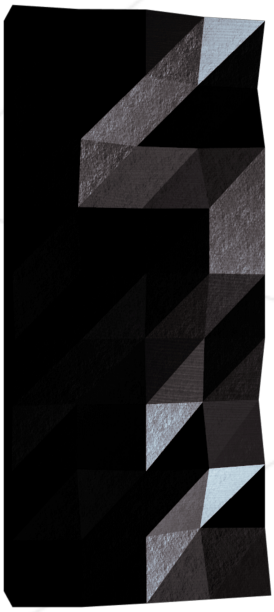


Gomoku

Yeah, well, your brain has to fry sometime

42 staff staff@42.fr

Abstract: The goal of this project is to make an AI capable of beating human players at Gomoku

Contents

Chapter I

Foreword

Here are the approximate lyrics to the beginning song of `Jay and Silent Bob Strike Back`:

```
Fuck.
Fuck.
Fuck.
Fuck.
Fuck.
Fuck, fuck, fuck !
Mother mother fuck. Mother mother fuck fuck. Mother fuck, mother fuck.
Noise noise noise.
1 2 1 2 3 4
Noise, noise, noise.
Smokin weed, smokin weed.
Doin' coke, drinkin beers.
Drinkin beers, beers, beers.
Rollin' fatties, smokin blunts.
Who smokes the blunts? We smoke the blunts.
Rollin' blunts and smokin um'
[...]
15 bucks, little man, put that shit in my hand.
If that money doesn't show then you owe me, owe me, owe !
My jungle love,
Oh eee oh eee oh,
I think I wanna know ya know ya ... yeah, what.
```

Chapter II

The project

II.1 The game

Gomoku is a strategy board game traditionally played on a Go board with stones of two different colors.

Two players take turns placing stones of their color on an intersection of the board, and the game ends when one player manages to align five stones. Sometimes, only an alignment of 5 can win, and sometimes 5 or more is okay. In the context of this projet, we will consider 5 or more to be a win.

There are different interpretations on what the board size for Gomoku should be. In the context of this project, Gomoku will be played on a 19x19 Goban, without limit to the number of stones.

There are a great many additional rules to Gomoku (Google it!) aimed at making the game more fair (regular Gomoku is proven to be unfair, a perfect first player wins 100% of the time) and more interesting.

In the context of this project, you will play with the following additional rules :

- Capture (As in the Ninuki-renju or Pente variants) : You can remove a pair of your opponent's stones from the board by flanking them with your own stones (See the appendix). This rule adds a win condition : If you manage to capture ten of your opponent's stones, you win the game.

- Game-ending capture : A player that manages to align five stones only wins if the opponent can not break this alignment by capturing a pair, or if he has already lost four pairs and the opponent can capture one more, therefore winning by capture. There is no need for the game to go on if there is no possibility of this happening.

- No double-threes : It is forbidden to play a move that introduces two free-three alignments, which would guarantee a win by alignment (See the appendix).

II.2 What you have to do

You must make a program that lets you play Gomoku (with the rules specified previously) :

- Against your program : The most interesting case, of course. The goal here is that the program wins the game, without you letting it win. It must be able to adapt its strategy to the player's moves.

- Against another human player on the same computer (hotseat), but with a move-suggestion feature. Easy to do once you have made a whole computer player.

The AI will, basically, generate a possible-solution tree, and choose the best move according to this tree. You have to use a Min-Max algorithm for this. While this is rather easy in itself, since these algorithms are well-documented and you pretty much only have to implement them, for them to actually be useful, you need an efficient heuristic function to evaluate the value of a terminal node in your tree. You will have to experiment and refine it until it is sufficiently accurate while remaining fast enough. Don't be fooled : The heuristic is actually the hardest part.

You must also provide a usable graphical interface to allow one to actually play Gomoku. After all, while the AI is the main focus here, your game still has to be playable. You're free to use whichever graphical or ncurses-like library you want for this, as long as the end result is at least vaguely pleasing to the eye, and is easily playable.

Also, it would be a very good idea to implement some sort of debugging process that lets you examine the reasoning process of your AI while it's running. Not only would it be helpful to refine your AI's tactics, but it would help during your defense sessions, since you will have to actually explain how it works.

Last point, and this is mandatory : You have to display, somewhere in your user interface, a timer that counts how much time your AI takes to find its next move. This is so your grader can precisely evaluate the performance of your AI. And yes, I mean mandatory : No timer, no project validation. It's that serious.

II.3 Bonuses

When you're finished, why not implement the possibility of choosing the game rules you want to apply when you start a game ? There are a LOT of possibilities.

You can especially look at starting conditions (Standard, Pro, Swap, Swap2...).

Of course, any other bonus you can think of that is actually interesting and/or useful will be taken into account.

II.4 Language constraints

You are free to use whatever language and graphical interface library you want. Although you must keep in mind that while there will not be a grading scale on the time/space performance of your AI relative to others (We'll see about that kind of thing later on), if your AI takes more than half a second to find a move, you will not validate the project. That'd be bad, wouldn't it ? Also, you must still keep performance for its own sake in mind : You will not get all the points if your implementation wins too slowly, or if your implementation seems lazily done (Low search depth, naive implementation, etc...)

