PP/018/SMS ke IV/ 3th Revision

LABORATORY
INFORMATICS ENGINEERING
INDUSTRIAL TECHNOLOGY
FACULTY
AHMAD DAHLAN UNIVERSITY

PRACTICUM INSTRUCTIONS

ARTIFICIAL INTELLIGENCE

Editor:
Anna Hendri Soleliza Jones

2020

PREFACE

Alhamdulillah, the author of the presence of Allah SWT who has provided health and opportunities to the author, so that this 3rd Edition of Artificial Intelligence Practices Manual can be completed. Quite a lot of changes were made in revising this Artificial Intelligence Practicum Manual based on the input given by the Lecturers in the Informatics Engineering Study Program when reviewing material in the curriculum. Based on this input, the Artificial Intelligence Practicum Handbook was summarized with expectations in accordance with the course material provided. Changes to the material are very much found in this revised edition, especially the use of tools that as appropriate use Python software. This is aimed so that students can get to know programming languages that intersect with other subjects, such as machine learning. Once again, the authors would like to thank my colleagues in Informatics Engineering who have indirectly contributed ideas / thoughts to support the formation of this Artificial Intelligence Practices Manual. Hopefully this practical guide can be useful in accordance with the expectations of the author.

Yogyakarta, 16 January 2020
Author

Author

Authors

Anna Hendri Soleliza Jones, S.Kom., M.Cs.

REVISION PAGE

The undersigned below:

Name	: ………………………………
NIK/NIY	: ………………………………
Position	: ………………………………

Hereby declare the implementation of the Practicum Revision ……………………….. for the Informatics Engineering Study Program has been carried out with the following explanation:

Num	Revision Details Information (Per Meeting)	Revision Date	Module Number
1	Changes to the material compiled according to the RPS, and the cases provided. The software used is matlab, turbo prologue and GUI	2014	
2	Changes to the material, namely the addition of software agent material and hierarchy planning using the matlab application. Thus the tools used in addition to the Turbo Prologh also use MATLAB	2016	
3	• Software changes in material 1 - 6, and 8 using Python • Addition of material 9 Custom Python Package Creation • Material 10 uses GUI software	2019	
4	Changes to module templates	2020	

Yogyakarta, 16 January 2020
Author

Author

HALAMAN PERNYATAAN

Yang bertanda tangan di bawah ini:

Nama	: ....
NIK/NIY	: ....
Jabatan	: ....

Menerangkan dengan sesungguhnya bahwa Petunjuk Praktikum ini telah direview dan akan digunakan untuk pelaksanaan praktikum di Semester .... Tahun Akademik .... di Laboratorium ...., Program Studi Teknik Informatika, Fakultas Teknologi Industri, Universitas Ahmad Dahlan.

Yogyakarta, Tanggal Bulan Tahun

Mengetahui,	
Ketua Kelompok Keilmuan	Kepala Laboratorium ....
....	....
NIK/NIY.	NIK/NIY. ....

VISION AND MISSION OF INFORMATICS ENGINEERING

VISION

To become an internationally recognized and superior Information Studies Program in the field of Informatics and based on Islamic values.

MISSION

1. Carry out education in accordance with competencies in the field of Informatics that are recognized nationally and internationally
2. Increasing the research of lecturers and students in the field of Informatics that is creative, innovative and effective.
3. Increasing the quantity and quality of national and international scientific publications
4. Carry out and improve community service activities by lecturers and students in the field of Informatics.
5. Organizing activities that support the development of study programs by involving lecturers and students.
6. Carry out cooperation with national and international level institutions.
7. Creating Islamic life in the study program environment.

REGULATION OF INFORMATICS ENGINEERING LABORATORY

LECTURER / COORDINATOR OF PRACTICUM

1. The lecturer must be present at the practicum for at least 15 minutes at the beginning of the practicum and sign the presence of practicum.
2. The lecturer prepares the practicum module, about the selection of assistant, pre-test, post-test, and response by coordinating with the assistant and practicum expert.
3. The lecturer coordinates with the practicum assistant coordinator for the practicum evaluation every week.
4. The lecturer signs the practicum assistant contract letter and the practicum assistant coordinator.
5. Lecturers who are not present in certain practicum slots without notification for 2 consecutive weeks receive a reprimand from the Head of the Laboratory, if it continues for the next 2 weeks then the Head of Laboratory has the right to change the practicum coordinator in that slot.

STUDENT

1. Students must be present 15 minutes before the practicum activities begin, and dispensation is 15 minutes late for a clear reason (unless the assistant determines otherwise and the standard hours are hours in the Laboratory, being late more than 15 minutes may not enter the practicum & are considered Inhal).
2. Students who do not take the practicum for any reason, must take INHAL, a maximum of 4 times the practicum and if more than 4 times the practicum is considered FAIL.
3. Students must dress neatly in accordance with the provisions of the University, as follows:
   a. Must not wear T-shirts, including if they are covered by Almamater Jackets / Jackets (Men / Women) and Hats must be removed.
   b. It is not allowed to wear tight clothes, minimal headscarves and hair must be completely covered with headscarves, should not be seen on the forehead or on the back (especially for women).
   c. Must not wear minimal clothes, even when sitting, the waist must be tightly closed (Male / Female).
   d. Men shouldn't wear bracelets, earrings or women's accessories.
4. Students must not eat and drink during practicum activities, must maintain cleanliness, safety and order while attending practical activities or while in the laboratory (may not litter the paper, pieces of paper, candy wrappers on the carpet floor or on in CPU space).
5. Students are prohibited from leaving the practicum without permission from the Assistant or Laboratory Assistant.
6. Students must put shoes and bags on the shelves / lockers that have been provided.
7. During the practicum, NGENET / NGE-GAME is prohibited, except for practicum eyes that require or use Internet facilities.
8. Students are prohibited from removing network cables or practicum power cables without the knowledge of the laboratory assistant
9. Students must have FILE Practicum instructions and be used during the practicum and must be prepared before the practicum takes place.
10. Students are prohibited from cheating such as cheating or copying other students' work when practicum is taking place or post-test which is a practicum assignment.

11. Students are prohibited from changing computer software / hardware settings to either add or subtract without the request of an assistant or laboratory assistant and do something that can harm the laboratory or other practicum.
12. Assistant, Practicum Coordinator, Head of Laboratory and Laboratory Assistant have the right to reprimand, warn and even ask students to leave the practicum room if it is felt that you are disturbing other students or not carrying out practicum activities as appropriate and or do not comply with applicable laboratory rules.
13. Violation of one or more of the rules above, the practicum value at the meeting is considered 0 (ZERO) with INHAL status.

LAB ASSISTANT

1. Assistant must be present 15 minutes before the practicum begins (confirmation to the coordinator if there is a delay or unable to attend).
2. Assistant who cannot attend MUST find a replacement, and report to the Assistant Coordinator.
3. Assistants must dress neatly in accordance with the provisions of the University, as follows:
   a. Must not wear T-shirts, including if they are covered by Almamater Jackets / Jackets (Men / Women) and Hats must be removed.
   b. It is not allowed to wear tight clothes, minimal headscarves and hair must be completely covered with headscarves, should not be seen on the forehead or on the back (especially for women).
   c. Must not wear minimal clothes, even when sitting, the waist must be tightly closed (Male / Female).
   d. Men shouldn't wear bracelets, earrings or women's accessories.
4. Assistants must maintain cleanliness, security and order while attending practicum activities or while in the laboratory, reprimanding or reminding if there are students who cannot maintain cleanliness, order or politeness.
5. Assistant must be able to tidy up and secure the presence of practicum, Value Card and orderly in entering / Input values online / offline.
6. The assistant must be able to act professionally as a practicum assistant and can be an example for students.
7. Assistants must be able to provide explanations / understanding needed by students regarding practicum material that is assisted so that students can carry out and do practical work well and clearly.
8. Assistants are not allowed to talk alone let alone to make noise.
9. The assistant is asked to coordinate to ask the student to turn off the computer for the last schedule and an assessment of the student's work has been done.
10. Assistants are required to turn off the LCD Projector and computer assistant / student when not in use.
11. Assistants are not permitted to use internet access other than for practicum activities, such as Youtube / Game / SocMed / Streaming Movies on student computers.

OTHERS

1. At the time of response, you should use shirts for men and women for students and assistants.
2. The absence of practicums for any reason is considered INHAL.
3. The practicum permit follows the SIMERU / LECTURAL permit rules.
4. Those who have no interest in practicum are prohibited from disturbing students or making a commotion / noise.

5. The use of the lab outside the practicum hours is maximum until 21.00 by showing a permit from the Head of the Laboratory of Informatics Engineering Study Program.

   Yogyakarta, 15 February 2019
   Head of the Informatics Engineering Practicum Laboratory

   ……………………………………..………
   NIK/NIY. ……………………………....

TABLE OF CONTENTS

LIST OF PICTURE

LIST OF TABLES

PRACTICUM 1 TRACKING CONCEPT WITH PYTHON

	Meeting	: 1
	Total Time Allocation	: 90 minutes
•	Pre-Test	: 15 minutes
•	Practicum	: 60 minutes
•	Post-Test	: 15 minutes

	Total Rating Score	: 100%
•	Pre-Test	: 25%
•	Practicum	: 35%
•	Post-Test	: 40%

1.1 OBJECTIVES AND INDICATORS OF ACHIEVEMENTS

After following this practicum, students are expected to:

1. Knowing how to use python tracking.

The achievement indicator is measured by: (adjusting to the RPS)

1. Able to apply algorithmic strategy theory, data structure, statistical inference to solve problems in the tracking process into the programming language

1.2 SUPPORTING THEORY

History of Python

- Python is a multipurpose programming language because it can be used for various applications such as website development, mathematical computing, graphical interfaces, and others.
- Developed by Guido Van Rossum since the 1980s and first released in February 1991

Difference between Python and Other AI Languages

In the field of artificial intelligence Prolog and Lisp are popular languages, especially among academics. Prolog is popular in Europe, while in America researchers develop a similar programming language, namely LISP. Lately Python has emerged as a very popular programming language in the field of artificial intelligence.

Unlike Prolog which uses the concept of logic programming, Python can support several programming concepts, such as object-oriented programming, imperative programming, functional programming, procedural programming. The amount of support for this type of programming is because the syntax of the Python programming language is very simple and flexible with a very comprehensive package support. The very large and active open-source community produces a large number of packages that can be used to solve problems in the field of artificial intelligence. For example TensorFlow developed by the Google Brain team, Scikit-learn is widely used for machine learning, Hard for the development of neural networks, NLTK for natural language processing, and many more.

1.3 PRACTICUM STEPS

Package in Python

One of the strengths of Python is the very diverse and broad package support. Therefore, applications developed using Python can adjust their complexity by utilizing packages available in Python. Package installation can be done in stages depending on the needs of the application.

1.1.1.1 Package Installation

Adding packages to Python can be done via the pip install <package name> command. If the pip is not installed, complete the following steps:

- Download `get-pip.py` via the following link: https://bootstrap.pypa.io/get-pip.py.
- Run the following command for pip installation:

```
python get-pip.py
```

- If pip is installed, the Python package can be installed using the following command:

```
pip install <nama_package>
```

Packages that have been registered with PyPI (the official repository for python packages) can be installed directly by calling the package name. For example, if you want to install the Numpy package for numerical computing in Python, then the following command can be run:

```
pip install numpy
```

1.1.1.2 Using Modules in the Python Package

A module is a .py extension file that contains Python code and contains several functions and classes. Modules can be put together in one package so that the collection of modules can be structured and can be distributed easily. Module calling can be done using "dot / dot", for example if you want to use the core module in numpy, it can be done with numpy.core

The contents of the module can be accessed by importing. There are three ways to import a module:

- `import <nama_module>`

  Ex:

```
import numpy.core
subtract(3,2)
```

- `from <nama_ module> import <nama_obyek>`

  contoh: `from numpy.core import substract`

```
subtract(3,2)
```

- `from <nama_ module> import <nama_obyek> as <inisial>`

  contoh:

```
from numpy.core import subtract as sub
sub(3, 2)
```

1.1.1.3 Kanren Package (LogPy)

Basically, the standard Python package does not support logic programming like other AI languages like Prolog. Therefore, a package or logic programming module is needed to apply the concept of tracking in Python. One of the packages used is Kanren (formerly called LogPy). By using Kanren, we can express relationships, facts and query to find the desired value. Kanren stores data in the form of facts that explain the relationship between terms.

Kanren installation can be done using pip:

```
pip install kanren
```

Once installed, the Kanren package can be used in the program by importing the program.

1.1.1.4 Logic Module in Kanren Package

Logic Variable

Logical variables are variables that can contain any value, but only one value at a time. How to use logical variables using Kanren is to declare variables using var. For example x = var () where x is a logical variable.

Constraint dan Goal

Constraint is an expression that limits the value of a logical variable.

Ex 1:

After declaring x = var (), the variable x can hold any value. If you want to limit or give constraints that the value of variable x is equal to 5, then the constraint variable x can be written as eq (x, 5).

Goal states the final results to be achieved.

Ex 2:

For example the variable y is declared as a variable using the same method as example 1, y = var (). The goal to be achieved is equation (x, y) = (y, 3). In this equation the goal can be written using the eq syntax, which states that both expressions are equal. The goal can be written as follows eq ((x, y), (y, 3)) .

Logic Expression

Logical expression consists of a collection of logic variables and a collection of constraints to the value of the logical variable. If x is a variable, then each logical expression contains a constraint that limits the value of the x variable.

Based on example 1, logical expressions can be poured into the program as follows:

```
1  from kanren.core import var, eq, run
2  x = var()
3  output = run(1, x, eq(5, x))
4  print(output)
```

Listing 0.1 Program Code for example 1

First, import the syntax from the Kanren package that will be used in the program, namely var, eq, and run as in the first line. Furthermore, the declaration of the logic x variable as written in line 2. Then the logic expression that is run using the run syntax on line 3, shows that the program asks for 1 value, namely variable x, with the constraint x = 5. The results of the program are displayed on line 4 so The output of the program is as follows: `(5,)` Example 2 can be written into the program as follows:

```
1  from kanren.core import var, eq, run
2  x = var()
3  y = var()
4  output = run(1, x, eq((x, y), (y, 3)))
5  print(output)
```

Listing 0.2 Program code for example 2

The variable declaration and import package are almost the same as the program in example 1. The difference is that now there are two declared variables, x and y. Furthermore, the syntax in line 4 shows that the program asks for 1 value, namely variable x, with goal eq ((x, y), (y, 3)). In other words, look for the value of variable x that satisfies the constraint y == 3 and meets the requirements (x, y) = (y, 3) so that when run the program will display the following output: `(3,)`

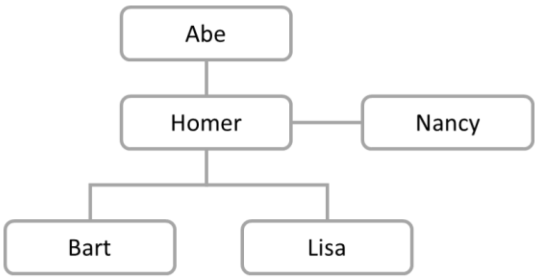
Facts and Relationships

The declaration of facts and relations using Kanren is similar to the declaration using the language Prolog. Examples of the use of facts and relations using Kanren can be seen in table 1.

