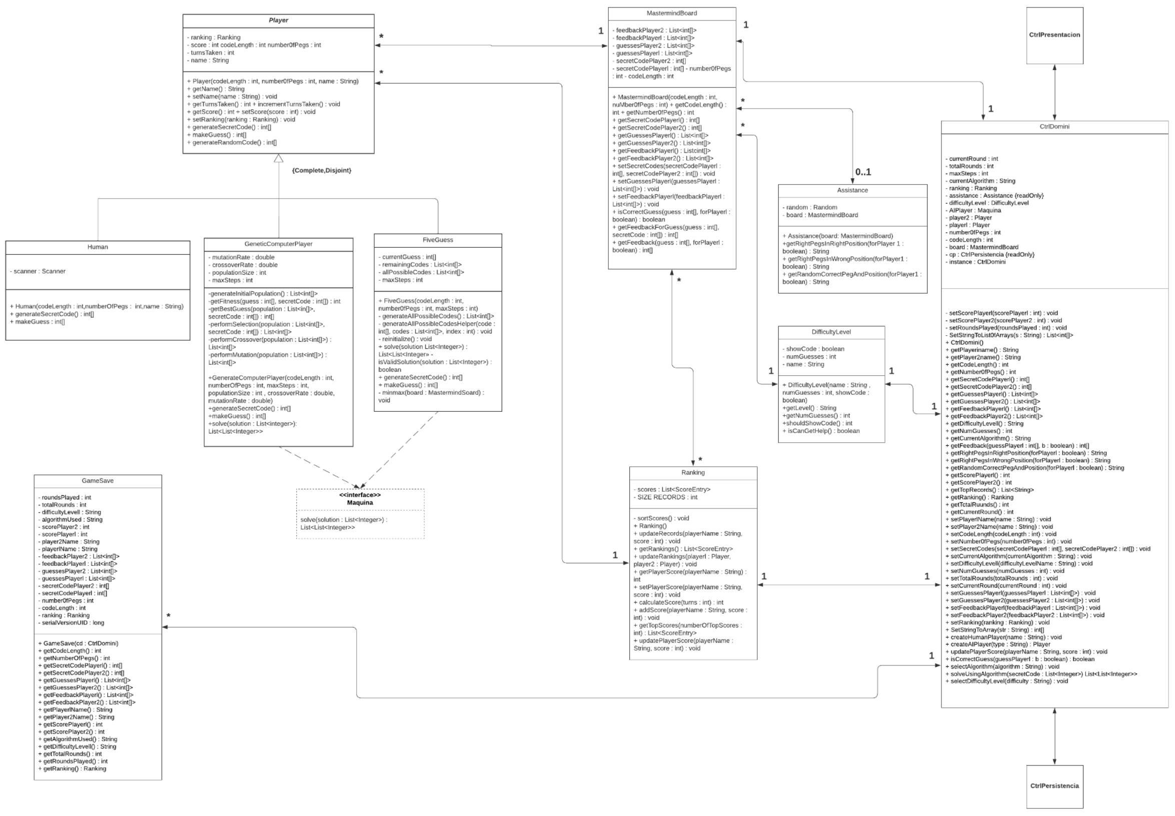
MASTERMIND

1a Entrega PROP, Grup 13.4

Aashish Bhusal : aashish.bhusal@estudiantat.upc.edu
Paolo Milner : paolo.neville.milner@estudiantat.upc.ed
Mario Wang : mario.wang@estudiantat.upc.edu

Description of Class Diagrams

Capa de Domini



1) Class *Player*

This class represents the abstract class player. It has two subclasses Humanplayer and computerplayer. Object of the class player cannot be created, it only contains abstract methods.

Attributes:

a) private String name : Name of the player
b) private int turnsTaken : To keep track of turns taken by each player.
c) protected int numberOfPegs : The total number of pegs in a game.
d) protected int codeLength : Codelength of the game.

Methodes

a) public Player(int codeLength, int numberOfPegs, String name) :

   It is the constructor of class Player.

   - codeLength is the length of the secret code.
   - numberOfPegs is the total number of pegs in the game.
   - The parameter name is the name of the player.

b) public int getTurnsTaken() :

   - It is the getter method to get the turns taken by the player.
   - It returns the number of turns taken by the player.

c) public void incrementTurnsTaken() :
   - This is the method to increment the turns taken by the player.

d) public abstract int[]generateSecretCode() :
   - It is an abstract method to generate a secret code.

e) public abstract int[] makeGuess() :
   - It is an abstract method for the player to make a guess.
   - It returns an Array representing the player's guess.

f) protected int[] generateRandomCode() :
   - It generates a random code to be used as the secret code.

g) public void setScore(int score):
   - Method to set the player's score

2) Class Human

The Human class extends the abstract Player class and provides implementation for the abstract methods of Player class. It represents a human player in the Mastermind game who can act as both CodeMaker and CodeBreaker.

Attributes:

a) private final Scanner scanner :
   - A scanner object to read the user's input.

Methodes:

a) public Human(int codeLength, int numberOfPegs, String name) :
   - It is the Constructor for the Human class that initializes the parent Player class with CodeLength, numberOfPegs and name and creates a new Scanner object for reading input
   - The parameter codeLength is the length of the secret code.
   - The parameter numberOfPegs is the number of pegs used in the game.

- The parameter name is the name of the player.

b) public int[]generateSecretCode() :
   - It overrides the abstract method from the Player class to allow Human to create secret code.
   - Returns an array representing the human player's secret code.

c) public int[] makeGuess() :
   - It overrides the abstract method from the Player class to allow Human to guess the secret code.
   - Returns an array representing the human player's guess.

3) Class FiveGuess

It implements the five guess algorithm to solve the Mastermind game.

Attributes:

a) private int maxSteps
   - Maximum number of steps allowed for the Five Guess algorithm to find the solution.

b) private List<int[]> allPossibleCodes
   - List containing all the possible code combinations.

c) private Set<int[]> remainingCodes
   - Set containing the remaining code combinations that have not been tested yet.

d) private int[] currentGuess
   - The current guess of the algorithm

Methodes:

a) public FiveGuess(int codeLength, int numberOfPegs, int maxSteps)
   - Constructor for the FiveGuess class that initializes the parent Player class with codeLength, numberOfPegs, and a fixed name.
   - It also initializes maxSteps, allPossibleCodes, and remainingCodes.