If you use C, you have to respect the Norm, as usual.

II.5 Defense sessions

For the defense session, be prepared to :

- **Thoroughly** explain your implementation of the Minimax algorithm (or the variant you chose). For the love of `Tartiflette`, I can not stress the "`thoroughly`" part enough: If you can not explain, in detail, your implementation of the algorithm to a grader who potentially knows nothing about Minimax, then you will not get any points for it. That would be bad.

- **Thoroughly** explain your heuristic. You really need to spend some time on this, it has to be accurate, fast, AND you need to understand it well enough to explain it ! Same as before, if you can't explain it well, then you don't understand it well enough, and you won't get points for it.

- Show that you have correctly implemented the game rules imposed by the subject.

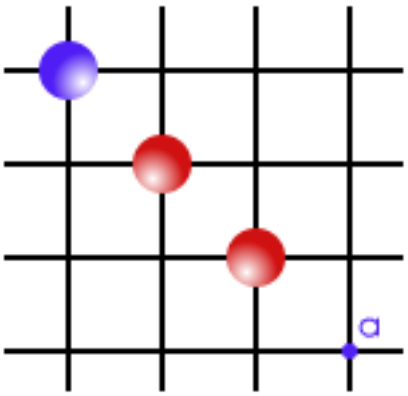
- Run your program ... and hope your AI doesn't get completely fragged by the guy who's grading you.

Chapter III

Appendix

III.1 Captures

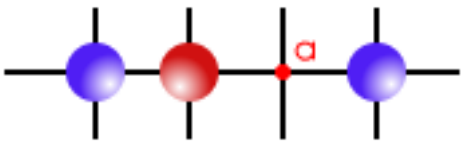
Captures are made by flanking a pair of the opponent's stones, as demonstrated below :



In this scenario, by playing in `a`, Blue captures the red pair and removes the stones from the game. The now-free intersections can be played on as if they were never occupied.

One can only capture PAIRS, not single stones, and not more than 2 stones in a row.

Also note that one can not move into a capture. Example :

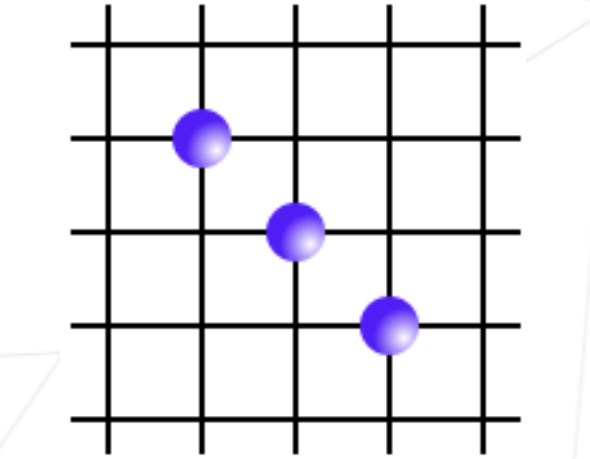


In this scenario, Red can play in `a` without losing the pair. However, if later Red takes one of the Blue stones, his position becomes vulnerable to capture...

III.2 Free-threes

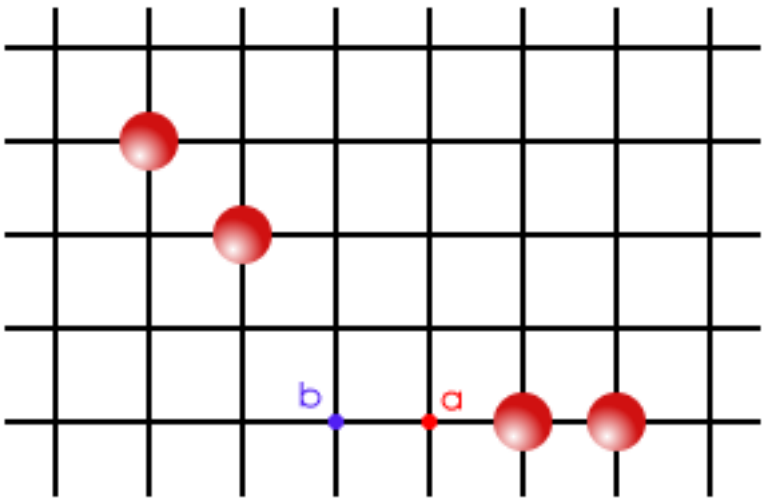
A free-three is an alignement of three stones that, if not immediately blocked, allows for an indefendable alignment of four stones (that's to say an alignment of four stones with two unobstructed extremities).

Both of these scenarios are free-threes:



A double-three is a move that introduces two simultaneous free-three alignments. This is an indefendable scenario.

Example:



In this scenario, by playing in `a`, Red would introduce a double-three, therefore this is a forbidden move. However, if there were a blue stone in `b`, one of the three-aligned would be obstructed, therefore the move in `a` would be legal.

It is important to note that it is not forbidden to introduce a double-three by capturing a pair.