Fight the Landlord (Dou Di Zhu)

Zhennan Yan, Xiang Yu, Tinglin Liu, Xiaoye Han

Introduction

Fight the Landlord (Dou Di Zhu) is a climbing game primarily for three players. In the game, one player, the "landlord", plays alone and the others form a team. The landlord's aim is to be the first to play out all his cards in valid combinations, and the 'Farmer' team wins if any one of them manages to play all their cards before the landlord[1-2].
The game is very popular all over China, and is also extensively played on line.[3-7]
Because the game is very complicated, we properly cannot accomplish it in one month. So we made some abridgement to the original rule to make it simple enough to be finished and easy to be understood.

Players, Cards and Deal

This game uses a 54-card pack including two jokers, red and black. 4 suits(Spade, Heart, Club, Diamond) with 13 cards in each.
The cards rank from high to low:
red joker, black joker, 2, A, K, Q, J, 10, 9, 8, 7, 6, 5, 4, 3.
Suits are irrelevant (no ranking between suits).

Cards are shuffled randomly, distributed to players counter-clockwise around the table until each player has 17 cards. Each player can only see his own cards. The last three cards are visible to everyone and belong to the landlord. So landlord has a total of 20 cards.

Valid cards types (combinations)

In this game, there are five types of combination that can be played:

1. **Single card -** ranking from '**3**' (low) up to red **joker** (high) as explained above
2. **Pair -** two cards of the same rank, from '**3**' (low) up to '**2**' (high), two **jokers** are not pair
3. **Triplet** - three cards of the same rank
4. **Quad -** four cards of the same rank
5. **Sequence -** at least five cards of consecutive rank, from '**3**' up to **ace**, for example '8-9-10-J-Q'. **Twos** and **jokers** cannot be used.

Play rules

The landlord plays first, and may play any single card or any legal combination. The played cards can been seen by everyone. Each subsequent player in counter-clockwise order must either pass (play no card) or beat the previous play by playing a higher-ranking combination of the **same number of cards** and **same type**. (No ranking between different types or different number of cards).
Example: 2 **aces** cannot beat 3 **tens**.

The play continues around the table for as many circuits as necessary until two consecutive players pass. The person who played the last card(s) begins again, leading any card or legal combination.
Note that passing does not prevent you from playing on a future turn.

Aim of designing

Fight the Landlord is a partially observable, multi-agent adversarial game.
We wanted to modify the game to be standalone version instead of 3 people involved. So we needed two agents as farmer to fight with the landlord. As the game itself is complex, the agents should be smart enough to cooperate and beat human beings in the game. This is an challenging attempt to achieve artificial intelligence.
We can learn and review what we learned from the class by practicing, also we can get a good game for entertainment at some free time.

Example of expected cooperation

Player A (the landlord) leads 3-3-3 to get rid of some low cards, player B passes, player C plays 5-5-5, player A plays K-K-K and player B plays A-A-A. C and A pass, so B can start again with anything. He leads a single 4.
Note B could have played his aces on his the first turn, but preferred to pass to give his partner a chance to get rid of some cards. C will now play if possible, so as not to give the landlord (A) a free chance to lead again. Having beaten A's second play, B leads a low card to give C the choice of playing another unwanted card or putting the landlord under pressure by playing a high card.

Design

Framework:

- Play table: shows cards discarded by each player, displaces current status, announces the winner
- Game control: shuffles and distributes cards at the beginning, judges validity of each pile of cards discarded, judges when to terminate the game
- Agent as farmer: two agents with different levels of intelligence representing farmers
- Human or Agent as landlord: the landlord could be either human or agent designed.
- Logging system: keeps logs for testing and evaluations

Agents: Assume all agents can analyze their own cards.

- **Random Agent**: believes the playing is stochastic process, plays valid cards randomly, doesn't consider cooperation, doesn't remember history, doesn't plan for future
- **Simple Agent**: plans for future, plays the low-ranking valid card(s) first if possible
- **Advanced Agent**: analyzes different contexts, and plays conditionally according to decision tree and sometimes probabilities.
- **Predictive Agent**: remembers cards played and related context, predicts the possible remaining cards in other players' hands, plays card(s) as optimal as possible