   - @param codeLength Length of the secret code to be guessed.
   - @param numberOfPegs Number of possible pegs that can be used in the code.
   - @param maxSteps Maximum number of steps that the algorithm will take to find a solution.

b) private List<int[]> generateAllPossibleCodes()
   - It generates a list of all possible code combinations based on codeLength and numberOfPegs.

c) private void generateAllPossibleCodesHelper(int[] code, List<int[]> codes, int index)
   - It generates all possible code combinations recursively.

- @param code The Current code being constructed.
- @param codes List of codes being filled.
- @param index Current index in the code.

d) public List<List<Integer>> solve(List<Integer> solution) throws Exception
    - It Implements the solve method of the Maquina interface.
    - It takes a List of Integers as the secret code to break and returns a List of List of Integers representing the steps taken to find the solution.

    - @param solution The secret code that needs to be guessed.
    - @return List of codes guessed by the algorithm in order.
    - @throws Exception If the provided solution is not valid.

e) private boolean isValidSolution(List<Integer> solution)
    - Check if the solution is valid.

    - @param solution The solution to be checked.
    - @return True if the solution is valid, false otherwise.

f) public int[] generateSecretCode()
    - It overrides the abstract method from the parent Player class to generate a random secret code.

g) public int[] makeGuess()
    - It Overrides the abstract method from the parent Player class to make a guess based on the remaining possible codes.

h) private void reinitialize()
    - Reinitializes the values of the remaining possible solutions and the current guess once the solution is found.

i) private void minmax(MastermindBoard board)
    - Implements the MinMax part of the Five Guess algorithm.
    - chooses the guess that minimizes the maximum possible remaining codes in the worst case.

    - @param board Current game board.

4) Class GeneticComputerPlayer

This class implements a genetic algorithm to solve the Mastermind game. It inherits methods and attributes from class Player and implements the interface Maquina.

It generates an initial population, calculates the fitness of each individual, selects the best individuals, performs crossover and mutation, and iteratively repeats these steps until the solution is found or the maximum number of attempts is reached.

Attributes:

a) private int maxSteps
   It is the maximum number of attempts allowed to find the solution.
b) private int populationSize
   - It is the size of the population of possible solutions.
c) private double crossoverRate
   - It is the probability of crossover between two selected individuals.
d) private double mutationRate
   - It is the probability of mutation.
e) private final List<int[]> guessHistory
   - Tracks the previous made guesses so that it won't be repeated

Methodes:

a) public GeneticComputerPlayer(int codeLength, int numberOfPegs, int maxSteps, int populationSize, double crossoverRate, double mutationRate)

   - It is the constructor for the GeneticComputerPlayer class.

b) public int[] generateSecretCode()
   -It generates a random secret code.

c) public int[] makeGuess()
   - It returns a random code.

d) public List<List<Integer>> solve(List<Integer> solution)
   - It solves the Mastermind game using the genetic algorithm.

   - @param solution The secret code to be guessed.

e) private List<int[]> generateInitialPopulation()
   - It generates the initial population of random codes.

f) private int getFitness(int[] guess, int[] secretCode)
   - It calculates the fitness of a guess by comparing it to the secret code.

   - @parameter guess The guess to calculate the fitness for.
   - @param secretCode The secret code to compare against.
   - @return The fitness value of the guess.

g) private int[] getBestGuess(List<int[]> population, int[] secretCode)
   - It returns the best guess from the population based on the lowest fitness value.

   - @param population The current population.
   - @param secretCode The secret code to be guessed.
   - @return The best guess from the population.

h) private List<int[]> performSelection(List<int[]> population, int[] secretCode)
   - It selects the top 50% of the population based on fitness.

   - @param population The current population.
   - @param secretCode The secret code to be guessed.
   - @return The selected population.

i) private List<int[]> performCrossover(List<int[]> population)
   - It performs a crossover to generate offspring from selected individuals in the population.
   - @param population The current population.

j) private List<int[]> performMutation(List<int[]> population)
   - It randomly mutates the population.

   - @param population The current population.
   - @return The best guess from the population.

5) Class MastermindBoard

This class represents the state of a Mastermind game. It stores the secret codes, guesses and feedback for both the players.

Attributes:

a) private int codeLength
   -The length of the secret code

b) private int numberOfPegs
   -The number of different colored pegs available.

c) private int[] secretCodePlayer1
   - The secret code for Player 1.

d) private int[] secretCodePlayer2
   -The secret code for Player 2.

e) private List<int[]> guessesPlayer1
   - A list of guesses made by Player 1.

f) private List<int[]> guessesPlayer2
   - A list of guesses made by Player 2.

g) private List<int[]> feedbackPlayer1
   -A list of feedback given to Player 1.

h) private List<int[]> feedbackPlayer2
   -A list of feedback given to Player 2.

Methodes:

a) public MastermindBoard(int codeLength, int numberOfPegs)
- Constructor for the MastermindBoard class.

- @param codeLength is The length of the secret code.
- @param numberOfPegs The number of pegs used in the game.

b) public void setSecretCodes(int[] secretCodePlayer1, int[] secretCodePlayer2)
- Sets the secret codes for both players.

- @param secretCodePlayer1: The secret code for player 1.
- @param secretCodePlayer2 : The secret code for player 2.

c) public boolean isCorrectGuess(int[] guess, boolean forPlayer1)
- Checks if the given guess matches the secret code, either for player1 or player2.

- guess: The guess to check.
- forPlayer1: A boolean indicating whether the guess is for player 1.

d) public int[] getFeedbackForGuess(int[] guess, int[] secretCode)
- Returns feedback for a given guess compared to the secret code.
- Feedback consists of the number of pegs in the correct position and the number of pegs in the wrong position.

- @param guess The guess to check.
- @param secretCode The secret code to compare the guess to.
- @return An array with the number of pegs in the correct position and the number of pegs in the wrong position.

e) public int[] getFeedback(int[] guess, boolean forPlayer1)
- Provides feedback for a given guess, stores the guess and feedback in the appropriate lists,and returns the feedback. The feedback consists of the number of pegs in the correct position and the number of pegs in the wrong position.
- @param guess The guess to check.
- @param forPlayer1 A boolean indicating whether the guess is for player 1.

