University of Groningen
Faculty of Science and Engineering

Examining the Effects of Theory of Mind on playing the Game of 'No Thanks!' using ACT-R-based Cognitive Models

Master's Thesis
Human-Machine Communication

Tom Renkema

Primary supervisor:
Prof. dr. Niels A. Taatgen
Artificial Intelligence, University of Groningen

Secondary supervisor:
dr. Jennifer K. Spenader
Artificial Intelligence, University of Groningen

August 20, 2018

Abstract

Theory of mind is the ability for an individual to reason about the mental state of themselves and others. This project studied the effects of theory of mind on playing the card game of *No Thanks*! using cognitive models. Multiple cognitive models using different orders of theory of mind were developed using instance-based decision making and a partial implementation of the cognitive architecture ACT-R. Additionally, the performance of these models against human players was examined. An iOS application was developed for the Apple iPad with which a user could play the game of *No Thanks!* against the developed cognitive models. Both an experiment with human subjects and simulations in which the models played one other were conducted. The results of both experiments showed that models using first-order or second-order theory of mind had a significant advantage over models using zero-order theory of mind. However, the advantage of using second-order theory of mind over first-order theory of mind appears to be not as substantial. Furthermore, this project demonstrated that an academic AI method like instance-based decision making is suitable for creating a competitive game AI for a competitive game.

Contents

1 Introduction

Humans are capable of not only understanding their own beliefs and intentions, but are also able to infer these of others and to acknowledge that these can be different from their own. *Theory of mind* is the ability for an individual to reason about the mental state of themselves and others (Premack and Woodruff, 1978). An individual can use different orders of theory of mind. One is said to have *zero-order* theory of mind (ToM_0) if they cannot reason about the mental state of others (thus they are not using theory of mind abilities). Whereas an individual has *first-order* theory of mind (ToM_1) when they do reason about the mental state of others. However, an individual using first-order theory of mind does not recognize that others may have theory of mind as well. Instead they assume others have zero-order theory of mind.

In contrast, when an individual does recognize the theory of mind abilities of others, it is said one uses a higher-order theory of mind approach (i.e. second-order or higher). When using n-order theory of mind, one assumes others use n-1-order theory of mind. Therefore, having *second-order* theory of mind (ToM_2) assumes others use first-order theory of mind and so on. In other words, an individual using second-order theory of mind one is able to reason: "I think that you think that I think...".

Multiple studies have found that humans are able to use theory of mind recursively and can reason about the theory of mind abilities of others in their decision making process (i.e. have higher-order theory of mind). Children can have this ability as early as age 5 to 9 (Perner and Wimmer, 1985; Miller, 2012). Evidence has also been found of humans using higher-order theory of mind in playing strategic games (e.g. Hedden and Zhang (2002); Meijering et al. (2011)).

Besides conducting experiments with human subjects, agent-based computational cognitive models can also be used to study whether using a different order of theory of mind leads to different outcomes. The aim for this study was to examine the effects of theory of mind on playing the card game of *No Thanks!* by using cognitive models. Multiple cognitive models using different orders of theory of mind were developed using instance-based decision making and a partial implementation of the cognitive architecture ACT-R (Gonzalez and Lebiere (2005); Anderson (2009) respectively). Additionally, the performance of these models against human players was studied. To that end, an iOS application was developed for the Apple iPad with which a user can play the game of *No Thanks!* in with these models serve as their opponents.

Section 1.1 gives an overview of the game artificial intelligence used in the video game industry, the ACT-R architecture, instance-based decision making, and modelling theory of mind. Section 2 explains the game of *No Thanks!*, its presentation on the iPad, and why it was chosen for this project. Section 3 explains how the developed cognitive models were implemented. Section 4 discusses how the application was coded. The conducted experiments and results are discussed in section 5. The implications of these results are discussed in section 6.

1.1 Background

1.1.1 Game artificial intelligence

In video games, techniques drawn from the field of artificial intelligence are used to define the behaviour of non-playable characters (NPCs). A NPC is an agent in a video game that is not controlled by the human player. Such an agent can for example serve as an opponent in a game of chess or as an enemy in a first-person shooter. For game developers, the goal is to program such agents in a way so they show believable and reasonable behaviour to the player. Multiple techniques have been applied in the video game industry with varying degrees of complexity. This section will give an overview of the common techniques used for game AI in the video game industry.

Finite-state machines

Finite-state machines (FSMs) are one of the most commonly used technique in game AI (Buckland, 2005). In a FSM, the agent's behaviour is split into a collection of discrete states. Each state represents a specific behaviour or internal configuration. States are connected by transitions. These links are responsible from switching from one state to another when a certain condition is met. The agent can only be in one state at a time.

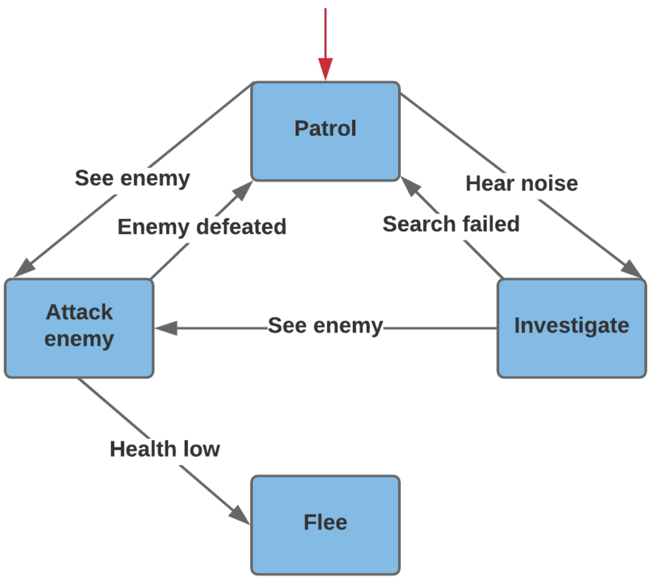

Figure 1: FSM diagram showing the behaviour of an agent guarding a castle (adapted from Dawe et al. (2013).

Figure 1 shows an example of FSM modelling a simple behaviour of an agent (NPC) that has to guard a castle. This agent patrols the castle until it hears a noise or sees an enemy. If one of those conditions are met, it will either investigate the noise or attack the enemy. It will continue attacking until either the enemy is defeated or its own health drops too low. In case of the former it will return to its patrol state, while in case of the latter it will flee.

This technique allows a developer to fully specify the agent's behaviour with relative ease. However, a downside of FSMs is that they scale poorly due to its low-level logic. For more complex behaviours a FSM can become very large, which makes it inflexible and increasingly more difficult for developers to interpret (Rabin, 2000).

Behaviour trees

A *behaviour tree* consists of nodes that each define a specific behaviour (i.e. an individual action that the agent can perform) (Isla, 2005). Every decision making process starts at the root node of the tree. Each behaviour node can have one or more child nodes (thus creating a tree-like data structure). Also, each node in the tree contains a precondition and an action. The agent will only access a node when its precondition is met and consequently executes the node's corresponding action. It then checks whether any precondition of its child nodes are met. If so, it performs that child node's action. This process continues until no child nodes' preconditions are met or a node does not have any child nodes left.

Behaviour trees differ from FSMs in that they are stateless and do not require transitions. This means the algorithm does not need to keep track of what behaviours were executed previously for its decision making process. As a result, behaviour trees do not suffer from the scalability issues of FSMs, because nodes can be designed independent from each other. Thus, when adding or removing new nodes in a small part of the tree, it is not required to alter other parts of the tree. Whereas adding or removing a state in a FSM requires changing the conditions of all other states that have transitions to that altered state. Therefore, behaviour trees are more flexible than FSMs overall. However, processing time can be significantly longer as the decision making process always starts at the root of the tree regardless of what the agent has executed before. This issue becomes more severe when the tree is larger and thus more nodes need to traversed (Rabin, 2000).

Utility-based system

A utility-based system uses a heuristic function to assign a floating-point value (often referred to as the utility value) to each possible action the agent can perform (Dill et al., 2012). This utility value shows the preferability of each action in the current game state. One approach is to let the agent execute the action with the highest utility value. Alternatively, one can use the obtained utility values to seed a weighted random selection, meaning the most preferable actions have a higher chance of being selected (thus adding more randomness to the decision making process).

A utility-based system is suitable for games in which an agent needs to make decision based on multiple continues values (e.g. distance to enemy, number of bullets left etc.). FSMs and behaviour trees are fundamentally built on Boolean conditionals, which can quickly results in complicated models when dealing with many game state parameters. An utility-based system is more suitable for such situations.

Planners

Planners have a fundamentally different approach in defining the behaviour of an agent than the aforementioned techniques. Goal-oriented action planners (GOAP) plan a sequence of actions to achieve a specified goal state (Gorniak and Davis, 2007). Contrary to FSMs and behaviour trees, the order in which an agent can execute actions is not directly determined by the scripted transitions and the fixed structure of the tree respectively. Instead the agent is given the current state of the game, its available actions and their effects on the game state, the requirements for each action (i.e. the precondition), and the goal that needs to be achieved. Given this information, the agent then attempts to chain the actions together to get from its initial state to the goal state. GOAP applies backwards chaining to construct this sequence (Feigenbaum et al., 1988). This method starts with the goal state and works backwards to assert which actions need to be satisfied so the goal state can be achieved.

Another popular planner applied in game AI is the hierarchical task-network planner (HTN ; Gorniak and Davis (2007)). Similar to GOAP, this planning algorithm attempts to build a sequence of actions that will get the agent to the goal state. However, HTN uses forward chaining as opposed to backward chaining. This method starts with the current world state and asserts which actions need to be performed to achieve the goal state.

Gap between academic AI and industrial game AI

A common theme among the aforementioned techniques used for game AI in the video game industry is that these are relatively old AI techniques. Academic research in game AI has largely shifted focus to topics such as reinforcement learning, neural networks, and cognitive modelling in games (e.g. Ghory (2004); Silver et al. (2016); Sanner et al. (2000) respectively). Despite being mature research subjects in the field of AI, these state of the art tools are rarely used in the video game industry.

To that end, Yannakakis (2012) and Champandard (2003) state that there is a considerable gap between academic game AI and industrial game AI. Yannakakis (2012) argues that ”The key message of academic AI has been that industry does not attempt to use sophisticated AI techniques with high potential (e.g. neural networks) in their games. On the other end, the central complaint of industrial game AI has been the lack of domain-knowledge and practical wisdom when it comes to realistic problems and challenges faced during game production.”
(p. 1).

One of these novel techniques in academic AI is instance-based decision making (Gonzalez and Lebiere, 2005). This approach has been studied considerably in the field of cognitive modelling. This technique relies on the cognitive architecture ACT-R, which will now be discussed.

1.1.2 ACT-R architecture

ACT-R is a cognitive architecture consisting of multiple modules that each have a particular cognitive function (Anderson et al., 2004; Anderson, 2009). Although there are several modules, the cognitive models developed during this project are based exclusively on the mechanisms of the declarative module. Therefore only this module will be discussed. For an extensive overview of the entire ACT-R architecture, please refer to Anderson (2009).

