Fighting the Landlord with Monte Carlo Simulation

Zhi (Richie) Li , NetID: zhil7

Outline

- Introduction
- Program Designing
- Validation
- Experiments
- Summary and Future Works

Introduction
Fighting the Landlord (斗地主, Dou DiZhu)

- The game is played with Poker cards with Jokers included
- There are three people in a game, one landlord and two peasants as a team
- Everyone is dealt 17 shuffled cards before the role assignment
- Players bid to be the landlord, who gets the remaining three cards shown to all
- Does not care about suits, which means 7 of spades is not bigger than 7 of hearts
- Objective: Play all the cards on hand to win!

Program Designing – Objects – Deck

- Attributes
  - Points: Used for bidding for the landlord
  - Cards: What cards does a deck have (in int)

```
def __init__(self):
    self.points = 0
    self.cards = []
```

- Methods
  - Add new deck: Adds 54 cards to self.cards
  - Shuffle cards: random.shuffle(self.cards)
  - Get deck points: sum(self.cards)
  - Get deck length: len(self.cards)
  - Remove card from hand: uses list remove self.cards.remove(card)

```
def add_new_deck(self):...

2 usages (1 dynamic)  r10641001
def shuffle_cards(self, shuffles=1):...

2 usages (1 dynamic)  r10641001
def get_deck_points(self):...

1 usage (1 dynamic)  r10641001
def get_deck_length(self):...

2 usages (1 dynamic)  r10641001 *
def remove_card_from_hand(self, move):...
```

Program Designing – Objects – Player

- Attributes
  - Character: Landlord, Peasant 1 , or Peasant 2
  - Hand: Card a player has in the hand
  - Hand points: Used for bidding
  - First player next round: Used to determine if a player plays first in each round
  - Strength: How big a player plays a move among the available moves

```
def __init__(self):
    self.character = -1
    self.hand = Deck()
    self.hand_points = 0
    self.first_player_next_round = False
    self.strength = 0  # 0-9, the higher
```

- Methods:
  - Update hand points
  - Assign character

```
def update_hand_points(self):...

3 usages (3 dynamic)    r10641001
def assign_character(self, char_num):...
```

Program Designing – Moves – Weight

```
        self.rule = rule
        self.move_type_weight_and_function = {
            TYPE_1_SINGLE: {'weight': 1, 'function': self.gen_type_1_single},
            TYPE_2_PAIR: {'weight': 2, 'function': self.gen_type_2_pair},
            TYPE_3_TRIPLE: {'weight': 4, 'function': self.gen_type_3_triple},
            TYPE_4_BOMB: {'weight': 10, 'function': self.gen_type_4_bomb},
            TYPE_5_KING_BOMB: {'weight': 16, 'function': self.gen_type_5_king_bomb},
            TYPE_6_3_1: {'weight': 4, 'function': self.gen_type_6_3_1},
            TYPE_7_3_2: {'weight': 4, 'function': self.gen_type_7_3_2},
            TYPE_8_SERIAL_SINGLE: {'weight': 6, 'function': self.gen_type_8_serial_single},
            TYPE_9_SERIAL_PAIR: {'weight': 6, 'function': self.gen_type_9_serial_pair},
            TYPE_10_SERIAL_TRIPLE: {'weight': 8, 'function': self.gen_type_10_serial_triple},
            TYPE_11_SERIAL_3_1: {'weight': 8, 'function': self.gen_type_11_serial_3_1},
            TYPE_12_SERIAL_3_2: {'weight': 8, 'function': self.gen_type_12_serial_3_2},
            TYPE_13_4_2: {'weight': 8, 'function': self.gen_type_13_4_2},
            TYPE_14_4_22: {'weight': 8, 'function': self.gen_type_14_4_22}
        }
        if self.rule:
            self.move_type_weight_and_function[TYPE_16_2_1] = {'weight': 3, 'function': self.gen_type_16_2_1}
```












https://www.cnblogs.com/moyiii-/p/7260517.html

Program Designing – Moves – Generation

```python
class MoveGeneration:
    """generate legal moves
    def __init__(self, cards, rival_move):
        self.cards = cards
        self.cards_unique = sorted(list(set(self.cards)))
        self.cards_dict = Counter(cards)
        self.rival_move = rival_move
        self.rival_move_length = len(rival_move)
        self.new_move = []
        self.move_type_weight_and_function = {
    def generate_move(self):
        """get rival move and generate corresponding moves
        rival_move_dict = get_move_type(self.rival_move)

        if rival_move_dict['type'] == TYPE_0_PASS:
            self.gen_all_moves()

        else:
            self.move_type_weight_and_function[rival_move_dict['type']]['function']()

        self.gen_type_4_bomb()
        self.gen_type_5_king_bomb()
```

