ENTREGA 2

PROP GRUP 12.1

Aina Gomez
Gerard Oliva
Pere Mir
Pol Contreras

Índex

1. Capa Presentació

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

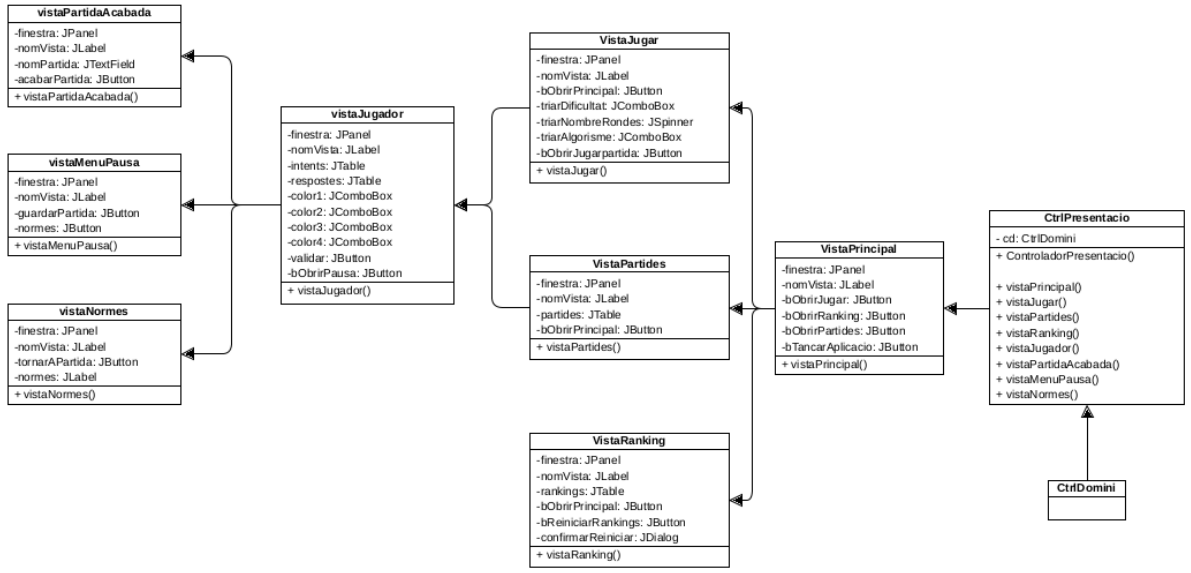

Figura 1: Diagrama de la Capa Presentació

1.2. Descripció

Classe Vista Principal.

Nom de la classe: vistaPrincipal
Breu descripció: Aquesta és la vista principal que s'obre quan executem l'aplicació.
Cardinalitat: Una sola classe vistaPrincipal per cada classe controladorPresentacio.
Descripció dels atributs:

- finestra: JPanel: Panell on s'inclouen tots els elements de la finestra.
- nomVista: JLabel: Títol de l'aplicació.
- bObrirJugar: JButton: Botó per anar a la vista jugar.
- bObrirRanking: JButton: Botó per anar a la vista rànquings.
- bObrirPartides: JButton: Botó per anar a la vista partides.
- bTancarAplicacio: JButton: Botó per sortir de l'aplicació.

Descripció de les relacions:

- Relació amb la classe "ControladorPresentacio": Fa les crides necessàries a gestor partides per a canviar de vista, o interectual amb el controlador de domini.

Descripció dels mètodes públics:

- vistaPrincipal(): Creadora de la classe vistaPrincipal.

Classe implementada per: Gerard Oliva

Classe Vista Jugar.

Nom de la classe: vistaJugar
Breu descripció: Aquesta és la vista on s'introduiran els paràmetres de la partida que es vol començar.

Cardinalitat: Una sola classe vistaJugar per cada classe controladorPresentacio.
Descripció dels atributs:
- finestra: JPanel: Panell on s'inclouen tots els elements de la finestra.
- nomVista: JLabel: Títol de la vista.
- bObrirPrincipal: JButton: Botó per anar a la vista principal.
- triarDificultat: JComboBox: Dropdown menu per escollir la dificultat que es vol jugar.
- triarNombreRondes: JSpinner: Spinner per escollir el nombre de rondes a jugar (nombre parell).
- triarAlgorisme: JComboBox: Dropdown menu per escollir contra quin algorisme vols jugar.
- bObrirJugarpartida: JButton: Botó per obrir la vista jugador.

Descripció de les relacions:
- Relació amb la classe "ControladorPresentacio": Fa les crides necessàries a gestor partides per a canviar de vista, o interactuar amb el controlador de domini.

