**CHAPTER I INTRODUCTION**

The introduction consists of the Background, Problem Statement, Research Objectives, Scope and Limitation, Research Methodology and Thesis Outline. The Background explains the current condition of the technology and information accessibility and the main problem that is created by the accessibility. The Problem Statement describes concisely some issues that is solved by the application.

**1.1 Background**

To prosper the quality of human’s life, human began to develop a man-made intelligence, or what we usually called *Artificial Intelligence (AI)*. The long development goal of AI is to achieve the ability for the machine to think and act both rationally and humanlyin solving any intellectual human task, which is called *Artificial General Intelligence (AGI)*. In this era, the recent development of AI is in creating an AI that could both think and act rationally.

AI become a popular technology trends in business that enables human to predict, cluster and classify data. Copeland (2016), a Silicon Valley Journalist writes for NVIDIA, the biggest graphic card companies, believes that AI will not only be utilized by technology-driven businesses such as Google, Microsoft, and Amazon. However, another business fields such as sports, oil, personal loans, and other companies will also utilizes AI to help them wins the business. One example is mentioned by Copeland in the article created by Caulfield (2015), NVIDIA’s chief blogger, which report a beer’s business that utilizes machine learning to help the craft brewers crafting a better beer by gaining a knowledge from their customers.

Google DeepMind and OpenAI are companies that shows AI potential in solving problem that can be trained in simulated environment. RL is utilized by these companies to train an expert agent that outperforms humans in game. The agent created by DeepMind, the *AlphaGo Zero*, is able to defeat the 18-time world champion Lee Sedol in the game of Go (“The Google DeepMind Challenge Match”, n.d). OpenAI agent is also able to defeat the three best Dota 2 player in the world in 1v1 match and it puts a tough battle in five bots versus five players mode (“OpenAI Five”, n.d.).

By this achievement, the author deduct that using game as the environment to train an agent is the first step in solving complex real world problem. For example, researcher that would want to create a self-driving car could create an agent which excel in playing racing game. Strategic games also exist in the business such as the strategy to wins marketing, stock exchange, etc.

Seeing the potential of RL, in this study, the author implement *Reinforcement Learning (RL)* algorithms to create an agent that is excel in playing *Atari games*. The training results will be an agent that is excel in playing different kinds of Atari games environment.

**Problem** **Statement**

Kuder, D., Hans, S., & Mittal, N. (2019) stated that AI will become a powerful business tools that could help people to win at business. In order to studies the recent surge of AI trends in business, the author intent to studies and compares different kinds of *reinforcement learning* algorithm which can train an agent to learn and interact within the specified environment to reach a specified goal.

**Research Objective**

This study is conducted with the following objectives:

1. To implement reinforcement learning as the machine learning algorithm in creating an agent that could outperform human in Atari games.
2. To analyse the algorithm performance in the simulated environment to achieve the best result.
3. To provide a reference materials for students who are interested in this field that would become a base for a more suitable learning process to solve real world problem.

**Scope and Limitation**

This study was conducted using Q-learning algorithm (and Deep Q Network algorithms) in the Atari games environment. The aspects looked into are the implementation of the algorithm and the agents performance in different environments (a specific environment).

**Methodology**

The author use the Waterfall Model of software development process. In this model, the whole software development process are divided into seven phase where the outcome of one phase will become the input for the next phase as shown by Figure 1. The seven phases in this model are:

* **Requirement Gathering and analysis** − All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.
* **System Design** − the requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.
* **Implementation** − with inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.
* **Integration and Testing** − All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
* **Deployment of system** − Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.
* **Maintenance** − To fix the appearing issues and enhance the product.

**Thesis Outline**

The thesis consists of seven chapters, which are:

1. Chapter I: Introduction

Introduction consists of Thesis Background, Problem Statement, Research Objective, Scope and Limitation, Methodology, and Thesis Outline.

1. Chapter II: Literature Study

Literature Study describes the theoretical basis of references and guidance in the thesis creation.

1. Chapter III: System Analysis

System Analysis describes the analysis of the program – its behaviour and function. It consists of System Overview, Hardware and Software Requirement, Use Case Diagram, Use Case Narrative, and Activity Diagram.

1. Chapter IV: System Design

System Design describes the definition of the program’s architecture, components, and modules. It defines User Interface Design, Physical Design, Data Design, and Class Diagram of the program.