Program Designing – Moves – Detection

```
def get_move_type(move: list) -> dict:
```

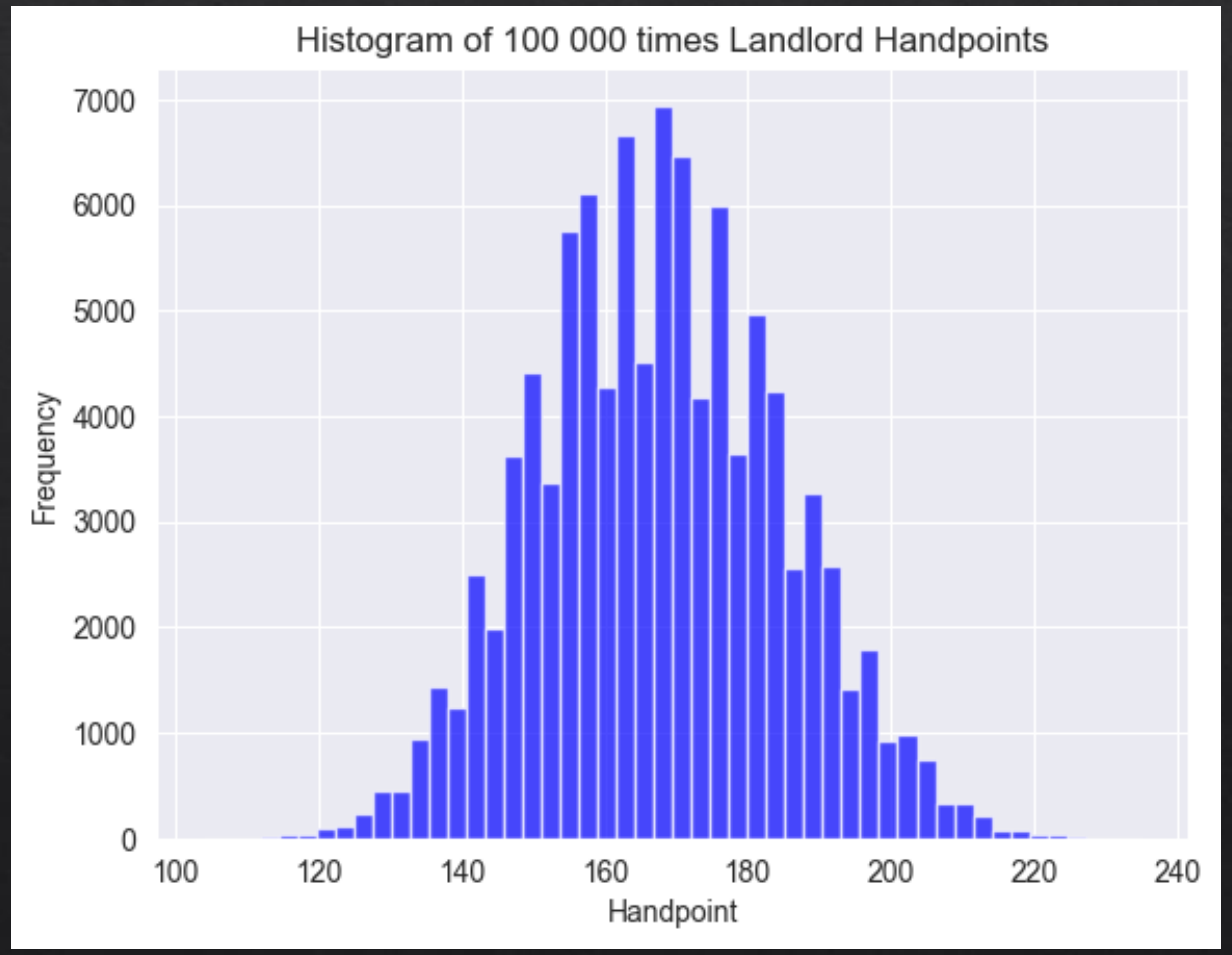
- Gets a list of cards
- Use the length of cards to do a preliminary filter with match, case (switch, case in C)
- Use collections.Counter() to get the amount of cards of each number
- Also a function for detecting if a straight exists
- Planes (3+1*2 or 3+2*2), 4+1+1, 4+2+2 are more complicated and are detected later