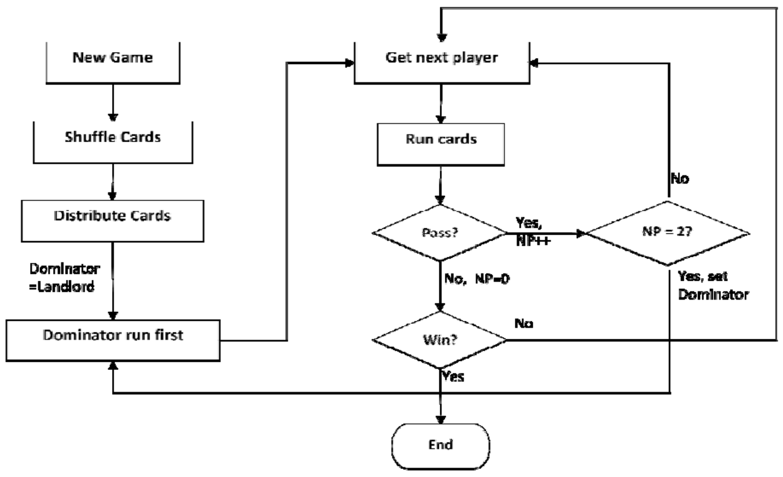
Experiments:

- Comparisons among different agents by competing with human and competing with each other kind of agent

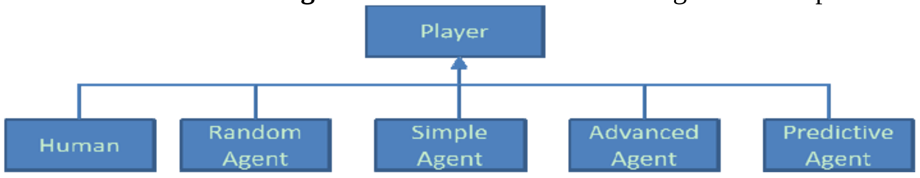
Implementation

Development environment: We use QT 4.7.0, C++ Compiler, CMake. (These make a cross-platform project).

Game main flow diagram:



Interfaces for different agents: Same in framework through common parent class.



Basic and key operation for all agents: Cards state division.
Implementation aim: get a reasonable state division to help to analyze own cards and to guide how to play. This is an optimal heuristic distance to win.
if cards are divided into 5 states, player needs at least 5 steps to win.

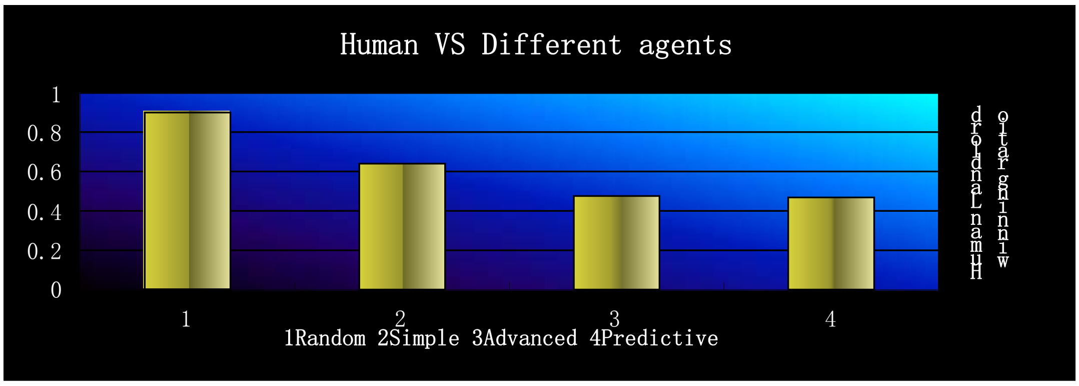
Testing & feedback

- This game is a partially observed game involves three players
- It is hard to tell what play is good at that very moment.
- We have to improve our designs and implementations by testing and feedback.
- So our game provides some extra features:

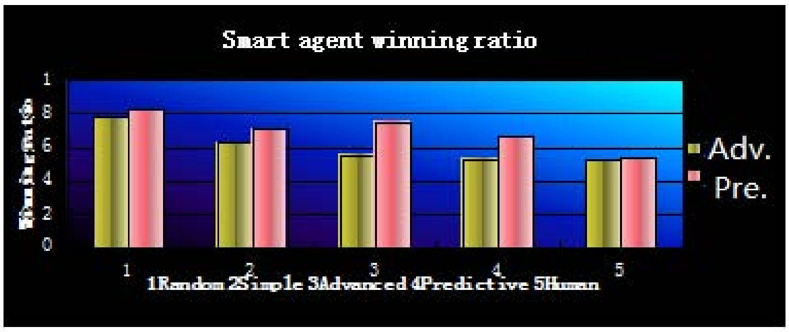
  Have option to show farmers' cards, which actually cannot be seen in real game
  Can log game histories for later analysis
  Can load a custom-assigned cards distribution to players (random distribution default)