1. Chapter V: System Implementation

System Implementation describes how the application is implemented. It consists of User Interface Development and Application Details.

1. Chapter VI: System Testing

System Testing contains the testing documentation of the application. Included here are Testing Environment and Testing Scenarios, along with the results.

1. Chapter VII: Conclusion and Future Work

This chapter contains conclusion of the research. It also describes possible future improvements in section Future Work.

**CHAPTER II** **LITERATURE STUDY**

**CHAPTER III SYSTEM ANALYSIS**

System Analysis explains the software – both in its function and behavior – in order to satisfy the prescribed requirements. This chapter consists of six sub chapters that will fulfill the system analysis process which are System Overview, Functional Analysis, Hardware and Software Requirement, Use-Case Diagram, Use-Case Narrative, and Activity Diagram.

**3.1 System Overview**

This thesis is intended to implement Deep Q Network Algorithm to train an agent in playing Atari game. The system will train the agent to the point that the agent is able to exploit the environment to achieve the best score as fast as possible. The main objective of the system is to create a system that were able to outperform human in playing games.

**3.2 Functional Analysis**

Functional analysis is the next step in the Systems Engineering process after setting goal and requirements. Functional analysis divides a system into smaller parts, called functional elements, which describe **what** we want each part to do. We do not include the **how** of the design or solution yet. At this point we don't want to limit the design choices, because it might leave out the best answer. In later steps we will identify alternatives, optimize them, and select the best ones to make up the complete system. The name **Function** comes from mathematical functions, which act on an input value and produce a different output value. Similarly, in the Systems Engineering method, functions transform a set of inputs to a set of outputs.

Table 2 Functionality Table for Safe Hooker

No Function Description

1 Able to detect mature content on the website

2 Able to prevent access to the website if the mature content is detected

3 Able to generate email report

**3.3 Hardware and Software Requirement**

Listed below are the software and hardware that are needed to developed this application:

1. Personal Computer

A personal computer is where the application resides and be developed, from its earliest stage to its deployment. This application is developed on a PC running 64-bit Windows 10 education.

1. JetBrains PyCharm Professional 2019.2.1

PyCharm is a Python Integrated Development Environment (IDE) that can be downloaded for free with a student email. Developed by a Czech company, JetBrains, PyCharm provides easy code analysis, jupyter notebook supports, an integrated unit tester, integration with version control systems (VCS), and supports web development with Django.

1. Microsoft Office

Microsoft Office application, specifically Microsoft Word is used in the making of the application’s documentation.

1. Python 3.7.4

Python 3.7.4 is the programming language that is used in the development of the application. Some APIs listed below are used to support the development of the application:

* + Matplotlib, a python 2D plotting library which can produce a variety of plot and histogram which are interactive and cross platform.
  + Tensorflow, Google’s open source machine learning platform that could deliver an easy yet robust machine learning model.
  + Gym, OpenAI’s toolkit for developing and comparing a variety of reinforcement learning algorithm. Using Gym also allows the researchers to create and define their own environment to test their algorithm.

**3.4 Use-Case Diagram**

Use case diagram is a diagram describing dynamic behavior or technical concept of a system. It defines:

1. Actors

The internal or external factors interacting with the system depicted by stickman

1. Use Cases Nodes

A sequence of actions that provide something of measurable value to an actor depicted by circle.

1. Associations Arrow

Dotted arrows which explains a node dependency toward another node.

1. Unidirectional Associations Arrow

Arrow which explains a node association toward another figures.

The application’s use case diagram is shown in Figure 1.

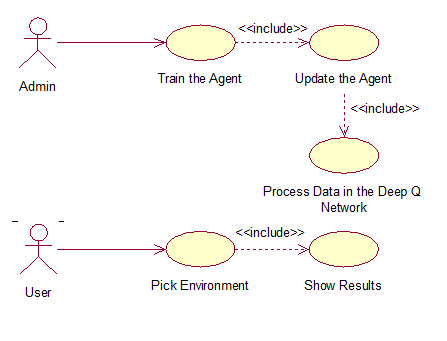


Figure 1. The Use Case Diagram

**3.5 Use-Case Narrative**

A detailed textual representation of the sequence of events occurred during the interaction between an actor and the system in use case diagram will be explained in the narratives. The narrative is a table that will explains the prerequisites, conditions, expected results, alternative scenario, exception, goal, and post condition of a use case node. Use case narrative aims is to clarify the system’s behaviors from the early stages.

