ENTREGA 1

PROP GRUP 12.1

Aina Gomez
Gerard Oliva
Pere Mir
Pol Contreras

Índex

1. Casos d'ús.

1.1. Diagrama casos d'ús.

A continuació adjuntem una imatge del diagrama dels casos d'ús de la nostra aplicació 'Mastermind'. Podreu trobar una versió ".pdf" el repositori de gitlab.

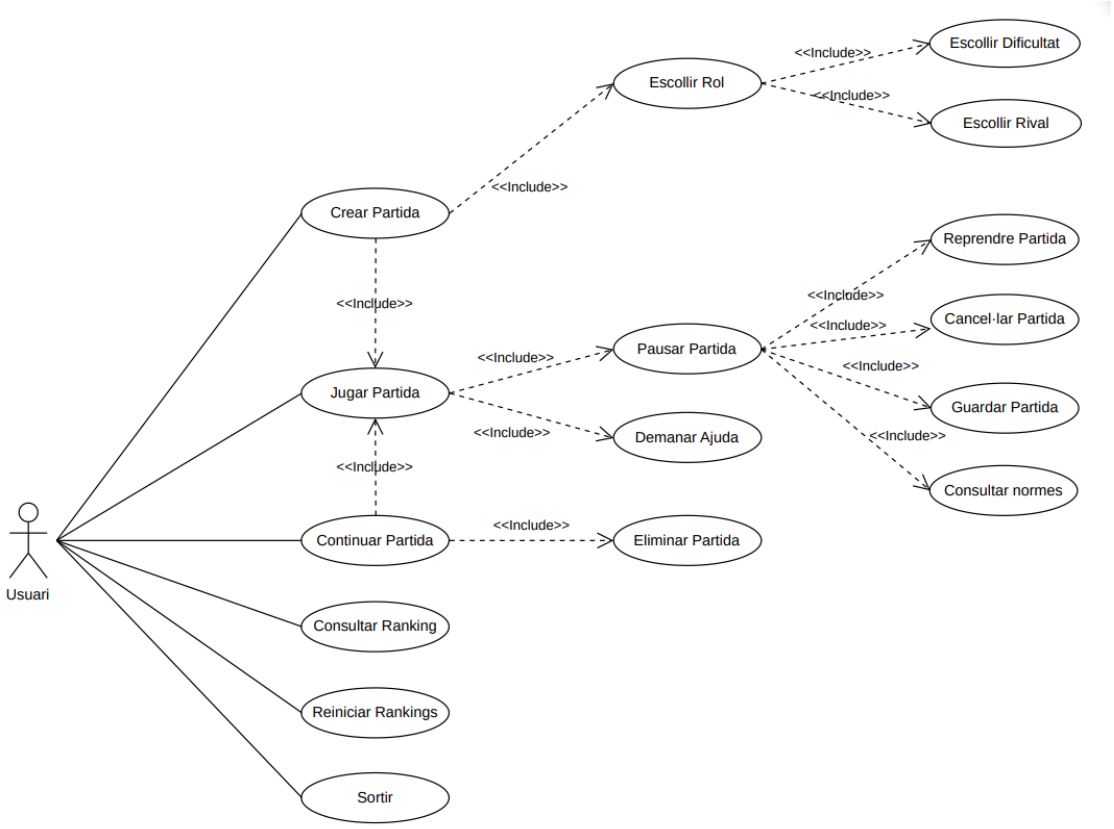

Figura 1: Diagrama de casos d'ús.

1.2. Descripcions Casos d'ús

Crear Partida

Nom: Crear partida.
Actor: Usuari.
Comportament: 1. L'usuari indica que vol crear una partida. 2. L'usuari pot indicar el rol que desitja jugar, salta al cas d'ús "Escollir rol". 3. El sistema comença la partida amb la confirmació de l'usuari i salta al cas d'ús "Jugar Partida".
Errors possibles i cursos alternatius: • L'usuari no ha seleccionat cap rol, el sistema avisa de l'error.

Escollir Rol

Nom: Escollir rol.
Actor: Usuari.
Comportament: 1. L'usuari selecciona el tipus de rol que vol jugar: *codebreaker* o *codemaker*. 2. Si el rol escollit és *codebreaker*, saltarà al cas d'ús "Escollir Dificultat".

Errors possibles i cursos alternatius: • L'usuari no ha seleccionat cap rol, el sistema avisa de l'error. • L'usuari selecciona el rol *codemaker*. Saltarà al cas d'ús "Escollir Rival".

Escollir Dificultat

Nom: Escollir dificultat.
Actor: Usuari.
Comportament: 1. L'Usuari selecciona la dificultat de la partida que vol jugar: fàcil, normal o difícil.
Errors possibles i cursos alternatius: • L'usuari no ha seleccionat cap dificultat, el sistema avisa de l'error.

Escollir Rival

Nom: Escollir rival.
Actor: Usuari.
Comportament: 1. L'Usuari selecciona un dels dos rivals: *five guess* o *genetic.*
Errors possibles i cursos alternatius: • L'usuari no ha seleccionat cap rival, el sistema avisa de l'error.

Continuar Partida

Nom: Continuar partida.
Actor: Usuari.
Comportament: 1. L'usuari indica que vol continuar una partida. 2. El sistema mostra les diferents partides guardades (20 com a màxim). 3. L'usuari selecciona la partida guardada que vol continuar. 4. El sistema càrrega la partida que ha seleccionat i salta al cas d'ús "Jugar Partida".
Errors possibles i cursos alternatius: • L'usuari no té cap partida guardada, el sistema mostra un missatge indicant que l'usuari no té partides guardades.