In ACT-R, declarative memory is used to store facts that can be retrieved upon request. These facts are stored in chunks. Whether a fact can be retrieved from declarative memory depends on the activation of the chunk that fact is stored in. If a chunk has an activation value above the retrieval threshold, it can be retrieved and accessed by the model. In other words, that piece of information can be remembered by the model. The activation of a chunk depends on the time that has passed since it was stored in declarative memory and how frequently it has been retrieved. This means that chunks decay in activation over time and that the speed of this decay depends on how often a chunk has been retrieved before. This results in model behaviour in which commonly used information is remembered more easily than information that is rarely utilized.

1.1.3 Instance-based decision making

Instance-based decision making is at the core of the decision making process of this project's cognitive models. In instance-based decision making, a cognitive model makes decisions based on a collection of prior episodes (Gonzalez and Lebiere, 2005). In other words, an agent chooses its actions through the use of accumulated experience. This theory is based on evidence that humans use this approach when under conditions involving uncertainty, stress or task overload (Pew and Mavor, 1998; Klein et al., 1993; Zsambok and Klein, 2014).

Instance-based decision making has been successfully applied in modelling human behaviour in a variety of decision making tasks and games, such as the prisoner's dilemma (Gonzalez and Lebiere, 2005), backgammon (Sanner et al., 2000), and the lemonade game (Reitter et al., 2010). The cognitive architecture ACT-R is used to implement the approach.

In ACT-R, instances are represented as chunks in declarative memory. When a model encounters a new scenario and has to make a decision, it attempts to find the instance (i.e. chunk) that is most similar to the current context. Partial matching is used to determine the similarity of each chunk. This process will be further explained in section 3.3. The idea behind this approach is that the more similar a past experience is to the current context, the more useful the corresponding chunk is at that moment. Additionally, an advantage of this approach is that a model can still make a considered decision if it encounters a novel situation, because it chooses to perform the action that it did in a previous situation (i.e. instance) that most closely resembles the current one.

An agent using instance-based decision making does not acquire new information and thus does not learn from new experiences. Gonzalez et al. (2003) have proposed an expansion on IBD referred to as *instance-based learning theory* (IBLT), which does provide a framework for agents to acquire and update instances. However, the cognitive models presented in this project do not learn from new experiences and make their decisions based on a fixed set of predetermined instances. Therefore these models make use of instance-based decision making and not of the instance-based learning theory.

1.1.4 Modelling theory of mind

Agent-based computational models have been used to study theory of mind. Such models allow researchers to precisely control the mental capabilities of their test subjects. As such, an agent can be given a particular theory of mind capacity and compared to another agent with a different capacity.

De Weerd et al. (2013) let such computational models compete against one another in a number of zero-sum games. The goal of this study was to determine whether the ability to make use of a higher-order theory of mind is advantageous in a competitive setting. Its results showed that using higher-order theory mind does indeed benefit agents competitively. However, whereas using both first-order and second-order theory of mind had a significant advantage over agents with a lower order of theory of mind, using an order higher than second resulted in only marginal gains.

Van Maanen and Verbrugge (2010) presented a computational cognitive model of second-order theory of mind. This model was developed in ACT-R and attempted to model how people apply second-order theory of mind and the cognitive limitations of this task. The model relied on a decision tree strategy to reason about the opponent. This decision tree was implemented by distinguishing between declarative memory and working memory. The former is responsible for retrieving successive reasoning steps, whereas the latter is required to temporarily store these reasoning steps while the next step is retrieved from declarative memory.

In order to test the validity of this model, it was fit on the experimental data obtained from Meijering et al. (2010). This study let participants play a variant of the centipede game called Mable Drop. This is a turn-based game in which it is essential for a player to reason about their opponent and therefore requires higher-order theory of mind reasoning. The model was able to successfully predict the participants' data.

1.2 Project goals

This project attempts to develop a game AI for a strategic turn-based game using the novel approach of using instance-based decision making models. The card game *No Thanks!* is used during the project. Although academic game AI research has introduced newer methods, the video game industry still very much relies on traditional game AI methods. Therefore this project attempts to demonstrate that an academic AI method like instance-based decision making is suitable for creating a competitive game AI for a competitive game.

Additionally, the effects of theory of mind on playing the game of *No Thanks!* is studied by using these cognitive models. Multiple cognitive models using different orders of theory of mind will be introduced. As mentioned before, these models rely on instance-based decision making and a partial implementation of the cognitive architecture ACT-R. In doing so, theory of mind is used in a gaming scenario and the effects of this ability are studied by conducting two main experiments. The first experiment attempts to study the performance of the developed cognitive models against human players. The second experiment focuses on model versus model behaviour by conducting simulations to further examine whether using a different order of theory of mind has an advantage over another.

2 The game of No *Thanks!*

2.1 The rules

No Thanks! is a card filler game which can be played by 3 up to 7 players and takes about 20 minutes to play. The game is designed by Thorsten Gimmler and was published in 2004 in more than 15 countries [1]. The game contains 33 unique cards numbered 3 to 35 and a number of pass tokens. Each card is worth its face value in points, while a pass token subtracts one point for each token a player owns (worth -1 point). Each player gets 11 pass tokens at the start of the game. The goal of the game is to get the fewest points possible.

During each turn, the top card of the shuffled deck of cards is placed face-up on the playing field (referred to as the playing card). When it is their turn, a player can choose between two possible actions.

1. Do not pick up the playing card by placing a pass token on it. The turn then continues to the next player (moving clockwise). If a player does not have any pass tokens left, they are forced to pick up the playing card.
2. Pick up the playing card, along with any pass tokens that have already been placed on that card, and turn over the next card from the deck. The same player then again has to decide which action to take with the new playing card (it is still their turn).

Picking up a card means its face value is added to the player's score minus any number of pass tokens that were on that card. Every card a player takes is placed face-up on the playing field in front of the player. This means that all players can see each other's cards. However, the number of pass tokens that players own is only visible to the player themselves.

The value of every card owned by a player is added to their score. However, if a player owns a sequence of cards, then only the value of the lowest card of that sequence is added to their score. For example, if a player owns the cards 20 and 22, their total score would be $20 + 22 = 42$ points. Yet if that player completes the sequence 20 to 22 by picking up card 21 (thus creating 20, 21, 22), these cards would only be worth 20 points (21 and 22 are then worth 0 points). A sequence starts at two cards and has no maximum length. A player can have multiple separate sequences.

However, 9 random cards are removed from play before the game starts. These cards are unknown to all players. This means a card that would complete a player's sequence may actually not be in the game (unknowingly to the player) and thus cannot be taken by anyone. The game ends when all 24 cards of the deck have been taken by the players. The player with the lowest score (cards points minus number of owned pass tokens) wins the game.

[1] https://www.boardgamegeek.com/boardgame/12942/no-thanks

The simple set of rules of *No Thanks!* provoke strategic gameplay, as a player not only needs to keep track of their own cards, but also of those of their opponents. This means that each player can see if their opponents have any (partial) sequences of cards and infer how important a specific playing card is for them to adjust their own strategy accordingly.

Additionally, it is important to predict how many pass tokens each player has left, as these are private and are vital in the strategy of a player. This is because a player is forced to pick up a card if they have no tokens left, regardless how bad that card is for them. Therefore, it can still be good practise to pick up a high card that no other player wants in order to get the pass tokens that are placed on them. This is also opens up the possibility to create a sequence with this high card. More strategies in this game will be discussed in more detail later in this report.

These set of rules make *No Thanks!* a suitable game for this project due to a number of reasons. Firstly, having 9 random cards removed from play and the opponent's token supply being unknown, means that uncertainty is always a significant factor in the player's decision making process. Instance-based decision making is therefore a suitable approach, as evidence suggests that people make experience-based decisions when dealing with uncertainty (Pew and Mavor, 1998; Gonzalez and Lebiere, 2005).

Secondly, using theory of mind in playing *No Thanks!* is vital to one's strategy, as it is important for a player to try to predict which action their opponents will take. A player can predict their opponent's action by primarily assessing whether the playing card would be part of sequence if taken. This is possible because the owned cards of all players are open knowledge. Therefore a cognitive model should be able to use theory of mind to successfully play the game of *No Thanks!*. Additionally, the effect of using different orders of theory of mind can be studied.

Lastly, despite provoking strategic gameplay, the set of rules of *No Thanks!* are relatively easy to understand for experimental participants. This makes it a suitable choice for this project, as a human subject who does not understand the rules could heavily influence the results. A game with more complex rules would therefore be undesirable.

2.2 User interface design and usability

Figure 2 shows a screenshot of the app at the start of the game. The app is designed for 4 players, because this number of players makes optimal use of the iPad's screen space (one player on every side). Additionally, an online poll among players of the *No Thanks!* board game showed a preference for playing with 4 or 5 players to ensure the best gameplay [2].

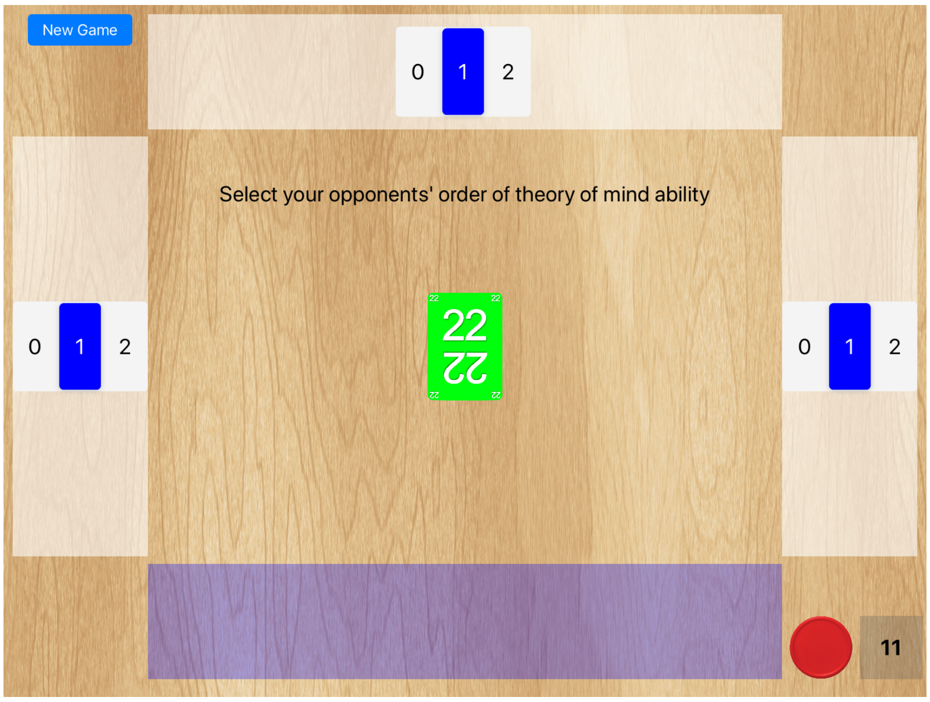

Figure 2: Screenshot of the app at the start of the game

The user interface (UI) is designed as follows. The four white rectangles represent the field of each player where they will place their owned cards. The field turns blue if it is the turn of its corresponding player. The user is the player at the bottom (their field being blue indicates it is their turn). The card in the middle of the screen is the current playing card (card 22 in Figure 2). The label on the bottom right of the screen shows how many tokens the user currently owns.