Program Designing – Process of the Code

- Setup new game with 3 player objects
  - Deal cards
  - Bid for landlord
- While loop for playing a round until a player has no cards
  - Each player plays a move
  - Check if a winner appears, end game if there is a winner
  - If other two players pass, end round and start a new one (can play different kind of card sets)
- Record the winner of the game
- Repeat the above many times

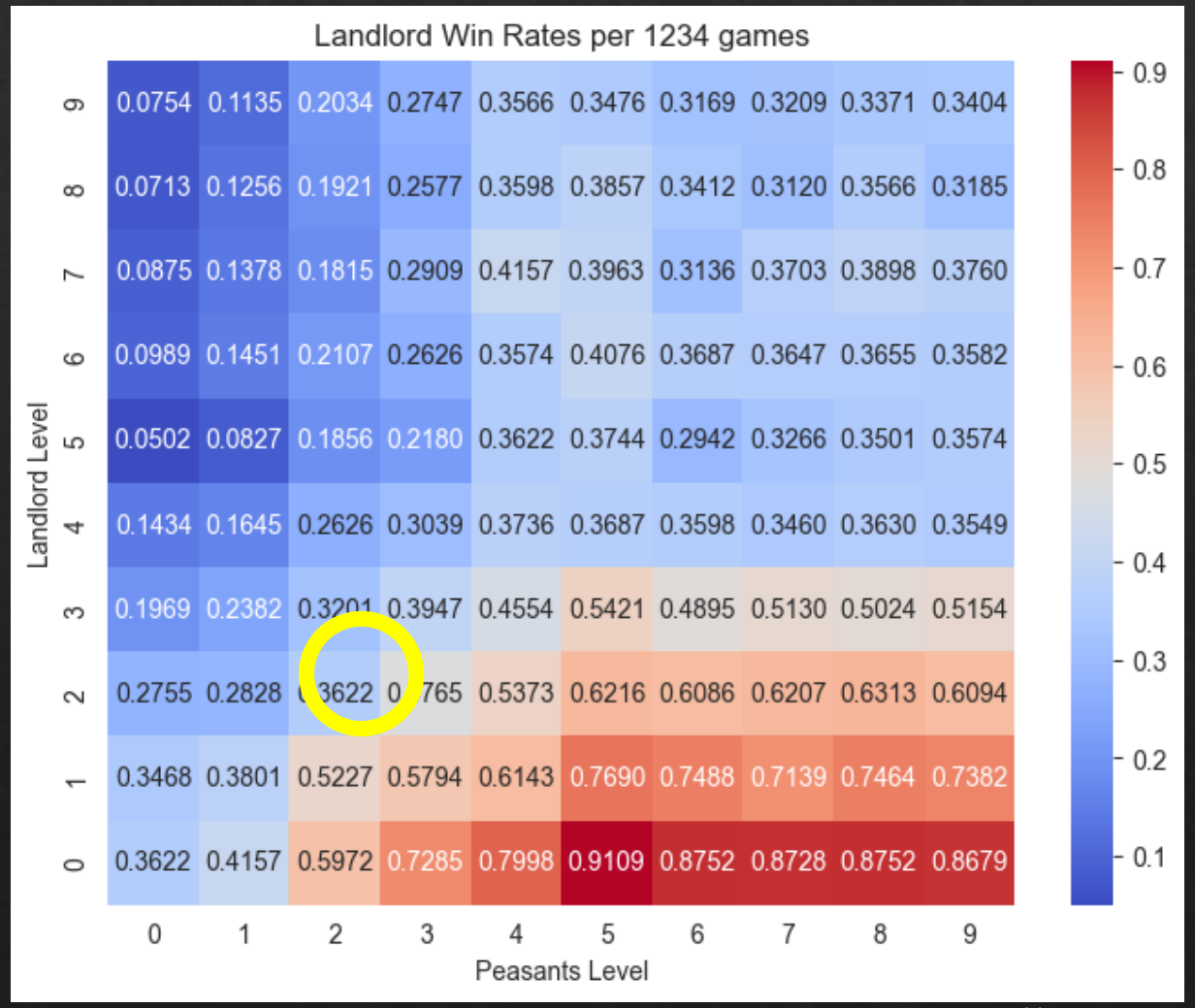
Validation – Points Landlord Get

- The point system:
  - 3~A are 3~14, 2 is 16
  - Small king (black joker) is **20**
  - Big king (colored joker) is **30**
- Landlord has 17 cards before getting the 3 cards, ranging from 79 to 258 points
- Result of testing 100 000 times:
  - Figure on the right
  - Average: 167.64 ± 16.98



Validation – Win Rates with Different Levels

- Without any calculation of future steps or memorization of cards played, it is hard to say whether playing a small or a big card gives a player an advantage.
- However, if the peasants play out all big cards in the beginning, they might not have cards to beat the landlord in later moves
