Logical Aspects of Multi-agent Systems
Modelling Kemps using Public Announcements and Common Knowledge

Andreea Minculescu
s3932222
a.minculescu@student.rug.nl

David Coslar
s3954943
d.b.coslar@student.rug.nl

Ruhi Mahadeshwar
s4014456
r.u.mahadeshwar@student.rug.nl

June 25, 2023

1 Game explanation

Traditionally, Kemps is a card game played in pairs with the goal of obtaining four cards of the same number and using a secret signal to get your partner to call out "Kemps!". If this secret signal is intercepted by an opposing team, one of the team members can call out "Block Kemps!". If the partner successfully calls out Kemps without Block Kemps, then their team wins a point, otherwise the team who calls out Block Kemps wins a point. If the partner calls out Kemps without their team member having the correct cards, then their team loses a point. The game ends when there are no more groups of four to be made. The winner is the team with the highest number of points. A round ends when a team scores a point.

The game starts by shuffling the cards and dealing four cards to each player. Each player holds a maximum of four cards in his or her hand at any given time. Then, deal six cards to the middle of the table, face up. At the start of the round, each player can discard a card from their hand and then grab any card from the center of the table. There are no turns; players simply exchange single cards from their hand with cards that show up on the table. Here, to exchange cards means to first pick up a card from the table and then to discard a card from your hand. Once no player wants to pick up the cards from the table, new cards are dealt from the leftover deck, and the previous cards on the table are put into a discard pile. Once the leftover deck runs out of cards, the discard pile acts as the leftover deck and the game goes on until there are no more groups of four to be made.

Kemps is usually played with the standard card deck of 52 cards and two teams of two players. However, we have decided to simplify the gameplay:

- We do not have teams in our implementation of Kemps, each player calls out "Kemps!" by themselves. This was done to simplify the implementation of the game since keeping track of secret signals in a computational manner would have been difficult. This means that the Block Kemps aspect of the game is also removed since it is intended to be used in teams.
- We also decided to remove the no turns aspect of the game and introduced a turn based gameplay so each player will only exchange cards from their hand and the cards on the table when it is their turn. This was done since it would have been difficult to implement reaction times of looking at the card and picking up the card.
- For simplicity, we restrict the game to three players.
- Since the Block Kemps aspect is removed, we still wanted a way for players to block other players from achieving Kemps. To do so we implemented a blocking strategy which is explained in Blocking.
- In our implementation, new table cards are laid out when in three turns, no player has picked a card from the table. This mimics the idea that no player wants to pick up the cards from the table.

We decided to implement the gameplay in Python.

2 Terminology

Some of the terminology used in the report is as follows:

- Model/agent/player: the simulated agent that implemented the described strategies
- User: the human player that interacts with the GUI
- Value/number and suit: a card has a number and a suit; e.g. for 5♠ the number/value is 5 and the suit is ♠
- Kemps: formation of four cards with the same value
- Table (cards): the six cards dealt face-up from the playing deck
- Hand (cards): the four cards dealt face-down to each player and visible only to that player
- Playing/leftover (cards/pile): the part of the deck from which new cards are dealt
- Discard (cards/pile): removed table cards, which will be reshuffled into the playing cards once the current playing deck has finished
- To pick/collect a card: to take a card from the table cards and add to the hand cards
- To discard a card: to take a card from the hand cards and add to the table cards
- To deal cards: to pop cards from the playing deck
- Knowledge base: simulated memory storage of an agent in which they store the cards seen on the table and the cards picked/discarded by other agents; used to decide which cards to collect for Kemps (see strategies).

3 Game implementation

The standard 52 card deck was implemented using the Pygame library. The visualization of the game was also implemented using Pygame in `view.py`.

3.1 Card class

An instance of the Card class in `card.py` represents a card from the card deck. Each card has a suit, a value, and an image. The image was used to visualize the card in the game. The cards Ace, Jack, Queen and King have the numbers 1, 11, 12 and 13 respectively. All other cards have the same number as on their card in real life.

