**Project Report: Implementing Go-Moku (five-in-row) Game**

**Team members: Piran Caci, Simen Sverdrup-Thygeson, Ojin Kim**

**Abstract**

This report provides a brief overview of the game five-in-row, also known as Go-Moku (from Japanese: five-stones), the basic structure of our program, including the single-player and multi-player mode. The multi-player mode consists of two player giving input from the command line, whereas the single-player mode is a more complicated one, since we had to implement a computer-based opponent player, which takes reasonable decisions. In the single-player mode the player is playing against the computer. The program doesn’t provide a GUI (Graphical User Interface) and will only be played by giving input in the command line.

**Introduction**

The five-in-row game is a two-player abstract strategy board game often played with Go board and pieces. Its rules are simple, but they lead to a highly complex game. The board consists of n horizontal lines and n vertical lines, where two players Black and White take turns to mark their own color on one of the empty squares. The board is more often seen in sizes of 15 x 15 or 19 x 19. Five-in-row game is a game with long history. The Black player is the one who shall start the game and is believed that the first player has an advantage of winning the game. For that purpose, we choose to set the size of the board by 15 x 15, which is believed to reduce the advantage of moving first. Our board has rows marked with alphabetically ordered capital letters, whereas colons are marked with numbers. In our program, since we don’t provide Graphical User Interface, our players are marked with O for the Black player and X for the White one. These are the rules of the game; once a marker is placed on the board (in our case O for the black player starting the game ), it can’t be moved to another position later. A marker can’t be taken over by another player, either. The player, who creates a line of five consecutive markers of his color/symbol horizontally, vertically or diagonally, wins the game. A consecutive line of 6 markers can be theoretically created but will not be counted for the player who creates it, while the opponent player will not have any restriction. If no one creates five-in-row before the board is completely filled, the game is drawn.

**Description of the program**

The program is based on the following classes;

* Main Class, which contains the driving code. The main method runs through a infinite loop which presents the user with the main menu and the different alternatives. The user can choose between singleplayer, multiplayer, ranking or quit. If the user chooses ranking, he/she will be presented with a printout of the score saved in the highscore.txt file and will also have the option to reset the score. The singleplayer and multiplayer will allow the user to play the Go-moku game in single- and multiplayer mode, respectively.
* Board Class, which is responsible for for creating a board of size 15 X 15. The class is also using a HashMap and an alphabet string to simplify getting the 2D array coordinates from the user input (e.g. “D5”). The Board class has two fundamental public methods, in addition to the constructor method;

***public void displayBoard();*** this method is for displaying the actual board every time a player’s turn comes.

***public void insertRock(String position, char rock);*** this method is for allowing each player to place a rock in the board. In our case a symbol X/O. The method will take two parameters; the position of the rock and the rock itself (i.e. ‘o’ or ‘x’ depending on which player is inserting the rock).

* Move Class, which contains numerous public methods, in addition to the constructor;

**public boolean checkEmpty(int a, int b);** this method checks whether the board position, in which the player tends to place its rock, is free. It takes to parameters a & b, which represent the coordinates of the position to be checked.

**Public Boolean withinBoard(int a, int b);** this method checks whether the position coordinates are within the board. It also takes two parameters, which represent the coordinates of the position.

**Public Boolean legalMove(String position);** this method checks whether the intended move is legal. The method works by calling the two previous methods to check whether the position is within the board and at the same time free. It takes a string parameter consisting of the position in the format “A12”. It also makes use of a HashMap to simplify coordinate extraction and the toUpperCase-embedded method to make sure that the position string won’t be case-sensitive.

**Public Boolean gameFinished(String position);** this method checks whether the game is finished. Like the above methods, it also takes a string position as parameter. The method works by iteratively looking through all the adjacent slots around the position where the latest stone was added, checking horizontally, vertically and both the diagonal possibilities. By only checking the adjacent slots to the latest inserted rock, the method works quite efficiently. The for loops in the method are also tweaked so that they never go outside the board by trying to access an out of bounds array index.

**Public Boolean ThreeThreeRule(int a, int b);** this method checks whether a given move breaks the so-called three-and-three-rule. It does this similarly to the gameFinished()-method.

* Ranking Class, which provides rankings and score-results of the players after a game is finished. The method makes use of an external file called highscores.txt, to provide a non-volatile way of saving the information between program runs. Of course, the directory path of the highscore file will vary from computer to computer and must be changed accordingly. It provides following methods:

**Public void displayscore();** this method accesses the highscore-file, which provides the scores of the players and displays them.

**Public void appendScore(String player1, String player2, int moves);** this methods gets the scores of each player and the scores are based on the number of moves that led the winning player to win the game. It gets three parameters as reference; Two string parameters with player’s names and the third integer parameter, which represents the number of moves.

**Public void resetScore();** this method resets the highscore records, it works by deleting the file of the previous highscores and creating a new one.

* AI Class, which is the class in charge of generating the computer-controlled moves. It uses two HashMaps to simplify coordinate extraction and generation and has a constructor class which takes the updated board as input. In addition to the constructor, it has one method, which makes use of the board input:

**Public String AIMove();** this method generates the opponent’s move for the singleplayer. It works by creating a 2D array of integers called valueBoard. This 2D array represents the potential-score for every position in the board. It looks through the entire board and gives a score (indicated with the value-variable) to every position based on the adjacent stones. If the position is non-empty, it will automatically be assigned a score of 0 (as the computer cannot place another stone there). After the valueBoard has been generated and filled, the method chooses the position with the maximum score as its move. If several positions share the maximum score, the move will be determined pseudo-randomly between them.

* Singleplayer Class, which is for the single-player mode of the game. It inherits the Board class and creates a Move & Ranking object as well. In addition to the constructor method, it contains;

**Public void play();** this is the only method of this class. The method puts all the other classes together by running in a loop as long as the gameFinished()-method returns false. For every iteration, it queries the player for a move, checks whether the move is legal or not (by the legalMove()-method), inserts the rock and displays the board. It also creates an AI object with the updated board and uses the AIMove()-method to determine the moves of the computer-controlled opponent.

* Multiplayer Class, which is for the multi-player mode of the game. It creates objects from the Board, Move & Ranking classes. Similar to the Singleplayer class also provides following method, in addition to its constructor method;

**Public void play();** this method works almost exacltly as the play method of the Singleplayer class, just alternating between two players inputing positions in the terminal instead of using the AI-class.

**External Tools**

We established a GitHub repository (<https://github.com/SimenSverdrup/Gomoku>), to handle version control and manage collaboration. We also used KakaoTalk to discuss various aspects of the development and cooperation, when we weren’t able to meet in person.