# CISC 204 Modelling Project Report

Group 8: Aleeza Balbas, Ripley Visentin, Kylie Wong

# Project Summary

The objective of Gin Rummy is to earn points by collecting a hand where the majority or all the cards can be combined into different sets and runs, trying to group up the remaining unmatched cards on hand before the opponent chooses to end the round, checking to see who has the lowest point value. There are three ways to end the game: knock, gin, and big gin, where knock does not guarantee a 100% win-rate, the win depending on the opponent’s remaining card point value. If one knocks and their point value is higher or equal to the opponent, it is the opponent’s victory, and vice versa. Whoever has a gin, a big gin or the lowest point value between you and the opponent wins the round.

In this project, we intend to model the ideal move choice against the opponent’s sets of predetermined cards. We determine which melds are possible based on the opponent’s decision to either pick up a new card or one that we dropped, as well as analyze what the opponent has based on the cards they choose and drop. As we determine the optimal strategy, we strategize what the player’s next move should be, defining when it is the best time to pick up a new card or take the top card from the discarded pile based on which melds are possible.

# Propositions

|  |  |
| --- | --- |
| player\_a\_b | The player has a card with “a” rank and “b” suit |
| player\_run\_b\_v\_u | The player has a run of 3 or more consecutive cards of the same suit, b = suit, v= lower bound of the run, u = upper bound of the run, v<u  (e.g. player\_run\_A\_1\_3 = [player\_1\_A, player\_2\_A, player\_3\_A]) |
| player\_set\_k\_n | Considered true if the player has a set of 3 or 4 cards of rank k, n = the suit that the set does NOT include, with Z indicates that the set include all four suits  (e.g. player\_set\_1\_A= [player\_1\_B, player\_1\_C, player\_1\_D]) |
| pl\_want\_a\_b | Considered true if obtaining a card of “a” rank and “b” suit would create a meld |
| opp\_a\_b | The opponent has a card with “a” rank and “b” suit |
| opp\_pick\_a\_b | The opponent picks a card with “a” rank and “b” suit from the discarding pile |
| opp\_discard\_a\_b | The opponent discards a card with “a” rank and “b” suit |
| deck\_a\_b | A card with “a” rank and “b” suit is still in the deck |
| dump\_a\_b | A card with “a” rank and “b” suit is being discarded and is not picked up by any player. In our model, this card will never appear in the game again. |

# Constraints

1. A card with “a” rank and “b” suit can only appear in one place, either in the opponent, the player, the deck, or the dump.

* (player\_a\_b ∧ ¬opp\_a\_b ∧ ¬deck\_a\_b ∧ ¬dump\_a\_b) ∨ (¬player\_a\_b ∧ opp\_a\_b ∧ ¬deck\_a\_b ∧ ¬dump\_a\_b) ∨ (¬player\_a\_b ∧ ¬opp\_a\_b ∧ deck\_a\_b ∧ ¬dump\_a\_b) ∨ (¬player\_a\_b ∧ ¬opp\_a\_b ∧ ¬deck\_a\_b ∧ dump\_a\_b)

1. Definition of a run:the player has 3 or more consecutive cards of the same suit. b = suit of the cards, u = lower bound of the run, v = upper bound of the run with u<v.

* player\_u\_b player\_u+1\_b  **…** player\_v\_b player\_run\_b\_v\_u

1. Definition of a set: the player has 3 or 4 cards of the same rank. k = rank, n = the excluded suit with ‘Z’ = all suits are included.

* (player\_k\_j player\_k\_l player\_k\_m) (player\_k\_A player\_k\_B player\_k\_C ∧ player\_k\_D) player\_set\_k\_n

1. If the opponent picks a card(a, b), that card must create a meld or contribute to an existing meld.

* opp\_pick\_a\_b (opp\_a-1\_b ∧ opp\_a+1\_b) ∨ (opp\_a+1\_b ∧ opp\_a+2\_b) ∨ (opp\_a-1\_b ∧ opp\_a-2\_b) ∨ (opp\_a\_j ∧ opp\_a\_k) ∨ (opp\_a\_j ∧ opp\_a\_k ∧ opp\_a\_m) ∨…….

