Sprawozdanie z projektu Programowanie III

Karolina Kozubik

Styczeń 2024

1 Temat projektu

Tematem projektu jest zrealizowanie wybranego zadania konkursowego w języku Java. Zadanie pochodzi z konkursu Algorytmion, edycja 2016. 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]

2 Wejście

Program nie przyjmuje danych wejściowych. Implementuje utworzenie talii składającej się z 24 kart w sposób losowy.

3 Wyjście

Na wyjściu program podaje liczbę partii rozegranych aż do uzyskania partii zwycięskiej, oraz przebieg ostatniej partii.

Przykładowe wyjście:

```
liczba partii: 6
XXX     XXX     XXX     XXX
XXX     XXX     XXX     XXX
XXX     XXX     XXX     XXX
XXX     XXX     XXX     XXX
XXX     XXX     XXX     XXX
XXX     XXX     XXX
hand: Q♡

XXX     XXX     XXX     XXX
XXX     XXX     XXX     XXX
XXX     XXX     XXX     XXX
Q♡      XXX     XXX     XXX
XXX     XXX     XXX     XXX
XXX     XXX     XXX
hand: 9♣

XXX     XXX     9♣      XXX
XXX     XXX     XXX     XXX
XXX     XXX     XXX     XXX
Q♡      XXX     XXX     XXX
XXX     XXX     XXX     XXX
XXX     XXX     XXX
hand: 10♡

XXX     XXX     9♣      XXX
10♡     XXX     XXX     XXX
XXX     XXX     XXX     XXX
Q♡      XXX     XXX     XXX
XXX     XXX     XXX     XXX
XXX     XXX     XXX
hand: A♣

XXX     XXX     9♣      XXX
10♡     XXX     XXX     XXX
XXX     XXX     XXX     XXX
Q♡      XXX     XXX     XXX
XXX     XXX     XXX     XXX
XXX     XXX     A♣
hand: A♢

XXX     XXX     9♣      XXX
10♡     XXX     XXX     XXX
XXX     XXX     XXX     XXX
Q♡      XXX     XXX     XXX
XXX     XXX     XXX     XXX
XXX     A♢      A♣
hand: Q♣

XXX     XXX     9♣      XXX
10♡     XXX     XXX     XXX
XXX     XXX     XXX     XXX
Q♡      XXX     Q♣      XXX
XXX     XXX     XXX     XXX
XXX     A♢      A♣
hand: J♠

XXX     XXX     9♣      XXX
10♡     XXX     XXX     XXX
XXX     XXX     XXX     J♠
Q♡      XXX     Q♣      XXX
XXX     XXX     XXX     XXX
XXX     A♢      A♣
hand: J♣

XXX     XXX     9♣      XXX
10♡     XXX     XXX     XXX
XXX     XXX     J♣      J♠
Q♡      XXX     Q♣      XXX
XXX     XXX     XXX     XXX
XXX     A♢      A♣
hand: K♣

XXX     XXX     9♣      XXX
10♡     XXX     XXX     XXX
XXX     XXX     J♣      J♠
Q♡      XXX     Q♣      XXX
XXX     XXX     K♣      XXX
XXX     A♢      A♣
hand: 9♢
```

```
XXX     9♢      9♣      XXX
10♡     XXX     XXX     XXX
XXX     XXX     J♣      J♠
