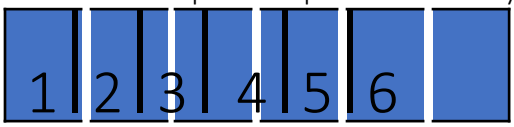
The University of Texas at Dallas
CS 6364
Artificial Intelligence
Fall 2020
Instructor: Dr. Sanda Harabagiu
Grader/ Teaching Assistant: Maxwell Weinzierl

Homework 1: 100 points (70 points extra-credit)
Issued September 2, 2020
Due September 21, 2020 before midnight

Name _____KAPIL GAUTAM_____
Student ID _____KXG180032_____
CS 6364

PROBLEM 1: Intelligent Agents (15 points)

Enhance the Kitty robot discussed in class to a model-based agent in which the state has 3 components: [position, happiness, battery]; where the battery has a value of 100 for FULL initially and 0 when it is discharged. Also add an admissible new action: TURN-AROUND. The possible positions of Kitty are:



You are asked to:

1. Describe the entire state space (2 points)
2. Write a new State-Update function that changes the value of the battery by -1 for every new change of position by walking; by -4 when it turns around; by -5 when it bumps against the wall; by -3 every time Kitty purrs and by -2 every time it meows You should consider that when the battery is discharged, no more actions are possible. (10 points)
3. Write a utility function that combines the happiness with the state of the battery. (3 points)

SOLUTION 1:

1. Kitty state space
   State: {position, happiness, battery status}

Kitty robot Battery status: {0 to 100}
Current: 100
Happiness of kitty: {Very Happy (VH), Happy(H), Sad(S), Curious(C)}
Current: Curious
Positions of kitty: {1, 2, 3, 4, 5, 6}
Current: 6

2. State-update

Action	Battery change
No action if battery discharged	Not Applicable
Bump against wall	-5
Turn around	-4
Kitty purr	-3
Kitty meow	-2
Change of position by walking	-1

Update-Kitty [State, Action, Percept]

- {Position, Happiness+1, Battery>2 && Battery-2} **<-** UPDATE-KITTY [previous state, Meow, [Clap, Clap]]
- {Position, Happiness+2, Battery>3 && Battery-3} **<-** UPDATE-KITTY [previous state, Purr, [Pet]]
- {Position, Happiness, Battery>4 && Battery-4} **<-** {Position, Happiness-1, Battery>5 && Battery-5} **<-** UPDATE-KITTY [previous state, Turn Around, [Bump]]
- {Position-1, Happiness, Battery>1 && Battery-1} **<-** UPDATE-KITTY [previous state, Walk, [Clap]]

3. Utility function: combining happiness and battery status

```
function UTILITY-BASED-AGENT (percept)
   returns an action
 static: state, a description of the current world state
           action, the most recent action, initially none
           model, a description of how the next state depends on current state and action
         utilities, a set of weighted outcomes for maximizing performance

state ← UPDATE-STATE (state, action, percept, model)
utility ← UTILITY-FUNCTION (state, utilities)
action ← UTILITY-ACTION [utility]
return action
```

Utilities:

State - Happiness	State – Battery status	Utility measure (maximum = 10)	Action
Happiness +2	battery >= 90%	Utility = 8	Walk
Happiness +1	battery < 90% and battery >= 50%	Utility = 6	Turn Around
Happiness	battery < 50% and battery >= 20%	Utility = 4	Meow, Walk
Happiness -1	battery < 20% and battery >= 5%	Utility = 8	Meow, Purr
Happiness -2	battery < 5%	Utility = 10	Purr, Blink, Stop

PROBLEM 2: Problem Formulation for Search (55 points)

The Missionaries and Cannibals Problem is stated as follows. Three missionaries and three cannibals are on one side of the river, along with a boat that can hold one or two people.

