JOHNS HOPKINS
APPLIED PHYSICS LABORATORY

# Reconnaissance Blind Chess (RBC): A Challenge Problem for Planning and Autonomy

## NVIDIA GTC: March 21, 2019

Casey Richardson, Ph.D.
casey.richardson@jhuapl.edu

JOHNS HOPKINS
APPLIED PHYSICS LABORATORY

ISC INTELLIGENT SYSTEMS CENTER



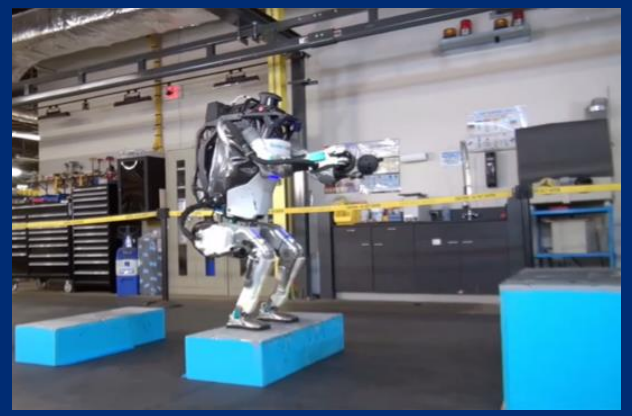
***Inventing the future of intelligent systems for our Nation***

# AI & Decision-Making: *What Intelligent Systems Do*

**Perceive**

**Decide**


**Act**



**Team**

# What Is Reconnaissance Blind Chess (RBC)?

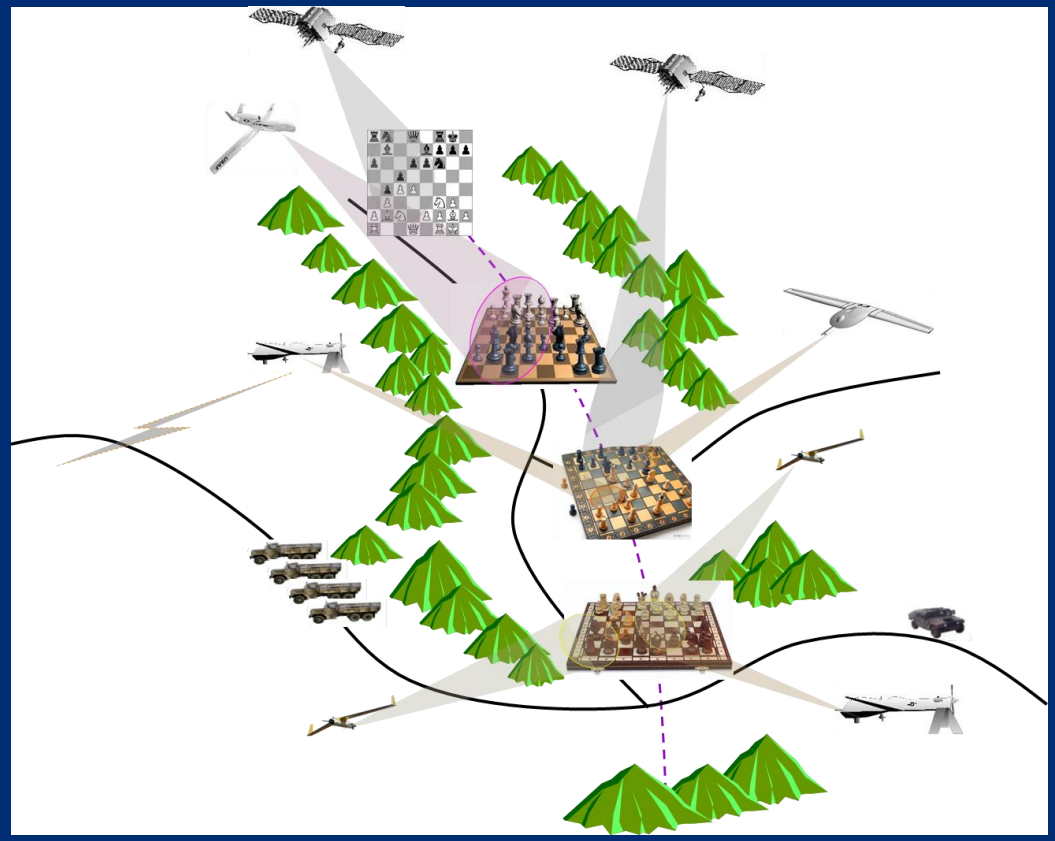
- Chess variant(s) where players:
  - Cannot see the opponent's moves, pieces, or board position (blind)
  - Gain information about the ground truth board through active sensing actions (reconnaissance)

- RBC adds the following elements to standard chess
  - Sensing (potentially noisy and multi-modality)
  - Incomplete information
  - Decision making under uncertainty
  - Joint decision / battle management and sensor management
  - Multiple, simultaneous, competing objectives

# Why Did We Invent RBC?

- RBC is a new reference/challenge problem for research in autonomy and intelligent systems
  - Emphasizes strategic and tactical decision making under uncertainty in a dynamic, adversarial environment

- RBC provides an **accessible challenge problem** in fusion and resource management enabling open collaboration
  - Abstracts and incorporates the many common elements (e.g., value of information) in areas such as
    - Autonomy
    - Intelligence, Surveillance, and Reconnaissance (ISR)
    - Optimal Sensor Tasking for Disaster Relief scenarios
  - There is currently no common, unclassified, open experimentation platform



# Reconnaissance Chess Illustrative Example

**Sensing**

## Controls

white elects to move piece

Start | Back | Forward | End

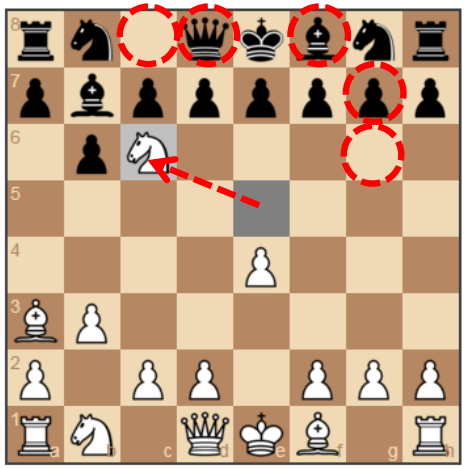
**Game History**

| Move # | W Sense | W Move | B Sense | B Move |
|---|---|---|---|---|
| 1 | b4 | b2-b3 | f3 | b7-b6 |
| 2 | f6 | c1-a3 | d4 | c8-b7 |
| 3 | d6 | g1-f3 | c4 | d8-c8 |
| 4 | b5 | e2-e4 | g4 | g7-g6 |
| 5 | c6 | g1-e5 | c4 | f8-g7 |
| 6 | d6 | e5-c6 | d4 | g7-a1 |
| 7 | e7 | g1-d8 | f3 | a1-d4 |
| 8 | c7 | d1-f3 | g4 | d7-d6 |
| 9 | f7 | d1-f7 | f7 | e8-d7 |
| 10 | e7 | f1-b5 | e7 | c8-d5 |
| 11 | d7 | f1-d7 | | |

## Truth

**Confusion**

## White



## Black

# Chess Variants

## *Historical Context*

- There is a very long history of chess variants: blindfold chess, simultaneous chess (multiple boards), simultaneous blindfold chess, and kriegspiel
  - First recorded game of blindfold chess: Jubair (665–714), Middle East

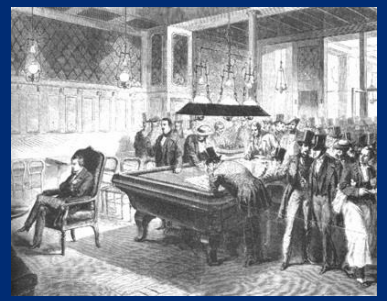
- Blindfold chess and simultaneous chess
  - Tests of memory capacity and cognitive processing power
  - Often used by experts to exhibit their mastery over common players
  - World record simultaneous blindfold chess: Najdorf (1947) versus 45 opponents; 39 wins 2 losses 4 draws

- **Kriegspiel**
  - Game invented in 1812 to train Prussian military officers[1] (translates to *war game*)
  - Inspired invention of chess variant by the same name[2]
  - Test of ability to deal with incomplete information