Q♡      XXX     Q♣      XXX
XXX     XXX     K♣      XXX
XXX     A♢      A♣
hand: 9♡
```

```
9♡      9♢      9♣      XXX
10♡     XXX     XXX     XXX
XXX     XXX     J♣      J♠
Q♡      XXX     Q♣      XXX
XXX     XXX     K♣      XXX
XXX     A♢      A♣
hand: J♢
```

```
9♡      9♢      9♣      XXX
10♡     XXX     XXX     XXX
XXX     J♢      J♣      J♠
Q♡      XXX     Q♣      XXX
XXX     XXX     K♣      XXX
XXX     A♢      A♣
hand: 10♣
```

```
9♡      9♢      9♣      XXX
10♡     XXX     10♣     XXX
XXX     J♢      J♣      J♠
Q♡      XXX     Q♣      XXX
XXX     XXX     K♣      XXX
XXX     A♢      A♣
hand: K♡
```

```
9♡      9♢      9♣      XXX
10♡     XXX     10♣     XXX
XXX     J♢      J♣      J♠
Q♡      XXX     Q♣      XXX
K♡      XXX     K♣      XXX
XXX     A♢      A♣
hand: 10♢
```

```
9♡      9♢      9♣      XXX
10♡     10♢     10♣     XXX
XXX     J♢      J♣      J♠
Q♡      XXX     Q♣      XXX
```

```
K♡      XXX     K♣      XXX
XXX     A♢      A♣
hand: Q♢
```

```
9♡      9♢      9♣      XXX
10♡     10♢     10♣     XXX
XXX     J♢      J♣      J♠
Q♡      Q♢      Q♣      XXX
K♡      XXX     K♣      XXX
XXX     A♢      A♣
hand: 10♠
```

```
9♡      9♢      9♣      XXX
10♡     10♢     10♣     10♠
XXX     J♢      J♣      J♠
Q♡      Q♢      Q♣      XXX
K♡      XXX     K♣      XXX
XXX     A♢      A♣
hand: A♡
```

```
9♡      9♢      9♣      XXX
10♡     10♢     10♣     10♠
XXX     J♢      J♣      J♠
Q♡      Q♢      Q♣      XXX
K♡      XXX     K♣      XXX
A♡      A♢      A♣
hand: K♠
```

```
9♡      9♢      9♣      XXX
10♡     10♢     10♣     10♠
XXX     J♢      J♣      J♠
Q♡      Q♢      Q♣      XXX
K♡      XXX     K♣      K♠
A♡      A♢      A♣
hand: J♡
```

```
9♡      9♢      9♣      XXX
10♡     10♢     10♣     10♠
J♡      J♢      J♣      J♠
Q♡      Q♢      Q♣      XXX
K♡      XXX     K♣      K♠
A♡      A♢      A♣
hand: K♢
```

```
9♡      9♢      9♣      XXX
10♡     10♢     10♣     10♠
```

```
J♡      J♢      J♣      J♠
Q♡      Q♢      Q♣      XXX
K♡      K♢      K♣      K♠
A♡      A♢      A♣
hand: Q♠

9♡      9♢      9♣      XXX
10♡     10♢     10♣     10♠
J♡      J♢      J♣      J♠
Q♡      Q♢      Q♣      Q♠
```

```
K♡      K♢      K♣      K♠
A♡      A♢      A♣
hand: 9♠

9♡      9♢      9♣      9♠
10♡     10♢     10♣     10♠
J♡      J♢      J♣      J♠
Q♡      Q♢      Q♣      Q♠
K♡      K♢      K♣      K♠
A♡      A♢      A♣
hand: A♠
```

