Hiding Information From The AIs

Noah Morris

July 21, 2023

1 Inspiration

Let us consider a time of leisure, and suppose we find ourselves in the company of ≥ 2 other people, and that there is rain. The odds are slim that the Game chosen will feature perfect information: likely, there will be randomly drawn cards, hidden from view of the other players. Over the past 25 years, the board game industry has grown massively, with thousands of new games released every year[2]. and as such the average modern board game involves many more strategic variables, at least one of which is usually a specialized deck of cards. Consider modern classics such as Catan, Ticket to Ride, and Pandemic: three undeniably engaging and strategically dense games, all utilizing chance to some extent. The effect of the inclusion of unknowns into a game is variable: among humans, there exists a sentiment that games of chance or hiding are less difficult to play well than more "pure" abstract strategy games: where in the latter one's win is a sheer show of force, some unknowable portion of the credit in one's win in the former is thought to be due only to fate. Regardless of whether this sentiment is true, it is certain that games of incomplete information are more difficult for the AI algorithms in Ludii, a powerful open-source General Game System hosted in Java.

A General Game System consists of two primary features: a syntax through which one can describe the components and rules of an arbitrary game, and an AI that compiles the game description and engages in dynamic, intelligent play. The most common algorithm for this purpose is that mentioned above, MCTS. For a reasonable AI performance, it is sufficient to provide a ruleset. All Ludii AIs take in a tokenized version of the game description at outset in order to judge the value of any legal move[9]. They can become quite complicated, but below is a sample of the implementation of Tic-Tac-Toe, which is not.

```
(game "Tic-Tac-Toe"
    (players 2)
    (equipment {
        (board (square 3))
```

```
        (piece "Disc" P1)
        (piece "Cross" P2)
    })
    (rules
        (play (move Add (to (sites Empty))))
        (end (if (is Line 3) (result Mover Win)))
    )
)
```

Ludii's syntax, when it released in 2017[11], was a massive improvement on existing game description languages. It takes far less time to write a Ludii description, because the language is very robust, allowing complex statements to be expressed succinctly. The project has about 1,500 games available on its website, an incredible, simultaneous feat of computation and anthropology. However, at the beginning of the summer, there were no card games, because the AI cheats.

2 In Search of An Honest UCT Search

2.1 The AIs Cheat

The first essential issue that I faced this summer was that information could not really be hidden in Ludii. Syntactically, there was a robust suite of aspects of an in-game situation that can be "set Hidden", but doing so did not impact what the AIs were able to see in any way. This is because, on setting something invisible to a particular player, one passes the "context", which is essentially the board position, through a function called "InformationContext" which removes the necessary information. This method, on occurance, only updated the GUI, meaning that the AIs were free to cheat. Reading through the source code makes this clear, but I used a couple of tools to verify this as I tested potential solutions.

2.2 The Monty Hall Problem

The Monty Hall Problem is a classic, counterintuitive probability puzzle from the '70s, which is as follows. At outset, there exist three doors, behind two of which lie goats (worthless), and one, a car (good). The protagonist (person A) chooses a door, gaining some affinity for it in the process, and then the host (person B) opens one of the two remaining doors, revealing a goat. Thus, of the three doors, one is totally out of the running, one is the current choice, and one remains a possibility. Person B is about to ask if you want to switch.

Monty Hall Problem		
	A chose Car ($P(\frac{1}{3})$)	A Chose Goat ($P(\frac{2}{3})$)
Car	Choice	Switch
Goat 1	Out	Out
Goat 2	Switch	Choice

This means that there is a 2 in 3 probability that you're on the right side of the chart, where switching will get you the car. The Ludii implementation, as a one-player game, attempts to set what is behind the doors hidden: one would expect that a good AI would approach a 2/3 win rate. However, I have found that all existing Ludii AIs win 100% of the time, demonstrating their ability to see things that have been "set Hidden". A similar, slightly faster test, that I called "The Game Of Nil" (in which both players lay down a white or black Piece, then reveal what they chose. If they're the same, player 2 wins, otherwise Player 1 does), is in the attached Github repository of finished Ludii games[3]. It, too, showed that the Ludii AIs all cheat, although it demonstrated this a bit quicker.

