Gobang

Licheng Yu
Department of Computer Science
University of North Carolina at Chapel
Hill
licheng@cs.unc.edu

ABSTRACT

Gobang is an abstract strategy board game, also called Five in a Row or Gomoku. It is traditionally played with Go pieces (black and white stones) on a go board with 15 x 15 intersections. Players alternate in placing a stone of theirs color (black or white) on an empty intersection. The winner is the first player to get an unbroken row of fines stones horizontally, vertically, or diagonally. In this project, we implement an Android App of this game, supporting two-player mode and one-player mode.

CCS CONCEPTS

• **Algorithm** → MinMax, MinMax with alpha-beta pruning• **Android** → Application, Game

KEYWORDS

MinMax, alpha-beta pruning, Android, Gobang

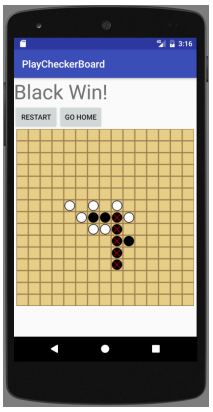

Figure 1: Gobang. Figure shows when player (black) has won the game.

Table I. Functions used for the GUI interface

Function Name	Function
void gohome (…)	Go back to Main Activity
void restart(…)	Restart this game
void addButton(…)	Add 15x15 buttons
void setButtonClick(…)	Button OnClick function
void SwitchTurn(…)	Swith player turn
bool CheckIfWin(…)	Check if some player just won
void Red(…)	Highlight the five wining stones

1 INTRODUCTION

Gobang is widely popular in Asia. Most parents would love to teach their children to play this game. In this project, we implement an android app – Gobang. It has two features. The first feature is supporting two players playing against each other. The second feature is supporting one player playing against artificial intelligence (AI). In this mode, we provide two variants, where one is difficulty selection (easy, medium, hard) and the other is who goes first selection (player goes first or AI goest first). In Figure 1, we show the GUI interface of our designed app.

2 Implementation details

2.1 GUI interface

In the MainActivity, we provide two buttons – one-player mode and two-player mode. Each button points to a new view. As there are 15x15 buttons insides a full checker board. It is impossible to manually add these buttons in layout files. So we only define LinearLayout in the layout file, and programmatically add the 15x15 buttons in the java code. We list the function names used in the GUI interface in Table I.

For the play-against-AI interface, we use the same functions. We also design a new java class called chessboard.java for the AI computing.

2.2 Algorithm: MinMax with alpha-beta pruning

2.2.1 MinMax Algorithm. MinMax is a decision rule used in decision theory, game theory, statistics and philosophy for minimizing the possible loss for a worst case (maximum loss) scenario. Originally formulated for two-player zero-sum game theory, covering both the cases where players take alternative moves and those where they make simultaneous moves, it has also been extended to more complex games and to general decision-making in the presence of uncertainty.

The key to the Minmax algorithm is a back and forth between two players, where the player whose “turn it is” desires to pick the move with the maximum score. In turn, the scores for each of the available moves are determined by the opposing player deciding which of its available moves has the minimum score. And the scores for the opposing players moves are again determined by the

turn – taking player trying to maximize its score and so on all the way down the move tree to an end state.

This algorithm is recursive, it flips back and forth between the players until a final score is found.

2.2.2 Alpha-beta pruning. Alpha-beta pruning is a search algorithm that seeks to decrease the number of nodes that are evaluated by the minmax algorithm in its search tree. It is an adversarial search algorithm. It stops completely evaluating a move when at lease one possibility has been found that proves the move to be worse than a previously examined move. Such moves need not be evaluated further. When applied to a standard minmax true, it returns the same move as minmax would, but prunes away branches that cannot possibly influence the final decision.

In this app, we do not explore all cases by searching till the end of the tree. Instead, we pre-define the depth to control the difficulty of this game. Here, we set depth=0 for easy mode, depth=2 for medium mode, and depth=4 for hard mode.

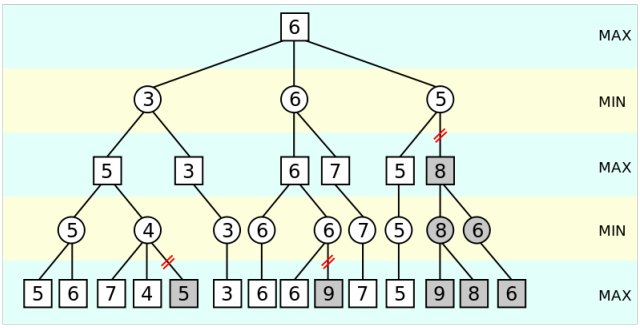

Figure 2: An illustration of alpha-beta pruning. The grayed-out subtrees need not be explored (when moves are evaluated from left to right), since we know the group of subtress as a whole yields the value of an equivalent subtree or worse, and as such cannot influence the final result. The max and min levels represent the turn of the player and the adversary, respectively.

3 Examples

In this section, we show more examples of our game, specifically the one-player mode, which is us playing against AI system in Fig. 3. To give a rough sense about how advanced is the AI. When playing against AI, my winning rate is below 10%, which indicates the powerful intelligence of MinMax approach.

In order to make this happen, we need to evaluate the reward for each stone placing. The used functions are listed in Table. II.

Table II. Functions used for the AI ChessBoard

Function Name	Function
void count (…)	#same-color stones in one direction
bool makeSense(…)	if it is worthwhile to place the stone
int getvalue(…)	evaluate local value placing some stone
int getreward(…)	evaluate global reward placing some stone
int[][] getBests(…)	local optimal best 8 places for color
int Max(…)	Max in MinMax algorithm
int Min(…)	Min in MinMax algorithm
int[] putOne(…)	place (i, j) to put color stone

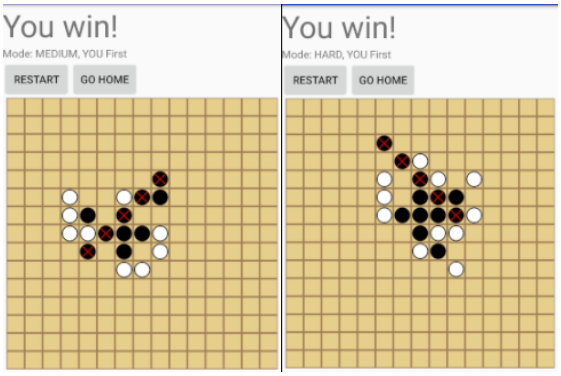

Figure 3: Two examples showing “me” wining the game against AI’s medium and hard level.