Facts in sentences	Relation	Fact
Abe is the father of Homer "	`father`	`fact(father("Abe","Homer"))`
California is a coastal	`coastal`	`fact(coastal("California"))`
California is adjacent to Arizona	`adjacent`	`fact(adjacent("California, "Arizona"))`

Tabel 0.1 Facts and Relationships

Examples of family genealogical cases like in Figure 1.1 can be used to clarify the use of facts and relations in Python programs using Kanren. Facts and relationships can be made using the references as in Table 1.1.



Gambar 0.1 Examples of Family Genealogy

Relations and facts are imported from the Kanren package on line 1 along with var and run which are used to hold relations, facts, logical variables, and execute logical expressions sequentially. The relation used in this example is father. A list of facts in the example of family tree using the father relation can be seen in Tabel 0.2 below:

Facts in sentences	Fact
Homer is the father of Bart	`fact(father( "Homer", "Bart"))`
Homer is the father of Lisa	`fact(father( "Homer", "Lisa"))`
Abe is the father of Homer	`fact(father( "Abe", "Homer"))`

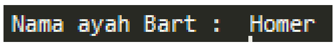
Tabel 0.2 Family Genealogy Facts

The declaration of the father's relation is written on line 3. Because of the facts on Tabel 0.2 using the same relation that is father, then the collection of facts can be written into one using facts as written on line 4.

```
1  from kanren.facts import Relation, facts
2  from kanren.core import var, run

3  father = Relation()
4  facts(father, ("Homer", "Bart"),
                 ("Homer", "Lisa"),
                 ("Abe",   "Homer"))
5  x = var()
6  output = run(1, x, father(x, "Bart"))
7  print("\nNama ayah Bart : ", output[0])
```

Listing 0.3 Program Code for Family Genealogy in Figure 1.1

Lines 5 are logic variables and line 6 shows the execution of logical expressions to find father from Bart. If run, the program will produce the following output:



```
Nama ayah Bart :  Homer
```

New relations can be declared in two ways, namely:

By making a declaration of a new relation using Relationships

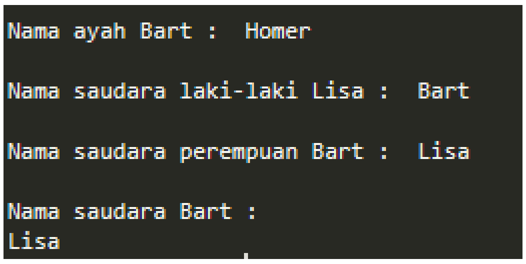
Use existing relationships to form other relations without creating a new relation

Implementation of the first method can be seen in lines 13-19 in the program below. Sibling relations are declared using Relationships and then need to be defined new facts based on these relations. This method is ineffective because there is data redundancy and if you have a large number of brothers, you must add facts one by one according to the number of facts. For cases like this we can use existing relations to detect new relations. In this case the Parent relation can be used to detect relatives without making a new relationship. The implementation of using Parent relations to detect your name can be seen in the get_sibling function in lines 2-4. The function call and output display can be seen in lines 20-24.

```
1   from kanren.facts import Relation, facts
    from kanren.core import var, run, conde

2   def get_sibling(x, y):
3       temp = var()
4       return conde((parent(temp, x), parent(temp, y)))

5   if __name__=='__main__':
6       parent = Relation()
7       facts(parent, ("Homer", "Bart"),
8                     ("Homer", "Lisa"),
9                     ("Abe",   "Homer"))
10      x = var()
11      output = run(1, x, parent(x, "Bart"))
12      print("\nNama ayah Bart : ", output[0])

        # contoh definisi saudara (sibling) menggunakan relasi baru
13      sibling = Relation()
14      facts(sibling, ("Bart", "Lisa"),
15                     ("Lisa", "Bart"))
16      brother = run(0, x, sibling(x, "Lisa"))
17      print("\nNama saudara laki-laki Lisa : ", brother[0])
18      sister = run(0, x, sibling(x, "Bart"))
19      print("\nNama saudara perempuan Bart : ", sister[0])

        '''
            contoh definisi saudara (sibling)
            menggunakan relasi yang sudah ada (parent)
        '''
20      siblings = run(0, x, get_sibling(x, "Bart"))
21      siblings = [x for x in siblings if x != "Bart"]
22      print("\nNama saudara Bart : ")
23      for item in siblings:
24          print(item)
```

Listing 0.4 Program Code for Family Genealogy Using New Relationships

If the program is run, it will output something like this:



```
Nama ayah Bart :  Homer

Nama saudara laki-laki Lisa :  Bart

Nama saudara perempuan Bart :  Lisa

Nama saudara Bart :
Lisa
```

1.4 PRACTICUM ASSIGNMENTS

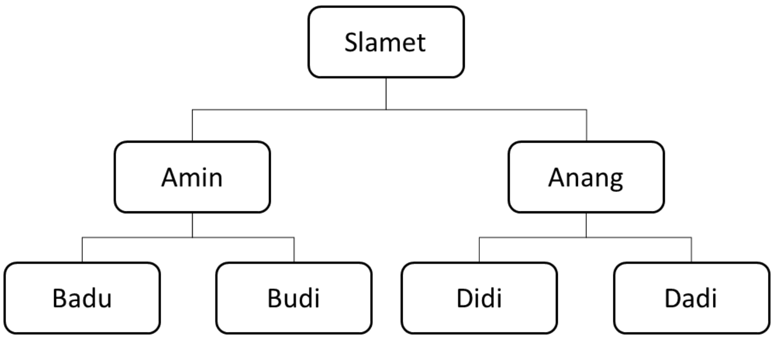


Figure 1.2 Example of Family Genealogy

Example Program: Family Genealogy Case in Figure 1.2

```
from kanren.facts import Relation, facts
from kanren.core import var, run

parent = Relation()
facts(parent, ("Slamet", "Amin"),
              ("Slamet", "Anang"),
              ("Amin", "Badu"),
              ("Amin", "Budi"),
              ("Anang", "Didi"),
              ("Anang", "Dadi"))
x = var()
child = "Amin"
ayah = run(1, x, parent(x, child))
print("\nNama ayah " + child + ": ")
for item in ayah:
    print(item)
```

Listing 0.5 Example of a Family Genealogy Program Using the Kanren Package

1. **(Value 40)** Try typing the above program in Python. How:
   - Open the editor window as desired
   - To write the writing program syntax in the editor window
   - Type all the programs above (listing 1.5)
   - Run the program using the console
   - Watch what happens

Task: Try multiple queries (minimum 3 queries). Record the results

(Value 60) Based on listing 1.5, try to develop the program, if you add the facts of grandfather, child or uncle in two ways:
   - Making new relationships, grandfathers, children, and uncle
   - Without creating new relations (only use parent relations to form other relations. Hint: Look at the example in Listing 1.4 again)

PRACTICUM 2 TRACKING, ROOMS AND PROBLEMS

	Meeting	: 2
	Total Time Allocation	: 90 minutes
•	Pre-Test	: 15 minutes
•	Practicum	: 60 minutes
•	Post-Test	: 15 minutes

	Total Rating Score	: 100%
•	Pre-Test	: 25%
•	Practicum	: 35%
•	Post-Test	: 40%

2.1 OBJECTIVES AND INDICATORS OF ACHIEVEMENTS

After following this practicum, students are expected to:

1. Can do tracking in programming.

The achievement indicator is measured by: (adjusting to the RPS)

1. Able to apply algorithmic strategy theory, data structure, statistical inference to solve problems in the tracking process into the programming language

2.2 SUPPORTING THEORY

Problem Representation

As is well known on systems that use artificial intelligence will try to provide output in the form of a solution to a problem based on a collection of existing knowledge. This is represented in **Error! Reference source not found.**.

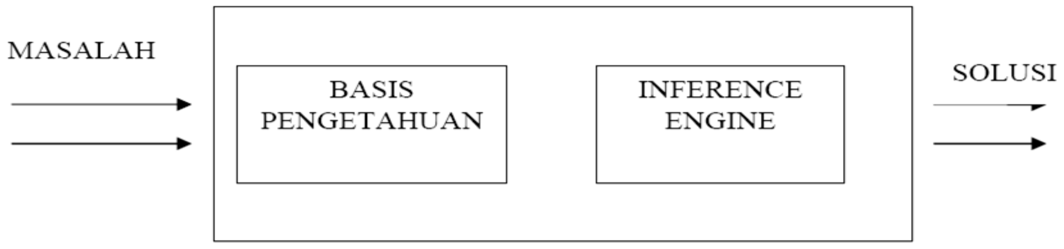


Figure 2.1 Artificial Intelligence System

Based on **Error! Reference source not found.**, input provided on systems that use artificial intelligence in the form of problems. In the system must be equipped with a set of knowledge that is on a knowledge base (knowlage base). The system must have an inference engine so that the system is able to draw conclusions based on facts or knowledge. The output given is a solution to the problem as a result of interference.

In general, to build a money system capable of solving problems, it is necessary to consider 4 things:

1. Define the problem correctly. This definition includes the specification of a cash advance regarding the initial state (initial state) and the expected solution.
2. Analyze the problem and look for several points to solve the problem of money accordingly.
3. Represent the knowledge needed to solve the problem.
4. Choose the best money solving techniques.

Troubleshooting in AI

In solving problems with AI techniques involving several steps, namely:

- Problem analysis
- Representation of Problems and Knowledge
- Inference
- Use of AI Language

1.1.1.5 Problem analysis

Problem analysis is the first step in AI in AI. This step analyzes the problem of money being faced and expresses the problem in a symbol system. The system can be diagrams, schemes, graphs, or other symbols. This symbol system must be translated into the AI programming language. This system must be able to express precisely the initial state (initial state). The final state or the intended target (Goal State). For example, traders visited 10 cities. The initial state is the existing travel route and can be described as follows:

$R(K_1, K_2, ..., K_N)$

$R(K_{J1}, K_{J2}, ..., K_{JN})$

With city distance K_1 to city K_{J1} is d_{ij}.

Where K_{J1}, K_{J2} are cities in the list of traders.

The target situation is one of the travel routes that has a minimum number of d ij.

In general, defining the problem as a state space includes 3 things, i.e:

- Initial State
- Rule
- Goal

Example

For example the problem faced is "Chess Game", then it must be determined:

1. The initial position on the chessboard

   The starting position of each chess game is always the same, i.e. all the pieces are placed on the chessboard on two sides, namely the white and black camps.

2. Rules for making illegal movements (Rule)

   The rules (rules) are useful for determining the movement of a piece, which is moving from one state to another. For example, to make it easier to show the position of the pieces, each box is shown in letters (a, b, c, d, e, f, g, h) in the horizontal direction and numbers (1, 2, 3, 4, 5, 6, 7, 8) in the vertical direction. A rule for moving a piece from position (e, 2) to (e, 4) can be indicated by the rule:

```
IF white piece on box (e, 2),
        AND Box (e, 3) Empty,
        AND Box (e, 4) Empty,
THEN Move pieces from (e, 2) to (e, 4)
```

1. Goal

   The goal to be achieved is the position on the chessboard that shows a person's victory against his opponent. This victory is marked by the position of KING, the money has moved again.

2.3 PRACTICUM ASSIGNMENTS

1. **(Value 50)** Try typing the following Python program:

Program 1:

```
# Program aktivitas 1
from kanren.facts import Relation, facts, fact
from kanren.core import var, run
from kanren.goals import membero

suka = Relation()

facts(suka, ("ellen", "tenis"),
            ("john", "football"),
            ("john", "tenis"),
            ("mary", "renang"),
            ("tom", "tenis"),
            ("tom", "basket"),
            ("eric", "renang"),
            ("mary", "tenis"))

x = var()
tom_hobbies =  run(0, x, suka("tom", x))
print("Tom: ", tom_hobbies)

for hobby in tom_hobbies:
    fact(suka, ("bill"), hobby)
bill_hobbies =  run(0, x, suka("bill", x))
print("Bill: ", bill_hobbies)

mary_hobbies =  run(0, x, suka("mary", x))
print("Mary: ", mary_hobbies)

for hobby in mary_hobbies:
    fact(suka, ("ann"), hobby)
ann_hobbies =  run(0, x, suka("ann", x))
print("Ann: ", ann_hobbies)
```

Listing 0.1 Hobby Program

Program 2 :

```
# Program aktivitas 2

from kanren.facts import Relation, facts, fact
from kanren.core import var, run
from kanren.goals import membero

suka = Relation()

facts(suka, ("ellen", "tenis"),
            ("john", "football"),
            ("mary", "renang"),
            ("tom", "tenis"),
            ("eric", "renang"))

x = var()
tom_hobbies =  run(0, x, suka("tom", x))
print("Tom: ", tom_hobbies)

for hobby in tom_hobbies:
    fact(suka, ("bill"), hobby)
bill_hobbies =  run(0, x, suka("bill", x))
print("Bill: ", bill_hobbies)

mary_hobbies =  run(0, x, suka("mary", x))
```

```
print("Mary: ", mary_hobbies)

for hobby in mary_hobbies:
    fact(suka, ("ann"), hobby)
ann_hobbies =  run(0, x, suka("ann", x))
print("Ann: ", ann_hobbies)
```

Listing 0.2 Hobby Program 2

a. For the programs in Listing 2.1 and 2.2 what are the results?
b. Try creating both programs by providing the goals below for both programs. How is the result? (hint: use membero)
   `Query 1 : suka("ellen",X),suka("tom",X)`
   `Query 2 : suka("mary",X),suka("ann",X)`
c. Try to give new facts for development.

Try the program in Listing 2.3 below and watch the results

```
    from kanren.facts import Relation, facts, fact
    from kanren.core import var, run
    from kanren.goals import membero
    from kanren import vars

    ukuran = Relation()
    warna = Relation()
    gelap = Relation()

    facts(ukuran, ("beruang", "besar"),
                  ("gajah", "besar"),
                  ("kucing", "kecil"))

    facts(warna,  ("beruang", "cokelat"),
                  ("kucing", "hitam"),
                  ("gajah", "kelabu"))

    fact(gelap, "hitam")
    fact(gelap, "cokelat")

    x = var()
    kecil =  run(0, z, ukuran(z, "kecil"))
    print("hewan berukuran kecil: ", kecil)
```

Listing 0.3 Contoh program Python

2. **(Value 15)** Run the program by adding the following goals:
   `ukuran(z, "besar")`. Write into the program and display the program output
   `warna(z, "cokelat")`. Write into the program and display the program output
   Add code to the program to display the sized animals besar **and** colored cokelat
3. **(Value 15)** Add code to the program to display animals that:
   colored gelap
   sized besar **and** colored gelap
4. **(Value 20)** After trying the query, try adding new facts and relations to the program. Add relation types and facts `jenis("beruang", "karnivora")` and `jenis("kucing", "karnivora")`. Add codes to the program to display carnivorous animals

PRACTICUM 3: HEURISTIC SEARCH

	Meeting	: 3
	Total Time Allocation	: 90 minutes
•	Pre-Test	: 15 minutes
•	Practicum	: 60 minutes
•	Post-Test	: 15 minutes

	Total Rating Score	: 100%
•	Pre-Test	: 25%
•	Practicum	: 35%
•	Post-Test	: 40%

3.1 OBJECTIVES AND INDICATORS OF ACHIEVEMENTS

After following this practicum, students are expected to:

1. Able to do heuristic search percentages with a programming language.

Achievement indicators are measured by:

1. Able to apply algorithmic strategy theory, data structure, statistical inference to solve problems in the tracking process into the programming language

3.2 SUPPORTING THEORY

Custom Python Package for Searching

For this practicum material and so on will be used a custom package that defines the data structure and basic functions contained in the practicum module material. The custom package is packaged in the form of Python wheels, which is the standard Python package distribution.