Jugar Partida

Nom: Jugar partida.
Actor: Usuari.
Comportament: 1. El sistema pregunta els torns de la partida. 2. L'Usuari introdueix el nombre de torns que desitja jugar. 3. El sistema inicia una partida amb els valors definits per l'usuari prèviament. 4. Si l'usuari juga com a *codemaker*, el sistema demanarà que introdueixi el codi a encertar. 5. El sistema mostrarà la resposta del *codebreaker* i esperarà que introdueixi el nombre de fitxes encertades i el nombre de colors encertats. 6. Es repeteixen els casos 4 i 5 fins a la finalització dels torns o fins que el *codebreaker* encerta el codi proposat pel *codebreaker.* 7. Un cop finalitzada la partida, el sistema actualitzarà les estadístiques de l'usuari i el ranking dels jugadors.

Errors possibles i cursos alternatius: • L'Usuari introdueix un nombre de torns fora del interval [6,12]. • Si l'usuari juga com a *codebreaker*. A cada torn, el sistema li demanarà que introdueixi el possible codi i li mostrarà el nombre d'encerts. • L'usuari introdueix un color que no es troba dintre dels colors disponibles en la dificultat seleccionada, el sistema avisa del error. • L'usuari (jugant com a *codemaker*) introdueix el nombre d'encerts incorrectament, el sistema avisa de l'error i repeteix la operació. • L'usuari tanca l'aplicació sense guardar, el sistema no guardarà l'estat de la partida. • L'usuari pausa la partida, el sistemons: a saltarà al cas d'ús "Pausar Partida" i aturarà el temps. • Al finalitzar la partida, si l'usuari jugava com a *codebreaker*, el sistema li demanarà un nom d'usuari per introduir les dades de la partida al sistema de ranking.

Consultar Ranking

Nom: Consultar ranking.
Actor: Usuari.
Comportament: 1. L'usuari indica que desitja consultar el ranking de la aplicació. 2. El sistema mostra un llistat ordenat de millor a pitjor d'entre totes les partides guanyades per usuaris com a codebreaker.
Errors possibles i cursos alternatius: • No hi ha estadístiques de cap usuari, per tant, el llistat és buit, el sistema ho indica amb un missatge d'error.

Reiniciar Ranking

Nom: Reiniciar ranking.
Actor: Usuari.
Comportament: 1. L'usuari indica que desitja reiniciar el ranking de la aplicació. 2. S'esborren totes les partides guardades del ranking i aquest queda un altre cop buit.
Errors possibles i cursos alternatius: • No hi ha estadístiques de cap usuari, per tant, no hi ha res a borrar, el sistema ho indica amb un missatge d'error.

Pausar Partida

Nom: Pausar partida.
Actor: Usuari.
Comportament: 1. L'usuari indica que desitja pausar la partida, aturant així el comptador de temps. 2. El sistema mostra un "menú" de pausa, on podem trobar les opcions: reprendre la partida, guardar-la, cancel·lar-a o bé consultar les normes de joc. Cada opció salta al cas d'ús "Reprendre Partida", "Guardar Partida", "Cancel·lar Partida" i "Consultar normes", respectivament.
Errors possibles i cursos alternatius:

• La partida ja estava pausada, aquesta acció no té cap efecte i la partida segueix pausada.

Reprendre Partida

Nom: Reprendre partida.
Actor: Usuari.
Comportament: 1. L'usuari indica que desitja reprendre la partida que prèviament ha pausat, tornant a posar en marxa el comptador de temps. 2. El sistema torna al cas d'ús "Jugar Partida".
Errors possibles i cursos alternatius: • La partida no estava en pausa, no es produeix cap canvi.

Guardar Partida

Nom: Guardar partida.
Actor: Usuari.
Comportament: 1. L'usuari indica que desitja guardar la partida. 2. El sistema guarda la partida a les partides guardades del jugador. 3. El sistema pregunta si l'usuari desitja seguir amb la partida. 4. Si l'usuari confirma, continua la partida.
Errors possibles i cursos alternatius: • L'usuari no desitja seguir amb la partida, el sistema surt de "Jugar Partida" i va al menú principal. • El sistema no té prou memòria, el sistema informa de l'error. • L'usuari ja té 20 partides guardades, el sistema sobreescriurà la partida guardada més antiga, si l'usuari confirma l'acció. • Si la partida ja es troba a partides guardades, es sobreescriu a ella mateixa.

Cancel·lar Partida

Nom: Cancel·lar partida.
Actor: Usuari.
Comportament: 1. L'usuari indica que desitja cancel·lar la partida que està jugant actualment. 2. El sistema demana a l'usuari que confirmi l'acció. 3. Si l'usuari confirma, el sistema cancel·la la partida i surt de "Jugar Partida".
Errors possibles i cursos alternatius: • L'usuari no confirma l'acció, el sistema avorta la operació. • La partida que es vol cancel·lar ja ha estat finalitzada, el sistema avisa de l'error.

Consultar normes

Nom: Consultar normes.
Actor: Usuari
Comportament: 1. L'usuari indica que vol consultar les normes del joc. 2. El sistema mostra la normativa del joc a l'usuari.

Eliminar Partida

Nom: Eliminar partida.
Actor: Usuari.
Comportament: 1. L'usuari selecciona una de les seves partides guardades i indica que vol eliminar-la. 2. El sistema mostra un missatge de confirmació. 3. L'usuari confirma que desitja eliminar la partida.
Errors possibles i cursos alternatius: • L'usuari no confirma la eliminació de la partida, el sistema avorta la operació. • L'usuari no té cap partida guardada, el sistema avisa de l'error.

Demanar Ajuda

Nom: Demanar ajuda.
Actor: Usuari.
Comportament: 1. L'usuari (que juga com a *codebreaker*) indica que vol rebre ajuda per al torn en el que es troba. 2. El sistema li mostra el codi proposat a l'usuari. 3. L'usuari pot decidir si jugar el codi proposat o amb un altre escollit per ell.
Errors possibles i cursos alternatius: • L'usuari ha arribat al màxim de les ajudes disponibles, el sistema avisa de l'error.

