CodeMaker

Una nova forma de jugar al Mastermind

Treball Final de Grau

28 d'octubre de 2016

Autor: Lluc Pont Rojas

Director: Lluís Belanche

Titulació: Grau en Enginyeria Informàtica

Especialitat: Computació

Facultat informàtica de Catalunya

Universitat Politècnica de Catalunya

Índex

Índex de taules

Índex de il·lustracions

1 Resum

Mastermind és un joc de lògica per a dos jugadors, el qual consisteix en que un dels jugadors (el CodeBreaker) intenta encertar una combinació secreta pensada per l'altre jugador (CodeMaker).

En aquest projecte s'han afegit dos noves normes al joc; la primera permet que el CodeMaker pugui canviar la seva combinació secreta sempre que vulgui (però sense incomplir les anteriors respostes del taulell); la segona consisteix en que només es conservin les tres últimes preguntes al taulell (després de respondre la quarta, la més antiga s'elimina).

Per comprovar com afecten les normes al joc s'ha creat un sistema que juga partides de forma automàtica per obtenir informació estadística.

S'ha creat un algorisme que juga amb les noves normes, com també un algorisme de Machine Learning que usa la mateixa lògica.

Per últim, s'ha implementat una aplicació mòbil que permet jugar en els dos rols existents contra les diferents estratègies de la màquina, o fer que dos estratègies juguin entre ells.

* * *

Mastermind es un juego de lógica para dos jugadores, el cual consiste en que un jugador (CodeBreaker) intenta acertar la combinación secreta pensada por el otro jugador (CodeMaker).

En este proyecto se han añadido dos nuevas reglas al juego; la primera permite que el CodeMaker cambie la combinación secreta siempre que quiera (pero sin incumplir las respuestas de tablero); la segunda consiste en que solo se conserven las tres ultimas preguntas del tablero (después de responder la cuarta, la mas antigua se elimina).

Para comprobar como afectan las nuevas reglas al juego se ha creado un sistema que juegue partida de forma automática para obtener información estadística.

Se ha creado un algoritmo que juega con las nuevas reglas, como también un algoritmo de Machine Learning que usa la misma lógica.

Por ultimo, se ha implementado una aplicación móvil que permite jugar en los dos roles existentes contra las distintas estrategias de la maquina, o hacer que dos estrategias jueguen entre ellas.

* * *

Mastermind is a logic game for two players. The first player (CodeBreaker) tries to answer the secret code created by the other player (CodeMaker).

In this project we added two new rules; the first one enables the CodeMaker to change his secret code whenever he wants (however, he has to respect the old guess on the board); the second rule says that only the last three guesses are stored on the board (after answering the forth guesses, the last one is deleted).

To check the impact of the new rules, we created a program that plays games automatically; therefore, we will recollect a lot of statistical information.

We created an algorithm that plays with the new rules; moreover, we created a Machine Learning algorithm that plays with this new logic.

Last but not least, we implemented a mobile application in which you can play in both roles (CodeMaker and CodeBreaker) against a lot of machine's algorithm; or you can see two algorithm playing against.

2 Introducció

El joc Mastermind és un joc de lògica per a dos jugadors, en el que només un interactua i realitza accions (el que intenta encertar la combinació secreta (CodeBreaker)), mentre que l'altre només l'hi respon seguint unes pautes molt rígides (CodeMaker).

En el joc clàssic, es juga amb 4 columnes, on a cada columna hi pot haver un dels 6 possibles colors, generant un total de 6^4 = 1296 possibles combinacions secretes. Encara que pot semblar difícil encertar la combinació secreta en pocs torns, gràcies als algorismes computacionals actuals, aquesta s'encerta en una mitjana de 4.5 torns (sense passar dels 7 torns).

En aquest projecte s'ha plantejat una variació en les normes del joc amb l'objectiu que els dos jugadors tenen un rol actiu; alhora que augmenta la complexitat del joc. Aquesta variació permetrà que el CodeMaker canviï la combinació secreta sempre que vulgui (fins i tot després se saber la combinació preguntada pel CodeBreaker (i abans de respondre'l)), però sempre respectant que la nova combinació secreta compleixi amb els pins que s'han respòs en les 3 preguntes anteriors .

Un requeriment de tots els jocs és que siguin equilibrats, es a dir, que els dos jugadors tenen unes possibilitats similars de guanyar la partida. Per comprovar-ho, en aquest projecte s'ha estudiat com interactuen els algorismes actuals que juguen al Mastermind (en el rol de CodeBreaker) amb les noves normes; també s'han adaptat per poder jugar en el rol de CodeMaker.

Addicionalment, s'ha dissenyat i implementat un algorisme perquè jugui en el rol del CodeMaker, aprofitant al màxim les noves normes. Per optimitzar el temps de decisió i resposta, es s'ha dissenyat i entrenat un algorisme de Machine Learning que usa la nova lògica; s'ha implementat un algorisme per la generació de les dades d'entrenament; s'han aplicat diverses optimitzacions a les dades...

Gràcies a aquests algorismes (els clàssics i el nou) s'ha comprovat (estadísticament) si el joc és equilibrat o no (si un dels dos rol té més possibilitats de guanyar que l'altre o es equitatiu).

S'ha creat una aplicació per mòbil Android que implementa totes aquestes característiques: per poder jugar tant al joc clàssic com amb les noves normes; tant en el rol de CodeBreaker com en el de CodeMaker; poder triar entre una gran diversitat d'estratègies de la màquina...

3 Mastermind

3.1 Que és el Mastermind

El Mastermind és un joc d'ingeni i lògica per a dos jugadors.

El joc està format per 3 components: el taulell, les fitxes de colors, i els pins de resposta.

Un jugador, en el rol del CodeMaker (creador del codi), s'inventa un combinació que es mantindrà secreta per l'altre jugador, el qual juga en el rol del CodeBreaker (trencador del codi). CodeBreaker l'intentarà d'endevinar preguntant si una combinació (creada per ell) és la secreta o no.

El CodeMaker li respondrà informant-lo com el codi proposat s'assembla al secret o no, usant els pins de resposta. Gràcies a aquestes pistes, el CodeBreaker podrà deduir la combinació secreta pensada pel CodeMaker.

Il·lustració 1: Taulell Mastermind[1]

3.2 Història

El joc es va inventar en el 1970 per en Mordecai Meirowitz. Mordecai era Israelià, carter i expert en telecomunicacions. Per la creació del joc, Mordecai es va basar en el joc 'Bulls and Cows'[2] (a diferència del Mastermind, aquest no té cap versió de taulell coneguda).

[1] En algunes versió del joc, enlloc de pins de resposta negre s'usen vermells
[2] Aquest joc també es anomenat Numerello (en italià)

Gràcies a la simplicitat de la seva representació, era molt comú jugar amb llapis i paper. A l'any 1971, la companyia Britànica 'Invicta Plastics Ltd' va ser la primera que va comercialitzar una versió del joc en format física.

El 1977 es va crear la primera versió del joc en un format no físic. Gràcies a les companyies Sears i Atari es va implementar el joc en per la videoconsola Atari 2600. A partir del 1980 es van començar a crear les primeres versions per PC. Actualment, hi ha moltes implementacions del joc per dispositius mòbils.

3.3 Normes[3]

La preparació del joc consta d'un pas: el CodeMaker es crea la combinació secreta (formada per 4 fitxes, en la qual es pot repetir els colors) usada per la partida. Aquesta combinació s'haurà de mantenir oculta a l'altre jugador, que l'haurà d'endevinar (el CodeBreaker).

Cada torn transcorre de la següent manera:

- El CodeBreaker crea una combinació (amb les fitxes del colors) i l'hi pregunta al CodeMaker si la combinació preguntada és igual a la secreta.
- El CodeMaker l'hi respon, gràcies als pins de respostes negres i blancs, quan s'assembla la combinació proposada a la secreta. Cada pin negre significa que una fitxa de color està en la mateixa posició en ambdues combinacions; cada pin blanc significa que hi ha una fitxa de color que està en les dos combinacions, però no en la posició correcte.
- El CodeBreaker, gràcies a la informació que obté de les respostes de les preguntes realitzades, pensa una nova combinació a preguntar.

La partida s'acaba quan el CodeBreaker encerta la combinació secreta (el CodeMaker li respon amb 4 pins negres) o realitza 10 preguntes.

3.3.1 Aclariments

- En la primera pregunta, el CodeBreaker no té cap informació, per tant, la primera pregunta sempre és a cegues.
- El CodeBreaker pot preguntar una combinació encara que sàpiga que no pot ser la secreta.
- Quan en CodeMaker respon a la proposada del CodeMaker, cada fitxa de color (de les dues combinacions) només s'avaluarà una vegada (primer es mirarà si està al lloc correcte, i després si hi està però en l'una o altra posició).
- L'ordre o la posició dels pins de resposta no té relació amb l'ordre o la posició de cap de les dues combinacions.

[3] Degut a que hi ha moltes variacions en el joc, en aquesta secció tan sols es comentaran les normes clàssiques

4 Formulació del problema

4.1 Context

Com a jugador de Mastermind, he apreciat que té 2 punts febles, els quals crec, que si es milloressin, els jugadors gaudirien molt més del joc. El primer és que el jugador que juga com a CodeMaker té un rol molt passiu; l'altre es que, actualment, gràcies als algorismes computacionals, les partides es poden acabar amb poques rondes (el CodeBreaker encerta la combinació secreta ràpidament).

El CodeMaker té un rol molt passiu

En el joc clàssic del Mastermind (i en les variacions més comunes) el CodeMaker només pot realitzar 2 accions: pensar la combinació secreta i respondre amb els pins de resposta a les preguntes que l'hi fa el CodeBreaker.

L'acció de decidir la combinació secreta és més complexa que triar 4 colors aleatòriament, ja que hi ha combinacions secretes millors que altres (per exemple, 4 fitxes vermelles és una combinació més fàcil de trobar que vermell-blau-verd-vermell). El fet de que el CodeBreaker sigui amateur o expert, o si aquest segueix una estratègia determinada, afecta la qualitat de les possibles combinacions secretes. Aquesta acció només es fa un cop, al principi de la partida.

En la segona acció que pot realitzar, la de respondre a la pregunta del CodeBreaker, el CodeMaker no realitza cap elecció, ja que, per cada parella de combinació preguntada i combinació secreta, només hi ha una resposta de pins blancs i negres possible. Addicionalment, la resposta donada es fàcil de calcular.

Les partides són curtes

Actualment, hi ha molts algorismes informàtics que juguen al Mastermind (en el rol de CodeBreaker) i encerten la combinació secreta ràpidament (per exemple, en el Mastermind clàssic, quasi tots els algorisme acostumen a encertar la combinació secreta en 4 o 5 torns.

4.2 Abast

Per millorar els dos punts mencionats anteriorment es vol crear una nova versió del joc, la qual afegirà dues noves regles. Amb aquesta nova versió s'espera, per una banda, que el CodeMaker tingui un rol molt més actiu i, per l'altra, augmenten la complexitat del joc. S'espera que, a conseqüència d'això, els algorismes actuals que s'usen en el rol de CodeBreaker en el Mastermind no seran tan eficients (necessitaran més torns per encertar la combinació secreta).

Les noves regles són les següents:

- El CodeMaker pot canviar el codi secret quan vulgui; sempre i quan no contradigui les respostes donades en els torns anteriors.
- Només es conserven les últimes tres preguntes de codi secret; al realitzar una quarta, la més antiga s'eliminarà (després de que el CodeMaker respongui). El CodeMaker pot canviar la seva combinació secreta només tenint en compte les preguntes i respostes no eliminades.

Quan es modifica un joc (per exemple, afegint noves normes), s'ha d'aconseguir que el joc sigui equilibrat, és a dir, que el joc no sigui ni trivial ni impossible per cap de les dues parts. Per comprovar si és equilibrat o no, es crearà un programa el qual jugaran partides de forma automàtica (amb les noves normes) contra els algorismes més comuns del Mastermind; per així obtenir dades estadístiques dels torns necessaris per encertar la combinació secreta. S'espera que els algorismes actuals necessitin bastant més torns per encertar la combinació secreta; o fins i tot que algun d'aquests no la pugui encertar mai.

En el projecte, es crearà un algorisme que jugui en el rol de Machine Learning, aprofitant totes les noves normes del joc. També s'entrenarà un algorisme de Machine Learning, que seguirà la mateixa estratègia.

També es desenvoluparà una aplicació per mòbil Android, en la qual es podrà jugar al joc Mastermind, tant amb el rol de CodeMaker com el de CodeBreaker, amb les noves normes o amb les clàssiques. S'implementaran diversos algorismes per l'estratègia de màquina, en ambdós rols. Gràcies a l'aplicació s'espera, per una banda, popularitzar la nova variació del joc clàssic, i per l'altra, permetre obtenir beneficis econòmics.

4.3 Objectius

Pel projecte s'han definit tres grans objectius, els quals es poden desglossar en altres objectius. Aquests es mostren a continuació:

- **Noves normes**
  - Rol actiu del CodeMaker
  - Incrementar la dificultat del CodeBreaker
  - Joc equilibrat
- **Algorisme Machine Learning**
- **Aplicació mòbil**
  - Joc clàssic
  - Joc amb les noves normes

- Estratègies de la màquina
  - *Primera opció possible*
  - *Pitjor cas*
- Millorar l'aparença de l'aplicació
- Pujar l'aplicació a la Play Store
  - *Anuncis en l'aplicació*

4.4 Estat de l'art

4.4.1 Estudis anteriors

Hi ha molts estudis sobre el Mastermind i les seves estratègies per guanyar el joc. La complexitat de les estratègies és molt diversa: des d'alguns que estan pensats per poder usar-los sense ajuda d'ordenadors (mentalment o amb llapis i paper), fins als que es necessita un gran computador per poder-los dur a terme. Segons la seva metodologia, es poden classificar en 3 categories, simples, força bruta i aprenentatge màquina.

Simples

La principal característica dels algorismes classificats en la categoria de simples és que es poden realitzar sense ajuda d'ordinadors, com a molt requereix l'ajuda de llapis i paper.

- **Primera opció possible:** Aquest algorisme es basa en respondre sempre la primera combinació (ordenades segons l'ordre de les fitxes de colors) que pot ser la combinació secreta[4]. Un cop s'obtenen els pins de resposta del CodeMaker, es torna a calcular quina és la primera combinació possible.
- **Patró:** Aquest consisteix en seguir una guia on està indicat quina tirada s'ha de realitzar, depenent de les preguntes realitzades i resultats obtinguts. Aquest mètode és el més eficient de tots (es guanya amb menys torns). El problema d'aquest mètode és que està pensat, específicament, per les normes clàssiques, és a dir, si s'afegeix alguna variació en les normes o mida de la partida, s'ha de generar un nou patró des de zero, on es tinguin en compte totes les noves possibilitats, i calcular la millor forma de tractar-les.

Computacionals/força bruta

Aquests algorismes usen computadores per saber quina és la millor combinació a preguntar. Aquests es basen en calcular els resultats de totes les possibles combinacions per cada una de les combinacions secretes possibles.

La diferència entre les estratègies d'aquest tipus són els resultats de les comparacions que valoren a l'hora de prendre la decisió de quina tirada a realitzar.

[4] Existeix una variació d'aquest mètode que consisteix en respondre una de les opcions possibles de forma aleatòria. Aquesta variació es mes eficient (en necessiten menys torn), però una mica mes complexa.

- **Pitjor cas:** Es realitza la tirada en la que, en el pitjor escenari, el resultat sigui el millor possible.
- **Esperança:** Es realitza la tirada en la que, estadísticament, hi hagui el menor número de combinacions secretes possibles.
- **Entropia:** Es realitza la tirada que té una major entropia.

Machine Learning (Aprenentatge Automàtic)

Aquesta categoria engloba les estratègies que es basen en que el mateix algorisme aprengui el joc i ell descobreixi quina és la millor tirada a realitzar. Aquests algorismes tenen una fase d'aprenentatge, en la que se li mostren unes dades que representen l'estat actual. Gràcies a les dades, decideix com actuar. Depenent de com s'entreni (la seva estructura i les dades usades en l'entrenament), el resultat pot variar molt.

La majoria són algorismes genètics i evolutius específics. També en trobem alguns que usen les Xarxes Neuronals (són poc comuns ja que necessiten una gran quantitat de temps i de dades per entrenar la Xarxa Neuronal).

En general, aquests algorismes no són tan bons pel joc clàssic, però són molt escalables per jugar amb variacions complexes, com més fitxes en la combinació secreta o més fitxes de colors.

* * *

Tipus	**Intents estimats**	**Intents en el pitjor dels casos**
Primera opció possible	6	10+ [5]
Patró	4	5
Pitjor cas	4.5	5
Esperançar	[4.5]	6
Entropia	4.5	6

Taula 1: Torns necessaris per estratègia

4.4.2 Variacions del joc

Hi ha moltes variacions del Mastermind, algunes es basen a afegir noves normes, mentre que en d'altres es afegir més o noves peces. Les classifico segons si faciliten o dificulten al CodeBreaker a encertar la combinació secreta.

[5] Es necessiten 10 o mes torns, per tant, en alguns casos no troba la combinació secreta.