Pierwsza linia wyjścia informuje o liczbie rozegranych partii. Następnie wyświetlane są kolejne układy planszy, rozpoczynając od ułożenia startowego.

Każdy widok planszy składa się z układu czterech rzędów, zawierających po sześć kart. Karty zasłonięte są widoczne jako `XXX`. Odsłonięte karty reprezentowane są przez ich wartość (`9`, `10`, `J`, `Q`, `K` lub `A`) oraz kolor (♡, ♢, ♣ lub ♠). Pod rzędami kart, po słowie `hand:`, widoczna jest karta trzymana aktualnie na ręce.

Kolejne plansze reprezentują ułożenia kart po podmianie karty z ręki na odpowiadającą jej miejcu kartę z planszy.

4 Algorytm rozwiązania

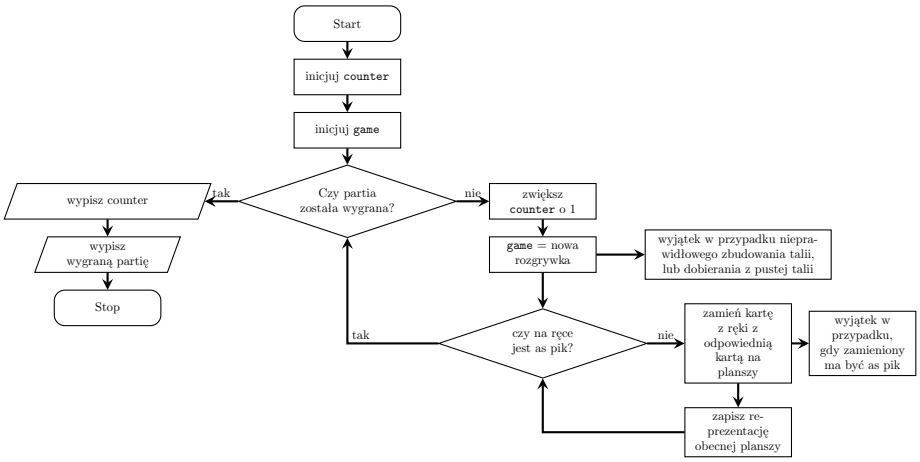
4.1 Opis

Główny algorytm rozwiązania znajduje się w metodzie `int main()` klasy `App`. Inicjowana jest zmienna `int counter`, z wartością 0. Następnie tworzony jest obiekt klasy `Game`. W pętli `while` sprawdzane jest, czy rozegrana partia była zwycięska. Jeśli nie, wartość zmiennej `counter` zostaje zwiększona o 1, a następnie tworzony jest nowy obiekt klasy `Game`, na którym rozgrywana jest partia. Po zakończeniu pętli `while`, na wyjście przekazywany jest komunikat informujący o ilości rozegranych partii, oraz przebieg zwycięskiej rozgrywki.

Każda z rozgrywek rozpoczyna się od inicjalizacji planszy (obiektu klasy `Board`). Inicjalizacja planszy polega na utworzeniu obiektu klasy `Deck`, który zawiera kolekcję 24 losowo ułożonych kart. W trakcie inicjalizacji planszy tworzona jest dwuwymiarowa lista reprezentująca ułożenie kart (kolejno pobieranych z talii) w poszczególnych rzędach. Ostatnia z kart jest dobierana na rękę.

Główna pętla rozgrywki (metoda `void run()` w klasie `Game`) przebiega, dopóki kartą na ręce nie jest as pik. Znajdowana jest pozycja na planszy odpowiadająca karcie na ręce (dzięki wartościom numerycznym przypisanym do wartości oraz koloru karty). Karta na znalezionej pozycji dobierana jest do ręki, a na planszy zastępowana kartą, która poprzednio znajdowała się na ręce. W obiekcie zapisywana jest reprezentacja układu na planszy po zamianie kart.

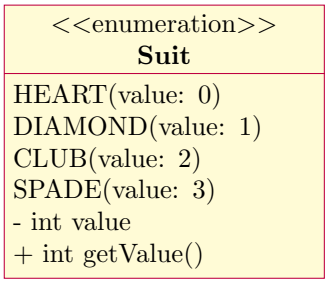
4.2 Schemat blokowy



4.3 Metody i klasy w programie

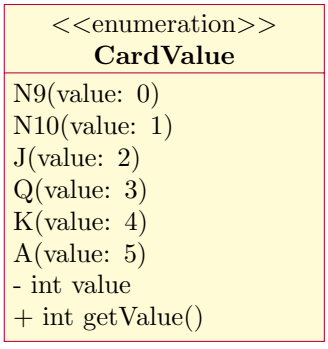
enum Suit

`Suit` reprezentuje kolor karty. Każdemu z jego wariantów przypisana jest wartość liczbowa, która odpowiada indeksowi rzędu na planszy (licząc od 0). Wartość przechowywana jest w atrybucie `value`. Enumerator ma metodę `int getValue()`, która zwraca wartość atrybutu `value`.



