ACT-R model for cooperative board game Hanabi

Gavin Zhu (feiyuz)

April 29, 2022

1 Introduction and Background

1.1 Brief Introduction to Hanabi

Hanabi is a two to five players card game designed by Antoine Bauza in 2010. It won the best board game of the year (*Spiel des Jahres award*) in 2013 and is proposed to be the benchmark for developing cooperative AI [1]. The most distinctive feature of the game is its cooperative nature: all the players are working toward the same goal, each player cannot see their own cards, and they have to rely on the imperfect information from their partner to infer what their own cards are.

Rules

Each game has the following components which are shared by all players.

- 50 cards: five suits each of its own color; within each suite, there are 3 cards with rank 1, 2 cards each with rank $2, 3, 4$, and one card with rank 5
- 8 hint tokens
- 3 strike tokens

The goal of the players is to play the cards in order without repetition within each suit. That is, Red 2 can only be played after Red 1 is played, but is irrelevant to cards in other colors. Initially, each player is dealt with some cards (5 for 2 or 3 players, 4 for more players), they have to hold the cards outward such that they cannot see their own cards but their partner(s) can. Then each player takes turn to perform an *action* as listed below.

- **Give a hint**: pick a color or rank and tell one partner *all* his/her cards with that color or rank.
  - This costs one hint token, and can't be performed if there are no hint tokens left.
- **Discard a card**: pick a card in their own hand and discard it.
  - This will acquire one hint token, but can't be performed if there are 8 hint tokens left.
- **Play a card**: pick a card in their own hand and attempt to play it.
  - If the card is playable (of all the cards in the same color, one of each of the lower rank cards has been played, and none of the higher or equal rank cards has been played), it will be placed on to the board.
  - Otherwise, it will be put into the trash pile and one strike token will be taken away. But no hint token will be given back.

The game ends if one of the following happens:

- Full board: when there are 25 cards successfully played to the board.
- Strikeout: when there are no strike tokens left.
- Run out of cards: after the deck is empty, each player has one last turn before the game ends.

In the end, the score of *all* players is the number of successful plays (25 maximum).

Challenges

The main challenge of Hanabi arises from the limited information a single hint can provide. Suppose a player is in the situation illustrated in **Figure** 1, where the partner has a playable card Yellow 2 which is the only playable card. And the partner currently knows nothing of the hand. At this time hinting *two* will inform the partner the second and four cards (from the left) are twos, and hinting *Yellow* will inform the partner the first, second, and fifth cards are yellow. In both cases the hint is ambiguous and the partner cannot know for certain which card to play. Each player may have there own way of resolving the conflict (e.g. refer to the rightest one, wait for next hint, etc.), and misinterpretation may lead to sooner end of the game (such as in this case referring to the rightest one will play Red 2 or Yellow 1 which are not playable).

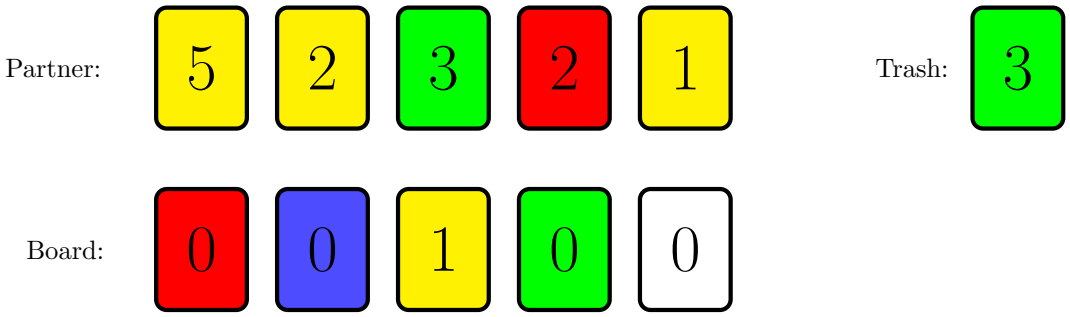


Figure 1: A scenario that may cause ambiguity (best view in color)

Other challenges are concerned with strategies. Discarding a 5 will result in not being possible to get a perfect score, but will increase the chances of getting a high score, how to decide what to do? If there are no hint tokens left and none of the cards in the hand is known to be certainly playable, should the player discard for safety or take the risk and try to play a card with partial information?

1.2 Existing Hanabi Agents

There are many approaches to developing Hanabi playing agents. The most popular paradigm is to formulate Hanabi as a multi-agent reinforcement learning problem. Hu et al. [2, 3] formulates the game as a partially observable Markov decision process, and trains deep reinforcement learning agents based on self-play and cross-play (where the agents play with similar agents in the same family but with different parameter settings). It is shown that self-play training converges to a descent strategy that yields an average of 24 out of 25 per game, but it is not generalized such that it does not perform well when paired with other agents (average about 15 out of 25). Techniques such as cross-play and auxiliary losses enable more generalization, yielding around 23 when paired with agents in the family, but still cannot coordinate well with humans (average about 16).