Evaluation

Winning ratio for human landlord VS different Farmer agents



Winning ratio for Advanced or Predictive Agent as farmers VS different Landlord



Results table for different players combination

Farmers\Lanlord	1 Random	2 Simple	3 Advanced	4 Predictive	Human
3 Advanced	0.78125	0.62645348837	0.5462962963	0.52777777778	0.52459016393
4 Predictive	0.83018867925	0.708502024429	0.75	0.659793814433	0.53125

References

[1] http://www.pagat.com/climbing/doudizhu.html
[2] http://en.wikipedia.org/wiki/Dou_Di_Zhu
[3] http://www.chinagames.net/mygames/cardgames/landlord/
[4] http://en.auway.net/Games/play/Dou-Dizhu.htm
[5] http://www.viwawa.com/en_US/game/HarvestTime
[6] http://www.facebook.com/apps/application.php?id=54153958094
[7] http://mgame.ourgame.com/game.aspx?gameid=20010&modeid=0
[8] http://itunes.apple.com/us/app/id342994828?mt=8
[9] http://www.pokerddz.com/

Fight the Landlord - Subtask description

Zhennan Yan

1. SVN server management

At the beginning of the project, considering there may be many joint efforts between team members, and many modifications during the project implementation, I decide to use SVN application to manage the subversions of our project. This is proved very necessary at later time.

2. Game framework implementation, GUI design

Referring to the GUI of some online games for Dou Di Zhu, I design our GUI using QT, a cross-platform application and UI framework. So we can build different versions for different platforms. So far we have Windows version and Mac version of our game.
Because the framework should provide all the interfaces with humans as well as different kinds of agents, we used class inheritance skill by object-oriented programming language like C++. And provided common interfaces for different kinds of agent.
I constructed the whole project framework, so my team member can implement respective subtasks easily.

3. Implementation for Random Agent

3.1 Cards state division

Before implementation for agent, we need a basic function for dividing cards into some valid type states (we call this result as state space). This is used by agent to analyze its own cards. A good division of some cards should have the least number of states. This number (N) is an optimal heuristic distance to winning. Assuming the other two players pass all the time, the agent has to play at least N times to win.

At first, we used greedy algorithm: search the longest cards type first, typically the sequences; then search quads, trips, pairs in turn; at last, the rest cards are singles.

This method works well but not optimally. For example, if we have cards 3-4-5-6-6-7-7-7-8-8-9-9-10, the greedy algorithm will output 5 states after division: first is the longest sequence 3-4-5-6-7-8-9-10, then pair of 7-7, at last three single cards 6, 8 and 9. However the optimal division should be 3-4-5-6-7-8-9, 6-7-8-9-10, 7, which only contains 3 states.

So the improvement of the algorithm is that: firstly, does division using greedy method; secondly, excludes all sequences with 5 cards, because these cannot be improved; thirdly, removes some cards from the sequences with more than 5 cards, puts these removed cards together with the cards not in any sequence and does division using greedy method again on these pieced-together cards, keeps the division which has less states. Repeat the third step until no possible reorganization exists. At last, we can get the optimal division with the least number of states.

Actually, this is like the backtracking search.

3.2 **Random Agent**

Random Agent plays cards only based on the optimal state division results. So it has enough intelligence that won't play any invalid card(s) or won't play single 3 when it has triplet 3.

The agent will play a valid state in the optimal state space randomly. So if previous player just played a ‘3’, possibly it will play a Joker when it also has a ‘4’. The agent may regret in future when it really needs a Joker to beat other player. But we can't say this is so bad strategy, since the game has many uncertainties. For example, if another player, say farmer 1, has one last card in hands which is a single K, and the landlord has just 4 and joker left, then playing joker first definitely is a successful strategy.

5. Testing and feedback for modification of all agents

This game is a partially observed game involves three players, and two farmers should cooperate to beat landlord, it is impossible to tell what play is good at that very moment. Even different people have different prefers faced on the same situation. So it is very hard to generate fixed strategies for the game. And hard for us to implement the agent once-and-for-all.

