Autonomous Self-Learning Agents in 3D Virtual Worlds

Evaluating and extending the Q-Cog platform

William Grant
University of Cape Town
grnwil006@myuct.ac.za

ABSTRACT

This paper details the evaluation and extension of the Q-Cog experimental platform. The Q-Cog platform was evaluated through the integration and evaluation of various agent architectures using new performance metrics and testing scenarios. In order create these new performance metrics and testing scenarios an investigation into what is currently used in the field was done. From this the common performance metrics were gathered as a starting point. Once the performance metrics and testing scenarios were implemented formal experimentation was done. The results gathered regarding the various agents were shown to be useful and had significance which verified the use of the Q-Cog platform as an evaluation tool as well as extended it. Additional features and extensions have also been added to the Q-Cog platform and discussed. The purpose of these extensions was to elevate the Q-Cog platform towards being a mature research tool.

CCS CONCEPTS

• **General and reference** → **Metrics**; **Evaluation**;

KEYWORDS

Q-Cog, Unity3D, Metrics, Agent Evaluation, Agent Performance, Artificial Intelligence, Reinforcement Learning, MDP, BDI, POMDP

1 INTRODUCTION

The aim for this paper is to further the vision of for the Q-Cog platform which is to create an independent, pure and generic artificial intelligence testing and simulation platform. This platform facilitates the independent development of artificial intelligence agents that can then be evaluated through simulated testing scenarios. The Q-Cog platform has a lot of potential to be extended and improved in line with this vision. These improvements are outlined in section 7 and mainly fall under network stability and system robustness, error handling as well as the extensions made to the platform. The Q-Cog platform was evaluated through assessing and comparing the performance for a number of different agent architectures.

To aid this endeavour an important objective of this research paper was to investigate what evaluative testbeds/frameworks, testing scenarios and performance metrics are currently used for evaluating self-learning and adaptive agents. This was done in order to design and implement new performance metrics and testing scenarios which were then assessed in order to see whether they can measure an agent's performance meaningfully and accurately. Inspiration was also taken from previous works already done using the Q-Cog platform[19]. The agent architectures used for evaluation[10, 14] were developed by Y. Dollie and J. Hooper independently and then integrated into the Q-Cog Agent module[19]. These agents were then evaluated in comparison with the default Q-Cog agent that is built into the Q-Cog testbed. This agent utilizes reinforcement learning with a dynamic policy selection mechanism[19].

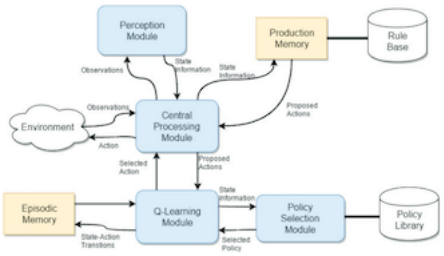

Figure 1: Default Q-Cog Agent Architecture

The agent architectures developed by Y. Dollie used a Belief-Desire-Intention (**BDI**) agent architecture with the primary one using an additional reinforcement learning mechanism (**R-BDI**)[10]. The agent architectures developed by J. Hooper used Partially Observable Markov Decision Processes (**POMDP**) in its design with the primary one utilizing model learning(**PCog**)[14]. Within this work the performance of a self-learning agent refers to the effectiveness of a cognitive agent within a given testing scenario using a means of scoring its performance. For example the score could be based on the time taken for an agent to complete a given goal/task. The secondary objective of this project was to further extend and improve the Q-Cog platform. This involved the development of an executable version of the Q-Cog testbed referred to as Q-Cog Lite and is discussed in section 7.2. This runtime version of the Q-Cog testbed removed the dependency on the Unity3D Game Engine[11]. Q-Cog Lite was designed for researchers who wish to evaluate their own agent architectures using pre-existing testing scenarios and who do not wish to design their own testing scenarios.

2 BACKGROUND

2.1 The Q-Cog Platform

Q-Cog is an experimental artificial intelligence platform designed for developing and testing of adaptive and self-learning agents in 3D environments that are both complex and unpredictable. It is owned by the Centre for Artificial Intelligence Research (CAIR)[2]. It consists of an experimental 3D simulation testbed used for agent development and evaluation and an agent integration module henceforth known as the Q-Cog Agent module[19]. The Q-Cog Agent module is designed using the Java Programming Language and allows an agent architecture to be developed independently from the Q-Cog platform while still allowing it to be tested through the Q-Cog platform. This is done in order to keep the agent architecture as

independent and unbiased as possible from the Q-Cog platform. The Q-Cog testbed was developed using the Unity3D Game Engine[11] and features adjustable simulation settings which can be used to set the number of iterations for a simulation, a simulation speed control mechanism, a data recorder, and finally a playback engine that allows the playback of prior simulations for further analysis whenever the user desires[19].