enum CardValue

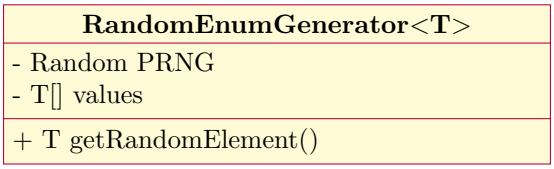
`CardValue` reprezentuje wartość karty. Każdemu z jego wariantów przypisana jest wartość liczbowa, która odpowiada indeksowi karty w rzędzie na planszy (licząc od 0). Wartość przechowywana jest w atrybucie `value`. Enumerator ma metodę `int getValue()`, która zwraca wartość atrybutu `value`.



class RandomEnumGenerator

Klasa `RandomEnumGenerator<T>` jest klasą generyczną dla typu, który jest enumeratorem. Umożliwia wybór losowego wariantu podanego enumeratora. Posiada atrybut `Random PRNG`, który przechowuje obiekt

klasy `Random` z pakietu `java.util.Random`,
oraz atrybut `T[] values`, który jest tablicą wariantów obsługiwanego enumeratora. W konstruktorze atrybutowi `T[] values` przypisywane są wartości wariantów enumeratora. Klasa posiada metodę `T getRandomElement()`, który zwraca losowy element tablicy `T[] values`.



class Card

`Card` to klasa reprezentująca pojedynczą kartę w grze. Jej atrybuty to:

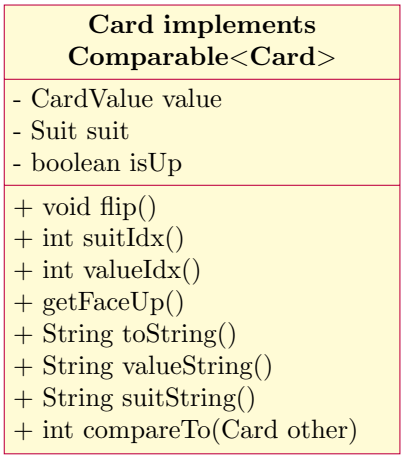
`CardValue value` - wartość karty.

`Suit suit` - kolor karty.

`boolean isUp` - wartość określająca, czy karta jest odwrócona górą.

Klasa posiada metody:

Konstruktor klasy `Card(CardValue value, Suit suit)` przypisuje podane w argumentach wartości odpowiednim atrybutom klasy. Dodatkowo, atrybutowi `isUp` przypisywana jest wartość domyślna `false`.



`void flip()` zmienia wartość atrybutu `boolean isUp` na przeciwną, reprezentując odwrócenie karty na drugą stronę.

`int suitIdx()` zwraca wartość numeryczną przypisaną do koloru karty, która jest jednocześnie indeksem rzędu na planszy.

`int valueIdx()` zwraca wartość numeryczną przypisaną do wartości karty, która jest jednocześnie indeksem karty w rzędzie na planszy.

`boolean getFaceUp()` zwraca wartość atrybutu `boolean isUp`.

`String toString()` zwraca reprezentację karty w postaci zmiennej typu `String`.

`String valueString()` zwraca reprezentację wartości karty w postaci zmiennej typu `String`. Metoda rzuca wyjątek, jeśli wartość karty nie zostaje rozpoznana.

`String suitString()` zwraca reprezentację koloru karty w posraci zmiennej typu `String`. Metoda rzuca wyjątek, jeśli kolor karty nie zostaje rozpoznany.

`int compareTo(Card other)` to metoda porównująca kartę do innej karty w oparciu o ich kolory i wartości. Dla takich samych kolorów, porządek wartości

to 9, 10, walet, królowa, król, as. W innym przypadku porządek oparty jest o kolor kart - w kolejności kier, karo, trefl, pik.

class Deck

Deck
- ArrayList<Card> content - RandomEnumGenerator<CardValue> valueGenerator - RandomEnumGenerator<Suit> suitGenerator
+ void shuffle() + void sort() + Card draw() + boolean isEmpty() + String toString()

Klasa `Deck` reprezentuje talię kart.

