***Name and Title of the Report***

**Name:** Introduction to Artificial Intelligence; laboratory exercise 2

**Title:** ROBOT ARM

***Authors***

AKINDE EMMANUEL CSC/2012/021 ENGINEERING

OLADELE IDOWU CSC/2012/044 ENGINEERING

OLUWADAMILER YETUNDE CSC/2012/077 ECONOMICS

BOLUWATIFE FASUGBA CSC/2013/123 ENGINEERING

OLUWAYEMI ONI CSC/2013/137 ECONOMICS

OLUWASHINA KURE-OJO CSC/2012/053 ENGINEERING

***Contents***

**Page**

Name and Title of the report 1

Author 1

Contents 1

Glossary of Notations and Terms 2

Introduction 3

Problem Statement 4

Discussion of Result 4

Interpretation and Application of results 23

Summary and Conclusion 24

**GLOSSARY OF NOTATION AND TERMS**

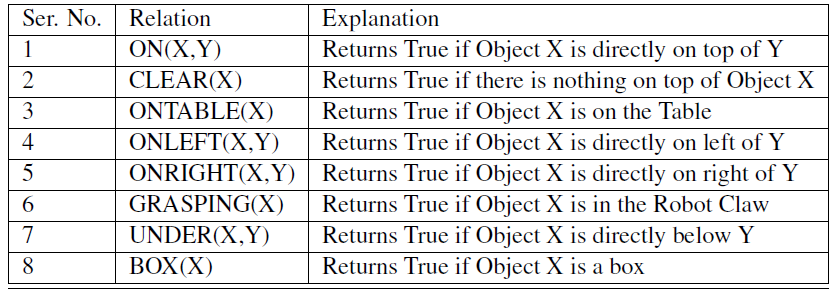
W – The robot world domain

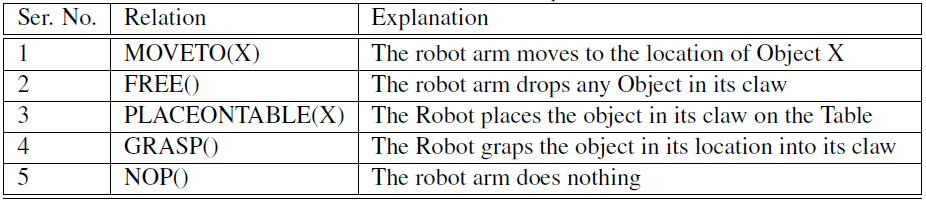
R – The robot manipulator

T – The supporting table

B – A set of blocks (three in numbers), with the same sizes, dimensions and weights.

FOPC – First Order Predicate Calculus



****

**INTRODUCTION**

Planning is the process of thinking about and synthesising the sequence of actions required for solving

a problem. Planning allows us to abstract and think about the problem world before

carrying out the activity.

In a monotonic environment, the state (or world) of the problem does not change

while the problem is being solved. In non-monotonic environment, the state of the problem

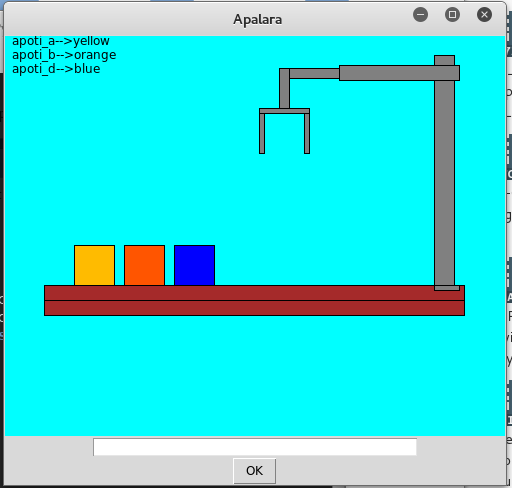
may change at some point before the solution is generated. In this exercise, we shall

consider planning in a monotonic robot motion environment. We shall be working with Ap´alar´a the robot.

Considering planning in a monotonic robot motion environment (where the state or world of the problem does not change while the problem is being solved), the robot can arrange boxes in its world, e.g by stacking and un-stacking the boxes. These actions will require the robot to determine some intermediate steps. This environment can be used to simulate and demonstrate planning, which is an important task in intelligent system.

The state space can be represented and manipulated using a language for expressing declarative knowledge.