2.2 Defining Agent Performance, Metrics & Scenarios

Agent performance can be described as how effective an agent is in a given scenario using some means of scoring[6, 18, 20, 22]. The score itself is generally dependent on the complexity of the environment and testing scenario. A classic example of a score would be the time taken for an agent to complete a given goal/task. Agent performance can be further broken down into a collection of specific characteristics related to the agent and its architecture[8, 9, 15, 20–22, 27].
Examples of these characteristics are:

- Planning time
- Action execution time
- Number of actions taken to complete a task/goal
- Relative costs involved when running the agent during a simulation

The relative costs referred to here relate to areas such as sensing the surrounding environment or the cost to plan/replan a series of actions. In order to evaluate the performance of an agent and its adopted learning strategies which is essentially its architecture, a controlled environment is required where experiments are run using appropriate performance measures (metrics) so that the agents learning strategy can be effectively evaluated and possibly compared with others[16]. The environment for this is known as a simulated world and within the context of this research project refers to the environments designed within the Q-Cog testbed[19]. Any experiments that are done to evaluate an agents performance are conducted using the Q-Cog testbed with different testing scenarios that have an underlying goal that the agent is required to attempt. It should be noted that metrics do not strictly refer to an agent's performance but also refer to the measurement of the changes made to the parameters of the environment or actions taken by an agent where applicable[22].

2.3 Interactive Learning and Self-Adaptation

Given the nature of this project, it was important to define the common types of learning found in the field. An agent in this context must be self-learning and adaptive thus it must show some sort of change in performance when performing future tasks after making observations in the world[23]. This section will give a brief introduction to the common types of interactive learning. The concept of learning is often found to be quite ambiguous, this is because it is used in many different areas for problem solving. Bearing this in mind, in this project *learning* will be defined as "a change in a system that allows it to perform better when a task of the system is repeated, or on a similar task is given"[25].

2.3.1 Reinforcement Learning. This is where an agent learns from a series of reinforcements. Reinforcements refer to rewards, punishments or both[23, 25, 26]. The task of reinforcement learning is to use observed rewards to as best as possible learn an optimal policy for an environment[23]. Some examples of reinforcement learning are value iteration, policy iteration and Q-learning. In algorithmic terms reinforcement learning estimates the action-value function through the use of the Bellman equation as an iterative update. Thereafter a reinforcement algorithm will converge to an optimal action-value function[7, 15, 20, 25, 27].

2.3.2 Supervised Learning. Given a training set of example input-output value pairs the agent will learn a function that will approximate the output through mapping inputs to outputs. The final goal is to get the approximate output value to be equal or as close as possible to the desired output. This approximate output is the primary form of evaluation for an agent under this learning mechanism. Supervised learning is useful to problems where the expected outputs are well known[25]. Within supervised learning, learning is defined as a search through the space of possible hypotheses (outputs) for one that will perform well, even when given new data that is beyond the original training set[23]. Some examples of supervised learning can be found are classification and regression and neural networks[23, 25].

2.3.3 Unsupervised Learning. This is where an agent notices patterns within inputs passed down to it while no form of feedback is explicitly supplied. Essentially this means that the agent will learn everything on its own while reaching some sort of convergence criteria with no outside interference to guide it[13, 25]. The most common example of unsupervised learning is clustering. Clustering is where a collection of inputs are grouped (clustered) together based around a chosen property potentially in a useful manner[23]. An additional example of where unsupervised learning is used is neural networks[13, 25].

2.3.4 Deep Learning. Deep learning can be found in reinforcement, supervised and unsupervised learning, it utilizes data representations, hierarchies and neural network architectures to extract high level features from very large datasets through training. Deep learning has been used in various areas of research such as speech recognition, visual processing, robotics, lifelong learning and cognitive agent training[12, 20, 26, 27].

3 REVIEW OF LITERATURE & RELATED WORK

3.1 Existing Testing Frameworks & Platforms

There already exist a number of testing frameworks that are used to evaluate the performance of adaptive and self-learning agent architectures as well as the various learning strategies that an agent may have. Examples of these learning strategies that are assessed can be found under reinforcement, supervised, unsupervised and deep learning as mentioned above. Fundamentally there are two types of testing frameworks/platforms that are used. The first type are designed using their own custom software that create simulated worlds with which agents can interact with[8, 9, 18, 22]. The second type of testing framework make use of existing video game engines

as a foundation that are then further extended to allow for AI research[5–7, 15, 20, 21, 27].