Faciliten la partida

- **Colors no repetits:** En aquesta variació del joc no es permet que hi hagí colors repetits en la combinació secreta (però sí que n'hi poden haver en la combinació proposada pel CodeBreaker). Redueix bastant el número de combinacions secretes possibles.
- **La posició dels pins de resposta indica a quina fitxa fa referència:** Aquesta variació es basa en que els pins de resposta són assignats a les fitxes de la combinació preguntada. Per exemple, enlloc de respondre informant que dos fitxes estan bé (lloc i color) i una està però en el lloc incorrecte, la resposta seria del següent estil: la segona i la quarta fitxa estan en el seu lloc, en canvi, la primera fitxa hi és però en un lloc incorrecte (fàcilment es pot deduir que la fitxa de la primera posició hauria d'estar a la tercera). Les respostes donades pel CodeMaker donen molta més informació.

Compliquen la partida

- **Forats:** En la combinació secreta poden no haver-hi fitxes (anomenat forats). La diferència entre un forat i jugar amb un color addicional és que pels forats no es generen pins de resposta, de cap dels dos tipus (ni negres ni blancs). Augmenta el número de combinacions secretes i redueix la informació de les respostes del CodeMaker.
- **Jugar només amb pins negres:** En aquesta variació, el CodeMaker només respon amb pins negres (no informa dels blancs), per exemple, enlloc de respondre informant que dos fitxes estan be (lloc i color) i una està però en el lloc incorrecte, la resposta seria que dos fitxes estan bé. Aquesta versió normalment s'implementa en algorismes d'aprenentatge autònom i amb combinacions secretes amb moltes fitxes (els pins blancs donen molt poca informació). Les respostes donades pel CodeMaker donen menys informació.

Poden facilitar o complicar la partida

- **Mida de les combinacions:** El número de fitxes de la combinació secreta i el número de colors possibles son triades abans de començar la partida (enlloc del 4x6 clàssic (4 columnes, 6 colors)). Com més gran siguin les mides, el joc és més difícil pel CodeBreaker; com més petit, més fàcil. La variació més popular, apart de la de 4x6, és la de 5x8 (anomenada Logik). Una altra variació comuna és una amb només 2 colors.

* * *

Variació	**Combinacions possibles**	**Respostes possibles**
No repetits	360	15
La posició dels pins importa	1296	80
Forats	2401	15[6]
Només pins negres	1296	5

Taula 2: Numero de combinacions i respostes possibles

[6] Els pins de resposta donen menys informació.

5 Algorisme Machine Learning

5.1 Que és el Machine Learning

Machine Learning (o Aprenentatge Màquina) és una de les àrees en que es divideix la Intel·ligència Artificial (IA). Aquests algorismes es caracteritzen per que se'ls presenten unes dades amb les quals són capaços d'aprendre automàticament els patrons més discriminants d'aquestes.

El Machine Learning és usat principalment en motors de cerca (com Google), diagnòstics mèdics, detecció de fraus (en targetes de crèdit), reconeixements de so i imatges (i lletres), en jocs (com Forza Motorsports) o per robòtica.

Exemple d'algorismes de Machine Learning

Segons el sistema d'aprenentatge, els algorismes de Machine Learning es poden classificar en 3 categories: Aprenentatge supervisat (l'usat per l'estratègia del CodeMaker), Aprenentatge no supervisat i Aprenentatge per reforç.

L'entrenament dels algorismes d'Aprenentatge supervisat es basa en mostrar-li molts casos formats per una entrada (informació de l'estat actual) i la resposta que s'espera d'aquesta (etiqueta). Gràcies a tots els casos mostrats, l'algorisme és capaç de deduir quins són els patrons més discriminants entre els exemples de diferents etiquetes. Gràcies a aquest procés, l'algorisme és capaç de, donat un estat, triar quina és l'etiqueta que li correspon.

En l'Aprenentatge no supervisat se li mostren a l'algorisme una sèrie d'estats a analitzar, però cap etiqueta. Aquest algorisme analitza les dades dels estats i busca els patrons comuns en aquests per poder-los agrupar entre ells.

En l'Aprenentatge per reforç, l'algorisme interactua de forma directe amb l'entorn del qual rep dades constantment. Després de cada acció que modifica l'estat de l'entorn, aquest obté una puntuació en funció de quan pròxim està de la situació objectiu, per així analitzant, en cada acció realitzada sobre l'entorn, si aquesta acció ha estat beneficiosa o contraproduent. Gràcies a això, l'algorisme aprèn quines accions a de realitzar en cada estat per arribar a la situació objectiu.

Objectius a predir

Segons quin sigui l'objectiu dels algorismes de Machine Learning, aquests es poden classificar en Classificació (la que s'usa per l'estratègia de CodeMaker), Regressió, Agrupació i Reducció de dimensions.

Els algorismes de Classificació són entrenats amb Aprenentatge supervisat. Aquests algorismes, un cop entrenats, són capaços d'organitzar les entrades assignant a cada cas, la etiqueta que li correspon. Un exemple és la classificació de missatges de correu en spam o no spam.

Els algorismes de Regressió són entrenats usant també Aprenentatge supervisat. Aquest tipus d'algorismes, assignen a cada estat de l'entrada, un valor numèric (com a etiqueta). Un exemple seria calcular el valor d'un pis segons les seves característiques (número d'habitacions, mida, localització, orientació...).

Els algorismes d'Agrupació usen el mètode d'Aprenentatge no supervisat. Aquests algorismes agrupen les dades (estats) en diversos grups, on cada element d'un grup té característiques similars als altres elements que pertanyen al mateix grup. Un exemple és analitzar els usuaris d'alguna xarxa social segons els seus gustos i interessos.

Els algorismes de Reducció de dimensions usen tant l'Aprenentatge supervisat com l'Aprenentatge no supervisat. Un conjunt de dades (estats) està format per varis casos on cada cas està definit per un conjunt de característiques. Aquest algorismes intenten reduir del nombre de característiques que defineix cada estat, procurant mantenir la major discriminalitat de les dades.

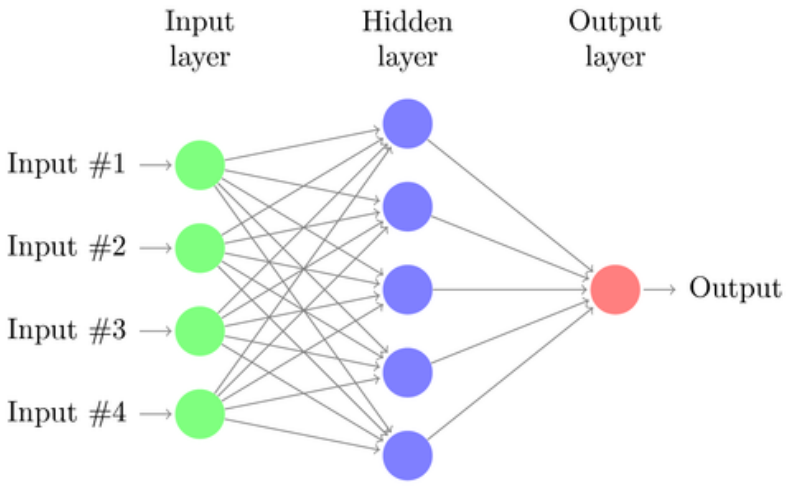
5.2 Tipus de Machine Learning

Hi ha una gran diversitat de tipus d'algorismes de Machine Learning, com els Arbre de decisions, Regles d'associació, Algorismes genètics, Algoritmes d'agrupament o Xarxes Bayesianes. En aquest projecte, ens hem centrat en els que s'adapten millor al problema actual i donen millors resultats: les Xarxa Neuronal, les Support Vector Machine i el Random Forest.

5.2.1 Xarxa Neuronal

Les Xarxes Neuronals (Neural Networks) son models de Machine Learning inspirats en les connexions neuronals del cervell humà. Les Xarxes Neuronals estan formades per neurones; una neurona és la modelització matemàtica d'una biològica, la qual rep diverses entrades (valors numèrics), amb els que realitza una suma ponderada per obtenir una resposta numèrica com a sortida.

La Xarxa Neuronal és el conjunt de neurones connectades entre si de forma estructurada. Aquest conjunt de neurones s'organitzen per capes, així anomenant a la primera capa de neurones la capa d'entrada (Input layer), l'última capa la capa de sortida (Output layer) i la resta de les capes intermiges, capes ocultes (Hidden layers). Cada neurona d'una capa qualsevol està connectada a totes les neurones de la capa anterior, definint les connexions d'entrada. En el cas de la primera capa, les connexions de cada neurona son amb les dades d'entrada de la Xarxa Neuronal (per això el nom de capa d'entrada). Cada una d'aquestes connexions conté un pes assignat, i el conjunt de tots aquests pesos defineixen el model. Cal tenir en compte, però, que a cada capa hi ha una neurona addicional (anomenada bias) per ajustar millor les entrades i sortides de les neurones (descentrament).



Il·lustració 2: Esquema capes Xarxa Neuronal

Una Xarxa Neuronal pot realitzar 2 operacions, predir un valor (donada una entrada (estat), retorna la millor sortida (etiqueta) segons els seus criteris), i entrenar la Xarxa Neuronal (gràcies a una entrada i la seva sortida, ajusta els pesos de relacions entre les neurones).

En l'operació de predir, s'assigna, a cada entrada de la Xarxa Neuronal, el valor corresponent de l'estat donat, i s'opera per ordre cada capa de neurones des de l'inici al final. L'operació realitzada per cada capa de neurones es matricial (operacions amb matrius), i aquesta es compon per una suma ponderada seguit d'una operació de no-linealitat (típicament usant la funció sigmoidal, tangent hiperbòlica, o ReLU (Rectified Linear Unit) (comú en les Xarxes de Deep Learning)). Així doncs, s'opera capa per capa fins a arribar a les sortides de les neurones de l'última capa, les quals definiran quina és la sortida (etiqueta) de la Xarxa Neuronal i, per tant, la sortida corresponent a la dada d'entrada.

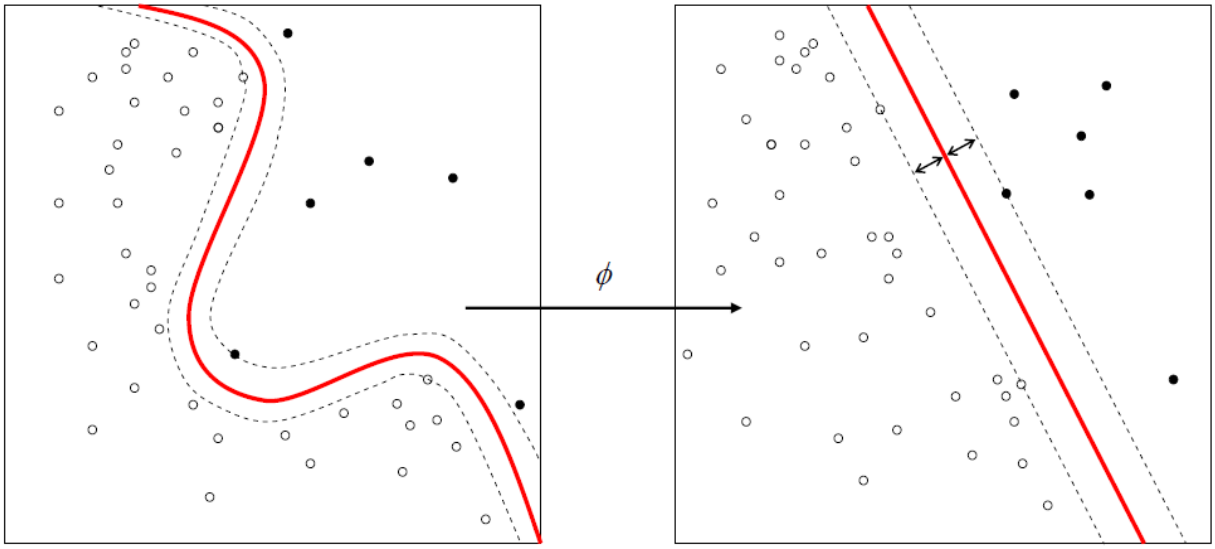
La segona operació és la d'entrenament (és més laboriosa que la de predir). La primera fase d'aquesta operació és la pròpia operació de predicció per obtenir una sortida de valors numèrics de la Xarxa Neuronal. Amb el valor numèrics que correspon la predicció de l'algorisme (que pot ser errònia), es compara amb la predicció real que ens han donat, i així li fem saber a la Xarxa Neuronal per quan s'ha confós realitzant la resta entre la sortida de valors presentada i la sortida de valors ideal corresponent a la predicció real. Amb els errors obtinguts de la última capa de la Xarxa Neuronal, és propaguen els errors a les capes anteriors usant el back-propagation. Aquest és un algorisme específic per a propagar els errors entre les neurones de capes anteriors.

Quan s'han propagat tots els errors i cada neurona de la Xarxa Neuronal té un error assignat, s'aplica un algorisme d'optimització (normalment s'utilitza l'algorisme Gradient Descent) per a realitzar la modificació dels pesos de la Xarxa Neuronal. Així doncs, aquest procés es repeteix de forma iterativa fins a arribar a convergir en una solució (combinació de pesos) suficientment pròxim a la solució ideal (o si es supera el número màxim d'iteracions permeses (definit per l'usuari)).

Un exemple d'ús de les Xarxes Neuronals és la classificació d'imatges (utilitzant un tipus molt concret anomenat Xarxa Neuronal Convolucional).

5.2.2 Support Vector Machine

L'ús principal de les Support Vector Machine (Màquina de Vectors de Suport) és classificar les dades en 2 classes. Per fer-ho, l'algorisme intenta trobar un hiperplà (de la mateixa dimensionalitat que l'entrada) que separi linealment (perfectament i amb el màxim marge possible) totes les dades en les 2 classes definides.



Il·lustració 3: Exemple de funció mapeig Support Vector Machine

És possible que les dades no es poguin separar linealment en les dos classes; en aquest cas, es transformen les dades d'entrada transformant-les, linealment, en un espai amb més dimensions, per així poder trobar un nou hiperplà (amb les noves dimensions) per separar linealment l'entrada en 2 classes.

La funció que genera l'hiperplà per separar les dades en les classes s'anomena funció de Kernel.

Hi ha diverses variacions per millorar l'algorisme:

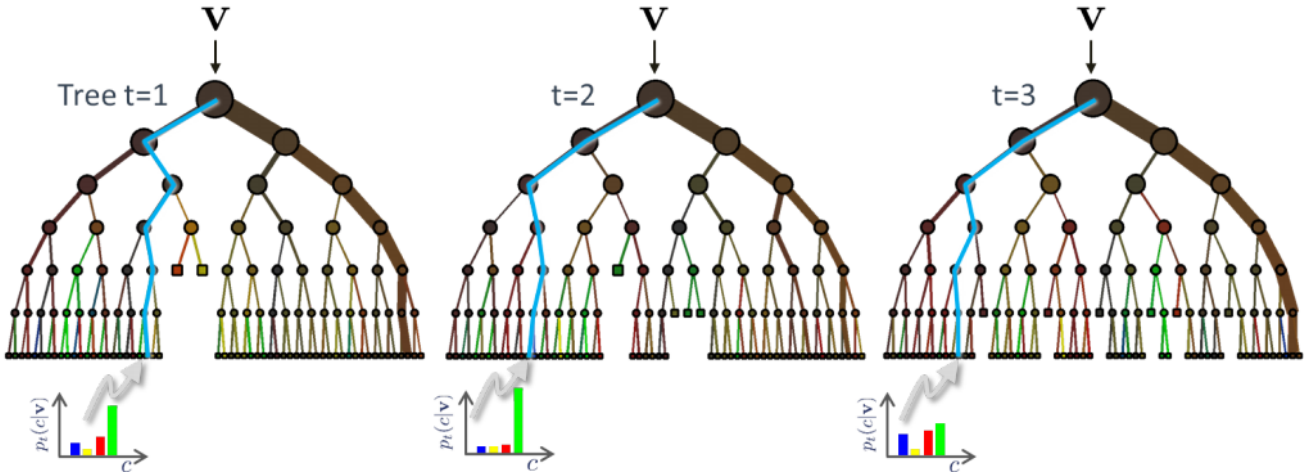
- **Marge Suau:** No es força que les dades es separin perfectament. Aquest intenta separar les dades amb un hiperplà deixant el màxim marge possible entre elles.
- **Classificacions no lineals:** La funció de mappeig de les dades inicials a espais amb més dimensions no lineals.
- **Multi-classe:** Gràcies a aquest mètode es poden classificar les dades en més de 2 classes. Per fer-ho, en creen diverses Support Vector Machine que intenta classificar, per cada classe, si les dades són d'aquella classes o de la resta (també es pot comparar cada classe amb cada una de les altres). Les noves dades a classificar, s'executen un cop per cada una de les Support Vector Machine, i gràcies als resultats obtinguts de cada una, es decideix a quina classe pertany.
- **Support Vector Regressió (Regracio de Suport de Vectors):** Una variació permet, enlloc de classificar les dades en 2 classes, permet retornar un escalar.

Aquests s'entrenen gràcies a Aprenentatge supervisat.

Les Support Vector Machine s'usen per classificació binaria i regressió (en general).

5.2.3 Random Forest

L'algorisme de Random forest (Selva Aleatòria) es basen en el coneixement col·lectiu; enlloc de tenir un gran algorisme que prediu una resposta única, s'entrenen molts algorismes petits i es tenen totes les respostes en compte.



Il·lustració 4: Exemple de decisió de diversos arbres de la Random Forest

Com el nom indica, el Random Forest es basa en un conjunt d'arbres de decisió. Cada arbre s'entrena de forma independent, per així aconseguir que cada arbre tingui pesos independents. Per l'entrenament de cada arbre, en el moment de decidir que es divideixi en subarbres (2 o més), no es tria la opció òptima de totes les possibles, sinó que es es crea un subconjunt de totes les possibles opcions i es tria l'òptima d'aquest subconjunt.