The robot world is represented in the figure below



**PROBLEM STATEMENT**

Task 1

Write the plan to implement the following:

1. Swap any two boxes
2. Place a box under another box
3. Place a box on top of another box
4. Stack all the boxes on A
5. Stack all the boxes on B
6. Stack all the boxes on C

Generate the matrices for the state transitions of the above plans

Task 2

1. Using the python programming language, develop a software that can implement the plans you wrote in Task 1 above
2. Test your system extensively and discuss its limitations
3. Discuss the language of your robot plan.

**DISCUSSION OF RESULTS**

**Task 1**

**[0 0 0] BOX A -1**

**[0 0 0] BOX B - 2**

**[1 2 3 ] BOX C- 3 (initial matrix)**

Swap any two Boxes:

Swapping Box A and C [0 0 0]

START [0 0 0]

MOVE TO (A) [3 2 1]

GRASP ()

PLACE ON (B,A)

FREE ()

MOVE TO (C)

GRASP ()

PLACE TO\_THE\_LEFT\_OF (B)

FREE ()

MOVE TO (A)

GRASP ()

PLACE TO\_THE\_RIGHT\_OF (B)

FREE

END

Place a box under another box:

The operation in not possible

Place box A on B:

START

MOVE TO (C) [0 0 0]

GRASP () [3 0 0]

PLACE ON (A,C) [1 2 0]

FREE ()

END

Stack all the boxes on A:

START

MOVE TO (C) [2 0 0]

GRASP () [3 0 0]

PLACE ON (A,C) [1 0 0]

FREE ()

MOVE TO (B)

GRASP ()

PLACE ON (C,B)

FREE ()

END

Stack all the boxes on B:

START [0 1 0]

MOVE TO (C) [0 3 0]

GRASP () [0 2 0]

PLACE ON (B,C)

FREE ()

MOVE TO (A)

GRASP ()

PLACE ON (C,A)

FREE ()

END

Stack all the boxes on B:

START

MOVE TO (A) [0 0 2]

GRASP () [0 0 1]

PLACE ON (C,A) [0 0 3]

FREE ()

MOVE TO (B)

GRASP ()

PLACE ON (A,B)

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

FREE ()

END

**Task 2**

Python code to implement the plans in task 1

class Apalara:

row1=[0, 0, 0, 0, 0, 0, 1, 2, 3]

input=0

box1=0

box2=0

box3=0

position=0

holder1=0

holder2=0

def GameStart(self):

print self.row1[0:3]

print self.row1[3:6]

print self.row1[6:9]

while(True):

GameOption(Apalara)

if(self.input==5):

SystemExit

def GameOption(self):

print "Press 1 key to Swap any 2 boxes"

print "Press 2 key to place a box under another box"

print "Press 3 key to place a box on top another box"

print "Press 4 key to exit"

self.input=input()

if(self.input==1):

GameSwap(Apalara)

elif(self.input==2):

GameUnder(Apalara)

elif(self.input==3):

GameTop(Apalara)

elif(self.input==4):

Gameend(Apalara)

def GameSwap(self):

z1=0

z2=0

print "Enter the boxes to swap"

self.box1=input()

print "and?"

self.box2=input()

if((self.box1==1)&(self.box2==2)|(self.box1==2)&(self.box2==1)):

print "Checking"

if(CheckingClear1(Apalara)&CheckingClear2(Apalara)):

i=0

print "clear"

while(True):

i=i+1

self.holder1=self.row1[i]

z1=i

if(self.row1[i]==1):

print self.holder1

break;

i=0

while(True):

i=i+1

self.holder2=self.row1[i]

z2=i

if(self.row1[i]==2):

print self.holder2

break;

self.row1[z1]=self.holder2

self.row1[z2]=self.holder1

else:

print "Not Swappabless"

print self.row1[0:3]

print self.row1[3:6]

print self.row1[6:9]

elif((self.box1==1)&(self.box2==3)|(self.box1==3)&(self.box2==1)):

if(CheckingClear1(Apalara)&CheckingClear3(Apalara)):

i=0

print "clear"

while(True):

i=i+1

self.holder1=self.row1[i]

z1=i

if(self.row1[i]==1):

print self.holder1

break;

i=0

while(True):

i=i+1

self.holder2=self.row1[i]

z2=i

if(self.row1[i]==3):

print self.holder2

break;

self.row1[z1]=self.holder2

self.row1[z2]=self.holder1

else:

print "Not Swappabless"

print self.row1[0:3]

print self.row1[3:6]

print self.row1[6:9]

elif((self.box1==2)&(self.box2==3)|(self.box1==3)&(self.box2==2)):

