Assignment 2

Håkon Mork
ECSE 526 Artificial Intelligence

February 28, 2013

1 Two-dimensional navigation

1.1 State utilities and optimal policy

After 42 rounds of value iteration, the values converged to:

0.078	0.027		-0.187
0.169	0.084	0.102	-0.154
0.283		0.257	-1
0.392	0.549	0.699	1

After 6 iterations, the optimal policy converged to:

$\rightarrow$	$\downarrow$		$\leftarrow$
$\downarrow$	$\rightarrow$	$\downarrow$	$\uparrow$
$\downarrow$		$\downarrow$	
$\rightarrow$	$\rightarrow$	$\rightarrow$	

It's curious that the optimal policy differs in some places from following the gradient of the convergence values.

1.2 Linear algebra policy evaluation

To construct the matrix A, observe that the final utility b_{ij} in a given state (i, j) on the board depends linearly on the utilities of its neighbors—and possibly itself, if the policy we've chosen causes us to face a wall and bounce back when we try to walk into it. For example, we have that $u_{11} \leftarrow r + 0.8u_{11} + 0.2u_{21}$ because going south will take us either to the same place or to the state to the west with 0.8 and 0.2 probability, respectively. On the other hand, going south from $(3, 3)$ yields $u_{33} \leftarrow r + 0.1u_{23} + 0.7u_{32} + 0.2u_{43}$ as per the general rule, which applies when there are no obstacles present to obstruct movement, such as blocked squares or board boundaries.

Following this line of reasoning, we can construct the (16×16) matrix A whose entries are the weights by which the utility in each state depend on other states, following the format on page 657 of Russell and Norvig, except that we also have to add the reward for each state: $A\mathbf{u}_n + \mathbf{r} = \mathbf{u}_{n+1}$. Rearranging, we get $A\mathbf{u}_n = \mathbf{u}_{n+1} - \mathbf{r} = \mathbf{b}$. The entries in every row of A clearly have to sum to 1, since every step we take must lead somewhere. Since the matrix is too big to fit here, please see appendix A.1.

I get the following final utilities for $r = -0.02$:

0.347	0.142		-0.917
0.434	0.141	0.095	-0.888
0.546		0.397	-1
0.575	0.675	0.825	1

and $r = -0.04$:

-0.107	-0.230		-0.992
-0.015	-0.191	-0.079	-0.935
0.093		0.239	-1
0.150	0.350	0.650	1

See also appendix A.2 for further comments regarding construction and implementation.

1.3 State utilities are linear in r

Proof. Consider equation 17.10 in Russell and Norvig:

$$U_i(s) = R(s) + \gamma \sum_{s'} T(s, \pi_i(s), s') U_i(s')$$

The utility vector $\mathbf{u}$ consists of utilities such as this for all the states s in the state space $\mathcal{S}$; that is, $\mathbf{u} = [U_i(s)]_{s \in \mathcal{S}}$. We can then, by linearity of vector addition, split each U_i into its constituent parts, one for the reward per state and one for the discounted utility of future state:

$$\begin{aligned}
\mathbf{u} &= [U_i(s)]_{s \in \mathcal{S}} \\
&= \left[R(s) + \gamma \sum_{s'} T(s, \pi_i(s), s') U_i(s') \right]_{s \in \mathcal{S}} \\
&= [R(s)]_{s \in \mathcal{S}} + \left[\gamma \sum_{s'} T(s, \pi_i(s), s') U_i(s') \right]_{s \in \mathcal{S}} \\
&= r\mathbf{b} + \mathbf{c}
\end{aligned}$$

where $[R(s)] = r\mathbf{b}$ since all states get an addition of r (to a constant) in their utility for each iteration. Consider, for example, the vector

$$\mathbf{b}[s] = \begin{cases} 1 & \text{if state } s \text{ is a nonterminal} \\ 0 & \text{otherwise} \end{cases}$$

which accurately describes both our state space and those in the book: only the nonterminals have a reward added to them in each iteration. Note that this representation is essentially the same as the one in problem 1.2; see appendix A.2 for comments. □

1.4 Utility plot I

See figure 1 in appendix B.1. The relationship between r and the utilities is clearly linear.