- The more a player plays aggressively, the more possible they will lose



Experiment 1 – Add 2+1 Type

- Landlord level = 2, Peasants level = 4 for a more appropriate playing
  - Not too aggressive or conservative
- Now player can play cards like [7, 7, 8] and [3, 9, 9]
- *Peasants might win more for there are more possibilities?*
- Weight = 3, between Pairs and Triples
- After playing both original rules and special rules, the win rates are:

```
Among 1234 original games played, the win rates are:
    LANDLORD: 53.00% with level 2
    PEASANTS: 47.00% with level 4
Runtime: 2.015625 seconds

Among 1234 special games played, the win rates are:
    LANDLORD: 53.89% with level 2
    PEASANTS: 46.11% with level 4
Runtime: 1.75 seconds
```

x 10

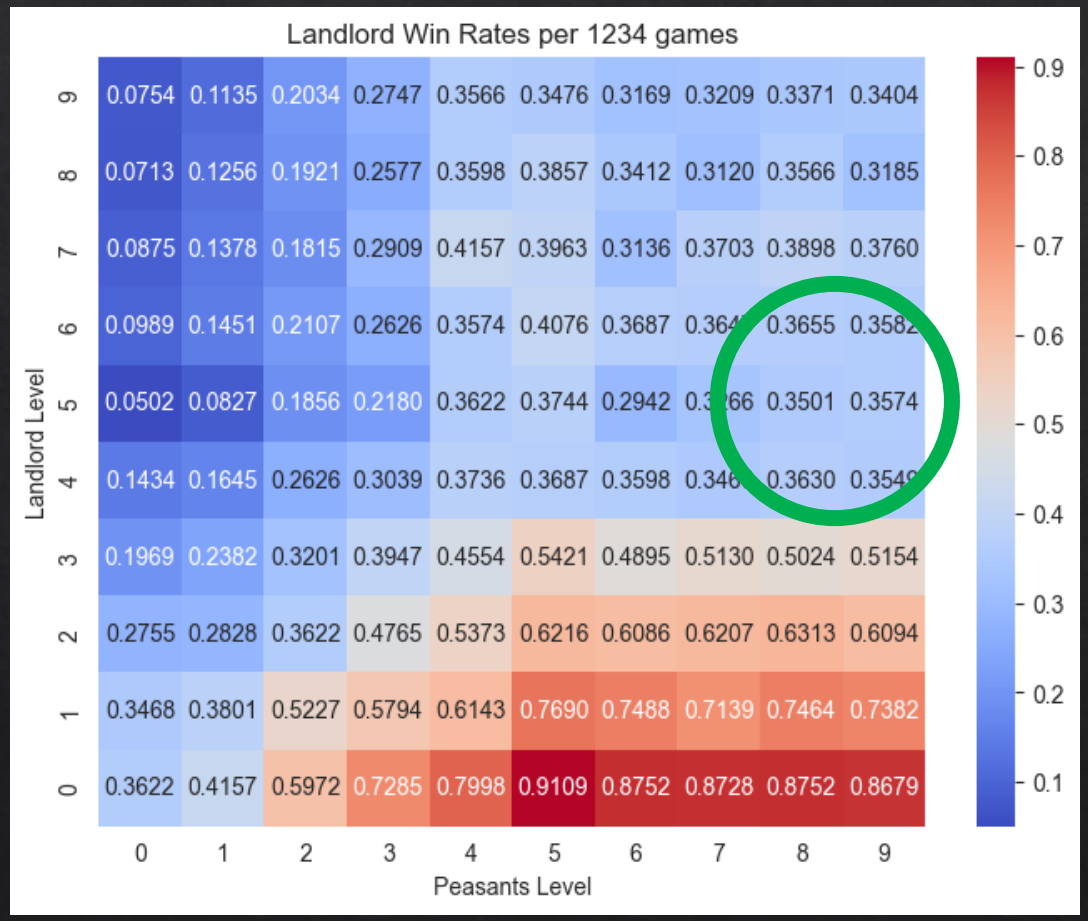
```
Among 12345 original games played, the win rates are:
    LANDLORD: 53.89% with level 2
    PEASANTS: 46.11% with level 4
Runtime: 19.328125 seconds

Among 12345 special games played, the win rates are:
    LANDLORD: 53.41% with level 2
    PEASANTS: 46.59% with level 4
Runtime: 17.390625 seconds
```

