1.

Robot Mouse Races:

Performance measure: destination, speed

Environment: maze (road, barrier)

Actuators: movement, trace path

Sensors: displacement, distance

Robothespian:

Performance measure: level of human-like behavior

Environment: any object it can contact

Actuators: response human-like behavior

Sensors: optical recognize, GPS, sound recognize and so on. Generally, needing all sensors to recognize external environment change.

2(a).

Solution by HW1_q2(a).cpp (at the bottom of document)

start at: AT

expand: AM AN AS AX IT

choose: AM

expand:

choose: AN

expand: IN ON

choose: AS

expand: IS US

choose: AX

expand: OX

choose: IT

expand: IF

goal at: IN

path = AT => AN => IN

2(b).

If we consider this problem as a graph, we can transit from s1 which is “ab” to s2 which is “12” by at least one step if ‘a’ equal to ‘1’ or ‘b’ equal to ‘2’. Hence, if we want to transit s1 to s2 with ‘a’ not equal to ‘1’ and ‘b’ not equal to ‘2’, it needs at least two steps to make it.

By above conclusion, hamming distance can be considered as a heuristic for this problem because number of unmatchable characters between strings means the minimum steps to transit between 2 strings.

2(c).

Solution by HW1_q2(c).cpp (at the bottom of document)

start at: AT

expand: AM AN AS AX IT

choose: AN

expand: IN ON

goal at: IN

AT => AN => IN

3.

Init(): $D_X = \{0, 1, ..., 9\}$, $D_Y = \{0, 1, ..., 9\}$, $D_Z = \{0, 1, ..., 9\}$

Arc_checking_Loop(time = 1; ; time++):

time = 1:

Arc(Y~X): $D_X = \{0, 1, ..., 9\}$, $D_Y = \{0, 1, 2, 3\}$, $D_Z = \{0, 1, ..., 9\}$

Arc(X~Y): $D_X = \{0, 1, 4, 9\}$, $D_Y = \{0, 1, 2, 3\}$, $D_Z = \{0, 1, ..., 9\}$

Arc(Z~X): $D_X = \{0, 1, 4, 9\}$, $D_Y = \{0, 1, 2, 3\}$, $D_Z = \{0, 1\}$

Arc(X~Z): $D_X = \{0, 1\}$, $D_Y = \{0, 1, 2, 3\}$, $D_Z = \{0, 1\}$

time = 2:

Arc(Y~X): $D_X = \{0, 1\}$, $D_Y = \{0, 1\}$, $D_Z = \{0, 1\}$

Arc(X~Y): $D_X = \{0, 1\}$, $D_Y = \{0, 1\}$, $D_Z = \{0, 1\}$

Arc(Z~X): $D_X = \{0, 1\}$, $D_Y = \{0, 1\}$, $D_Z = \{0, 1\}$

Arc(X~Z): $D_X = \{0, 1\}$, $D_Y = \{0, 1\}$, $D_Z = \{0, 1\}$

time = 3:

Arc(Y~X): $D_X = \{0, 1\}$, $D_Y = \{0, 1\}$, $D_Z = \{0, 1\}$

Exit because 3 domains are halting

$D_X = \{0, 1\}$, $D_Y = \{0, 1\}$, $D_Z = \{0, 1\}$ => (X, Y, Z) = (0, 0, 0), (1, 1, 1)

4(a).

append 3 more variable b1, b2, b3 where denotes second digit borrows b1 from first digit, third digit borrows b2 from second digit, last digit borrows b3 from third digit.

0. 100I + 10V - 200O = 10U + 10N + R

simplify from

1000F + 100I + 10V + E - 1000F – 100O – 10U – R = 100O + 10N + E

1. F, I, V, E, O, U, R, N ∈ {0, 1, ..., 9}

2. F≠0 (first digit can’t be zero)

3. F≠I≠V≠E≠O≠U≠R≠N

4. b1, b2, b3 ∈ {0, 1}

5. Digit constrain

F – b1 – F = 0 => b1 = 0

I – O + 10b1 – b2 = O => I – O – b2 = O

V – U + 10b2 – b3 = N

E + 10b3 – R = E

4(b).

arrange constrain:

R = 0, b3 = 0

consider last digit, it may be

E – R = E => R = 0

or 10 + E – R = E => R = 10 => conflict to R's domain

100I + 10V - 200O = 10U + 10N (constrain1)

F, I, V, E, O, U, N ∈ {1, 2, ..., 9} (domain)

