HW1

Puzzles

Index

Chinese Dark Chess

- All assignments of this course will be based on the popular park-bench pastime.
- Chinese Chess with extra quirks
  - **Stochastic**: pieces are randomly distributed face down
  - Simpler movement rules
  - Played on half a board: 4x8 squares
- For HW1, we will play a slightly different version of the game

Chinese Dark Chess Rules

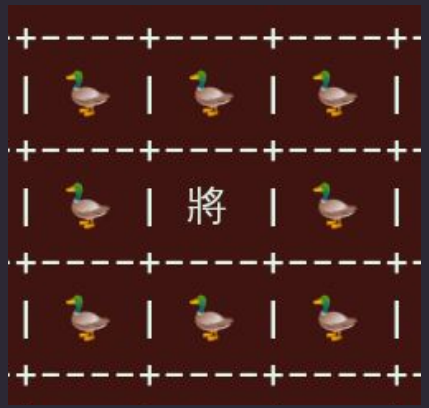
- Movements
  - All pieces move one square in any of 4 directions
  - Except the cannon
- Ranks
  - There is a hierarchy of pieces, only certain pieces can capture others
  - General (將) > Advisors (士) > Elephant (象) > Chariot (車) > Horse (馬) > Cannon (包) > Soldier (卒)
    - Cannons can capture anyone
    - Soldiers can capture Generals, and *not* vice versa
  - You don’t really need to remember this, the code provided will take care of it for you

Chinese Chess Puzzles

You will solve one-player **puzzles** of Chinese Dark Chess

- **Fullbright**: no hidden pieces
- **Ramming speed**: chariots (車) can move any distance
- **Ducks(🦆)**: new special pieces that just sits there
  - Cannot be captured or moved
  - Ducks will always be on the Black side (in the code)
- **Cold weaponry**: cannons (炮) have been banned
  - Because they are too hard to deal with

Your ducks may vary



Your goal

End the puzzle by winning:

- Move your pieces to capture all your opponent's pieces

⭐ Achieve victory **in the fewest moves possible**.

- You can also win by reducing your opponent's number of legal moves to zero (i.e by blocking). But we will just ignore that for now.
- A solution is guaranteed to exist.

Programming - Baselines

- Baseline 1 - 25%
- Baseline 2 - 30%
- Baseline 3 - 25%
- SuperHard Bonus - 10%

The private test cases should be no harder than the hardest public ones.

*to the extent (not far) that we can ensure such claims

Programming - Wakasagi

- Your code will play the chess game through the trusty referee program **Wakasagihime**.
- The main file you'll be editing is "solver.cpp"

This slide will go through the basic usage of the engine,

for a more detailed guide, check out the documentation here.

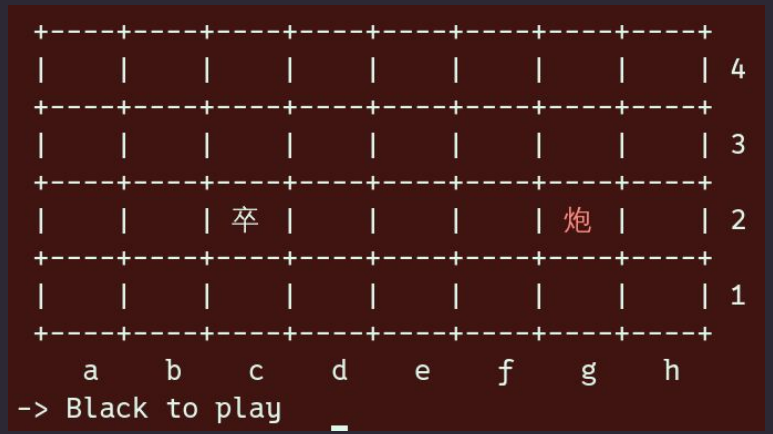
Wakasagi - compilation note

- Only gcc is supported
  - We use intrinsics here
- Feel free to ask if something doesn’t work

Wakasagi - Input & Output format

- Wakasagi can take a FEN-like string as input
  - “8/2P3c1/8/8 b”
  - 4 ranks, separated by a slash ‘/’, space, then the side to play (always ‘b’ in HW1)
  - The first part is rank 1, which is displayed at the bottom
- Output
  - Time taken (in seconds)
  - Number of steps
  - All moves, in order