THERE IS NO BIG DIFFERENCE!!

Maybe see the heatmap of the special rule?

Experiment 1 – 2+1 Type Win Rates



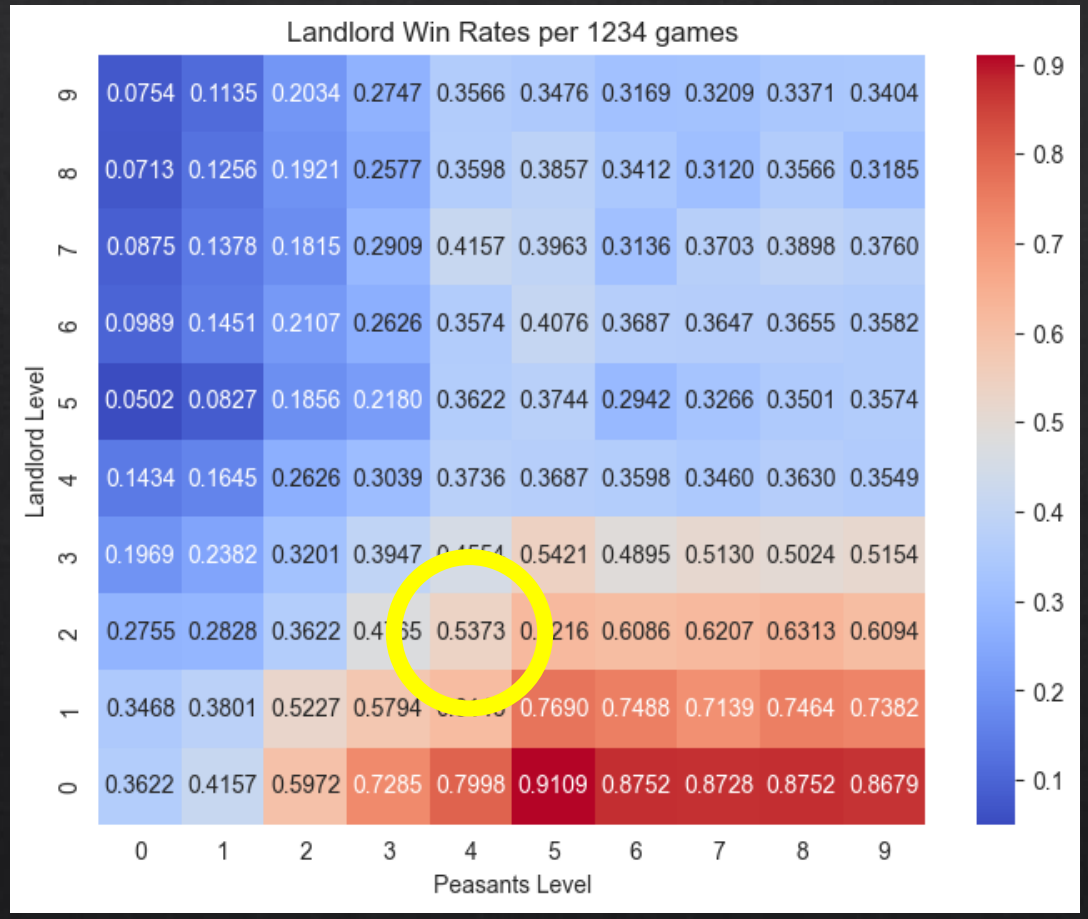
STILL SIMILAR!!

