ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal
Regular Issue, Vol. 7 N. 4 (2018), 17-26
eISSN: 2255-2863
DOI: http://dx.doi.org/10.14201/ADCAIJ2018741726

Multi-Agent Word Guessing Game

Gabino Luis, Jaime de la Peña, David Suárez, and Alfonso J. Mateos

u147689@usal.es, jpenara11@usal.es, davidsm@usal.es, xanthux@usal.es
Universidad de Salamanca

KEYWORD

Gaming; Multi-agent Systems; Execution time

ABSTRACT

The task of creating algorithms to solve a problem is surely a hard thing as it can be the fact of evaluating them. A well designed algorithm can be very powerful but, it may lack of efficiency at some aspects. This paper proposes a multi-agent system based game with three types of agents: CBot, ABot and QBot, which stands for Coordinator, Answer and Question. They will play a game based on questions and answers, where each of the QBots uses a different algorithm to guess a word. The CBot has the responsibility of the efficiency measurements, receiving and manipulating the ABot reports. The game will finish once all QBots give the correct answer and after that, the efficiency of the algorithms thanks to the CBot. Using this method, it is easier to determine which algorithm is the best with a given performance measurement.

1. Introduction

Measuring the efficiency of an algorithm, and especially of an algorithm in comparison to others may become a hard task in complex projects. Keeping in mind that runtime will not be always the factor to optimize and that the structural difference between algorithms can be vast, developing a general tool to measure algorithms' performance is an arduous work. This paper proposes the initial step to develop a general testing environment: a smaller model designed to evaluate a certain type of algorithm in a pre-defined scenario. The objectives the mentioned project is meant to fulfill are correctly implementing the algorithms' functionalities inside the testing environment simultaneously, running and measuring them several times to obtain a set of results, and evaluating those results to extract conclusions about their performance. Completing these objectives will provide valuable information about the algorithms' efficiency, useful to decide which one to implement on the program it is designed to, or simply to observe the differences in the behaviors depending on the inputs.

This project proposes a word-guessing game as an example, where the measure factor will be the turns that the algorithms take to guess the word correctly. The objectives on this particular case are running the algorithms simultaneously, obtaining the number of turns taken of each algorithm for the same word, and evaluating those results, considering the word length together with the turns taken to extract more useful conclusions. Solving this problem successfully will assure a solid base for a more general testing tool, which may be used for testing more diverse algorithms.

In order to accomplish this, a multi-agent environment is chosen as the backbone of the system. It has been decided to use the JADE platform, a multi-agent framework for the Java programming language; however, any other multi-agent platform can be used for this problem. The main motivation as why a multi-agent scheme

is preferred over other types is the decentralized operating mode they provide, because it is easy to add new agents to the testing procedures in real time, so good scalability is assured. In the testing example for his paper three types of agents are used, called C-Bot, A-Bot and Q-Bot that stands for Coordinator Bot, Answer Bot and Question Bot. The Answer Bot only has access to the word that needs to be guessed correctly by the Question Bot, which in turn has the list with all the possible words that can be the correct one. In agent theory there are broadly three main groups of agents: reactive, deliberative and hybrid agents. Reactive means there are no problem-solving involved in the agent when an external precept is given to him, he uses an internal table with all the recorded precepts he might get and the responses he should give to every single one of them. This structure is very inflexible and scales badly as new information is added but, as an advantage, it makes the agent responsive. The A-Bot in the proposed system follows this pattern, because the inputs he might receive are controlled and known and he is servicing multiple Q-Bots on his own, so speed in the response is critical. Deliberative agents, on the other hand, have internally some type of problem-solving mechanism and need to use a symbolic representation of the knowledge they receive; the structure can have multiple levels of complexity but, in the end, it makes the agent less responsive than working with a input-response table, because he needs to search for an answer given an input; in return, however, the agent can solve different problems that belong to some predefined domain. The Q-Bots use this architecture because: first, they have the algorithms that are subject for testing; and second, they need to work with the knowledge of discarded words in the given dataset in order to guess the keyword correctly. The communication between agents is done via messaging. In the proposed example the agents communicate between them with direct messages because there are only four Q-Bots; however, there are no restrictions with the number of A-Bots and Q-Bots the system might have and, in order to make it scalable, the Q-Bots should ask first for an agent that can give answers to their questions; so the A-Bot with the least workload could process the request.