Sortir

Nom: Sortir.
Actor: Usuari.
Comportament: 1. L'usuari informa que desitja tancar la aplicació. 2. El sistema farà un guardat de l'estat de la aplicació. 3. El sistema tanca la aplicació.
Errors possibles i cursos alternatius: • Si l'usuari estava jugant a una partida no guardada, el sistema no guardarà l'estat d'aquesta.

2. Diagrama del model conceptual

2.1. Disseny

A continuació podeu veure el diagrama del model conceptual de la nostra aplicació. Aquest diagrama també el podeu trobar a la carpeta 'DOCS' del repositori gitlab.

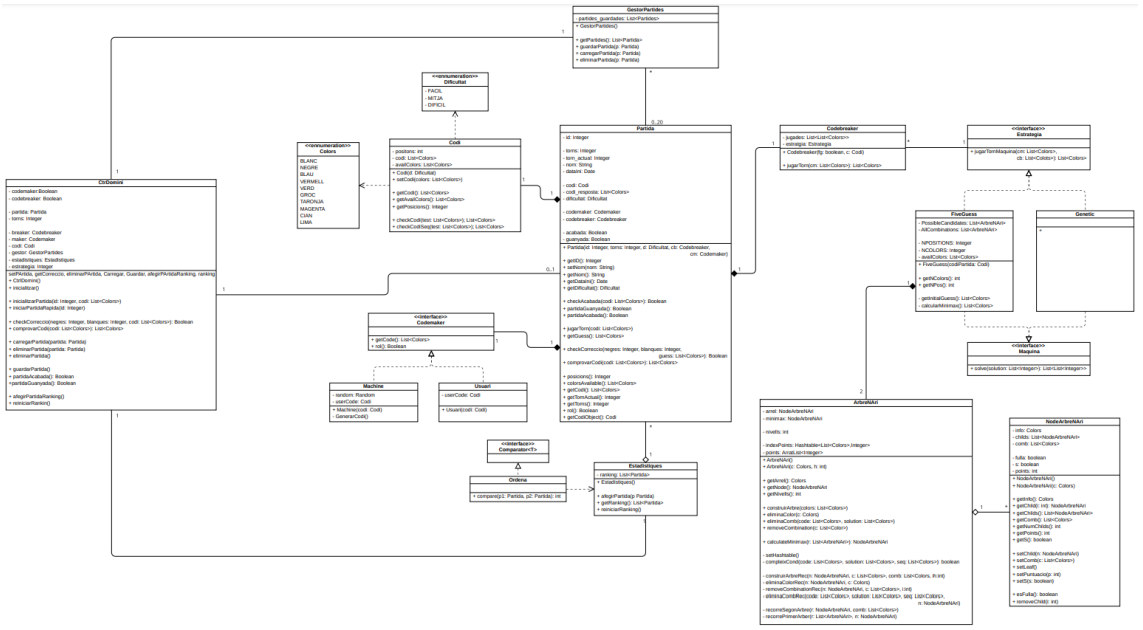

Figura 2: Diagrama del model conceptual de l'aplicació 'Mastermind'.

2.2. Descripció

Classe Gestor de Partides.

Nom de la classe: GestorPartides
Breu descripció: Classe que guarda les partides sense acabar que l'usuari podrà reprendre en un altre moment.
Cardinalitat: Una sola classe de GestorPartides per a totes les partides que es guardin, i moltes partides (amb un màxim de 20) poden ser guardades pel gestor.
Descripció dels atributs:

- partides_guardades: conjunt de partides guardades

Descripció de les relacions:

- Relació amb la classe "Partida": indica les partides no acabades que s'han guardat en el gestor

Descripció dels mètodes públics:

- GestorPartides(): constructora per defecte que inicialitza un gestor de partides sense cap partida guardada.
- void guardarPartida(Partida p): s'afegeix la partida p al conjunt de partides guardades
- Partida carregarPartida(Partida p): retorna la partida p
- void eliminarPartida(Partida p): s'elimina la partida p del conjunt de partides guardades

Classe implementada per: Pol Contreras

Classe Partida

Nom de la classe: Partida.
Breu descripció: La classe Partida fa referència amb les altres classes per tal de modelar les funcionalitats d'una partida al MasterMind.
Cardinalitat: Una sola classe partida, per als diferents atributs que conté.
Descripció dels atributs:

- Id: Identificador de la partida.
- dataIni: Data d'inici de la partida.
- torns: Nombre màxim de torns que el codebreaker té per acabar la partida.
- torn_actual: Torn actual.
- codi: Codi de la partida, que ha escollit el codemaker al principi d'aquesta.
- Dificultat: dificultat de la partida.
- codemaker: Classe codemaker.
- codebreaker: Classe codebreaker.
- codi_resposta: Guarda l'última resposta que li ha donat el codemaker respecte a l'últim intent del codebreaker de trobar una solució.
- acabada: Si la partida ja està acabada.
- guanyada: Si la partida ja ha estat guanyada.
- nom: Nom que li donem a la partida, per després mostrar-lo al rànquing.

Descripció de les relacions:

- Relació amb la classe "Codi": ens indica quin és el codi d'una partida en qüestió.

Descripció dels mètodes públics:

- Partida(Integer id, Integer torns, Dificultat dificultat, Codebreaker codebreaker, Codemaker codemaker): Creadora de partida, el codi de la partida l'obtenim de la classe codemaker que passem com a paràmetre.
- List<Colors> getGuess(): Retorna una solució al codi del codemaker la qual ens retorna la classe codebreaker.
- Boolean checkCorreccio(Integer negres, Integer blanques, List<Colors> guess): Donades el nombre de posicions que corresponen amb color i posició (blanques), i el nombre de posicions que només corresponen amb color (negres), comprova si blanques i negres són els números correctes pel codi donat per codebreaker guess.
- List<Colors> comprovarCodi(List<Colors> guess): Donada una solució per codebreaker (guess), la funció retorna una llista que és la correcció d'aquest codi.

