2022-10-29
Author: Robert Okuła
robert.okula@pg.edu.pl

1 Basics of the game of Go

The goal of the project is the creation of a simple implementation of a Go game as a console application. Go is an abstract strategy board game for exactly two players. The goal of the game is to surround ("conquer") more territory than the opponent.
The rules of (a simplified version of) the game are as follows:

1. The game is played on a square boards (ja: *goban*), with 19 lines crossing the board horizontally and 19 lines crossing the board vertically (board 19x19 – 361 intersections). Sometimes, the board size is 13x13 or even 9x9.
2. Players put black and white stones on the intersections, one stone at the time. The player that plays with the black ones is the one that starts first.
3. The goal is to surround the largest surface on the board using your own stones.
4. A stone that was put on the board stays there unless it is captured. The stone is captured when it is surrounded from all four sides by either the stone stones of a opposing colour or the edge of the board. The free sides are called *liberties*, e.g. a stone that is surrounded by two stones of a opposing colour has two liberties.
5. The player is not allowed to put a suicide stone on the board (that is – a stone that after the placement immediately does not have any liberties). The player can put a stone on a intersection with no liberties if and only if after the placement one of the stones of a opposing colour are captured.
6. The Ko rule has to be kept – *the stones on the board must never repeat a previous position of stones. Moves which would do so are forbidden, and thus only moves elsewhere on the board are permitted that turn*.

These rules can be however elaborated further.

1.1 Liberties

As mentioned before, a stone is captured when it is surrounded from all four sides by either the stone stones of a opposing colour or the edge of the board (a.i. it has no remaining liberties). However, vertically and horizontally (but not diagonally) adjacent stons of the sam colour form a chain. The chain cannot be divided and its stones share liberties – they have to be captured together. The chains can be expanded not just in one line, but also form branches – the player can put additional stones on all adjacent intersections, not just at the ends.

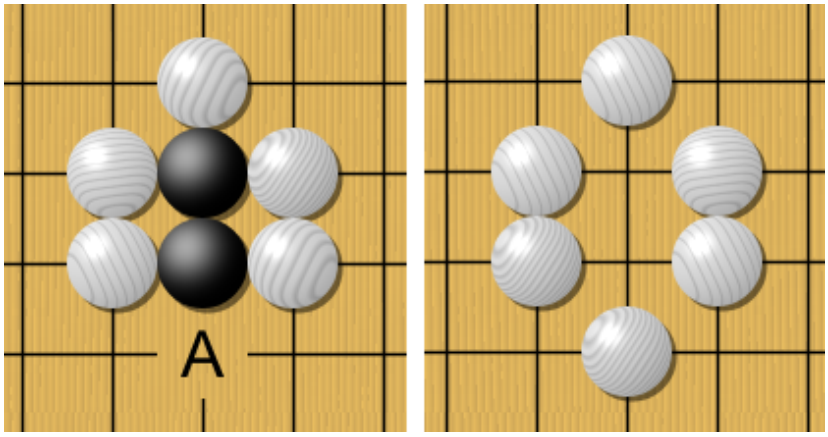

Figure 1: The capture process of a two-stone chain. Source: Stevertigo, Wikicommons, CC BY-SA 2.0

1.2 Ko rule

The player is not allowed to make a move that returns the game to the previous position – in other words, the infinite and repetitve loop with two steps is not possible. This could occur when the so-called *ko fight* happens – on the left side of the picture blacks can capture the white stone marked by the red circle. However, after that move, the possibility to capture the black stone in a similar fashion appears, which, after it is used, brings the situation to the initial state. This situation is also known as Ko. Therefore, the Ko rule demands that if one player captures the Ko, the other one is prohibited doing the same immediately, they have to wait to the next round.

