Computer programs solving I.Q. questions: Number sequences, direction, and pick the odd one out problems

Kelvin Benzali (kben19@student.monash.edu)
Bachelor of Computer Science
Monash University, VIC 3800 Australia

Supervisor:
David L. Dowe
School of Computer Science and Information Technology
Monash University, Clayton, VIC 3800 Australia

Abstract

Artificial intelligence is a combination of complex techniques and algorithm to imitate the intelligence of living beings. Despite the facts that it is been researched for over half a century, the artificial intelligence has not yet reached to human intelligence. Turing test is an example of measuring the A.I. capability of imitating the human intelligence perfectly. Additionally, I.Q. test in which a method usually used to quantify human intelligence, can also be used to help measure a computer program as well. I.Q. test consist of various type of questions in which each type can measure different attributes of intelligence such as number, memory, linguistic, spatial, and many more. Therefore, the program must be able at least solve every types of questions in order to be recognized with the same level of human intelligence. This report would provide the techniques, tests, and analysis of the A.I. behavior to solve the number pattern, direction, and odd one out problems in particular. The additional details also include the analysis results of the output given various test cases, the evaluation of the program practicality, the comparison between human and program performance and addressing the program methodology in detail. Nonetheless, the project for a computer that can pass the I.Q. test by human standard might contribute to A.I. field study and generate the baseline idea on how the intelligence characteristics work in respect from computer approach.

Keywords

Artificial intelligence, I.Q. tests, Reinforcement learning, Sequence pattern prediction, Arithmetic and geometric sequence, Odd one out.

1.0 Introduction

The project main objective is to develop a program that can handle and solve I.Q. questions. The purposes of the project is to explore and analyze the capabilities of artificial intelligence program in respect to solve I.Q. questions. Furthermore, the report analysis can also be considered to help measure the current technology of algorithm technique in terms of I.Q. tests problems that usually only used to measure human intelligence but not a computer program. In other words, a program can also be created to handle such test and generate the score in which the score can be used to measure the program or algorithm capabilities. Development of this project can be considered as the application of various algorithm to solve various I.Q. test problem in order to create an agent capable of passing the I.Q. test.

The program requirements consist of functional requirements and non-functional requirements. The functional requirements usually includes the program behavior, program reasoning towards the problem, and the program capabilities in passing the test. The program main functional requirements can be explicitly stated to only develop the computer that can solves number sequences problem, direction problem, and odd one out problem. In addition, the program can also be able to understand the context of the question independently without any user feedback and the ability to receive a file input of questions in case of large input. Hence, the program is able to analyze the question pattern, recognize the question type and solves it with respective agent, otherwise the program should behave responsibly even without any knowledge of the question.

The program non-functional requirements include the usability, performance, documentation, extensibility, and fault tolerance of the program. The usability of the program is related to the program user interface. The program is required to have a graphic user interface for at least able to interact with the user which consist of a field that able to take a single question from the user, the option to edit the agent knowledge for testing purposes, and the option to input a file for large input or analysis purposes. The performance of the program is required to be able to solve the question that includes large input as well for at most a reasonable amount of time. The reasonable amount of time can be specified to be in quadratic complexity in the worst scenario. The project also needs to be well documented both in report, specification and in code program. The design software architecture of the program must be as flexible as possible to allow more extension, customization, and modification without any risk. The fault tolerance of the program is also needed in order to handle the error received due to unknown reason without hindering the program operation.

The project scope constraint is only limited to handle I.Q. questions type only explicitly for number sequences, direction, and odd one out problems. The scope constraint also includes the implementation of simple graphic user interface but it does not need to emphasize the usability of the interface.

2.0 Background

2.1 Research Studies

Turing test is one of an interesting method to identifying an intelligence by the means of observing. The test used to be called as Turing imitation game in which a machine and human are interviewed and observed by the judge with an exception that the judge does not know which one is the artificial intelligence. Therefore, the purposes of this game is too deceive the judge to prove that the machine is indistinguishable with human intelligence (Turing, 1950). An intelligence can be identified with a basic approach of observation and analyze its result to decide if the machine meet the standard of intelligence. However, an intelligence that can rival human intelligence is very complicated as such it would be nearly impossible or would be invented in distant future to create such intelligence that can imitate human just by observation approach. For instance, the modern artificial intelligence still even cannot imitate human just by observation method (J. Hernandez-Orallo, 2016).

I.Q. or Intelligent Quotient is a quantitative measurement of a human intelligence with a tool of questions and tests. The purposes of I.Q. test is to measure the capability of human cognitive process and the amount of human knowledge that a person has learned. It is also known to be used to check person mental growth. Based on the previously stated statement, I.Q. test might be a plausible way to identify a machine intelligent as a machine can be created specifically to solve the I.Q. test and yield some results. The results would then judge to identify the machine possibilities and capabilities of holding an intelligence. However, both of Turing test and I.Q. test are a test designed for human rather than intelligence itself. Hence, the these tests are not suitable to measure an artificial intelligence as the machine might get a pass score in the certain test that the machine can understand, but on the other hand the machine get a low score in other tests. The reason is because these tests are initially designed to test humanity but not for machine specifically such as philosophy and a misconception of intelligence (Dowe D. L., 2012). Even though I.Q. tests is not suitable to measure the machine, this project is not entirely to discover and prove the capabilities of machine in I.Q. test, rather as an application of techniques and algorithms to solve I.Q. questions.

The I.Q. test consist of various type of questions in purpose of measuring various abilities and characteristic of intelligence such as reasoning, spatial, logical, and many others. Therefore, the machine needs to be able to understand the questions before solving it with respective techniques. The method can be identifying keywords and pattern in the questions itself for example considering a number sequence question would usually accompanied by a keywords of sequence, digit, series, or next. By checking the keywords and the patterns, the program might identify the question and proceed to its respective algorithm to solve the specific problem type (Pritika Sanghi, 2003). However, these types of method is not heuristic enough to be able to understand every questions as it relies on the specified keywords and might unable to identify the question if the question pattern is uncommon. Nonetheless, the machine is good enough for at least to be able to identify questions under normal circumstances.

Odd one out type of questions is popular among the I.Q. test as it requires not only knowledge to identify the odd one between the choices but also need several other factors to make the decision such as reasoning, logical, and bias. This type of question can be solved by applying dictionary comparison techniques but it requires a vast database of words. The other plausible way to solve the question is the application of reinforcement learning technique of artificial intelligence since the machine does not need the vast database as a baseline to compare the relations between words, rather the machine would work independently. Reinforcement learning is a method of a machine to learn the environment and takes action to maximize the reward value under uncertainty. Hence, the method is a goal oriented approach of machine learning as the machine will choose an action to further maximize the reward. In addition, reinforcement learning is different with other type of learning due to the willingness to explore the uncertainty for the possibility of better reward (Richard S. Sutton, 2018). However, this odd one out characteristic does not necessarily need the exploration attributes since relation between words is not abstract once the machine has learned its meaning, categories, and relation.

2.2 Project Risk

The project risk management which includes the risk management table, risk probability, risk reduction strategy, and risk encountered can be seen below as stated in project specification.

Table 1 Risk Management Table

No	Risk	Description	Risk Reduction Strategy	Probability	Impact	Score	Threat
1	Errors and bugs	Errors and bugs appear in the program implementation phase	Execute the unit testing process every time a new features is added to remove the unnecessary minor bugs. Do the iterations to ensure that the program is always stable at any time.	10	1	10	Very Low
2	Algorithm completeness	Algorithm designed to solve the problem cannot solve the problem completely	Before implementing the algorithm into practice, the algorithm itself need to be researched thoroughly and proved and tested its theory in paper or in testing environment.	4	10	40	Moderate
3	Program time complexity	The time needed to solve a problem must be in reasonable time	Analyze the time complexity of the algorithms that will be used into the program and modify it into more efficient algorithm if possible. Otherwise, try to find other methods or abandon the technique altogether to prevent the risk	7	5	35	Moderate
4	Platform problem	The probability of facing problems regarding the software or hardware used to develop this project	Check the updates of the software and libraries used in the program on weekly basis at least. Make sure to update the backup of the program on weekly basis as well	2	9	18	Low

5	Unexpected events	The unexpected events occurs to the developer of the program. For example away for important events, sickness or accident	The only available backup for this risk is to develop the program early in order to replace the time loss during the absent of the developer.	2	7	14	Low
6	Schedule overrun	The project does not meet its schedule constraint and meet its expected deadline with all the main requirement	Avoid any changes to project scope whenever possible to prevent any scope creep and add more time requirement to implement the function. Review the progress of the project development every week to ensure the project is on track and identify any time overrun early.	7	9	63	High
7	Test cases completeness	Test cases sample and methods are not covering all boundary and bugs possibilities. Hence, unable to pointed out the error and bugs left by the program	Analyze carefully the nature of the problem and all the samples possibilities of the problem to cover every bugs. Develop more detailed test cases plan for each problem and functionality. Develop an automated test cases generator to help the testing stages of the project and reduce the possibility of bugs.	3	7	21	Moderate
8	Lack of usability	The difficulty experienced by the user to operate the program due to lack of usability. The user cannot grasp the full potential of the program because of the steep learning curve	Revise the project specification and requirements regarding the usability to be at least meet the minimum user acceptance criteria.	5	4	20	Moderate

Risk Encountered

There are several risks that are encountered during the development stage of the project. Although the number of risks encountered is small, but it is still important to state the risks encountered for future references. The risks encountered are as follows:

- Error and bugs risk
  Error and bugs always occurred during the implementation and the testing stages of the project. Sometime the error is big enough to break the program and it needs some time to fix it. However, most of the errors are minor bugs in which can be fix quickly. The final version of the program does not have any known errors and bugs but it is not guaranteed to free of errors.
- Algorithm completeness risk
  The implementation of algorithm for number sequences, and odd one out problems shows the weakness in algorithm design. Number sequences algorithm can only predict the sequences that are already implemented inside the program. The odd one out algorithm shows the small percentage of wrong answers even after an ideal number of learning samples.
- Program time complexity
  The algorithm for odd one out problems used to have quadratic complexity. The risk is not harmful if the input size is not large. However, it will become a problem and risk once the input size is large enough to multiply its time processing. However, in the later implementation, the problem is fixed by modifying the algorithm technique. The detailed information can be seen in the performance section

2.3 Resource Requirements

The resources of hardware and software required to run the program successfully can be seen below as stated in the project specification proposal.