1. If opponent does not pick card (a, b), then for sure they do not have card(a, b).

* opp\_pick\_a\_ b opp\_a\_b

1. If opponent discard card (a, b), then for sure they do not have card(a, b)

* opp\_discard\_a\_ b opp\_a\_b

1. If the opponent does not pick the face-up card, the face-up card will be in the dump and cannot be picked up anymore because the most recently discarded card will be on top.

* ¬opp\_pick\_a\_b → dump\_a\_b

1. If the opponent either does not pick a card(a, b) or discards, then they do not have related cards that make card (a, b) into a meld.

* opp\_pick\_a\_ b opp\_discard\_a\_ b (opp\_a-1\_b ∧ opp\_a+1\_b) ∨ (opp\_a+1\_b ∧ opp\_a+2\_b) ∨ (opp\_a-1\_b ∧ opp\_a-2\_b) ∨ (opp\_a\_j ∧ opp\_a\_k) ∨ (opp\_a\_j ∧ opp\_a\_k ∧ opp\_a\_m) ∨…….

1. The player wants a card(a, b) if it would create a meld or contribute to an existing one

* (player\_a-1\_b ∧ player\_a+1\_b) ∨ (player\_a-1\_b ∧ player\_a-2\_b) ∨ (player\_a+1\_b ∧ player\_a+2\_b) ∨ (player\_a\_j ∧ player\_a\_k) ∨ (player\_a\_j ∧ player\_a\_k ∧ player\_a\_m) → pl\_want\_a\_b

1. If a card(a,b) is not in the deck, then it must either be in the player’s hand, the opponent’s hand, or the dump. In all these cases, the player does not want the card.

* ¬deck\_a\_b ∨ dump\_a\_b ∨ opp\_a\_b pl\_want\_a\_b

# Model Exploration

In the beginning, we intended to use our model to be focused on the winning methods of the game. We try to suggest the player the best time to knock that win the most points while reducing the chance of being undercut by the opponent. But we realized that without analysing what the opponent potentially having and the chance of picking up a card that the player wants.

So, we simplify the game and focus on dealing with a total of 36 cards with ranks = {1, 2, 3, 4, 5, 6, 7, 8, 9} and suits = {A, B, C, D}. We have made up some rules of how the player and opponent behave: they will pick up a card from the discarding pile if and only if the card can be added into an existing meld OR the card can make into a meld. The game ends when any player has a deadwood of 0, i.e. all cards are in either a set or a run (this is also known as Gin).

## Exploration 1: Analyzing the potential cards Opponent has

First, we explore the possibility of cards that the opponent is holding based on how the opponent behaves in the game. The only two inputs we have from the opponent is whether they pick the card facing up from the discard pile or not and the card that they discarded. For instance, if the opponent picks up the initial card facing up, then the initial card makes into a meld or adds into existing meld that the opponent is holding. The same logic applies to the card that the opponent discards. If a card is not in a meld, or it does not have potential being a meld, the opponent will discard that card. Given we know the cards the player is holding, we can conclude the cards that the opponent potentially has by considering all possible meld combinations that the opponent has.

### Bug – Accuracy of guessing the potentially

We tried to model a 1-turn base model and found that if the opponent does not pick up the card, the accuracy of the analysis of the opponent’s potential cards decreases from 80% to 40%. Since if the opponent does not pick up the card, there will be no input of what opponent potentially have. However, not picking up also gives information about the opponent that they don’t want the card because it does not make into a meld. After reviewing the constraints we have, “If the opponent picks up the card, that implies that they have related cards that makes into a meld”, we found out we are missing the negation of it, that is “if the opponent does not pick up the card, that implies that opponent does not have related cards”. Noted that these two constraints are similar, but both is important to the model.

Furthermore, to improve the accuracy, we added multiple turns to the model which increase the number of solutions exponentially. This makes intuitive sense since the more cards we know the opponent is picking up and discarding, it increases the possibility of the combination of cards that the opponent is holding. We compared the models of playing 3 rounds and 7 rounds. The accuracy is calculated by dividing the total correct guess by the total guesses, and we took 30 experiments and took the average of each model. The accuracy of 3 rounds is approximately 50%, and for 7 rounds is 75%. This is what we are expecting, since the more rounds we play, there is more information about the opponent cards, and the output of the model would be more accurate.

