Artificial Intelligence for Go

CIS 499 SENIOR PROJECT DESIGN DOCUMENT

Kristen Ying
Advisors: Dr. Norm Badler, Dr. Maxim Likhachev
University of Pennsylvania

PROJECT ABSTRACT

Computers have beaten pro human chess players through their superior computational ability. With the ancient board game Go, however, the sheer number of possibilities, as well as the subjective nature of what a more desirable board state is make it a more challenging problem. On February 7th, 2008, a computer finally beat a pro Go player – but with a 9-stone handicap on the human player, and with processing power over 1000 times that of Deep Blue.

Crazy Stone by Rémi Coulom uses Monte-Carlo Tree Search and in 2006 began the current trend of using algorithms from this family. The Monte-Carlo method creates playouts, or played games with random (light playout) or heuristic-based (heavy playout) moves. Applied to game trees, each node keeps a win rate, remembering the number of playouts that were won from this position. As is often desirable in Go, such analysis favors the potential of winning, regardless of the potential margin by which the game may be won. Thus such algorithms may often result in winning by a small margin. Crazy Stone also utilizes pattern learning. The program MoGo, which received some early inspiration from Crazy Stone, made headlines when it defeated a pro player in a full-scale 19x19 game of Go on February 7th, 2008. The program uses the algorithm UTC (Upper Confidence bounds applied to Trees) for Monte-Carlo.

This project is an investigation into designing AI for the board game Go, Using a Monte Carlo Search Tree algorithm. The goal here is to create a program that is playable with limited computational resources, allowing it to be presented in the form of a console game. The end product should be a game on the XBox 360 which can be played by novice Go players.

Project blog:
http://kaising.wordpress.com

1. INTROUDUCTION

Computers have beaten pro human chess players through their superior computational ability. With the ancient board game Go, however, the sheer number of possibilities, as well as the subjective nature of what a more desirable board state is make it a more challenging problem. On February 7th, 2008, a computer finally beat a pro Go player – but with a 9-stone handicap on the human player, and with processing power over 1000 times that of Deep Blue.

1.1. Significance of Problem or Production/Development Need

The purpose of this project is to apply the Monte Carlo Tree Search algorithm to the complicated problem of having a program with limited computational resources play Go. Additionally, it is also intended as an investigation into the field of AI and XNA, to gain personal experience in preparation for entry into the video game industry.

1.2. Technology

This project will use C# and eventually XNA to produce code, hopefully progressing to use an XBox360 developer kit. The main papers will be technical papers written by the creators of Crazy Stone and MoGo.

1.3. Design Goals

1.3.1 Target Audience.

The target audience of the final product is recreational / amateur Go players; people who would like to practice new knowledge of Go. The aim is not to be the best Go AI, but to be playable with only the computational resources of an XBox 360.

1.3.2 User goals and objectives

The user should be able to play a game of Go against a program with reasonably fast responses, as a recreational game or learning tool to reinforce beginners' knowledge of the game.

1.3.3 Project features and functionality

The main features of the game are 1. The ability to play Go against a human component, and 2. A visual interface for the game on the XBox 360.

2. Prior Work

One of the first, if not the very first implementations of AI for Go was by PhD student Albert Zobrist in 1970. It used two influence functions to assign numeric values to the possible move locations. One influence function was based on which colors occupied what locations; it gave +50 to a location with a black stone, and – 50 to a location with a white stone. Then, for four iterations, positions received -1 for each adjacent location with a negative value, and received +1 for each adjacent location with a positive value. The other influence function was based on which locations were occupied. Based on available information, then, the program pattern matched against a database. Via scanning the board searching for various rotations of each pattern, it would decide what the next move should be. In order to make the program's decision more sound, the game was divided into stages (e.g. beginning, endgame, etc.), and only patterns appropriate for the given stage were searched for. Some lookahead (3 moves) was added to incorporate particular aspects of the game that require some planning (e.g. saving/capturing strings, connecting/cutting strings, ladders, making eyes). This program was able to defeat novices. Some other earlier programs were also based on variations of influence functions and pattern matching, though a number tried to account for more aspects of the game, such as attempting to build models analogous to the way Go players structure their perception of the game.

Crazy Stone by Rémi Coulom uses Monte-Carlo Tree Search and in 2006 began the current trend of using algorithms from this family. The Monte-Carlo method creates playouts, or played games with random (light playout) or heuristic-based (heavy playout) moves. Applied to game trees, each node keeps a win rate, remembering the number of playouts that were won from this position. As is often desirable in Go, such analysis favors the potential of winning, regardless of the potential margin by which the game may be won. Thus such algorithms may often result in winning by a small margin. Crazy Stone also utilizes pattern learning. The program MoGo, which received some early inspiration from Crazy Stone, made headlines when it defeated a pro player in a full-scale 19x19 game of Go on February 7th, 2008. This program uses the algorithm UTC (Upper Confidence bounds applied to Trees) for Monte-Carlo.