Installation

To do the installation, simply download the package wheels with the .whl extension and execute the command like the package installation from the Python server

```
pip install <nama_file_whl>
```

Module Search Structure in Package

The custom package that will be used for the lab is called ai_pkg. Within this module there are several modules, one of which is the search module. In this module there are three classes that define the data structures used for the search method.

```
ai_pkg/
    ...
    search.py
        Graph <class>
        Node <class>
        Problem <class>
    utils.py
```

...

Hill Climbing

This method is a type of heuristic tracking, because in searching for a solution always consider the node that has the best value or with a gradual model.

- Is a tracking method that combines Generate and Test tracking with Backtracking.
- For the initial step of tracking by selecting the node that has the best / largest value (heuristic function)

This method is almost the same as the Generate and Test method, it's just that the testing process is done by using the heuristic function. The test value is a certain range. Pay attention to **Error! Reference source not found.**.

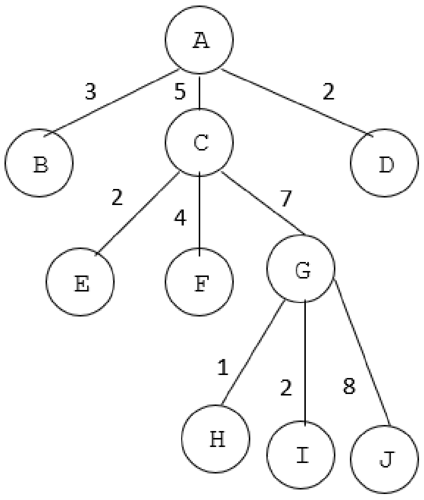


Figure 3.1 Searching for Hill Climbing

The search starts from A as the initial state. At the first level, C has a better value (in this case the highest heuristic function in one level). Thus C is chosen as the next node. At the second level, G has the best value, so G is chosen as the next node. And so on.

Algorithm:

1. Starting from the initial state, do the test: if it is a solution, then stop; and if not, continue with the active region as the initial state.
2. Work through the following steps until a solution is found, or until no new operators will be applied to the active region:
   a) If it is a goal, then the process is successful and exits
   b) If it is not a destination, but it is better than the current state, then set the point as an active state.
   c) If it is worse than the current situation, then repeat step - 2.

3.3 PRACTICUM STEPS

Following is an example of tracking application using the Hill Climbing method to answer the Traveling Salesman Problem (TSP) problem. The problem is if given several cities and the distance between each city is known, then the solution is the shortest route that can reach all cities and return to the city of origin.

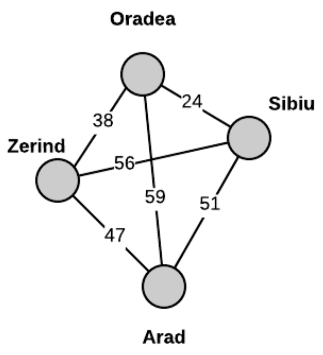


Figure 3.2 City Map

Example of a TSP problem:

There are 4 cities namely Zerind, Oradea, Sibiu, and Arad and the distance between cities can be seen in Figure 3.1. Assuming the distance between cities is symmetrical so the distance from Zerind to Oradea and vice versa is the same that is 38.

The TSP case can be modeled as a weighted and undirected weighted graph. Each city is represented as a "node" (vertex or node) and each path between cities is represented as a "edge".

```
city_map = Graph(dict(
    Oradea=dict(Oradea=0, Sibiu=24, Arad=59, Zerind=38),
    Sibiu=dict(Oradea=24, Sibiu=0,   Arad=51, Zerind=56),
    Arad=dict(Oradea=59, Sibiu=51, Arad=0,   Zerind=47),
    Zerind=dict(Oradea=38, Sibiu=56, Arad=47,  Zerind=0)),
    directed=False)
```

Listing 3.1 Code for City Graph Representation in Figure 3.1

First, Graph is formed with parameters in the form of dict and directed. The city name and the distance of each city can be defined directly using dict. In the case of TSP used undirected graphs the directed parameter is set to False. By creating a new Graph object, the graph node will be formed automatically using the existing Node class in the search module. The city graph representation in Figure 3.1 can be written as in Listing 3.1.

```
distances = {}
class TSP_problem(Problem):
    def generate_neighbour(self, state):
        neighbour_state = state[:]
        left = random.randint(0, len(neighbour_state) - 1)
        right = random.randint(0, len(neighbour_state) - 1)
        if left > right:
            left, right = right, left
        neighbour_state[left: right + 1] = reversed(neighbour_state[left: right +
1])
        return neighbour_state

    def actions(self, state):
        return [self.generate_neighbour]

    def result(self, state, action):
        return action(state)

    def path_cost(self, state):
        cost = 0
        for i in range(len(state) - 1):
            current_city = state[i]
            next_city = state[i + 1]
            cost += distances[current_city][next_city]
        cost += distances[state[0]][state[-1]]
        return cost
    def value(self, state):
        return -1 * self.path_cost(state)
```

Listing 3.2 Code for the Definition of the TSP Problem

Next we create a TSP problem by creating a class named TSP_problem which implements the abstract class Problem in the package used. There are four basic functions, namely: actions, result, path_cost, and value. These four basic functions must be implemented because they are used in TSP execution in the Node class.

The code for the Hill Climbing method can be seen in Listing 3.3.

```
def hill_climbing(problem):
    def find_neighbors(state, number_of_neighbors=100):
        neighbors = []
        for i in range(number_of_neighbors):
            new_state = problem.generate_neighbour(state)
            neighbors.append(Node(new_state))
            state = new_state
        return neighbors

    current = Node(problem.initial)
    while True:
        neighbors = find_neighbors(current.state)
        if not neighbors:
            break
        neighbor      =      argmax_random_tie(neighbors,      key=lambda      node:
problem.value(node.state))
        if problem.value(neighbor.state) <= problem.value(current.state):
            break
        current.state = neighbor.state
    return current.state
```

Listing 3.3 Code for the Hill Climbing Method

3.4 PRACTICUM ASSIGNMENTS

1. **(Value 40)** Run the Traveling Salesman Problem program in a way:
   a. Download the custom package via the URL provided by the assistant and install the package and import the modules that are in the package to use it in the program
   b. Write down the program in the example case in Figure 3.1 and add the following listing to run the program

```
if __name__=='__main__':
    all_cities = []
    cities_graph = city_map.graph_dict
    for city_1 in cities_graph.keys():
        distances[city_1] = {}
        if(city_1 not in all_cities):
            all_cities.append(city_1)
        for city_2 in cities_graph.keys():
            if(cities_graph.get(city_1).get(city_2) is not None):
                distances[city_1][city_2] = cities_graph.get(city_1).get(city_2)
```

Listing 3.4 Code for Defining City Distance

   c. The TSP program can be run by creating a new TSP_problem object with the all_cities parameter. Next, call the hill_climbing function with the parameters of the object you just created.
   d. Run the program and display the results

(Value 60) Create a TSP program using city data as shown below and record the results:

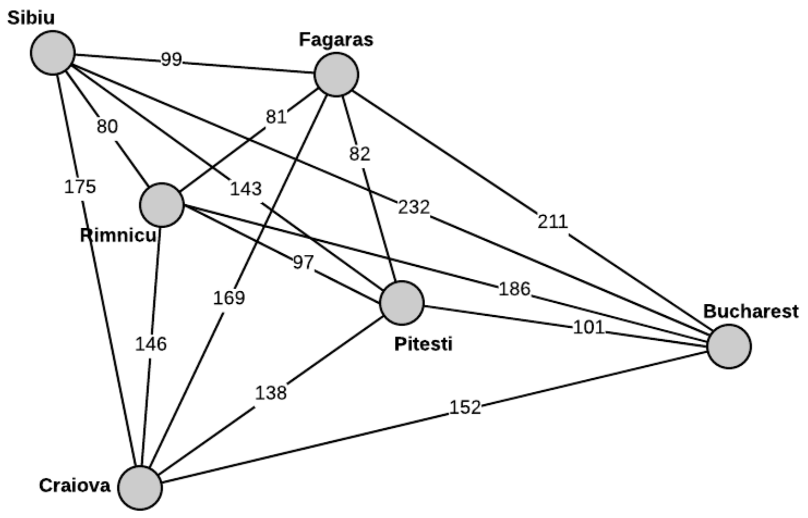


Figure 3.3 City Map

PRACTICUM 4 REPRESENTATION OF KNOWLEDGE

Meeting	: 4
Total Time Allocation	: 90 minutes
• Pre-Test	: 15 minutes
• Practicum	: 60 minutes
• Post-Test	: 15 minutes

Total Rating Score	: 100%
• Pre-Test	: 25%
• Practicum	: 35%
• Post-Test	: 40%

4.1 OBJECTIVES AND INDICATORS OF ACHIEVEMENTS

After following this practicum, students are expected to:

1. Able to represent knowledge to present knowledge.

Achievement indicators are measured by:

1. Can explain the representation model to present knowledge.

4.2 SUPPORTING THEORY

Knowledge:

- General definition: fact or condition of something or condition that arises from an experience.
- The branch of philosophy, namely Epistemology, with regard to the nature, structure and authenticity of knowledge.

Knowledge Representation Techniques:

1) Production Rules
2) Semantic Networks
3) Graph
4) Frame and Scemata
5) Proposition Logic
6) List
7) Decision table

Language of representation must be able to make a programmer able to express knowledge to get the solution of a problem.

In brief, Mylopoulos and Levesque classify the composition or pattern of representation into four categories:

1) Logic Representation - This representation uses expressions in formal logic to represent the knowledge base.
2) Procedural Representation - Describes knowledge as a set of instructions for solving a problem. In a rule-based system, if-then rules can be interpreted as a procedure to achieve the goal of problem solving.
3) Network Representation - Capturing knowledge as a graph in which vertices represent objects or concepts in the problem at hand, while the arches describe relationships or associations between them. An example is semantic networks.

4) Structured Representation - Expanding the network by making each node into a complex data structure that contains places named slots with certain values. These values can be simple numeric or symbolic data, pointers to other frames, or even procedures to perform certain tasks.

Example: scripts (frames), frames (frames) and objects (objects).

In mathematics, not all sentences relate to logic. Only sentences that are true or false are used in reasoning. The sentence is called a proposition (preposition).

Propositions are declarative sentences that are true or false, but they cannot be both at the same time. The truth or error of a sentence is called the truth value.

The following three examples can illustrate which sentences are propositions and which are not. The following statements,

(a) 6 is an even number.

(b) Sukarno was the first President of Indonesia.

(c) 2 + 2 = 4.

(d) The capital of West Java Province is Semarang.

(e) 12 ³ 19.

(f) Yesterday was a rainy day.

(g) The temperature at sea level is 21 degrees Celsius.

(h) Youth is tall.

(i) Life only exists on planet Earth.

everything is a proposition. Propositions a, b, and c are true, but proposition d is false because the capital of West Java should be Bandung and proposition e is false because it should be 12 pounds. - these propositions cannot be right or wrong at once. We can determine the value of the proposition is true or false. For example, we can assume that the proposition f is true (yesterday was indeed raining) or false (yesterday was not raining). The same goes for propositions g and h. Proposition i can be right or wrong, because until now there has been no scientist who can confirm the truth.

Symbolically, propositions are usually denoted by lowercase letters such as p, q, r, ... For example, p: 6 is an even number.

To define p as the proposition "6 is an even number". Likewise for q: Soekarno was the first President of Indonesia.

r : 2 + 2 = 4.

etc. We can form new propositions by combining one or more propositions. The operators used to combine propositions are called logical operators. The basic logical operators used are and (and), or (or), and not (not). The first two operators are called binary operators because they operate two propositions, while the third operator is called uner operators because it only requires one proposition. The new proposition obtained from the combination is called a compound proposition (compound proposition). Propositions that are not a combination of other propositions are called atomic propositions. In other words, compound propositions are composed of atomic propositions. The method of combining propositions was discussed by an English mathematician named George Boole in 1854 in his famous book, The Laws of Thought. There are three kinds of compound propositions, namely conjunctions, disjunctions, and denials. The three are defined as follows:

Suppose p and q are propositions. The conjunction (conjunction) p and q, expressed by the $p \wedge q$ notation, are propositions p and q. Disjunction (disjunction) p and q, expressed by the notation $p \vee q$, is the proposition p or q. The denial or (negation) of p, expressed by ~ p notation, is a proposition not p.

Note:

1. Note1. Some literature uses the notation "Øp", "p", or "not p" to express a denial.
2. The word "no" can be written in the middle of the statement. If the word "no" is given at the beginning of the statement then it is usually connected with the word "true" to "incorrect".

The word "no" can also be replaced by "no" depending on the sense of language that is appropriate for the statement

Example :
The following propositions are known:
p : Today is raining
q : Students are dismissed from school
then
$p \wedge q$: Today it is raining and students are suspended from school
$p \vee q$: Today it is raining or students are dismissed from school
$\sim p$: It is not true today it rains (or in another more usual sentence: Today it doesn't rain)

4.3 PRACTICUM ASSIGNMENTS

Type the coding program logic proposition in listing 4.1.

```
from ai_pkg.utils import Expr

def is_prop_symbol(s):
    return isinstance(s, str) and s[:1].isalpha() and s[0].isupper()

def is_true(exp, model={}):
    if exp in (True, False):
        return exp
    op, args = exp.op, exp.args
    if is_prop_symbol(op):
        return model.get(exp)
    elif op == '~':
        p = is_true(args[0], model)
        if p is None:
            return None
        else:
            return not p
    elif op == '|':
        result = False
        for arg in args:
            p = is_true(arg, model)
            if p is True:
                return True
            if p is None:
                result = None
        return result
    elif op == '&':
        result = True
        for arg in args:
            p = is_true(arg, model)
            if p is False:
                return False
            if p is None:
                result = None
        return result
    p, q = args
    if op == '==>':
        return is_true(~p | q, model)
    elif op == '<==':
        return is_true(p | ~q, model)
    pt = is_true(p, model)
    if pt is None:
        return None
    qt = is_true(q, model)
    if qt is None:
        return None
    if op == '<=>':
        return pt == qt
    elif op == '^':
        return pt != qt
    else:
        raise ValueError("illegal operator" + str(exp))

if __name__=='__main__':
    A, B = map(Expr, 'AB')
    model = {A: False, B: True}
    query = (A & B)
    print(query , ' : ' , is_true(query, model))
```

Listing 4.1 Proposition Logic Code

Operator	Simbol
not	~
and	&
or	|
implication	==>
if and only if	<=>

Figure 4.1 List of operators and symbols in the logic program proposition

1. **(Value 40)** Add some of the queries below to the program in Listing 4.1 to prove the logical expression of the proposition. Record the results.
    - Query 1: not (A and B)
    - Query 2: not (A and B) or not (A or B)
    - Query 3: not (A or B) and not (A and B)
2. **(Value 60)** By using the program in listing 4.1:
    a. Add Expr C and give False then give the following query to the program:
        Query 1 : (A or B) and (C or D)
        Query 2 : (A and B) and (C or D)
    b. Add Expr D and give the value True then run the query again in point 2.a and compare the results

PRACTICUM 5 REPRESENTATION OF KNOWLEDGE WITH THE TREE MODEL

Meeting	: 5
Total Time Allocation	: 90 minutes
• Pre-Test	: 15 minutes
• Practicum	: 60 minutes
• Post-Test	: 15 minutes

Total Rating Score	: 100%
• Pre-Test	: 25%
• Practicum	: 35%
• Post-Test	: 40%

5.1 OBJECTIVES AND INDICATORS OF ACHIEVEMENTS

After following this practicum, students are expected to:

1. Can represent knowledge with a tree model.

Achievement indicators are measured by:

1. Can explain the Tree model as a form of knowledge representation.

5.2 SUPPORTING THEORY

In this practicum will use the example of a tree model to present knowledge

In this practicum, only a few will be discussed as a sample application of tracking in artificial intelligence.

