HOW I LEARNED TO ESCAPE THE MONSTER

ISAAC J. LEE

Abstract. We consider a discrete variation of the princess and monster game and use Q-learning to safely guide the princess to the exit. The princess keeps an optimal policy for each possible state the monster can be in, and her policy is to use the optimal policy corresponding to the state the monster is in currently. We look at how the learning parameter λ and the number of monsters present affect the princess' rate of success from simulations.

Keywords and Phrases: reinforcement learning, Q-learning, princess and monster game

1. Introduction

Reinforcement naturally appears when an organism learns from interacting with an environment: a rat in a Skinner box presses down a lever to acquire food or to avoid being shocked, a child says politely "hello" for social acceptance or "thank you" in return for a candy for future, possibly greater gifts, a person drinks to spend a good time with his mates or to put aside his troubles for awhile, etc.

Reinforcement learning (RL) concerns "how to map situations to actions [...] to maximize a numerical reward signal" [3]. The learner is not explicitly told what actions to take and what not to take, but instead discovers through trial-and-error interactions with the environment the action that will maximize his reward.

RL is different from supervised learning, in which a supervisor gives the learner sets of input and output. The learner attempts to define a relation between them so that he can respond on his own to future input. We already see limitations to this approach. The supervisor must provide "examples of desired behavior that are both correct and representative of all the situations in which the agent has to act" for a successful learning to occur [3]. Would it not be better if we let the agent roam around an environment and learn from his own accomplishments and mistakes?

Among many methods to implement RL, such as policy or value iteration, TD(λ), and residual algorithms, we consider Q-learning, which is known for easiness of implementation [3, 2]:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \cdot max_a\{Q(s_{t+1}, a)\} - Q(s_t, a_t) \right].$$

In a sense, the value of Q tells the agent how good his action a_t is given the state s_t. So when the Q values are "nearly converged to their optimal values," he is "to act greedily, taking, in each situation, the action with the highest Q value" [2]. In the simulations, if there were multiple actions all leading to the highest Q value, we let the princess randomly choose one, thereby allowing some degree of exploration.

RL has been applied extensively in artificial intelligence. For example, Tesauro's TD-Gammon uses a neural network to approximate the value function and plays backgammon (which has "approximately 10^{20} states") at a near-expert level [5, 2].

Schaal and Atkeson taught a robot to juggle a devil-stick, which is "a complex non-linear control task involving a six-dimensional state space and less than 200 msecs per control decision," whereas Crites and Barto simulated an "elevator dispatching task [...] to minimize the average squared wait time for passengers" [2].

Now, the princess and monster game originates from Rufus Isaacs' book *Differential Games*. The princess and the monster move in a bounded region, and each player cannot see the other until the distance between them is less than a fixed constant, at which the capture occurs [1]. We consider a discrete variation of the game, described in detail below.

2. The Problem

There is a finite set of rooms $\mathscr{R} = \{0, 1, \cdots, N-1\}$. The game starts with the princess and the monster in different rooms, with the princess in a room other than the exit (room 0). At discrete times $t = 0, 1, \cdots$, they remain inside a room. Between t and $t+1$, the monster moves to an adjacent room in a fierce search of the princess, whereas the princess can move to an adjacent room to look for room 0 or choose to stay where she was at time t. No light has dared to enter this fortress in which she is held captive. So even should they pass by a nick while they move to another room, they do not recognize each other. At the discrete times, the monster does not know where she is (unless they happen to be in the same room), but the princess is sage and knows where the monster is. If the princess reaches room 0 (regardless of whether the monster is there also), she will have successfully escaped the monster's gripes and will live happily ever after with a typical, charming prince. If the princess and the monster are in the same room any time before, well, things will be very grim.

The princess uses Q-learning to learn what action to take when she is in room 0 (stay in room 0, clearly) and when the monster is in room $1, \cdots, N-1$ at time t. For each of these N cases, she considers the transition rule [4]

$$Q(s_t, a_t) = R(s_t, a_t) + \gamma \cdot max_{a_{t+1}}\{Q(s_{t+1}, a_{t+1})\},$$

- Q, the value of the princess' experience from (s_t, a_t)
- R, the reward she gets from (s_t, a_t)
- s_t, the state (room) she is in at time t
- a_t, the action she takes (to which possible room?) at time t
- γ, a learning parameter $(0 \leq \gamma < 1)$
- s_{t+1}, the state she will be in at time $t+1$ after a_t is taken
- a_{t+1}, the action she can take at time $t+1$ once s_{t+1} is reached.

Q is initially set to 0, i.e. the princess has no experience with the environment. The parameter γ is such that if γ is close to 0, she tends to consider immediate reward, and if γ is close to 1, she will consider future reward with greater weight [4]. (For $\gamma \geq 1$, Q diverges.) We assign a value of $reward > 0$ to R if an action takes us to room 0, and of $penalty < 0$ if it takes us to a room with the monster.