Descripció dels mètodes privats:

- Boolean checkAcabada(List<Colors> solucio): Donada una solució a un codi, comprova si amb aquest codi ja s'ha trobat la solució, si és així retorna true, en cas contrari retorna false.
- void jugarTorn(List<Colors> solucio): Augmenta en 1 el nombre de torns actuals, i crida a checkAcabada().

Classe implementada per: Gerard Oliva

Classe Codi

Nom de la classe: Codi.
Breu descripció: Representa un codi d'una partida del MasterMind.
Cardinalitat: Un sol codi per cada partida.
Descripció dels atributs:

- availColors: Llista dels colors que estan disponibles, varia depenent de la dificultat.
- posicions: Nombre de posicions que té un codi, varia depenent de la dificultat.
- codi: Codi que s'utilitzarà com a codi de partida.

Descripció de les relacions:

- Relació amb la classe "Partida": Cada classe té un Codi, que el codemaker decideix i el codebreaker intenta desxifrar.

Descripció dels mètodes públics:

- Codi(Dificultat dificultat): Creadora de codi, segons la dificultat que escollim, el nombre de posicions i els colors disponibles variaran.
- List<Colors> checkCodi(List<Colors> test): Donat un codi (test), la funció compara aquest codi amb el codi que té com a atribut. Retorna una llista amb tantes posicions blanques com test hagi encertat en posició i color, i tantes posicions negres com test només hagi encertat en color.
- List<Colors> checkCodiSeq(List<Colors> test): Mateix funcionament que checkCodi, però no comprova que la mida de test hagi de ser la mateixa que el nombre de posicions que té el codi.

Classe implementada per: Gerard Oliva

Classe Codebreaker

Nom de la classe: Codebreaker.
Breu descripció: Representa al jugador que intenta encertar el codi proposat pel *codemaker* abans que acabin els torns.
Cardinalitat: Un sol jugador *codebreaker* per a cada partida.
Descripció dels atributs:

- estrategia: Instancia de la classe "Estratègia" que utilitzarà el *codebreaker*.
- jugades: Llista amb tots els codis proposats per la estratègia utilitzada.

Descripció de les relacions:

- Relació amb la classe "Partida": correspon al jugador amb rol *codebreaker* a la partida.
- Relació amb la interfície "Estrategia": correspon a la estrategia utilitzada a la partida.

Descripció dels mètodes públics:

- Codebreaker(boolean fg, Codi c). Creadora per defecte de la classe Codebreaker. Si el paràmetre *fg* és cert, el jugador utilitzarà la estratègia *fiveGuess*. Altrament, estrategia *Genetic*.

- jugarTorn(List<Colors> cm): List<Colors>. Retorna una llista de colors amb el codi proposat per el torn en qüestió. El paràmetre *cm* conté la resposta al codi jugat al torn anterior.

  Aquest mètode crida a un altra mètode de la classe "Estratègia" i guarda el codi proposat al seu atribut privat "jugades".

Classe implementada per: Aina Gomez.

Interfície Estrategia

Nom de la interfície: Estrategia.
Breu descripció: Representa les diferents estratègies per resoldre el joc 'Mastermind'. Aquestes estratègies són *FiveGuess* i *Genetic*. Per aquesta entrega hem decidit implementar la primera.
Cardinalitat: Una estrategia per cada jugador *codebreaker* quan l'usuari presenta el rol *codemaker*.
Descripció de les relacions:

- Relació amb la classe "Codebreaker": correspon al jugador que utilitza aquesta estrategia.

Descripció dels mètodes públics:

- jugarTornMaquina(List<Colors> cm, List<Colors> cb): List<Colors>. Mètode on l'estrategia rep com a parametres el codi utilitzat al torn anterior (*cb*) i la resposta d'aquest (*cm*).
  Cada estrategia presenta el seu mètode per resoldre el torn.

Classe implementada per: Aina Gomez.

Classe FiveGuess.

Nom de la classe: FiveGuess.
Breu descripció: Representa una de les estratègies que el jugador *codebreaker* coneix.
Cardinalitat: Una per jugador *codebreaker* (si aquest és la Màquina i no l'Usuari).
Descripció dels atributs:

- PossibleCandidates. Llista d'arbres N aris. Cada element d'aquesta llista representa un arbre per cada color disponible a la partida. És a dir, si tenim 6 colors, aquesta llista tindrà 6 arbres on l'arrel d'aquests serà un color.
  Aquesta llista representa les combinacions que tenen possibilitats a ser el codi pensat pel *codemaker*.
- AllCombinations. Llista d'arbres N aris. Igual que amb "PossibleCandidates", conté un arbre per color.
  Aquesta llista representa totes les combinacions existents amb el nombre de posicions i colors.
- NPOSITIONS. Enter amb el nombre de posicions disponibles a la partida.
- NCOLORS. Enter amb el nombre de colors disponibles a la partida.
- availColors. Llista amb tots els colors disponibles a la partida.

Descripció de les relacions:

- Relació amb la classe "Codebreaker": correspon a la estrategia utilitzada.
- Relació amb la classe "ArbreNAri": correspon a la estructura de dades de la Estrategia.

Descripció dels mètodes públics:

- FiveGuess(Codi c). Creadora per defecte de la estratègia *FiveGuess*. Inicialitza tots els atributs explicats anteriorment.
- getNColors(): int. Retorna el nombre de colors.
- getNPos(): int. Retorna el nombre de posicions.

Descripció dels mètodes privats:

- getInitialGuess(): List<Colors>. Retorna el codi del primer torn, format per dues parelles. Si *a* i *b* son els dos primers colors de l'atribut *availColors*, el resultat serà la llista {a,a,b,b}.
- calcularMinimax(): List<Colors>. Per cada arbre de la llista *AllCombinations*, calcula la combinació amb més probabilitat d'eliminar més codis dins de *PossibleCandidates*. De tots aquests resultats, ens quedarem amb la combinació de cada arbre amb millors resultats.
- reiniciar(c: Codi). Reinicia els atributs de FiveGuess per poder utilitzar-lo per altres partides.