**This work is motivated by the elements of incomplete information and competing priorities**

**Philidor (1783)**

**Morphy (1858)**



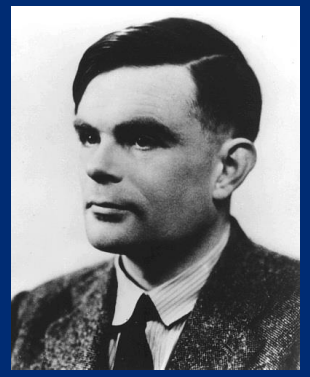
**Alekhine (1925)**

**Modern Blindfold Chess**

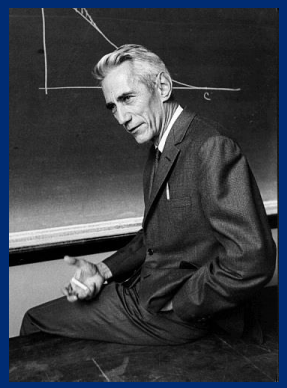
**Kriegspiel**

[1] Georg von Rassewitz (1812)
[2] Henry Temple (1899)

# Computer Chess
## *Historical Context*

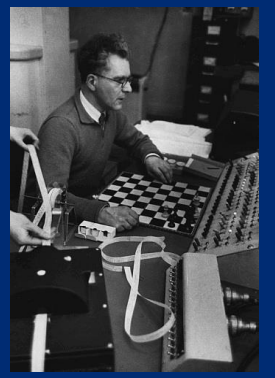


1947: Alan Turing designed first program to play chess (paper & pencil)



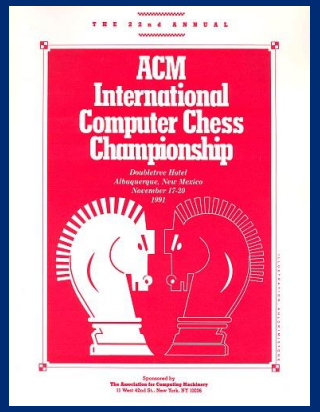
1950: Claude Shannon - relay-based chess machine and groundbreaking paper

Shannon, C. E., "Programming a Computer for Playing Chess," *Philosophical Magazine*, Ser. 7, Vol. 41, No. 314 (March 1950)



1955: Dietrich Prinz - first working chess program

1958: Allen Newell (r) and Herbert Simon (l) developed pioneering algorithms




1970's-1990's: computer chess tournaments and consumer electronics

1997: Deep Blue defeats human world champion Garry Kasparov


2017: AlphaZero defeats Stockfish 10-0 after 4 hours of self-play training

**We hope to stimulate (and expand) an already large community of interest on a new quest to solve a more complex machine intelligence problem**

# Reconnaissance Blind Chess
## *Key Differentiators*

- Blindfold chess: The moves are announced and known to the players (who must keep track mentally)

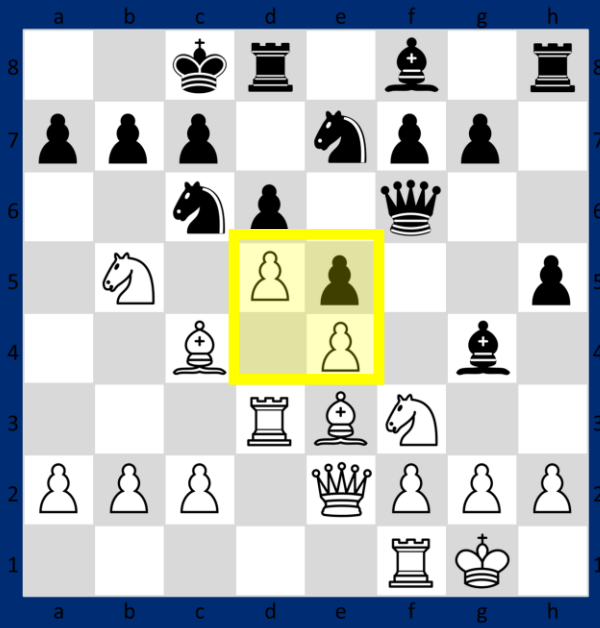
- Blind chess (Kriegspiel): The moves (actions) are covert; each player can see his own pieces but not those of the opponent; players must decide and act with incomplete information

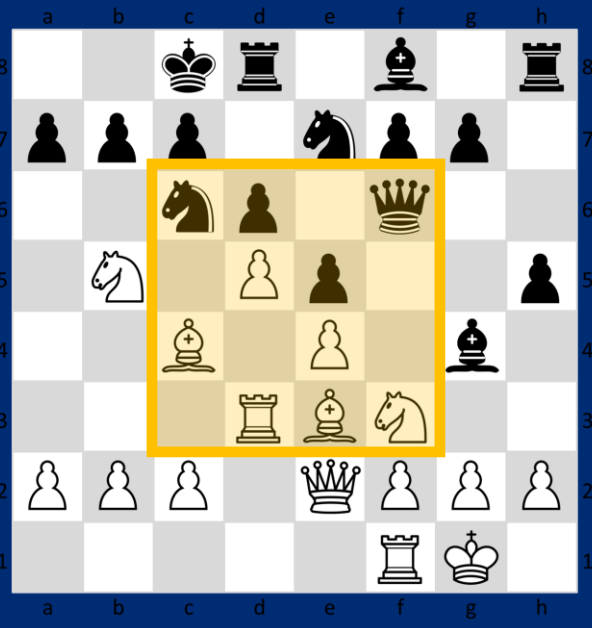
- Reconnaissance blind chess: The moves (actions) are covert; players must acquire and infer all information through "sensor" actions and subsequent inferencing

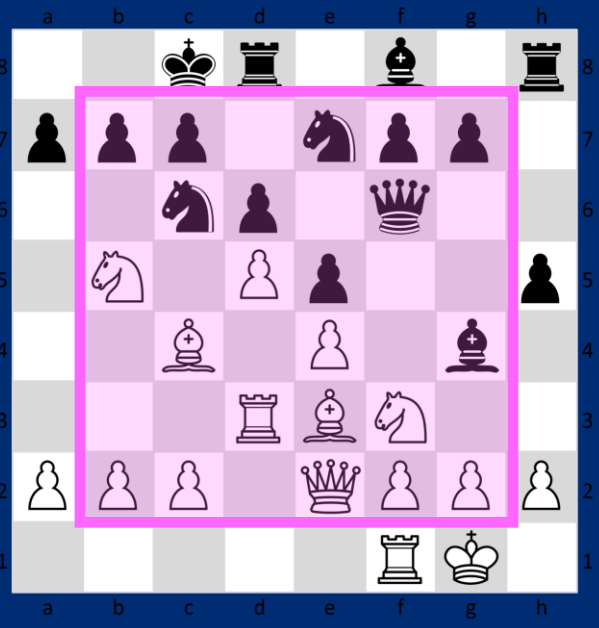

- Reconnaissance blind multi-chess: Players must allocate scarce sensing resources among multiple boards (competing objectives)

**The reconnaissance element is new and the driver of this research problem**

# Sensor Concepts in Reconnaissance Chess







RBC can include typical sources of sensor and processing error; and trade-off coverage against resolution