`ArrayList<Card> content` zawiera listę kart w talii.

`RandomEnumGenerator<CardValue> valueGenerator` to generator losowych wartości kart.

`RandomEnumGenerator<Suit> suitGenerator` to generator losowych kolorów kart.

Klasa posiada następujące metody:

Konstruktor `Deck()` przypisuje atrybutom `valueGenerator` oraz `suitGenerator` nowo utworzone obiekty. Ponadto wywołuje metodę `void shuffle()`, która inicjuje listę kart w talii.

`void shuffle()` tworzy talię kart. Metoda tworzy kolejne karty w sposób losowy, a następnie sprawdza, czy takiej karty nie ma jeszcze w liście `ArrayList<Card> content`. W takim przypadku karta jest dodawana do talii. Proces jest powtarzany, aż talia nie osiągnie rozmiaru 24 kart.

`void sort()` sortuje listę przechowywaną w atrybucie `ArrayList<Card> content`.

`Card draw()` służy do dobierania kart z talii. Jeśli lista `ArrayList<Card> content` jest pusta, metoda zwraca wyjątek. W innym przypadku zwracany jest ostatni element tej listy, który jest również z niej usuwany. Metoda rzuca wyjątek, jeśli jest wywołana w sytuacji, gdy talia jest pusta.

`boolean isEmpty()` zwraca wartość `true`, jeśli lista `ArrayList<Card> content` jest pusta.

`String toString()` zwraca reprezentację talii w postaci zmiennej typu `String`.

class Board

Board
- ArrayList<ArrayList<Card>> content - Card hand
+ void replace() + Card getHand() + boolean isFaceUp() + String toString()

Klasa `Board` reprezentuje planszę. Posiada następujące atrybuty:

`ArrayList<ArrayList<Card>> content` to dwuwymiarowa lista zawierająca układ planszy.

`Card hand` przechowuje kartę, która aktualnie znajduje się w ręce gracza.

Klasa posiada poniższe metody:

Konstruktor `Board()` tworzy obiekt klasy `Deck`, a następnie pobierając z niego kolejne karty, tworzy dwuwymiarową listę (zewnętrzna lista o rozmiarze 4, wewnętrzne listy o rozmiarze 6). Następnie usuwany z listy jest ostatni element ostatniej listy wewnętrznej, który zostaje przypisany atrybutowi `Card hand`. Karta trzymana w ręce zostaje odwrócona.

`void replace()` reprezentuje podmianę karty na ręce na odpowiadającą jej miejscu kartę z planszy. W pierwszej kolejności zmiennej `Card newHand` przypisywana jest karta pobrana z listy, z pozycji odpowiadającej karcie przechowywanej w atrybucie `Card hand`. Następnie, na tą samą pozycję, wstawiana jest karta obecnie trzymana na ręce. Na sam koniec wartość zmiennej `Card newHand` jest przypisywana atrybutowi `Card hand`, karta ta jest również odwracana. Metoda rzuca wyjątek, jeśli kartą na ręce jest as pik.

`Card getHand()` to metoda, która zwraca kartę obecnie trzymaną na ręce.

`boolean isFaceUp()` zwraca wartość `true`, jeśli wszyskie karty w planszy są odwrócone górą na wierzch.

`String toString()` zwraca reprezentację planszy w postaci zmiennej typu `String`. Zapisywane są cztery rzędy zawierające reprezentacje poszczególnych kart, pod którymi, po napisie `hand:`, widoczna jest karta obecnie trzymana na ręce.

class Game

Klasa `Game` reprezentuje pojedynczą rozgrywkę pasjansa.

Atrybut stały `Card STOP_CARD` przechowuje kartę as pik, której dobranie na rękę oznacza koniec rozgrywki.

`Board board` to atrybut przechowujący planszę.

`ArrayString<String> visualization` przechowuje listę re-

Game
- Card STOP_CARD - Board board - ArrayList<String> visualization - boolean isWon
+ void run() + boolean getResult() + void checkResult() + String toString()

prezentacji kolejnych układów kart na planszy.