Classe implementada per: Aina Gomez.

Classe ArbreNAri.

Nom de la classe: ArbreNAri.
Breu descripció: Classe pensada per guardar les combinacions possibles en forma d'arbre N ari i realitzar els càlculs necessàris per obtenir el següent codi per l'estratègia "FiveGuess".
Cardinalitat: Per cada estrategia "FiveGuess" tindrem dos llistats d'arbres N aris on cada arbre representa totes les combinacions possibles començant per un color determinat. Per tant, si tenim *c* colors, tindrem 2*c arbres N aris.
Descripció dels atributs:

- arrel. Conté el primer node on comença l'arbre.
- minimax. Conté el node amb millor puntuació pel següent torn de la partida.
- nivells: Conté el nombre de posicions de la partida, el que també és l'alçada de l'arbre.
- indexPoints. Una Hashtable que tradueix les respostes obtingudes al comparar dos codis a un enter dins de l'interval [0,13].
- points. Un array que utilitzem per calcular la millor combinació a jugar.

Descripció de les relacions:

- Relació amb la classe "FiveGuess". Podriem definir-la com la eina que utilitza "FiveGuess" per calcular els seus proxims moviments.
- Relació amb la classe "NodeArbreNAri". Conté tota la informació de cada Node de l'arbre.

Descripció dels mètodes públics:

- ArbreNAri(). Crea un arbre N ari buit.
- ArbreNAri(Colors c, int h). Crea un arbre N ari amb el primer node de color *c*, amb *h* com nivells i declara la taula indexPoints.
- getArrel(): Colors. Retorna el color que conté el primer node.
- getNode(): NodeArbreNAri. Retorna el primer node.
- construirArbre(List<Colors> colors). Declara l'arbre amb alçada *nivells* i per cada node, *colors.size()* fills.
- eliminaColor(Colors c). Elimina de tot l'arbre el color *c*.
- eliminaComb(List<Colors> code, List<Colors> solution). Elimina de l'arbre aquelles combinacions que, al comparar-les amb la combinació *solution*, donen una combinació de blancs i negres diferent a *code*.
- removeCombination(List<Colors> c). Elimina de l'arbre la combinació *c*.
- calculateMinimax(List<ArbreNAri> r): NodeArbreNAri. Calcula la combinació de l'arbre del paràmetre implicit que té més probabilitat d'eliminar més combinacions del llistat *r*.

Descripció dels mètodes privats:

- setHashtable(). Inicialitza la taula *indexPoints*.
- construirArbreRec(NodeArbreNAri n, List<Colors> c, List<Colors> comb, int h). És el mètode recursiu per la funció *construirArbre*. Per cada node que es trobi a l'alçada *nivells-1*, el inicialitzarem com a fulla i hi guardarem la combinació *comb*.
- eliminaColorRec(NodeArbreNAri n, Colors c). Mètode privat de la funcionalitat *eliminarColor*.
- compleixCond(List<Colors> code, List<Colors> solution, List<Colors> seq): boolean. Comprova si al comparar les combinacions *seq* i *solution* donen el mateix nombre de blancs i negres que hi ha a *code*. Si és així, retorna cert, fals altrament.
- eliminaCombRec(List<Colors> code, List<Colors> solution, List<Colors> seq, NodeArbreNAri n). Mètode privat de la funcionalitat *eliminaComb.*
- removeCombinationRec(NodeArbreNAri n, List<Colors> c, int i). Mètode privat per la funcionalitat *removeCombination.*
- recorrePrimerArbre(List<ArbreNAri> r, NodeArbreNAri n). Mètode recursiu que recòrre el arbre del parametre implicit i, un cop arriba a una fulla, recòrre tots els arbres que es troben a *r*.
- recorreSegonArbre(NodeArbreNAri r, List<Colors> comb). Mètode recursiu que recorre els fills de *r* fins arribar a una fulla. Un cop arribem, comparem la combinació del node *r* amb *comb,* segons la resposta obtinguda, actualitzarem l'array *points*.

Classe implementada per: Aina Gomez.

Classe NodeArbreNAri.

Nom de la classe: NodeArbreNAri.
Breu descripció: Conté tota la informació que guarda un Node de la classe ArbreNAri.
Cardinalitat: Si tenim que *c* són el nombre de colors i *p* les posicions. Per cada arbre N ari hi haurà com a màxim: $1 + c^1+c^2+ ... + c^{(p-1)}$.
Descripció dels atributs:

- info. Conté el color del Node.
- childs. Conté una llista amb els fills del Node.

- fulla. Booleà que indica si el Node és fulla (és a dir, esta al nivell p-1 de l'arbre).
- comb. En cas de ser fulla, hi guardarem el color dels nodes pares d'aquest.
- points. Hi guardem els punts maxims de la combinació *comb* en cas de calcular el *minimax.*
- s. Booleà que ens indica que la combinació *comb* del Node es troba dintre de les possibles solucions.

Descripció de les relacions:

- Relació amb la classe "ArbreNAri". La classe NodeArbreNAri conté la informació de tots els nodes de la classe ArbreNAri.