if(CheckingClear2(Apalara)&CheckingClear3(Apalara)):

i=0

print "clear"

while(True):

i=i+1

self.holder1=self.row1[i]

z1=i

if(self.row1[i]==2):

print self.holder1

break;

i=0

while(True):

i=i+1

self.holder2=self.row1[i]

z2=i

if(self.row1[i]==3):

print self.holder2

break;

self.row1[z1]=self.holder2

self.row1[z2]=self.holder1

else:

print "Not Swappabless"

print self.row1[0:3]

print self.row1[3:6]

print self.row1[6:9]

def CheckingClear1(self):

i=0

while(True):

i=i+1

self.position=i

if(self.row1[i]==1):

break

if(self.position>3):

self.position=self.position-3

if(self.row1[self.position]>=1):

return False

else:

return True

else:

return True

def CheckingClear2(self):

i=0

while(True):

i=i+1

self.position=i

if(self.row1[i]==2):

break

if(self.position>3):

self.position=self.position-3

if(self.row1[self.position]>=1):

return False

else:

return True

else:

return True

def CheckingClear3(self):

i=0

while(True):

i=i+1

self.position=i

if(self.row1[i]==3):

break

if(self.position>3):

self.position=self.position-3

if(self.row1[self.position]>=1):

return False

else:

return True

else:

return True

def GameUnder(self):

z1=0

z2=0

print "Enter the boxes to place under the other"

self.box1=input()

print "under?"

self.box2=input()

if((self.box1==1)&(self.box2==2)):

if(CheckingClear1(Apalara)):

i=0

while(True):

i=i+1

z1=i

self.holder1=self.row1[i]

if(self.row1[i]==2):

break;

if(CheckingClear2(Apalara)):

print "clear"

else:

i=0

while(True):

i=i+1

z2=i

self.holder2=self.row1[i]

if(self.row1[i]==3):

self.row1[z2-3]=self.holder2

break;

self.row1[z1-3]=self.holder1

i=0

while(True):

i=i+1

z2=i

self.holder1=self.row1[i]

if(self.row1[i]==1):

self.row1[z1]=1

self.row1[z2]=0

break;

else:

print "Not Movable"

print self.row1[0:3]

print self.row1[3:6]

print self.row1[6:9]

elif((self.box1==2)&(self.box2==1)):

if(CheckingClear2(Apalara)):

i=0

while(True):

i=i+1

z1=i

self.holder1=self.row1[i]

if(self.row1[i]==1):

z1=z1-3

break;

if(CheckingClear1(Apalara)):

print "clear"

else:

i=0

while(True):

i=i+1

z2=i

self.holder2=self.row1[i]

if(self.row1[i]==3):

z2=z2-3

self.row1[z2]=self.holder2

break;

self.row1[z1]=self.holder1

i=0

while(True):

i=i+1

z2=i

self.holder1=self.row1[i]

if(self.row1[i]==2):

self.row1[z1+3]=2

self.row1[z2]=0

break;

else:

print "Not Movable"

print self.row1[0:3]

print self.row1[3:6]

print self.row1[6:9]

elif((self.box1==1)&(self.box2==3)):

if(CheckingClear1(Apalara)):

i=0

while(True):

i=i+1

z1=i

self.holder1=self.row1[i]

if(self.row1[i]==3):

z1=z1-3

break;

if(CheckingClear3(Apalara)):

print "clear"

else:

i=0

while(True):

i=i+1

z2=i

self.holder2=self.row1[i]

if(self.row1[i]==2):

z2=z2-3

self.row1[z2]=self.holder2

break;

self.row1[z1]=self.holder1

i=0

while(True):

i=i+1

z2=i

self.holder1=self.row1[i]

if(self.row1[i]==1):

self.row1[z1+3]=1

self.row1[z2]=0

break;

else:

print "Not Movable"

print self.row1[0:3]

print self.row1[3:6]

print self.row1[6:9]

elif((self.box1==3)&(self.box2==1)):

if(CheckingClear3(Apalara)):

i=0

while(True):

i=i+1

z1=i

self.holder1=self.row1[i]

if(self.row1[i]==1):

z1=z1-3

break;

if(CheckingClear1(Apalara)):

print "clear"

else:

i=0

while(True):

i=i+1

z2=i

self.holder2=self.row1[i]

if(self.row1[i]==2):

z2=z2-3

self.row1[z2]=self.holder2

break;

