chesh

february 25, 2020
nprg035
jooh@cuni.cz

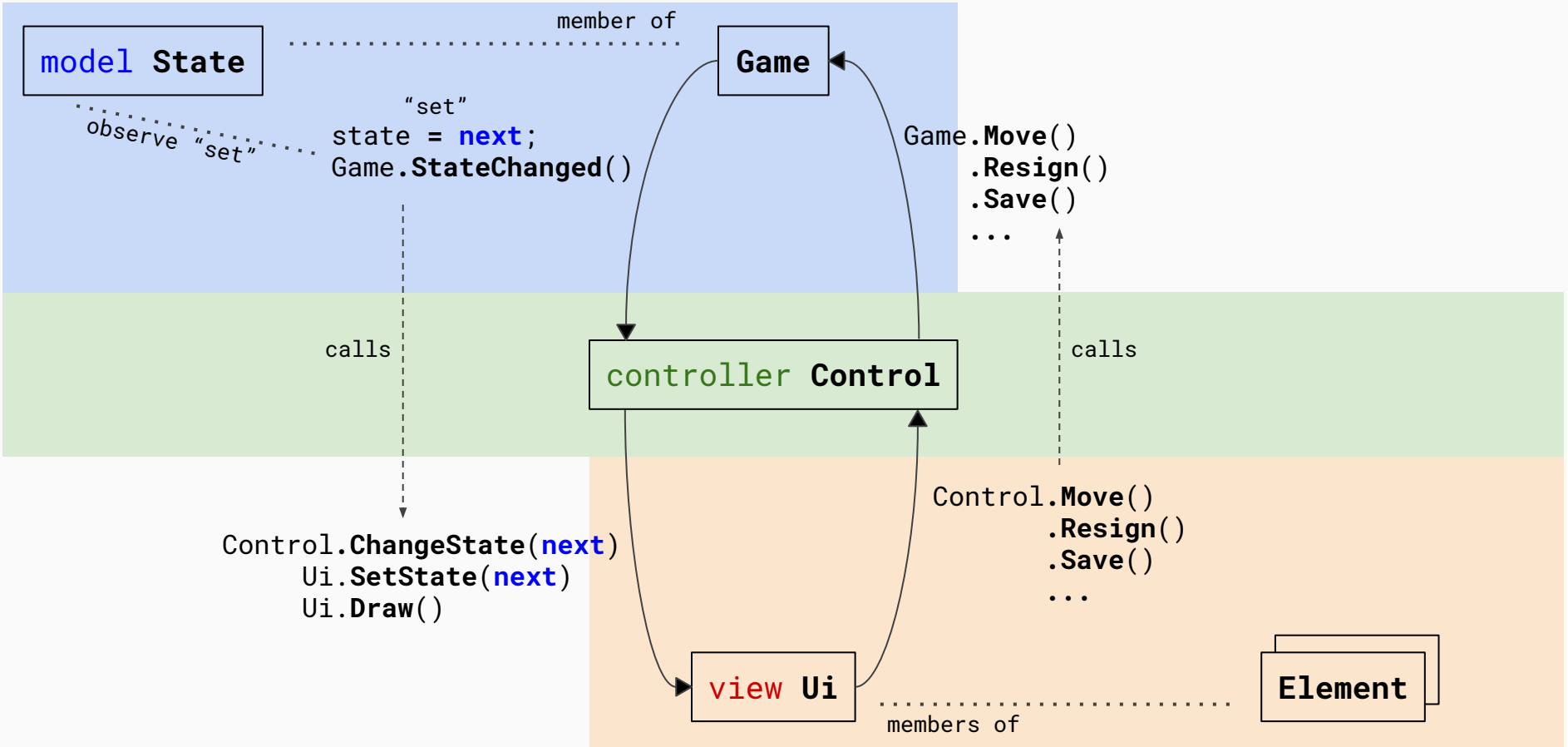
```
b>
 1. Pe2e4   Pe7e5   x N P N R B Q                    x
 2. Ng1f3   Pd7d6
 3. Pd2d4   Bc8g4       a   b   c   d   e   f   g   h
 4. Pd4e5:  Bg4f3:    .---.---.---.---.---.---.---.---.
 5. Qd1f3:  Pd6e5:  8 |   | n |   | R | k | b |   | r | 8
 6. Bf1c4   Ng8f6     |---+---+---+---+---+---+---+---|
 7. Qf3b3   Qd8e7   7 | o |   |   |   |   | o | o | o | 7
 8. Nb1c3   Pc7c6     |---+---+---+---+---+---+---+---|
 9. Bc1g5   Pb7b5   6 |   |   |   |   | q |   |   |   | 6
10. Nc3b5:  Pc6b5:    |---+---+---+---+---+---+---+---|
11. Bc4b5*  Nb8d7   5 |   |   |   |   | o |   | B |   | 5
12. Ke1c1%  Ra8d8     |---+---+---+---+---+---+---+---|
13. Rd1d7:  Rd8d7:  4 |   |   |   |   | P |   |   |   | 4
14. Rh1d1   Qe7e6     |---+---+---+---+---+---+---+---|
15. Bb5d7*  Nf6d7:  3 |   |   |   |   |   |   |   |   | 3
16. Qb3b8+  Nd7b8:    |---+---+---+---+---+---+---+---|
17. Rd1d8#          2 | P | P | P |   |   | P | P | P | 2
                      |---+---+---+---+---+---+---+---|
                    1 |   |   | K |   |   |   |   |   | 1
                      '---'---'---'---'---'---'---'---'
                        a   b   c   d   e   f   g   h

                    x o b o o n r                   x
w>
Checkmate. White wins!
```