### Bug – False interpretation of a fundamental constraint: “Card(a, b) can only appear in one place, either in the opponent, the player, the deck, or the discarded pile”

At the beginning, the constraint we have that describe the uniqueness of the card is that if player has the card, the opponent does not have it. That is,

for card in player\_cards:

E.add\_constraint(Player(card[0], card[1]))

E.add\_constraint(Player(card[0], card[1]) >> ~Opponent(card[0], card[1]))

But this constraint is not accurate since it does not describe all the cases of the problem. The proper constraint would be taking all possible places of the card can be, that is a card only appears in one place, either in the player’s hand, the opponent’s hand, the deck that is still hasn’t been draw, or it has already been discarded.

Then we translate the constraint as ¬ (player\_a\_b ∧ opp\_a\_b ∧ deck\_a\_b ∧ dump\_a\_b), but this does not capture that property of the card that can only be in one place. For example, if player\_a\_b = T, opp\_a\_b = T, deck\_a\_b = F, dump\_a\_b = F, the constraint will still be true.

A way to express “a card can either be” is using an exclusive disjunction. In other words, the statement is true if and only if one is true and the others are false.

The modified constraint would be: (player\_a\_b ∧ ¬opp\_a\_b ∧ ¬deck\_a\_b ∧ ¬dump\_a\_b) ∨ (¬player\_a\_b ∧ opp\_a\_b ∧ ¬deck\_a\_b ∧ ¬dump\_a\_b) ∨ (¬player\_a\_b ∧ ¬opp\_a\_b ∧ deck\_a\_b ∧ ¬dump\_a\_b) ∨ (¬player\_a\_b ∧ ¬opp\_a\_b ∧ ¬deck\_a\_b ∧ dump\_a\_b)

However, this does not affect the constraint in the code since the function add\_exactly\_one from the package Bauhaus captures the uniqueness of the card:

for card in deck:

constraint.add\_exactly\_one(E, Player(card[0], card[1]), Opponent(card[0], card[1]), Deck(card[0], card[1]), Dump(card[0], card[1]))

### Bug – Model is not satisfiable

In the model, we assume that the opponent has related cards if they pick up a card and does not have related cards if they do not pick up a card. If the opponent’s behavior is not consistent, then the model would falsely calculate a card that the opponent has or does not have. For example, if the player picks up card(x,y) but then decides to discard the same card, card(x,y) after a few rounds, the model will have a conflict where opponent has card(x,y) and opponent does not have card(x,y). This makes the model not satisfiable. In a real situation, we would interpret it as the opponent potentially has or does not have these related cards. However, we wanted to keep our logic in absolutes, so instead saying that the opponent potentially has these cards, the model is interpreted as the opponent has these cards. Therefore, the model is sometimes not satisfiable because it is both true that the opponent has and does not have card(x,y), which, from our constraint, is stated that every card is unique in the model that it can only appears in one place.

## Exploration 2: Suggest the best move for Player in a given game status and cards that Player need in order to win

The intermediate version of the model is determining whether the player should pick the card from the discarding pile or draw a card. This is done by counting and comparing the solutions of the player picking up the card or not and counting the changes of the number of melds that the player cards forms. The model would either suggest the player to pick up the card or not, which would be the best move to make the round.

We also explore the cards that the player should look out for that would add into a meld. If a card can add into an existing meld that the player has or create a meld, then the player would want that card. For a card that the player should want, the card must be still in the deck. If the card is in the opponent’s hand or already in the dump, there is no chance that the player would draw that card in upcoming rounds during the game. Based on the cards that opponent is potentially holding and the discarded cards, we can derive the cards that the player want to pick up according to the player cards and the game status.

