Primera Entrega

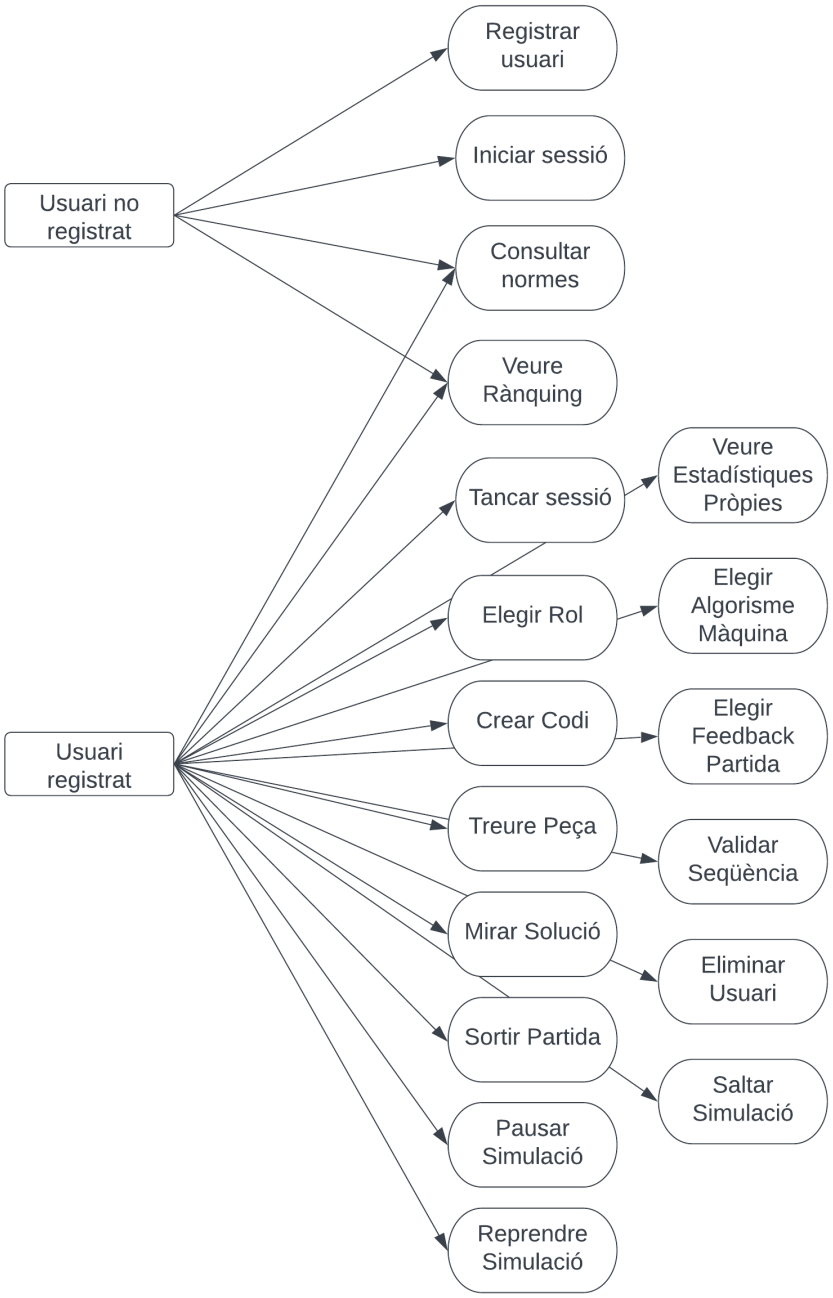
Albert Canales Ros
Mar Gonzàlez Català
Kamil Przybyszewski
Arnau Valls Fusté