Find a way to get everyone to the other side without ever leaving a group of missionaries outnumbered by cannibals in that place.

a. Formulate the search problem formally. State clearly how you represent states, (including the initial state and the goal state(s)), what actions are available in each state, including the cost of each action. (15 points)
b. Sketch a solution of the problem. Show the path and its cost. Describe the search strategy that you have used and motivate it. (15 points for the manual solution; 5 points for its motivation). TOTAL 20 points

Programming Assignment for Problem 2:

1. Use the implementations of the search strategies available in the aima code; e.g. https://github.com/aimacode/aima-java to write the program that will search for the solution to Missionaries and Cannibals Problem. (10 points)

2. Provide the path to the solution generated by your code, indicating each state it has visited when using: (a) uniform-cost search; (b) iterative deepening search; (c) greedy best-first search; (d) A* search and (e) recursive best-first search (1 point/strategy). Describe clearly how you have implemented each state in the search space. (5 points)

TOTAL: 10 points

Extra-credit: List the content of the frontier and the list of explored nodes for the first 5 steps of each of the strategies used in 2. (2 points/step/strategy)

TOTAL: 50 points ONLY if it is possible to comprehend (1) the current node; (2) the content of the frontier; (3) the content of the list of explored/expanded nodes; (4) the children of the current node for each step.

SOLUTION 2:

a. Missionaries and Cannibals Problem formulation:

# of cannibals	3
# of missionaries	3
Side of river	left, right

State Representation: {
# of missionaries on left-side,
# of cannibals on left-side,
Side of river where the boat is
}

Initial State: { 3, 3, left }

Goal State: { 0, 0, right}

Actions:

CC: Two cannibals crosses the river on boat
MM: Two missionaries crosses the river on boat
MC: One cannibal and one missionary cross the river on a boat
M: Only one missionary crosses the river on a boat
C: Only one cannibal crosses the river on a boat

Constraints:

1. No side can have a greater number of cannibals than missionaries.

Path Cost: The boat has to move with one or two people, so each move costs one. (There is no mention if the fuel efficiency changes with different number of people.)

b. Motivation: The missionaries and cannibals problem focus how through a structured search strategy we can solve complicated problems. There can be several search strategies like uniform cost search, iterative deepening search, greedy best-first search, or recursive best-first search. A structured representation of the state space and how we move from one state to other is the core of this problem. As given in the problem, initially we have 3 cannibals and 3 missionaries on one side of the river, let's assume, left. Now they all need to cross the river with a boat and arrive at the right-side. During each state we need to be sure that at any side the missionaries are not outnumbered by the cannibals.

Path:
We represent cannibals by 'C', missionaries by 'M' and side of river by either 'L' or 'R'. Initially 3C and 3M are a L side of river.

1. 1M and 1C decide to cross the river. Now the boat is at right-side, and there are 2M and 2C left. Safe state for both side of river.

2. 1M at the right-side of river moves back with the boat to left-side. Now there are 3M and 2C left. Still safe state.

3. Now we cannot send 2M, or 1M 1C on the right as that will be an unsafe situation. So, we send 2C on the right-side of the river. This leaves all the missionaries on the left-side, and all cannibals on the right-side of the river.

4. We bring back 1C from the boat at right to left-side. Now there are 3 M and 1C on the left-side, which is a safe state.

5. With the boat at the left-side, we take 2M on the boat for crossing the river from left to right. There are 1M 1C on the left-side, and 2M 2C on the right-side of the river making it a safe state.

6. The boat is now at the right-side of the river. To maintain the safe state, we have to move back with 1M 1C, to make the left-side total to be 2M 2C and 1M 1C on the right-side.

7. Easily now we move 2M on the boat from the left-side to the right-side. So now only 2 cannibals are left at the left-side of river.

8. Now we should not bring back any missionaries on the left side, and just bring back the boat to the left-side of the river by 1C. So, we have all cannibals on the left and all missionaries on the right. (Just opposite state of step 3)

9. With the only choice left, we move 2C to the right. This leaves us with only 1C on the left.

10. Now we can send 1M or 1C on the boat from right to left to maintain the safe state.

11. Remaining people now travel back to the right side, which is our goal state.

Cost: 11, since the boat had to travel 11 times to move all the cannibals and missionaries from the left-side to right-side of the river.

Search strategy used: Uniform cost search is used to arrive at a solution in the manual solution by self.

