Georgio Zeilaa

17014182

ANimal name report

CS3S668 - Intelligent Systems

Table of Content

Introduction 3

Logic and workings 3

How the program runs 4

Dealing with uncertain information explained 4

Issues encountered 5

Dealing with uncertain information 5

References 6

Appendix 7

# Introduction

The problem analysis is figuring out the animal name by asking questions according to the characteristics of the animal. A decision will be produced according to the inputs to the questions asked. This does follow the self-appointed expert’s simple expressions by a decision tree which is show in figure 1.

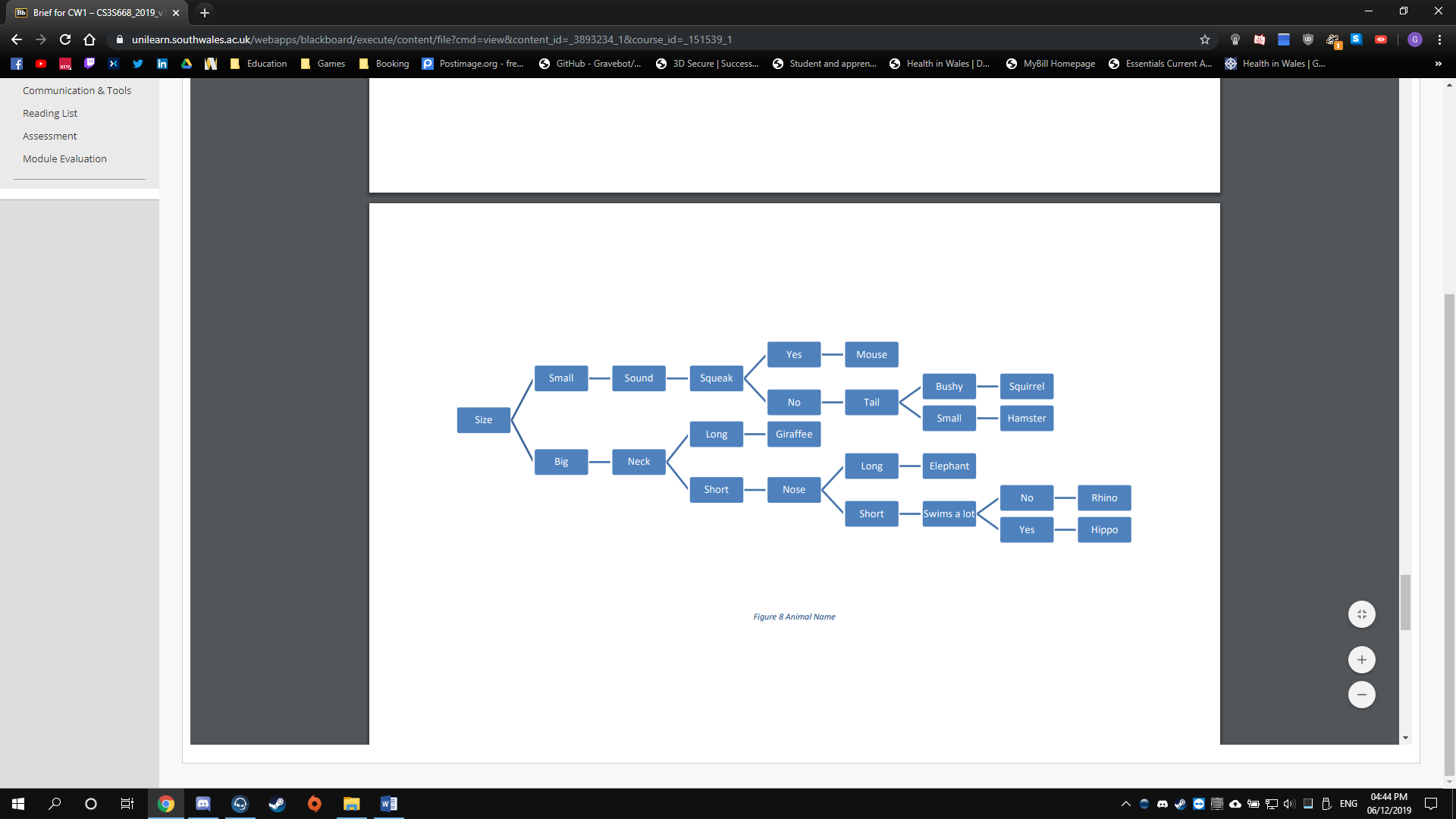
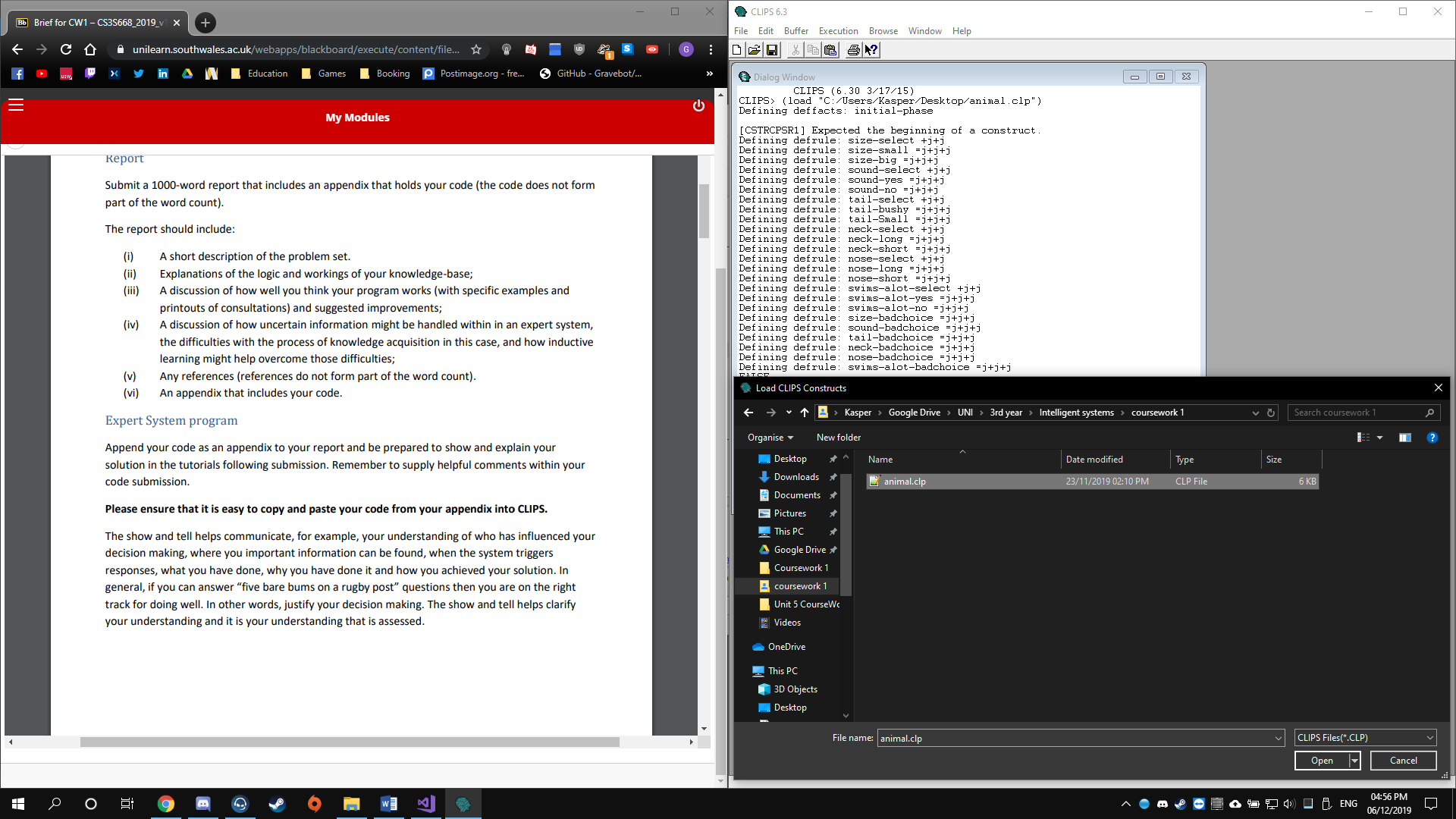