Breadth First Search (BFS)

BFS is a search that starts from the root node and continues to the next level from left to right, then moves to the next level. The case example in Figure 5.1 shows a city with a directed graph type.

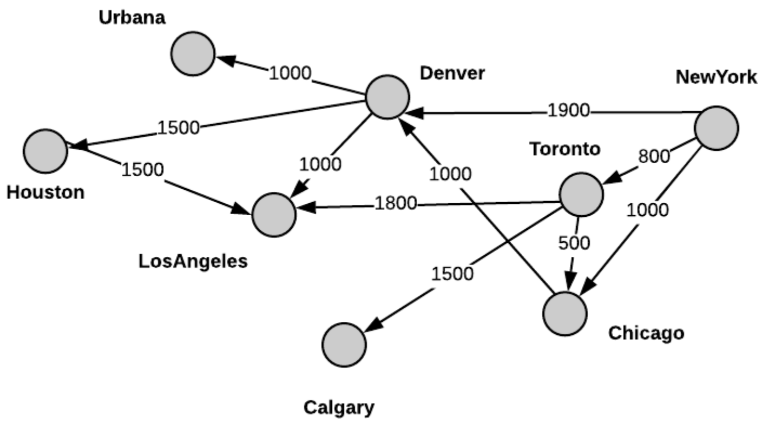


Figure 5.1 City Map

If the home city is New York and the destination city is Los Angeles, the tracking flow using the BFS method is shown in Figure 5.2:

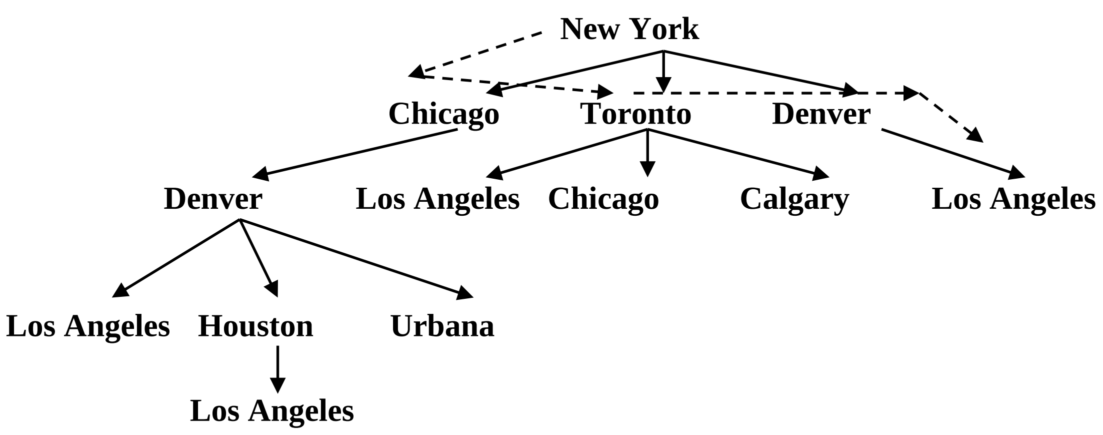
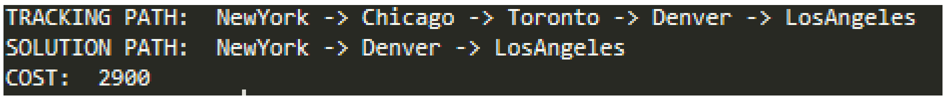


Figure 5.2 Tracking Trees Using BFS

The solution of the above case can be seen in Figure 5.3 below



```
TRACKING PATH:  NewYork -> Chicago -> Toronto -> Denver -> LosAngeles
SOLUTION PATH:  NewYork -> Denver -> LosAngeles
COST:  2900
```

Figure 5.3 BFS Problem Solution

5.3 PRACTICUM ASSIGNMENTS

Write the code below

```
city_map = Graph(dict(
        NewYork=dict(Chicago=1000, Toronto=800, Denver=1900),
        Chicago=dict(Denver=1000),
        Denver=dict(LosAngeles=1000, Houston=1500, Urbana=1000),
        Houston=dict(LosAngeles=1500),
        Toronto=dict(LosAngeles=1800, Chicago=500, Calgary=1500)), directed=True)

class CityProblem(Problem):
    def __init__(self, initial, goal, graph):
        Problem.__init__(self, initial, goal)
        self.graph = graph

    def actions(self, A):
        return list(self.graph.get(A).keys())

    def result(self, state, action):
        return action

    def path_cost(self, cost, A, action, B):
        return cost + (self.graph.get(A, B) or infinity)

def breadth_first_search(problem):
    global track_path
    frontier = deque([Node(problem.initial)])
    explored = set()
    track_path = [problem.initial]
    while frontier:
        node = frontier.popleft()
        if problem.goal_test(node.state):
            return node
        explored.add(node.state)
        expanded = node.expand(problem)
        for child in expanded:
            track_path.append(child.state)
            if child.state not in explored and child not in frontier:
                if problem.goal_test(child.state):
                    return child
                frontier.append(child)
    return None

if __name__=='__main__':
    track_path = []
    romania_problem = CityProblem(start, goal , city_map)
    node = breadth_first_search(romania_problem)
    if node is not None:
        final_path = node.solution()
        final_path.insert(0, start)
        print('TRACKING PATH: ', ' -> '.join(track_path))
        print('SOLUTION PATH: ', ' -> '.join(final_path))
```

Listing 5.1 Tracking Program Code Using BFS

1. **(Value 30)** Run the BFS program in a way:
   a. Download the custom package via the URL provided by the assistant and install the package and import the modules that are in the package to use it in the program
   b. Write the code in listing 5.1 and add the hometown of "New York" and the destination city of "Los Angeles". Run the program and show the results
   c. Try using another destination and show the results
2. **(Value 70)** After you run and try to interact with the system, try to make a program developed from Listing 5.1 above with reference to the following tracking tree:

Origin : Kota_A
Dest : Kota_T

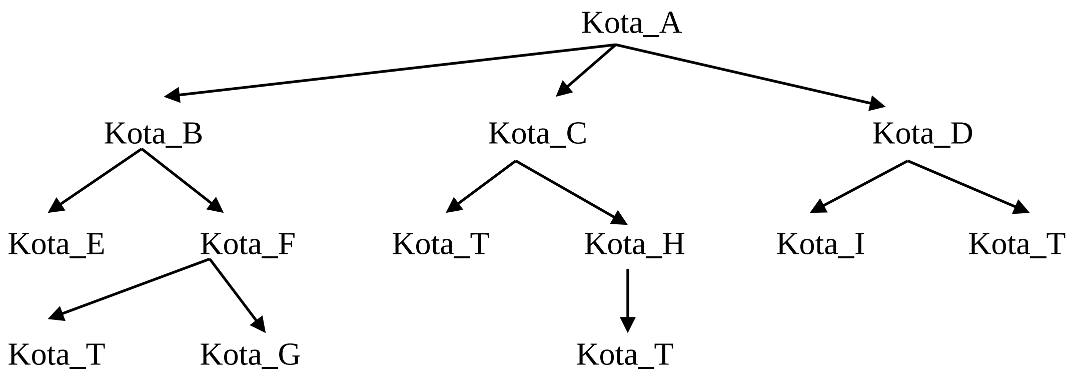


Figure 5.4 Tracking tree for the 2nd case of BFS

PRACTICUM 6 HIERARCY PLANNING

	Meeting	: 6
	Total Time Allocation	: 90 minutes
•	Pre-Test	: 15 minutes
•	Practicum	: 60 minutes
•	Post-Test	: 15 minutes

	Total Rating Score	: 100%
•	Pre-Test	: 25%
•	Practicum	: 35%
•	Post-Test	: 40%

6.1 OBJECTIVES AND INDICATORS OF ACHIEVEMENTS

After following this practicum, students are expected to:

1. Able to do hierarchy planning to build efficient plans.

Achievement indicators are measured by:

1. Able to explain hierarchy planning using HLA.

6.2 SUPPORTING THEORY

Hierarchical planning is used to build efficient plans because it can divide problems into abstract levels before proceeding to more detailed steps. By using a hierarchical structure, each level in the hierarchy can be divided into smaller steps.

High-Level Action (HLA)

Each HLA can have one or more refinements in the form of a more detailed set of actions. For example: the action "Go to the airport" can be represented in HLA Go (Home, Airport). To be able to go to the airport can be done in two ways, namely by driving your own car or taking a taxi. If the option of driving a car is selected, the action can be further divided into smaller steps, namely driving from the house to the airport car park, then taking the shuttle bus from the airport car park to the terminal at the airport. So, HLA Go (Home, Airport) has two refinements, i.e.:

- `Drive(Home, AirportParking)` and `Shuttle(AirportParking, Airport)`
- `Taxi(Home, Airport)`

Planning Problem

To solve Hierarchical Planning problems, you can use the planning module in the custom package. Input from planning algorithm are Problem and hierarchy. hierarchy is a collection of all actions (actions) and the sequence of these actions.

As explained in the example case of going to the airport, to go to the airport can be done in two ways, namely driving yourself to the airport or using a taxi. Both of these actions have several initial conditions (preconditions) and effects (effects). When we use a taxi, the initial conditions we have to have cash, while if we want to drive our own meals we must have a car. hierarchy of actions that can be done can be written as follows:

```
    library = {
            'HLA': ['Go(Home,Airport)',
                    'Go(Home,Airport)',
                    'Drive(Home, AirportParking)',
                    'Shuttle(AirportParking, Airport)',
                    'Taxi(Home, Airport)'],
            'steps':[['Drive(Home,AirportParking)','Shuttle(AirportParking,
Airport)'],
                    ['Taxi(Home, Airport)'],
                    [],
                    [],
                    []],
            'precond': [['At(Home) & Have(Car)'],
                    ['At(Home)'],
                    ['At(Home) & Have(Car)'],
                    ['At(AirportParking)'],
                    ['At(Home)']],
            'effect': [['At(Airport) & ~At(Home)'],
                    ['At(Airport) & ~At(Home) & ~Have(Cash)'],
                    ['At(AirportParking) & ~At(Home)'],
                    ['At(Airport) & ~At(LongTermParking)'],
                    ['At(Airport) & ~At(Home) & ~Have(Cash)']] }
```

Listing 0.1 Kode Hirarki untuk Kasus Pergi ke Airport

Action with the highest level that can be done is to go to the airport (`goto_airport`). Action `goto_airport` can be written as in Listing 6.2.

```
    goto_airport = HLA('Go(Home, Airport)', precond='At(Home)', effect='At(Airport)
 & ~At(Home)'
```

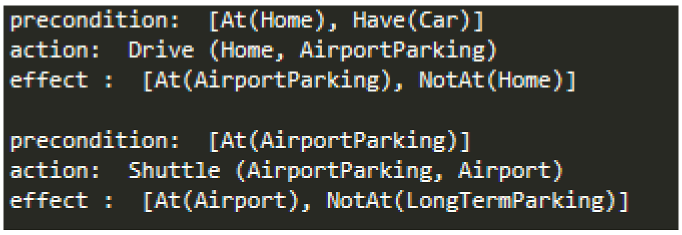
Listing 6.2 Action Code `goto_airport`

After defining the hierarchy and action, the next step is to create a new problem. Problems can be formed as in Listing 6.3. When creating a new Problem, the parameters required are initial conditions, final conditions, and actions taken. In this case the initial condition is being at 'At (Home)' house and having Have (Cash) money. The expected final condition (goal) is at the At (Airport) airport and still have cash (cash).

```
    PlanProblem('At(Home) & Have(Cash) & Have(Car) ', 'At(Airport) & Have(Cash)',
              [goto_airport])
```

Listing 6.3 Problem Codes `goto_airport`

The action that can be done is goto_airport so that the output of the problem above is the following actions: Drive (Home, AirportParking) and Shuttle (AirportParking, Airport). This means that in order to reach the destination at the airport and have money ('At (Airport) & Have (Cash)') fulfilled, the solution is to drive your own car. Driving a car can be divided into two detailed actions namely driving from the house to the airport parking lot then taking the shuttle bus from the airport parking lot to the airport gate. Details of each action can be seen in Figure 6.1:



```
precondition:  [At(Home), Have(Car)]
action:  Drive (Home, AirportParking)
effect :  [At(AirportParking), NotAt(Home)]

precondition:  [At(AirportParking)]
action:  Shuttle (AirportParking, Airport)
effect :  [At(Airport), NotAt(LongTermParking)]
```

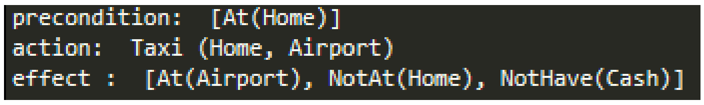
Figure 6.1 Program Output in the Form of Detail from the goto_airport Action

6.3 PRACTICUM ASSIGNMENTS

```
from ai_pkg.planning import (HLA, Problem as PlanProblem)

solution = PlanProblem.hierarchical_search(problem, library)
for i in range(0, len(solution)):
   print('precondition: ', solution[i].precond)
   print('action: ' , solution[i].name,  solution[i].args)
   print('effect : ',  solution[i].effect)
```

Listing 6.4 Code of the Hierarchical Planning Program for Cases Going to the Airport

1. **(Value 40)** Run the program for the case of going to the airport by the way:
   - Write the code in Listing 6.4
   - Add the hierarch to Listing 6.1, the goto_airport action to Listing 6.2, and the PlanProblem to Listing 6.3 before calling the hierarchical_search function.
   - Run the program and compare the output with Figure 6.1
2. **(Value 60)** Change the program above so that the results that come out are to go by taxi as the output in Figure 6.2 follows:



```
precondition:  [At(Home)]
action:  Taxi (Home, Airport)
effect :  [At(Airport), NotAt(Home), NotHave(Cash)]
```

Figure 6.2 Program Outcome in the Form of Action Going Using a Taxi

PRACTICUM 7 THEORY OF PROBABILISTICS IN UNCERTAINITY

Meeting	: 7
Total Time Allocation	: 90 minutes
• Pre-Test	: 15 minutes
• Practicum	: 60 minutes
• Post-Test	: 15 minutes

Total Rating Score	: 100%
• Pre-Test	: 25%
• Practicum	: 35%
• Post-Test	: 40%

7.1 OBJECTIVES AND INDICATORS OF ACHIEVEMENTS

After following this practicum, students are expected to:

1. Able to use probabilistic theories to overcome uncertainty in the implementation of tracking solutions in a programming language.