Programming Assignment for Problem 2:

1. Please check the code solution provided in search_problem.py

2. Implementation of state in search space:
   I have used Python programming to complete the solution. In order to represent the states, I have used a tuple to represent the state of the search space as described in the manual solution. The structure of tuple is comprised of 3 tuple elements as follows:

- First element of tuple denotes the number of missionaries remaining on the left side of the river.
- Second element of the tuple denotes the number of cannibals remaining on the right side of the river.
- Third element of the tuple denotes which side of the river the boat is currently at, i.e., left or right.

Path to solution:

a. Uniform-cost search

Solution path: [(3, 3, 'left'), (2, 2, 'right'), (3, 2, 'left'), (3, 0, 'right'), (3, 1, 'left'), (1, 1, 'right'), (2, 2, 'left'), (0, 2, 'right'), (0, 3, 'left'), (0, 1, 'right'), (1, 1, 'left'), (0, 0, 'right')]

Step1	Frontier	(3, 3, 'left')
	Explored	None
	Current Node	(3, 3, 'left')
	Children of current node	(1, 3, 'right'), (2, 3, 'right'), (2, 2, 'right'), (3, 2, 'right'), (3, 1, 'right')
Step2	Frontier	(1, 3, 'right'), (2, 3, 'right'), (2, 2, 'right'), (3, 2, 'right'), (3, 1, 'right')
	Explored	(3, 3, 'left')
	Current Node	(1, 3, 'right')
	Children of current node	None
Step3	Frontier	(2, 2, 'right'), (2, 3, 'right'), (3, 1, 'right'), (3, 2, 'right')
	Explored	(1, 3, 'right'), (3, 3, 'left')
	Current Node	(2, 2, 'right')
	Children of current node	(3, 2, 'left'), (2, 3, 'left')
Step4	Frontier	(2, 3, 'right'), (3, 2, 'right'), (3, 1, 'right'), (3, 2, 'left'), (2, 3, 'left')
	Explored	(2, 2, 'right'), (1, 3, 'right'), (3, 3, 'left')
	Current Node	(2, 3, 'right')
	Children of current node	None
Step5	Frontier	(3, 1, 'right'), (3, 2, 'right'), (2, 3, 'left'), (3, 2, 'left')
	Explored	(2, 2, 'right'), (1, 3, 'right'), (2, 3, 'right'), (3, 3, 'left')
	Current Node	(3, 1, 'right')
	Children of current node	None

b. Iterative deepening search

Solution path: [(3, 3, 'left'), (2, 2, 'right'), (3, 2, 'left'), (3, 0, 'right'), (3, 1, 'left'), (1, 1, 'right'), (2, 2, 'left'), (0, 2, 'right'), (0, 3, 'left'), (0, 1, 'right'), (1, 1, 'left'), (0, 0, 'right')]

Step1	Frontier	No concept
	Explored	No concept
	Current Node	(3, 3, 'left')
	Children of current node	(1, 3, 'right'), (2, 3, 'right'), (2, 2, 'right'), (3, 2,

		'right'), (3, 1, 'right')
Step2	Frontier	No concept
	Explored	No concept
	Current Node	(1, 3, 'right')
	Children of current node	None
Step3	Frontier	No concept
	Explored	No concept
	Current Node	(2, 3, 'right')
	Children of current node	None
Step4	Frontier	No concept
	Explored	No concept
	Current Node	(2, 2, 'right')
	Children of current node	None
Step5	Frontier	No concept
	Explored	No concept
	Current Node	(3, 2, 'right')
	Children of current node	None

c. Greedy best-first search

Solution path: [(3, 3, 'left'), (2, 2, 'right'), (3, 2, 'left'), (3, 0, 'right'), (3, 1, 'left'), (1, 1, 'right'), (2, 2, 'left'), (0, 2, 'right'), (0, 3, 'left'), (0, 1, 'right'), (1, 1, 'left'), (0, 0, 'right')]