Un cop es tenen tots els arbres entrenats, es decideix quina és la resposta de l'algorisme de Random Forest donada una entrada en dos passos. El primer consisteix en obtenir les respostes individuals de cada un dels arbres de decisió entrenats del Random Forest. El segon depèn de si l'algorisme s'usa per regressió o classificació. En el cas de regressió, es calcula la mitjana de tots els arbres; en el cas de classificació, es tria la etiqueta (classe) més comuna.

Cada un d'aquest arbres s'entrena gràcies a agregació de Bootstrap aggregating.

El Random Forest s'usa per classificació i regressió.

5.3 Entrenament

El algorismes de Machine Learning que s'ha desenvolupat s'usarà en el rol del CodeMaker. Aquest algorisme tindrà en compte el següent:

- Els taulells seran de 4x6 (4 fitxes en la combinació secreta, de 6 possibles colors). Per usar-lo una mida diferent, s'hauria d'entrenat específicament per la nova mida (lògica, optimitzacions...).

- S'ha centrat en que, després de fer (i respondre) una pregunta de combinació secreta, la pregunta feta 4 torns abans deixa de ser vàlida per la resta de la partida.
- El CodeMaker pot canviar la combinació secreta després de veure la pregunta realitzada pel CodeBreaker i abans de respondre-li amb els pins de resposta.

5.3.1 Dades

Entrada de l'algorisme

L'entrada usada per entrenar l'algorisme de Machine Learning consisteix una codificació de l'estat del d'una partida. L'estat de la partida es divideix en 3 parts: Les preguntes realitzades anteriorment (amb les respostes), la pregunta actual (de la qual es vol donar la millor resposta), informació addicional (obtinguda gràcies a les preguntes i respostes realitzades) per ajudar a l'algorisme de Machine Learning a obtenir la millor resposta possible.

Per cada una de les preguntes anteriors (tres) es codifica la següent informació:

- 4 enters que representen la combinació preguntada pel CodeBreaker. La primera sempre comença igual (mirar l'apartat Optimitzacions aplicades), per tant, aquesta només té 3 enters.
- 1 enter que indica de quin tipus és la combinació preguntada.[7]
- 2 enters que indiquen el número de pins de resposta negres i blancs retornat pel CodeMaker.

Per la combinació preguntada es codifica la següent informació:

- 4 enters que representen la combinació preguntada pel CodeBreaker.
- 1 enter que indica de quin tipus és la combinació preguntada.

La informació addicional conte:

- 1 enter que indica quantes combinacions secretes possibles hi ha
- 6 enters que indiques quantes fitxes hi ha de cada color
- 2 enters que indiques quants pins de resposta negres i blancs ha respost en CodeMaker.

En total, es passen (3*(4 + 1 + 2) - 1) + (4 + 1) + (1 + 6 + 2) = 34 valors.

1 2 3 4 0 2 2 0 2 0 2 2 1 2 3 2 0 3 2 1 4 4 1 1 2 1 4 3 5 2 2 0 4 4
1 2 3 4 0 3 0 5 5 2 3 1 1 5 0 2 5 3 1 1 4 4 2 1 3 5 3 2 4 1 2 4 2 5
0 1 1 2 1 0 1 2 3 0 4 0 3 0 3 4 1 4 0 2 3 4 4 4 1 5 4 4 1 3 4 0 1 5
0 1 2 3 0 2 0 2 1 0 3 2 0 2 3 2 0 3 2 2 2 2 3 0 3 1 6 2 6 2 0 0 4 4
0 1 2 3 1 3 0 1 2 0 3 4 0 0 1 1 5 3 2 0 2 0 3 4 4 1 6 4 3 1 1 1 7 3

Taula 3: Exemple de 5 entrades de l'entrenament de l'algorisme de Machine Learning

[7] En un taulell de 4x6 hi ha 5 possibles tipus, 0000, 0001, 0011, 0012 i 0123.

Sortida de l'algorisme

L'objectiu de la sortida de l'algorisme és respondre a la pregunta sobre la semblança entre la combinació secreta que ha realitzat el CodeBreaker i la combinació secreta pensada pel CodeMaker; enlloc de respondre amb la combinació secreta pensada (mirar l'apartat Optimitzacions aplicades).

La sortida dels algorismes de Machine Learning són només un valor, però se'n volen retornar dos (el número de pins negres i el número de blancs), per tant, s'han de codificar el número de pins negres i blancs a un únic valor, el qual retornarà l'algorisme[8].

Resposta Machine Learning	**Pins negres**	**Pins blanc**
0	0	0
1	0	1
2	0	2
3	0	3
4	0	4
5	1	0
6	1	1
7	1	2
8	1	3
10	2	0
11	2	1
12	2	2
15	3	0
20	4	0

Taula 4: Exemple sortida entrenament algorismes Machine Learning

0 2 5 12 20

Taula 5: Exemple de 5 respostes de l'entrenament de l'algorisme de Machine Learning

[8] La formula per transformar la resposta a els pins es la següent : resposta = (numero de pins negre) * (numero de columnes) + (numero de pins blancs)

5.3.2 Casos especials

Taulell no complert

En els primers torns (abans de la quarta pregunta), no es té suficient informació com per omplir totes les dades d'entrada de l'algorisme de Machine Learning (no s'han realitzat les tres preguntes que es tenen en compte en l'entrada).

Per solucionar el problema una possibilitat seria crear un valor específic per indicar que no es té informació del codi preguntat (fent referència a que la posició està buida). El problema d'aquesta alternativa és que incrementa molt els casos d'entrada possibles.

S'ha optat per emplenar les dades de les preguntes que no es tenen (les fitxes de colors de combinació secreta preguntada, el tipus i els seus corresponents pins de resposta) pel valors de la primera pregunta.

Aquest procés de copiar les dades es transparent per l'algorisme de Machine Learning, ja que els programes que l'usen i l'entrenen (com l'aplicació mòbil) són els encarregats de transformar l'entrada abans de passar-li les dades.

Primera pregunta

Com en el cas del "Taulell no complert", hi ha dades que no es tenen, per tant, s'han d'omplir abans de passar-li l'entrada a l'algorisme de Machine Learning. El problema és que no es té cap dada que pugui repetir (com passava en el cas anterior), ja que el CodeBreaker només ha realitzat una pregunta (de la que espera la resposta).

Gràcies a les optimitzacions del codi (explicades a posteriorment), només hi ha cinc possibles combinacions de fitxes de colors per a la primera combinació preguntada; per cada una d'aquestes cinc possibles preguntes s'ha calculat manualment quina és la millor combinació de pins de resposta a donar (la que maximitza el número de combinacions secretes possibles).

Aquest procés de copiar les dades, no el té en compte l'algorisme de Machine Learning, sinó que l'encarregat de entrenar-lo i usar-lo (com l'aplicació mòbil) és el que transforma l'entrada abans de donar-li les dades.

Combinació	Pins resposta negre	Pins resposta blancs	Numero de respostes possibles
0000	0	0	625
0001	1	0	317
0011[9]	0	1	256
	1	0	
0012	0	1	276
0123	0	2	312

Taula 6: Resposta automàtica a la primera pregunta

[9] En aquest cas, tant les respostes 0-0, 0-1 i 1-0 donen el mateix numero de respostes possibles. S'ha descartat el 0-0 ja que dir que no hi ha cap del dos colors dona mes informació pels següents torns que indicar que un dels dos està. Tant el 0-1 com el 1-0 son equivalent.

5.3.3 Optimitzacions aplicades

Per facilitar l'entrenament de l'algorisme, i reduir el numero d'escenaris possibles, s'apliquen diverses optimitzacions a les dades usades per l'entrenament.

En aquest apartat, a cada color se l'hi ha assignat un identificador numèric (del 0 al 5), per així facilitar l'explicació del mètode[10]. També s'usarà el terme fila per referir-se a una pregunta realitzada pel CodeBreaker.

L'optimització de les dades de l'entrada està basa en 3 punts, un canvi dels colors usats, una ordenació de les columnes (a totes les files) i una ordenació de les preguntes dintre del taulell.

Primer es calcula, gràcies a la primera fila, quins seran els canvis de colors i de posicions (en aquest ordre) que s'aplicaran a totes les files; després, per cada fila, s'aplicaran els canvis de colors i de files segons les pautes que abans s'han decidit. A continuació, es calcula un pes per cada fila, segons les fitxes de colors que té cada fila. Per acabar, s'ordenen totes les files del taulell (entre elles) segons aquest pes calculat.

També s'ha optimitzat la sortida de l'algorisme. L'optimització consisteix en, enlloc de respondre amb la combinació secreta, respon amb el número de pins negres i blancs que aquesta retornaria.

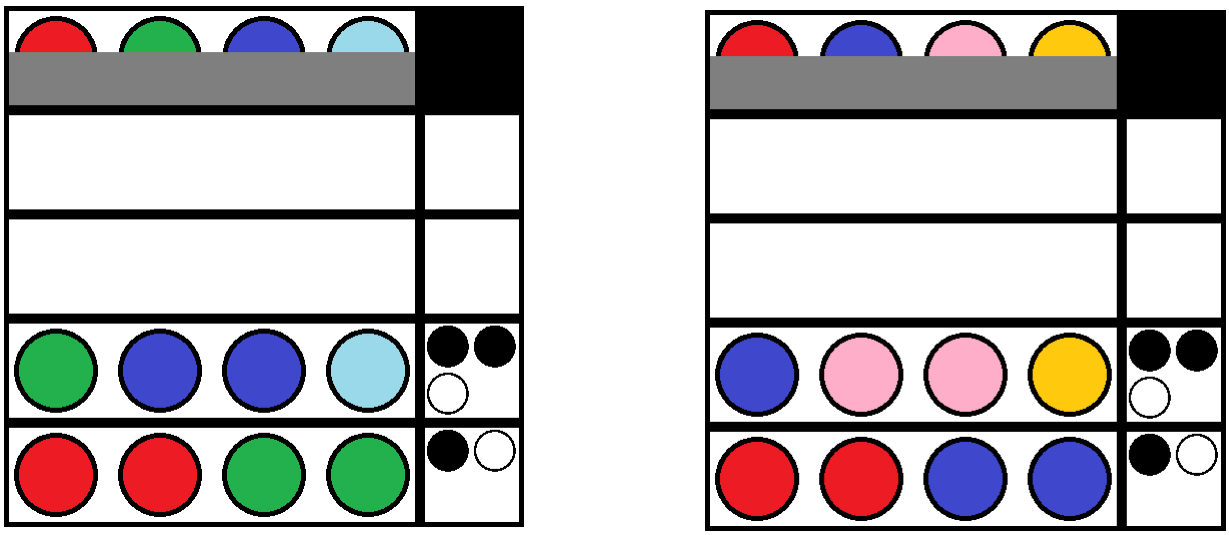
Aquestes optimitzacions de les dades, no el té en compte l'algorisme de Machine Learning, sinó que l'encarregat de entrenar-lo i usar-lo (com l'aplicació mòbil) és qui transforma l'entrada abans de donar-li les dades.

Canvi de colors de les fitxes

L'objectiu d'aquesta optimització és evitar que dos combinacions que tenen la mateixa estructura però que tots els colors d'un tipus estan canviats pels d'un altre (una o més vegades) es tractin de forma separada, ja que la lògica a aplicar és la mateixa. Per exemple, si la combinació secreta es 0123 i es pregunta per 0011 i 1223 s'ha d'actuar igual que si la combinació secreta es 0215 i es pregunta per 0022 i 2445 (en ambdós casos es respon la primera amb un pin de resposta negre i un blanc, i la segona només amb un de blanc).

El que s'intenta és modificar l'entrada a l'algorisme de Machine Learning canviant els números de cada una de les files per així usar sempre números baixos (i aconseguir que si falta algun color en el taulell, que sempre sigui el mateix color (el de valor major)).

[10] Per exemple, si els colors son vermell, verd, blau, cian, magenta i groc, a cada un se'ls assigna el 0, 1, 2, 3, 4 i 5; respectivament.



Il·lustració 5: Exemple de taulells iguals segons l'optimització per colors

Per fer-ho, es miren les fitxes de colors de la combinació més antiga preguntada pel CodeBreaker i assignar al número que està més cops repetit el valor 0, al següent més repetit el 1, i així successivament per totes les fitxes de la combinació més antiga.

(Apèndix: Algorisme generació de vector d'intercanvi dels colors de les fitxes de l'entrada (Java))

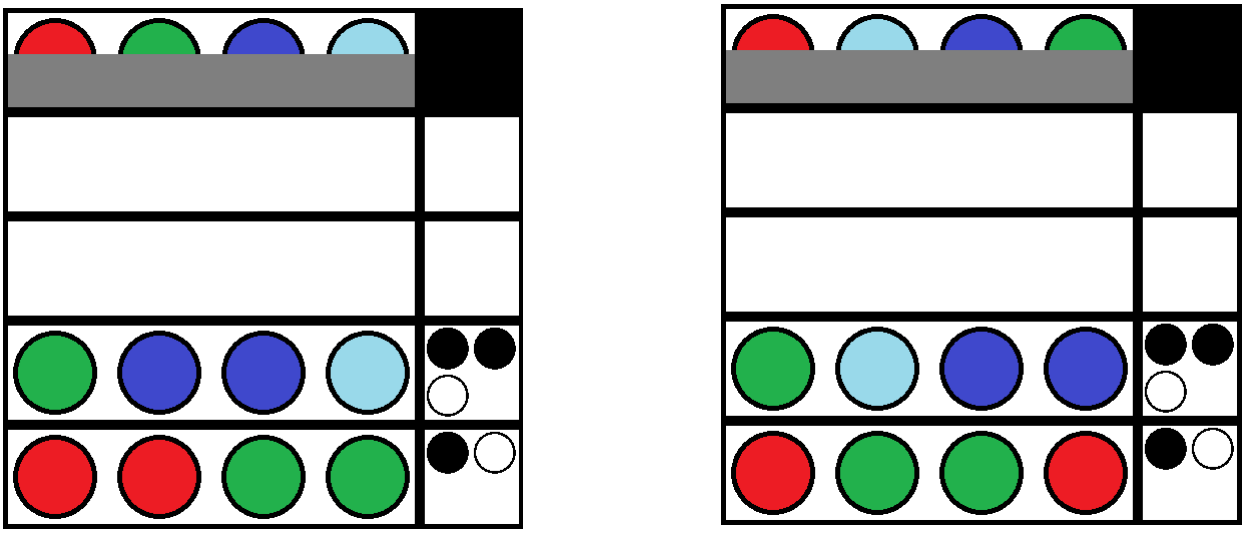
Un cop es tenen decidits tots els canvis a realitzar, aquests s'apliquen a totes les combinacions de les fitxes de colors preguntades pel CodeBreaker. Pels colors que no apareixen en la pregunta ms antiga, però si en les altres preguntes del CodeBreaker, se'ls assignarà un número quan es vagin a usar, que s'utilitzarà per les següents files.

(Apèndix: Algorisme d'intercanvi dels colors de les fitxes de l'entrada (Java))

Ordenació de l'entrada per columnes

L'objectiu d'aquesta optimització és evitar que dos combinacions que són iguals però que totes les fitxes d'una columna estan canviades per una altra (una o més vegades) es tractin de forma separada, ja que la lògica a aplicar és la mateixa. Per exemple, si la combinació secreta es 0123 i es pregunta per 0011 i 1223 s'ha d'actuar igual que si combinació secreta es 0321 i es pregunta per 0110 i 1322 (en ambdós casos es respon la primera amb un pin de resposta negre i un blanc, i la segona només amb un de blanc).

El que s'intenta és modificar l'entrada a l'algorisme de Machine Learning canviant les columnes de cada una de les files deixant els números mes petits (de la primera pregunta) tan a l'esquerra com sigui possible, i els grans a la dreta.



Il·lustració 6: Exemple de taulells iguals segons l'optimització per columnes

Per fer-ho, es miren les fitxes de colors de la combinació més antiga preguntada pel CodeBreaker i es busca a quina columna hi ha el número més petit (després de l'optimització 'canvi de colors de les fitxes'). Un cop es té, s'assigna que aquella columna ha de ser la primera després de la transformació, el següent més petit a la segona posició, i així successivament per totes les columnes de la combinació més antiga.

(Apèndix: Algorisme de generació de vector d'intercanvi de les columnes de l'entrada (Java))

Un cop es tenen decidits tots els canvis a realitzar, aquests s'apliquen a totes les combinacions de les fitxes de colors preguntades pel CodeBreaker.

(Apèndix: Algorisme d'intercanvi de les columnes de l'entrada (Java))

Després de les dos combinacions mencionades (la de canvi de colors i la de ordenació de les columnes), s'aconsegueix que la primera primera fila sempre quedi amb un dels 5 tipus possibles de combinacions en els taulell 4x6 (4 columnes, 6 files): 0000, 0001, 0011, 0012, 0123.

Gràcies a aquestes dues primeres transformacions reduïm, de les 1296 possibles valors de la fila més antiga, a només 5 possibles combinacions , generant un joc de proves complert però reduint a menys d'un 0.4% de la seva mida inicial[11].