We have to improve our designs and implementations by testing and feedback. So our game framework provides some extra features:

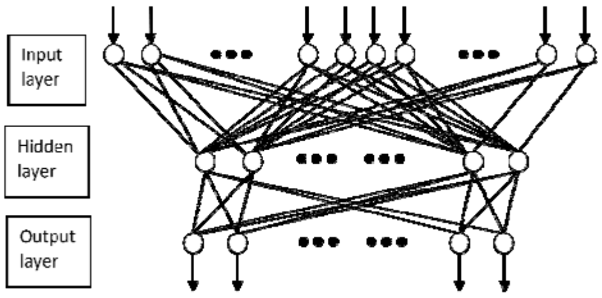
1) Has option to show farmers' cards to observer (human), which actually cannot be seen in real game
2) Can log detail game histories for later analysis
3) Can load a custom-assigned cards distribution to players (random distribution default)

6. Other works

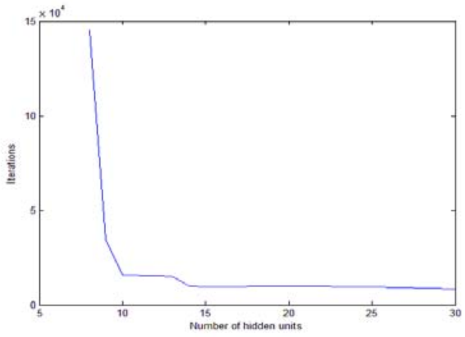
Other works include debugging and testing on the framework and all agents to make our project work well as a whole, searching for references and rearranging our final overview documents.

7. Future work

We wanted to try some learning methods for an agent. And designed a neural network as fig.1:



(fig. 1)



(fig. 2)

There are 30 units for input layer, in which 15 units for status of cards in hand, 15 units for current play in table; there are 15 units for output layer, representing desired play. The number of units in hidden layer is not sure, so I made some experiments: using 44 training data to train the network with different number of hidden units, set a fixed threshold, then record the number of iterations for different network. The result is showed in fig. 2. So we choose to use 20 hidden units in case it won't converge well for large number of data. But unfortunately, when I applied more complicated data to it, the network didn't converge after 2 days' training. Considering the limited time, we just gave up.

In future, we may do more work on some learning methods for our agent.

Report by Xiang Yu

Part A. content I present in the related slides

1. The Agent Design

What I concern about——the agents with different levels. First one is the Random Agent, which is also the simplest one. The Random Agent believes that the process of playing is stochastic. There are two conditions each turn during the play: one is currently the agent is the dominator, that means no other players can discard larger value cards; the other is the agent is not the dominator, it needs discard larger value cards in order to get the domination. For the first condition, the random agent will randomly choose what it has in hand to discard; for the second condition, the agent will randomly choose among those cards whose card power is larger than the current dominator.

The second one is Simple Agent. Simple agent design is based on several basic experiences from human beings. These are: (1) run out as many cards at a time as it can; (2) try to maintain the largest card power in hand; (3) the domination is very important. Thus, we designed the card power to evaluate one’s cards. This number indicates how likely the agent is going to win the game.

$$C_i(v, n) = \begin{cases} \alpha_v \cdot v \cdot n, \text{ if } \ n \le 4 \\ \alpha_v \cdot (v + (v + n - 1)) \cdot n / 2, \text{ otherwise} \end{cases}$$

The third agent is so called Advanced Agent. The Advanced agent treats each time of discarding as **decision making**. The same as simple agent, we will build in experiences from human beings. But this time the volume of experiences is large, which we call the **knowledge base**. And we construct this knowledge base as a **decision tree**. We prepare a lot of conditions which the agent may encounter. These conditions are the nodes of the tree. And we design different actions when different conditions are satisfied. The actions are the arrows of the tree. Another important thing is that when alternative conditions appear, we may use **probability theory** to deal with it. Firstly we may collect the same situations. Then we do **statistics** about the result if each choice is taken. After that, we obtain the probability of each choice. When in the real game the agent meets such a situation, it will make the decision according to the probability we have trained.

The fourth agent is predictive agent. It will firstly to remember what cards are discarded. And then it will try to predict what exact cards in each other player’s hand by probability. After that, suppose all the card types are known, the agent will search the optimal strategy to run out its own cards. If all the card types are inferred, the agent will take 3 agents’ adversarial minimax algorithm to get the terminal node.