Step1	Frontier	(3, 3, 'left')
	Explored	None
	Current Node	(3, 3, 'left')
	Children of current node	(1, 3, 'right'), (2, 3, 'right'), (2, 2, 'right'), (3, 2, 'right'), (3, 1, 'right')
Step2	Frontier	(1, 3, 'right'), (2, 3, 'right'), (2, 2, 'right'), (3, 2, 'right'), (3, 1, 'right')
	Explored	(3, 3, 'left')
	Current Node	(1, 3, 'right')
	Children of current node	None
Step3	Frontier	(2, 2, 'right'), (2, 3, 'right'), (3, 1, 'right'), (3, 2, 'right')
	Explored	(1, 3, 'right'), (3, 3, 'left')
	Current Node	(2, 2, 'right')
	Children of current node	(3, 2, 'left'), (2, 3, 'left')
Step4	Frontier	(2, 3, 'right'), (3, 2, 'right'), (3, 1, 'right'), (3, 2, 'left'), (2, 3, 'left')
	Explored	(2, 2, 'right'), (1, 3, 'right'), (3, 3, 'left')
	Current Node	(2, 3, 'right')
	Children of current node	None
Step5	Frontier	(3, 1, 'right'), (3, 2, 'right'), (2, 3, 'left'), (3, 2, 'left')
	Explored	(2, 2, 'right'), (1, 3, 'right'), (2, 3, 'right'), (3, 3, 'left')
	Current Node	(3, 1, 'right')
	Children of current node	None

d. A* search

Solution path: [(3, 3, 'left'), (2, 2, 'right'), (3, 2, 'left'), (3, 0, 'right'), (3, 1, 'left'), (1, 1, 'right'), (2, 2, 'left'), (0, 2, 'right'), (0, 3, 'left'), (0, 1, 'right'), (1, 1, 'left'), (0, 0, 'right')]

Step1	Frontier	(3, 3, 'left')
	Explored	None
	Current Node	(3, 3, 'left')
	Children of current node	(1, 3, 'right'), (2, 3, 'right'), (2, 2, 'right'), (3, 2, 'right'), (3, 1, 'right')
Step2	Frontier	(1, 3, 'right'), (2, 3, 'right'), (2, 2, 'right'), (3, 2, 'right'), (3, 1, 'right')
	Explored	(3, 3, 'left')
	Current Node	(1, 3, 'right')
	Children of current node	None
Step3	Frontier	(2, 2, 'right'), (2, 3, 'right'), (3, 1, 'right'), (3, 2, 'right')
	Explored	(1, 3, 'right'), (3, 3, 'left')
	Current Node	(2, 2, 'right')
	Children of current node	(3, 2, 'left'), (2, 3, 'left')
Step4	Frontier	(2, 3, 'right'), (3, 2, 'right'), (3, 1, 'right'), (3, 2, 'left'), (2, 3, 'left')
	Explored	(2, 2, 'right'), (1, 3, 'right'), (3, 3, 'left')
	Current Node	(2, 3, 'right')
	Children of current node	None
Step5	Frontier	(3, 1, 'right'), (3, 2, 'right'), (2, 3, 'left'), (3, 2, 'left')
	Explored	(2, 2, 'right'), (1, 3, 'right'), (2, 3, 'right'), (3, 3, 'left')
	Current Node	(3, 1, 'right')
	Children of current node	None

e. Recursive best-first search

Solution path: [(3, 3, 'left'), (3, 1, 'right'), (3, 2, 'left'), (3, 0, 'right'), (3, 1, 'left'), (1, 1, 'right'), (2, 2, 'left'), (0, 2, 'right'), (0, 3, 'left'), (0, 1, 'right'), (1, 1, 'left'), (0, 0, 'right')]

Step1	Flimit	inf
	Next Node	(1, 3, 'right')
	Current Node	(3, 3, 'left')
	Children of current node	(1, 3, 'right'), (2, 3, 'right'), (2, 2, 'right'), (3, 2, 'right'), (3, 1, 'right')
Step2	Flimit	inf
	Next Node	(2, 3, 'right')
	Current Node	(3, 3, 'left')
	Children of current node	(2, 3, 'right'), (2, 2, 'right'), (3, 2, 'right'), (3, 1, 'right'), (1, 3, 'right')
Step3	Flimit	inf
	Next Node	(2, 2, 'right')
	Current Node	(3, 3, 'left')
	Children of current node	(2, 2, 'right'), (3, 2, 'right'), (3, 1, 'right'), (2, 3, 'right'), (1, 3, 'right')

Step4	Flimit	4
	Next Node	(2, 2, 'right'), (1, 3, 'right'), (3, 3, 'left')
	Current Node	(2, 2, 'right')
	Children of current node	(3, 2, 'left'), (3, 3, 'left'), (2, 3, 'left')
Step5	Flimit	inf
	Next Node	(3, 2, 'right')
	Current Node	(3, 3, 'left')
	Children of current node	(3, 2, 'right'), (3, 1, 'right'), (2, 2, 'right'), (2, 3, 'right'), (1, 3, 'right')

PROBLEM 3: Searching for Road Trips in the U.S.A. (30 points)

Considering the following table:
Table of direct/flight distances to Dallas (in miles).