F≠I≠V≠E≠O≠U≠N (constrain2)

b2 ∈ {0, 1} (domain)

b1 = 0

I = 2O + b2 (constrain3)

V + 10b2 = N + U (constrain4)

explicit var: R = b1 = b3 = 0

implicit var: F, I, V, E, O, U, N, b2

Var(MRV, degree): b2(2, 2), I(9,3), V(9,3), O(9,3), U(9,3), N(9,3), F(9, 1), E(9,1)

Iteration1:

choose b2 = 0;

explicit var: R = 0, b1 = b2 = b3 = 0

stack = {$,b2=1} // $ is bottom of stack

constrain1: 100I + 10V - 200O = 10U + 10N

constrain2: F≠I≠V≠E≠O≠U≠N

constrain3: I = 2O => I ∈ {2, 4, 6, 8}

constrain4: V = N + U

Var(MRV, degree): I(4,3), V(9,3), O(9,3), U(9,3), N(9,3), F(9, 1), E(9,1)

Iteration2:

choose I = 8;

explicit var: I = 8, R = 0, b1 = b2 = b3 = 0

stack = {$,b2=1,I=2,I=4,I=6} // $ is bottom of stack

constrain3: 8 = 2O => O ∈ {4} => O = 4 becomes explicit (SOLVE)

constrain1: 800 + 10V - 800 = 10U + 10N => 10V = 10U + 10N

=> V = N + U => constrain4 (SOLVE)

constrain2: F≠V≠E≠U≠N≠4, 8 => F, V, E, U, N ∈ {1, 2, ..., 9} - {4, 8}

constrain4: V = N + U

Var(MRV, degree): V(7,2), U(7,2), N(7,2), F(7, 1), E(7,1)

Iteration3:

choose V = 9;

explicit var: I = 8, V = 9, O = 4, R = 0, b1 = b2 = b3 = 0

stack = {$,b2=1,I=2,I=4,I=6,V=1,V=2,V=3,V=5,V=6,V=7}

constrain4: 9 = N + U => N, U ∈ {2 ,3, 6, 7}

constrain2: F≠E≠U≠N≠4, 8, 9 => F, E, U, N ∈ {1, 2, 3, 5, 6, 7}

Var(MRV, degree): U(4,2), N(4,2), F(6, 1), E(6,1)

Iteration4:

choose U = 7;

explicit var: I = 8, V = 9, O = 4, U = 7, R = 0, b1 = b2 = b3 = 0

stack = {$,b2=1,I=2,I=4,I=6,V=1,V=2,V=3,V=5,V=6,V=7,U=2,U=3,U=6}

constrain4: 2 = N => N ∈ {2} => N = 2 becomes explicit (SOLVE)

constrain2: F≠E≠2, 4, 7, 8, 9 => F, E ∈ {1, 3, 5, 6}

Var(MRV, degree): F(4, 1), E(4,1)

Iteration4:

choose F = 6;

explicit var: F = 6, I = 8, V = 9, O = 4, U = 7, R = 0, N = 2, b1 = b2 = b3 = 0

stack = {$,b2=1,I=2,I=4,I=6,V=1,V=2,V=3,V=5,V=6,V=7,U=2,U=3,U=6,F=1,F=3,F=5}

constrain2: E≠2, 3, 4, 7, 8, 9 => E ∈ {1, 3, 5}

Var(MRV, degree): E(3,1)

Iteration5:

choose E = 5;

explicit var: F = 6, I = 8, V = 9, E = 5, O = 4, U = 7, R = 0, N = 2, b1 = b2 = b3 = 0

stack = {$,b2=1,I=2,I=4,I=6,V=1,V=2,V=3,V=5,V=6,V=7,U=2,U=3,U=6,F=1,F=3,F=5,E=1,E=5}