Unlike deep reinforcement learning which is only concerned with scores in the game, the Intentional agent [4] is designed based on how humans typically approach the game. They reasoned that each action must have an "intention", and try to interpret the hints based on possible intentions. Overall the performance is not as good as the deep reinforcement learning agents (average about 17), but because the way it plays is intuitive to humans, the score, when paired with human partners, is roughly the same.

There are also theoretically based works that analyze the game in terms of information theory. Bouzy [5] abstracts the game into an NP-hard problem and comes up with an approximation algorithm using tree searches that play near optimally. Although it achieves the great performance (achieving a perfect score of more than 90% of the time), it can only be used in games with more than 2 players, where there is a sufficient number of cards disclosed for each player, and cannot coordinate well with other types of agents.

1.3 Baseline Agent

The first step for the ACT-R model is to learn to coordinate with an existing agent. In this project, an agent with a simple, fixed strategy is used to be the baseline that ACT-R model learns from and compares to. The general idea of the baseline agent is to prioritize play over hint, hint over discard, and never take risks. Specifically, it will linear pattern match the following set of rules during the gameplay.

1. If a card is known to be playable, play the card

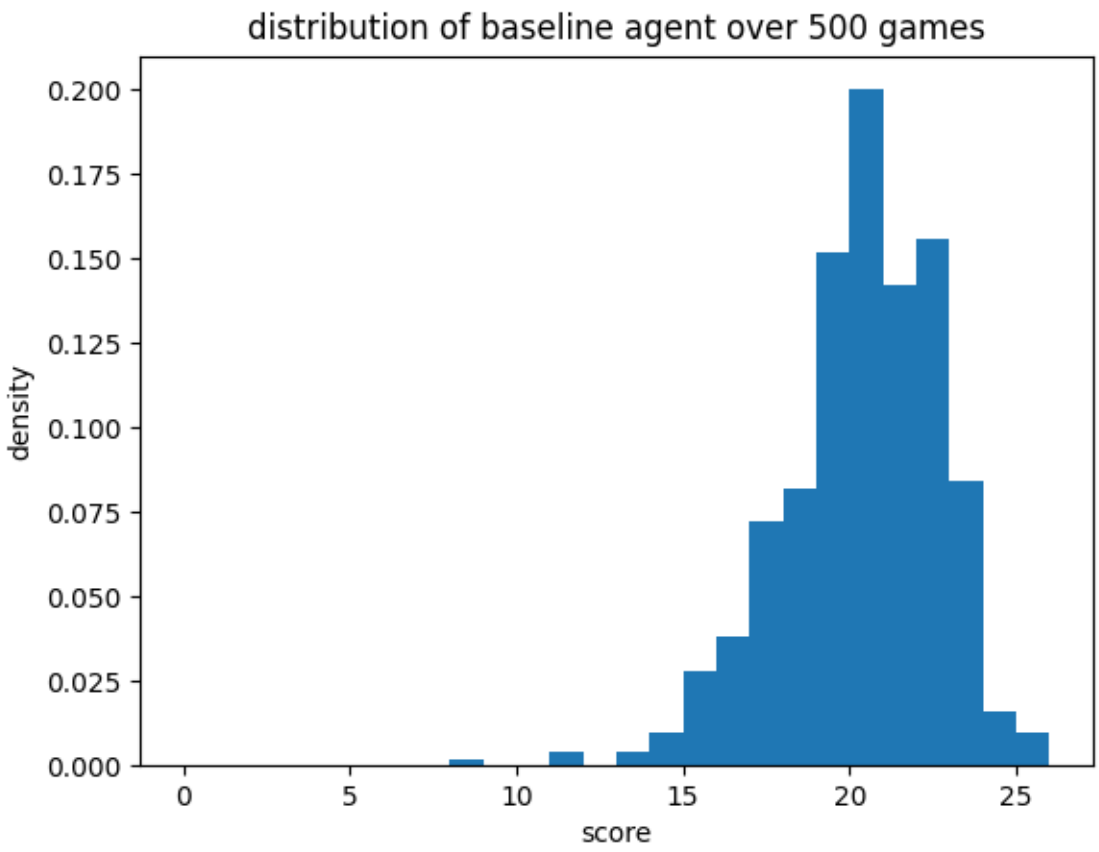


Figure 2: Score distribution of the baseline agent

2. If a hinted card is not known to be unplayable, play the card (interpret the partner's hint as a hint to play)
3. If a hint refers to multiple cards, associate it with the rightmost (newest card)
4. If a partner's card is playable, and can be hinted at unambiguously, hint at the card
5. If a card is known to be unplayable, discard the card
6. If a partner's card is unplayable, and hinting at it will not be misinterpreted as a hint to play, hint at the card
7. If there are sufficient number of hint tokens left, hint the partner for most information gain
8. If none of the rules above matched, discard the leftmost (oldest) card that has never been hinted