Descripció dels mètodes públics:
- vistaJugar(): Creadora de la classe de la classe vistaJugar.

Classe implementada per: Gerard Oliva

Classe Vista Rànquings.

Nom de la classe: vistaRanking
Breu descripció: Aquesta és la vista on podrem veure els rànquings de totes les nostres partides.
Descripció dels atributs:
- finestra: JPanel: Panell on s'inclouen tots els elements de la finestra.
- nomVista: JLabel: Títol de la vista.
- rankings: JTable: Taula amb els rànquings de les diferents partides jugades.
- bObrirPrincipal: JButton: Botó per tornar a la vista principal.
- bReiniciarRankigs: JButton: Botó per reiniciar el rànquing.
- confirmarReiniciar: JDialog: Confirma que realment vols reiniciar el rànquing.

Descripció de les relacions:
- Relació amb la classe "ControladorPresentacio": Fa les crides necessàries a gestor partides per a canviar de vista, o interactuar amb el controlador de domini.

Descripció dels mètodes públics:
- vistaRanking(): Creadora de la classe vistaRanking.

Classe implementada per: Pere Mir

Classe Vista partides.

Nom de la classe: vistaPartides
Breu descripció: Aquesta és la vista on podrem veure les diferents partides guardades en el sistema.
Descripció dels atributs:
- finestra: JPanel: Panell on s'inclouen tots els elements de la finestra.
- nomVista: JLabel: Títol de la vista.

- partides: JTable: Taula amb les partides guardades al sistema, amb l'opció de continuar una partida o eliminar-la.
- bObrirPrincipal: JButton: Botó per tornar a la vista principal.

Descripció de les relacions:

- Relació amb la classe "ControladorPresentacio": Fa les crides necessàries a gestor partides per a canviar de vista, o interactuar amb el controlador de domini.

Descripció dels mètodes públics:

- vistaPartides(): Creadora de la classe vistaPartides.

Classe implementada per: Pere Mir

Classe Vista jugador.

Nom de la classe: vistaJugador
Breu descripció: Aquesta és la vista on podrem jugar al MasterMind, depenent del rol que tinguem en el moment tindrem unes funcions o unes altres.
Descripció dels atributs:

- finestra: JPanel: Panell on s'inclouen tots els elements de la finestra.
- nomVista: JLabel: Títol de la vista.
- intents: JTable: Llistat d'intents que ha fet el codebreaker.
- respostes: JTable: Llistat de respostes del codemaker.
- color1: JComboBox: Color de la posició 1.
- color2: JComboBox: Color de la posició 2.
- color3: JComboBox: Color de la posició 3.
- color4: JComboBox: Color de la posició 4.
- validar: JButton: Valida que els colors que s'han introduït estiguin bé, en cas de jugar com a codemaker, i retorna una resposta en cas de jugar com a codebreaker.
- bObrirPausa: JButton: Botó per anar al menú de pausa.

Descripció de les relacions:

- Relació amb la classe "ControladorPresentacio": Fa les crides necessàries a gestor partides per a canviar de vista, o interactuar amb el controlador de domini.

Descripció dels mètodes públics:

- vistaJugador(): Creadora de la classe vistaJugador.

Classe implementada per: Gerard Oliva

Classe Vista partida acabada.

Nom de la classe: VistaPartidaAcabada
Breu descripció: Aquesta és la vista on un cop hàgim acabat la partida podrem posar-li nom perquè aparegui als rànquings.
Descripció dels atributs:

- finestra: JPanel: Panell on s'inclouen tots els elements de la finestra.
- nomVista: JLabel: Títol de la vista.
- nomPartida: JTextField: Nom que li donarem a la partida acabada.
- acabarPartida: JButton: Un cop se li hagi donat un nom a la partida es guardarà a rànquings i desapareixerà de partides guardades.

Descripció de les relacions:

- Relació amb la classe “ControladorPresentacio”: Fa les crides necessàries a gestor partides per a canviar de vista, o interactuar amb el controlador de domini.

Descripció dels mètodes públics:

- vistaPartidaAcabada(): Creadora de la classe vistaPartidaAcabada.

Classe implementada per: Gerard Oliva

Classe Vista menú de pausa.

Nom de la classe: VistaMenuPausa

Breu descripció: Aquesta és la vista on podrem guardar una partida que estigui a mitges així com, accedir a les normes, etc.

Descripció dels atributs:

- finestra: JPanel: Panell on s’inclouen tots els elements de la finestra.
- nomVista: JLabel: Títol de la vista.
- guardarPartida: JButton: Botó per guardar la partida.
- normes: JButton: Boto per anar a la vista normes.

