Autor: Èric Bitrià Ribes

DNI: 49755704B

Grup: 1

Pràctica 2: SENKU

Informe

Índex

[0. Continguts 2](#_Toc133576205)

[1. Introducció 3](#_Toc133576206)

[2. Desenvolupament 4](#_Toc133576207)

[2.1 boardFiller 4](#_Toc133576208)

[2.2 La classe Game 5](#_Toc133576209)

[2.3 xyToCell 6](#_Toc133576210)

[3. Conclusions 7](#_Toc133576211)

# 0. Continguts

Aquest informe conté l’explicació del raonament i desenvolupament de diferents funcions implementades a l’hora de realitzar la Pràctica 2: Senku.

S’inicia amb una explicació del plantejament general de la pràctica juntament amb les dificultats a nivell de programació, comprensió del enunciat i us del entorn de programació.

A continuació es mostraran les tres funcions que s’han considerat més complicades d’implementar, cada funció comentada tindrà la següent estructura:

* Plantejament
* Diagrames o imatges
* Explicació del Codi
* Mètodes alternatius o descartats
* Dificultats trobades

Finalment s’exposaran les conclusions amb el resum general de la pràctica, així com les observacions i valoracions personals sobre com s’ha realitzat.

# 1. Introducció

La pràctica 2 es basa en el joc Senku, es tracta d’un joc individual que consta d’un tauler de fixes que salten per sobre d’altres per matar-se entre si.

La practica demanava completar 6 classes que s’utilitzarien per poder executar el joc final, és a dir la pràctica no ha estat creada des de 0 si no que simplement era necessari crear funcions auxiliars que executarien el resultat final.

El conjunt de classes que conformaven la pràctica eren aquestes:

* Position
* Direction
* Cell
* Board
* Game
* Geometry

Juntament es disposaven de les classes:

* Palette
* Display
* Senku

Aquestes tres classes ja estaven completament implementades i solament eren necessàries implementar les primeres sis.

A part de les classes principals es disposava d’un conjunt de tests creats amb JUnit amb l’objectiu facilitar la creació de les funcions de cada classe.

# 2. Desenvolupament

Al llarg de tot el desenvolupament de la pràctica, no s’han trobat grans complicacions a l’hora de realitzar cada funció. Amb el guió de la pràctica, i el fet que molts sistemes estiguin subdividits entre diferents classes, feien que les funcions i constructors no ocupessin més de dues o tres línies.

Encara així, voldria considerar les tres funcions més complicades d’implementar que s’han torbat en el transcurs de la pràctica.

## 2.1 boardFiller

La funció boardFiller, es un mètode intern de la classe Board que permet al constructor donat una String emmagatzemar-lo dins una matriu de caselles, la qual representa el tauler.

Les seves característiques principals son que es basa en un bucle “for” que recorre totes les posicions d’un tauler temporal anomenat tempBoard:

Cell[][] tempBoard = new Cell[this.height][this.width];

Dins del bucle trobem un constructor de Cell que crea una nova casella utilitzant el caràcter corresponent a la posició del String.

Cell add = Cell.fromChar(board.charAt(k));

Seguidament hem d’afegir la casella add al tauler, primer comprovant que aquesta sigui vàlida, com que el constructor Cell només pot retornar null en cas que no sigui vàlida simplement comprovem que no sigui null i l’afegim.

if(add != null){

tempBoard[i][j]= add;

}

Ara solament incrementem el índex del string “k” i finalment retornem la matriu.

Durant la realització d’aquesta funció auxiliar no es van trobar cap dificultat, però posterior a la seva creació vaig trobar l’import del stringTokenizer, es va intentar realitzar un mètode amb aquesta llibreria però el resultat era bastant similar, així que es va mantenir el resultat original.

## 2.2 La classe Game

En aquest apartat de les funcions més complicades o a les que han sorgit dificultats s’inclou tota la classe game per un problema que afectava a totes en general. És a dir no ha resultat cap complicació la implementació de les funcions de Game sinó que el problema es trobava en una funció de la classe Board.

El error existent era un array Index Out Of Bounds Exception, y es trobava en pràcticament totes les funcions del Game. Amb un anàlisi del error recorrent la propagació d’aquest es va poder determinar que el bug es trobava a la funció auxiliar ouOfBounds de la Classe Board:

private boolean outOfBounds(Position pos){

return (pos.getX()<this.width && pos.getX()>=0 && pos.getY()<this.height &&

pos.getY()>=0);