The baseline agent performs reasonably well given its simple implementation, averaging around 19 out of 25 per game when playing with itself. The distribution of scores is shown in Figure 2. Most games are in the 18 to 23 range indicating the baseline agent's performance is pretty consistent, but since the strategy does not consider many of the edge cases getting a perfect score is very rare for the baseline agent.

2 Model Details

2.1 Representations

Basic information about the game includes the number of hint tokens, the number of hit tokens, and the board (how many cards have been played in each color). As they only hold a small set of information, it is reasonable to assume human players can keep them in mind without having to revisit the game. Therefore in the model they

are directly set in the goal buffer by the game environment. Under this setting, the model can directly test the number of hint tokens left to determine whether hint action is available, and can also make use of dynamic pattern matching to check the board easily.

For a card that is fully known (i.e. partner's hand or the trash pile), it is represented as a customized visicon in ACT-R with the following chunk type

```
(chunk-type (card-obj (:include visual-object))
   color     ;; one of {blue green red white yellow}
   rank      ;; one of {one two three four five}
   owner     ;; one of {partner, trash}
   index     ;; nil for trash, 1 - 5 for partner's hand
             ;; (1 being the leftmost, 5 being the rightmost)
   count     ;; nil for partner's hand, number of cards discarded for trash (possibly zero)
)
```

There are a total of 30 such objects visible at a time for the model, 5 for the partner's hand, and 25 for cards in the trash. The model will use the visual module to attend to one card at a time and have access to all the information needed, similar to human players.

To represent the model's knowledge of its own card, and also keep track of the partner's knowledge of their card, another customized visicon with the following chunk type is used

```
(chunk-type (knowledge-obj (:include visual-object))
   blue green red white yellow    ;; whether it is possible that the card is of each color
   one two three four five        ;; whether it is possible that the card is of each rank
   color rank                     ;; set to the color / rank if full known, otherwise set to nil
   hinted                         ;; whether the card is hinted at in last round
   index                          ;; 1 - 5 for each of the player
   owner                          ;; one of {partner, model}
)
```

The representation of knowledge is inspired by an online Hanabi playing platform[1] where the slot values are rendered in colors and shapes for humans. There are a total of 10 such objects in the game, 5 for each player, and similar to the card object the knowledge object can also be attended by the visual module. Below are examples of a newly drawn card for the model and a red card in the partner's hand just hinted at by the model.

```
;; newly drawn card, knows nothing yet
(
   blue t  green t  red   t  white t  yellow t  ;; all colors are possible
   one  t  two   t  three t  four  t  five t    ;; all ranks are possible
   color nil   rank nil                        ;; none of the color or rank is known
   hinted nil                                  ;; freshly drawn, not hinted yet
   index 5                                     ;; 5 corresponds to the rightmost card
   owner model
)

;; a red card in the partner's hand just hinted by the model
(
   blue nil  green nil  red   t  white nil  yellow nil  ;; only red is possible
   one  t    two   t    three t  four  t    five   t    ;; all ranks are possible
   color red   rank nil                                ;; color is known to be red, rank unknown
   hinted t                                            ;; just hinted by the model
   index 5                                             ;; 5 corresponds to the rightmost card
   owner partner
)
```

