Hanabi Ai

Members:
Sam Shareski 998581556 g3shares
Lev Ufimtsev 999366635 g2lev

April 2016
CSC384
University of Toronto

Member Roles:

- Sam : Came up with the Idea, Coding,
- Lev : Documentation, Diagrams, Testing,

Type of Project:

- Constraint Satisfaction Problem with limited & unknown information.
- Developing heuristics that use probabilities & likelihoods to achieve higher score than a simple constraint heuristic.

Project Motivation/Background (TODO)

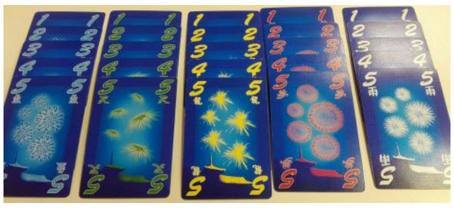
The game (model)

Hanabi is a deductive and cooperative card game. It bears resemblance to the game "Cluedo/Clue" (where you had to deduct what cards the opponent players have and figure out the murder/location/weapon); but Hanabi is sort of the inverse.
In Hanabi you have cards that have a color and a number:

In contrast to Clue, the difference in Hanabi is that you:

1) You don't see your own cards, but you see the cards of the other players:

2) Your goal is to **work together** to win the game cooperatively instead of fighting opponent players.
3) You cannot freely discuss the other player's hand. They cannot tell you exactly which cards you're holding, but can only give you specific "**hints**" like "these cards in your hand are red (point to the red cards)". You can use these hints to **deduce** which cards you are holding. When giving a hint about a colour or number you must point out all cards of that number or colour.
4) Instead of trying to guess a single correct murderer/weapon/location, the objective in hanabi is to lay cards into a decks in a sequential order, where color and card number match. The game is won if all "5" are placed correctly. For example:



5) The player can "discard" a card if he does not things it's useful and pick up another.
6) The limiting factors is that the deck can run out of cards and there is a limit on how many clues can be provided. There is also a limit on the number of 'wrong cards' that can be played (Fuse).

Motivation (A.I functionality)

The job of the A.I is to do the deductive reasoning behind the game.

1. **Information provider**
   It has a mechanism to give other players information about their hand, that helps them deduct which cards they hold.
2. **Move decision maker**
   Given the hints provided by other players, the A.I has to decide which card to play next, or if the player should discard the current card in hand.

It is a constraint satisfaction problem, but with incomplete information. I.e, the A.I has to decide which card to play, sometimes without knowing everything about a cards in the hand.

We have different heuristics that decide how to play:

1) A simplistic heuristic will only play if it has explicit and complete information.
2) A probabilistic heuristic that calculates the probability of a card being playable and acts based on a threshold. It has input parameter that can be tweaked (explained below).

We then evaluate which heuristic function performs better, we also try various parameters to the probabilistic heuristic to gain insight into attaining an optimal play heuristic.

A human cannot keep in mind all the cards that are played, where as a computer can. However, similar to chess, a human can have a ‘strategy’ to guide his moves, where as a deterministic A.I would not. The goal of this assignment is to mimic a ‘strategy-like’ behaviour via probabilistic reasoning. The A.I. also has the advantage of easily synthesizing ‘negative’ information. That is, if the A.I. is told the a card is red, it also knows that all other cards are not red.

Methods

Models

Variables: Card in hand
You don't know what cards you are holding, but you have information about your cards that you can use to deduce which cards you might hold. Similar to sudoku, with enough information, you can deduce what cards can be. As such we keep track of the following:

