

# MASTERMIND

## Descripción Diagrama de Clases Dominio

**Projectes de Programació:** Quadrimestre Primavera 2022-23

**Grup 42.5:**

Elsa Boix Solina

Joel Macías Rojas

Ricard Medina Amado

Marcel Sánchez Roca

# Descripción del diagrama de clases

## 1. Usuario

**Breve descripción de la clase:** Registra la información del usuario en nuestro sistema.

**Descripción de los atributos:**

- username – Nombre que identifica al usuario
- maxScore – Vector que guarda el orden de las puntuaciones de las partidas realizadas por el usuario.
- partidas – Array que guarda las fechas de las partidas jugadas por el usuario.

**Descripción de los métodos:**

- deletePartida(Date fecha): Borra una partida con la fecha pasada por parámetro de las partidas del usuario.

**Descripción de las relaciones:**

- Relación de asociación con la clase “Partida”: Indica quién es el que realiza la partida.
- Relación de asociación con la clase “CtrlUsuario”: Indica que el usuario es gestionado por el CtrlUsuario.

## 2. CtrlUsuario

**Breve descripción de la clase:** Tiene el conjunto de instancias de “Usuario” y es el encargado de comunicarse con el usuario y sus partidas.

**Descripción de los atributos:**

- usuarios – ArrayList con los nombres de los usuarios del sistema.
- userAct – Instancia del usuario que interactúa actualmente con el sistema.

**Descripción de los métodos:**

- addUsuario(): Añade un usuario a la lista de usuarios.
- loginUser(String username): Si el usuario no existe se añade a la lista de usuarios. Se crea una nueva instancia de Usuario usando el username pasado por parámetro y se iguala a userAct.

- crearPartida(int dificultadEscogida, boolean ayuda, boolean rol): Se crea una partida usando los valores pasados por parámetro.
- crearPartidaGuardada(String data, int nTurno, boolean rol, ArrayList<Color> solucion, boolean ayuda, int puntuacion, int dificultadEscogida, ArrayList<ArrayList<Color>> combinaciones, int rondasMaquina): Crea una partida a partir de los datos de una partida guardada.
- borrarPartida(Date data): Borra una partida tanto del historial de partidas como de la lista de partidas del usuario.
- solicitarAyuda(): Activa la ayuda en la partida actual.
- salirPartida(): Sale de la partida actual.
- existsPartidaActual(): Devuelve un valor booleano dependiendo de si existe alguna partida actual o no.
- reiniciarPartida(): Reinicia una partida actual.
- cambiarEstadoPartida(): Cambia el estado de la partida actual al que le llega por parámetro en forma de String.

#### **Descripción de las relaciones:**

- Relación de asociación con la clase "CtrlDominio": El Controlador Domini crea una instancia de Controlador Usuario y se utiliza en las funcionalidades necesarias del sistema.
- Relación de asociación con la clase "Usuario": Gestiona a los usuarios del sistema.
- Relación de asociación con la clase "CtrlPartida": Necesario para que el usuario pueda interactuar con la partida.

### **3. Partida**

**Breve descripción de la clase:** Registra la información de una partida que ha sido creada por la aplicación.

#### **Descripción de los atributos:**

- data – Date que sirve como identificador de la partida.
- puntos – Int que indica la puntuación resultante de la partida.
- ayuda – Bool que si es falso indica que no se quiere ayuda en la partida, true si se quiere ayuda en la partida.

- estadoPartida – Indica el estado actual de la partida, RUNNING, PAUSED, SAVED.
- solutions – ArrayList que contiene la solución de la partida.
- nivel – Indica el nivel de dificultad de la partida: BAJO, MEDIO, ALTO
- username – Indica el nombre del usuario encargado de la partida.
- turnos – Indica el turno actual de la partida.