1.5 Utility plot II

See figure 2 in appendix B.2. The utilities appear to be piecewise linear, and unexpectedly jagged. All the three states have a large negative jump in utilities at $r = -2.333$ and other places, which may be because a new path through the state space is chosen over the old one. Of course it may also be a bug.

1.6 Policy differences

It would make sense for the optimal policy to change from a very negative r to one closer to zero. At $r = -3.5$, the agent is in a grim situation, and even jumping into the pit with utility -1 would be better than staying in the nonterminal squares and endure the penalty of a large negative r. The less negative penalty of $r = -0.5$, on the other hand, may be sufficient for the agent to prefer going the long way around to the positive terminal node. Since there appear to be discontinuities in the utility graph, I guess that happens here.

1.7 Equation modification

Equation 17.5 in Russell and Norvig is

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U(s')$$

where we assume that the immediate reward $R(s)$ in state s does not depend on what action we take; that is, $R(s) = R(s, a, s')$ for all a and s'. If, however, the reward does depend on the action taken, the utility maximization over a must extend to the reward as well, with the appropriate weighting T for each direction. Therefore my guess is that we should write

$$U(s) = \gamma \max_a \sum_{s'} T(s, a, s') \left(U(s') + R(s, a, s') \right).$$

Still, this means that R is discounted by γ along with the utilities from neighboring states, which I'm not convinced is the right thing to do.

2 Feeding the kangaroo

2.1 State representation

I see no reason not to represent the state space in the most straightforward way: as a one-dimensional array of utilities. I put the complexity of managing what states are terminals and not in the evaluation function. The convoluted jumping system is handled here too, since putting those instructions into the state representation would be needlessly complicating.

2.2 Terminals and nonterminals

We have three terminals: holes 3 and 7, and the food square 5. It also looks like we have *seven* nonterminals: regular squares 1, 2, 4, 6, 8, and 9, as well as the food square 5: since the kangaroo has to perform a jump of magnitude zero to eat the food and win the game while standing in square 5, that square should be considered to be both a terminal and a nonterminal state.

2.3 Possible actions

Jump	1	2	3	4	5	6	7	8	9
2 left	✓	✓		✓		✓		✓	
1 left	✓	✓			✓	✓			✓
In place	✓	✓		✓	✓	✓		✓	✓
1 right	✓			✓	✓			✓	✓
2 right		✓		✓		✓		✓	✓

I only consider actions that are "allowed" in the sense that we don't lose if we perform that action; for example, jumping one step to the right from square 2 would make the kangaroo tumble into the pit and lose, so that option is not considered feasible. I also consider squares 3 and 7 to be of little interest because being in one of them means that we've lost and the game is over, so it doesn't much matter what actions are possible there.

Keep in mind that these are the actions that are possible in general; what is allowed in practice at any stage in the game depends on the previous jump, because of the restriction on absolute difference in jump magnitude.

2.4 State utilities and optimal policy

A couple of comments: to save horizontal space I elided the utilities in the terminal states, since we already know what they are. I also left out the policies for the negative terminals, since they're irrelevant because the game ends when we enter those states.

Double arrows mean a jump of magnitude 2 in that direction. An upward arrow means a jump of magnitude 0.

Value iteration with $r = -0.04$:

0.621	0.734		0.86		0.86		0.734	0.621

Policy iteration with $r = -0.04$:

$\rightarrow$	$\rightrightarrows$		$\rightarrow$	$\uparrow$	$\leftarrow$		$\leftrightarrows$	$\leftarrow$

Value iteration with $r = -0.5$:

-0.626	-0.14		0.4		0.4		-0.14	-0.626

Policy iteration with $r = -0.5$:

$\rightarrow$	$\rightrightarrows$		$\rightarrow$	$\uparrow$	$\leftarrow$		$\leftrightarrows$	$\leftarrow$

Value iteration with $r = -1$:

-1.9	-1.09		-0.1		-0.1		-1.09	-1.9

Policy iteration with $r = -1$:

$\rightrightarrows$	$\rightrightarrows$		$\rightarrow$	$\uparrow$	$\leftarrow$		$\leftrightarrows$	$\leftrightarrows$

For a negative enough reward, e.g. $r = -1$, the kangaroo evidently prefers jumping into the pit rather than enduring the negative reward and get the food.