Variables: Color of a card, Number the card can be

Domains: Color: {white, yellow, green, blue, red},   Number: { 1,2,3,4,5}

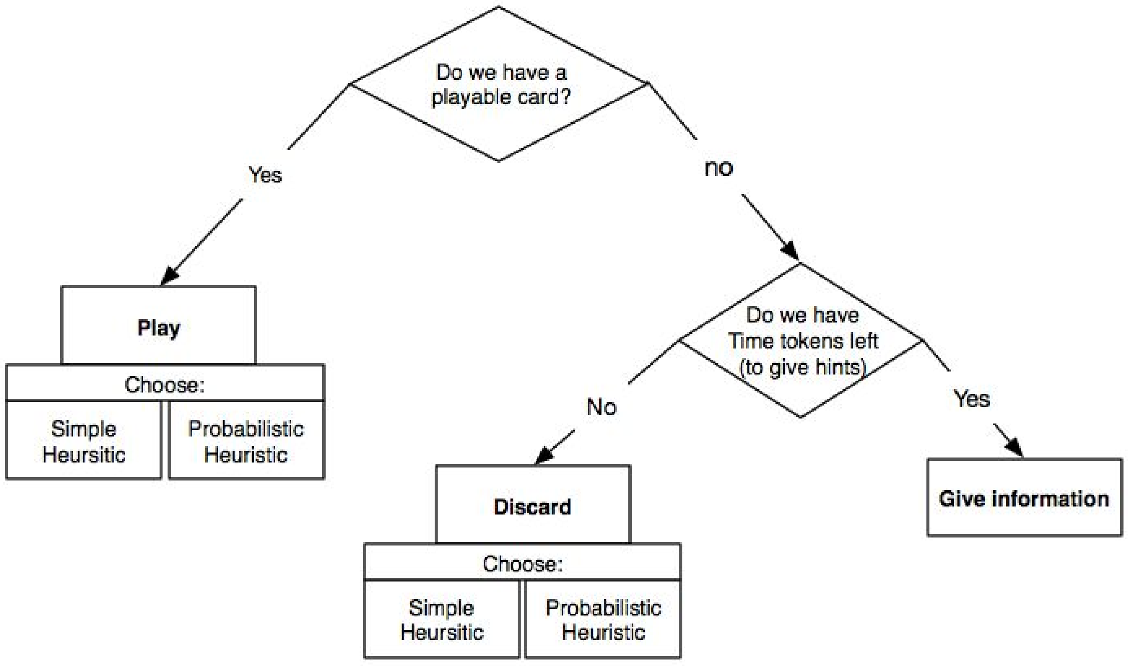
As hints are provided, the domain of these variables is reduced to until a specific color and number can be determined.

State:
A state is encoded as the following:

- Cards in play area (stacks already built)

- Cards in discard pile
- Card in-hand information for the current player
- Cards held by other player
- A set of tokens called "Time", which represent how much info can be given to other players.
- Number of remaining Fuses. One fuse allows you to make one mistake (play wrong card). You start with 3 fuses, if you make 3 mistakes, the game is over).

Successor function:
The successor function is a function that decides what to do next. It can be either one of "Play", "Discard" or "Give Information".
It decides based on a decision tree:



The 'play' and 'discard' nodes can either use a simple heuristic or use a probabilistic heuristic.

Play:
The play function finds out if a card can be played. A card is playable if it can be put onto one of the existing decks, such that the color matches the stack and the number increments by one when you put it on a deck. For example below you can play a 'red' 5 on the pile on the left:

Play - Simple heuristic:
The simple heuristic will wait till it has complete information about all the cards in one's hand, only then will it play a card. Otherwise it will try to give hints or discard a card.

This A.I plays "safe", but it risks of not being able to complete the game, or it may get a poor score.

It takes into account all cards that have been played so far, the hints that it was given for the cards it holds, the cards of the other player and the cards in the discarded pile.

Play - Probabilistic heuristic:

The probabilistic heuristic also takes into account the complete state and it's history (as above), but it decides whether to play or not based on the probability of a card being playable.
It acts given a threshold (e.g 60%, 70%, 80% certainty). This is an optimistic heuristic, that "tries its luck". However, if too many mistakes are made it becomes pessimistic, it has a 'safety threshold' such that it will only play a card if it is 90%, 95% or more certain. I.e, it is an adaptive heuristic that takes account the current state of the game.

As the game plays out the AI is able to make much better decisions. This is because as the discard pile and play area accumulate more cards it can be more certain about the identity of any particular card in the player's hand.

Give information:

The 'give information' function figures out the best possible information that can be given and gives it to the player. Information is something like "these 3 cards are red", or "these two cards are a 2". Good information is when your piece of information gives information about as many cards as possible, or cards that are worth more (i.e cards you can play, cards which are more rare etc.).