Hardware Requirements:

- 1 gigahertz (GHz) processor
- 1 gigabyte (GB) RAM (32-bit/64-bit)
- 1 GB available hard disk space
- Basic integrated video card graphic
- No internet connection required

Software Requirements:

- Windows 7 operating system (32-bit / 64-bit)
- PyCharm Integrated Development Environment (IDE)
- Python programming language version 3.5.1
- Python libraries used:
  - Appjar library for GUI
  - abc module (Abstract Base Class for Python)

2.4 Tasks Management

Table 2 Task Management Table

No.	Task	Start Date	End Date	Duration	Week	Dependency
1.0	**Initiating**					
1.1	Deciding Project	Mon 2/26/18	Fri 3/2/18	5 days	1	
1.2	Deciding Program Platform	Sat 3/3/18	Mon 3/5/18	3 days	2	1.1
2.0	**Planning**					
2.1	Gathering Requirements	Tue 3/6/18	Fri 3/16/18	11 days	2,3	1.2
2.2	Develop Project Specification	Sat 3/17/18	Sun 4/8/18	23 days	4,5	2.1
3.0	**Implementation**					
3.1	Number Sequence Agent	Mon 4/9/18	Sun 4/15/18	7 days	6	2.2
3.2	Direction Agent	Mon 4/16/18	Sun 4/22/18	7 days	7	3.1
3.3	Odd One Out Agent	Mon 4/23/18	Wed 5/2/18	10 days	8,9	3.2
3.4	Graphic User Interface	Thu 5/3/18	Sun 5/6/18	4 days	9	3.3
4.0	**Testing**					
4.1	Unit Testing	Mon 5/7/18	Thu 5/10/18	4 days	10	3.4
4.2	System Testing	Fri 5/11/18	Sun 5/13/18	3 days	10	4.1
4.3	Usability Testing	Mon 5/14/18	Thu 5/17/18	4 days	11	4.2
4.4	User Acceptance Testing	Fri 5/18/18	Sun 5/20/18	3 days	11	4.3
5.0	**Closing**					
5.1	Create Final Project Report	Mon 5/14/18	Sun 5/27/18	14 days	11,12	3.4
5.2	Create Testing Report	Mon 5/21/18	Sun 5/27/18	7 days	11,12	4.4
5.3	Create Final Presentation	Mon 5/7/18	Fri 5/11/18	5 days	10	4.2
6.0	**Maintenance**					
6.1	Workbook Report	Tue 3/6/18	Fri 5/11/18	67 days	2-12	1.2

3.0 Method

3.1 Methodology

The program methodology consist of several parts. The first part is classified as question parsing where the agent analyze the question pattern and determine its type, and get the multiple choice in the question if any. The second part is classified as an agent controller where several different agents inherited from agent class grouped together and managed by the agent controller to be an intermediate between the agents and user action. The last part is the functionality of each respective agents which this program currently has implemented 3 agents of number sequences, direction, and odd one out agent. Each of these agents contain different functionality of solve and sequence parsing functionality since each problem type has its unique pattern and characteristics that need to be solved.

The question parsing methodology for identifying the problem type from the question works by parsing the question and check for its important keywords. The function would loop for each word in the question and add a score to a respective problem type for example number sequence types usually related to a keywords of “sequence”, “digit”, or “series” while the direction problem would always contains “south”, “west”, “left”, or other direction words in the question. After the program has check every words in the question, then it would determine the type by checking each problem score. The scores problem that has meet its requirement means that the question is mostly related to the problem. However, the question string needs to be generalized before looping for all the words by removing any special characters that are not related to prevent some miss words in the checking process.

The question parsing for multiple choice is assumed that the question has “a.”, “b.”, “c.”, “d.” format for the multiple choices. Since the algorithm is completely relies on the keywords of the multiple choices character, any unknown character for the keywords need to be added into the code manually. The algorithm works by replacing any newline character (”\n”) into space character then it would parse the question for every words. For each words, check if the current word of the first 2 characters contain the choices keywords such as “a.”, “a,”, “a)” or other format. Then, mark the current index and add the rest of choices string into the multiple choice array once the next choice founded. If the choice string and the choice format is connected as one word, then the index number is used to identify the problem by checking if the current number of word pointer subtract by the index is one.

The agent controller basically manages the user action with the responsible agent. The agent controller class would have a private variable of agent collection which holds all of the agent concrete class that inherit agent class in this case it would be number, direction, and odd one out class. When the agent controller given an action to solve a certain question, the agent controller would first check the question type by calling the agent function of checkType(). Then, the agent controller will solve the question by assigning the question into the responsible agent based on the type returned by the check type function since every agent has different method to solve the question. Lastly, the agent controller return the result output from the respective agent into the view. In addition, agent controller can also handle large input file by opening the received file path and loop for each line (assuming each line representing each question) then call the function mentioned before for solving a certain question in which the question is represented as the line. Afterwards, holds all the answer values into an array of answer that can be used later to compare the solution.

The agent class is an abstract class that contains abstracts method of solve and get sequences. Therefore, any agent class that inherits the agent abstract class would implement these methods in which solve function represent the technique to solve the problem and get sequence function represent the technique to get the data needed for the agent to solve the problem and usually heavily affected by the question pattern.

The get sequence function of number class is simply checking every word in the question and determine the start of the sequence by finding the position of the first two numbers that are right next to each other. Then, the algorithm can start adding the rest of the numbers into the sequence array. The figure 1 below shows the pseudocode of the get sequence for number class.

```
def getSequence(self):
    seqList = []
    Generalized input text() # Remove special characters

    for i in range(length(QuestionWordArray)-1):
         # If the current word and next word are number
        if QuestionWordArray[i].isdigit() and QuestionWordArray[i+1].isdigit():
            if length(seqList) == 0:   # Add the first digit of the sequence
                seqList.append(int(QuestionWordArray[i]))
            seqList.append(int(QuestionWordArray[i+1]))

    return seqList
```

Figure 1 Number getSequence Pseudocode

The solve function of number class is done by predicting every arithmetic and geometric sequences which every prediction is implemented as a function that would return a Boolean representing if the sequence match the prediction criteria, and an integer represent the next number in the sequence. The number return value is -1 if the sequence does not match with the pattern criteria.

The multiplication prediction algorithm works by creating a difference list that consist of number represented as the difference between two numbers next to each other in the sequence. Based on difference list, if all the difference number are all the same then generate the next number sequence.

$$x_i where\ x\ is\ the\ sequence\ list\ and\ i\ is\ index$$

$$diffList_j = x_{i+1} - x_i$$

The power prediction algorithm works by checking the results of dividing every two numbers next each other are all the same with the later number as the numerator and previous number as the denominator. If the results of all the division is the same then generate the next number sequence simply by multiply it with the number from the division result.

```
def power(self, number):
    tempNum = None

    basePower = number[1] / number[0]
    for i in range(1, length(number)-2):
        if number[i+1] / number[i] != basePower:
            return [False, tempNum]

    tempNum = number[length(number)-1] * basePower
    return [True, tempNum]
```

Figure 2 Power Pseudocode

The fixed power sequence is the opposite of the power sequence mentioned before since the power become the constant number while the base number is the sequence. The algorithm has 2 steps of verification which are identifying the index of the sequences, and verifying the fixed power pattern. Identifying the index sequences is done by verifying every number is divisible by some arithmetic sequence of $x_i = x_{i-1} + 1$ where x_0 is a subset of natural number and i is the sequential number. For example, n_0 is divisible by x_0, n_0 is divisible by x_0 and so on. Once the algorithm has secured the index of the sequence, the next step would be checking the power pattern by calculating power $= log_{x_i}(n_i)$. Calculate this formula for every sequence number and check if it has the same results for all number. Then, generate the next sequence number by $n_i = x_i \times power$, where i is the next index of the sequence and power is the result of previous formula.

```
def fixedPower(self):
    j = 2    # Start by 2 to prevent zero division error
    if self.sequence[0] == 1:    # Remove number 1 in the sequence
        self.sequence.remove(1)
    match = False
    tempNum = None
    while j < 100:   # limitations of 100 arithmetic sequence
        num1 = self.sequence[0] % j
        num2 = self.sequence[1] % (j+1)
        # Base check of the arithmetic sequence
        if num1 == 0 and num2 == 0:
            match = True
            # Check if all sequence number are divisible by the arithmetic sequence number
            for i in range(length(self.sequence)-1):
                # If the arithmetic sequence do not match with the sequence number
                if self.sequence[i] % (j+i) != 0 or self.sequence[i+1] % (j+1+i) != 0:
                    match = False
                    break
        if match:
            break
        j += 1

    for i in range(length(self.sequence)-1):
        power1 = $log_{i+j}(self.sequence[i])$
        power2 = $log_{i+j+1}(self.sequence[i+1])$
        if power1 != power2:     # Second check of power pattern using logarithm function
            return [False, tempNum]
    tempNum = $(length(self.sequence) + j)^{power\ 1}$
    return [True, tempNum]
```

Figure 3 Fixed Power Pseudocode

Fibonacci sequence can be predicted by using its recurrence relationship. The algorithm works by verifying that every number is the sum of the previous number and the second previous number.$x_i = x_{i-1} + x_{i-2}$. Hence, the sequence need to have at least 3 numbers of sequence and the algorithm need to check it first before predicting the pattern.