6) Class DifficultyLevel

This class represents a difficulty level for the game. Each difficulty level has a name, a number of guesses and an indicator if help is available.

Attributes:

a) private String name
   - Could be Easy, Medium Or Hard

b) private int numGuesses
   - The number of allowed guesses for the given difficulty level.

c) private boolean canGetHelp
   - A boolean indicating if the secret code should be shown for the given difficulty level.

Methodes:

a) public DifficultyLevel(String name, int numGuesses, boolean showCode)
   - Constructor for the DifficultyLevel class.
   - @param name The name of the difficulty level.
   - @param numGuesses The number of guesses allowed in this difficulty level.
   - @param canGetHelp A boolean indicating if help can be obtained in this difficulty level.

b) public int getNumGuesses()
   - Getter for the numGuesses attribute.

c) public boolean isCanGetHelp()
   - If true code will be shown for the given difficulty level

7) Class Assistance

This class helps to provide hints and feedback to the player based on their guesses and also current game state.

Attributes:

a) private MastermindBoard board
   -The current MastermindBoard instance for the game.

b) private Random random
   - To generate random numbers, a random object

c) private int index
   -The index used to track the guesses for assistance purpose

Methodes:

a) public Assistance(MastermindBoard board)
   - Constructor for the Assistance class, taking a MastermindBoard as a

parameter.
- @param board The board of the Mastermind game.

b) public String getRightPegsInRightPosition(boolean forPlayer1)
- It returns a string message indicating the number of pegs in the right position for the last guess.

- @param forPlayer1 Boolean to specify if feedback is for player 1.

c) public String getRightPegsInWrongPosition(boolean forPlayer1)
-It returns a string message indicating the number of pegs that are correct but in the wrong position for the last guess.

- @param forPlayer1 Boolean to specify if feedback is for player 1.

d) public String getNextCorrectPegAndPosition(boolean forPlayer1)
-Gives a hint to the player by revealing the position of the next peg in the secret code.
- @param forPlayer1 Boolean to specify if the hint is for player 1.

e) public void nextPeg()
- Keep track of the index to find the next peg for hint.

8) Class Ranking

This class represents the ranking system for the Mastermind game. It maintains a list of player scores and provides methods to update and retrieve rankings as well.

Attributes:

a) private List<ScoreEntry> scores
- A list of ScoreEntry objects representing player scores.

b) private static final int SIZE_RECORDS= 10
- Size of the list of leaderboard (Records)

Methods:

a) public Ranking()
- Constructor for the Ranking class, initializing the scores list.

b) public void updateRecords(String playerName, int score)
- Updates the records of ranking with the player's name and score

c) public List<ScoreEntry>getRankings()

- Returns the list of scores in the ranking.

d) private void addScore(String playerName, int score)
   - Adds a new ScoreEntry to the scores list and sorts the scores.
   - The score is added if it's either in the top 10 scores or the ranking has fewer than 10 scores.

   - @param playerName The name of the player.
   - @param score The score to add.

e) public List<ScoreEntry> getTopScores(int numberOfTopScores)
   - Returns a list of the top N scores, where N is specified by the numberOfTopScores parameter.

   - @param numberOfTopScores Number of top scores to return.

f) private void sortScores()
   - Sorts the scores list in descending order by score.

9) Static Class ScoreEntry

This class is an inner class within the Ranking class that represents an individual score entry for a player in the game. It saves the player's name and the score achieved by that player in a single game.

Attributes:

a) private String playerName
   - The name of the player.

b) private int score
   - The score achieved by the player.

Methodes:

a) public ScoreEntry(String playerName, int score)
   - Constructor for the ScoreEntry class, taking a player name and score as parameters.

   - @param playerName Name of the player.
   - @param score Score of the player

b) public String getPlayerName()
   - Returns the name of the player.

c) public int getScore()

- Returns the player's score.

d) public void setScore(int score)
- Sets the score for this score entry.

- @param score Score to set

e) public int compareTo(ScoreEntry other)
- Compare this score entry with another score entry based on the score value.

- @param other The other score entry to compare to.

f) public void updatePlayerScore(String playerName, int score)
- Updates the player's score in the ranking system.
- If the player is found in the ranking, it updates their score.
- If the player is not found, it creates a new entry for the player.

- @param playerName Name of the player.
- @param score New score of the player.

10) Serializable class GameSave

This class is a serializable class which is used to save and load the instance of the game to resume play.

Attributes:

a) private static final long serialVersionUID
-Necessary to deserialize same class that was serialized

b) private final int codeLength
- game of code length

c) private final int numberOfPegs
- game of number of pegs

d) private final int[] secretCodePlayer1
- Secret code of player 1

e) private final int[] secretCodePlayer2
- Secret code of player 2

f) private final List<int[]> guessesPlayer1
- Guesses made by player 1

g) private final List<int[]> guessesPlayer2

- Guesses Made by player 2

h) private final List<int[]> feedbackPlayer1
   - Feedbacks obtained by player1

i) private final List<int[]> feedbackPlayer2
   - Feedbacks obtained by player2

j) private final String player1Name
   - Name of player 1

k) private final String player2Name
   - Name of player 2

l) private final int scorePlayer1
   - Score of player 1

m) private final int scorePlayer2
   - Score of player 2

n) private String algorithmUsed
   - algorithm used, either genetic or five guess

o) private final String difficultyLevel1
   - to store the difficulty level of the game

p) private final int totalRounds
   - to store total number of rounds and rounds played

q) private final int roundsPlayed
   - to store the total rounds Played

Methodes:

a) Methods of this Serializable class contain all the getters for the above attributes of the class.

11) CtrlDomini

This is the controller class responsible for managing the game. It interacts with the persistence layer and presentation layer. All the necessary data that we need to send to the presentation layer are sent via this class in the form of Strings.