In Section 4 two experiments are presented, using two different datasets. They consist in running the multi-agent system created several times, gathering data and classifying them to analyse and extract valuable information afterwards; information that is the main goal of this paper. Having these experiments working successfully allows to move into the next step which would be to scalate the created model to have it working on many other different type of scenarios. Finally, some of the results are also presented and explained, having some unexpected information that wouldn't have been possible to detect without this new model.

2. Related work

Most important works that use agents to communicate for a specific goal use image captioning (Kiros *et al.*, 2014); (Xu *et al.*, 2015); (Vinyals *et al.*, 2015); (Johnson *et al.*, 2016); (Lu *et al.*, 2017); (Yao *et al.*, 2017) and visual question-answering (Antol *et al.*, 2015); (Zhang *et al.*, 2016); (Goyal *et al.*, 2017) which was first introduced by (Das *et al.*, 2016).

The first half of the table are articles focused on image captioning, using neural networks in most of them and trying to improve the results of the current state of art by experimenting with some of the current models or even introducing new ideas. The second half is about visual question answering, where image captioning is also necessary. They also develop a learning system where the main goal is having the agents involved to be more accurate as they learn.

Table 1: Article references and important features

Article	Main Feature
(Kiros *et al.*, 2014)	Multimodal neural language
(Xu *et al.,* 2015)	Neural image caption generation
(Vinyals *et al.,* 2015)	Neural image caption generation
(Johnson *et al.*, 2016)	Dense captioning
(Lu *et al.*, 2017)	A visual sentinel for image captioning
(Yao *et al.*, 2017)	Boosting image captioning with attributes
(Antol *et al.*, 2015)	Visual question answering
(Zhang *et al.*, 2016)	Answering binary visual questions
(Goyal *et al.*, 2017)	Visual question answering
(De la Prieta *et al.*, 2013)	Adaptive learning system
(García *et al.*, 2017)	Intelligent systems
(Das *et al.*, 2016)	Visual dialog
(Agarwal *et al.*, 2018)	Visually grounded dialog

In reference to (De la Prieta *et al.*, 2013), the multi-agent system proposed in this paper has as similarity the adaptive and evolutionary process of the agents in a social environment, in the article referenced use primary school students to improve these agents and adapt them to the current needs of these children, on the other hand not only improve the intellectual capacity of children, furthermore, teach them to relax and stimulate their senses.

In reference to (García *et al.*, 2017), an adaptive multi-agent system is being used which, depending on the needs of the user and the interests of the user, can control and regulate the conditions of the surrounding environment. With this it is achieved that the system evolves and that progressively with the passage of time a greater efficiency of it can be obtained, in this proposed paper the use of sensors nor regulators has not been served, but it is pending an improvement of the algorithms used. Therefore, both can be improved even more.
The last article in the table(Agarwal *et al.*, 2018) talks about some problems that these methods present, proposing a new architecture trying to solve them. Those problems are in fact that those agents tend to develop their own language. This is really important to mention, the reason is that it doesn't really matter how much they learn if the human cannot understand that language. The same happens between them: a single pair of agents would have developed their own private language. This would be for nothing because a third agent wouldn't understand them. The mentioned article proposes a framework where they change this philosophy of learning. Instead of allowing single pairs of agents to learn, they opened them to a big community, where they would learn together and develop a common language. The idea of creating the proposed system in this paper came from investigating on this topic. It will consist of a famous word guessing game, where agents compete and show their efficiency.

