CMPUT 657 Heuristic Search Assignment 4 Project Presentation

Instructor: Prof. Michael Buro

Talha Ibn Aziz
Thesis-based Master's Student
Student ID: 1669108

Introduction

- Project: Using Monte-Carlo Tree Search (MCTS) to generate moves in a tactical domain: Ataxx
- MCTS – selection, expansion, simluation, propagation
- Why MCTS?
  - random simulation – stochastic domains
  - can stop at any moment in the search
  - if the number of iterations is high enough, MCTS converges to Minimax (without pruning)[1]
  - searches better in large sample spaces

Introduction

- Rules of the game Ataxx
  - board – size from 4x4, upto 8x8, obstacles
  - cloning and jumping moves
  - game over if no moves present for both players
  - game over if any player attempts more than 50 jumps
- MCTS working principle
  - generate trees for every move and remove when done
  - at every iteration add a new node in the tree
  - simulation and selection is semi random

Design

- Three components
  - Interface – implements the menu and drives others
  - State – implements the game environment
  - Search – implements the search algorithms
- State is the most independent component and is only called by other components
- Search uses State to modify the board states
- Why the encapsulated forms?

Design

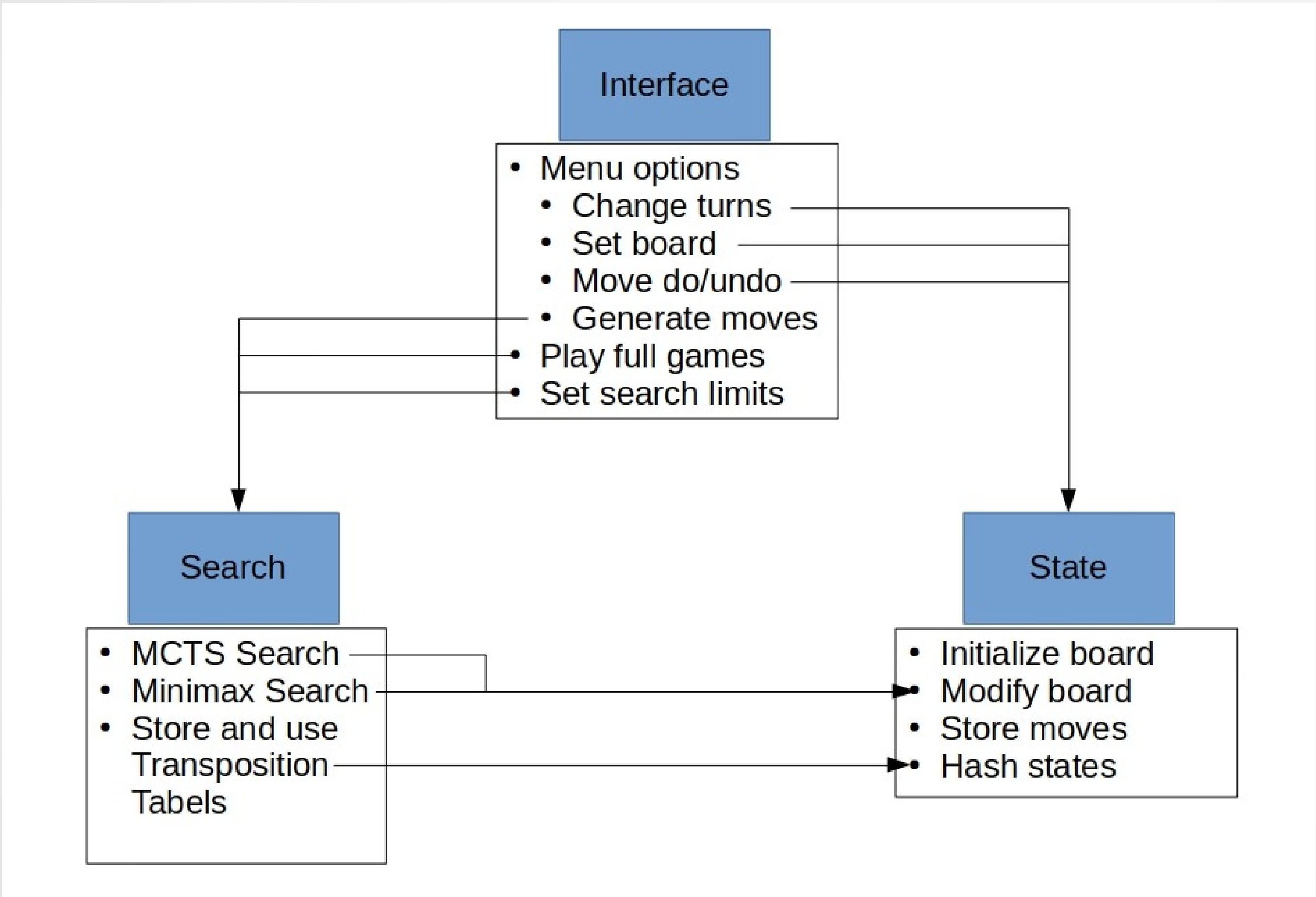
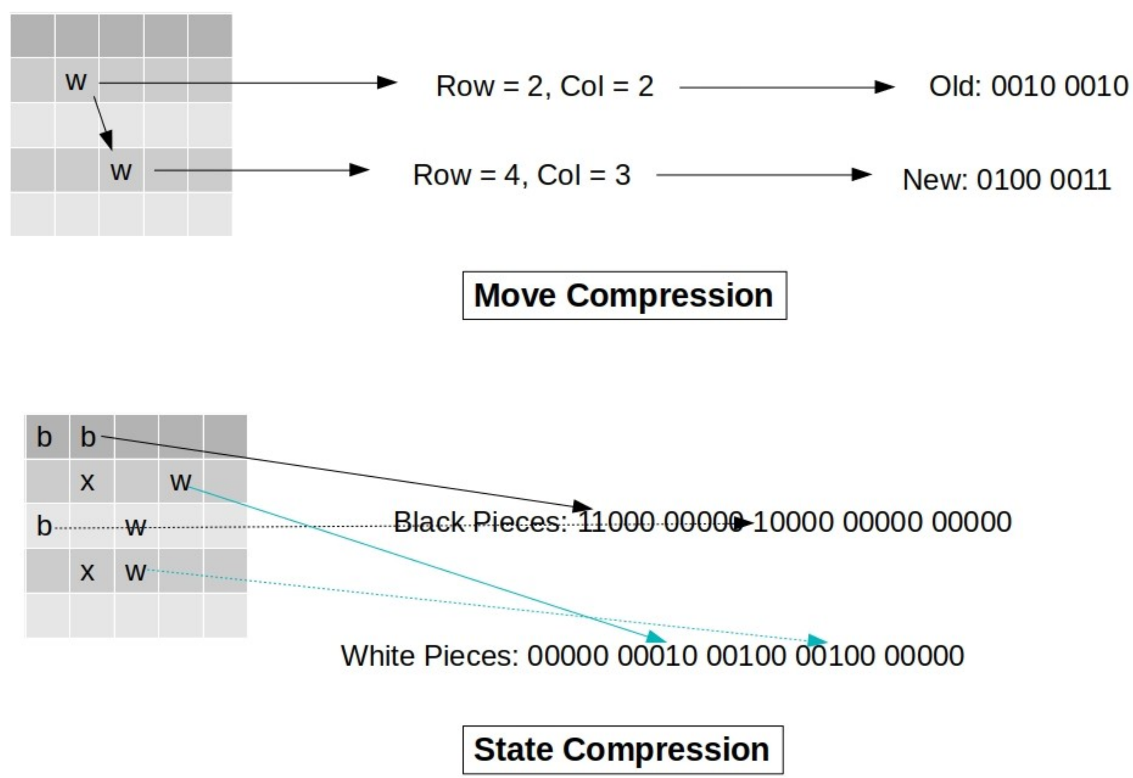


Figure: Project Design Diagram

Implementation

- Interface component – main function (nothing worthwhile)
- State component – class “ataxx_state”
  - all functions are public, most variables are private
  - few get and set functions, do/undo, game over, etc.
  - board is stored in a character array (string)
  - hashing states using Zobrist Table
  - move history using compressed move

Implementation



Move Compression: Convert the move locations into bit positions and store as character/integers

State Compression: Convert each piece locations into bits and store as 64-bit integers

Move Generation:

```
/**
Direction matrices for piece capture
This is also used to store history
*/
int row_d1[8] = {+1,-1,0,0,+1,-1,+1,-1};
int col_d1[8] = {0,0,+1,-1,-1,+1,+1,-1};

///Direction matrices for both one and two-cell moves
int row_d2[24] = {+1,-1,0,0,+1,-1,+1,-1, 2,2,2,2,2, 1,0,-1, -2,-2,-2,-2,-2, -1,0,1};
int col_d2[24] = {0,0,+1,-1,-1,+1,+1,-1, 2,1,0,-1,-2, -2,-2,-2, -2,-1,0,1,2, 2,2,2};
```