Table 1. Use Case Narrative of Train the Agent Node

|  |  |
| --- | --- |
| Use Case Name | Train the Agent |
| Goal in Context | The accuracy of the Agent is increased |
| Primary Actor | The admin |
| Secondary Actor | None |
| Precondition | The environment must exist |
| Trigger | The admin initializes the environment |
| Scenario | 1. The admin initializes the environment 2. The admin starts the training process 3. The application train the agent to maximize the reward it could get 4. The training results are recorded and plotted into a chart |
| Alternate Scenario | None |
| Exception | The environment is not exist |
| Post Condition | The training data are saved |

Table 2. Use Case Narrative of Process Data in Deep Q Network Node

|  |  |
| --- | --- |
| Use Case Name | Process Data in Deep Q Network |
| Goal in Context | The environment data and the agent knowledge are updated |
| Primary Actor | The admin |
| Secondary Actor | None |
| Precondition | The agent is initialized |
| Trigger | The agent is taking an action that changes the environment |
| Scenario | 1. The agent is taking an action that changes the environment 2. Both of the environment and agent data are processed into the Convolutional Neural Network 3. The agent knowledge base is updated from the retrieved data |
| Alternate Scenario | None |
| Exception | None |
| Post Condition | The agent got the knowledge from the past experience enables it to learn with more accuracy |

Table 3. Use Case Narrative of Update the Agent Node

|  |  |
| --- | --- |
| Use Case Name | Update the Agent |
| Goal in Context | The agent making a progress in every episode |
| Primary Actor | The admin |
| Secondary Actor | None |
| Precondition | The agent is initialized |
| Trigger | The training to a certain iteration is started by the admin |
| Scenario | 1. The training is started by the admin 2. The agent is initialized with the default environment information 3. The agent takes random action in the environment 4. The action updated the environment which puts the agent in a new environment 5. The agent stores the knowledges and continue taking random action |
| Alternate Scenario | None |
| Exception | If the agent exploits the short-term reward and do not seeks a long-term reward, the agent would not get the best performance. |
| Post Condition | The agent made a progress in making a decision until it get the best performance or until it reaches the end of the iterations. |

Table 4. Use Case Narrative of Pick Environment Node

|  |  |
| --- | --- |
| Use Case Name | Pick Environment |
| Goal in Context | The environment is initialized |
| Primary Actor | The user |
| Secondary Actor | None |
| Precondition | The application is running |
| Trigger | The user chooses a specific environment from the combo box |
| Scenario | 1. The user chooses an environment 2. The environment is initialized |
| Alternate Scenario | None |
| Exception | None |
| Post Condition | The environment is shown to the user |

Table 5. Use Case Narrative of Show Results Node

|  |  |
| --- | --- |
| Use Case Name | Show Results |
| Goal in Context | The training results is shown to the user through a chart |
| Primary Actor | The user |
| Secondary Actor | None |
| Precondition | The user has chosen a specific environment |
| Trigger | The environment is initialized |
| Scenario | 1. The environment is shown to the user 2. The results of the agent’s training are shown through a chart |
| Alternate Scenario | None |
| Exception | None |
| Post Condition | The result chart is shown to the user |

**3.6 Activity Diagram**

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. The diagram defines both concurrent and sequential activities which is used by the developers as a line up of the technical flow of their works. This sub chapter will explain the activities diagram that are used in building this application.

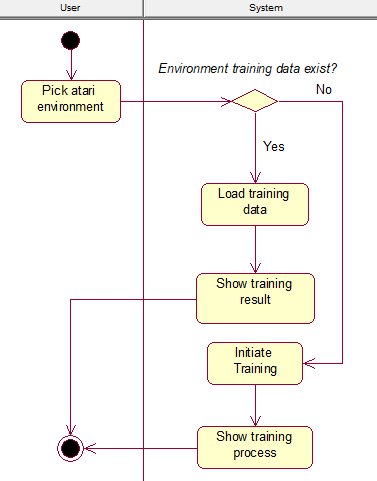


Figure 2. Pick Environment Activity Diagram

The Figure 2 above is derived from the *Pick Environment* node of the Use Case Diagram. The user who picks the environment will trigger the system to check whether a training data is exist in the environment. If the training already conducted, then the application will simply show the training result to the user. However, in the case where a training is not yet conducted, then the training will be initiated and the process will be visible to the user through the chart generated on the fly when the training is running. The *Initiate Training* node itself is explained in more detail through the Figure 3 below.

