Języki skryptowe

dokumentacja projektu Pasjans

Karolina Kozubik
grupa 3/5, Informatyka st. I, rok II
Wydział Matematyki Stosowanej

23 stycznia 2024

Część I

Opis programu

Celem projektu było zaproponowanie infrastruktury serwerowej realizującej Zadanie 5 z konkursu Algorytmion z roku 2016. Pełna treść zadania:

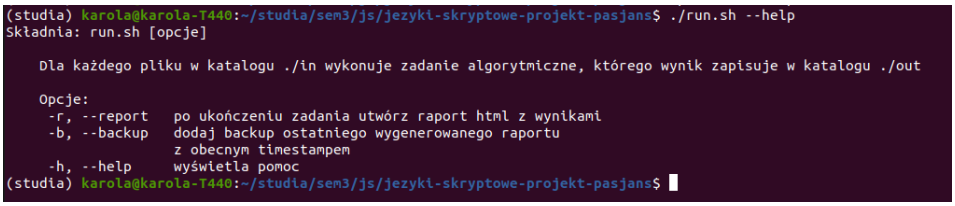
Pasjans jest grą karcianą (najczęściej jednoosobową), której celem jest ułożenie krat wg pewnego wzorca. W zdecydowanej większości pasjansów powodzenie gracza zależy od jego umiejętności, istnieje jednak i taki, który zależy tylko od „szczęścia" gracza. W pasjansie tym, po przetasowaniu talii 24 krat, układamy (koszulkami do góry) cztery rzędy krat po sześć – z wyjątkiem rzędu czwartego, w którym ostatnią kartę zatrzymujemy w dłoni. Celem gry jest ułożenie kart we właściwej kolejności: w pierwszym rzędzie kiery (9, 10, walet, dama, król i as), w kolejnych odpowiednie kolory to: karo, trefl i pik (kolejność figur jak w kierach). Gra kończy się, gdy w dłoni mieć będziemy asa pik. Jeśli kartą w dłoni nie jest as pik, to kładziemy ją (obrazkiem do góry) na odpowiadające jej miejsce (np. gdyby była to dama tref, to odłożylibyśmy ją do rzędu trzeciego do czwartej kolumny), biorąc w dłoń leżącą tam kartę. Jeśli natrafimy ostatecznie na asa pik, to (o ile będzie taka potrzeba) odsłaniamy, nie zmieniając ich położeń w układzie, pozostałe nieodsłonięte (leżące koszulkami do góry) dotychczas karty. Jeżeli wszystkie karty leżeć będą we właściwej kolejności – wygraliśmy, jeśli nie – przegraliśmy. Napisz program, który generował będzie losowe rozłożenie kart, a następnie rozgrywał będzie partie tego pasjansa. Program działał będzie do pierwszego zwycięstwa. Partie przegrane nie interesują nas, chcemy jedynie wiedzieć, za którym razem wygraliśmy oraz zobaczyć wizualizację zwycięskiej partii. Przez wizualizację rozumiemy tutaj kolejne ruchy gracza począwszy od wejściowego układu, a skończywszy na asie pik „w dłoni" (łącznie z ewentualnym odsłanianiem kart nieodsłoniętych). Sposób tej wizualizacji pozostawiamy w gestii rozwiązującego. [1]

W celu spełnienia założeń projektu, zadanie zostało zmodyfikowane. Algorytm przyjmuje liczbę oznaczającą ilość partii, jaką może maksymalnie rozegrać. Algorytm kończy działanie, jeśli rozegra daną ilość partii lub wcześniej, jeśli rozegrana partia zakończy się sukcesem. Algorytm zwraca liczbę faktycznie rozegranych partii, informację o tym, czy udało się wygrać, oraz reprezentację ostatniej rozegranej partii.

Program zarządzający realizacją algorytmu pobiera dane wejściowe z katalogu `/in/`, a dane wyjściowe zapisuje w katalogu `/out/`. Program po podaniu odpowiednich opcji ma możliwość wygenerowania raportu w pliku `html`, oraz stworzenia zapasowej kopii raportu opatrzonej znacznikiem czasowym.