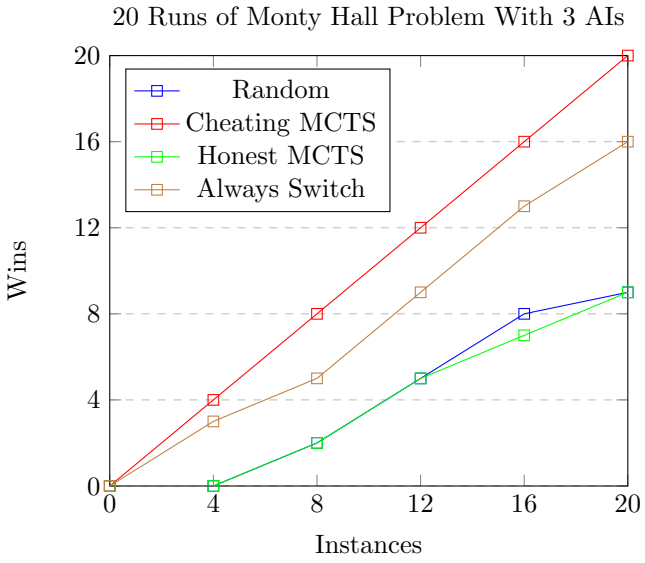
2.3 An Example Shell Method

To remedy this, the solution that I found was to make a shell method in the "LudiiExampleAI" package[18], which passes a context through the InformationContext method, filtering out the specified information and doing nothing otherwise. One can then pass the resulting context into the AI, provided, on instantiation in the method AI.java, the boolean "wantsCheatRNG" is set to false. The below method does this for the standard "UCT" implementation. In the future, I plan to integrate this solution into the AI methods within the Ludii source code, so one does not need to run the separate AI package to ensure an honest AI.

```
package mcts;

import game.Game;
import other.AI;
import other.context.Context;
import other.context.InformationContext;
import other.move.Move;

import search.mcts.MCTS;

//  Thank you to Victor Putrich, who did this in
    https://github.com/schererl/FinalYearProject/tree/main/src.

public class MCTSStoHI extends AI
{
```

```
  protected int player = -1;
  private MCTS UCT_Ludii;
  public MCTSStoHI()
  {
     this.friendlyName = "MCTSStoHI";
  }

  @Override
  public Move selectAction
  (
     final Game game,
     final Context context,
     final double maxSeconds,
     final int maxIterations,
     final int maxDepth
  )
  {
     Move selectedMove=UCT_Ludii.selectAction(game, new
         InformationContext(context,player), maxSeconds, maxIterations,
         maxDepth);
     return selectedMove;
  }

  @Override
  public void initAI(final Game game, final int playerID)
  {
     this.player = playerID;
     UCT_Ludii = MCTS.createUCT();
     UCT_Ludii.initAI(game, playerID);
  }

  @Override
  public boolean supportsGame(final Game game)
  {
     if (!game.isAlternatingMoveGame())
        return false;

     return true;
  }
}
```

With this AI, I have found honest behavior within The Monty Hall Problem and The Game of Nil, and competitive play with the normal UCT in perfect-information games like Hex. The below chart shows 20 instances of the Monty Hall Problem, with this AI.



The Honest MCTS loses sometimes at the Monty Hall Problem,which demonstrates that it has genuinely removed the hidden information from its context. This is good, but not optimal. The Brown Line shows the optimal strategy of always switching, which the AI does not do, and performs worse than. Most games of chance involve weighing probabilities with far more complexity than this.

This aligns with I've found after various more involved experiments in other games of hidden information: the Ludii AIs designed for perfect information games are not well-suited to hidden information games, once made to play them honestly, usually performing comparably to a random player, and being demolished by an intelligent, cheating player. The current UCT implementation does not account in any way for what it doesn't know, leading to poor play. I plan on working toward a more strategic Honest MCTS in Ludii over the next year, taking inspiration from the implementation that my advisor, Dr. Mark Goadrich, uses for this purpose in CardStock[13]. Despite its skill, the ability to communicate incomplete situations to the Ludii AI is the first step toward generalizing the system to be card-capable, and this simple solution does not degrade decision time or performance in perfect information games.