Prime sequence can be predicted by verifying that all numbers in the sequence is prime. The approach used to check if the number is prime is by checking that n is not divisible with every number of x where $x = 6k \pm 1$ and $x \leq \sqrt{n}$. However, to generate the next prime number, the algorithm needs to keep aware that prime numbers is a subset of x but not all x numbers are prime number.

```
def prime(self):
    tempNum = None
    for i in self.sequence:
        if not(self.isPrime(i)): # Check if the number is prime
            return [False, tempNum]

    generate the next prime number as tempNum
    return [True, tempNum]

def isPrime(self, num):
    # First check of primality test before proceeding into (6x±1) test
    if num % 2 != 0 and num % 3 != 0:
        n = $\sqrt{num}$
        j = 1
        while 6*j - 1 <= n or 6*j +1 <= n:
            if 6*j - 1 <= n and num % (6*j-1) == 0:
                return False
            if 6*j + 1 <= n and num % (6*j+1) == 0:
                return False
            j += 1
    else:
        return False
    return True
```

Figure 4 Prime Pseudocode

The two differences prediction algorithm is one of the example that the sequence has two arithmetic sequences. For example, when i is odd then add the previous number with x, but when i is even then subtract the previous number with y. The pattern can be predict by generating the differences list just like in multiplication algorithm and check every differences with a range of 2. Generate the next sequence number by identifying the differences of both even and odd and apply it into the calculation based on the next index.

The get sequence function of direction class is heavily relies on the keywords commonly used in the direction question. Further addition of keywords due to the results of new research is possible. The algorithm get the direction data by checking every word in the question if the word exist in the keywords. If a word is verified to be exist in the keywords, then add the direction word and the distance number into the sequence array. This algorithm is always successful with an assumption that the direction used exist in keywords database and the distance travel number has the same order with the directions array. The current keywords used includes “West”, “South”, “East”, “South”, “Left”, and “Right”.

```
def getSequence(self):
    self.direction = []
    self.sequence = []
    directionName = ["east", "south", "west", "north", "left", "right"]
    Remove any special character in Question String
    tempList = make Question String into list with space as denominator
    index = 0
    for i in tempList:
        if i contain words in directionName:
            self.direction.append(i)
        elif i.isdigit():
            self.sequence.append(int(i))
```

Figure 5 Direction getSequence Pseudocode

The solve function of the direction class can be represented as a graph with x-axis and y-axis. For every moves in the sequence, the algorithm update the x value and y value respectively. However, for left and right action, the algorithm need to check previous direction used to determine the next direction for example the previous direction used was east action, then the current left action would represent going to the north direction. The final distance can be calculated by Pythagoras formula and the direction can be verified based on x value and y value to represent the position. Obviously, a minus y value would represent south and positive y value represent north while negative x value would represent west and positive x value represent east.

```
def solve(self):
    x = 0
    y = 0
    prev = ""
    self.getSequence()
    for i in range(0, length(self.sequence)):
        if direction[i] == "east"¥
                or (direction[i] == "left" and prev == "south")¥
                or (direction[i] == "right" and prev == "north"):
            x += distanceSequence[i]
            prev = "east"
        elif direction[i] == "south"¥
                or (direction[i] == "left" and prev == "west")¥
                or (direction[i] == "right" and prev == "east"):
            y -= distanceSequence[i]
            prev = "south"
        elif direction[i] == "west"¥
                or (direction[i] == "left" and prev == "north")¥
                or (direction[i] == "right" and prev == "south"):
            x -= distanceSequence[i]
            prev = "west"
        elif direction[i] == "north"¥
                or (direction[i] == "left" and prev == "east")¥
                or (direction[i] == "right" and prev == "west"):
            y += distanceSequence[i]
            prev = "north"
    distance = $\sqrt{x^2 + y^2}$
    dir = ""
    if y > 0:
        dir += "north"
    elif y < 0:
        dir += "south"
    if x > 0:
        dir += "east"
    elif x < 0:
        dir += "west"
    return [distance, dir]
```

Figure 6 Direction Solve Pseudocode

The get sequence function for odd one out class cannot identify the sequence if the question format does not meet the algorithm constraint. The reason is because odd one out data sequence consist mostly of word which can blend into the question itself that can make it harder for the program to find the odd one out words. Hence, the program made a constraint and limitation about the question format for the odd one out to be split between the question and the words by ":" character.

The solve function for odd one out class is a combination of node data structure that represent each word and hash table to store all the learned words. Firstly, each word node consist of the word itself and the relationship with other words accompanying with the score for each relationship. Then, the algorithm of reinforcement learning to choose the odd one word works by checking every word points. The word point can be obtained by summing all the relation points with other words in the data sequence. The word that holds the lowest point is assumed by the machine to be the odd one word based on the current machine knowledge. Therefore, in order to make the odd one out agent successful and efficient, the agent needs to learn a large sample of questions to gain the knowledge.

```
def solve(self):
    point = 16       # Maximum point a node can have is 15
    self.getSequence() # Parse the question to get the odd one out words
        # If the words score is the same then choose randomly
        if every odd one out words has the same points:
            decision = choose random number between 0 and total number of words -1)
            LowestWord = OddOneOutWords[decision]
        # Otherwise choose the lowest one
        else:
            for every word in odd one out words:
                if point > currentWord.getPoint():
                    point = currentWord.getPoint()
                    LowestWord = currentWord

        return LowestWord
```

Figure 7 Odd One Out Solve Pseudocode

The learning function would give a feedback to the machine if the machine tries to answer a question with a wrong answer, then the wrong chosen word get a reward of positive points with other words relationship, while the other words get the negative relationship point with the rest of the words. Meanwhile, when the machine tried to answer successfully, the words chosen is known to be unrelated with other words. Therefore, the chosen correct word get a feedback of negative relationship point with other words, while the other words get positive points between each other. Repeating this learning process will make the machine to have a knowledge about words that the machine has learned and would answer successfully if the questions has the learned words. However, to prevent the overfitting of the learning process, the relationship points need to be limited up to certain points, in this case it would be 5 and -5.

```
def learning(self, questions, answers):
    for i in range(0, length(questions)-1):
        decision = NULL
        while decision != answers[i]:
            set agent input text with questions[i]
            decision = self.solve()
            self.reinforce(decision, answers[i])

def reinforce(self, decision, correct):
    for i in range(0, length(wordList)-2):
        for j in range(i+1, length(wordList)-1):
            if wordList[i] != decision and wordList[j] != decision:
                wordList[i].addReward(wordList[j] by 1 if decision == correct else -1)
                wordList[j].addReward(wordList[i] by 1 if decision == correct else -1)
            else:
                wordList[i].addReward(wordList[j] by -1 if decision == correct else 1)
                wordList[j].addReward(wordList[i] by -1 if decision == correct else 1)
```

Figure 8 Reinforcement Learning Pseudocode

3.2 Statistics

Table 3 Odd One Out Knowledge Example

Country	Continent Category	Region Category
Japan	3	11
China	3	11
South Korea	3	11
Indonesia	3	12
Australia	5	19

The statistics used in this project is exclusively used on the odd one out class. The purposes of the statistics is to experiment and prove the theory of the reinforcement learning in context of odd one out questions. The statistics used consist of all the country names in the world with a total of 236 names. These country names will be used to generate a set of questions based on their categories such as continents for the program to learn and solve the odd one out country names questions. The table 3 above shows an example of country names with their own categories generate a set of questions based on the categories. Furthermore, with a help of testing tool of custom made class questionGenerator class, these country names can be collected, generated and outputted as a large sets of questions accompanied by their answers that will be used in odd one out agent. The table 4 below shows the complete categories of the country as a reference for generating the questions.

Table 4 Country Names Categories

Category Number	Region	Continent
1	Eastern Africa	Africa
2	Middle Africa	America
3	Northern Africa	Asia
4	Southern Africa	Europe
5	Western Africa	Oceania
6	Caribbean	
7	Central America	
8	Northern America	
9	Southern America	
10	Central Asia	
11	Eastern Asia	
12	South East Asia	
13	Southern Asia	
14	Western Asia	
15	Eastern Europe	
16	Northern Europe	
17	Southern Europe	
18	Western Europe	
19	Australia & New Zealand	
20	Melanesia	
21	Micronesia	
22	Polynesia	

This example of country names used to generate the questions to analyze the performance and capabilities of odd one out reinforcement learning can also be applied to other example as well. In other words, the methodology of experimental statistics is generalized for odd one out agent to learn as long as the agent can learn the words and the feedback.

