Dokumentti

Henkilötiedot

Nimi: Joakim Engström
Koulutusohjelma: Automaatio- ja informaatioteknologia
Vuosikurssi: 2014
Päiväys: 30.4.2018

Yleiskuvaus

Tehty ohjelma on kasino-korttipeli tekstipohjaisena versiona. Toteutettu versio on ns. ”pakka-kasino”. Pelissä kerätään pisteitä, jotka lasketaan aina jokaisen pelikierroksen lopussa ja peli loppuu kun joku pelaajista saavuttaa 16 pistettä.

Jokaisen pelikierroksen alussa pakka sekoitetaan ja jakaja jakaa jokaiselle pelaajalle 4 korttia (eivät näy muille) sekä 4 korttia pöytään (näkyvät kaikille). Loput kortit jätetään pöydälle pinoon ylösalaisin. Jakajasta seuraava aloittaa pelaamisen. Seuraavalla pelikierroksella hän on jakaja.

Pelaaja voi kerrallaan käyttää jonkin kädessään olevista korteista: joko ottaa sillä pöydästä kortteja tai laittaa kortin pöytään. Jos pelaaja ei voi ottaa mitään pöydästä täytyy hänen laittaa jokin korteistaan pöytään. Jos pelaaja ottaa pöydästä kortteja hän kerää ne itselleen pinoon. Pinon sisällöstä lasketaan korttien loputtua pisteet. Pöydässä olevien korttien määrä voi vaihdella vapaasti. Jos joku vaikkapa ottaa kaikki kortit, täytyy seuraavan laittaa jokin korteistaan tyhjään pöytään. Aina käytettyään kortin, pelaaja ottaa käteensä pakasta uuden, niin että kädessä on aina 4 korttia. (Kun pöydällä oleva pakka loppuu, ei oteta enää lisää vaan pelataan niin kauan kuin kenelläkään on kortteja kädessä. Tällöin siis tietenkin alle 4 korttia on mahdollinen tilanne.) Kortilla voi ottaa pöydästä yhden tai useampia samanarvoisia kortteja ja kortteja, joiden summa on yhtä suuri, kuin kortin jolla otetaan.

Pelissä on muutama kortti, joiden arvo kädessä on arvokkaampi kuin pöydässä:
- Ässät : kädessä 14, pöydässä 1
- Ruutu-10 : kädessä 16, pöydässä 10
- Pata-2 : kädessä 15, pöydässä 2

Muiden korttien arvot ovat samat sekä pöydässä että kädessä.

Kun kaikilta loppuvat kädestä kortit saa viimeksi pöydästä kortteja ottanut loput pöydässä olevat kortit. Tämän jälkeen lasketaan pisteet ja lisätään ne entisiin pisteisiin.

Seuraavista asioista saa pisteitä:

-Jokaisesta mökistä(pelaaja saa kaikki kortit pöydästä kerralla) saa yhden pisteen
-Jokaisesta ässästä saa yhden pisteen
-Eniten kortteja saanut saa yhden pisteen
-Eniten patoja saanut saa 2 pistettä
-Ruutu-10 kortin omistaja saa 2 pistettä
-Pata-2 kortin omistaja saa pisteen

Toteutettu versio on helppo, jossa käytetään merkkipohjaista käyttöliittymää ja tallennuksessa ja latauksessa peli tallentaa ainoastaan pelaajien sen hetkiset pistemäärät.

Käyttöohje

Ohjelma käynnistetään komentoriviltä projektin src kansiossa ajamalla Pythonilla `main.py`. Ohjelmassa voi ensin valita, haluaako ladata aiemmin tallennetun pelin vai aloittaa uuden pelin. Jos valitsee pelin lataamisen niin pitää pitää syöttää ladattavan tiedoston nimi. Mikäli valitsee uuden pelin käyttäjää pyydetään syöttämään pelaajien lukumäärä ja tämän jälkeen pelaajien nimet. Peliä pelataan siihen asti kunnes se loppuu itsestään (joku pelaajista saavuttaa 16 pisteen rajan) tai käyttäjä lopettaa pelin syöttämällä kirjaimen q, jolloin käyttäjä voi halutessaan tallentaa pelin. Pelaajat pelaavat vuorotellen. Vuoron alussa pelaaja saa valita haluaako hän ottaa pöydästä kortteja vai ei, syöttämällä kirjaimen k tai e. Jos hän ottaa kortteja niin hän valitsee otettavat kortit. Mikäli hän aiemmin päätti olla ottamatta kortteja hänen täytyy valita kortin josta hän haluaa luopua (otettavat/luovutettavat kortit valitaan niiden indeksin mukaan). Sitten vuoro päättyy ja seuraavan pelaajan vuoro alkaa. Pelin voi lopettaa syöttämällä kirjaimen q. Tällöin pelaajalta kysytään mikäli hän haluaa tallentaa pelin. Jos hän haluaa tallentaa pelin niin häneltä pyydetään minkä nimiseen tiedostoon hän haluaa tallentaa pelaajien pisteet.

Ulkoiset kirjastot

Ohjelmassa käytetään ainoastaan Pythoniin sisäänrakennettuja kirjastoja.