[1] https://hanabi.cards/

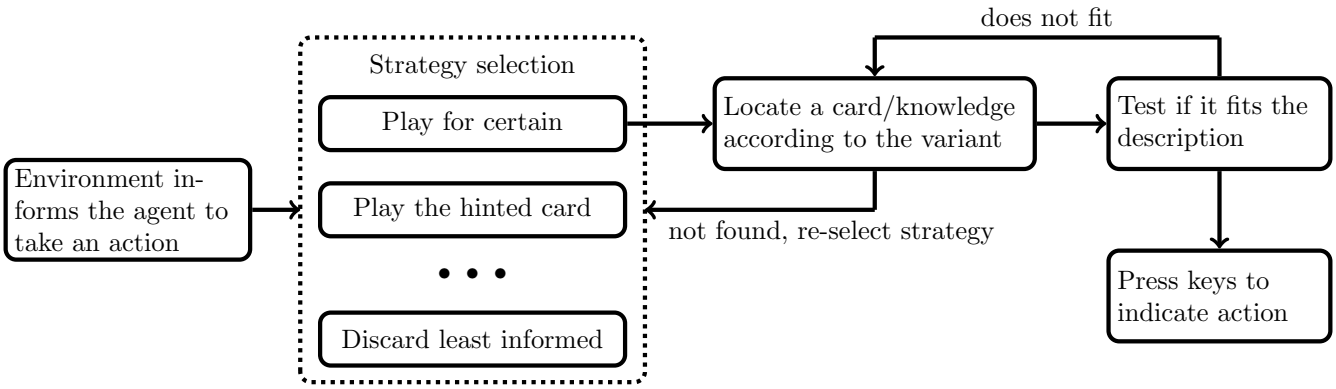


Figure 3: Model overview

2.2 Model Implementation

Model overview

As shown in Figure 3, the model starts when it is its turn. First, it will choose, based on the learned utility, a strategy and its variant (details covered in the following subsections). Then based on the variant, it tries to find and attend to a card / knowledge which can potentially fit the precondition of the strategy. If no such object exists, the model falls back to re-select another strategy. Otherwise, the model will test if the card / knowledge fits the precondition, if it fits then proceed to press keys, which ends this turn. But if the attended card / knowledge does not fit the description, the model will fall back to locating a new card / knowledge.

To keep track of strategies tried and avoid an infinite loop, there is a set of binary variables in the goal buffer indicating whether a strategy has been tried or not. When a strategy is attempted but failed, its corresponding variable will be set to nil so that it will not be attempted again. At the beginning of each turn, the variables are reset so that every strategy has the opportunity to be executed.

Play for certain

If a card is known to be playable, it is intuitive to play the card. When determining whether a card is certainly playable, one of the following scenarios will occur

- If both the rank and the color are known, the playability check is easy: dynamic pattern matches on the board in the goal buffer and test if the rank of the card is the next to the one on the board.
- If only the color is known, but the rank is unknown (at least two possibilities), the card can definitely not be playable, since for each color there is at most one rank be playable at a time.
- Similar to the previous case, if none of the rank or color is known for certain, the card cannot be definitely playable.
- If the rank is known but the color is not, the model will have to iterate through all the possible colors and check if the rank fits the board. If all of the colors are either not possible or have the appropriate rank on the board, the card is known to be playable (e.g. at the very beginning when no card is played, a rank 1 card is known to be playable even if the player knows nothing about its color), otherwise it is not.

The variants of this strategy include play from left or right, that is if there are multiple playable cards, which specific one to play.

Play in uncertain

Playing a card when it is not fully known to be playable is not an available option in the baseline agents. However, since it is possible in human plays so it is included as an option in the model, with an assumption that humans don't ever play a card that has never been hinted at before.

The model implementation makes use of visual search to perform this action. Instead of iterating through each card, the model will start with the board and do a visual search to find whether there are potentially playable

cards in its hand. The variants of the strategy are to only consider playing in uncertain when there is no misplay previously in the game (i.e. the game will not end even if the current play is unsuccessful).

Play the most recently hinted

Playing the most recently hinted works similarly to the play in uncertain since both of them requires the card to have at least one of the color / rank information (which is automatically satisfied after a hint) and only requires it to be potentially playable instead of certainly playable. So the testing procedure is the same as play in uncertain, and the only difference is instead of any cards in the hand this strategy focused on the hinted cards (which can be tested using the "hinted" slot in the knowledge). The variants of this strategy are to play the leftmost or the rightmost hinted cards when a hint is associated with multiple cards. Note that the variant of play the rightmost is compatible with the baseline player while play the leftmost is not.

Hint playable card

As mentioned in the previous section, hint for play is a commonly used strategy. The model always tries to hint unambiguously assuming the partner follows some deterministic conflict resolution procedure (e.g. associate the hint to the rightmost hinted card). For the hint from the right variant, the model will iterate through all the partner's cards from right to left, and for each of the cards, check if there are other cards with the same color or rank on the right. If there is none, then it means the playable card is the rightmost card with its color or rank and the hint is unambiguous to a right-associative agent. And therefore a hint will be made. If both the color and the rank are not available (e.g. the scenario illustrate in Figure 1 for yellow 2), the model will not give a hint. Another variant of this strategy is to assume the partner associates the hint with the leftmost card.