We note that the rooms and their adjacency can be represented as a graph. So Q and R can be represented as a matrix in which the rows and columns indicate the rooms the princess is in before and after the transition, respectively. In particular,

the graph will often be sparse (a room is unlikely connected to many other rooms), so we consider Q and R as sparse matrices, i.e. store each as three vectors (two index vectors, one value vector).

After a series of trainings, the princess embarks on escaping from the fortress. She uses her knowledge of the whereabouts of the monster to choose the Q function that was designed for that particular case and then follow the optimal policy. We look at all $(N-1)^2$ possible starting positions of the princess and the monster, and count how many times she has escaped.

3. Examples

We consider two distinct maps and look at subcases. All our simulations involved $reward = 400$, $penalty = -2000$.

Example 1. Consider the map in Figure 1.

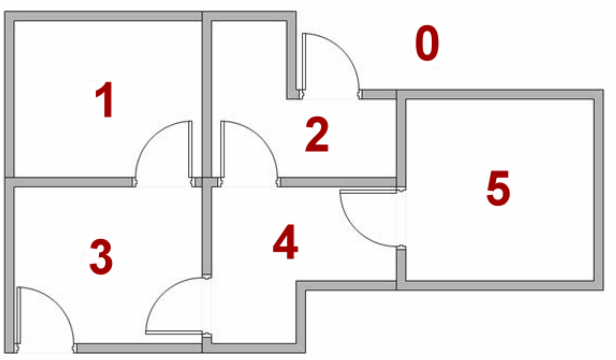


Figure 1. $|\mathscr{R}| = 6$

To demonstrate how Q-learning works, we take $\gamma = 0.8$ and consider what to do when the monster is in room 2. Initially,

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & & 0 & 0 & & \\ & 0 & & 0 & & \\ 0 & & 0 & & 0 & \\ 0 & 0 & & 0 & 0 & \\ & & 0 & 0 & 0 & 0 \\ & & & & 0 & 0 \end{bmatrix} \end{array}, \quad R = \begin{bmatrix} 400 & & -2000 & 0 & & \\ & 0 & & 0 & & \\ -2000 & & -2000 & & -2000 & \\ 400 & 0 & & 0 & 0 & \\ & & -2000 & 0 & 0 & 0 \\ & & & & 0 & 0 \end{bmatrix}$$

An empty cell in the matrices $Q = [q_{ij}]$ and $R = [r_{ij}]$ indicates that rooms i and j are not adjacent. Recall, we store Q and R above in sparse format to save resources. Let us begin the training. Suppose the princess is in room 3 and takes the action to move to room 0. Then,

$$\begin{aligned} Q(3,\,0) &= R(3,\,0) + 0.8 \cdot max_{a_{t+1}}\{Q(0,\,a_{t+1})\} \\ &= 400 + 0.8 \cdot max\{Q(0,\,0),\,Q(0,\,2),\,Q(0,\,3)\} \\ &= 400 + 0.8 \cdot 0 \\ &= 400. \end{aligned}$$

So Q is now,

$$Q = \begin{bmatrix} 0 & & 0 & 0 & & \\ & 0 & & 0 & & \\ 0 & & 0 & & 0 & \\ 400 & 0 & & 0 & 0 & \\ & & 0 & 0 & 0 & 0 \\ & & & & 0 & 0 \end{bmatrix}.$$

Now, suppose in our second training, the princess is in room 4 and takes the action to move to room 3. Then,

$$\begin{aligned} Q(4,\,3) &= R(4,\,3) + 0.8 \cdot max_{a_{t+1}}\{Q(3,\,a_{t+1})\} \\ &= 0 + 0.8 \cdot max\{Q(3,\,0),\,Q(3,\,1),\,Q(3,\,3),\,Q(3,\,4)\} \\ &= 0 + 0.8 \cdot 400 \\ &= 320. \end{aligned}$$

and

$$Q = \begin{bmatrix} 0 & & 0 & 0 & & \\ & 0 & & 0 & & \\ 0 & & 0 & & 0 & \\ 400 & 0 & & 0 & 0 & \\ & & 0 & 320 & 0 & 0 \\ & & & & 0 & 0 \end{bmatrix}.$$

Note that in the training scenario above, we just happen to have chosen the best action for the princess when she is in rooms 3 and 4 to illustrate how the values of Q are updated. In practice, out of the possible actions a_t, she would randomly select one and gain experience (i.e. update Q accordingly), leading her to perform that action or not during the real run. Given enough training (1000 iterations is sufficient for this example), we will get

$$Q = \begin{bmatrix} 2000 & & -2320 & 1600 & & \\ & 1280 & & 1600 & & \\ -400 & & -2320 & & -720 & \\ 2000 & 1280 & & 1600 & 1280 & \\ & & -2320 & 1600 & 1280 & 1024 \\ & & & & 1280 & 1024 \end{bmatrix}.$$

So when the monster is in room 2, the princess' optimal policy is as follows: If she is in room 0, then stay in room 0 (obvious). If she is in room 1 (3, 4, 5), then move to room 3 (0, 3, 4). Note, we easily see from the transition rule that once Q converges, $sup\{q_{ij}\} = reward/(1-\gamma)$. We similarly find an optimal policy for the case that the princess is in room 0 and the case that the monster is in room 1, 3, 4, or 5.