#### **Descripción de los métodos:**

- convertIntegerToColor(List<Integer> combInteger): Función encargada de cambiar cada uno de los números guardados en la lista combInteger por su correspondiente Color.
- convertColorToInteger(Combinacion comb): Función encargada de cambiar cada uno de los colores guardados en la combinación comb por su correspondiente valor entero.
- donePartida(): Método que se llama para asegurar que una partida ha sido finalizada y añade la partida a partidas guardadas.
- checkIfReps(ArrayList<Color> combinacion): Revisa si en la combinación pasada por parámetro se repite algún Color. En caso de repetirse se lanza una excepción.
- checkLevelExceptions(ArrayList<Color> combinacion): Revisa que el número de columnas sea correspondiente al del nivel.
- checkIfAllCorrects(ArrayList<ColorFeedBack> feedBackSolution): Revisa si en el feedback devuelto todas las bolas están en la posición correcta.
- setSolution(ArrayList<Color> combSolution): Se introduce la solución a adivinar para este turno.
- getAllCombLastTurno(): Retorna todas las combinaciones hechas en el turno anterior.
- setCombinacion(ArrayList<Color> combSolution): Se introduce uno de los 10 posibles intentos para encontrar la combinación solución. En caso de acertar se llama a la función encargada de gestionar el final de partida(donePartida).
- reiniciarPartida(): Se encarga de borrar las combinaciones ya hechas.
- guardarPartida(): Guarda la partida en la persistencia.

#### **Descripción de las relaciones:**

- Relación de asociación con la clase “Usuario”: Indica de quién es la partida.

- Relación de asociación con la clase “CtrlPartida”: Indica que la partida es gestionada por el controlador de partida.
- Relación de asociación con la clase “Ranking” Indica que la partida estará en un ranking.
- Relación de asociación con la clase “NivelDificultad”: Necesaria para tratar la partida de una manera u otra dependiendo del nivel de dificultad.
- Relación de asociación con la clase “Turno”: Necesaria para tratar de una manera u otra dependiendo de si ha de jugar como CodeBraker o CodeMaker.
- Relación de asociación con la clase “Combinación”: Necesaria para tener la funcionalidad de enviar combinaciones i soluciones.

## 4. CtrlPartida

**Nombre de la clase:** CtrlPartida

**Breve descripción de la clase:** Tiene el conjunto de instancias de la clase Partida

**Descripción de los atributos:**

- partidaActual – Instancia actual de la partida que se está tratando en el sistema.

**Descripción de los métodos:**

- crearPartida(int dificultadEscogida, boolean ayuda, boolean rol): Se crea una partida usando los valores pasados por parámetro.
- crearPartidaGuardada(String data, int nTurno, boolean rol, ArrayList<Color> solucion, boolean ayuda, int puntuacion, int dificultadEscogida, ArrayList<ArrayList<Color>> combinaciones, int rondasMaquina): Crea una partida a partir de los datos de una partida guardada.
- borrarPartida(Date data): Borra una partida tanto del historial de partidas como de la lista de partidas del usuario.
- existsPartidaActual(): Devuelve un valor booleano dependiendo de si existe alguna partida actual o no.
- solicitarAyuda(): Activa la ayuda en la partida actual.

- salirPartida(): Sale de la partida poniendo el atributo de partidaAct igual a nulo.
- reiniciarPartida(): Reinicia una partida actual.
- cambiarEstadoPartida(): Cambia el estado de la partida actual al que le llega por parámetro en forma de String.

#### **Descripción de las relaciones:**

- Relación de asociación con la clase "Partida". Gestiona las partidas del sistema
- Relación de asociación con la clase "CtrlUsuario". Necesario para que el usuario pueda interactuar con la partida.

## **5. CtrlDominio**

**Breve descripción de la clase:** Se utiliza para la funcionalidad de todas las clases y comunicarse con la persistencia y la presentación.

#### **Descripción de los métodos:**

- restoreRanking(): Restaura los rankings de los tres niveles de dificultad desde la persistencia.
- restoreRecord(): Restaura el ranking personal del usuario desde la persistencia.
- loginUser(): Hace el login del usuario, creándolo si no existía y si existía carga sus datos desde la persistencia.
- newUser(String username): Añade a un nuevo usuario a la lista de usuarios con el nombre de usuario pasado por parámetro.
- cargarPartida(String id): Carga una partida que había sido previamente guardada en la persistencia de un usuario. La partida está identificada con el String id pasado por parámetro.
- newCombinacion(ArrayList<Color> combination): Añade la nueva combinación de colores pasada por parámetro.
- salirPartida(): Sale de la partida poniendo el atributo de partidaAct igual a nulo.
- reiniciarPartida(): Reinicia una partida actual.