Discard useless card

If a card is known to be unplayable, it should be discarded. The model tests if a card is definitely unplayable using the following procedure

1. attend the card
2. find the minimum rank on the board with one of the possible colors for the card
3. find the maximum possible rank of the card
4. if the maximum possible rank of the card is less than the minimum playable rank, then the card is definitely unplayable
5. otherwise the card is still potentially playable

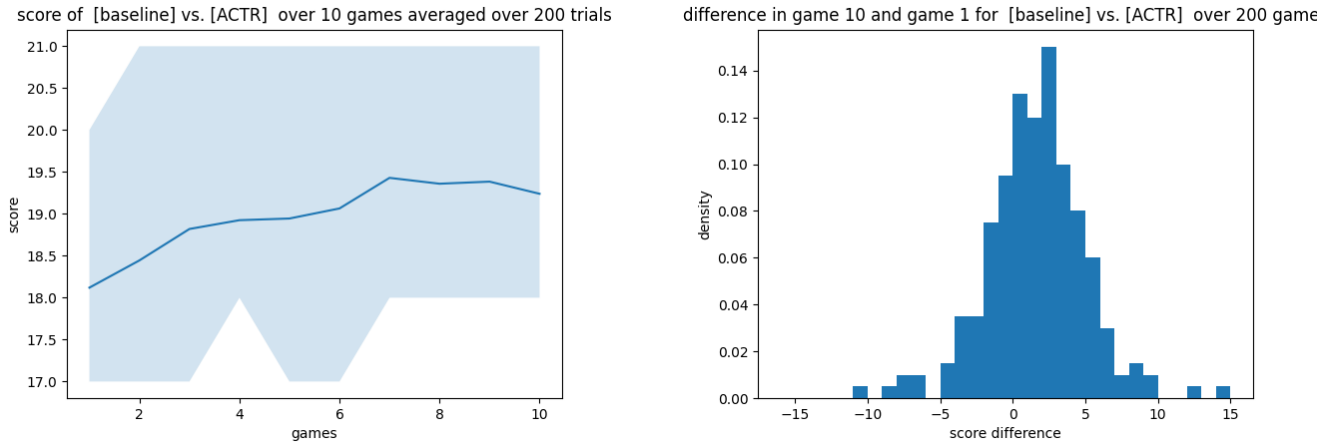
The variants of this strategy include discard from left or right, that is if there are multiple useless cards, which specific ones to discard.

Discard unhinted

Since it takes more turns to know about unhinted cards than those that have been hinted at before, it is reasonable that the unhinted ones should be prioritized at discard. The test for an unhinted card is straightforward with the current knowledge representation: if both the color and rank are unknown for a card, then the card is unhinted. The variants of this strategy include discarding from left or right, that is if there are multiple unhinted cards, which specific ones to discard.

Discard at random & hint at random

Sometimes there are no or all eight hint tokens, making many of the strategies not applicable. Under these circumstances, the model has to have some default actions. Discard at random and hint at random, as the name implies, randomly attend to a knowledge in the model's own or a card in the partner's hand and try to discard or hint at it. These are a pair of fallback strategies when all other strategies are not applicable.



(a) Performance curve of preference learning

(b) Distribution of difference between last and first game

Figure 4: Results for learning preferences

2.3 Utility Learning

The model makes use of utility learning to learn to cooperate with the partner. Specifically, a reward is given when either the model or the partner plays or discards a card. If a card is successfully played, or a useless card is discarded, the model is provided with (different) positive rewards. And if a card is unsuccessfully played, or a playable card is discarded, the model is provided with (different) negative rewards. If an unimportant card is discarded the model learned nothing and the utility trace is cleared.

Under this paradigm, play and discard actions will get immediate feedback, but hint productions will have to wait for the next play or discard. Also, the reward only acts on the executed strategy. If the model attempts a strategy but failed (e.g. try to play for certain but cannot find any known playable card), the utility learning trace will be cleared and the strategy's utility will not be changed.

2.4 Code Distribution

The implementation for the model and experiment environments is provided at:
https://github.com/zfy0314/hanabi-actr

3 Learning Strategy Preference

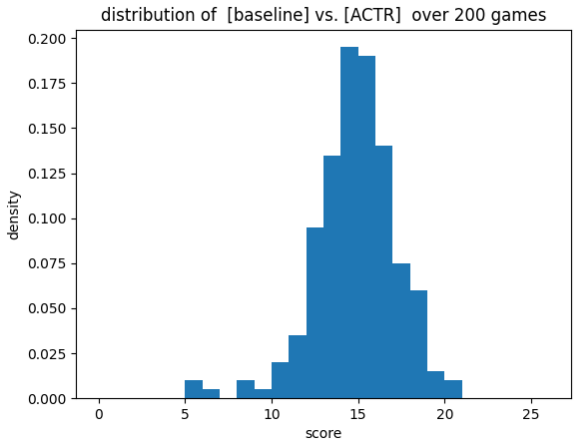
Setup

The model is only provided with the strategies and their specific variants in the baseline agents (e.g. there is only "play the rightmost hinted card" but no "play the left most hinted card"), but all with equal utility (i.e. no preference between the strategies). And it is expected that through utility learning the model can learn a reasonable ordering of the strategies by playing with the baseline agent so that it can cooperate with it in the end.