Discard card

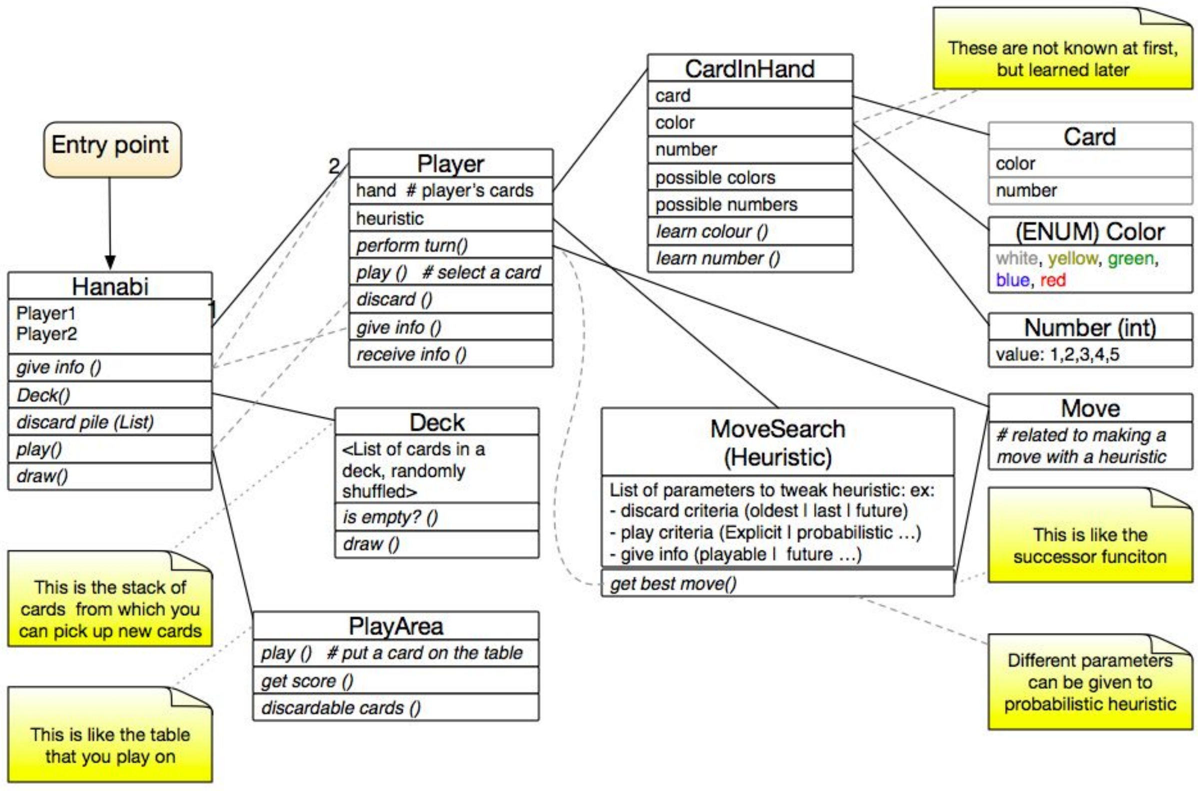
This is similar to play card, but it's the inverse. It decides which card has the least value. The simple heuristic will always discard the 'oldest' card. But the probabilistic heuristic can be run with different parameters:

- Least likely playable:
  This might be a card that will never be used again because it's color/number has already been put down.
- Least likely playable in the future:
  Similar to above, but it considers how useful the card will be in the future.
- Least rare card:
  Rare cards are those of which only few remain. For example there is only one '5' card of each color, but three '1's of each color. As such the '5' is more rare. Also it takes into account cards that have already been played.

Code Documentation

The project is coded in Python.
Hanabi is the main class that interacts with all the other classes.
The player class represents players one and two. They are connected to each other via Hanabi.
CardInHand is information about the possible characteristics of a card that you're holding, for example we might know that it's a '4 or 5' and 'red or yellow'.
The Deck contains a list of cards, that are randomly shuffled upon creation.
The play area is like the desk where cards are put down.

The Heuristic class is an abstract class. Alternative heuristics extend the main heuristic class.



Entry point
Hanabi
Player1
Player2
give info ()
Deck()
discard pile (List)
play()
draw()
1
2
Player
hand # player's cards
heuristic
perform turn()
play () # select a card
discard ()
give info ()
receive info ()
CardInHand
card
color
number
possible colors
possible numbers
learn colour ()
learn number ()
These are not known at first, but learned later
Card
color
number
(ENUM) Color
white, yellow, green, blue, red
Number (int)
value: 1,2,3,4,5
Move
# related to making a move with a heuristic
This is like the successor funciton
Deck
<List of cards in a deck, randomly shuffled>
is empty? ()
draw ()
MoveSearch (Heuristic)
List of parameters to tweak heuristic: ex:
- discard criteria (oldest | last | future)
- play criteria (Explicit | probabilistic ...)
- give info (playable | future ...)
get best move()
Different parameters can be given to probabilistic heuristic
PlayArea
play () # put a card on the table
get score ()
discardable cards ()
This is the stack of cards from which you can pick up new cards
This is like the table that you play on

Evaluation and Results

Score:
We evaluate the quality of our A.I & it's heuristics based on the score at the end of the game. The score is calculated based on the cards that were put down correctly. It takes the sum of the biggest card of each stack. For example in the image below we have: 2 + 2 + 3 + 5 = 12

On average, skilled human players with clever strategies can achieve a score of around 17. The highest possible is 25 (5 * 5).

Our goal is to try to outcompete the human. Our A.I 'works' if it produces a high enough score.