3. Architecture

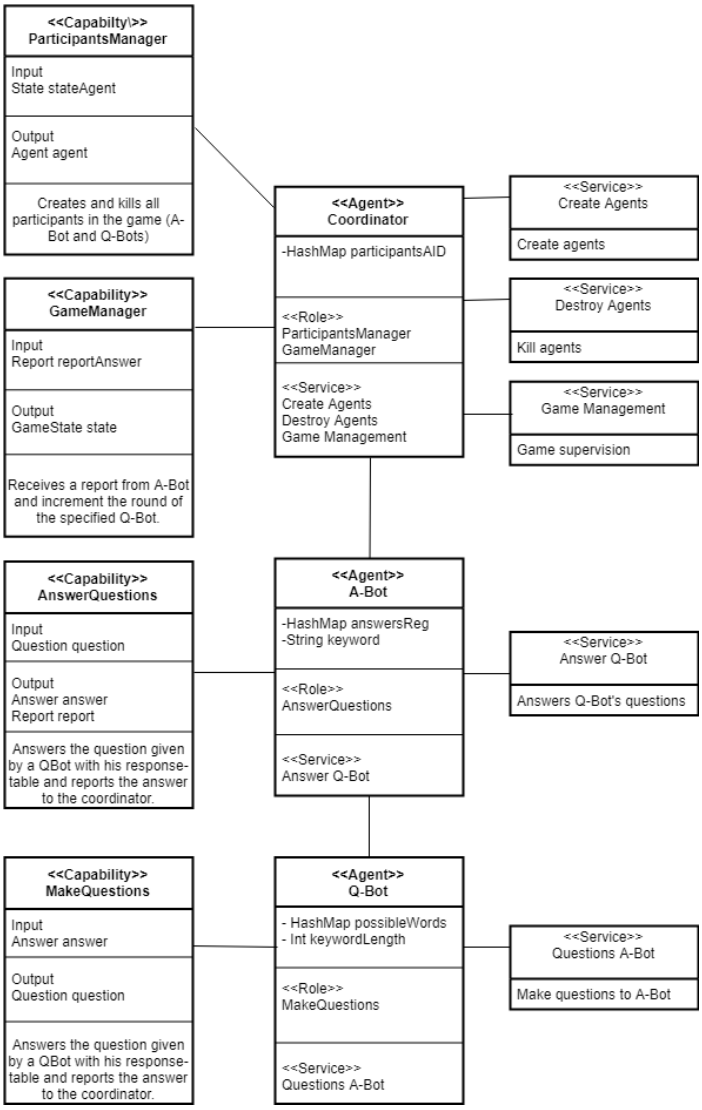

Figure 1: AUML class diagram

In this section the architecture details of the agents in the system are explained, starting with the main roles:

Coordinator: this is the main agent for managing the game. His main resposability is to maintain count of the number of questions answered by the A-Bot to each of the Q-Bots; he also creates and kills all the participants in the game. The Coordinator maintains a hashmap with the keys being the AIDs of the Q-Bots and the values the number of questions answered to that particular Q-Bot by the A-Bot; this way at the end of the game we can determine a ranking giving more points to the agents that resolved the game with the least questions answered. In order to make this management work, the coordinator receives updates from the A-Bot in the form of the message class Report (that will be explained in detail in the A-Bot architecture section).

A-Bot: this bot follows a simple Reflex-agent type of structure. In the moment of birth, he is given a word that will be the keyword the Q-Bots have to guess. With that word he creates a simple hashTable which is his condition-action rules as explained in (Russell and Norvig, 2016). This bot answers the questions from the

Q-Bots using this table in the form of the message class Answer, that has the type of answer given as well as the content. Also, after an answer is sent, the A-Bot forms a Report object message that is sent to the Coordinator; in this message the AID of the QBot the last answer was for is reported and if it was a correct guess of the keyword. Q-Bot: because the implementation of this bot can be very different and can follow a various problem-solving architectures, it is advised to work with the general interface that a Q-Bot in the system needs. The bot has to be able to receive messages from the message class Answer and to send messages from the message class Question to an A-Bot, this class has the type of the question asked and the content. Q-Bot will ask questions to the A-Bot until he receives an Answer object that says the last word guess query was correct; then he should kill himself. Answer-Question types: the answers and questions between the Q-Bots and the A-Bot have two types; a letter query, that are answered with the total amount of ocurrences of that letter in the keyword; and a guess query, that is a complete word question and must be responded with 1 or 0, depending if the guess was correct or no.

