Politechnika Krakowska

Wydział Fizyki, Matematyki i Informatyki

ZADANIE 5

Zaawansowane techniki programowania

Prowadzący: dr inż. Jerzy Jaworowski

II stopień - rok 1, semestr 1

wykonanie:

Piotr Adam Tomaszewski

Nr albumu: 104896

Wstęp

Celem zadania było zaimplementowanie gry o nazwie MasterMind. Na samym początku należało utworzyć ‘połączenie’ z metodą IGameMonitor a następnie zarejestrowanie się przez wywołanie metody register. Jeśli przebiegła ona pomyślnie, to należy pobrać przekazane w żądaniu (url) parametry n, k i s jako informację przeznaczoną do zainicjowania rozgrywki. W celu zainicjowania rozgrywki servlet przekazuje dane do komponentu menadżera gry (metoda initGame udostępniona interfejsem IGameMonitor komponentu GameMonitor). Korzystając z wykonanej i udostępnionej (interfejs IMasterMind) metody komponentu MasterMind realizującej proces rozgrywki servlet ustala i wyprowadza prawidłowo obliczoną ilość wykonanych kroków (ruchów) algorytmu gry.

Organizacja plików

IGameMonitor.java

Interfejs ***IGameMonitor*** posiadający deklarację metod, która pozwalają na zarejestrowanie użytkownika w systemie, rozpoczęcie rozgrywki i zweryfikowania rozwiązania zaproponowanego przez algorytm

Deklaracja metody ***register*** rejestrująca użytkownika w systemie - zwraca true jeżeli proces rejestracji zakończył się poprawnie. Jeżeli rejestracja zakończyła się niepowodzeniem, metoda register zwraca wartość false.

@param hwork - numer zadania

@param album – numer albumu studenta

@return wartość logiczna czy połączenie przebiegło pomyślnie

```
public boolean register(int hwork, String album);
```

Rysunek 1 Deklaracja metody register

Deklaracja metody ***initGame*** umożliwia zainicjowanie gry. Ustala poszukiwane przez gracza w kolejnych ruchach początkowe ustawienie pionków.

@param n - ilość kolorów

@param k - ilość kolumn

@param seed - parametr niezbędny do zainicjowania gry

```
public void initGame(int n, int k, long seed);
```

Rysunek 2 Deklaracja metody initGame

Deklaracja metody ***verify*** sprawdza aktualne ustawienie kolorów przez gracza z kolorami ustalonymi podczas inicjowania gry. Zwraca dwucyfrową informację.

@param state - aktualne ustawienie kolorów przez gracza

@return - dwie cyfry: pierwsza z nich określa ilość pionów o właściwym kolorze ustawionych we właściwej kolumnie, drugi z nich określa ilość pionów o właściwym kolorze lecz ustawionych w niewłaściwej kolumnie.

```
public String verify(String state);
```

Rysunek 3 Deklaracja metody verify

IMasterMind.java

Definicja interfejsu komponentu zdefiniowanego w pliku MasterMind.java

Deklaracja metody ***masterMind*** określa kolejność realizacji procesu rozgrywki i zwraca informację o ilości wykonanych kroków.

@return ilość wykonanych weryfikacji niezbędnych do znalezienia rozwiązania

```
public int masterMind();
```

Rysunek 4 Deklaracja metody masterMind

Deklaracja metody ***start*** jest metodą startową uruchamianą z servletu MGame pobierającą parametry.

@param n - ilość kolorów

@param k - ilość kolumn

@param seed - parametr niezbędny do zainicjowania gry

@param game - połączenie dzięki któremu możemy wywoływać metody z komponentu GameMonitor

@return ilość wykonanych weryfikacji niezbędnych do znalezienia rozwiązania

```
public int start(int n, int k, long seed, pl.jrj.game.IGameMonitor game);
```

Rysunek 5 Deklaracja metody start

MasterMind.java

Klasa ***MasterMind*** implementująca interfejs IMasterMind. Zawiera implementację algorytmu to szybkiego znajdywania rozwiązania gry.

```
public class MasterMind implements IMasterMind{
    int n;
    int k;
    long seed;
    char[] znaki;
    int [] gdziePoprawne;
    int interacja=0;
    pl.jrj.game.IGameMonitor game;
    Random r = new Random();
```

Rysunek 6 Deklaracja klasy MasterMind

Metoda ***pierwszeVerify*** pozwala na wypełnienie tablicy losowymi kolorami i sprawdzaniu czy pierwsza liczba po weryfikacji (liczba poprawnych kolorów znajdujących się na swoim miejscu) jest większa od drugiej (liczby kolorów poprawnych ale nie znajdujących się na swoim miejscu).