Table 1 records the number of times the princess has escaped the monster for $\gamma = 0.2,\, 0.4,\, 0.6,\, 0.8$ ($trainingSize = 20000$, $trialSize = 10000$). The rows and columns denote in which room the princess and the monster began, respectively.

$\gamma = 0.2$:

05051	-----	10000	03741	06510	10000
10000	10000	-----	10000	10000	10000
10000	10000	10000	-----	10000	10000
04269	04836	10000	10000	-----	10000
10000	10000	02051	02966	01791	-----

$\gamma = 0.4$:

04864	-----	10000	03813	06670	10000
10000	10000	-----	10000	10000	10000
10000	10000	10000	-----	10000	10000
04532	04755	10000	10000	-----	10000
10000	10000	01997	03064	01719	-----

$\gamma = 0.6$:

04791	———	10000	03801	06676	10000
10000	10000	———	10000	10000	10000
10000	10000	10000	———	10000	10000
04711	04709	10000	10000	———	10000
10000	10000	02064	03079	01736	———

$\gamma = 0.8$:

04877	———	10000	03854	06731	10000
10000	10000	———	10000	10000	10000
10000	10000	10000	———	10000	10000
04575	04640	10000	10000	———	10000
10000	10000	02067	03055	01720	———

Table 1. $|\mathscr{R}| = 6$, $\gamma = 0.2, 0.4, 0.6, 0.8$

We want to compare how well the princess has fared for different values of γ. So for each γ, we create 11 bins (0%, 10%, $\cdots$, 100%) and find what proportion falls into each bin. We use the floor function for categorization; for example, 05051 and 03741 under $\gamma = 0.2$ would fall into the 50% and the 30% bin, respectively. We then make a "reverse" ogive (i.e. 1 - cdf) to easily see the probability that the princess' success rate is a given percentage or higher. We get the following:

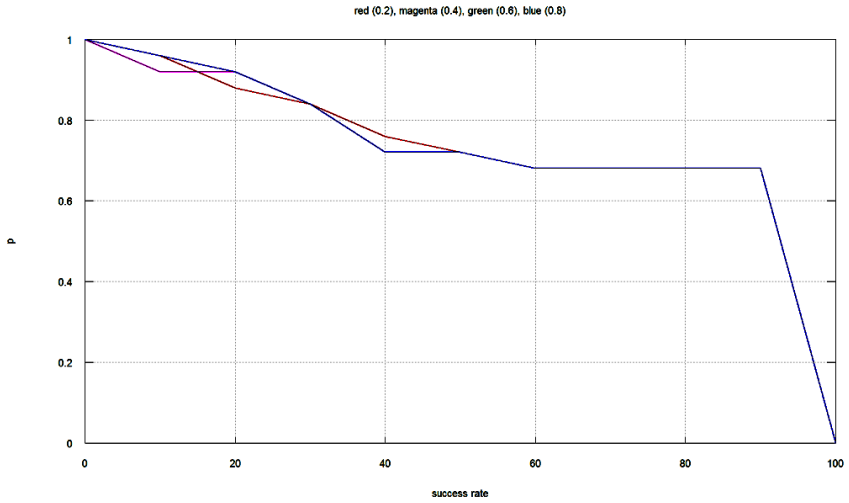


Figure 2. $|\mathscr{R}| = 6$, 1 monster

Note that the distributions for $\gamma = 0.6$ and $\gamma = 0.8$ come out to be the same, so the graph for $\gamma = 0.6$ (green) is overlaid by that for $\gamma = 0.8$ (blue). We see little variation among the graphs, but because we have a small sample size of $(N-1)^2 = 25$, with most samples in the 100% bin, it is difficult to tell whether γ truly has no effect on the princess' success rate.

Example 2. Consider a more elaborate map in Figure 3.

After training, the princess comes up with 30 optimal policies ($trainingSize = 20000$): one to be used when she is in room 0 and the other 29 corresponding to which room the monster is in. As before, we perform $trialSize = 10000$ trials for $\gamma = 0.2, 0.4, 0.6, 0.8$ (Tables 2-1, 2-2). For each γ, we create 21 bins (0%, 5%, $\cdots$, 100%) and find the reverse ogive (Figure 4).