Attributes:

a) private static CtrlDomini instance

- instance of this class

b) private final CtrlPersistencia cp
- For loading and saving of game instance and records

c) private MastermindBoard board
- Mastermind game board

d) private int codeLength
- game with code length

e) private int numberOfPegs
- game with number of pegs

f) private Player player1
- Player in the game (Human)

g) private Player player2
- Player in the game (AI)

h) private Maquina AIPlayer
- instance of the maquina interface

i) private DifficultyLevel difficultyLevel
- difficulty level of the game

j) private final Assistance assistance
- assistance for providing hints during the game

k) private Ranking ranking
- ranking system for the game, also useful to pull records

l) private String currentAlgorithm
- Algorithm used by the AI for the game

m) private int maxSteps
- Maximum attempts allowed in the game

n) private int totalRounds
- Total rounds in a game

o) private int currentRound
- current round in the game

p) private int scorePlayer2
-Score of player2

q) private int scorePlayer1
-Score of player1

Methodes:

a) public CtrlDomini()
   - Constructor for the CtrlDomini class
   - Initializes the game with a certain codeLength and numberOfPegs
   - Loads ranking from a persistent file (Records)
   - Initializes players with the specified game configurations

b) public String getPlayer1name()
   - Get the name of player1.

c) public String getPlayer2name()
   -Get the name of player2.

d) public String getCodeLength()
   - the codeLength used in the game.

e) public String getNumberOfPegs()
   -the number of pegs used in the game.

f) public int[] getSecretCodePlayer1()
   -Player1's Secret code

g) public String getSecretCodePlayer2()
   - PLayer2's Secret code

h) public String getGuessesPlayer1()
   - Guesses made by player 1

i) public List<int[]> getGuessesPlayer2()
   - Guesses made by player2

j) public String getFeedbackPlayer1()
   -Feedback obtained by player1

k) public List<int[]> getFeedbackPlayer2()
   -Feedback obtained by player2

l) public String getDifficultyLevel1()
   -Returns the difficulty level of the game.

m) public String getNumGuesses()
   -Returns the maximum number of guesses allowed

n) public String getCurrentAlgorithm()
   -Returns the current Algorithm used by the AI player

o) public StringgetFeedback(int[] guessPlayer1, boolean b)
- Feedbacks based on the guesses made by the player

p) public String getRightPegsInRightPosition(boolean forPlayer1)
- Provides feedback about the number of pegs placed in the right position
- @param forPlayer1 Boolean to specify if feedback is for player 1.

q) public String getRightPegsInWrongPosition(boolean forPlayer1)
- Provides feedback about the number of correct pegs placed in the wrong position

r) public String getRandomCorrectPegAndPosition(boolean forPlayer1)
- Gives a hint to the player by revealing the position of a random peg in the secret code.

s) public String getScorePlayer1()
-Returns the score of player1

t) public String getScorePlayer2()
- Returns the score of player2

u) public List<String> getTopRecords()
- Top 10 scores for the records functionality

v) public String getTotalRounds()
-Gets the total number of rounds set by the player in the current round

w) public String getCurrentRound()
- the current round number in the game

x) public int[] SetStringToArray(String str)
- Converts a string representation of an integer array into an integer array.

- @param str The string to be converted.
- @return An integer array.

y) public void createHumanPlayer(String name)
- Creates a Human player
- @param name the name of the human player.

z) public Player createAIPlayer(String type)
- Create an AI player of a given type.
- @param type the type of AI player, either "genetic" or "fiveGuess".
- @return a Player object representing the AI player.

aa) public void updatePlayerScore(String playerName, int score)
- Update the score for a player.

bb) public boolean isCorrectGuess(int[] guessPlayer1, boolean b)
- If the guess made by the player is correct or not
- @param guessPlayer1 guesses made by player1.
- @param b if the guess is made by player1 or player2, true if player1.
- @return boolean value, true if guess is correct, false otherwise

cc) public void selectAlgorithm(String algorithm)
- Allows us to choose between two algorithm, be it fiveGuess or Genetic.
- @param algorithm the algorithm selected.

dd) public List<List<Integer>> solveUsingAlgorithm(List<Integer> secretCode)
- Use the AI player to solve the game after selecting the algorithm
- Select algorithm should be called before using solveUsingAlgorithm.

- @param secretCode The code to be solved by the AI player.
- @return A list of lists of integers, where each list represents a guess by the AI player.
- @throws Exception If any errors occur during solving.

ee) public void selectDifficultyLevel(String difficulty)
- Sets the difficulty level of the game.

- @param difficulty A string representing the desired difficulty level.

ff) public boolean playerNotNull()
- Checks if the player is not null.

- @return true if the player is not null, false otherwise.

gg) public int[] generateSecretCodePlayer2()
- Generates the secret code for player 2.
- @return An integer array representing the secret code.

hh) public void updateRecords(String playerName, int score)
- Updates the records with the given player name and score.

ii) public String getInfo()
- Provides basic information about the game.
- @return A string containing the game information.

jj) public void resetGameSameSettings()
- Resets the game with the same settings.