Ohjelman rakenne

Ohjelmassa on neljä eri luokkaa Card, Player, Round ja Game. Game kuvaa peliä ylimmällä tasolla ja sisältää listan pelaajista ja korteista ja hoitaa pelin aloittamiseen ja lopettamiseen sisältyvät toiminnot. Se myös tallentaa pelaajien pisteet tiedostoon ja lataa pisteet tiedostosta. Round kuvaa yksittäistä pelikierrosta, pitää huolen pakan sekoittamisesta, korttien jakamisesta, pelivuoron vaihtamisesta, korttien paikkojen vaihtamisesta ja pisteiden laskemisesta. Player sisältää tiedot pelaajasta ja pelaajan korteista. Card on luokka korteille, jossa on kortin nimi, ja kortin pistearvot pöydässä ja kädessä.

Game

- players (lista Player objekteja)

- cards (lista Card objekteja)
- dealer (osoittaa pelaajaan joka on jakaja seuraavalla kierroksella

- give_players
    - Pyytää käyttäjältä tietoa pelaajista ja tallentaa tiedot players listaan Player objekteina.
- start_game
    - Kutsutaan pelin alussa. Toivottaa käyttäjän tervetulleeksi pelaamaan ja kysyy käyttäjältä mikäli hän haluaa aloittaa uuden pelin vai ladata kesken jääneen pelin.
- check_end
    - Tarkistaa jokaisen kierroksen jälkeen mikäli jollain pelaajalla on 16 pistettä.
- save_game
    - Tallentaa pelaajien pisteet tekstitiedostoon
- load_game
    - lataa pelaajien pisteet tekstitiedostosta
- set_dealer
    - Laittaa jokaisen kierroksen jälkeen dealer muuttujan osoittamaan seuraavaan pelaajaan.
- initialize_cards
    - Luo listan cards joka sisältää Card objekteja
- final_results
    - Tulostaa lopputulokset
- round
    - Hoitaa pelin yksittäistä kierrosta. Luo Round olion ja kutsuu Round luokan metodeja hoitamaan tiettyjä tehtäviä. Tarkistaa aina pelivuoron vaihtuessa onko pelaajilla kortteja jäljellä, mikäli ei ole pelaajien pisteet lasketaan ja tulostetaan.

Round

- table_cards (lista pöydässä olevista korteista)
- deck_cards (lista pakassa olevista korteista)
- next_player (pelivuorossa oleva pelaaja)

- deal_cards
    - Sekoittaa korttipakan ja jakaa jokaiselle pelaajalle 4 korttia, lisää table_cards listaan 4 korttia ja lisää loput kortit deck listaan.
- change_player
    - Mikäli kierros on juuri alkanut osoittaa jakasta seuraavaan pelaajaan muuten siirtyy askeleen eteenpäin pelaajalistassa ja palauttaa pelaajan (eli pelivuorossa olevan pelaajan.
- the_play
    - Hoitaa yksittäistä pelivuoroa. Pyytää pelaajalta syötettä siitä mitä hän haluaa tehdä ja kutsuu muita metodeja hoitamaan pelivuoron eri osia.
- take_cards

    - Kutsutaan mikäli pelaaja haluaa ottaa kortteja pöydästä. Kutsuu muita metodeja hoitamaan eri tehtäviä.
- pic_cards
    - Kysyy pelaajalta mitkä kortit hän haluaa ottaa pöydästä.
- add_card
    - Kysyy pelaajalta minkä kädessä olevan kortin hän haluaa käyttää.
- cards_to_player_deck
    - Siirtää kortteja pöydästä ja pelaajan kädestä pelaajan pakkaan.
- cards_to_player_hand
    - Siirtää kortin pakasta pelaajan käteen mikäli pakassa on vielä kortteja.
- add_table_card
    - Siirtää kortin pelaajan kädestä pöytään.
- check_validity
    - Tarkistaa onko pelaajan liike laillinen.
- calculate_points
    - Jakaa pelaajille pisteet kierroksen päätyttyä tai antaa pelaajalle mökki-pisteen.
- check_end
    - Tarkistaa onko pelaajilla edelleen kortteja kädessä.

Player

- name (pelaajan nimi)
- points (pelaajan pisteet)
- mokki (pelaajan mökit kaynnissa olevan kierroksen aikana)
- cards_hand (lista pelaajan kädessä olevista korteista tietyllä hetkellä)
- cards_deck (lista pelaajan pakassa olevista korteista tietyllä hetkellä)

Card

- name (kortin nimi joka näkyy pelaajille, esimerkiksi Herttajätkä-11)
- table_value (kortin arvo pöydässä)
- hand_value (kortin arvo kädessä)

Luokat Player ja Card olivat selviä valintoja sillä ne koostuvat yhteen kuuluvista tiedoista. Luokat Game ja Round taas ovat sopivan kokoisia kokonaisuuksia tiettyjen ongelmien ratkaisemiseen ja ne muodostavat myös loogisesti yhteenkuuluvat kokonaisuudet. Mahdollisia luokkia olisivat myös olleet luokka “Board” jossa olisi ollut tiedot korttien sijainnista mutta mielestäni oli toimivampaa jakaa ne Player ja Round luokkiin ja luokka “Play” joka olisi kuvannut pelitilannetta jossa pelaaja tekee siirron mutta se olisi muodostanut suhteellisen pienen kokonaisuuden joten päätin sisällyttää vaaditut toiminnallisuudet luokan Round metodeihin.