Achievement indicators are measured by:

1. Able to use probabilistic theory to overcome uncertainty in the implementation of tracking solutions in a programming language.

7.2 SUPPORTING THEORY

PROBABILITY

Probability indicates whether something will happen or not.

$$p(x) = \frac{\text{jumlah kejadian berhasil}}{\text{jumlah semua kejadian}}$$

For example from 10 scholars, 3 people master cisco, so the opportunity to choose a bachelor who master cisco is: p (cisco) = 3/10 = 0.3

The method of certainty with the Bayes Theorem

$$p(H_i \mid E) = \frac{p(E \mid H_i) * p(H_i)}{\sum_{k=1}^{n} p(E \mid H_k) * p(H_k)}$$

with:

$p(H_i \mid E)$	= the probability of the Hi hypothesis is true if given evidence (fact) E
$p(E \mid H_i)$	= probability of the emergence of evidence (fact) E if known hypothesis Hi is true
$p(H_i)$	= probability of Hi hypothesis (according to previous results) regardless of any evidence
n	= number of possible hypotheses

Example :

Asih has symptoms of freckles on her face. The doctor suspected that Asih had smallpox:

- the probability of appearance of spots on the face, if Asih is exposed to smallpox → p(bintik2|cacar) = 0,8
- Asih's probability of getting smallpox regardless of any symptoms → p(cacar) = 0,4
- the probability of appearance of spots on the face, if Asih is allergic → p(bintik2|alergi) = 0,3

- Asih's probability of being allergic regardless of any symptoms → p(alergi) = 0,7
- probability of appearance of spots on the face, if Asih is spotty → p(bintik2|jerawatan) = 0,9
- • Asih spotty probabilities regardless of any symptoms → p(jerawatan) = 0,5

then :

$$p(H_i \mid E) = \frac{p(E \mid H_i) * p(H_i)}{\sum_{k=1}^{n} p(E \mid H_k) * p(H_k)}$$

- Asih's probability of getting smallpox because there are spots on his face:

$$\text{p (cacar | bintik)} = \frac{\text{p (bintik | cacar) * p (cacar)}}{\text{p (bintik | cacar) * p (cacar) + p (bintik | alergi) * p (alergi) + p (bintik | jerawat) * p (jerawat)}}$$

$$\text{p (cacar | bintik)} = \frac{(0.8)*(0.4)}{(0.8)*(0.4)+(0.3)*(0.7)+(0.9)*(0.5)} = \frac{0.32}{0.98} = 0.327$$

- The probability of Asih being allergic because there are spots on his face:

$$\text{p (alergi | bintik)} = \frac{\text{p (bintik | alergi) * p (alergi)}}{\text{p (bintik | cacar) * p (cacar) + p (bintik | alergi) * p (alergi) + p (bintik | jerawat) * p (jerawat)}}$$

$$\text{p (alergi | bintik)} = \frac{(0.3)*(0.7)}{(0.8)*(0.4)+(0.3)*(0.7)+(0.9)*(0.5)} = \frac{0.21}{0.98} = 0.214$$

- Asih spotty probability because there are spots on his face:

$$\text{p (jerawat | bintik)} = \frac{\text{p (bintik | jerawat) * p (jerawat)}}{\text{p (bintik | cacar) * p (cacar) + p (bintik | alergi) * p (alergi) + p (bintik | jerawat) * p (jerawat)}}$$

$$\text{p (jerawat | bintik)} = \frac{(0.9)*(0.5)}{(0.8)*(0.4)+(0.3)*(0.7)+(0.9)*(0.5)} = \frac{0.45}{0.98} = 0.459$$

If after testing the hypothesis one or more evidence (facts) or new observations appear then:

$$p(H \mid E,e) = p(H \mid E) * \frac{p(e \mid E,H)}{p(e \mid E)}$$

with:

e	= old evidence
E	= evidence or new observations
p(H|E,e)	= the probability of the H hypothesis is true if new E evidence emerges from the old evidence
p(H|E)	= the probability of the H hypothesis is true if given evidence E.
p(e|E,H)	= the link between e and E if the hypothesis is true
P(e|E)	= the link between e and E regardless of any hypothesis.

7.3 PRACTICUM ASSIGNMENTS

Post Test 1

It is known that data from the Department of Health has data on the number of patients staying at hospitals in the Jogjakarta area in VIP, class I, class II, class III and VVIP. The data is shown in Table 8.1

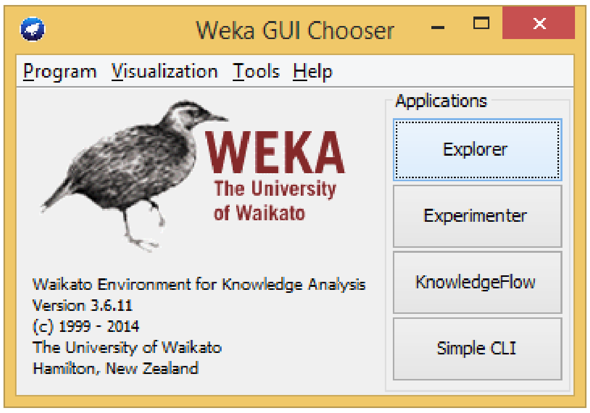
Table 8.1 Data on the number of hospitalized patients in the Jogjakarta area

Halte	VVIP	VIP	CLS I	CLS II	CLS III	class
Hospital A	40	45	44	33	20	high
Hospital B	21	33	43	50	42	medium
Hospital C	46	30	16	36	22	high
Hospital D	53	26	26	39	21	high
Hospital E	33	37	39	32	18	high
Hospital F	21	27	35	42	39	low
Hospital G	37	47	38	32	18	high
Hospital H	39	38	33	38	25	medium

By using weka calculate probability using the Bayes theorem Certainty method.

Practicum steps using WEKA

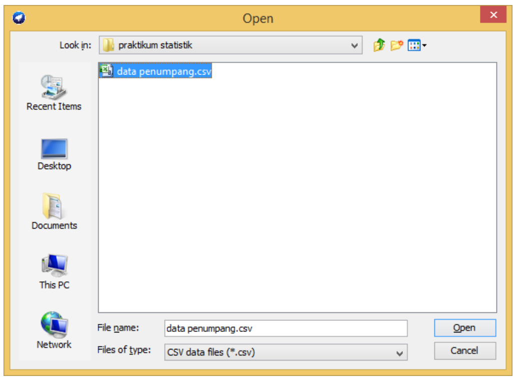
1. Type the data in table 8.1 in excel and save it as .CSV
2. Open the WEKA program where it looks like in Figure 8.1



Waikato Environment for Knowledge Analysis
Version 3.6.11
(c) 1999 - 2014
The University of Waikato
Hamilton, New Zealand

Figure 7.1 Initial Display of WEKA

3. 3. Open the file and select where to place the file. What needs to be noted is that files from Excel are stored in CSV so they can be read by the WEKA program.



Look in: praktikum statistik
data penumpang.csv
File name: data penumpang.csv
Files of type: CSV data files (*.csv)

Figure 8.2 Open .CSV file

4. 4. In the top Tabulation select classification -> Select Bayes -> star

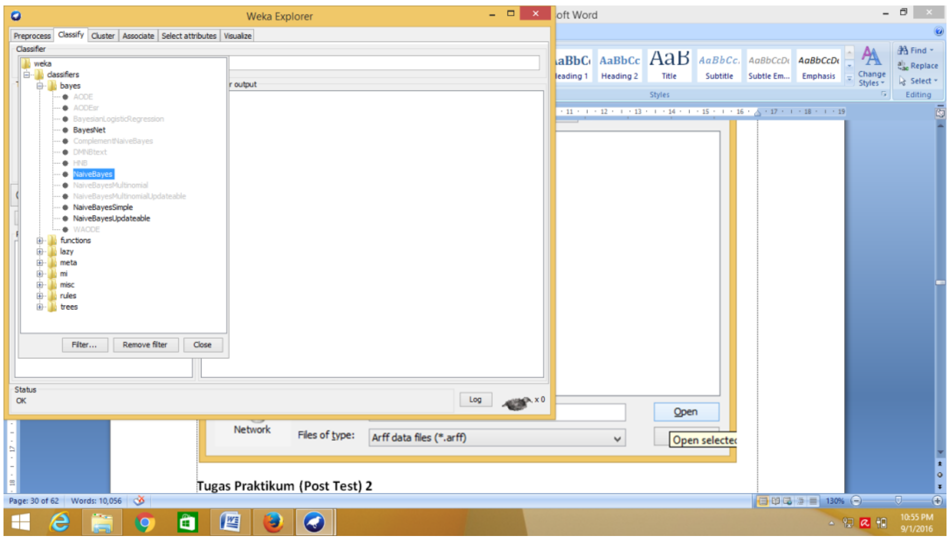


Figure 8.3 Selection of the BAYES method

5. 5. Analyze the results of the WEKA program, the results of the BAYES calculation process using WEKA

Post Test 2

From the table below, do it like in Task Postest 1. And make an analysis.

Table 8.2 Old Data Writing Student thesis

Students	Writing period for Chapter 1 and data collection (mth)	Writing time for chapter 2 (mo)	Writing period for chapter 3 (mo)	Writing times for chapters 4 & 5 (mo)	Duration of thesis completion period
1	3.5	3.60	3.0	1	Fast
2	4	3.25	2.5	2.5	Fast
3	4	3.10	3.5	2.7	Slow
4	5	2.98	4.2	1.5	Normal
5	4.5	3.00	2.1	2.0	Normal
6	6	3.20	3.2	3.0	Slow
7	4	2.75	2.1	1	Slow
8	4.5	2.90	3;0	0.7	Slow
9	6	3.35	3.1	1.2	Normal
10	4	3.10	4.3	2	Fast

PRACTICUM 8 MULTI AGENT

	Meeting	: 8
	Total Time Allocation	: 90 minutes
•	Pre-Test	: 15 minutes
•	Practicum	: 60 minutes
•	Post-Test	: 15 minutes

	Total Rating Score	: 100%
•	Pre-Test	: 25%
•	Practicum	: 35%
•	Post-Test	: 40%

8.1 PURPOSE AND ACHIEVEMENTS

After following this practicum, students are expected to:

1. Able to solve multi agent problems.

Achievement indicators are measured by:

1. Able to make multi agent system designs based on agent system design and analysis methods, as well as implementing multi agent simulations

8.2 SUPPORTING THEORY

Multiagent Planning Problem

In Handout 6, it has been explained about planning for a single agent. In fact, some problems require several agents working together. Each agent faces a multiagent planning problem where each agent tries to achieve his goals with the help and obstruction of other agents.

When there is only one agent doing planning, then there is only one goal achieved by that agent. But when there are two or more agents planning, each agent shares the same goal. For example, in the problem of a double tennis game where both players in a team have the same goal of being able to return the ball passed by the opposing team. In multiagents there is a problem that is if the existing agents have different goals. In the case of double tennis, goals on the opposite team are the opposite.

Planning with multiple simultaneous actions

The purpose of this section is to find a plan of action that is compatible with multiagent issues. The plan is called true or appropriate, ie if the action is carried out by agents, then the goal can be achieved.

The problem of double tennis is one example of a simple case of multiagents. In this case two tennis players A and B play in one team and can be in four positions: back left (LeftBaseLine), right back (RightBaseLine), front left (LeftNet), and front right (RightNet). The ball can be returned if the player is in the same place as the ball position.

The conditions for this problem are:

- `Approaching(Ball, loc)` : Ball passed to position `loc`
- `Returned(Ball)` : one of the players hits and returns the ball from the correct position which causes the ball to return to the opposing team's area.
- `At(player, loc)` : the player is in the loc position
- `~At(player, loc)` : the player is not in the loc position

If in the initial condition the position of Player A is in the back left and player B is in the front right position, the ball is hit by the opponent towards the right back, then the initial conditions can be written as follows:

```
    (A,LeftBaseLine) & At(B,RightNet) & Approaching(Ball,RightBaseLine) & Team(A,B)
& Team(B,A)
```

Some actions that can be taken to achieve goals include:

- `Hit(player, Ball, loc)` : the action returns Ball if a player is in a loc position. The complete action can be written as follows:

```
    Action('Hit(player, Ball, loc)',
       precond='Approaching(Ball, loc) & At(player, loc)',
       effect='Returned(Ball)')
```

- `Go(player, to, loc)` : player moves from loc position to position to. The complete action can be written as follows:

```
Action('Go(player, to, loc)',
      precond='At(player, loc)',
      effect='At(player, to) & ~At(player, loc)')
```

The goal of this game is:

```
    Returned(Ball) & At(a, RightNet) & At(a, LeftNet)
```

In the goal above there is a new variable, namely a. This can happen in a multiagent case because we don't care which players end up in the RightNet or LeftNet position. SeSlow is still a player in one of these positions so the goal can still be achieved.