3 Experimental Modifications

In almost every card game, one hides some cards from the other players: this was not possible for me with the current Deck Ludeme. It posits itself as a container distinct from the rest of the board, but is in actuality a Stack. In line

2725 of Game.java, if a Deck has been created in the (equipment) section, the Card components get shuffled and then placed like this.

```
    for (int i = 0; i < nbCards; i++)
      {
        final int j = (context.rng().nextInt(components.size()));
        final int index = components.getQuick(j);
        final StartRule rule = new PlaceCustomStack("Card" + index,...
```

The Stack, in Ludii, is not defined as a Component or Container. It is a set of components placed with one command, but does not function as an object. As it stands, they can only exist up to a size of 32, at which point the Stack must become a "LargeStack", the presence of which slows the engine noticeably. This is not specified in the reference, is remedied by activating the boolean for any game with a deck. to the equipment breaks the engine if the largeStack boolean is not specified as True. For example, in the case of "largeStack:False", the following program is syntactically correct, according to the compiler in the Ludii Editor, but breaks the engine, returning the below stacktrace.

```
(game "DECK"
  (players 3)
  (equipment {
    (board (square 10))
    (deck {(card Seven rank:0 value:0 trumpRank:0 trumpValue:0)
           (card Eight rank:1 value:0 trumpRank:1 trumpValue:0)
           (card Nine rank:2 value:0 trumpRank:6 trumpValue:14)
           (card Ten rank:3 value:10 trumpRank:4 trumpValue:10)
           (card Jack rank:4 value:2 trumpRank:7 trumpValue:20)
           (card Queen rank:5 value:3 trumpRank:2 trumpValue:3)
           (card King rank:6 value:4 trumpRank:3 trumpValue:4)
           (card Ace rank:7 value:11 trumpRank:5 trumpValue:11)})
    (hand Each size:2)
  })
  (rules
    (start {(deal Cards 2)})
    (play (move PlayCard))
    (end (if "HandEmpty" (result Mover Win)))
  )
)
```

```
java.lang.ArrayIndexOutOfBoundsException: 0
  at gnu.trove.list.array.TIntArrayList.getQuick(TIntArrayList.java:305)
  at main.collections.ListStack.isHidden(ListStack.java:745)
  at
      other.state.stacking.ContainerStateStacksLarge.isHidden(ContainerStateStacksLarge.java:579)
  at other.action.move.ActionAdd.apply(ActionAdd.java:269)
  at game.rules.start.Start.placePieces(Start.java:113)
```

```
  at
      game.rules.start.place.stack.PlaceCustomStack.eval(PlaceCustomStack.java:195)
  at game.Game.start(Game.java:2728)
  at app.utils.GameUtil.startGame(GameUtil.java:186)
  at app.utils.GameUtil.resetGame(GameUtil.java:85)
  at app.utils.GameSetup.compileAndShowGame(GameSetup.java:49)
```

The one other Dealable Type is the Domino, which renders correctly, but deals only half of what it's told.

```
(game "DOMINO"
    (players 4)
    (equipment
        {
            (board (square 20))
            (dominoes)
            (hand Each size:7)
        }
    )
    (rules
        (start {
            (deal Dominoes 7)
        }       )
        (play (move (from (sites Hand Mover)) (to (sites Empty))))
        (end (if "HandEmpty" (result Mover Win)))
    )
)
```

This issue is comparatively easy to fix: change line 161 of Deal.java in the source code from

```
while (dealed < (count * 2))
```

to

```
while (dealed < (count * context.game().players().count()))
```

This makes the above code properly deal. I was not able to find a similar fix to the issue with dealing cards: the largeStack has many methods that still must be done (see ContainerStateStacksLarge.java, lines 864-908), and hiding < 32-tall stacks is difficult, often breaking the engine. I found some mixed results modifying the source code, adding filters to avoid IndexOutOfBoundsExceptions on each "isHidden" boolean, and was able to make games compile, but not function properly. After a while, I decided to use a different tactic to render card games in Ludii.