Algoritmit

Pelin merkittävin ja monimutkaisin algoritmi on algoritmi joka tarkistaa pöydästä otettujen korttien laillisuus. Jokaisella kortilla on oma pistearvonsa. Pelaajan pöydästä valitsemat kortit ovat listassa ja tämä listan mahdollisia kombinaatioita aletaan käydä läpi ja mikäli löytyy kombinaatio joka vastaa pistearvoltaan pelaajan omaa valitsemaa korttia, poistetaan kombinaatio listasta ja käydään pienempää listaa läpi kunnes jokainen mahdollinen kombinaatio on käyty läpi. Mikäli lista on tyhjä pelaajan valitsemat kortit ovat sallittuja ja mikäli lista ei ole tyhjä ne eivät ole sallittuja.

Toinen vaihtoehtoinen toteutus olisi ollut tehdä algoritmi joka käy läpi kaikki pöydässä olevien korttien kombinaatiot, pelaajan valitseman kortin perusteella ja pelaaja voisi sitten valita näistä kombinaatiosta mitkä kortit hän haluaa ottaa. Eli sen sijaan että pelaaja kertoo ensin itse mitkä kortit hän haluaa pöydästä, hänelle kerrottaisiin mitkä kortit ovat otettavissa. Tämä algoritmi olisi hieman monimutkaisempi ja mikäli kortteja olisi pöydässä paljon saattaisi se viedä hieman enemmän aikaa.

Muita haastavimpia algoritmeja ei ohjelmassa vaadita kun myös korttien sekoittamiseen löytyy Pythonilta valmis funktio random.shuffle() moduulista random.

Tietorakenteet

Tässä ohjelmassa Pythonin taulukkorakenne on riittävä tiedon varastoimiseen. Pythonilla on taulukkorakenteeseen laajat sisäänrakennetut funktiot, taulukosta hakemiseen, lisäämiseen ja poistamiseen, mikä helpottaa koodaamista ja lyhentää tarvitun koodin määrää. Mahdollinen vaihtoehto olisi ollut linkitetyt listat mutta tässä tapauksessa taulukot riittävät. Varastoitavan tiedon määrä on myös sen verran pieni että taulukot ajavat asiaansa.

Tiedostot

Ohjelma käsittelee CSV(Comma-separated values) tiedostoja. Kyseessä on tekstitiedosto jossa tässä tapauksessa tiedot on eritelty toisistaan puolipisteillä ja rivinvaihdolla. Ensimmäisellä rivillä on tiedoston nimi “Kasino” ja tallennuspäivämäärä. Toisella rivillä “Nimi” ja Pisteet” ja sen jälkeen jokaisen pelaajan nimet ja pisteet on omilla riveillään. Src kansiosta löytyy test.csv josta voi katsoa miten tiedosto on rakennettu.

Testaus

Ohelmaa varten tehtiin jonkin verran yksikkötestejä jossa testattiin eri metodeja. Olennaisin testattava metodi oli check_validity metodi kuten suunnitelmassa mainittiin. Suurin osa testauksesta oli kuitenkin yksikkötestien lisäksi tehtävää ohjelman testaamista pelaamalla pelaajien pelivuoroja, kierroksia ja kokonaisia pelejä läpi. Ohjelman eri mahdollisia haaroja testattiin ja annettiin erilaisia oikeita ja virheellisiä syötteitä. Ohjelmaa testattiin aina kun lisää toiminnallisuutta lisättiin.

Ohjelman puutteet ja viat

Mikäli jatkaisin ohjelman eteenpäin kehittämistä pyrkisin tekemään siitä graafisesti miellyttävämmän näköisen. Ohjelmassa voisi myös eritellä mistä pelaajien pisteet koostuivat.

3 parasta ja 3 heikointa kohtaa

Mielestäni ohjelma ohjeistaa käyttäjää hyvin jokaisessa kohdassa siitä mitä hän voi tai mitä hänen pitää tehdä ja mitkä ovat mahdollisia syötteitä. Myös luokat on jaettu loogisiin kokonaisuuksiin.

Ohjelmaan voisi kirjoittaa enemmän yksikkötestejä, kommentteja ja koodia voisi jäsennellä hieman paremmin. Koodissa on myös jonkin verran toisteisuutta.

Poikkeamat suunnitelmasta

Aikataulu muuttui hieman ja ohjelman tekeminen meni myöhäisemmäksi kuin olin suunnitellut. Tämä johtui muista opiskelukiireistä. Muuten etenin suunnitelman mukaan ja myös ajankäyttöarvio oli suhteellisen osuva.

