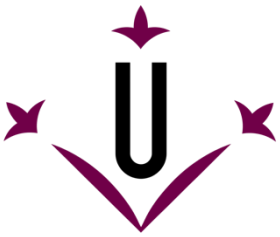


Universitat de Lleida

Escola Politècnica Superior

Grau en Enginyeria Informàtica

Programació II

Pràctica 2

Senku

Aniol Serrano Ortega

21161168H

01/05/2023

GPraLab 2

Índex

Índex de figures

No se encuentran elementos de tabla de ilustraciones.

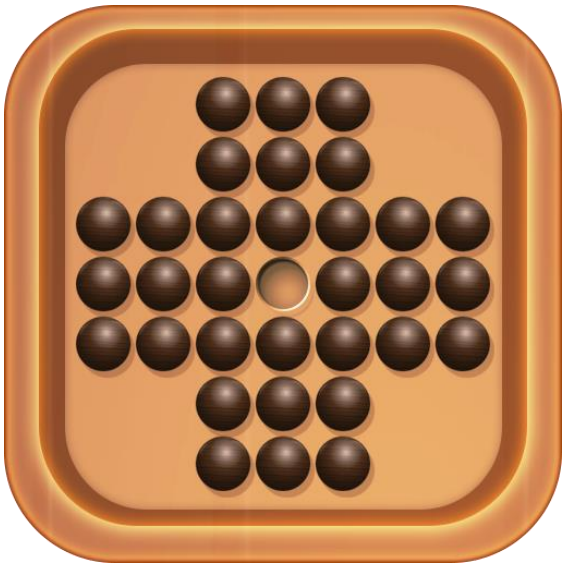
Índex d'il·lustracions

1. Introducció

En aquesta pràctica es tracta d'implementar un joc en Java anomenat Senku. La implementació d'aquest joc és una forma per a treballar la programació orientada a objectes. La implementació del joc se separa en diverses classes que implementen diversos mètodes que en conjunció d'aquests formen la programació del joc.

El joc del Senku és un joc de taula que es juga en un tauler i unes fitxes o boles. L'objectiu del joc és acabar-lo amb una única fitxa al tauler sense quedar-se sense moviments. Al principi del joc totes les posicions estan ocupades menys la posició central. El jugador ha de moure només una peça per torn.

Les peces només es poden moure saltant per sobre d'una altra en línia recta, com en les dames, però només en horitzontal o vertical, mai en diagonal. Així, al principi només unes poques tenen la possibilitat de moure's capturant-ne una. L'objectiu del joc és eliminar totes les peces, deixant només una al tauler. A continuació es mostra un exemple de taulell en la posició inicial de Senku.



Il·lustració 1: Tauler del Senku

2. Problemes en el desenvolupament de la pràctica

En aquesta pràctica al ser més complexa i llarga que l'anterior, ha comportat un major repte per a la implementació d'aquesta. Primerament, en l'enunciat al ser extens s'ha hagut de fer una llegida superficial per a entendre el context del joc que és el que es demana. Un cop fet la primera llegida s'ha fet una lectura més pausada sobre cada funció, en particular aquelles que s'havien d'implementar.

És rellevant considerar el fet que a l'estar algunes funcions ja implementades i els tests fets, és més fàcil seguir el desenvolupament del joc. A més, el fet que a l'informe cada classe i mètode estan detallats i descrits en ordre d'implementació facilita considerablement la implementació.

Segonament, en el cas de la implementació del codi ha hagut certs problemes que s'han propagat en altres implementacions i ha dificultat la programació. Malgrat això gràcies als tests s'han pogut solucionar ja que veure la composició dels test és de gran ajuda per a veure on falla la funció. Per exemple, en el cas de la classe *Direction* primerament les direccions s'havien situat suposant un pla cartesià, és a dir, suposant l'eix de coordenades a la cantonada inferior esquerra. Aquest supòsit només afectava en el cas de la direcció vertical, per tant, el codi primerament era així:

```
public static final Direction UP = new Direction(0, 1);
public static final Direction DOWN = new Direction(0, -1);
```

Després d'analitzar els tests es va fer la següent modificació:

```
public static final Direction UP = new Direction(0, -1);
public static final Direction DOWN = new Direction(0, 1);
```

Aquest és un exemple trivial però serveix per a exemplificar la metodologia de resolució de problemes. En cas de problemes més complexos també és adequat usar l'*output* que proporcionen els tests i l'ús del *debugger* per a tindre una major comprensió del error. En cas que aquestes dos eines no siguin d'ajuda, llavors és adequat fer un replantejament de la solució usant una idea diferent per a solucionar el problema. Totes aquestes eines resulten ser molt més potents si es domina l'entorn de desenvolupament, és a dir, l'IDE[1] usat.