self.row1[z1]=self.holder1

i=0

while(True):

i=i+1

z2=i

self.holder1=self.row1[i]

if(self.row1[i]==3):

self.row1[z1+3]=3

self.row1[z2]=0

break;

else:

print "Not Movable"

print self.row1[0:3]

print self.row1[3:6]

print self.row1[6:9]

elif((self.box1==2)&(self.box2==3)):

if(CheckingClear2(Apalara)):

i=0

while(True):

i=i+1

z1=i

self.holder1=self.row1[i]

if(self.row1[i]==3):

z1=z1-3

break;

if(CheckingClear1(Apalara)):

print "clear"

else:

i=0

while(True):

i=i+1

z2=i

self.holder2=self.row1[i]

if(self.row1[i]==1):

z2=z2-3

self.row1[z2]=self.holder2

break;

self.row1[z1]=self.holder1

i=0

while(True):

i=i+1

z2=i

self.holder1=self.row1[i]

if(self.row1[i]==2):

self.row1[z1+3]=2

self.row1[z2]=0

break;

else:

print "Not Movable"

print self.row1[0:3]

print self.row1[3:6]

print self.row1[6:9]

elif((self.box1==3)&(self.box2==2)):

if(CheckingClear3(Apalara)):

i=0

while(True):

i=i+1

z1=i

self.holder1=self.row1[i]

if(self.row1[i]==2):

z1=z1-3

break;

if(CheckingClear1(Apalara)):

print "clear"

else:

i=0

while(True):

i=i+1

z2=i

self.holder2=self.row1[i]

if(self.row1[i]==1):

z2=z2-3

self.row1[z2]=self.holder2

break;

self.row1[z1]=self.holder1

i=0

while(True):

i=i+1

z2=i

self.holder1=self.row1[i]

if(self.row1[i]==3):

self.row1[z1+3]=3

self.row1[z2]=0

break;

else:

print "Not Movable"

print self.row1[0:3]

print self.row1[3:6]

print self.row1[6:9]

def GameTop(self):

l=0

z1=0

print "Enter the boxes to place ONTOP the other"

self.box1=input()

print "ON TOP?"

self.box2=input()

if((self.box1==1)&(self.box2==2)):

if((CheckingClear1(Apalara))&(CheckingClear2(Apalara))):

i=0

while(True):

k=0

z1=i

if(self.row1[i]==2):

self.row1[z1-3]=1

k=z1-3

if(i<8):

i=i+1

if(self.row1[i]==1):

l=i

if((self.row1[l]==1)&(self.row1[k]==1)):

print "debug"

self.row1[l]=0

break

if(i<8):

i=i+1

else:

print "Not Movable"

print self.row1[0:3]

print self.row1[3:6]

print self.row1[6:9]

elif((self.box1==2)&(self.box2==1)):

if((CheckingClear1(Apalara))&(CheckingClear2(Apalara))):

i=0

while(True):

k=0

z1=i

if(self.row1[i]==1):

print "printing"

self.row1[z1-3]=2

k=z1-3

if(i<8):

i=i+1

if(self.row1[i]==2):

print "printing1"

l=i

if((self.row1[l]==2)&(self.row1[k]==2)):

self.row1[l]=0

print "printing2"

break

if(i<8):

i=i+1

else:

print "Not Movable"

print self.row1[0:3]

print self.row1[3:6]

print self.row1[6:9]

elif((self.box1==1)&(self.box2==3)):

if((CheckingClear1(Apalara))&(CheckingClear3(Apalara))):

i=0

while(True):

k=0

z1=i

if(self.row1[i]==3):

self.row1[z1-3]=1

k=z1-3

if(i<8):

i=i+1

if(self.row1[i]==1):

l=i

if((self.row1[l]==1)&(self.row1[k]==1)):

self.row1[l]=0

break

if(i<8):

i=i+1

else:

print "Not Movable"

print self.row1[0:3]

print self.row1[3:6]

print self.row1[6:9]

elif((self.box1==3)&(self.box2==1)):

if((CheckingClear1(Apalara))&(CheckingClear3(Apalara))):

i=0

while(True):

k=0

z1=i

if(self.row1[i]==1):

self.row1[z1-3]=3

print i

k=z1-3

if(i<8):

i=i+1

if(self.row1[i]==3):

l=i

if((self.row1[i]==3)&(self.row1[k]==3)):

print "see3"

self.row1[l]=0

break

if(i<8):

i=i+1

print self.row1[i]

else:

print "Not Movable"