Toteutunut työjärjestys ja aikataulu

Ohjelma toteutettiin luomalla aluksi kaikki luokat ja olioiden muuttujat. Tämän jälkeen luotiin metodi toisensa jälkeen aloittamalla keskeisimmistä ja haastavimmista metodeista. Ohjelmaa testattiin aina kun uutta toiminnallisuutta luotiin. Lopuksi testattiin isompia kokonaisuuksia jonka jälkeen löytyi jonkin verran korjattavaa.

Arvio lopputuloksesta

Ohjelma toimii mielestäni hyvin ja se on jaettu järkeenkäypiin kokonaisuuksiin. Ohjelmaa on testattu aika paljon joten bugeja ei pitäisi olla ainakaan kovin paljon. Olisin mielelläni tehnyt graafisen käyttöliittymän mutta itselläni ei löytynyt riittävästi aikaa muiden kiireiden takia.

Koodia voisi varmaan parantaa vähentämällä toisteisuutta ja jäsentämällä sitä hieman paremmin. Yksikkötestejä olisi voinut kirjoittaa enemmän. Jotkin metodit ovat aika isoja ja niitä olisi voinut jakaa pienempii osiin.

Kirjallisuusviitteet ja linkit

https://docs.python.org
https://www.tutorialspoint.com/python/index.htm

https://www.programiz.com/python-programming/examples/shuffle-card
https://stackoverflow.com/
https://plus.cs.hut.fi/y2/2018/

Liitteet

Lähdekoodi

main.py

```
from game import Game

while True:
    game = Game()
    game.initialize_cards()
    while True:
        if game.start_game() == -1:
            if game.quit():
                exit()
        else:
            break
    while True:
        if game.round() == -1:
            game.end_game()
            break
        elif game.check_end():
            game.show_results()
            break
    if not game.new_game():
        break
```

game.py

```
import csv, datetime
from card import Card
from round import Round
from player import Player

class Game:

    card_numbers = ['2', '3', '4', '5', '6', '7', '8', '9', '10',
```

```
'jatka', 'rouva', 'kuningas', 'assa']
    card_suits = ['Pata', 'Risti', 'Hertta', 'Ruutu']


    def __init__(self):
        self.players = []
        self.cards = []
        self.dealer = None

    # Creates deck of cards
    def initialize_cards(self):
        i = 2
        for card_number in self.card_numbers:
            for suit in self.card_suits:
                if card_number == "jatka" or card_number ==
"rouva" or card_number == "kuningas" or card_number == "assa":
                    name = suit + card_number
                else:
                    name = suit + "-" + card_number
                if i == 14:
                    hand_value = 14
                    table_value = 1
                elif i == 10 and suit == "Ruutu":
                    hand_value = 16
                    table_value = 10
                elif i == 2 and suit =="Pata":
                    hand_value = 15
                    table_value = 2
                else:
                    hand_value, table_value = i, i
                self.cards.append(Card(name, table_value,
hand_value))
            i += 1

    # A round in game
    def round(self):
        r = Round(self.set_dealer())
        cards = list(self.cards)
        r.deal_cards(cards, self.players)
        # While loop continues until all players are out of cards
or player wants to quit game
        while not r.check_end(self.players):
            # the.play() returns -1 when player wants to quit game
            if r.the_play() == -1:
                return -1
            r.change_player(self.players)
```

```
        r.calculate_points(self.players)
        r.empty_players_decks(self.players)


    def set_dealer(self):
        if not self.dealer or self.dealer == self.players[-1]:
            self.dealer = self.players[0]
        else:
            self.dealer =
self.players[self.players.index(self.dealer) + 1]
        return self.dealer

    def save_game(self):
        while True:
            file = input("Minka nimen haluat antaa
tiedostolle?\n")
            csv_file = file + (".csv")
            date = datetime.datetime.today().strftime ("%d.%m.%Y")
            data = ["Kasino", date]
            headings = ["Nimi", "Pisteet"]
            try:
                with open (csv_file, "w", newline="") as csvfile:
                    writer = csv.writer(csvfile, delimiter=";")
                    writer.writerow(data)
                    writer.writerow(headings)
                    for player in self.players:
                        data = [player.name, player.points]
                        writer.writerow(data)
                    break
            except Exception:
                print("Jotain meni pieleen ja tiedoston
tallentaminen epaonnistui! Tarkista etta samanniminen tiedosto ei
ole parhailaan auki\n")

    def load_game(self):
        while True:
            csv_file = input("Minka tiedoston haluat avata?\n")
            try:
                with open(csv_file) as csvfile:
                    reader = csv.reader(csvfile, delimiter=";")
                    row1 = next(reader)
                    row2 = next(reader)
                    try:
                        if row1[0] != "Kasino" or row2[0] !=
"Nimi" or row2[1] != "Pisteet":
                            print("Virheellinen tiedosto!\n")
```

```
                        for row in reader:
                            try:
                                success = True
                                player = Player(row[0])
                                player.points = int(row[1])
                                self.players.append(player)
                            except ValueError:
                                print("Virheellinen tiedosto!\n")
                                success = False
                        if success:
                            return success
                    except Exception:
                        print("Virheellinen tiedosto!\n")
            except Exception:
                print("Tiedostoa ei loytynyt!\n")
            while True:
                choice = input("Haluatko yrittaa tiedoston
avaamista uudelleen? (e/k)\n")
                choice = choice.lower()
                if choice == "k":
                    break
                elif choice == "e":
                    return False
                elif choice == "q":
                    return -1
                else:
                    self.invalid_input()

    def check_end(self):
        for player in self.players:
            if player.points >= 16:
                return True
        print("Pisteet kierroksen jalkeen:\n")
        self.print_scores()
        return False

    def show_results(self):
        print("Peli loppui!\nTassa ovat lopulliset pisteet:\n")
        self.print_scores()

    def start_game(self):
        print("Tervetuloa pelaamaan kasinoa!\nMikali haluat
lopettaa pelaamisen kesken pelin, syota q.")
        while True:
            choice = input("Haluatko ladata kesken jaaneen pelin?
(k/e)?\n")
```

```
            choice = choice.lower()
            if choice == "k":
                load_game = self.load_game()
                if load_game:
                    # load_game == -1 if player wants to quit
                    if load_game == -1:
                        return -1
                    else:
                        break
                else:
                    if self.give_players() == -1:
                        return -1
                    break
            elif choice == "e":
                if self.give_players() == -1:
                    return -1
                break
            elif choice == "q":
                return -1
            else:
                self.invalid_input()

    # Asks user to give players
    def give_players(self):
        print("Aloitetaan uusi peli!\n")
        print("Sallittu pelaajien lukumaara on 2-6.\n")
        while True:
            amount = input("Anna pelaajien lukumaara:\n")
            if amount == "q" or amount == "Q":
                return -1
            try:
                amount = int(amount)
                if amount > 6 or amount < 2:
                    print("Virheellinen syote! Syotteen pitaa olla
2 ja 6 valissa!")
                else:
                    for i in range(amount):
                        name = input("Anna pelaajan nimi: ")
                        self.players.append(Player(name))
                    break
            except ValueError:
                print("Virheellinen syote! Syotteen pitaa olla
kokonaisluku!\n")

    def end_game(self):
        while True:
```

```
            choice = input("Halutko tallentaa pelin? (k/e)?\n")
            choice = choice.lower()
            if choice == "k":
                self.save_game()
                break
            elif choice == "e":
                break
            else:
                self.invalid_input()

    def invalid_input(self):
        print("Virheellinen syote! Syota joko k tai e")

    def print_scores(self):
        ordered = sorted(self.players, key=lambda x: x.points,
reverse = True)
        for player in ordered:
            print("{}: {} pistetta".format(player.name,
player.points))
        print("\n")

    def new_game(self):
        while True:
            choice = input("Haluatko pelata uudestaan? (k/e)\n")
            choice = choice.lower()
            if choice == "k":
                return True
            elif choice == "e":
                return False
            else:
                self.invalid_input()

    def quit(self):
        while True:
            choice = input("Haluatko varmasti lopettaa pelin
(k/e)?\n")
            choice = choice.lower()
            if choice == "e":
                return False
            elif choice == "k":
                return True
            else:
                self.invalid_input()
```

