Da Vinci Code

Game AI Agent Design

CS181 Artificial Intelligence Project Report

Team Member：秦朝 张涵锐 刘峻上

CONTENTS

Section 1

Problem Background

Background

SPIELE-BESTSELLER!
INTERNATIONAL CODE-BREAKING SENSATION
EL JUEGO MÁS INTERNACIONAL DE CÓDIGOS
Eiji Wakasugi
DaVinci
code
SPIELEN MACHT SCHULE

01

Guess-Number Game

Da Vinci Code game is basically a **number-guessing game**. Players can see their own cards, meanwhile must utilize hidden information to inference the number of opponent's card.

02

Turn-Based Game

In this game, players **act in turns**, making guesses and adjusting their strategies based on feedback from correct or incorrect predictions, so as to **maximize its own profit**.

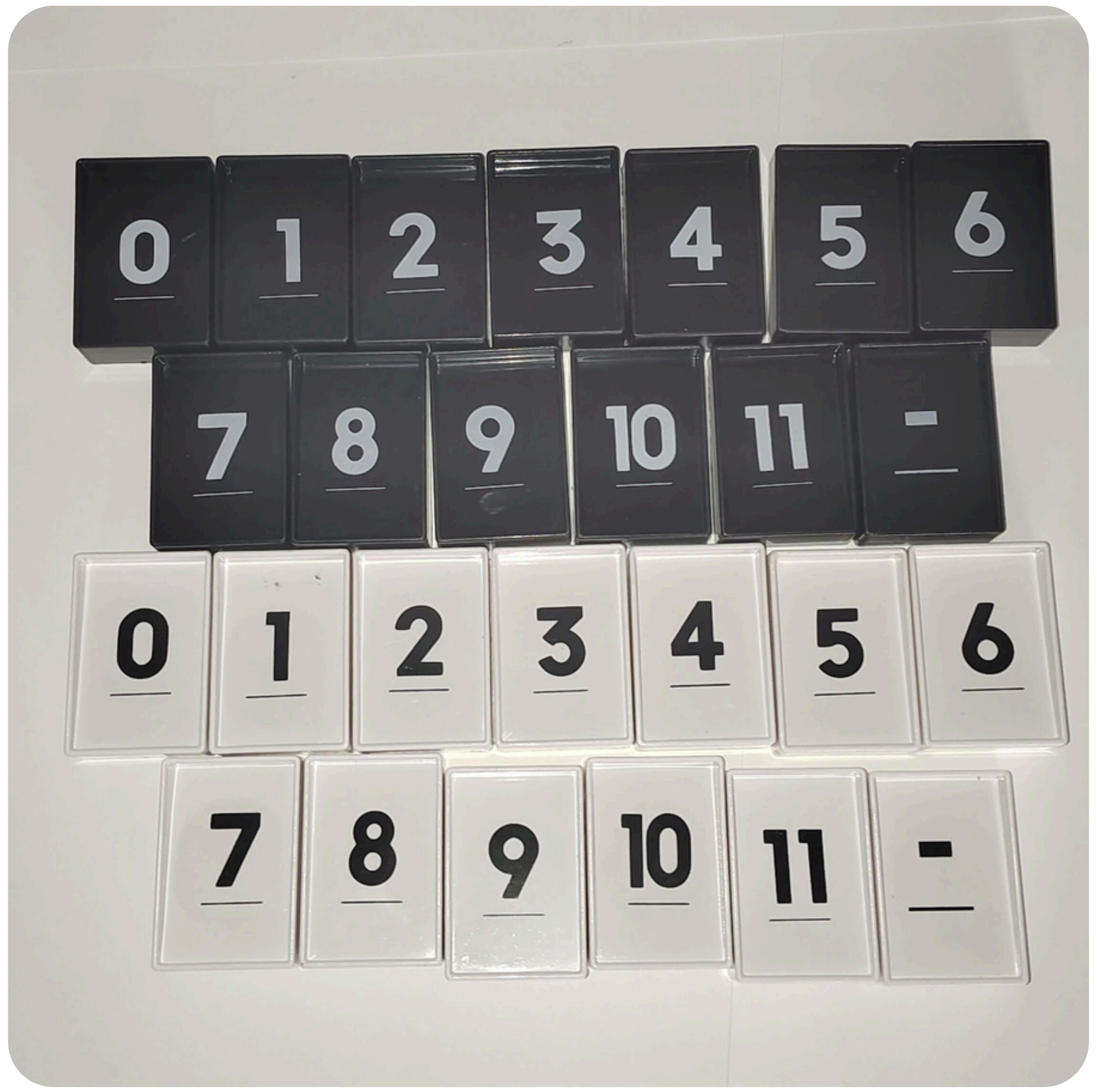
Game Rules

Gameflow

Each player **draws four hidden cards** and arranges them in the order **from small to big**. On their turn, a player **draws a new card** of either white or black, then **guesses** an opponent's hidden card number. If correct, they can **guess again or insert the new card** into their hand; if wrong, they **reveal the card** and insert it.