Figure Animal Tree

# Logic and workings

This expert system uses a decision tree diagram so there will be a question and the answers are either left or right answers. This will then lead to another question which will eventually give the answer at the end depending on what path you took. In this case Questions about animal characteristics are asked and then either right or left answer, this can be for example yes or not, big or small and long or short.

To load in the animal.clp, when running CLIPS 6.3, File > Load as shown in figure 2:

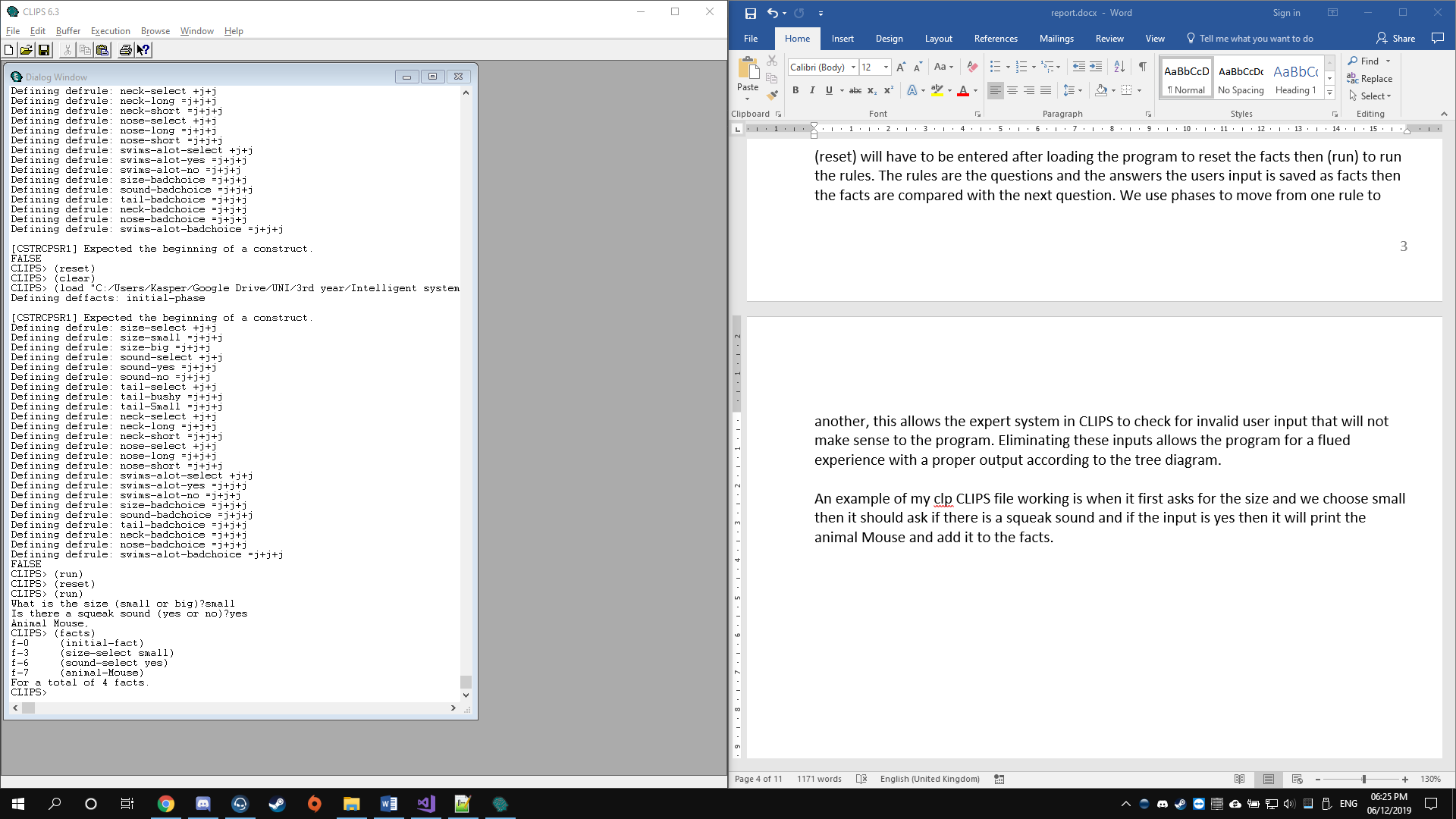
Figure Loading clp in CLIPS

(reset) will have to be entered after loading the program to reset the facts then (run) to run the rules. The rules are the questions and the answers the users input is saved as facts then the facts are compared with the next question. We use phases to move from one rule to another, this allows the expert system in CLIPS to check for invalid user input that will not make sense to the program. Eliminating these inputs allows the program for a flued experience with a proper output according to the tree diagram. Phases also helps with scalability so this can be used in most expert systems and this is done easily on large sized decision tree diagrams.

## How the program runs

An example of my clp CLIPS file working is when it first asks for the size and we choose small then it should ask if there is a squeak sound and if the input is yes then it will print the animal Mouse and add it to the facts.

Questions Asked according to the previous answer.



The final answer saved as fact.