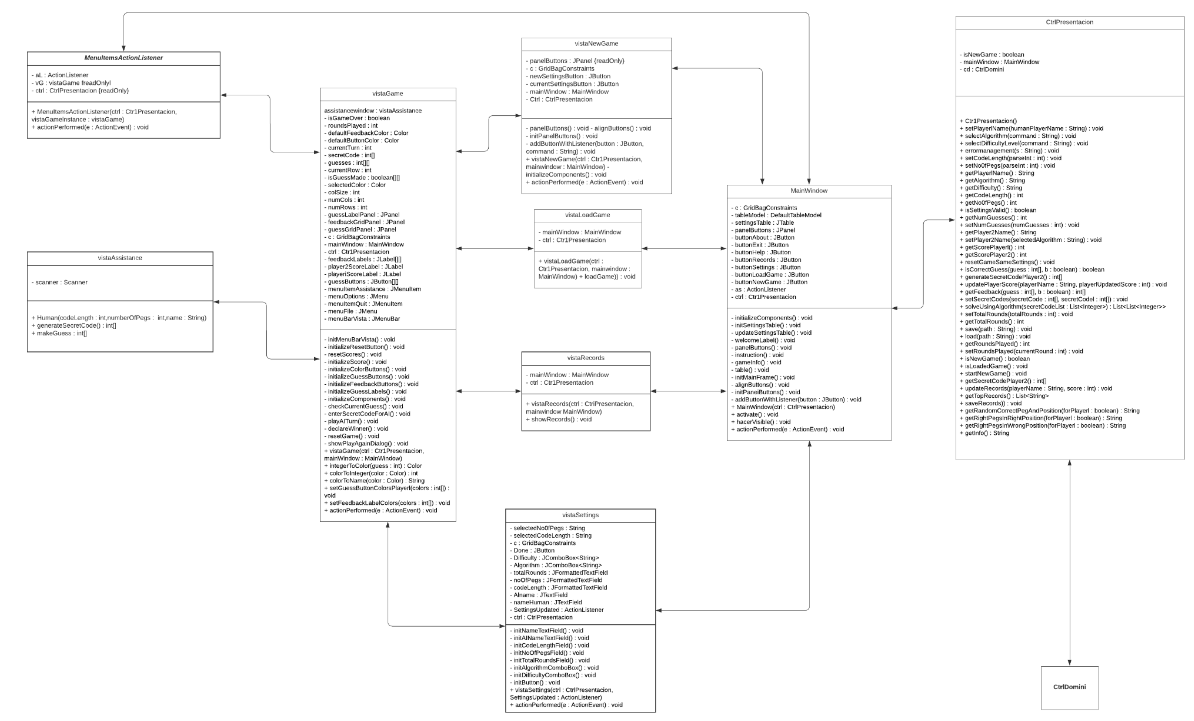
kk) public void save(String path)
- Saves the current state of the game to the specified path.
- @param path The path where the game state should be saved.

ll) public void saveRecordsToFile()
  - Saves the records to a file.

mm) public void load(String path)
  -Loads a game state from the specified path.
  - @param path The path where the game state is stored.

Capa de Presentacion

This layer simply handles the interface for the game. It communicates with CtrlDomini for necessary elements it needs for the interface to work properly. Below is the class diagram and description of the presentation layer and all it’s related classes:



1) Class MainWindow

This is the vista principle of the application. Whenever we run the application we will see the view designed here. It represents a mainWindow with several actions to perform such as:

starting a game, loading a game, viewing records, managing settings, learning about the application, and knowing the rules of the game.

Attributes:

a) private final CtrlPresentacion ctrl
   - instance of the CtrlPresentacion class

b) private ActionListener as

c) private final JButton buttonNewGame
   - initialization of New Game button

d) private final JButton buttonLoadGame
   - initialization of LoadGame button

e) private final JButton buttonSettings
   - initialization of Settings button

f) private final JButton buttonRecords
   - initialization of records button

g) private final JButton buttonHelp
   - initialization of help button

h) private final JButton buttonExit
   - initialization of Exit button

i) private final JButton buttonAbout
   - initialization of About button

j) private final JPanel panelButtons
   - initialization of buttons panel

k) private JTable settingsTable
   - initialization of settings table

l) private DefaultTableModel tableModel

m) private final GridBagConstraints c
   - initialization of GridBagConstraints

n) private Dimension resolution
   -To maintain a perfect window size across devices

Methodes:

a) public MainWindow(CtrlPresentacion ctrl)
   - Constructor for the MainWindow class.

b) public void activate()
   - Activate the mainWindow, make it visible.

c) public void hacerVisible()
- Adjust dimensions, make it visible

d) private void initializeComponents()
- Initializes all the components we have in the main window

e) private void initSettingsTable()
- To initialize the table where we can view current settings of the game.

f) private void updateSettingsTable()
- To update the settings table with the settings we configured

g) private void welcomeLabel()
- Initializes the welcome label

h) private void panelButtons()
- Initializes the panel buttons

i) private void instruction()
- A small instruction info shown on mainWindow for the users

j) private void gameInfo()
- Name of the table of Settings in the mainWindow

k) private void table()
- The table where we update the settings info.

l) private void initMainFrame()
- Sets the dimension of the mainWindow,allows resizable,sets layout calls various init functions to initialize the main window such as panelButtons(), table(), welcomeLabel(),instruction().

m) private void alignButtons()
- To align the buttons in the center of the mainWindow and rearranging them a bit down the y-axis

n) private void initPanelButtons()
- Initialization of All the buttons available in the mainWindow

o) public void actionPerformed(ActionEvent e)
- handles the events performed from the buttons of the mainWindow

p) private void addButtonWithListener(JButton button)
- Add a button to the button panel and sets up an ActionListener for the button

2) Class vistaSettings

This class represents the GUI for the settings view in the game.

Attributes:

a) private CtrlPresentacion ctrl;

b) private ActionListener SettingsUpdated;

c) private JTextField nameHuman;

d) private JTextField AIname;

e) private JFormattedTextField codeLength;

f) private JFormattedTextField noOfPegs;

g) private JFormattedTextField totalRounds;

h) private JComboBox<String> Algorithm;

i) private JComboBox<String> Difficulty;

j) private JButton Done;

k) private GridBagConstraints c

l) String selectedCodeLength

m) String selectedNoOfPegs

Methodes:

a) public vistaSettings(CtrlPresentacion ctrl, ActionListener SettingsUpdated)
- Constructor of the vistaSettings initiates the settings view by initializing all UI components and their action listeners.
-@param ctrl - the controller object for the settings view to interact with.
- @param SettingsUpdated - action listener to handle actions when settings are updated.

b) private void initNameTextField()
- initializes the text field for the human player name.

c) private void initAINameTextField()
- initializes the text field for the AI player name.

d) private void initCodeLengthField()
- initializes the text field for the code length.

e) private void initNoOfPegsField()
- initializes the text field for the number of pegs.

f) private void initTotalRoundsField()

- initializes the text field for the total rounds of play.

g) private void initAlgorithmComboBox()
   - initializes the combo box for selecting the AI's algorithm.

h) private void initDifficultyComboBox()
   - initializes the combo box for selecting the game difficulty level.

i) private void initButton()
   - initializes the done button. It verifies and validates the inputs and passes it to the controller.

3) class vistaGame

This class implements the view of the game in the GUI. For example either we start a new game or we load a game, we will eventually enter the vistaGame view. Below are the attributes and methods for vistaGame.

