Collaboration analysis in multi-player based simulations

Bruno Carreira
bruno.carreira@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal
December 2020

Abstract

This work aims at helping developers of interactive software to test collaboration inducing scenarios. When creating a training simulation for team building, developers must make sure that their scenarios promote collaboration but also, don’t force it, meaning a scenario must allow users to behave freely, otherwise did they really collaborated or were just forced to? This creates a difficulty, how can developers test their scenarios on their capability of allowing different behaviours? Our approach is based on using two different automated agents behavioural traces, one specifying the scenario’s Design Goal, collaboration, and the other an example of a non-Design Goal, acting individually. After training said agents and by comparing the agents optimal behaviour when solving each scenario to the two policies, we can determine if the scenarios allow to differentiate between the Design Goal and the non-Design Goal. With this approach we are also able to order the scenarios from easiest to differentiate to hardest. Our approach was tested in two different environments, in a custom built simulator and in the iv4XR game Lab Recruits.

Keywords: Multi-player based simulation; scenario testing; collaboration detection; automated agents behavioural traces, automated testing.

1. Introduction

We want to develop a method that helps designers of interactive software to test their scenarios and see if they can distinguish different behaviours.

Companies that aim at developing interactive software must take into consideration the different components that can bring different contributions to their products as well as the problems that come with said contributions, like their dynamic and subjective concepts as well as their financially expensive and time demanding testing procedures.

One example of this, concretely in line with our work, is in the use of serious games for team training simulation. When a company is focused on creating a training simulation to improve team collaboration, several questions arise. Can the developers conclude that there was teamwork by simply observing if the team reaches the objective? What if the individual members didn’t work as a team and they actually just acted by themselves but luckily reached the objective? How about if the scenario allowed them to have a choice between collaborating and not? Did the scenario simply forced them to work together? If so, can they really say that they learned to collaborate? Answering these questions can be somewhat of a difficult task, especially during the developing phase of scenarios where there are constant changes in the maps, objectives, among others.

Emerging from the iv4XR project\(^1\), our main goal is to use Reinforcement Learning (RL) policies to evaluate designed scenarios on their capability of allowing and distinguishing collaborative and non-collaborative behaviours. These policies will be extracted from automated agents developed for this work and corroborated from real-life users.

We will consider different scenarios in two different examples:

- Squary-Shappy - Self-developed game/simulator for this work.
- Lab Recruits - AIGym and game developed for the iv4XR project.

2. Related Work

The approach taken for this work derives from areas such as Automated playtesting with agents and Collaborative behaviour, particularly regarding understanding how collaboration can be identified in interactions between humans and structured between agents.

\(^1\)www.iv4xr-project.eu
2.1. Automated playtesting with Agents
The main objective behind automated playtesting consists in the use of autonomous agents that will test the software and gather data only previously obtained by real-life users. Correlated to our work, automated playtesting with agents is commonly used in the gaming industry where automatically testing maps and specific scenarios is one of the main focuses.

Silva et al. [3] expanded on their previous work [2] and demonstrated how the use of different intelligent agents allowed them to evaluate the Ticket To Ride game. Their work consisted on simulating common user strategies, gathered from the game’s community forums, with game related heuristic-based automated agents and, with said agents, analyse several map variations on their impact in different strategies and number of players. Their work proved to be successful since they were able to show that it is possible to both characterize the desirability of various parts of the maps, the relative strengths of strategies in the map variants and with different number of total players. They were also able to determine the differences between maps in terms of what strategies work best as an unexpected result, they also identified two failure cases in the game, where agents found game states that were not covered by the game rules.

Holmgard et al. [6] presented a method that consisted on using player modeling to create automated agents as game-personas\(^2\) and enabled them to automatically playtest and evaluate content in the game MiniDungeons 2. After several experiences they observed that use of automated persona-like agents, in games with similar complexity to the MiniDungeons 2, proved to be helpful when evaluating the game levels, more significantly, identifying which maps were better suited for each type of players.