round.py

```
import random, itertools

class Round:
    # Last player who took cards from table
    last_player = None

    def __init__(self, next_player):
        self.table_cards = []
        self.deck_cards = []
        self.next_player = next_player

    def deal_cards(self, cards, players):
        random.shuffle(cards)
        for i in range(4):
            for player in players:
                player.cards_hand.append(cards.pop(0))
            self.table_cards.append(cards.pop(0))
        self.deck_cards = cards

    def change_player(self, players):
        if players.index(self.next_player) == len(players) - 1:
            self.next_player = players[0]
        else:
            self.next_player =
players[players.index(self.next_player) + 1]

    def show_table_cards(self):
        print("Poyta:")
        for idx, card in enumerate(self.table_cards):
            print("{}. [{}]".format(idx + 1, card), end=" ")
        print("\n")

    def show_player_cards(self):
        print("Katesi:")
        for idx, card in enumerate(self.next_player.cards_hand):
            print("{}. [{}]".format(idx + 1, card), end=" ")
        print("\n")

    # Method that deals with a player's turn. Calls other methods
to handle specific tasks.
    def the_play(self):
        print("Vuorossa oleva pelaaja: {}
\n".format(self.next_player))
        if not self.next_player.cards_hand:
            print("Koska sinulla ei ole kortteja, siirtyy vuoro
seuraavalle pelaajalle\n")
```

```
        elif not self.table_cards:
            print("Koska poyta on tyhja, joudut laittamaan yhden
korteistasi poytaan.\n")
            # Returns -1 when player wants to quit game
            if self.add_table_card(self.add_card()) == -1:
                return -1
        else:
            self.show_table_cards()
            self.show_player_cards()
            while True:
                choice = input("Haluatko ottaa poydasta kortteja
(k/e)?\n")
                choice = choice.lower()
                if choice == "k":
                    success = self.take_cards()
                    # success == -1 when player wants to quit game
                    if success == -1:
                        return -1
                    # success == false when player decided not to
take cards from table
                    elif success:
                        break
                    else:
                        print("Joudut laittamaan yhden korteistasi
poytaan\n")
                        self.add_table_card(self.add_card())
                        break
                elif choice == "e":
                    print("Joudut laittaamaan yhden korteistasi
poytaan\n")
                    if self.add_table_card(self.add_card()) == -1:
                        return -1
                    break
                elif choice == "q":
                        if self.quit():
                            return -1
                else:
                    self.invalid_input()
    # Method for taking cards from table. Calls other methods for
specific tasks
    def take_cards(self):
        while True:
            player_card = self.add_card()
            if player_card == -1:
                return -1
            table_cards = self.pic_cards()
```

```
            if table_cards == -1:
                return -1
            if self.check_validity(player_card, table_cards):
                self.cards_to_player_deck(player_card,
table_cards)
                self.check_mokki()
                self.card_to_player_hand()
                Round.last_player = self.next_player
                return True
            else:
                print("Liikkeesi on laiton!\n")
                while True:
                    choice = input("Haluatko yrittaa uudestaan
ottaa poydasta kortteja (k/e)?\n")
                    choice = choice.lower()
                    if choice == "k":
                        break
                    elif choice == "e":
                        return False
                    else:
                        self.invalid_input()

    # Asks player which of his cards he wants to use
    def add_card(self):
        while True:
            self.show_table_cards()
            self.show_player_cards()
            card = input("Valitse minka kortin kaytat. Anna kortin
indeksi: \n")
            try:
                card = int(card)
                if card > len(self.next_player.cards_hand) or card
< 1:
                    print("Virheellinen numero!\n")
                else:
                    return self.next_player.cards_hand[card - 1]
            except ValueError:
                if card == "q":
                        if self.quit():
                            return -1
                else:
                    print("Syotteen pitaa olla kokonaisluku!\n")

    # Moves one card from player's hand to table
    def add_table_card(self, card):
        # Card is -1 if player wants to quit game
```

```
        if card == -1:
            return -1
        else:
            self.table_cards.append(card)
            self.next_player.cards_hand.remove(card)
            self.card_to_player_hand()

    # Asks player which cards he wants to take from table
    def pic_cards(self):
        while True:
            cards = []
            try:
                # success will stay true if all user inputs are
valid
                success = True
                self.show_table_cards()