3.2 Common Performance Metrics

Through investigation the most commonly found areas of a cognitive agent that are used for evaluating performance through metrics were time, sensing, planning, deliberation, rate of world change and commitment towards goals. Additionally there were other areas that allowed for measurement of complexity and unpredictable(noisy) elements within a testing scenario. These additional areas can be broken down further into both rate of occurrence and the total number of occurrences. All of the above can be represented as a factor of time[5–7, 9, 15–17, 20–24, 27]. When investigating areas relating to time it had been found that there were issues when evaluating an agent's performance in real time and it was suggested that it would be better to use a simulated clock for the environment instead[18, 22]. Planning was also found to have the possibility of including a subset of actions that were reflexive to changes in the environment and therefore required separate measures for evaluation[9]. Re-planning otherwise referred to as deliberation was shown to be costly on performance and in order to counter one could filter out the possible actions that could be deliberated on which decreased the search space when planning. This indicates that deliberation could have its own set of metrics that are independent of planning[9, 17, 18, 21, 22, 24].

4 PERFORMANCE METRICS

This section details the evaluative metrics that have been added to the Q-Cog platform during this project. At the time of writing the Q-Cog platform has already implemented metrics per simulation iteration relating to the number of predators killed and the success or failure for a given scenario task/goal[19]. Additionally the reasoning behind why certain areas from section 3.2 did not have metrics assigned to them is discussed later in section 9.2.

4.1 Performance Score

The performance score of an agent is a numerical value calculated using the proposed formula below which scores an agent's performance per iteration that can then be averaged to give an overall simulation score at the end of a simulation. This metric gives an overview evaluation of an agent's performance which can be used for preliminary analysis. As the analysis of an agent progresses one can then delve deeper into individual metrics used to calculate the performance score in order to assess the strengths and weaknesses of an agent in specific areas which will indicate where changes should be made to improve performance. This individual metrics are described below in detail.
The formula for calculating a performance score is as follows:

- **G** = Goal Success or Failure per iteration
- **T** = Previous Survival Time / Current Survival Time
- **A** = Successful Attacks / Total Attacks
- **F** = Successful Flees / Total Flees
- **P** = Predators Killed / Number of Starting Predators
- Performance Score $\mathbf{I} = \mathbf{G} + (\mathbf{T} * (\mathbf{A} + \mathbf{F} + \mathbf{P})/3)$
- Simulation Score **S** = **I** / No. of iterations per simulation

4.2 Goal Success

The success or failure for a given scenario task/goal is a numerical value where originally if an agent failed the value was set to 0 and if an agent succeed then the value was set to 100. In this project the value for goal success was changed and was set to 1. While this is a simple change to the metric it is still critical change for calculating the performance score above.

4.3 Survival Time Ratio

In order to record the survival time of an agent per iteration the simulation clock built into the Q-Cog testbed was used. To clear any ambiguity the testing scenarios of this project were solely focused on agent survival and thus we refer to this time metric strictly as survival time. The survival time metric is a cumulative value based on the simulator clock as stated and from that a new cumulative time metric was added to the Q-Cog platform. This proposed survival time ratio (**T**) is represented by the ratio between survival time at the completion of a previous iteration divided by survival time at the completion of the current iteration. The benefit of this metric was that we can immediately see the change in an agents survivability per iteration which can then be used to improve or penalize an agent's performance score.

4.4 Action Ratios

An action ratio refers to the number of successful attempts for a specific action during an iteration divided by the total number of times that the specific action took place during that same iteration. This measure was created because as an agent progresses through a series of iterations it should learn when it is most appropriate and effective to perform an action within a given testing scenario. Therefore whenever an agent fails to perform an action in an effective manner it will then be reflected in it's action ratio which can be used as a performance score penalty. A number of action ratios have been implemented so far and more could be potentially added in future. The action ratios currently implemented within the Q-Cog platform are based on the following actions: the agent attacking (**A**); the agent fleeing (**F**); and the number of predators killed during an iteration (**P**). While killing predators is not strictly an action it can be considered a pseudo action as it is the task that an agent must perform in order to survive in the testing scenario as discussed in section 5.2.

5 DESIGNED SCENARIOS

5.1 The Sandbox

The Sandbox was not strictly designed as the final testing scenario but more of a starting point for attempting to perform integration testing of the various agent architectures[10, 14]. This scenario was deliberately made simplistic with limited features in order to aid with the integration testing process. The purpose of the Sandbox was to make sure that the actions performed by an agent were done correctly as defined within the Q-Cog platform and to see if there were any errors with the commands sent to the agent via the Q-Cog Agent module. Additionally the Sandbox facilitates the testing of new features that are being designed and integrated into the Q-Cog testbed without having to worry about the complexity of

a complete testing scenario interfering during testing. The Sandbox was designed to be a very small 3D world (30 units x 30 units) with only one of any feature or entity at a given time with that an agent can interact with. This lowered the difficulty of goal/task completion for an agent allowing for many quick simulations to be run during the early stages of development and integration of a new agent architecture into the Q-Cog platform.