3. Descripció dels 3 mètodes més complicats

En aquesta pràctica s'ha implementat un seguit de funcions sobre operacions matemàtiques amb vectors. La forma de tractar amb els vectors on el bit més significatiu és a la dreta i dígit té una posició, provoca haver de pensar per a una solució més creativa per a resoldre el problema.

El fet de disposar d'una bona previsió de pràctica ha facilitat el fet de poder resoldre els problemes de múltiples formes i trobar aquella que resulta més entenedora (Annex a.). També s'han descartat aquelles funcions que no resultaven útils per a resolució de la pràctica, però sí que han estat el camí per a entendre el problema i trobar una bona solució (Annex b.).

Finalment, aquesta pràctica ha estat molt útil per a construir uns bons fonaments sobre Java i millorar les meves habilitats, especialment en el tractament de vectors.

[1] **IDE** (Entorn integrat de desenvolupament): És una aplicació de *software* que ajuda als programadors a desenvolupar codi de manera eficient.

Annex

a. Versió alternativa de la funció *multiplyByDigit*

En aquesta versió en comptes d'utilitzar les funcions prèviament definides com són la funció *zero()* i *add()*, es resol el problema de forma totalment aïllada de les altres funcions. Aquesta solució funciona correctament, però la seva llegibilitat és clarament pitjor a més que no cohesiona el codi i, per tant, no fa ús del treball prèviament fet.

```
public int[] multiplyByDigit(int[] number, int digit) {
        if ((number.length == 1 && number[0] == 0) || digit == 0) {
            return new int[]{0};
        }

        int carry = 0, product;
        int[] result = new int[number.length + 1];
        for (int i = 0; i < number.length; i++) {
            product = (digit * number[i]) + carry;
            result[i] = product % 10;
            carry = product / 10;
        }

        if (carry > 0) {
            result[number.length] = carry;
            return result;
        }

        int[] shortResult = new int[result.length - 1];
        for (int i = 0; i < shortResult.length; i++) {
            shortResult[i] = result[i];
        }
        return shortResult;
    }
```

En canvi, el codi final ha estat molt més esclaridor, ja que sí que usa les funcions prèviament realitzades i, per tant, el codi resulta més senzill i intuïtiu.

```
public int[] multiplyByDigit(int[] number, int digit) {
        int[] result = zero();
        for (int i = 1; i <= digit; i++) {
            result = add(result, number);
        }
        return result;
    }
```

b. Funcions auxiliars descartades

Aquestes versions són funcions que s'havien implementat primerament però no funcionen per a números molt grans, i per tant, no implementen correctament la solució d'acord amb l'enunciat de la pràctica. Igualment aquestes funcions auxiliars són útils en cas que s'usés per a números que no provoquen desbordament, és a dir, petits.

```
// From int gets the length as if it was in one array
public int getLength(int num) {
        int length = 1;
        while (num >= 10) {
            num /= 10;
            length++;
        }
```

```
        return length;
}
```

Aquesta funció finalment s'ha implementat de forma similar però amb l'estructura adequada per ser usada al fibonacci.

```
// From array returns it in an int
public int getIntFromArray(int[] num) {
        int result = 0;
        for (int i = 0; i < num.length; i++) {
            int currentNum = num[i];
            result += currentNum * Math.pow(10, (num.length - i - 1));
        }
        return result;
}
```

Aquest codi, es trobaba inicialment en la funció *add* però finalment s'ha descartat ja que el codi principal ja contempla la posibilitat que un nombre sigui igual a zero.

```
if (equals(num1, zero())) {
        return num2;
} else if (equals(num2, zero())) {
        return num1;
}
```