                self.show_player_cards()
                choice = input("Anna kokonaislukuina haluamasi
korttien indeksit valilyonnein erotettuna: \n")
                if choice == "q" or choice == "Q":
                    return -1
                card_numbers = [int(i) for i in choice.split()]
                if not card_numbers:
                    raise ValueError
                # Removes duplicates
                card_numbers = list(set(card_numbers))
                for number in card_numbers:
                    if number < 1 or number >
len(self.table_cards):
                        print("Virheellinen kokonaisluku!\n")
                        success = False
                        break
                    else:
                        cards.append(self.table_cards[number - 1])
                if success:
                    return cards
            except ValueError:
                print("Syotteiden pitaa olla kokonaislukuja! \n")

    # Moves cards from table and player's hand to player's deck
    def cards_to_player_deck(self, player_card, table_cards):
        # Removes a card from player's hand and inserts it in
player's deck
        self.next_player.cards_deck.append(player_card)
        self.next_player.cards_hand.remove(player_card)
        for card in table_cards:
```

```
            # Removes cards from table and inserts them in
player's deck
            self.next_player.cards_deck.append(card)
            self.table_cards.remove(card)

    # Moves card from deck to player's hand if there are still
cards left in deck
    def card_to_player_hand(self):
        if len(self.next_player.cards_hand) < 4 and
self.deck_cards:

self.next_player.cards_hand.append(self.deck_cards.pop())


    # Checks if a player can take the cards he wants from the
table.
    def check_validity(self, player_card, table_cards):
        #Copies the table_cards list to an new list "table"
        table = list(table_cards)
        i = len(table)
        while i > 0:
            comb_sum = 0
            found = False
            for comb in itertools.combinations(table, i):
                for item in comb:
                    comb_sum += item.table_value
                if comb_sum == player_card.hand_value:
                    sublist = list(comb)
                    for value in sublist:
                        table.remove(value)
                    # When a sublist is found we need to update
table list before looking for new sublists
                    found = True
                    break
                comb_sum = 0
            if found:
                pass
            elif i > len(table):
                i = len(table)
            else:
                i -= 1
        if len(table) == 0:
            return True
        else:
            return False
```

```
    def check_mokki(self):
        if not self.table_cards:
            self.next_player.mokki += 1

    def check_end(self, players):
        for player in players:
            if player.cards_hand:
                return False
        table_cards = list(self.table_cards)
        # The player who last took cards from table gets the rest
of the cards in table
        for card in table_cards:
            # Removes cards from table and inserts them in
player's deck
            Round.last_player.cards_deck.append(card)
            self.table_cards.remove(card)
        return True

    def calculate_points(self, players):
        most_cards = [players[0]]
        most_spades = [players[0]]
        # Keeps track of the highest spade amount
        high_spades = 0
        for player in players:
            spades = 0
            if player.mokki:
                player.points += 1
                player.mokki = 0
            for card in player.cards_deck:
                if card.name == "Ruutu-10":
                    player.points += 2
                elif card.name == "Pata-2":
                    player.points += 1
                # Gives points for aces
                elif card.hand_value == 14:
                    player.points += 1
                if "Pata" in card.name:
                    spades += 1
            # Checks which player has most cards and spades
            if player is not players[0]:
                if len(player.cards_deck) ==
len(most_cards[0].cards_deck):
                    most_cards.append(player)
                elif len(player.cards_deck) >
len(most_cards[0].cards_deck):
                    most_cards = [player]
```

```
                if spades == high_spades:
                    most_spades.append(player)
                elif spades > high_spades:
                    most_spades = [player]
            if spades > high_spades:
                high_spades = spades
        for player in most_spades:
            player.points += 2
        for player in most_cards:
            player.points += 1

    def invalid_input(self):
        print("Virheellinen syote! Syota joko k tai e")

    def quit(self):
        while True:
            choice = input("Haluatko varmasti lopettaa pelin
(k/e)?\n")
            choice = choice.lower()
            if choice == "e":
                return False
            elif choice == "k":
                return True
            else:
                self.invalid_input()

    # Empties players' decks after each round
    def empty_players_decks(self, players):
        for player in players:
            player.cards_deck = []
```