Descripció de les relacions:

- Relació amb la classe “ControladorPresentacio”: Fa les crides necessàries a gestor partides per a canviar de vista, o interactuar amb el controlador de domini.

Descripció dels mètodes públics:

- vistaMenuPausa(): Creadora de la classe vistaMenuPausa.

Classe implementada per: Gerard Oliva

Classe Vista Normes.

Nom de la classe: VistaNormes

Breu descripció: Aquesta és la vista on podrem llegir les normes del joc.

Descripció dels atributs:

- finestra: JPanel: Panell on s’inclouen tots els elements de la finestra.
- nomVista: JLabel: Títol de la vista.
- tornarAPartida: JButton: Botó per continuar la partida
- normes: JLabel: Normes del joc.

Descripció de les relacions:

- Relació amb la classe “ControladorPresentacio”: Fa les crides necessàries a gestor partides per a canviar de vista, o interactuar amb el controlador de domini.

Descripció dels mètodes públics:

- vistaNormes(): Creadora de la classe vistaNormes.

Classe implementada per: Pere Mir

2. Capa Domini

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

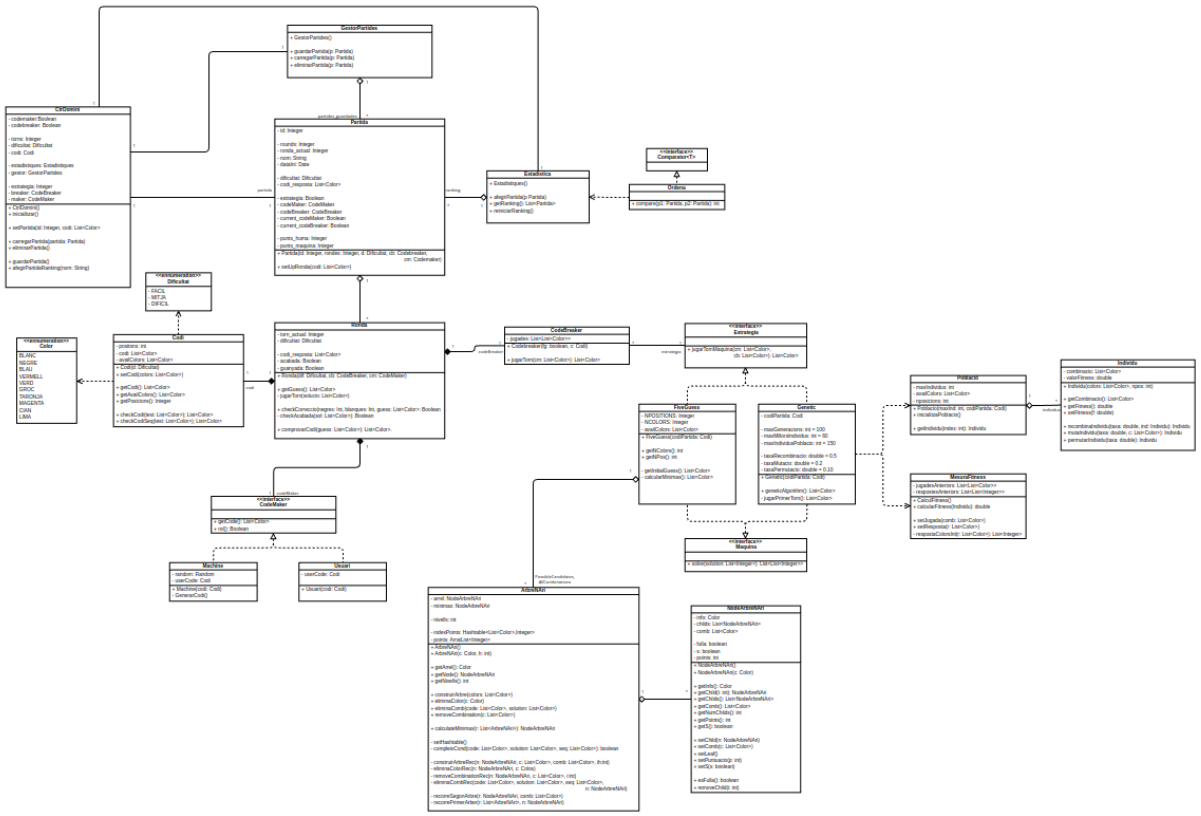

Figura 2: Diagrama de la Capa Domini.

A part d'afegir les classes relacionades amb l'algorisme genètic, també hem afegit la classe Ronda.