There is also a commercial computer program called "Many Faces of Go" that uses a variant of Monte-Carlo Tree Search, indicating that this can be playable without the great computing power that MoGo had access to.

The main challenges of Go seem to be the sheer number of possible moves for each player, as well as the issue of evaluating how "good" a given board configuration is for a player. For example of the latter, even a beginning player may be able to recognize the potential for a stone pattern called the 'ladder' and look 40+ turns ahead to see how much it could benefit them (a very deep search for a naive branching algorithm). Thus encoding such evaluation in a program is not trivial.

3. PROJECT DEVELOPMENT APPROACH

3.1. Algorithm Details

The algorithm for the Go AI will be in the family of Monte-Carlo Tree Search algorithms. As described above, this algorithm creates playouts, and evaluates moves based on win counts at a given node in the stored tree. This algorithm was first popularized for Go by Rémi Coulom's Crazy Stone in 2006.

3.2. Target Platforms

3.2.1 Hardware

XBox 360

3.2.2 Software

C# and XNA

4. WORK PLAN

4.1.1. Project Milestone Report (Alpha Version)

A program against which a user may play a full game of Go. The interface may be as simple as command-line entry of positions to place the stones down on, and an ASCII representation of the current board state.

Dates:

01.21.2009	Meeting with Dr. Likhachev to discuss project concept
01.26.2009	Acquisition of basic knowledge of Go
02.02.2009	Investigation of XNA's capabilities, bare bones C# data structures / rules begun (should be able to play a 2 person command line version of Go).

02.09.2009 Finished reading through background papers and those recommended by Dr. Likhachev; final evaluation on which algorithmic approach to take
*Evaluation of whether Go is doable, or if a different game should be selected
02.16.2009 Begin coding AI
02.23.2009 Pre-alpha evaluation of what is reasonable to expect by alpha version
03.02.2009 Finished alpha version (playable)

4.1.2. Project Final Deliverables

The final product will build on the AI from the Alpha version, and be a program against which a user can play Go on an XBox 360.

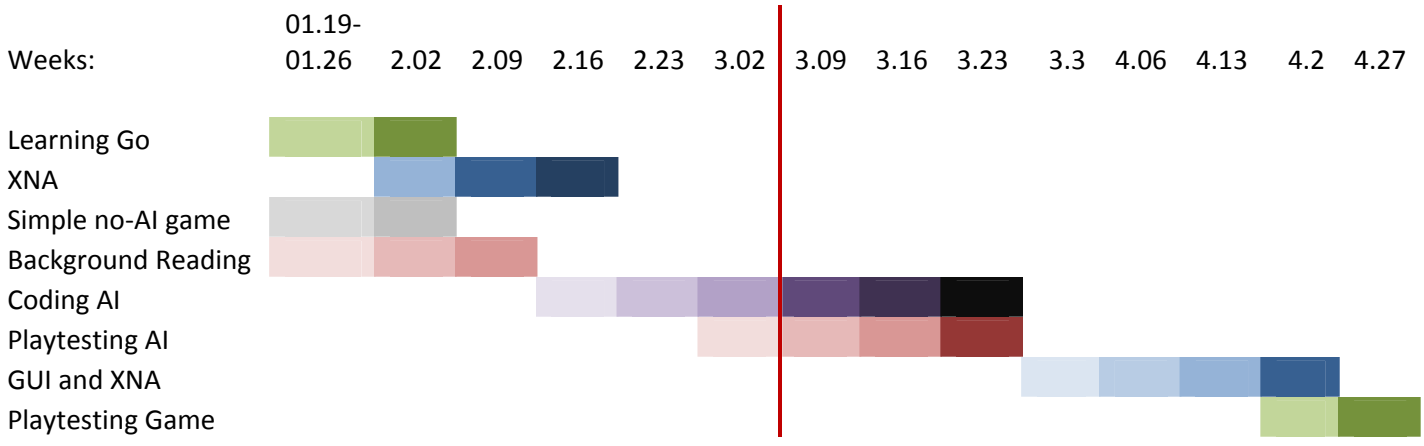
Dates:

03.09.2009 Solidify what will be expected of final version
03.16.2009 More coding and evaluation
03.23.2009 Finished AI
03.30.2009 Preliminary user interface
04.06.2009 Playtesting user interface
04.13.2009 Testable 'finished' product
04.20.2009 Evaluation of product
04.27.2009 Incorporation of findings from evaluation

4.1.3 Project timeline.

Please see 4.1.1 and 4.1.2.