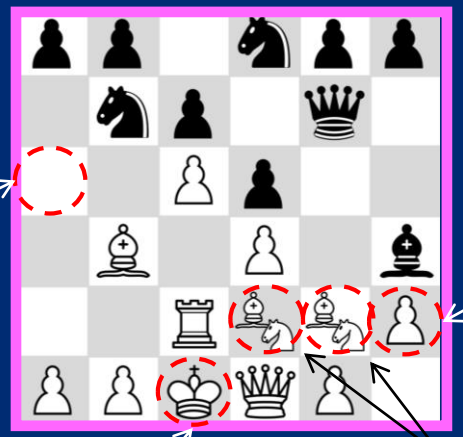
High-Res

**Sensor Output For Three Example Sensing Actions**

Classifier Confusion $P_{recog} < 1$

Missed Detect $P_{detect} < 1$

Medium-Res



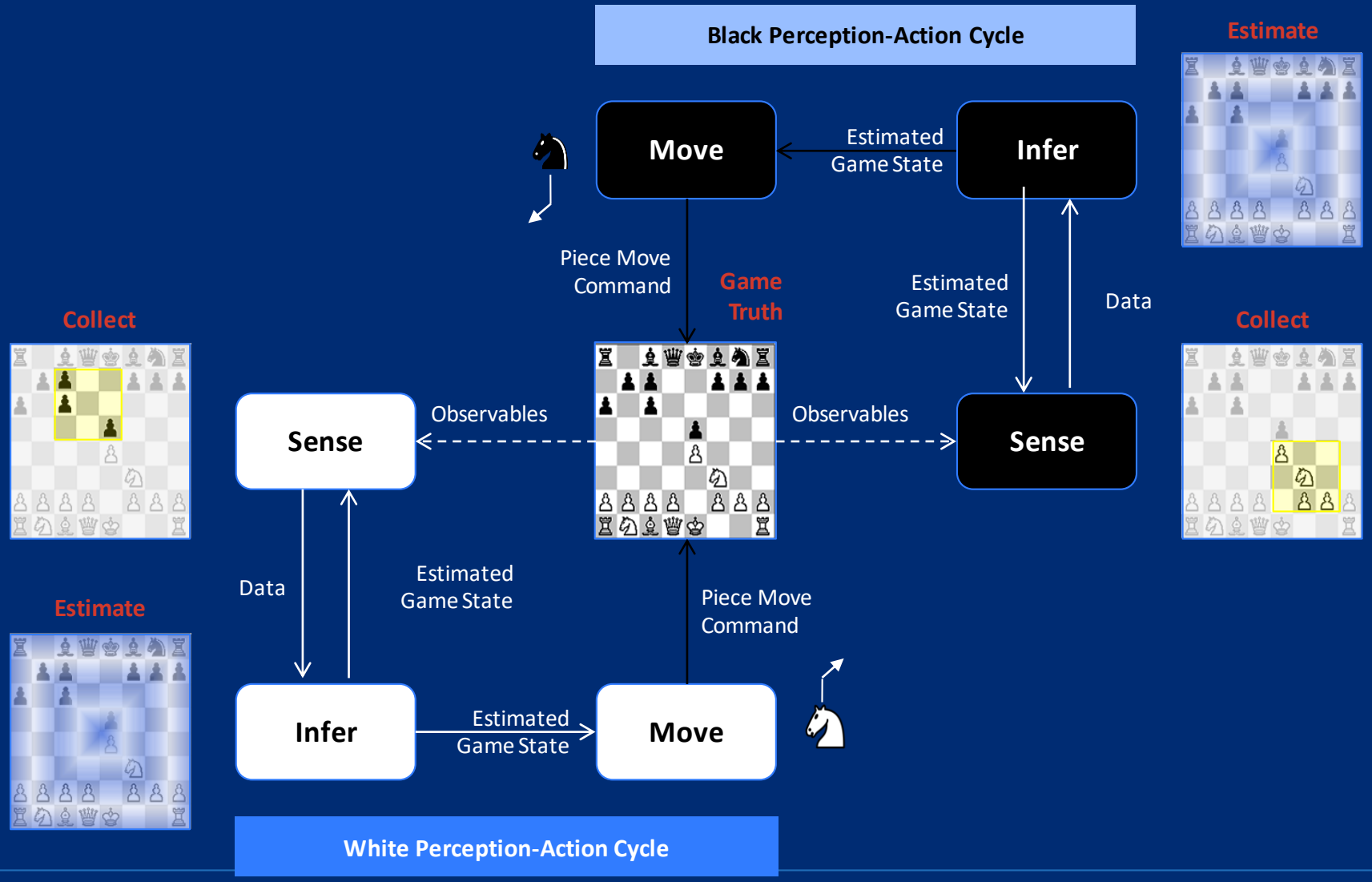
False Alarm $P_{fa} > 0$

Low-Res

Localization Noise

Classifier Confusion $P_{recog} < 1$

# Interacting / Competing Decision Processes (Perception-Action Cycles)



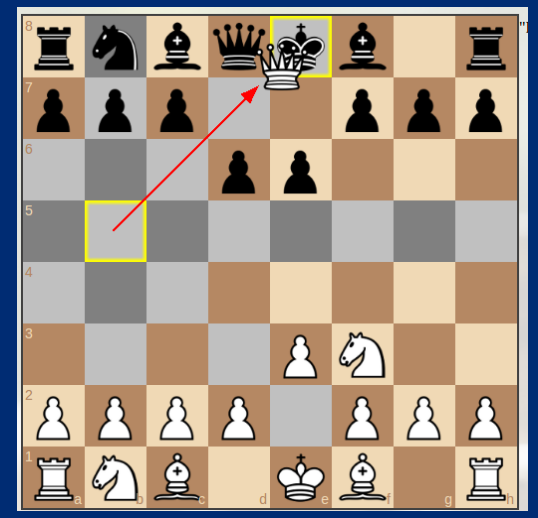
# Reconnaissance Blind Chess: Current Status

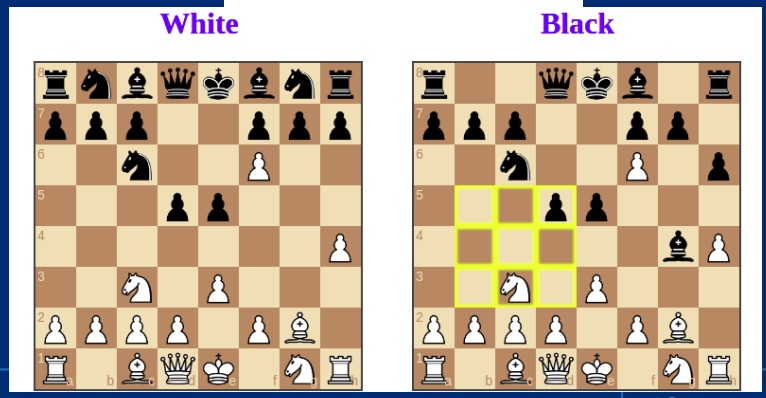
- Defined a basic rule set as foundation for initial implementation (Reconnaissance Chess “1.0”)
- Defined a framework for creating more complex variations as models
- Developed a web based software realization to serve as an experimentation platform
  - http://rbmc.jhuapl.edu
- The following slides will outline JHU/APL R&D using RBC:
  - Development of open source python package with streamlined game core and bot API (on github)
  - Development some initial machine intelligence approaches and exemplar AI-player clients
  - Hosting of internal Autonomy Challenges 2017/2018, APL Intelligent Systems Center (ISC)
  - Research into game complexity analysis
  - Design and implementation of initial concept for human-machine teaming with RBC
  - Development of additional RBC variants

# Rules of Reconnaissance Chess

1. Rules of standard chess apply, with some modifications (below)
2. Objective of the game is to capture the opponent's king
3. There is no check or checkmate, all rules associated with check are eliminated (including with castling)
4. No automatic draws (stalemate, repetition of position, 50 move rule, etc.)
5. Each players turn has turn start phase, sense phase, move phase
   a. Sensor: player chooses a square, ground truth revealed (no error) in 3x3 window around that square
   b. Player is not told where opponent sensed (or vice versa)
6. Players are told:
   a. When their piece is captured (but not the enemy piece that did it!)
   b. When they capture a piece (but not which one!)
   c. When they submit a move that is illegal in ground truth (and they lose a turn)
   d. When their submitted move is modified (and the actual move that was played)

   (above rules allow players to always track their pieces exactly).