Descripció dels mètodes públics:

- NodeArbreNAri(). Creadora buida de la classe NodeArbreNAri.
- NodeArbreNAri(Colors c). Creadora de la classe NodeArbreNAri on declara el node amb color igual a *c*.
- getInfo(): Colors. Retorna el color que conté el node.
- getChild(int i): NodeArbreNAri. Retorna el fill amb index *i* del node del parametre implicit.
- setChild(NodeArbreNAri info). Declara el node *info* com a fill del parametre implicit.
- setComb(List<Colors> c). Declara l'atribut *comb* com a la llista *c*.
- getComb(): List<Colors>. Retorna la combinació que guardem a *comb*.
- setLeaf(). Marca al node del parametre implicit com a fulla.
- esFulla(): boolean. El resultat és cert si el parametre implicit és fulla, fals altrament.
- getNumChilds(): int. Retorna el nombre de fills que conté el node del p.i.
- setPuntuacio(int p). Actualitza l'atribut points.
- getPoints(). Retorna la puntuació màxima per al node del p.i.
- setS(boolean s). Actualitza l'atribut s de la classe.
- getS(): boolean. Retorna l'atribut s.

Classe implementada per: Aina Gomez.

Interfície Codemaker

Nom de la classe: Codemaker.
Breu descripció: Representa el jugador que proposa el codi a endivinar de la partida.
Cardinalitat: Una classe per partida i diverses partides poden tenir un Codemaker.
Descripció de les relacions:

- Relació d'agregació amb classe "Partida": Control de les classes Usuari i Machine.

Descripció dels mètodes públics:

- List<colors> getCode(): Funció que retorna el codi de colors de l'atribut codi com a llista..
- Boolean rol(): Retorna un boolean segons si es Machine (false) o Usuari (true) el codemaker.

Classe implementada per: Pere Mir.

Classe Usuari

Nom de la classe: Usuari.

Breu descripció: Representa l'usuari quan juga com a codemaker.

Cardinalitat: .

Descripció dels atributs:

- userCode: descripció.

Descripció de les relacions:

- Relació d'agregació amb classe "Partida": Control del codi de l'Usuari.

Descripció dels mètodes públics:

- Usuari(Codi codi): Constructora per defecte la qual es passa per paràmetre un codi per saber el llistat de colors disponibles i la dificultat.
- List<colors> getCode(): Funció que retorna el codi de colors de l'atribut codi com a llista..
- Boolean rol(): Retorna un boolean true perquè està jugant l'usuari com a codemaker.

Classe implementada per: Pere Mir.

Classe Machine

Nom de la classe: Machine.

Breu descripció: La classe Machine representa la maquina quan juga com a codemaker i genera un codi aleatori.

Cardinalitat: .

Descripció dels atributs:

- random: descripció.
- userCode: descripció.

Descripció de les relacions:

- Relació d'agregació amb classe "Partida": Genera el codi aleatori per jugar la partida.

Descripció dels mètodes públics:

- Machine(Codi codi): Constructora per defecte la qual es passa per paràmetre un codi per saber el llistat de colors disponibles i la dificultat.
- void GenerarCodi(): Funció que genera de manera aleatoria un codi i substitueix l'atribut codi pel nou codi generat.
- List<Colors> GetCode(): Funció que retorna el codi de colors de l'atribut codi com a llista.
- Boolean rol(): Retorna un boolean false perquè està jugant la màquina com a codemaker.

Descripció dels mètodes privats:

- void GenerarCodi(): Genera un codi pseudoaleatori de la longitud i amb els colors disponibles de l'atribut userCode.

Classe implementada per: Pere Mir.

Classe Estadístiques.

Nom de la classe: Estadístiques
Breu descripció: Classe que enregistra aquelles partides guanyades per l'usuari com a codebreaker, ordenant el conjunt segons la dificultat i els torns utilitzats en guanyar cada partida.
Cardinalitat: Una sola classe Estadístiques per a totes les partides i moltes partides hi pertanyen.
Descripció dels atributs:

- ranking: Conjunt de partides guanyades

Descripció de les relacions:

- Relació amb la classe "Partida": indica quines partides acabades estan al ranking.

Descripció dels mètodes públics:

- Estadistiques(): constructora per defecte que inicialitza un ranking buit
- void afegirPartida(Partida p): afegeix la partida p al ranking a la seva posició corresponent segons l'ordre definit.
- void reiniciarRanking(): S'esborren totes les partides del ranking i aquest torna a estar buit

Classe implementada per: Pol Contreras

Classe Controlador de Domini.

Nom de la classe: CtrDomini
Breu descripció: La classe controlador de partida, s'encarrega de la partida que està en curs, així com de gestionar gestor partides i estadístiques.
Cardinalitat: Una sola classe "CtrDomini" per les diferents partides i per cada gestor partida i estadístiques.
Descripció dels atributs:

- codemaker i codebreaker: Són booleans que valen "true" si l'humà és qui controla el codemaker/codebreaker, i "false" en cas contrari.
- partida: Representa la partida que hi ha carregada en el moment d'ús.
- dificultat: La dificultat de la partida que hi ha carregada en el moment d'ús.
- torns: Nombre màxim de torns que té el codebreaker per endivinar el codi, de la partida carregada en el moment d'ús.
- maker i breaker: Representen els codebreakers i codemakers que s'inicialitzen i es passen a la creadora de partida.
- codi: Codi de la partida carregada en el moment d'ús.
- gestor: Gestor de partides, que conté les diferents partides guardades.
- estadístiques: Guarda de forma ordenada les partides, per tal de portar un control dels rànquings.
- estretegia: Serveix per saber quina estratègia es vol utilitzar.

Descripció de les relacions:

- Relació amb la classe "Partida": Indica quina és la partida que està en curs en el moment d'ús.
- Relació amb la classe "Codi": Indica el codi de la partida que està en curs en el moment d'ús.
- Relació amb la classe "Codemaker": Inicia el codemaker que la creadora de partida necessita.
- Relació amb la classe "Codebreaker": Inicia el codebreaker que la creadora de partida necessita.
- Relació amb la classe "GestorPartides": Indica on hi ha totes les diferents partides que estan guardades.
- Relació amb la classe "Estadistiques": Indica les partides que s'han guanyat com a codebreaker.