2. The hypothesis

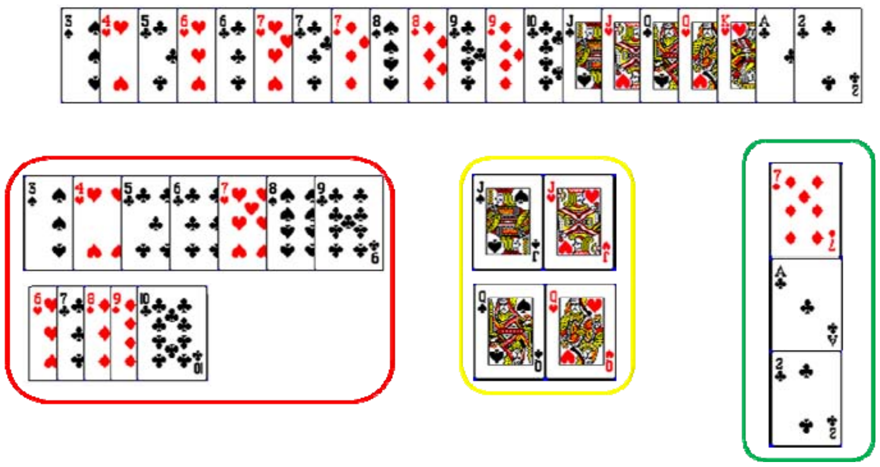
From the agent design, we can see that we imported experiences from human beings and the prediction of other players' card types step by step. So we want to propose 2 hypotheses: **(1)** the more real situation experiences from human beings imported to the agent, the performance will be better; **(2)** The more and exact information the agent can guess from the rest cards in other players' hands, it will be more rational and more possible to win the game.

And we can see the random agent never import anything. All it does is random. So the performance is expected to be poor. The simple agent takes in several essential experiences and it helps the agent to improve the performance a lot. While a large volume of experiences are taken by the advanced agent, we can expect its performance will achieve the level of human beings. For the predictive one, it not only get the aid of experience but also try to obtain all the information from the play table and guess from other players' card types by probability. In this way, we hope it can beat people by chance.

Part B. detail work I have accomplished in our group

To begin this project, given a pile of cards, the first and essential thing we should do is to cut the cards into different card types. We name it "Card state division". All other functions are based on this step. Then we want to find the best division with least card types so that we expect to take least steps to win. Then after discussing the design of the four different agents, I implemented simple agent and advanced agent using Visual C++ platform. And we work together to debug and make each part of the project consistent.

1. Card state division



As can be seen from the picture above, obtaining a pile of cards, the most first step is to divide all those cards into different types, which we introduced as single, pair, triplet, quad and flush. Each pile of cards can be divided into different card state divisions. The division is not unique.

In this section, we basically take **Depth First Search** to firstly search out the longest flush. And then we may go on to search for quads, triplets, pairs and single cards.

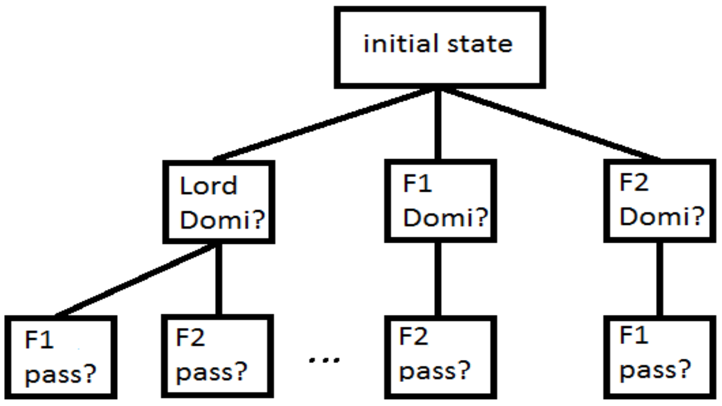
2. State transition

Of course we have to find out all the states so that we can get the best state division among those divisions. As states can be arbitrarily translated from one state to another, we start from the current state. By cutting off the flushes we have found based on the basic state division step, we can obtain all the possible flushes derived from the original one. Then we collect the cut off cards with the left cards to do the state division again. But this time the state division will not find any flush. It will start directly to find quad, triplets, pairs and single cards. During the cutting off step, we search the possible sub-flushes by **greedy best first search** in order to find all the candidates. Then based on card power, we choose the largest one to be the optimal one.

3. Simple Agent and Advanced Agent Implementation

The simple agent is based on basic experiences, which we believe to deal with most common cases. It based on the following experiences: (1) the domination is important; (2) Farmers' card power should be large enough; (3) try to choose the lower rank card type to go first; (4) leave as few states as possible.

We can see there is seldom cooperation between two agents in the simple agent. Then we expand the experiences as a knowledge base, which can cover almost all the cases that people can meet in the real play. We translate this knowledge base into decision tree form to guide our agents. In this knowledge base, we add some special cases which must need cooperation between two agents. For each turn, the decision tree structure is like the picture below. Each node represents one situation. Each branch represents the choice the agent or landlord makes. We also import probability theory to deal with the alternative situations. This time, the performance enhanced significantly as can be seen from our evaluation results.