}

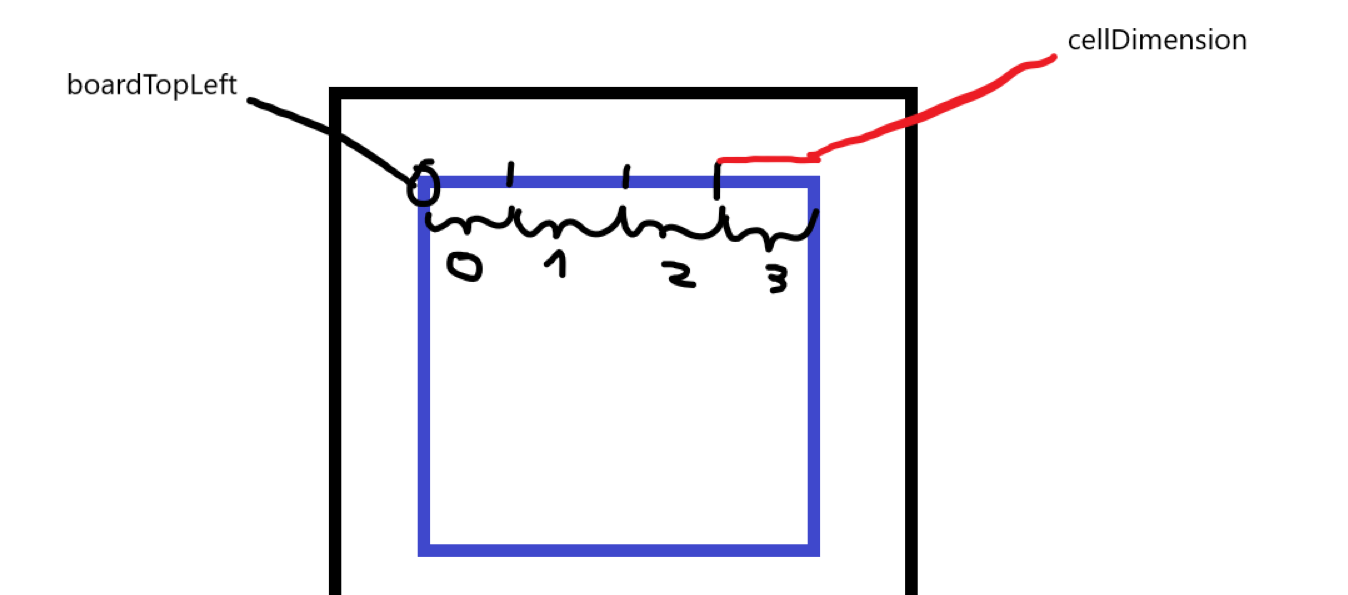
Aquesta funció comprovava si la posició pos es situava dins els marges del tauler. La primera versió d’aquesta funció no incloïa el més gran o **igual** a 0 (>=0), però encara així les funcions de prova CellTest no detectaven el error i validaven l’execució del codi.

Al no trobar el error a primera vista les funcions de la classe Game fallaven i no era per un problema de disseny sinó per el problema d’aquesta funció.

Per haver solucionat el problema des d’un primer moment s’hauria tingut que considerar totes les possibilitats del outOfBounds, ja que es un error molt “tonto” que ha complicat la pràctica en unes quantes hores de frustració sense trobar el motiu del error. Encara que hauria set molt més fàcil si simplement s’hagués llegit el error que sortia en consola i recorregut la propagació d’aquest des d’un inici.

## 2.3 xyToCell

Aquesta funció té l’objectiu de convertir les coordenades del cursor sobre la pantalla en la posició de casella a la qual correspon. El seu plantejament és mol senzill:



El que hem de comprovar es que les coordenades x, y es situïn entre dos marges: La posició més a l’esquerra de la casella i la posició més a la dreta de la casella:

i \* cellDimension().getWidth() <= x && x <= x\_marging + (i + 1) \* cellDimension().getWidth()

De esquerra a dreta, el índex “i” recorre les caselles de dreta a esquerra i comprova si el valor de la x es igual o més gran a la casella que estem buscant.

A picture containing icon

Description automatically generated

La segona condició comprova si és més petita o està a l’extrem de la següent casella (i+1).

A picture containing icon

Description automatically generated

Ara simplement repliquem el codi per el eix de les “y” i ja tindríem les comprovacions.

Però a aquestes comprovacions els hi hem d’afegir un punt d’origen, ja que en aquest moment comença a contar des del 0, però el nostre tauler té un tant per cent de marge que hem de considerar abans de entrar en les dimensions on es troben les caselles per això li afegim:

double x\_marging = boardTopLeft().getX();

double y\_marging = boardTopLeft().getY();

Una vegada trobat la posició que correspon a les coordenades, les retornem en format de Posició.

return new Position(col,row);

Una vegada realitzada la funció es va intentar trobar altres mètodes, per no quedar-se amb la primera opció. Encara així no va ser possible ni un altra manera de fer-ho ni de millorar el codi, encara que es podria utilitzar la funció cellTopLeft(x,y) per agafar les posicions de la casella però a efectes pràctics s’obtenia el mateix resultat, i es va descartar.

# 3. Conclusions

Aquesta pràctica incloïa una gran varietat de tipus de classes, cadascuna amb propietats i característiques diferents. S’ha observat com a la classe direction el constructor no té per que ser públic cosa que en un primer moment no vaig entendre i em va resultar estrany.

També s’ha considerat molt la forma en la que s’interconnecten totes les classes, és a dir, el disseny descendent ja no solament és limita a la creació de funcions auxiliars, sinó que tot un conjunt de classes amb característiques pròpies que en conjunt poden abordar un problema molt més complex.

A més la divisió de les classes permet una clara i estructurada lectura del codi i permet una gran escalabilitat del joc al utilitzar paràmetres que depenen d’altres i no deixar valors hard coded dins el programa.

Si es tingués que refer la pràctica, valoraria més la creació de diagrames i formes clares i visuals de les intencions del codi. El fet que el guió aportes imatges visuals indicant cada mètode i el que representa ha facilitat molt la creació de la pràctica, encara així crear els teus propis diagrames aporta un esforç de comprensió del perquè i com ho estàs fent.

A part de tot el tema de comprensió i codi també valoraria molt més l’anàlisi d’erros que aporta intelliJ, saber que signifiquen i com abordar-los. També valoraria com no dependre tant dels tests, i que encara que els passi tots, ser crític a l’hora d’analitzar el teu codi i tenir en compte totes les possibilitats que possiblement no han estat aplicades al diferents tests.

En general és una bona pràctica per agafar el costum al disseny descendent de funcions utilitzant moltes classes i mètodes més senzills per abordar una solució molt més complexa. I també el fet de valorar els tests i errors per saber com entendre’ls i que siguin una eina útil a l’hora de programar.