At the start of the game, the user is asked to select the order of theory of mind their opponents should use in the game with their corresponding slider. Three cognitive models were developed, which each used a different order of theory of mind: zero-order, first-order or second-order. Their implementation and functionalities will be discussed in the next section. When the user is done selecting their opponents, they can start the game by performing their action. Furthermore, the user can start a new game at any time by tapping the corresponding button in the top left corner.

[2]https://www.boardgamegeek.com/boardgame/12942/no-thanks

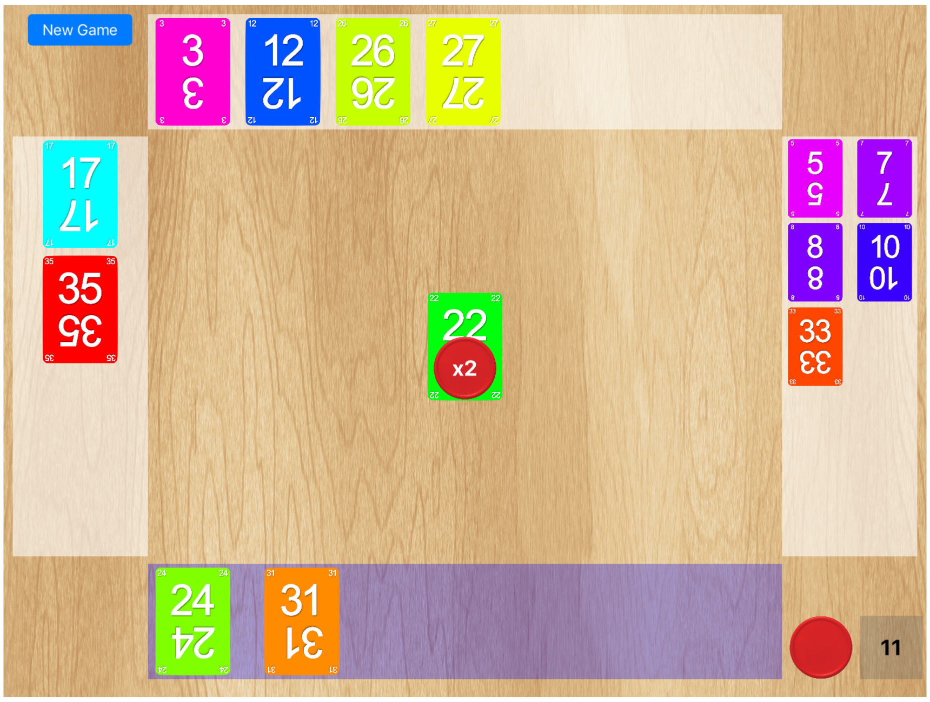

Figure 3: Screenshot of the app midway through the game

In order to pick up the card, the user can swipe down on the playing card. The playing card is then animated to the user's field to add the card to their collection. Any pass tokens on the playing card are also animated onto the user's token in the bottom right and added to their token supply. Alternatively, the user can swipe up from the pass token displayed in the bottom right to place a token on the playing card. This token is also animated onto the playing card.

Figure 3 shows a screenshot of the app midway through a game. The owned cards of all players are clearly displayed and shown in order. The current card at play is card 22 with 2 pass tokens on it.

3 Model implementation

Three different models were developed for this project, which each used a different order of theory of mind (ToM). A model was made using zero-order ToM, first-order ToM, and second-order ToM. The implementation of each model will be discussed in this section.

3.1 Zero-order theory of mind model

The zero-order theory of mind model (ToM_0) only looks at its own cards and its number of owned tokens. It does not take its opponents' cards into account nor the number of tokens that they own, as this would be a first-order theory of mind skill. The model does the following when it is its turn and it has to decide which action to take.

Firstly, the model interprets the current state of the game by looking at the card at play, the number of tokens that are on that card, its owned cards, and its number of owned tokens. Additionally, it checks which sequence category the card at play belongs to. These sequence categories are a measurement to assess how valuable a playing card is to a player given the cards they currently own (not taking the number of tokens on the playing card into account). There are three different categories:

No sequence

The card at play would not be part of any sequence if taken, thus meaning it is worth its face value in points (a lonely card).
Scenario:
Cards owned: 4,5 (value: 4 points)
Card 7 is taken
Cards owned: 4,5,7 (value: 11 points)

Sequence extension

The card at play would be part of a sequence if taken and is positioned either at the start or at the end of the sequence. This means the player would gain a point if the card belongs at the start of a sequence.
Scenario:
Cards owned: 4,5 (value: 4 points)
Card 3 is taken
Cards owned: 3,4,5 (value: 3 points)

Alternatively, the player can add a card with no gain if the card belongs at the end of sequence.
Scenario:
Cards owned: 4,5 (value: 4 points)
Card 6 is taken
Cards owned: 4,5,6 (value: 4 points)

Sequence completion

If taken, the card at play would complete a new sequence by adding a card between two cards. This category reflects the highest gain in points, as completing

a sequence means the value of the card(s) on the right side of the taken playing card are negated.
Scenario:
Cards owned: 3,5,6 (value: 8 points)
Card 4 is taken
Cards owned: 3,4,5,6 (value: 3 points)

Using this criteria, the model assigns a sequence category to the current game state and uses this information to decide which move to take using instance-based decision making in the next step.

3.2 Instance-based decision making

Instance-based decision making is at the core of the model's decision making process, meaning the model's declarative memory contains instance chunks which define its actions. The instances present in declarative memory are manually entered beforehand and are grouped by the aforementioned sequence category. All instance chunks contain the following slots: the sequence category, playing card value, number of tokens on the playing card, number of tokens owned, and the action to perform. Table 1 shows the slots present in an instance chunk and their range of values.

Slot	**Range**
Category	no sequence / sequence extension / sequence completion
Playing card value	3 to 35
Tokens on playing card	0 or higher
Tokens owned	0 or higher
Action to perform	take card / place token

Table 1: The slots present in an instance chunk and their range of values

The instances are designed from a zero-order theory of mind perspective. From this perspective, whenever the card at play belongs to either the sequence extension or sequence completion category, the model should take the card. This is because such a card can be added freely or even with a gain in points (i.e. a lower score). To achieve this behaviour, all instances belonging to these categories always have the *take card* action in their corresponding slot. In contrast, instances belonging to the no-sequence category are much more varied. These chunks are mainly structured around the playing card value. Generally, how higher the card value, the more tokens have to be on the card before the model decides to take it. However, when the model's token supply is low, less tokens are needed on the card before the model chooses to take it.

When the model has assessed the game state as described earlier, it creates a chunk of that current state. This current state chunk contains the same slots as the instances, excluding the action slot (because this is the thing the model needs to decide). *Partial matching* is then used to retrieve the instance chunk that is most similar to the current state chunk (i.e. has the highest activation = lowest mismatch penalty). The action present in the corresponding slot of the retrieved chunk is then executed by the model. Figure 4 shows a flowchart of the decision making process of the ToM_0 model.

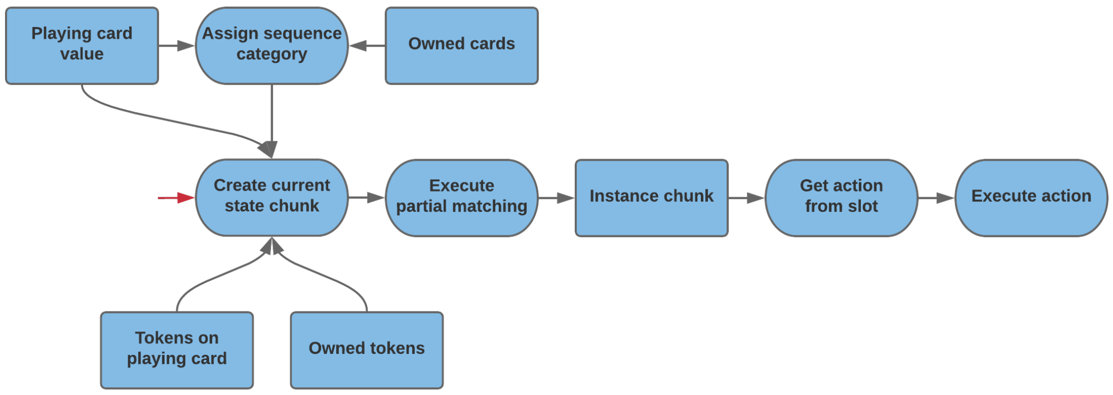

Figure 4: Flowchart of the decision making process of the ToM_0 model

3.3 Partial matching and its mismatch function

A mismatch function is used to determine which instance chunk is the most similar to the current state chunk that is given as the partial matching input. Because all instance chunks have a fixed activation of 0 (thus no decay), the retrieval activation of a chunk is entirely dependant on the mismatch penalty it receives. This means that the retrieval activation is always negative, except if the current state chunk exactly matches an instance chunk (disregarding activation noise).

Slot	Mismatch penalty
Category	-100
Playing card value	$-\lvert x-y \rvert$ / 32
Tokens on playing card	$-\lvert x-y \rvert$ / $max(x,y)$ *Weber's law*
Tokens owned	$-\lvert x-y \rvert$ / $max(x,y)$ *Weber's law*

Table 2: Mismatch penalty per slot as used in the mismatch function

Some aspects of the game state are more important than others. Defining this distinction is an important task of the mismatch function, because it specifies the proportional penalty that slots should receive relative to each other. Table 2 shows the mismatch penalty used for each slot. In the case of *No Thanks!*, the sequence category is the biggest factor in the decision making process. Therefore, the category slot has a large mismatch penalty, which in turn always ensures a chunk of the correct category is retrieved. This is also relevant for the token chunks used only by the higher order theory of mind models, which will be discussed later.

As shown in Table 2, a linear mismatch penalty is used for the playing card value slot by dividing the absolute difference of input x and y by the range of cards (i.e. 32). However, the tokens on playing card and owned token slots use a different penalty function based on *Weber's law*.

According to Weber's law, the minimum difference that a human can perceive between two stimuli is proportional to the size of those stimuli (Whalen et al., 1999). This means that numbers that are closer together are considered to be more similar, but larger pairs of numbers are considered more similar than smaller pairs (i.e. the former receive a smaller penalty). This approach is suitable for determining the mismatch penalty for the number of tokens on the playing card and the model's token supply. This is because a difference between a smaller number of tokens should have a higher mismatch penalty than that same difference between a higher amount. For example, owning 1 token instead of 2 should have a bigger impact on a player's strategy than the difference between owning 10 or 11 tokens.

