Documentación de clases

Tabla de contenidos

ExceptionsHandler	0
InGame	0
ViewBoard	0
ViewResult	0
ViewPause	0
ViewCM	0
ViewCB	0

Diagrama del model conceptual de dades en versió disseny

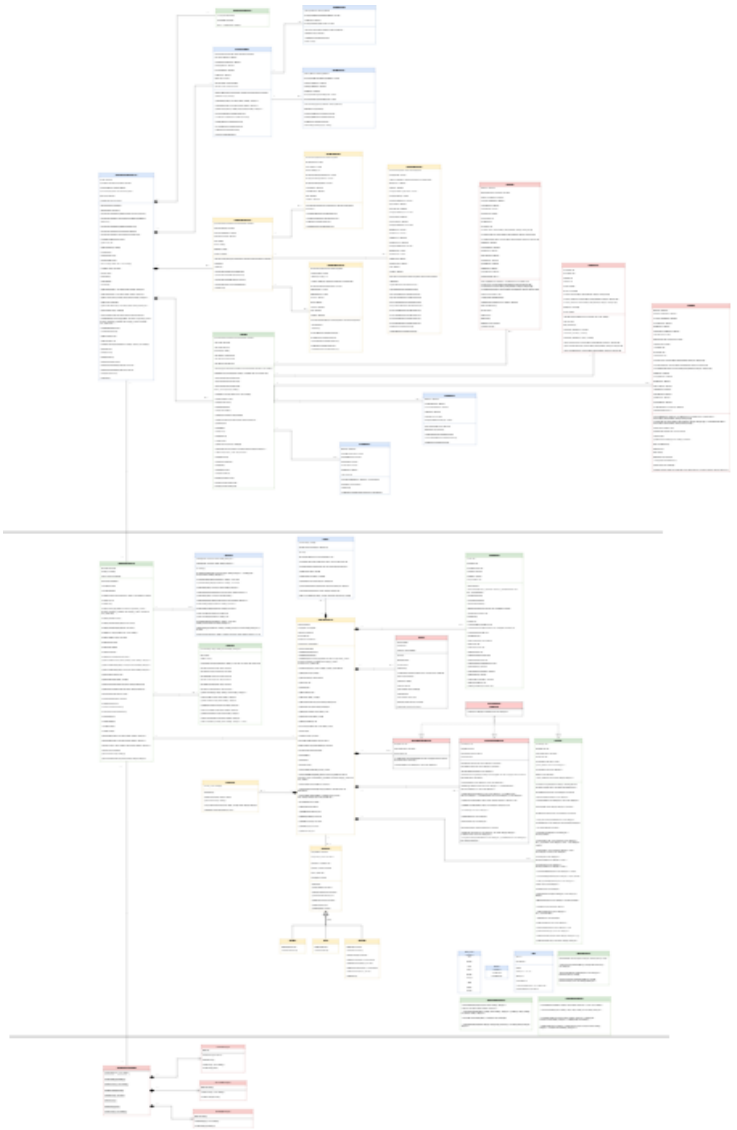

Figura 1. Diagrama UML

Clases principales de dominio

DomainController

Esta clase consiste en ser el controlador de dominio el cual se encargará de toda la comunicación entre las capas de presentación y de persistencia. Este controlador se encarga de llamar al controlador de partida para tener todo lo necesario para jugar una partida. Además, esta clase tiene la responsabilidad de crear y actualizar los rankings y récords cada vez que acaba una partida si se dan los requisitos necesarios.

- Atributos:
    - game: instancia de la clase GameController que controla el juego.
    - ranking: instancia de la clase Ranking que almacena los rankings de los jugadores (no máquina).
    - records: instancia de la clase Records que almacena los pódios escogidos del juego.
- Métodos:
    - DomainController()
        - Creadora de la cosa que inicializa la instancia de GameController y también inicializa los récords y los rankings.
    - createRecords()
        - crea e inicializa los récords
    - createRanking()
        - crea e inicializa los rankings
    - notifyNewPlayer(String playerName, String playerMode)
        - con el nombre del jugador y el modo que escoge, se llama al controlador de juego para que pueda crear la instancia de jugador
    - notifyNormal()
        - notifica al controlador del juego que el usuario quiere jugar una partida en dificultad normal
    - notifyHard()
        - notifica al controlador del juego que el usuario quiere jugar una partida en dificultad difícil
    - notifyCustom(boolean algorithm,boolean repeated_colors, int number_positions, boolean empty_colors, int max_turns)
        - notifica al controlador del juego que el usuario quiere jugar una partida en dificultad custom con los parámetros recibidos
    - notifyUpdateRecords()
        - notifica a Records para actualizar los récords con la información de la partida acabada
    - notifyUpdateRankingCM(boolean win)
        - notifica a Ranking para actualizar los rankings de codemaker si la partida que ha acabado ha sido una victoria
    - notifyUpdateRankingCB(boolean win)

        - notifica a Ranking para actualizar los rankings de codebreaker si la partida que ha acabado ha sido una victoria
    - notifyNewColor(int position, String color)
        - notifica al controlador de la partida con un color una posición para almacenar el nuevo color en la posición correspondiente del código del turno actual.
    - notifyConfirmCode(): boolean
        - notificación al controlador de la partida para confirmar el código seleccionado por el jugador. Devolverá true dependiendo de si ha acertado el código secreto o no.
    - notifyCleanCode()
        - notifica al controlador de la partida que el jugador quiere limpiar todo el código actual
    - notifyAskForHint()
        - notifica al controlador de la partida que el jugador ha pedido una ayuda.
    - notifySurrender()
        - notifica al controlador de la partida que el jugador se ha rendido.
    - notifySetComputerSecretCode()
        - Notifica al controlador de la partida que se cree un código secreto para cuando el jugador es codebreaker.
    - getDomainRecords(): Map<String, Pair<String, Integer>>
        - devuelve el estado actual de los records de la clase records.
    - getDomainRankingCM(): Map<String, Pair<String, Integer>>
        - devuelve el estado actual de los rankings de codemaker de la clase rankings.
    - getDomainRankingCB():Map<String, Pair<String, Integer>>
        - devuelve el estado actual de los rankings de codemaker de la clase rankings.
    - getGameMaxTurns(): int
        - le pregunta al controlador de partida el número máximo de turnos
    - getGameBoard(): Ball_color[][]
        - le pide al controlador de partida por el tablero actual
    - getGameFeedbackBoard(): Pair<Integer,Integer>[]
        - le pide al controlador de partida por el tablero actual de feedback
    - getPlayerCurrentFeedback(): Pair<Integer,Integer>
        - le pide al controlador de partida por el feedback del turno actual
    - getCurrentCode(): Ball_color[]
        - le pide al controlador de partida por el código actual del jugador que está seleccionando
    - savedGameExists(): boolean
        - pregunta a persistencia si existe una partida guardada (solo dejamos tener una)
    - notifyEndedGame()
        - avisa a persistencia de que la partida ha acabado y por lo tanto, debe borrar su partida guardada (si es que la tiene)
    - saveGame2Persistence()

        - envía una lista de strings a persistencia con todos los datos necesarios para poder luego continuar la partida.
    - loadGameFromPersistence()
        - Consigue la lista de strings con los datos necesarios de la partida, los pasa a valores que se entiendan en dominio y crea la partida a partir de esos datos.
    - saveRankings()
        - coge los dos rankings (CM y CB), los junta en una misma estructura que persistencia pueda leer y los envía para que se lo guarde.
    - loadRankings()
        - pide la estructura conjunta de rankings a persistencia y la trata para poder separarla y así guardarla en dominio.
    - saveRecords()
        - coge los récords, los traduce a una estructura que persistencia pueda leer y se lo envía para que lo guarde.
    - loadRecords()
        - pide a persistencia la estructura suya de récords para que la pueda traducir y setear en dominio.
    - giveRankingCM(): List<Map.Entry<String, Integer>>
        - le devuelve a presentación el ranking de codemaker
    - giveRankingCB(): List<Map.Entry<String, Integer>>
        - le devuelve a presentación el ranking de codebreaker
    - giveRecords(): Map<String, Map.Entry<String, Integer>>
        - le devuelve a presentación todos los récords
    - giveBoard(): String[][]
        - le devuelve a presentación el tablero
    - giveFeedbackBoard(): List<Map.Entry<Integer,Integer>>
        - le devuelve a presentación el tablero de feedback
    - giveCurrentCode(): String[]
        - le devuelve un array de String con el código actual que está poniendo el jugador para que presentación pueda pintarlo
    - giveCurrentFeedback(): Map.Entry<Integer,Integer>
        - le devuelve un map entry con el feedback del turno actual para que presentación pueda tratarlo.
    - getGameTurn()
        - pide al controlador de partida que devuelve el turno actual
    - notifyPause()
        - avisa al controlador de partida que pare el timer.
    - notifyResume()
        - avisa al controlador de partida que encienda otra vez el timer

Ranking

Clase responsable de almacenar la información necesaria respecto a la tabla de rankings. Además, se encarga de actualizar los rankings y ordenarlos según vayan entrando nuevas estadísticas.

- Atributos:
    - rankingCB: arraylist que representa el ranking de codebreaker.
    - rankingCM: arraylist que representa el ranking de codemaker.
- Métodos:
    - Ranking()
        - creadora de valores vacíos.
    - Ranking(ArrayList<Pair<String,Integer>> rankingCB, ArrayList<Pair<String,Integer>> rankingCM)
        - Crea un nuevo ranking a partir de rankings especificados como parámetros.
    - updateRankingCB(String playerName, int score)
        - Actualiza la clasificación CodeBreaker con una nueva entrada para el jugador y la puntuación especificada.
          Si la clasificación no está completa (menos de 20 entradas), o si la nueva puntuación es mayor que la puntuación más baja de la clasificación, la nueva entrada se añade a la clasificación. Si la clasificación tiene más de 20 entradas, se elimina la entrada más baja.

    - updateRankingCM(String playerName, int score)
        - Actualiza la clasificación CodeMaker con una nueva entrada para el jugador y la puntuación especificada.
          Si la clasificación no está completa (menos de 20 entradas), o si la nueva puntuación es mayor que la puntuación más baja de la clasificación, la nueva entrada se añade a la clasificación. Si la clasificación tiene más de 20 entradas, se elimina la entrada más baja.
    - getRankingCB(): ArrayList<Pair<String,Integer>>
    - setRankingCB(ArrayList<Pair<String,Integer>> rankingCB)
    - getRankingCM(): ArrayList<Pair<String,Integer>>
    - setRankingCM(ArrayList<Pair<String,Integer>> rankingCM)
    - isInRankingCB(String playerName): boolean
        - Comprueba si el jugador especificado está en el ranking de codebreaker.
    - isInRankingCM(String playerName): boolean
        - Comprueba si el jugador especificado está en el ranking de codemaker.
    - getScoreCB(String playerName): int
    - getScoreCM(String playerName): int
    - updateRanking(String playerName, int score, ArrayList<Pair<String,Integer>> ranking)
        - Método privado que actualiza la clasificación especificada con una nueva entrada para el jugador y la puntuación especificada.