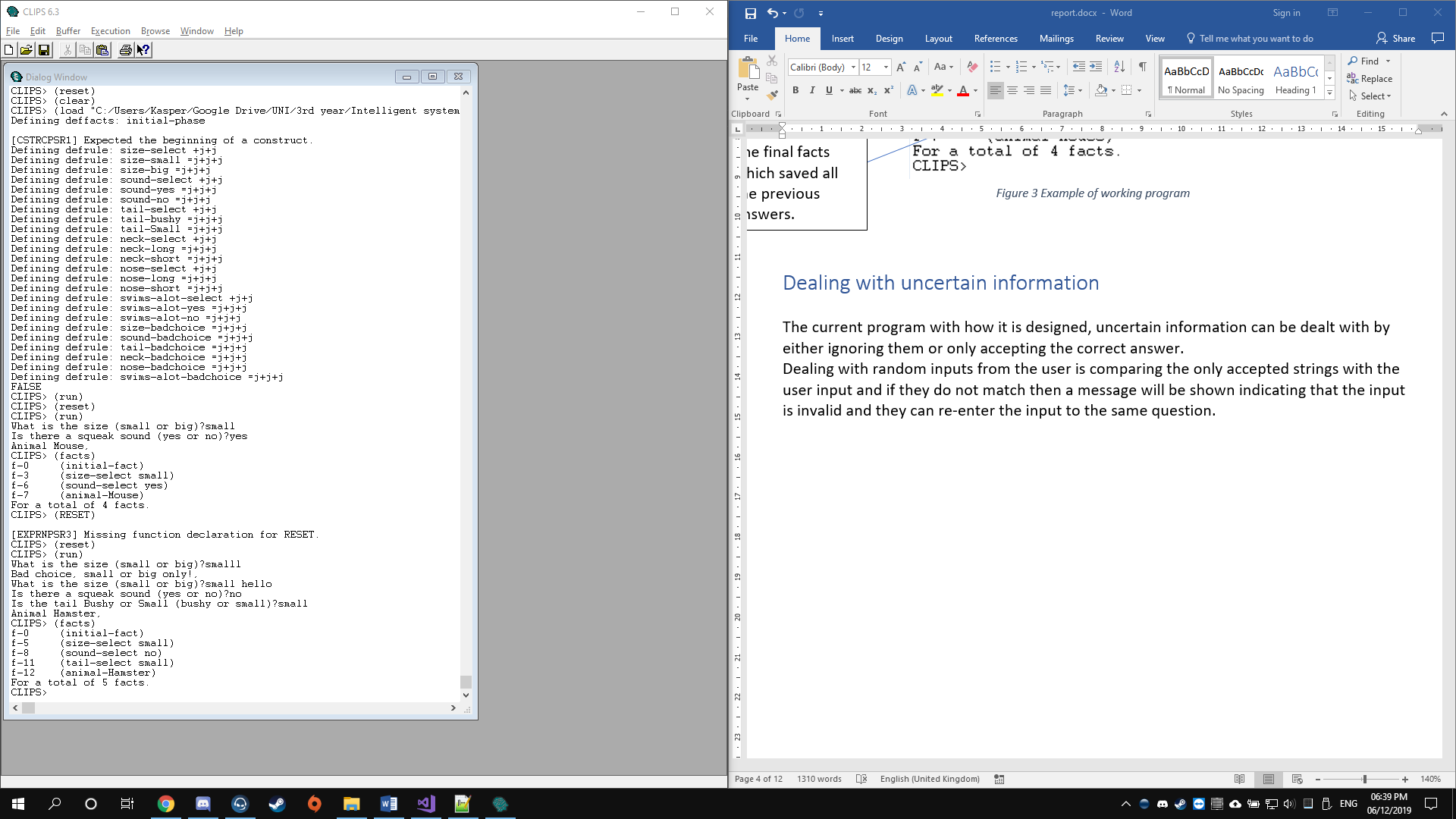
Figure Example of working program

The final facts which saved all the previous answers.

## Dealing with uncertain information explained

The current program with how it is designed, uncertain information can be dealt with by either ignoring them or only accepting the correct answer.

Dealing with random inputs from the user is comparing the only accepted strings with the user input and if they do not match then a message will be shown indicating that the input is invalid and they can re-enter the input to the same question.



“small” instead of “small”

“small hello” instead of “small”

Figure Uncertain information testing

The facts show what was accepted in figure 4. The first question was answered with input of “smalll” which is incorrect, according to the question, it only accepts small or big as either answer. The second part of the questions was about the size again which it was asked due to the first input being incorrect and not accepting it. This time it accepted the input “small hello” by ignoring anything after small which is the correct answer and adding small to the fact of size-select. The second question was asked and the user input was correct so no issue there. I have included the code in the Appendix below.

## Issues encountered

There is a small issue that occurred while programming the entering the questions for the decisions to be made. The first row where the size of the animal is small and then it says sound and the next branch is squeak with no other branch connected to sound so it is just one connection within the sound branch. This got the decision process a bit of a challenge by having to add another rule however this was encountered by combining the sound and squeak into one rule so there will be only one question asked during the runtime of the program. Squeak branch does connect later to two different outcomes depending on the user’s input. This eliminated any unnecessary questions asked.

