Course Project Demo: Isolation

Gao Fangshu 高方澍
Li Jiawei 李佳蔚

Overview

- Implement a solver and GUI for game Isolation in 4*4, 4*5 and 5*5.
- For human vs. human, you can play on board with any size you want.

Language and Tool

- Both use Python and C++, Python for GUI and C++ for solver.
- Use Visual Studio for C++ and PyCharm for Python.
- Developed in Windows, but also use a Linux server to run the solver.

Division of Work

- Gao Fangshu was in charge of GUI.
- Li Jiawei was in charge of solver.

- Both person were in charge of testing and analysis.

Play Time

- Anybody wants to try play with our computer (or human vs. human)?

About Solver

- Upper bound for the # of the positions:
  - For a N*M board: $2^{N*M} * (N * M)^2$
  - 4*4: 16777216 (16.8M)
  - 4*5: 419430400 (420M)
  - 5*5: 20971520000 (21G)
- Actual # of positions:
  - 4*4: 3617229 (3.6M)
  - 4*5: 97976968 (98M)
  - 5*5: 4967156731 (5.0G)
- The actual # of positions is about 4 times smaller than the upper bound.
- Both of them are very big!

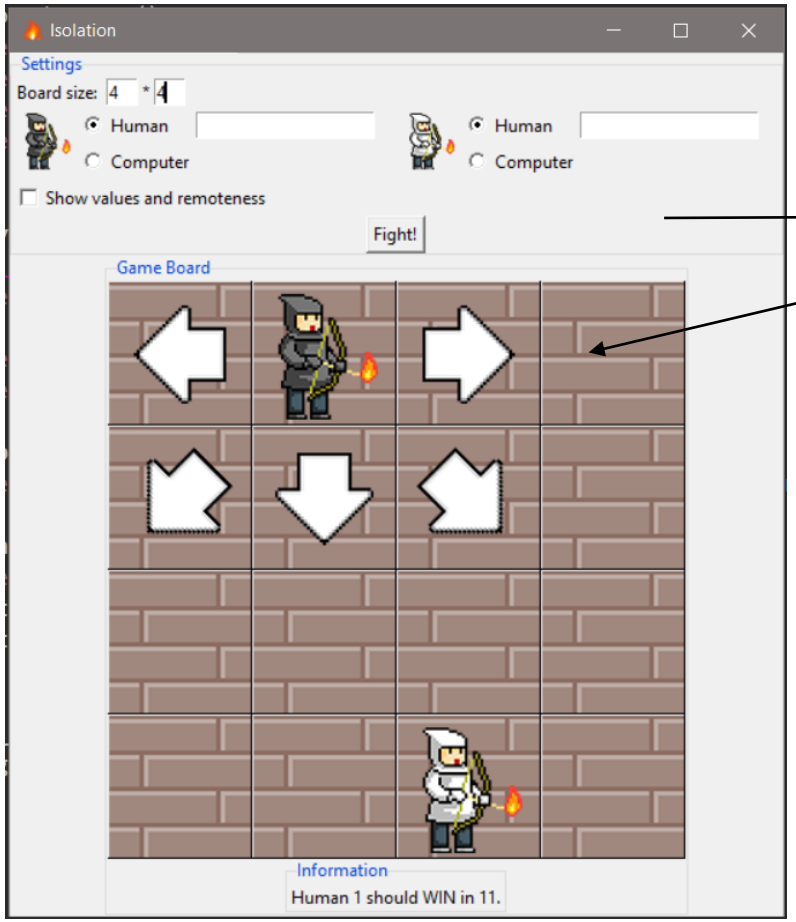
About Solver

- It’s **impossible** for Python to solve 5*5 case in one day.
- Use a very delicate way to hash the position and the value/remoteness pair, so that the memory usage and the size of the database file is same as the upper bound.
- Time for running the solver:
  - 4*4: 3.938s
  - 4*5: 106.075s
  - 5*5: 114m 57.963s
- The database file is really big, so a fast C++ reader is also needed.

Difficult Problems in Solver

- The #actual positions for 5*5 was too big, the algorithm must be efficient both in time and space.
- Difficult to read big database file. The file size even exceeds the range of 32bit integer, a special function "`_fseeki64`" can solve this problem under MSVC, but we still don't know how to do in other environment.
- Several wrong way during coding:
  - Use JSON to store the database. JSON is not suitable for big data.
  - Use unordered_map(a automatic hash container like dict in Python) to hash position.
- Hard to debug
- Many overflow in 32-bits integers…

About GUI



orders etc.
(size, player, predict or not, move/delete)

class GameGUI
- Settings
- class GameBoard
  - class Board
  - class Information

class Env
- class Game
- other dynamic variables (current_position, turn, etc.)

feedback

value
remoteness

position

Database
C++ .exe file

Difficult Problems in GUI

- It's complicated to deliver data and command among classes.
- Computer can play two steps (move, delete) each time, but human only plays one step by clicking mouse. Controlling this in GUI/game environment can easily cause bugs. (We still have bugs in Computer vs. Computer).

Analysis

- On 4*4 board:
  - #Upper bound is 16777216, #Actual position is 3617229
  - Start position's value is 0(WIN), remoteness is 11, smaller than longest possible step 14.
  - #Primitive position is 231480
  - $\#WIN = 2737177, \#LOSE = 648572$, $\frac{\#WIN}{\#LOSE}$= 4.22, which means that go first has great advantages.
  - #Average children for all position is 17.246, which is not a small number.
- On 5*5 board:
  - #Upper bound is 20971520000, #Actual position is 4967156731
  - Start position's value is 0(WIN), remoteness is 13, only 2 steps slower than 4*4 case.
  - #Primitive position is 213427647
  - $\#WIN = 3712154601$, $\#LOSE = 1041574483$, $\frac{\#WIN}{\#LOSE} = 3.56$, go first's advantages shrank a little, maybe because you can't kill opponent easily in a larger board
  - #Average children is 32.788

How to get our game

- *https://github.com/GaoFangshu/solve-games/tree/master/final_project*
- If you only play Human vs. Human, python files are enough. (just run GameGUI.py)
- If you want to play with computer or need prediction, run C++ first to get database.

Our to-do list

- Speed up the Python code when make prediction or generate best move
- Finish report
- Debug