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.
- rounds: Nombre parell de rondes a jugar.
- ronda_actual: Ronda actual.
- 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.
- punts_humà: Puntuació actual del jugardor humà.
- punts_màquina: Puntuació actual de la màquina.
- current_codemaker: Indica si el jugador és codemaker.
- current_codebreaker: Indica si el jugador és codebreaker.
- estrategia: estrategia escollida per el jugador màquina.
- 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, hem d'indicar dificultat, nombre de rondes i qui començarà jugant com a codebreaker i codemaker.
- void setupRonda(List<Color> codi): Configura una nova ronda amb el codi que li passa per paràmetre.

Classe implementada per: Gerard Oliva

Classe Ronda

Nom de la classe: Ronda.

Breu descripció: La classe Ronda fa referència amb les altres classes per tal de modelar les funcionalitats d'una ronda al MasterMind.

Cardinalitat: Diverses classes Ronda per cada classe Partida.

Descripció dels atributs:

- 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.

Descripció de les relacions:

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

Descripció dels mètodes públics:

- Ronda(Dificultat dificultat, Codebreaker codebreaker, Codemaker codemaker): Crea una Ronda nova amb la dificultat que li passem per paràmetre, a més d'utilitzar el codebreaker i codemaker que també li passem.
- List<Color> getGuess(): Retorna una solució al codi del codemaker la qual ens retorna la classe codebreaker.
- Boolean checkCorreccio(Integer negres, Integer blanques, List<Color> 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<Color> comprovarCodi(List<Color> 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<Color> 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<Color> 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:

- availColor: 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<Color> checkCodi(List<Color> 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<Color> checkCodiSeq(List<Color> 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 recomanats per l'estratègia usada.

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 l'estratègia feta servir 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 farà servir l'estratègia *fiveGuess*. Altrament, estrategia *Genetic*.
- jugarTorn(List<Color> cm): List<Color>. Retorna una llista de colors amb el codi suggerit pel torn en qüestió. El paràmetre *cm* conté la resposta al codi jugat al torn anterior.

  Aquest mètode crida a un altre 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 estratègia 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 estratègia.

Descripció dels mètodes públics:

- jugarTornMaquina(List<Color> cm, List<Color> cb): List<Color>. Mètode on l'estratègia rep com a paràmetres el codi emprat al torn anterior (*cb*) i la resposta d'aquest (*cm*).
  Cada estratègia 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 l'estratègia utilitzada.
- Relació amb la classe "ArbreNAri": correspon a l'estructura de dades de l'Estrategia.

Descripció dels mètodes públics:

- FiveGuess(Codi c). Creadora per defecte de l'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<Color>. 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<Color>. 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 aconseguides en 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 l'eina que fa servir "FiveGuess" per calcular els seus pròxims 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(Color 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(): Color. Retorna el color que conté el primer node.
- getNode(): NodeArbreNAri. Retorna el primer node.
- construirArbre(List<Color> colors). Declara l'arbre amb alçada *nivells* i per cada node, *colors.size()* fills.
- eliminaColor(Color c). Elimina de tot l'arbre el color *c*.
- eliminaComb(List<Color> code, List<Color> solution). Elimina de l'arbre aquelles combinacions que, al comparar-les amb la combinació *solution*, donen una combinació de blancs i negres diferent de code.
- removeCombination(List<Color> 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<Color> c, List<Color> comb, int h). És el mètode recursiu per la funció *construirArbre*. Per cada node que es trobi a l'alçada *nivells-1*, l'inicialitzarem com a fulla i hi guardarem la combinació *comb*.
- eliminaColorRec(NodeArbreNAri n, Color c). Mètode privat de la funcionalitat *eliminarColor*.
- compleixCond(List<Color> code, List<Color> solution, List<Color> 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<Color> code, List<Color> solution, List<Color> seq, NodeArbreNAri n). Mètode privat de la funcionalitat *eliminaComb.*
- removeCombinationRec(NodeArbreNAri n, List<Color> 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<Color> 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 màxims 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(Color c). Creadora de la classe NodeArbreNAri on declara el node amb color igual a *c*.
- getInfo(): Color. 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<Color> c). Declara l'atribut *comb* com a la llista *c*.
- getComb(): List<Color>. 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.

Classe Genetic.

Nom de la classe: Genetic.
Breu descripció: És la segona estratègia que coneix el jugador *codeBreaker* quan juga com a màquina.
Cardinalitat: Una per *codeBreaker.*
Descripció dels atributs:

- codiPartida: el codi de la partida actual del qual podem obtenir els Color disponibles i el nombre de posicions.
- maxGeneracions: una constant que guarda el nombre màxim de generacions que pot fer l'algorisme en un torn.
- maxMillorsIndividus: constant que guarda el nombre màxim d'Individus que presenten la millor mesura de fitness.
- maxIndividusPoblacio: constant que guarda el nombre màxim d'individus per població.
- taxaRecombinació: de tipus *double*, guarda un index pel càlcul de recombinació d'individus.
- taxaMutacio: de tipus *double*, guarda un index pel càlcul de mutuació d'individus.
- taxaPermutació: de tipus *double*, guarda un index pel càlcul de permutació d'individus.