Part C. Conclusion

In this project, we designed a game Fight the Landlord which needs sophisticated skills and prediction of cards types. We designed and implemented four different levels of intelligent agents according to our hypotheses. From the evaluation results we can see that our agents meet with our hypotheses and especially the advanced and predictive agents can challenge the opponent of human beings. During the process of design and implementation, we took the professional AI thinking of what we learnt from Professor Kulikowski. And it proves to be essential and effective. We want to thank Professor Kulikowski for his diligent teaching and good inspirations. We also appreciate each other for good team spirit and cooperation.

The Work Description

- **Predictive Agent**

In this project, we implemented the intelligent version of one popular game in China – Fight the Landlord, which is a typical example of the partially observable, adversarial poker game. In this game, the human expert generally has the ability to flexibly employ the different maneuver strategies at different circumstances, e.g. he/she can master the current state of the game through the clear and thorough analysis and then try to make the perfect strike at the beautiful timing. Here in order to imitate or even challenge the intelligent performance of the human expert, I designed one smart agent, called the Predictive Agent.

Generally, the design philosophy of the Predictive Agent is just to let the agent make its decisions not only referring to the general guidance from the expert experiences, but also based on the acute analysis of the specific situations in the current game. Naturally the combination could render the agent to be flexible and stable at the same time.

Accordingly, two different methods were utilized in the fulfillment of the Predictive Agent. 1) The Decision Three was constructed, due to its high efficiency and expandability, to reflect the knowledge base, e.g. the general guidance from the expert experiences. 2) Inference + minimax adversary strategy was used to do the analysis job. Inference is the reasoning process, which tries to gradually eliminate the uncertainty; Minimax strategy is the searching process, which manages to calculate the possible optimal actions.

To be specific, in the inference part, what the Predictive Agent predicts/inferences are the possible remaining card states in the other players' hands by referring to the context information, like the actions of the players and all the remaining cards. That's why we call it the Predictive Agent. And the exact strategy the Predictive Agent uses is the greedy exclusive method, which assumes that at most times the other players would run the cards as small as possible, and thus based on their previous actions the Predictive Agent can separately exclude the impossible card states from his/her card states space. The more rounds pass and the less cards left, the more accurate predictions the Predictive Agent can get. Therefore, the partially observable problem is transferred to the one, which as if was fully observable, especially at the second half of the game.

Furthermore, what will happen if you know every possible card state of the other two players? The answer is definitely positive. In the project, I chose the simplified minimax adversary strategy to search out the possible optimal order to run the rest cards, to help the partner run his/her last cards, to avoid running the cards which the opponent need. The cooperation characteristic is added into the adversary strategy to make the two farmer Predictive Agents work as a team, to prevent them fight with each other. Besides, it's clear that the performance of the minimax search strategy is dependent on the quality of the previous predictive job. The expected value is calculated to handle the uncertainty. The more precise the prediction is, the better performance the search strategy can get.

How to combine the specific analysis with the general guidance? Generally speaking, the general guidance dominates at the information shortage situations, especially at the beginning of the game, while the inference + minimax strategy helps mainly at the end of the game, when the remaining states are convergent and limited. So they dominate primarily at the different periods, working well throughout the game.

The extensive experiments demonstrated the smart performance of the Predictive Agent (as showed at the evaluation part of the report). A typical real example is illustrated as follows.

Landlord: (10, 10, 10), (6), (A)

Farmer1: (5, 5), (J)

Turn to run → Farmer2: (3, 3, 3), (5, 6, 7, 8, 9), (Q, Q), (A)

If the Famer2 run (3, 3, 3) first, as directed by the general guidance, then Landlord would get the domination by following (10, 10, 10) and then run out of his cards first. However, the Predictive Agent can foresee this result because of the accurate prediction of the remaining cards in the Landlord's hand, and thus avoid running (3, 3, 3) first, but running the rest cards in the order of (5, 6, 7, 8, 9), (Q, Q), (A) and final (3, 3, 3), which contrarily makes the farmer team win.

- **Something else**

Except the design and implementation of the Predictive Agent, the works I did in the project also include:

1) the implementation of the game board, which controls the flow of the whole game, like shuffle the cards, distribute the cards and arrange the running order of the three players.

2) the design and the implementation of the Simple Agent, which takes the basic strategies, involving just some simple cooperation and competition principles. The Simple Agent greedily keeps the powerful cards in hands, and it doesn't observe the current situations, but just stick to the pre-defined patterns to run its cards. The Simple Agent simulates the performances of the novices for the game.

3) the implementation work of the definition of the interfaces at the early stage and of course a long period's debugging works.

Thanks

Tinglin Liu

Computer Science Dept.