@param weryf - deklaracja zmiennej przechowującej informację z metody verify

@param x - deklaracja zmiennej przechowującej liczbę poprawnych kolorów w poprawnych miejscach

@param xx - deklaracja zmiennej przechowującej liczbę poprawnych kolorów w niepoprawnych miejscach

@return - liczba poprawnych kolorów w poprawnych miejscach

```
private int pierwszeVerify(String weryf, int x, int xx){
    do{
        wypelnijTab();
        weryf=game.verify(naString());
        x= Character.getNumericValue(weryf.charAt(0));
        xx= Character.getNumericValue(weryf.charAt(1));
    }while(x<=xx);
    return x;
}
```

Rysunek 7 Implementacja metody pierwszeVerify

Metoda ***wypelnijTab*** wypełniająca tablicę kolorami z przedziału od A do A+n

```
private void wypelnijTab(){
    for (int i=0;i<this.k;i++){
        this.znaki[i]=(char)(r.nextInt(this.n+1)+65);
        this.gdziePoprawne[i]=0;
    }
}
```

Rysunek 8 Implementacja metody wypelnijTab

Metoda ***naString*** zamieniająca elementy tablicy na wartość tekstową

@return ciąg wyrazów tablicy przedstawiony w tekstowej

```
private String naString(){
    String z="" ;
    for (int i=0;i<this.k;i++){
        z+=znaki[i];
    }
    return z;
}
```

Rysunek 9 Implementacja metody naString

Metoda ***gdziePoprawne*** która sprawdza czy w tablicy o konkretnym indeksie kolor jest poprawny czy nie. Jeśli kolor jest poprawny, to zapisuje to tablicy gdziePoprawne, o odpowiednim indeksie, wartość 1.

@param x - stara wartość pierwszej liczby po poprzedniej weryfikacji

@param stary - zmienna pomocnicza przechowująca kolor tablicy przed zmianą

@param weryf - zmienna pomocnicza przechowująca wynik weryfikacji.

```
private void gdziePoprawne(int x, char stary, String weryf){
    for (int i=0;i<this.k;i++){
        stary = this.znaki[i];
        this.znaki[i]='0';
        weryf=game.verify(naString());
        if (Character.getNumericValue(weryf.charAt(0))<x){
            this.gdziePoprawne[i]=1;
            this.znaki[i]=stary;
        }
        interacja++;

    }
}
```

Rysunek 10 Implementacja metody gdziePoprawne

Metoda ***znajdzRozwiazanie*** sprawdza kolejne ułożenia kolorów w tablicy o tych indeksach, w których tablica gdziePoprawne nie przyjmuje wartości 1.

@param weryf - deklaracja zmiennej przechowującej informację z metody verify

@param x - liczbę poprawnych kolorów w poprawnych miejscach po pierwszej weryfikacji

@param i - deklaracja zmiennej będącej iteratorem pętli

@param j - deklaracja zmiennej będącej iteratorem pętli

```
private void znajdzRozwiazanie(String weryf, int x, int i, int j){
    for (i=0;i<this.k;i++){

        if (this.gdziePoprawne[i]!=1){
            for (j=65;j<(65+this.n);j++){
                this.znaki[i]=(char)j;
                weryf=game.verify(naString());
                interacja++;
                if (x<Character.getNumericValue(weryf.charAt(0))){
                    x=Character.getNumericValue(weryf.charAt(0));
                    break;
                }
            }
        }
    }
}
```

Rysunek 11 Implementacja metody znajdzRozwiazania

Metoda ***masterMind*** zaimplementowana z interfejsu IMasreMind określa kolejność realizacji procesu rozgrywki i zwraca informację o ilości wykonanych kroków.