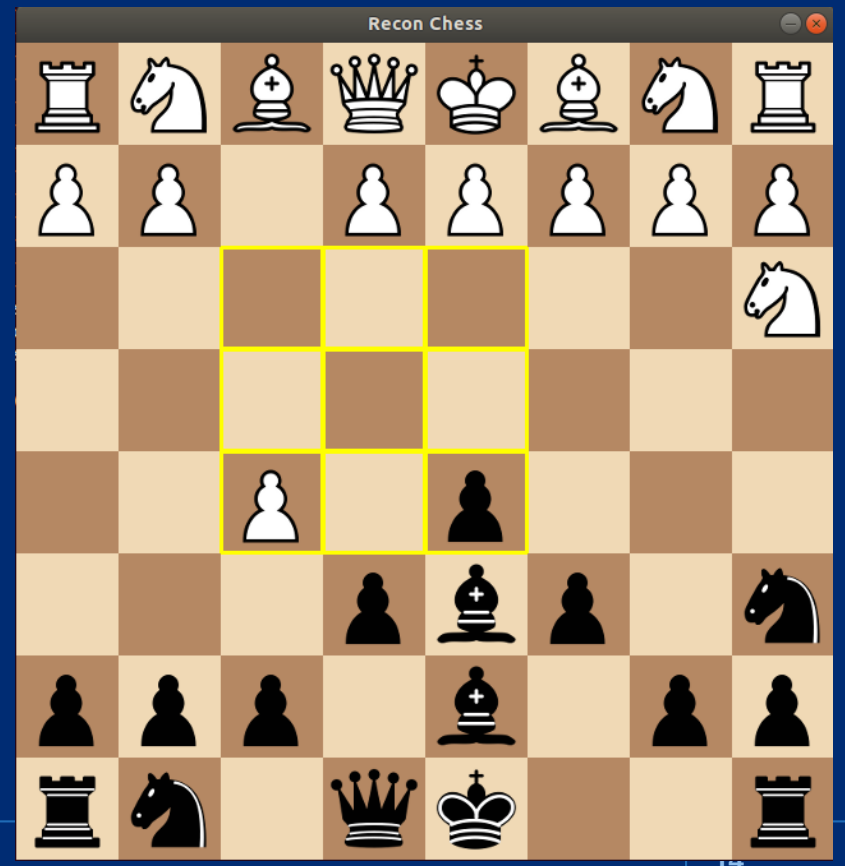



# Open Source Python Package “reconchess”

- We developed an open source python implementation of “Recon Chess” that has
  - An implementation of the game engine / “arbiter” built on the excellent python-chess package
  - A bot player API for developers to easily implement and experiment with RBC bot algorithms
  - Example bots for illustration
  - Simple UI (based on pygame) for playing bots locally (and debugging your implementation), and replaying games

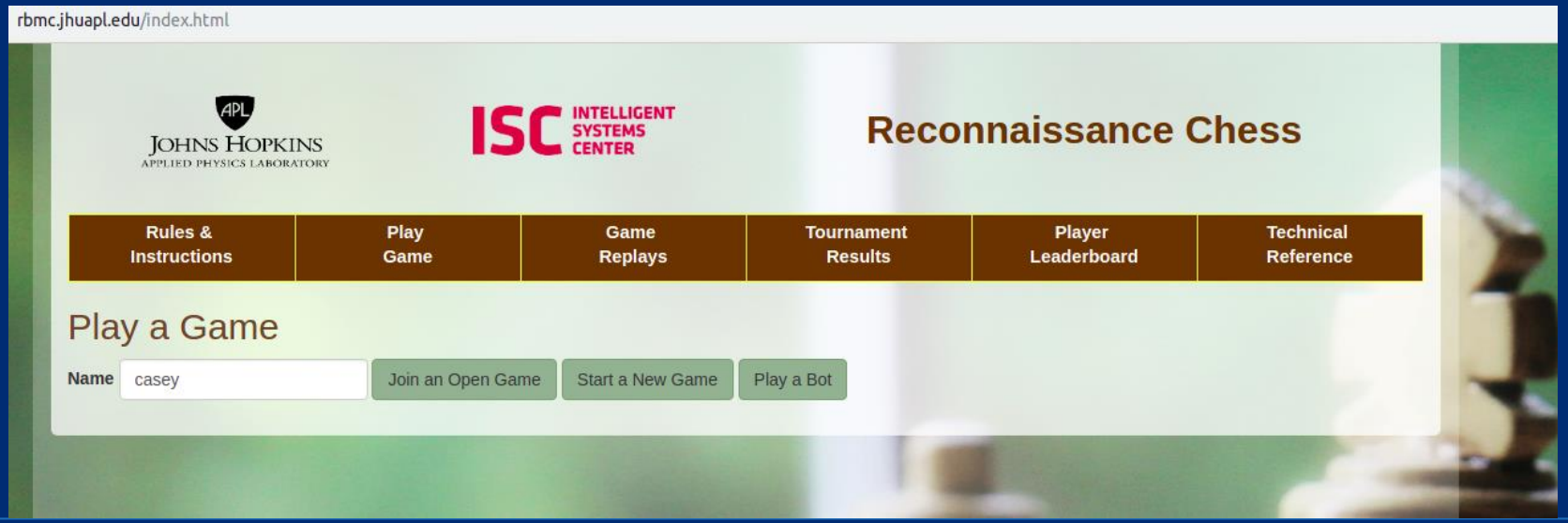
```python
class RandomBot(Player):
    def handle_game_start(self, color: Color, board: chess.Board):
        pass

    def handle_opponent_move_result(self, captured_my_piece: bool, capture_square: Optional[Square]):
        pass

    def choose_sense(self, sense_actions: List[Square], move_actions: List[chess.Move], seconds_left: float) -> Square:
        return random.choice(sense_actions)

    def handle_sense_result(self, sense_result: List[Tuple[Square, Optional[chess.Piece]]]):
        pass

    def choose_move(self, move_actions: List[chess.Move], seconds_left: float) -> Optional[chess.Move]:
        return random.choice(move_actions + [None])

    def handle_move_result(self, requested_move: Optional[chess.Move], taken_move: Optional[chess.Move],
                           captured_opponent_piece: bool, capture_square: Optional[Square]):
        pass

    def handle_game_end(self, winner_color: Optional[Color], win_reason: Optional[WinReason],
                        game_history: GameHistory):
        pass
```

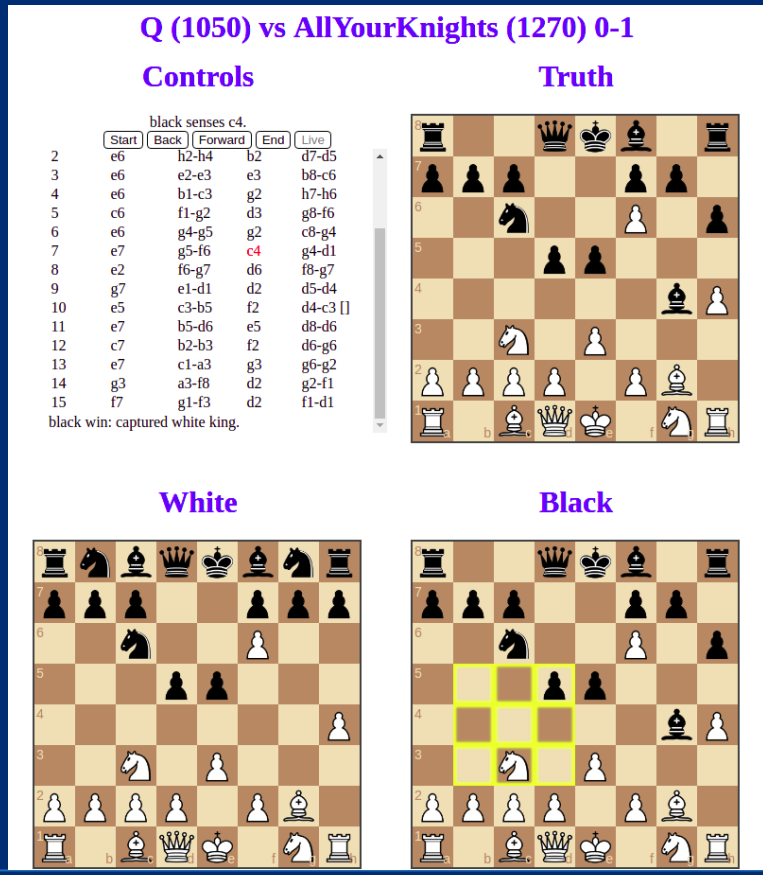
```
rc-bot-match <my_bot_file> rbc.bots.random_bot
rc-replay <game_json_file>
```



# RBC: Getting Started

- Python package available on pypy: “pip install reconchess”
- Getting started and documentation: https://reconchess.readthedocs.io/
- Get the code and bot API: https://github.com/reconnaissanceblindchess/reconchess
- At our website: http://rbmc.jhuapl.edu
  - Play Recon Chess against other humans (a bit clunky) or bots
  - Game rules
  - Replay games (including from internal challenge)
  - Links to papers