Si la clasificación no está completa (menos de 20 entradas), o si la nueva puntuación es mayor que la puntuación más baja de la clasificación, la nueva entrada se añade a la clasificación. Si la clasificación tiene más de 20 entradas, se elimina la entrada más baja.
La clasificación se ordena por puntuación en orden descendente.

- isInRanking(String playerName, ArrayList<Pair<String,Integer>> ranking): boolean
  - Método privado para comprobar si el jugador especificado está en el ranking especificado.
- getScore(String playerName, ArrayList<Pair<String,Integer>> ranking): int
  - Método privado que devuelve la puntuación de un jugador en el ranking. Si el jugador no está en el ranking, devuelve 0.

Records

Esta clase es responsable de almacenar los registros de los récords del juego. Los récords que se guardan son: la partida más rápida y lenta, partida con más y menos puntuación y la partida con más y menos puntos.

- Atributos:
    - recordsMap: variable privada donde se almacena, en un mapa, el conjunto de todos los récords del juego.
- Métodos:
    - Records()
        - creadora del mapa de récords y los inicializa.
    - initRecords()
        - inicialización del mapa de récords vacíos.
    - updateRecords(String playerName, int score, int seconds, int turns)
        - según los parámetros pasados por parámetros, actualiza el mapa de récords. Como la forma de puntuar una partida de codemaker es distinto a los de codebreaker, solo se actualizará los records cuando la partida es de codebreaker. Ya que sinó, no sería justo para ambos roles.
    - checkFastest(int seconds): boolean
        - Comprueba si el parámetro pasado es mejor que el récord del mismo. Devuelve true si un tiempo mejor.
    - checkSlowest(int seconds): boolean
        - Comprueba si el parámetro pasado es mejor que el récord del mismo. Devuelve true si un tiempo peor.
    - checkHighScore(int score): boolean
        - Comprueba si el parámetro pasado es mejor que el récord del mismo. Devuelve true si la puntuación es mejor.
    - checkLowScore(int score): boolean
        - Comprueba si el parámetro pasado es mejor que el récord del mismo. Devuelve true si la puntuación es peor.
    - checkMoreTurns(int turns): boolean
        - Comprueba si el parámetro pasado es mejor que el récord del mismo. Devuelve true si una partida con más turnos.
    - checkFewerTurns(int turns): boolean
        - Comprueba si el parámetro pasado es mejor que el récord del mismo. Devuelve true si una partida con menos turnos.
    - setRecords(Map<String, Pair<String, Integer> > data)
    - getRecordsMap(): Map<String, Pair<String, Integer>>
    - getFastestRecord(): Pair<String, Integer>
    - getSlowestRecord(): Pair<String, Integer>
    - getHighScoreRecord(): Pair<String, Integer>
    - getLowScoreRecord(): Pair<String, Integer>
    - getMoreTurnsRecord(): Pair<String, Integer>
    - getFewerTurnsRecord(): Pair<String, Integer>

GameController

La clase GameController es la responsable del funcionamiento de la partida. Se encarga de que todas las clases referentes a la partida funcionen correctamente pasando siempre por el controlador de juego. Así pues la comunicación entre ellas viene dada por este controlador.

- Atributos:
    - player: variable privada donde se almacena la instancia de jugador (clase Player) de la partida.
    - computer: variable privada donde se almacena la instancia de máquina (clase Computer) de la partida.
    - difficulty: variable privada donde se almacena la instancia de dificultad (clase Difficulty) de la partida.
    - board: variable privada donde se almacena la instancia de tablero (clase Board) de la partida.
    - feedback: variable privada donde se almacena la instancia de lógica del feedback (clase Feedback) de la partida.
    - gamestats: variable privada donde se almacena la instancia de estadísticas (clase GameStats) de la partida.
- Métodos:
    - GameController()
        - Creadora de un controlador de partida vacío.
    - setDifficultyToNormal()
        - Método para instanciar una dificultad con opciones de dif. normal.
    - setDifficultyToHard()
        - Método para instanciar una dificultad con opciones de dif. difícil.
    - setDifficultyToCustom(boolean algorithm,boolean repeated_colors, int number_positions, boolean empty_colors, int max_turns)
        - Método para instanciar una dificultad custom con las opciones pasadas por parámetro. Estas opciones las recibe del controlador de dominio.
    - startGame(String player_name, Mode player_role)
        - crea una instancia vacía de las clases que quedaban por instanciar juntamente con el nombre y rol del jugador. Así pues se puede crear el jugador.
    - getPlayerName(): String
        - pide a la clase player el nombre del jugador
    - getGameStats(): GameStats
        - le pide a la clase GameStats todo el objeto GameStats
    - getScore(): int
        - le pide a GameStats la puntuación actual
    - getTurn(): int
        - le pide a GameStats el turno actual
    - getMaxTurns(): int
        - le pide a Difficulty el número máximo de turnos
    - getBoard(): Ball_color[][]
        - le pide a la clase Board el tablero actual

- getBoardFeedback(): Pair<Integer, Integer>
  - le pide a la clase Board el tablero de feedbacks actual
- getCurrentFeedback(): Pair<Integer, Integer>
  - le pide a la clase Board solo el feedback del turno actual
- getPlayerCurrentCode(): Ball_color[]
  - le pide a la clase Player el código actual que está poniendo
- getSecretCode(): Ball_color[]
  - le pide a la clase Feedback el código secreto
- getElapsedTime(): int
  - le pide a la clase GameStats el tiempo que ha pasado de partida
- newColor(int position, Ball_color color)
  - Envía el color y la posición deseada al jugador para almacenarla en su código actual.
- cleanCode()
  - Notifica al jugador que limpie las casillas de su código.
- confirmCode(): boolean
  - Indica al jugador que el código ha sido completado y confirmado.
    1) Si el rol del jugador es CodeMaker, esta será la confirmación de lo que será el código secreto. El código devuelto por la confirmación del Jugador será establecido en Feedback como el código secreto y será enviado a Computer para resolver el juego. El resultado dado por el Computer será enviado al Board para guardar la información.
    2) Si el rol del jugador es CodeBreaker, simplemente se enviará a Board para actualizar la fila correspondiente y se comparará con el código secreto guardado en Feedback.

    RETORNO) Esta función también dice si el juego ha terminado dentro del max_turns devolviendo true, o si todavía está en curso devolviendo false.

- checkIfGameWon(int first): boolean
  - Método auxiliar para completar el confirmCode() que calcula el estado de la partida.
- setBoardToComputerSolution(): boolean
  - Función auxiliar para ayudar a confirmCode() a poner al tablero la solución completa dada por Computer y la respuesta a cada fila. Esta función sólo será llamada si el rol del jugador es CodeMaker. Esto significa que esta función completará el juego con la solución dada por Computer y devolverá TRUE si el ordenador fue capaz de descifrar el código o FALSE si el ordenador no fue capaz de descifrar el código dentro de los turnos máximos.

- newSecretCodeComputer()
  - Método para avisar de que la máquina cree aleatoriamente un código secreto y se almacenará en la clase feedback.
- desiredHint()
  - Método que se llama cuando el jugador pide una pista. Directamente coge la pista y la coloca en el código actual del jugador.

- surrender()
  - Avisa a gamestats de que la partida ha acabado por que el jugador se ha rendido. Y pone su puntuación a 0.
- restoreGame():
  - método que se llama después de settear todos los valores necesarios para la partida. Genera el computer i pide continuar la partida a los game stats.
- setSavedPlayer(String player_name, Mode player_role, int num_positions)
  - settea los valores del jugador guardados en el dominio.
- setSavedDifficulty(String diff, boolean algorithm,boolean repeated_colors, int number_positions, boolean empty_colors, int max_turns)
  - settea las opciones de la dificultad que previamente estaban guardados, a su respectiva clase.
- setSavedSecretCode(Ball_color[] sc)
  - settea el codigo secreto guardado a dominio.
- setSavedGameStats(int elapsedTime, int numTurn, int hintsUsed)
  - settea los valores necesarios de gamestats guardados, para su clase.
- setSavedBoardState(Ball_color[][] saved_board, Pair<Integer,Integer>[] saved_feedback)
  - settea los valores necesarios de la clase tablero para seguir la partida que estaban guardados en persistencia.
- getPlayerRole(): Mode
  - le pide a Player el rol del jugador
- getDiffNumPositions(): int
  - le pide a la clase Difficulty el número máximo de posiciones posibles
- getDiffAlgorithm(): boolean
  - le pide a Difficulty qué algoritmo se ha escogido
- getDiffRepeated(): boolean
  - le pide a Difficulty si se permiten colores repetidos
- getDiffEmpty(): boolean
  - le pide a Difficulty si se permiten colores vacíos
- getHintsUsed(): int
  - le pide a GameStats el número de pistas usadas
- notifyPause()
  - pide a gamestats que pare el timer.
- notifyResume()
  - pide a gamestats que encienda el timer otra vez.

Difficulty

Clase de dificultad del juego. Esta clase proporciona la información de los ajustes de dificultad. Tiene tres subclases, una para cada dificultad disponible.

- Atributos:
    - algorithm: booleano que describe el algoritmo a usar. True es five-guess y false se considerará el genético.
    - repeated_colors: booleano que permite colores repetidos o no.
    - number_positions: variable para almacenar el número de bolas por código.
    - empty_colors: booleano que permite colocar o dejar colores vacíos.
    - max_turns: variable que almacena el número máximo de turnos por partida.
    - multiplier: variable que sirve para escoger un factor que multiplicará a la puntuación del jugador. Este multiplicador se calculará dependiendo de las opciones escogidas. Cuanta más complejidad, mayor será el multiplicador.
- Métodos:
    - Difficulty()
        - creadora de instancia dificultad vacía
    - getAlgorithm(): boolean
    - getRepeatedColors(): boolean
    - getNumberPositions(): int
    - getEmptyColors(): boolean
    - getMaxTurns(): int
    - getMultiplier(): double

DifficultyNormal

Clase DifficultyNormal, extiende de Difficulty. Esta clase establece un grupo de valores considerados "normales" en la clase Difficulty. Estos valores son, algoritmo 5-guess, colores repetidos no permitidos, 4 bolas por código, no se permiten colores vacíos y un máximo de 8 turnos.

- Métodos:
    - DifficultyNormal()
        - Crea una instancia de dificultad normal como subclase de dificultad.
    - setValuesNormal()
        - Define los valores de la clase padre dificultad con los ya documentados en la descripción de la subclase.

DifficultyHard

Clase DifficultyHard, extiende de Difficulty. Esta clase establece un grupo de valores considerados "difíciles" en la clase Difficulty. Estos valores son, algoritmo 5-guess, colores repetidos no permitidos, 7 bolas por código, no se permiten colores vacíos y un máximo de 8 turnos.

- Métodos:
    - DifficultyHard()
        - Crea una instancia de dificultad difícil como subclase de dificultad.
    - setValuesHard()

        - Define los valores de la clase padre dificultad con los ya documentados en la descripción de la subclase.

DifficultyCustom

La clase DifficultyCustom, extiende a Difficulty. Esta clase establece un grupo de valores escogidos por el usuario en la clase Difficulty.
Permite modificar cada uno de los valores de la clase Difficulty para obtener un juego totalmente personalizado.
Esta clase también recalcula el multiplicador para ajustarlo a los valores seleccionados.