4.1.4 Gant Chart



5. REFERENCES

Learning Go:

British Go Association. "Introduction to the Game of Go." *British Go Association*. URL: http://www.britgo.org/intro/intro1.html. April 16, 2008.

Mori, Hiroki. "The Interactive Way To Go." *PlayGO.to.* URL: http://www.playgo.to/interactive/. 1997-2001.

Reiss, Michael. "Go in Five Minutes Flat." *Mick's Computer Go Page*. URL: http://www.reiss.demon.co.uk/webgo/rules.htm. 2004.

Existing Programs and Algorithms:

"AITopics/Go." *Association for the Advancement of Artificial Intelligence.* URL: http://www.aaai.org/AITopics/pmwiki/pmwiki.php/AITopics/Go. January 22, 2009.

Has useful summaries and links for topics such as Crazy Stone and MoGo.

Burmeister, Jay, and Wiles, Janet. "CS-TR-339 Computer Go Tech Report." *The University of Queensland Australia*. URL: http://www.itee.uq.edu.au/~janetw/Computer%20Go/CS-TR-339.html#3.0. 1995.

This describes an overview of various academic implementations of Computer Go, as well as more recent competitive ones.

CiteULike. "Group: computer-go." *CiteULike.* URL: http://www.citeulike.org/group/5884/library. January 22, 2009.

There are 420 articles here, many of which refer to Monte-Carlo – based techniques, and a few of which refer to genetic algorithms. Once it has been decided which will be used as resources, those chosen articles will be listed as individual sources.

Coulom, Rémi. "Efficient Selectivity and Back-up Operators in Monte-Carlo Tree Search." *Rémi Coulom's Home Page.* URL: http://remi.coulom.free.fr/CG2006/. January 22, 2009.

This is the page for Crazy Stone, by programmer Rémi Coulom. It includes a technical paper describing the program.

Gelly, Sylvain. "MoGo." *Sylvain Gelly's Home Page*. URL: http://www.lri.fr/~gelly/MoGo.htm. January 22, 2009.

Home Page of one of the programmers behind MoGo; contains links to a technical report and other information.

Gelly, Sylvain, and Wang, Yizao. "Exploration and Exploitation in Go: UCT for Monte-Carlo Go." *University College London.* URL: http://www.homepages.ucl.ac.uk/~ucabzhu/workshop_talks/mogoNIPSWorkshop.pdf. December 9, 2006.

MoGo project. "MoGo: a software for the Game of Go." URL: http://www.lri.fr/~teytaud/mogo.html. January 22, 2009.

Home page of MoGo, which is considered to be the first program to beat a pro player at Go.

"Monte Carlo Tree Search." *Sensei's Library*. URL: http://senseis.xmp.net/?MonteCarloTreeSearch. September 11, 2008.

This describes the family of algorithms used by competitive Go programs such as Crazy Stone and MoGo.

Reiss, Mick. "Go++, the world's strongest Go playing program." *Go++.* URL: http://www.goplusplus.com/. January 22, 2009.

This program features board sizes of 19x19, 13x13, and 9x9. It has features such as handicaps and load/save, and has won several tournaments. It seems that at least Reiss' Go4++ program would require too great computational resources for this senior project.

Shiba, Kojiro, and Mori, Kunihiko. "Detection of Go-board Contour in Real Image using Genetic Algorithm." *IEEE Xplore.* URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=01491921. August 4, 2004.

This is a report available from IEEE Xplore, originally from the 2004 SICE Annual Conference in Sapporo.

Online Go Servers:

IGS the Internet Go Server. URL: http://www.pandanet.co.jp/English/.

Online Go Server. URL: http://www.online-go.com/.

The KGS Go Server. URL: http://www.gokgs.com/.

Learning XNA and C#:

Dolan, Mike. "XNA: Make your own XBOX games in 10 steps." *Fierce Developer.* URL: http://www.fiercedeveloper.com/story/xbox-make-your-own-games-xna-10-steps-diy. December 16, 2006.

Liberty, Jesse, and Xie, Donald. "Programming C# 3.0, 5th Edition." *ProQuest.* URL: http://proquest.safaribooksonline.com/9780596527433. December 20, 2007.

Microsoft Corporation. "Visual C#." *Microsoft.* URL: http://msdn.microsoft.com/en-us/library/kx37x362.aspx. 2009.

Microsoft Corporation. "Learn XNA Game Studio Express." *Microsoft*. URL: http://msdn.microsoft.com/en-us/xna/bb219593.aspx. 2009.

Omark, Jöran. "XNA Tutorial." *XNAtutorial.com.* URL: http://www.xnatutorial.com/?page_id=46. 2006.