Let us up the ante and place the princess against two monsters. (Here, we only consider the case where she begins in room 29.) The setting is similar. Neither monster begins in the same room as the princess, and their movements are independent of one another's. At each discrete time, she uses one of $(N-1)^2 = 841$ optimal policies that corresponds to the locations of the monsters. She must get to room 0 without being caught by either. We get Table 2-3, 2-4 and Figure 5.

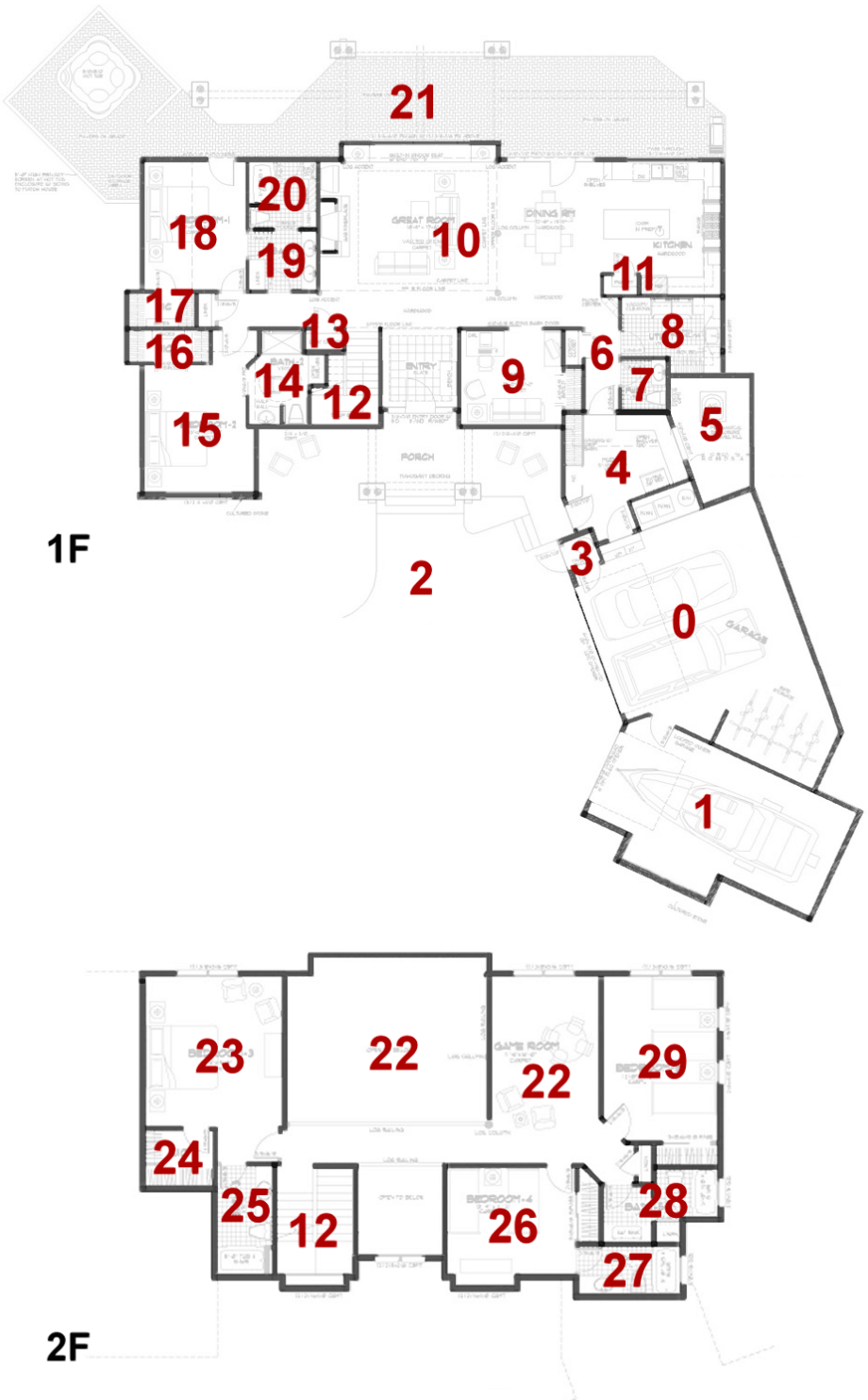


Figure 3. $|\mathscr{R}| = 30$

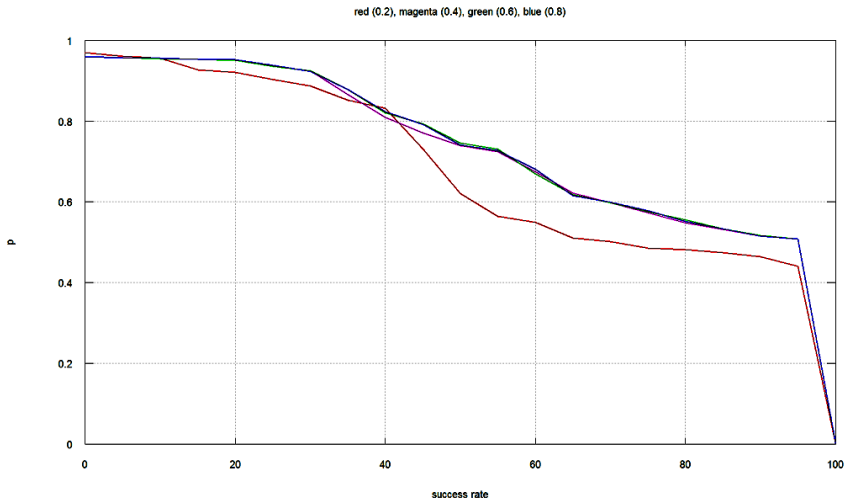


Figure 4. $|\mathscr{R}| = 30$, 1 monster

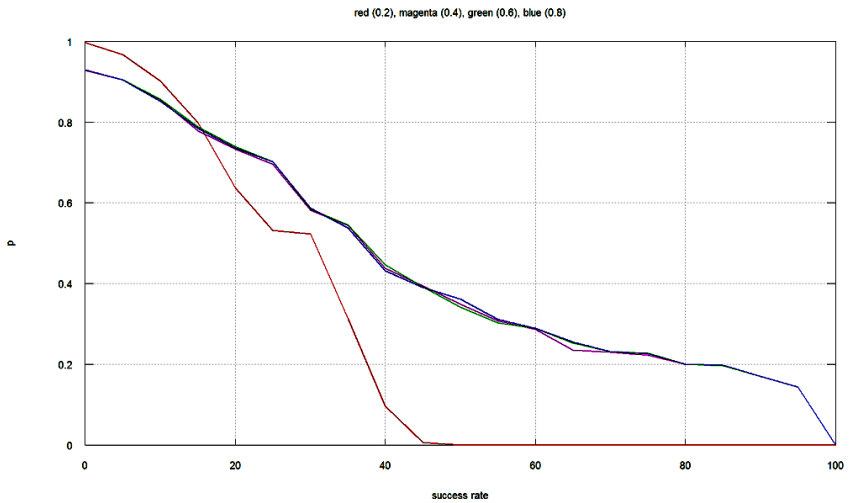


Figure 5. $|\mathscr{R}| = 30$, 2 monsters

In both Figures 4 and 5, we see that the princess markedly underperforms with $\gamma = 0.2$, i.e. when she heavily focuses on the immediate reward. (With two monsters roaming around, the probability that she escapes at least half the time is 0! She does a better job for $\gamma = 0.4, 0.6, 0.8$, but the probability that she escapes two monsters at least half the time is only mid-30%.)

This makes sense. Before, in Example 1, the rooms were close to one another, so the likelihood of her ending up in the same room with the monster would be consistent across γ. Now, there are more rooms and they are more spread apart from one another. In particular, the princess must go through rooms 10 and 22 on her way to room 0, and these two rooms have a high degree of 10 and 5, respectively. The monster, who always moves during transition, would likely end up in either of these two rooms, so she would have to give more weight to the future reward.