From understanding which strategy is better for a certain map to determining how the design of levels affect different playstyles, the work mentioned in this Section provides an indication that analysing game maps and scenarios can be done using specifications of design goals (strategies, game objectives, among others) in automated agents, which is in-line with this works objective of using automatic agents as a way of evaluating game-like scenarios on their capability of distinguishing collaborative and non-collaborative behaviours (design goals).

2.2. Collaborative Behaviour
To understand how to differentiate and recreate collaborative and non-collaborative behaviours for our automated agents, we must first understand how collaboration works and is defined in human-human interactions to then define how to replicate it between agents.

2.2.1 Collaboration between humans
Although every situation is unique and every group interacts in a different manner, there are several collaboration aspects that are considered universal such as, clear and open communication [8], consensus about goals and methods for completing tasks [5], clear definitions on each contributors role [9], placing group goals above individual satisfactions [10], among others. The successful implementation of these elements through communication and coordination is the key for teams/groups to work effectively in a collaborative manner. An example of this is presented in [1] where Bagshaw et al. displayed their work and strategies regarding collaboration in international research projects and how to promote it.

We’ve shown capable of pointing out different elements of collaboration but can we explain how teams connect all these strategies and define a structure for collaborative behaviour? Stephen et al. [4] researching high performing teams suggested that experienced teams develop a shared mental model in order effectively coordinate and predict each others movements while also improving their communication. SMM are described as knowledge structures for group members to similarly understand the team’s objectives, peers individual needs and everyone's responsibilities. The use of this structure allows for all members of a team to have the shared understanding of a problem and to have a clear understanding of the most effective way to approach the problem, meaning the techniques and resources the team should utilize as well as everyone’s role in them. An example of a shared mental model can easily be explained in a professional team sports environment where the members of a team must rely on their shared understating of a game situation and play according to their knowledge of what is the best strategy for the team. The use of shared mental models allows for all members of a team to visualize a problem at hand with the same mindset and anticipate future actions and states. No matter the complexity or simplicity of a task, the use of shared mental models has been identified as a collaborative indicator in human-human interactions [7].

2.2.2 Collaboration between agents
Creating collaborative agents can be a difficult task. How can we have artificial agents working as team? Putting group goals above their own?
How can we create agents that really work as a team and not just happened to have the same individuals goals? Different methods have been used but a common technique on helping agents collaborating is to create a system similar to what human teams develop, a shared mental model. The SMM method has been widely researched for intelligent agents ([13], [14]) and proven to help agents act collaboratively. Similarly to humans, a mental model means an internal representation of a situation and the shared knowledge of the best actions for the group. More specifically for automated agents, a SMM defines a cognitive structure for common information access and that connects and extends the notion of individual goals/needs to a team context [11]. The use of an SMM is proven to be an effective method to help agents collaborating, but since for this work we will evaluate collaborative behaviour, we want to create inherently collaborative agents. One way to do that in to use a centralized method. A centralized approach aims at providing a complete scheme of the general current state [12] to all agents, simulating intrinsic communication, as well guaranteeing that by executing coordinated joint actions, the best action for both agents (collaboration) is always chosen.

The work presented in this Section showed us that based on interactions between humans we can define a structure, SMM, to help agents act collaboratively. Since we want to have inherently collaborative agents we can also utilize the centralized method that possesses homogeneous properties. These structures/methods served as the base inspiration for our automated agents architectures, further explained in Section 3.

3. Methodology

In this section, we will describe the approach used to create, detect and analyse behaviour in each scenario and a detailed explanation of the two environments used to test our solution. For both environments we will detail the map designing process, the different architectures of the automatic agents, the algorithms implemented to train the agents as well the optimization methods for said agents. We will also present a description of the real-life users playtesting experiment made with the Lab Recruits game.

3.1. Behaviour Comparison Approach

In a software testing method the main premise is to check if the software reaches its main purpose, the Design Goal (DG). For a game developer the DG can be to create an engaging experience, for a developer of an intelligent tutoring system a DG could be to ensure that the students acquire the knowledge components and for a developer of a serious game for team training simulation, the DG may be that the users work as a team. Although this is strongly related to regular software testing, the main difference is that in our case the Design Goal can not be specified only in terms of logic properties of the software but also need to take into account human factors.