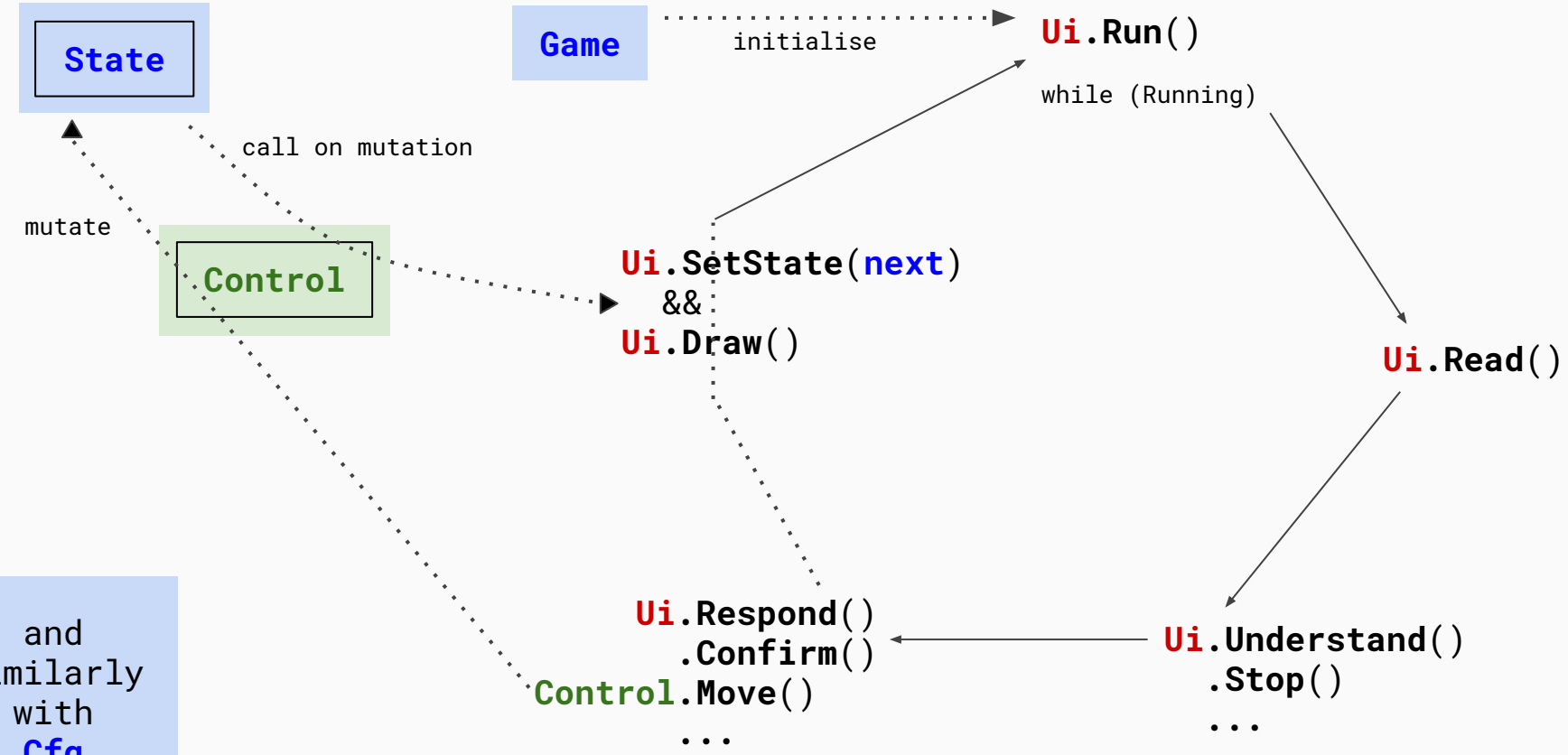
the Opera Game: Morphy vs. Karl & Isouard, 1858