# Dealing with uncertain information

Uncertain information means working with vague data. Types of uncertainty data can be uncertain information that cannot be a full fact for example “I am 300 years old”. Other types of uncertainty are: imprecision. Vagueness, inconsistency and ambiguity. There can be set valued approaches to check if the answer equals the answer that is excepted. This is what the expert program that was done on animal names does. [2]

There are theories that can help with uncertain information and dealing with them. One of them is the Dempster-Shafer Theory of Evidence. This is where it finds a reasoning with uncertainty in artificial intelligence. It creates belief functions to separate between uncertainty and ignorance. It can also handle lack of information by expressing explicitly. This type of theory is more for a large artificial intelligent system, this does help with hypothesis points which can lead to bigger image because of the set of all possible hypothesis. [2]

Inductive learning is when the person discovers rules by learning examples. Inductive learning algorithm uses if-then statements and helps to generate a set of classification rule. This is how the expert system that was created for the animal names was created. So it does use the rule of checking if the input of the user equals the input we expect then goes to the next question until an answer is found. [3]

# References

[1] CLIPS Tutorial 4 CLIPS Tutorial 4 Modular Design, Execution Modular Design, Execution Control, and Rule Efficiency Control, and Rule Efficiency. (n.d.). [online] Available at: https://www.cs.odu.edu/~mukka/cs480f09/Lecturenotes/Expertsystems/clips/tutorial/tutorial4.pdf [Accessed 6 Dec. 2019].