Descripció de les relacions:

- Relació amb la classe "CodeBreaker": correspon a la estratègia que utilitza el jugador durant la partida (quan el *codeBreaker* no és l'usuari).
- Relació amb la classe "Població": és la classe que genera les diferents poblacions d'individus pel càlcul de l'algorisme.

- Relació amb la classe "MesuraFitness": és la classe que calcula les mesures de fitness de cada individu.

Descripció dels mètodes públics:

- Genetic(Codi c). La creadora de la classe Genetic. Inicialitza l'atribut codiPartida de la classe.

Descripció dels mètodes privats:

- geneticAlgorithm(): List<Color>. Mètode on es resol el procediment general de l'algorisme. Retorna una llista de colors amb el codi amb la millors puntuació de mesura de fitness.
- jugarPrimerTorn(): List<Color>. Mètode que retorna una llista de Color amb una combinació aleatòria.

Classe implementada per: Aina Gomez.

Classe Població.

Nom de la classe: Poblacio.
Breu descripció: És la classe que genera i inicialitza les poblacions necessàries per realitzar l'algorisme genètic.
Cardinalitat: Una o moltes per cada algorisme genètic i conté cap o molts individus.
Descripció dels atributs:

- maxIndividus: enter que conté el nombre màxim d'individus per població.
- individus: llista d'individus.
- availColors: llista dels colors disponibles en la partida actual.
- nposicions: nombre de posicions per combinació a la partida actual.

Descripció de les relacions:

- Relació amb la classe "Genetic": genera totes les poblacions per al càlcul de l'algorisme.
- Relació amb la classe "Individu": engloba un conjunt d'individus per al càlcul de l'algorisme.

Descripció dels mètodes públics:

- Poblacio(int maxInd, Codi codiPartida). Creadora de la classe Població. Inicialitza l'atribut *maxIndividus* amb el valor de la variable *maxInd* que passem per parametre. També inicialitza les variables *availColors* i *nposicions*.
- inicialitzaPoblació(). Inicialitza la població del parametre implicit amb *maxIndividus* individus.
- getIndividu(int i): Individu. Retorna l'Individu amb l'index *i* del conjunt *individus*.

Classe implementada per: Aina Gomez.

Classe Individu.

Nom de la classe: Individu.
Breu descripció: la classe individu conté una possible combinació per la partida actual i un nombre obtingut de la *MesuraFitness* que indica les possiblitats de ser el codi resultant.
Cardinalitat: cap o molts per cada instància de la classe Població.
Descripció dels atributs:

- combinacio: llista de colors de llargaria determinada per cada població.

- valorFitness: valor obtingut de la mesura de fitness. Determina les possiblitats de que la combinació pugui ser el codi creat pel *codeMaker*.

Descripció de les relacions:

- Relació amb la classe "Població": cada individu és un element de cada població. Cada població presenta un nombre determinat d'individus.

Descripció dels mètodes públics:

- Individu(List<Color> colors, int npos). Creadora de la classe Individu. Inicialitza la combinació de llargaria *npos*. Els colors de la combinació els genera aleatòriament.
- getCombinacio(): List<Color>. Retorna l'atribut *combinacio.*
- getFitness(): double. Retorna l'atribut *valorFitness*.
- setFitness(double f). Actualitza l'atribut *valorFitness* pel parametre *f*.
- recombinaIndividu(double taxaRec, Individu ind): Individu. Genera un nou Individu i l'inicialitza sel·leccionant un color de l'individu *ind* o l'individu del parametre implicit segons la taxa de recombinació *taxaRec*.
- mutaIndividu(double taxaMut, List<Color> colors): Individu. Genera un indivudu partint de l'individu del parametre implicit. Segons la taxa de mutació *taxaMut*, modificara certs colors de la combinació per nous sel·leccionats aleatoriament del llistat *colors*.
- permutaIndividu(double taxaPer): Individu. Genera un nou individu modificant l'ordre dels colors del parametre implicit segons la taxa de permutació *taxaPer*.

Classe implementada per: Aina Gomez.

Classe Mesura de Fitness.

Nom de la classe: MesuraFitness

Breu descripció: conté els mètodes per a calcular la mesura de fitness de cada individu.