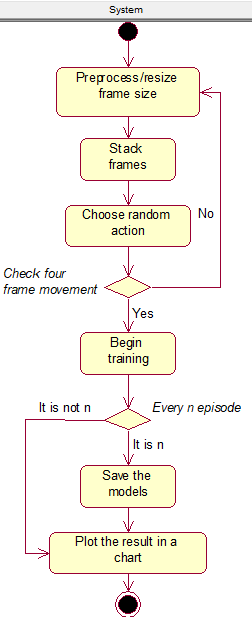


Figure 3. Initiate Training Activity Diagram

The system will be the sole actor of the *Initiate Training Activity Diagram*. To begin the training, the system will first *Preprocess/Resize* the environment’s frame to delete meaningless details (extra spaces, scoreboard, etc.). Then, the next process is to *Stack Frames* so that the system could get the sense of motion. Next, the system will initiate the agent to take a *Random Action* to update the environment. If the system did not detect an update to the frames in *n-steps*, it will restart the process all over again from the *Preprocess* node. Otherwise, the system will *Begin the Training Session* that will be furtherly explained in the next section that is depicted by Figure 4. The last steps that will end the training session are to *Save* the trained model for each defined step and *Plot* the training results into the chart.

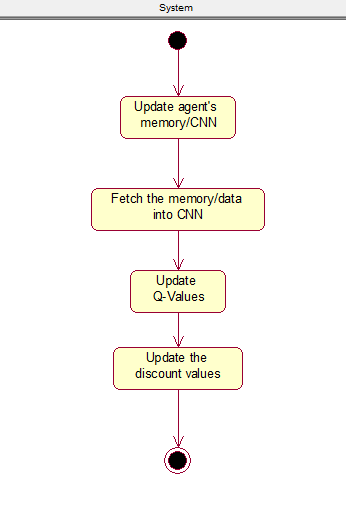


Figure 4. Begin Training Activity Diagram

The *Begin Training Activity Diagram* which is extended from the Figure 3 is the last activity diagram in this application. The training is started by initializing or *updating the agent’s memory*. Then, the knowledge that the agent got will be *Fetched into the Convolutional Neural Network (CNN)*. After that, the *Q-Values will be updated* by the new reward function and the *Discount Values* will also be updated by the CNN.

**CHAPTER IV SYSTEM DESIGN**

System Design is the process of defining the architecture, components, modules, and data of a system to satisfy the specified requirements. It can be considered as the intermediate stage between System Analysis and Product Development where the analyses that have been made in the previous chapter are now visualized in detail. The application comprises of four sections: user interface design, physical design, data design, and class diagram.

**4.1 User Interface Design**

User interface design is one of the important components of any application, be it a desktop or a web application. The user interface will allow users to use the application easily. As a desktop application, this application will one interface for the sake of simplicity which is shown by Figure 5.

The user may input the URL inside the text box. The check button is used to initiate the mature content filtering. The admin side will not have any user interface design for the current application.

**4.2 Physical Design**

Physical Design defines the minimum requirements of software and hardware used in development process to ensure that the created application runs without problems. The Table 6 and 7 below show respectively the software and hardware requirements to runs this application.

Table 6. Software Requirements

|  |  |  |
| --- | --- | --- |
| **No** | **Field** | **Description** |
| 1 | Operating System | Windows 10 Education |
| 2 | Programming Language | Python |
| 3 | Program Development | JetBrains PyCharm Professional 2019.2.1 |
| 4 | Documentation | Microsoft Word 2019 |

Table 7. Hardware Requirements

|  |  |  |
| --- | --- | --- |
| **No** | **Field** | **Description** |
| 1 | Processor | Minimum requirement: Intel Core i series |
| 2 | Memory | Minimum requirement: 2GB of RAM |
| 3 | Monitor | Minimum requirement: Resolution 1280x720 |
| 4 | Hard Drive | Minimum requirement: Free space of 100 MB |

**4.3 Data Design**

Database is not needed in this application as the application purpose is only to train the agent to be excel in a specific game environment. Therefore, there only exist three data that will be generated to save the checkpoint for the training, evaluation, and taken action in the format of checkpoint (.ckpt) file. Table 8 below depict the data design for the application.