# JHU/APL ISC Challenge Overview


- 14 teams from across APL fielded bots
- 1 winning algorithm (“Petrosian”)
- Humans won the man-vs-machine final event 8 to 4 (Jeff & Tom vs. Petrosian)

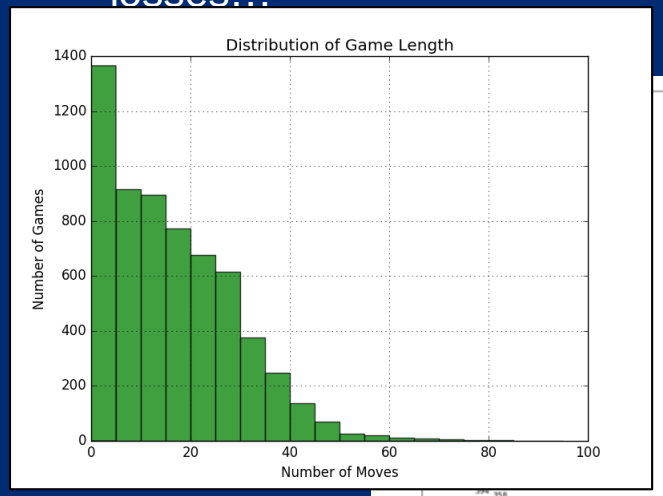

# Win Percentage Cross Table (All Tournaments)

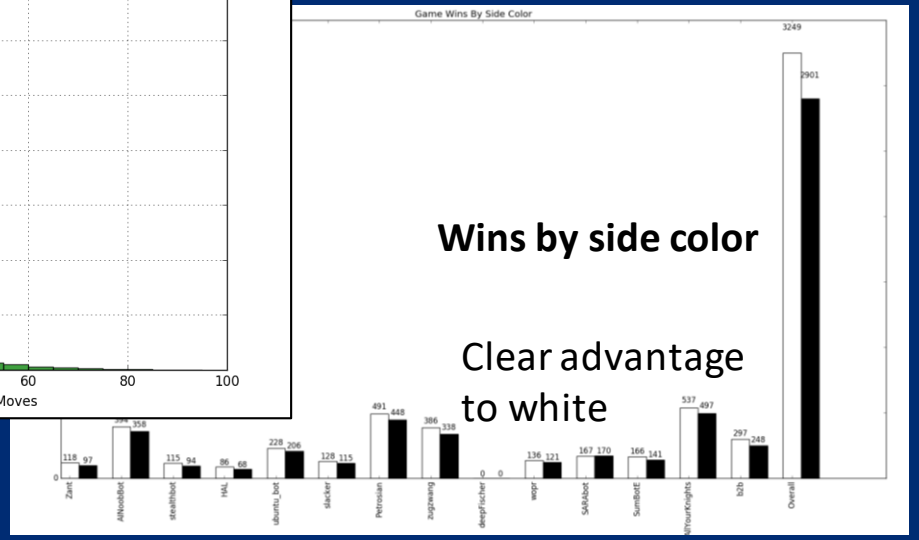
| playerName | AllYourKnights | Petrosian | slacker | AINoobBot | zugzwang | Zant | stealthbot | b2b | HAL | SumBotE | ubuntu_bot | wopr | SARAbot | deep Fischer | Overall |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AllYourKnights | | 44% | 80% | 75% | 82% | 93% | 80% | 92% | 93% | 81% | 87% | 97% | 95% | 100% | 80% |
| Petrosian | 56% | | 77% | 65% | 58% | 97% | 70% | 79% | 77% | 82% | 79% | 80% | 86% | 100% | 73% |
| slacker | 20% | 23% | | 23% | 63% | 100% | 43% | 57% | 73% | 67% | 63% | 83% | 93% | 100% | 62% |
| AINoobBot | 25% | 35% | 77% | | 28% | 50% | 60% | 66% | 87% | 86% | 90% | 93% | 75% | 100% | 61% |
| zugzwang | 18% | 42% | 37% | 72% | | 7% | 70% | 64% | 53% | 64% | 79% | 77% | 70% | 100% | 59% |
| Zant | 7% | 3% | 0% | 50% | 93% | | 73% | 3% | 73% | 90% | 93% | 77% | 53% | 100% | 55% |
| stealthbot | 20% | 30% | 57% | 40% | 30% | 27% | | 60% | 73% | 60% | 43% | 57% | 100% | 100% | 54% |
| b2b | 8% | 21% | 43% | 34% | 36% | 97% | 40% | | 53% | 51% | 75% | 47% | 65% | 100% | 44% |
| HAL | 7% | 23% | 27% | 13% | 47% | 27% | 27% | 47% | | 53% | 17% | 40% | 87% | 100% | 39% |
| SumBotE | 19% | 18% | 33% | 14% | 36% | 10% | 40% | 49% | 47% | | 31% | 43% | 83% | 100% | 38% |
| ubuntu_bot | 13% | 21% | 37% | 10% | 21% | 7% | 57% | 25% | 83% | 69% | | 61% | 64% | 100% | 35% |
| wopr | 3% | 20% | 17% | 7% | 23% | 23% | 43% | 53% | 60% | 57% | 39% | | 40% | 100% | 32% |
| SARAbot | 5% | 14% | 7% | 25% | 30% | 47% | 0% | 35% | 13% | 17% | 36% | 60% | | 100% | 27% |
| deepFischer | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | | 0% |

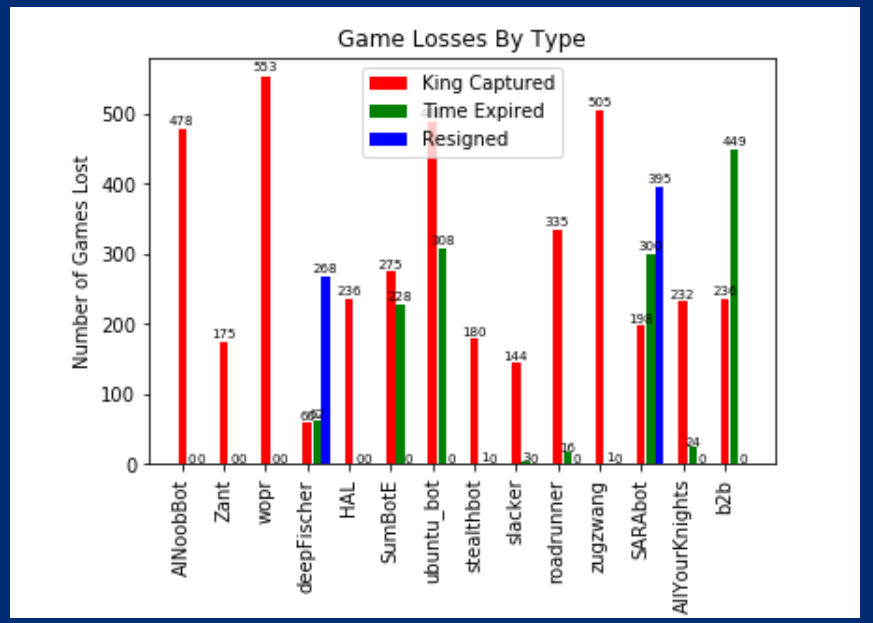
- Win Percentage > 80%
- Win Percentage between 60% and 80%
- Win Percentage between 40% and 60%
- Win Percentage between 20% and 40%
- Win Percentage < 20%