print self.row1[0:3]

print self.row1[3:6]

print self.row1[6:9]

elif((self.box1==2)&(self.box2==3)):

if((CheckingClear2(Apalara))&(CheckingClear3(Apalara))):

i=0

while(True):

k=0

z1=i

if(self.row1[i]==3):

self.row1[z1-3]=2

k=z1-3

if(i<8):

i=i+1

if(self.row1[i]==2):

l=i

if((self.row1[l]==2)&(self.row1[k]==2)):

self.row1[l]=0

break

if(i<8):

i=i+1

else:

print "Not Movable"

print self.row1[0:3]

print self.row1[3:6]

print self.row1[6:9]

elif((self.box1==3)&(self.box2==2)):

if((CheckingClear3(Apalara))&(CheckingClear2(Apalara))):

i=0

while(True):

k=0

z1=i

if(self.row1[i]==2):

self.row1[z1-3]=3

k=z1-3

if(i<8):

i=i+1

if(self.row1[i]==3):

l=i

if((self.row1[l]==3)&(self.row1[k]==3)):

self.row1[l]=0

break

if(i<8):

i=i+1

else:

print "Not Movable"

print self.row1[0:3]

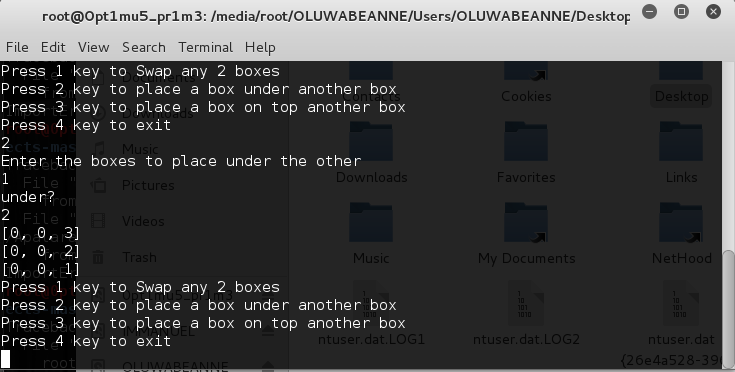
print self.row1[3:6]

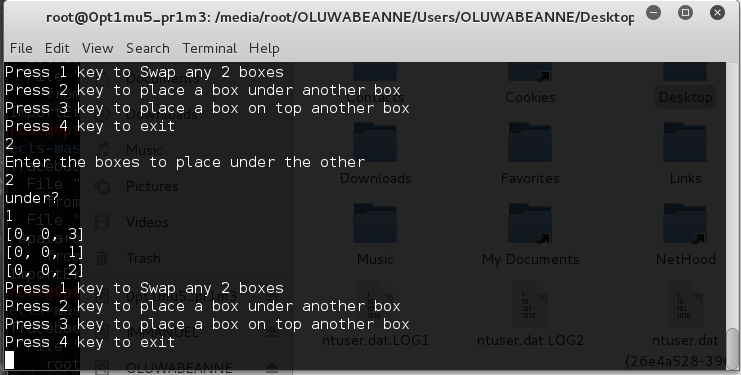
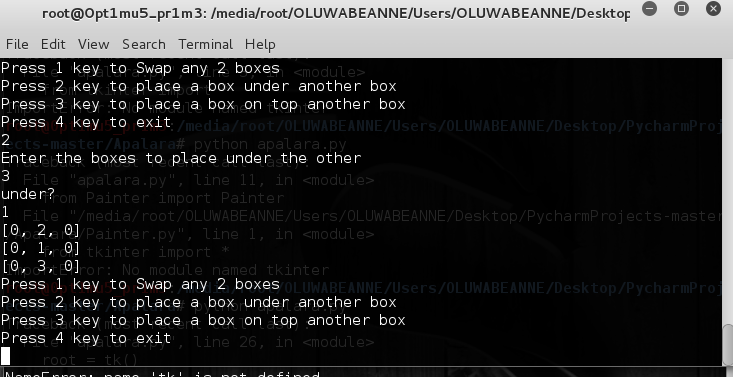
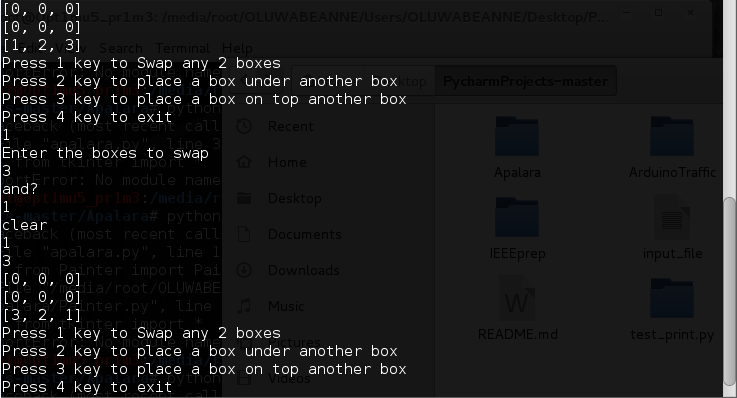
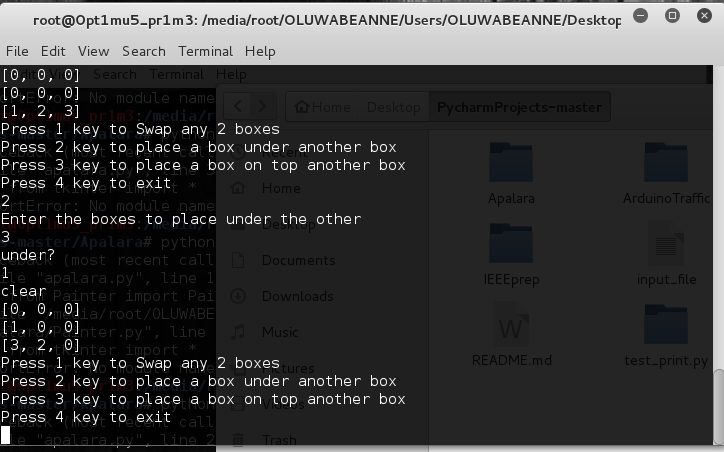
print self.row1[6:9]