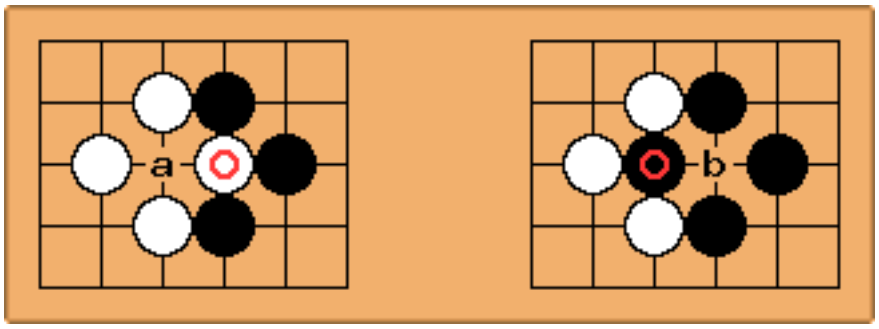

Figure 2: Ko fight. Source: Sensei's Library

1.3 Grading (and *Komi*)

In general, two types of scoring systems are used, however in this project students should use the Japanese method, sometimes called **territory scoring**. Players should keep the captured stones, which are called *prisoners*. The score is the number of empty intersections enclosed by the player's stones + the number of prisoners that were captured by the player. The game is finished either when status of every stone is determined directly or the players can just notice that some stones could be captured, but for that you need another round, so the players can just count it as if the movement was already made.
To show it in practice: picture 3 shows a simplified game at its end. A and B can be directly added to each player's score, points in F don't belong to any player. The D chain of whits has been effectively captured, as black can easily add two stones and therefore remove all liberties from the chain. The C chain (and the additional stone) however effectively conquered part of the white's territory, reducing their score.

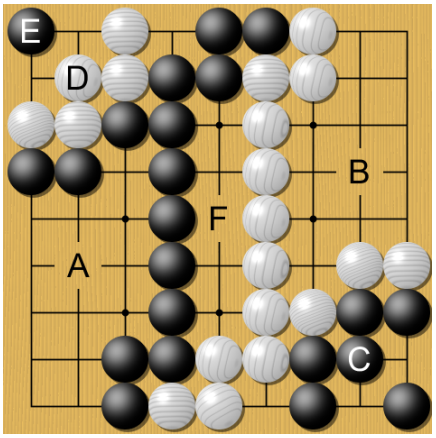

Figure 3: A simplified game at its end. Source: Scsc, Wikicommons, CC BY-SA 3.0

Additionally, as the black has the advantage of playing the first move, sometimes the compensation for that is awarded to the white. This behaviour is called *komi* and often provides a 6.5 point bonus. Sometimes when parties play in the handicapped mode (that is when some of the black stones are predefined and put on the board at start) the bonus is just 0.5-point, just to break a tie.

2 General guidelines of the project

The program should be written with the usage of the provided template. The template provides the ability to use advanced capabilities of a console in a Windows system. Usage of instructions cin/cout/printf/scanf is forbidden. In order to write on a screen and read from a keyboard only methods provided in the template should be used.
Attention: the template works in a Windows system and there is no easy way to move it to other operating systems. However, it is acceptable to prepare a similarly organized input/output without using our template (e.g. by using the ncurses library). By a ”similarly organized input/output” we in particular mean that writing to the screen should not happen in a sequential way resulting in ”scrolling” the contents.

Remember to prepare for the grading – check if you can effectively run your program on a computer in a laboratory!

The project should be written such that making visual changes should be easy (usually by modifying some constants), including (but not limited to):

- changing the placement of parts of the interface on the screen (board position),
- dimensions of the board (it should be possible to use a larger or smaller board).

2.1 Program controls

The program should be driven by a keyboard input with the following meaning assigned to keys:

- `arrows`: moving the cursor over the board;
- `q`: quit the program;
- `n`: start a new game;
- `enter`: confirm choice and end player’s turn;
- `esc`: cancel current action;
- `i`: place a stone on the board;
- `s`: save the game state;
- `l`: load the game state;
- `f`: finish the game.

2.2 Mandatory requirements (5 points)

All elements stated here have to be implemented. Lack of any of the following elements results in gaining 0 points from this project.