A Linear algebra

A.1 System from problem 1.2

This is the system $A\mathbf{u}_n = \mathbf{b}_n$ from problem 1.2, where $\mathbf{b}_n = \mathbf{u}_{n+1} - \mathbf{r}$. We can repeatedly solve this system until convergence, with $\mathbf{u}_n = \mathbf{b}_n$. I added lines in A to identify (4×4) submatrices for the sake of readability.

$$\left[\begin{array}{cccc|cccc|cccc|cccc}
0.8 & 0 & 0 & 0 & 0.2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.7 & 0.3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0.7 & 0.1 & 0 & 0 & 0 & 0.2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0.7 & 0.1 & 0 & 0 & 0 & 0.2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0.1 & 0 & 0 & 0 & 0.7 & 0 & 0 & 0 & 0.2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0.1 & 0 & 0 & 0 & 0.7 & 0 & 0 & 0 & 0.2 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.1 & 0 & 0 & 0.7 & 0.2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0.1 & 0 & 0 & 0 & 0.7 & 0 & 0 & 0 & 0.2 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.7 & 0.1 & 0 & 0 & 0 & 0.2 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0 & 0 & 0.7 & 0 & 0 & 0 & 0 & 0.2 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0 & 0 & 0.7 & 0.2 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.7 & 0.3
\end{array}\right]
\begin{bmatrix} u_{11} \\ u_{12} \\ u_{13} \\ u_{14} \\ u_{21} \\ u_{22} \\ u_{23} \\ u_{24} \\ u_{31} \\ u_{32} \\ u_{33} \\ u_{34} \\ u_{41} \\ u_{42} \\ u_{43} \\ u_{44} \end{bmatrix}_n
=
\begin{bmatrix} u_{11} \\ u_{12} \\ u_{13} \\ u_{14} \\ u_{21} \\ u_{22} \\ u_{23} \\ u_{24} \\ u_{31} \\ u_{32} \\ u_{33} \\ u_{34} \\ u_{41} \\ u_{42} \\ u_{43} \\ u_{44} \end{bmatrix}_{n+1}
-
\begin{bmatrix} r \\ r \\ r \\ r \\ r \\ 0 \\ r \\ r \\ r \\ r \\ r \\ 0 \\ 0 \\ 0 \\ r \\ r \end{bmatrix}$$

A.2 Further comments

Note that the elements of $\mathbf{r}$ are r except for some particular states. The "special" states, i.e., the terminals $(4,1)$ and $(4,2)$ as well as the blocked squares $(2,2)$ and $(3,4)$, are not influenced by any other squares, so they just retain their old value from one iteration to the next. This is represented by these rows having a weight of 1 on the corresponding column in A, as well as getting no reward when we iterate. Technically, there is no need to assign a utility to the blocked squares, but I did so for the sake of concistency and ease of implementation.

The iteration is actually implemented in code as an augmented matrix multiplied by an elongated vector: $[\mathbf{r} \mid A][1 \circ \mathbf{u}_n] = \mathbf{u}_{n+1}$. That is, we attach a column vector $\mathbf{r}$ of rewards to the side of A, each element of which is multiplied by the 1 we put on the top of $\mathbf{u}$. This representation is clearly equivalent to the one stated earlier, in problem 1.2.

