N-in-a-Row – The Web Application

Engine Module

**Interface GameLogic:**

* boolean play(int col, boolean popout);
* String timeFromBegining();
* int getCols();
* int getRows();
* PlayerTypes getTypeOfCurrentPlayer();
* int getIdOfCurrentPlayer();
* boolean getHasWinner();
* Set<Integer> getWinners();
* boolean getIsBoardFull();
* void setRoundFromSettings(boolean restartPlayers);
* int getNumberOfPlayersToInitialized();
* boolean isPopout();
* List<Player> getPlayers();
* Move getLastMove();
* void resignPlayer();
* GameVariant getGameVariant();
* int getNumberOfRoundsPlayed();

void increaseRoundPlayed();

**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 PlayersTypes**: HUMAN, ROBOT

**Enum class GameType**: BASIC, MULTIPLAYER, DYNAMIC\_MULTIPLAYERboolean play(int col, boolean popout);

String timeFromBegining();

int getCols();

int getRows();

PlayerTypes getTypeOfCurrentPlayer();

int getIdOfCurrentPlayer();

boolean getHasWinner();

Set<Integer> getWinners();

boolean getIsBoardFull();

void setRoundFromSettings(boolean restartPlayers);

int getNumberOfPlayersToInitialized();

boolean isPopout();

List<Player> getPlayers();

Move getLastMove();

void resignPlayer();

GameVariant getGameVariant();

int getNumberOfRoundsPlayed();

void increaseRoundPlayed();

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

Common Module

**Class** **GameSettings** + **Class** **SettingsFileException**

**Class** **Lock**

**Class** **PlayerSettings**

**Enum class** **MoveType**: INSERT, POPOUT

**Class SingleChatEntry**:

* String chatString
* String username
* long time

**Enum class GameStatus**: PLAYING, PENDING\_PLAYERS

**Class SingleGameEntry**:

* String userName
* int gameId
* GameLogic gameLogic
* String gameName
* GameStatus gameStatus
* List<UserSettings> viewers
* boolean hasWinner
* boolean isTie

WebApplication\_LogicEngine Module

**Class UserManager**:

* HashMap<String, UserSettings> usersMap

**Class ChatManager**:

* Map<Integer, SingleChatManager> chats

**Class SingleChatManager**:

* List<UserSettings> chatUsers
* List<SingleChatEntry> chatDataList

**Class GameListManager**:

* static int gamesCount = 0
* Map<Integer, SingleGameEntry> gameEntriesMap
* GameFactory gameFactory