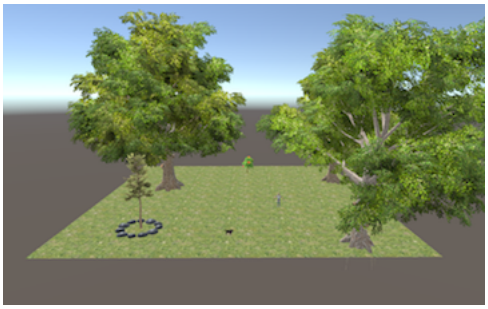

Figure 2: The Sandbox

5.2 The Testing Scenario

Inspired by a previous scenario used in the Q-Cog platform[19] the goal for this scenario was to kill all enemies in the world while attempting to survive. What differentiated this testing scenario from its predecessor were the following factors:

- The terrain of the world was deliberately designed to be more complex in order to obscure visibility for both agents and predators therefore increasing the difficulty of perception and navigation. This included having obstacles such as trees, dunes and hills.
- More food sources were provided in order for an agent to recover health more frequently at any given time giving the agent greater survivability and learning time which increased the richness of the data gathered.
- Two fixed fleeing locations were added so that an agent can flee from an enemy towards the direction of the flee location if it was in range as opposed to just running in the opposite direction of an enemy. These fixed flee locations were chosen to increase pressure on an agent as it had to deliberate between the two possible flee locations while under threat from enemies. It also allowed for a better indication of when an agent had successfully fled from an enemy.
- Six predators of three different types (Normal, Medium Strong) were used. The parameters for the predators had been adjusted to increase randomness and unpredictability within the testing scenario. There are four normal predators as well as one medium and strong predator. The predator's vision, wandering and chasing distance had been increased and the attacking damage and range had been given more variety based on its type. The physical sizes of the predators had also been adjusted based on their type. This enabled the agent to more effectively perceive a dangerous predator since it had a greater chance of being seen.

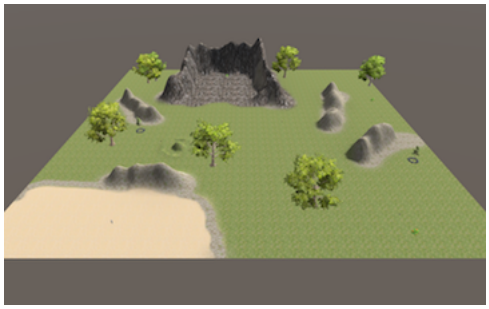

Figure 3: The Testing Scenario

6 EVALUATION APPROACH

The approach for this project was broken up into two distinct phases.

6.1 Phase 1

This phase focused on the integration of the new agent architectures[10, 14] into the Q-Cog platform as well as the development of any extensions that assisted with successful integration of the new agents while being in line with the vision for the Q-Cog platform. The phase followed an iterative design process where any development changes required by Y.Dollie and J. Hooper were prioritised with any improvements that would be useful for research purposes also being added. Throughout this process any errors or bugs found were dealt with as soon as reported in order to not delay moving onto phase 2.

6.2 Phase 2

This phase focused on the implementation of the performance metrics and the testing scenario as well as the final evaluation of the various self-learning agent architectures mentioned in section 1 under experimental conditions. The experimental conditions for the final evaluation were that a minimum of 3 simulations had to be run with each simulation executing 20 iterations. The results from the experiments were then statistically analysed and compared by Y. Dollie and J. Hooper and any observations they had during the experiment were recorded and then later discussed. The statistical analysis involved a comparison of not only the simulation scores for the various agent architectures but also an analysis of the individual metrics that are mentioned in section 4. Furthermore with these results Y. Dollie and J. Hooper could analyse any areas that they felt were significant. If the results gathered were found to be accurate and meaningful through the use of both the testing scenario and the performance metrics then this project can be considered a success.

7 PHASE 1 RESULTS

This section covers the development and integration of the agent architectures designed by Y. Dollie and J. Hooper[10, 14] as well as the improvements, changes and extensions made to Q-Cog platform. Additional details can be found in the found in the Q-Cog platform repository[1]. This includes code documentation and all minor changes made to the Q-Cog platform throughout the duration of this project.

7.1 System Improvements

During the integration process of the new agent architectures[10, 14] the most critical area that needed to be assessed for stability and possible improvement was the Q-Cog Agent Module which functions as both a framework for agent development & integration but also as a bridge for communication between an agent architecture and the Q-Cog testbed. During testing it was found that the network communication interface had not been designed to handle network errors and client disconnections effectively.