Instrukcja obsługi

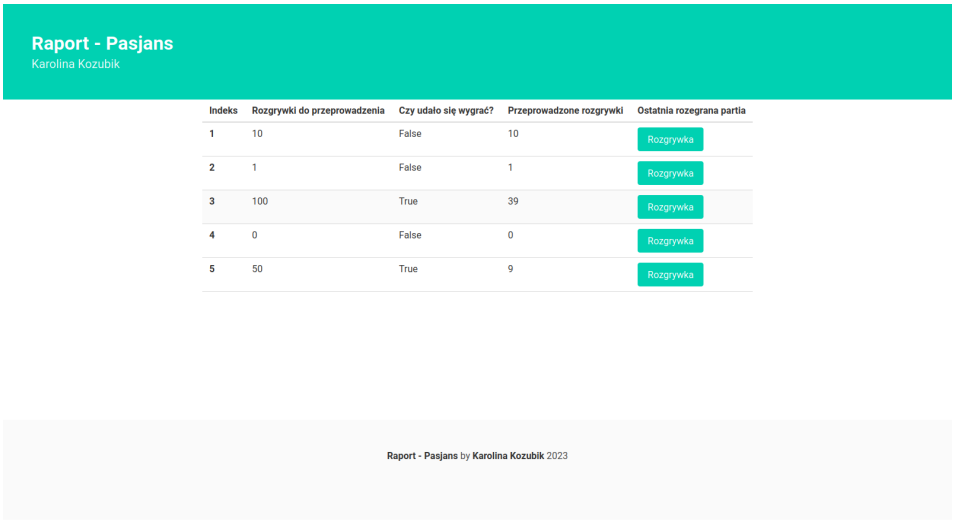
Program jest wzorowany na aplikacjach konsolowych systemów typu Unix. Aby uruchomić program należy z poziomu konsoli systemowej wykonać skrypt `run.sh`. Aby uzyskać więcej informacji o programie, należy jako argument podać opcję `-h` lub `--help`.



Rysunek 1: Uruchomienie programu z opcją --help

Program wypisuje na wyjście składnię polecenia oraz opisy opcji, z których można skorzystać.

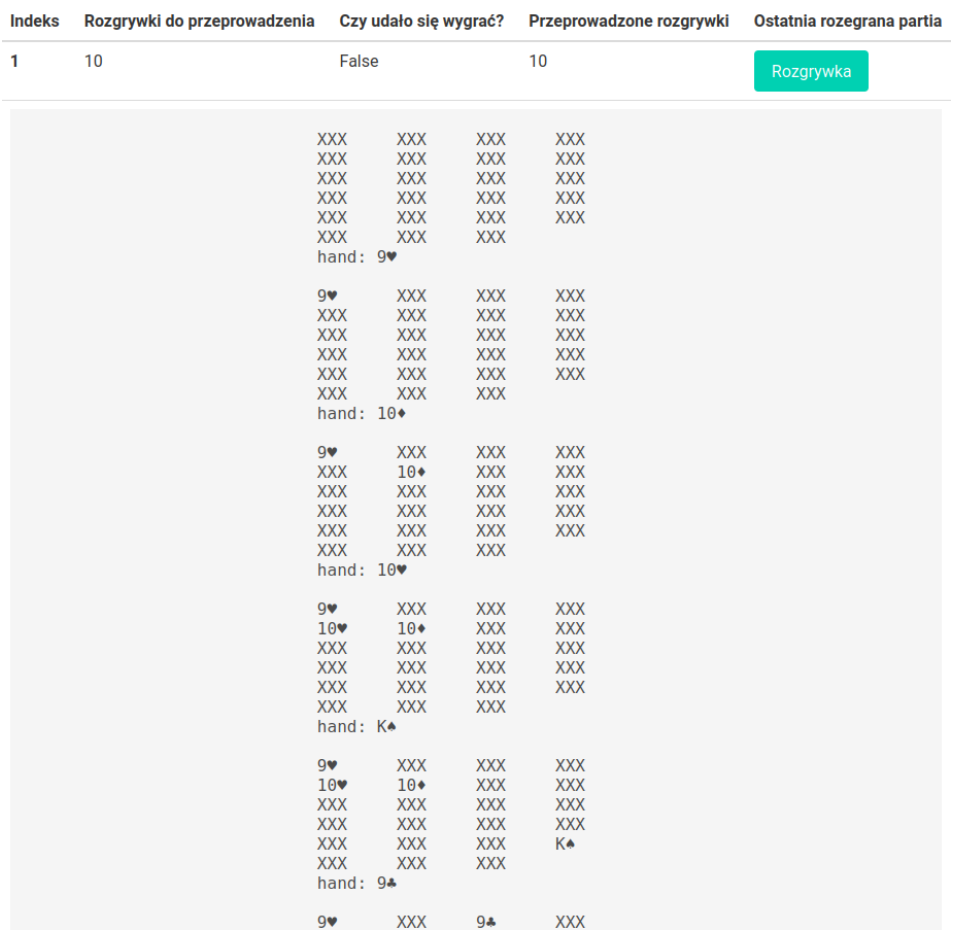
Przy podaniu opcji `-r` lub `--report` w katalogu `/report/` tworzony jest plik `index.html`, który automatycznie otwierany jest w przeglądarce internetowej.