- Basic subset of the game of Chess
- Implemented in C# .NET Core
- Windows/Linux compatible
- Console-based
- 2 Human players

MVC pattern, Observer pattern



User interface



Model-view “move” contract

Model		View
• **Assume** input square is within bounds. • Comprehend move (ie. determine good or bad).	⟷	• Determine if input square is within bounds.
• Change State only if move is good.	⟷	• **Can assume** State changes only on good moves. ◦ But can **ignore** this because view simply reflects State.
• **Assume** .Move() called with <int> coordinates.	⟷	• Call .Move() with coordinates as <int>s.

Data interchange format: JSON

State

Piece **Selection**

List<Piece> **Live**

List<Piece> **Dead**

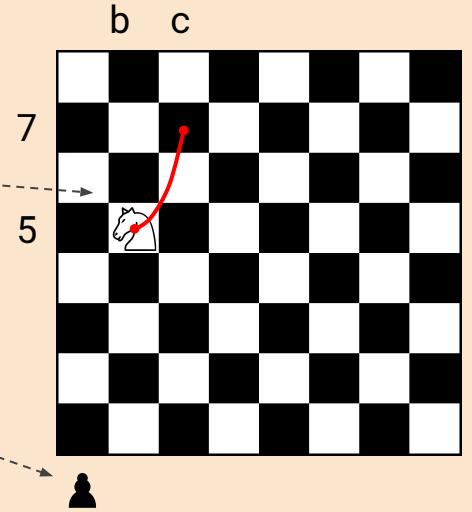
List<(string,long)>
History

json string

```
{
  Selection: <Piece>,
  Live: [
    <Piece>
    ...
  ],
  Dead: [
    <Piece>
    ...
  ],
  History: [
    <(string,long)>
    ...
  ]
}
```