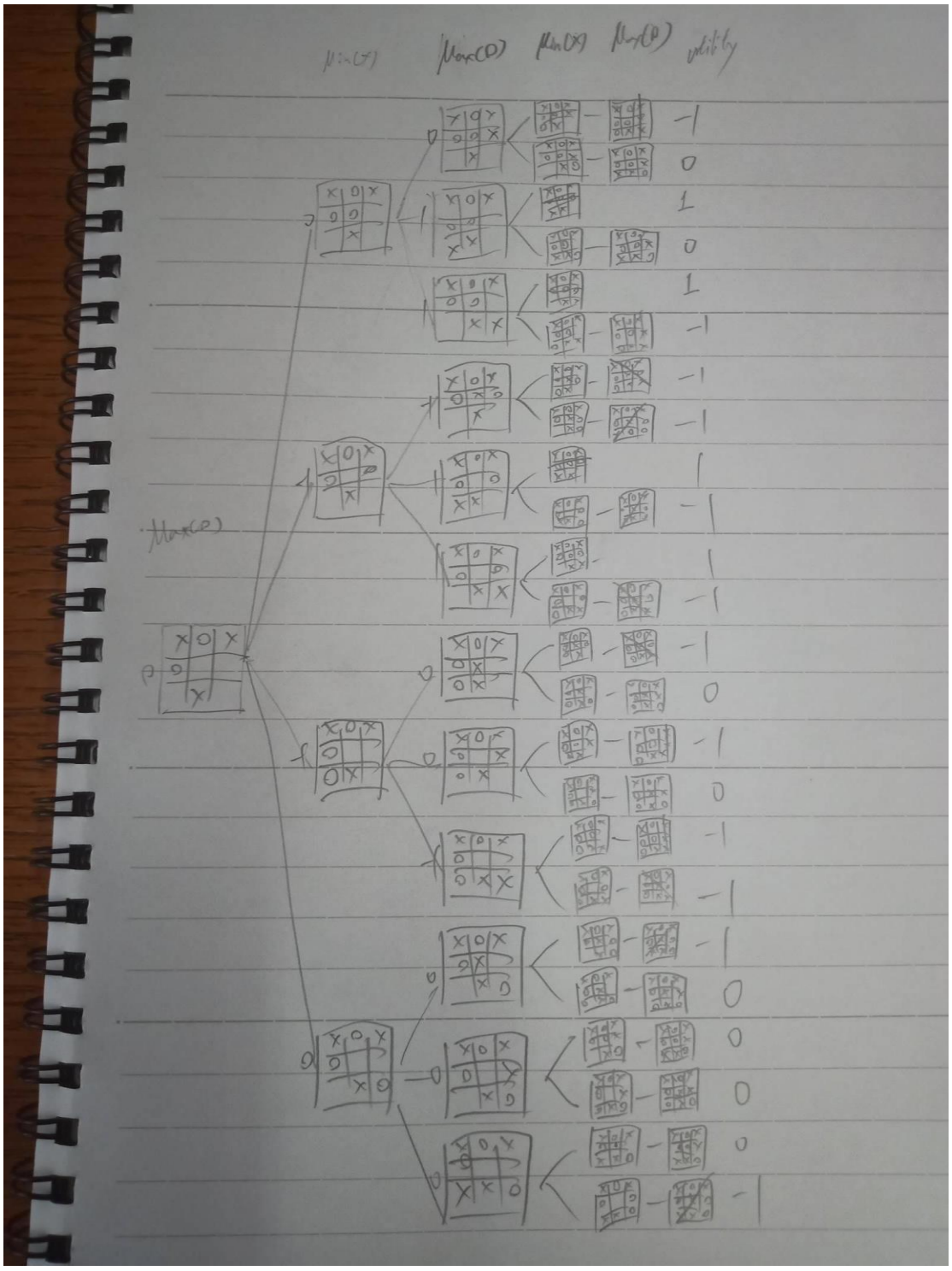
constrain2: (EMPTY) => (SOLVE)

(No unsolved constrain)

Solution: F = 6, I = 8, V = 9, E = 5, O = 4, U = 7, R = 0, N = 2, b1 = b2 = b3 = 0

This solution denotes to 6895 – 6470 = 425

5(a).



5(b).

9 nodes, first expansion will return 0, every value smaller or equal to 0 will drop all expansion.

6(a).

A

B

P => Q: ¬P ∨ Q

L ∧ M => P: ¬L ∨ ¬M ∨ P
L ∧ B => M: ¬L ∨ ¬B ∨ M
A ∧ B => L: ¬A ∨ ¬B ∨ L
A ∧ P => L: ¬A ∨ ¬P ∨ L

6(b).

(1) A
(2) B
(3) ¬P ∨ Q
(4) ¬L ∨ ¬M ∨ P
(5) ¬L ∨ ¬B ∨ M
(6) ¬A ∨ ¬B ∨ L
(7) ¬A ∨ ¬P ∨ L
(8) ¬Q // complement of proof target
(9) L by (1)(2)(6)
(10) M by (2)(5)(9)
(11) P by (4)(9)(10)
(12) Q by (3)(11)
(13) False by (8)(12)

Get contradiction, which means Q can't be False.
We can say Q is True by this proof.