8.3 PRACTICUM ASSIGNMENTS

Examples of cases in the double tennis game can be written as in listing 8.1 below:

```
    from ai_pkg.planning import PlanningProblem, Action, goal_test
    from ai_pkg.utils import expr

    def double_tennis_problem():
        initial  =  'At(A,  LeftBaseLine)  &  At(B,  RightNet)  &  Approaching(Ball,
RightBaseLine) & Team(A, B) & Team(B, A)'
        goal = 'Returned(Ball) & At(a, RightNet) & At(a, LeftNet)'
        action = [Action('Hit(player, Ball, loc)',
                          precond='Approaching(Ball, loc) & At(player, loc)',
                          effect='Returned(Ball)'),
                   Action('Go(player, to, loc)',
                          precond='At(player, loc)',
                          effect='At(player, to)')]

        return PlanningProblem(init=initial,
                                goals=goal,
                                actions=action)
    if __name__=’__main__’:
        p = double_tennis_problem()
        print(goal_test(p.goals, p.init))

        solution = [expr(“Go(A, RightBaseLine, LeftBaseLine)”),
                    expr(“Hit(A, Ball, RightBaseLine)”),
                    expr(“Go(A, LeftNet, RightBaseLine)”)]

        for action in solution:
            p.act(action)

        print(goal_test(p.goals, p.init))
```

Listing 8.1 Code Program of the Double Tennis Game

1. **(Value 40)** Run the multiagent Python double tennis game program in a way:
   - Write down all program code in listing 8.1.
   - Run the program using the console
   - Record the results
   - Change or add some actions to the solution to prove that the solution mentioned in Listing 8.1 has reached the goal
2. **(Value 60)** In the case of the double tennis game there are other solutions, namely:
3. Player A: go from the back left (LeftBaseLine) to the front left (LeftNet) and do nothing (NoOp)
4. Player B: go from front right (RightNet) to back right (RightBaseLine) and return the ball (Hit)
   - Write the solution above into the program by:
   - Add a NoOp (player) action to the action in PlanningProblem with preconds and effects blank (no action whatsoever so the conditions before and after the action don't change)
   - Replace the solution in the solution with the solution mentioned above

PRACTICUM 9 MAKING CUSTOM PACKAGE PYTHON

Meeting	: 9
Total Time Allocation	: 90 minutes
• Pre-Test	: 15 minutes
• Practicum	: 60 minutes
• Post-Test	: 15 minutes

Total Rating Score	: 100%
• Pre-Test	: 25%
• Practicum	: 35%
• Post-Test	: 40%

9.1 PURPOSE AND ACHIEVEMENTS

After following this practicum, students are expected to:

1. Able to complete and distribute custom packages in python.

Achievement indicators are measured by:

2. Able to create and distribute custom packages for applications using Python

9.2 SUPPORTING THEORY

Package Structure in Python

To create a custom package we need to create a Python project with the initial arrangement as follows:

```
/ai_pkg
  /ai_pkg
    __init__.py
```

`ai_pkg` is the name of the package to be created. Therefore, declaring the package name needs to be done in the file `__init__.py`. Add the following code to the file `__init__.py`.

```
name = "ai_pkg"
```

Creating Files in Package

Next we create other files that will be integrated into this project and distributed as a custom package. Add file `setup.py`, `LICENSE`, and `README.md` in the project directory.

```
/ai_pkg
  /ai_pkg
    __init__.py
  setup.py
  LICENSE
  README.md
```

Create setup.py

`setup.py` is a script to form packages using setuptools. This file defines all things about the package, such as name, version, description, author, and other information.

```
import setuptools

with open("README.md", "r") as fh:
    long_description = fh.read()

setuptools.setup(
    name="ai_pkg",
    version="0.0.1",
    author="Example Author",
    author_email="author@example.com",
    description="A small example package",
    long_description=long_description,
    long_description_content_type="text/markdown",
    url="https://github.com/pypa/sampleproject",
    packages=setuptools.find_packages(),
    classifiers=[
        "Programming Language :: Python :: 3",
        "License :: OSI Approved :: MIT License",
        "Operating System :: OS Independent",
    ],
)
```

Create README.md

This file contains the package description. Can be filled with any information from package usage, usage examples, etc.

```
# Example Package

This is a simple example package. You can use
[Github-flavored          Markdown](https://guides.github.com/features/mastering-markdown/)
to write your content.
```

Create LICENSE

The creation of this file is important if the package that we created is uploaded and distributed

```
Copyright (c) 2018 The Python Packaging Authority

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.
```

```
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

Creating Package Distribution Using Python Wheel

The next step is to create a package distribution in the form of an archive that can be installed using pip. To create a package distribution, **setuptools** and **wheel** installation is required.

```
pip install --user --upgrade setuptools wheel
```

After **setuptools** and **wheels** are installed, run the following command in the directory containing the **setup.py** file.

```
python setup.py sdist bdist_wheel
```

This command will produce two files in the **dist** folder, **.tar.gz** is the code archive and **.whl** is the distribution file. The pip command will install the package using a distribution **(.whl)** file but an archive file **(.tar.gz)** can be used if the distribution file is not available.