[2] Dubey, S., Pandey, R. and Gautam, S. (2014). Dealing with Uncertainty in Expert Systems. International Journal of Soft Computing and Engineering (IJSCE), [online] (3), pp.2231–2307. Available at: https://pdfs.semanticscholar.org/d58c/5c4cc41d5f92a7cbe9c5b919a7a5293604bc.pdf [Accessed 12 Dec. 2019].

‌[3] GeeksforGeeks. (2019). Inductive Learning Algorithm - GeeksforGeeks. [online] Available at: https://www.geeksforgeeks.org/inductive-learning-algorithm/.

# Appendix

## Appendix A

**PLEASE NOTE: COPY AND PASTE CODE INTO A .CLP FORMATTED FILE THEN IN CLIPS, FILE>> LOAD>> [FILENAME].CLP FILE. ONCE IT HAS BEEN LOADED: RUN THE FOLLOWING COMMANDS: (reset), (run)**

(deffacts initial-phase

(phase choose-size))

;list of phases saved as facts to transfer between the rules, the use of phases are used.

;size small or big----------------------------------------

(defrule size-select ;This is where the question will be asked

(phase choose-size) ;states which phase that this is at

=>

(printout t "What is the size of the animal (small or big)?") ;prints the question to the user

(assert (size-select (read))));inserts the input of the user in the size-select fact

(defrule size-small ;the left side of the answer for the above question

?phase <- (phase choose-size) ;if the phase is coming from the choose-size phase then this will run

?choice <- (size-select ?size&small);checks the input from user from the fact size-select, in this case it accepts only small

=>

(retract ?phase ?choice);receives the phase and choice from the previous question

(assert (size-select ?size)(phase choose-sound))) ;this moves to the phase choose-sound and inserts the correct size in size-select fact

(defrule size-big ;the right side of the answer for the above question

?phase <- (phase choose-size) ;checks again the phase if it is coming from choose-size

?choice <- (size-select ?size&big) ;checks the input from user from the fact size-select, in this case it accepts only big

=>

(retract ?phase ?choice);receives the phase and choice from the previous question

(assert (size-select ?size)(phase choose-neck))) ;for phase it goes to that one of choose-neck and the user input is inserted into the fact size-select

;--------------------------------------------------------

;from this point it is the same general code except for different variables and outcomes to different phases and fact variables different naming.

;sound yes or no-----------------------------------------

(defrule sound-select

(phase choose-sound)

=>

(printout t "Does it squeak (yes or no)?")

(assert (sound-select (read))))

(defrule sound-yes

?phase <- (phase choose-sound)

?choice <- (sound-select ?sound&yes)

=>

(retract ?phase ?choice)

(assert (sound-select ?sound)(animal-Mouse))

(printout t "Animal Mouse", crlf))

(defrule sound-no

?phase <- (phase choose-sound)

?choice <- (sound-select ?sound&no)

=>

(retract ?phase ?choice)

(assert (sound-select ?sound)(phase choose-tail)))

;-----------------------------------------------------------

;tail brushy or small---------------------------------------

(defrule tail-select

(phase choose-tail)

=>

(printout t "Is the tail bushy or Small (bushy or small)?")

(assert (tail-select (read))))

(defrule tail-bushy

?phase <- (phase choose-tail)

?choice <- (tail-select ?tail&bushy)

=>

(retract ?phase ?choice)

(assert (tail-select ?tail)(animal-Squirrel))

(printout t "Animal Squirrel", crlf))

(defrule tail-Small

?phase <- (phase choose-tail)

?choice <- (tail-select ?tail&small)

=>

(retract ?phase ?choice)

(assert (tail-select ?tail)(animal-Hamster))

(printout t "Animal Hamster", crlf))

;-----------------------------------------------------------

;neck long or short---------------------------------------

(defrule neck-select

(phase choose-neck)

=>

(printout t "What is the neck size (long or short)?")