Projectes de Programació

Facultat d'Informàtica de Barcelona
Universitat Politècnica de Catalunya

Contents

1 Casos d'Ús


Usuari no registrat
Usuari registrat
Registrar usuari
Iniciar sessió
Consultar normes
Veure Rànquing
Tancar sessió
Elegir Rol
Crear Codi
Treure Peça
Mirar Solució
Sortir Partida
Pausar Simulació
Reprendre Simulació
Veure Estadístiques Pròpies
Elegir Algorisme Màquina
Elegir Feedback Partida
Validar Seqüència
Eliminar Usuari
Saltar Simulació

1.1 Registrar Usuari

Actor: Usuari no registrat

Comportament:

L'actor escull fer un registre d'usuari. Ha d'indicar els seu nom i cognoms, codi d'usuari (*username*) i clau de pas (*password*). L'última ha de de ser inserida dos cops.

Errors possibles i cursos alternatius:

El codi d'usuari ja ha estat assignat en un altre registre: canviar o abandonar. El nom i cognoms, o *username*, o *password* son invàlids: tornar a introduir-les. Les dues contrasenyes donades no coincideixen: tornar a introduir-les.

1.2 Iniciar Sessió

Actor: Usuari no registrat

Comportament:

L'actor demana accedir al seu usuari associat. Indica el seu codi d'usuari (*username*) i la seva clau de pas (*password*).

Errors possibles i cursos alternatius:

La combinació de codi d'usuari i clau de pas no és correcta: tornar a introduir-les. No existeix cap usuari registrat amb aquest codi d'usuari: introduir un codi d'usuari diferent.

1.3 Consultar Normes

Actor: Usuari no registrat, Usuari registrat

Comportament:

L'actor demana veure la normativa del joc. El sistema presenta la normativa del joc.

1.4 Veure Rànquing

Actor: Usuari no registrat, Usuari registrat

Comportament:

L'actor demana veure el rànquing de puntuacions de les partides. El sistema presenta el llistat de les partides ordenat per puntuació obtinguda.

Errors possibles i cursos alternatius:

No hi ha partides: s'indica amb un missatge i es presenta una llista buida.

1.5 Veure Estadístiques Pròpies

Actor: Usuari registrat

Comportament:

L'actor demana veure les seves estadístiques generals. El sistema presenta els valors de les estadístiques com a CodeBreaker i com a CodeMaker de l'usuari.

Errors possibles i cursos alternatius:

L'usuari no té partides: s'indica amb un missatge i es presenten estadístiques buides.

1.6 Tancar Sessió

Actor: Usuari registrat

Comportament:

L'actor decideix deixar d'estar associat amb l'usuari indicat prèviament en l'inici de sessió. El sistema li ofereix les opcions d'inici de sessió i de registre d'un nou usuari.

1.7 Elegir Algorisme Màquina

Actor: Usuari registrat

Comportament:

L'actor indica quin dels algorismes disponibles vol que la màquina empri per a exercir de CodeBreaker. El sistema guarda la informació.

1.8 Elegir Rol

Actor: Usuari registrat

Comportament:

L'actor comunica una partida no inicialitzada i el rol que hi vol prendre: codemaker o codebreaker. El sistema registra les actualitzacions.

Errors possibles i cursos alternatius:

La partida ja està inicialitzada: s'indica amb un missatge i no s'actualitza res

1.9 Crear codi

Actor: Usuari registrat

Comportament:

L'actor indica la seqüència de colors que serà codi ocult en la partida. El sistema guarda la informació, que utilitzarà a posteriori per comprovar si les seqüències proposades pel codebreaker són el codi ocult (i per tant, el joc ha de finalitzar).

Errors possibles i cursos alternatius:

La seqüència no és de quatre colors. El sistema informarà a l'actor de l'error i li demanarà de fer una correcció a la seva proposta de seqüència no guardant cap informació fins que se'n proposi una de vàlida.

1.10 Elegir Feedback Partida

Actor: Usuari registrat

Comportament:

L'actor comunica una partida no inicialitzada i el mode de joc que tindrà la partida: això és, el tipus de feedback que rebrà l'usuari actuant com a codebreaker. El sistema registra les actualitzacions.