- addPartidaRanking(String jugador, Integer puntuacion, Integer nivelDificultad): Añade una partida al ranking del nivel de dificultad pasado por parámetro con el nombre de usuario y fecha indicados en caso de que el usuario no existiera en el ranking o que la puntuación que tuviera en el ranking fuera inferior.
- guardarPartida(): Guarda la partida actual en la persistencia del jugador.

#### **Descripción de las relaciones:**

- Relación de asociación con “CtrlUsuario”: el controlador de dominio crea una instancia de CtrlUsuario y accede a ella cuando quiere empezar con las funcionalidades del sistema.
- Relació d’associació amb “Ranking”: el controlador de dominio crea y gestiona el ranking del sistema.

## **6. Turno**

**Breve descripción de la clase:** Se utiliza para dar unas u otras funcionalidades dependiendo de si se es CodeMaker o CodeBreaker.

#### **Descripción de los atributos:**

- rol: bool que false=CodeBreaker, true=CodeMaker
- combinations: ArrayList con las combinaciones que se han enviado en la partida.

#### **Descripción de los métodos:**

- changeTurno(): Modifica el atributo rol poniendo el inverso del valor guardado actualmente.
- eraseCombinations: Borra todas las combinaciones que hay guardadas.

#### **Descripción de las relaciones:**

- Relación de asociación con “Partida”: Necesaria para tener todas las funcionalidades relacionadas con enviar combinaciones.
- Relación de asociación con “Combinacion”: Necesaria para poder crear combinaciones.

## 7. Combinacion

**Breve descripción de la clase:** Se utiliza para crear las combinaciones que servirán para la funcionalidad del juego

**Descripción de los atributos:**

- combination: ArrayList de colores que crean la combinación.

**Descripción de los métodos:**

- contains(Color color): Devuelve un valor booleano dependiendo de si la combinación tiene el color pasado por parámetro.
- add(Color colorAñadido): Añade un color pasado por parámetro a la combinación.
- remove(Color colorEliminado): Quita el color pasado por parámetro de la combinación. En caso de no tener ese color en la combinación devuelve una excepción.

**Descripción de las relaciones:**

- Relación de asociación con "Turno": Necesaria para que Turno pueda enviar combinaciones a Partida.
- Relación de asociación con "Partida": Necesaria para que Partida tenga una solución.

## 8. Ranking

**Breve descripción de la clase:** Se utiliza para tener un registro de las mejores puntuaciones de las partidas del sistema.

**Descripción de los atributos:**

- posiciones: map que se encarga de ordenar las puntuaciones de cada partida y las asocia al usuario que ha jugado la partida.

**Descripción de los métodos:**

- addPartida(String jugador, Integer puntuacion, Integer nivelDificultad): Dependiendo del nivelDificultad recibido añade al ranking bajo, medio o alto la puntuación y el jugador que llegan por parámetro en caso de tener una mayor puntuación de la que tenía con anterioridad o no estar registrado en el ranking.



- cargarRanking(int dificultad, TreeMap<String, Integer> ranking):  
Dependiendo del valor de dificultad carga uno de los tres rankings en memoria.

#### **Descripción de las relaciones:**

- Relación de asociación con CtrlDomini; Necesaria para la gestión del mismo.
- Relación de asociación con partida: Necesaria para poder obtener tanto la puntuación de la partida como el usuario responsable de ella.

## **9. NivelDificultad**

**Breve descripción de la clase:** Contiene los métodos comunes para los diferentes tipos de dificultad, además de las funciones encargadas de las comprobaciones entre envíos y soluciones.