The first issue was solved by outputting error messages when network errors arose. The second issue was dealt with by immediately closing the network connection if either side of the network interface disconnected at any time. This was done because a simulation would then be considered corrupted as there was no point in continuing a simulation if network commands were not being sent.

An additional issue found was that communication between the Q-Cog module and testbed was not synchronous. This meant that the module would continuously send commands across the network connection interface without any form of feedback when activities such as action completion, agent death, or task/goal success occurred. This issue was resolved through the addition of completion flags that were sent back to the module from the testbed when any of the above events had occurred. The completion flags enabled Y. Dollie and J. Hooper to improve their agent architecture's overall performance across the network by drastically reducing the network packet overhead that occurred when attempting to run an agent since the number of packets sent had drastically decreased. During the assessment and further development of the Q-Cog platform a number of improvements and extensions were made to the existing functionalities & tools available. They were as follows:

Figure 4: The Advanced HUD

- The simulation information display now has an added advanced mode that can be toggled which details more in-depth information concerning the simulation, which entities are currently still in the world and the network traffic between the module and testbed.
- A simulation eventlog was created that stored the network messages sent between the module and the testbed as well as feedback when an agent interacted with any objects in the simulated world. All logged events had a timestamp attached based on the simulator clock time.
- All output files from the Q-Cog platform has been formatted to a standardized style and the simulation results are now saved using the .csv file format to enhance readability and compatibility with various software programs for statistical analysis (i.e. Microsoft Excel, LibreOffice and R-Studio). Furthermore all output files are now named and stored in a unique folder based on the simulation's number.
- The simulation playback engine[19] was extended to include not only the position of an entity during a simulation but also its actions and health changes giving a more informative simulation playback.

7.2 Q-Cog Lite

Since the aim for the Q-Cog platform is to create an independent, pure and generic artificial intelligence testing and simulation platform which can evaluate adaptive and self-learning agents. There should be little or no biases or dependencies within the Q-Cog platform. Originally the Q-Cog testbed had a dependency on the Unity3D Game Engine in order to be used. This can be seen as a severe limitation for potential users, particularly for users on Linux distributions due to the lack of Unity3D support. The solution to this problem was the development of a runtime version of the Q-Cog testbed. This Q-Cog runtime version which is an executable of the Unity3D project henceforth is referred to as Q-Cog Lite. Q-Cog Lite allows researchers to run simulations using pre-designed testing scenarios with configurable parameters without the need for the Unity3D Game Engine[11].

Figure 5: Main Menu

7.2.1 *Q-Cog Lite Features.*

- No need for the Unity3D Game Engine to run the Q-Cog testbed.
- Cross-Platform support for Windows, macOS & Linux distributions.
- A Main Menu where testing scenarios can be selected.
- A Settings Menu to configure simulation parameters such as: number of iterations per simulation; maximum health predators and the agent; set the simulation recording you wish to play using the playback engine, and finally which port you want the TCP network connection to be on to connect to the Q-Cog Agent Module.
- A Loading screen when waiting for the Q-Cog Agent Module to connect.
- Two testing scenarios that can run an external agent architecture or the default Q-Cog agent.

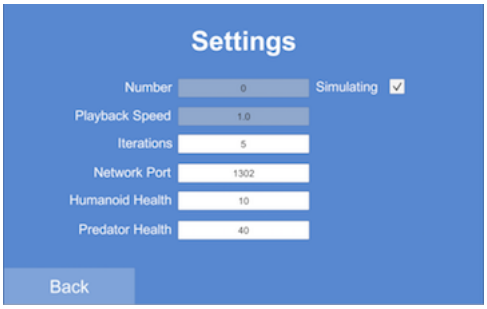

Figure 6: Settings Menu

7.3 Q-Cog Documentation

Given the inherent complexity of the Q-Cog project, easily accessible code documentation was a critical requirement that had to be addressed. With this in mind an important deliverable of this project was to have comprehensive and accurate documentation for the existing classes in the Q-Cog platform. This documentation also had to include instructions for using the testbed and for setting up an external agent architecture that would integrate with the Q-Cog Agent module. This has been successfully achieved through the use of Doxygen, a documentation generation tool used for a variety of programming languages[3], and a specialized Unity3D package that utilizes Doxygen.

7.3.1 Doxygen Unity3D Package. Using a *.unitypackage* created by Jacob Pennock[4] a documentation website was generated for all the C# scripts used within the Q-Cog testbed and additional setup instructions for the Q-Cog platform were added.