The distributions for $\gamma = 0.4, 0.6, 0.8$ are much the same, suggesting there may be a threshold γ^* under her policy such that we do not see any drastic improvement in her success rate for $\gamma > \gamma^*$. From Tables 2-3 ($\gamma = 0.4$) and 2-4 ($\gamma = 0.6, 0.8$), we see that the princess never succeeded in escaping when either monster was in room 28, for then they would both be in room 22 after transition. In addition to the monster's current state, if she can deliberate the actions that the monster can make and avoid actions that are risky, she may have a better chance of escape.

4. Conclusion

We considered a discrete variation of the princess and monster game. The princess, who used Q-learning to learn the environment, kept a record of the optimal move for all possible positions that the monster(s) could be in and followed it at discrete times to get to room 0.

We observed from the simulations that for a map with a small number of rooms with approximately equal number of degree, the value of γ did not seem to provide her any more or less advantage. However, for a map with a large number of rooms with some having higher degree than do others, a γ close to 0 lowered her chance of escape. From this second map, we concluded that there may be a threshold γ^* such that there is no drastic improvement in her success rate for $\gamma > \gamma^*$.

We note that Figures 2, 4, and 5 do not tell us the entire story, for we treated the samples as if they came from a homogeneous pool. In actuality, some samples were obtained when the game began with the princess and the monster close to each other (e.g. in room 29 and room 28, respectively), whereas others when they were far away from each other. Furthermore, the number of rooms to which a room is adjacent (i.e. the degree) may play a factor as well.