Rysunek 2: Wygenerowany raport

W raporcie prezentowana jest tabela informująca o przebiegu programu dla kolejnych danych wejściowych. W kolumnie `Rozgrywki do przeprowadzenia` widoczne są dane wejściowe dla programu. Następnie program informuje, czy udało się wygrać oraz ile faktycznie rozgrywek przeprowadzono. Ostatnia kolumna zawiera przycisk umożliwiający rozwinięcie wizualizacji ostatniej rozegranej w danym przebiegu partii.

Za każdym razem, gdy nie udało się osiągnąć zwycięstwa, liczba przeprowadzonych partii nie będzie się różniła od ilości partii zadanych do rozegrania. Jednak gdy wykonanie programu zakończyło się wcześniej ze względu na wygraną rozgrywkę, jesteśmy w stanie ustalić, jak szybko to nastąpiło.



Rysunek 3: Widok rozgrywki w raporcie

Reprezentacja rozgrywki zawiera kolejne układy planszy. Każda karta na planszy jest reprezentowana odpowiadającym jej symbolem, lub ciągiem `XXX`, w przypadku, gdy karta jest zasłonięta. Układ pokazuje cztery kolumny po 6 kart, oraz kartę obecnie znajdującą się na ręce. Każda kolejna plansza reprezentuje układ powstały po podmianie karty z ręki z odpowiadającą jej pozycji kartą z planszy. W przypadku, gdy rozgrywka zakończona jest niepowodzeniem, ostatnia plansza nie wskazuje wszystkich odsłoniętych kart, a jedynie te, które udało się odwrócić w czasie gry.

Ostatnią dostępną w programie opcją jest `-b` lub `--backup`, która odpowiada za generowanie kopii zapasowej raportu opatrzonej znacznikiem czasowym. Przykładowy wygenerowany w ten sposób plik może nazywać się:
`backup-2024-01-20T17-58-31-663963.html`

Zawarty w nazwie znacznik czasowy jest reprezentacją obecnej daty oraz czasu w formacie ISO, ze wszystkimi znakami interpunkcyjnymi zastąpionymi myślnikami.

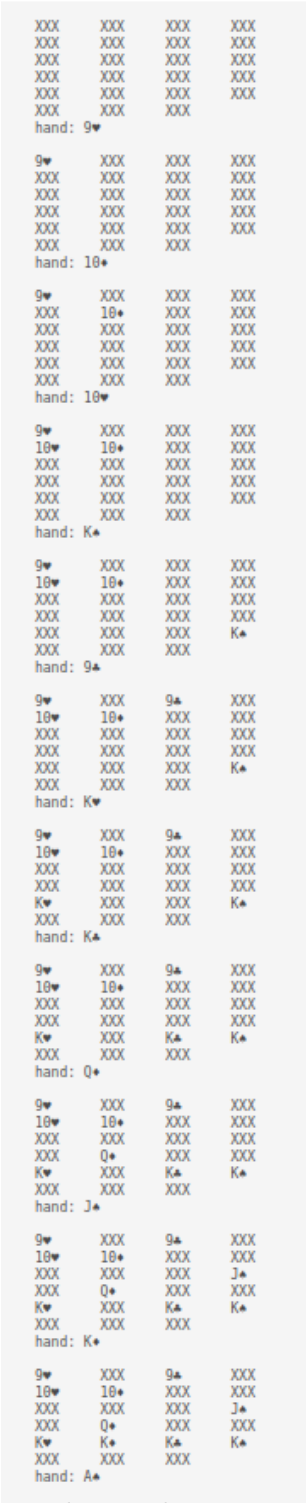
Utworzony w ten sposób plik zawiera również oznaczenie czasowe w nagłówku raportu.