Furthermore, based on the opponent cards that we logically derived from opponent’s behavior and the discarded cards as the game progress, the player would have a sense what the opponent is holding, and the deadwood points that the opponent has. From this, we can derive when would be the best time for player to knock so that the player scores the most points and prevent from being undercut.

# Jape Proof Ideas

|  |  |
| --- | --- |
| DESCRIPTION | JAPE PROOF |
| Player (P) does not have card(a, b) implies that either opponent (T) has the card, or the card is in the card deck (D). Given that opponent does not have the card, therefore, we can conclude that the card is in the deck. | A screenshot of a computer  Description automatically generated |
| If the opponent has a card (TAB) then the player does not want this card (¬PWAB). Premises 2 and 3 are constraints. |  |
| If the player has a card (PAB) then the player does not want this card (¬PWAB). Premises 2 and 3 are constraints. |  |
| If a card is in the dump (DuAB) then the player does not want this card (¬PWAB).  Premises 2 and 3 are constraints. |  |

# Jape Proof Ideas

|  |  |
| --- | --- |
| DESCRIPTION | JAPE PROOF |
| Within the same turn, if the opponent (**T**) picks up a discarded card of one rank (**TP6B**, where the rank is **6** of the suit B in this case) and drops a card (**TD7B**) of the same suit (**B)**, but of a different rank that is numerically adjacent to the discarded card, we can conclude that they have or aiming for a set of (*SBAC ∨ SBBD ∨ SBAD*). | A screenshot of a computer  Description automatically generated  A screenshot of a computer  Description automatically generated |

# Jape Proof Ideas

|  |  |
| --- | --- |
| DESCRIPTION | JAPE PROOF |
| If the opponent (T) drops a card of one suit (TD3A, where 3 is the rank and **A** is the suit) and picks up a discarded card of the same rank, but a different suit **B** (TP3B) within a turn, we can conclude that they have or aiming for a run of the suit **B** (*RB13 ∨ RB14 ∨ RB15 ∨ RB24 ∨ RB25 ∨ RB35).*  *----------*  NOTES:   * We don’t consider if the opponent possibly has adjacent cards of the discarded TD3A since we’re trying to look for a run of the suit B in a certain range of ranking.   + But we do consider what they don’t have so we can remove it from the possibilities in what the opponent could possibly have. (*TD3A (¬(T3C ∧ T3D))* | A screenshot of a computer  Description automatically generated  A screenshot of a computer  Description automatically generated  A screenshot of a computer  Description automatically generated |

# First-Order Extension

## Predicates

Redefining our propositions with predicate logic

|  |  |
| --- | --- |
| PL(a, b) | The player has a card with “a” rank and “b” suit |
| PL\_R(b, v, u) | The player has a run of 3 or more consecutive cards of the same suit, b = suit, v= lower bound of the run, u = upper bound of the run, v<u  (e.g. PL\_R(A, 1, 3) [PL(1, A), PL(2, A), PL(3, A)]) |
| PL\_S(a, n) | Considered true if the player has a set of 3 or 4 cards of the rank a, n = the suit that the set does NOT include, with Z indicates that the set include all four suits (e.g. PL\_S(1, A) = [PL(1, B), PL(1, C), PL(1, D)]) |
| PL\_W(a, b) | Considered true if obtaining a card of “a” rank and “b” suit would create a meld. |
| O(a, b) | The opponent has a card with “a” rank and “b” suit |
| O\_P (a, b) | The opponent picks a card with “a” rank and “b” suit from the discarding pile. |
| O\_D(a, b) | The opponent discards a card with “a” rank and “b” suit |
| DE(a, b) | A card with “a” rank and “b” is in the deck |
| DP(a, b) | A card with “a” rank and “b” suit is being discarded and is not picked up by any player. In our model, this card will never appear in the game again. |

## Functions

* incl\_suit(n, x): Function that returns an object of the next suits that exclude suit n, with x indicating which element in the suit list.
* next\_suit(b): Function that returns an object of the next suit in the suit list.
* prev\_rank(a): Function that returns an object of the previous element of the ranks from a.
* next\_rank(a): Function that returns an object of the next consecutive element of the ranks from a

## Constraints

* For all cards of rank “a” and suit “b”, the card is either in the player’s hand, the opponent’s hand, the deck, or in the dump.