`boolean isWon` to wartość przechowująca wynik rozgrywki.

Klasa posiada metody:

`void run()` rozgrywa partię. Dopóki kartą na ręce nie jest as pik, karta z ręki jest podmieniania na odpowiadającą jej pozycji kartę z planszy. Przy każdej podmianie do `ArrayList<String> visualization` dodawana jest reprezentacja planszy w postaci zmiennej typu `String`. Na koniec sprawdzany jest wynik rozgrywki za pomocą metody `void checkResult()`.

`boolean getResult()` zwraca wartość atrybutu `boolean isWon()`.

`void checkResult()` zmienia wartość atrybutut `boolean isWon`, jeśli wszystkie karty obecne na planszy są odkryte.

`String toString()` zwraca reprezentację rozgrywki z postaci zmiennej typu `String`. Zapisywane są reprezentacje kolejnych plansz rozdzielone znakiem nowej linii.

class App

Klasa `App` jest główną klasą programu, posiadającą jedynie metodę `void main()`. Metoda ta inicjuje `int counter` wartością 0. Następnie rozgrywane są poszczególne partie, po każdej zwiększana o 1 jest wartość zmiennej `counter`. Partie są rozgrywane aż do uzyskania partii zwycięskiej, po czym na wyjście wypisywana jest liczba rozegranych partii oraz wizualizacja ostatniej rozgrywki.

5 Testy programu

W celu przetestowania programu sprawdzam wyniki dla 100 kolejnych przebiegów algorytmu. W kolejnych przebiegach liczba rozegranych partii wynosiła: 9, 2, 28, 5, 54, 26, 11, 52, 40, 5, 22, 30, 2, 51, 11, 8, 2, 9, 3, 4, 1, 20, 13, 29, 54, 75, 14, 29, 1, 31, 15, 15, 10, 11, 31, 26, 12, 1, 2, 1, 45, 22, 31, 51, 4, 3, 35, 19, 65, 11, 5, 50, 3, 30, 10, 38, 12, 1, 30, 36, 9, 96, 45, 38, 57, 11, 30, 27, 96, 35, 4, 18, 4, 9, 84, 32, 6, 64, 11, 10, 8, 13, 39, 15, 5, 60, 13, 22, 2, 36, 15, 81, 3, 48, 14, 9, 46, 19, 12, 16. Brak powtarzających się zestawów liczb potwierdza losowość utworzonych talii.

W trakcie przebiegu programu walidowana jest zawartość ręki oraz zawartość talii, także nie dochodzi do rzucania wyjątków. Jednakże możliwe jest sprawdzenie tychże walidacji poprzez zasymulowanie utworzenia nieprawidłowej talii.

Przy zasymulowaniu pustej talii na ekran zostaje wypisany `java.lang.Exception: Cannot draw from the empty deck.`

Przy usunięciu warunku zapewniającego zakończenie rozgrywki po dobraniu asa pik na rękę, na ekran zostaje wypisany wyjątek `java.lang.Exception: Cannot replace the ace of spades from hand.`

6 Wnioski

Program podaje duże rozbieżności w danych wyjściowych, co spowodowane jest losowym sposobem generowania talii. Metoda generująca talię jest najmniej optymalnym elementem programu - w pętli `while` tworzona jest nowa karta, następnie sprawdza się, czy karta o tej samej wartości i kolorze nie znajduje się już w talii, w którym to razie proces jest powtarzany.

Reprezentacja przebiegu rozgrywki stosuje uniwersalną symbolikę dotyczącą kart w standardowych taliach, co sprawia, że jest ona czytelna dla użytkownika.

W procesie tworzenia programowania zastosowano system kontroli wersji git.

Literatura

[1] dr inż. Mariusz Pleszczyński. Zadanie 5 - pasjans (zadanie finałowe edycji 2014/2015). https://algorytmion.ms.polsl.pl/storage/files/Zadania2016.pdf.