Austin	182
Charlotte	929
San Francisco	1230
Los Angeles	1100
New York	1368
Chicago	800
Seattle	1670
Santa Fe	560
Bakersville	1282
Boston	1551

1/ You contemplate to search for a trip back to Dallas from Seattle, having access to a road map which should be implemented as a graph. Use the aima code to find the solution and return a simulation of the RBFS strategy by generating automatically the following five

values: (1*) f_limit*; (2) *best*; (3) *alternative*; (4) *current-city*; and (5) *next-city* for each node visited. (10 points)

2/ Find the same solution manually and specify how you have computed the following five values: (1*) f_limit*; (2) *best*; (3) *alternative*; (4) *current-city*; and (5) *next-city* for each node visited. Specify at each step the current node and the next node. (5 points)

The road graph is (in miles):

```
Los Angeles --- San Francisco ::: 383
Los Angeles --- Austin ::: 1377
Los Angeles --- Bakersville ::: 153
San Francisco --- Bakersville ::: 283
San Francisco --- Seattle ::: 807
```

```
Seattle --- Santa Fe ::: 1463
Seattle --- Chicago ::: 2064
Bakersville -- Santa Fe ::: 864
Austin --- Dallas ::: 195
Santa Fe --- Dallas ::: 640
Boston --- Austin ::: 1963
Dallas --- New York ::: 1548
Austin --- Charlotte ::: 1200
Charlotte -- New York ::: 634
New York --- Boston ::: 225
Boston --- Chicago ::: 983
Chicago -- Santa Fe ::: 1272
```

3/ Perform the same search for your optimal road trip from Seattle to Dallas when using A*. List the contents of the frontier and explored list for each node that you visit during search, in the format:
[Current node: X; Evaluation function (current node)=???;
Explored Cities: [….]; Frontier: [(City_X, f(City_X)), ….];] (10 points)
4/ Find the same solution manually, specifying:

(a) the current node; If it is not a goal node then also:
(b) the children of the current node and the value of their evaluated function (detailing how you have computed it)
(c) The current path from Seattle to the current city + the cost
(d) The Contents of the Explored Cities list
(e) The Contents of the Frontier
(f) Next node.

(5 points)

Extra-credit Write a program that will check if the heuristic provided in this problem is consistent, given the road graph TOTAL: 20 points

SOLUTION 3:

1. RBFS - Recursive best-first search - code solution:

	Step1	Step2	Step3	Step4	Step5	Step6
flimit	inf	2037	inf	2103	inf	2290
best	2023	2103	2037	2290	2103	2103
alternate	2037	None	2103	None	2290	3535
current city	Seattle	SantaFe	Seattle	SanFrancisco	Seattle	SantaFe
next city	SantaFe	None	SanFrancisco	None	SantaFe	Dallas

Solution path: Seattle -> SantaFe -> Dallas

2. RBFS manual solution:

Here, the best and alternate node value is chosen by ordering the evaluation function values of node. Each successor is evaluated by the evaluation function and the top 2

values are chosen. The top value is the best node value, and the second best is chosen as the alternate node value.