Implementation

- Search Component – class “ataxx_search”
  - Implements MCTS with a driver function and a recursive function
  - Minimax is also implemented
  - Stores and maintains the Transposition Tables and MCTS tree (size = 5000011)
- Minimax with Alpha-Beta pruning (nothing new)
  - keep searching and pruning (until timeout)
  - no move ordering except cloning and TT move

Implementation

- MCTS uses fixed memory (pre-allocated) for speed
- Selection: choose node with best win probability
  - Keep choosing until end of tree is reached
  - <u>Randomly</u> choose a non-best node
  - <u>Randomly</u> stop choosing (breakout)
- Expansion: create new node and add (from fixed memory)
- Simulation: call the recursive function (explained later)
- Propagate: pass along the result (while modifying state)

Implementation

- Simulation: Recursive function
  - Keep searching until game over
  - Select <u>random</u> move in every step
  - Return the score when done
- In all random steps,
  - Different extents of determinism may be used

Results

- Simulation Environment
  - Language: C++, Lightweight IDE: Geany
  - Competitor: Minimax Search with Alpha-Beta pruning and using Transposition Table
- The Transposition Table and MCTS tree size are same
- Assignment 1 code refactored for encapsulation
- Comparison between old and new code
  - old code uses arrays and is faster (array vs vector)
  - draw at fixed depth, old code wins in fixed time

Results

- In short: COULD NOT BEAT Minimax Search
- Reasons:
  - Random moves are worse in Ataxx
  - Even if converges, still not using pruning and TT
  - Duplicate states are not handled (also in Minimax)
- Used various levels of randomness for experiments
- MCTS tends to perform wrong jump moves even when probabilistically forced to clone

Results

Step 1: Selection	Step 3: Simulation	Tree Structure
Deterministic	Random	Linear
Semi-Random	Random	Spread-out and shallow
Semi-Random	Forced Cloning	Spread-out and shallow

Table 1: Search Tree Variations

- As MCTS is dominated in open boards, used a board with varying obstacles to compare the results
- Why?
  - Obstacles reduce the number of possible moves
  - Easy to study and analyse the players

Results

- Map 1:
  - Center is not reachable
  - Actually a linear map
  - Optimal move: clone until meeting

```
  a b c d e f g h
8 w _ _ _ _ _ _ _
7 _ x x x x x x _
6 _ x x x x x x _
5 _ x x _ _ x x _
4 _ x x _ _ x x _
3 _ x x x x x x _
2 _ x x x x x x _
1 _ _ _ _ _ _ _ b
```

- MCTS loses (MCTS: 9, Minimax: -19)
- Even if there is little probability of jumping, MCTS loses as one early jump move reduces score (for sure)
- Winning Strategy: only jump when you can jump adjacent to the opponent (blocking off their advance)

Results

- Map 2:
  - Center is reachable
  - Non-linear map, complex strategy
- MCTS loses (MCTS: 0, Minimax: 44)
- Number of moves: 39 (may change)
- Winning Strategy: keep cloning and jump when enemy is closeby (but not always)

```
  a b c d e f g h
8 w _ _ _ _ _ _ _
7 _ x x x x x x _
6 _ x _ _ _ _ x _
5 _ x _ _ _ _ x _
4 _ x _ _ _ _ x _
3 _ x _ _ _ _ x _
2 _ x x x x x x _
1 _ _ _ _ _ _ _ b
```

Results

- Map 3:
  - Standard map (no blocked areas)
  - Players start from neutral positions
- Whitewash in 55 moves or
- MCTS loses (MCTS: 2, Minimax: 55)
- Winning Strategy: Same as before (maybe?)
- Minimax sometimes shoots to the opponent (with backup)

```
  a b c d e f g h
8 w _ _ _ _ _ _ _
7 _ _ _ _ _ _ _ _
6 _ x _ _ _ _ x _
5 _ _ _ _ _ _ _ _
4 _ _ _ _ _ _ _ _
3 _ x _ _ _ _ x _
2 _ _ _ _ _ _ _ _
1 _ _ _ _ _ _ _ b
```

```
  a b c d e f g h
8 w b b b b b b b
7 w b b b b b b b
6 b x b b b b x b
5 b b b b b b b b
4 b b b b b b b b
3 b x b b b b x b
2 b b b b b b b b
1 b b b b b b b _
```

Reference

(1) Bouzy, Bruno. "Old-fashioned Computer Go vs Monte-Carlo Go" (PDF). IEEE Symposium on Computational Intelligence and Games, April 1–5, 2007, Hilton Hawaiian Village, Honolulu, Hawaii