3.2 Deck class

During the game there are three sets of cards in front of the players: the cards face up on the table, the cards in the discard pile and the cards in the leftover deck pile. The Deck class initializes these three sets and keeps a track of them as the game progresses. For example when a turn ends, the cards which no one has picked from the table go into the discard pile: the cards in the table set are now moved to the discard pile set.

In the Deck class, `deal_table()` deals cards to the table which the players can exchange from. Cards are dealt to the table at the start of the game and every time a turn or a round ends. `deal_cards_player()` deals cards to the players of the game. At the beginning of the game, all three players get dealt 4 cards from the leftover deck pile. When a player achieves Kemps, the player who achieved Kemps gets dealt new cards from the leftover deck pile and the Kemps cards are removed from all deck sets. If there are no more new cards to be dealt from the leftover deck pile for either the table or the players, the shuffled discard pile then acts as the leftover deck pile. Once there are no cards left to deal from the leftover deck pile and the discard pile is empty, the game ends.

3.3 Game GUI

The game GUI can accessed by running `main.py`, which opens a Pygame terminal. The GUI shows six cards at the top of the window, namely the table cards, and four cards at the bottom of the window, namely the user's cards. Additionally, there are two face-down cards on each side of as placeholders for the models (they are not clickable). The user can interact with the interface by clicking on one table card and one hand card, in which case a swap will occur. The swapping was implemented by continuously listening for click events in the game window and checking the coordinates of the clicks to identify which cards should be swapped. The turn of the user ends when the "Next turn" button is clicked, in which case the event listening loop is exited and the models are updated with regards to the moves made by the user.

3.4 Agent class

An instance of the Agent class has a name, a set of cards which are in the agent's hand, a set of cards to represent the cards that the agent sees on the table, their score in the game so far, a knowledge base (KB) associated with the agent (more information about the implementation of a KB in Knowledge Base) and a set of cards which represent the values of the cards that the agent is not collecting in the game at that moment.

An agent thus represents a player in the game and it can employ one of the three strategies to play the game.

3.4.1 Greedy

In the greedy strategy, the agent only looks at the cards visible to it which are the cards in its hand and the cards on the table. The agent does not keep a KB of which agent has which card. The agent always picks the number(s) from the table which has the highest occurrence in their own hand and on the table. If there are no frequently occurring number(s) in the current round the agent makes no moves, otherwise the agent picks out the most frequently occurring number(s) and swaps them out with the least frequently occurring number(s) in their hand and on the table. The greedy strategy is implemented in the Agent class using `greedy_strategy(kb_based=False, blocking=False)`.

Table 1 shows an example run-through where all agents use the Greedy strategy. For simplicity, we considered only cards with values from 2 to 6 (so a playing deck of 20 cards) because, in this situation, the game ends after one Kemps (not enough hand cards left). Player 1 starts the game: they collect all cards with the value 5 from the table and discard the 4♥ in exchange. Player 1 decided selected the value 5 because a formation of three of a kind is as best as they can do at the moment (all other cards are in pairs or unique). Similarly, player 2 collects all cards with value 6 from the table, as three of a kind is the best outcome. In exchange, player 2 discards 2♠ and 4♣ (randomly selected cards, not part of the formation of 6s). Lastly, player 3 collects all cards with a value of 4 and achieves a Kemps (there are three cards on the table and one in the player's hand).

Player	Table cards	Hand cards	Move	Comment
Player 1	6♥, 5♦, 3♦, 3♣, 4♠, 6♣	2♥, 5♥, 4♥, 5♣	Collects 5	Three of a kind is the highest
Player 2	6♥, 4♥, 3♦, 3♣, 4♠, 6♣	5♠, 6♦, 2♠, 4♣	Collects 6	Three of a kind is the highest
Player 3	2♠, 4♥, 3♦, 3♣, 4♠, 4♣	2♦, 4♦, 2♣, 3♥	Collects 4	Kemps!