3.4 First-order theory of mind model

Tokens play an essential part in *No Thanks!*, as a token is worth -1 point and is required for the ability to skip a bad card. Therefore, it is beneficial for the model to maximize the number of owned tokens. A good strategy for this is to assess how good or bad the current card at play is for the opponents and thus to predict how likely it is for an opponent to take the card or not. It can occur that a particular card is valuable to the model (because it would be part of a sequence in its collection of cards), while it is a bad card for its opponents (they cannot place it in a sequence). The model can then decide to place a token on that card, even though taking the card now would already be profitable. This way the same card could rotate back to the model with three extra tokens on it (one from each opponent, as a result of them skipping it).

This ability is a first-order theory of mind skill, as it involves reasoning about what the opponent would do from a zero-order theory of mind perspective. This means the model has to make a prediction based on the owned cards of its opponents and how many tokens they own. While the former is open knowledge (the cards are on display), the latter is not. However, this knowledge can be fully tracked, as every player can fully observe the flow of tokens in the game and the number of starting tokens is equal for all players. This skill involves counting and memorization, and is not an easy task for humans. In order to let the model memorize the token flow in a realistic human-like way, a token memory system was implemented. This system will be explained later in this section.

3.4.1 Predicting actions of opponents

When it is the turn of the model, it attempts to predict the next action of its opponents first. It does this by projecting the state of the opponent onto itself. In other words, it decides what it would do if it were in its opponent's position if it would apply zero-order theory of mind. Thus it executes the ToM_0-protocol with the opponent's current state plus an extra token on the playing card. This additional token will be on the card if the model decides to skip the card. Therefore this has to be taken into account.

This procedure continues until a prediction has been made for all opponents (note an extra token is added for each player). However, if it predicts an opponent will take the card, it will not make a prediction about the player(s) that is (are) next in turn. Because the model will not risk placing a token anyway. Next, the model executes the ToM_0-protocol on its own state. If its best ToM_0 action is to take the card, then it looks at the predictions. Only if all players are predicted to place a token, then the model decides to skip the card in the expectation that the card will come back again with extra tokens on it. Otherwise, the model takes the card (like a ToM_0 model would do). Note that the ToM_1 model still uses the same zero-order ToM instances as the ToM_0 model. Thus the fundamental aspect of the first-order ToM model is that it takes its opponent's cards and their number of owned tokens into account. Figure 5 shows a flowchart of the decision making process of the ToM_1 model.

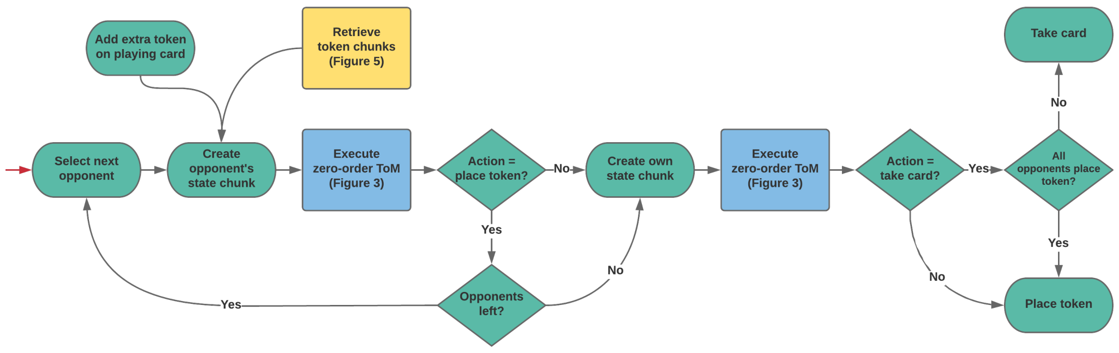
Add extra token on playing card
Retrieve token chunks (Figure 5)
Take card
No
Select next opponent
Create opponent's state chunk
Execute zero-order ToM (Figure 3)
Action = place token?
No
Create own state chunk
Execute zero-order ToM (Figure 3)
Action = take card?
Yes
All opponents place token?
Yes
No
Yes
No
Yes
Opponents left?
Place token

Figure 5: Flowchart of the decision making process of the ToM_1 model

3.4.2 Remembering the token supply of opponents

A human player with perfect memory could in theory derive the exact number of tokens for every player at any point in the game. However, a human player makes mistakes and so should the model. Additionally, the model would have an unfair advantage over human players if it simply logged the token distribution with no error. Instead, the model uses token chunks stored in declarative memory to remember the token distribution of the game. These chunks decay over time and thus it can happen that they cannot be retrieved and thus cannot be remembered by the model.

Slot	**Range**
Category	Leaf
Player	0-3
Leaf decay	*Constant value*
Linked root	Root that was created or referenced during leaf creation
Token number	0 or higher

Table 3: Slots and their content range of the token *leaf* chunk

Slot	**Range**
Category	Root
Player	0-3
Token class	Low / Sufficient / Many

Table 4: Slots and their content range of the token *root* chunk

]

A frequent occurrence for human players is that they are unable to remember the exact number of tokens, yet are able to remember an approximation. To model this behaviour, two types of token chunks were implemented: token leaf chunks and token root chunks. Table 3 & 4 show the slots and their content range of the token leaf chunk and token root chunk respectively. The model stores both a root and a leaf chunk for each opponent. The leaf chunk stores an exact number of tokens, while the root chunk holds a categorical amount (i.e. token class). The leaf chunk is harder to retrieve than a root chunk, because it receives an additional mismatch penalty (thus decreasing its activation). This models the behaviour that a human remembers an approximation easier than an exact number.

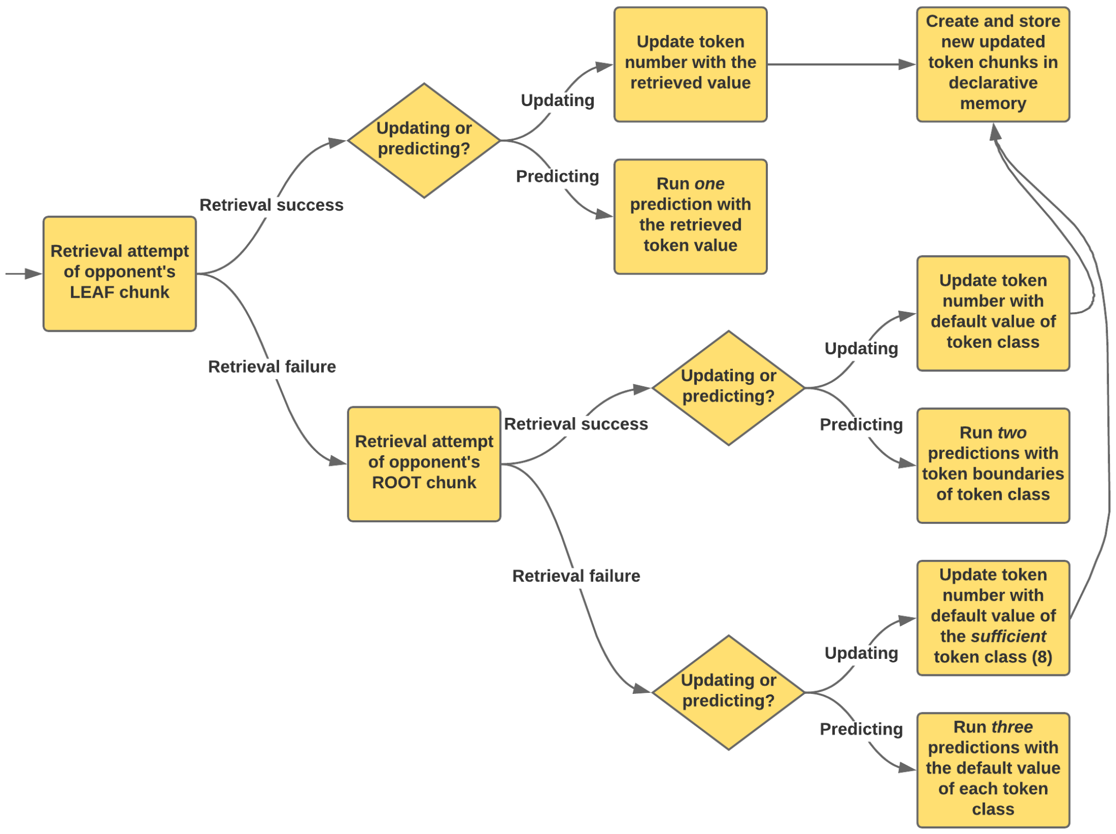
Retrieval attempt of opponent's LEAF chunk
Retrieval success
Updating or predicting?
Updating
Update token number with the retrieved value
Create and store new updated token chunks in declarative memory
Predicting
Run *one* prediction with the retrieved token value
Retrieval failure
Retrieval attempt of opponent's ROOT chunk
Retrieval success
Updating or predicting?
Updating
Update token number with default value of token class
Predicting
Run *two* predictions with token boundaries of token class
Retrieval failure
Updating or predicting?
Updating
Update token number with default value of the *sufficient* token class (8)
Predicting
Run *three* predictions with the default value of each token class

Figure 6: Flowchart of the implementation of the token memory system in updating the token chunks and determining which token value(s) to use for its prediction(s).

Figure 6 shows the flowchart of the implementation of the token memory system. When the model tries to remember the number of tokens of an opponent, it first attempts to retrieve the opponent's leaf chunk using partial matching. If this retrieval is successful, it uses the retrieved number of tokens for its prediction. However, if the retrieval fails, it attempts to retrieve the root chunk (which is easier to retrieve). If this does succeed, the model executes two predictions with the prediction values of the retrieved class (Table 5). In the event the root chunk can also not be retrieved, the model runs a prediction which the default token value of each token class as shown in Table 5.

Token class	**Range**	**Prediction values**	**Default value**
Low	0-5	2, 5	3
Sufficient	6-10	6, 10	8
Many	11 and up	11, 16	11
Default (no retrieval)	N/A	3, 8, 11	8

Table 5: Properties of each token class.

By using declarative memory, whether a token chunk can be retrieved depends on the decay. While the base activation of a token chunk is also dependant on noise, it is primarily determined by the model time that has passed. Thus this means how longer it takes before a particular chunk needs to be retrieved, how harder it is to retrieve that chunk. In *No Thanks!* this translates to how many turns have passed, because a constant amount of model time is added for every turn. The minimum number of turns is 3 (if all opponents skip the card), but can be more if a player decides to take a card. Therefore the chance of a retrieval failure increases if players take cards during a round (and thus take more time).