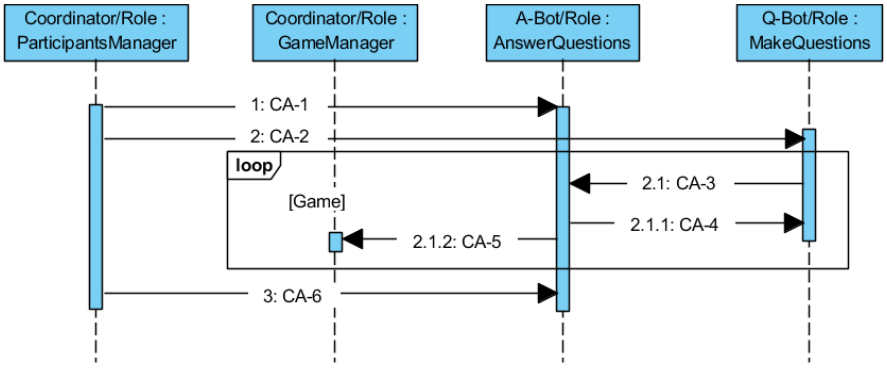

Figure 2: Bots AUML Sequence diagram

The Figure 1 illustrates the AUML class diagram of the system, with the capabilities and services each participant has. As explained, the coordinator must have at least two main roles which are: 1) Participants Manager, that gives the agent the services of creating and deleting agents in the system. 2) Game Manager, where he knows the exact state of the game and is used at the end for ranking the algorithms. The A-Bot is a simple agent with just one role, which is answering the questions he is given to. From his eyes, he doesn't know how many Q-Bots are there or the state of the game, he just gives answers based on the table built at the start and reports each one to the Coordinator. The Q-Bots have the algorithms that are being subject to test. In this particular example, the Q-Bots only knows the existence of the A-Bot, which is the source of all the knowledge they might need. For this reason, the only role the Q-Bots have is to make questions until the problem is solved. In a sense, the Q-Bots can be treated as an intermediary between all the testing system and the algorithm, that uses the service of making questions from the Q-Bot in order to receive new knowledge from the environment.

4. Experiments and Results

Section 4 is divided by two, for an easier explanation and understanding of the experiments made. In Section 4.1, there are the QBot pseudocodes with short paragraphs explaining them and in the Section 4.2 there are explained the two experiments done, using two different datasets which are described as well. These experiments consists on running the multi-agent system built several times. One run is one word to guess, once all the QBots guessed the word the run finishes and starts another one. In the first experiment the words20k dataset is used, in a total of 1000 runs. In the second one the words100k is used and 100 runs were done.

4.1. QBots description

For the experiments and results shown in the following sections, there are two different datasets which are going to be used. Also, there will be four participants named as: QBotA, QBotD, QBotJ and QBotG respectively, for an easier explanation and of course, for a better differentiation. Those are going to be the players, on the other hand, two aditional agents are placed in the game: ABot and CBot. The first one, ABot, it is seen as the link between the QBots and the CBot, understanding by this that he answers QBot questions and send reports to the CBot so he can manage their game rounds. The second one, CBot, has two important roles: 1) participant manager, where he is responsible of loading the dictionary and randomly choosing a word and, after that, he has to create the ABot, who receives the chosen word, and afterwards the QBots. Once he has settled all these required things, he must switch to 2) game manager, where he only waits for ABot reports and updates QBots turns. Agents and roles are better explained in Section 3.

Here, it is listed the algorithms that the agents used for this experiment:

Algorithm 1 QBotJ

```
1: procedure GUESSING ALGORITHM 1
2: loop:
3:     if possibleWords = 1 or noMoreLetters() then
4:         askWord(possibleWords[0])
5:         doDelete()
6:
       ocurrences = askLetter(getNextLetter())
7:     updateList(letter, ocurrences)
```

Figure 3: Pseudocode algorithm 1

QBotJ asks all letters in the dictionary given. He discards words which not contain the asked letter or haven't the same number of occurrences and, in the end, asks for a word when the list size is 1 or doesn't have more letters to ask.