(assert (neck-select (read))))

(defrule neck-long

?phase <- (phase choose-neck)

?choice <- (neck-select ?neck&long)

=>

(retract ?phase ?choice)

(assert (neck-select ?neck)(animal-Giraffe))

(printout t "Animal Giraffe", crlf))

(defrule neck-short

?phase <- (phase choose-neck)

?choice <- (neck-select ?neck&short)

=>

(retract ?phase ?choice)

(assert (neck-select ?neck)(phase choose-nose)))

;-----------------------------------------------------------

;Nose long or short---------------------------------------

(defrule nose-select

(phase choose-nose)

=>

(printout t "What is the size of the nose (long or short)?")

(assert (nose-select (read))))

(defrule nose-long

?phase <- (phase choose-nose)

?choice <- (nose-select ?nose&long)

=>

(retract ?phase ?choice)

(assert (nose-select ?nose)(animal-Elephant))

(printout t "Animal Elephant", crlf))

(defrule nose-short

?phase <- (phase choose-nose)

?choice <- (nose-select ?nose&short)

=>

(retract ?phase ?choice)

(assert (nose-select ?nose)(phase choose-swims-alot)))

;-----------------------------------------------------------

;SwimsAlot yes or no---------------------------------------

(defrule swims-alot-select

(phase choose-swims-alot)

=>

(printout t "Does it swim a lot (yes or no)?")

(assert (swims-alot-select (read))))

(defrule swims-alot-yes

?phase <- (phase choose-swims-alot)

?choice <- (swims-alot-select ?swims-alot&yes)

=>

(retract ?phase ?choice)

(assert (swims-alot-select ?swims-alot)(animal-Hippo))

(printout t "Animal Hippo", crlf))

(defrule swims-alot-no

?phase <- (phase choose-swims-alot)

?choice <- (swims-alot-select ?swims-alot&no)

=>

(retract ?phase ?choice)

(assert (swims-alot-select ?swims-alot)(animal-Phino))

(printout t "Animal Rhino", crlf))

;-----------------------------------------------------------

;at bad choices, all the wrong inputs goes here, this is where it is proccessed.

;bad choices -----------------------------------------------

(defrule size-badchoice

?phase <- (phase choose-size) ;if they are coming from that phase it keeps running.

?choice <- (size-select ?size&~small&~big) ;if the inputs are not small or big then it will continue.

=>

(retract ?phase ?choice)

(assert (phase choose-size)) ;goes to the phase choose-size.

(printout t "Bad choice, small or big only!", crlf) ;this prints the message stating that this is a bad choice and it only accepts the inputs shown.

)

;from this point the rest are the same with the usual different variable namings.

(defrule sound-badchoice

?phase <- (phase choose-sound)

?choice <- (sound-select ?sound&~yes&~no)

=>

(retract ?phase ?choice)

(assert (phase choose-sound))

(printout t "Bad choice, yes or no only!", crlf)

)

(defrule tail-badchoice

?phase <- (phase choose-tail)

?choice <- (tail-select ?tail&~bushy&~small)

=>

(retract ?phase ?choice)

(assert (phase choose-tail))

(printout t "Bad choice, bushy or small only!", crlf)

)

(defrule neck-badchoice

?phase <- (phase choose-neck)

?choice <- (neck-select ?neck&~long&~short)

=>

(retract ?phase ?choice)

(assert (phase choose-neck))

(printout t "Bad choice, long or short only!", crlf)

)

(defrule nose-badchoice

?phase <- (phase choose-nose)

?choice <- (nose-select ?nose&~long&~short)

=>

(retract ?phase ?choice)

(assert (phase choose-nose))

(printout t "Bad choice, long or short only!", crlf)

)

(defrule swims-alot-badchoice

?phase <- (phase choose-swims-alot)

?choice <- (swims-alot-select ?swims-alot&~yes&~no)

=>

(retract ?phase ?choice)

(assert (phase choose-swims-alot))

(printout t "Bad choice, yes or no only!", crlf)

)

;-----------------------------------------------------------