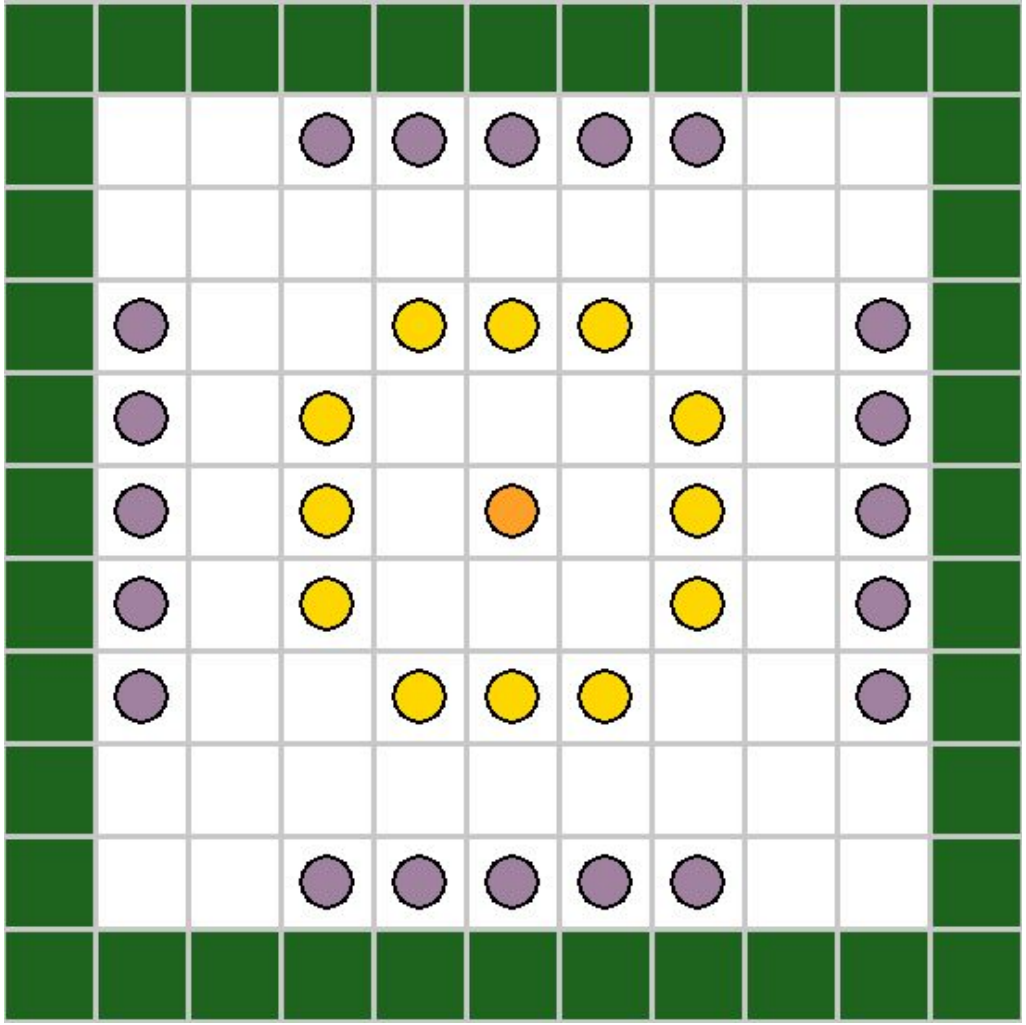
Breakthru

Dag Grinberg

Breakthru

- Game implementation
- Game AI



Game Implementation

- Python, pygame - GUI

Classes:

- Grid > provides layout of the pieces, can easily modify the initial position
- Piece > represents pieces
- Board > manages piece position and actions
- Turn > manages actions, keeps turn stats (moves, captures, last piece moved, etc.)
- Game > coordinates player input, keeps history, game stats

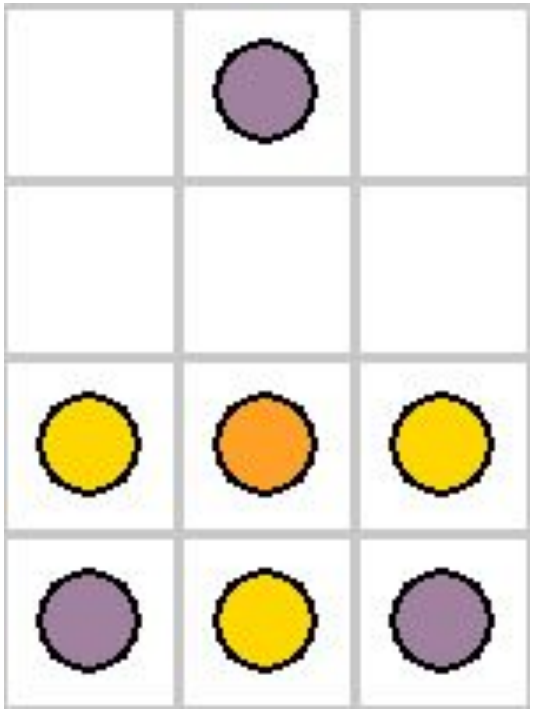
Game State Representation

- actions = moves + captures
- 1 turn = 2 moves or 1 capture or 1 flagship (king) move
- 1 ply = 1 action
  - See further, than combining 2 moves into one ply
  - More expensive
- 1 turn = 1 or 2 plies!

Evaluation function

Heuristic evaluation, gold maximises

- Winning conditions
- Pieces = (Gold - Silver)
- King surrounded
- King distance from edge
- King mobility
- King exposed



MiniMax + AlphaBeta pruning

Fixed depth search

- 1 ply ~ 0.2s
- 2ply ~ 20s
- 3ply ~ long time

NegaMax + AlphaBeta pruning

1 ply = 1 action, the turn doesn’t change every ply!

Solution > a flag

- If Gold player > don’t reverse the values
- If Silver player > reverse the values
- Reverse alpha beta values ONLY if the player of the child node is opposite of the current player

NegaMax + AlphaBeta pruning

Fixed depth search + Zobrist Hashing + TT + Heuristic Move ordering

- 1 ply ~ 0.2s
- 2ply ~ 20s
- 3ply ~ long time

Iterative Deepening

Use the values calculated in previous search to sort child nodes for the next search

Implementation not stable :(

Resume

- Basic MiniMax and NegaMax with AlphaBeta pruning works yet is slow
- ZobristHashing + TT + Simple Heuristic move ordering doesn’t give much improvement
- Could be due to the overhead of the state representation and Python
- Iterative Deepening could provide real improvements by using TT