- Métodos:
    - setValuesCustom()
        - inicializa los valores de Difficulty por defecto (dificultad normal), para luego poder ser cambiados, si es necesario.
    - setAlgCus(boolean alg)
    - setRepColCus(boolean rep_col)
    - setNumPosCus(int num_pos)
    - setEmptyColCus(boolean emp_col)
    - setMaxTurCus(int max_tur)
    - setMulCus()
        - calcula el multiplicador de la dificultad según las opciones almacenadas.

Player

La clase Player representa al jugador que juega contra la máquina en una partida, tanto de codebreaker o de codemaker. Esta clase es responsable de guardar tanto la información del jugador: nombre y rol, como el código del turno que está actualmente jugando.

- Atributos:
    - Nombre: Nombre que identifica al jugador. Es de tipo String.
    - Rol: Rol del jugador en la partida. Es un enum “Mode” que puede tomar los valores Codemaker y Codebreaker.
    - currentCode: Código del turno o código secreto a llenar con colores. Es un array de enums “Ball_color”.
- Métodos:
    - Player()
        - creadora vacía de Player.
    - Player(String name)
        - creadora con nombre.
    - addColor(int pos, Ball_color col)
        - Añade el color col en la posición pos del currentCode, devuelve una excepción PositionOutOfBounds en caso de que la posición esté fuera del rango del código.
    - cleanCode()
        - Llena el currentCode de espacios vacíos, es decir de Ball_Color.Empty.
    - confirmedCode(boolean repeatedColors, emptyColors): Ball_color[]
        - Comprueba si el código currentCode es consistente con los dos booleanos. En el caso de que repeatedColors sea falso comprueba que no tenga colores repetidos, y en el caso de que emptyColors sea falso, comprueba que el código no tenga colores repetidos. En el caso de ser consistente, retorna el currentCode, y en caso contrario lanza las excepciones NotRepeatedColors o NotEmptyColors.
    - initCode(int numElementos)
        - Inicializa el currentCode con una medida de numElementos, y llena el currentCode de espacios vacíos.
    - hasRepeatedColors():boolean
        - Comprueba si currentCode tiene códigos repetidos.
    - hasEmptyColors():boolean
        - Comprueba si currentCode tiene espacios vacíos
    - getName()
    - getRole()
    - getCurrentCode()
    - setRole()
    - setCurrentCode(Ball_color[] code)

Computer

La interfaz Computer determina cómo tiene que ser la máquina en una partida. Solo contiene el método solve.

CodeMakerComputer

La clase CodeMakerComputer es una clase que implementa la interfaz Computer. Esta clase representa a la máquina cuando juega de code maker, es decir es la encargada de generar el código secreto cuando la máquina juega de code maker.

- Atributos:
  - lengthcode: Indica la medida del código secreto a generar
  - repeatedColors: Indica si en el código secreto se pueden repetir los colores.
  - numColors: Indica el número máximo de colores que puede utilizar el código secreto, varía según si se pueden dejar espacios vacíos
- Métodos:
  - CodeMakerComputer(int lengthcode, boolean emptyColors, boolean repeatedColors)
    - Creadora con parámetros de la clase CodeMakerComputer
  - solve(List<Integer> solution): List<List<Integer>>
    - El método solve de la clase CodeMakerComputer crea un secret code aleatorio consistente con los atributos de la clase. En este caso no utiliza el parámetro solution. Retorna una lista de arrays de Ball_Color de un solo elemento con el código generado.

FiveGuessComputer

La clase FiveGuessComputer representa a la máquina cuando juega de code breaker y utiliza el algoritmo Five Guess para dar una solución a la partida. És la encargada de contener la inteligencia del algoritmo Five Guess.

- Atributos:
  - maxSteps: Indica el número máximo de turnos que tiene el algoritmo para resolver.
  - lengthcode: Indica la medida del código secreto a generar
  - repeatedColors: Indica si en el código secreto se pueden repetir los colores.
  - maxColor: Indica el número máximo de colores que puede utilizar el código secreto, varía según si se pueden dejar espacios vacíos
- Métodos:
  - FiveGuessComputer(int maxSteps, int lengthcode, boolean emptyColors, boolean repeatedColors)
    - Creadora con parámetros de la clase FiveGuessComputer.

- solve(List<Integer> solution): List<List<Integer>>
  - El método solve de la clase FiveGuessComputer resuelve el algoritmo Five Guess de Mastermind dado un código solution. Este método devuelve una lista de códigos (List<List<Integer>>) donde cada código corresponde a cada uno de los intentos en cada turno. En caso de que el código solution proporcionado no sea consistente con los atributos, es decir, si se pueden repetir colores, los colores que se pueden utilizar y la medida del código, este lanzará una de las excepciones: NotRepeatedColors, NotEmptyColors, DifferentLengthCode o IncorrectColor.
- checkSolution(List<Integer> solution)
  - Comprueba que la solución sea consistente con los parámetros de la partida.
- getNextGuess(List<List<Integer>> possibleCodes, List<List<Integer>> combinations)
  - Genera el próximo código de entre todas las combinations que en el peor de los casos elimine al máximo de los possibleCodes.
- createSet(): List<List<Integer>>
  - Crea el set con todas las permutaciones compatibles con los parámetros de la partida.
- getFirstGuess(): List<Integer>
  - Genera la primera guess de la partida.
- randomGuess(): List<Integer>
  - Genera un código random compatible con la partida
- isSolved(Pair<Integer,Integer> feedback)
  - Dado un feedback comprueba si este es perfecto.
- eliminateCodes(List<Integer>guess,Pair<Integer,Integer>fd, List<List<Integer>> possibleCodes): List<List<Integer>>
  - Dado la guess actual y su feedback elimina los códigos de possibleCodes que ya no son posibles.
- compareCodes(List<Integer> code, List<Integer> guess): Pair<Integer,Integer>
  - Compara dos códigos y devuelve el feedback con los colores correctos en posiciones correctas y los colores correctos en posiciones incorrectas.
- hasRepeatedColors(List<Integer> code): boolean
- hasEmptyColors(List<Integer> code): boolean
- hasOtherColor(List<Integer> code): boolean
- minCount(Map<List<Integer>,Integer> scores): int
  - busca, entre todos los scores, el mínino y lo devuelve
- getMaxCount(Map<Pair<Integer,Integer>,Integer> count): int
  - busca, entre todos los count, el máximo y lo devuelve

GeneticComputer

La clase GeneticComputer representa a la máquina cuando juega de code breaker y utiliza el algoritmo “Genético” para dar una solución a la partida. Es la encargada de contener la inteligencia del algoritmo genético.

- Atributos:
    - maxSteps: Indica el número máximo de turnos que tiene el algoritmo para resolver.
    - lengthCode: Indica la medida del código secreto a generar.
    - repeatedColors: Indica si en el código secreto se pueden repetir los colores.
    - maxColor: Indica el número máximo de colores que puede utilizar el código secreto, varía según si se pueden dejar espacios vacíos
    - population_size: máximo tamaño que dejamos para cualquier población generada. No hace falta que llegue al máximo.
    - game_history: lista para almacenar todas las soluciones propuestas de la partida por turnos. Para así poder ver el historial y aprender de él.
    - population: lista con las posibles soluciones para encontrar luego la más ideal.
    - fitness: lista con los valores de fitness calculados de cada uno de los elementos de la lista population. Se usa para poder ordenar por mejor fitness la lista population.
    - prev_fitness: lista para almacenar todos los feedbacks otorgados en cada turno.
- Métodos:
    - GeneticComputer(int maxSteps, int lengthCode, boolean emptyColors, boolean repeatedColors)
        - Creadora inicial con los parámetros de la clase.
    - hasRepeatedColors(List<Integer> code): boolean
        -
    - hasEmptyColors(List<Integer> code): boolean
    - hasOtherColor(List<Integer> code): boolean
    - checkSolution(List<Integer> solution)
        - Comprueba que la solución sea consistente con los parámetros de la partida.
    - compareCodes(List<Integer> code, List<Integer> guess): Pair<Integer,Integer>
        - Compara dos códigos y devuelve el feedback con los colores correctos en posiciones correctas y los colores correctos en posiciones incorrectas.
    - isSolved(Pair<Integer,Integer> feedback)
        - Dado un feedback comprueba si este es perfecto.
    - solve(List<Integer> solution): List<List<Integer>>
        - El método solve de la clase GeneticComputer resuelve el algoritmo genético de Mastermind dado un código solution. Este método devuelve una lista de códigos (List<List<Integer>>) donde cada código corresponde a cada uno de los intentos en cada turno. En caso de que el código solution proporcionado no sea consistente con los atributos, es decir, si se pueden repetir colores, los colores que se

pueden utilizar y la medida del código, este lanzará una de las excepciones: NotRepeatedColors, NotEmptyColors, DifferentLengthCode o IncorrectColor.

- createStartingPopulation(): void
  - crea la primera población para el algoritmo genético. Se generan de forma aleatoria pero el primer guess lo prepara dependiendo de la longitud del código.
- evolvePopulation(List<List<Integer>> newPopulation): void
  - evoluciona la población de códigos aplicando operadores de cruce y mutación. Toma como parámetro una lista llamada newPopulation que se llenará con otros códigos posibles. Inicialmente, esta lista comienza con dos códigos que son los padres de la población. Al finalizar, se establece la población global del programa con la nueva población generada.
- permutation_full_crossover(List<List<Integer>> new_population, List<Integer> code1, List<Integer> code2): void
  - Se llama a permutation_crossover de las dos maneras para crear nuevas posibles soluciones mediante crossover.
- permutation_crossover(List<Integer> code1, List<Integer> code2): List<Integer>
  - realiza un cruce por permutación entre dos códigos padres para generar un nuevo código. Toma como parámetros dos códigos padres, code1 y code2, para este cruce específico. El método devuelve el nuevo código generado por el cruce por permutación.
- mutate(List<List<Integer>> newPopulation,List<Integer> code1)
  - muta un código cambiando aleatoriamente algunos de sus elementos. Toma como parámetros la población newPopulation a la cual se agregará el código mutado y el código original a ser mutado (code1).
- inversion(List<List<Integer>> newPopulation,List<Integer> code1)
  - realiza una inversión en un código intercambiando dos posiciones aleatorias. Recibe como parámetros la población newPopulation a la cual se agregará el código mutado y el código original a ser mutado (code1).
- calculateAllFitness(List<List<Integer>> codes)
  - este método guarda a fitness los fitness calculados para cada uno de la población pasada en el parámetro codes. Esto lo hace llamando a una función que calcula solamente un fitness así pues recorre la población para ello.
- calculateSingleFitness(List<Integer> code): double
  - calcula el valor de aptitud de un único código basándose en las soluciones previas recibidas durante el juego. El método acepta un parámetro code, que representa el código para el cual se calculará la fitness, y devuelve el valor de fitness calculado.
- sortPopulationByFitness(List<List<Integer>> codes): List<List<Integer>>
  - El método sortPopulationByFitness ordena la población de códigos en función de su aptitud. El primer elemento de la lista resultante será la mejor solución en la población. El método acepta un parámetro codes, que representa la población de códigos a ser ordenada, y

devuelve la lista ordenada de códigos. Usa bubblesort com algoritmo de ordenación.