Descripció dels mètodes públics:

- CtrDomini(): Constructora de controlador domini, crida a inicialitzar().
- inicialitzar(): inicia els atributs amb valors per defecte.
- void setPartida(Integer id, List<Colors> codi): inicialitza l'atribut partida amb l'id i el codi que li passem per paràmetre. Per poder cridar aquesta funció ja s'han d'haver assignat els rols, ja que també inicialitza els paràmetres maker i breaker.
- void carregarPartida(Partida partida): Donada una partida que la passem per paràmetre, es configuren els diferents atributs per carregar la partida que es desitja.
- void eliminarPartida(): S'elimina de gestor de partides la partida que està actualment carregada.
- void guardarPartida(): Es guarda la partida actualment carregada al gestor de partides.
- void afegirPartidaARanking(String nom): Li donem un nom a la partida actual, i la posem a estadístiques.

Classe implementada per: Gerard Oliva

Les classes ControladorPresentacio i vistaTerminal han estat implementades per: Gerard Oliva.

3. Estructura de dades i Algorismes.

3.1. Implementació Five Guess Algorithm.

L'algorisme Five Guess és una de les millors estratègies implementades quan es tracta de codis de 4 dígits i amb 6 colors possibles (permeten repeticions). L'algorisme en qüestió ve definit de la següent manera:

1. Creem un *Set S* amb 1.296 combinacions possibles.
2. Comencem amb el codi {1,1,2,2}.
3. Si la resposta donada pel *codemaker* son 4 fitxes blanques, hem acabat.
4. Sinó, borrem de *S* tots aquells codis que no donin la mateixa resposta en comparar-lo amb la combinació jugada.
5. La següent combinació a jugar, l'escollim amb la tècnica *minimax*. De totes les possibles combinacions, calculem quines d'elles tenen més probabilitat d'eliminar més codis de *S*.
6. Repetim des del pas 3.

De manera, que haurem de tenir dos conjunts, un on guardem les possibles combinacions (*S*) i un altre per guardar totes les combinacions possibles.

A cada torn haurem de recórrer contínuament aquests dos conjunts: (1) haurem de recórrer *S* per descartar les noves combinacions i (2) per cada combinació de les 1.296, haurem de recórrer totes les combinacions de *S* per calcular la combinació amb més possibilitats de guanyar.

Sabem que a cada torn que passi, *S'anirà* fent-se més petit, però no és el cas del segon conjunt. En implementar-ho amb un *Set* o amb una Llista, no tenim més remei que recorre a cada torn els dos conjunts. Per aquest motiu, vam decidir plantejar l'algorisme amb una altra estructura de dades.

3.1.1. Estructura de Dades.

Amb l'objectiu de reduir el cost de l'algorisme, vam decidir tenir dues llistes que contenen arbres N aris, on N sería el número de colors disponibles (6, en el nostre cas).

```
Llista<ArbreNAri> AllCombinations;
Llista<ArbreNAri> PossibleCandidates;
```

Sent la llista 'AllCombinations' totes les combinacions possibles i 'PossibleCandidates' les possibles combinacions guanyadores (*S*).

Hem decidit fer una llista d'arbres, ja que d'aquesta manera podem tenir un arbre per cada color i emmagatzemar en aquest totes les combinacions existents començant per aquest color.

És a dir, si tenim que *availColors* (atribut on hi guardem els colors disponibles) és {BLAU, VERD, GROC, TARONJA}, tindrem, per cada llista, quatre arbres amb totes les combinacions que comencen per el color escollit. Sent {BLAU} el color amb index 0, l'arbre que correspon a totes les seves combinacions que comencen per blau, tindrà el index 0.

Per l'algorisme Five Guess utilitzarem 6 colors, per tant, les dues llistes tindran mida 6. Més endavant veurem que si algun arbre de colors es queda sense combinacions, l'eliminarem del llistat.

Per tant, partint de l'exemple anterior, la estructura del primer arbre seria la següent (4 colors i 4 posicions):

```
{BLAU{
        BLAU{
                BLAU{BLAU {},VERD {},GROC{},TARONJA{}},
                VERD{BLAU{},,VERD {},GROC{},TARONJA{}},
                GROC{BLAU{},,VERD {},GROC{},TARONJA{}},
                TARONJA{BLAU{},,VERD {},GROC{},TARONJA{}},
        }, VERD{...},
        }, TARONJA{..}
}
```

Figura 3: Exemple estructura ArbreNAri.

3.1.2. Funcionalitat de l'Algorisme: JugarTornMaquina.

La funcionalitat principal de la classe FiveGuess és "JugarTornMaquina". Aquest mètode retorna una llista de colors, que representa la proposta del *codebreaker* i rep com a parametres dues llistes de colors: *cm* i *cb*. La primera correspon a la resposta obtinguda del *codemaker* i la segona a la jugada anterior del *codebreaker*.

Es presenen tres diferents casos:

1. La llista *cb* és buida.

   En aquest cas significa que ens trobem en el primer torn de la partida, de manera que la resposta del *codebreaker* és un codi de dues parelles de colors (com explica l'algorisme).

   És per això que cridem al mètode *getInirialGuess()* i retornem el seu resultat.
2. La llista *cm* és buida.

   En aquest cas, significa que la solució de la partida no conté cap dels colors que es troben a la llista *cb*.

   Un dels avantatges d'utilitzar arbres enlloc de llistes, és que en el cas d'eliminar tot un color *c*, és tan senzill com anar a cada arbre i podar tots aquells subarbres que conten el color *c*.

Abans de cridar al mètode que s'encarrega d'això, eliminem l'arbre de la llista que correspon amb el color a eliminar. De manera que totes les combinacions que comencen per aquest color seran eliminades simplement borrant l'arbre de la llista.

Per la resta d'arbres de la llista, cridarem al metode de la classe ArbreNAri *eliminarColor(Color c)* per podar tots els subarbres que presentin com primer node el color *c*.