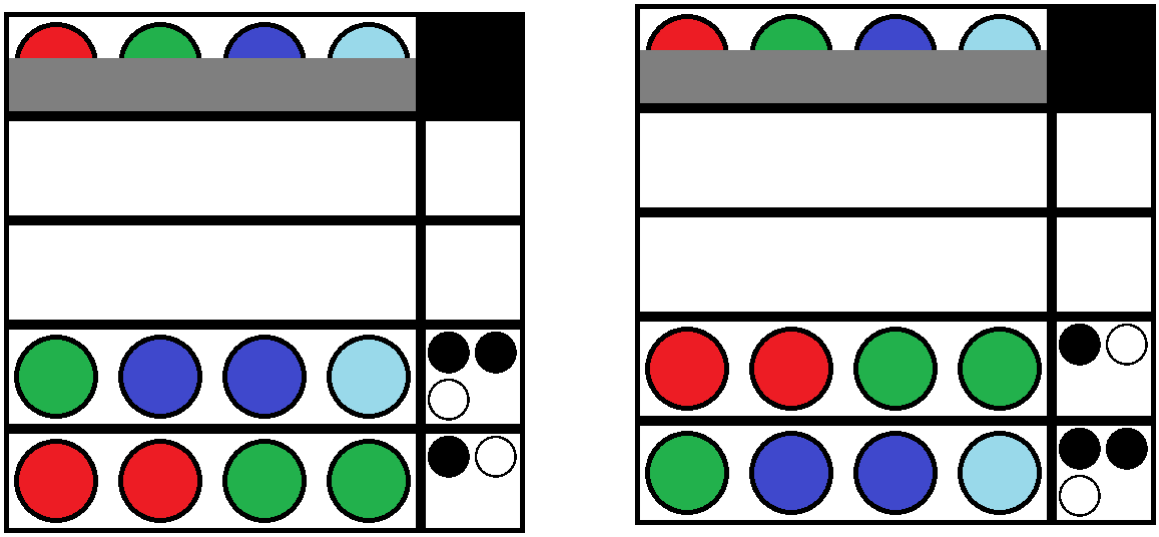
Ordenació de l'entrada per files

L'objectiu d'aquesta optimització és evitar que dos combinacions que són iguals però que una de les preguntes fetes pel CodeBreaker està canviada per una altra (una o més vegades) es tractin de forma separada, ja que la lògica a aplicar és la mateixa. Per exemple, si la combinació secreta es 0123 i es

[11] (5/(6^4))*100 = 0,3858

pregunta per 0011 i 1223 s'ha d'actuar igual que si la combinació secreta és 0123 i es pregunta per 1223 i 0011 (en ambdós casos es respon la primera amb un pin de resposta negre i un de blanc, i la segona només amb un de blanc)[12].

El que s'intenta és modificar l'entrada a l'algorisme de Machine Learning canviant les files d'ordre del taulell, deixant els que tenen números mes petits com a les primeres preguntes, i les més grans com a les últimes noves.



Il·lustració 7: Exemple de taulells iguals segons l'optimització per files

Per fer-ho, primer s'assigna un pes a cada una de les files. Com més petits siguin els números, més gran és el seu pes. A continuació es mira quin és i quina es la fila amb el número més gran i s'assigna com a pregunta més antiga. El procés és repeteix fins que s'han posat totes les preguntes.

(Apèndix: Algorisme de assignació de pesos a les files de l'entrada (Java))

(Apèndix: Algorisme de ordenació de les files de l'entrada (Java))

Gràcies a aquesta transformació reduïm a un 50% el número de taulells possibles[13].

Ja que comporta poca millora, es planteja l'opció de no usar aquesta optimització (només usar-la en taulells amb més files).

Respondre amb el pins de resposta, no amb el codi secret

Coneixent el joc, el que sembla més lògic és que l'algorisme de Machine Learning respongués amb la combinació secreta que el CodeMaker ha pensat. Un cop s'obtingui la combinació secreta, es calcularan els pins de resposta, tant negres com blans. A continuació s'informarà al CodeBreaker

[12] En veritat, no es el mateix, ja que com mes antiga sigui la pregunta, abans desapareixerà. En l'algorisme actual de Machine Learning només es te en compte la posició de la pregunta mes antiga, considerant que la posició de les altres preguntes es pot ignorar. Com es pot veure, l'exemple no es correcte (ja que canvia la posició de la pregunta mes antiga), però amb ell s'espera que els lectors entenguin la idea de la optimització.

[13] De les 4 files del taulell, només 2 es veuen afectades (la primera i la última no es modifiques), per tant, reduïm de les 2 possibles combinacions a 1; 100/2 = 50

aquests pins de resposta. A partir d'aquest punt, ja no es necessitaria més la combinació secreta. El problema d'aquesta opció és que l'algorisme té moltes possibles respostes a donar.

Per solucionar aquest problema, s'ha triat l'opció de respondre directament amb els pins de resposta; reduint així el número de respostes a només 14[14].

El possible problema d'aquest mètode seria que la informació és poc precisa (ja que els mateixos pins de resposta poden derivar de diverses combinacions secretes). Però això no comporta cap problema ja que al CodeBreaker només se li informarà dels pins de resposta, i no de la combinació secreta. Si es necessités una combinació secreta, sempre se'n podria buscar una qualsevol que complís els pins de resposta donats, ja que el CodeMaker sempre te la possibilitat canviar de codi secret si el generat si ho necessita.

Amb aquesta transformació reduïm de les 1296 possibles valors de la resposta a només 14, generant una sortida igual de potent però reduït a poc més d'un 1% de la seva mida inicial[15].

Optimitzacions externes

Un cop es té un exemple de les entrades i sortides per entrenar l'algorisme de Machine Learning, s'executarà un programa en R que avaluarà la importància de les dades de l'entrada i decidirà quines dades són importants i quines despreciables.

Per exemple, gràcies a aquesta optimització, es va eliminar la primera columna de l'entrada, ja que sempre era 0 (després de les optimitzacions).

Millores futures

Actualment, s'obtenen tant el patró de canvi dels colors com el patró d'ordenació de les columnes de la fila més antiga. Hi ha casos on seria millor triar una altra fila de les del taulell per generar els patrons més genèrics (per exemple, obtenir els patrons de la fila que tingui més colors repetits; per tenir així números més baixos després de la transformació de canvi de colors). El problema és que aquest algorismes és costós i no comporta una gran optimització (menys del 50%). S'ha decidit no implementar-lo però no es descarta usar-lo en un futur.

[14] Les possibles parelles de pins negre i pins blancs son les següents: 0-0, 0-1, 0-2, 0-3, 0-4, 1-0, 1-1, 1-2, 1-3, 2-0, 2-1, 2-2, 3-0 i 4-0

[15] (14/(6^4))*100 = 1,08

5.4 Estratègia aplicada

La primera estratègia que es va aplicar per generar les dades de l'entrenament de l'algorisme de Machine Learning era molt simple; responia amb els pins de resposta que donessin menys combinacions possibles al CodeBreaker.

La segona fase de l'estratègia, ja era més intel·ligent: tenia en compte que la pregunta més antiga que ha realitzat el CodeBreaker s'eliminarà, després de que respongui, per tant, responia amb els pins de resposta que generessin més cassos possibles sense tenir en compte la pregunta més antiga que havia fet (però comprovant que existissin, com a mínim, dos possibles combinacions que complissin totes les preguntes que encara eren vàlides).

En la tercera fase ja es va tenir molt en compte el joc i les noves estratègies que aquest comportava. L'objectiu passa a intentar evitar que la resposta digui quines són les quatre fitxes que formen la combinació secreta, ja sigui en el lloc correcte o incorrecte, ja que això dóna molta informació al CodeBreaker, i limita molt el número de combinacions secretes possibles. A més a més, aquesta informació es mantindrà moltes rondes i és fàcil de conservar.

La quarta fase ja és una millora de la tercera. Enlloc de evitar respondre amb 4 pins (blancs o negres), es crea un sistema de pesos; a cada possible combinació de pins de resposta se li assigna un valor segons la quantitat d'informació que dóna. Les combinacions que donen més informació s'intenten evitar, mentre que les que en donen poca s'usaran més cops.

	0 - 0	**0 - 1**	**0 - 2**	**0 - 3**	**0 - 4**	**1 - 0**	**1 - 1**	**1 - 2**	**1 - 3**	**2 - 0**	**2 - 1**	**2 - 2**	**3 - 0**	**4 - 0**
0000	625	0	0	0	0	500	0	0	0	150	0	0	20	1
0001	256	308	61	0	0	317	156	27	0	123	24	3	20	1
0011	256	256	96	16	1	256	208	36	0	114	32	4	20	1
0012	81	276	222	44	2	182	230	84	4	105	40	5	20	1
0123	16	152	312	136	9	108	252	132	8	96	48	6	20	1

Taula 7: Respostes possibles per cada tipus de combinació (colors de les posicions) i resposta (Negre - Blanc)

(Apèndix: Codi calcul pesos individuals entrenament (C++))

5.5 Resultats

S'han executat proves de Machine Learning amb els algorismes de Xarxa Neuronal, Suport Vector Machine i Random Forest.

En tots els entrenament s'ha usat un joc d'entrenament de 14000 partida; 1000 dades per cada possible sortida.

Xarxa Neuronal

Per les proves d'entrenament amb les Xarxes Neuronals s'han provat diferents valors del numero de neurones de la capa oculta, com del número màxim d'iteracions.

Neurones en la capa oculta	**Iteracions**	**Valor**	**Error**
10	500/500	18286,03	71,63%
10	970/1000	19220,63	73,30%
20	500/500	17125,98	71,09%
20	890/1000	18579.95	73,09%
40	500/500	15276,83	71,89%
40	940/1000	15862,53	72%
80	500/500	12880,88	74.24%
80	1000/1000	12017,41	74,05%
80	1510/2000	12417,05	73,94%

Taula 8: Resultat entrenament Xarxa Neuronal

Variant el número de neurones o de iteracions, el resultat no varia molt (l'error es manté entre el 70% i el 75%).

Support Vector Machine

S'han realitzat diversos entrenaments de la Support Vector Machine, cada un amb diferents pesos de C (el cost de classificar erròniament les classes), però en tots els casos, l'error en la predicció de la resposta era elevat.

C	Numero de Vectors de Suport	Error de training	Error de Cross Validation	Error de test	Comentari
0,1 (molt baix)	9300 (66.42%)	72,2%	79,2%	74,5%	Tots dos són elevats però no massa diferents; indicació de d'infraajust
1	9142 (65,3%)	58,6%	72,8%	68,9%	L'error de l'entrenament s'ha reduït molt, però no el de Cross Validation, indicació d'inici de sobre-ajust
10	91,25 (65,2%)	25,44%	74,1%	69,9%	L'error de l'entrenament s'ha reduït molt més, però no el de Cross Validation, indicació d'inici de sobre-ajust
100 (molt alt)	9177 (65,5%)	0,0857%	76,1%	73,5%	L'error de l'entrenament s'ha fet negligible, però el de Cross Validation s'ha mantingut; indicació de sobre-ajust complet

Taula 9: Resultat entrenament Suport Vector Machine

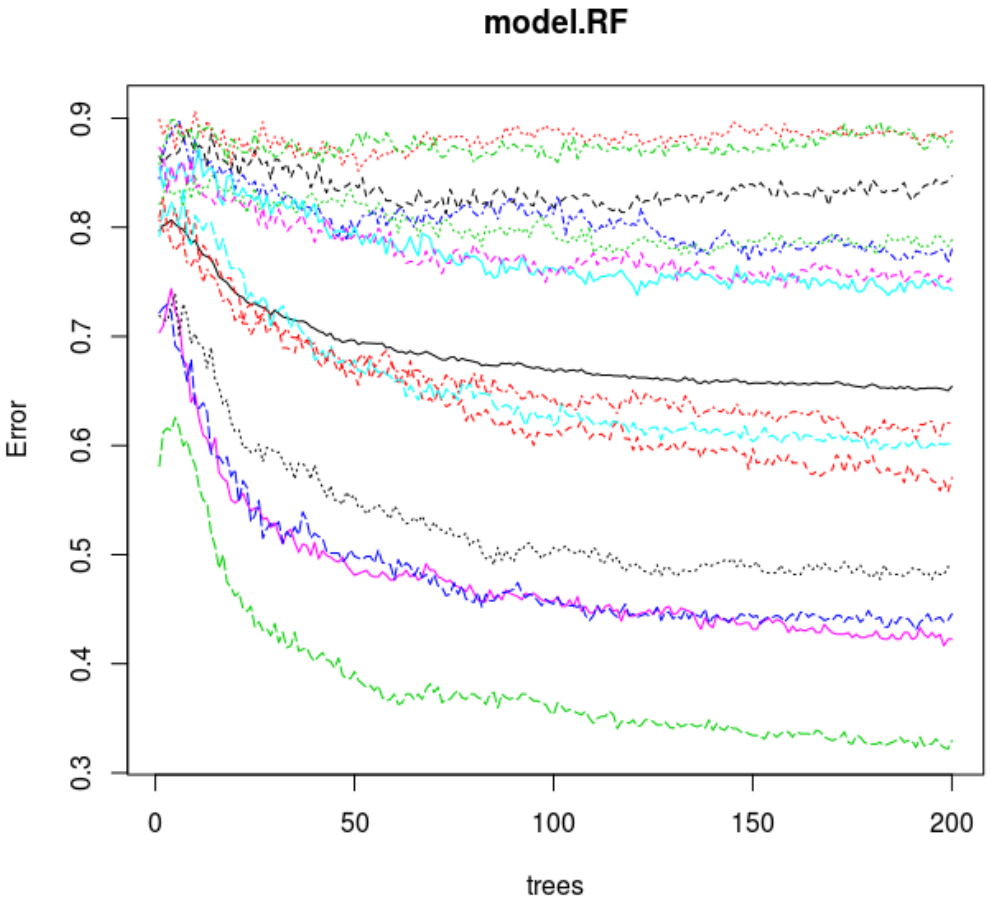
Random Forest

L'entrenament de Random Forest de classificació s'han usat 200 arbre de decisió; on cada arbre s'ha dividit cinc cops.

L'error del Random Forest es del 63% (abans d'executar l'algorisme s'havia estimat un error del 65.23%).

Les etiquetes (classes) a predir tenen una varietat; algunes són molt fàcils de predir, mentre que altres són molt complicades. Les etiquetes més difícils són les 1,2,5,6,7[16] (per sobre del 75% d'error) i les més fàcils son les 4,8,12,20[17] (per sota del 50% d'error).

[16] 0 - 1; 0 - 2; 1 - 0; 1 - 1; 1 - 2 respectivament (quantitat de pins negres i blancs).
[17] 0 - 4; 1 - 3; 2 - 2; 4 - 0 respectivament (quantitat de pins negres i blancs).



Il·lustració 8: Evolució de l'error de cada resposta segons el numero de arbre en el Random Forest

En el cas de un únic arbre de decisió, s'ha aconseguit un error de predicció del 63.8%.

6 Aplicació mòbil

6.1 Característiques

L'aplicació mòbil s'ha implementat pel sistema operatiu mòbil Android; en la versió 4.0 o superior. Per la seva implementació s'ha usat el framework AndroidStudio, framework específic per crear aplicacions d'Android.

Les proves de rendiment s'han executat en un mòbil Motorola, el "Moto G" de 2013, amb 13 GB de memòria; aquest usa la versió 5.1 d'Android (la versió més actual disponible pel mòbil).

6.2 Funcionalitats

S'ha creat una aplicació mòbil perquè l'usuari sempre que vulgui pugui jugar al Mastermind de forma fàcil.

Apart del joc clàssic, s'ha creat l'aplicació amb moltes funcionalitats addicionals, per millorar l'experiència de joc. Aquestes funcionalitats es descriuen a continuació.

Mastermind clàssic

L'aplicació mòbil es permet jugar al joc clàssic del Mastermind, amb l'usuari en el rol CodeBreaker i una màquina com a CodeMaker (que crearà la combinació secreta al principi de la partida de forma aleatòria, i respondrà a les pregunta del jugador).

Mastermind versió CodeMaker

El l'aplicació mòbil es permet jugar al la versió CodeMaker del joc Mastermind, amb les noves normes, permetent que l'usuari triï en quin rol jugar, si el de CodeMaker o el de CodeBreaker. En aquesta versió, només es guardaran les quatre últimes preguntes realitzades, i el CodeMaker podrà canviar la combinació secreta sempre que vulgui.

Estratègies de la màquina

Per jugar contra una maquina amb un rol actiu (ja sigui en el rol del CodeBreaker o en el rol del CodeMaker en la nova versió del joc), aquestes han de seguir una estratègia per actuar en el seu torn. La majoria de les estratègies son les descrites en l'apartat de 'Estudis anteriors'.

Primera resposta possible

La màquina obté una llista de totes les combinacions secretes que son possibles, i tria la primera (segons un ordre intern, usualment el 0000 es el primer i el 5555 l'últim). Aquest algorisme funciona igual per el rol de CodeMaker (la combinació que obté es la que usarà per determinar els pins de resposta) com pel de CodeBreaker (la combinació que obté és la que pregunta).

Combinació secreta actual

La màquina sempre respondrà amb la combinació secreta actual. El el cas del rol de CodeMaker, aquesta estratègia no canvia mai de combinació secreta; en canvi, en el rol de CodeBreaker, la combinació que la màquina sempre preguntarà per la combinació secreta actual (encara que no l'hauria de saber).

Respon de forma aleatòria

Aquesta és una variació de l'estratègia de "Primera resposta possible". Un cop la màquina obté la llista de totes les combinacions secretes que són possibles, enlloc de triar la primera, tria una de forma aleatòria.