Opcje można dowolnie łączyć, a kolejność ich wykonania zależy od kolejności pojawienia się w linii polecenia.

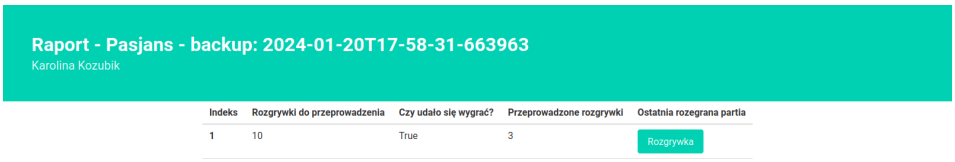
Dodatkowe informacje

Do wygenerowania raportu wykorzystano bibliotekę `jinja2`, która pozwala na wstawianie zmiennych do przygotowanego szablonu z poziomu programu języka Python. Szablon ten znajduje się w pliku `index.html` w katalogu `template`.

Raport korzysta z technologii HTML5, JavaScript oraz Bulma CSS. Napisany w JavaScript skrypt zawarty w pliku raportu umożliwia rozwijanie oraz ukrywanie rozgrywek, co zapewnia jednolitość oraz spójność tabeli i wygodę w przeglądaniu danych. Bulma CSS to pakiet służący do stylowania elementów HTML za pomocą klas, co zapewnia estetyczność raportu.



Rysunek 4: Pełna rozgrywka



Rysunek 5: Znacznik czasowy w nagłówku raportu

Część II

Opis działania

Jak ujęto w treści zadania, zwycięstwo w zaproponowanym pasjansie zależy jedynie od losowego ułożenia talii.

Talia używana w grze składa się z 24 kart. Istnieje 24!, a zatem $6,20448402 \cdot 10^{23}$ sposobów ułożenia takiej talii.

Prawdopodobieństwo zwycięstwa równa się prawdopodobieństwu dobierania kolejnych kart w taki sposób, aby nie był to as pik. W przypadku dobierania pierwszej karty mamy możliwość dobrania jednej z 24 kart, z których 23 nie są asem pik. Następnie musimy dobrać jedną z pozostałych 23 kart, z których 22 nie są asem pik. W ten sposób możemy obliczyć prawdopodobieństwo zwyciężenia rozgrywki:

$$\frac{23}{24} \cdot \frac{22}{23} \cdot \frac{21}{22} \cdot \ldots \cdot \frac{1}{2} = \frac{23!}{24!} \approx 0,04166666667 \tag{1}$$

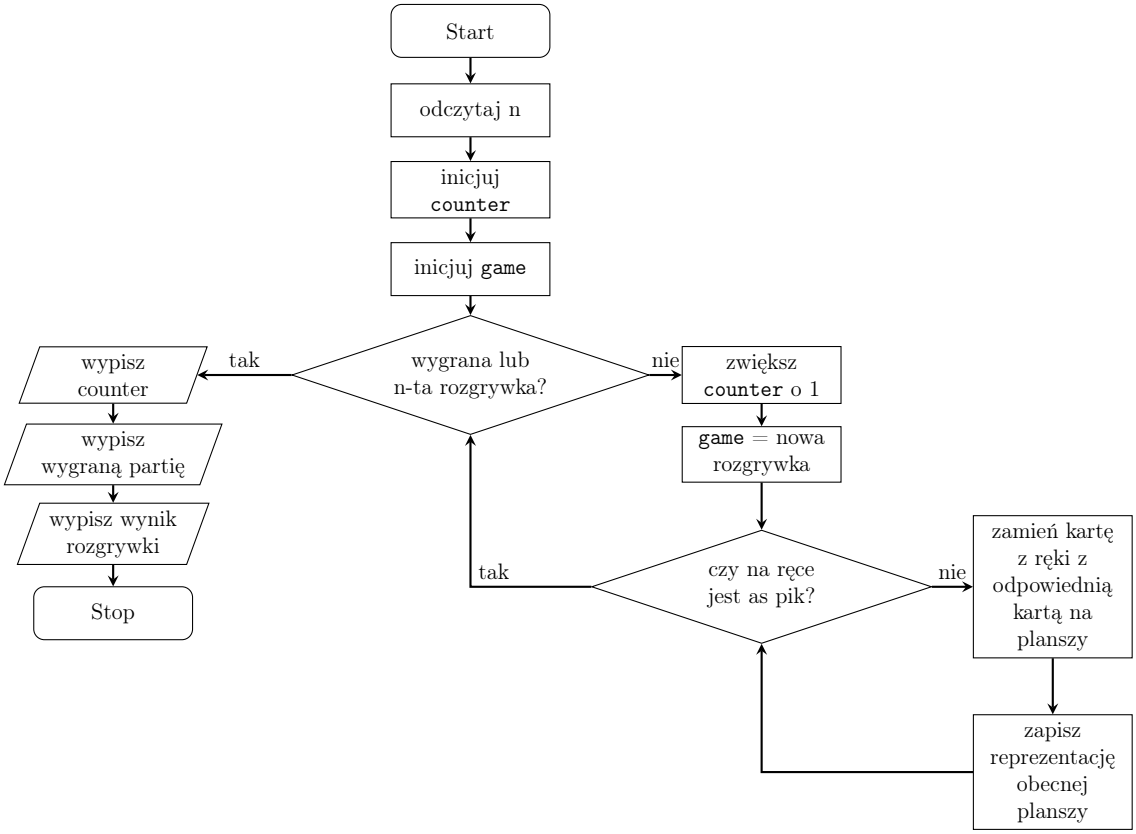
Zatem prawdopodobieństwo wygranej wynosi około 4%.