Cardinalitat: una per cada algorisme genètic.

Descripció dels atributs:

- jugadesAnteriors: llistat que conté totes les combinacions jugades com *codeBreaker* als torns anteriors.
- respostesAnteriors: llistat que conté totes les respostes a les combinacions jugades als torns anteriors. Aquestes respostes les guarda en llistats en forma de (#B, #N), és a dir, (nombre de blancs, nombre de negres).

Descripció de les relacions:

- Relació amb la classe "Genetic": la classe *MesuraFitness* conté alguns dels mètodes per la realització correcta de l'algorisme genetic.

Descripció dels mètodes públics:

- CalculFitness(). Creadora de la classe.
- calcularFitness(Individu ind): Double. Calcula la mesura de fitness de l'individu *ind.*
- setJugada(List<Color> comb). Guarda a l'atribut *jugadesAnteriors* la combinació *comb*.
- setResposta(List<Color> resposta). Guarda a l'atribut *respostesAnteriors* el nombre de blancs i el nombre de negres de la variable *resposta*.

Descripció dels mètodes privats:

- respostaColorsInt(List<Color> res): List<Integer>. Pasa la llista de colors *res* a una llista d'enters que guarda el nombre de colors blancs i negres de forma (#B, #N).

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<Color> 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: Conte i representa la combinació del codi de colors que ha generat l'usuari.

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<Color> 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: Atribut de la classe Random el qual representa un numero aleatori utilitzat per generar una combinació de colors per la Màquina.
- userCode: Conte i representa la combinació del codi de colors que ha generat la Maquina.

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<Color> 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<Color> 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

3. Capa Persistència.

En la imatge podeu veure el diagrama de la Capa de Persistència. Recordeu que al gitlab podeu veure la imatge amb més qualitat.

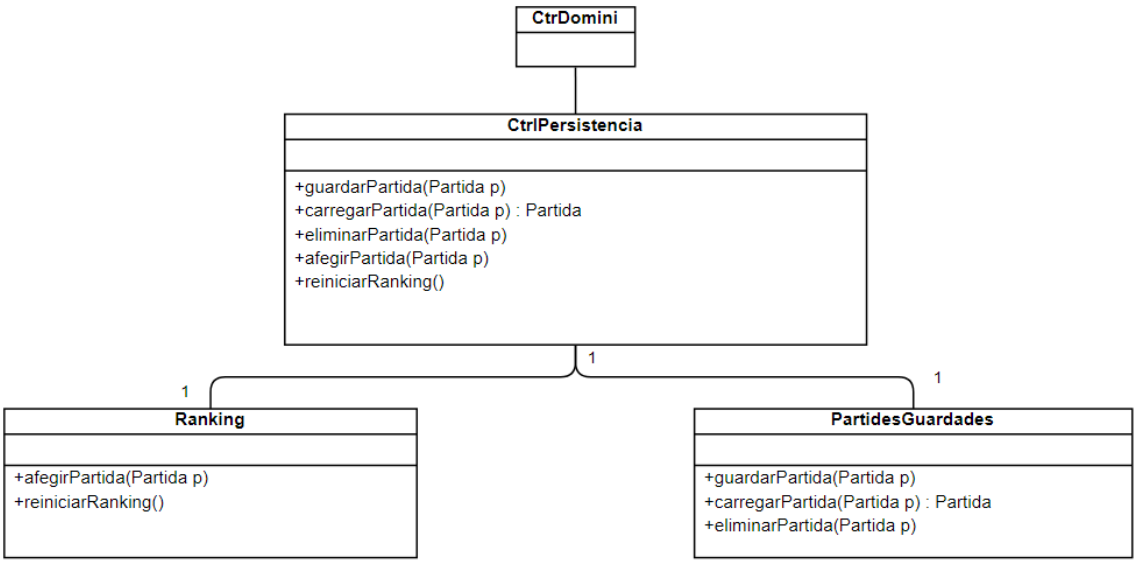

Figura 3: Diagrama de la Capa Persistència.

3.1. Descripció de les classes.

Classe CtrlPersistencia.

Nom de la classe: CtrlPersistencia
Breu descripció: Controlador de persistència que s'encarrega de la gestió de disc amb les partides guardades i el ranking de l'aplicació
Descripció de les relacions:

- Relació amb la classe "CtrlDomini".
- Relació amb la classe "Ranking".
- Relació amb la classe "PartidesGuardades".