**Class ViewerResignServlet**

**Class RegisterViewerToGameServlet**

**Class UsersListServlet**

**Class GameDataServlet**

**Class ServletResponse**:

* Boolean result
* String msg

**Class SessionUtils**

**Class RegisterPlayerToGameServlet**

**Class GamesListServlet**

**Class ServletUtils**

**Class SendChatServlet**

**Class PlayMoveServlet**

**Class GetChatServlet**

**Class PlayerResignServlet**

**Class LogoutServlet**

**Class UploadGameServlet**

**Class BoardDataServlet**

**Class LoginToChatServlet**

**Class LoginServlet**

**Class GetChatUsersServlet**

**Class ExitChatServlet**

**Class Constants**

**Class CheckLoginServlet**

WebApplication\_NinaRow Module

**General structure**

* The game has 3 main modules: Engine, WebApplication\_LogicEngine, WebApplication\_NinaRow.
* The game has 1 common module: Common. This module is being used by all modules – Engine, WebApplication\_LogicEngine and WebApplication\_NinaRow.
* 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).
* The WebApplication\_NinaRow module holds all the resources – HTML pages, Bootstrap framework and the resources folder – web/src/dist, with all the resources used for the game’s display, mostly including .css and images.  
  The WebApplication\_NinaRow holds all the servlets:
  + BoardDataServlet – this servlet responds with the board information, who is the current player, the actual board, if there is a winner/tie, etc.
  + CheckLoginServlet – this servlet responds with true/false as it checks if the JSESSION received is valid. If the JSESSION is not valid or there is no JSESSION at all – it responds with false (and the front end in turn will direct the user to the registration page).
  + ExitChatServlet – this servlet removes the user from the chat room of the game he is currently registered to.
  + GameDataServlet – this servlet responds with the metadata of the game – information such as who are the players, viewers, moves made, etc.
  + GameListServlet – this servlet responds with the list of all games that were uploaded by all users and their status.
  + GetCharServlet – this servlet responds with all the messages that where sent by the users in the chatroom. The information is pull by deltas and not by pulling the entire chat history each time.
  + GetChatUsersServlet – this servlet responds with all users that are logged in to the chat.
  + LoginServlet – this servlet is used for registration purposes as a new user wants to register to the game.
  + LoginToChatServlet – this servlet is used as a user is already registered and clicks on the envelop sign in the gameBoard.html page – a registration to the chat occurs.
  + LogoutServlet – this servlet is used as a logged in user clicks on the sign out buttom (click on the user name on the top right and then the sign out button).
  + PlayerResignServlet – this servlet is used when a user is already registered to a game and decides to resign. As he clicks the resign button, the servlet deletes the user from the game and the gameBoard.html page in turn redirects the user to the gamesList.html page.
  + PlayerMoveServlet – this servlet is used for making a move in the game.
  + RegisterPlayerToGameServlet – this servlet is used to register a logged in user to the game. The servlet can be used only for a game that has not started yet (As a user clicks on the button "Play" in the gamesList.html page).
  + RegisterViewToGameServlet – this servlet is used to register a logged in user to a game as a viewer. The servlet can be used only for a game that has begun (As a user clicks on the button "View" in the gamesList.html page).
  + SendChatServlet – this servlet is used while a logged in user wants to send a message in the chat room.
  + UploadGameServlet – this servlet is used as a user selects a file and uploads a new game. The servlet responds with true while the XML file is legal and no other game with the same name was uploaded. The servlet responds with false as a game with the same name was uploaded or the XML file is illegal, and a corresponding message is displayed to the user.
  + UsersListServlet – this servlet responds with a list of all the logged in users (the list is displayed in the gamesList.html page, in the sidebar section).
  + ViewerResignServlet – this servlet acts the same as PlayerResignServlet with the difference of resigning as a viewer.
* Each servlet sends a response which is an instance of a unique class, designed for the servlet. All servlets’ responses inherite from the general ServletResponse class.
* The WebApplication\_LogicEngine – this module is comprised of 3 main parts: one which is responsible for the games, another one which is focused on the users and a third which serves the chat functionality. Each one of these parts has a dedicated manager which is created and associated (via attribute) with the servlet context.
* The visual components of the web application is based on the Bootstrap framework.

**Main Choices that were made**

* Computer’s moves are executed with a slight delay, to give a clear separation between the moves that are played.
* When a round of game is completed, either by a win or a tie, the user is redirected to the gamesList.html page and the game performs a reset. If the user was a player in the last game – the last board status is displayed. If the user was not a player in the last game – an empty board is displayed.
* At any point of the game, the user/s can choose to exit the game all together.
* Bonuses that were implemented:
  + Chat in the game.
  + Viewer – a logged in user can join the game as a viewer. We decided that a user can join as a viewer only when the game has begun – Approved by Aviad.
  + Sign in – a logged in user can play more then one game at a time.

**Important notes for running the project**

1. In order to open the startup page of the project (after copying the NinaRow.war file to Tomcat’s webapps folder), it is necessary to navigate to the following URL in the browser: <http://localhost:8080/NinaRow/index.html>
2. In order to use the chat functionality from a game board screen, the browser should be configured to allow pop-ups (from the project’s pages)

**Students’ details**

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