Figure 7: Q-Cog Documentation Website

8 PHASE 2 RESULTS

8.1 R-BDI Agent Results

Using the results produced by the Q-Cog testbed statistical analysis was performed on the agent's performance score, the number of predators killed and the survival time per iteration in order to determine if there was any statistical significance between the various agent architectures. All of the statistical tests were carried out using RStudio. The results indicated that on average the R-BDI agent had the best performance score with the highest number of predators killed. Furthermore the average survival time was better than all the other agents with the only exception being the default Q-Cog agent[10]. The averages of the results for the various agent architectures can be seen below as well as which were found to have statistical significance in comparison to the R-BDI agent.

Table 1: Agent Results on Average

Agent	Predators Killed	Survival Time	Score
R-BDI	5.4	236.1	1.628813
BDI	5.2	323.9	1.52148
POMDP	5	300.35	1.190819
MDP	2.3	178	0.7691907

Table 2: Statistical Significance between Agents

Comparison	Predators Killed	Survival Time	Score
R-BDI vs BDI	-	-	-
R-BDI vs POMDP	-	-	Significance
R-BDI vs MDP	Significance	-	Significance

8.2 PCog Agent Results

Using the results produced by the Q-Cog testbed statistical analysis was performed on the same areas of interest as in Y. Dollie's approach[10]. The PCog agent architecture was compared with a differing set of agent architectures compared to Y. Dollie which included a manually defined POMDP agent (**spCog**) and an additional agent architecture that picks actions at random (**RCog**). The results indicated that the PCog agent successfully outperformed the other agent architectures being tested with the exception of the BDI agent architecture (referred to as BCog in the work)[10] which had the best overall performance[14]. The averages of the results for the various agent architectures in comparison to the PCog agent can be seen below.

Table 3: Agent Results on Average

Agent	PCog	SPCog	BCog	QCog	RCog
Predators Killed	4.75	4.45	5.01	2.5	0.1333
Survival Time	257.4	308.11	283.61	206.15	87.86
Score	1.22	1.034	1.545	0.77	0.43

9 DISCUSSION

9.1 Agent Integration

It has been shown that the various agent architectures implemented by both Y. Dollie and J. Hooper were successfully integrated into the Q-Cog platform and the results outputted by the testbed matched with what was seen while running the experiments with regards to responsiveness and performance in comparison to the default Q-Cog agent. An important point discussed by Y. Dollie was that random chance cannot be completely ruled out when assessing an agent architecture's performance given the unpredictable nature

of the testing scenarios involved[10]. Both Y. Dollie and J. Hooper made use of Q-Cog Lite for development and evaluation of their respective agent architectures allowing them to make use of the Q-Cog platform without the need to install the Unity3D Game Engine which improved the overall efficiency and speed of the integration process. This also avoided setting up the project from source code negating possible dependency and corruptions issues that often occur with video-game based projects that make use of version control. Given that the Q-Cog platform is not a mature project and is still very much in the early stages of development there were a two major limitations found within the platform that are listed below.

- The default Q-Cog agent which had been used as a foundation for agent development had been designed for specific use cases and testing scenarios that were created previously. This could be seen as contradictory to the vision for the Q-Cog platform. What this means is that the agent logic during a simulation was not as generic and decoupled as it should be which did slow down the integration process as previous methods and agent logic had to be updated continuously during the integration process.
- The only available goal/objective for an agent was to kill all the predators and to survive. Work has been done during this project to further extend the possible goals/objectives that an agent could be assigned to do in a testing scenario but the integration process using the existing functionalities of the Q-Cog platform was prioritised. The work that had been started to resolve this issue was an additional agent goal/objective using a key/door system where an agent had to find a key in it's surroundings and take that to a door while fighting off enemies in order to successfully complete an iteration but this functionality is still immature and has not yet been tested.

9.2 Performance Metrics

During the evaluation of the performance metrics and testing scenarios both Y. Dollie & J. Hooper had noticed areas of improvement for their respective agent architectures by analysing the individual performance metrics and thereafter updating their code accordingly. Particularly in areas related to an agent attacking or fleeing a large number of improvements were made during project based on how the agent interacted with the environment and from the action ratio performance metric results which indicated if their agents were using actions correctly and effectively within the testing scenario.

During the design and development of the performance metrics there were areas where additional performance metrics could have been added based on the investigation of current works and literature in section 3.2. Unfortunately there were certain limitations and considerations that had to be taken into account during the project. One of the primary reasons behind the implementation of the current performance metric collection was due to time awareness required so that both Y. Dollie & J.Hooper had sufficient time allocated for performing experiments in order to get enough useful data for their analysis of their respective agents. Therefore certain areas of evaluation such as time, agent actions, predator kills and overall task/goal success were prioritised over other areas such as sensing, deliberation and planning. In particular with regards to sensing due to the nature of the perception engine in the Q-Cog platform[19].