4 Making Card Games

4.1 Shuffling

The most important existing game in the library for my development of card games was Quarto, an excellent abstract-strategy game that I always played with cards. It involves sixteen pieces, each with four dichotomous attributes: one or two, small or large, black or red, square or circle. The players take turns laying one down of their opponent's choice, attempting to make a line of four with at least one attribute entirely in common. With the implementation of this game, I had pieces that carried various integer values, which were capable of affecting scoring and visuals. After some light trial and error, I had a functional adding game, 98, where cards were instantiated by a simple random number generator, written in Ludii as "(value Random (range 1 13))". However, in many card games, the fact that cards are selected without replacement is crucial to strategy: one such example is "Hearts". My first strategy, of a stack that is either placed randomly or selected from randomly, works, as shown in the "ShuffleStack.lud" file[3]. However, as detailed above, I ran into a litany of issues setting stacks Hidden, and (after some failed experiments with manual Knuth-shuffling with a sea of variables, modular multiplication, and the "RememberValues" ludeme), I went with Place Random (sites), hiding the board one unit at a time from the AIs. This solution can lead to large, unused swaths of boardspace, but moving items secretly becomes much more reliable when everything has a unique index, allowing a developer to tweak things without risk of IndexOutOfBoundsExceptions. Additionally, large stacks jut out from the game board along a z-axis, visibly slowing startup and AI speeds. The method below, the complete version of which can be found in the "LudiiGames" repository by the name "Shuffle", which creates and deals a 35-card deck, does not.

```
(game "Shuffle"
  (players 4)
  (equipment {
    (board (rectangle 1 16))
    (piece "Square" Shared)   (piece "Square1" Shared)
    (piece "Square2" Shared)  (piece "Square3" Shared)
    (piece "Square4" Shared)  (piece "Square5" Shared)
    (piece "Square6" Shared)  (piece "Square7" Shared)
    (piece "Square8" Shared)  (piece "Square9" Shared)
    (piece "Square10" Shared) (piece "Square11" Shared)
    (piece "Square12" Shared) (piece "Square13" Shared)
    (piece "Square14" Shared) (piece "Square15" Shared)
    (piece "Square16" Shared) (piece "Square17" Shared)
    (piece "Square18" Shared) (piece "Square19" Shared)
    (piece "Square20" Shared) (piece "Square21" Shared)
    (piece "Square22" Shared) (piece "Square23" Shared)
    (piece "Square24" Shared) (piece "Square25" Shared)
```

```
    (piece "Square26" Shared) (piece "Square27" Shared)
    (piece "Square28" Shared) (piece "Square29" Shared)
    (piece "Square30" Shared) (piece "Square31" Shared)
    (piece "Square32" Shared) (piece "Square33" Shared)
    (piece "Square34" Shared) (piece "Square35" Shared)
    (hand Each size:6)
  })
  (rules
    (start {
      (place Random (sites (union (array {1..5}) (array {10..39})))
        {"Square1" "Square2" "Square3" "Square4" "Square5"
         "Square6" "Square7" "Square8" "Square9" "Square10"
         "Square11" "Square12" "Square13" "Square14" "Square15"
         "Square16" "Square17" "Square18" "Square19" "Square20"
         "Square21" "Square22" "Square23" "Square24" "Square25"
         "Square26" "Square27" "Square28" "Square29" "Square30"
         "Square31" "Square32" "Square33" "Square34" "Square35"})
      (forEach Player (set Hidden (sites (union (array {1..5})
        (array {10..15}))) to:Player))
    })
    (play (move Pass))
    (end {(if ("HandEmpty" Next) (byScore))})
  )
)
```

4.2 Game Overview

At the beginning of the summer I wrote Coup and Poker in CardStock, then I moved to Ludii, writing three Combinatoric games to learn the ropes: Teeko, Hexade, and Catchup. As I developed my methods, I made four games of hidden information in Ludii: Agram, Hearts, 98, and Minesweeper. I began work on many more games, which I plan on continuing to work on over the next year. All finished games can be found in "LudiiGames"[13], which I will update as I complete more of them.