Appendix:

HW1_q2(a).cpp:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <queue>

using namespace std;

class Frontier{
public:
	int pos,pos_p;
	Frontier(int p,int p_p){
		pos = p;
		pos_p = p_p;
	}
};
int main(int argc,char const *argv[]){
```

```
	string sa[] =
{"AN","AM","AS","AT","AX","BE","BY","GO","HE","HI","IT","IS","IN","IF","ME
","MY","NO","OF","OH","OK","ON","OR","OX","SO","TO","UP","US","WE"};
	vector<string> v(sa,sa + sizeof(sa) / sizeof(string));
	sort(v.begin(), v.end());
	vector<int> parent(v.size(),-1);
	vector<bool> inq(v.size(),false);
	int start,goal;
	for(int i = 0;i < v.size();i++){
		if(v[i] == "AT")
			start = i;
		else if(v[i] == "IN")
			goal = i;
	}
	queue<Frontier> q;
	q.push(Frontier(start,9999)); // no parent
	inq[start] = true;
	while(!q.empty()){
		Frontier node = q.front();
		q.pop();
		parent[node.pos] = node.pos_p;
		string s = v[node.pos];
		cout << "choose: " << s << '\n';
		if(s == v[goal])
			break;
		cout << "expand: ";
		for(int i = 0;i < v.size();i++)
			if(!inq[i] && (s[0] == v[i][0] || s[1] == v[i][1])){
				inq[i] = true;
				cout << v[i] << ' ';
				q.push(Frontier(i,node.pos));
			}
		cout << '\n';
	}
	vector<int> path;
	int pos = goal;
	while(true){
		if(pos == 9999)
```

```
                break;
            path.push_back(pos);
            pos = parent[pos];
        }
        cout << v[path[path.size() - 1]];
        for(int i = path.size() - 2;i >= 0;i--)
            cout << " => " << v[path[i]];
        cout << '\n';
        return 0;
}
```

HW1_q2(c).cpp:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <queue>

using namespace std;

int h(string s1,string s2){ // heuristic function
        int n = s1.size();
        for(int i = 0;i < s1.length();i++)
            if(s1[i] == s2[i])
                n--;
        return n;
}
class Frontier{
public:
        int pos,pos_p,fval,gval;
        Frontier(int p,int p_p,int f){
            pos = p;
            pos_p = p_p;
            fval = f;
        }
        friend bool operator>(const Frontier & obj1,const Frontier & obj2){
            return obj1.fval > obj2.fval;
        }
};
int main(int argc,char const *argv[]){
```

```
	string sa[] =
{"AN","AM","AS","AT","AX","BE","BY","GO","HE","HI","IT","IS","IN","IF","ME
","MY","NO","OF","OH","OK","ON","OR","OX","SO","TO","UP","US","WE"};
	vector<string> v(sa,sa + sizeof(sa) / sizeof(string));
	sort(v.begin(), v.end());
	vector<int> parent(v.size(),-1);
	vector<bool> inq(v.size(),false);
	int start,goal;
	for(int i = 0;i < v.size();i++){
		if(v[i] == "AT")
			start = i;
		else if(v[i] == "IN")
			goal = i;
	}
	priority_queue<Frontier,vector<Frontier>,greater<Frontier> > q;
	q.push(Frontier(start,9999,h(v[start],v[goal]))); // no parent
	inq[start] = true;
	while(!q.empty()){
		Frontier node = q.top();
		q.pop();
		parent[node.pos] = node.pos_p;
		string s = v[node.pos];
		cout << "choose: " << s << '\n';
		if(s == v[goal])
			break;
		cout << "expand: ";
		for(int i = 0;i < v.size();i++)
			if(!inq[i] && (s[0] == v[i][0] || s[1] == v[i][1])){
				inq[i] = true;
				cout << v[i] << ' ';
				q.push(Frontier(i,node.pos,h(v[i],v[goal])));
			}
		cout << '\n';
	}
	vector<int> path;
	int pos = goal;
	while(true){
		if(pos == 9999)
```

```
                break;
            path.push_back(pos);
            pos = parent[pos];
        }
        cout << v[path[path.size() - 1]];
        for(int i = path.size() - 2;i >= 0;i--)
            cout << " => " << v[path[i]];
        cout << '\n';
        return 0;
}
```