The perception engine creates perceptions of an agent's environment continuously which adds an additional complexity when attempting to design a meaningful metric for an agent as it perceives. In addition when designing the performance metrics it was important that given the vision for the Q-Cog platform to design the performance metrics to be as generic as possible and to have as little reliance on an agent's underlying architecture as can be done. The reason being that analysing areas relating to planning and deliberation through an agent's architecture would create a dependency on an agent's architecture in order to generate data for the desired performance metric which would have to be accounted for when developing a new agent architecture. The performance metrics implemented currently in the Q-Cog platform are generated strictly using the Q-Cog testbed data and do not rely on an external agent's architecture. Both Y. Dollie and J. Hooper found statistical significance in agent performance in comparison with the default Q-Cog agent which indicated that the score metric was meaningful as well as successful for measuring an agent's performance within the testing scenario. However an interesting observation was made that the individual metrics used to calculate the performance score had no statistical significance between agents.

9.3 Testing Scenarios

The Sandbox used for integration and testing during phase 1 had been approved by both Y. Dollie and J.Hooper given usefulness of having a low complexity testing scenario that would facilitate quick and easy testing at the early stages of development. The agent integration testing could be done on a feature by feature with minimal effort until both agent architecture designers were confident enough in their agents to perform formal experiments. This indicated that the Sandbox was successful in performing its given function. The testing scenario for formal experiment had a sufficient level of difficulty and complexity that offered insight into the improvements and changes that could be made to both Y. Dollie and J. Hooper's agent architectures[10, 14].

The testing scenario can then be considered successful since the various agent architectures were able to learn and adapt effectively to the scenario given enough time. It should be noted that Y. Dollie did query the suitability of the testing scenario for agent learning given the results produced for the agent architectures based on reinforcement learning mechanisms[10]. However as stated overall the testing scenario can be seen as successful given that the agents were able to successfully complete the given task at least 1 or more times during the early stages of testing and in the later stages the agents where achieving higher levels of goal success of up to 50 percent. For future researchers it is important to be aware that meaningful data can and should still gathered even if an agent has failed it's given task/goal in a simulation since this can used to further improve an agent's underlying performance given the extensions made to the Q-Cog platform.

10 CONCLUSIONS

In conclusion the improvement and extension of the Q-Cog platform has been a success given the successful development, integration and evaluation of the various new agent architectures designed by both Y.Dollie and J. Hooper. As stated in section 1 large improvements have been made to overall network stability, system robustness, system performance and usability particularly through the improvements made to not only the network communication interface between the Agent Module and the testbed but also through the introduction of Q-Cog Lite. The performance metrics and testing scenarios that have been implemented have been shown to be meaningful as well as being a successful addition to the Q-Cog platform for evaluating an agent architecture's performance. However while some of the individual performance metrics were shown to not have statistical significance between agent architectures they were still crucial for the overall improvement of the various agent architectures during the early stages of development and testing. Finally while the Q-Cog platform is still in its early stages of development it has been shown to be an effective research tool for adaptive and self-learning agent development and evaluation with the improvements and extensions made here only further increasing its usefulness in the artificial intelligence community.

11 FUTURE WORK

There are a number of areas for future work in order to elevate the Q-Cog platform towards being a mature research tool. These areas are as follows:

- There exist a number of features and functionalities within the Q-Cog code base that are not utilized in the currently implemented testing scenarios and the implementation of them could be seen as a considerable portion of future work.
- Given that a large focus of this project was the implemented performance metrics there is still a lot of possible research that can be done towards expanding the use of action ratios in order to improve agent architecture development and integration while improving the accuracy of the agent performance score.
- The agent framework within the Q-Cog platform is still heavily dependent on the previously designed testing scenarios and therefore there is a need to make it more generic in order to be an effective foundation for the development and integration of new agent architectures. By doing this new agents can be designed and implemented with minimal corrections made to the existing code base.
- There is a need extend the Q-Cog testbed to support varying agent goals/tasks that can then be set and used in designing testing scenarios. An example of this would be the key/door functionality that has been implemented.
- There is a need for the development of more testing scenarios that differ from the existing ones particularly with regards to goals/objectives. This is in order to improve agent learning and performance by using a collection of unique testing scenarios which would give a more robust evaluation of an agent architecture.
- The network communications that occur between the Agent module and the testbed can be further optimized in order to reduce network packet overhead in order for the Q-Cog platform to run with better performance and lower system resource usage.
- Finally, Q-Cog Lite has great potential to be further extended so that it gives more control over simulation parameters and possibly to allow for the initialization of the Q-Cog Agent module without the use of the command-line.

12 ACKNOWLEDGEMENTS

Thanks to the primary developer of the Q-Cog Platform, Michael Waltham. Gratitude to my supervisor Deshen Moodley for his guidance throughout the duration of this project. Finally, special thanks to Yusri Dollie and Jonah Hooper for their contributions towards this project.