ID: 134008489

Test part: Xiaoye Han

Experiment Design

Test1. Cards divide Strategies of Farmer agent (Advanced agent + Predictive agent)

Test2. Rationality of the Cardpower Evaluation by agents logic

Test3. Comparisons among different agents

Implementation:

Part 1: Test1 and Test 2 (will not be shown in the presentation)

Methods：

Let Farmer agent deal with certain types of cards, see how it divides & compare with cards division samples of experts.

Given certain situations, see how it reacts & compare with human heuristic

Goal :

Evaluate the preformance of Farmer agents

The results are used as feedbacks to adjust agent heuristic & strategies along the project process.

It isn’t shown in the presentation.

Part 2: Test 3

Method:

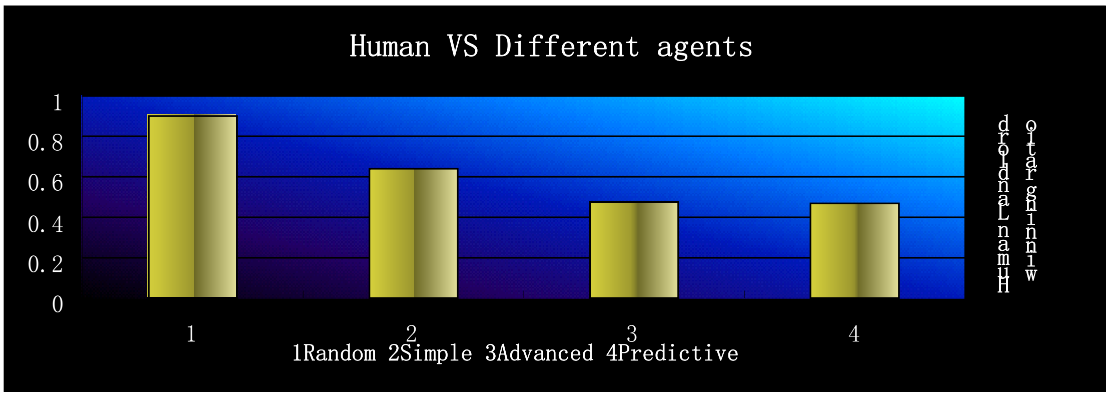
Design 4 different agents based on different heuristic. Name: 1 Random 2 Simple 3 Advanced 4 Predictive. Let human play against different agents, calculate the landlord winning ratio based on data sets.

Goal: Evaluate the preformance of different Farmer agents. Focus on the much smarter Advanced agent and Predictive agent. See if they can control the human player winning ratio below 50%.

Result:

According to the winning ratio, we can see human can easily beat Random agent. However, when faced with our smart agent, the winning ratio decreases to below 0.5.

Farmers	Human (Lanlord) winning ratio
1 Random	0.9
2 Simple	0.64
3 Advanced	0.47540983607
4 Predictive	0.46875



Part 3: Test 3

Average Round number:

Method:

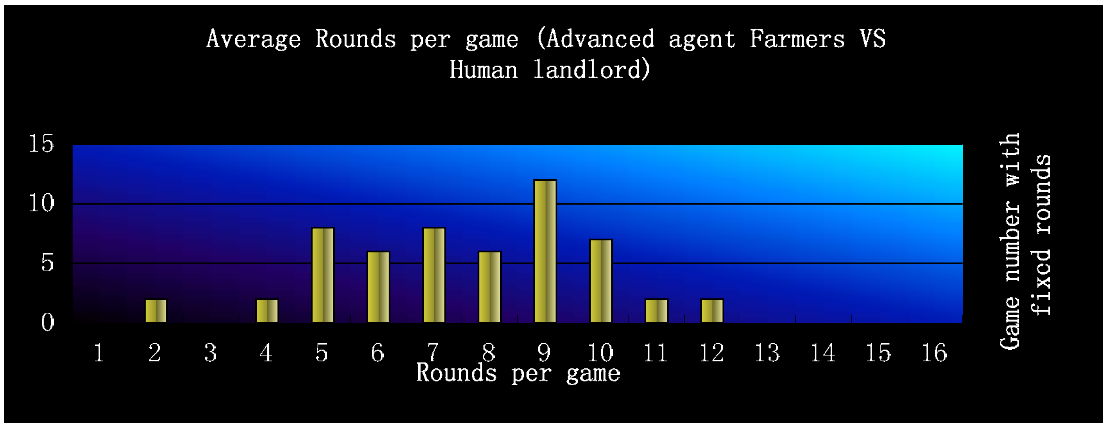
Based on Human Landlord VS Predictive & Advanced data sets, calculate average rounds per game. Compare preformance.

