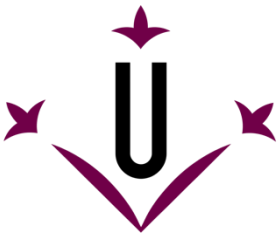


Universitat de Lleida

Escola Politècnica Superior

Grau en Enginyeria Informàtica

Programació II

Pràctica 2

Senku

Aniol Serrano Ortega

21161168H

01/05/2023

GPraLab 2

Índex

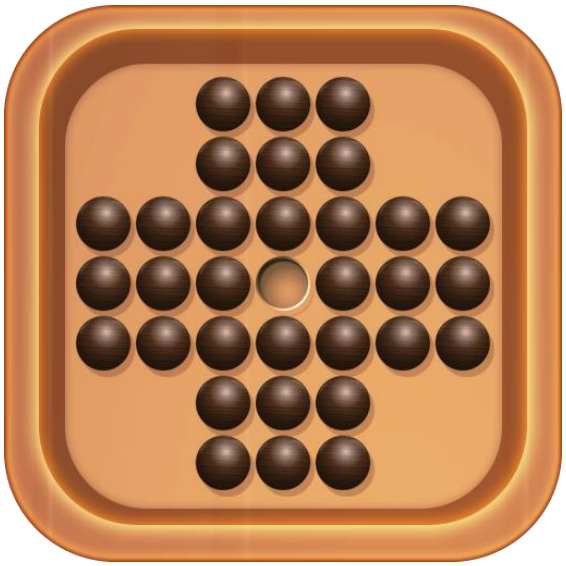
Índex d'il·lustracions

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 i que és el que es demana. Un cop feta 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.

Finalment, en el cas de la implementació del codi han sorgit certs problemes, que a més s'han propagat en altres implementacions i ha dificultat la programació. Malgrat això, gràcies als testos s'han pogut solucionar, ja que veure la composició d'aquests és de gran ajuda per a veure la estructura de la solució correcta.

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 testos 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 de l'error. En cas que aquestes dues eines no siguin d'ajuda, llavors és adequat fer un replantejament de la solució fent ús d'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] utilitzat.

3. Descripció dels 3 mètodes més complicats

A continuació es procedeix a fer una descripció detallada dels mètodes que han resultat més complicats en el desenvolupament de la pràctica.

3.1. Mètode *isValidFrom* (*Game*)

En la classe *Game* el mètode que ha resultat ser el més complex ha estat el *isValidFrom*. Aquest mètode verifica si una posició donada en el tauler és un punt de partida vàlid per a un moviment. Per a implementar-lo, primerament es recorre totes les direccions possibles i verifica si existeix un moviment vàlid en cada direcció. En aquesta implementació de s'ha usat la descomposició en funcions auxiliars de l'objecte *board*.

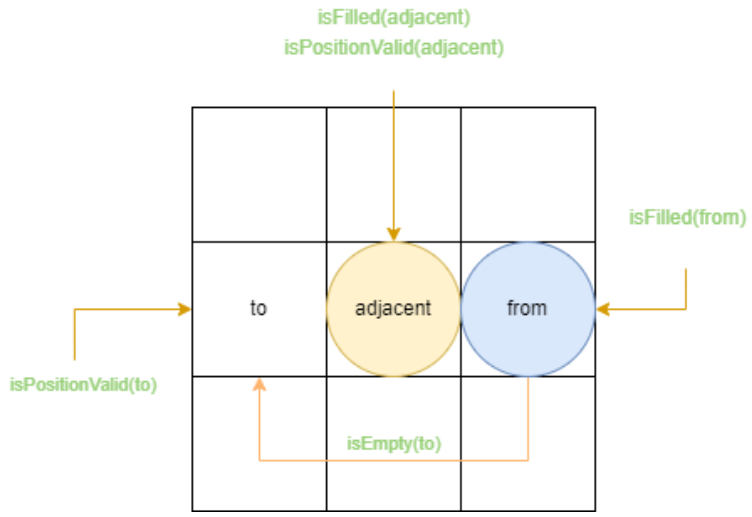
Per a verificar que un moviment s'han de complir que totes aquestes casuístiques siguin certes, a continuació es fa una descripció de cada casuística.

- ***isFilled(from)*:** Verifica si la posició actual està ocupada.
- ***isFilled(adjacent)*:** Verifica si la posició adjacent a la posició actual està ocupada.
- ***isEmpty(to)*:** Verifica que efectivament la posició a la que es vol fer el moviment estigui buida.

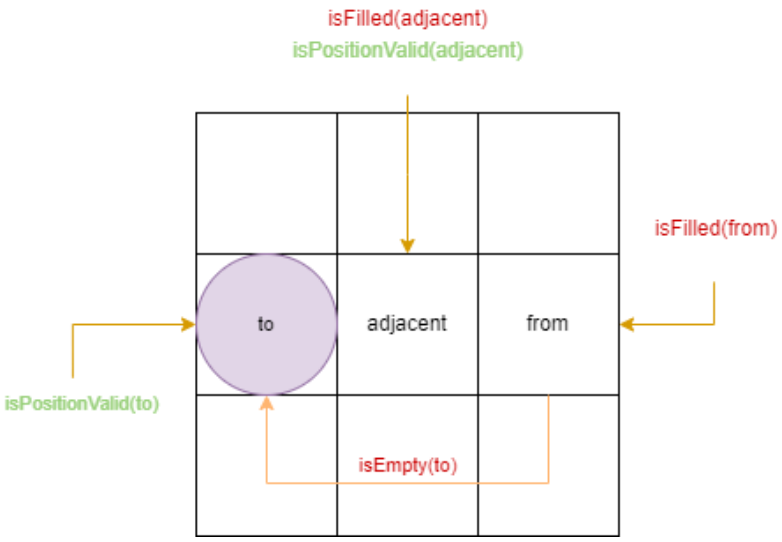
[1] **IDE** (Entorn integrat de desenvolupament): És una aplicació de *software* que ajuda als programadors a desenvolupar codi de manera eficient.