Step1:

f (SantaFe) = 2023
f (Chicago) = 2864
f (SanFransisco) = 2037

flimit: inf (Initial value)
best: 2023
alternate: 2037
current city: Seattle
next city: Santa Fe

Step2:

f (Dallas) = 2103
f (Seattle) = 4596
f (Bakersville) = 3609
f (Chicago) = 3535

flimit: 2037 (=min(2037, inf))
best: 2103

best > flimit => backtrack

alternate: None
current city: Santa Fe
next city: None

Step3:

flimit: inf (= due to backtrack)
best: 2037
alternate: 2103
current city: Seattle
next city: San Francisco

Step4:

f (Bakersville) = 2372
f (Seattle) = 3284
f (LosAngeles) = 2290
f (Boston) = 5453

flimit: 2103 (=min(2103, inf0)
best: 2290

best > flimit => backtrack

alternate: None

current city: San Francisco
next city: None

Step5:

flimit: inf (= due to backtrack)
best: 2103
alternate: 2290
current city: Seattle
next city: Santa Fe

Step6:

f (Dallas) = 2103
f (Seattle) = 4596
f (Bakersville) = 3609
f (Chicago) = 3535

flimit: 2290 (=min(2290, inf))
best: 2103
alternate: 3535
current city: Santa Fe
next city: Dallas

3. A* code solution:

Step0:
[Current node: Seattle; Evaluation function (current node)=1670;
Explored Cities: []; Frontier: [(1670, Seattle)]

Step1:
[Current node: SantaFe; Evaluation function (current node)=2023;
Explored Cities: ['Seattle']; Frontier: [(2023, SantaFe), (2864, Chicago), (2037, SanFrancisco)]

Step2:
[Current node: SanFrancisco; Evaluation function (current node)=2037;
Explored Cities: ['SantaFe', 'Seattle']; Frontier: [(2037, SanFrancisco), (2864, Chicago), (2103, Dallas), (3609, Bakersville)]

Step3:
[Current node: Dallas; Evaluation function (current node)=2103;
Explored Cities: ['SantaFe', 'SanFrancisco', 'Seattle']; Frontier: [(2103, Dallas), (2290, LosAngeles), (2372, Bakersville), (2864, Chicago), (5453, Boston)]

Solution path: Seattle -> SantaFe -> Dallas

4. A* manual solution:

Step0:

Current node: Seattle

Successors(Seattle) = {San Francisco, Santa Fe, Chicago}
f(Santa Fe) = 1463+560 = 2023,
f(Chicago) = 2064+800 = 2864,
f(San Francisco) = 807+1230=2037

Path: Seattle, Cost: 1670
Explored: None
Frontier: (1670, Seattle)
Next Node: Santa Fe

Step1:

Current node: Santa Fe
Children of current node: [(2037, SanFrancisco), (2864, Chicago), (2103, Dallas), (3609, Bakersville)]
Path: Seattle -> Santa Fe, Cost: 2023
Explored: Seattle
Frontier: [(2023, SantaFe), (2864, Chicago), (2037, SanFrancisco)]
Next Node: San Francisco

Step2:

Current node: San Francisco
Children of current node: [(2103, Dallas), (2290, LosAngeles), (2372, Bakersville), (2864, Chicago), (5453, Boston)]
Path: Seattle -> San Francisco, Cost: 2037
Explored: 'SantaFe', 'Seattle'
Frontier: [(2037, SanFrancisco), (2864, Chicago), (2103, Dallas), (3609, Bakersville)]
Next Node: Dallas

Step3:

Current node: Dallas ← <- <- <- <-GOAL
Children of current node: [(, Austin), (,New York)]
Path: Seattle -> Santa Fe -> Dallas, Cost: 2103
Explored: 'SantaFe', 'SanFrancisco', 'Seattle'
Frontier: [(2103, Dallas), (2290, LosAngeles), (2372, Bakersville), (2864, Chicago), (5453, Boston)]
Next Node: Algorithm stops

Extra-credit: The code for checking if the heuristic is consistent is given in the seattle_to_dallas.py file using the triangle inequality. It is computed for each node in the path in the Road_Problem class under the path_cost function.

Software Engineering (includes documentation for your programming assignments)

Your README file must include the following:

- Your name and email address.
- *Homework number* for this class (AI CS6364), and the *number of the problem* it solves.
- A description of every file for your solution, the programming language used, supporting files from the aima code used, etc.
- How your code operates, in detail.
- A description of special features (or limitations) of your code.

Within Code Documentation:

- Methods/functions/procedures should be documented in a meaningful way. This can mean expressive function/variable names as well as explicit documentation.
- Informative method/procedure/function/variable names.
- Efficient implementation
- Don't hardcode variable values, etc