Attributes:

a) private final JMenuBar menuBarVista
   - New JmenuBar initialization

b) private final JMenu menuFile
   - Jmenu initialization

c) private final JMenuItem menuItemQuit
   - Initialization of menu item quit

d) private final JMenu menuOptions
   - initialization of menuOptions

e) private final JMenuItem menuItemAssistance
   - initialization of menuitem assistance

f) private static JButton[][] guessButtons
   - buttons for guesses which is a 2D array

g) private JLabel player1ScoreLabel
   - Label for player1 score

h) private JLabel player2ScoreLabel
   - Label for player2 score

i) private static JLabel[][] feedbackLabels;

- feedback labels which is a 2D array

j) private final CtrlPresentacion ctrl
- instance of the ctrl presentacion

k) private MainWindow mainWindow
- instance of the mainWindow

l) private final GridBagConstraints c
- initialization of gridBagConstraints

m) private JPanel guessGridPanel

n) private JPanel feedbackGridPanel

o) private JPanel guessLabelPanel

p) private static int numRows
- to check the rows clicked

q) private static int numCols
- to check the buttons clicked

r) private final int colSize
- size of the column which is equal to the no. of pegs

s) private Color selectedColor

t) private boolean[][] isGuessMade
- boolean value to check if the button is initialized

u) private int currentRow

v) private int[][] guesses
- the guesses made by the players

w) private int[] secretCode
- the secretCode for each turn

x) private int currentTurn
- Turn 1 is of Human Player

y) Color defaultButtonColor
- Default color for guess button

z) Color defaultFeedbackColor
- Default color for feedback label

aa) private int roundsPlayed
- no of rounds played until now

bb) private boolean isGameOver
- game is over when roundsPlayed equal total rounds

cc) private vistaAssistance assistanceWindow
- instance of vistaAssistance class

dd) private int defaultButtonSize

ee) private Dimenson window

ff) private vistaSecretCode secretCodeWindow

Methodes:

a) public vistaGame(CtrlPresentacion ctrl, MainWindow mainWindow)
- Constructor for vistaGame.

b) private void initMenuBarVista()
- Initialization of the menu bar for the game.

c) public static Color integerToColor(int guess)
- Convert an integer to a corresponding Color.

d) public static int colorToInteger(Color color)
- Convert a Color to a corresponding integer.

e) private String colorToName(Color color)
- Convert a Color to a corresponding color name in string.

f) private void initializeResetButton()
- Initialization of the reset button.

g) private void resetScores()
- Reset the scores for the players.

h) private void initializeScore()
- Initialization of the score labels.

i) private void initializeColorButtons()
- Initialization of the color buttons

j) private void initializeGuessButtons()
- Initialization of guess buttons

k) private void initializeFeedbackButtons()
- Initialization of feedback labels

l) private void initializeGuessLabels()
- Initialization of Guess labels

m) private void initializeComponents()
- Initialization of all the components of the game

n) private void checkCurrentGuess()
- To check the current Guess made by the human player

o) private void enterSecretCodeForAI()

- JOptionPane for the input dialog to set the secretCode for AI to guess

p) private void playAITurn()
- Guesses made by the AI player

q) private void declareWinner()
- If the game is over, compares the scores and declares the winner

r) private void resetGame()
- Resets the game completely when a RESTART button is pressed

s) private void showPlayAgainDialog()
- Asks for continuity or not when the game is over

t) public static void setGuessButtonColorsPlayer1(int[][] colors)
- Getters and setters to save colors of buttons to load later while loading

u) public static void setFeedbackLabelColors(int[][] colors)

4) class vistaRecords

This class represents the GUI for the records view in the game.

Attributes:

a) private CtrlPresentacion ctrl;
- Instance of the CtrlPresentacion class

b) private MainWindow mainWindow;
- Instance of the MainWindow class

Methodes:

a) public vistaRecords(CtrlPresentacion ctrl, MainWindow mainWindow)

b) public void showRecords()
- It shows the top 10 records of the game. The records should be a table with 3 fields.. Player name, Score, Ranks

5) class vistaNewGame

This class represents the GUI for the new game view in the game.

Attributes:

a) private CtrlPresentacion ctrl;
- Instance of the CtrlPresentacion class

b) private MainWindow mainWindow;
- Instance of the MainWindow class

c) private JButton currentSettingsButton
- Initialization of current settings button

d) private JButton newSettingsButton
- Initialization of new settings button

e) private GridBagConstraints c
- Initialization of GridBagConstraints

f) private final JPanel panelButtons
- Initialization of panel buttons

Methodes:

a) public vistaNewGame(CtrlPresentacion ctrl, MainWindow mainWindow)
- Constructor for the New Game view. Sets the control and main window.

b) private void initializeComponents()
- Initializes all the components for the New Game view.Ç

c) private void panelButtons()
- Initializes the panel for the buttons.

d) private void alignButtons()
- Aligns the buttons in the center.

e) private void initPanelButtons()
- Initializes the buttons for the panel and adds them to the panel.

f) private void addButtonWithListener(JButton button,String command)
- Adds a button with an action listener to the panel.

g) public void actionPerformed(ActionEvent e)
- Handles the action events generated by the buttons.

6) class vistaLoadGame

This class represents the GUI for the load game view in the game.

Attributes:

a) private CtrlPresentacion ctrl
   - Instance of the CtrlPresentacion class

b) private MainWindow mainWindow;
   - Instance of the MainWindow class

Methodes:

a) public vistaLoadGame(CtrlPresentacion ctrl, MainWindow mainWindow)
   - Constructor for the Load Game view. Set the control and main window.

b) public void loadGame()
   - It Loads a game from a saved file. If the game is loaded successfully, the game window is opened and the main window is disposed of. If the game fails to load, an error message is shown.

7) class vistaAssistance

This class represents the GUI for the assistance view in the game.

Attributes:

a) private CtrlPresentacion ctrl
   - Instance of the CtrlPresentacion class

Methodes:

a) public vistaAssistance(CtrlPresentacion ctrl)
   - Constructor for the Assistance view.

b) private void initWindow()
   - Initializes window settings: layout, size, location and visibility.

c) private void initButtons()
   - Initializes the buttons on the window. Displays the secret code as a set of buttons with colors corresponding to the code. And we highlight one random button with the correct color.

d) public static Color integerToColor(int guess)
   - Converts an integer to a corresponding color.

