N-in-a-Row – The Console Application

Engine Module

**Interface GameLogic:**

* boolean play(int col)
* int playerTurns(int player)
* String timeFromBegining()
* int getCols()
* void loadSettingsFile(String filePath) throws Exception
* char[][] boardReadyToPrint()
* void initPlayer(PlayersTypes playerType, int id, String name)
* PlayersTypes getTypeOfCurrentPlayer()
* boolean undoLastMove()
* int getNumberOfInitializedPlayers()
* int getIdOfCurrentPlayer()
* List<Move> getMovesHistory()
* boolean getHasWinner()
* boolean getIsBoardFull()
* void setBoardFromSettings(boolean restartPlayers)
* int getSequenceLength()
* int getNumberOfPlayersToInitialized()

**Class Game** implements GameLogic:

* final int maxNumberOfPlayers
* boolean hasWinner, isBoardFull
* Board board
* Date startingTime
* GameSettings gameSettings
* List<Player> players
* Player currentPlayer
* List<Move> playedMoves

**Enum class Directions**: LEFT, LEFTUP, UP, UPRIGHT, RIGHT, RIGHTDOWN, DOWN, LEFTDOWN

**Class Player**:

* int numOfTurnsPlayed, id
* PlayerTypes playerType
* String name

**Class Board**:

* int rows, cols, winningPlayer, emptySapces
* Col[] board
* boolean hasWinner

**Class GameSettings**:

* int target, boardNumRows, boardNumCols, numOfPlayers
* GameVariant gameVariant
* GameType gameType
* String settingFilesPath

**Class Move**:

* Static int movesCount
* int moveIndex, playerID, col

**Class Col**:

* int colNumber, freeSpaces, lastRowInserted
* Disc[] discs

**Class Disc**:

* Position position
* int discOfPlayer
* Map<Directions, Disc> discsArround

**Enum class MenuChoice**: LOADXML, RESTARTGAME, EXIT, MAKETURN, HISTORY, UNDO, SAVEGAME, LOADGAME, INVALIDCHOICE

**class UI**:

* String xmlPath, savedGamePath
* PrintMessages winningMessage, endGame
* Menu primaryMenu\_noRestart, primaryMenu\_wRestart , gameMenu
* **GameLogic** gameLogic
* boolean isValidXML

Common Module

**Enum class PlayersTypes**: HUMAN, ROBOT

**Enum class GameVariant**: REGULAR, CIRCULAR, POPOUT

**Enum class GameType**: BASIC, MULTIPLAYER, DYNAMIC\_MULTIPLAYER

UI Module

**Interface PrintMessage**:

* Void printMessage(int player)

**Interface Menu**:

* MenuChoice showMenu()

**General structure**

* The game has 2 main modules: Engine, UI.
* The game has 1 common module: Common. It's being used by both the Engine module and UI module.
* The central class in the engine module is Game, which implements the GameLogic interface:
  + It holds information about the players, the board (and its state) and the moves that were made in the game.
  + It is responsible for enforcing and validating the rules of the game.
  + The Board consists of a set of columns, each of which contains Discs, in number matching the number of rows.
  + Each Disc has its own position in the board (represented as a matrix) and may or may not have a player’s ID (depends on whether a player dropped a disc that ended up in the Disc’s position). ID 0 represents a disc that does not belong to any player.
* The central class in the UI module is UI, and it includes the the PSVM function. UI class holds an instance of a GameLogic object
  + This class interacts with the user via the console.
  + Interaction consists of printing the game board and messages with relevant information and displaying menus that allow the user to perform different choices regarding the game.
* The Common module includes ENUM classes with static values. Both UI module and Engine module should be familiar with the Common module.

**Main Choices that were made**

* The flow of the game is managed by a set of menus, as opposed to a single constant menu which accompanies the entire game – the menus offer the user options that are relevant to the specific point the game flow.
* When a round of game is completed, either by a win or a tie, the user can choose between (1) starting a new game, by loading an XML file with different configuration and defining new players, (2) loading a game that was previously saved to file and continue it from the point of the save, (3) restarting the most recent game that was played, whether it was a new game from the fresh XML configuration or a loaded game.
* At any point of the game, the user can choose to exit the game all together.
* Besides implementing the bonus that allows the user to continue playing after a game is finished (playing additional games), by restarting a game or load a game that was saved to a file in some point in the past (detailed above), we also implemented the UNDO functionality and the options to save/load games (mentioned above).
* Games that are saved to files are saved to the directory from where the game is run, in the following format: N-in-a-Row\_<timestamp>.nar
* When a game is in progress, any time that a message it shown to user regarding the latest action selected (Ex. no moves available for UNDO, no moves history to show etc.), the main thread is in Sleep mode for a defined duration (long enough for the user to get the relevant information).

**Students’ details**

* Alon Boxiner (203485248), [alonboxiner@gmail.com](mailto:alonboxiner@gmail.com)
* Miri Kogan (066719279), [mirikog@gmail.com](mailto:mirikog@gmail.com)