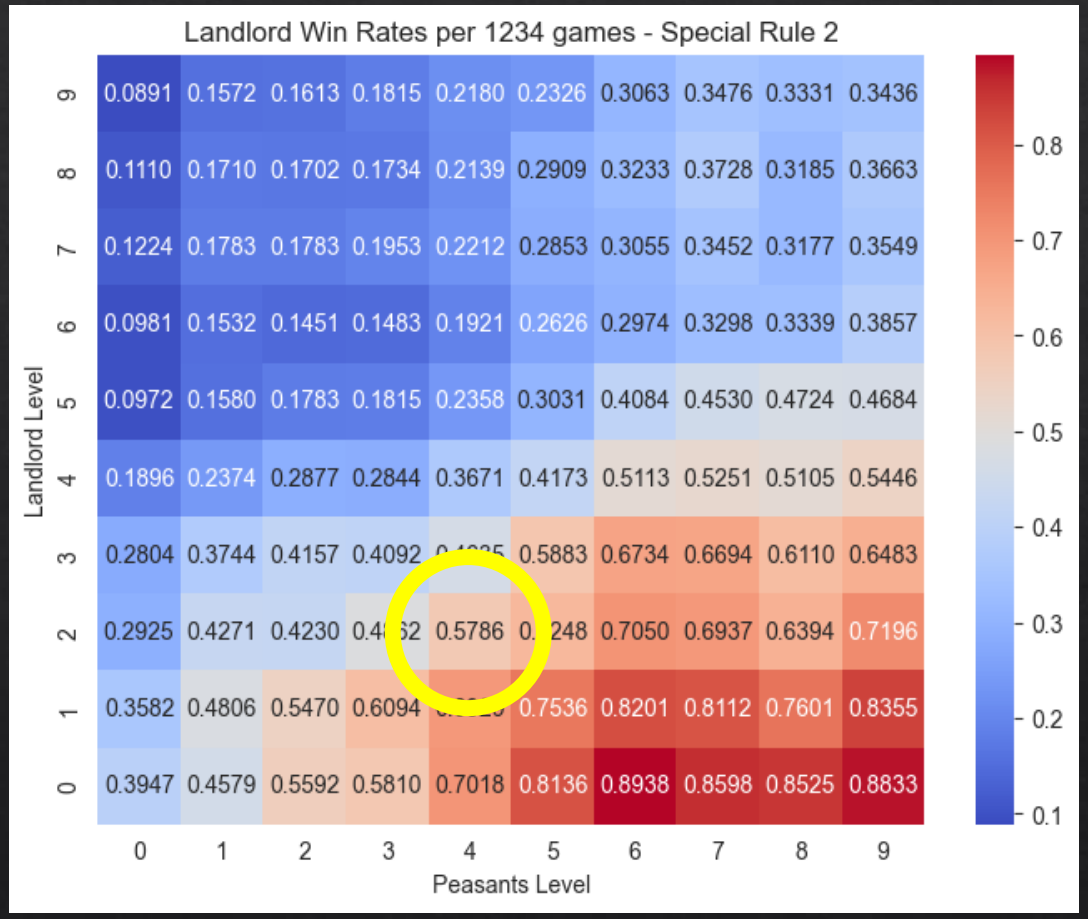
Maybe 2+1 differs too little from pairs and 3+1?