3.3 Internal Design

Name: Solve a question use case
Actors: Users
Purpose: Solve the IQ question with the program

Typical Course of Events		
Step	**Actor Action**	**System Response**
1	User input the I.Q. question into the text field on the main window	
2	User click the solve button	
3		System get the question string, parse the question, solve the problem with respective agent, and show the result to the interface
Alternative Courses		
3.1	System does not understand the question or received empty string in which the system would return an empty string into the answer label on the interface	

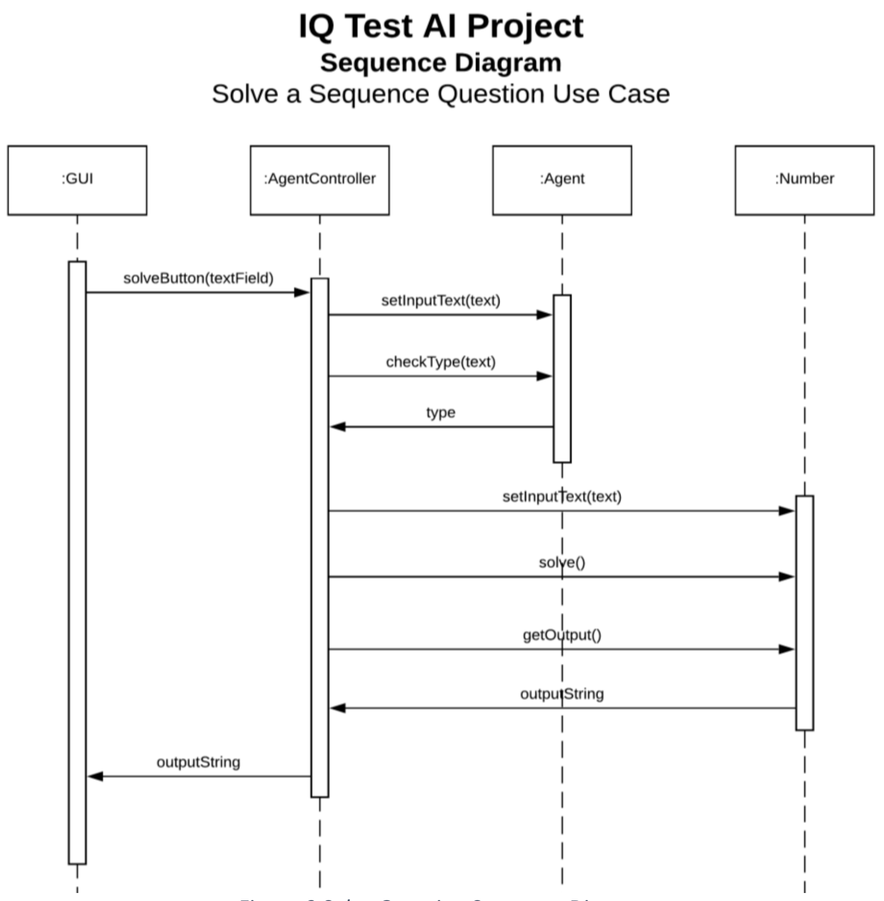


Figure 9 Solve Question Sequence Diagram

Name: Learn samples and test large inputs in odd one out agent use case
Actors: Users
Purpose: Make the agent learn samples and test it with large inputs file

Typical Course of Events		
Step	**Actor Action**	**System Response**
1	User click knowledge button on main window	
2		System open knowledge sub window
3	User select the knowledge file and click load knowledge button	
4		System load the knowledge file into the odd one out agent knowledge
5	User click the solve file button on main window	
6		System open the file management sub window
7	User select the question file and answer file then click Run button	
8		System load both question file and answer file, then run the odd one out agent for each question and compare the answer with answer file data. Show the result of the total correct and wrong answer to the interface
Alternative Courses		
2.1	System check there is no knowledge file attached in the field	
8.1	System check there is no question file nor answer file attached in the field and return error feedback	

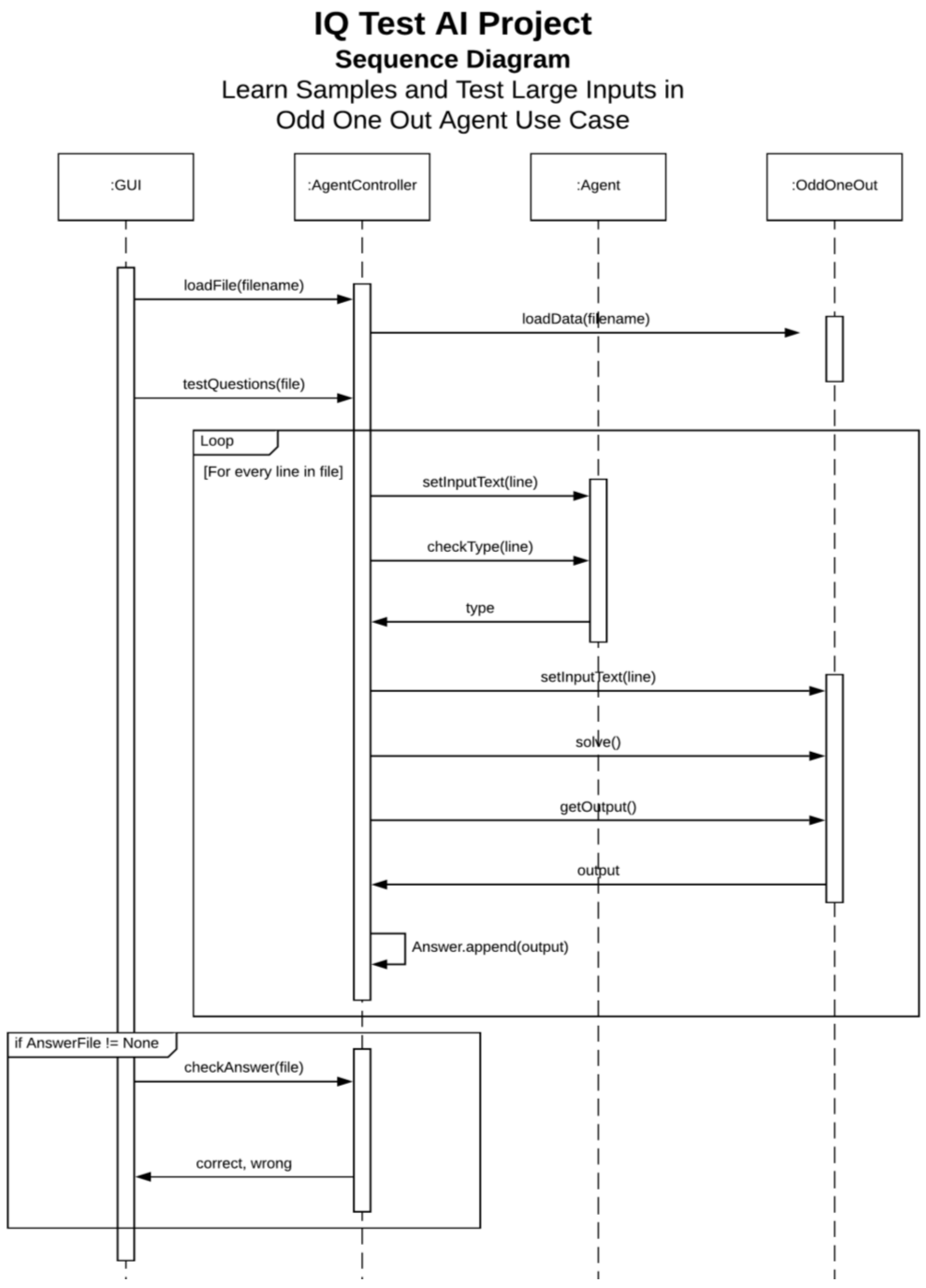


Figure 10 Reinforcement Learning Sequence Diagram

Assumptions of the sequence diagrams and use cases:

- Controller of the interface such as buttons is inside the GUI class
- GUI class manages both the view and controller functionalities
- All of sub windows are managed within the GUI class
- The checkType return the integer type that represent the respective agent
- loadFile and loadData for loading knowledge are exclusively for OddOneOut agent

3.4 Software Architecture

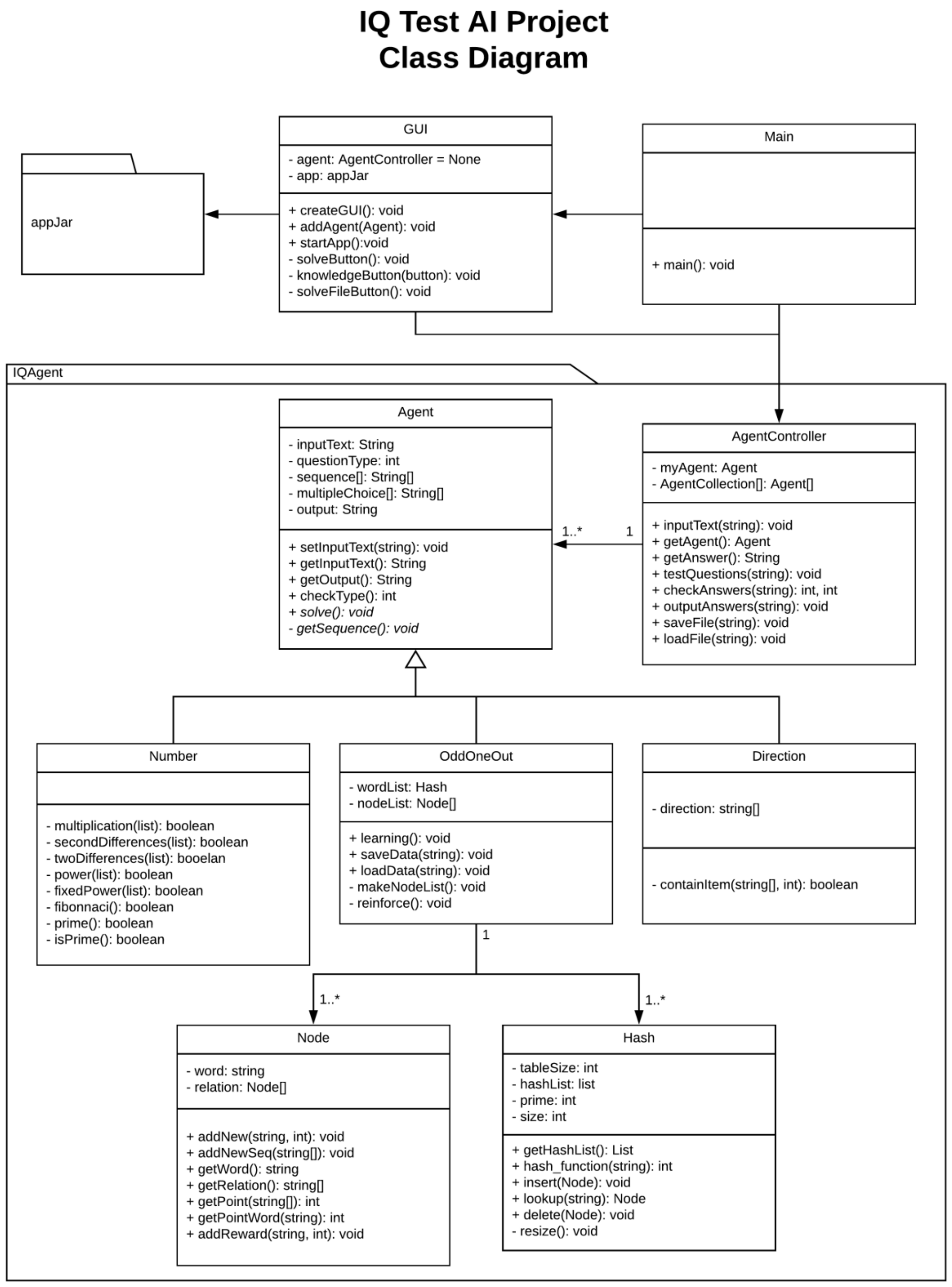


Figure 11 Program Class Diagram

The program is built in object oriented programming language design. Hence, the program consist of various classes and dependency. The classes of the program includes:

- Main class
  Class that has a purposes as a driver which manages both GUI class and AgentController class to work together as one system.
- GUI class
  A Class that manages both controller and view functionalities. Hence, this class has dependency with model package which can be represented as AgentController class
- AgentController class
  Class that represent as a driver of model package. The class manages the agent class and incoming input from GUI class. However, since the current version of the program does not need the model to notify the view, there is no dependency to the view class. This class has dependency with every agent implemented in the program.
- Agent class
  An abstract class that represent the agent to solve the question problem. This class two abstract methods.
- Number class
  A concrete class that inherited Agent class. This class represent the agent to solve number sequence problem and implemented the abstract method in relation with the problem.
- Direction class
  A concrete class that inherited Agent class. This class represent the agent to solve the direction problem and implemented the abstract method in relation with the problem.
- OddOneOut class
  A concrete class that inherited Agent class. This class represent the agent to solve the odd one out problem and implemented the abstract method in relation with the problem. In addition, this class has dependency with Node class and Hash class that works as a data structure.
- Node class
  A class that represent as node data structure. This class used as a basis data structure for the odd one out agent class methodology.
- Hash class
  A class that represent as hash table data structure.

4.0 Results

4.1 Outcomes

The results of the project is a deliverable in form of a program that is capable in solving various I.Q. questions type includes number sequence, direction problem, and odd one out question. The program can handle various pattern of the question that are commonly show in the real I.Q. test sample quite well. The quality of the program can be identified by the range of questions the program can solve which in this case the program can solve most of the direction problem except for some cases of two subject in direction problem, and solve the most common mathematical arithmetic, geometric, and recurrence sequence which can be considered bare minimum for solving the number sequence in I.Q. questions. In case of odd one out, the program can reliably solve most of the question if and only if the program has learned the words in the question before. Based on the functionalities and limitations mentioned before, the program quality can be considered as a good quality of a program that can pass the I.Q. test with minimum score. Therefore, the program can at least compete with human in I.Q. test that only has number, direction, and odd one out problems.

The program generate outputs and result of the solution based on the inputted question since the sole purpose of the program is to solve I.Q. question. However, additional results of statistics in odd one out question can be generated after a large file input represented as question is solved the program. The results contains the statistic and percentage of the number the odd one out questions are correctly solved and the wrong answers.

4.2 Experimental Statistics

The only experimental statistics used in this project is the learning process of odd one out agent. In order to identify and analyzed the AI behavior toward reinforcement learning in terms of odd one out problems, the statistics used are the set of questions generated by custom made testing class. The experiment performed by generating the set random odd one out question in large numbers and the questions is used as a material for the program to learn with reinforcement learning. The number of questions would be the variable of the experiment where hypothetically as the number of questions increased, the more efficient and knowledgeable the program becomes. Hence, if a program learned a very large number of question, the percentage rate of wrong answer would be very minimal. The table below shows the statistics of odd one out agent efficiency in different number of learning sample with 1000 questions tested or verification after learning stage.

Table 5 Reinforcement Learning Experimental Statistics

Number of Learning	**Wrong Answer Mean**	**Wrong Answer Median**	**Standard Deviation**
1000 questions	257.36	254	13.83
2000 questions	148.5	151.5	16.32
3000 questions	98.7	100.5	9.73
4000 questions	72.7	76.5	13.92
5000 questions	51.1	51	8.49
6000 questions	42.9	43	5.02
7000 questions	34.2	34.5	4.63
8000 questions	25.4	27	4.27
9000 questions	23.1	23.5	4.72
10000 questions	23.9	25	3.95

4.3 Performance

The performance of the program can be classified as linear complexity $O(n)$ in general since most of the agent has linear complexity on average and best scenario. The number agent complexity can be calculated with the sum of the parsing algorithm and all the prediction algorithm time complexity. The parsing question has $n + m$ time processing where n is the sequence length and m is the words of the question since the program is parsing the question for every word. All prediction algorithm would have $n + c$ time processing where n is the sequence length and c as constant with an exception of prime sequence prediction algorithm. Predicting prime needs $\left(n \times {}^{n}/_{3}\right) + c$ where n is the sequence length and c is the constant. Therefore prime prediction has $O(n^2)$ time complexity. The total time complexity for number agent would be quadratic time complexity $O(n^2)$ in worst case scenario.

The time complexity of direction agent can be calculated with the sum of parsing algorithm processing time and the solve function algorithm time complexity. The parsing algorithm is the same as in the number agent which has linear time complexity $O(n)$. The direction solve agent has processing time of $n + c$ where n is the sequence length and c as constant. Therefore, the time complexity for direction agent in all scenario would be linear time complexity $O(n)$.

The time complexity for odd one out agent can be calculated with the sum of parsing algorithm and solve algorithm complexity. The parsing algorithm has the same complexity as other agent which is linear complexity $O(n)$. The solve algorithm only consist of lookup of the words in hash table which has linear complexity on worst scenario and comparing the all the word points which also has linear complexity. Therefore, the time complexity for solving an odd one out problem is linear complexity $O(n)$. However, this does not apply to reinforcement learning of the agent. The learning algorithm consist of solve algorithm and reinforcement algorithm which has quadratic time complexity because it needs to give the reward for every words relationship in the question. Each word has a relationship number of n-1 with n as the total words in the question. Hence, the processing time for updating every word relationship would be $n(n-1)$. In other words, the learning algorithm for odd one out agent is quadratic time complexity $O(n^2)$.

The space complexity of every agent and algorithm in this program is linear space complexity because every algorithm used an array to store the sequences data only. Therefore, the space complexity would be the same as the number of the sequence data. However, with a little exception of the odd one out agent which uses hash table that has procured specific designated space for the table, but the space would not increase even if the sequence data increased. Hence, the hash table can be still considered as linear space complexity.

Table 6 Agent Time Complexity

Algorithm	Time Complexity	Space Complexity
Number Sequence Agent	$O(n^2)$	$O(n)$
Direction Agent	$O(n)$	$O(n)$
Odd One Out Agent	$O(n)$	$O(n)$
Odd One Out Learning	$O(n^2)$	$O(n)$

5.0 Analysis

5.1 Analyze Statistics and Outputs

The outputs of the experimental statistics shown on the table 5 give an explanation of the methodology efficiency in solving the odd one out problems. The statistics suggest that as the number of samples for the program to learn increased, the percentage of correct answers is increase as well. The percentage of correct answers show a similarities with logarithm function $y = log_b(x)$ where $b > 1$ and for the wrong answers, $b < 1, b > 0$. The reason is because the program has learned most of the words relationship if the number of samples is big enough. Hence, the differences of percentage correct answers rate between later number samples is small. The figure 12 below shows the line graph of number samples versus both answers rate.

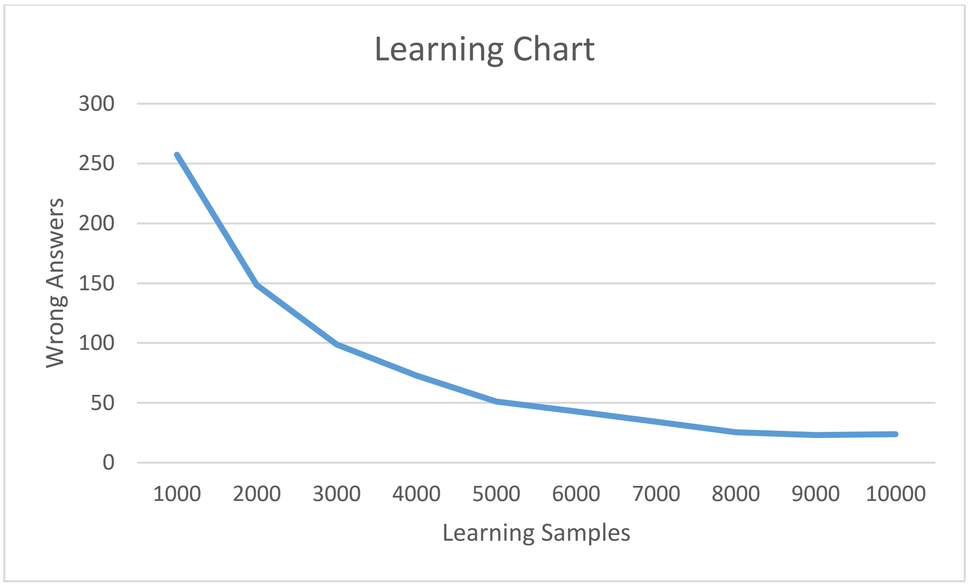


Figure 12 Learning Samples Line Chart

Each number of learning variable is calculated with 10 samples. Therefore, the statistics produced in the experiment does not represent the whole population but reliable enough to be used for statistical analysis. The median in the small learning samples has a tendency to produce a skewed graph which means majority of the results are actually bigger than the mean itself and there are some extreme datum that would affect the means and produce skewed graph. This symptom explain the inefficiency of the program with small learning samples that would produce correct/wrong answers rate outside the range of the graph. However, these symptom does not show in the larger learning sample because the program is stable enough to solve the problems.