3.1.1 How to define the Design Goals?

Design Goals can be defined in many different ways such as examples of the intended behaviour, a reward/cost function that induces the behavior, a set of rules and many others. For this work we used Behavioural traces where the DG can be directly specified via designer provided demonstrations of the desired behavior as well as different behaviours that are to be tested. For instance if the DG is to train users on how to work as a team, we provide traces of a group of people working as a team and traces of people not working as a team.

3.1.2 How to test a scenario?

We will now understand how we can test if a given scenario achieves a desired Design Goal.

Considering that some DG are very abstract there are numerous situations where we cannot guarantee if a DG is achieved, but, in most cases, we can determinate if a DG is not achieved. A clear example is if our DG is to understand if a user can learn basic arithmetic’s. If we present a user with $(2 \times 2 + 2)$ and he responds $6$, although is the correct answer, we are still not sure if the user understood that $\times$ has priority in relation to $+$. Meaning that for any of the different ways to specify the Design Goals we must get a set of behaviours that achieve the desired goals $\pi_g$ and another set that does not verify the design goals $\bar{\pi}_g$.

Figure 1 explains this difference. On the left side we have a system that given only a desired behaviour is able to decide if given scenarios fulfilled the DG or not, but, like we’ve seen, this doesn’t always work. Whereas on the right side we consider that by having the definition of the desired behaviour and the undesired behavior allows the system to then decide if not only the DG is fulfilled but also if the non DG ($\neg DG$) is not.

Given this difference, our approach is based on testing each scenario on the DG and on a specification of $\neg$ DG to determine if we can distinguish them and order the scenarios on their ability to differentiate both behaviours. Although there are clearly many different amounts of possible $\neg DG$, for specific problems, such as ours, the main effects can be easily identified.
3.2. Squary-Shappy
In order to perform initial tests on our approach, we started by creating our own simulation gym, a Python-based program called Squary-Shappy that, due to its simplicity, served as the initial environment to create automatic agents and implement our approach. The Squary-Shappy scenarios simulate a 2D object collecting game where agents roam around the map trying to eat as much food as they find. The food is positioned around the map and does not regenerate, meaning the food is limited by the number initially deployed. The maps are also all designed in a closed room format, meaning the agents cannot move out of the established bounds.

3.2.1 Map Designing
Our initial work in map designing consisted of creating 1-Dimensional maps to test our agents. Since 1-Dimensional maps are somewhat restrictive we quickly moved on to 2-Dimensional maps. The maps in the Squary-Shappy environment were designed in individual .txt files which allowed us to quickly create new scenarios and alter existing ones, since that, at their core, they were a simple deterministic matrix using a basic symbol format.

3.2.2 Agents
To automatically create the previously mentioned desired and undesired behaviors we started by developing two different agent architectures and training them using RL.

To define our Desired Behaviour we implemented the Centralized architecture since our goal is not to teach agents to collaborate, but instead to create inherently collaborative and coordinated agents to then use their behavior to evaluate the scenarios. The Centralized agent type means that all agents are a part of a shared mind that takes into account the effort made by every single agent and fundamentally makes them function as a team. On the other hand, for our Undesired Behaviour we decided to use the Individual architecture meaning that for these type of agents, each have an individual "mind" and make decisions based on their own personal gain, meaning they prioritize actions that provide themselves with the highest individual reward and don’t take into consideration other agents.

In order to train the agents we chose to use a simple reinforcement learning approach called Markov Decision Process (MDP) with Q-Learning. An MDP is predicated on the Markov Property “The future is independent of the past given the present” which implies that in a RL problem, the next state $s_{t+1}$ only depends on the current state $s_t$.

The MDP with Q-Learning algorithm cycle is easily explained in Figure 2.