Experiment 1 – Add 2+1 Type

- Independent samples t-test for proportion:
  - Null Hypothesis (H0): The win rates are equal for both rules.
  - Alternative Hypothesis (H1): The win rates are not equal for both rules.
- Get p-value < 0.05, reject H0.
- There is a significant difference between two rules

Experiment 2 – Add 2+2+1 Type





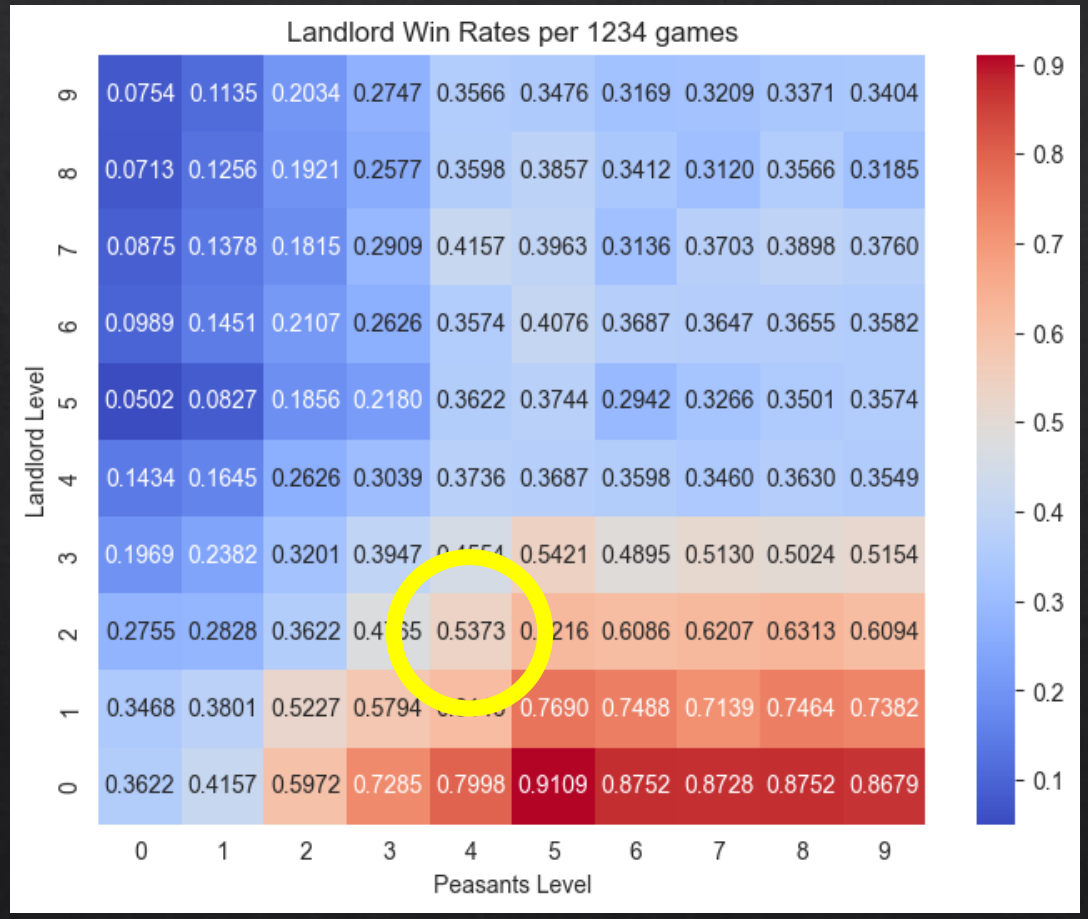
New rule seems to have higher win rate at the difficulty circles by yellow