B Figures

B.1 From problem 1.4

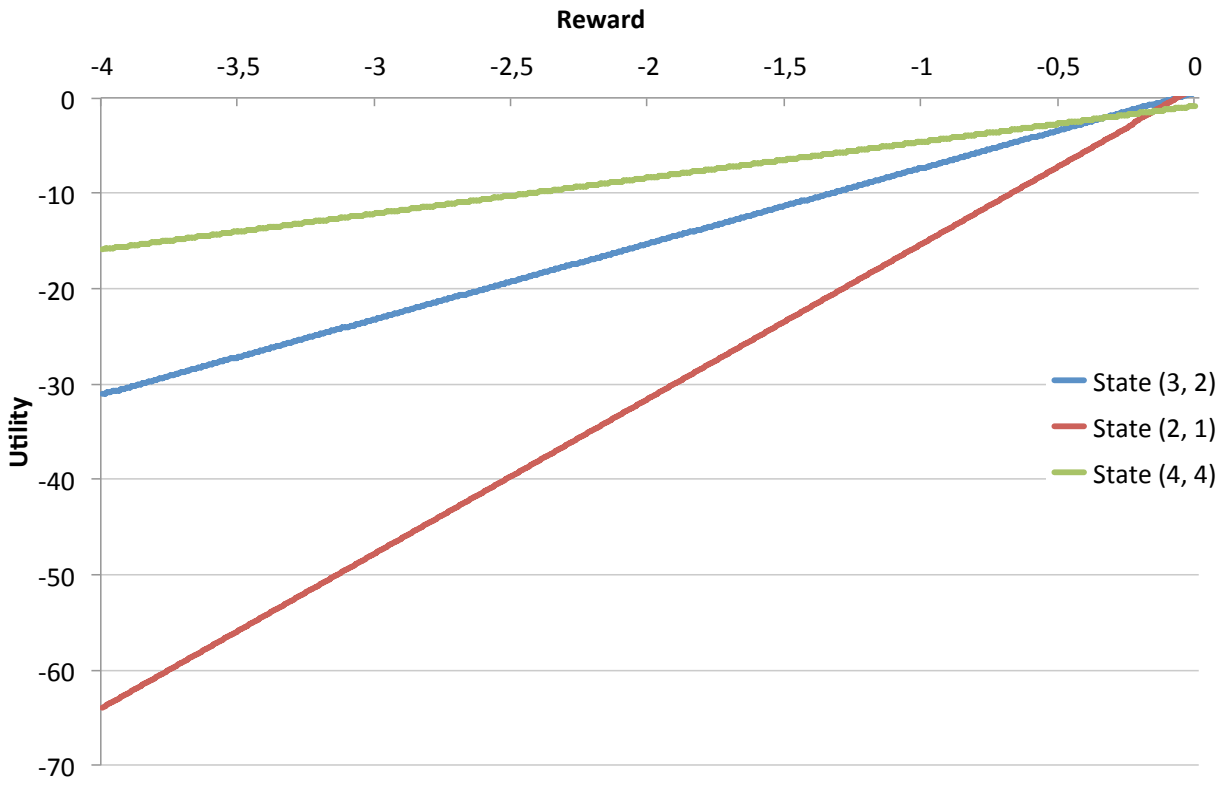

Figure 1

B.2 From problem 1.5

Figure 2

C Usage

Assuming the code resides in the current directory, it can be run with `python ass2.py [args]`. The arguments are the problems to be solved, i.e., any combination of `11`, `12`, `14`, `15`, and `24`. Use `all` to solve all problems in sequence. Examples:

- `python ass2.py 11`
- `python ass2.py 12 14 15 24`
- `python ass2.py all`

Note that the code for problem 1.5 prints out 4000 lines of values, which takes a little while to run and is not overly exciting to watch.

D Design discussion

The code is structured into functions according to problems to solve, with some helper functions for code reuse and modularity. The value iteration and policy iteration algorithms are described in detail in Russell and Norvig, so I can't see that there is much room for discussing them. However, I'll point out that I don't use the exact same algorithm for part one and part two, but rather slightly different variations on the same algorithms described in the textbook. This is because there are enough differences between the two scenarios, such as the two-dimensionality in the first problem and the wall bouncing and momentum restrictions in the second one, that I think it's worthwhile to deal with separate algorithms instead of a more general one. This separation makes e.g. the state transition logic much easier to deal with.