5 Further Work

An ancillary goal is the application of AI capable of handling hidden information towards a heuristic that CardStock cannot account for: Solvability. Ludii contains syntactic support for deduction puzzles, and the Ludii team has written about utilizing the XCSP format in tandem with Ludii to solve deduction puzzles like Sudoku and Nonograms.[12] I will explore applications of this approach taken by Piette et. al toward deducing what positions are possible in

multi-player hidden information games. Going further, with a randomly set board, and an AI that assures a threshold "Solvability" metric, I believe that procedurally generated deduction-puzzles and stochastic environments can be generated within Ludii. I have created Minesweeper in the language, and would like to explore methods of ensuring unique solutions and solvability[12,19], as they spring from improving the current AI's performance.

I also plan to implement more card games within Ludii, potentially attempting to translate some of the suite of more than 30 available games in CardStock. CardStock gathers Heuristics about games hosted within it, such as Convergence, Drama, Fairness, and Spread, which are valuable to a game designer who wants to assure that these factors are in balance as they playtest. I plan to implement similar heuristics in Ludii.

Over the next year, my primary goal will be to create a more capable honest AI for games of hidden information in Ludii. I plan on utilizing methods employed by Dr. Goadrich in CardStock, including both those in the source code and these, and game theoretical methods introduced by Nash and Conway[4,5], and existing Ludii AIs.

6 Sources

1. "Go master Lee says he quits unable to win over AI Go players", Yoo Cheong-mo, November 27 2019, https://en.yna.co.kr/view/AEN20191127004800315

2. "Board games' golden age: sociable, brilliant and driven by the internet", Owen Duffy, November 25 2014, https://www.theguardian.com/technology/2014/nov/25/board-games-internet-playstation-xbox

3. "Ludii Games", Noah Morris, July 21 2023, https://github.com/brimstonetrader/LudiiGames

4. "Winning Ways for your Mathematical Plays", John Conway & Elwyn Berlekamp & Richard Guy, 1982, https://archive.org/details/winningwaysforyo02berl

5. "Non-Cooperative Games", John Nash, 1951, https://web.archive.org/web/20150420144847/http://www.princeton.edu/mudd/news/faq/topics/Non-Cooperative_Games_Nash.pdf

6. "Ludii Game Logic Guide", Eric Piette & Cameron Browne & Dennis Soemers, https://ludii.games/downloads/LudiiGameLogicGuide.pdf

7. "RECYCLEd CardStock", Mark Goadrich, 2018, https://cardstock.readthedocs.io/en/latest/index.html

8. "A Practical Introduction to the Ludii General Game System", Eric Piette & Cameron Browne & Dennis Soemers & Matthew Stephenson, 2019

9. "Ludii - The Ludemic General Game System", Eric Piette & Cameron Browne & Dennis Soemers & Matthew Stephenson & Chiara F. Sironi & Mark H. M. Winands, 2020

10. "General Board Game Concepts", Eric Piette & Cameron Browne & Dennis Soemers & Matthew Stephenson, 2021

11. "Ludii Language Reference", Eric Piette & Cameron Browne & Dennis Soemers, May 16 2023, https://ludii.games/downloads/LudiiLanguageReference.pdf

12. "Ludii and XCSP: Playing and Solving Logic Puzzles", Eric Piette & Cameron Browne & Dennis Soemers & Matthew Stephenson, August 23 2019, https://cris.maastrichtuniversity.nl/ws/portalfiles/portal/123479428/Soemers-2019-Ludii-and-XCSP-Playing-and.pdf

13. "CardStock", mgoadrich, https://github.com/mgoadric/cardstock

14. "GGP in Ludii", brayo303, https://github.com/brayo303/GGPinLudii

15. "Ludii AI Dev", z5164964, https://github.com/z5164964/LudiiAIDev

16. "Ludii Stuff", mgrider, https://github.com/mgrider/ludii-stuff/ tree/-main

17. "Final Year Project", schererl, https://github.com/schererl/FinalYearProject/tree/main/src

18. "Ludii Example AI", Ludeme, https://github.com/Ludeme/LudiiExampleAI

19. "Improving Solvability for Procedurally Generated Challenges", Mark Goadrich & James Droscha, May 26 2019, https://arxiv.org/pdf/1810.01926.pdf