Descripció dels mètodes públics:

+ guardarPartida(Partida p). Guarda a disc, amb la resta de partides guardades, la partida p
+ carregarPartida(Partida p):Partida. Busca i retorna, entre les partides guardades a disc, la partida p per a poder continuar.
+ eliminarPartida(Partida p). Busca i elimina la partida p d'entre totes les partides guardades a disc.
+ afegirPartida(Partida p). Afegeix la partida p al ranking de partides guardat a disc.
+ reiniciarRanking(). Buida i elimina totes les partides del ranking guardat a disc.

Classe implementada per: Pol Contreras.

Classe Ranking.

Nom de la classe: Ranking
Breu descripció: Classe que gestiona el ranking de partides, que es guarda a disc per mantenir els rècords passats encara que reiniciem l'aplicació.
Descripció de les relacions:
- Relació amb la classe "CtrlPersistencia". Indica que aquesta és gestionada pel controlador de persistència.

Descripció dels mètodes públics:
+ afegirPartida(Partida p). Afegeix la partida p al ranking de partides guardat a disc.
+ reiniciarRanking(). Buida i elimina totes les partides del ranking guardat a disc.

Classe implementada per: Pol Contreras.

Classe PartidesGuardades.

Nom de la classe: PartidesGuardades
Breu descripció: Classe que gestiona les partides guardades a disc.
Descripció de les relacions:
- Relació amb la classe "CtrlPersistencia". Indica que aquesta és gestionada pel controlador de persistència.

Descripció dels mètodes públics:
+ guardarPartida(Partida p). Guarda a disc, amb la resta de partides guardades, la partida p
+ carregarPartida(Partida p):Partida. Busca i retorna, entre les partides guardades a disc, la partida p per a poder continuar.
+ eliminarPartida(Partida p). Busca i elimina la partida p d'entre totes les partides guardades a disc.

Classe implementada per: Pol Contreras.

4. Estructura de Dades i Algoritmes

4.1. Canvis a FiveGuess

Hem realitzat alguns canvis de nomenclatura a algunes variables com, per exemple, de la classe NodeArbreNAri, l'atribut childs ara es diu children. També hem canviat el nom d'alguns paràmetres amb l'objectiu que el codi sigui més llegible i fàcil d'entendre.

En últim lloc, hem simplificat el codi de solve. Nosaltres inicialitzàvem dues *Hash Tables* amb l'objectiu de passar de colors a enters i a l'invers. Però ens hem adonat que podem fer-ho més senzillament amb la utilització de les funcionalitats *getInfexOf()* i *get()* de l'atribut *availColors*.

4.2. Algorisme Genètic.

El segon algorisme a implementar ha sigut l'algorisme genètic. Aquest algoritme està basat en el procés de selecció natural per obtenir solucions a problemes d'optimització i de cerques.

Per la implementació d'aquest algorisme hem estudiat els tres exemples penjats a la web oficial de l'assignatura. Un cop hem entès el procediment d'aquests, hem pogut realitzar la nostra pròpia implementació.

Per això, hem decidit fer una implementació orientada a objectes de l'algorisme. Podeu veure el diagrama de classes a la següent imatge.

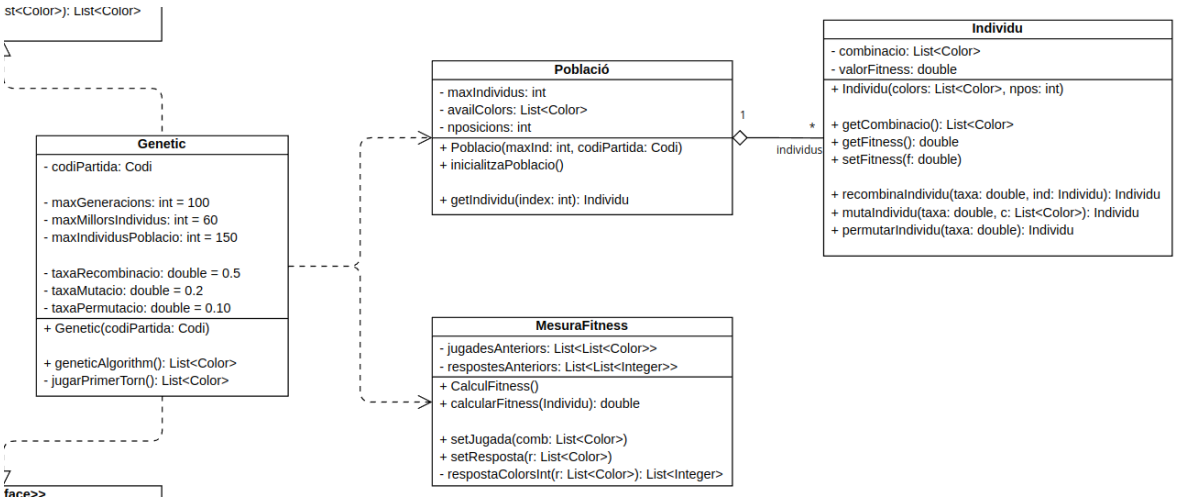

Figura 4: Zoom de les classes de l'algorisme genètic del diagrama de la Capa Domini.