8) class MenuItemsActionListener

This class represents the action listener for the menu items in the game.

Attributes:

a) private CtrlPresentacion ctrl
   - Instance of the CtrlPresentacion class

b) private final vistaGame vG
   - Instance of the vistaGame class

c) private ActionListener aL
   - Instance of the ActionListener class

Methodes:

a) public MenuItemsActionListener(CtrlPresentacion ctrl, vistaGame vistaGameInstance)
   - Constructor for the MenuItemsActionListener.

b) public void actionPerformed(ActionEvent e)

9)Class CtrlPresentacion

This is the controller class of the presentation layer. This class is responsible for communicating with CtrlDomini which is the controller of the Domain layer.

Attributes:

a) private CtrlDomini cd
   - Instance of the CtrlDomini class

c) private MainWindow mainWindow
   - Instance of the vista MainWindow class

d) Private boolean isNewGame
   - boolean that indicates if the game is new or loaded

Methodes:

a) public CtrlPresentacion()
   - Constructor for the CtrlPresentacion class. Initializes the controller for the game.

b) public void setPlayer1Name(String humanPlayerName)
   - Sets Player 1's name.

- @param name the new name for player1.

c) public void setPlayer2Name(String humanPlayerName)
- Sets Player 2's name.

- @param name the new name for player2.

d) public void selectAlgorithm(String command)
- Selects the algorithm for the game.

e) public void selectDifficultyLevel(String command)
-Selects the difficulty level for the game.

f) public void setCodeLength(int parseInt)
- Sets the length of the code for the game.

- @param codeLength The length of the secretCode.

g) public void setNumberOfPegs(int parseInt)
- Sets the number of pegs for the game and initializes a new board.

- @param numberOfPegs The number of different colors available.

h) public String getPlayer1Name()
-Gets Player 1's name.

i) public String getAlgorithm()
- Gets the current algorithm used in the game.

j) public String getDifficulty()
-Gets the current difficulty level of the game.

k) public int getCodeLength()
-Gets the length of the code in the game.

l) public int getNoOfPegs()
- Gets the number of pegs in the game.

m) public boolean isSettingsValid()
- Checks if the game settings are valid.

n) public int getNumGuesses()
- Gets the number of guesses made in the game.

o) public int getScorePlayer1()
- Gets Player 1's score.

p) public int getScorePlayer2()

- Gets Player 2's score.

q) public int[] generateSecretCodePlayer2()
- Generates a secret code for Player 2.

r) public void updatePlayerScore(String player1Name, int player1UpdatedScore)
- Updates Player 1's score.

s) public int[] getFeedback(int[] guess, boolean b)
- Provides feedback for the provided guess.

t) public void setSecretCodes(int[] secretCode, int[] secretCode1)
- Sets the secret codes for the game.

u) public List<List<Integer>> solveUsingAlgorithm(List<Integer> secretCodeList)
- Solves the game using the specified algorithm.

v) public void setTotalRounds(int totalRounds)
- Sets the total number of rounds for the game.

- @param totalRounds The total number of rounds to be played.

w) public int getTotalRounds()
-Gets the total number of rounds set for the game.

x) public void save(String path)
-Saves the current state of the game.

y) public void load(String path)
-Loads a saved game state.

z) public void setRoundsPlayed(int currentRound)
-Sets the current round of the game.

- @param roundsPlayed The number of rounds played.

aa) public boolean isNewGame()
-Checks if it's a new game.

bb) public boolean isNewGame()
-Checks if it's a new game.

cc) public void isLoadedGame()
- Marks the game as a loaded game.

dd) public void startNewGame()
-Marks the game as a new game.

ee) public int[] getSecretCodePlayer2()
- Gets the secret code of Player 2.

ff) public void updateRecords(String playerName, int score)
- Updates the records of the game.

gg) public List<String> getTopRecords()
- Gets the top records of the game.

hh) public void saveRecords()
- Saves the records to a file.

ii) public String getRandomCorrectPegAndPosition(boolean forPlayer1)
- Gets a random correct peg and its position.

jj) public String getRightPegsInRightPosition(boolean forPlayer1)
- Gets the right pegs in the right position.

kk) public String getRightPegsInWrongPosition(boolean forPlayer1)
- Gets the right pegs in the wrong position.

ll) public String getInfo()
- Gets information about the game.

mm) getNextCorrectPegAndPosition()
- Gives a hint to the player by revealing the position of a peg in the secret code.

- @param forPlayer1 Boolean to specify if the hint is for player 1.
- @return String hint about a peg and its position in the secret code.

nn) public String getFeedback(int[] guessPlayer1, boolean b)
- Feedbacks based on the guesses made by the player.

- @param guessPlayer1 The guess made by player1.
- @param b boolean to represent if the feedback is for player1 or player2
- @return An array of integers representing the feedback.

oo) public void setCurrentAlgorithm(String currentAlgorithm)
- Sets the algorithm that the game will use to solve the secretCode.
- @param currentAlgorithm The name of the algorithm to be used.

pp) public void setDifficultyLevel1(String difficultyLevelName)
- Sets the difficulty level of the game.
- @param difficultyLevelName The name of the difficulty level.

qq) public void setNumGuesses(int numGuesses)

- Sets the number of guesses allowed in the game.
- @param numGuesses The maximum number of guesses allowed.

rr) public void setCurrentRound(int currentRound)
- Sets the current round number.
- @param currentRound The round number to be set.

ss) public void setGuessesPlayer1(List<int[]> guessesPlayer1)
- Sets the guesses made by player 1.
- @param guessesPlayer1 list of integer arrays, each representing a guess by player 1.

tt) public void setGuessesPlayer2(List<int[]> guessesPlayer2)
- Sets the guesses made by player 2.
- @param guessesPlayer2 list of integer arrays, each representing a guess by player 2.

uu) public void setFeedbackPlayer1(List<int[]> feedbackPlayer1)
- Sets the feedback for player 1's guesses.
- @param feedbackPlayer1 list of integer arrays, each representing the feedback for a guess by player 1.