Table 1: Example of a game run where all agents use the Greedy strategy.

Note how all agents selected the card value with the highest number of occurrences at the moment. Player 3 won the game by pure chance, due to the fact that the other two players discarded cards with value 4. However, this example proves an important point regarding the importance of knowledge of other players' strategies. Firstly, it helps guide the selection of cards from the table: note how it is disadvantageous for player 1 to collect 5s, as player 2 has a 5 in their hand and can choose to block player 1 for an indefinite number of rounds. Thus, it is arguably better to wait and learn about other players' cards before committing to a value for a Kemps formation. Secondly, players can make a more informed decision with regards to the cards they should discard, in order to block other players from forming Kemps: if players 1 and 2 had known that player 3 was going to collect 4s, then it would have been more advantageous for themselves to discard different cards instead.

3.4.2 KB greedy

Similar to the greedy strategy but this time, the agent keeps a KB and keeps track of which cards are picked and discarded by the other agent. Here the agent only picks up a card number if it is sure that no other agent is collecting that number, i.e., the agent knows that the cards corresponding to that number are either in the discard pile, in the leftover cards pile, on the table, or in the agent's hand and that the number is not being collected by any other agent. If the agent's KB has no knowledge about such cards, such as uncertainty whether or not a card is in another agent's hand or in the leftover card pile, then the agent does not pick any cards from the table. Note that it is then theoretically possible for agents to get stuck in an infinite loop due to not having enough information about any of the cards on the table (they need information about cards in other players' hands who also need information about cards in other players' hands etc). Since this is a natural consequence of a greedy strategy (namely, agents unwilling to make a less optimal action to escape an infinite loop), we simply decided to end the game after 10000 turns. The KB greedy strategy is implemented in the Agent class using `greedy_strategy(kb_based=True, blocking=False)`.

Table 2 shows an example run-through of where all agents use the KB Greedy strategy. Since we used the same seed as for the game described in Table 1, the card distribution is the same and only the strategy determines the different

outcomes. In the first round of the game (above the darker horizontal line) neither players makes a move. This is because they do not yet have enough information about the card distribution and the other players' hands. As discussed above, this is the right strategy for player 1 because collecting 5s is disadvantageous. In the second round of the game, the table cards get redistributed following three turns with no moves. Player 1 still makes no move because it does not have enough information about the distribution of the cards: they do not know if 3♥ is in another player's hand (and indeed it is). Player 2 collects 6s because by this point they have determined that all 6s are either in their hand or in the deck so now it is safe to collect 6s. Lastly, player 3 collects all 3s and achieves Kemps.

Player	Table cards	Hand cards	Move	Comment
Player 1	6♥, 5♦, 3♦, 3♣, 4♠, 6♣	2♥, 5♥, 4♥, 5♣	No move	Wants 5 but not sure if available
Player 2	6♥, 5♦, 3♦, 3♣, 4♠, 6♣	5♠, 6♦, 2♠, 4♣	No move	Wants 6 but not sure if available
Player 3	6♥, 5♦, 3♦, 3♣, 4♠, 6♣	2♦, 4♦, 2♣, 3♥	No move	Wants 3 but not sure if available
Player 1	6♥, 3♦, 3♠, 6♠, 4♠, 3♣	2♥, 5♥, 4♥, 5♣	No move	Wants 3 but not sure if available
Player 2	6♥, 3♦, 3♠, 6♠, 4♠, 3♣	5♠, 6♦, 2♠, 4♣	Collects 6	Knows that all 6 are available
Player 3	2♠, 3♦, 3♠, 4♣, 4♠, 3♣	2♦, 4♦, 2♣, 3♥	Collects 3	Kemps!

Table 2: Example of a game run where all agents use the KB Greedy strategy. The bolded vertical line separates different rounds of the game.