We leave two main questions in the hands of the brave knights. First, we would like to find out if such γ^* exists and what its value is, analytically or empirically. Whether γ^* depends on the number of monsters present, and if so, how we can efficiently store and compute the data whose size grows exponentially, are related to this problem. Secondly, because the monster randomly moves to one of the adjacent rooms, the princess may want to consider not only the optimal policy for the room the monster is in at time t, but also the room the monster can be in at time $t + 1$. We seek to know whether this policy or some other can improve her success rate so that she may live happily ever after.

References

[1] Shmuel Gal, Search Games with Mobile and Immobile Hider, Journal of Control and Optimization **17** (1979), 99-122.

[2] Leslie Pack Kaelbling, Michael L. Littman, Andrew W. Moore, Reinforcement Learning: A Survey, Journal of Artificial Intelligence Research **4** (1996), 237-285.

[3] Richard S. Sutton and Andrew G. Barto, Reinforcement Learning: An Introduction, The MIT Press, 1998.

[4] Kardi Teknomo. 2005. Q-Learning by Examples. http://people.revoledu.com/kardi/tutorial/ReinforcementLearning/index.html.

[5] Gerald Tesauro, Temporal Difference Learning and TD-Gammon, Communications of the ACM **38** (1995), 58-68.

Table 2-1. $|\mathscr{R}| = 30$, 1 monster, $\gamma = 0.2$ (top), 0.4 (bottom)

Table 2-2. $|\mathscr{R}| = 30$, 1 monster, $\gamma = 0.6$ (top), 0.8 (bottom)

