PART A   Student Name：Haoyu He

1.

Agent Type	Performance Measure	Environment	Actuators	Sensors
Playing soccer	Score	Field, teammates, opponents, ball	Jointed feet and leg	Camera,force sensors
Shopping for used AI books on the Internet	Finding used AI books,reduce cost,condition	Internet	Screen, http request	Web browser
Practicing tennis against a wall	Round,	Court, wall	Racket, jointed arm	Range sensors, force sensors
Performing a high jump	Height	Measuring stick	legs	Balancing sensors

2.

agent: Anything that can react to environment.
agent function: A corresponding table of actions and percept sequence.
agent program: Implementation of agent function.
rationality: Agents aim to enhance performance measure.
autonomy: Agents can update its prior knowledge automatically.
reflex agent: Agents select actions without percept history.
model-based agent: Agents act with their percepts and knowledge of real world.
goal-based agent: Agents act with goal information.
utility-based agent: Agents measure their actions by how their actions can contribute to performance measure.
learning agent: Agents improve overall performance by modifying their own components..

3.

Goal-based agents:

function GOAL-BASED-REFLEX-AGENT(percept) returns an action
    persistent:
        state, the agent’s current conception of the world state
        model, a description of the next state depends on current state and action

goals, a set of goals the agent needs to accomplish
action, the most recent action
state <- UPDATE-STATE(state,action,percept,model)
action <- BEST-ACTION(goals,state)
return action

Utility-based agents:

```
function UTILITY-BASED-AGENT(percept) returns an action
    persistent:
        state, the agent's current conception of the world state
        model, a description of the next state depends on current state and action
        action, the most recent action
state <- UPDATE-STATE(state,action,percept,model)
score <- SCORE-STATE(state)
action <- BEST-ACTION(score)
```

4.

3.6(a):
States: Any region colored by one of four colors is a state.
Initial State: No regions colored.
Actions: Color a region with one of four colors.
Transition model: Color a region then update this map.
Goal test: All regions are colored and no two adjacent regions have the same color.
Path cost: Number of assignments.
3.6(b):
States: Monkey's positions
Initial State: The monkey is in a room and bananas are suspended from the 8-foot ceiling.
Actions: Push a crate, jump onto crate, jump off crate.
Transition model: Monkey do an action and result in a new place.
Goal test: Monkey gets the bananas.
Path cost: Number of actions.

5.

state: A condition that agent can be in.
state space: A set of all possible states.
search tree: A tree in which the nodes represent reachable states.
search node: A node in the search tree.
goal: A goal is a state where agents do not need to do further actions.
action: Something agents can choose to do.
transition model: Description of what actions do.
branching factor: Actions agent can do when it is in one state.

6.

a.
A 9*9 square which has elements from 1-9 and 0 for empty. Only valid squares are allowed(no duplication in rows, column or 3*3 squares)
b.

```
function SUCCESSOR(current_state) returns successor_list:,
    successor_list = []
    for each i in [1,2,3,4,5,6,7,8,9]
        for each j in [1,2,3,4,5,6,7,8,9]
            if S[i,j] == 0
                A<-current_state
                for each k in [1,2,3,4,5,6,7,8,9]
                    A[i,j] = k
                    if(VALID (A))
                        successor_list.INSERT(A)
    function VALID(state) returns True or False
        for each i in [1,2,3,4,5,6,7,8,9]
            if duplicated(state[i,:])
                return False
        for each i in [1,2,3,4,5,6,7,8,9]
            if duplicated(state[:,i])
                return False
        for each i in [1,2,3,4,5,6,7,8,9]
            for each j in [1,2,3,4,5,6,7,8,9]

                if(duplicated(state[i:i+2,j:j+2]))

                    return False

        return True
```

c.

```
function GOAL(current_state) returns True or False
        for each i in [1,2,3,4,5,6,7,8,9]
            for each j in [1,2,3,4,5,6,7,8,9]
                if current_state[i,j] == 0
                    return False
        return True
```

d.
L=9*9-28=53

7.

BFS: 0->1,4->2,3,5,6
DFS: 0->1->2->3->4->5->6
Python implementation:
graph={'0':['1','4'],'1':['2','3'],'4':['5','6']}
BFS:

```
visited = {}
queue = []
result = []
def BFS(graph,root):
    visited[root]=1
    queue.append(root)
    while queue:
        s= queue.pop(0)
        result.append(s)
        if graph.get(s) == None:
            continue
        for child in graph[s]:
            if visited.get(child) == None:
                queue.append(child)
                visited[child] = 1
    print('->'.join(result))
BFS(graph,'0')
```

Result: 0->1->4->2->3->5->6

DFS:

```
visited = {}
result = []
def DFS(graph,node):
    if visited.get(node) is not None:
        return
    result.append(node)
    if graph.get(node) == None:
        visited[node]=1
        return
    else:
        for child in graph[node]:
            DFS(graph,child)
DFS(graph,'0')
print('->'.join(result))
```

result: 0->1->2->3->4->5->6