This image shows that during the training, agents observe their current state and choose an action to execute. They will then receive a reward (positive or negative) and observe the state of the environment after performing the selected action. This information will then be used to update the Q-Table, the agents’ “brain”, by means of the $Q$-Learning formula:

$$Q(s, a) = Q(s, a) + \alpha \times [r + \gamma \times \max (Q(s', a')) - Q(s, a)]$$

Where $Q(s, a)$ represents the $Q$-value of the performed action in the current state, $\alpha$ represents the learning rate, $r$ represents the reward, $\gamma$ is discount factor, a real value in the range $[0, 1]$ and is used to balance the relevance of the immediate reward with respect to future actions. $\max(Q(s', a'))$ is the maximum $Q$-value for any action of the next state.
The scenarios were formulated into a RL problem by defining the MDP parameters as the following.

- **States:** Since the agents are only able to move one step at a time, the food has a static position and is limited, we decided to describe the state of the environment using the positions of the agents as well as the position of the food objects that still exist. Every time a food object was eaten, that position was removed from the state.
  - Individual: \{PosAgent, PosFood1, PosFood2, PosFood3, ...\}
  - Centralized: \{PosAgent1, PosAgent2, PosFood1, PosFood2, PosFood3, ...\}

- **Actions:** In the Squary-Shappy simulator the agents can only perform low-level actions. It is also relevant to mention that in this environment, a character eats an object by simply occupying its position, no additional move was required. For the Individual agents they had the following action space:
  - NOTHING = Stay in the same place.
  - UP = Take one step up.
  - DOWN = Take one step down.
  - LEFT = Take one step left.
  - RIGHT = Take one step right.

For the Centralized architecture, the agents acted in joint actions e.g. \[RIGHT, RIGHT\] = Both agents take one step right, and all remaining permutations.

- **Rewards:** Every time an agent consumed a food object they were positively rewarded with a value of +100. If the agent didn’t move they didn’t receive any reward, positive or negative, meaning the reward was 0, but for every step an agent took they would lose energy, meaning they would get a small punishment of −1. The reward system for the Centralized agents functioned as a collective, meaning the actual centralized agent, the shared mind controlling two characters, received the combined amount, whereas for the Individual type agents, they were attributed their own rewards.

### 3.3. Lab Recruits

Upon creating the Squary-Shappy gym and implementing our approach in it, we moved on to a more high-end environment, the Lab Recruits game designed by the University of Utrecht for the iv4XR project. In this game, the players objective is to click the target green button(s). For this to happen, the characters have to roam around the maps and click red buttons to open doors and access new rooms. Contrary to the Squary-Shappy food objects, these buttons do not disappear and could be clicked unlimited times, turning them ON/OFF to open/close doors.

For the Lab Recruits game, the iv4XR framework ran all its tests and simulations in a Unity environment. Every move/action/change done in the map had to be passed to the Unity application through a socket, applied using the Unity game physics engine and relayed back to the Java application through the same socket. Although this method provides more accurate data regarding to map positioning, physics, among others, the amount of time needed to train agents in real-time would be too much. For this reason, and since for this work and in our scenarios the Unity physics could be considered meaningless, we decided to create a model of the map that could be simulated in a 2D matrix, similar to the Squary-Shappy environment, to help speed up the agents training process and still provide the wanted policies.

#### 3.3.1 Map Designing

In the Lab Recruits gym, the map designing followed a very similar approach to the one we took in the Squary-Shappy scenarios, meaning that we used a clear symbol format, previously defined by the University of Utrecht, that could easily build/alter scenarios in a .csv file. Although the game itself (Unity executable) possessed physics, the maps were designed as a 2D matrix using discrete units which also lead for the agents to train in that manner, meaning that 1 step in the agents training meant several physical steps to go through the same distance in the game.

#### 3.3.2 Agents

For this environment, we used the same approach for the agents as in the Squary-Sghappy environment, we again developed Centralized agents who are a part of a shared mental mind and Individual agents that have their own individual decision process and only take into account their own state and benefit. The development and training of the agents was done inside the Java iv4XR framework and in order to train the agents, we once again used an MDP with Q-Learning since it provided simplicity in the implementation. Instead of training the agents using the game itself, we developed a model to simulate the environment of the Lab Recruits. For the agents training we defined the following MDP parameters:
• States: In the Lab Recruits games, the agents interact with buttons to open and close doors until reaching the target button. Since the state of the doors (open or closed) are the only visual feedback that real life players have, we choose to use this as part of the agents MDP state. The doors internal state can be described as '0' if the door is closed or '1' if the door is open. Appending this to the agents position resulted in the following policy states:

- Centralized: {PosAgent1, PosAgent2, IntStateDoor1, IntStateDoor2, ...}
- Individual: {PosAgent, IntStateDoor1, IntStateDoor2, ...}

• Actions: Although the iv4Xr had a built-in pathfinding system with high-level actions we decided to use the basic low-level actions and added one specific action related to the Lab Recruits game, pressing on a game button. The Individual agents could perform the following actions:

- NOTHING = Stay in the same place.
- UP = Take one step up.
- DOWN = Take one step down.
- LEFT = Take one step left.
- RIGHT = Take one step right.
- PRESS = If the character is on top of a button, press it.

For the Centralized type agents to act coordinately, their actions were again in pairs with all possible permutations, similar to the Squary-Shappy environment.

• Rewards: Agents will get positively rewarded(+100) for reaching the target (turning ON a target button), no reward for doing nothing and a negative reward (-1) for all other actions. The only thing we had to make sure was, since the buttons could be clicked unlimited times, the agents would only get a positive reward the first time a target button was turned ON.

3.3.3 Optimizations - Dyna Q

One difficulty that we encountered in the Lab Recruits game was the time taken to train the agents, even using a 2D matrix model representation of the game. In order to try speed up the process, we implemented an optimization method for the agents training process, the Dyna-Q.

The Dyna-Q algorithm aims at collecting the agents past experiences and using them to further update the policy without the need for the agents to physically perform an action in the simulation world. Upon each cycle (after each Q table update) the agents save their experience by updating a Transition table. The Transition table contains the information that if an agent is in a state S and performs action A, then the resulting state will be S’ and will receive reward R’. Upon each training episode, the agents enter a state of “hallucination”, commonly used to describe a Dyna-Q cycle since no actions are done in the real world, only the information in the agents memory (Transition table), where they use the information inside the table to virtually replay several random experiences and update the Q table with the resulting values. The agents replay random movements by selecting a random state and action that they already experienced, followed by searching in the Transition Table for the corresponding reward and resulting state to then update the Q-Table. This means that the agents are using their “memory”, from past experiences, to reproduce real values and use them to update the Q-Tables.

3.3.4 User Playtesting

Although the purpose of our work is to help developers test scenarios that need to promote collaborative behaviours without the need of real-life users, in order to understand if the agents policies could be deemed as “human-like behaviour”, we decided to have a playtesting experience with users to corroborate, or not, the agents behaviour and our results.

An online form was created containing all the information regarding the experience and a link to download the game. To try and create the same environment as our automated agents, users, in pairs, were asked to play a number of scenarios three times (since agents had several training episodes), while trying to reach the objectives as fast as possible and also avoiding having any type of communication between themselves. The questions selected were created to help understand the internal decision process from the users to verify our systems accuracy, further explained in Section 4.3.

The data gathered from the users playtesting were their behavioural traces, meaning that at every frame and using the same centralized state format from our MDP Centralized implementation. We recorded every action made the users and linked them the game state, resulting in the traces being a sequence of state-action elements. Here we present an example:

< (3, 0, 7), (9, 0, 7), [1, 0], (Left, Nothing) >
3.4. How to identify behaviours?

Our main objective is to help developers of collaborative inducing software to test their scenarios by comparing behavioural traces of a DG and a ¬DG to understand if the scenarios allow to distinguish between collaborative and non-collaborative behaviour. In order to achieve that, we had to create a method of comparing any behavioural trace with the Centralized policy (DG) and the Individual policy (¬DG).