Winning Condition

When the deck runs out, players no longer draw new cards. The goal is to reveal all opponents' hidden cards while keeping your own hidden. A player wins by revealing all other opponents' cards before being revealed his own cards.



Section 2

Problem Formulation

Game state

Self Variable State

Own Hand Cards	hc[]	Ordered list of own cards (number, color, revealed status).	hc[0] = [7, 'black', False] means black 7 in my hand, not revealed, and leftmost.
Opponent Cards	oc[]	Ordered list of opponent’s revealed cards (number, color, revealed status).	oc[2] = [5, 'white', True] means opponent’s third smallest card is revealed as white 5.
Own Action History	hist_self[]	List of past actions taken by self (card index, guessed number, result).	hist_self[-1] = [1, 5, True] means last guess was opponent’s second card guessed as 5, and was correct.
Opponent Action History	hist_opp[]	List of past actions taken by opponent (card index, guessed number, result).	hist_opp[-1] = [0, 7, False] means opponent guessed my first card as 7 and was wrong.

Game state

Global Variable State

Remaining Deck Cards	deck_remain_set	Set of all remaining cards in the public deck	deck_remain_set = {(0, 'white'), (2, 'black'), (7, 'white')} means these cards are still in the deck
Current Turn Player	turn	Indicates whose turn it is to act: represented by 0 or 1	turn = 0 means it is my turn to act

Initial State

Initial State	hc.size() == oc.size() == 4
	turn.update()
	deck_remain_set.init()
		Tribute 4 cards to 2 players	Each player should be able to choose white or black

Game state

Actions

Pick new card	!deck_remain_set.size() new_card = randInt(...)	randomly pick new card from deck if exists

Guess opponent card	game.check(turn)
	game.check(success)
		before guessing, confirm state of has_guessed

insert card	if guess_success: self.insert()	before inserting, must have guess_successed

Game state

Successor

update hand card	hc.update()
	oc.update()
		update one player's hand card record, for the one who's at his turn

update history guess	hist.update()	update history guess according to current guess action

update deck data	deck_remain_set.update()
	turn = 1 - turn
		deck data reduces its number of cards if exists

Goal

all cards revealed	all_revealed(hc) ‖ all_revealed(oc)	Game ends when one player has all cards revealed

Section 3

Proposed Solution & Comparative Experiments

Naive Solution

Pick new card

- Which color to pick if both exists? Random!

$$\begin{tikzcd} \text{turn} \arrow[r] & \text{action} \end{tikzcd}$$

Guess opponent card

- Find all possible solution for each unrevealed card
- Guess the card with least possible solutions
- Consider equal possibility

$$\begin{tikzcd} \text{hc [ ]} \quad \text{oc [ ]} & \\ \text{hist\_self} \arrow[r] & \text{action} \\ \text{deck\_remain\_set} & \end{tikzcd}$$

Insert card

- Choose to insert card if exists failure probability
- Otherwise guess instead of insert

$$\begin{tikzcd} \text{last action} \arrow[r] & \text{action} \end{tikzcd}$$

Bayes Inference

Improve

- Consider hidden information instead of random guess!
- Make an optimal decision under the same prior distribution!

Definition

- P(Ci=n | obs): Under current observation, probability of the number of the opponent's i-th card is n
- Considering variable: hist, oc, naive

Bayes' Rule

- Multiple false guess by myself: hist_self, produces high sink costs, therefore should continue guessing
- opponent has guessed number around n
- opponent has guessed number n

- From Naive Solution

$$P(C_i = n \mid \text{obs}) = \frac{P(\text{obs} \mid C_i = n) \cdot P(C_i = n)}{\sum_{n'} P(\text{obs} \mid C_i = n') \cdot P(C_i = n')}$$