unimplemented

Ui

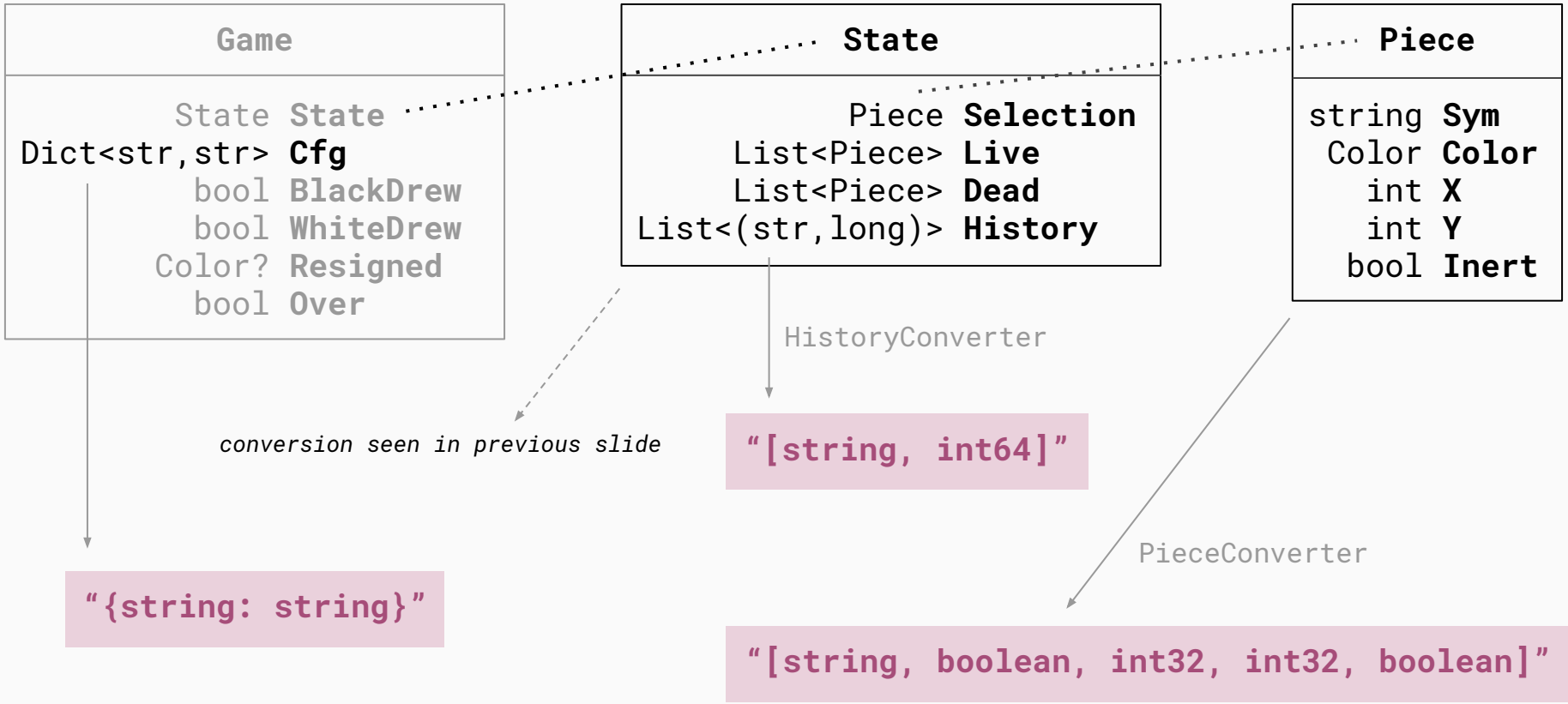


Nc7b5

Helper.ToJson()

Helper.FromJson*()

(Model) Classes, JSON representation



Review: a sample move sequence

User enters “b2 b4” into game prompt

Ui does **.Move**(2, 2, 2, 4)
And calls **Control.Move**(2, 2, 2, 4)

Control calls **Game.Move**(2, 2, 2, 4)

Game tries to move Piece from b2 to b4
(Assume success)

Game mutates State

Control detects State mutation

The direct avenue of function returning is used to pass the move’s **Ret** value to **Ui.Move**()

Control sets Ui’s state-json-string
And calls **Ui.Draw**()

Finish **Ui.Move**() with appropriate **.Respond**()

Notation, save/load

Pc2c4+

piece type	source	destination	move type
[PRNBQK]	[a-h][1-8]	[a-h][1-8]	[&*#+:p%RNBQ]+

save

- Opt to save on quit (to “chesh.log”)

load

- Pass filename as argument to Chesh.exe
- Ignores whitespace

Move annotation (yet another algebraic notation ...)

mutually exclusive
castling AND capturing
castling AND promoting
castling AND enpassant
enpassant AND capturing
enpassant AND promoting
checking AND checkmating

	: cap	+ chk	# cmt
[RNBQ]=@ promote	@:	@+	@#
% castle		%+	%#
p en passant		**p**+	**p**#
: capture		*	&

	: AND +	: AND #
[RNBQ]=@ promote	@*	@&

Examples

A Pawn's regular move.