Algorytmy

Algorytm działa w pętli kończącej się po rozegraniu zadanej ilości rozgrywek lub zwyciężeniu partii. W każdym przebiegu pętli inicjowany jest obiekt klasy `Game`. Inicjalizacja ta pociąga za sobą inicjalizację obiektów klasy `Board`, następnie klasy `Deck`. W czasie inicjalizacji obiektu klasy `Deck` (talii) tworzone są 24 różne karty (obiekty klasy `Card`), układane w sposób losowy. Wewnątrz konstruktora klasy `Board` karty te rozkładane są w dwuwymiarowej tablicy reprezentującej planszę. Po inicjalizacji wszystkich obiektów następuje rozegranie partii.

Rozgrywka odbywa się w pętli zakańczanej, gdy kartą na ręce jest as pik. Wewnątrz pętli karta z ręki podmieniana jest na odpowiadającą jej pozycji kartę z planszy. Następnie w obiekcie zapisywana jest wizualizacji obecnego stanu planszy.

Po zakończeniu rozgrywki ustalany jest jej wynik. Na sam koniec na wyjście standardowe wypisywana jest reprezentacja ostatniej rozegranej partii, ilość przeprowadzonych rozgrywek oraz informacja o tym, czy udało się wygrać.


Start
odczytaj n
inicjuj counter
inicjuj game
wygrana lub n-ta rozgrywka?
tak
nie
wypisz counter
wypisz wygraną partię
wypisz wynik rozgrywki
Stop
zwiększ counter o 1
game = nowa rozgrywka
czy na ręce jest as pik?
tak
nie
zamień kartę z ręki z odpowiednią kartą na planszy
zapisz reprezentację obecnej planszy

Implementacja systemu

Struktura programu wygląda następująco:

```
/
├── main.py
├── app
│   ├── Board.py
│   ├── Card.py
│   ├── Deck.py
│   └── Game.py
├── run.sh
├── in
│   ├── in1.txt
│   ├── in2.txt
│   ├── in3.txt
│   └── ...
├── out
│   ├── out1.txt
│   ├── out2.txt
│   ├── out3.txt
│   └── ...
├── html_report.py
├── report
│   └── index.html
├── template
│   └── index.html
├── backup.py
└── backup
    ├── backup-2024-01-20T17-34-06-838936.html
    ├── backup-2024-01-20T17-35-36-367595.html
    ├── backup-2024-01-20T17-39-57-441331.html
    └── ...
```

Przebiegiem programu steruje plik `run.sh`, który pobiera dane wejściowe z katalogu `/in/`. Główny algorytm programu znajduje się w pliku `main.py`, który korzysta z klas umieszczonych w katalogu `app`. Program otrzymuje dane wejściowe i przekazuje dane wyjściowe, które skrypt `run.sh` zapisuje w katalogu `/out/`. Jeśli uruchomiono program z argumentami wejściowymi, wykonywana jest pętla, która je kolejno przetwarza.