- nextGuess(): List<Integer>
  - devuelve la mejor solución. Es decir, la primera posición, pero por si por alguna razón la mejor opción ya ha sido usada anteriormente, se envía la siguiente que no esté usada.
- giveRandomColor(List<Integer> code, int pos): Integer
  - devuelve un color posible al azar para una posición específica de un código. Toma como parámetros el código en el que se buscará un nuevo color y la posición del color que se reemplazará.
- fillPopulation(List<List<Integer>> newPopulation): void
  - se encarga de llenar la población con códigos generados aleatoriamente hasta que se alcance el tamaño de población deseado.
- setFitness(List<Double> fitness)
- setPopulation(List<List<Integer>> new_populationlist)
- getFitness(): List<Double>
- getPopulation(): List<List<Integer>>
- setGameHistory(List<List<Integer>> gh)
- getGameHistory(): List<List<Integer>>
- setPrevFeedback(List<Pair<Integer, Integer>> pg)
- getPrevFeedback(): List<Pair<Integer, Integer>>

Board

Clase Tablero que se encarga de todo lo relacionado con el tablero de la partida. El tablero implica los colores de los códigos y además de tener los colores de los pines del feedback de cada código. Esta clase, igual que el resto, recibirá información del controlador de juego para actualizar y guardar los códigos que irá recibiendo.

- Atributos:
    - board: estructura de tablero de tipo Ball_color[][] donde se almacenarán los códigos pasados por el codebreaker.
    - feedback: estructura de feedbacks de tipo Pair<Integer, Integer> que deberá tener el tamaño igual al número de turnos máximo.
- Métodos:
    - Board()
        - Constructora que crea una instancia nueva de tablero vacía. Tanto de tablero de códigos, como de feedbacks.
    - Board(int numBalls, int numTurnsMax)
        - Constructora de una nueva instancia de tablero con los valores especificados por parámetro.
    - updateRowBoard(Ball_color[] codePlayer, int numTurn)
        - Actualiza los valores de la fila de código con el código confirmado por el usuario.
    - updateFeedback(int numTurn, Pair<Integer,Integer> fd)
        - Actualiza la estructura de feedbacks con el Pair recibido por parte de la clase Feedback pasada por el controlador de juego.
    - getBoard(): Ball_color[][]
    - setBoard(Ball_color[][] board)
    - getFeedback(): Pair<Integer,Integer>[]
    - getCurrentFeedback(int numTurn): Pair<Integer, Integer>
    - setFeedback(Pair<Integer,Integer>[] feedback)
    - setRow(Ball_color[] codePlayer, int numTurn): Ball_color[][]
        - Método que coloca en la fila del tablero corresponidente el código especificado por el jugador.

GameStats

La clase GameStats es la encargada de almacenar las estadísticas de una partida en curso y la responsable de calcular la puntuación final. Además,es la clase que debe controlar los timers de la partida. Esta clase será llamada solamente por el game controller para que el resto de clases referentes a la partida sepan las estadísticas de la partida.

- Atributos:
    - score: variable privada que almacena puntuación de la partida que es calculada una vez se ha finalizado.
    - numTurn: variable privada que almacena el número de turno actual de la partida.
    - numHintsUsed: variable privada que almacena el número de pistas pedidas por el jugador.
    - startTime: variable privada donde se almacena la instancia de tiempo inicial de la partida.
    - endTime: variable privada donde se almacena la instancia de tiempo final de la partida.
    - elapsedTime:variable privada que almacena la duración en segundos de la partida acabada.
- Métodos:
    - GameStats()
        - creadora donde se inicializan los atributos a 0, menos el tiempo.
    - GameStats(int new_numTurn, int new_numHintsUsed, int new_elapsedTime)
        - creadora donde se inicializa la clase pero con los valores de una partida guardada
    - startGameStats()
        - método para empezar la partida, turno es 1 y empieza el temporizador.
    - pauseGameStats()
        - Método para pausar el juego y controlar el tiempo. Además de controlar que las puntuaciones, cuando se reanuda la partida, sigan en orden.
    - restoreGameStats()
        - Método para reaunudar una partida que había sido anteriormente pausada para así poder volver a encender el temporizador.
    - finishGameStats(Mode playerRole, int maxTurns, double multiplier)
        - con los parámetros pasados, hará todos los pasos necesarios para poder calcular y obtener toda la información necesaria respecto a una partida finalizada. Como sería parar el temporizador, calcular la puntuación y guardarla.
    - addTurn(int maxTurns)
        - Método para añadir un turno al contador de turnos.
    - addHintUsed()
        - método para añadir una pedida más de ayuda.
    - startTimer()
        - Método para encender el temporizador.
    - endTimer()

        - Método para parar el temporizador.
    - calculatedElapsedTime(): int
        - Método que calcula la duración de la partida, en segundos, teniendo en cuenta el startTime y el endTime establecido anteriormente. Sinó, salta una excepción.
    - calculateScoreCB(int maxTurns, double multiplier): int
        - El método para calcular la puntuación de la partida acaba como codebreaker. Primero calcula la duración, y a partir de ahí, calcula la puntuación final siguiendo una fórmula personalizada original.
    - calculateScoreCM(): int
        - Método para calcular la puntuación de la partida acabada como codemaker. Como los tiempos de las partidas como codemaker son distintos a los de codebreaker, la puntuación refleja solamente la cantidad de turnos que la máquina ha tardado a resolver tu código.
    - getGameStats(): GameStats
    - getScore(): int
    - setScore(int score)
    - getNumTurn(): int
    - setNumTurn(int numTurn)
    - getNumHintsUsed(): int
    - setNumHintsUsed()
    - getStartTime(): Instant
    - setStartTime(Instant startTime)
    - getEndTime(): Instant
    - setEndTime(Instant endTime)
    - getElapsedTime(): int
    - setElapsedTime(int seconds)

Feedback

Clase con la lógica del feedback del juego. Esta clase almacenará el código secreto y dará el feedback necesario a los códigos de entrada para poder jugar los turnos.
Esta clase también proporcionará información del código secreto cuando se solicite una pista.

- Atributos:
    - secret_code: variable privada de tipo Ball_color[] donde se almacena el código secreto escogido por el Codemaker.
- Métodos:
    - Feedback()
        - Creadora de Feedback con valores vacíos.
    - setSecretCode(Ball_color[] sc)
    - getSecretCode(): Ball_color[]
    - compareSecretCode(Ball_color[] code): Pair<Integer, Integer>
        - Este método compara el código nuevo pasado por parámetro con el código secreto ya previamente almacenado. Devuelve un pair de integers que corresponden a el número de bolas en la posición correcta y el número de colores correctos pero no en la posición correcta.
    - askHint(): Pair<Integer,Ball_color>
        - Método llamado cuando el jugador solicita una pista. Coge un color y una posición del código secreto y lo devuelve como Pair.

Utils

Pair

Para una mayor comodidad de representación de valores conjuntos, hemos decidido implementar una clase Pair con el tipo de variables general. Así pues, toda implementación puede hacer uso simplemente declarando de qué tipos es el pair.

- Atributos:
    - first: variable privada con el primer valor del pair sin especificar el tipo.
    - second: variable privada con el segundo valor del pair sin especificar el tipo.
- Métodos:
    - Pair()
        - creadora del pair vacío
    - Pair(T1 i1, T2 i2)
        - creadora del pair con tipos de valores definidos
    - first(): T1
        - consultora del primer valor del pair
    - second(): T2
        - consultora del segundo valor del pair
    - equals(Pair<T1,T2> p): boolean
        - método comparador
    - equals(Object o): boolean
        - método comparador

ParserIntColor

Para hacer el algoritmo de 5-guess se nos proporcionó una cabecera donde se nos restringía usar una lista de integers cuyo número representa un color. Como en el resto del proyecto queremos jugar y enseñar el enum de colores, nos hemos visto con la necesidad de crear una clase parser para poder pasar toda estructura de List<Integer> a Ball_color[]. También es el caso de pasar de List<List<Integer>> a Ball_color[][].

- Métodos:
    - intListToBallColorArray(List<Integer> intList): Ball_color[]
        - Convertir lista de integers a un array de Ball_color
    - ballColorArrayToIntList(Ball_color[] ballColorArray): List<Integer>
        - Convertir un array de Ball_colora una lista de integers
    - intListListToBallColorMatrix(List<List<Integer>> intListList): Ball_color[][]
        - Convertir una lista de listas de integers a una matriz de Ball_color
    - ballColorArrayToListOfIntLists(Ball_color[][]ballColorArray): List<List<Integer>>
        - Convertir una matriz de Ball_color a una lista de listas de integers

Ball_color

Para comprobar qué colores son válidos y para un uso más cómodo de ello, hemos implementado un enum con los colores posibles del juego. Así pues, en lugar de mostrar números o jugar con ellos, podemos directamente referenciar a colores.
Los colores son: rojo, verde, azul, amarillo, lila, naranja, rosa, blanco y vacío.

Mode

Para mayor comodidad y flexibilidad hemos implementado un enum para guardar qué rol tiene el jugador. Roles: CodeMaker y CodeBreaker.

ParserPresentation

Clase que contiene los métodos necesarios para traducir estructuras de datos para que la presentación pueda tratarlos sin problemas.

- Métodos:
    - convertToEntryList(ArrayList<Pair<String, Integer>> pairList): List<Map.Entry<String, Integer>>
        - convierte este arraylist de pairs a una lista de map.entry para que la presentación pueda tratarlo.
    - convertToEntryMap(Map<String, Pair<String, Integer>> pairMap): Map<String, Map.Entry<String, Integer>>
        - convierte el mapa de pairs a mapa de map.entry para que la presentación pueda tratarlo.
    - convertBoardToString(Ball_color[][] board): String[][]
        - convierte el tablero de ball_color a uno de String[][] para que la presentación pueda entenderlo.
    - convertToEntryList(Pair<Integer, Integer>[] pairArray): List<Map.Entry<Integer, Integer>>
        - convierte el array de pairs a una lista de map.entry para que la presentación pueda entenderlo.

ParserPersistence

Clase que contiene los métodos necesarios para poder traducir estructuras de datos a estructuras que persistencia pueda leer. Y también en viceversa. Así pues nos aseguramos de que haya una buena comunicación y que si hay algún cambio, no afecte a la otra clase.

- Métodos:
    - parseColor(String color): Ball_color
        - transforma una string de un color a su Ball_color correspondiente.

- convertMapToList(Map<String, Pair<String, Integer>> map): List<String>
  - parser para pasar un mapa a una lista de strings (para persistencia)
- convertListToMap(List<String> list): Map<String, Pair<String, Integer>>
  - parser para pasar una lista de string al map correspondiente de su estructura de dominio.
- combineRankings(ArrayList<Pair<String, Integer>> rankingCB, ArrayList<Pair<String, Integer>> rankingCM): List<String>
  - coge los dos rankings (CB y CM), los junta en una solo y los pasa a una lista de strings para que persistencia pueda leerlo.
- splitRankings(List<String> combinedList): Pair<ArrayList<Pair<String, Integer>>, ArrayList<Pair<String, Integer>>>
  - coge la lista de strings de los rankings, los separa y los envía separados con sus respectivas estructuras de dominio.