```
dist/
  ai_pkg-0.0.1-py3-none-any.whl
  ai_pkg-0.0.1.tar.gz
```

9.3 PRACTICUM ASSIGNMENTS

Create **ai_pkg** version 0.0.2 custom package in a way:

- Download all module files in the custom package **ai_pkg** from the URL provided by the assistant.
- Add codes that are worked on practicum material 3, 4, 5, 6, and 8 into the module
- Change version to version 0.0.2
- Add the required files according to instructions in handout 9 and create the custome package file.
- Collect **.whl** and **.tar.gz** files in the **dist** folder

PRACTICUM 10 APPLICATION IN ARTIFICIAL INTELLIGENCE (PAKAR SYSTEM, ARTIFICIAL NEURAL NETWORK)

Meeting	: 10
Total Time Allocation	: 90 minutes
• Pre-Test	: 15 minutes
• Practicum	: 60 minutes
• Post-Test	: 15 minutes

Total Rating Score	: 100%
• Pre-Test	: 25%
• Practicum	: 35%
Post-Test	: 40%

10.1 PURPOSE AND ACHIEVEMENTS

After following this practicum, students are expected to:

1. Able to use AI applications and can create system interface designs using the GUI.Python application.

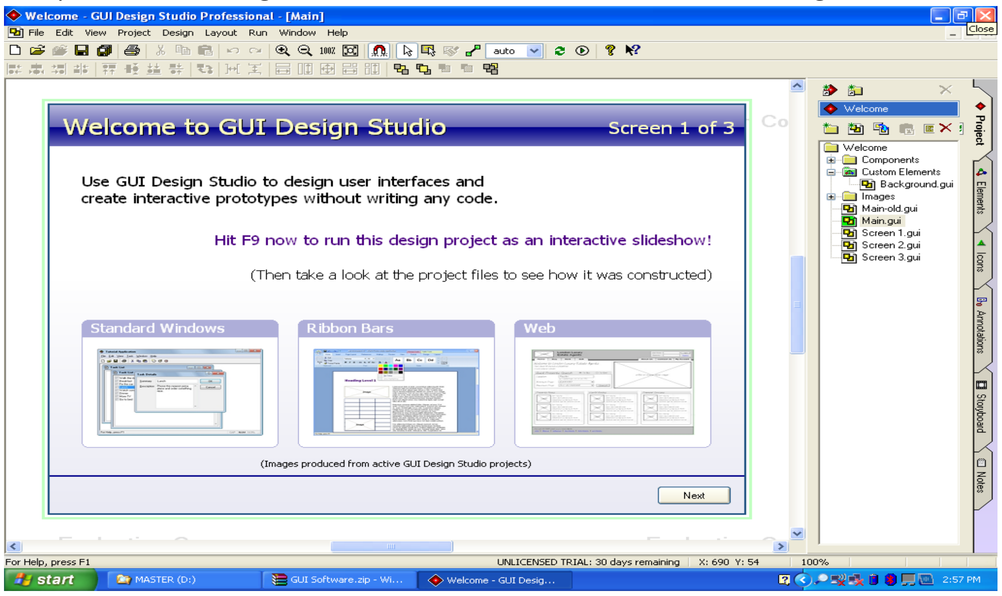
Achievement indicators are measured by:

1. Students understand various applications in AI and can create system interface designs using the GUI.Python application.

10.2 PRACTICUM STEPS

Steps to use GUI Design Studio

a. Open the GUI Design Studio then the main form can be seen in Figure 10.1.


Welcome - GUI Design Studio Professional - [Main]

Welcome to GUI Design Studio   Screen 1 of 3

Use GUI Design Studio to design user interfaces and create interactive prototypes without writing any code.

Hit F9 now to run this design project as an interactive slideshow!

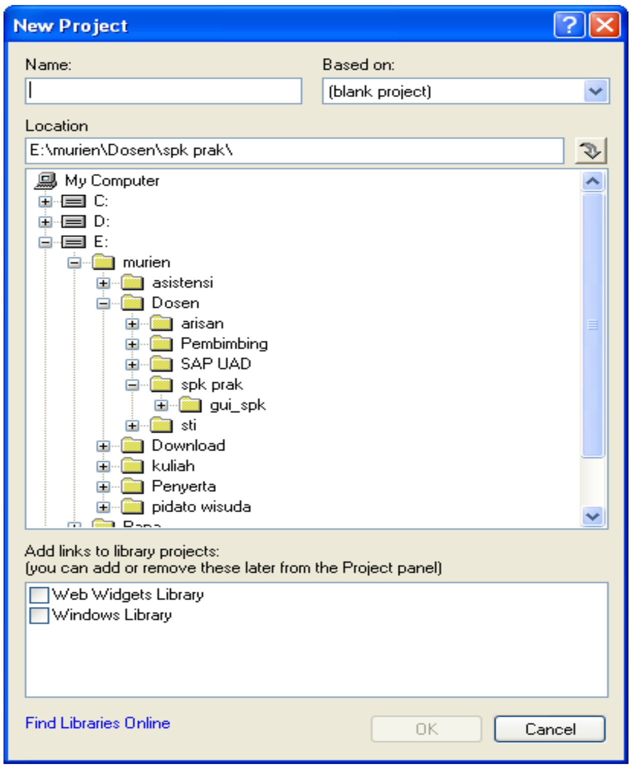
(Then take a look at the project files to see how it was constructed)

(Images produced from active GUI Design Studio projects)

For Help, press F1   UNLICENSED TRIAL: 30 days remaining   X: 690 Y: 54   100%

Figure 10.1 Display of the Design Studio GUI Main Form

b. After the main form opens, select New Project and give the name of your project and don't forget to choose the location of the drive where you will save it. If you want to add a link to the library, it is checked. Like Figure 10.2.



New Project

Name:

Based on: (blank project)

Location: E:\murien\Dosen\spk prak\

My Computer
C:
D:
E:
murien
asistensi
Dosen
arisan
Pembimbing
SAP UAD
spk prak
gui_spk
sti
Download
kuliah
Penyerta
pidato wisuda

Add links to library projects:
(you can add or remove these later from the Project panel)

Web Widgets Library
Windows Library

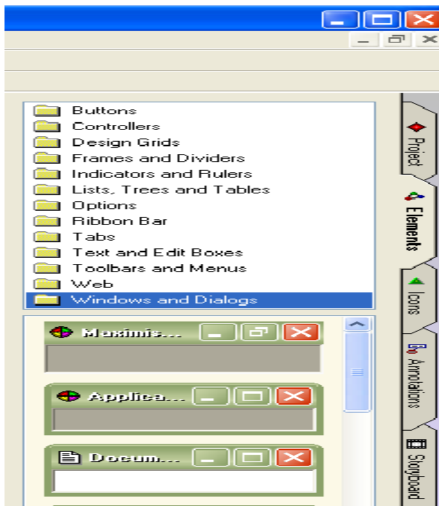
Find Libraries Online

OK

Cancel

Figure 10.2 Form for selecting the New Project repository

c. Once saved with a new project, you are ready to design the design you want. You can use various components on the left side of the screen. There are Project, Element, Icon and others. Like Figure 10.3.



Buttons
Controllers
Design Grids
Frames and Dividers
Indicators and Rulers
Lists, Trees and Tables
Options
Ribbon Bar
Tabs
Text and Edit Boxes
Toolbars and Menus
Web
Windows and Dialogs

Figure 10.3 Components for building interfaces

d. To start the design, first please create a design form, by clicking New Design on the project or using Crtl + N. After the new dialog window exits, please add a component. Drag and drop the selected components according to the design theme. Like Figure 10.4.

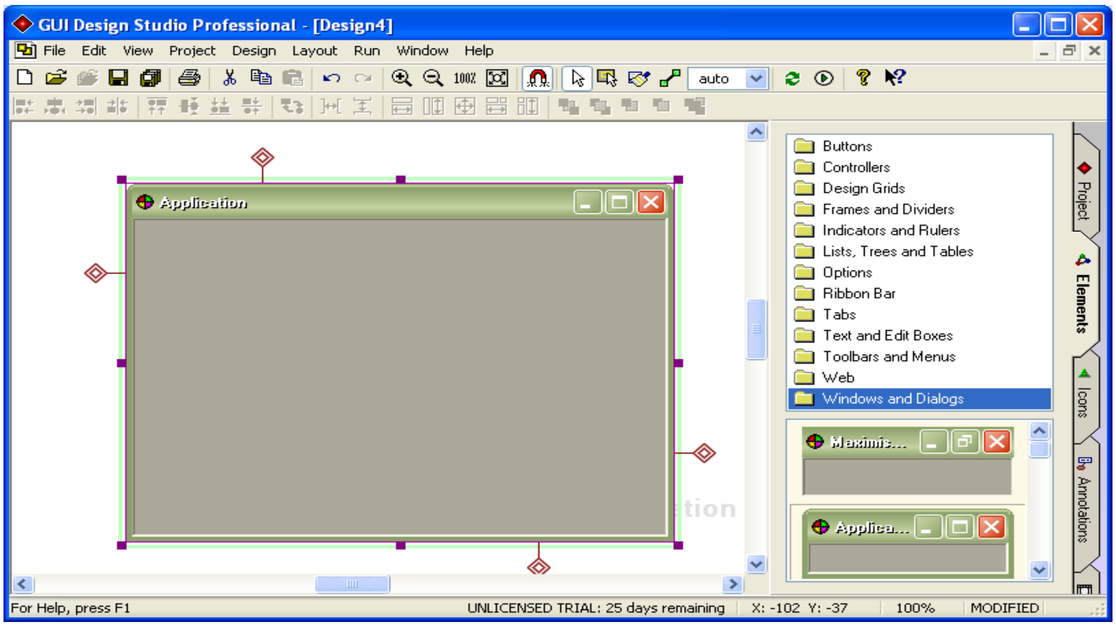


Figure 10.4 Display of the worksheet on which to create interface design

e. To arrange each component or element can be done by double clicking the component to be modified. And to move an element, you can use the up-down and left-right arrow keys on the keyboard, to minimize and freeze element size, you can use the combination of the Shift key and the 4-way key mentioned earlier..

Interface Design

a. Main Menu

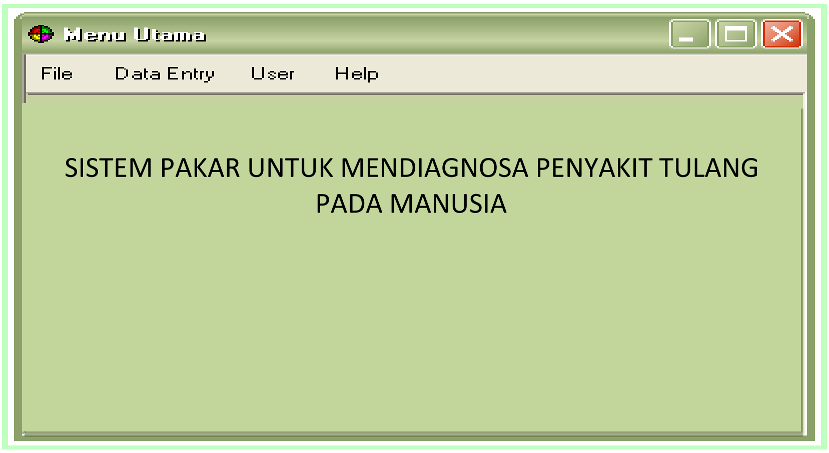
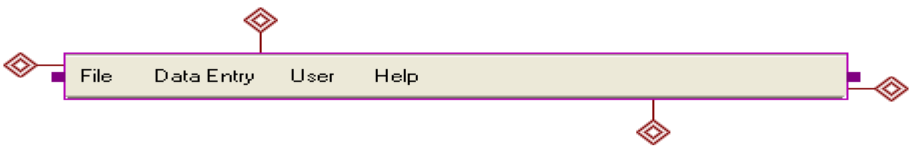

Menu Utama
File Data Entry User Help
SISTEM PAKAR UNTUK MENDIAGNOSA PENYAKIT TULANG PADA MANUSIA

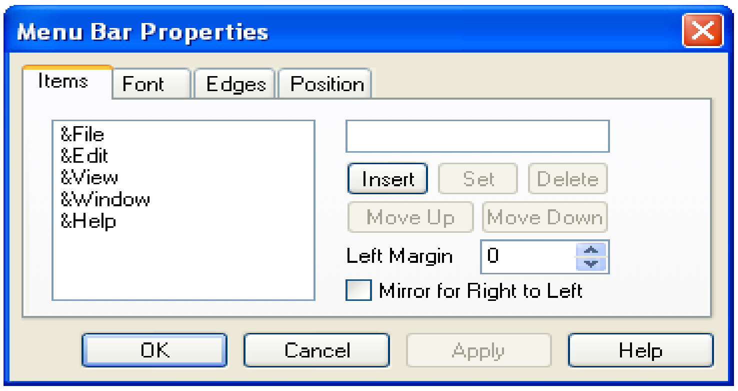
Figure 10.5 The main menu design

On the main menu there are several menus, namely the file menu, data entry, user and help. The file menu has a login, change password and exit sub menu. Data entry has a sub menu of diseases, causes, solutions, symptoms and rule base. Users have sub menus of user data and diagnostics, and help has sub menus about, instructions for use and programmers. To create a task bar as below:



Gambar 0.1 Image Task Bar

- Select Elements → Toolbars dan Menus
- Drag the Menu Bar image, if you want to make changes double click on the Menu Bar image, then do the settings in the Menu Bar properties as shown below.

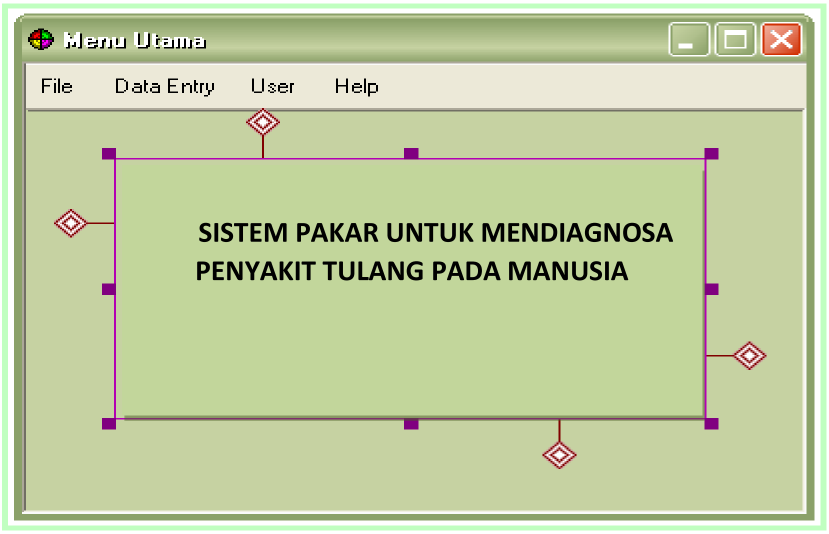


&File
&Edit
&View
&Window
&Help

Left Margin 0
Mirror for Right to Left

Figure 10.7 Display of the Properties Bar Menu

For giving the title on the main menu:



SISTEM PAKAR UNTUK MENDIAGNOSA PENYAKIT TULANG PADA MANUSIA

Figure 10.8 Title Design in the Main Menu

- Select Elements → Text and Edit Boxes
- Then do the settings in the Text Properties