The KB Greedy strategy is clearly an improved version of the pure Greedy strategy. Even though the same player won in both cases, player 3 won purely due to luck because the three table cards with value 3 were dealt from the beginning, rather than having been discarded by other agents. Moreover, player 1 does not immediately start collecting 5s but instead waits for more information (as explained before, this is the right decision). Overall, this strategy informs the selection of cards from the table but it still does not inform the agent which cards it is advantageous to discard. Moreover, given the knowledge base information, the agent could also purposefully try to block other agents from forming a Kemps. Next, we propose an even more advanced strategy.

3.4.3 Blocking

This strategy can be seen as an extension of the KB greedy strategy. Here, first priority is given to the needed cards from the table (KB greedy strategy). Once all cards needed are in the player's hand, the player will then look at its KB and try to take cards from the table which the player knows are being collected by other players as well, which is the blocking extension. The blocking strategy is implemented in the Agent class using `greedy_strategy(kb_based=True, blocking=True)`

Table 3 shows an example run-through where all agents use the Blocking strategy. Here, we used a different setup than for the other two strategies because the games described above would end too quickly to show the effects of the Blocking strategy. Thus, we considered cards with values from 2 to 8 (so a playing deck of 28 cards). In the first round of the game, the agents make no move because there is not yet enough information on the cards' distribution. In the second round, player 2 starts collecting 3s because they now know that three of the 3s are on the table or in the discard pile and the fourth is in their own hand. Similarly, player 3 starts collecting 4s. The behaviour shown in rounds 2 and 3 are consistent with the KB Greedy behaviour as described in KB greedy. Now let us skip a few rounds ahead. Player 2 is collecting 3s at the moment and has an extra inconsequential card (the 8♦). Moreover, player 2 knows that player 3 picked up 4s before. Thus, player 2 decides to discard the inconsequential card in favour of one of the 4s on the table, in order to block player 3 from achieving Kemps. Now, player 3 saw player 2 pick one of the 4s and, as a consequence of the KB Greedy strategy, decides that they should not collect 4s anymore (since another player has a 4) and switches to collecting 8s instead.

The Blocking strategy is clearly an improved version of the KB Greedy strategy: players focus on achieving Kemps, which is ultimately the goal of the game, while trying to block other players from achieving Kemps in the process. This is done by replacing in the hand that are inconsequential for the Kemps formation with cards on the table that other players need to collect and holding onto those cards for as long as possible (it should not prevent the player from forming their own Kemps). The effects of this strategy are clear in Table 3: player 3 would have achieved Kemps had player 2 not blocked them from doing so. This strategy could still be improved: player 3 does not understand that player 2 did not pick the 4♠ to collect for Kemps and that player 2 might discard the card again soon in favour for a card they need. Thus, higher-order theory of mind could further improve performance.

Player	Table cards	Hand cards	Move	Comment
Player 1	2♥6♥, 4♥, 3♣, 2♠, 4♠	2♦, 8♥, 8♣, 6♣	No move	Wants 2 but not sure if available
Player 2	2♥6♥, 4♥, 3♣, 2♠, 4♠	6♠, 3♠, 8♦, 8♠	No move	Wants 3 or 6 but not sure if available
Player 3	2♥6♥, 4♥, 3♣, 2♠, 4♠	2♣, 6♦, 5♥, 4♣	No move	Wants 2or 4 but not sure if available
Player 1	3♥, 5♣, 4♦, 7♥, 3♦, 5♦	2♦, 8♥, 8♣, 6♣	No move	Wants 5 or 3 but not sure if available
Player 2	3♥, 5♣, 4♦, 7♥, 3♦, 5♦	6♠, 3♠, 8♦, 8♠	Collects 3 Discards 6♠ and 8♠	Knows that all 3 are available
Player 3	8♠, 5♣, 4♦, 7♥, 6♠, 5♦	2♣, 6♦, 5♥, 4♣	Collects 4 Discards 6♦	Knows all 4 are available
Player 1	8♠, 5♣, 6♦, 7♥, 6♠, 5♦	2♦, 8♥, 8♣, 6♣	Collects 6 Discards 8♣ and 2♦	Knows that all 6 are available
Player 2	8♠, 5♣, 8♣, 7♥, 2♦, 5♦	3♦, 3♠, 8♦, 3♥	No move	Nothing to collect
Player 3	8♠, 5♣, 8♣, 7♥, 2♦, 5♦	2♣, 4♦, 5♥, 4♣	Collects 2 Discards 5♥	Knows all 2 are available
...				
Player 2	4♠, 8♥, 5♣, 4♥, 5♠, 7♣	3♦, 3♠, 8♦, 3♥	Picks 4♠ Discards 8♦	Blocks player 3
Player 3	8♦, 8♥, 5♣, 4♥, 5♠, 7♣	2♣, 4♦, 5♥, 4♣	Collects 8 Discards 4♦ and 4♣	Changes strategy after block
...				