3.4.3 Keeping track of tokens

Whereas the ToM_0 model only attends to the game during its turn, the ToM_1 model also needs to perform when it is the turn of others. In order to keep track of tokens, it has to observe and remember whether an opponent either receives tokens (by taking the card) or loses a token (by skipping the card). For it to update the number of owned tokens, it first has to remember the relevant token chunks. The model uses the same partial retrieval system as previously explained (Figure 6). If it can retrieve a leaf, it will create a new leaf chunk with the updated number of tokens and store it in declarative memory. Additionally, it will create and store a root chunk with the corresponding token class. If it can only retrieve the root chunk, it will add or subtract the change in tokens to the default value of that class (shown in Table 5). A new leaf and root chunk are then created and stored in declarative memory with the new token number and its associated class respectively. In the case retrieving both the leaf and the root chunk result in a retrieval failure, the model assumes the player had 8 tokens (the default value of the *sufficient* token class) and creates the new chunks using that value.

Note that the ToM_1 model executes this process during every opponent's turn and that each opponent has its own set of token chunks. The mismatch penalty for the player slot is large, which means the model does not mix up the tokens supllies between players and does not incorrectly retrieve a token chunk belonging to the wrong opponent. This possible confusion was not added to the model, because it was found to add unnecessary complexity to the model.

Additionally, a result of this token system is that the number of token chunks in declarative memory increases as the game progresses (because a new chunk is created with every update). However, because all chunks decay over time, the model retrieves the most recent chunk (unless in the rare case activation noise causes an older chunk to have a higher activation).

3.5 Second-order theory of mind model

The second-order theory of mind model makes use of the same tools available to the ToM_1 model. It also uses the ToM_0 instances, the same token memory system and it makes predictions about its opponents' move. However, whereas the ToM_1 model assumes its opponents reason with zero-order ToM, the ToM_2 model assumes they possess ToM_1 reasoning.

When it is the turn of the model, it makes a prediction about the player next in turn. It projects its own ToM_1-protocol on the current state of that player, plus an additional token. Just like the ToM_1 model, it makes a prediction for all its opponents and only places a token on a favourable card if it predicts that all opponents will skip that card. Note that the model still only uses its own declarative memory for remembering the token chunks. It does not attempt to make a prediction about how well its opponents remember the token flow of the game. The reason for this is that there is no reliable way to make a prognosis about how well an opponent may remember the same thing the model itself attempts to remember. Additionally, it would add redundant complexity to the model. Therefore, the main difference with the ToM_1 model is that it projects the ToM_1 protocol onto its opponents, instead of the ToM_0 protocol.

4 Code implementation

This section will present an overview of the code implementation and an explanation about a number of features that were not discussed in the previous section (Model Implementation). Additionally, the most important properties (instance variables) will be discussed.

4.1 Structure

The entire application has been coded using Swift 4.0 [3] and is designed exclusively for the iPad. The app is targeted to run on iOS version 10.3 and higher. The app has been designed following the Model-View-Controller (MVC) design pattern as recommended by Apple [4]. The application consists of the following Swift files, which each represent a class.

Models

- NoThanks.swift: Primary core model class responsible for running the game of *No Thanks!*. The game is initialized and run by this class, e.g. it shuffles the cards, initializes the players, communicates the game state to the cognitive models, logs the game etc.
- PlayerModel.swift: Core model class responsible for running the cognitive model used as a player in *No Thanks!* and thus defines the behaviour of the model. Inherits from Model, which originates from the ACT-R Core files. All theory of mind and token memory system functions are present in this class.
- Log.swift: Class responsible for logging game progress and model behaviour for debugging and experimental data analysis. 39 variables are logged and written to a text file by this class.
- Player.swift: Class which defines the properties of the players in the game. It stores which cards a player owns, how many tokens they own, what their current score is, and what model defines its behaviour.
- Card.swift: Class which defines the properties of a playing card in the game.

Controllers

- ViewController.swift: Primary view controller class responsible for everything UI related in the app. It manages animations, gestures, displaying scores etc. Communicates with an instance of NoThanks.

[3]https://developer.apple.com/documentation/swift

[4]https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html

Views

- CardView.swift: UIView class to depict a card in the UI.
- CardViewCell.swift: UICollectionViewCell class to depict a card exclusively in the UICollectionViews used to depict the players' owned cards in the UI.
- TokenView.swift: UIView class to depict a token in the UI

ACT-R Core Swift files

A collection of Swift files which form a partial implementation of ACT-R, created by Niels Taatgen [5]. PlayerModel inherits from the Model class present in these files. This allows the cognitive model to use ACT-R functionalities such as declarative memory, chunks and partial matching.

4.2 NoThanks

Key properties

This class has multiple Boolean properties that can be set to change the behaviour of the game. Setting modelOnly to true lets the app play the game with only models and no user involved. The fastMode variable sets the duration of animations to near 0 time. This was enabled when running model versus model simulations to significantly decrease game duration. Lastly, when the debuggingMode is enabled, a more detailed model trace is printed in the Xcode console. This was very valuable in debugging, because it gives a detailed view of the model's decision process. Additionally, a new debugging button is also displayed in the UI. Pressing this button allows the model that is next in turn to execute its action. The game will not proceed to the next turn automatically, allowing the user to debug without time constraints.

The user can also setup an experiment with multiple games from this class. The property array modelsToMSchedule holds the distribution of which order of theory of mind each model will use for each game. This can be altered to the user's needs. This class is reinitialized to start a new game.

Key functions

The most important task of this class is to control the game. At initialization, it shuffles the cards, gives out the tokens and assigns an instance of the PlayerModel class to each of the three players (or four in case of model only gameplay). During the game it asks the linked model of each player (or human player) which action it (they) will take when it is their turn. It then communicates to the ViewController what to display on screen.

This class is also responsible to inform the game state to each model (i.e. each PlayerModel instance) at every turn so the cognitive models can assess the current game state to decide on their action or to update their token memory system (this is done by the giveGameStateToModels function).

[5] https://github.com/ntaatgen/ACT-R.git

Additionally, this class also passes logging data to an instance of the Log class at every turn. The data is written to a dictionary which is a property of that class. This Log instance is passed as an initialization parameter of the NoThanks class when it is reinitialized for a new game. This is done so the data of multiple games can be written to the same log.

4.3 PlayerModel

Key properties

This class has multiple Boolean properties that are primarily intended for debugging. The first- and second order ToM models use their token memory system when useTokenMemory is set to true. When set to false, the model 'cheats' by looking directly at the correct number of tokens its opponents own. This was used for debugging and testing the model. Furthermore, setting the tokenDebugging and instancesDebugging properties to true results in a more detailed model trace in the console about the token memory system and instance-based decision making respectively.

The model's main ACT-R parameters can also be set here. These are the retrieval threshold, baselevel decay, activation noise, and the mismatch penalty. Many simulations and tests were necessary to set these parameter values, due to the fact that these parameters have a strong influence on both instance-based decision making and on the token memory system at the same time. For example, increasing the activation noise leads to more variation in the partial matching process of retrieving an instance. This also increases the variation of whether the model can retrieve a leaf chunk, only a root chunk, or neither. Additionally, more activation noise increases the chance that the model does not retrieve the most recent token chunk, as noise can cause a more decayed chunk to get a higher activation value than a less decayed chunk (i.e. the more recent chunk). Both factors result in the model making more mistakes in remembering token values. This in turn results again in more variation in instance-based decision making, as the current state chunk used for partial matching has more varied token values in it.

Therefore, due to the compounded effect of parameters and the interaction effects between them, it was challenging to find parameters that lead to the most realistic model behaviour. It was attempted to balance these parameters in such a way that the model does make mistakes and has variation in its decision making process, while also not underutilizing the process by adding too much randomness.

Key functions

The PlayerModel is responsible for the behaviour of the model. This means that all theory of mind reasoning and the token memory system are defined by this class. In other words, everything discussed in the Model Implementation section is executed by this class. However, the class does inherit from the Model class of the ACT-R core files. It depends on this superclass, because it makes use of ACT-R functionalities such as chunks, declarative memory and partial matching. These are defined in these files.

At every turn, the model updates its own game state according to the information given by the NoThanks class. It then continues performing the steps as explained in the Model Implementation section. Note that a higher order theory of mind model not only uses the respective executeToM function, but also the function(s) of the lower order(s) of theory of mind. For example, the ToM_2 model starts by executing the executeSecondOrderToM function. Then it makes predictions of its opponents by using the executeFirstOrderToM function three times, which each call the executeZeroOrderToM function three times to make predictions from that opponent's perspective. Lastly, it makes a decision based on its own cards and own tokens collection by calling executeZeroOrderToM one final time.

4.4 ViewController

The ViewController controls everything that is UI related. It is responsible for animations, recognizing gestures and thus presenting the game to the player.

An UICollectionView [6] was used to display the cards of players on the screen. This type was found to be very suitable for presenting this data, because an UICollectionView provides its own internal animations when adding, deleting or moving objects of its data source. This means that when adding a new card to a player's card collection to, for example, somewhere in the middle, all cards to the right side of that new card are automatically animated to the right to make space for that new card. This allowed for a smooth and clear presentation to the user.

Many functions were needed to smoothly move a playing card into a player's card collection. This procedure is done by creating an identical transition card exactly above the playing card immediately when the card is taken. The original playing card view is hidden, and the new card is added to the player's corresponding UICollectionView as a hidden object. The transition card (which is a CardView) is animated to the exact position of the (still hidden) card view in the UICollectionView. Then the hidden card view is made visible and the transition card is deleted simultaneously. This procedure results in the "illusion" that the same playing card is being moved to a player. It is also dynamic, which means that elements of the UI can be moved without breaking any of the animations. The same process with a transition token is used for moving tokens to a player and on a playing card .

Additionally, the capabilities of the UICollectionView were significantly modified to let cards dynamically shrink in size in case a player has too many cards to fit on screen in full size. The cards also form rows to optimally make use the available screen space.

[6]https://developer.apple.com/documentation/uikit/uicollectionview

4.5 Additional notes

- The instances used by the cognitive model are present in the Instances_ZeroToM_Final.actr file. These instances are imported by the PlayerModel class.
- An open-source library was used to create the controls shown at the start of the game. The user can use these to select which order of theory of mind its opponent should use during the game (see Figure 2). These were developed by Tapptitude and are available on Github [7].
- The log file which records the game progress is saved locally. When running the app on a Mac using the simulator, the directory path is shown in the Xcode console. When using an iPad, the file is located in the app's file container. This container can be downloaded from the iPad when connecting it to Xcode.

[7] https://github.com/tapptitude/TTSegmentedControl

5 Experimental results

5.1 Experiment with human subjects

An experiment with human subjects was conducted. The goal of this experiment was to study the performance of the developed cognitive models against human players. Its results would give insight into how suitable a model using instance-based decision making is in a strategic turn-based game such as *No Thanks!*. Additionally, the experiment was also meant to examine the effects of theory of mind and to study whether human players perform differently against models using different orders of theory of mind.

5.1.1 Methods

30 people participated in this experiment (20 female). Each were given €12 for their participation. All participants were international students at the University of Groningen.