Errors possibles i cursos alternatius:

La partida ja està inicialitzada: s'indica amb un missatge i no s'actualitza res

1.11 Treure Bola

Actor: Usuari registrat

Comportament:

L'actor escull la bola que vol treure de la seqüència i el sistema la deixa al lloc pertanyent depenent del seu color

1.12 Validar seqüència

Actor: Usuari registrat

Comportament:

L'actor demana comprovar la semblança entre la seqüència que proposa i el codi ocult.

1.13 Mirar Solució

Actor: Usuari registrat

Comportament:

L'actor comunica que vol veure el codi ocult de la partida actual i el sistema li facilita.

1.14 Eliminar usuari

Actor: Usuari registrat

Comportament:

L'actor indica que desitja esborrar l'usuari que té associat de la base de dades. El sistema esborra totes les seves dades personals del sistema i dels rànquings i automàticament tanca la sessió.

1.15 Sortir Partida

Actor: Usuari registrat

Comportament:

L'actor comunica que vol sortir de la partida i el sistema s'encarrega d'aturar-la.

1.16 Saltar simulació

Actor: Usuari registrat

Comportament:

L'usuari indica que desitja saltar la simulació. El sistema mostra el taulell resultant de la partida.

1.17 Pausar simulació

Actor: Usuari registrat

Comportament:

L'usuari indica que desitja pausar la simulació. El sistema atura la simulació deixant visible la situació actual de la partida.

1.18 Reprendre simulació

Actor: Usuari registrat

Comportament:

L'usuari indica que desitja reprendre la simulació després d'haver-la pausada. El sistema reprèn la simulació des del punt on havia estat aturada.

2 Breu descripció de les estructures de dades i algorismes

2.1 Boles, Nivells de Dificultat i Tipus d'Algorismes

Hem representat aquests tres objectes diferents amb una mateixa estructura, l'*enum*. La raó principal per la qual no hem utilitzat un tipus primitiu com l'enter és per tenir un codi més llegible i mantenible.

A més, com que Java ens permet afegir petits mètodes als *enum*, també hi hem incorporat en el cas de bola un pocs mètodes estàtics que ens permeten evitar la duplicació de codi.

Finalment també és important destacar que l'ús d'aquestes estructures no fa el codi més lent, ja que quan la resta de classes les utilitzen i les guarden, sempre ho fan amb els seus valors numèrics.

2.2 Seqüències

Utilitzem la paraula *Sequència* per referir-nos a una llista ordenada de *Boles*. Tot i que no tingui una classe com a tal ja que no era necessària, aquest concepte apareix repetidament en el codi en forma de:

- Solució. És la seqüència secret a descobrir en la partida.
- Intent. És una seqüència en la qual es pretén endevinar la solució.
- Feedback. És la resposta que el joc al Breaker a un intent.

Sabent que aquests tres conceptes estan representats per un mateix tipus, aquest tipus és una *ArrayList<Integer>*. Per ser més generals en el codi, en canvi, les guardem com a la seva superclasse *List<Integer>*.

La raó per la qual hem optat per una *ArrayList* en comptes de, per exemple, una *LinkedList* és purament per eficiència. Com que és comú fer modificacions en posicions arbitràries, el tipus escollit ho farà en temps constant respecte la llargada de la llista.

A més, tots els tipus de seqüència anomenats tenen una mida fixa (en cas de ser vàlids) que està especificada a la classe *Taulell*, en la variable `NUMBOLES`. Per temes de l'algorisme, hem elegit que en el nostre joc aquest sigui 4. Això també coincideix amb la capacity per defecte de *ArrayList*, així que també estem fent un ús eficient de la memòria en aquest aspecte.