∀a.∀b. ((PL(a,b) ∧ ¬ O(a,b) ∧ ¬DE(a,b) ∧ ¬ DP(a,b)) ∨ (¬PL(a,b)∧ O(a,b) ∧ ¬ DE(a,b) ∧ ¬ DP(a,b)) ∨ (¬PL(a,b) ∧ ¬ O(a,b) ∧ DE(a,b) ∧ ¬ DP(a,b)) ∨ (¬PL(a,b)∧ ¬ O(a,b)∧ ¬ DE(a,b) ∧ DP(a,b)))

* Definition of a run: for all “b” suits, if there exists a lower bound rank “v” and upper bound rank “u” between which the player has all consecutive cards of suit b, then the player has a run.

b. (∃v.∃u. PL(v,b) ∧ PL(next\_rank(v), b) ∧ … ∧ PL(u,b) PL\_R(b,v,u))

* Definition of a set: the player has 3 or 4 cards of the same rank. k = rank, n = the excluded suit (when the player has 3 cards) with ‘Z’ = all suits are in the set (when the player has 4 cards)

k.n. ((PL(k, incl\_suit(n,0)) ∧ PL(k, incl\_suit(n,1)) ∧ PL(k, incl\_suit(n,3)) ∧ … )PL\_S(k,n))

* For all cards(a, b) that the opponent picks up, it follows that the opponent has related cards that form a run or set.

a.b. (O\_P(a,b)(O\_P(a,b)∧ O\_P(prev\_rank(a), b)∧ O\_P(next\_rank(a),b))∨ (O\_P(a,b)∧ O\_P(prev\_rank(a), b)∧ O\_P(next\_rank(a),b) ∧ O\_P(next\_rank(next\_rank(a)),b))∨ … ∨ (O\_P(a,b) ∧ O\_P(a, next\_suit(b)) ∧ O\_P(a, next\_suit(next\_suit(b))) ) ∨ (O\_P(a,b) ∧ O\_P(a, next\_suit(b)) ∧ O\_P(a, next\_suit(next\_suit(b))) ∧ O\_P(a, next\_suit(next\_suit(next\_suit(b)))) ))

* If there exists a card(a,b) that, along with 2 cards currently in the player’s hand, will form a 3-card consecutive run, or is the same rank as 2 cards in the player’s hand, then the player wants that card.

∃a.∃b.((PL(prev\_rank(a), b) ∧ PL(next\_rank(a), b)) ∨ (PL(prev\_rank(a), b) ∧ PL(prev\_rank(prev\_rank(a)), b)) ∨ (PL(next\_rank(a), b) ∧ PL(next\_rank(next\_rank(a)), b)) ∨ (PL(a, next\_suit(b)) ∧ PL(a, next\_suit(next\_suit(b)))) → PL\_W(a, b))

* For all cards(a,b) if the opponent does not pick a card, then the opponent does not have that card  
  ∀a.∀b.(O\_P(a, b) O(a, b))
* For all cards(a,b) if the opponent discards a card, then the opponent does not have that card  
  ∀a.∀b.(O\_D(a, b) O(a, b))
* If there exists a card(a, b) such that the opponent either does not pick or discards with rank ‘a’ and suit ‘b’, then there are no adjacent or related cards in the opponent’s hand that forms a meld.  
  ∃a.∃b.(O\_P(a, b) ∧ O\_D(a, b)) (a-1, b) ∧ a+1, b)) ∨ (a+1, b) ∧ a+2, b)) ∨ (a-1, b) ∧ a-2, b)) ∨ (a, j) ∧ a, k)) ∨ (a, j) ∧ a, k) ∧ a, m))
* For all card(a,b) that is not in the deck, or in the opponent’s hand, or the dump, the player does not want that card.

a.b. ((¬DE(a, b) ∨ DP(a,b) ∨ O\_P(a,b)) PL\_W(a, b))

* For all card(a,b) that the opponent does not pick up, that card would be in the dump.

a.b. (¬O\_P(a, b) → DP(a, b))