Before the experiment started, instruction sheets were handed to the participants who were then given sufficient time to read it through. This instruction sheet consisted of a detailed explanation of the rules of *No Thanks!*, an explanation of the user interface of the app, and additional information about the proceedings of the experiment (Appendix A). After each participant was finished reading the instructions, the supervisor read the instructions out loud and the participants were asked to read along. This was done to reduce the chance of a participant not understanding the rules. Each subject was allowed to ask questions at any time. Once the instructions were clear, the participants moved to their own separate room where they would play *No Thanks!* on an iPad Mini 4 (2015) running iOS 11.3.

Block	1			2			3			4			5			6			7			8		
Game	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Models' order of ToM	0	1	2	0	0	0	1	1	1	2	2	2	0	0	0	1	1	1	2	2	2	0	1	2

Table 6: Overview of which model type served as the subjects' opponents during the 24 games of the experiment

All participants played against the three different ToM models for a total of 24 games; 8 games against each model. Subjects took 72 minutes on average to complete the experiment. Table 6 shows which type of model served as the subject's opponent for each game. All three opponents were always the same type of model (thus used the same order of theory of mind), meaning different models types never competed against each other during this experiment. This was done to prevent any additional interaction effect that the different model types may have on each other, as the goal of this experiment was to study whether a human subject performs differently against different order theory of mind models in *No Thanks!*. An additional experiment was conducted in which simulations were run to study the model versus model behaviour. This experiment will be discussed in the next section.

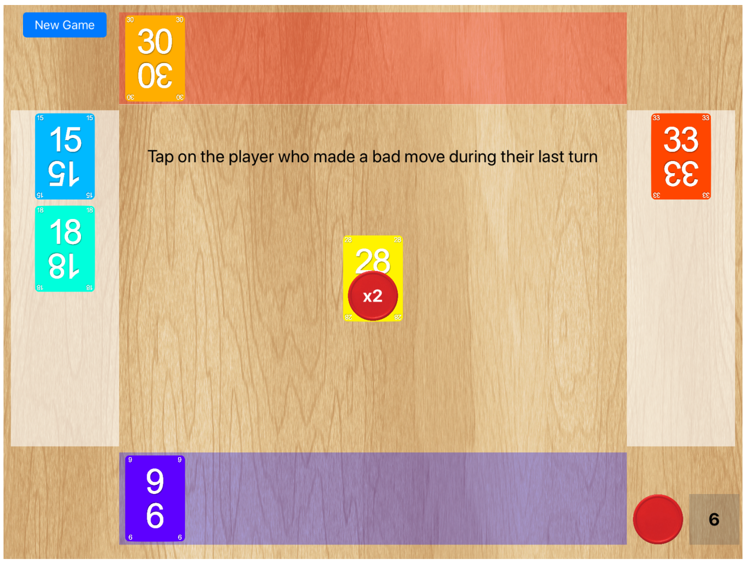
New Game
30
15
18
33
Tap on the player who made a bad move during their last turn
28
x2
9
6

Figure 7: Screenshot of the game when the subject is asked to provide feedback. The opponent at the top of the screen (player 2) is selected (and thus marked red).

Additionally, participants were asked to give feedback about the models' actions. During each of their turns, participants were asked whether any of their opponents made a bad move during their last turn (in their opinion). This message was displayed on screen. If they found that was the case, the subject could tap on the concerning opponent(s). This marked the opponent(s) red on screen (see Figure 7). The subjects could then continue with the game. It was noted to the subjects that it was not required to give feedback during every turn, only if they found that any opponent made a bad move.

5.1.2 Results

The data of all 30 subjects were used. However, the first three games of the experiment (block 1) were meant as practice games and its results were therefore omitted in the data analysis. This means the following results are based on 21 games in which each subject played 7 games against each model.

Figure 8 shows the proportional ranking of the human subjects against each model. Note that a human player always played against three models at the same time. This means that if a participant won 25% of the games against a particular model, its proportional winning performance would be equal to that of the model (because the three models won the remaining 75% of the games; 25% by each player). Therefore, a player ranking above 25% indicates it scored that ranking more often than the model.

In other words, Figure 8 shows the distribution of where the human subjects finished at the end of each game against each model type. This means these plotted frequencies add up to 100% for each model type. For example, the participants became first in 17.7% of the games when playing against the ToM_1 model. Whereas they became second in 20.5% of the games, and ended up in third and fourth place in 23.3% and 38.5% of the games respectively (summing to 100%).

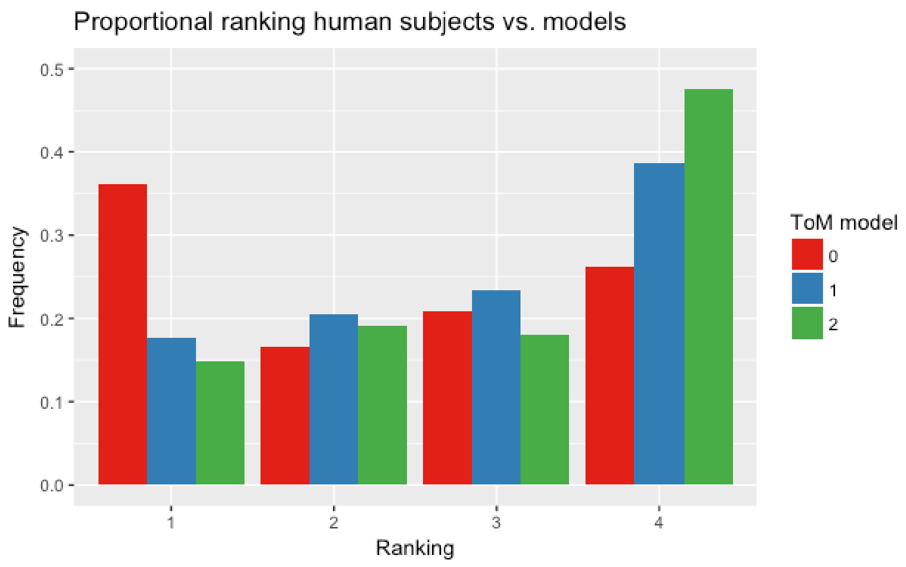

Figure 8: Proportional ranking of the human subjects versus each model over 21 games

Figure 8 shows that the participants won more often against the ToM_0 model than versus the ToM_1 and ToM_2 model. The subjects also became second and last less often than versus the other models. Additionally, subjects won 36.2% of the games against the ToM_0 model. This means they won more often from the ToM_0 model than vice versa (because this occurred more often than 25% of the games).

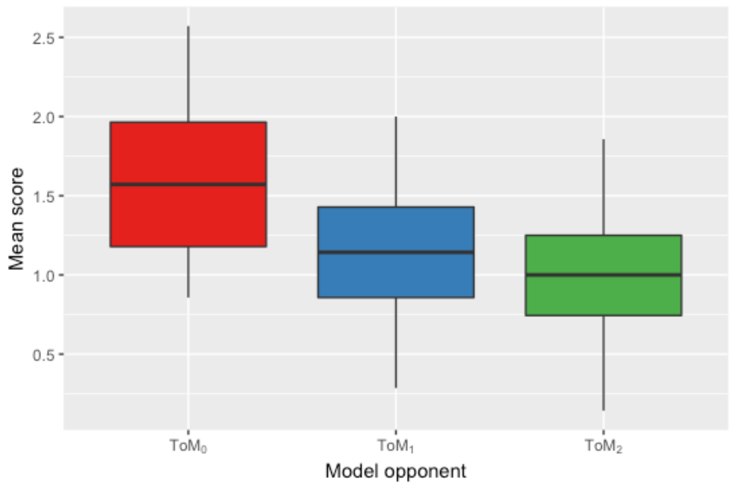

Figure 9: Boxplots showing the distribution of the mean score of the human subjects against each model

To test the significance of the performance difference between models, subjects were given a score based on their ranking. 3 points were assigned for winning, 2 for becoming second, 1 for becoming third, and no points for becoming last. For each subject, the mean scored points against each model was calculated, thus resulting in three mean scores per subject. The distribution of these mean scores per model opponent are shown by the boxplots in Figure 9. Similar to the results of Figure 8, it can be seen that subjects on average score worse when the model's order of theory of mind becomes higher.

A paired t-test was run between the subjects' mean scores against the ToM_0 and ToM_1 model. The result of this shows that this difference in performance is significant ($t(29) = 4.63, p < 0.005$). The same procedure shows that the performance difference between the ToM_0 and ToM_2 model is also significant ($t(29) = 5.28, p < 0.005$).

Additionally, Figure 8 and 9 show that the participants performed slightly better against the ToM_1 model than against the ToM_2 model. This is because the subjects became first, second and third marginally more often versus the ToM_1 model, whereas they became fourth considerably more often against the ToM_2 model. However, the overall performance difference between these models appeared to be not significant when assigning a mean score to each participant ($t(29) = 1.55, p = 0.16$). Also, it can be derived from Figure 8 that subjects won less often from both the ToM_1 and ToM_2 model than vice versa (17.7% and 14.8% respectively). This being in contrast with the subjects' performance against the ToM_0 model (subjects winning 36.2%).

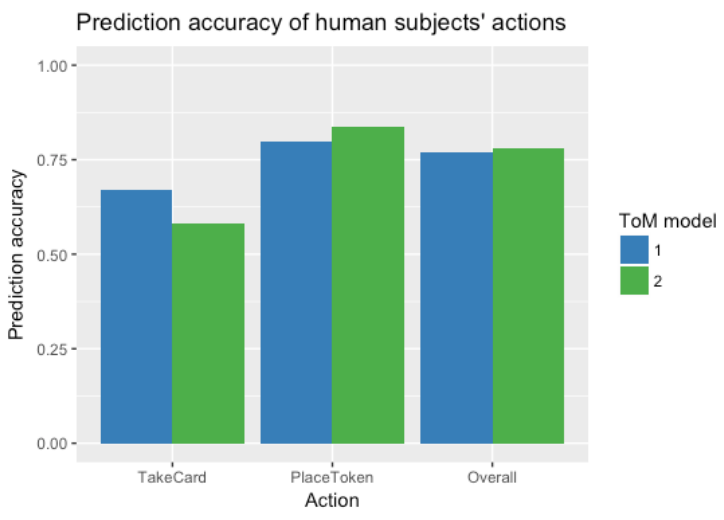

Figure 10: Prediction accuracy of the ToM_1 and ToM_2 model of the human subjects' actions

Figure 10 shows the prediction accuracy of the ToM_1 and ToM_2 model of the human subjects' actions. Because the ToM_0 model does not make predictions about its opponent's actions, it is not present in this graph. It can be seen that that both models have almost the same prediction accuracy overall (77.0% and

77.9% for ToM_1 and ToM_2 respectively). This is significantly above chance. Both models predicted more accurately when a subject would place a token than they would take a card. However, the distributions between the two models are different. Whether a subject would take a card was correctly predicted in 67.1% of the cases by the ToM_1 model, compared to 58.0% by the ToM_2 model. Whereas the ToM_2 model beats out the ToM_1 model in predicting more accurately when a participant will place a token (83.5% over 79.8%). Despite this smaller margin, the overall prediction accuracy of both models are almost identical. This is because players place a token significantly more often than they take a card (approximately 80% of actions on average).