To connect one interface to another interface, a link can be made from the Data Entry Menu to the Popup Menu by selecting the icon

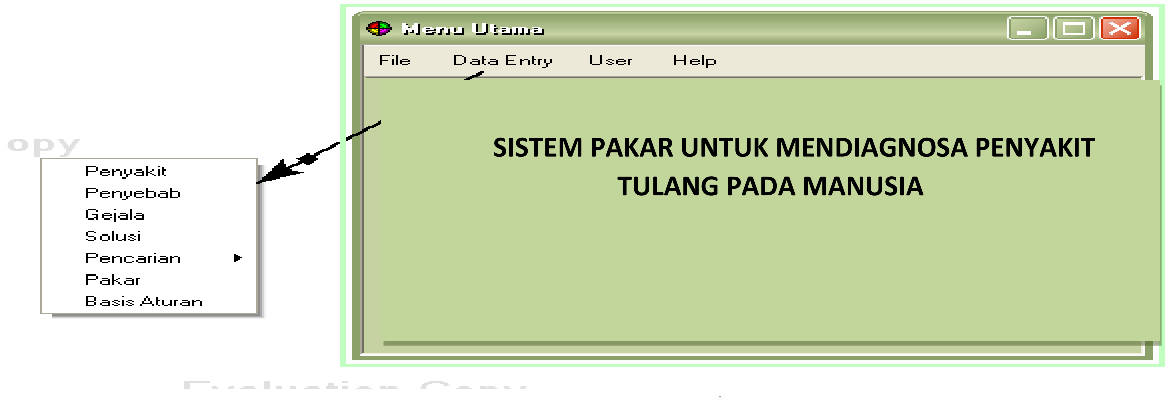


Figure 10.9 Main Menu Design with Popup Menu

For making Popup Menus can be done in a way:

- Select Elements → Toolbars dan Menu

- Drag the Popup image, if you want to make changes double click on the image popup menu then do the settings in the popup menu properties.

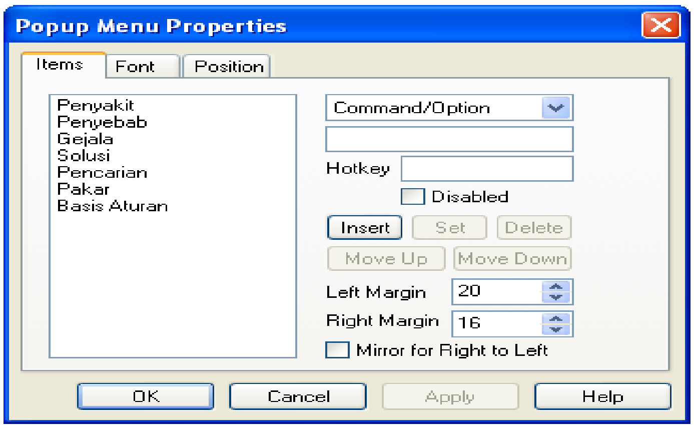


Figure 10.10 Display Popup Menu Properties

b. Login Menu

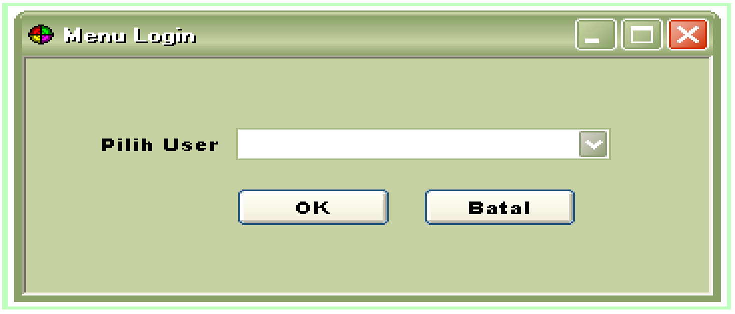


Figure 10.11 The design of the login menu

To make a button:

- Select Elements → Buttons
- • Then select the desired button, for example on this button

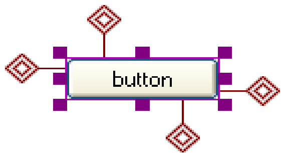
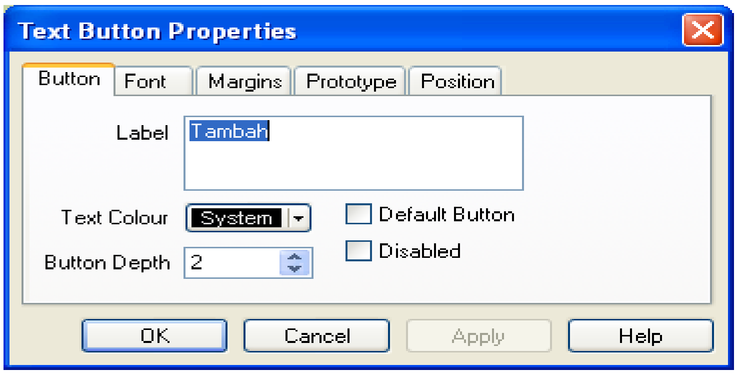


Figure 10.12 Image Button

- If you want to make changes double click on the button then make adjustments to the Text Button Properties as below:



Label: Tambah
Text Colour: System
Button Depth: 2
Default Button
Disabled

Figure 10.13 Display Text Button Properties

c. Disease Data Menu

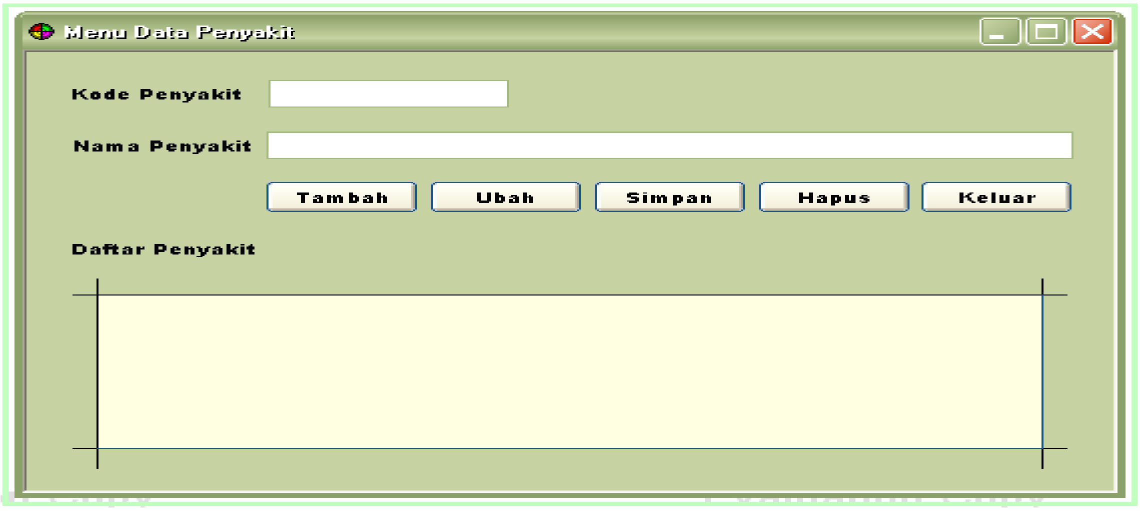


Figure 10.14 Disease data menu design

This menu is used to input disease data to be stored in the system. It is possible to add new types of diseases, or to remove one type of disease.

To make edit text:

- Select Elements → Text and Edit Boxes
- Select image

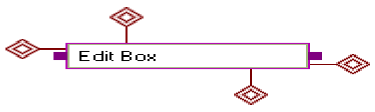
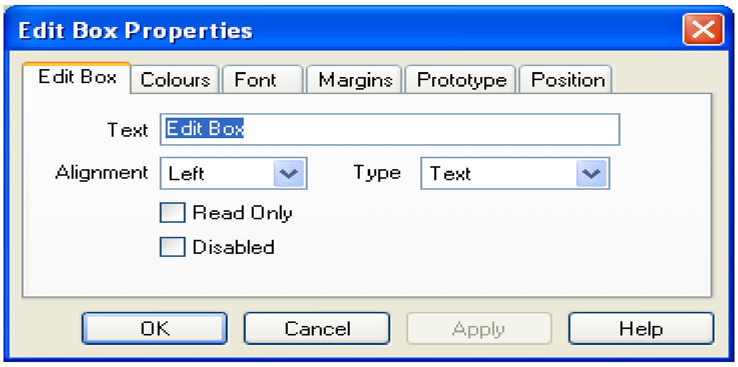


Figure 10.15 Image Edit Box

Then do the settings in the Edit Box Proporties like the image below by double clicking.


Edit Box Properties

Edit Box | Colours | Font | Margins | Prototype | Position

Text: Edit Box

Alignment: Left

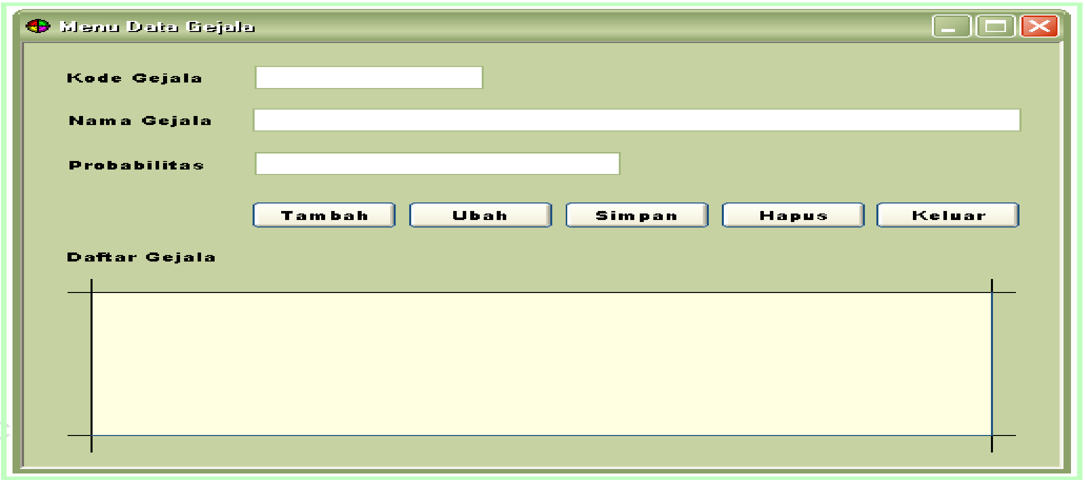
Type: Text

Read Only: unchecked

Disabled: unchecked

Figure 10.19. Display Edit Box Properties

d. Symptoms Data Menu


Menu Data Gejala

Kode Gejala

Nama Gejala

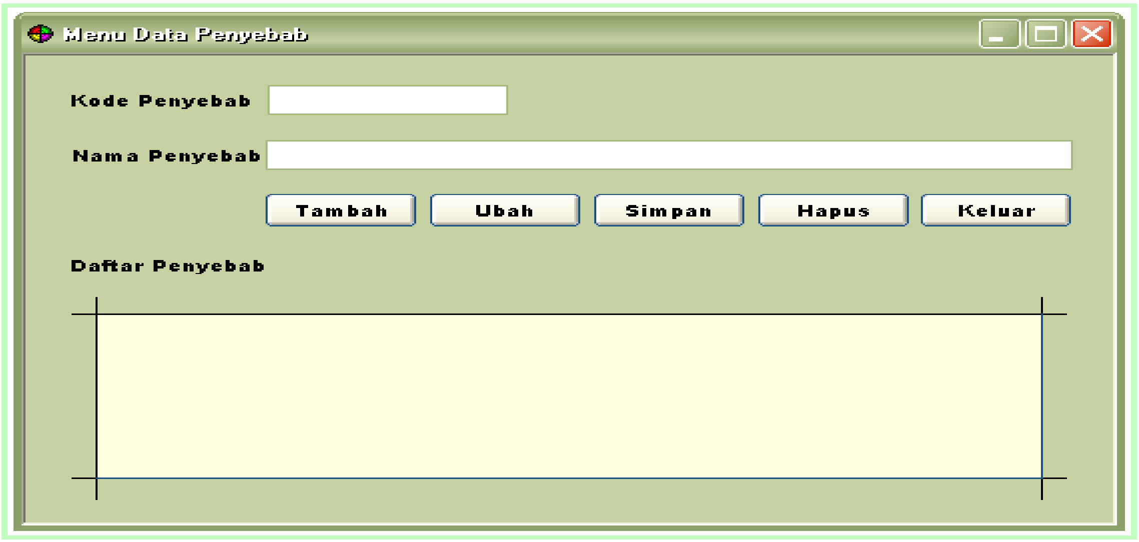
Probabilitas

Daftar Gejala

Figure 10.16 Symptoms data menu design

This menu is used to input symptoms and probability data for each disease. It's possible to add a new symptom, or to delete one of the symptoms.

e. Data Menu Cause


Menu Data Penyebab

Kode Penyebab

Nama Penyebab

Daftar Penyebab

Figure 10.17 Design of the causal data menu

This menu is used to input disease-causing data. Possible to add new causes, or to remove one of the causes of the disease.

f. Data Solutions Menu

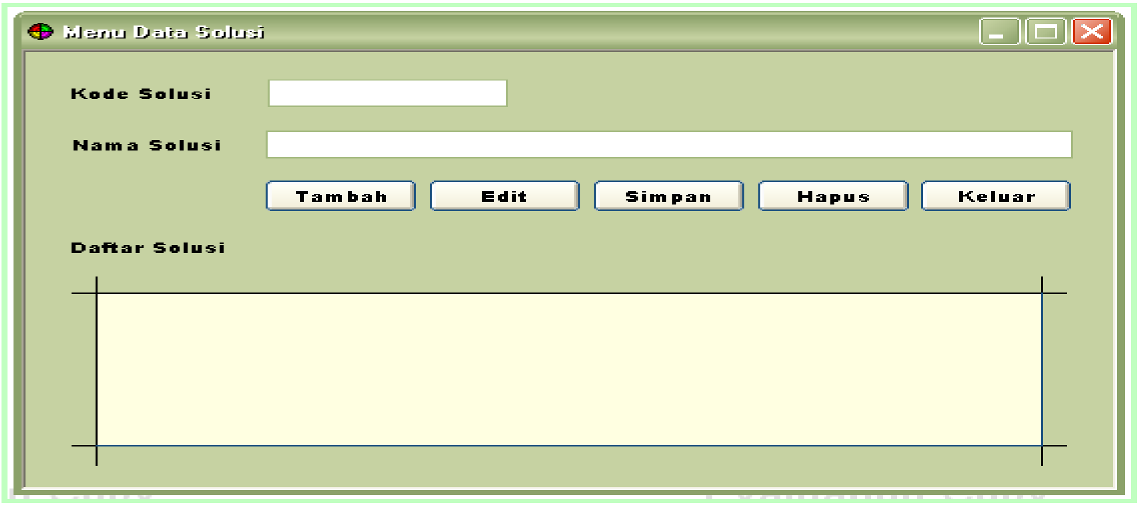


Figure 10.18 The design of the solution data menu

This menu is used to input data solutions. It is possible to add a new solution, or to remove one solution from each disease.

g. Patient Data Menu

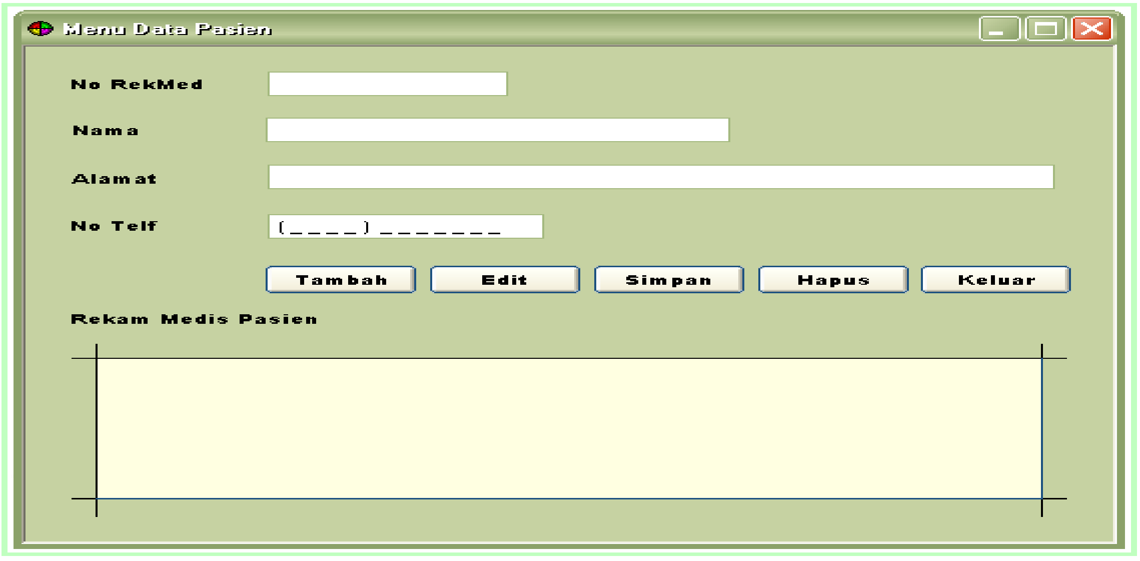


Figure 10.19 Patient data menu design

This menu is used to input patient data. It is possible to add new patients who will use this system to consult, or to delete one patient.

h. Rules menu

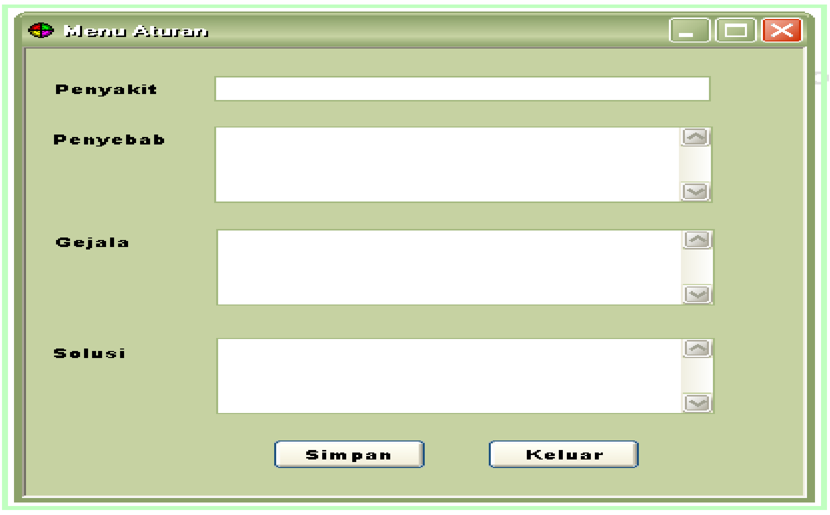


Figure 10.20 Rules menu design

This menu is used to display the rules of the rules used to search for diseases and solutions according to the disease based on symptoms.

i. Diagnosis Menu

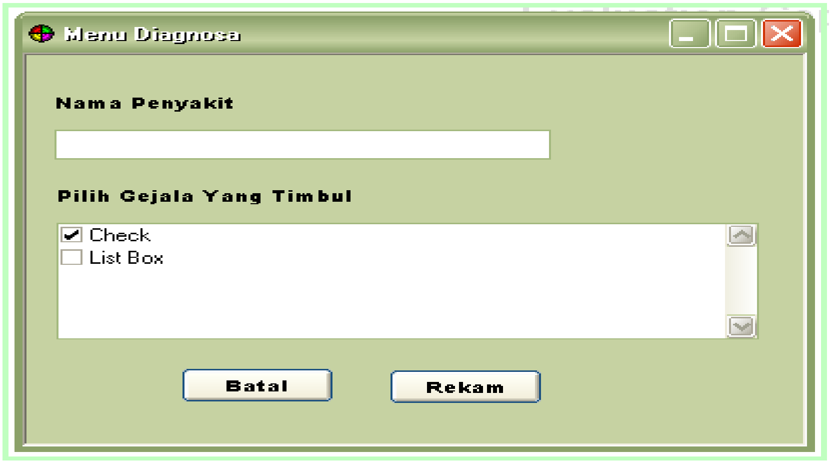


Figure 10.21 Consultation menu design

This menu is used for users who want to do a consultation. Later the user will choose the symptoms that are felt.

j. Record Form

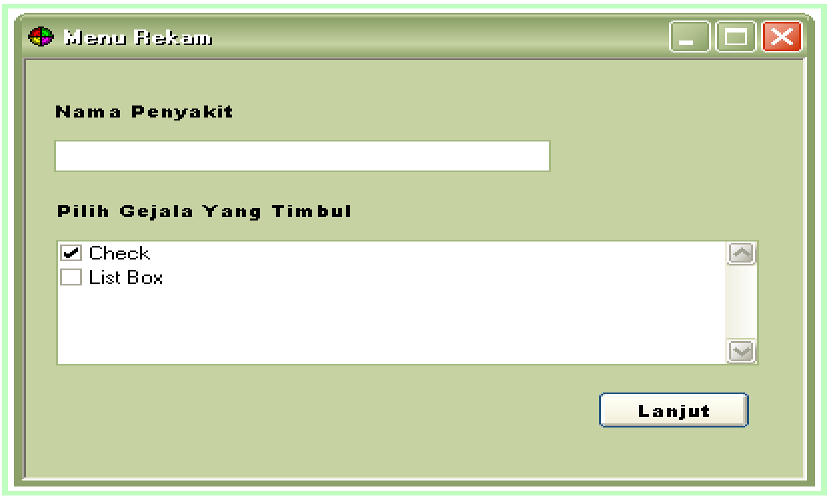


Figure 10.22 Record menu design

This menu is continued from the consultation menu. If on the consultation menu the user chooses to record, the symptoms that have been selected on the consultation menu will be displayed on this form.

k. Advanced Menu Design

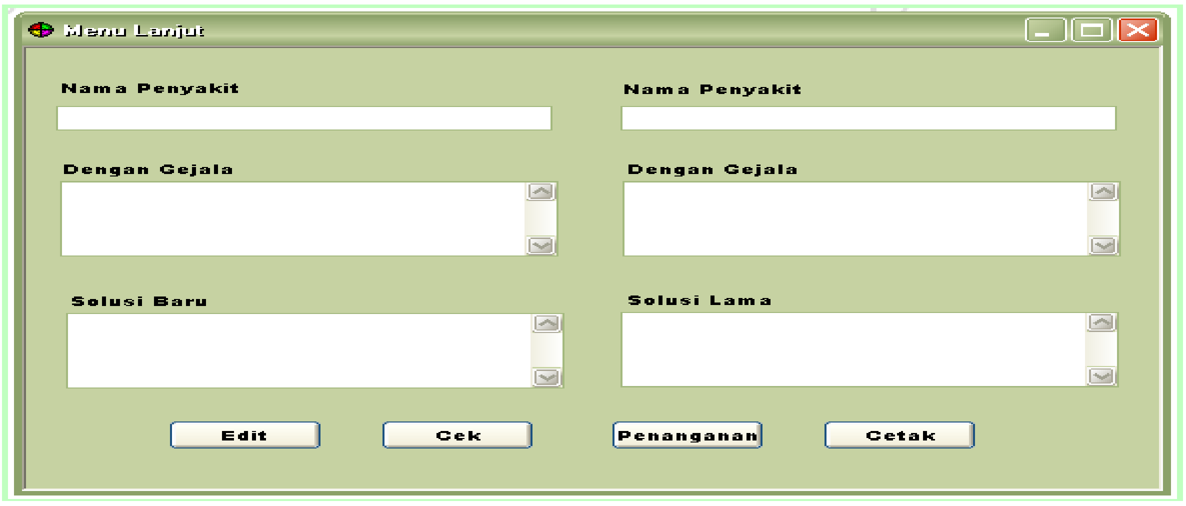
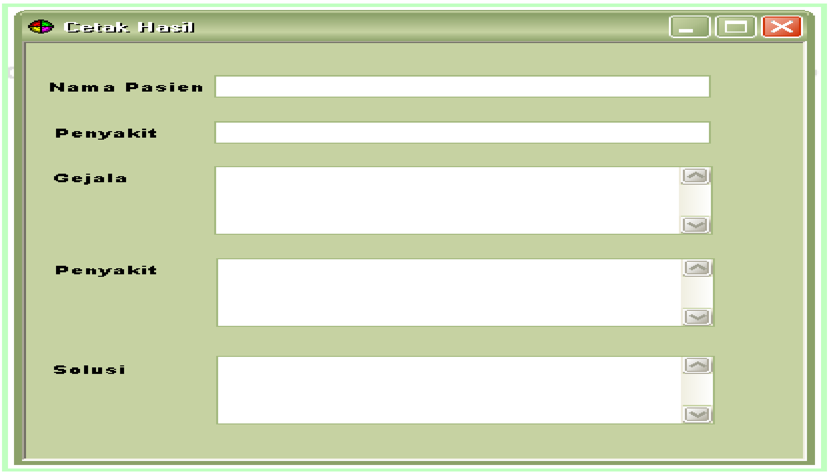


Figure 10.23 Advanced menu design

This menu is continued from the record form. In this form, the system will display a disease solution that has been stored in a database whose symptoms are similar to the new symptoms entered by the user.

l. Print Results



Nama Pasien

Penyakit

Gejala

Penyakit

Solusi

Figure 10.24 Draft print menu design

This menu is continued from the advanced form. If in the advanced form the user chooses to print, the results of the consultation can be printed.

m. Help menu

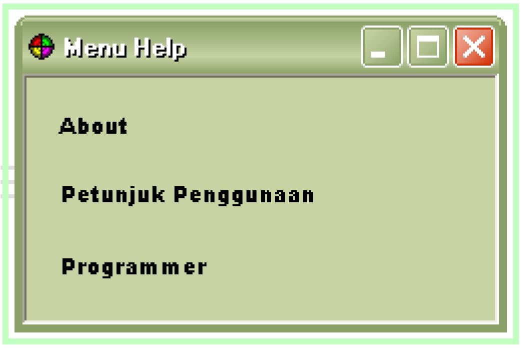


Figure 10.25 Design help menu

This menu will display about options that contain about the program, instructions for using the program and the programmer's identity.

BIBLIOGRAPHY

Kusumadewim S. 2003. Artificial Intelligence (Teknik dan Aplikasinya). Graha Ilmu, Yogyakarta.
Hermawan, A. 2006. Jaringan Syaraf Tiruan. ANDI OFFSET, Yogyakarta
Prateek, Joshi, Artificial Intelligence with Python, 2017.
Russel, Stuart, Artificial Intelligence a Modern Aproach, 2016
https://github.com/logpy/logpy
https://packaging.python.org/tutorials/packaging-projects/
http://mbp.sourceforge.net/index.html, tanggal akses 9 Agustus 2014, jam 13.10 WIB.
Winiarti.S, Buku Praktikum Kuliah Sistem Informasi, 2014.
Modul praktikum Statistika Informatika UAD 2016

LABORATORIUM
TEKNIK INFORMATIKA
UNIVERSITAS AHMAD DAHLAN

UNIVERSITAS AHMAD DAHLAN
1960