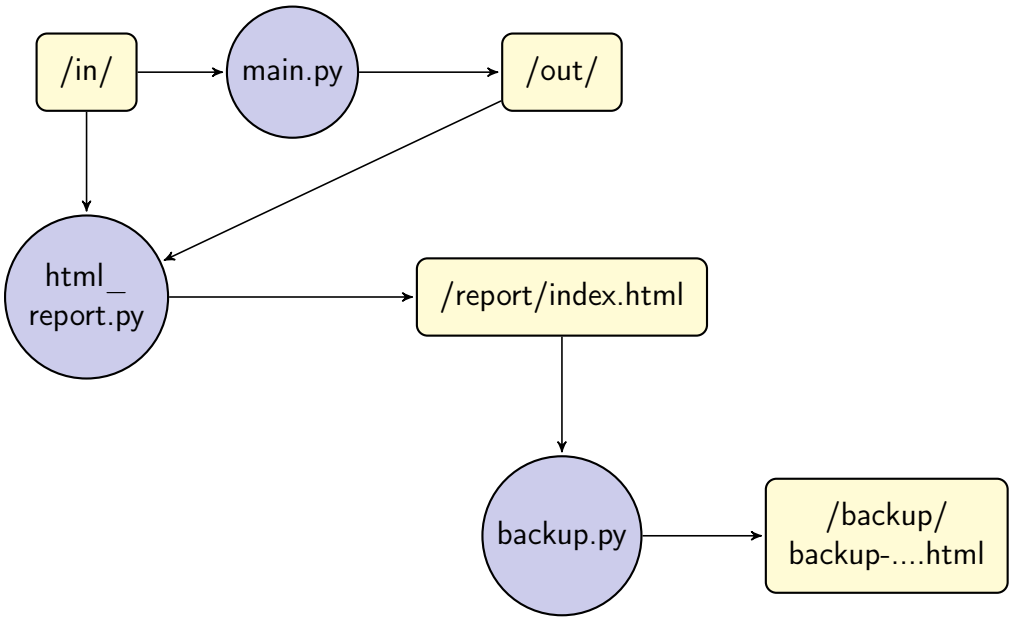
```
while [[ $# -gt 0 ]]; do
    case "$1" in
        -r | --report) python -m html_report ${files[*]} > ./report/
           index.html; firefox ./report/index.html ;;
        -b | --backup) python -m backup ;;
        -h | --help) help ;;
    esac
    shift
done
```

Pętla pobiera kolejne argumenty wejściowe i sprawdza, czy są one równe jednemu z podanych przypadków.

Argument wejścia `-r` lub `--report` powoduje uruchomienie pliku `html_report.py`, któremu przekazywana jest lista ścieżek do plików z zapisanymi danymi wyjściowymi. Program ten odczytuje dane ze wskazanych plików i je przetwarza, następnie przy ich pomocy generuje raport, korzystając z pliku `/template/index.html`. Wygenerowany raport jest przekazywany na wyjście, a następnie zapisywany w pliku `/report/index.html`. Na sam koniec plik ten jest uruchamiany przy pomocy przeglądarki internetowej.

Przy argumencie `-b` lub `--backup` wykonywany jest plik `backup.py`, który odczytuje zawartość pliku `/report/index.html` i generuje jego kopię zapasową, którą zapisuje w katalogu `/backup/` pod odpowiednią nazwą.

Przepływ danych w całej infrastrukturze prezentuje poniższy diagram:



Implementacja algorytmu

Główna pętla algorytmu zawarta jest w pliku `main.py`.

```
game = Game()
counter = 0
while counter < games_num and not game.get_result():
    counter += 1
    game = Game()
    game.run()
print(game)
print(counter)
print(game.get_result())
```

Pętla zlicza swoje kolejne przebiegi, w któych tworzy kolejne obiekty klasy `Game`, na których rozgrywa partię. Na sam koniec program przekazuje odpowiednie dane na wyjście.

Metoda `run` klasy `Game` wygląda następująco:

```
def run(self):
    self.visualization.append(str(self.board))
    while self.board.get_hand() != self.STOP_CARD:
        self.board.replace()
        self.visualization.append(str(self.board))
    self.check_result()
```

Klasa korzysta z tablicy `visualization`, w której przechowuje reprezentacje kolejnych plansz w postaci zmiennej typu `str`.