We had to define a voting system to make sure every action was properly accounted for. We decided to go with the softmax function using the policies action Q-values. The softmax function, commonly used in Machine Learning, serves as a function to calculate the likelihood of a given input, state-action pair, belonging to the target classes, Centralized and Individual. One relevant property of this function is the output values since they’re in the range [0, 1] which is useful because we can avoid a binary classification and allows for easy interpretation. The softmax function goes as following:

\[
L(s,a) = \frac{e^{Q(s,a)}}{\sum_{k \in \text{actions}} e^{Q(s,a_k)}}
\]

(1)

Where \((s,a)\) represents a given state-action pair, \(Q(s,a)\) is the Q-value of a policy for said state-action pair and \(Q(s,a_k)\) is the Q-value of an state-action pair where \(a_k\) is a specific action belonging to the centralized action space. By calculating the highest likelihood, this voting process will allow us to identify if any given behavioural trace followed the DG or the ¬DG and since the system outputs a normalized similarity value for both policies we will be able to calculate the difference of said values which will be used to order the scenarios by their capability of allowing and distinguishing collaborative and non-collaborative behaviours, from best (highest difference) to worst.

4. Results

In this chapter, we will explained the results gathered from our experiments in both environments, the Squary-Shappy simulation and the Lab Recruits game.

4.1. Squary-Shappy

We began our experiments using our custom simulator, Squary - Shappy. In this environment the objective is for two agents to roam around the map and eat all the food objects. For this experiment we created three scenarios to test, depicted in Figures 3, 4 and 5.

In order to understand if we could differentiate the DG, collaborative behaviour, and the ¬DG, non-collaborative behaviour, we used both agent types optimal behaviour (always perform the best actions) when solving each of the three scenarios and, using the likelihood voting system, compared those behaviours to both policies, Centralized and Individual.

![Figure 3: SS: Scenario 1 - “Advantage red”](image)

![Figure 4: SS: Scenario 2 - “Perfect division”](image)

![Figure 5: SS: Scenario 3 - “Center focus”](image)

![Figure 6: Results: SS Centralized agents optimal behaviour](image)

![Figure 7: Results: SS Individual agents optimal behaviour](image)
Figures 6 and 7 depict the results for the centralized agents optimal behaviour. The green columns are representative of the likelihood values for agents optimal behaviour to belong to the Centralized policy, whereas the orange columns represents the value of likelihood of the same behaviour belonging to the Individual policy.

Figure 8 shows us the likelihood difference, meaning these were calculated by summing the difference between the likelihood values of the centralized agents optimal behaviour and the likelihood values of the individual agents optimal behaviour belonging to the Centralized and Individual policy. From this figure we extrapolated the order of the scenarios that allow and better distinguish collaborative and non-collaborative behaviours. With the largest likelihood difference value of 0.032 Scenario 3 “Center focused” is considered the best scenario to differentiate the two behaviours, followed by Scenario 1 “Advantage red” with 0.024 and finally Scenario 2 “Perfect division” with 0.002.

4.2. Lab Recruits - Automated Agents

Moving to the Lab Recruits game where agents, and now also real-life users, had to click on a sequence of buttons to achieve each map’s final objective. The maps created to test our approach in the Lab Recruits game are depicted in Figures 9, 10 and 11.

Similar to the Squary-Shappy environment, we wanted understand if the scenarios could differentiate collaborative and non-collaborative behaviour. We again used the centralized and individual agents optimal behaviour when solving the three mentioned scenarios and using the likelihood voting system calculated the likelihood of those behavioural traces belonging to the Centralized and Individual policy. We now present the results gathered.

Figures 12 and 13 show the results regarding the centralized and individual agents optimal behaviours likelihood of belonging the Centralized and Individual policies. The green columns are again relative to the likelihood of the agents optimal policy belonging to the Centralized policy, while the orange columns indicates the likelihood of the same behaviour being more representative of the Individual policy.

By again creating a figure to display the likelihood differences on all scenarios to order them by their ability to allow and distinguish collaborative and non-collaborative behaviours, we obtained the results depicted in Figure 14. This figure shows a considerable difference between the scenarios, especially regarding Scenario 3 “Random maze” with
a likelihood difference of 0.57, and gives us the scenario order regarding the scenarios capability to distinguish collaborative and non-collaborative behaviour. The order, from best to worst, starts with Scenario 3 “Random maze”, followed by Scenario 1 “One man’s choice” with 0.032 and lastly Scenario 2 “Even distribution” with 0.008.