#### **Descripción de los atributos:**

- numColumnas – Número de columnas que tendrá el juego.
- numColores – Número de colores que se podrá elegir para jugar.
- sePuedeRepetir – Bool que si true == se pueden repetir colores tanto en la solución como en la combinación, de lo contrario == false.
- turn – Indica el turno en que esta la partida.
- solución – solución de la partida

#### **Descripción de los métodos:**

- calculaPuntuacion(int numIntentCodeMaker, int numIntentCodeBreaker):  
Calcula la puntuación correspondiente al nivel de dificultad en el que se esté, teniendo en cuenta el número de intentos hechos en cada uno de los roles.
- comprobarCombinacion(List<Integer> solution, List<Integer> solEnviada):  
Comprueba las diferencias entre la solución realizada por el usuario y la solución real del turno y retorna un feedback resultado de esta comparación.
- comprobarCombinacionPista(List<Integer> solution, List<Integer> solEnviada):  
Comprueba las diferencias entre la solución realizada por el usuario y la solución real del turno y retorna un feedback ordenando el feedback para cada bola, debido a que estamos en el modo ayuda.
- getCombinacion():  
Genera una combinación aleatoria que se usará como solución para que el usuario adivine en su turno de CodeBreaker.

**Descripción de las relaciones:**

- Relación de asociación con “Partida”: Necesaria para que la Partida pueda disponer de una lógica para poder jugar e interactuar con la máquina.
- Relación de asociación con “Maquina”: Necesaria para poder acceder al algoritmo encargado de hallar la solución cuando se juega como *codemaker*.

## 10. NivelDificultadBajo

**Breve descripción de la clase:** Contiene los métodos específicos del nivel de dificultad bajo, dónde no hay repeticiones y el número total de columnas son 4.

**Descripción de los atributos:**

- numColumnas – Número de columnas que tendrá el juego.
- numColores – Número de colores que se podrá elegir para jugar.
- sePuedeRepetir – Bool que si true == se pueden repetir colores tanto en la solución como en la combinación, de lo contrario == false.
- turn – Indica el turno en que esta la partida.
- solucion – solución de la partida

**Descripción de los métodos:**

- calculaPuntuacion(int numIntentCodeMaker, int numIntentCodeBreaker):  
Calcula la puntuación correspondiente al nivel de dificultad bajo, teniendo en cuenta el número de intentos hechos en cada uno de los roles.

**Descripción de las relaciones:**

Recibe la herencia de la clase padre “NivelDificultad”: Necesaria para evitar repetición de código y darle atributos propios de la clase como número de columnas o nivel de dificultad que se requerirán en 5Guess para así adaptarse a las especificaciones del nivel de dificultad.

## 11. NivelDificultadMedio

**Breve descripción de la clase:** Contiene los métodos específicos del nivel de dificultad medio, dónde sí hay repeticiones y el número total de columnas son 4.

**Descripción de los atributos:**

- numColumnas – Número de columnas que tendrá el juego.

- numColores – Número de colores que se podrá elegir para jugar.
- sePuedeRepetir – Bool que si true == se pueden repetir colores tanto en la solución como en la combinación, de lo contrario == false.
- turn – Indica el turno en que esta la partida.
- solucion – solución de la partida

#### **Descripción de los métodos:**

- calculaPuntuacion(int numIntentCodeMaker, int numIntentCodeBreaker):  
Calcula la puntuación correspondiente al nivel de dificultad medio, teniendo en cuenta el número de intentos hechos en cada uno de los roles.

#### **Descripción de las relaciones:**

- Recibe la herencia de la clase padre “NivelDificultad”: Necesaria para evitar repetición de código y darle atributos propios de la clase como número de columnas o nivel de dificultad que se requerirán en 5Guess para así adaptarse a las especificaciones del nivel de dificultad.

## **12. NivelDificultadAlto**

**Breve descripción de la clase:** Contiene los métodos específicos del nivel de dificultad dificultad alto, dónde sí hay repeticiones y el número total de columnas son 5.