SQ_A1 = 0, SQ_H4 = 31



Wakasagi - Data Structures

- **Piece**

  Contains a Color and a PieceType.
- **Square**

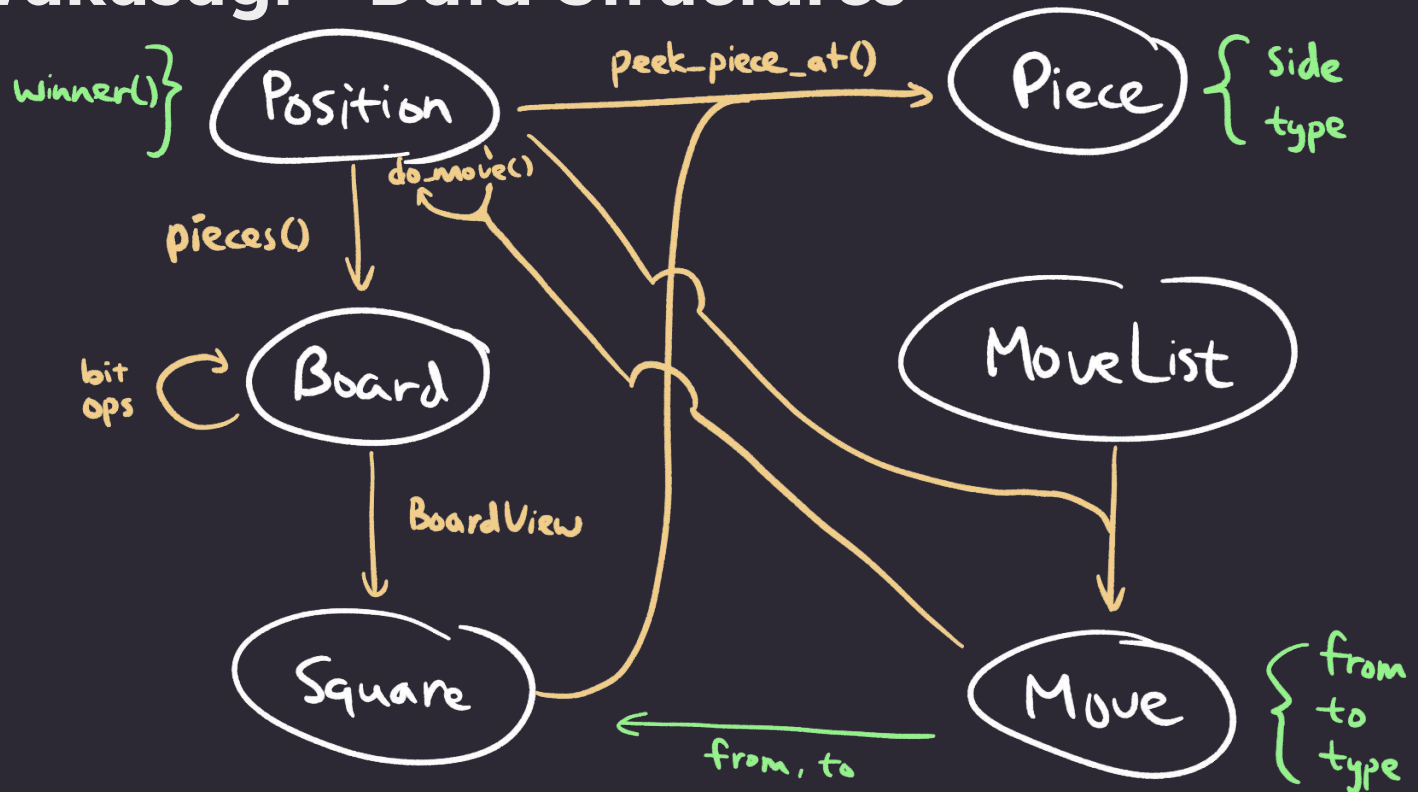
  Enum 0~31. Can be acquired from Boards.
- **Board & BoardView**

  32-bit bitboards acquired from Positions and can yield Squares via BoardView.
- **Position**

  Contains Boards for each PieceType and Color, along with other information about a position.
- **MoveList & Move**

  A list of Moves generated from a Position.

Wakasagi - Data Structures



Wakasagi - Position

- Use the pieces() function to get the Board for some pieces
  - pos.pieces(Red)
  - pos.pieces(Black, Chariot)
- peek_piece_at(Square) to get full info on a piece at a Square
- do_move(Move &) to apply a move
- winner() to check for game endings
  - Check (pos.winner() == Black)

Wakasagi - Board

- 32-bit integers
  - One bit for each square
  - Bit high (1): this square is occupied by *something*
- BoardView provides an iterator of set bits (occupied Squares)
  - for (Square sq: BoardView(board)) { /* iterate Squares */ }
  - You can also use the BoardView::to_vector() function for a mutable vector of Squares
- You can also modify Boards directly with bit operations
  - Use ~ to invert a Board (probably the most useful one)
  - See lib/chess.h.