- Distinguish optimal solution from the "same" probability

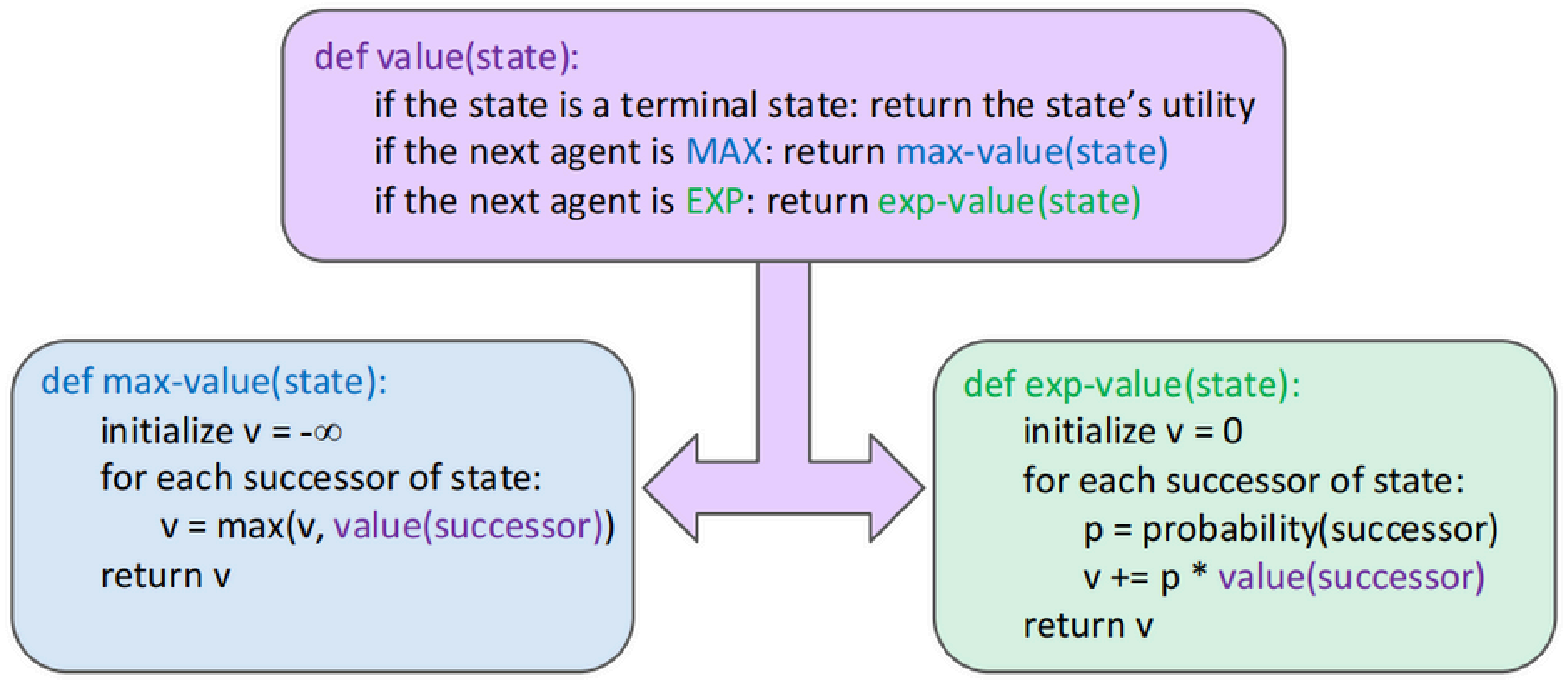
- Normalization by summing all probabilities

Adversarial Search

Expectimax

- First conquer an **inline loop** of choosing to guess or insert
- Inline loop guessing uses **probability from previous bayes inference** (since it's my card and my view), it's used in evaluation function.
- **Probability of successor (p)** uses naive solution (directly or with exponential trend) with guessing finished
- In evaluate function, consider number of my revealed cards (risk), the opponent's revealed cards, hand size, the Bayesian probability of guessing opponent cards, opponent owned information and so on.

