City University of Hong Kong

Department of Computer Science

CS4386 AI Game Programming

Semester B, 2022-2023

Assignment II report

Name: LI Xiao Yang

SID: 56638660

Contents

1 Algorithm

1.1 Minimax

In comparison to the previous assignment, the nature of Wolf-Eat-Sheep enables static evaluation of certain game states. Therefore, minimax algorithm is adopted in this assignment. In order to decrease the number of the nodes evaluated by minimax, alpha beta pruning is facilitated in the implementation of minimax algorithm. Besides, to significantly reduce the number of steps needed for one game, a **Checkmate** scheme is introduced to check whether there is a trivial move to win.

The maximin value is the highest value that the player can be sure to get without knowing the actions of the other players. Equivalently, it is the lowest value the other players can force the player to receive when they know the player's action. Calculating the maximin value of a player is done in a worst-case approach. For each possible action of the player, all possible actions of the other players are checked and the worst possible combination of actions are determined, namely the one that gives current player the smallest value. Then, we determine which action to take in order to make sure that this smallest value is the highest possible.

1.2 Alpha beta pruning

Alpha beta pruning maintains two values, alpha α and beta β, which respectively represent the minimum score that the maximizing player is assured of and the maximum score that the minimizing player is assured of. Initially, alpha is negative infinity and beta is positive infinity. That is to say, both players start with their worst possible score. Whenever the maximum score that the minimizing player is assured of becomes less than the minimum score that the maximizing player is assured of ($\beta < \alpha$), the maximizing player need not consider further descendants of this node, as they will never be reached in the actual play.

A critical problem in alpha beta pruning is whether a pruning is required when alpha is equal to beta. In my implementation of alpha beta pruning, a pruning will be conducted with a probability $p = 0.5$ if $\alpha = \beta$, which avoids being trapped into the local optimal. In real practice, this eliminates duplicate moves, thereby significantly improve the performance of algorithm.

1.3 Zobrist hashing

Zobrist hashing starts by randomly generating bitstrings for each possible element of a board game. In the game of Wolf-Eat-Sheep, that is 2 pieces $\times$ 25 board positions. Now any board configuration can be broken up into independent piece and position components, which are mapped to the random bitstrings generated beforehand. The final Zobrist hash is computed by combining those bitstrings using bitwise XOR.

Zobrist hashing is utilized to avoid evaluating repeated game states. Taking memory cost into consideration, the hash table will be clear once the number of entries in it exceeds the threshold. An emperical value of the threshold is $2^{20} = 1024 * 1024$ in my trial and error.

2 Pseudocode

Algorithm 1 MaxPlay

```
Input: alpha $\alpha$, beta $\beta$, state $s$, depth $d$, hash key $k$, hash table $t$
if $k \in t$ then
    return $Score_{hash}, Move_{hash}$
end if
if $d = 0$ then
    return $heuristic(s)$
end if
$value \leftarrow -\infty$
for $move \in moves$ do
    $\widetilde{s} \leftarrow Move(s, move)$
    $score, move \leftarrow MinPlay(\widetilde{s}, \alpha, \beta, d - 1)$
    if $score > value$ then
        $value, bestMove \leftarrow score, move$
    end if
    if $value \geq \beta$ then
        break
    end if
end for
$t \leftarrow t \cup s$
return $value, bestMove$
```

Algorithm 2 MinPlay

```
Input: alpha $\alpha$, beta $\beta$, state $s$, depth $d$, hash key $k$, hash table $t$
if $k \in t$ then
    return $Score_{hash}, Move_{hash}$
end if
if $d = 0$ then
    return $heuristic(s)$
end if
$value \leftarrow \infty$
for $move \in moves$ do
    $\widetilde{s} \leftarrow Move(s, move)$
    $score, move \leftarrow MaxPlay(\widetilde{s}, \alpha, \beta, d - 1)$
    if $score < value$ then
        $value, bestMove \leftarrow score, move$
    end if
    if $value \leq \alpha$ then
        break
    end if
end for
$t \leftarrow t \cup s$
return $value, bestMove$
```

Algorithm 3 Minimax + Alpha beta pruning + Checkmate

```
Input: alpha $\alpha$, beta $\beta$, state $s$, depth $d$, hash key $k$, hash table $t$, move turn $turn$
if Checkmate then
    return $W_{win}$ or $S_{win}$, trivial move
end if
if maximize player then
    return $MaxPlay(s, \alpha, \beta, d - 1)$
else if minimize player then
    return $MinPlay(s, \alpha, \beta, d - 1)$
end if
```

3 Trick

3.1 Checkmate verification

Checkmate scheme refers to the verification before minmax search. If there is a trivial **Checkmate** move for either wolf or sheep, then return it without searching for a best move. The simple trick is inspired by the assessment of average steps for win. Also, it reduces the overall runtime in real practice.

3.2 Non-copy implementation

Deep copy of matrix is both time-consuming and memory-consuming in my trial and error. Therefore, all deep copy operations are replaced by equivalent operations in my implementation, which proves to be very efficient.

4 Setup

SEARCH DEPTH	8
MAX HASH TABLE SIZE	$2^{20} = 1048576$
WOLF TRAPPED (heuristic)	-50
EAT SHEEP (heuristic)	10
EMPTY ADJACENT CELL (heuristic)	1
WOLF WIN (heuristic)	1000
SHEEP WIN (heuristic)	-1000