Rozgrywka wykonywana jest w pętli, w której karta z ręki zamieniana jest na kartę pod odpowiednią pozycją na planszy. Następnie nowa reprezentacja planszy jest zapisywana w klasie. Pętla zakańcza się, jeśli kartą na rękę zostanie as pik.

Podmiana karty z ręki przedstawiona jest w metodzie `replace` klasy `Board`.

```
def replace(self):
    new_hand = self.content[self.hand.suit_idx()][self.hand.value_idx()]
    self.content[self.hand.suit_idx()][self.hand.value_idx()] = self.
        hand
    self.hand = new_hand
```

Klasa `Board` posiada atrybut `content`, który jest tablicą dwuwymiarową, reprezentującą cztery kolumny po sześć kart. Nową kartą na ręce zostaje karta, która w tablicy znajduje się w kolumnie odpowiadającej kolorze obecnej ręki, oraz w rzędzie odpowiadającym wartości obecnej karty na ręce. Następnie tej pozycji zostaje przypisana karta z obecnej ręki, a nowa karta na ręce przypisana jest odpowiedniemu atrybutowi.

Testy

W celu zmierzenia czasu działania algorytmu wykonano 10 000 uruchomień programu, gdzie jako dane wejściowe za każdym razem podano 1. Dzięki temu każdorazowo algorytm był wykonany jednokrotnie, co oznacza, że przeprowadzono jedną rozgrywkę pasjansa.

Przez x oznaczamy czas wykonania programu.

Średnia arytmetyczna czasu wykonania wynosi:

$$\overline{x} = 0.02467981641880251s \tag{2}$$

Mediana czasu wykonania wynosi:

$$\tilde{x} = 0.0241208679999545s \tag{3}$$

Odchylenie standardowe wynosi:

$$\sigma_x = 0.00260178745211724s \tag{4}$$

Pełen kod aplikacji

Plik `main.py`:

```
import sys
from app.Game import Game

if __name__ == "__main__":
    try:
        games_num = int(sys.argv[1])
    except IndexError:
        print("nie podano danych wejsciowych")
    except ValueError:
        print("nieprawidlowe dane wejsciowe")
    game = Game()
    counter = 0
    while counter < games_num and not game.get_result():
        counter += 1
        game = Game()
        game.run()
    print(game)
    print(counter)
    print(game.get_result())
```

Plik `Game.py`:

```
from app.Board import Board
from app.Card import Card


class Game:
    def __init__(self):
        self.board: Board = Board()
        self.STOP_CARD: Card = Card("A", "♠")
        self.visualization = []
        self.is_won = False

    def run(self):
        self.visualization.append(str(self.board))
        while self.board.get_hand() != self.STOP_CARD:
            self.board.replace()
            self.visualization.append(str(self.board))
        self.check_result()

    def get_result(self):
        return self.is_won

    def check_result(self):
        if self.board.is_face_up():
            self.is_won = True

    def __str__(self):
        return "\n".join(self.visualization)
```

Plik Board.py:

```
from app.Deck import Deck
from app.Card import Card


class Board:
    def __init__(self):
        self.content: list[list[Card]] = []
        self.hand: Card = None
        self._place()

    def _place(self):
        deck = Deck()
        rows_counter = 0
        while rows_counter < 4:
            self.content.append([])
            cards_counter = 0
            while cards_counter < 6:
                try:
                    self.content[rows_counter].append(deck.draw())
                    cards_counter += 1
                except ValueError as e:
                    print(e)
            rows_counter += 1
        self.hand = self.content[3].pop()
        self.hand.flip()

    def replace(self):
        new_hand = self.content[self.hand.suit_idx()][self.hand.
            value_idx()]
        self.content[self.hand.suit_idx()][self.hand.value_idx()] = self
            .hand
        self.hand = new_hand
        self.hand.flip()

    def get_hand(self):
        return self.hand

    def is_face_up(self):
        return all(all(card.get_face_up() for card in row) for row in
            self.content)

    def __str__(self):
        string = ""
        for i in range(6):
            row = ""
            for j in range(4):
                if i == 5 and j == 3:
                    continue
                row += str(self.content[j][i]) + "\t"
            row += "\n"
            string += row
        string += "hand: " + str(self.hand) + "\n"
        return string
```