Usar l'algorisme de pitjor cas possible

Calcula totes els possibles resultats de totes les possibles combinacions per decidir quina combinació triar. En el cas del rol de CodeMaker respondrà amb la combinació que deixi mes combinacions secretes possibles; en el cas del CodeBreaker preguntarà amb la combinació que, en el pitjor dels casos, elimini mes combinacions secretes possibles.

Usar l'algorisme de mida esperada

Calcula totes els possibles resultats de totes les possibles combinacions per decidir quina combinació triar. En el cas del rol de CodeMaker respondrà amb la combinació que, estadísticament, deixi més combinacions secretes possibles[18]; en el cas del CodeBreaker preguntarà amb la combinació que, estadísticament, elimini més combinacions secretes possibles.

Learning Algorithm

S'ha dissenyat un algorisme específicament pensat per les noves normes. L'estratègia i la lògica està explicada en l'apartat 'Estratègia aplicada'. Aquesta mateixa estratègia serà usada per la generació de les dades de l'entrenament de l'algorisme de Machine Learning.

Machine Learning

S'usa el resultat obtingut al preguntar a l'algorisme de Machine Learning creat i entrenat gràcies a R. Aquest només funciona si juga en la nova versió del joc Mastermind, i en el rol de CodeMaker, ja que, enlloc de retornar una combinació, aquest retorna els pins de resposta de la pregunta realitzada per el CodeBreaker.

[18] La versió de CodeMaker d'aquest algorisme es només es útil en el cas que no se sàpiga la combinació secreta. Si se sap, l'algorisme de Pitjor resposta sempre donarà millors resultats.

Primera tirada intel·ligent

La primera pregunta que el CodeBreaker realitza al CodeMaker és la més costosa de calcular ja que hi ha molts codis secrets possibles. Per altra banda, el CodeBreaker no té cap informació, per tant, sempre proposa la mateixa combinació. Per això, un cop es realitza la primera pregunta, hi ha l'opció de guardar-la en el mòbil i, quan es torni a jugar una nova partida amb les mateixes característiques (número de columnes, colors i l'estratègia de l'algorisme que juga en el rol de CodeBreaker), la usi directament, sense necessitar a pensar quina és la millor combinació.

Màquina contra màquina

Es tria l'opció que el dos rols (tant el CodeMaker com el CodeBreaker) siguin controlats per màquines amb alguna de les estratègies descrites anteriorment (poden ser diferents entre elles), es a dir, sense que cap dels dos rols sigui jugat per usuari, es jugarà una partida de forma automàtica, mostrant a l'usuari de l'aplicació mòbil el resultat final de la partida, qui ha guanyat la partida, les preguntes que s'han realitzat i les seves respostes.

Obtenir estadístiques de les partides[19]

Al acabar cada partida, es guarda la informació referent a com ha anat la partida (com el número de torns necessaris per guanyar). Tota la informació s'usarà en un futur, per la generació d'estadístiques de les estratègies.

Addicionalment, si es tria que dos màquines juguin entre elles, hi ha l'opció de fer que comencin, automàticament, una nova partida després d'acabar l'actual i guardar la informació estadística de com ha anat.

Aquesta funcionalitat no serà visible per l'usuari final; només serà usada per generar la informació estadística de, com han anat les partides.

6.3 Implementació algorisme Machine Learning

Com s'ha descrit en l'apartat anterior, l'entrenament de l'algorisme de Machine Learning s'ha realitzar amb l'eina R. Actualment, no hi ha cap mètode per migrar programes de R a Java o C++, per tant, s'ha hagut de programar l'algorisme de Machine Learning manualment, i simular les operacions que R realitzaria.

El fet de que s'han d'implementar manualment ha comportar que alguns algorismes no s'usin per l'aplicació mòbil, ja que la seva implementació seria massa costosa. Un exemple d'aquests cas es l'algorisme de Random Forest.

[19] Aquesta funcionalitat no està disponible pels usuaris.

7 Planificació

7.1 Tasques

El projecte s'ha dividit en 6 tasques: aprendre Machine Learning; crear un programa per generar les dades necessàries per entrenar l'algorisme de Machine Learning i un algorisme per comprovar la validesa d'aquest; crear i entrenar l'algorisme de Machine Learning (gràcies a les dades del punt anterior); crear una aplicació per mòbils que permetin jugar al joc Mastermind; i escriure la memòria del projecte. També hi ha una llista de tasques addicionals i extres per millorar l'aplicació (si el temps ho permet).

Aprendre Machine Learning

Les dos parts en les que es centra el projecte són les noves nores del joc Mastermind i l'algorisme que jugar en aquesta nova versió. L'algorisme que hi juga usarà Machine Learning.

El joc Mastermind l'he jugat des que era petit, per això conec totes les normes bàsiques i moltes estratègies avançades. Pel projecte, tornaré a investigar sobre el joc, tant per saber l'estat de l'art, com els seus algorismes de Machine Learning.

El projecte serà la meva primera interacció amb el Machine Learning (a excepció de una petita introducció en la universitat[20]), per tant, hauré d'aprendre com funcionen i com utilitzar-les.

Seguiré uns videocursos sobre l'aprenentatge automàtic i les Xarxes neuronals. Ambdós cursos seran per la pàgina web Coursera.

Per entrenar l'algorisme de Machine Learning, s'usarà el programa R (amb les seves llibreries NNET i CARET). Com el meu coneixement d'aquest programa es bastant limitat, hauré d'aprendre com usar-lo. També s'estudiarà usar el programa TensorFlow, com a alternativa a R per usar les Xarxes Neuronals.

Generar dades d'entrenament i validació

Per poder usar un algorisme de Machine Learning, primer se l'ha d'entrenar. L'entrenament consisteix en mostrar-li molts exemples d'estats de partides i els seus resultats esperats. Per obtenir aquestes dades, s'haurà de buscar dades ja existents, o crear un programa que les generi.

Per validar que l'estratègia de l'algorisme de Machine Learning sigui millor que la dels algorismes clàssics de Mastermind, també s'haurà de generar un programa que jugui partides comparant els dos algorismes (en nou CodeMaker i els clàssics CodeBreaker), per així poder obtenir resultats estadístics de moltes partides jugades de forma automàtica.

[20] Assignatura de Intel·ligència Artificial, FIB, UPC.

Algorisme CodeMaker

Aquesta és la part més important del projecte, la creació de l'algorisme de CodeMaker que juga aplicant les noves normes.

Aquest procés es divideix en els següents passos:

1. Pensar la idea i la lògica de l'algorisme. També en les dades que necessita.
2. Si es necessari, modificar el programa de generació de dades per obtenir els nou joc d'entrada
3. Entrenar l'algorisme (gràcies a l'eina R).
4. Comprovar i comprovar si l'algorisme de CodeMaker es prou bo com per guanyar als algorismes clàssics de Mastermind.
5. Si la solució no és suficient bona, es repeteix el procés des del punt 1.

Per causa d'aquest procés cíclic, és molt difícil calcular el temps requerit.

Aplicació mòbil

Es crearà una aplicació mòbil que permetrà jugar al joc de Mastermind, tant amb les normes clàssiques com amb les noves normes de la versió de CodeMaker.

Aquesta aplicació es programarà amb Java, usant el Framework Android Studio.

Amb aquesta aplicació es vol popularitzar el joc, tant perquè la gent hi jugui més, com per augmentar el número d'usuaris del joc. També s'espera poder obtenir beneficis econòmics en un futur.

Memòria

La memòria és una de les parts més importants del projecte. En aquesta part, es menciona, de forma detallada, el que s'ha fet en el projecte, com s'ha organitzat, l'evolució de la seva planificació, costs, metodologia...

Funcionalitats extra

Com a últim, hi ha una part d'idees les quals s'intentaran implementar en el cas que el temps ho permeti.

Algorisme CodeBreaker:

- Es crearà un nou algorisme de Machine Learning, el qual jugui en el rol de CodeBreaker amb les noves normes. Aquest algorisme serà la última validació per comprovar si el joc es equilibrat o no.

Millorar l'aplicació mòbil:

- Pujar l'aplicació a Google Play.
- Sistema de micró-pagament i anuncis incorporada a per obtenir un benefici econòmic.
- Implementació algorisme CodeBreaker: L'usuari juga conta el CodeBreaker.
- Millorar aspectes visuals de l'aplicació.
- Crear versió per iOS.

7.2 Planificació Inicial

Un cop ja es tenia la idea del projecte (modificar el joc clàssic del Mastermind i comprovar si es viable), i també es tenia decidit que es volia implementar (crear algorisme per jugar amb les noves normes i l'aplicació mòbil), es va intentar organitzar-ho en tasques més petites, d'on van sortir les 6 descrites en l'apartat anterior. Per poder decidir el millor ordre en que es realitzen les tasques, s'havia de tenir en compte els requisit de cada una respecte les altres. S'han classificat en requisits necessaris per començar una tasca, i els necessaris per poder-la acabar.

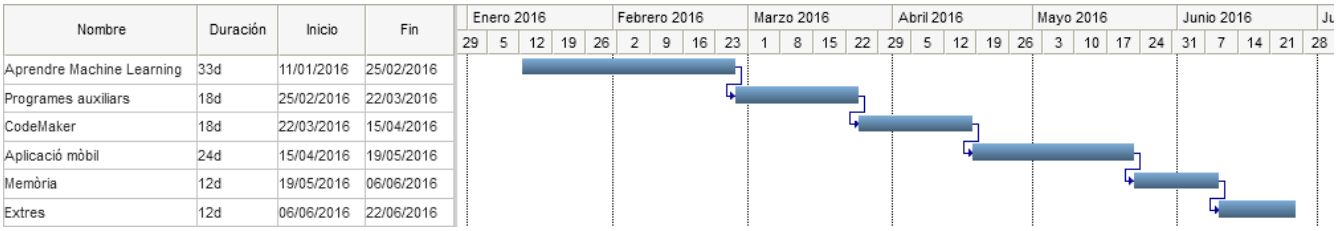
Tasca	**Necessaris per començar**	**Necessaris per acabar**
Aprendre	-	-
Programes auxiliars	Aprendre Algorisme CodeMaker (validació) *	Algorisme CodeMaker *
Algorisme CodeMaker	Aprendre Programes auxiliar (dades) *	-
Aplicació mòbil	-	Algorisme CodeMaker *
Memòria	Totes les tasques *	Totes les tasques *
Extres	Depèn de la tasca	Depèn de la tasca

* La tasca ha d'estar parcialment començada; no és necessari que estigui finalitzada.

Taula 10: Requisits en la planificació de les tasques

L'ordre que es va decidir per realitzar les tasques va ser el següent:

1. Aprendre
2. Programes auxiliars
3. Algorisme CodeMaker
4. Aplicació mòbil
5. Memòria
6. Extres



Nombre	Duración	Inicio	Fin
Aprendre Machine Learning	33d	11/01/2016	25/02/2016
Programes auxiliars	18d	25/02/2016	22/03/2016
CodeMaker	18d	22/03/2016	15/04/2016
Aplicació mòbil	24d	15/04/2016	19/05/2016
Memòria	12d	19/05/2016	06/06/2016
Extres	12d	06/06/2016	22/06/2016

Il·lustració 9: Diagrama de GANTT de la planificació inicial

7.3 Canvis en la planificació

Un cop ja s'havia acabat la tasca d'aprenentatge, i mentre s'estava realitzant la tasca de programes auxiliars, em vaig adonar que per la tasca de programes auxiliars (la part de l'algorisme de validació), es necessitaria que l'algorisme de CodeMaker estigués fet (com a mínim, parcialment), ja que necessitava saber quines dades tindria per la validació.

Es van reestructurar les tasques restants de la següent manera:

~~Aprendre~~

1. Programes auxiliars (dades)
2. Idea genèrica del algorisme CodeMaker
3. Programes auxiliars (validació)
4. Algorisme CodeMaker
5. Aplicació mòbil
6. Memòria
7. Extres

Quan es va començar a realitzar l'algorisme de CodeMaker, van sorgir complicacions, pel que es va decidir retrasar aquesta tasca i començar a realitzar l'aplicació mòbil.

El nou aspecte de les tasques era el següent:

~~Aprendre~~

~~Programes auxiliars (dades)~~

1. Aplicació mòbil
2. Idea genèrica del algorisme CodeMaker
3. Programes auxiliars (validació)
4. Algorisme CodeMaker
5. Memòria
6. Extres

7.4 Planificació Final

Després de la fita de seguiment, ja s'havia acabat la tasca de l'aplicació mòbil, es tenia la idea genèrica que com funcionaria l'algorisme i s'estava creant el programa auxiliar de validació.

Avaluant el temps restant amb les tasques que s'havien de realitzar, es va decidir que no es realitzarien les tasques extres, ja que no es tenia prou temps disponibles.

L'última planificació de les tasques era la següent:

~~Aprendre~~

~~Programes auxiliars (dades)~~

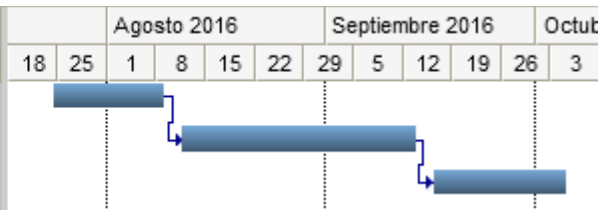
~~Aplicació mòbil~~

~~Idea genèrica del algorisme CodeMaker~~

1. Programes auxiliars (validació)
2. Algorisme CodeMaker
3. Memòria

~~Extres~~

Nombre	Duración	Inicio	Fin
Programa validació	12d	21/07/2016	05/08/2016
CodeMaker	25d	08/08/2016	09/09/2016
Memòria	15d	12/09/2016	30/09/2016



Il·lustració 10: Diagrama de GANTT de la planificació final

7.5 Justificació de desviacions

Durant l'evolució del projecte, en cada una de les tasques, s'han pres les següents decisions:

Usar R

S'usarà el programa R enlloc del TensorFlow per entrenar l'algorisme de Machine Learning. Aquesta decisió es basa en 3 punts:

- El coneixement del tutor (Lluís Belanche) de l'eina R.
- R te molta mes diversitat en tipus d'algorismes de Machine Learning (Random Forest, Support Vector Machine, Xarxa Neuronal...).
- TensorFlow està molt orientat al Deeplearning, el qual no s'aprofitaria en aquest projecte.

Generar dades entrenament

No s'ha trobat cap Base de Dades amb informació de partides del Mastermind la qual ens permetés entrenar l'algorisme de Machine Learning. S'ha creat un programa que ens generarà les dades, en les quals es podrà adaptar el seu format i mida fàcilment.

Dividir tasca algorisme CodeMaker

S'ha vist que tant la generació de dades com el testeig de l'algorisme de Machine Learning estan molt lligats a l'estructura de la d'aquest, ja que un canvi en l'algorisme (per exemple, afegir un node en l'entrada esperada) comporta la adaptació dels programes de generació de dades i validació. Aquests es poden començar un cop es coneix una idea aproximada de com serà l'algorisme de Machine Learning, però es finalitzaran alhora.

Per això, s'ha decidit dividir la tasca de l'algorisme CodeMaker en dos; la primera consistirà en decidir, aproximadament, quina estructura tindrà l'algorisme, i la segona serà la seva implementació i testeig. La primera tasca es realitzarà abans del programa auxiliar de validació.

Dividir tasca programes auxiliars

Encara que en un principi s'havien agrupar les dos tasques de crear programes auxiliars, ja que el seu us i requisits són molt diferents, s'ha decidit separar-los en dos; el programa de generació de dades per l'entrenament de l'algorisme de Machine Learning, i la validació de la seva correctesa.

Aquests dos s'implementaran en diferents punts de la planificació.

Validació de l'algorisme en l'aplicació mòbil

Enlloc de crear un programa específic per validar l'algorisme de Machine Learning, s'usarà l'aplicació mòbil per obtenir els resultats de les partides. Aquest canvi comportarà el següent:

- Re-aprofitament de codi: en l'aplicació mòbil ja es crearan diversos algorisme que jugaran com a CodeMaker i com a CodeBreaker. També es reaprofita el flux d'interaccions entre els dos rols controlats per les màquines.
- Llenguatge poc eficient: com l'aplicació mòbil es programarà en J2ME, s'obligarà que aquest algorisme també usi el mateix llenguatge, enlloc d'usar-ne algun altre més potent, com C++. També força a córrer el programa en el dispositiu mòbil, i no en un ordinador.

No reaprofitar una aplicació mòbils existents

No s'ha trobat cap aplicació mòbil la qual em pugui evitar implementar el joc per mi mateix, ja que no s'ha trobar cap codi que compleixi amb els requisits demanats. Entre ells, es demana que aquesta aplicació a reaprofitar sigui lliure, fàcil de modificar, que permeti afegir les variacions del joc. Per a això, es crearà una nova aplicació mòbil.

Triar algorismes de Machine Learning

Els algorismes que sembla que s'apropen més a les necessitats del problema són les Xarxes Neuronals, el Support Vector Machine, el Random Forest i els arbres binaris de decisió.

Cap dels algorismes provats han donat un resultat suficientment bo com per usar-lo en l'aplicació, ja que no arribaven ni al 50% (i es demanava, com a mínim, un del 10%). El que dóna millors resultat és l'algorisme de Random Forest, però és el més difícil d'implementar en l'aplicació mòbil.