Persistence

PersistenceController

Esta clase consiste en ser el controlador de persistencia el cual se encargará de la comunicación entre la capa de dominio y los managers de persisténcia. Este controlador se encarga de llamar a cada uno de los managers de persistencia para guardar o cargaar partidas, rankings y récords, además de crearlos.

- Atributos
    - gameManager: Atributo que contiene la instancia de GameManager
    - rankingsManager: Atributo que contiene la instancia de RankingsManager
    - recordsManager: Atributo que contiene la instancia de RecordsManager

- Métodos
    - PersistenceController()
        - Creadora vacía de la clase.
    - loadRanking() : List<String>
        - Llama a RankingsManager para que cargue los datos del archivo Rankings.csv.
    - saveRankings(rankings)
        - Llama a RankingsManager que guarda los elementos de rankings en el archivo Rankings.csv.
    - loadRecords() : List<String>
        - Llama a RecordsManager para que cargue los datos del archivo Records.csv.
    - saveRecords(records)
        - Llama a RecordsManager para que guarde los elementos de records en el archivo Records.csv.
    - loadGame() : List<String>
        - Llama a GameManager para que cargue los datos del archivo Game.csv.
    - saveGame(game)
        - Llama a GameManager para que guarde los elementos de una partida en el archivo Game.csv.
    - existsGame : boolean
        - Llama a GameManager para que consulte si existe una archivo Game.csv.
    - deleteGame()
        - Llama a GameManager para que elimine el archivo Game.csv si existe.

GameManager

Esta clase consiste en gestionar el archivo Game.csv, donde se persiste la información relacionada con una partida.

- Atributos
    - filePath: Consiste en una String que contiene el path relativo al archivo Game.csv

- Métodos
    - GameManager()
        - Creadora vacía de la clase GameManager.
    - loadGame() : List<String>
        - Llama a GameManager para que cargue los datos del archivo Game.csv.
    - saveGame(game)
        - Llama a GameManager para que guarde los elementos de una partida en el archivo Game.csv.
    - existsGame : boolean
        - Llama a GameManager para que consulte si existe una archivo Game.csv.
    - deleteGame()
        - Llama a GameManager para que elimine el archivo Game.csv si existe.

RankingsManager

Esta clase consiste en gestionar el archivo Rankings.csv, donde se persiste la información relacionada con los rankings de la aplicación.

- Atributos
    - filePath: Consiste en una String que contiene el path relativo al archivo Rankings.csv

- Métodos
    - RankingsManager()
        - Creadora vacía de la clase RankingsManager.
    - loadRankings() : List<String>
        - Llama a RankingsManager para que cargue los datos del archivo Rankings.csv.
    - saveRankings(rankings)
        - Llama a RankingsaManager para que guarde los rankings de la aplicación en el archivo Rankings.csv.

RecordsManager

Esta clase consiste en gestionar el archivo Records.csv, donde se persiste la información relacionada con los récords de la aplicación.

- Atributos
    - filePath: Consiste en una String que contiene el path relativo al archivo Records.csv

- Métodos:
    - RecordsManager()
        - Creadora vacía de la clase RecordsManager.
    - loadRecords() : List<String>
        - Llama a RecordsManager para que cargue los datos del archivo Records.csv.
    - saveRecords(records)
        - Llama a RecordsManager para que guarde los records de la aplicación en el archivo Records.csv.

Presentation

PresentationController

Esta clase es la controladora de la capa de presentación. Ésta se encargará de comunicarle todos los eventos y cambios que puedan haber en la interfaz gráfica y se lo comunicará a la capa de dominio. Hemos dividido la capa de presentación en tres pantallas, las cuales podrán comunicarse con el resto mediante esta clase controladora: PrincipalMenu, NewGameOption y InGame.

- Atributos:
    - main: frame principal en el cual se trabaja el entorno gráfico.
    - domainController: variable privada donde se inicializará el controlador de dominio.
    - principalMenu: variable privada donde se almacena la clase que controla la pantalla del menú principal.
    - newGameOptions: variable privada donde se almacena la clase que controla la pantalla de una nueva partida y sus opciones
    - inGame: variable privada donde se almacena la clase que controla la pantalla de la partida.
    - confirmedCode: variable privada de tipo boolean para indicar si se ha confirmado o no el código actual.
- Métodos:
    - PresentationController():
        - Creadora de la clase controladora de presentación que además inicializa el resto de clases de la capa de presentación.
    - initializePresentation():
        - Este método público se encarga de inicializar toda la parte gráfica inicial. Establece un tamaño de ventana de 1024x720 píxeles y carga una imagen de fondo. Configura el título de la ventana como "MasterMind", el cursor como el cursor predeterminado y evita que la ventana sea redimensionable. A continuación, agrega el menú principal y las nuevas opciones de juego a la ventana principal, muestra el menú principal y oculta las nuevas opciones de juego.
    - synchronizationPrincipalMenuToNewGameOptions()
        - Este método público se utiliza para sincronizar el menú principal con las nuevas opciones de juego. Oculta el menú principal y muestra las nuevas opciones de juego.
    - synchronizationNewGameOptionsToPrincipalMenu()
        - Este método público se utiliza para sincronizar las nuevas opciones de juego con el menú principal. Oculta las nuevas opciones de juego y muestra el menú principal.
    - initGame()
        - Método público donde se crea la instancia de partida cuando se da inicio al juego.

- synchronizationPrincipalMenuToInGame()
  - Este método público se utiliza para sincronizar el menú principal con el panel de juego. Desactiva el menú principal, lo oculta y activa el panel de juego, mostrándolo en pantalla.
- synchronizationInGameToPrincipalMenu()
  - Este método público se utiliza para sincronizar el panel de juego con el menú principal. Desactiva el panel de juego, lo oculta y activa el menú principal.
- synchronizationNewGameOptionsToInGame()
  - Este método público se utiliza para sincronizar las nuevas opciones de juego con el panel de juego. Activa el panel de juego y lo muestra en pantalla.
- savedGameExist(): boolean
  - Pregunta a la capa de dominio (su controlador) si existe una partida guardada
- getScore(): int
  - Pide a la capa de dominio la puntuación del jugador.
- getPlayerRole(): String
  - Pide a la capa de dominio el rol del jugador.
- saveGame()
  - Avisa a dominio para que guarde la partida
- notifyStartGame()
  - Avisa a dominio que la partida ha empezado.
- loadSavedGame()
  - Pide a dominio que se cargue la partida, que estaba en persistencia, para poder continuarla.
- newColor(int position, String color)
  - Avisa a dominio que se ha escogido un nuevo color en el código actual del jugador
- confirmCode(): boolean
  - Avisa al controlador de dominio que el jugador ha confirmado su turno. Éste devuelve true si el código confirmado resulta ser el secreto.
- cleanCode()
  - Avisa al controlador de dominio que borre el código actual.
- surrender()
  - Avisa al controlador de dominio que el jugador se ha rendido, por lo tanto el juego debe acabar.
- askForHint()
  - avisa al controlador de dominio que el usuario ha pedido una pista.
- loadStart()
  - método para poner en dominio la partida guardada, los rankings y récords.
- getRankingCM(): List<Map.Entry<String, Integer>>
  - Le pide al controlador de dominio el ranking de codemaker actual
- getRankingCB(): List<Map.Entry<String, Integer>>

        - Le pide al controlador de dominio el ranking de codebreaker actual.
    - getRecords(): Map<String, Map.Entry<String, Integer>>
        - Le pide al controlador de dominio los récords actuales
    - getBoard(): String[][]
        - Le pide al controlador de dominio el tablero actual.
    - getFeedbackBoard(): List<Map.Entry<Integer,Integer>>
        - Le pide al controlador de dominio el tablero de feedbacks actual.
    - getCurrentCode(): String[]
        - Le pide a dominio al controlador de dominio el código actual de jugador.
    - getCurrentFeedback(): Map.Entry<Integer,Integer>
        - Le pide a dominio al controlador de dominio el feedback del turno actual.
    - sendDifficultyCustom(boolean algorithm,boolean repeated_colors, int number_positions, boolean empty_colors, int max_turns)
        - Avisa al controlador de dominio que se quiere crear una partida con la dificultad custom junto a las opciones pedidas.
    - sendDifficultyNormal()
        - Avisa al controlador de dominio que se quiere crear una partida con la dificultad normal.
    - sendDifficultyHard()
        - Avisa al controlador de dominio que se quiere crear una partida con dificultad difícil.
    - getMaxTurns(): int
        - Pregunta a dominio por los turnos máximos.
    - getMaxBalls(): int
        - Pregunta a dominio por las bolas máximas.
    - sendNewPlayer(String playerName, String player_role)
        - Avisa al controlador de dominio que se ha establecido un nombre de jugador y entonces empieza la partida. Se debe haber llamado antes los métodos de dificultad.
    - getTurn(): int
        - Pregunta a dominio por el turno actual.
    - notifyPause()
        - método que notifica a dominio que la partida ha sido pausada, para así, poder parar el timer.
    - notifyResume()
        - método que notifica a dominio que la partida ha sido restaurada del pause, para así, poder continuar el timer.
    - notifyUpdateRecords()
        - notifica a dominio para que actualice los récords.
    - notifyUpdateRankingCM(boolean win)
        - notifica a dominio para que actualice los rankings de code maker dependiendo de si ha ganado la partida o no.
    - notifyUpdateRankingCB(boolean win)
        - notifica a dominio para que actualice los rankings de code breaker dependiendo de si ha ganado la partida o no.
    - notifyEndedGame()

        - notifica a dominio que la partida ha acabado. Así sabrá cuándo avisar a persistencia que borre su partida guardada, si es que existe.
    - exitGame()
        - Método encargado de guardar los rankings y récords cuando el jugador sale del juego.

PrincipalMenu

Esta clase es la encargada de mostrar e imprimir por pantalla el menú principal del juego, es la primera ventana que se visualiza al iniciar el juego. Desde esta clase se puede iniciar y continuar partida, mostrar ranking y records y salir del juego, guardando el estado de ranking y records y una partida si esta no ha terminado.

- Atributos:
    - presentationController: Instancia de la clase PresentationController encargada de las sincronizaciones y la comunicación con dominio.
    - newGameButton: Variable privada de tipo JButton, es el botón con el que se inicia una nueva partida y da paso a la elección de dificultad.
    - continueGameButton: Variable privada de tipo JButton, botón con el que se continúa una partida ya iniciada con anterioridad.
    - rankingButton: Variable privada de tipo JButton, botón con el que se muestran los rankings del CodeMaker y CodeBreaker por pantalla.
    - recordsButton: Variable privada de tipo JButton, es el botón con el que se muestran los records del CodeBreaker por pantalla.
    - exitButton: Variable privada de tipo JButton, es el botón que se utiliza para salir del juego e implícitamente guardar el estado actual del juego.
    - titleLabel: Variable privada de tipo JLabel que contiene el título del juego.
    - viewRanking: instancia de la clase ViewRanking encargada de mostrar los rankings.
    - viewRecords: instancia de la clase ViewRecords encargada de mostrar los records.