The standard deviation of the wrong answers in the statistics indicates the reliability of the program to solve the problem. The statistics outputs from 1000 learning samples up to 5000 learning samples shows the instability of standard deviation in which at some learning samples the standard deviation is big while on the other cases the standard deviation is small without any trend. This results indicates that the program has answered wrong questions in a big range on one learning sample type. In addition, it proves that the program is not reliable enough to solve the problem as there might be a possibility the program produce a large number of wrong answer in some cases. However, starting 6000 learning samples and more, the program starts to generate a stable standard deviation with a decreasing trend every time learning samples increases. In other words, the program is assumed to be reliable enough if the program has learned 6000 learning samples.

5.2 Discussions

Based on the researches collected for this project relating the methodology and theory supporting the odd one out reinforcement learning program, the analysis from experimental statistic has proved the base theory of reinforcement learning methodology to be efficient and reliable for odd one out problems. The previous statement is backed up by the results of the statistics showing more than 95% questions answered correctly most of the time when the program has reached the point of good reliability for solving the odd one out problem. Furthermore, its effectiveness keep increasing as the learning samples increases which also proves the statement of reinforcement learning that the program would make a moves and learn its past mistakes for getting bigger reward in the future.

When a program learned a certain learning samples for very long time or very big samples, the program is deemed to be over fitted to the certain learning samples. In other words, the program is too familiar for the learning samples which makes the program seems to be designed to solve that particular learning samples only. Hence, the program efficiency is not increasing when given a verification test sample different from the learning samples. However, in this case the statistic outcomes for the odd one out program does not show any significant symptom of overfitting. The most possibility of overfitting occurred after 9000 learning samples when the program start to output the similar wrong answer mean and standard deviation with very little difference and without constant trend. But these analysis can be wrong due to low number of statistic sample and statistic reliability. Therefore, the hypothesis of overfitting on reinforcement learning specifically for this program has not yet formally supported.

6.0 Future Work

Future research that can emanate from this project are the addition of mathematical arithmetic, geometric, and recurrence that might come out in I.Q. test. In addition with the mathematical sequence, the research of efficient algorithm to predict the specific sequence is also needed in order to solve a problem before generating the next number in the sequence. If possible, the prediction algorithm for number sequence need to be at most quadratic time complexity to prevent the program process too much time in worst case scenario.

Another future research plan includes researching a new methodology and techniques to fine more efficient time algorithm for reinforcement learning in odd one out AI. Currently the machine learns processing time is significantly affected by the total number of words in the machine knowledge database and the number of odd one out words in the question. Hence, the worst case scenario of holding a vast words of database while handling set of questions that has large number of odd one out words choices, the program would have a long processing time. A further research of improving the data structure, algorithm design and methodology of the algorithm regarding the reinforcement learning can greatly help the understanding of reinforcement learning subject and odd one out problems characteristics.

Other minor improvement plan for the program quality includes more research on the I.Q. test sample to gather the keywords that are usually used n I.Q. question for better question recognition reliability. Other plan includes an implementation of direction problem functionality regarding two subject direction problem. For example, the program can handle direction problem that has two people walking at the same time and asked the distance between the two people.

7.0 Conclusion

In conclusion, the project of computer solving I.Q. question specifically for number sequence, direction, and odd one out problems has meet its project main requirements based on the project specification proposal. The project has successfully deliver a working program with proper graphic user interface along with all the agents that are able to solve the problem reliably. Furthermore, the testing report indicates that the program does not have any significant problem and able to work properly without any logical and technical error. An experimental statistics also being done for the purposes to analyze the odd one out agent behavior in solving the problem. The results of the experiment are able to prove the base hypothesis of the program concept methodology. In overall, the project can be considered as a successful project and additional implementation and research based on this project are encouraged to improve the quality of the project and to learn the characteristics of the I.Q. questions problem with program algorithm techniques.

8.0 Bibliography

Benzali, K. (2018). *Computers Doing IQ Tests, and Pick the 'Odd One Out' Project Specification Proposal*. Monash University.

Carter, P. J. (2008). *Advanced IQ Tests : The Toughest Practice Questions to Test Your Lateral Thinking Problem Solving and Reasoning Skills*. London, United Kingdom: Kogan Page Ltd.

Dowe D. L., H.-O. J. (2012). IQ tests are not for machines, yet. *Intelligence, 40*(2), 77-81. doi:10.1016/ j.intell.2011.12.001

Erwin Hilton, Q. L. a. T. P. (2017). *Spatial IQ Test for AI.* Massachusetts Institute of Technology, Retrieved from https://dspace.mit.edu/handle/1721.1/113004

J. Hernandez-Orallo, F. M.-P., U. Schmid, M. Siebers, D. L. Dowe (2016). Computer models solving intelligence test problems: Progress and implications. *Artificial Intelligence, 230,* 74-107. doi:doi:10.1016/j.artint.2015.09.011

Martina, M. (2014). *Applying Inductive Programming to Solving Number Series Problems -- Comparing Performance of Igor with Humans.* (Master of Applied Computer Science), University of Bamberg,

Phillip J. Carter, K. R. (2007). *The Ultimate IQ Test Book : 1000 Practice Test Questions to Boost Your Brain Power*. London, United Kingdom: Kogan Page Ltd.

Pritika Sanghi, D. L. D. (2003, 13-17 July 2003). *A Computer Program Capable of Passing I.Q. Test.* Paper presented at the 4th ICCS International Conference on Cognitive Science & 7th ASCS Australasian Society for Cognitive Science, Sydney, NSW, Australia.

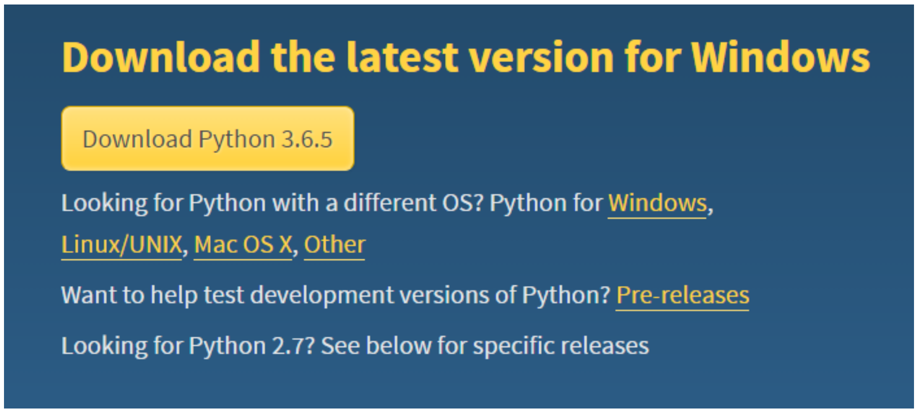
Richard S. Sutton, A. G. B. (2018). 1.1 Reinforcement Learning. In *Reinforcement Learning: An Introduction, Second Edition (Draft)* (pp. 548). Massachusetts: The MIT Press.

Turing, A. M. (1950). COMPUTING MACHINERY AND INTELLIGENCE. In *Mind* (Vol. 59, pp. 433-460).

9.0 Appendices

9.1 Production and Deployment

The program is built in Python programming language environment. Hence, it is important to install the Python language package into the computer environment. The package can be downloaded on https://www.python.org/downloads/ website and download the latest version.



Download the latest version for Windows

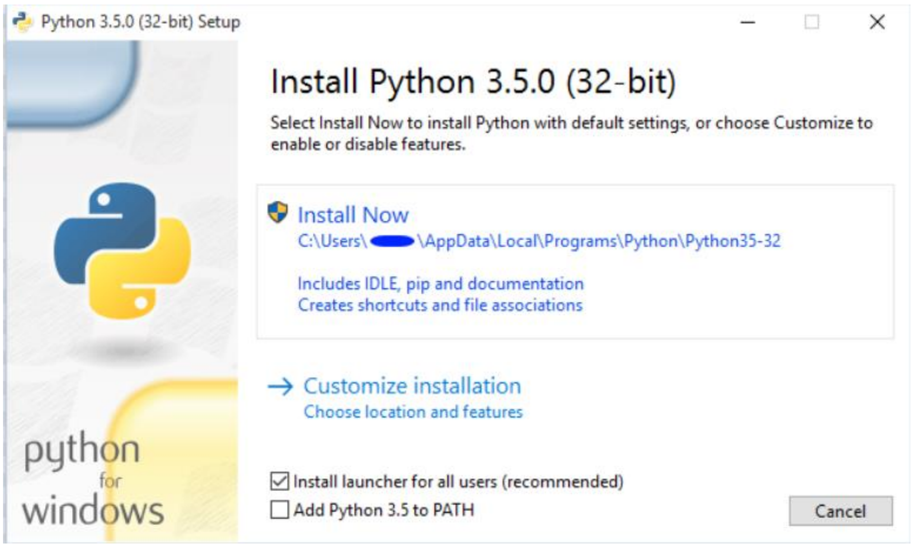
Download Python 3.6.5

Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other

Want to help test development versions of Python? Pre-releases

Looking for Python 2.7? See below for specific releases

After downloaded the Python installer, run the installer and proceed the installation with default or custom preferences.