player.py

```
class Player:

    def __init__(self, name):
        self.name = name
        self.points = 0
        self.mokki = 0
        self.cards_hand = []
        self.cards_deck = []

    def __str__(self):
        return self.name
```

card.py

```
class Card:

    def __init__(self, name, table_value, hand_value):
        self.name = name
        self.table_value = table_value
        self.hand_value = hand_value

    def __str__(self):
        return self.name
```

test.py

```
import unittest

from round import Round
from game import Game
from player import Player
from card import Card

class Test(unittest.TestCase):

    def test_methods(self):

        game = Game()
        game.initialize_cards()
        game.players = [Player("Mike"), Player("Tim"),
Player("Jack"), Player("Nick")]

        # Points for Mike
        game.players[0].cards_deck = game.cards[0:26]
        # Points for Tim
        game.players[1].cards_deck = game.cards[26:37]
        # Points for Jack
        game.players[2].cards_deck = game.cards[37:49]
        # Points for Nick
        game.players[3].cards_deck = game.cards[49:52]

        rnd = Round(game.players[0])
        game.players[1].mokki = 2
        rnd.calculate_points(game.players)

        # Check that the points of each player is correct
        self.assertEqual(4, game.players[0].points, "Wrong amount
```

```
of points!")
        self.assertEqual(3, game.players[1].points, "Wrong amount
of points!")
        self.assertEqual(1, game.players[2].points, "Wrong amount
of points!")
        self.assertEqual(3, game.players[3].points, "Wrong amount
of points!")


        # Check that the validation algorithm works properly
        self.assertEqual(True, Round.check_validity(self,
Card("Pata-7", 7, 7), [Card("Ruutu-3", 3, 3), Card("Hertta-4", 4,
4)]), "Validation failed")
        self.assertNotEqual(True, Round.check_validity(self,
Card("Pata-8", 8, 8), [Card("Ruutu-3", 3, 3), Card("Hertta-4", 4,
4)]), "Validation failed")
        self.assertEqual(True, Round.check_validity(self,
Card("Pata-2", 2, 15), [Card("Pata-8", 8, 8), Card("Hertta-7", 7,
7)]), "Validation failed")
        self.assertNotEqual(True, Round.check_validity(self,
Card("Pata-10", 10, 10), [Card("Ruutu-5", 5, 5), Card("Hertta-5",
5, 5), Card("Pata-5", 5, 5)]), "Validation failed")

        # Check that the dealer changes correctly after each round
        game.set_dealer()
        self.assertEqual(game.players[0], game.dealer, "Wrong
dealer!")
        game.set_dealer()
        self.assertEqual(game.players[1], game.dealer, "Wrong
dealer!")
        game.set_dealer()
        game.set_dealer()
        game.set_dealer()
        self.assertEqual(game.players[0], game.dealer, "Wrong
dealer!")


if __name__ == '__main__':
    unittest.main()
```

Ajoesimerkit

Tervetuloa pelaamaan kasinoa!
Mikali haluat lopettaa pelaamisen kesken pelin, syota q.
Haluatko ladata kesken jaaneen pelin (k/e)?
e
Aloitetaan uusi peli!

Sallittu pelaajien lukumaara on 2-6.