Plik Deck.py:

```
from app.Card import Card


class Deck:
    def __init__(self):
        self.content = []
        self._shuffle()

    def sort(self):
        self.content = sorted(self.content)

    def draw(self):
        if self._is_empty():
            raise ValueError("Cannot draw from empty deck.")
        return self.content.pop()

    def __str__(self):
        return " ".join(str(card) for card in self.content)

    def _is_empty(self):
        return not self.content

    def _shuffle(self):
        self.content = []
        while len(self.content) != 24:
            card = Card.random()
            if card not in self.content:
                self.content.append(card)
```

Plik Card.py:

```
import random


class Card:
    SUIT = ("♡", "♢", "♣", "♠")
    VALUE = ("9", "10", "J", "Q", "K", "A")

    def __init__(self, value, suit):
        if value not in self.VALUE or suit not in self.SUIT:
            raise ValueError("Unrecognized card value or suit.")
        self.value = value
        self.suit = suit
        self.is_up = False

    def flip(self):
        self.is_up = not self.is_up

    def suit_idx(self):
        return self.SUIT.index(self.suit)

    def value_idx(self):
        return self.VALUE.index(self.value)

```

```
    def get_face_up(self):
        return self.is_up

    def __str__(self):
        if not self.is_up:
            return "XXX"
        return self.value + self.suit

    def __eq__(self, other):
        return self.value == other.value and self.suit == other.suit

    def __lt__(self, other):
        if self.suit == other.suit:
            return self.value_idx() > other.value_idx()
        return self.suit_idx() > other.suit_idx()

    @classmethod
    def random(cls):
        return Card(random.choice(cls.VALUE), random.choice(cls.SUIT))
```

Plik `html_report.py`:

```
import sys
from jinja2 import Template


def get_input_data(files_num):
    inputs = []
    for num in range(1, files_num + 1):
        with open(f"./in/in{num}.txt") as f:
            inputs.append(f.read())
    return inputs


def parse(files):
    data = []
    inputs = get_input_data(len(files))
    for idx, file in enumerate(files):
        with open(file) as f:
            content = f.readlines()
        data.append(
            {
                "idx": idx + 1,
                "input": inputs[idx],
                "result": content[-1],
                "counter": content[-2],
                "game": "".join(content[:-2]),
            }
        )
    return data


if __name__ == "__main__":
    try:
        files = sys.argv[1:]
```

```
    except ValueError:
        print("podano bledne dane wejsciowe")

    template = Template(open("./template/index.html").read())
    data = parse(files)

    print(template.render(data=data))
```

Plik backup.py:

```
import datetime
import re

SEP = r"[\-\:\.]"

if __name__ == "__main__":
    timestamp = re.sub(SEP, "-", str(datetime.datetime.now().isoformat()
        ))
    with open("./report/index.html") as file:
        content = file.read()
    content = content.replace(
        '<p class="title">Raport - Pasjans</p>',
        f'<p class="title">Raport - Pasjans - backup: {timestamp}</p>',
    )
    with open(f"./backup/backup-{timestamp}.html", "x") as file:
        file.write(content)
```

Plik run.sh

```
#!/bin/bash

help() {
    echo """Skladnia: run.sh [opcje]

    Dla kazdego pliku w katalogu ./in wykonuje zadanie algorytmiczne,
       ktorego wynik zapisuje w katalogu ./out

    Opcje:
     -r, --report   po ukonczeniu zadania utworz raport html z wynikami
     -b, --backup   dodaj backup ostatniego wygenerowanego raportu
                    z obecnym timestampem
     -h, --help     wyswietla pomoc"""
}

wd=$(dirname $0);

files=()

for file in "./in/*"; do
    file_number=${file#./in/in}
    file_name=./out/out$file_number
    games_counter=$(cat $file)
    python -m main $games_counter > $file_name
    files+=($file_name)
done

```

```
27 while [[ $# -gt 0 ]]; do
28     case "$1" in
29         -r | --report) python -m html_report ${files[*]} > ./report/
               index.html; firefox ./report/index.html ;;
30         -b | --backup) python -m backup ;;
31         -h | --help) help ;;
32     esac
33     shift
34 done
```

Literatura

[1] dr inż. Mariusz Pleszczyński. Zadanie 5 – pasjans (zadanie finałowe edycji 2014/15). https://algorytmion.ms.polsl.pl/storage/files/Zadania2016.pdf.