def Gameend(self):

exit()

GameStart(Apalara)

**TEST OF SYSTEM **

****

**LIMITATION OF THE SYSTEM**

1. The maximum amount of boxes allowed for this implementation is just 3
2. Lack of graphics to fully show the activity of the ROBOT ARM

LANGUAGE OF ROBOT PLAN

The robot plans can be represented using First Order Predicate Calculus (FOPC). The FOPC is a system of logic in which it is possible to formally express much of the statements of everyday language. The FOPC state-space is closed. This implies that when its logics and relations are applied on its objects and terms, they (i.e. the objects and terms) retain their features. The space may change state but its original feature remains unchanged. This way, the state space cannot be extended or contrasted, but can be made to transit from one configuration, or form, to another.

In the FOPC system, there is no anticipation of the possibility of latter revision of the state space. A term in the system can assume only one of the set of two values: True (Accepted) or False (Rejected). Every situation is wholly false or wholly true: there is nothing in between. In the FOPC, an expression can either be true or false not partly true and partly false.

This formal system can be used to model the state space in a primitive dialogue for a robot arm domain. The robot world state space is specified by a set of propositions and connectives (or operators). The proposition and connectives expresses truth relations among objects in a configuration of the robot world. There are also robot action propositions which express the trustfulness of the robot arm actions. The objects in the robot world, i.e. the boxes, can be abstractly manipulated using the robot action propositions. If an action can be (or had been) carried out successfully then, the proposition will return True. It will return False otherwise.

predicate calculus = proposition logic + universal quantifiers

**TECHNOLOGICAL INTERPRETATION AND APPLICATION OF RESULTS**

The ROBOT ARM is designed on python platform which gives the design view of the real life system to be designed based on Artificial Intelligent system.

The knowledge gotten from the ROBOT ARM is applicable in real life situations e.g. THE MASS PRODUCTION LINE,SEA PORT

* + The mass production line includes d moving of parts of production like, part of cars the arm is used to fix the parts together, and to move them off the line
  + In sea port to move containers around and place them in good positions



* + The caterpillar arms is a robot arm

**SUMMARY AND CONCLUSION**

This experiment is based on an analogy of a ROBOT arm, and the simulation and manipulation of boxes A, B, C represented by 1, 2, 3, respectively. We have been able to determine the ROBOT ARM movement and also implement the game with a programming language (PYTHON).

We have been able to evolve a ROBOT ARM that can emulate the crane in sea port used in carrying containers in order to place them as wanted on top or under each other, it expands our understanding of human intelligent processes of arrangement and carrying through computer simulation. The ROBOT is computationally effective and can improve AI performance and make computer programmed robot more adaptable and responsive.