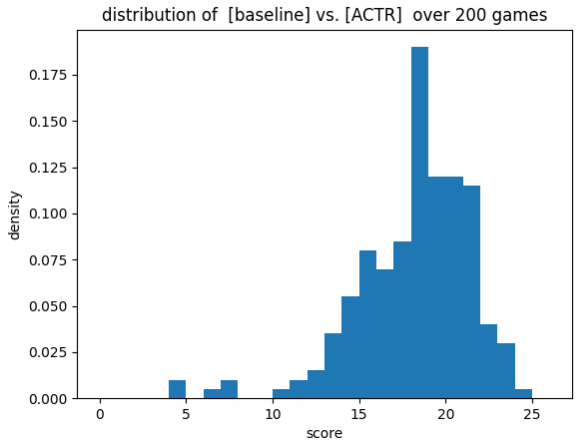
At the start of each trial, the model is reset to have no preference, then it will play 10 games with the baseline agent, using utility learning through the games to adjust the utility for each strategy. In the end, the effect of learning is evaluated by the change in the average score of different games in the trial. If the model is learning, the last game should have a higher score than the first. To mitigate the variance of the game, there are 200 repeated trials and a total of 1000 games.

Results

The performance curve is shown in Figure 4a, where the solid blue line represents the average of the games, and the upper and lower bounds of the shaded area are the first and third quantile of the 200 games. The figure shows a clear upward trend, and a simple linear regression using R [6] for the score on the index of the game shows that the slope is significant on the 0.001 level, indicating that the model is indeed learning. Also Figure 4b shows the



(a) Distribution of one game without utility learning



(b) Distribution of one game with utility learning

Figure 5: Comparison between with and without utility learning for preference learning

distribution of the difference between the last game and the first game (last − first). 175 out of 200 (87.5%) of trials shows an positive improve in score.

Figure 5 compares the performance of with and without utility learning. It is shown that a model with no utility learning yields an average of around 14.5, but the first game in each trial with utility learning averages around 18, showing that the learning is effective as early as the first game in each trial.

The learned utility is shown in Table 1. In general, it learns to fit the baseline agent's priority preference, that is play over hint and hint over discard. But it ranks *play recently hinted* to be the first instead of *play for certain*, which is unintuitive. Further experiment on monitoring the utilities of different strategy over 20 games (Figure 6) suggests that this is because at the beginning there is no definitely playable card, so the *hint to play* and *play recently hinted* are fired and receive positive reward when the partner successful play the hinted card or hinted at a playable card, which give them a head start in the increasing utility. At the end of the 20th game the utility of *play for certain* is catching up, and is basically at the same level as *hint to play* and *play recently hinted*.

Another interesting phenomenon is that the model ranks *discard random* over other discard strategies. This is probably because it takes the least number of productions to decide, so it gets more reward. And that most cards are neutral so discarding randomly will not incur many penalties.

Strategy	Rank	Strategy	Rank
Play for certain	3	Play in uncertain	4
Play recently hinted	1	Hint playable card	2
Discard unhinted	7	Discard useless	6
Discard at random	5	Hint at random	8

Table 1: Strategy preference after 10 games for preference learning

4 Learning Strategy (Variant) Selection

Setup

In this experiment, in addition to the variants in the baseline agent, the model is also provided with strategy variants that are incompatible with the baseline agent (e.g. associate hints to the leftmost card). It is expected that the model should learn which variant is better for pairing with the baseline agent and give them higher utility. Similar to the previous experiment, the learning will be evaluated as the score increase in the game, and there are 200 trials and each trial has 10 games.