Table 3: Example of a game run where all agents use the Blocking strategy. The bolded vertical lines separate different rounds of the game.

4 Model Formalization

The game states and logic can be formalized using Kripke models. However, this is not an easy task, as the game state space is large (combinations of 4 hand cards over 52 deck cards for 3 agents) and each action can change the entire state space instead of restricting the Kripke model: an agent that seems to be collecting cards with the number 2 can change their mind completely halfway through the game and start collecting cards with the number 5 instead. As a result, the entire knowledge base of the other two agents is reset.

4.1 Kripke Model

Consider the following notations. Let $A = \{1, 2, 3\}$ be the set of agents and let $m = |A| = 3$ be the number of agents. Let $C = \{2\heartsuit, 2\diamondsuit, 2\clubsuit, 2\spadesuit, ..., A\heartsuit, A\diamondsuit, A\clubsuit, A\spadesuit\}$ be the set of playing cards from a standard deck without Jokers. Lastly, let P be the set of propositions $hand_i^c$, which can be interpreted as "player i has card $c \in C$ in their hand". Now, we define a Kripke model $M = \langle S, \pi, R \rangle$ as follows:

- S is the set of states $S_c = \{s_1^c, s_2^c, s_3^c, s_0^c\}_c$ for all $c \in C$, which means that each card is associated with a set of four states.
- $\pi : P \to (S_c \to \{\text{t}, \text{f}\})$ such that:
  - $\pi(s_1^c)(hand_1^c) = \text{t}$; $\pi(s_1^c)(hand_2^c) = \text{f}$; $\pi(s_1^c)(hand_3^c) = \text{f}$ for all $c \in C$. Intuitively, in state s_1^c agent 1 has card c in their hand.
  - $\pi(s_2^c)(hand_1^c) = \text{f}$; $\pi(s_2^c)(hand_2^c) = \text{t}$; $\pi(s_2^c)(hand_3^c) = \text{f}$ for all $c \in C$. Intuitively, in state s_2^c agent 2 has card c in their hand.
  - $\pi(s_3^c)(hand_1^c) = \text{f}$; $\pi(s_3^c)(hand_2^c) = \text{f}$; $\pi(s_1^c)(hand_3^c) = \text{t}$ for all $c \in C$. Intuitively, in state s_3^c agent 1 has card c in their hand.
  - $\pi(s_0^c)(hand_1^c) = \text{f}$; $\pi(s_0^c)(hand_2^c) = \text{f}$; $\pi(s_0^c)(hand_3^c) = \text{f}$ for all $c \in C$. Intuitively, in state s_0^c no agent has card c (i.e. card c is in the deck, on the table or in the discard pile).
  - For simplicity, we don't consider impossible situations as part of the model (e.g. more than one agent having a card at the same time).
- $R = \{R_1, R_2, R_3\}$ is the set of relations $R_i \subseteq S_c \times S_c$ for all $c \in C$ and $i \in \{1, 2, 3\}$. For convenience, let us define a "memory" predicate such that $M_{i,T}$ means "player i saw cards c' on the table" for all cards $c' \in C$ that were at some point throughout the game on the table and an equivalent $M_{i,H}$ predicate for cards seen by player i in their

hand. Now, for arbitrary states $(t, u) \in S_c \setminus \{s_i^c\} \times S_c \setminus \{s_i^c\}$ we have $t \to_{R_i} u$ for all $c \in C$ such that $c \notin M_{i,T}$ and $c \notin M_{i,H}$. Intuitively, this states that agent i knows its own cards and the cards that have been shown on the table throughout the game.