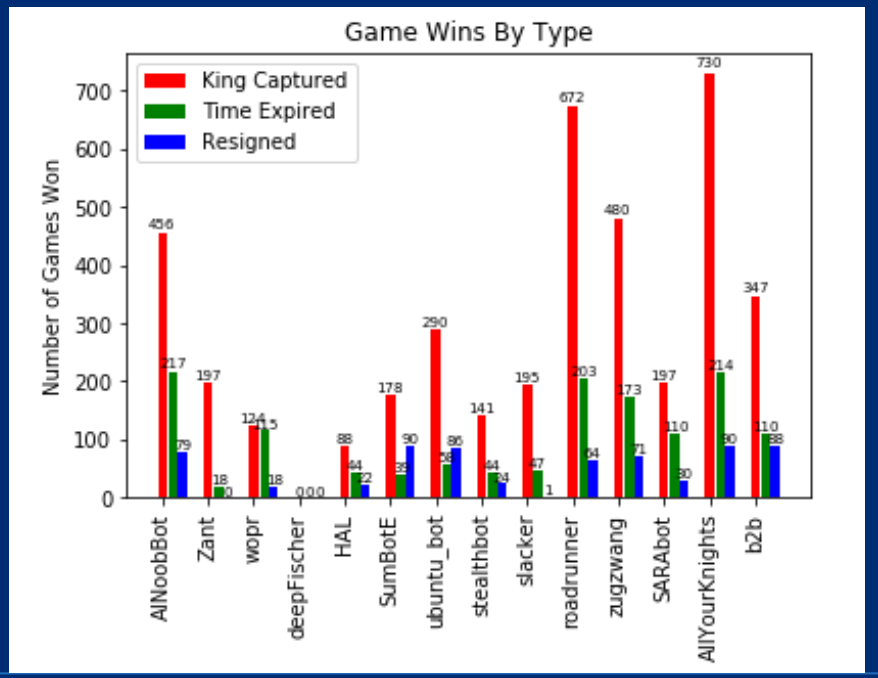
# Competition Statistics

- 6150 games over 3 tournaments
- Longest games (most moves):
  - stealthbot-zugzwang (0-1): 153 moves
  - roadrunner-zugzwang (0-1): 234 moves
- Shortest games (least moves where game ended in king capture):
  - 3 moves (116 times)
  - ubuntu_bot 56 such wins… but also 50 such losses…



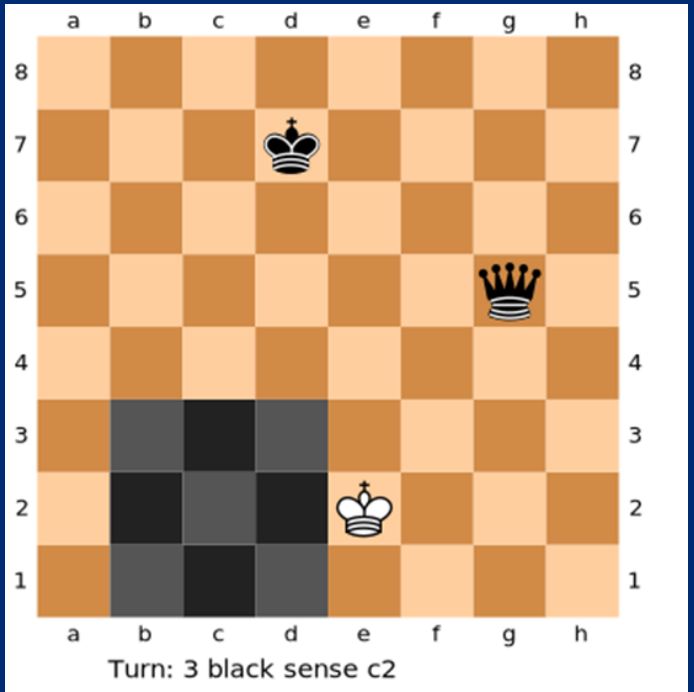



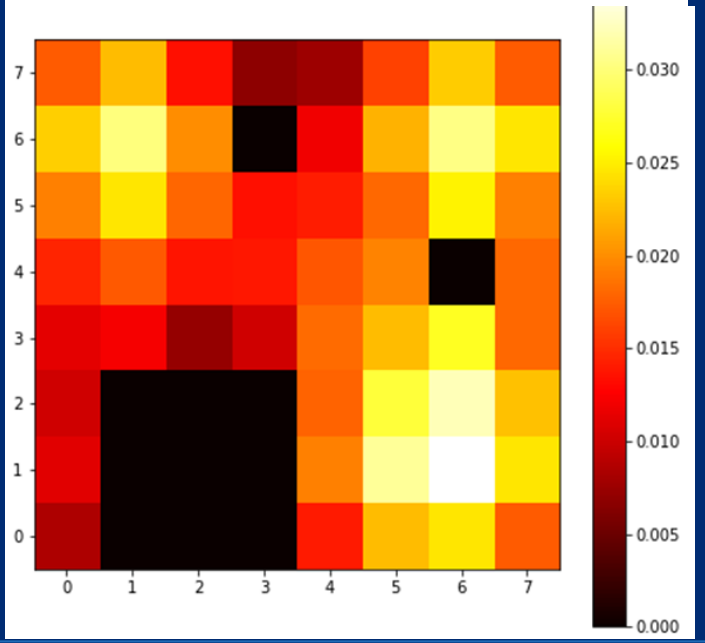



# Reinforcement Learning

- Technical Approach: Use Reinforcement Learning (coupled with other ML techniques like CNN) to learn the optimal choice of sense and move (PO-MDP)
- Working to adapt the many ideas from the recent successful DeepMind results (AlphaGo, AlphaGo Zero, AlphaZero for Chess)
  - Adapting the algorithms to partially observable setting is non-trivial
  - Board representations, MCTS assume perfect knowledge
  - Credit assignment problem for moves and senses (value of sense information)
- Early positive results training a simplified version of the game (Reconnaissance Chess starting in a K vs Q+K “endgame”)
  - Encoding the distribution of opponent king is relatively easy
  - Do we use Bayes rule as an innate algorithm? Do “learn” it?
- Plan is to build up complexity and understand how to scale the algorithms (work in progress)



# RBC Complexity: *Analysis Approach*

- Measure changes in size of players information set:
  - (Wikipedia) "Information set for a player establishes all the possible moves that could have taken place in the game so far, given what that player has observed" (imperfect information)
- In RBC, effective game complexity is heavily dependent on strategy.
- To analyze the complexity of RBC, we played our bots against each other (all bots use Stockfish in base move strategy except RandomBot25):
  - **RandomBot25** senses and moves randomly, passing 25% of the time.
  - **NaiveBot** senses by minimizing per-square time since last sense
  - **MHTBot** senses to minimize the expected information set size (using a multi-hypothesis tracker)
  - **PredictorBot** senses using a prediction of the opponent's move based by assuming Stockfish as base strategy
  - **PerfectInfoBot** is granted perfect information

# RBC: *Estimated Complexity*

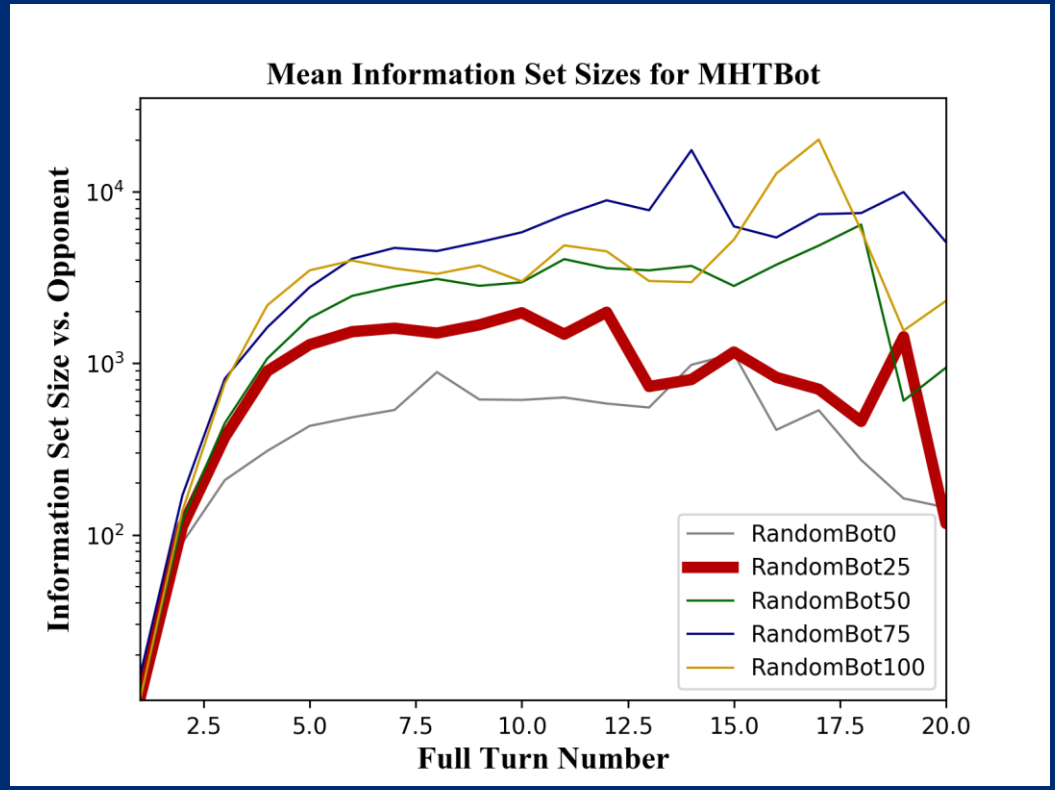
| | Chess | Go (19x19) | Large No Limit Heads Up Poker | RBC |
|---|---|---|---|---|
| **Game Size** | $10^{43}$ | $10^{170}$ | $10^{162}$ | $10^{178}$ |
| **Average Information Set Size** | 1 | 1 | $6.4 \times 10^{14}$ | 653 |
| **Above + *Opponent's Knowledge*** | 1 | 1 | $6.4 \times 10^{14}$ | $1.3 \times 10^{66}$ |

Conservative estimate obtained by playing MHTBot against itself

- Our analysis shows that RBC ***compares to Go and No Limit Heads up Poker*** in overall game size (number of possible play-outs)
- Uncertainty around what the opponent knows creates adds ***significant additional complexity*** (in terms of average information set size).