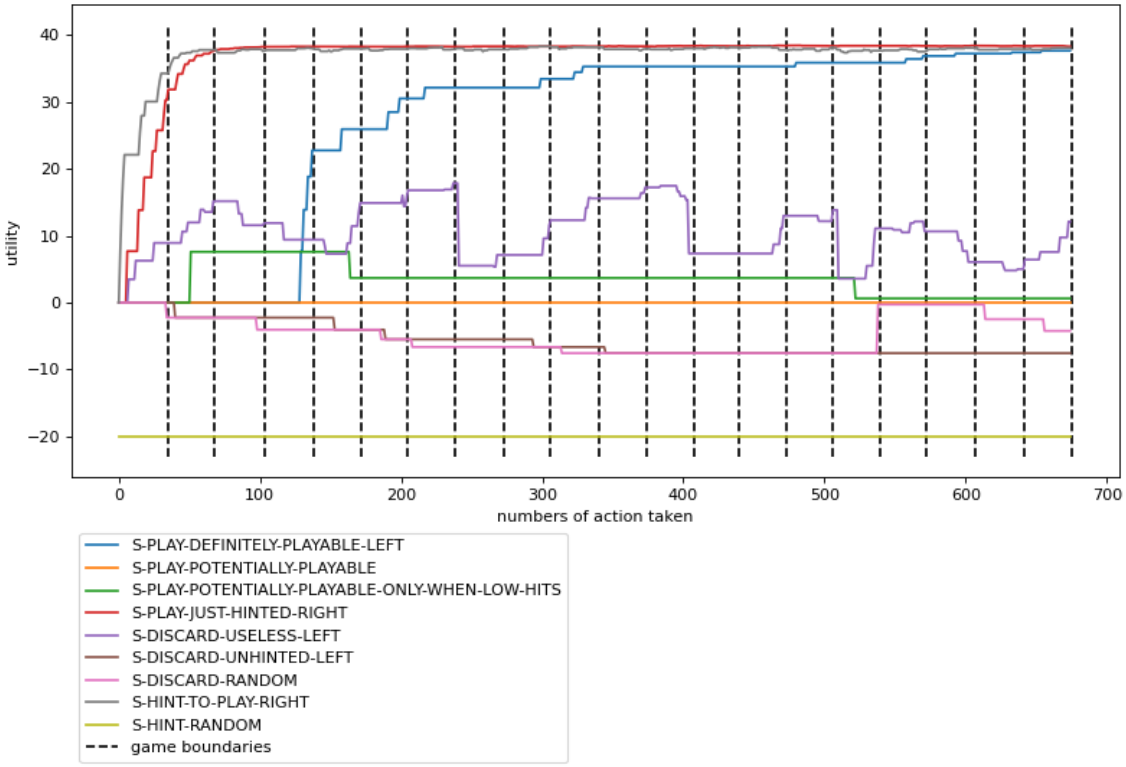


Figure 6: Utilities of each production for preference learning over 20 games (best view in color)

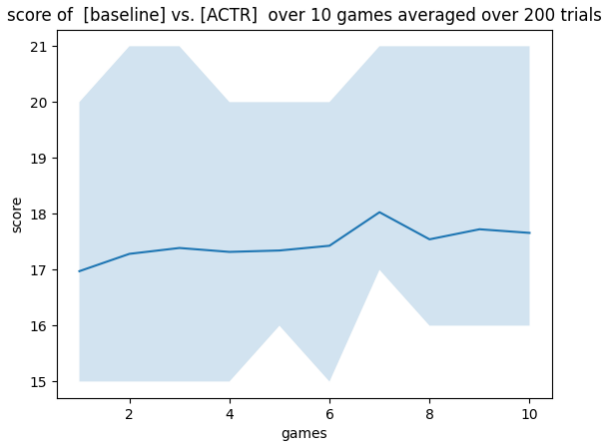
Strategy	Rank	Strategy	Rank
Play for certain from left	11	Play for certain from right	3
Play in uncertain at all times	10	Play in uncertain when no strikes	5
Play recently hinted from left	9	**Play recently hinted from right**	1
Hint playable card from left	7	**Hint playable card from right**	2
Discard unhinted from left	13	Discard unhinted from right	12
Discard useless from left	4	Discard useless from right	8
Discard at random	6	Hint at random	14

Table 2: Strategy preference after 10 games for variant selection

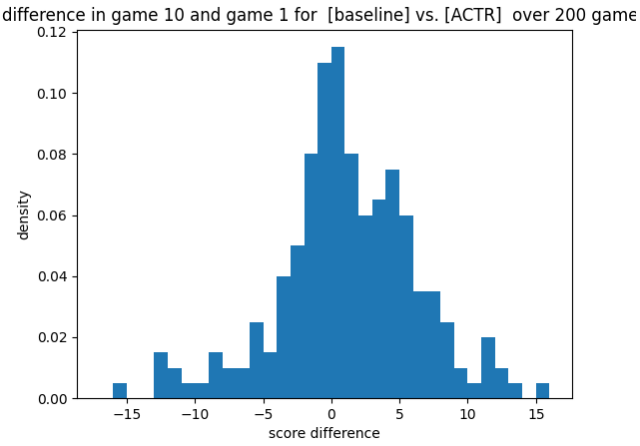
Result

The performance curve is shown in Figure 7a, and like the previous, the solid blue line represents the average of the games, and the upper and lower bounds of the shaded area are the first and third quantile of the 200 games. The upward trend in the graph is less clear in the graph compared to the previous experiment, but simple linear regression shows that the slope is still significant at the 0.05 level. The distribution of difference in Figure 7b shows more variance than the preference learning. 121 out of 200 (60.5%) of the trials achieve higher score in the last game than the first game. And similar to the preference learning, Figure 8 shows a model without utility learning has an average of 14.3 while the one with utility learning has an average of 17 at the first game. However, the average of the games dropped. This is because the variants have overlaps (e.g. if only one card fits the hint, hinting from left or right is identical), so the model may oscillate between different variants, hampering the game performance.