Algorithm 2 QBotA

```
1: procedure GUESSING ALGORITHM 2
2: loop:
3:     if possibleWords = 1 or possibleWords < askConstant and wordMatches() then
4:         if possibleWords = 1 then
5:             word = possibleWords(0)
6:         else
7:             word = matchedWord
8:         if askWord(word) = true then
9:             doDelete()
10:    else
11:        letter = getMostFrequentLetter()
12:        occurrences = ask(letter)
13:        updateList(letter, ocurrences)
```

Figure 4: Pseudocode algorithm 2

QBotA algorithm asks the most frequent letters in the remaining words list. This is very efficient because if that letter isn't contained in keyword, he discards a huge amount of words in the remaining words list. Also, if the keyword contains that letter, all the words which hasn't the same occurrences are discarded. In order to ask

a word, this bot waits until the list size is one, which means that he's got the keyword already, or the list size is less or equal to askConstant (totalWordListSize / 1000) and there's at least one word with no more letters to guess, then he'll choose one of those randomly.

Algorithm 3 QBotD

```
1: procedure GUESSING ALGORITHM 3
2: loop:
3:     if possibleWords >= 2 or possibleWordsHaveSameLetters() then
4:         if askWord(possibleWords[0]) = true then
5:             doDelete()
6:     else
7:         letter = getMostFrequentLetter()
8:         occurrences = ask(letter)
9:         updateList(letter, ocurrences)
```

Figure 5: Pseudocode algorithm 3

QBotD works similar than the last algorithm explained. It has a slight difference, when asking a word, he waits until the remaining word list size is 2 (or less), then it's a 50% chance to success, or until all words have all their letters guessed, which means there's no need to ask for more letters, then he'll start asking those words in order.

Algorithm 4 QBotG

```
1: procedure GUESSING ALGORITHM 4
2: loop:
3:     if checkTotalMatch() < (1.0 - aggressiveness) then
4:         status = askNextLetter()
5:         if status not true then
6:             getNewKeywordBelief()
7:     else
8:         status = tryGuessWord()
9:         if status not true then
10:            getNewKeywordBelief()
11:        else
12:            doDelete()
13:
14: procedure GETNEWKEYWORDBELIEF
15: loop:
16:    word = getRandomNotDiscardedWord()
17:    if checkWordWithCurrentKnowledge(word) then
18:        return word
19:    else
20:        discardWord()
```

Figure 6: Pseudocode algorithm 4

This particular algorithm follows a different general strategy. Instead of maintaining an updated collection of possible words given a knowledge, the bot chooses a word thinking it will be the correct one, then proceeds to ask letters of that particular "belief" word until either an answer discards that word as the good one or the threshold of aggressiveness is surpassed. The aggressiveness is a factor between 0 and 1 that determines the

amount of coincidences between the belief word and the actual knowledge of letter-amount has to have before guessing the word. Being 0 the most conservative (he only guesses the word if all the letters of the belief word has been asked and coincide) and 1 the most aggressive (the bot never ask for a letter, he directly guess his belief). If the answer received from the ABot discards the current belief, then a procedure for choosing another one is triggered. The bot chooses one random word from the dataset that are not discarded and checks if the word has potential to be valid with the current knowledge, if it isn't then it is marked as discarded and another one is chosen randomly from the pool until one valid is reached, that will be the new belief word. This way of choosing a new word is the reason why it eliminates the need of an updated collection of possible words.

4.2 Datasets and results

Datasets are text files which contains words of all kinds, length and characters. Specifically, 'words20k' dataset contains 20.000 different words. They are formed with the letters of the English alphabet (A-Z, 26 letters) and all in lower case. This simple dataset has the perfect amount of words and also free the QBots from a more complex operating mode.

Figure 3 shows the results of the experiment with words20k dataset. It consisted of 1000 rounds, where each round is one different word to guess. From this graphic it can be seen that the trend is for bots to have better performance the longer the word is. QBotA and QBotD have similiar results as expected; although, QBotD is slightly better than QBotA. QBotG has a lot of fluctuations in the performance, especially in shorter words. Finally, it can be seen that QBotJ, while he follows the same trend as the others, it seems to give worse results.