Randomization:
A heuristic could produce biased results if the starter deck is static. As such, we randomize the starting deck and run many experiments (hundreds/thousands) until we have a stable average score. Each move decider being compared plays the game on a copy of the same shuffled deck.

Simple vs probabilistic

The simple (explicit) heuristic did very poorly (9.7) and it is very inconsistent. For some decks it does very poorly (~2.5) and for some it does better, but not that good. The probabilistic heuristic was much more stable and on average performed much better (13). The propabalistic function would yield consitent averages after only 100 runs, where as the simple heuristic would still produce inconsistent averages even after 100,000 runs.

	Explicit	Probabilistic
Failure %	0	0
Average Score	9.733	13.013
Max Score	16	19

Play Criteria (Discard Oldest, Maximize # cards told)

Tweaking probabilistic heuristic.

The probabilistic heuristic can accept various parameters. E.g ‘how risky’ it should play, discard strategies, safety threshold. We attempt to find the best possible parameters.

Discard parameter:
We have tried the different discard heuristics mentioned above (least likely playable card, least likely future, rare card). We thought that ‘least rare’ heuristic would always be the best. However, the data shows that “Least likely playable card” was actually on average the highest scoring discard heuristic:

Discard heuristic	Average score	Maximum score	Failure rate
Oldest card	12.48	17	0.0
Least playable card	**14.55**	20	0.0
Least future playable card	11.43	20	0.0
Least rare	10.86	19	0.0

We achieved reasonable averages with a sample size of 100 for each runs.

However, if we tinker with how advice is given, and change it so that players give as much advice about ‘rare’ cards as possible, then although we have low ‘average’ scores, the maximum achieved is higher. I.e, *bigger jackpot, but less often.*

Discard heuristic	Average score	Maximum score	Failure rate
Playability	14.02	18	0
Rarity, players give info on rare cards	10.67	**21**	0

It follows that playability has the best average, but rarity has the best chance of getting the highest maximum.

Probability parameter:
We have experimented with changing the probability at which a card is selected as playable. E.x select a card if we are only 60% sure it is the correct one, 70% etc.. We also experimented with different 'safety thresholds'.

Probability	Safety threshold	Average	Max	Failure rate
60%	90%	14.14	**22**	0.0
70%	90%	**14.54**	19	0.0
80%	90%	13.99	19	0.0
90%	90%	13.6	19	0.0
70%	**70%**	14.3	19	**0.04**
70%	**80%**	14.32	19	**0.02**
70%	**90%**	14.31	19	0.0
70%	**99%**	14.30	19	0.0

Changing the safety threshold to something low lead to more games failing (4%). However increasing it beyond 90% did not improve the score. I.e, if the A.I starts to make mistakes, it's best to resort to to playing it 'more safe'.

Playing at 70% probability yielded the highest average score. Playing too risky (60%) lead to lower average but higher maximum score.

Player advise Parameter
We experimented with how players give hints to each other. We had a lot of data, but the most interesting one was when additional points were given to 'playable cards'. I.e, if a card was playable, it was given more value by the other player.

Bonus points given to playable card	Average	max	Failure rate
1	14.08	21	0
2	15.52	20	0.
3	15.84	20	0
4	**15.87**	20	0

We found that giving more bonus to playable cards yielded the highest score.

Limitations / Obstacles

Variability of simple heuristic:

We found that the simple heuristic's performance heavily relies on the initial deck. In situations where the deck is favourable (i.e, one you draw playable cards early), the simple heuristic does very well, but it does very poorly on other decks. This is a limitation of the pessimistic nature of the simple heuristic to only act upon complete information. The simple A.I. also took many more runs to average out since a deck has roughly 25! combinations.

Performance of probabilistic heuristic:

The probabilistic move decider is fairly slow as it must calculate probabilities for every card in hand when deciding to play and when to discard. As such some of the test runs were on fewer number of games. Fortunately, because the amount of information that accumulates as the game goes on causes the average score to settle very quickly.

Conclusion

Our probabilistic A.I, with the best parameters achieved a top score 15.87. This is a much better score than a naive pessimistic A.I that achieved only a score of 9. While 15.87 is a good score, it is slightly below a human average of 17.

A human player can only keep limited information in mind about played cards. In particular humans are very bad at dealing with negative information (for example when told that one card is red, they can't recall that the other cards are not red).

While human players have a much more difficulty synthesizing information in the game, they can compensate for this by relying on conventions to do more with less information. For example, human players may tend to favour giving information about fewer playable cards rather than more total cards. Then the players assume that cards they were recently told about are playable.

Future work would probably involve adding special conventions so that the AI can be more efficient with information.

References:

- https://en.wikipedia.org/wiki/Hanabi_(card_game)