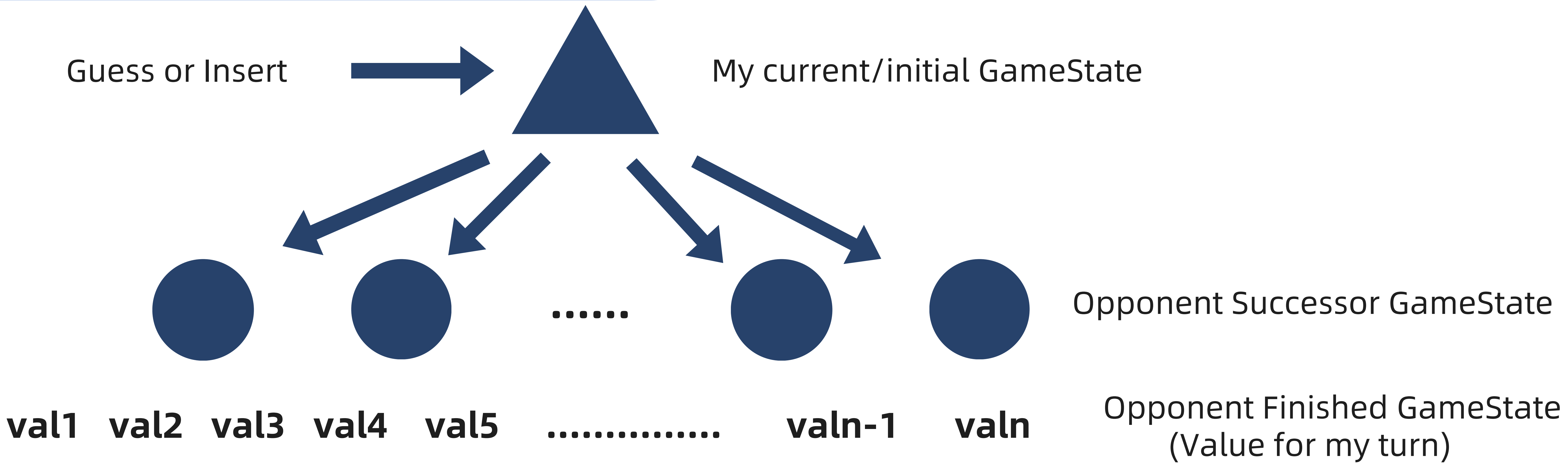
Depth: 3 ~ approximate 8



```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is EXP: return exp-value(state)

def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v

def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```



Guess or Insert
My current/initial GameState
......
Opponent Successor GameState
val1 val2 val3 val4 val5 .... valn-1 valn
Opponent Finished GameState
(Value for my turn)

Comparative Experiments

Naive Solution

Act like a **new player with very good memory**, however lacks strategy and gives its winning chance to “god”.

Bayes Inference

Consider more **potential informations** including opponent’s guessing information, develop more advanced strategy and should work better than naive solution. However it only **sees the past and ignores future**.

Expectimax

Take both naive solution and bayes inference into consideration, and start considering more besides only guessing number with high correctness. It considers how to **win the game from total view**, making it acts more like a skilled player. **By adjusting parameters in evaluation, we can simulate different strategies like gambler or milquetoast.**

Compare With RL Method

There exists an RL game agent for this game in open resources, we are going to estimate the performance through hand testing and bot competing, in order find how they work under different situations.

Section 4

Expected Outcomes & Contributions

Expected Outcomes

It’s expected that the game agent can **appropriately make an optimal solution in a short time** (the size of game is very small) of whether it shoud guess or not, and which number to guess. **Higher winning rate** also belongs to our expectations. **Multiple special occasions would be simulated** in our test to confirm the correctness and effectiveness of the expectimax and bayesian method.

Outcomes against RL

The performance of expectimax and RL are expected to be similar, since they all consider common strategies we can see. Meanwhile, there might exists situations where an agent is **too clever that it might loses its game against “stupid” agent**, which we would consider after getting test results.

Contribution

Simple Game Agent

Since this card game is not that popular as other digital games, there are not many open source game agent like this **hand-made strategy** provided online.

Bayesian & Expectimax

With the help of **probabilistic inference and adversarial search**, game agent can act like a skilled player who consider both “history” and “future” , which is based on strong theory.

Evaluation Function

Evaluation function is a key element in our project. Through **adjusting parameters** and values, we can **enable better strategies**. Detailed method would be implemented in the next stage.

Comparison against RL

This contribution holds if our method works as well as or better than RL agent provided online. We would **analysis our method’s advantages and disadvantages to obtain further conclusions**.

Potential Improvement

Multiagents

Da Vinci Card game should be able to allow a maximum of 4 players! This requires more inference under ai agent.

Bar Card

Bar Cards are the last 2 components of this game, which can be placed anywhere. This improves uncertainty for guessing and entertainment.

Deceit or Naive?

How to purpose different strategies toward different opponents **according to their playing histories and strategy habits?**

THANK YOU