- Métodos:
    - PrincipalMenu(PresentationController presentationController)
        - Creadora de PrincipalMenu con una instancia ya inicializada de PresentationController con la que se hacen diferentes sincronizaciones.
    - initializeComponents()
        - Inicializa todos los componentes del menú principal pintándolos y colocándolos en su sitio.
    - getRankingCM(): List<Map.Entry<String, Integer>>
        - Le pide al controlador de presentación el ranking de codemaker actual.
    - getRankingCB(): List<Map.Entry<String, Integer>>

        - Le pide al controlador de presentación el ranking de codebreaker actual.
    - getRecords(): Map<String, Map.Entry<String, Integer>>
        - Le pide al controlador de presentación los récords actuales.
    - newGameButtonActionPerformed()
        - Al pulsar el botón de nueva partida dará paso a la selección de dificultades.
    - continueGameButtonActionPerformed()
        - Al pulsar el botón de continuar partida dará paso a la partida que haya guardado en ese momento.
    - rankingButtonActionPerformed()
        - Al pulsar el botón de ranking se mostrarán los rankings por pantalla.
    - recordsButtonActionPerformed()
        - Al pulsar el botón de récords se mostrarán los récords por pantalla.
    - exitButtonActionPerformed()
        - Al pulsar el botón de salir se cerrará el juego y guardarán los estados actuales.
    - updateContinueButton()
        - Método utilizado para actualizar la apariencia del botón de continuar partida. Sí detecta una partida guardada, este será seleccionable, de lo contrario no.

ViewRanking

Esta clase es la encargada de mostrar tanto el ranking de CodeMaker como el de CodeBreaker. Primeramente, se mostrará el ranking de CodeMaker y se puede hacer el cambio pulsando el botón de CodeBreaker y viceversa.

- Atributos:
    - principalMenu: Instancia de la clase PrincipalMenu encargada de las sincronizaciones del menú principal.
    - backgroundBackToPrincipalMenu: Variable privada tipo JLabel, vuelve a pintar el fondo por encima del panel para poder hacer la sincronización hacia el menú principal.
    - rankingCMButton: Variable privada de tipo JButton, botón con el que se muestra el ranking de CodeMaker por pantalla.
    - rankingCBButton: Variable privada de tipo JButton, botón con el que se muestra el ranking de CodeBreaker por pantalla.
    - exitButton: Variable privada de tipo JButton, es el botón que se utiliza para volver al menú principal.
    - backgroundImageRankingCM: Variable privada de tipo JLabel, almacena la imagen de fondo para el ranking de CodeMaker.
    - backgroundImageRankingCB: Variable privada de tipo JLabel, almacena la imagen de fondo para el ranking de CodeBreaker.

- Métodos:
    - ViewRanking(PrincipalMenu principalMenu)
        - Creadora de ViewRanking con una instancia ya inicializada de la clase PrincipalMenu con la que se hacen sincronizaciones al menú principal.
    - initializeComponents()
        - Inicializa todos los componentes de los rankings pintándolos y colocándolos en su sitio.
    - rankingCMButtonActionPerformed()
        - Al pulsar el botón de rankingCM se mostrará el ranking de CodeMaker por pantalla.
    - rankingCBButtonActionPerformed()
        - Al pulsar el botón de rankingCB se mostrará el ranking de CodeBreaker por pantalla.
    - exitButtonActionPerformed()
        - Al pulsar el botón de salir se volverá al menú principal.
    - setRanking(rankingType)
        - Método privado utilizado para imprimir por pantalla, en la posición correspondiente, el ranking indicado con rankingType (CodeMaker o CodeBreaker).

ViewRecords

Esta clase es la encargada de mostrar los records de CodeBreaker.

- Atributos:
    - principalMenu: Instancia de la clase PrincipalMenu encargada de las sincronizaciones del menú principal.
    - backgroundBackToPrincipalMenu: Variable privada tipo JLabel, vuelve a pintar el fondo por encima del panel para poder hacer la sincronización hacia el menú principal.
    - exitButton: Variable privada de tipo JButton, es el botón que se utiliza para volver al menú principal.
    - backgroundImageRecords: Variable privada de tipo JLabel, almacena la imagen de fondo para los récords de CodeBreaker.

- Métodos:
    - ViewRecords(PrincipalMenu principalMenu)
        - Creadora de ViewRecords con una instancia ya inicializada de la clase PrincipalMenu con la que se hacen sincronizaciones al menú principal.
    - initializeComponents()
        - Inicializa todos los componentes de los récords pintándolos y colocándolos en su sitio.

- exitButtonActionPerformed()
  - Al pulsar el botón de salir se volverá al menú principal.
- setRecords()
  - Método privado utilizado para imprimir por pantalla, en la posición correspondiente, los récords de CodeBreaker.

NewGameOptions

Esta clase es la encargada de mostrar e imprimir por pantalla el menú que se encuentra entre el Menú Principal y lo que es la partida en sí. Esta clase controla y gestiona las tres vistas que dependen de ella: ViewNewPlayer, ViewNewDifficulty y ViewNewCustom.

- Atributos
  - presentationController: Instancia de la clase PresentationController encargada de las sincronizaciones y la comunicación con dominio.
  - viewNewPlayer: Variable privada de tipo JPanel, panel donde se añaden todos los componentes necesarios de la clase.
  - viewNewDifficulty: Variable privada de tipo JPanel, panel donde se añaden todos los componentes necesarios de la clase.
  - viewNewCustom: Variable privada de tipo JPanel, panel donde se añaden todos los componentes necesarios de la clase.
  - role: Variable privada de tipo String, donde se guarda el rol escogido por el jugador.
  - name: Variable privada de tipo String, donde se guarda el nombre escogido por el jugador.
  - difficulty: Variable privada de tipo String, donde se guarda la dificultad escogida por el jugador.
  - values: Variable privada de tipo array de Strings, donde se guardan todos los parámetros necesarios que conforman una dificultad.

- Métodos:
  - NewGameOptions(presentationController)
    - Creadora del NewGameOptions. Al llamar a esta función se crea la clase, se le asigna el controlador de presentación y se crean todos los componentes que necesita la propia clase.
  - initialize()
    - Inicializa los componentes necesarios de la clase.
  - goBack()
    - Carga la pantalla del menú principal y desactiva la actual NewGameOptions.
  - synchronizationPlayerToDifficulty()
    - Pide al controlador de presentación que desactive la pantalla ViewNewPlayer y active la pantalla ViewNewDifficulty.
  - synchronizationDifficultyToPlayer()
    - Pide al controlador de presentación que desactive la pantalla ViewNewDifficulty y active la pantalla ViewNewPlayer.

    - synchronizationDifficultyToCustom()
        - Pide al controlador de presentación que desactive la pantalla ViewNewDifficulty y active la pantalla ViewNewCustom.
    - synchronizationCustomToDifficulty()
        - Pide al controlador de presentación que desactive la pantalla ViewNewCustom y active la pantalla ViewNewDifficulty.
    - startGame()
        - Indica al controlador de presentación que se puede empezar la partida con los parámetros establecidos y que, por tanto, se puede cargar la pantalla InGame.

ViewNewPlayer

Esta clase es la encargada de preguntar al jugador el nombre, con los que se va a registrar en los rankings y récords en el caso que entre las 20 mejores puntuaciones o se bata un récord respectivamente. Es en esta clase también donde el jugador decide si quiere jugar como CodeMaker o CodeBreaker.

- Atributos:
    - newGameOptions: Instancia de la clase NewGameOptions encargada de las sincronizaciones de la selección de opciones antes de dar inicio una partida.
    - background: variable JLabel que le asignará el fondo a la pantalla.
    - name: Variable privada de tipo String para almacenar el nombre introducido por el jugador.
    - backgroundImageName: Variable privada de tipo JLabel, almacena la imagen de fondo donde marca al jugador que tiene que introducir el nombre.
    - backgroundImageIntroduce: Variable privada de tipo JLabel, almacena la imagen de fondo donde el jugador debe introducir su nombre.
    - backgroundImageRole: Variable privada de tipo JLabel, almacena la imagen de fondo donde se muestran la selección de rol.
    - inputName: JLabel privada donde se almacena el asset del input.
    - codeMaker: Variable privada de tipo JButton, botón con el que se selecciona que se quiere jugar como CodeMaker.
    - codeBreaker: Variable privada de tipo JButton, botón con el que se selecciona que se quiere jugar como CodeMaker.
    - exit: Variable privada de tipo JButton, es el botón que se utiliza para volver al menú principal.
    - confirm: Variable privada de tipo JButton, es el botón que se utiliza para confirmar las opciones escogidas.

- Métodos:
    - ViewNewPlayer(newGameOptions)
        - Creadora de ViewNewPlayer con una instancia ya inicializada de la clase NewGameOptions con la que se hacen sincronizaciones de la selección de opciones antes de dar inicio una partida.

- initialize()
    - Inicializa todos los componentes de la selección de rol y nombre de jugador pintándolos y colocándolos en su sitio.
- codeMakerButtonActionPerformed()
    - Al pulsar el botón de codeMaker se quedará marcado para dar a entender que se ha seleccionado la opción de jugar como CodeMaker. Se establece en la variable rol, el rol codemaker.
- codeBreakerButtonActionPerformed()
    - Al pulsar el botón de codeBreaker se quedará marcado para dar a entender que se ha seleccionado la opción de jugar como CodeBreaker. Se establece en la variable rol, el rol codemaker.
- exitButtonActionPerformed()
    - Al pulsar el botón de salir se volverá al menú principal
- confirmButtonActionPerformed()
    - Al pulsar el botón de confirmar se aceptarán las opciones escogidas por el jugador y se dará paso a la selección de dificultades.

ViewNewDifficulty

Esta clase es la encargada de mostrar por pantalla las dificultades disponibles para una partida y dónde el jugador podrá decidir con qué dificultad jugarla.

- Atributos
    - newGameOptions: Atributo contiene una instancia de la clase NewGameOptions.
    - background: variable JLabel que le asignará el fondo a la pantalla.
    - Difficulty: variable privada donde almacenamos la palabra clave de dificultad.
    - values: array de string privado con las opciones predefinidas al inicializar. Éstas estarán sujetas a cambio al escoger distintas dificultades.
    - backgroundImageSelection : Atributo de tipo JLabel contiene la imagen de fondo de la selección de dificultad.
    - difficultyInfo: Atributo de tipo JLabel que describe las características de la dificultad.
    - difficultyText: atributo JLabel con el texto a mostrar de la dificultad.
    - normal : Atributo de tipo JButton que sirve para preseleccionar la dificultad normal.
    - hard : Atributo de tipo JButton que sirve para preseleccionar la dificultad difícil.
    - custom : Atributo de tipo JButton que sirve para seleccionar la dificultad custom.
    - exit : Atributo de tipo JButton que sirve para volver al menú principal.