In essence, this setup can be understood as building a separate sub-model for each card in the deck. Each sub-model is updated as a result of picking cards from the table, discarding cards from hand and replacing table cards. These actions were modeled as public announcements (see Public Announcements). Moreover, it is worth noting that it is not necessarily the case that each action restricts the Kripke model. Instead, if an agent changes the number it was collecting up to that point, it can be the case that the number of relations increases as parts of the knowledge base of the other two agents are reset. Additionally, as cards are picked from the table and discarded from hand, it can be the case that the true state within a Kripke sub-model changes. See Example Kripke Model on Simplified Game for an example of how to build and update the Kripke model explained above.

4.2 Knowledge Base

A knowledge base (KB) is constructed for the three players using the KnowledgeBase class. The KB of a player represents the knowledge or belief that the player has about every player during the game, including itself. The KB also has belief about which cards are or are not in every player's hand.

The KB is initialized before the start of game by considering it possible that every other player has the cards the player can't see, i.e., the player only knows that the cards in their hand, the cards laid out on the table and the cards that the player has seen go in the discard pile are not in any other players' hand. This is done through `set_knowledge_of_other_cards()` and `set_knowledge_own_deck()`. Every time the players see the cards on the table go to the discard pile and new cards being set on the table, `set_knowledge_of_other_cards()` and `update_discard_pile()` is used to update the KB of each player to represent the players seeing that the new cards and the discarded cards are not in any players hand. Every time an player is given a new hand of cards, `set_knowledge_own_deck()` is used to represent the player knowing that the cards in their hand are not in any other players hand and not in the discard pile, and not on the table and not in the leftover cards pile.

As player's pick up cards and discard them, the KB of each player is updated since the discarded cards are no longer in any player's hand. More detail about this is in Public Announcements.

The KB is a dictionary of dictionaries. Essentially, for each possible card in the standard 52 card deck, the player keeps a track of whether the other players, including itself have the card or not. A value of `True` for another player for a specific card represents that the player with the KB believes that the other player has that card. If the value is `False`, it represents the player with the KB believing that the other agent does not have that card. If the value of a given card is `True` for player 1 and `False` for player 2 and player 3, then the player with the KB knows that the card is with player 1. If the value of a given card is `False` for all players, then the player with the KB knows that the card is either on the table, in the discard pile or in the leftover deck. If the value of a given card is `True` for all players except for the player with the KB, then the player with the KB does not know for sure if any other player has this card, so the player with the KB considers it possible that all other players have this card.

The implemetation of the KB was inspired by the Logical Boerenbridge project for the 2021 Logical Aspects of Multiagent Systems course.