4.3. Lab Recruits - Users playtesting

We made a playtesting experience where we asked users to play each of the Lab Recruits scenarios three times and to individually answer a questionnaire. This experience was made to corroborate the results gathered from our approach regarding the order of how better each scenario can distinguish collaborative and non-collaborative behaviours and also aimed at collecting the users behavioural traces to verify if our agents could be deemed as human-like. We receive 13 playtesting experiences and questionnaires responses, totaling 26 individual participants.

The questionnaire consisted on three questions for each scenario, the main one being: When playing Scenario X, did you feel that you played as a team, played in an individually manner or neither? - This question helped us determine if our system was identifying the correct behaviour when comparing the users to our agents, meaning determining if our agents did simulated human-like behaviour. Another important question was we asked users to order the scenarios, from best to worst, on which scenario they felt promoted the most collaborative behaviour in order to compare this order the one we arrived on using the automated agents experiences.

By combining the answers given by users in question 1 of the questionnaire where “Played as a team” meant both users said that they played as a team, “Played individually” is when both users said that they had an individual play style and “Mixed behaviour” is referent to a mixed play style by the user-pair and by calculating the likelihood values of the user behavioural traces belonging to each policy with the same voting system as in our automated agents experiences, we got the results depicted in Figures 15, 16 and 17. Regarding Scenario 1 we can calculate that our system had an accuracy of $\approx 43\%$ when linking supposed collaborative behaviours to the Centralized policy and an accuracy of $\approx 33\%$ when linking supposed individual behaviours to the Individual policy. We also noticed that all user-pairs with mixed behaviours were categorized as more likely to belong to the Centralized policy. For Scenario 2 we observed that no supposed collaborative behaviours were more likely to belong to the Centralized policy and also that only two supposed individual behaviours were more likely to belong to the Individual policy. This can however be explained since all like-
likelihood values are extremely close to each other, which was somewhat expected due to our results from the automated agents and confirms that Scenario 2 has a lower capability of distinguishing behaviours. In Scenario 3 we observed the users didn’t had mixed behaviours but, our system attributed a higher likelihood value to the Individual policy on all behavioural traces.

For the last question of the questionnaire where we asked each user to order the scenarios by their ability of promoting collaborative behaviour, users answered that Scenario 3 promoted the most, followed by Scenario 1 and lastly Scenario 2. This subjective scenario order by the users corroborates the scenario order achieved in our automated agents experiences. A conflicting result were the likelihood comparisons between the users behavioural traces and both policies since the system couldn’t accurately label the users collaborative behaviours as belonging to the Centralized policy and individual behaviours to the Individual policy. This was easily demonstrated in Figure 17 were all traces are categorized as more likely to belong to the Individual policy even though 80% of the user-pairs answered that they played as team.

5. Conclusion and Future Work

Both the automated agents experiences and the users subjective answers to the questionnaire are an indication that our approach of using the Centralized and Individual RL policies can in fact test scenarios on their ability of distinguishing collaborative and non-collaborative behaviours. However the results from comparing the users behavioural traces to the policies raised questions regarding the likelihood method used.

Although this work proved to have some limitations, more precisely the likelihood comparison system, the results gathered give us an indication that this approach of using definitions of a Design Goal and a non-Design Goal as behavioural traces from automated agents provides positive results and allows to order the scenarios by their ability to allow and distinguish different behaviours. We believe that this is evidence that this approach can provide developers with initial data regarding their scenarios during the development process.

Regarding future work, different comparisons between the the policies and the behavioural traces should be done. We also believe that a more accurate definition of the Design Goal and the non-Design Goal can be achieved using other methods such as a combination of Reinforcement Learning with Imitation Learning where a group of experts can provide general traces of the DG and the $\neg DG$ to more accurately represent human-like behaviour. Another interesting possibility is the combination of this approach to the analysis of agents and users internal state since it could allow developers to not only determine if their scenarios are being developed according to the design goals but also allow them to have an indication of which areas/components of the scenarios are affecting the agents and users decision making.

To finalize, we would like to state that is also our believe that this approach is not limited to collaboration and could be used to analyse other types of behaviours as long as definitions of the DG and the $\neg DG$ can be properly provided.

References