Table 8. The Data Design

|  |  |  |
| --- | --- | --- |
| **No** | **Filename** | **Content Description** |
| 1 | training\_checkpoint.ckpt | The training checkpoint for a defined number of iterations |
| 2 | evaluation\_checkpoint.ckpt | The evaluation checkpoint for a defined number of iterations |
| 3 | next\_action\_checkpoint.ckpt | The next action checkpoint for a defined number of iterations |

**4.4 Class Diagram**

A class diagram, as specified in UML, is a static diagram that describes the classes of a system. Classes here are classes in OOP context; each is displayed with its attributes, methods, and relations with other classes (if exists). Class diagram of this application is shown in Figure 13.

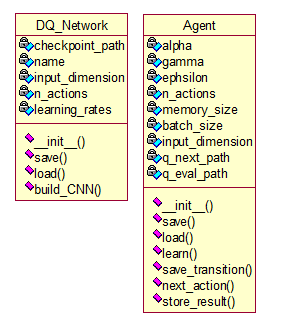


Figure 5. The Application Class Diagram

Two classes, *DQ\_Network* and *Agent*, are used in this application. There is no relation between classes in the diagrams because all of the classes are used in module. The further explanation of each class will be explained below:

1. DQ\_Network Class

The DQ\_Network Class main purpose is to build a Convolutional Neural Network for the agent. There exist five properties to define this class which are:

* + checkpoint\_path

The *checkpoint\_path* is used to store the path that will be utilized to either save or load the checkpoint file.

* + name

The *name* that will be referred from the agent’s name.

* + input\_dimension

The *input\_dimension* is used to defines the dimension of the input layer for the Convolutional Neural Network.

* + n\_actions

The *n\_actions* stands for a number of actions that the agent could take in the specific environment.

* + learning\_rates

The *learning\_rates* is a variable which hold a floating-point value from 0 to 1 that defines the learning speed of the agent.

Four methods of the *DQ\_Network class* are explained below:

* + \_\_init\_\_

The *\_\_init\_\_* method overwrite the class’s default initializer. It is utilized to initialize the class’s properties along with the TensorFlow session and the *build\_CNN* method.

* + build\_CNN

The *build\_CNN* method is the main method of this class. The neural network is configured inside this method.

* + save

The *save* method is utilized to save the trained models in checkpoint (.ckpt) format.

* + load

The *load* method is utilized to load the previously trained models in checkpoint (.ckpt) format.

1. Agent Class

The *Agent Class* represent the agent that the neural networks trained. Agent stores the environment and the taken action to feed it to the neural networks to determine the next taken action and also evaluates the previous action. Hence, the agent will be able to create a better decision in taking an action. There exist nine properties that is created for the *Agent Class* which are:

* + alpha

The *alpha* variable indicates the learning rate, which is stored in floating-point format, that will be used in the Q-values calculation.

* + batch\_size

The *batch\_size* indicates the

* + epsilon

The *epsilon* value stores a floating-point value between 0 and 1 to determines the agent characteristics to either explore or exploit the taken action.

* + gamma

The *gamma* value stores a floating-point value between 0 and 1 to determines the importance of the future reward that will be given to the agent.

* + input\_dimension

The *input\_dimension* stores the input layer dimension that will be feed to the network.

* + memory\_size

The *memory\_size* stores the memory size that the agent could hold.

* + n\_actions

The *n\_actions* stands for a number of actions that the agent could take in the specific environment.

* + q\_eval\_path

The *q\_eval\_path* stores the path string to save the checkpoint for the evaluation.

* + q\_next\_path

The *q\_next\_path* stores the path string to save the checkpoint for the next taken action training.

Seven methods in the *Agent Class* are explained below:

* + \_\_init\_\_

The *\_\_init\_\_* method overwrite the class’s default initializer. It is utilized to initialize the class’s properties.

* + save

The *save* method is utilized to save the trained models in checkpoint (.ckpt) format.

* + load

The *load* method is utilized to load the previously trained models in checkpoint (.ckpt) format.

* + learn

The *learn* method is the main method where the CNN will be run and the Q-values will be updated.

* + save\_transition

The *save\_transition* method is utilized to save the taken action and the environment states to the agent’ memory.

* + next\_action

The *next\_action* method is used to determine whether the agent should explore or exploit the environment.

* + store\_result

The *store\_result* method is used to store the training result and plot it into the chart.