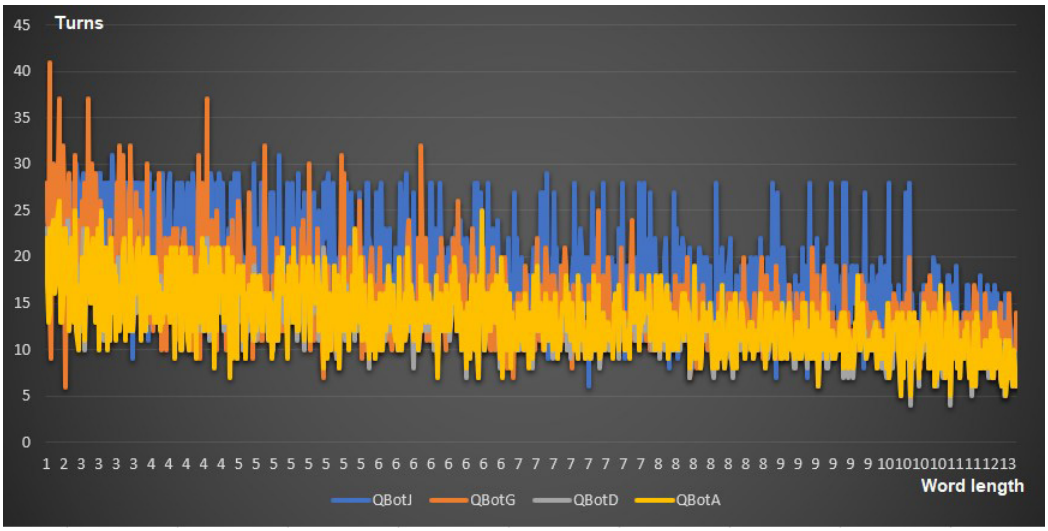

Figure 7: 20k Dataset experiment

Calculating the arithmetic average, it is obtained the results on the Figure 4. It can be observed that the algorithm with some randomness (QBotG) performs slightly worse that QBotA and QBotD. These two algorithms try to be as accurate as possible before starting to ask for words. On the other hand, QBotJ has very poor results because of the simplicity of the algorithm.

QBotD	QBotA	QBotG	QBotJ
12.806	13.428	14.959	18.209

Figure 8: Results words20k

In the second experiment the 'words100k' dataset is used. This one is a little bit bigger, contains 100.000 different words with lower and upper case letters and special characters. Figure 5 shows how this change affected the different algorithms. The global turns average increased, which makes sense. Although, there is something unexpected, QBotJ becomes much worse than before, compared with the rest.

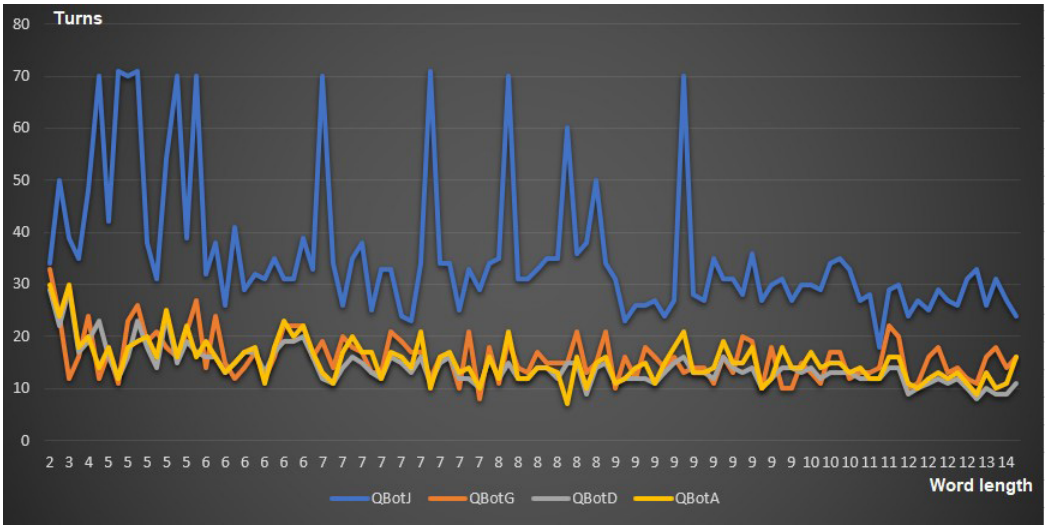

Figure 9: 100k Dataset experiment

More accurate results can be observed in Figure 5. QBotD is still the best so far. QBotA and QBotG are closer in performance this time because QBotG has a slightly better behaviour in this dataset. On the other hand, QBotJ has very bad results this time, because of the increased amount of different characters in the dataset hurts the performance of the algorithm.