    - confirm: Atributo de tipo JButton que sirve para confirmar la dificultad preseleccionada.
- Métodos
    - ViewNewDifficulty()
        - creadora de la clase ViewNewDifficulty donde setea valores predefinidos y llama a la función que inicializa todos los componentes.
    - updateText():
        - método para actualizar el contenido de difficultyText que se mostrará.
    - initialize()
        - método que inicializa todos los componentes de esta View.
    - normalButtonActionPerformed(event)
        - Método para preseleccionar la dificultad normal y mostrar su información en dificultyInfo.
    - hardButtonActionPerformed(event)
        - Método para preseleccionar la dificultad difícil y mostrar su información en dificultyInfo.
    - customButtonActionPerformed(event)
        - Método para preseleccionar la dificultad custom y mostrar su información en dificultyInfo.
    - exitButtonActionPerformed(event)
        - Método para volver a la view del menú principal.
    - confirmButtonActionPerformed(event)
        - Método que confirma la dificultad preseleccionada y pasa a la siguiente view (ViewNew custom, si se ha seleccionado custom, o ViewGame en caso contrario)

ViewNewCustom

Esta clase es la encargada de mostrar por pantalla un menú donde el usuario podrá escoger, dentro de la ya escogida dificultad custom, las opciones de la partida.

- Atributos:
    - newGameOptions: Este atributo contiene la instancia de la clase NewGameOptions.
    - background: variable JLabel que le asignará el fondo a la pantalla.
    - values: array de string privado con las opciones predefinidas al inicializar y puede ser modificado al cambiar las opciones.
    - backgroundImageValues: JLabel. Atributo de tipo JLabel que contiene la imagen de fondo de la selección de valores.
    - backgroundAlg: JLabel. Atributo de tipo JLabel que contiene la imagen de fondo del algoritmo.
    - fiveGuess: JButton. Atributo de tipo JButton relacionado con el algoritmo (true-5guess).
    - genetic: JButton. Atributo de tipo JButton relacionado con el algoritmo (false-Genetic).

- backgroundRepCol: JLabel. Atributo de tipo JLabel que contiene la imagen de fondo de colores repetidos.
- repColTrue: JButton. Atributo de tipo JButton relacionado con la opción de colores repetidos (true).
- repColFalse: JButton. Atributo de tipo JButton relacionado con la opción de colores repetidos (false).
- backgroundEmpCol: JLabel. Atributo de tipo JLabel que contiene la imagen de fondo de colores vacíos.
- empColTrue: JButton. Atributo de tipo JButton relacionado con la opción de colores vacíos (true).
- empColFalse: JButton. Atributo de tipo JButton relacionado con la opción de colores vacíos (true).
- backgroundNumCol: JLabel. Atributo de tipo JLabel que contiene la imagen de fondo de la opción longitud de código.
- infoNumCol: JLabel. Atributo de tipo JLabel que contiene el número de la longitud del código seleccionado.
- textNumCol: JLabel que contiene el texto correspondiente a mostrar para la opción.
- addNumCol: JButton. Atributo de tipo JButton relacionado con la opción de longitud de código.
- subNumCol: JButton. Atributo de tipo JButton relacionado con la opción de longitud de código.
- backgroundMaxTur: JLabel. Atributo de tipo JLabel que contiene la imagen de fondo de la opción de máximo de turnos.
- infoMaxTur: JLabel. Atributo de tipo JLabel que contiene el número de turnos máximos seleccionado.
- textMaxTur: JLabel que contiene el texto correspondiente a mostrar para la opción.
- addMaxTur: JButton. Atributo de tipo JButton relacionado con la opción de máximo de turnos.
- subMaxTur: JButton. Atributo de tipo JButton relacionado con la opción de máximo de turnos.
- exit: JButton. Botón de salir.
- confirm: JButton. Atributo de tipo JButton de confirmar opciones.

- Métodos:
  - ViewNewCustom(NewGameOptions newGameOptions)
    - creadora de la clase ViewNewCustom que setea los valores por defecto de la clase. También llama a la función que inicializa los componentes de la vista.
  - initialize()
    - método que inicializa los componentes de la vista.
  - algTrueButtonActionPerformed()
    - Método que pone la elección del algoritmo a true, es decir al algoritmo FirstGuess.
  - algFalseButtonActionPerformed()
    - Método que pone la elección del algoritmo a false, es decir al algoritmo genético.

    - repColTrueButtonActionPerformed()
        - Método que pone la elección de colores repetidos a true, es decir para que se puedan repetir colores.
    - repColFalseButtonActionPerformed()
        - Método que pone la elección de colores repetidos a false, es decir para que no se puedan repetir colores.
    - empColTrueButtonActionPerformed()
        - Método que pone la elección de poder dejar espacios vacíos a true, es decir para que se puedan dejar espacios vacíos.
    - empColFalseButtonActionPerformed()
        - Método que pone la elección de poder dejar espacios vacíos a false, es decir para que no se puedan dejar espacios vacíos.
    - maxColAddButtonActionPerformed(t)
        - Método que incrementa en uno la longitud máxima del código.
    - maxColSubButtonActionPerformed()
        - Método que decrementa en uno la longitud máxima del código.
    - maxTurAddButtonActionPerformed()
        - Método que incrementa en uno la cantidad máxima de turnos.
    - maxTurSubButtonActionPerformed()
        - Método que decrementa en uno la cantidad máxima de turnos.
    - exitButtonActionPerformed()
        - Método que carga la vista ViewNewDifficulty, al pulsar el exitButton.
    - confirmButtonActionPerformed()
        - Método que confirma todas las opciones preseleccionadas y pasa a la ViewGame.

ExceptionsHandler

Esta clase tiene la función de ser llamada cuando el controlador de presentación se encuentra con una excepción que llega desde el dominio. Si llega una excepción, implica que se está haciendo o accediendo a algo que el usuario no debería. Entonces, la finalidad de esta clase es ser llamada para mostrar mensajes de error con mensajes personalizados de cada una de ellas. Así pues, el usuario puede tener constancia de que, por ejemplo, ha intentado confirmar un código con colores repetidos, cuando no se le permite hacerlo.

- Métodos:
    - exceptionsHandler()
        - creadora default vacía.
    - handleException(Exception e)
        - recibe la excepción del controlador de presentación y la trata para poder, así, llamar al método que la enseñará por pantalla.
    - show_warning(String msg)
        - notificar al usuario sobre un error o una situación de advertencia en la aplicación mediante el uso de un cuadro de diálogo de advertencia en la interfaz gráfica del juego.

InGame

Esta clase es la encargada de mostrar por pantalla la parte jugable de la aplicación: la partida. Tiene como objetivo controlar las diferentes vistas que derivan de ella: ViewGame, ViewResult y ViewPause.

- Atributos
    - presentationController : Atributo que contiene la instancia de la clase PresentationController.
    - viewCM: Es el atributo que contiene la instancia de la clase ViewCM.
    - viewCB: Es el atributo que contiene la instancia de la clase ViewCB.
    - gameType: Es el atributo que contiene el tipo de juego que se va a jugar según lo haya seleccionado el jugador.
    - viewPause: Es el atributo que contiene la instancia de la clase ViewPause.
    - viewResult: Es el atributo que contiene la instancia de la clase ViewResult.
    - viewBoard: Es el atributo que contiene la instancia de la clase ViewBoard.

- Métodos
    - InGame(PresentationController presentationController, String role)
        - Creadora de la clase InGame con una instancia inicializada de la controladora y el rol escogido por el jugador.
    - initializeComponents(String role, int maxBalls, int maxTurns)
        - inicializa todos los componentes y atributos de la clase.
    - synchronizeGameToPause()
        - Este método hace visible el JPanel viewPause para poder interactuar con el menú de pausa.
    - synchronizePauseToGame()
        - Este método desvisibiliza el JPanel de viewPause para volver a la pantalla de viewGame.
    - sychronizeGameToResult()
        - Este método hace visible el JPanel viewResult para poder ver los resultados de la partida.
    - show_message(String msg)
        - Metodo que se encaraga de mostrar un mensaje, en forma de JFrame, cuando se acaba de jugar una partida.
    - notifyNewColor(int position, String color)
        - avisa de que el jugador ha propuesto un nuevo color en una posición en concreto.
    - notifyCleanCode()
        - avisa de que el jugador quiere borrar el código actual.
    - saveGame()
        - avisa de que el jugador quiere guardar la partida.
    - notifyAskForHint()
        - avisa de que el jugador quiere una pista.
    - notifyConfirmCode(): boolean

        - avisa de que el jugador confirma el código propuesto. Éste devolverá true si el código confirmado es el mismo que el secreto.
    - notifySurrender()
        - avisa de que el jugador quiere rendirse.
    - getPlayerCurrentCode()
        - pide el código actual que está en dominio.
    - getBoardComputer(): String[][]
        - pide el tablero actual que está en dominio.
    - getFeedbackComputer(): List<Map.Entry<Integer,Integer>>
        - pide el tablero de feedback actual que está en dominio.
    - getNewFeedback(): Map.Entry<Integer,Integer>
        - pide a la controladora de presentacion que pida el feedback del turno actual
    - getTurn(): int
        - pide a la controladora de presentación el turno actual.
    - notifyWin()
        - notifica que ha ganado para enseñar la pantalla de resultados en victoria.
    - notifyLose()
        - notifica que ha perdido para enseñar la pantalla de resultados en derrota.
    - notifyPause()
        - notifica a la controladora de presentación que el jugador ha pausado la partida.
    - notifyResume()
        - notifica a la controladora de presentación que el jugador ha vuelto a la partida.
    - validCode(int actual_turn): boolean
        - método que comprueba si el código que quería confirmar el jugador ha sido válido.
    - notifySaveAndExit()
        - método que guarda la partida y enseña el menú principal.
    - notifyExit()
        - método para la acción de un botón que sale al menú principal.
    - notifyGetScore()
        - Pide a la controladora la puntuación del jugador.
    - notifyResultExit()
        - cambia la vista de result a menú principal.
    - notifyUpdateRecords()
        - notifica a la controladora de presentación que avise que se tiene que actualizar los récords.
    - notifyUpdateRankingCM()
        - notifica a la controladora de presentación que avise que se tiene que actualizar el ranking de code maker.
    - notifyUpdateRankingCB()
        - notifica a la controladora de presentación que avise que se tiene que actualizar el ranking de code breaker.

ViewBoard

Esta clase es la encargada de construir e imprimir por pantalla el tablero. El tamaño de este cambiará y se adaptará según el número de tunos y bolas que el jugador haya seleccionado.