7.6 Valoració econòmica

No s'espera que el projecte sigui econòmicament viable, ja que els costos són elevats en comparació amb els seus ingressos (quasi inexistents). Els ingressos únicament vindran de l'aplicació mòbil i el seu sistema de micró-pagaments (realitzat en l'apartat de tasques extres).

8 Metodologia i rigor

El projecte s'ha dividit en 6 tasques ja que la seva implementació era independent entre elles. Per això, es va decidir que a cada una de les tasques, de forma independent de les altres, se'ls hi ha aplicat la metodologia clàssica de cascada.

Cada una de les tasques s'ha separat en les següents cinc etapes de la metodologia de cascada:

- **Anàlisi:** S'estudia i s'avalua quin és el problema a tractar i quina és la solució que s'espera.
- **Disseny:** Es pensa i decideix el mètode que solucionarà el problema, quins recursos s'usaran, i com es comportaran.
- **Construcció:** S'implementa el que s'ha decidit en l'etapa de Disseny.
- **Proves:** Es comprova si el codi implementat en l'etapa de Construcció funciona com es esperat, sense error ni bugs.
- **Implantació:** S'usa la versió (final) implementa amb la version els altres programes o tasques del projecte.

Cada dos setmanes s'han realitzat reunions de seguiment per avaluar l'evolució i l'estat de les tasques que s'estaven realitzant en el moment, tant com l'estat general del projecte. Gracies a aquestes reunions s'han ajustat el desenvolupament de la tasca per complir els terminis establerts.

En cada reunió també s'ha informat i es validat dels avanços fets i els problemes sorgits durant el desenvolupament.

8.1 Camí fins la solució final

Justificació us algorisme Machine Learning

El principal motiu per el qual s'ha decidit usar un algorisme de Machine Learning es pel poc temps que aquests necessiten per calcular la resposta optima. Abans de ser usats per predir la resposta optima, han estat entrenats amb molts exemples de possibles entrades, en les que han buscat diversos patrons en aquestes; per així, en el moment de la seva execució, només ha d'aplicar el seu patró, que és molt més simple que el càlculs que realitzen altres tipus d'algorismes.

El temps de resposta es molt important ja que l'algorisme s'usarà en una aplicació mòbil, com a resposta d'una interacció de l'usuari (quan l'usuari pregunti si una combinació es la correcte o nom l'algorisme calcularà la millor resposta possible). L'usuari no pot fer res mentre s'està calculant la resposta optima; si el temps d'espera es elevat, l'experiència de l'usuari no serà bona.

El segon punt pel qual s'ha decidit usar un algorisme de Machine Learning enlloc d'un altre és que aquest s'adapta molt be a la variabilitat que te el problema. El joc Mastermind te moltes possibles variacions, com permetre usar forats o que la posició dels pins de resposta importi. Amb un algorisme

de Machine Learning, nomes es necessita que es torni a entrenar amb les noves dades de resultats de partides, i automàticament aprèn i usa les noves normes. En canvi, si s'uses algun altre tipus d'algorisme, una petita variació en les normes comportaria que s'hauria de tornar a replantejar tota la lògica de l'algorisme, ja que una petita variació pot tenir un impacte molt gran en el resultat final.

Com a última motivació, també s'ha decidit implementar un algorisme del tipus Machine Learning perquè les dades que es tenen (per calcular quina és la millor resposta) son molt fàcils de expressar com a entrada d'aquest tipus d'algoritme:

- Les dades d'entrada son l'estat del taulell i dades de realitzar simples anàlisis.
- La resposta que es vol obtenir també s'adapta al tipus de dades que retornen dels algorisme de Machine Learning de classificació; donat un conjunt de respostes possibles etiquetes (els tipus de combinacions secretes, o els pins de resposta que aquesta retornaria) és vol que informi de quina és la correcta (o la que dóna millors resultats).

Significat de joc equilibrat

Un joc és equilibrat quan els dos jugadors poden guanyar la partida; a mes a mes, la probabilitat de guanyar és semblant a la de perdre (si per cada 10 partides un jugador en guanya 9, el joc no es equilibrat (suposant que els jugadors tenen un nivell similar)). Tampoc es pot donar la situació que en l'estat inicial de la partida, un dels dos jugadors pugui aplicar una estratègia que, independentment de les accions de l'altre jugador, l'hi permeti guanyar sempre[21].

Un altre punt important és el número de preguntes que se li permeten realitzar al CodeBreaker. En el joc clàssic, el CodeBreaker pot realitzar fins a 10 preguntes; suficient per encertar la combinació secreta pels algorismes de força bruta (normalment se'n necessiten entre 4 i 6). Els únics casos on el CodeMaker guanya són en algunes partides dels algorismes de Primera resposta o Resposta aleatòria, com algunes partides contra humans.

Pel joc clàssic, no té gaire sentit canviar el número de preguntes que el CodeBreaker pot realitzar, ja que el número de preguntes necessàries per encertar la combinació secreta està determinat per la combinació secreta inicial pensada pel CodeMaker, i aquest no pot fer res durant la partida. Per cada combinació secreta, sempre es necessiten el mateix número de combinacions secretes (a no ser que els algorismes de CodeBreaker usin algun component aleatori.

En canvi, en la versió CodeMaker del joc, reduir el número de preguntes que el CodeBreaker pot ser una bona opció per equilibrar el joc, ja que la nova estratègia van canviant la combinació secreta de forma dinàmica. Això genera una gran quantitat de possibles estats, evitant que, en major mesura, el número de preguntes necessàries per encertar la combinació secreta estigui determinat per les estratègies usades i la combinació secreta inicial.

Per determinar si el joc és equilibrat o no, s'ha de comprovar quin és el percentatge de victòries del CodeBreaker. Aquest percentatge ha d'estar entre el 33% i el 66%.

[21] Degut a que es un joc amb un numero finit de combinacions, i que no es pot empatar (no s'ha definit), segur que, amb suficient potencia computacional, es pot trobar un patró de tirades que permetin guanyar sempre al mateix jugador. Degut a això, només es tindran en consideració les estratègies que no siguin massa costoses de calcular o aplicar.

8.2 Validacions

Validació de l'algorisme de Machine Learning

La validesa de l'algorisme de Machine Learning es realitza durant el seu entrenament amb la eina R.

En el programa de R, un cop s'ha triar l'estructura de l'algorisme de Machine Learning i es tenen les dades d'entrada i sortida pel seu entrenament, enlloc d'usar-les totes per entrenar l'algorisme només s'usa una fracció d'elles (per exemple, dos terços). La fracció restant s'usa per comprovar si ha aprés bé o no.

Se li demana a l'algorisme de Machine Learning que predigui quin és el resultat per cada una de les dades d'entrada que no s'han fer servir pel seu entrenament. Es comparen els resultats predits amb les dades de sortida respectives. Com menor sigui el percentatge d'error, millor.

Com a mínim, s'esperà que el percentatge d'error sigui menor del 10%, valor al quan no s'ha aconseguit arribar (el Random Forest era el més precís, però tenia un error major al 60%).

Per cada error, també s'ha de tenir en compte la seva gravetat. No és el mateix que l'algorisme respongui amb la segona millor opció que si ho fa amb la penúltima. Fins i tot es poden classificar algunes respostes no òptimes (però quasi) com a vàlides; ja que la diferència entre aquesta resposta i la millor poden ser molt petita. El problema d'analitzar la gravetat, es que canvia en cada partira. Potser en la primera entrada la resposta 2 sigui una mica pitjor que la 1 (la optima), però segons les dades de la segona estrada, la resposta 2 sigui de les pitjors (mentre que la 1 continuï siguen la millor).

Hi ha diversos motius principals pels quals l'algorisme de Machine Learning no és prou acurat:

- L'algorisme no ha après suficient. Això es pot donar perquè se l'ha entrenat amb poques dades o perquè les dades donades no són suficientment bones (li falta informació).
- La seva estructura interna no és prou potent per trobar la relació (per exemple, una Xarxa Neuronal amb molt poques neurones).
- No existeix cap relació entre les dades i el resultat (per exemple, en el Mastermind, intentar predir la combinació secreta només mirant els pins de resposta).
- L'algorisme s'ha centrat en intentar satisfer els casos concrets amb els que s'ha entrenat enlloc de descobrir quin patró hi ha en les dades de l'entrenament. Normalment aquest problema es dóna quan l'algorisme té una estructura interna massa potent (per exemple, una Xarxa Neuronal amb masses neurones en una capa interna).

Validació de joc equilibrat

Per comprovar si el joc és equilibrat o no, es realitzaran partides de forma automàtiques fent que les màquines juguin entre elles (amb diferents estratègies). No s'ha avaluat l'estratègia de Primera resposta possible, ja que es tracta la seva versió millora, la Resposta aleatòria.

Tampoc s'ha avaluat que jugui una màquina (amb qualsevol algorisme) contra humans, ja que aquestes

estadístiques podrien no ser vàlides per diversos motius: Es possible que el jugador actuï diferent depenent de contra quina màquina jugui, per així obtenir el màxim rendiment (i enlloc de avaluar com juguen els humans, s'avalua com juga per contrarestar una estratègia específica). L'altre motiu és que no totes les personen tenen el mateix nivell (s'avaluaria com de bona és una persona concreta, i no les persones en general). També s'ha de tenir en compte que les persones no juguen sempre igual, per exemple, depenent de l'estat d'ànim o el cansament la qualitat del joc pot ser molt diferent.

Encara que les estratègies de Pitjor cas i Mida esperada tenen moltes semblances, ja que el CodeMaker pot canviar la combinació secreta sempre que vulgui, la estratègia de Mida esperada sembla pitjor que la de Pitjor cas.

CodeMaker: Resposta aleatòria				
CodeBreaker	**Torns mínims**	**Torns màxims**	**Mitjana torns**	**Casos sense resoldre**
Resposta aleatòria	5	10	171/27=6,33	> 50%
Pitjor cas	4	10	2740/504 = 5.44	Sempre la resol
Mida esperada	4	21	3839/583=6,58	Sempre la resol

Taula 11: Torns per partida contra estratègia de Resposta aleatòria en rol de CodeMaker

CodeMaker: Pitjor cas				
CodeBreaker	**Torns mínims**	**Torns màxims**	**Mitjana torns**	**Casos sense resoldre**
Resposta aleatòria	3	28	3235/503 = 6.43	Sempre la resol
Pitjor cas	4	11	2597/479 =5.42	Sempre la resol
Mida esperada	4	9	2710/502 =5.4	Sempre la resol

Taula 12: Torns per partida contra estratègia de Pitjor cas en rol de CodeMaker

CodeMaker: Mida esperada				
CodeBreaker	**Torns mínims**	**Torns màxims**	**Mitjana torns**	**Casos sense resoldre**
Resposta aleatòria	3	21	4804/754 = 6.37	Sempre la resol
Pitjor cas	4	10	2357/441 = 5.34	Sempre la resol
Mida esperada	4	18	485/73 = 6.64	Sempre la resol

Taula 13: Torns per partida contra estratègia de Mida esperada en rol de CodeMaker

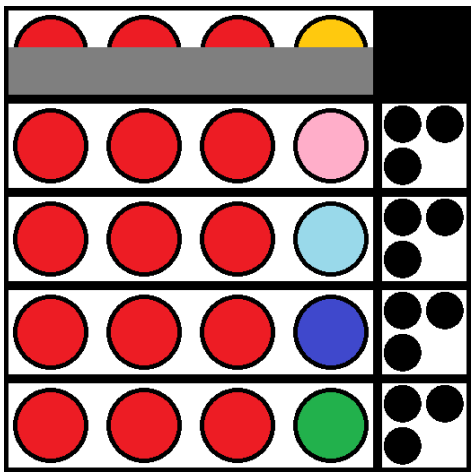
CodeMaker: Algorisme d'aprenentatge				
CodeBreaker	**Torns mínims**	**Torns màxims**	**Mitjana torns**	**Casos sense resoldre**
Resposta aleatòria	6	53	8401/556=15.11	444
Pitjor cas	7	83	12907/1000=12.91	0
Mida esperada	6	88	11441/998=11.46	2

(Apèndix: Estadístiques contra el CodeMaker amb la nova estratègia)

Taula 14: Torns per partida contra estratègia de Algorisme d'aprenentatge en rol de CodeMaker

Analitzant les taules anteriors, es pot deduir que:

- Els algorismes de CodeBreaker encerten, en el millor dels casos, la combinació secreta en només 3 o 4 torns. Això es dóna perquè l'adaptació dels algorisme existents al rol de CodeMaker no és òptima, i només aprofita la norma de canvi de la combinació.
- Els resultats dels algorisme de Pitjor cas i Mida esperada són molt semblants (encara que els de Pitjor cas son millors, tant en el rol de CodeMaker com en el de CodeBreaker). El cas de Mida esperada contra Mida esperada, el CodeMaker ho fa millor que el CodeBreaker.
- Tant el número de torns mínims necessaris per encertar la combinació secreta, com el nombre mig de preguntes necessàries per encertar la combinació secreta és el doble.
- Encara que l'estratègia de Mida esperada necessita menys torns (en mitjana) per encertar la combinació secreta que la de Pitjor cas, hi ha casos en que no l'encerta.
- Curiosament, contra el nou algorisme, l'estratègia de Pitjor cas és la que necessita mes torns mínims per encertar la combinació secreta.
- Sembla que només s'entra en bucle (el CodeBreaker no pot encertar la combinació secreta per més preguntes que realitzi) en el cas de Resposta aleatòria contra Resposta aleatòria (bàsicament si es dóna un cas de l'estil 0000-30, 0001-30, 0002-30, 0003-30).



Il·lustració 11: Exemple de bucle del CodeBreaker amb l'estratègia Resposta aleatòria

9 Pressupost i valoració dels costos

En aquest apartat es farà una estimació dels costos que comporta portar a terme el projecte.

9.1 Pressupost estimat

Costos directes

Els costos directes són els que conformen els recursos, ja siguin humans o materials que es necessiten per la realització del projecte.

Recursos humans

Per aquest projecte només es disposarà d'una persona que aplicarà els 4 rols més comuns d'un projecte (cap de projecte, analista, programador i tester). S'ha decidit assignar un cost comú depenent de la complexitat de la tasca (i no dels rols que hi intervindrien).

En total, es treballaran 136 dies, 3.5 hores al dia, fent un total de 476 hores.

Fase	**Preu per hora**	**Hores estimades**	**Preu estimat**
Aprendre	10[22] €	133 hores	1 330 €
Dades	20 €	73.5 hores	1 470 €
CodeMaker	35 €	73.5 hores	2 572.5 €
Aplicació	30 €	98 hores	2 940 €
Memòria	25 €	49 hores	1 225 €
Extra	25 €	49 hores	1 225 €
Total	22.61[23] €	476 hores	10 762.5 €

Taula 15: Costos directes

Hardware i Software

Per aquest projecte no es necessita comprar res específic, ja que s'usarà l'ordenador i mòbil personal, amb software lliure. Tampoc es realitza cap activitat que generi costos específics.

[22] Aquesta fase te un cost barat, ja que el desenvolupador també vol conèixer mes sobre tema i podrà usar els coneixements adquirits per usos futurs.

[23] Preu mitja per hora

Costos indirectes

Els costos indirectes són els que es deriven per realitzar i mantenir les instal·lacions o serveis necessaris per realitzar el projecte (un cop ja es tenen tots els recursos). En aquest projecte con els següents:

- **Consum elèctric:** 112.97 € anuals.
- **Internet:** 30 € mensuals.
- **Local:** 15 € mensuals el metre quadrat. Local de 15 metres.

Recurs	**Cost anual**	**Hores laborables anuals**	**Cost per hora**	**Hores d'ús**	**Us exclusiu**[24]	**Cost**
Electricitat	112.97 €	2008	0.056 €	294 hores	No	8.67 €
Internet	360 €	2008	0.179 €	294 hores	No	26.31 €
Local	2700 €	2008	1.345 €	294 hores	Si	395.32 €
Total	-	-	-	-	-	430 €

Taula 16: Costos indirectes

Amortitzacions

Un dels elements a tenir en compte dels recursos materials és la seva amortització. L'amortització l'aplicarem als dispositius Hardware i Software que ja disposem, que s'usen en el projecte (aplicant així, només el cost del seu ús, no el total del producte).

Tipus	**Recurs**	**Valor**	**Hores laborables anuals**	**Anys amortitzables**	**Hores amortitzables totals**[25]	**Cost per hora**	**Hores d'ús**	**Cost**
Hardware	Portàtil personal	1100 €	2008 hores	4 anys	8032 hores	0.137 €	294 hores	40.26 €
	Mòbil personal	115 €	2008 hores	3 anys	6024 hores	0.019 €	112 hores	2.14 €
Software	Office	149 €	2008 hores	1 any	2008 hores	0.074 €	49 hores	3.64 €
Total	-	-	-	-	-	-	455 hores	10.04 €

Taula 17: Cost amortitzacions

[24] Si el recurs es comparteix, només es contarà un 50% del seu cost.
[25] Es calcula una jornada laboral de 8 hores diàries. També es suposa que un any te 251 dies laborables.

Contingències

Per aquest projecte es calcula un cost de contingència del 25%, ja que les parts del projecte estan poc detallades i es pot incrementar el cost de la seva realització. Hi ha algunes parts, com la de l'entrenament de l'algorisme de Machine Learning, en el qual, el temps necessari pot ser molt diferent al previst.

Imprevistos

L'únic imprevist d'aquest projecte es la falta de temps, el qual ja s'ha tingut en compte amb la fase de tasques Extres.

Cost total

El cost total del projecte ascendeix a 16 673.18 €. Aquest es calcula sumant de tots els costos descrits en els apartats anteriors.