Lastly, 1.18% of the actions made by the ToM_0 model were regarded as a bad move by the participants. This was 1.36% and 1.20% for the ToM_1 and ToM_2 model respectively. This might suggest that the ToM_1 model received marginally more negative feedback than the ToM_0 model, even though the former performed significantly better against the participants. However, the differences are very small thus there is too much margin of error to draw conclusions from these results. Additionally, further analysis shows that there was a large variation between subjects. 10% of the participants were responsible for 38% of the given feedback, while 43% of the participants gave feedback less than 10 times (on ±1800 model turns). This undermines the reliability of these results even further.

5.2 Experiment by simulations

In addition to conducting an experiment with human subjects, an experiment through simulations was also conducted. The goal of this experiment was to further examine the effects of theory of mind on playing *No Thanks!*. Instead of playing against human players, the different models played against each other in order study which order of theory of mind has an advantage over the other. It was hypothesized its results would give further explanation to the results found in the experiment against human subjects.

Simulation	**Player 0**	**Player 1**	**Player 2**	**Player 3**
ToM_0 vs. ToM_1	ToM_0	ToM_0	ToM_0	ToM_1
ToM_0 vs. ToM_2	ToM_0	ToM_0	ToM_0	ToM_2
ToM_1 vs. ToM_2	ToM_1	ToM_1	ToM_1	ToM_2

Table 7: Experimental setup of the run simulations for this experiment

5.2.1 Methods

Three simulations were run in which each model type played against each other. A similar setup to the experiment with human players was used for these simulations, with one model type taking the role of the human subject. This means that model type A played against three models of type B during each game. The experimental setup is shown in Table 7. The models played 500 games against each other in every simulation. This was done to mitigate the effect of game randomness, which *No Thanks!* possesses, as much as possible. Additionally, the models used in this experiment were identical to the ones in the human experiment (using the same parameters etc.).

5.2.2 Results

Figure 11 and 12 show the proportional ranking of the ToM_0 model versus the ToM_1 and ToM_2 model respectively. Analogous to the results of the human experiment, a model ranking above 25% indicates it scored that ranking more often than the opponent model. Firstly, it can be seen between both figures that the results are very similar. Both the ToM_1 and ToM_2 model perform considerably better than the ToM_0 model by winning over 48% of games and becoming second, third or last fewer times. These performance differences were both found to be significant when assigning points to each ranking: $t(909.8) = -15.8, p < 0.005$ and $t(854.3) = -14.2, p < 0.005$ for the ToM_1 and ToM_2 model respectively.

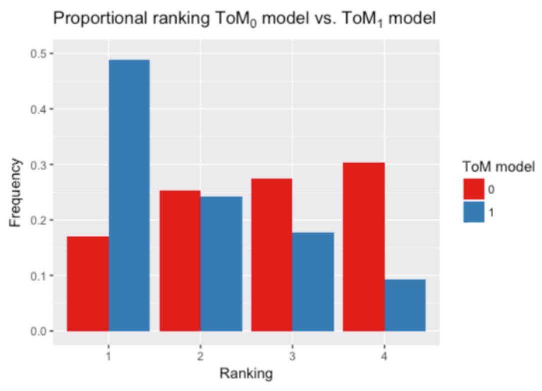

Figure 11: Proportional ranking ToM_0 versus ToM_1 model over 500 games

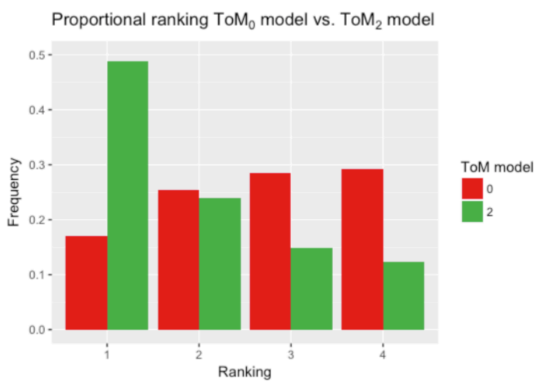

Figure 12: Proportional ranking ToM_0 versus ToM_2 model over 500 games

]

Figure 13 show the proportional ranking of the ToM_1 model versus the ToM_2 model. It shows that the ToM_2 model performs slightly better than the ToM_1 model, as it ranks first and second more often. Although the performance difference between these two models is not as substantial as comparing them to the ToM_0 model, it does appear to be significant ($t(865.0) = -2.0, p = 0.04$).

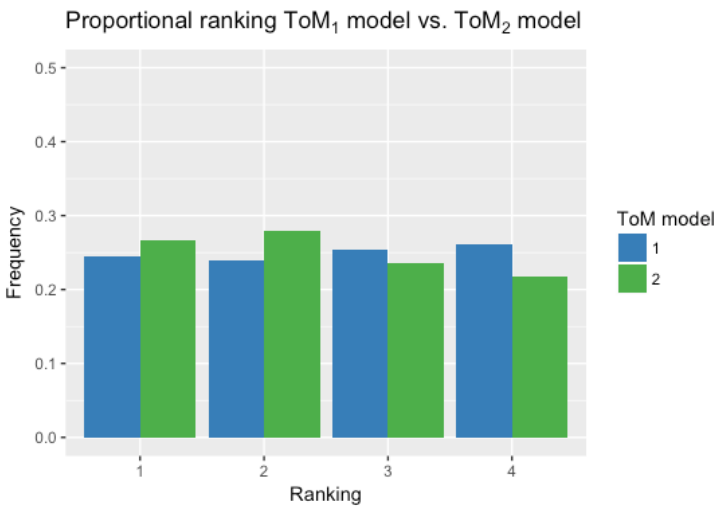

Figure 13: Proportional ranking ToM_1 versus ToM_2 model over 500 games

5.3 Effects of the token memory system

Both the ToM_1 and ToM_2 model use the token memory system to remember how many tokens their opponents own. Whether the correct number of tokens can be remembered depends largely on whether the token leaf and token root can be retrieved. A token chunk can be retrieved if its activation is above the retrieval threshold (RT). Decreasing the retrieval threshold thus means that a token chunk can be retrieved more easily and vice versa. Therefore, by adjusting the retrieval threshold, the effects of correctly remembering the token supply of opponents can be examined indirectly.

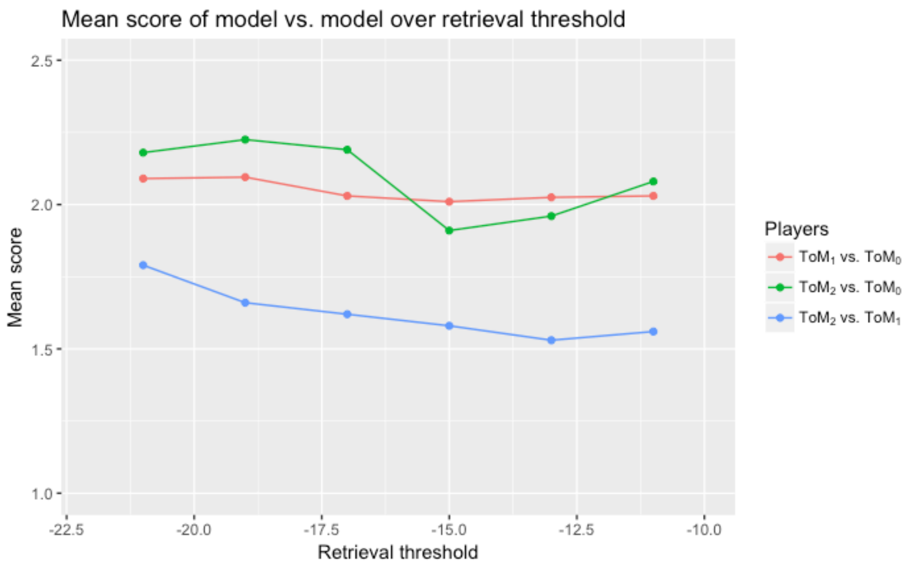

Figure 14: Mean score of both the ToM_1 and ToM_2 model playing against the ToM_0 model and one another over the retrieval threshold (RT)

Figure 14 shows the mean score of both the ToM_1 and ToM_2 model playing against the ToM_0 model and one another over the retrieval threshold. Like the previous analyses, these mean scores are obtained by assigning points to the models' rankings. The mean score of the model using the highest order of theory of mind is plotted (given in bold: $\boldsymbol{ToM_1}$ vs. ToM_0, $\boldsymbol{ToM_2}$ vs. ToM_0 and $\boldsymbol{ToM_2}$ vs. ToM_1).

It can be seen that the mean score of the ToM_1 model playing against the ToM_0 model (i.e. ToM_{10}) is only slightly affected by the retrieval threshold (RT) . There appears to be a downward trend when increasing the RT, but this may be due to the margin of error. In contrast, the mean score of ToM_{20} is decreasing until a RT of -15, but increases again afterwards. The mean score of ToM_{21} shows the most evident downward trend. All shown mean scores of the higher order theory of mind models were significantly higher than the respective mean scores of their opponents, with the exception of the ToM_{21} mean scores with a RT of -13 and -11.

Overall, the mean score appears to decrease when the RT is increased. The larger effect on the ToM_2 model may be caused by the fact that this model uses the token memory system more often than the ToM_1 model. This is because it assumes its opponents use ToM_1 and thus attempts to retrieve more token chunks. Therefore, this suggests that a higher RT causes more errors in retrieving token chunks, which leads to worse predictions, which in turn results in a lower mean score. This may also explain the smaller impact of the RT on the ToM_1 model, as it does not rely as much on retrieving tokens.

However, the overall effect of the retrieval threshold was smaller than expected. Given the impact of the number of owned tokens on which instance is retrieved (due to its high mismatch penalty), a larger effect on both the ToM_1 and ToM_2 model was anticipated. This may be due to the design of the instances and that all models use these same instances. The given instances avoid having a low token supply, as that can have a heavy negative influence in *No Thanks!*. The default number of tokens that the model assumes when it cannot retrieve any token chunks is 8. The results from both experiments show that models on average own approximately 10 tokens. Therefore, guessing that another model has 8 tokens is a pretty safe bet, which in turn may be the cause that the model's performance does not diminish as much as expected. The effect might be larger when playing against human players or models which use a different set of (more risky) instances. However, more research is needed to confirm this possible explanation.

6 Discussion

6.1 Implications of the results

This project attempted to study the effects of theory of mind on playing the game of *No Thanks!* using multiple cognitive models. The results of the experiment with human subjects showed that the cognitive models using first-order and second-order theory of mind performed significantly better against human players than the model using zero-order theory of mind. This suggests that making use of theory of mind in *No Thanks!* has an advantage over using no theory of mind. These results correspond with the findings of the simulation experiments, where both the ToM_1 and ToM_2 model significantly outperform the ToM_0 model. However, the advantage of using second-order theory of mind over first-order theory of mind appears to be not as substantial. Its performance against participants showed to be marginally better, although this was found to be insignificant ($p = 0.16$). The difference in the simulations was still unsubstantial, but larger and also (just) significant ($p = 0.04$).