The learned utility is shown in Table 2 where the boldface strategies are consistent with the baseline agent. It is shown that the model successfully learns to choose the consistent variants for *play recently hinted*, *hint playable card*, and *discard useless*. For *play for certain* the variants do not matter much since the card is known to be playable anyway, so it is not surprising that the model prefers playing from the right. And as for *discard unhinted* since it is no better than discard random so its variants also do not have much impact.

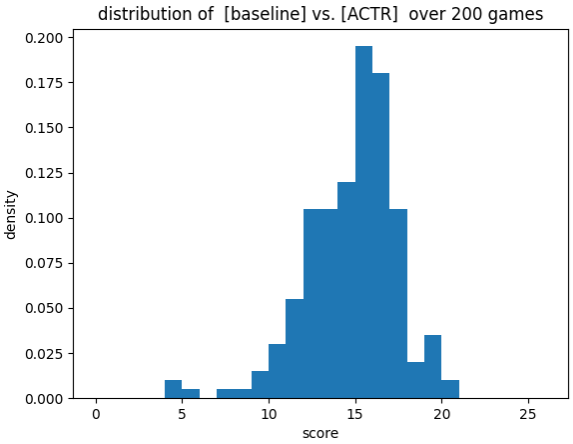


(a) Performance curve of variants learning

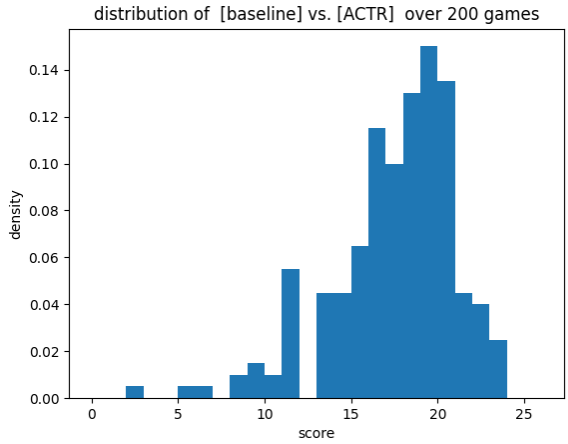


(b) Distribution of difference between last and first game

Figure 7: Results for learning variants



(a) Distribution of one game without utility learning



(b) Distribution of one game with utility learning

Figure 8: Comparison between with and without utility learning for variant learning

5 Concluding Remarks

In this project, an ACT-R model is designed to learn to coordinate with an existing agent in the cooperative board game Hanabi. It is shown that using utility learning the model achieves satisfactory results in learning the preference of the strategies. But on the strategy selection task, although the model still demonstrates learning and converges to a reasonable strategy ordering, it fails to achieve the same performance.

Compared to the deep learning models, the ACT-R model requires significantly fewer games to learn and has a learning curve closer to the learning curve of humans. And the ACT-R model, even under the strategy selection setting, also achieves better performance than the previous Intentional Agent which is based on the human theory of mind.

6 Acknowledgement

I would like to thank Prof. John Anderson and Dan Bothell for their feedback on the project idea and ACT-R implementation.

References

[1] N. Bard, J. N. Foerster, S. Chandar, N. Burch, M. Lanctot, H. F. Song, E. Parisotto, V. Dumoulin, S. Moitra, E. Hughes, *et al.*, "The hanabi challenge: A new frontier for ai research," *Artificial Intelligence*, vol. 280, p. 103216, 2020.

[2] H. Hu and J. N. Foerster, "Simplified action decoder for deep multi-agent reinforcement learning," *arXiv preprint arXiv:1912.02288*, 2019.

[3] H. Hu, A. Lerer, A. Peysakhovich, and J. Foerster, ""other-play" for zero-shot coordination," in *International Conference on Machine Learning*, pp. 4399–4410, PMLR, 2020.

[4] M. Eger, C. Martens, and M. A. Córdoba, "An intentional ai for hanabi," in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 68–75, IEEE, 2017.

[5] B. Bouzy, "Playing hanabi near-optimally," in *Advances in Computer Games*, pp. 51–62, Springer, 2017.

[6] R Core Team, *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013.