Com veieu a la imatge hem implementat quatre classes: *Genetic, Població, MesuraFitness* i *Individu*.

A la classe Genetic hi trobem les constants generals de l'algorisme com poden ser el nombre de generacions màximes, el nombre d'individus màxim o les taxes de recombinació, mutació i permutació. En aquesta classe també hi trobem el procediment principal de l'algorisme.

La idea d'aquest algorisme és tenir dos conjunts d'individus, és a dir, tenir dues poblacions. És per això que tenim dues constants diferents: *maxIndividusPoblacio* i *maxMillorsIndividus.* Com diuen els seus noms, tindrem la població inicial, que serà la que anirem modificant i després, tindrem una segona població anomenada *MillorsIndividus* amb aquelles combinacions que presenten una mesura de fitness igual a 0.

Tots els mètodes que permeten la modificació i actualització de poblacions estan definits a la classe Població. En aquesta també hi guardem una variable *availColors* on guardem tots els colors disponibles per aquella població. És a dir, si tenim una població P amb el conjunt de colors C, només podrà tenir individus formats pels colors que pertanyen a C.

L'algorisme genètic, a més de generar Individus de forma aleatòria, també demana els mètodes per recombinar dos individus i permutar i mutar un individu. Aquests mètodes es troben a la classe Individu.

Per acabar, tenim la classe *MesuraFitness*. En aquesta classe hi tenim els mètodes necessaris per a calcular la probabilitat de cada individu a ser l'encertat.

Sent *I* un Individu, *comb* la combinació d'*I*, *JA* sigui el conjunt de combinacions jugades i *RA* el conjunt de (#B, #N) de les combinacions jugades. El procediment que realitzem per a calcular el valor de fitness és el següent:

1. fitness = 0.
2. Per tota combinació de *JA (index i):*
    a. comprovem les combinació *JA[i]* amb *comb*.
    b. Obtenim un nombre de blancs (nb) i un nombre de negres (nn).
    c. Fem la resta absoluta del nombre de blancs obtinguts amb el nombre de blancs que hem obtingut de jugar la combinació *JA[i].* És a dir:
    DB = abs(nb - *RA[i].blancs*).
    d. Fem el mateix amb el nombre de negres:
    DN = abs(nn - *RA[i].negres*).
    e. fitness += *DB + DN*.

De manera que els codis amb millor puntuació seran aquells que presentin una mesura de fitness igual a 0. Perquè això voldrà dir que no s'ha trobat cap diferència amb les respostes de les combinacions jugades anteriorment, per tant, és un possible candidat. Com més torns juguem més combinacions tindrem per comparar, en conseqüència, el nombre de candidats serà menor.

Pel que fa al procediment de l'algorisme general, segueix la següent estructura:

1. Inicialitzem h = 0.
2. (si és el primer torn. Si no ens saltem aquest pas) Juguem el primer torn amb la combinació *G1* (combinació aleatòria).
3. Aconseguim el nombre de blancs (#B) i el nombre de negres (#N).
4. Guardem a les variables de *MesuraFitness* la combinació jugada i la resposta.
5. Creem *PoblacioInicial* i *MillorsCombinacions.*
6. Inicialitzem *PoblacioInicial*.

7. Repetim mentre *h* < *maxGeneracions* i *MillorsCombinacions* < *maxMillorsIndividus*.
    a. Generem una nova població *P* utilitzant dos individus de la *PoblacioInicial* i fent recombinació, mutació i permutació.
    b. Calculem el fitness de tots els individus de *P*.
    c. Guardem les combinacions dels individus que presenten un fitness = 0 a la població *MillorsIndividus.*
    d. Incrementem *h*.
    e. Actualitzem la *PoblacioInicial* eliminant totes les combinacions que inclou i afegint totes les que pertanyen a *MillorsIndividus*. L'espai restant a la mida del conjunt i a la mida màxima, hi afegim combinacions generades aleatoriament.
8. Juguem amb una combinació qualsevol que estigui inclosa en la població *MillorsIndividus.*
9. Esperem la resposta de la nova combinació.

A l'algorisme genètic, es generen molt individus i és possible generar dos individus iguals en una mateixa població. Aquest fenomen el controlarem i abans de guardar l'individu a la població comprovarem que no estigui repetit.