4.3 Public Announcements

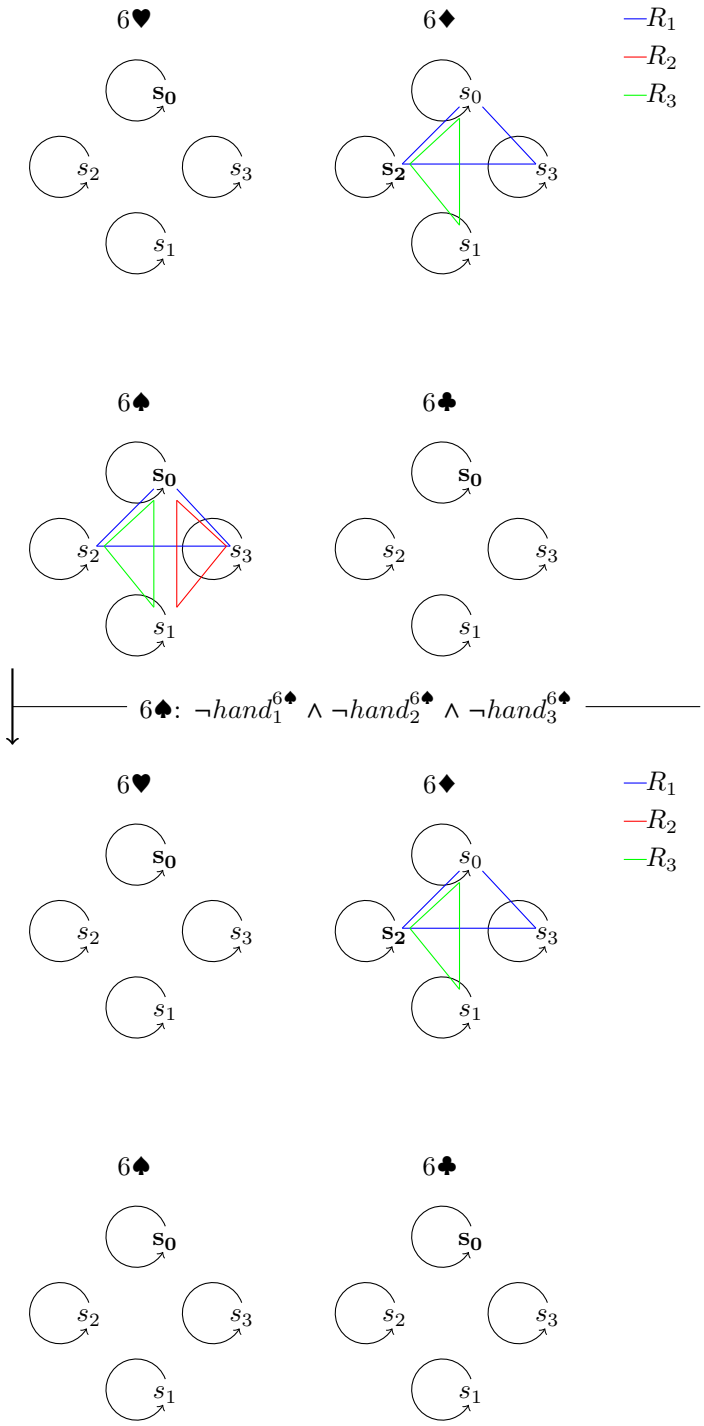
Each time a player picks up a card from the table and discards a card from their hand, two public announcements are made: one to describe which card (suit and number) was picked by which player (announcement type PICKED) and one to describe which card (suit and number) was discarded by which player (announcement type DISCARDED). Each time a player achieves Kemps, a public announcement of type KEMPS is made to describe which card number was part of the Kemps and which player has called out Kemps. Announcements are stored as a list during an agent's turn and when the agent's turn ends, the list of announcements is broadcasted to the other agents in the game using `make_announcements()` and the agent's recieve the broadcasted announcement using `receive_announcement()` in the Agent class.

When an announcement type PICKED reaches an agent who is not the sender of the announcement, the agent updates its KB through `set_card_knowledge_of_individual()` in the KnowledgeBase class by setting the value of the card for the sender as True to represent the agent knowing that the sender now has the card in their hand. The number of the card picked is also added to the do not collect set of the agent to make sure that the agent who is not the sender does not pick cards of the same number in the future. When an announcement of type DISCARDED reaches an agent,

the agent updates its KB through `set_discard_knowledge()` in the KnowledgeBase class by setting the value of the card for all the agents in the game as False to represent the agent knowing that the sender has discarded the card, meaning that the card is not in any agent's hand. The number of the card picked is also removed from the do not collect set of the agent to make sure that the agent who is not the sender does pick cards of the same number in the future. When an announcement of type KEMPS is received by an agent, the agent adds the number of the card picked to its do not collect set to make sure that the agent who is not the sender does not pick cards of the same number in the future.