Anna pelaajien lukumaara:
3
Anna pelaajan nimi: Tim
Anna pelaajan nimi: Jack
Anna pelaajan nimi: John
Vuorossa oleva pelaaja: Tim

Poyta:
1. [Ruutu-8] 2. [Ruutu-2] 3. [Ristirouva] 4. [Risti-4]

Katesi:
1. [Hertta-2] 2. [Herttaassa] 3. [Herttarouva] 4. [Ruutu-6]

Haluatko ottaa poydasta kortteja (k/e)?
k
Poyta:
1. [Ruutu-8] 2. [Ruutu-2] 3. [Ristirouva] 4. [Risti-4]

Katesi:
1. [Hertta-2] 2. [Herttaassa] 3. [Herttarouva] 4. [Ruutu-6]

Valitse minka kortin kaytat. Anna numero:
3
Poyta:
1. [Ruutu-8] 2. [Ruutu-2] 3. [Ristirouva] 4. [Risti-4]

Katesi:
1. [Hertta-2] 2. [Herttaassa] 3. [Herttarouva] 4. [Ruutu-6]

Anna kokonaislukuina haluamasi kortit valilyonnein erotettuna:

3
Vuorossa oleva pelaaja: Jack

Poyta:
1. [Ruutu-8] 2. [Ruutu-2] 3. [Risti-4]

Katesi:
1. [Ruutu-10] 2. [Ruutu-4] 3. [Ruutujatka] 4. [Risti-10]

Haluatko ottaa poydasta kortteja (k/e)?
k
Poyta:
1. [Ruutu-8] 2. [Ruutu-2] 3. [Risti-4]

Katesi:
1. [Ruutu-10] 2. [Ruutu-4] 3. [Ruutujatka] 4. [Risti-10]

Valitse minka kortin kaytat. Anna numero:
1
Poyta:
1. [Ruutu-8] 2. [Ruutu-2] 3. [Risti-4]

Katesi:
1. [Ruutu-10] 2. [Ruutu-4] 3. [Ruutujatka] 4. [Risti-10]

Anna kokonaislukuina haluamasi kortit valilyonnein erotettuna:
1 2 3
Liikkeesi on laiton!

Haluatko yrittaa uudestaan ottaa poydasta kortteja (k/e)?
k
Poyta:
1. [Ruutu-8] 2. [Ruutu-2] 3. [Risti-4]

Katesi:
1. [Ruutu-10] 2. [Ruutu-4] 3. [Ruutujatka] 4. [Risti-10]

Valitse minka kortin kaytat. Anna numero:
4
Poyta:
1. [Ruutu-8] 2. [Ruutu-2] 3. [Risti-4]

Katesi:
1. [Ruutu-10] 2. [Ruutu-4] 3. [Ruutujatka] 4. [Risti-10]

Anna kokonaislukuina haluamasi kortit valilyonnein erotettuna:

1 2

....

Vuorossa oleva pelaaja: John

Poyta:
1. [Herttaassa] 2. [Ruutuassa] 3. [Ruutu-10]

Katesi:
1. [Pata-3]

Haluatko ottaa poydasta kortteja (k/e)?
e
Joudut laittaamaan yhden korteistasi poytaan

Poyta:
1. [Herttaassa] 2. [Ruutuassa] 3. [Ruutu-10]

Katesi:
1. [Pata-3]

Valitse minka kortin kaytat. Anna numero:
1

Pisteet kierroksen jalkeen:

John: 9 pistetta
Jack: 2 pistetta
Tim: 1 pistetta

Vuorossa oleva pelaaja: Jack

Poyta:
1. [Hertta-3] 2. [Hertta-6] 3. [Risti-4] 4. [Hertta-8]

Katesi:
1. [Risti-9] 2. [Hertta-9] 3. [Ruutuassa] 4. [Pataassa]

Haluatko ottaa poydasta kortteja (k/e)?
q
Haluatko varmasti lopettaa pelin (k/e)?
k
Halutko tallentaa pelin? (k/e)?
k
Minka nimen haluat antaa tiedostolle?

ajoesimerkki
Haluatko pelata uudestaan? (k/e)
e

Tervetuloa pelaamaan kasinoa!
Mikali haluat lopettaa pelaamisen kesken pelin, syota q.
Haluatko ladata kesken jaaneen pelin (k/e)?
k
Minka tiedoston haluat avata?
test.csv
Vuorossa oleva pelaaja: Martin

Poyta:
1. [Hertta-7] 2. [Risti-10] 3. [Ruutu-8] 4. [Ruutu-6]

Katesi:
1. [Ristirouva] 2. [Pata-4] 3. [Herttajatka] 4. [Ruutu-9]

Haluatko ottaa poydasta kortteja (k/e)?
e
Joudut laittaamaan yhden korteistasi poytaan

Poyta:
1. [Hertta-7] 2. [Risti-10] 3. [Ruutu-8] 4. [Ruutu-6]

Katesi:
1. [Ristirouva] 2. [Pata-4] 3. [Herttajatka] 4. [Ruutu-9]

Valitse minka kortin kaytat. Anna kortin indeksi:
2

...

Vuorossa oleva pelaaja: Mike

Poyta:
1. [Ruutukuningas]

Katesi:
1. [Ruutuassa]

Haluatko ottaa poydasta kortteja (k/e)?
e
Joudut laittaamaan yhden korteistasi poytaan

Poyta:
1. [Ruutukuningas]

Katesi:
1. [Ruutuassa]

Valitse minka kortin kaytat. Anna kortin indeksi:
1

Peli loppui!
Tassa ovat lopulliset pisteet:

Martin: 17 pistetta
Adam: 17 pistetta
John: 16 pistetta
Stewart: 16 pistetta
Mike: 7 pistetta

Haluatko pelata uudestaan? (k/e)
e