(a) Display a board (in one of the sizes: 9x9, 13x13, 19x19) and a legend of the program. The board should have a border. The legend should contain name, surname and album number, list of the implemented functionalities (in the form of a list of points from this instruction and a list of working keys) and a score for both black and white players. By default the legend should be displayed on the left side of a screen, while the board should be displayed on the right. There should be an ability to easily move these elements with a change of respective constants in the code of the program.

(b) Move through the board with arrows. The legend should contain the current position of the cursor. It should be impossible for the cursor to leave the board and the current position should be visible on the board. The q key should end the program.

(c) New game. Pressing `n` key resets the player tiles and returns the board to the initial state.

(d) Simple stone placement – pressing key `i` places a stone on the board at cursor position, as long as the target cell is empty and and it is not an obvious suicide (the stone after placing has at least one liberty). The game is always played by two players – after the black stone is placed, the white turn begins and so on.

(e) Simple capturing – the program should correctly detect the capture of a single stone, remove it from the board and increment the score of the right player.

2.3 Optional requirements (9 points)

(f) (**1 pt.**) *Saving and restoring game state.* Pressing `s` key allows the user to enter the name of the file in which the state of the game is stored. Pressing `l` key asks for a file name restores the state from the given file (if such file exists). The order of stones drawn should also be preserved: when a game is saved and then 3 stones are placed, when the game is reloaded and again 3 stones are drawn, the have to be located at the same exact intersections.

(g) (**1 pt.**) *Forcing the following of the rule of Ko.* When the player try to make another move in a Ko fight, after the first player did it during their round, the game should not allow to do so. If the game is saved after the Ko fight was started, it also should keep the current state and don't allow the second player to make a Ko move after loading the game.

(h) (**1 pt.**) *Changing the size of the board.* The players should be able to choose every possible size of the board, as long as it fits within the borders of the window. The option should first show three standard, predefined options (9x9, 13x13 and 19x19) and then show the fourth option for choosing the custom size.

(i) (**1 pt.**) *Scrolling.* The players should be able to choose every possible size of the board, even if it does not fit within the borders of the window. This point should be implemented after the previous one.

(j) (**2 pt.**) *Capturing.* The program should correctly detect the capture of every possible chain of stones, remove it from the board and increment the score of the right player.

(k) (**1 pt.**) *Game state editor.* The player should be allowed to enter every possiible initial configuration of black stones before the start of the gameplay (to introduce a handicap).

(l) (**2 pt.**) *Scoring.* Pressing `f` key should finish the game and show the score, calculated according to the rules described in Section 1.3. The *komi* must also be taken into consideration – when some initial handicap is entered on the board, the value of the additional points that whites get must be different.

2.4 Additional requirements (3 points)

Attention: the following requirements are graded only if all previous points (a)-(l) are implemented **for the overall amount of 14 points**.

(m) (**2 pt.**) *Detection of the finished game.* The program should find out if any other move can still be made without reducing the score of one or both players and stop the game, showing the score.

(n) (**1 pt.**) *Atari.* The program should mark the stones that have just one liberty (reminder: **chains share their liberties**) and might be captured in the next move of the opponent.

2.5 Final remarks

- Configuration of the program should make it possible to easily change all parameters, not only those explicitly pointed in the above description. Easy change is understood as modification of a constant in the program.
- The project can be written in an object oriented manner, however it is forbidden to use the standard C++ library (including the string type, cin, cout, vector etc.)(Attention: the string type from the C++ library should not be confused with the string.h library in C – functions from string.h can be used).
- Handling of files should be implemented with the usage of the standard C library (family of functions f???? - e.g. fopen, fread, fclose etc.). C++ mechanisms (e.g. fstream) can not be used to this end.
- Each fragment of code subjected to grading should be written by the student independently. Using a code found on the Internet, received from other people or written with help from other people (excluding help received during project consultations) is forbidden.
- Pay attention to a proper division of code into functions in order to avoid duplication of code. For example it might occur, that in few given points a code which checks whether a given digit meets the requirements of a given rule in a given field. In this case placing such a code inside a function, which is next called in appropriate places, is natural and encouraged.
- Constant units in the program should be described by appropriate comments.