Tipus	**Cost**
Directes	10 762.5 €
Indirectes	430 €
Amortitzacions	10.04 €
Contingència	2 468.14[26] €
Imprevistos	0 €
Total	12 673.18 €

Taula 18: Cost total

9.2 Control de gestió

Al finalitzar cada una de les tasques definides en la planificació del projecte, es realitza un anàlisis dels costos, comparant la desviació entre els costos esperats i els reals. També s'ha realitzat la mateixa comparació pel projecte complert.

S'ha calculat els desviament de les hores estimades en comparació a les reals, i el cost total en comparació amb l'estimat.

[26] (Costos directes + Costos indirectes + amortitzacions) * percentatge de contingència.

Tasca	Tipus Desviació	Cost estimat	Cost real	Desviament[27]
Aprenentatge	Preu consum	133 hores	140 hores	5,263 %
	Preu total	1 330 €	1 400 €	
Programes auxiliars	Preu consum	73,5 hores	100 hores	36 %
	Preu total	1 470 €	2 000 €	
CodeMaker	Preu consum	73,5 hores	120 hores	63,265 %
	Preu total	2 572 ,5 €	4 200 €	
Aplicació mòbil	Preu consum	98 hores	115 hores	17,347 %
	Preu total	2 940 €	3 450 €	
Memòria	Preu consum	49 hores	60 hores	22,449 %
	Preu total	1 225 €	1 500 €	
Extres	Preu consum	49 hores	0 hores	-100%
	Preu total	1 225 €	0 €	
Total	Preu consum	476 hores	655 hores	37,605 %
	Preu total	10 762 ,5 €	12 550 €	16,609 %

Taula 19: Calcul desviacions

L'execució s'ha allargat més del previst en el càlcul inicial en totes les tasques (excepte la d'Extres, que era opcional), encara que els preus per hora s'han mantingut.

Com es pot veure, la principal desviació s'ha donat en la tasca de CodeMaker (l'algorisme de Machine Learning). Com s'ha comentat en els punts anteriors, s'ha hagut d'aplaçar un parell de cops; també s'ha invertit més temps del calculat en provar i entrenar les diferents possibilitats d'algorismes de Machine Learning.

També s'ha necessitat més temps del calculat pels programes auxiliars, sobretot pel de validació de que el joc sigui equilibrat (concretament la generació estadístiques, tant el codi com l'execució).

Com que no s'ha realitzat cap funcionalitat de la tasca de Extres, aquesta té una desviació positiva del 100%. Les hores i costos d'aquesta tasca s'han invertit en compensar les altres tasques amb una desviació negativa.

[27] Desviament = ((real / estimat) -1) * 100

10 Sostenibilitat i compromís social

Impacte social

Amb les noves normes del joc, es pretén tant que els jugadors actuals del Mastermind tinguin més diversitat en les partides; que els antics jugadors (que han jugat però ja no juguen) tornin a jugar, i que nous jugadors s'interessin pel joc. Això beneficiarà tant els jugadors les associacions i clubs on s'hi jugui.

Les partides de Mastermind són curtes, permeten als usuaris de l'aplicació jugar-hi durant els trajectes amb metro o autobús, característica important pels jugadors casuals. El Mastermind té una gran part de lògica, fet que pot limitar el seu mercat possible.

L'únic impacte negatiu que pot comportar el projecte és que la gent s'aïlli i comenci a jugar sola des de l'aplicació de mòbil, enlloc de conjuntament amb altra gent.

Impacte ambiental

Aquest projecte no té quasi impacte ambiental, ja que consisteix en crear noves normes, provar-les i crear una aplicació de mòbil, no genera ni consumeix res material (permet reaprofitar els taulells existents).

Durant la creació i programació del projecte, hi haurà un consum d'electricitat. L'ús de l'aplicació mòbil també generarà un increment l'ús de la bateria del mòbil que, de forma indirecte, podria afectar negativament al medi ambient.

Impacte econòmic

Aquest projecte no es econòmicament viable, ja que el seu cost es molt elevat i no té beneficis esperats. Tampoc satisfà cap necessitat especifica, ja que és un projecte amb un objectiu lúdic i d'aprenentatge, proposat pel mateix programador.

Els possibles beneficis d'aquest projecte serien els obtinguts en l'aplicació de mòbil, tant per Android o per iOS, gràcies un sistema d'anuncis i micró-pagaments.

Reduir impacte negatiu

Un mètode de reduir el temps del projecte, i també el seu cost, seria contractant algun desenvolupador amb experiència amb Machine Learning i la eina R, per així estalviar-se tota la part de l'aprenentatge. Això comportaria, segurament, incrementar en els preus de les altres tasques.

* * *

Sostenibilitat	**PPP**	**Vida útil**	**Riscs**
Ambiental	**Consum de disseny**	**Marca ecològica**	**Riscs ambientals**
	6	12	-2
Econòmic	**Factura**	**Pla de viabilitat**	**Riscs econòmics**
	3	6	-15
Social	**Impacte personal**	**Impacte social**	**Riscs socials**
	4	15	-5
Rang sostenibilitat	13	33	-22
	24		

Taula 20: Matriu de sostenibilitat

11 Treballs futurs

A continuació es detallen, de més a menys importància, les possible millores a realitzar en el projecte:

11.1 Millorar l'aprenentatge de Machine Learning

Una de les parts més importants del projecte era poder usar un algorisme de Machine Learning que jugués al Mastermind. No s'ha aconseguit una algorisme amb el percentatge mínim d'encert acceptable, per tant no s'ha aplicat.

S'hauria de intentar transformar les dades o canviar l'idea de com tractar les dades perquè l'algorisme de Machine Learning aprengués correctament els patrons de les dades i respondre de forma òptima.

11.2 Pujar l'aplicació a la Play Store

Una part molt important que no ha donat temps de realitzar en el desenvolupament del projecte és, pujar l'aplicació mòbil a la tenda de Google, la Play Store.

Si l'aplicació està a la Play Store, tothom podrà descarregar-se el joc; per tant, tothom qui vulgui tindrà accés al joc (tant al clàssic com amb la nova versió), augmentant així la seva popularitat.

Addicionalment, també s'obtindrà un feed-back dels usuaris, els quals informaran sobre que els sembla la aplicació i possibles millores.

11.3 Obtenir beneficis de l'aplicació mòbil

Per obtenir beneficis, s'ha de crear algun sistema amb el qual pugui guanyar diners gràcies a l'aplicació. Un cop l'aplicació estigui a la Google Play, hi ha tres mètodes amb els quals es poden guanyar diners: Requerint un pagament inicial per poder-se descarregar el joc, amb un sistema de micró-pagament i posant anuncis en l'aplicació.

Pagament per descarregar

La primera opció consisteix en demanar un petit pagament per poder-se descarregar la aplicació des de Play Store, per així ja obtenir beneficis econòmics només pel fet de que els usuaris se la descarreguin (la compren). Aquesta opció comporta tres problemes.

El primer problema de demanar diners per descarregar el joc és que aquest serà descarregat per molta menys gent. Com ja hi ha molts altres aplicacions gratuïtes que també simulen el joc Mastermind, per tant, un usuari que vulgui jugar o provar el Mastermind probablement es descarregarà una versió gratuïta que una que costi diners.

El sistema de pagament inicial és poc compatible amb els altres sistema per obtenir beneficis. Si un usuari ja ha pagat per l'aplicació un cop, no vol que se'l molesti amb anuncis o que se'ls demani que torni a pagar, ja que aquests ja han pagat un cop.

L'últim problema (però no per això menys important) és que, si es demana diners per descarregar l'aplicació, augmenta el pes de les opinions i comentaris dels usuaris, augmentant així la responsabilitat de donar un millor servei a aquests. S'han d' estar molt més a sobre de l'aplicació i les demandes que no pas amb una aplicació gratuïta.

Sistema de micró-pagaments

El sistema de micró-pagaments consisteix en demanar als usuaris que paguin per obtenir una funcionalitat, ja sigui temporalment o per sempre.

Un dels punts on es podria demanar a l'usuari que realitzi un micró-pagament seria després que aquests perdin una partida; se'ls proposarà l'opció de poder saber la combinació secreta (que no han pogut encertar) o permetrà realitzar alguna pregunta addicional.

Un altre moment on es pot realitzar aquest sistema és en mig d'una partida, permetent que amb un petit pagament obtinguin una pista de la combinació secreta, o una possible pregunta (o combinació secreta) que poden fer.

Com a últim, un dels usos més comuns del sistema de micró-pagaments és per eliminar els anuncis que es mostren en l'aplicació. Normalment els anuncis són una part molesta però acceptada. Gràcies a aquest petit pagament es milloraria l'experiència dels usuaris, i aquests també sentirien que col·laborarien en el desenvolupament del joc (amb la inversió).

Si es decideix usar aquest sistema, s'ha de decidir com si es paga directament per les funcionalitats o, si per altra banda, amb els diners reals es compren conjunts de diners del joc, amb els quals pots comprar les funcionalitats, per exemple, enlloc de pagar 0,25€ per veure la combinació secreta, pagues 5€ per obtenir diners del joc, els quals et permetran veure la solució 20 cops[28].

Anuncis

Una opció es posar anuncis que els usuaris veuran i, si els interessa, poden clicar-los per obtenir mes informació d'aquests. Els beneficis dels anuncis no s'obtenen dels usuaris, sinó directament de la Play Store.

Hi ha dos tipus d'anuncis, el de tipus banner i els de pantalla completa; aquests últims es divideixen en dos respectivament, imatges i vídeos.

Els anuncis de tipus banner són anuncis de mida petita i de forma rectangular que acostuma a posar-se a la part superior o inferior de l'aplicació mòbil. Normalment no estan de forma permanent en l'aplicació, sinó que estant durant tot el temps de vida de la pantalla (per exemple, l'anunci de tipus banner apareix al començar una partida clàssica, en acabar la partida, aquest desapareixerà).

Els anuncis de tipus pantalla completa són anuncis que apareixen de cop i ocupen tota la pantalla.

[28] El preus i les quantitats son inventades per l'exemple. No s'han de respectar en la versió final.

Normalment es mostren quan es realitza algun canvi de pantalla o s'acaba una acció (per exemple, mostrar-lo després d'acabar una partida); també hi ha l'opció que apareixen després que el jugador es passi molt de temps sense realitzar cap interacció amb l'aplicació. Encara que aquests estant menys temps que els de tipus banner, pel fet d'aparèixer de cop i ocupar tota la pantalla acostumen a ser més molestos pels usuaris, però generen més ingressos, ja que forces els usuaris a que centrin la seva atenció en aquests.

11.4 Machine Learning en el rol de CodeBreaker

Crear un algorisme de Machine Learning que jugui en el rol de CodeBreaker té una doble finalitat:

La primera finalitat és permetre que l'usuari pugui jugar en el rol de CodeMaker contra una màquina que conegui i usi les noves normes, ja que només s'ha dissenyat un algorisme en que la màquina juga en el rol de CodeMaker.

Actualment, l'usuari pot jugar contra màquines (amb estratègies simples o de força bruta) tant en el rol de CodeMaker com el de CodeBreaker, però encara que les màquines actuaran com si juguessin amb les normes clàssiques, sense aprofitar els canvis de la nova versió.

L'altre finalitat és fer jugar, de forma automàtica, a dos màquines entre elles, ambdues amb algorismes de Machine Learning, per així obtenir estadístiques sobre les partides quan ambdós jugador aprofiten les noves normes. Gràcies a aquestes estadístiques es podrà confirmar (de forma més exacta) si el joc és equilibrat o si un dels dos rols té avantatge respecte l'altre.

11.5 Adaptar l'algorisme de Machine Learning altres taulells

Un algorismes de Machine Learnings bo jugant amb les normes en les que ha estat entrenat, però depèn de quin canvi es realitzi, com per exemple la mida del taulell o el numero de colors, aquest passa a ser molt poc eficient. Per exemple, si el taulell té una columna més, aquesta no es tindria en compte, ja que en l'entrada de l'algorisme només se'n tenen en compte 4.

Per això en aquesta característica, s'han de generar jocs d'entrenament específics per cada mida de taulell (els nous jocs d'entrenament seran de diferent mida que els antics). A més a més, també s'ha de canviar l'estructura interna de l'algorisme de Machine Learning, perquè pugui llegir el nou joc d'entrenament i aprofitar-lo al màxim.

La primera part no és molt costosa, ja que el programa auxiliar de generació de dades permet generar-les amb taulell de mides diferents, encara que potser s'han de realitzar petites modificacions per generar informació addicional (com la de número de colors usats).

Modificar l'estructura de l'algorisme (i trobar l'òptima) és quasi tant costos com el que es va necessitar en el primer de l'algorisme de Machine Learning, el de la tasca de Algorisme CodeMaker.

11.6 Crear versió per iOS[29]

En temes d'aplicació mòbil, encara que Android sigui molt més popular i usat per mes usuaris, les aplicacions per sistema iOS generen molts més moviments de diners, ja que les aplicacions són més conegudes (n'hi ha menys), i els seus usuaris són més propensos a invertir diners en elles.

S'espera que una versió del del joc en iOS augmentarà els ingressos obtinguts.

El problema d'aquest punt, és que el codi que s'ha desenvolupat per l'aplicació mòbil Android no és compatible per les aplicacions per iOS, per tant, s'ha de tornar a desenvolupar tot el codi del projecte des de zero.

Un altre problema és que desenvolupar aplicacions per iOS i pujar-les a la App Store és més complicat (demanen molta més qualitat i control) i costos (les taxes per ser desenvolupador iOS són molt superiors a les d'Android).

11.7 Poder jugar amb les variacions del joc existents

En l'aplicació mòbil es permet jugar en el joc clàssic i en la versió CodeMaker. Com s'ha descrit en l'apartat de Variacions del joc (en Estat de l'art), hi ha moltes variacions del joc les quals canvien bastant la seva jugabilitat. Seria interessant permetre que els usuaris pugessin jugar de forma completa del joc i que triïn la combinació de normes que més els agradi.

Això tindrà un altre valor afegit a l'aplicació que la diferenciarà de les altres.

[29] Encara crear la versió per iOS es molt importa, degut a la complexitat i temps requerit per la seva implementació, s'ha reduït la seva prioritat.

12 Conclusions i objectius assolits

La variació de les normes del joc (el CodeMaker pot canviar la combinació secreta sempre que vulgui, però respectant les tres respostes anteriors) ha permès tant que el CodeMaker tingui un rol molt mes actiu, com augmentar la dificultat d'encertar la combinació secreta: el CodeBreaker necessita el doble de torns (mínims i en mitjana); fins i tot hi ha casos on no la pot encertar[30]. Gracies a això, el joc és més equilibrat, ja que hi ha molts casos (dels generats estadísticament) que necessiten més de deu torns per encertar la combinació secreta. L'únic punt que ha faltat és analitzar com juga una estratègia que aprofiti les noves normes en el rol de CodeBreaker.

L'entrenament i la implementació de l'algorisme de Machine Learning no ha donat els fruits esperats, ja que no s'ha arribat a obtenir-ne un prou bo com per ser usat. El motiu pel qual no ha pogut aprendre suficient bé recau en què les dades no contenen prou informació (les dades tenen molt soroll, en el sentit que un petit canvi en l'entrada varia molt la resposta final). Després de l'entrenament, l'error predictiu no s'ha pogut reduir per sota del 60% per les 14 respostes possibles (si es respongues de manera aleatòria, l'error esperat seria del 92%). L'algorisme que ha donat millor resultat ha sigut el de Random Forest (63 % d'error), seguides de les Suport Vector Machine i les Xarxes Neuronals (70% d'error).

Han sorgit dificultat en la implementació de l'algorisme de generació de dades per l'entrenament de l'algorisme de Machine Learning, ja que es molt difícil poder realitzar un joc de proves complet, ja que hi ha moltes combinacions possibles i es molt difícil arribar a cobrir tots els casos de partides possibles. També hi ha combinacions que no tenen sentit en una partida real (el joc clàssic), però que l'algorisme ha de tractar i conèixer, per no donar una resposta que ajudi al contrincant; també pot ser que amb les noves normes sí que tingui sentit (com per exemple, repetir una tirada).

S'ha implementat tota la lògica esperada en l'aplicació mòbil: poder jugar una partida amb les normes clàssiques i amb les noves normes; també s'han implementat les estratègies de la màquina bàsiques (així com algunes addicionals) pels dos rols. La part visual de l'aplicació, però, s'hauria de millorar, ja que és bastant simple i es veu inacabada. Tampoc no s'ha pujat l'aplicació a la botiga de Google, ni s'ha implementat cap sistema d'anuncis, però aquestes són tasques que no requereixen una dificultat intrínseca i es poden fer en qualsevol moment.

* * *

[30] Considerem que no pot encerta la combinació si necessita més de 100 preguntes.