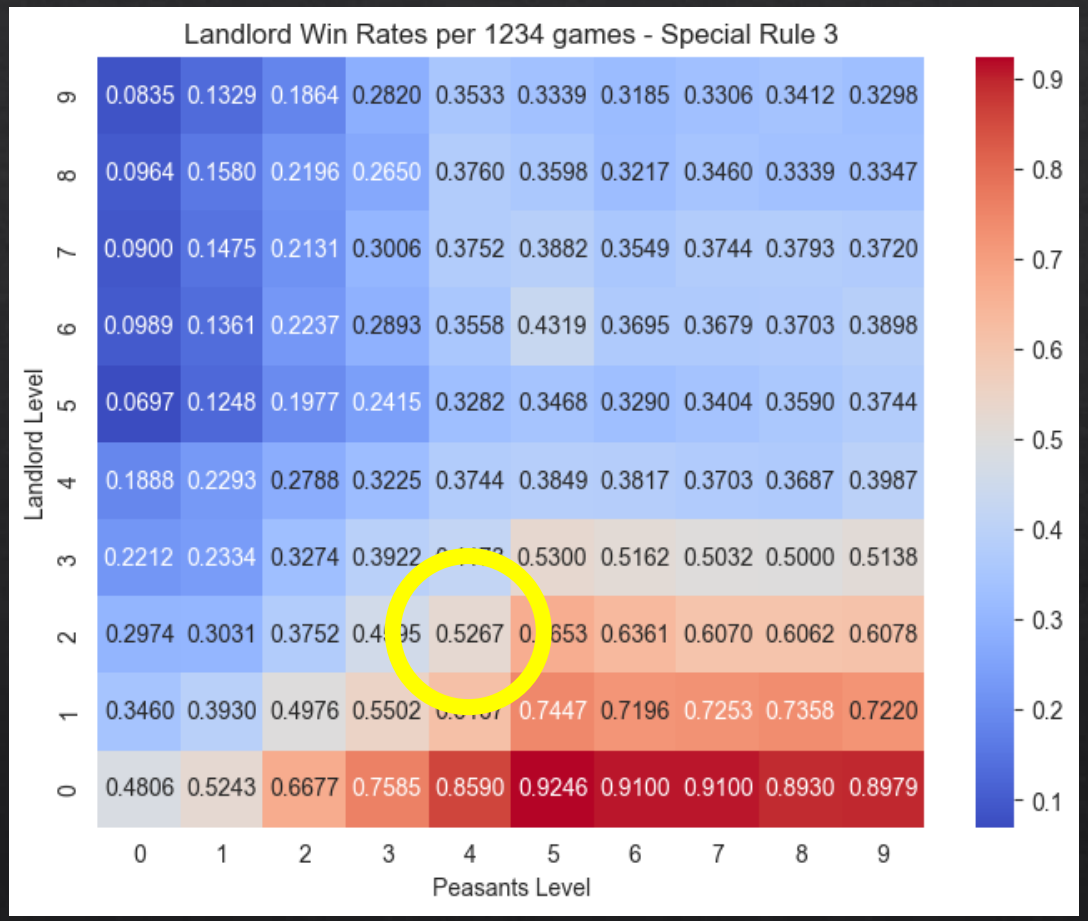
The overall pattern looks similar, do further analysis

Experiment 2 – Add 2+2+1 Type

- Independent samples t-test for proportion:
  - Null Hypothesis (H0): The win rates are equal for both rules.
  - Alternative Hypothesis (H1): The win rates are not equal for both rules.
- Get p-value < 0.05, reject H0.
- There is a significant difference between two rules

Experiment 3 – Landlord can Play an Additional Move before Game Starts





Seems the win rate at the difficulty circled by yellow is not higher than the original one

The overall pattern looks similar, do t-test on the circled player level

Experiment 3 – Landlord can Play an Additional Move before Game Starts

- Independent samples t-test for proportion:
  - Null Hypothesis (H0): The win rates are equal for both rules.
  - Alternative Hypothesis (H1): The win rates are not equal for both rules.
- Get p-value < 0.05, reject H0.
- There is a significant difference between two rules

Summary and Future Works

- The game design was more complicated than I thought, let alone doing deep learning on it
- We observed that the randomness of hand points has convergence
- The experiment of adding new rules made little impact on the win rates, consider making up another rule really different from the original to observe a big difference between them
- All new special rules showed significant difference from the original rule, but the statistics might not be that accurate since we only have 1234 games per simulation
- Too many factors that affect the results, making it hard to monitor the whole process
- Use the profiler to find out what functions occupy the most times and do enhancement
- Maybe consider adding functions that make players record previous moves by all players

References

- Zha, Daochen et al. “DouZero: Mastering DouDizhu with Self-Play Deep Reinforcement Learning.” ICML (2021)
  - Moves generation, detection
- Demo files (poker.py, my_lib.py) from "Programming for Business Computing" 2023 Spring at National Taiwan University
  - Classes of deck and player
- And of course, our best friend ChatGPT for generating some sample functions for testing

Thank you for your attention