04553	04084	04492	04079	04023	04513	04369	03969	03848	04143	03800	04147	02692	04168	04121	04479	04004	04077	04047	04245	04050	04218	01916	02196	02198	02124	01981	01888	01315	-----
03934	05029	03812	05004	04946	03907	03781	04819	04932	03666	04613	03590	02280	03663	04860	04058	04932	04398	04237	04165	04553	04086	02679	02013	02884	02954	01680	02723	00915	-----
04507	04045	04376	03907	03982	04514	04272	03808	03715	04111	03724	04082	02552	04177	04015	04256	03996	04080	04055	04139	03926	03996	01713	02229	02203	02197	01988	01884	01298	-----
04064	04978	03920	05014	04967	03967	03874	04857	04901	03791	04557	03757	02211	03668	04942	04089	04922	04502	04264	04066	04598	04054	02700	01995	02988	02909	01710	02731	00865	-----
03956	04974	03876	04970	05077	03989	03838	04936	04852	03677	04624	03847	02159	03721	04821	04056	04907	04441	04231	04024	04585	04043	02779	02012	02987	03019	01654	02779	00933	-----
04570	03968	04380	04049	03943	04585	04398	03945	03964	04224	03884	04131	02681	04215	03989	04526	04104	04160	04112	04208	04073	04208	01870	02258	02247	02242	01999	01931	01316	-----
04438	03830	04323	03805	03875	04404	04206	03794	03825	04019	03710	04082	02190	04129	03945	04197	03917	04058	04081	04191	03904	04135	01819	02045	02150	02213	01741	01846	01052	-----
03907	04928	03913	04939	04917	03919	03721	04862	04905	03616	04472	03697	02039	03672	04853	03951	04919	04476	04195	04137	04560	04124	02490	02023	02962	02902	01715	02705	00882	-----
03840	04912	03730	04954	05032	03945	03830	04887	04897	03714	04562	03717	02092	03683	04885	03972	04964	04453	04276	04051	04605	04077	02702	01983	03021	02954	01704	02695	00817	-----
04152	03563	04082	03692	03603	04209	04037	03589	03579	03789	03481	03850	02547	03880	03651	04016	03764	03809	03711	03929	03811	03941	01726	02012	01978	02013	01776	01776	01271	-----
03613	04630	03779	04714	04624	03669	03709	04553	04591	03573	04281	03530	01965	03511	04532	03985	04647	04224	04170	03908	04328	03875	02605	01854	02823	02819	01583	02667	00780	-----
04207	03727	04131	03704	03678	04230	04028	03654	03601	03841	03534	03936	02466	03883	03752	04026	03701	03854	03800	03881	03795	03893	01685	02002	02000	02033	01814	01782	01308	-----
02739	02232	02727	02321	02219	02749	02569	02314	02213	02645	02026	02581	01682	02711	02366	02749	02471	02497	02482	02574	02394	02580	00950	01477	01332	01245	01463	01183	01188	-----
04204	03740	04065	03714	03610	04102	03956	03607	03640	03884	03593	03812	02453	03823	03771	04072	03696	03748	03736	03910	03775	03970	01694	02018	02054	02024	01815	01804	01279	-----
03865	04909	03884	04765	04831	03984	03766	04769	04805	03757	04541	03627	02198	03731	04772	04106	04785	04380	04336	04132	04533	04060	02676	02108	02925	02929	01809	02624	00926	-----
04319	03988	04271	03964	03871	04493	04261	03851	03913	04066	03877	04079	02616	04052	03958	04325	04042	04104	03964	04085	04017	04010	01886	02209	02230	02261	01976	01969	01252	-----
03949	04955	03889	04906	04819	03863	03895	04759	04692	03742	04514	03716	02253	03761	04797	04066	04777	04400	04235	04080	04480	04000	02650	02108	02999	02939	01747	02632	00867	-----
03996	04331	04108	04331	04291	04125	03996	04311	04291	03979	04141	03957	02390	03987	04371	04245	04444	04167	04045	04047	04228	04020	02218	02054	02512	02540	01879	02273	01106	-----
04075	04250	03978	04210	04317	04021	03942	04144	04219	03807	04096	03881	02344	03967	04306	04165	04292	04112	04010	04026	04153	04035	02238	02041	02483	02567	01714	02192	01054	-----
04180	04098	04121	04111	03990	04143	04088	03933	04081	03928	03893	03884	02428	03871	04143	04173	04052	03995	04010	03947	04102	04029	02010	02059	02257	02319	01891	01976	01154	-----
03912	04498	03787	04510	04532	03954	03990	04615	04471	03747	04323	03757	02310	03691	04570	04013	04575	04252	04167	03987	04336	04052	02459	02037	02763	02731	01732	02378	00960	-----
04250	04032	04077	04054	03915	04198	04006	04007	03954	03904	03764	03907	02468	03980	04008	04139	03982	04070	03983	04011	04033	04056	01985	02101	02312	02263	01894	01995	01246	-----
01834	02650	01806	02732	02773	01786	01838	02708	02642	01744	02592	01724	01085	01687	02652	01978	02725	02323	02184	02023	02550	02010	01972	01336	02100	02194	01005	02138	00365	-----
02178	02018	02094	01988	01960	02140	02105	02028	02035	02044	01976	02061	01418	02083	02094	02187	02140	02108	01992	02011	02149	02102	01260	01758	01644	01605	01475	01416	01030	-----
02068	03079	02059	02929	02949	02132	02070	02960	02903	02094	02828	02037	01270	02084	03022	02359	03011	02663	02545	02327	02851	02346	02240	01613	02512	02478	01301	02173	00570	-----
02147	03079	02100	02948	02913	02045	02082	02850	02987	02096	02923	01996	01231	02099	02999	02289	02935	02725	02569	02365	02790	02320	02169	01645	02514	02490	01298	02299	00588	-----
01997	01762	01995	01754	01671	02000	01868	01694	01742	01868	01720	01910	01548	01894	01845	02058	01856	01973	01777	01950	01831	01853	01064	01466	01416	01394	01322	01251	00970	-----
02009	02853	02024	02870	02895	02061	01942	02886	02855	02014	02715	02004	01209	01954	02958	02143	02852	02442	02440	02258	02690	02274	02131	01527	02527	02401	01333	02194	00624	-----
01248	00874	01162	00850	00850	01293	01218	00821	00878	01294	00799	01326	01169	01342	01057	01321	01023	01079	01129	01163	00993	01166	00356	01016	00586	00603	00971	00481	00935	-----
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