Python 3.5.0 (32-bit) Setup

Install Python 3.5.0 (32-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

Install Now
C:\Users\[redacted]\AppData\Local\Programs\Python\Python35-32

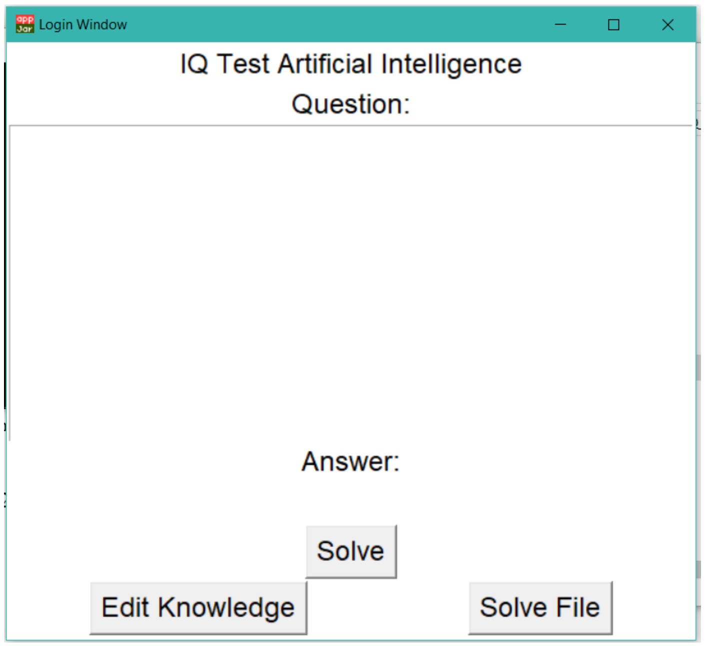
Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

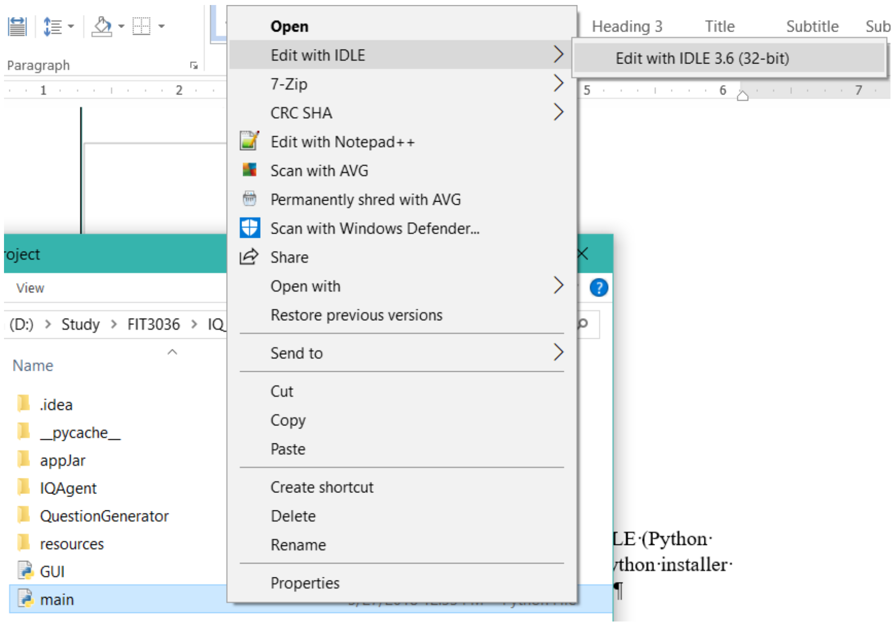
☑ Install launcher for all users (recommended)
☐ Add Python 3.5 to PATH

Cancel

After python is installed, the program can execute directly by double clicking the main.py. The other way to run it is through command prompt by changing the directory into the program path and run the program by entering “py main.py” or “python main.py”. The program GUI can be seen in the screenshot below.



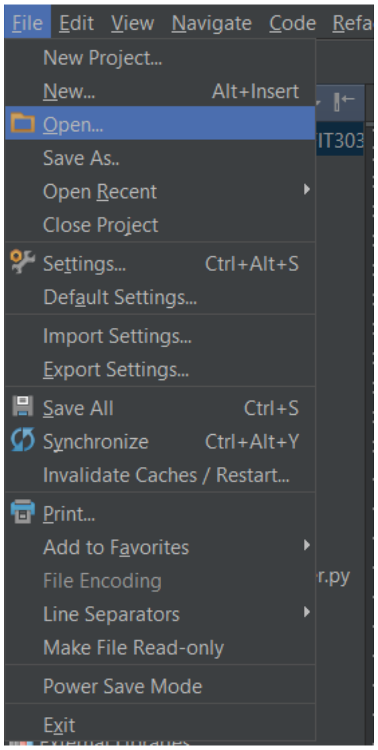
However, in order to see the code of the program, the file need to be opened with IDLE (Python Integrated Development and Learning Environment) which is installed together in Python installer package by default. Right click on the python file and click "edit with IDLE" option.

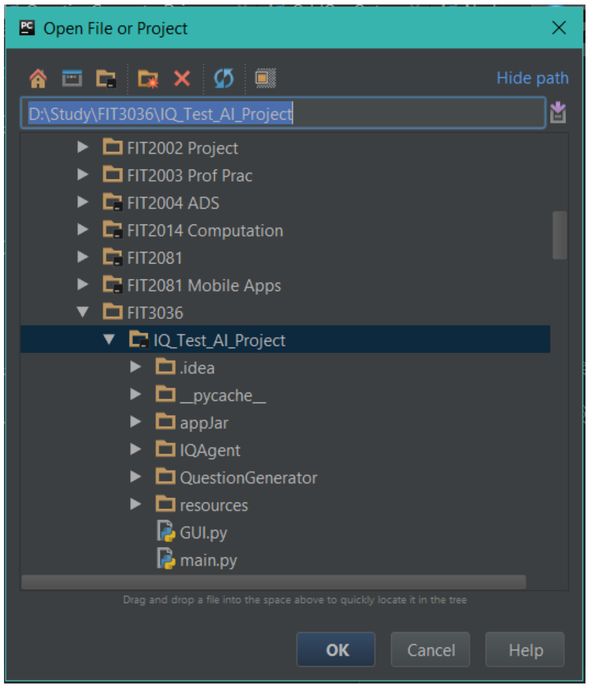


This project specifically uses PyCharm IDE in the process of implementation of the program. Additional option to open the program with PyCharm IDE can also be done. First, PyCharm needs to be installed by downloading the installer from https://www.jetbrains.com/pycharm/.

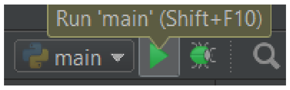
PC
PyCharm
Python IDE
for Professional Developers
DOWNLOAD NOW
Full-fledged Professional or Free Community

After installing the PyCharm, open the PyCharm and the program can be opened by choosing File > Open > Choose IQ Test AI Project File > OK.





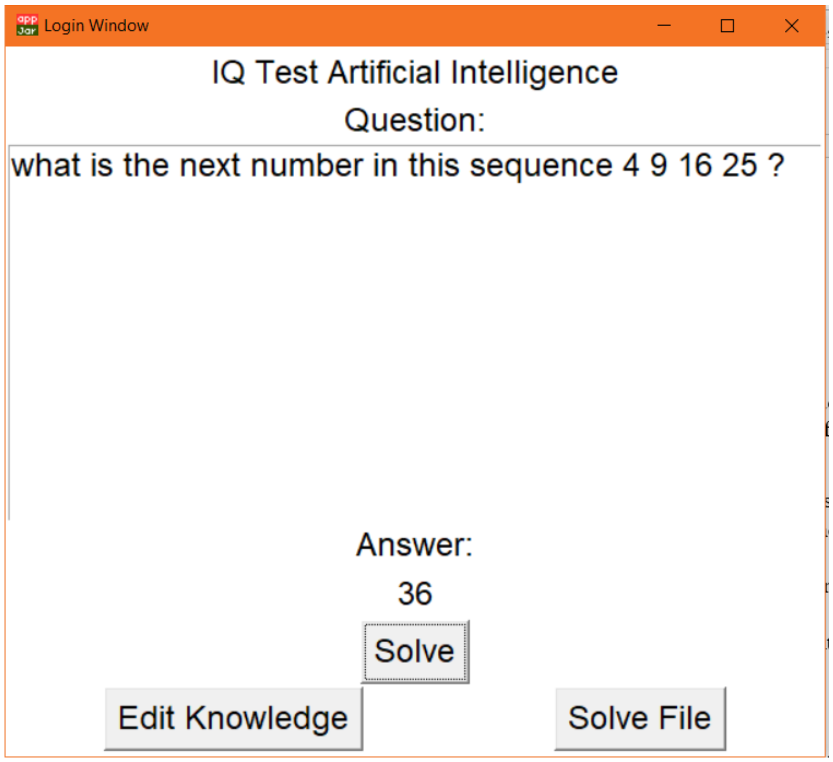
To run the program through PyCharm, simply click the run button on the top right of the program. Make sure that the main file is selected since the main function is inside the main file.



9.2 Program Guide

The program is launched with main window at first which consist of large text box in the middle of the program, answer label below the text box, solve button, edit knowledge button and solve file button. These elements have their own functionality and purposes which includes:

- Text box: allowing the user to input I.Q. questions in the text box for the program to solve it
- Answer label: the label will be updated with solution every time the program solve the question
- Solve button: the button that trigger the program to start solve the question inputted in the text box
- Edit knowledge button: show the sub window of odd one out knowledge management
- Solve file button: show the sub window of input output file management



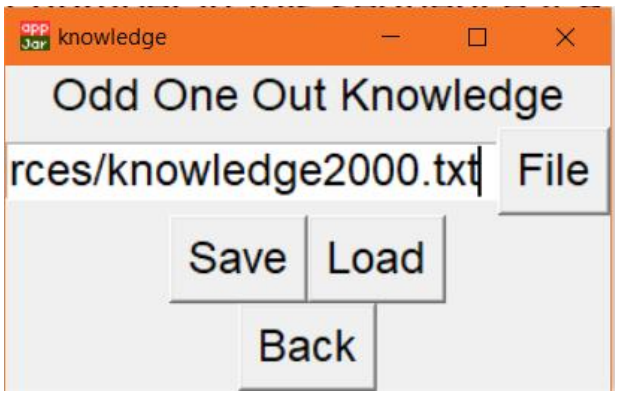
what is the next number in this sequence 4 9 16 25 ?