@return ilość wykonanych weryfikacji niezbędnych do znalezienia rozwiązania

```
@Override
public int masterMind(){
    interacja=0;
    znaki = new char[this.k];
    gdziePoprawne = new int[this.k];
    int x;

    game.initGame(this.n, this.k, this.seed);
    x=pierwszeVerify("",0,0);
    gdziePoprawne(x,'0',"0");

    znajdzRozwiazanie("",x,0,0);

return interacja;
}
```

Rysunek 12 Implementacja metody masterMind

Metoda ***start*** zaimplementowana z interfejsu IMasreMind jest metodą startową uruchamianą z servletu MGame pobierającą parametry.

@param n - ilość kolorów

@param k - ilość kolumn

@param seed - parametr niezbędny do zainicjowania gry

@param game - połączenie dzięki któremu możemy wywoływać metody z komponentu GameMonitor

@return ilość wykonanych weryfikacji niezbędnych do znalezienia rozwiązania

```java
@Override
public int start(int n, int k, long seed, pl.jrj.game.IGameMonitor game) {
    this.n=(n-1);
    this.k=k;
    this.seed=seed;
    this.game=game;

    return masterMind();
}
```

Rysunek 13 Implementacja metody start

MGame.java

Klasa ***servletu*** w której odbywa się sterowanie procesu rozgrywki. Metoda IGameMonitor, dzięki której po przekazaniu parametru będącym adresem połączenia, następuje wyszukiwanie paczki ejb-project i jeśli go znajdzie to następuje szukanie klasy GameMonitor i wykonuje działanie metod poprzez przesłonięcie interfejsem.

@param connect - adres do wyszukania i połączenia się z interfejsem

@return połączenie dzięki któremu możemy wywoływać metody z komponentu GameMonitor

```java
private pl.jrj.game.IGameMonitor getConnection(String connect){
    pl.jrj.game.IGameMonitor game = null;
    try {

        InitialContext con = new InitialContext();
        game = (pl.jrj.game.IGameMonitor) con.lookup(connect);
    } catch (NamingException ex ) {
        game=null;
    }
    return game;
}
```

Rysunek 14 Implementacja metody getConnection

Metoda ***doGet***, dzięki której po przekazaniu parametru w żądaniu URL, następuje rejestracja użytkownika i jeśli przebiegnie pomyślnie wykonywane są metody obliczające ilość kroków potrzebnych do znalezienia rozwiązania.

@param request - żądanie servletu

@param response - odpowiedź servletu

@throws ServletException – w przypadku wystąpienia błędu związanego z servletem

@throws IOException – w przypadku wystąpienia błędu wejścia/wyjścia

```
@Override
protected void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
    pl.jrj.game.IGameMonitor game;
    game=getConnection("java:global/ejb-project/GameMonitor"
            + "!pl.jrj.game.IGameMonitor");

    if (game.register(5, "104896")){
            response.getWriter().format(Locale.ENGLISH, "%d",
                imm.start(Integer.parseInt(request.getParameter("n")),
                Integer.parseInt(request.getParameter("k")),
                Long.parseLong(request.getParameter("s")),
                game)
            );
        } else
            response.getWriter().format(Locale.ENGLISH, "%.5f", 0.0);
     processRequest(request, response);
}
```

Rysunek 15 Implementacja metody doGet

Algorytm wyznaczania rozwiązania gry MasterMind

Algorytm pozwala w dość szybki sposób na znalezienie rozwiązania gry, w porównaniu do technik brute force, gdzie generowane byłyby wszystkie możliwe rozwiązania i porównywane w weryfikacji. Algorytm ten na samym początku losuje tablicę kolorów. Po sprawdzeniu w weryfikacji, czy pierwsza liczba jest większa od drugiej (czy ilość kolorów na poprawnym miejscu jest większa od ilości kolorów na niepoprawnym miejscy. Jeśli wystąpią takie wartości, algorytm przystępuje do sprawdzenia w którym miejscu tablicy znajdują się poprawne kolory. Sprawdzane jest to w ten sposób, że jeśli w danym indeksie tablicy zamienię stary kolor na nowy i pierwsza liczba z weryfikacji zostanie taka sama, to znaczy że kolor wcześniej też był niepoprawny. Jednak jeśli pierwsza liczba zmniejszy się o 1 to znaczy że poprzednia była poprawnym kolorem i w pomocniczej tabeli o aktualnym indeksie zaznaczamy flagę 1. Po sprawdzeniu następuje iteracyjne sprawdzenie komórek w których kolory są niepoprawne z przedziału [A;A+n]. Jako wynik zwracana jest ilość wywołań weryfikacji w algorytmie.