3. Cap dels llistats és buit.

   Per tant, ens trobem en que em jugat una combinació *cb* en el torn anterior i em obtingut una resposta amb blans i negres.

   L'algorisme FiveGuess ens diu que hem d'eliminar del llistat *PossibleCandidates* totes aquelles combinacions que al comparar-les amb el torn jugat *cb* donguin un resultat diferent a *cm*.

   Per poder eliminar aquestes combinacions, necessitarem recòrre cadascun dels arbres i anar eliminant aquelles combinacions que no ens interessen.

   Però, hi ha alguns casos que podem tallar abans d'arribar a l'ultim node de la combinació.

   Posem un exemple, sigui *seq* la seqüencia de colors que estem consultant:

   cm = [BLANC]

   cb = [GROC, BLAU, MAGENTA, VERD]

   (1) seq = {GROC}
   (2) comparar(cb, seq) = [BLANC] -> seguim pels seus fills.
   (3) seq = {GROC, BLAU}
   (4) comparar(cb, seq) = [BLANC, BLANC] -> descartem totes les combinacions que comencen per [GROC,BLAU].

   Com veiem en el pas (4), no val la pena seguir consultant les combinacions que comencen per [GROC, BLAU] ja que obtenim un nombre de blancs major que el que volem obtenir *cm*. De manera, que podem eliminar el fill {BLAU} del node {GROC}.

   Però hem de tenir en compte, que aquesta és la única condició a descartar, ja que els negres poden tornar-se blancs. Per exemple:

   cm = [BLANC]

   cb = [GROC, BLAU, MAGENTA, VERD]

   (5) seq = {VERMELL, MAGENTA}
   (6) comparar(cb, seq) = [NEGRE] -> seguim pels seus fills.
   (7) seq = {VERMELL, MAGENTA, MAGENTA}
   (8) comparar(cb, seq) = [BLANC] -> seguim pels seus fills.
   (9) ...

Un cop els casos 2 i 3 finalitzen, passem a utilitzar la tecnica *minimax* per calcular la combinació amb més probabilitats d'eliminar més codis de *PossibleCandidates.* Per fer-ho, cridem al mètode *calcularMinimax()*.

Cal destacar, que pels casos 2 i 3, com estem eliminant subarbres que formen part d'una combinació amb llargaria 4, cal eliminar també tots aquells nodes que es queden sense descendencia i estan a una alçada menor.

3.1.3. Funcionalitat de l'Algorisme: calcularMinimax

El següent pas de l'algorisme FiveGuess és calcular la pròxima jugada. Per fer això, ens indica que utilitzem la tècnica minimax. Aquesta consisteix en calcular el pitjor cas de cada combinació, és a dir, el nombre de codis que quedarien a *PossibleCombinations* al jugar-la. I de tots aquests casos, agafar el menor possible.

Com nosaltres no tenim totes les combinacions en un mateix arbre, sinó que tenim un llistat d'arbres que contenen aquestes combinacions, haurem de recòrre tots els arbres de *AllCombinations* i per cada combinació haurem de recòrrer tots els arbres de *PossibleCandidates*.

Per fer això, utilitzem dues instancies de tipus *NodeArbreNAri* anomenades *nodeAbs* i *nodeRec*. El primer correspon al node amb la millor puntuació dels arbres visitats, i el segon correspon al node amb millor puntuació de l'arbre que estem calculant.

En el codi de la funció *calcularMinmax()*, tenim un bucle que visita tots els arbres de *AllCombinations*. Per cada arbre crida al mètode de la classe "ArbreNAri" *calculateMinmax(PossibleCandidates)* on li passem el llistat amb les combinacions possibles.

En el mètode de la classe ArbreNAri farem el següent procediment:

1. Recòrrem l'arbre de *AllCombinations* amb el mètode recursiu *recorrerPrimerArbre,* on li passem per parametre la llista *PossibleCandidates*.
2. Un cop arribem a una fulla, farem un bucle per tots els arbres que es troben a la llista rebuda per paràmetre.
3. Per cada un d'aquests arbres, cridarem a un segon mètode annomenat *recorrerSegonArbre(n, comb),* on n és el primer node i *comb* la combinació de la fulla del primer arbre, és a dir, *AllCombinations*.
4. Un cop arribem a una fulla del segon arbre, compararem la combinació de la fulla del segon arbre amb la combinació *comb* rebuda com a paràmetre. Actualitzarem l'atribut points incrementant l'index del vector segons la resposta obtinguda.

   Siguin (#nombre blancs, #nombre negres), cada posició del vector serà:

   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), 9:(2,0),10:(2,1),11:(2,2),12:(3,0),13:(4,0).

   És a dir, si comparem dos combinacions i obtenim 2 blancs i 1 negre, incrementarem la posició 10 del vector points.
5. Un cop sortim de blucle per recòrre totes les combinacions del llistat *PossibleCandidates*, actualitzarem el node *minimax* amb la maxima puntuació (max(points)) i amb la combinació de la fulla del arbre *AllCombinations.*
6. Anirem actualitzant el valor de *minimax*, si trobem una altra combinació de *AllCombinations* que presenti una puntuació més alta. I en cas que presentin la mateixa, ens quedarem amb la combinació que estigui inclosa a *PossibleCandidates*.

Aquest procediment el repetirem per cada arbre que tinguem a *AllCombinations*, és a dir, el repetirem tants cops com colors, és a dir 6 vegades.

Cada resultat el guardarem a *nodeRec* i actualitzarem *nodeAbs* si i només si,

1. *nodeAbs* es buit.
2. *nodeAbs* presenta una puntuació més elevada que nodeRec.
3. *nodeAbs* i *nodeRec* contenen la mateixa puntuació, però la combinació de *nodeRec* es present a l'arbre *PossibleCandidates* i la combinació de *nodeAbs* no.

Un cop hem calculat la tecnica *minimax* per cada arbre de *AllCombinations*, retornarem la combinació de *nodeAbs*.