Implementing a 2-Player Strategy Game

Note Title

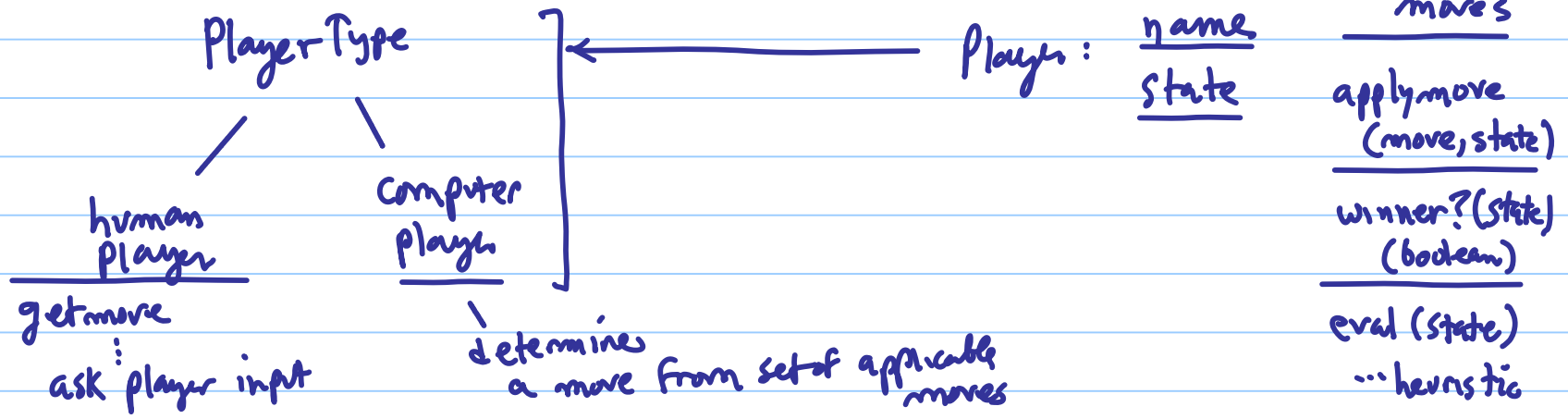
PlayerType = {human, computer}

Either player can go first

allows human vs. human

or

human vs. computer



validate (is this a legal move?)

IDEA: give player a list of legal moves as a menu (sometimes impractical)

get move

Minimax, Level 0

Select a move at random from set of legal moves

Minimax, Level 1

Select a move by applying eval f'n to set of legal moves

Minimax, Level $k \geq 2$

Select a move by applying eval f'n to a k-move lookahead (including opponent's potential moves)

allow α-β pruning =

include:

performance metrics —

# of nodes expanded

# of calls to eval with and without pruning

timer — to monitor actual amt. of time used by computer player

use iterative deepening

Start

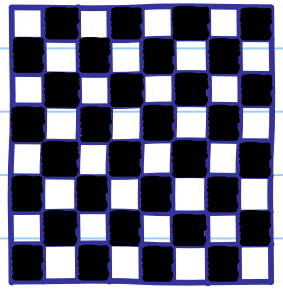
Name of player 1
Type of player 1 (human, computer)
Name of player 2
Type of player 2 (human, computer)
Which player goes first? (1 or 2)

State representation ⟶ might want to compress state so can save more in memory and then unpack to apply eval f'n

Rule/Move representation →
precondition (state, rule, player)
applicableMoves (player, state) =
applymove (state, rule, player)

↓

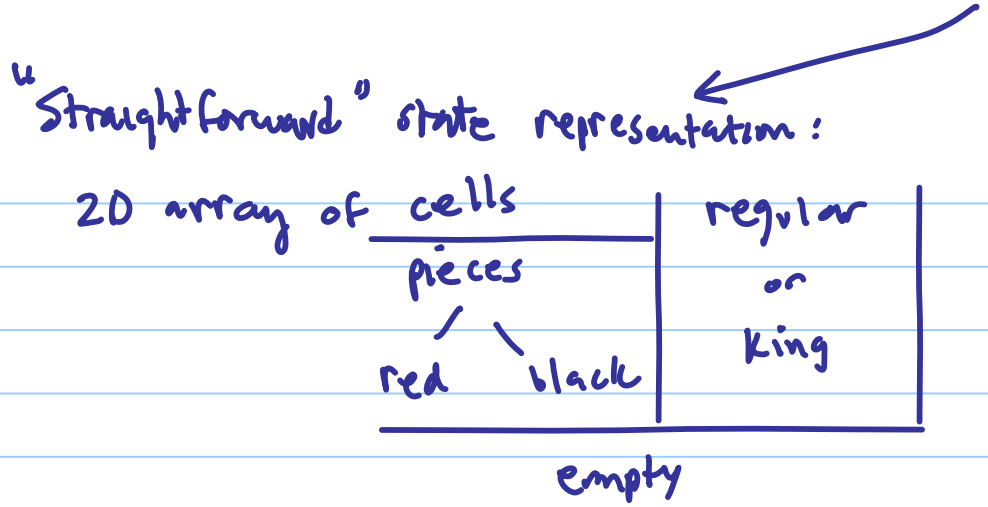
e.g. checkers



64 squares

only 32 on which pieces move

"Straightforward" state representation:

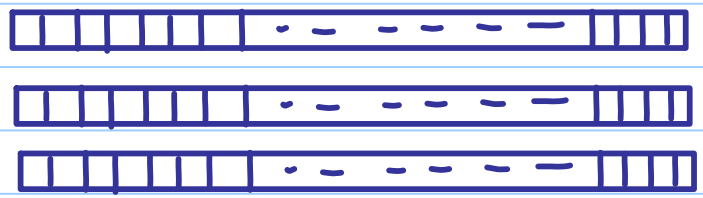


8x8 array:
of {empty, King red, King black, regular red, regular black}
unused ("other" 32 squares)

"Compact" Representation

3 32-bit words
1 bit per usable cell:

every cell's value can be determined from 3 bits



IDEA: store states in compact form
"unpack" into "straightforward" form
to do | evaluation
| move generation
| etc

Use of compact representation allows some info to be cached (kept in memory) for fast lookup -