Wakasagi - Moves

- Generate all legal moves for black!
  - MoveList<All, Black> moves(pos)
  - MoveList moves(pos) works just fine, too
  - Or only the horse moves: MoveList moves(pos, Horse)
- MoveList holds all the Moves
  - for (Move mv: moves) { pos.do_move(mv) } // Iterate like this!
  - Implements standard iterator - pretty handy
- To output the moves for your final answer, simply stream it to stdout
  - info << move;

Wakasagi - useful things

- Check if a piece type A can capture piece type B
  - Just use the greater than ‘>’ operator on the PieceTypes

    (it’s called “greater than,” but actually equal ranks returns true as well)
- distance<Square/File/Rank>(a, b)
  - Manhattan distance, doesn’t take into account blocking pieces
- You can run <algorithms> on a MoveList
  - Make a custom comparator and use std::sort()

Grading criteria - timing

“The clock is the 33rd chess piece.”

- The time limit is **10 seconds** for each test case
- You must output the time (in seconds) taken to calculate your answer
- The validator will also time your code
  - No points will be awarded if your timing is off by more than 0.1s
  - Correct timing grants a minimum of 0.5 points, even if you fail to deliver an answer

  *we time your program until it exits
- Remember to set a timeout
  - And leave some margin!

Grading criteria - suboptimal answer

We don’t need to be perfect, just enough to best your opponent.

- One extra step above optimal: 0.5% penalty
- Two or more extra steps above optimal: 0 points
    - But you can still get the 0.5% from the time

Using the Judge

- Usage: python validator/eiki.py
  - Add `--refresh` to recompile your updated code (it takes a while)

```
♣ validator ❯❯❯ python eiki.py
===== Judgement =====
1-0 - Timed @ 0.00s
★ Well done!
1-2 - Timed @ 0.00s
★ Well done!
1-3 - Timed @ 0.01s
❌ FAIL - Mistimed (Claimed 10.0s)
2-2 - Timed @ 1.90s
❌ FAIL - Mistimed (Claimed 1900.0s)
2-3 - Timed @ 5.62s
❌ FAIL - Mistimed (Claimed 5615.0s)
```

Judges your outputs

Judges your pitches

Judges you

Program rules

- Your code should run on the CSIE workstations.
- **You get one (1) thread.** No parallelism, forking, threading.
- No pragmas or any other similar gcc witchery.
  - We reserve the right to witch hunt.
- Memory limit: 10MiB (virtual address space)
- We will not compile your code if there is any warnings.
- **Do not edit:**
  - lib/*
  - makefile (note: put additional sources in sources.mk instead)
  - wakasagihime.cpp

Small tips

- No advanced data structures is needed or even recommended
- Please don’t use a self-balancing tree, it’s not worth it
  - But we won’t stop you if you insist...
- Just try to come up with a better heuristic, it’ll make all the difference

Written part - CTF

- This small capture-the-flag activity will give you a reason to learn to use a debugger & profiler.
- We recommend you do this part before tackling the programming part.
    - It takes like 5 minutes to complete

Debugger & profiler

- Debugger: gdb (compile with -g)
  - Print / examine a variable or memory address: p / x
  - Set a breakpoint at a line or a function: break (b)
    - Pauses when execution reaches that point
  - Set a watchpoint at a variable or memory address: watch
    - Pauses when the variable or memory content is changed
- Profiler: gprof (compile with -pg)
  - See how many times a function is run and how long it took
  - Run the program once to generate a profile, which you can check with gprof

CTF instructions

- **You should not modify the code**, simply use gdb and gprof to find the answers.
- Provide screenshots of gdb and gprof
- twin_prime.c
  - How many times is the function “is_prime()” executed? (gprof)
  - What is the 42nd pair of twin primes? (gdb)
  - How many pairs of positive twin primes below 23614847 are there? (gdb)
  - (Bonus) Prove or disprove that there are infinitely many twin primes.

Written part - Report

Your report should contain the following:

- Algorithm (10%)
  - What algorithm did you implement and how does it work?
- Heuristic (10%)
  - What is your heuristic and how does it work? (5%)
  - A proof that your heuristic is admissible. (5%)

Early Bird Bonus

- Submit your program before the early bird deadline for a flat 5% bonus
  - Only the programming part!
- You will receive your grades at the early bird deadline
- You can still make changes and submit after the early bird deadline
  - New submissions after the early bird deadline are not eligible for the 5% bonus
  - Your grade will be the higher of the early/regular submissions
- Total grade will not exceed 100%.
  - Extra points do not carry over to the next assignment.

Late policy

- Your submission time is server-sided, **do not submit at the last second**
- Each late day incurs a 0.9x penalty
  - Rounded up to the nearest day
  - 1 second of delay counts as a full day
- Maximum of 7 days of delay accepted

Submission

- Submit a zip file containing
  - report.pdf
  - code/
    - solver.h and solver.cpp
    - sources.mk
    - Any other source or header file you may need
- All code files will be placed in the same directory
- Submission link
  - www.csie.ntu.edu.tw/~tcg/2025/hw1
  - We will send your password (used for submissions) to your student ID email