Dels objectius definits inicialment valorem que s'han assolit amb la següent proporció:

- **Noves normes** **95%**
  - Rol actiu del CodeMaker 100%
  - Incrementar la dificultat del CodeBreaker 100%
  - Joc equilibrat 90%
- **Algorisme Machine Learning** **25%**
- **Aplicació mòbil** **75%**
  - Joc clàssic 100%
  - Joc amb les noves normes 100%
  - Estratègies de la màquina 100%
    - *Primera opció possible* *100%*
    - *Pitjor cas* *100%*
  - Millorar l'aparença de l'aplicació 50%
  - Pujar l'aplicació a la Play Store 00%
    - *Anuncis en l'aplicació* *00%*

13 Bibliografia

1. [Historia Mastermind] Ronald Retzel. *Bulls and Cows Mastermind ® Superhirn ® Codebreaker®* [Consultada: 14 Octubre 2016]. Disponible en internet: http://codebreaker-mastermind-superhirn.blogspot.com.es/2012/07/bulls-and-cows-mastermind-superhirn.html
2. [Mastermind] Justin Dowell. *Defeating Mastermind*. [Consultada: 27 Març 2016]. Disponible en internet: <http://mercury.webster.edu/aleshunas/Support%20Materials/Analysis/Dowelll%20-%20Mastermind%20v2-0.pdf>
3. [Mastermind] Barteld Kooi. *YET ANOTHER MASTERMIND STRATEGY*. [Consultada: 27 Març 2016]. Disponible en internet: <https://www.researchgate.net/publication/30485793_Yet_another_Mastermind_strategy>
4. [LOGIK] Albrecht Heeffer and Harold Heeffer. *NEAR-OPTIMAL STRATEGIES FOR THE GAME OF LOGIK*. [Consultada: 27 Març 2016]. Disponible en internet: <http://logica.ugent.be/albrecht/thesis/Logik.pdf>
5. [Entropy] Tom Davis. *Mastermind*. [Consultada: 27 Març 2016]. Disponible en internet: <http://www.geometer.org/mathcircles/mastermind.pdf>
6. [Evolutionary solution] Juan J. Merelo; Antonio M. Mora; Carlos Cotta; Antonio J. Fernández-Leiva. *Finding an evolutionary solution to the game of MasterMind with good scaling behavior.* [Consultada: 27 Març 2016]. Disponible en internet: <http://www.intelligent-optimization.org/LION7/Merelo-Leiva-Mora-Cotta_mm-lion13-6pages.pdf>
7. [Grid] Wayne Goddard. *A Computer/Human Mastermind Player using Grids*. [Consultada: 27 Març 2016]. Disponible en internet: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.84.8396>
8. [Expected case in 4x7 Mastermind] Geoffroy Ville. *An Optimal Mastermind (4,7) Strategy and More Results in the Expected Case* [Consultada: 10 Setembre 2016]. Disponible en internet: https://arxiv.org/pdf/1305.1010.pdf
9. [Knuth strategy] Donald E. Knuth. *THE COMPUTER AS MASTERMIND* [Consultada: 10 Setembre 2016]. Disponible en internet:
10. [Black Peg] Benjamin Doerr; Carola Doerr; Reto Spöhel; Henning Thomas. *Playing Mastermind with Many Color*. [Consultada: 27 Març 2016]. Disponible en internet: <http://arxiv.org/abs/1207.0773>
11. [Only black pegs] Michael T. Goodrich. *On the Algorithmic Complexity of the Mastermind | Game with Black-Peg Results* [Consultada: 10 Setembre 2016]. Disponible en internet: http://www.ics.uci.edu/~goodrich/pubs/ipl.pdfhttp://www.cs.uni.edu/~wallingf/teaching/cs3530/resources/knuth-mastermind.pdf
12. [Metodologia] Dr. Winston; W. Royce. *MANAGING THE DEVELOPMENT OF LARGE SOFTWARE SYSTEMS* [Consultada: 19 Juny 2016]. Disponible en internet: http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf

13. [Metodologia] Contributor Melonfire. *Understanding the pros and cons of the Waterfall Model of software development* [Consultada: 19 Juny 2016]. Disponible en internet: http://www.techrepublic.com/article/understanding-the-pros-and-cons-of-the-waterfall-model-of-software-development/
14. [Arbres de decisió] Jahdezp. *Árbol de decisión* [Consultada: 14 Octubre 2016]. Disponible en internet: http://www.utm.mx/~jahdezp/archivos%20estructuras/DESICION.pdf
15. [Random Forest] Andy Liaw; Matthew Wiener. *Classification and Regression by randomForest* [Consultada: 7 Octubre 2016]. Disponible en internet: http://www.bios.unc.edu/~dzeng/BIOS740/randomforest.pdf
16. [Type of algorithms] Justin Dowell. *Defeating Mastermind* [Consultada: 10 Setembre 2016]. Disponible en internet: http://mercury.webster.edu/aleshunas/Support%20Materials/Analysis/Dowelll%20-%20Mastermind%20v2-0.pdf
17. [Genetic algorithm] Lotte Berghman, Dries; Goossens, Roel Leus. *Efficient solutions for Mastermind using genetic algorithm*. [Consultada: 27 Març 2016]. Disponible en internet: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.483.8505>
18. [Algorismes genètics] Galeon. *ALGORITMOS GENETICOS* [Consultada: 14 Octubre 2016]. Disponible en internet: http://eddyalfaro.galeon.com/geneticos.html
19. [Neuronal Network] Christos Stergiou; Dimitrios Siganos. *NEURAL NETWORKS* [Consultada: 14 Octubre 2016]. Disponible en internet: https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html
20. [Support Vector Machine] Dell. *Support Vector Machines (SVM) Introductory Overview* [Consultada: 14 Octubre 2016]. Disponible en internet: http://www.statsoft.com/Textbook/Support-Vector-Machines
21. [Random Forest] Manish Saraswat. *A Complete Tutorial on Tree Based Modeling from Scratch (in R & Python)* [Consultada: 7 Octubre 2016]. Disponible en internet: https://www.analyticsvidhya.com/blog/2016/04/complete-tutorial-tree-based-modeling-scratch-in-python/
22. [Cursos Coursera] Andrew Ng. *Aprendizaje Automático* [Consultada: 02 Març 2016]. Disponible en internet: https://es.coursera.org/learn/machine-learning
23. [Cursos Coursera] Geoffrey Hinton. *Neural Networks for Machine Learning* [Consultada: 02 Març 2016]. Disponible en internet: https://es.coursera.org/course/neuralnets
24. [Joc Online] --- .*Guess the Colors* [Consultada: 5 Setembre 2016]. Disponible en internet: http://www.gamesforthebrain.com/game/guesscolors/
25. [Anuncis Google] Google. *Google Ads | Make money and advertise across screens and platforms.* [Consultada: 14 Octubre 2016]. Disponible en internet: https://developers.google.com/ads/#apps

14 Apèndix

14.1 Algorisme generació de vector d'intercanvi dels colors de les fitxes de l'entrada (Java)

```
std::vector<int> Row::getRenumVector() {
	// _colors     = vector, fitxes colors de la fila tractada
	// kNumColors  = numero de colors disponibles de la partida
	// kNumColumns = numero de columnes de la partida

	std::vector<int> reps = std::vector<int>(kNumColors, 0);
	std::vector<int> renumVector = std::vector<int>(kNumColors, -1);

	for (int i = 0; i < kNumColumns; ++i)
	{
		++reps[_colors[i]];
	}
	for (int i = 0; i < kNumColors; ++i)
	{
		int maxValue = -1;
		int maxPos = i;
		for (int j = 0; j < kNumColors; ++j)
		{
			if (reps[j] > maxValue)
			{
				maxValue = reps[j];
				maxPos = j;
			}
		}
		if(maxValue > 0) renumVector[maxPos] = i;
		reps[maxPos] = -2;
	}
	return renumVector;
}
```

14.2 Algorisme d'intercanvi dels colors de les fitxes de l'entrada (Java)

```
std::vector<int> Row::renumRow(std::vector<int> renumVector)
{
    // renumVector = el valor de la fitxa i passa valer j
(renumVector[i] = j)
    // _colors     = vector, fitxes colors de la fila tractada
    // kNumColumns = numero de columnes de la partida

    for(int i = 0; i < kNumColumns; ++i)
    {
           int newColor = renumVector[_colors[i]];
           if (newColor == -1)
           {
                 for (int j = 0; j < renumVector.size(); ++j)
                 {
                       if (renumVector[j] >= newColor) newColor =
renumVector[j] + 1;
                 }
                 renumVector[_colors[i]] = newColor;
           }
           _colors[i] = newColor;
    }
      return renumVector;
}
```

14.3 Algorisme de generació de vector d'intercanvi de les columnes de l'entrada (Java)

```
std::vector<int> Row::getReorderVector()
{
      // _colors     = vector, fitxes colors de la fila tractada
      // kNumColors  = numero de colors disponibles de la partida
      // kNumColumns = numero de columnes de la partida

    std::vector<int> base = _colors;
    std::vector<int> reorderVector = std::vector<int>(kNumColumns);
    for(int i = 0; i < kNumColumns; ++i)
    {
        int minColor = kNumColors+1;
        int minPos = 0;
        for(int j = 0; j < kNumColumns; ++j)
        {
            if(base[j] < minColor)
            {
                minColor = base[j];
                minPos = j;
            }
        }
        reorderVector[minPos] = i;
        base[minPos] = kNumColors+1;
    }

    return reorderVector;
}
```

14.4 Algorisme d'intercanvi de les columnes de l'entrada (Java)

```
void Row::reorderRow(std::vector<int> reorderVector)
{
    //reorderVector = el valor de la columna de la pos i passa a la pos j
(reorderVector[i] = j)
    // _colors      = vector, fitxes colors de la fila tractada
    // kNumColumns  = numero de columnes de la partida

    std::vector<int> base = _colors;
    for(int i = 0; i < kNumColumns; ++i)
    {
        _colors[reorderVector[i]] = base[i];
    }
}
```

14.5 Algorisme de assignació de pesos a les files de l'entrada (Java)

```
int Row::getRowValue(){
      // _colors     = vector, fitxes colors de la fila tractada
      // kNumColors  = numero de colors disponibles de la partida
      // kNumColumns = numero de columnes de la partida

    int rowValue = 0;
    for(int i = 0; i < kNumColumns; ++i)
    {
        rowValue += (int) pow(kNumColors+1, kNumColors - _colors[i] - 1);
    }
    return rowValue;
}
```

14.6 Algorisme de ordenació de les files de l'entrada (Java)

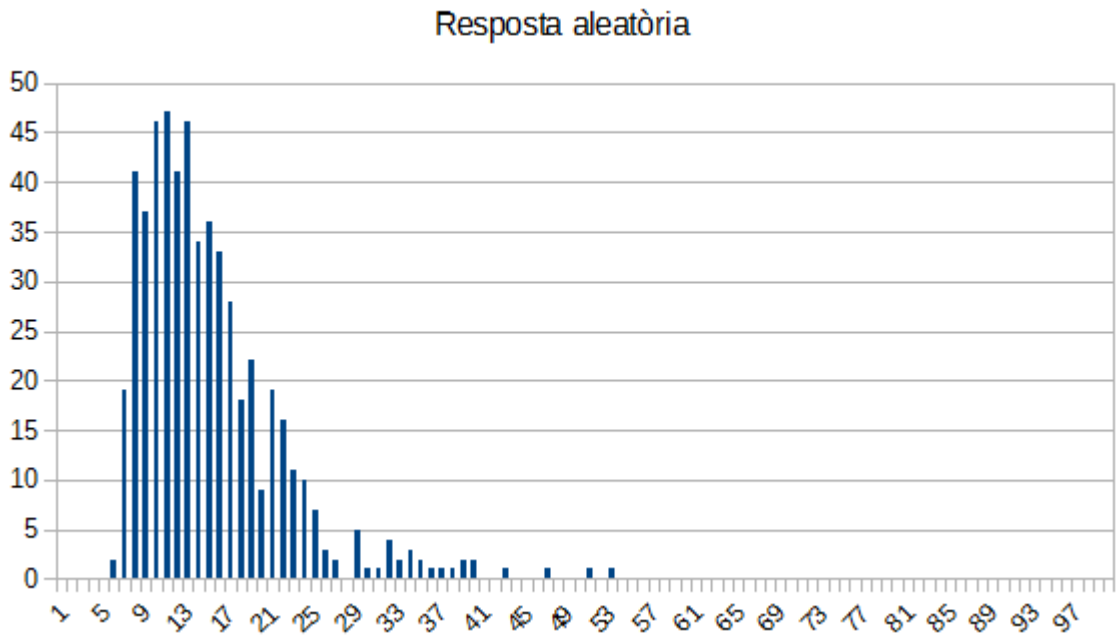
```
void Board::sortBoard()
{

    [...]

    // _colors  = vector, fitxes colors de la fila tractada
    // kNumRows = numero de files que hi ha en la partida

    for(int i = 1; i < kNumRows -1; ++i)
    {
        int maxValue = -1;
        int maxPos = i;
        for(int j = i; j < kNumRows; ++j)
        {
            if(rowValues[j] > maxValue)
            {
                maxValue = rowValues[j];
                maxPos = j;
            }
        }
        if(codeDebug) globals->log("swap: "+ globals->intToString(i) +" -
"+ globals->intToString(maxPos));
        Row* temp = _rows[maxPos];
        _rows[maxPos] = _rows[i];
        _rows[i] = temp;

        int tempInt = rowValues[maxPos];
        rowValues[maxPos] = rowValues[i];
        rowValues[i] = tempInt;
    }
}
```

14.7 Codi calcul pesos individuals entrenament (C++)

```
bool isValid(std::vector<int> res, std::vector<int> code, std::vector<int>
secret)
{
	for (int i = 0; i < code.size(); ++i)
	{
		if (code[i] == secret[i])
		{
			--res[0];
			code[i] = -1;
			secret[i] = -2;
		}
	}
	for (int i = 0; i < code.size(); ++i)
	{
		for (int j = 0; j < secret.size(); ++j)
		{
			if (code[i] == secret[j])
			{
				--res[1];
				code[i] = -1;
				secret[j] = -2;
			}
		}
	}

	return res[0] == 0 && res[1] == 0;
}


int main()
{
	// List with all possible results (0-0, 0-1, 0-2, 0-3, 0-4, 0-5, 0-6,
1-1, 1-2, 1-3, 2-0, 2-1, 2-2, 3-0, 4-0)
	std::vector<std::vector<int> > res;
	// List with all possible initial codes type (0000, 0001, 0011, 0012,
0123)
	std::vector<std::vector<int> > codes;
	// List with all possible codes (0000, 0001, 0002, 0003, 0004, 0005,
```

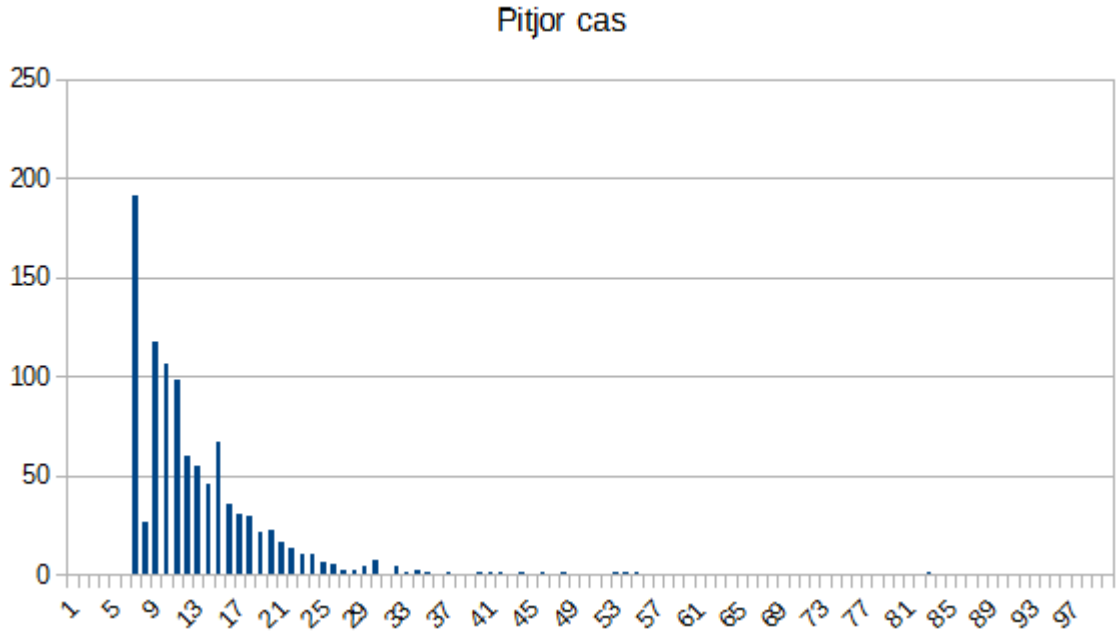
```
0010, ... , 5554, 5555)
	std::vector<std::vector<int> > allCodes;
	// Table with the result of possible answers
	std::vector<std::vector<int> > table;

	// Initialize the vectors with a 6 colors 4 columns board
	initVectors6x4(res, codes, allCodes, table);


	for (int i = 0; i < res.size(); ++i)
	{
		for (int j = 0; j < codes.size(); ++j)
		{
			int numPossibles = 0;
			for (int k = 0; k < allCodes.size(); ++k)
			{
				if (isValid(res[i], codes[j], allCodes[k]))
				{
					++numPossibles;
				}
			}
			table[i][j] = numPossibles;
		}

		std::cout << "Res: " << res[i][0] << " " << res[i][1] << "
DONE" << std::endl;
	}


	std::cout << std::endl;
	std::cout << "Table: " << std::endl;
	printTable(table, codes, res);
	std::cout << std::endl;

    return 0;
}
```

14.8 Estadístiques contra el CodeMaker amb la nova estratègia



(No te en compte les 444 partides que el CodeBreaker no ha encertat la combinació secreta)

* * *



* * *

(No te en compte les 2 partides que el CodeBreaker no ha encertat la combinació secreta)