- ***isPositionValid(adjacent):*** Verifica si la posició adjacent a la posició actual és vàlida en el tauler, és a dir, si la posició adjacent es dins dels límits del tauler.
- ***isPositionValid(to):*** Verifica si la posició a la qual es vol moure és una posició vàlida dins dels límits del tauler.

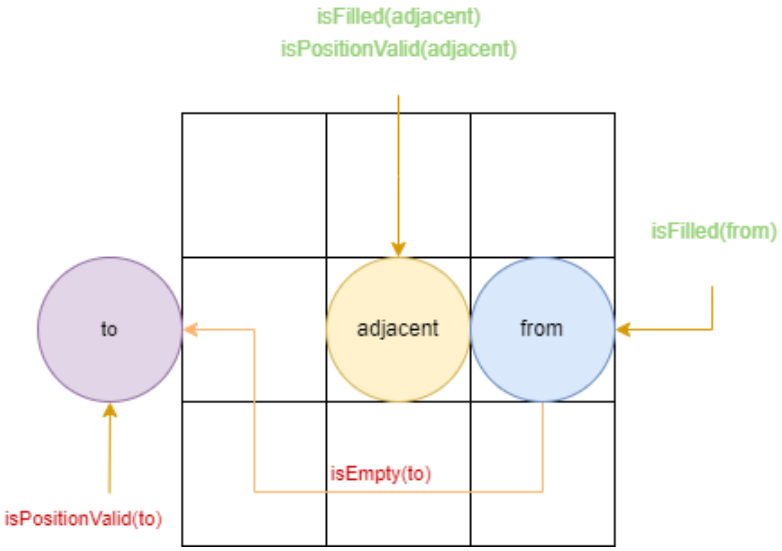
A continuació es mostren diverses casuístiques d'aquestes condicions. Val a dir que és necessari que totes les crides siguin verdaderes (en color verd) per a que el mètode retorni verdader.



Il·lustració 2: isValidFrom cas 1



Il·lustració 3: isValidFrom cas 2



Il·lustració 4: isValidFrom cas 3

3.2. Mètode constructor *Board* (Board)

Per a implementar la classe *Board* és necessari crear el constructor de la classe que rep com a paràmetres l'amplada i l'altura del tauler. A més, rep una cadena de text (*String board*) que representa les cel·les del tauler. Primerament, s'estableixen les mides del tauler amb les variables *width* i *height*.

Seguidament, es crea la matriu de les cel·les amb la mida especificada. El *string* de *board* s'ha de separar en línies i, per tant, s'usa el *String Tokenizer* amb el filtre '\n'. Es fa un bucle per a iterar sobre l'alçada on es llegeix cada fila del tauler. Dins d'aquest bucle es fa un altre bucle per a iterar sobre les columnes de la fila. En cada columna s'obté el caràcter corresponent a la cadena de text i es converteix en una cel·la. Per a convertir en una cel·la es fa ús del mètode *fromChar* de la classe *Cell*.

Finalment, aquesta cel·la creada es guarda en la posició que li correspon en la matriu. D'aquesta forma es construeix la matriu de cel·les amb el *string* donat.

3.3. Mètode *countValidMovesTo* (*Game*)

Aquest mètode retorna el nombre de moviments vàlids per a una posició vàlida en totes les direccions possibles. Primerament, es crea un *array* de totes les direccions possibles de la classe *Direction*. Seguidament, s'itera sobre totes les direccions possibles i es comprova si la posició de destí, és a dir, el *to* està dins del tauler i és buida. També es verifica si la posició adjacent és dins del tauler i es troba plena. En aquest cas, s'incrementa el nombre de moviments possibles en un comptador.

Per a obtenir les posicions adjacents i de destí, s'aplica la direcció sobre la posició actual amb la crida *apply* de la classe *Direction*. Finalment, es retorna el valor del comptador de moviments vàlids en totes les direccions possibles.

4. Conclusions

Aquesta pràctica ha estat útil per a entendre la programació orientada a objectes, ja que hi ha moltes classes que s'han d'implementar i usar per a mètodes més complexos. També ha ajudat per a fer us del disseny descendent de funcions, ja que, si els problemes se separen en problemes més petits, aquests resulten ser més manipulables.

A més, aquest projecte ha servit de pràctica introductòria als testos. Amb l'anàlisi d'aquests es pot veure quina és l'estructura de la solució. Malgrat passar un test no sigui garantia que la solució sigui la correcta o la més eficient, ajuda a saber que la implementació va per bon camí. D'altra banda, si la solució no passa els testos ajuda a identificar de forma ràpida i eficient els problemes.

En cas de tornar a començar la pràctica, es llegiria bé les descripcions de cada mètode, ja que hi ha, i hi ha hagut, comprovacions en el codi redundants. Com que certs mètodes ja tenen garantits certs paràmetres, no són necessàries aquestes comprovacions. També seria adequat afegir més testos per a comprovar aquells casos especials que no es tenen contemplats en un inici.