REFERENCES

[1] 2017. Autonomous self-learning agents in 3D virtual worlds. (2017). https://people.cs.uct.ac.za/~GRNWIL006/

[2] 2017. Centre for Artificial Intelligence Research CAIR. (2017). http://cair.za.net/

[3] 2017. Doxygen. (2017). http://www.stack.nl/~dimitri/doxygen/index.html

[4] 2017. Unity Automatic Documentation Generation. (2017). http://www.jacobpennock.com/Blog/unity-automatic-documentation-generation-an-editor-plugin/

[5] Rogelio Adobbati, Andrew N. Marshall, Andrew Scholer, Sheila Tejada, Gal A. Kaminka, Steven Schaffer, and Chris Sollitto. 2001. Gamebots: A 3d virtual world test-bed for multi-agent research. In *Proceedings of the second international workshop on Infrastructure for Agents, MAS, and Scalable MAS*, Vol. 5. Montreal, Canada.

[6] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. 2013. The Arcade Learning Environment: An evaluation platform for general agents. *J.Artif.Intell.Res.(JAIR)* 47 (2013), 253–279.

[7] Chris Brown, Peter Barnum, Dave Costello, George Ferguson, Bo Hu, and Mike Van Wie. 2004. Quake ii as a robotic and multi-agent platform. *Robotics and Vision Technical Reports,[Digital Repository],(2004 Oct.), Available at HTTP: http://hdl.handle.net/1802/1042* (2004).

[8] Paul R. Cohen, Michael L. Greenberg, David M. Hart, and Adele E. Howe. 1989. Trial by fire: Understanding the design requirements for agents in complex environments. *AI magazine* 10, 3 (1989), 32.

[9] Paul R. Cohen, Michael L. Greenberg, David M. Hart, and Adele E. Howe. 1990. Real-time problem solving in the Phoenix environment. (1990).

[10] Yusri Dollie. 2017. Development and Performance Comparison of a BDI Agent Utilizing Reinforcement Learning. Unpublished manuscript. (2017).

[11] Unity Game Engine. Unity Game Engine-Official Site. *Online][Cited: October 9, 2008.] http://unity3d. com* (????), 1534–4320.

[12] Ben Goertzel. 2014. Artificial general intelligence: concept, state of the art, and future prospects. *Journal of Artificial General Intelligence* 5, 1 (2014), 1–48.

[13] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2009. Unsupervised learning. In *The elements of statistical learning*. Springer, 485–585.

[14] Jonah Hooper. 2017. POMDP learning in partially observable 3d virtual worlds. Unpublished manuscript. (2017).

[15] Matthew Johnson, Katja Hofmann, Tim Hutton, and David Bignell. 2016. The malmo platform for artificial intelligence experimentation. In *International joint conference on artificial intelligence (IJCAI)*. 4246.

[16] D. Kinny and M. George. 1991. Commitment and Effectiveness of Situated Agents. In *Proceedings of the twelfth international joint conference on artificial intelligence (IJCAI-91)*. 82–88.

[17] David Kinny, Michael Georgeff, and James Hendler. 1992. Experiments in optimal sensing for situated agents. In *Proceedings of the Second Pacific Rim International Conference on Artificial Intelligence*. 1176–1182.

[18] Michael Lees. 2002. A history of the Tileworld agent testbed. *School of Computer Science and Information Technology, University of Nottingham, Nottingham* (2002), 2002–2001.

[19] Waltham Michael. Design and implementation of the Q-Cog Architecture. Unpublished manuscript. (????).

[20] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[21] Christopher Moriarty. 2007. Learning Human Behavior from Observation for Gaming Applications. (2007).

[22] Martha E. Pollack and Marc Ringuette. 1990. Introducing the Tileworld: Experimentally evaluating agent architectures. In *AAAI*, Vol. 90. 183.

[23] Stuart Rusell and Peter Norvig. 2003. *Artificial Intelligence: A Modern Approach* (third ed.). Prentice Hall. ID: Rusell-Norvig03.

[24] Stuart Russell and Eric Wefald. 1991. Principles of metareasoning. *Artificial Intelligence* 49, 1-3 (1991), 361–395.

[25] Christos Sioutis. 2006. *Reasoning and learning for intelligent agents*. Ph.D. Dissertation. University of South Australia.

[26] Chen Tessler, Shahar Givony, Tom Zahavy, Daniel J Mankowitz, and Shie Mannor. 2016. A deep hierarchical approach to lifelong learning in minecraft. *arXiv preprint arXiv:1604.07255* (2016).

[27] Hiroto Udagawa, Tarun Narasimhan, and Shim-Young Lee. 2016. Fighting Zombies in Minecraft With Deep Reinforcement Learning. (2016).