Overall, these results suggest that using higher orders of theory of mind are an advantage in a competitive turn-based game such as *No Thanks!*. However, the benefits appear to be diminishing. This partly complies with the findings of De Weerd et al. (2013), as that study observed diminishing returns on using orders of theory of mind beyond the second. The key difference being here is that this study observed these diminishing returns earlier; beyond first-order theory of mind. However, this study did not consider models of third-order theory of mind and higher, thus no conclusions can be made whether the additional advantage diminishes even further beyond second order theory of mind.

A possible explanation for why these diminishing returns occurred earlier, is that these may be caused by the limited action space that *No Thanks!* possesses. Because there are only two actions a player can do, there is a limited amount of influence a model's strategy can have. However, De Weerd et al. (2013) also found that their observed diminishing returns were not related to the available action space. Nevertheless, the rock-paper-scissors variants studied by De Weerd et al. (2013) had at least three possible actions. Therefore, it is possible that for games that have a smaller action space than three, like *No Thanks!*, the advantage of using higher orders of theory of mind diminishes earlier. However, future research is needed to verify this possible explanation.

The cause for the observed diminishing returns may also be specific to *No Thanks!*. The key advantage that the ToM_2 model has over the ToM_1 model, is that the former can predict when the latter decides to skip a favourable card in order to collect extra tokens. Although the results show that this 'skipping strategy' leads to significantly better performance, the ToM_1 does not have the opportunity to execute this strategy often (because such a game state is relatively rare). Therefore there is also a limit to how often the ToM_2 model can exploit the ToM_1 model. This may be the cause for the marginal advantage that second-order theory of mind has over first-order theory of mind in playing *No Thanks!*.

Additionally, the prediction accuracy results shown in Figure 10 suggest that the subjects overall did not consistently use either zero-order theory of mind or first-order theory of mind, as the overall prediction accuracy was almost identical between the ToM_1 and ToM_2 model. The lower prediction accuracy of the ToM_2 model of taking a card might indicate that the average participant did not make use of the 'skipping strategy'. The ToM_1 model assumes the subject would take a favourable card immediately (i.e. the ToM_0 strategy), whereas the ToM_2 model assumes the subject would skip it to collect more tokens, if possible (i.e. the ToM_1 strategy). Therefore the higher accuracy of the ToM_1 may be caused by the participants on average using a zero-order theory of mind approach.

Furthermore, this project has demonstrated that an academic AI method like instance-based decision making is suitable for creating a competitive game AI for a competitive game. By using instance-based decision making, competitive cognitive models were developed without relying on traditional game AI methods such as finite-state machines or behaviour trees. An added benefit of this approach is that is allows a developer to change the behaviour of the model by simply changing the set of instances, without having to modify the architecture.

The experiment with the human participants showed that especially the ToM_1 and ToM_2 models are a challenging, but not unbeatable opponent. This is an important balance, as a too easy opponent is not fun to play against for the player, while an opponent that is too hard to beat is not entertaining either. Also, because of the performance difference between the different orders of theory of mind, the user has the ability to select the opponent that matches their own skill as much as possible. This freedom may lead to a better game experience overall. However, it was not studied how much the participants enjoyed the game or how they would rate their opponents, as this was not the primary goal of this project. Future research is needed to study whether the developed cognitive models are entertaining and challenging opponents to play against in *No Thanks!*.

6.2 Future research

As previously mentioned, it would be interesting to study the performance of models using higher orders of theory of mind than second in playing *No Thanks!*. This would provide more insight to whether the advantage over using lower orders of theory of mind diminishes even further as observed by De Weerd et al. (2013).

Furthermore, for possible follow-up research, the developed models could be expanded to make use of instance-based learning. The current model has a fixed set of instances that define its core behaviour. It does not learn from its opponents (or itself) by dynamically changing its instances. It would be interesting to develop an instance-based learning model that adapts its strategy to its opponents by attempting to analyse which actions lead to a successful outcome and create new instances accordingly. This may lead to a better and more competitive model.

Additionally, it would be interesting to study whether the design of this project's cognitive model could be used as an opponent for other games. The combination of using instance-based decision making and a (token) memory system may be fitting for other games in which memory plays an supporting role. In card games such as Hearts and Belote for example, it is important to keep track of which cards have been played and which cards are still part of the game. A memory system similar to this project's token memory system may be suitable to model this behaviour.

References

Anderson, J. R. (2009). *How can the human mind occur in the physical universe?* Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., and Qin, Y. (2004). An integrated theory of the mind. *Psychological review*, 111(4):1036.

Buckland, M. (2005). Goal-driven agent behavior. *Programming Game AI by Example*, pages 379–414.

Champandard, A. J. (2003). *AI game development: Synthetic creatures with learning and reactive behaviors*. New Riders.

Dawe, M., Gargolinski, S., Dicken, L., Humphreys, T., and Mark, D. (2013). Behavior selection algorithms. *Game AI Pro: Collected Wisdom of Game AI Professionals*, page 47.

De Weerd, H., Verbrugge, R., and Verheij, B. (2013). How much does it help to know what she knows you know? an agent-based simulation study. *Artificial Intelligence*, 199:67–92.

Dill, K., Pursel, E. R., Garrity, P., Fragomeni, G., and Quantico, V. (2012). Design patterns for the configuration of utility-based ai. In *Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC)*, number 12146, pages 1–12.

Feigenbaum, E., McCorduck, P., and Nii, H. P. (1988). *The rise of the expert company*, volume 240. Times Books New York.

Ghory, I. (2004). Reinforcement learning in board games. *Department of Computer Science, University of Bristol, Tech. Rep*, page 105.

Gonzalez, C. and Lebiere, C. (2005). Instance-based cognitive models of decision-making.

Gonzalez, C., Lerch, J. F., and Lebiere, C. (2003). Instance-based learning in dynamic decision making. *Cognitive Science*, 27(4):591–635.

Gorniak, P. and Davis, I. (2007). Squadsmart: Hierarchical planning and coordinated plan execution for squads of characters. In *AIIDE*, pages 14–19.

Hedden, T. and Zhang, J. (2002). What do you think i think you think?: Strategic reasoning in matrix games. *Cognition*, 85(1):1–36.

Isla, D. (2005). Managing complexity in the halo2 ai.

Klein, G. A., Orasanu, J. E., Calderwood, R. E., and Zsambok, C. E. (1993). Decision making in action: Models and methods. In *This book is an outcome of a workshop held in Dayton, OH, Sep 25–27, 1989.* Ablex Publishing.

Meijering, B., Van Maanen, L., Van Rijn, H., and Verbrugge, R. (2010). The facilitative effect of context on second-order social reasoning. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 32.

Meijering, B., Van Rijn, H., Taatgen, N., and Verbrugge, R. (2011). I do know what you think i think: Second-order theory of mind in strategic games is not that difficult. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 33.

Miller, S. A. (2012). *Theory of mind: Beyond the preschool years*. Psychology Press.

Perner, J. and Wimmer, H. (1985). "john thinks that mary thinks that…" attribution of second-order beliefs by 5-to 10-year-old children. *Journal of experimental child psychology*, 39(3):437–471.

Pew, R. and Mavor, S. (1998). Modeling human and organizational behavior.

Premack, D. and Woodruff, G. (1978). Does the chimpanzee have a theory of mind? *Behavioral and brain sciences*, 1(4):515–526.

Rabin, S. (2000). Designing a general robust ai engine. *Game Programming Gems*, 1:221–236.

Reitter, D., Juvina, I., Stocco, A., and Lebiere, C. (2010). Resistance is futile: Winning lemonade market share through metacognitive reasoning in a three-agent cooperative game. *Proceedings of the 19th behavior representation in modeling & simulation (BRIMS). Charleston, SC.*

Sanner, S., Anderson, J. R., Lebiere, C., and Lovett, M. (2000). Achieving efficient and cognitively plausible learning in backgammon. In *ICML*, pages 823–830. Citeseer.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484.

Van Maanen, L. and Verbrugge, R. (2010). A computational model of second-order social reasoning. In *Proceedings of the 10th international conference on cognitive modeling*, pages 259–264. Citeseer.

Whalen, J., Gallistel, C. R., and Gelman, R. (1999). Nonverbal counting in humans: The psychophysics of number representation. *Psychological Science*, 10(2):130–137.

Yannakakis, G. N. (2012). Game ai revisited. In *Proceedings of the 9th conference on Computing Frontiers*, pages 285–292. ACM.

Zsambok, C. E. and Klein, G. (2014). *Naturalistic decision making*. Psychology Press.

Appendix A

Instructions Experiment

Thank you for participating in this experiment. Today you will be playing a game against three opponents (computer models) on an iPad. The goal of the game is to get as few points as possible, meaning the player with the least amount of points at the end wins the game.

The game you will be playing is called *No Thanks*. The game contains 33 unique cards numbered 3 to 35 and a number of *pass tokens*. Each card is worth its face value in points, while a pass token subtracts one point for each token a player owns (-1 point). Each player gets 11 pass tokens at the start of the round.

During each turn, the top card from the shuffled deck of cards is put face-up on the playing field. You always have two possible actions when it's your turn:

1. Do not pick up the face-up card by placing a pass token on it. The turn then continues to the next player (going clockwise). If you do not have any pass tokens left, you have to pick up the card.

2. Pick up the face-up card (along with any pass tokens that have already been played on that card) and turn over the next card from the pile. You can then decide your move with the new card (it is still your turn).

Picking up a card means its face value is added to your score minus any number of pass tokens that were on that card. Every card you take will be placed face-up on the playing field in front of you. This means that all players can see each other's cards. However, the number of pass tokens you own is only visible to you.

The value of every card you own is added to your score. However, if you own a sequence of cards, *only the value of the lowest card of that sequence is added to your score*. For example, if you own the cards 20 and 22 their total score would be 20 + 22 = 42 points. Yet if you complete the sequence 20 to 22 by picking up card 21 (thus creating 20, 21, 22), their score would only be 20 points (21 and 22 are then worth 0 points).

However, 9 random cards are removed from play before the game starts. This means a card that would complete your sequence may actually not be in the game and thus cannot be picked up by anyone.

The game ends when all 24 cards of the pile have been picked up by a player. The player with the lowest score (card points – pass tokens) wins the game.

You will be playing the game against 3 opponents (computer models) on an iPad. Swipe up on the pass token in the bottom right to place a token on the face-up card. Swipe down on the card to pick it up (and any pass tokens on it) and add it to your own cards. You will play 24 games, which will take about 70 minutes. A button which says 'start next game' appears at the end of each game until the experiment is finished.

Additionally, in this experiment you are also asked to give feedback about the models' actions. When it is your turn, you are asked whether any of your opponents made a bad move in its last turn. If so, you can tap on that opponent to select it (marking it red). You can then continue by making your move. Note that it is not at all required to select an opponent on every turn. Only if you think it made a bad move (you can select multiple opponents during the same turn).

Try to get as few points as possible!

Good luck!