2.3 Taullel

L'objecte *Taulell* és a nivell d'estructura de dades el més complex, ja que representa el contenidor amb els objectes necessaris per jugar una partida. Conté tres objectes:

- Solució. Seqüència solució de la partida
- Intents. Llista amb els intents de la partida.
- Feedbacks. Llista amb els feedbacks de la partida.

La primera, com s'ha dit abans, és una *List<Integer>*. Les altres dues són *List<List«Integer»*. A diferència del cas anterior, aquí no modifiquem ni accedim les posicions intermitges de la llista major i augmentem dinàmicament el tamany afegint valors al final de la llista.

Degut a això, ens seria indiferent optar per una *LinkedList* o una *ArrayList*, per comoditat d'implementació, hem seguit elegint una *Arraylist*. Cal notar que les llistes tampoc creixen indefinidament, ja que tenen un màxim de `NUMINTENTS+1` i `NUMINTENTS` respectivament.

2.4 User

L'objecte User emmagatzema les dades relacionades amb l'usuari que actualment ha iniciat sessió. Aquestes dades ja estan sempre disponibles des de Persistència (calculant-les a través del registre de partides), es copien a Domini en l'inici de sessió per una questió de eficiència.

L'objectiu de la classe és que després de l'inici de sessió les estadístiques pròpies de l'usuari no s'hagin de consultar de Persistència quan siguin requerides per Presentació, sinó que les ofereixi directament domini.

Per això cal guardar les estadístiques pròpies. Moltes d'aquestes (les que són llistes) tenen un valor per cada NivellDificultat o TipusAlgorisme possible. Per facilitar les crides, les dades es reben amb llistes, que un cop més són *Arraylists* ja que ens interessa fer un accés i modificació en una posició aleatòria.

2.5 FiveGuess

L'algorisme FiveGuess aconsegueix mitjançant l'ús de la tècnica minimax endevinar la seqüència oculta en cinc intents o menys. S'assumeix el feedback del mastermind clàssic: número de boles coincidents en color i posició -fitxes negres- i número de només coincidents en color -fitxes blanques-, donat un intent comparat amb la solució. També es pressuposa que el joc és amb seqüències de quatre boles i de sis colors possibles.

L'estratègia de l'algorisme per escollir el següent intent a enviar al CodeMaker es basa en la tècnica *minimax*. Per a cada possible seqüència, analitza de entre seqüències que encara romanen com possibles solucions -donats els feedbacks del CodeMaker previs- quantes compartirien cada possible feedback, és a dir, si enviéssim aquell intent i ens donés un cert feedback, a quant es reduiria la pool de possibles solucions. Guardem pel feedback que deixaria una pool de possibles solucions major, el tamany de la respectiva potencial pool. Com a

següent intent selecciona la seqüència amb menor màxima pool dels feedbacks - d'aquí la terminologia *minimax*. Amb aquest criteri es pot demostrar que arriba a la solució en com a màxim cinc intents.

Estructures de dades en la implementació del FiveGuess:

- Seqüència solució i seqüències intents (actual i finals): externament estan implementades com una List Integer perquè això ens demanava l'enunciat. Internament vam decidir implementar-les com una ArrayList<Integer>perquè la funció E set(int index, E element) ens permet fer l'equivalent a "col·locar una bola d'un color determinat", cosa que necessitem fer constantment internament en tot el projecte i en especial a la classe FiveGuess.
- Conjunt d'intents: externament està implementat com una List<List<Integer»perquè això ens demanava l'enunciat. Internament són una ArrayList<ArrayList<Integer»perquè aquesta estructura és coherent amb el que representa que és: un conjunt d'intents.
- Conjunt de possibles solucions per l'algorisme FiveGuess: l'estructura que hem utilitzat en aquest cas és un Hashmap<Integer>perquè és una estructura que es recorre molt ràpidament, una funcionalitat que minimax ens obliga a utilitzar constantment. A més, també ens ofereix la possibilitat de treure elements fet que l'algorisme de FiveGuess exigeix.