By using Public Announcement Logic (PAL) we give the agents a way to update their Kripke S5 models. When updating their Kripke models, the agents can reduce the number of possible states in the model and also increase common knowledge in the game. The public announcements made by the agents are always true and contain information which is accessible to every agent playing the game. Thus, after a public announcement of a message ϕ, ϕ becomes common knowledge amongst all players of the game. As the game progresses and more public announcements take place, the agent's receive more information about the real world which results in a decrease in the number of relations between possible states, and can also result in some possible states being eliminated. This allows a player to eliminate possibilities of who is or isn't collecting which cards from the deck, which again allows the agent to collect the card they think no one else is collecting in order to win and call Kemps. Thus, by incorporating announcements, we can introduce common knowledge in the Kripke model.

4.4 Example Kripke Model on Simplified Game



We now show an example of how to build and update the Kripke model, based on the game described in 1. Here, we only show the sub-models for the cards with value 6 but the process can be easily generalized to all other cards. The top four diagrams show the states and relations of Kripke the model in the first round and the bottom four diagram show the states and relations of the Kripke model in the second round.

Initially, 6♥ and 6♣ are on the table. This is illustrated in the corresponding sub-models: s_0 is the true state (the card is on the table) and all agents are aware of this (they can see the cards). Next, 6♦ is in the hand of player 2, and, thus, i) s_2 is the true state, ii) player 2 knows where the card is and iii) players 1 and 3 do not know anything about the card. Lastly, 6♠ is in the playing deck and no agent knows anything about its whereabouts. This clearly shows why player 2 did not make a move during the first round: they do not know whether 6♠ is in another player's hand.

At the beginning of the next round, the table cards are redistributed. Now, the 6♠ is on the table. We model this event as a public announcement that $\neg hand_1^{6♠} \wedge \neg hand_2^{6♠} \wedge \neg hand_3^{6♠}$ (no player has 6♠ in their hand). As a result it becomes common knowledge that 6♠ is on the table (formally, $C(\neg hand_1^{6♠} \wedge \neg hand_2^{6♠} \wedge \neg hand_3^{6♠})$) and the sub-model for 6♠ is updated accordingly. It is now clear why player 2 started collecting 6s: they now know that the other players do not have any 6s. The same cannot be said for players 1 and 3, as they still do not know where 6♦ is.

Next, player 2 discard 2♠ and picks up 6♥. Since the other players can see these events, picking and discarding cards are also modeled as public announcements. In this situation, two public announcements are created: $\neg hand_1^{2♠} \wedge \neg hand_2^{2♠} \wedge \neg hand_3^{2♠}$ and $hand_2^{6♥}$. As a result, we now have $C(\neg hand_1^{2♠} \wedge \neg hand_2^{2♠} \wedge \neg hand_3^{2♠})$ and $Chand_2^{6♥}$. This information is used to update corresponding Kripke sub-models such that the players can decide which cards it is advantageous to collect at any given point.

5 Final remarks

For simplicity, the model formalization is defined for exactly three players but it can easily be extended to a larger or smaller number of players (between 2 and 10). Moreover, we assume that the players are perfect logicians, meaning that they can reason in a logical way and have a perfect memory of the cards seen on the table.

Due to the large state space, we decided to greatly simplify the Kripke model by essentially creating independent sub-models for each card. As a consequence, an agent cannot reason about two cards at the same time based on one sub-model (e.g. "player 1 has 2♥ and player 2 has 5♠"). However, we consider that this is not a problem given the strategies implemented: an agent reasons whether to start collecting one value at a time.

Lastly, the game is highly luck-based, as it depends on how the card deck is shuffled. Therefore, the effectiveness of the different strategies may not be immediately clear unless multiple games are played against the models.

6 Experimental results

7 Conclusion

8 Discussion and future research