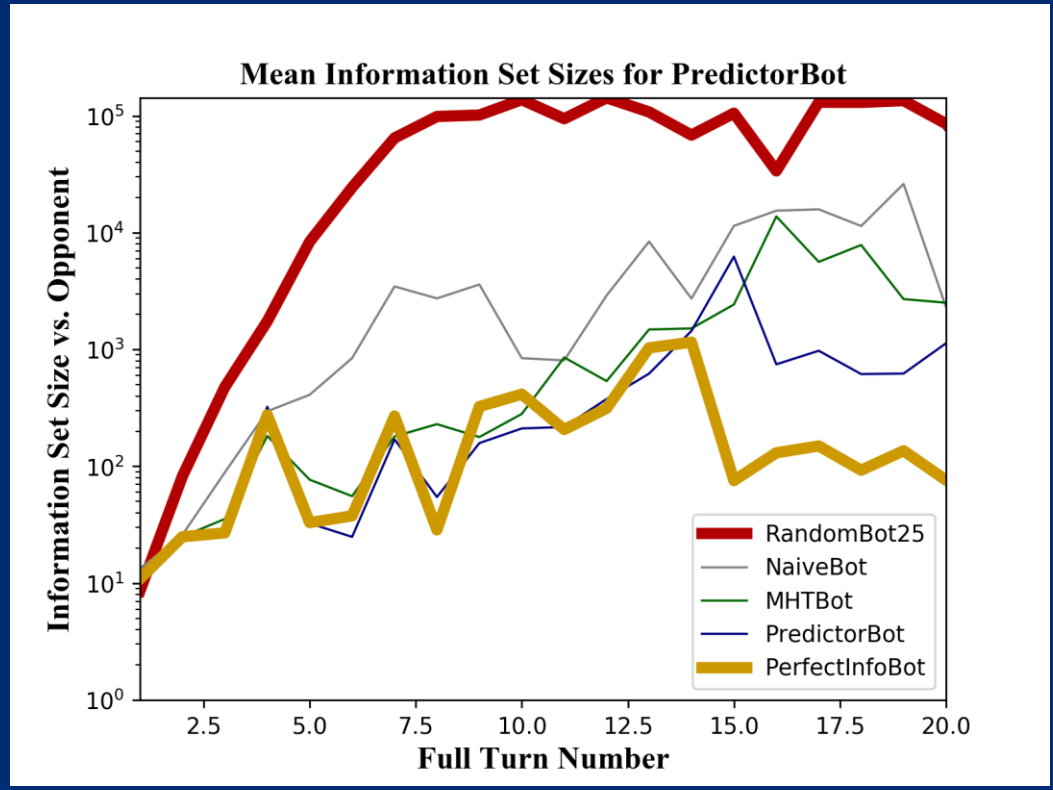
# Complexity Observations: *Novel Tactics*

- ***Moving unpredictably imposes complexity on the opponent***
- While not legal in standard chess, passing is legal in RBC
- Passing, while not necessarily leading to strategic progress, can lead to significant increases in the size of the opponent's information set



# **Observations:** *The Need for Mixed Strategies*

- ***Analysis results underscore the need for mixed strategies***
- PredictorBot was able to reduce the number of possible information sets effectively against PerfectInfoBot (since PerfectInfoBot matched its move assumptions)
- This underscores the need to mix strategies if there is a possibility that your opponent knows your base strategy

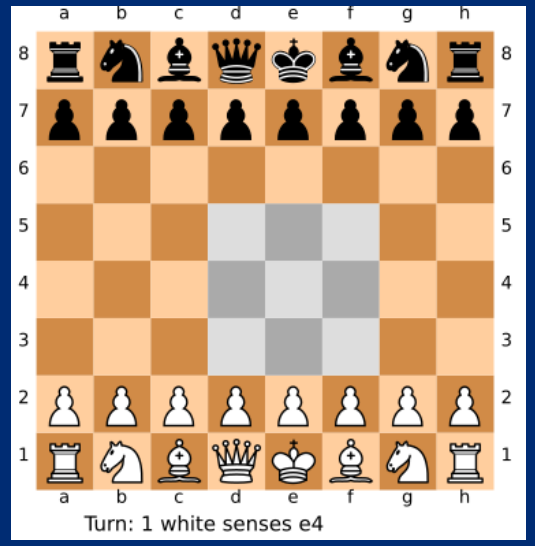


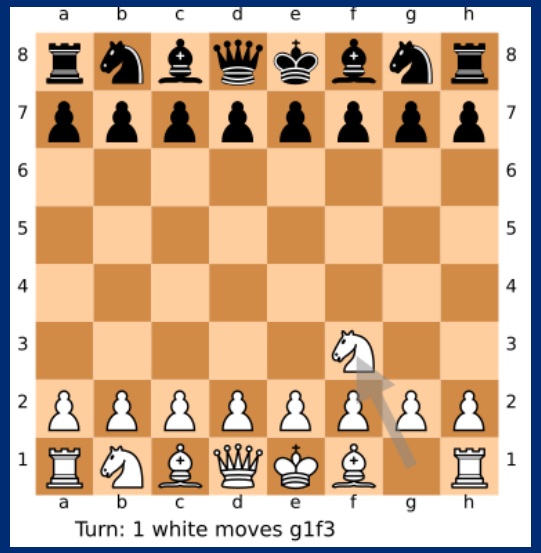
# Human-Machine Teaming with RBC: Concepts

- Have decision makers play RBC to get intuitions about how to manage uncertainty
- Computer provided sense and move suggestions
  - Possibly with explanations (principal variations, human readable descriptions)
- Computer assists the human in managing the uncertainty
  - As game progresses, have compute use multi-hypothesis algorithm to maintain distribution of ground truth states
  - Use information acquired during game play to prune hypothesis space
  - Display relevant information about board position probabilities to help human make the sense and move decisions

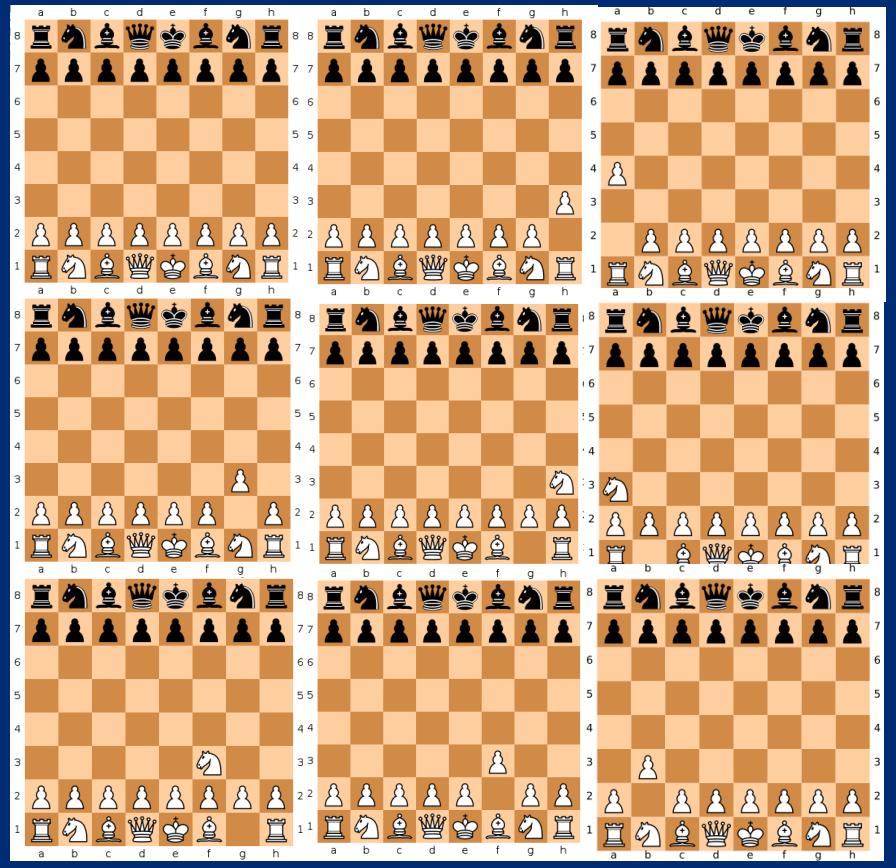
# HMT with RBC: Multi-Hypothesis Tracker