Pa2a4

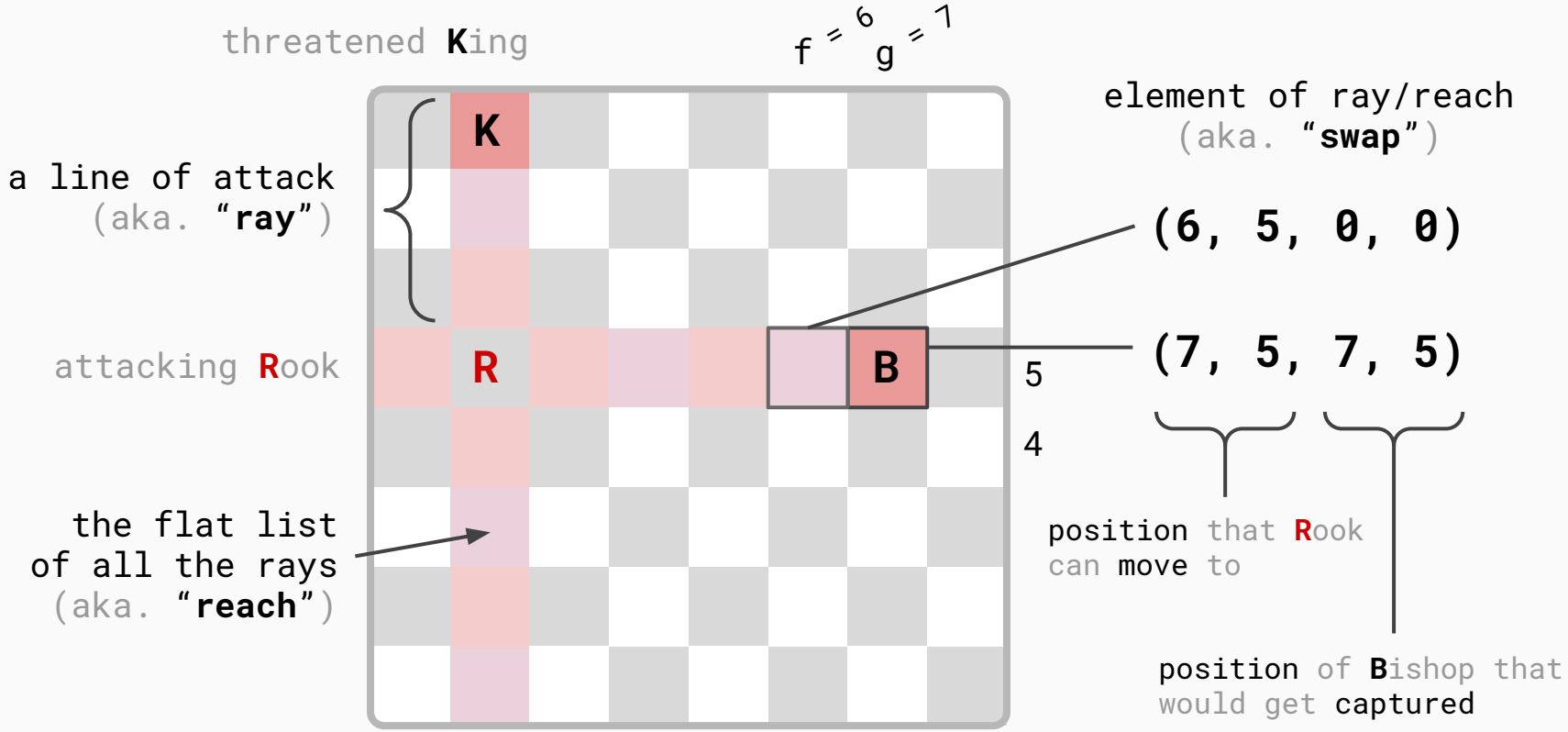
White's queenside castling.

Ke1c1%

A Pawn gets promoted to a Queen while simultaneously capturing an enemy and checkmating their King:

Pg7f8Q*

Attack


threatened King
f = 6
g = 7
a line of attack
(aka. "ray")
attacking Rook
the flat list
of all the rays
(aka. "reach")
K
R
B
5
4
element of ray/reach
(aka. "swap")
(6, 5, 0, 0)
(7, 5, 7, 5)
position that Rook
can move to
position of Bishop that
would get captured

Implemented functionality

Game mechanics

- Basic movement of pieces
  - Castling
- Capturing
  - En passant
- Promotion
- Check detection
- Checkmate detection
- Tie game
- Resign game
- Reset game
- Undo move

User interface

- Player input prompts
- Sub-prompts for responses and confirmations
- ASCII board
- Captured pieces
- History of moves
- Saving/loading
  - Playback
- Game menu
- Basic configuration

hindsight

More user friendly:

- Typing in the coordinates is not very user-friendly.
- Selecting, moving, possibility viewing, should all be done responsively, using arrow, tab, and enter keys.
- Use colors and/or unicode, at least.

More extensible UI structure:

- Elements need a more opaque and comprehensive interface with the main UI loop/control.
- The UI itself should be MVC.

More network-ready:

- Not so much calculation need be done by the “server”. The “client” (ie. view) is also capable of calculating move legality, detecting check, etc.
- Make the server minimal and the client heavy.

demo

Source:

https://github.com/agarick/mff/tree/master/chesh