- Atributos
  - maxBalls:Atributo de tipo entero que marca el número de bolas máximas.
  - maxTurns: Atributo de tipo entero que maraca el número de turnos máximos.
  - offsetX: Atributo utilizado para saber el offset entre imagenes en el eje X
  - offsetY:Atributo utilizado para saber el offset entre imagenes en el eje Y
  - scale: Atributo de tipo double utilizado para pintar el tablero con una escala determinada.
  - board: Matriz de tipo JLabel[][] que almacena las imagenes de las bolas dentro del tablero.
  - coords: Matriz de coordenadas donde colocar las bolas seleccionadas por el jugador.
  - coords_feedback: Matriz de coordenadas para pintar el feedback según el número de bolas que se haya seleccionado.
  - coords_number: Matriz de coordenadas para pintar los botones de selección de columna.
  - feedback: Atributo de tipo JLabel[] que almacena las imagenes que conforman el feedback
  - mode: Atributo que almacena el rol que ha seleccionado el jugador.
- Métodos
  - ViewBoard(int maxBalls, int maxTurns, String role)
    - Creadora de la clase ViewBoard con los parámetros necesarios para crear el tablero correctamente.
  - initBoard()
    - Método que inicializar el tablero.
  - init_feedback()
    - Método privado para inicializar el feedback.
  - calculate_feedback_coords()
    - Método privado para calcular las coordenadas para pintar por pantalla la posición donde debe ir el feedback en el tablero.
  - calculate_number_coords()
    - Método privado para calcular las coordenadas donde se pinta por pantalla los números de selección de columna.
  - calculate_feedback_code_coords()
    - Método privado para calcular las coordenadas donde se pinta por pantalla las bolas del feedback.
  - getCoordsColors(): AbstractMap.SimpleEntry<Integer, Integer>[][]
    - Método para obtener las coordenadas donde se pintan los colores en el tablero.

    - getCoordsFeedback(): AbstractMap.SimpleEntry<Integer, Integer>[][]
        - Método para obtener las coordenadas donde se pinta el feedback en el tablero.
    - getCoordsNumber(): AbstractMap.SimpleEntry<Integer, Integer>[][]
        - Método para obtener las coordenadas donde se pintan los números de selección de columna en el tablero.

ViewResult

Esta clase es la encargada de mostrar por pantalla la view de resultados, una vez la partida ha finalizado. Es la encargada de mostrar la puntuación del jugador o la máquina en la partida.

- Atributos
    - inGame: Atributo que contiene la instancia de la clase InGame
    - resultBackground : Atributo de tipo JLabel que almacena la imagen de fondo para la view de resultados.
    - resultTitleLabel: Atributo de tipo JLabel que almacena la imagen “Resultado”.
    - resultLabel: Atributo de tipo JLabel que almacena la imagen de victoria o derrota.
    - scoreLabel: Atributo de tipo JLabel que contiene la puntuación del jugador o máquina (en caso de jugar de codemaker) en la partida.
    - exitButton: Atributo de tipo JButton que sirve para salir de la view de resultados y volver a la view del menú principal
    - win: Atributo tipo boolean, que marca si el jugador ha ganado o no.
- Métodos
    - ViewResult(InGame inGame, boolean win)
        - Creadora de la clase ViewResult con una instancia iniciailizada de la clase InGame y el atributo win que marca si el jugador ha ganado la partida.
    - initializeComponents()
        - Método que inicializa los componentes en los atributos.
    - exitButtonActionPerformed().
        - Método que sale de la view de resultados y vuelve al menú principal. Se ejecuta al pulsar exitButon.

ViewPause

Esta clase es la encargada de mostrar por pantalla la view de pausa cuando se está en una partida.

- Atributos
    - inGame : Atributo que contiene la instancia de la clase InGame
    - continueButton : Atributo de tipo JButton que sirve para volver a la partida.
    - saveAndExitButton : Atributo de tipo JButton que sirve para guardar la partida y volver al menú principal.
    - exitButton : Atributo de tipo JButton que sirve para volver al menú principal sin guardar.
    - pauseImage : Atributo de tipo JLabel que guarda la imagen de la view de pausa.
    - backgroundPauseImage : Atributo de tipo JLabel que guarda la imagen de fondo de la view de pausa.
- Métodos
    - ViewPause(InGame inGame)
        - Creadora de la clase ViewPause con una instancia inicializada de la clase InGame.
    - initializeComponents()
        - Método que inicializa los componentes en los atributos.
    - continueButtonActionPerformed()
        - Método que invisiviliza la view de pausa y vuelve a poner la view de la partida al frente. Se ejecuta al pulsar el continueButton.
    - saveAndExitButtonActionPerformed()
        - Método que guarda la partida y vuelve al menú principal. Se ejecuta al pulsar el saveAndExitButton.
    - exitButtonActionPerformed()
        - Método que vuelva al menú principal. Se ejecuta al pulsar el exitButton.

ViewCM

Esta clase es la encargada de mostrar por pantalla la view de las partidas de CodeMaker.

- Atributos
    - inGame : Atributo que contiene la instancia de la clase InGame
    - displayedCode: Atributo de tipo ArrayList<JLabel> que almacena el conjunto de imágenes de las bolas que se quieren colocar en el tablero.
    - deleteCodeButton: Atributo de tipo JButton que sirve para borrar todo el código escrito en el turno actual en caso de codebreaker, y del código secreto en el caso de codemaker
    - acceptCodeButton: Atributo de tipo Jbutton que sirve para confirmar el código actual y poder recibir el feedback o el resultado de la partida, en el caso de

codebreaker y para confirmar el código secreto y dar lugar a la respuesta del algoritmo en caso de codemake.
    - pauseButton: Atributo de tipo JButton que sirve para pausar la partida.
    - colorsPanel: Atributo de tipo JLabel, almacena la imagen contenedora de los botones de bolas de colores.
    - currentColor: Atributo de tipo String, que contiene el código actual seleccionado por el jugador.
    - currentTurn: Atributo de tipo int que contiene el numero de tuno en el que se encuentra el jugador.
    - maxBalls:Atributo de tipo entero que marca el número de bolas máximas.
    - maxTurns: Atributo de tipo entero que maraca el número de turnos máximos.
    - coordsColors: Matriz donde se guardan las coordenadas donde colocar los colores en el tablero, una vez que el jugador haya aceptado su código actual.
    - coordsFeedback: Matriz de coordenadas para pintar el feedback según el número de bolas que se haya seleccionado.
    - coordsNumber: Matriz de coordenadas para pintar los botones de selección de columna.
    - Atributos de tipo JButton que sirven para seleccionar una bola de un color
        - redButton
        - greenButton
        - blueButton
        - yellowButton
        - orangeButton
        - pinkButton
        - whiteButton
        - purpleButton
    - feedbackSize: Atributo que marca el tamaño del feedback, inicialmente se le asigna un valor de 7.

- Métodos
    - ViewCB(InGame inGame, int maxBalls, int maxTurns, AbstractMap.SimpleEntry<Integer,Integer>[][] coordsColors , AbstractMap.SimpleEntry<Integer,Integer>[][] coordsFeedback, AbstractMap.SimpleEntry<Integer,Integer>[][] coordsButtonPosition)
        - Creadora de la clase ViewCM con los parámetros necesarios para jugar la partida.
    - printColor(int codePos)
        - Método privado para pintar un color en la posición pasada por parámetro.
    - configurePositionButtons()
        - Método que inicializa los botones de selección de columna y los pinta por pantalla.
    - getButtonFromString(String color): JButton
        - Método privado para obtener el botón de bola del color pasado por parámetro.
    - initColorsButtons()
        - Método privado que inicializa los botones de los colores seleccionables y los pinta por pantalla.

- initButtons()
    - Método privado que inicializa los botones básicos (pause, accept, delete) y los pinta por pantalla.
- initLabels()
    - Método privado que inicializa las imágenes que contienen los labels y los pinta por pantalla.
- initializeComponents()
    - Método privado que inicializa los componentes que se pintan por pantalla.
- printBoardCM()
    - Método encargado de pintar el tablero por pantalla en el caso de que se esté jugando como codeMaker.

ViewCB

Esta clase es la encargada de mostrar por pantalla la view de las partidas de CodeBreaker.

- Atributos
    - inGame : Atributo que contiene la instancia de la clase InGame
    - deleteCodeButton: Atributo de tipo JButton que sirve para borrar todo el código escrito en el turno actual en caso de codebreaker, y del código secreto en el caso de codemaker
    - acceptCodeButton: Atributo de tipo Jbutton que sirve para confirmar el código actual y poder recibir el feedback o el resultado de la partida, en el caso de codebreaker y para confirmar el código secreto y dar lugar a la respuesta del algoritmo en caso de codemake.
    - surrenderCodeButton : Es un atributo de tipo JButton que sirve para rendirse.
    - hintButton : Es un atributo de tipo JButton que sirve para pedir una pista.
    - colorsPanel: Atributo de tipo JLabel, almacena la imagen contenedora de los botones de bolas de colores.
    - displayedCode: Atributo de tipo ArrayList<JLabel> que almacena el conjunto de imágenes de las bolas que se quieren colocar en el tablero.
    - currentColor: Atributo de tipo String, que contiene el código actual seleccionado por el jugador.
    - currentTurn: Atributo de tipo int que contiene el numero de tuno en el que se encuentra el jugador.
    - maxBalls:Atributo de tipo entero que marca el número de bolas máximas.
    - maxTurns: Atributo de tipo entero que maraca el número de turnos máximos.
    - coordsColors: Matriz donde se guardan las coordenadas donde colocar los colores en el tablero, una vez que el jugador haya aceptado su código actual.
    - coordsFeedback: Matriz de coordenadas para pintar el feedback según el número de bolas que se haya seleccionado.
    - coordsNumber: Matriz de coordenadas para pintar los botones de selección de columna.

- Atributos de tipo JButton que sirven para seleccionar una bola de un color
  - redButton
  - greenButton
  - blueButton
  - yellowButton
  - orangeButton
  - pinkButton
  - whiteButton
  - purpleButton
- positionButtons: Atributo de tipo ArrayList<JButton> que almacena los botones de selección de posición de columna.
- feedbackSize: Atributo que marca el tamaño del feedback, inicialmente se le asigna un valor de 7.

- Métodos
  - ViewCB(InGame inGame, int maxBalls, int maxTurns, AbstractMap.SimpleEntry<Integer,Integer>[][] coordsColors , AbstractMap.SimpleEntry<Integer,Integer>[][] coordsFeedback, AbstractMap.SimpleEntry<Integer,Integer>[][] coordsButtonPosition)
    - Creadora de la clase ViewCB.
  - printColor(int codePos)
    - Método privado para pintar un color en la posición pasada por parámetro.
  - printFeedback(int turn, boolean win)
    - Método privado para pintar el feedback del turno pasado por parametro.
  - cleanCode()
    - Método privado para borrar el código actual.
  - getButtonFromString(String color): JButton
    - Método privado para obtener el botón de bola del color pasado por parámetro.
  - initColorsButtons()
    - Método privado que inicializa los botones de los colores seleccionables y los pinta por pantalla.
  - initButtons()
    - Método privado que inicializa los botones básicos (pause, accept, delete) y los pinta por pantalla.
  - initLabels()
    - Método privado que inicializa las imágenes que contienen los labels y los pinta por pantalla.
  - initializeComponents()
    - Método privado que inicializa los componentes que se pintan por pantalla.
  - configurePositionButtons()
    - Método que inicializa los botones de selección de columna y los pinta por pantalla.

- printCode(String[] code)
  - Método que pinta el estado del código actual del jugador.
- printBoard(String[][] domainBoard, List<Map.Entry<Integer,Integer>> domainFeedback)
  - Método que pinta el tablero completo, teniendo en cuenta el tablero y el feedback guardado en dominio.