vv) public void setFeedbackPlayer2(List<int[]> feedbackPlayer2)
- Sets the feedback for player 2's guesses.
- @param feedbackPlayer1 list of integer arrays, each representing the feedback for a guess by player 2.

ww) public void setScorePlayer1(int scorePlayer1)
- Sets the score of player 1.
- @param scorePlayer1 The score to be set for player 1.

xx) public void setScorePlayer2(int scorePlayer2)
- Sets the score of player 2.
- @param scorePlayer2 The score to be set for player 2.

yy) private List<int[]> SetStringToListOfArrays(String s)
- Converts a string representation of a list of integer arrays into a list of integer arrays.
- @param s The string to be converted.
- @return A list of integer arrays.

zz) public int[] SetStringToArray(String str)
- Converts a string representation of an integer array into an integer array.
- @param str The string to be converted.
- @return An integer array.

aaa) public String generateSecretCodePlayer2()
- Generates the secret code for player 2.
- @return An integer array representing the secret code.

10) vistaSecretCode

Attributes:

a) private CtrlPresentacion ctrl
b) private boolean[] painted
c) private Color selectedColor
d) private JPanel guessGridPanel
e) private JPanel colorButtonWrapperPanel
f) private JPanel submitButtonPanel
g) private boolean isCodeMade
h) private int[] SecretCode
i) private int numColors
j) private int CodeLength

Methodes:

a) public vistaSecretCode(CtrlPresentacion cp)
- Constructor of the class

b) private void initWindow()
- Initialization of the vista window

c) private void initColorsButtons()
- Initialization of the color buttons for the assistance.

d) void initSubmitButton()
- Initialization of the submit Button

e) private void initSecretCodeButtons()
- Initialization of the Secret code buttons

f) void initComponents()

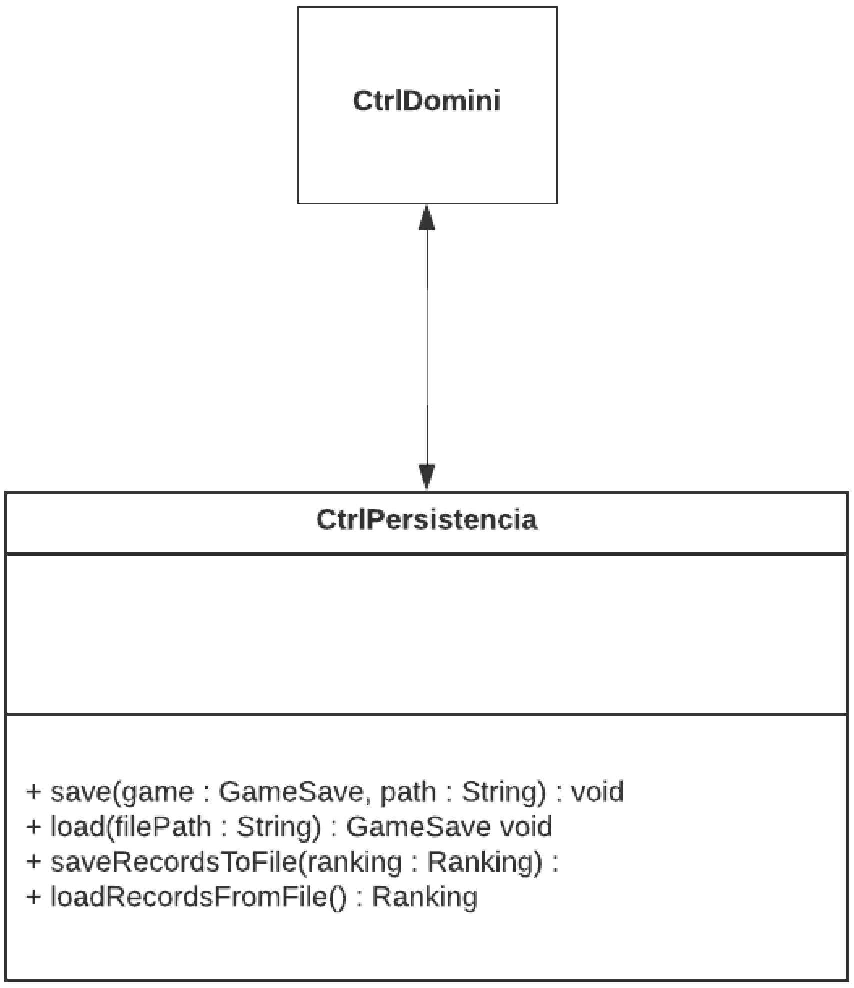
- Initialization of all the components necessary

g) private void checkCode()
- Checks if the secrecode provided as a hint has been completed or not

h) Public String getSecretCode()
- To get the current secret code to show.

Capa de Persistencia

This is the layer of the 3 tier architecture that is responsible for the handling of data i.e loading and saving of game states and files related to records. As the actual work of this layer is to save and load the state of the game and a file that handles the leaderboard we have included all these 4 functions in a single class such that there is no coupling of codes. Below is the Class diagram of the Persistence layer and its description.


CtrlDomini
CtrlPersistencia
+ save(game : GameSave, path : String) : void
+ load(filePath : String) : GameSave void
+ saveRecordsToFile(ranking : Ranking) :
+ loadRecordsFromFile() : Ranking

Class CtrlPersistencia

This class handles the loading and saving of game state and a records file that maintains the leaderboard.

Attributes: a) Class attributes none.
Methods: a) public static void save(GameSave game, String path)

- Saves a game state to a file at the given path.
- game: The GameSave object representing the state of the game to be saved.
- path The path to the file where the game state will be saved.

- @param game The GameSave object representing the state of the game to be saved.
- @param path The path to the file where the game state will be saved.

b) public static GameSave load(String filePath)
- Loads a game state from a file at the given path.
- filePath: The path to the file where the game state is stored.
- The GameSave object representing the loaded game state.

- @return The GameSave object representing the loaded game state.

c) public void saveRecordsToFile(Ranking ranking)
- Saves the game's ranking to a text file.

- @param ranking The Ranking object to be saved to file.

d) public Ranking loadRecordsFromFile()
- Loads the game's ranking from a text file.
- The Ranking object representing the loaded rankings.