#### **Descripción de los atributos:**

- numColumnas – Número de columnas que tendrá el juego.
- numColores – Número de colores que se podrá elegir para jugar.
- sePuedeRepetir – Bool que si true == se pueden repetir colores tanto en la solución como en la combinación, de lo contrario == false.
- turn – Indica el turno en que esta la partida.
- solucion – solución de la partida

#### **Descripción de los métodos:**

- calculaPuntuacion(int numIntentCodeMaker, int numIntentCodeBreaker):  
Calcula la puntuación correspondiente al nivel de dificultad alto, teniendo en cuenta el número de intentos hechos en cada uno de los roles.

#### **Descripción de las relaciones:**

- Recibe la herencia de la clase padre "NivelDificultad": Necesaria para evitar repetición de código y darle atributos propios de la clase como número de columnas o nivel de dificultad que se requerirán en 5Guess para así adaptarse a las especificaciones del nivel de dificultad.

## 13. FiveGuess

**Breve descripción de la clase:** Contiene los métodos para, mediante el algoritmo de *Five Guess*, generar los envíos cuando el jugador crea una solución como *codemaker*.

**Descripción de los atributos:**

- possibleCodes – Contiene las diferentes combinaciones de números que podrían ser la solución.
- totalcombinacionesPosibles – Contiene todas las diferentes combinaciones que se pueden generar en el nivel de dificultad especificado.
- solucionesEnviadas – Contiene los envíos realizados por el algoritmo para encontrar la solución
- enviosCandidatos – Se genera en cada ronda, contiene las combinaciones más adecuadas para ser enviadas.
- turn - turno actual
- envioActual - Última combinación comprobada,
- solucion - Solución que el algoritmo debe encontrar
- nivel - Nivel de dificultad, 1 = Bajo, 2 = Medio, 3= Alto.

**Descripción de las relaciones:**

- Relación de asociación con Máquina: Necesaria para una vez implementado el otro algoritmo, facilitar las comunicaciones entre

## 14. Juego

**Breve descripción de la clase:** Se encarga de mostrar las reglas del juego y el criterio que se usa en el sistema de puntuación.

**Descripción de los atributos:**

- InformaciónPuntuación: Texto que explica el criterio de puntuación.
- InformaciónSistema: Texto que explica como funciona el juego.

**Descripción de las relaciones:**

- Relación de asociación con CtrlDomini: Necesaria para poder proporcionar las informaciones de las reglas del juego.

## 15. HistorialPartidas

**Breve descripción de la clase:** Se encarga de tener un historial con todas las partidas, independientemente del estado de la partida.

**Descripción de los atributos:**

- partidas: ArrayList que contiene el usuario asociado a la partida y la fecha que identifica la partida.

**Descripción de los métodos:**

- agregarPartida(String username, Data dataIni): Añade una partida al total de partidas con el nombre y fecha pasados por parámetro.
- getPartidas(String username): Devuelve las partidas que tiene el usuario pasado por parámetro.
- (String username, Data dataIni): Borra del total de partidas la partida identificada con el nombre y fecha pasados por parámetro.

**Descripción de las relaciones:**

- Relación de asociación con CtrlDomini: Necesaria para la gestión de la clase.
- Relación de asociación con CtrlPartida: Necesaria para obtener la fecha de la partida.

## 14. HistorialPartidasGuardadas

**Breve descripción de la clase:** Se encarga de tener un historial con todas las partidas guardadas en el sistema en ese momento.

**Descripción de los atributos:**

- partidas: ArrayList que contiene el usuario asociado a la partida y la fecha que identifica la partida.

**Descripción de los métodos:**

- agregarPartidaGuardada(String username, Data dataIni): Añade una partida al total de partidas guardadas con el nombre y fecha pasados por parámetro.
- getPartidas(String username): Devuelve las partidas que tiene guardadas el usuario pasado por parámetro.
- borrarPartidaGuardada(String username, Data dataIni): Borra del total de partidas guardadas la partida identificada con el nombre y fecha pasados por parámetro.

**Descripción de las relaciones:**

- Relación de asociación con CtrlDomini: Necesaria para la gestión de la clase.
- Relación de asociación con CtrlPartida: Necesaria para obtener la fecha de la partida.