White Turn

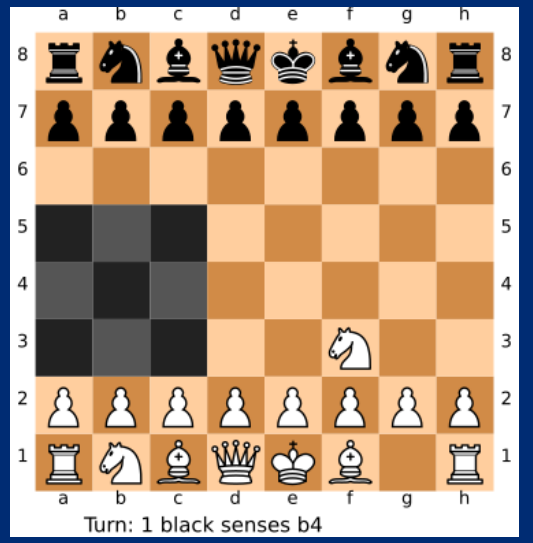



Start of Black Turn = 21 Hypotheses



Black Turn




After Black Sense = 21 – 8 = 13 Hypotheses

# HMT with RBC: Probability Display

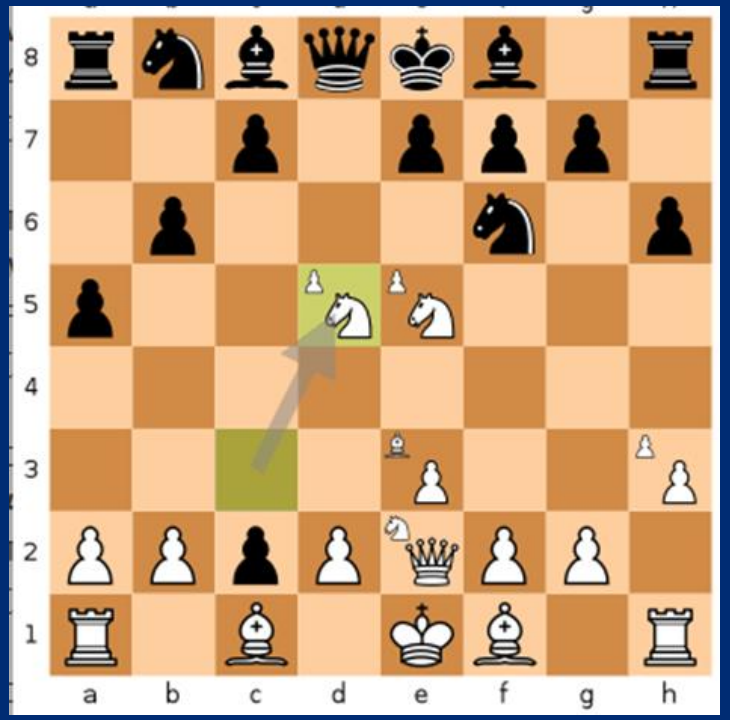
- How to summarize information in large hypothesis sets?
  - Weigh boards with: most risk? highest reward? most probable?
- Suggest senses that would gain the most information (reduce hypothesis space)



"Experimental" HMT Display from RBC website

# RBC: Potential Variations

- Sensors
  - Orbiting: periodic access to different parts of the board (or different boards)
  - Organic: tethered to a piece; limited sensing range
  - Localizing: square occupancy only; no piece identity information
  - Noisy: missed detections, false alarms, localization error, recognition error
- New Game Actions
  - Camouflage a piece to appear as another piece to the opponent's sensors
  - How to "coordinate deception"? Can enemy unravel the ruse?
- Game flow
  - Discrete / alternating with or without time limits
  - Continuous / asynchronous
- Command and control uncertainty and latency
  - Noisy moves (uncertainty in command execution)
  - Information transmission delays
  - Local versus global decisions and planning
  - Humans in part of the decision chain interacting with autonomous agents



# RBC: Potential Variations (cont'd)

- Multiple Boards (Reconnaissance Blind Multi-Chess)
  - Multiple concurrent boards (what is the objective?)
  - Must manage maneuver and sensing budgets over multiple boards
  - Each board weighted differently in importance
  - Model multiple domains (i.e., different rules govern each board)
  - Boards are separate but influence each other
  - Playing against multiple adversaries with complex alliance structure
    - Differing adversary capabilities (forces, sensing, algorithms) and objectives
- Multiple Objectives
  - RBC: different objectives in the game; RBMC: different objectives on different boards
  - Adversarial and competitive objectives (possibly hidden from opponent)
- …

# References


- Newman, A., Richardson, C., Kain, S., Stankewicz, P., Guseman, P., Schreurs, B., and Dunne, J., "Reconnaissance blind multi-chess: an experimentation platform for ISR sensor fusion and resource management," in Proc. SPIE Vol 9842, SPIE Defense and Commercial Sensing, Signal Processing, Sensor/Information Fusion, and Target Recognition XXV Conference, Baltimore, MD (April 2016).

- Newman, A., "Reconnaissance blind multi-chess: an experimentation platform for ISR sensor fusion and resource management," *Proc. 72nd Automatic Target Recognition Working Group Meeting*, NGA Campus East, Springfield, VA (August 2016).

- Newman, A., Richardson, C., Kain, S., Stankewicz, P., Guseman, P., Schreurs, B., and Dunne, J., "Reconnaissance blind multi-chess: an experimentation platform for ISR sensor fusion and resource management," in *Proc. 2016 Joint Meeting of the Military Sensing Symposia, National Symposium on Sensor and Data Fusion (NSSDF)*, Gaithersburg, MD (June 2016).

- Newman, A., Richardson, C., Li, W., and et al., "Autonomous ISR Applications of the JHU/APL 2017 Reconnaissance Chess Autonomy Challenge Competition," *Proc. 2017 Joint Meeting of the Military Sensing Symposia (MSS), National Symposium on Sensor and Data Fusion (NSSDF)*, Springfield, VA (31 October 2017)

- Llorens, A., Markowitz, J., Gardner, R., Newman, A., Richardson, C., Li, W., "Reconnaissance Chess: Towards Machine Decision-Making Under Uncertainty," NATO Science and Technology Organization, Information Systems Technology Panel, IST-160 Specialists' Meeeting, *Big Data and AI for Military Decision Making*, Bordeaux, France (30 May – 1 June 2018).

# Questions?

- Contact:
  - Casey Richardson: Casey.Richardson@jhuapl.edu
- Collaboration:
  - Conference Challenge with RBC
  - Bot algorithms R&D
  - Compute / Distributed RL training
  - Complexity
  - Variation design
  - Human Machine Teaming Concepts with RBC
  - Model New Domains with RBC
  - Software tools and platform development

- Getting Started:
  - Python package available on pypy: “pip install reconchess”
  - Getting started and documentation: https://reconchess.readthedocs.io/
  - Get the code and bot API: https://github.com/reconnaissanceblindchess/reconchess
  - RBC website: http://rbmc.jhuapl.edu


JOHNS HOPKINS
APPLIED PHYSICS LABORATORY