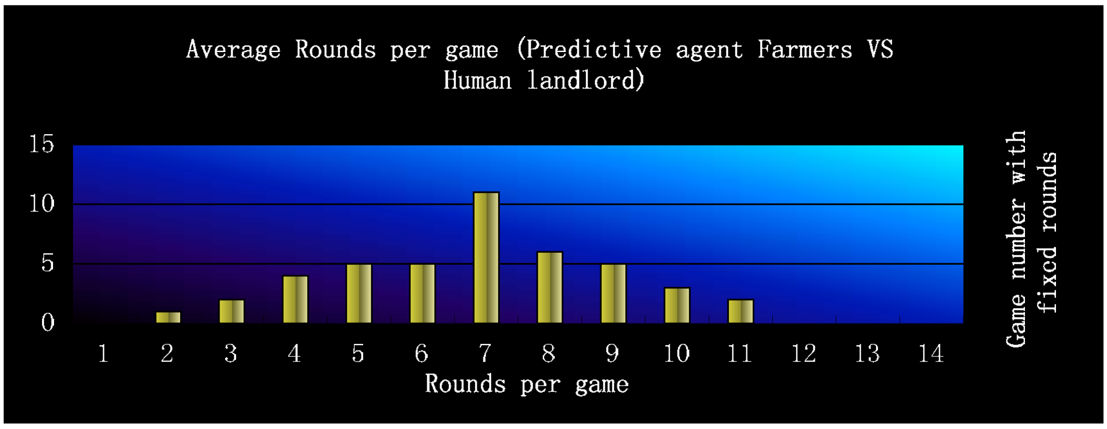
Goal:

Prove the intelligence of our agent. Predictive agent terminate the game in less rounds.

Result:

Human lanlord\Different Farmers	Predictive	Advanced
Average rounds/per game	8.1591	9.4363





Part 4: Test 3

Method:

Name: 1 Random 2 Simple 3 Advanced 4 Predictive. Let these four different agents plus human play as Landlord. Against Farmer played by 3Advanced agetns and Farmer played by 4Predictive agetns. Calulate the Farmer winning ratio based on data sets. Compare the preformance between 3 Advanced and 4 Predictive.

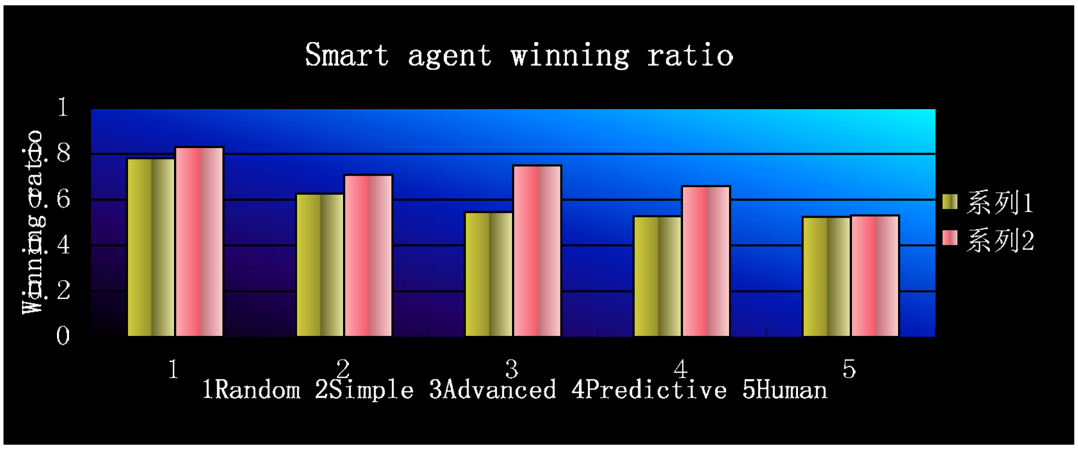
Goal: Test on smart Farmer agents. Focus on 3Advanced agent and 4Predictive agent. Evaluate the preformance of them against different landlord agents.

Result:

According to the diagram, we can see our Advanced agents and Predictive agents as farmers can easily beat the random/ simple agents as landlord. Besides, stay above 50% winning ratio against human. Comparisons between 4 Predictive and 3 Advanced shows that 4 agent plays a little better preformance.

Farmers\Lanlord	1 Random	2 Simple	3 Advanced	4 Predictive	Human
3 Advanced	0.78125	0.62645348837	0.5462962963	0.52777777778	0.52459016393
4 Predictive	0.83018867925	0.708502024429	0.75	0.659793814443	0.53125

The green is 3 Advanced. The pink is 4 Predictive



Conclusion: Fight the Landlord is a partially observable, multi-agent adversarial game.
The agents we designed are proved to be smart enough to cooperate and beat human beings in the game.