36

There are other sub windows as stated previously which are knowledge sub window that manages odd one out agent knowledge and file sub window that manages input and output of file. Each of the sub window has its own elements that are listed as below:

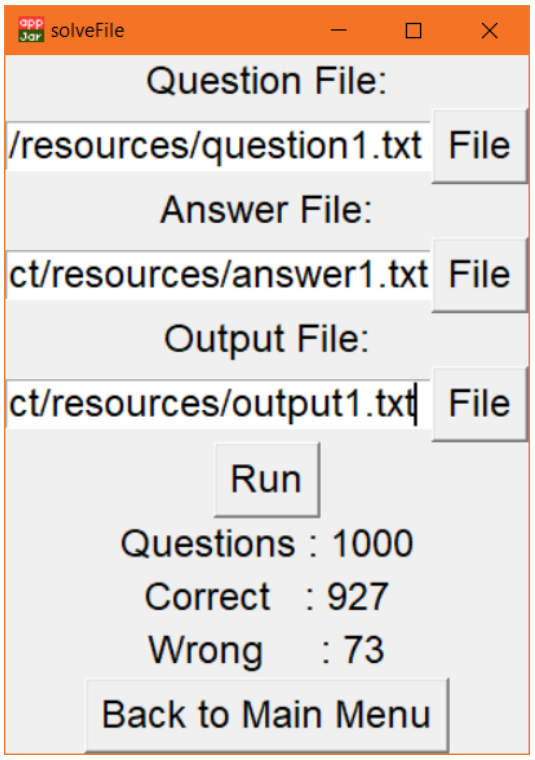
Knowledge sub window:

- Knowledge file slot: Input file slot for the odd one out agent to manage the knowledge
- Load button: button that load the knowledge file inputted into odd one out agent
- Save button: button that save the current odd one out agent into the inputted file name
- Back button: close the knowledge sub window



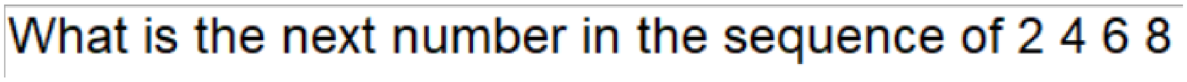
File management sub window

- Question file slot: input file slot for question file that consist of set of questions
- Answer file slot: input file slot for answer file that consist of the answer based on the set of question from question file
- Output file slot: input file for output file that would represent the output solution for each questions in question file after the program has solved all the questions
- Run button: a button that trigger the program to solve all the question in question file
- Back to main menu button: close file sub window

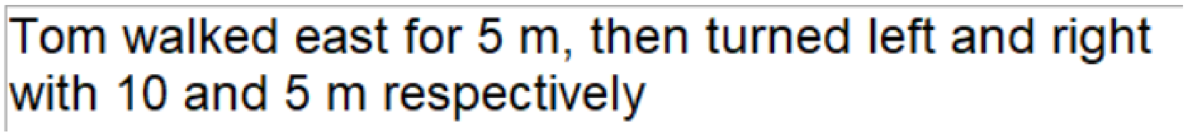


9.3 Parameter Range

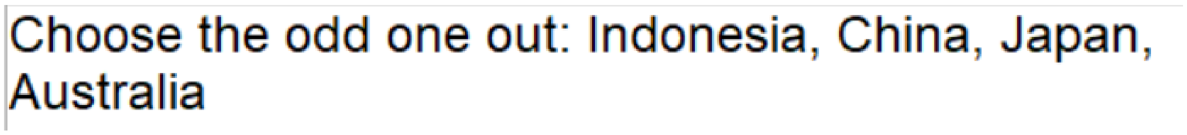
The program can handle I.Q. questions limited only to number sequence, direction problem, and odd one out problem only. For every problem type, the question needs to have the questions string before the data. For number sequence the question need to indicate that the question is sequence problem by including "sequence", "number" or other words related to the number sequence problem. The example can be seen in the pictures below.


What is the next number in the sequence of 2 4 6 8

For direction, the question need to include the direction words along with the distance with the same order and length.

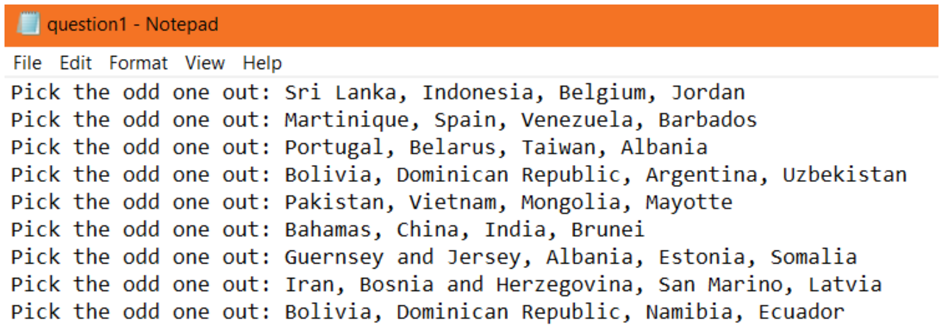

Tom walked east for 5 m, then turned left and right with 10 and 5 m respectively

For odd one out, the question need to have ":" special character to distinguish and split between the question string and the odd one out words data. In addition, every odd one out words need to have "," coma character as separator between each word


Choose the odd one out: Indonesia, China, Japan, Australia

The file management sub window manages both input and output file. The input file consist of question file and answer file. The only requirement for run button to work successfully is the existence of question file. If the question file is the only file inputted, the program still able to run without any output to the interface or file. Once answer file inputted as well, the wrong and correct answer on the interface automatically updated once run button clicked.

The structure of the question file that consist of set of questions need to have a format of each line for each question. The format also required in the answer file as well. Hence, each answer file line represent the answer of the respective question line.



```
Pick the odd one out: Sri Lanka, Indonesia, Belgium, Jordan
Pick the odd one out: Martinique, Spain, Venezuela, Barbados
Pick the odd one out: Portugal, Belarus, Taiwan, Albania
Pick the odd one out: Bolivia, Dominican Republic, Argentina, Uzbekistan
Pick the odd one out: Pakistan, Vietnam, Mongolia, Mayotte
Pick the odd one out: Bahamas, China, India, Brunei
Pick the odd one out: Guernsey and Jersey, Albania, Estonia, Somalia
Pick the odd one out: Iran, Bosnia and Herzegovina, San Marino, Latvia
Pick the odd one out: Bolivia, Dominican Republic, Namibia, Ecuador
```

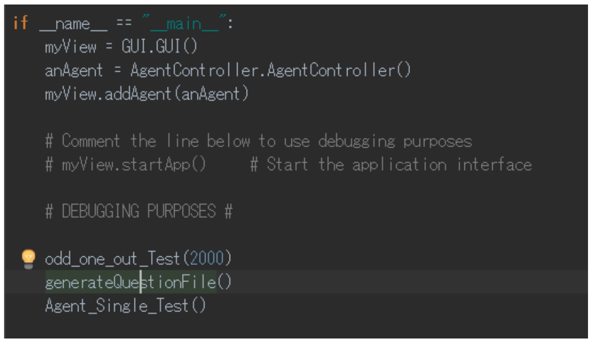
9.4 Automated Testing Procedure

There is an additional class designed exclusively to test the program of odd one out agent by generating a large numbers of randomly generated odd one out questions out of a database of words and its category. This custom made testing tool uses questionGenerator.py, questionGeneratorDriver.py classes and “odd_one_out_country.txt”, “odd_one_out_country_continent.txt”, “odd_one_out_country_region.txt” text files as the database for the question generator. The generator classes can be found under question generator file package and odd one out country names text files under resources file.

The guide how to use these classes can be handled just by instatiating the questionGeneratorDriver as an object. The questionGeneratorDriver has methods that are specifically design to make an agent learn a specified number of questions or test a specified number of questions. The method names are testSamples() and learnSamples() respectively. In addition, a method outputQuestions() has a functionality to generate the set of questions and answers then write it into the specified questions filename and answer filename in the parameter.

- learnSamples() parameter:
    - baseFile represent the words database file
    - categoryFile represent the category of the words
    - categoryNum integer represents the total type of the category in the category file
    - totalLearn integer represents the total number of samples.
- testSamples() parameter: basically the parameter is exactly the same with learnSamples function with totalTest act as number of samples.
- outputQuestions() parameter:
    - questionFile represents the question filename
    - AnswerFile represents the answer filename, baseFile represents the words database file
    - categoryFile represents the category of the words
    - catNum integer represents the total type if the category in category file
    - totalNum integer represents the total number of samples
    - answerType integer represent the type of answer output, 1 is the type of the real solution, 0 is the type of the index of the solution

Before trying the debugging tool in the main.py, make sure to comment the line of myView.startApp() since this line will start the application and stop the interpreter to process the next line.



```
if __name__ == "__main__":
    myView = GUI.GUI()
    anAgent = AgentController.AgentController()
    myView.addAgent(anAgent)

    # Comment the line below to use debugging purposes
    # myView.startApp()     # Start the application interface

    # DEBUGGING PURPOSES #

    odd_one_out_Test(2000)
    generateQuestionFile()
    Agent_Single_Test()
```