QBotD	QBotA	QBotG	QBotJ
14.26	15.37	16.1	35.91

Figure 10: Results words100k

5. Discussion and Conclusion

The reason that motivated this project was creating a system capable of evaluating algorithms given a performance measure. In order to accomplish this task a multi-agent system was used because of the flexibility it grants. As an instance of the system, it has been proposed a word guessing game the algorithms played, with the objective of providing their performance for an analysis to be made. Two datasets have been used, with 20k and 100k words respectively. The words in the smaller one had the 26 English lowercase alphabet letters, and the bigger one also had uppercase letters and symbols.

The results obtained in this paper demonstrates that in deterministic and static environments, stability in the algorithm processes is rewarded, in contrast to randomness. Randomness rises unpredictability in the algorithm, which is good in more competitive environments but lacks efficiency in this particular one. This conclusion was reached through the data the system collected, meaning that the initial approach of this multi-agent system has been proven useful.

All the system works in a deterministic fashion, it could be indeed very interesting to change this by giving the ABot the possibility to answer the questions wrong. This would increase the complexity of the QBots.

Working on this different system, it could happen that the best algorithms in the deterministic approach may not work properly in this new stochastic environment. Regarding to the environment, a possible improvement in the architecture could be made in order to make the system more adaptable to other type of problems, algorithms and performance measurements.

6. References

Agarwal, A., Gurumurthy, S., Sharma, V., and Sycara, K., 2018. Mind Your Language: Learning Visually Grounded Dialog in a Multi-Agent Setting. *arXiv preprint arXiv:1808.04359*.

Antol, S., Agrawal, A., Lu, J., Mitchell, M., Batra, D., Lawrence Zitnick, C., and Parikh, D., 2015. Vqa: Visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pages 2425–2433.

Das, A., Kottur, S., Gupta, K., Singh, A., Yadav, D., Moura, J., Parikh, D., and Batra, D., 2016. Visual dialog. *CoRR abs/1611.08669*.

García, O., Chamoso, P., Prieto, J., Rodríguez, S., and de la Prieta, F., 2017. A serious game to reduce consumption in smart buildings. In *International Conference on Practical Applications of Agents and Multi-Agent Systems*, pages 481–493. Springer.

Goyal, Y., Khot, T., Summers-Stay, D., Batra, D., and Parikh, D., 2017. Making the V in VQA matter: Elevating the role of image understanding in Visual Question Answering. In *CVPR*, page 3.

Johnson, J., Karpathy, A., and Fei-Fei, L., 2016. Densecap: Fully convolutional localization networks for dense captioning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4565–4574.

Kiros, R., Salakhutdinov, R., and Zemel, R., 2014. Multimodal neural language models. In *International Conference on Machine Learning*.

Lu, J., Xiong, C., Parikh, D., and Socher, R., 2017. Knowing when to look: Adaptive attention via a visual sentinel for image captioning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 6, page 2.

De la Prieta, F., Di Mascio, T., Marenzi, I., and Vittorini, P., 2013. Pedagogy-Driven Smart Games for Primary School Children. In *2nd International Workshop on Evidence-based Technology Enhanced Learning*, pages 33–41. Springer.

Russell, S. J. and Norvig, P., 2016. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,.

Vinyals, O., Toshev, A., Bengio, S., and Erhan, D., 2015. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., and Bengio, Y., 2015. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057.

Yao, T., Pan, Y., Li, Y., Qiu, Z., and Mei, T., 2017. Boosting image captioning with attributes. In *IEEE International Conference on Computer Vision, ICCV*, pages 22–29.

Zhang, P., Goyal, Y., Summers-Stay, D., Batra, D., and Parikh, D., 2016. Yin and yang: Balancing and answering binary visual questions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5014–5022.