01964	04390	01956	04378	04308	01802	01415	04391	04345	01432	04258	01462	00593	01467	04550	02433	04447	03500	03292	02587	04037	02471	04416	02938	04336	04371	02023	04360	00000	-----
04476	10000	04413	10000	10000	04309	03923	10000	10000	03388	09571	03381	01549	03257	10000	05569	10000	08117	06795	06072	09034	05256	10000	06851	10000	10000	04539	10000	00000	-----
01948	04316	01975	04530	04525	01845	01555	04329	04293	01449	04217	01410	00628	01431	04554	02498	04143	03385	03141	02610	03998	02328	04533	03018	04457	04463	02000	04500	00000	-----
04513	10000	04484	10000	10000	04372	03981	10000	10000	03277	09623	03272	01565	03314	10000	05595	10000	08111	06830	06073	09028	05319	10000	06781	10000	10000	04470	10000	00000	-----
04456	10000	04276	10000	10000	04376	03742	10000	10000	03413	09625	03340	01503	03401	10000	05552	10000	07987	07214	06024	09243	05918	10000	06784	10000	10000	04847	10000	00000	-----
01865	04343	01873	04351	04388	01858	01633	04325	04440	01457	04297	01419	00674	01554	04430	02497	04536	03610	03395	02690	04051	02394	04561	02987	04500	04398	02044	04565	00000	-----
01955	04651	01864	04348	04413	02078	01874	04658	04498	01550	04396	01549	00542	01209	04368	02355	04340	03749	03112	02579	04079	02535	04385	03058	04446	04294	01866	04461	00000	-----
04450	10000	04204	10000	10000	04591	03735	10000	10000	03336	09660	03311	01543	03290	10000	05388	10000	08266	06920	05733	09105	05560	10000	06511	10000	10000	04419	10000	00000	-----
04344	10000	04242	10000	10000	04614	03974	10000	10000	03279	09644	03340	01496	03420	10000	05347	10000	08318	06920	05830	09073	05699	10000	06663	10000	10000	04323	10000	00000	-----
01445	03383	01442	03326	03334	01480	01231	03324	03216	01036	03255	01080	00510	01098	03282	01920	03400	02723	02376	01994	03048	01833	03358	02266	03305	03312	01597	03398	00000	-----
04241	09645	04324	09625	09649	04185	03703	09677	09624	03195	09282	03053	01275	03185	09628	05248	09630	07858	06771	05734	08843	05221	09611	06325	09645	09644	03969	09625	00000	-----
01465	03254	01508	03316	03414	01448	01285	03436	03371	01059	03298	01101	00539	01085	03415	01819	03387	02733	02433	02046	03041	01832	03434	02238	03393	03284	01604	03301	00000	-----
00683	01578	00694	01530	01540	00675	00616	01535	01611	00515	01377	00481	00190	00516	01551	00877	01569	01178	01185	00874	01449	00957	01503	01072	01531	01533	00960	01532	00000	-----
01524	03313	01544	03286	03254	01457	01318	03357	03370	01107	03270	01027	00455	01091	03381	01874	03391	02701	02410	01930	03024	01855	03350	02282	03284	03389	01602	03383	00000	-----
04410	10000	04389	10000	10000	04269	03770	10000	10000	03343	09659	03257	01489	03343	10000	05562	10000	08132	06835	06080	09183	05313	10000	06770	10000	10000	04594	10000	00000	-----
02325	05523	02403	05524	05487	02386	01976	05494	05549	01834	05375	01887	00863	01853	05489	03093	05577	04432	03838	03399	05130	02934	05442	03722	05595	05520	02522	05616	00000	-----
04357	10000	04360	10000	10000	04382	03729	10000	10000	03266	09664	03370	01573	03261	10000	05549	10000	08125	06819	06050	09156	05377	10000	06743	10000	10000	04501	10000	00000	-----
03485	08043	03591	08115	08067	03489	03037	08149	08094	02784	07914	02860	01229	02748	08037	04509	08046	06418	05983	04802	07511	04386	08080	05352	08089	07990	03741	07999	00000	-----
03073	06808	03098	06777	06870	03046	02653	06674	06758	02318	06822	02305	01100	02365	06723	03724	06656	05615	04731	04131	06250	03614	07034	04689	06959	06798	02826	06670	00000	-----
02578	06043	02571	05980	05811	02536	02244	05963	06081	02017	05719	01990	00905	02021	06023	03339	05966	04806	04207	03620	05568	03199	05856	04069	05966	05945	02738	06069	00000	-----
04100	09178	04162	09099	09226	04107	03561	09131	09142	02967	08858	02946	01356	02910	09136	05142	09178	07478	06133	05587	08368	04908	09145	06312	09141	09174	04060	09166	00000	-----
02354	05152	02354	05292	05171	02312	01927	05233	05321	01856	05244	01826	00966	01822	05398	02913	05273	04453	03769	03197	05019	02916	05306	03623	05226	05401	02314	05287	00000	-----
04258	10000	04324	10000	10000	04247	03738	10000	10000	03310	09649	03299	01465	03247	10000	05413	10000	08162	07040	05944	09195	05541	10000	06760	10000	10000	04226	10000	00000	-----
02902	06766	02868	06741	06686	02773	02636	06781	06704	02207	06389	02251	01064	02300	06654	03673	06743	05377	04652	04088	06152	03743	06724	04660	06756	06654	03204	06807	00000	-----
04482	10000	04427	10000	10000	04263	03676	10000	10000	03372	09646	03335	01534	03320	10000	05452	10000	07862	06749	05923	09158	05234	10000	06738	10000	10000	04689	10000	00000	-----
04348	10000	04338	10000	10000	04257	03901	10000	10000	03383	09651	03264	01560	03315	10000	05612	10000	08027	06914	06051	09191	05304	10000	06779	10000	10000	04736	10000	00000	-----
02036	04551	02008	04599	04418	01971	01689	04683	04524	01629	03890	01564	00943	01575	04548	02552	04413	03863	03300	02843	04226	02367	05281	03474	04750	04734	01686	04795	00000	-----
04419	10000	04389	10000	10000	04284	03816	10000	10000	03388	09611	03348	01632	03367	10000	05633	10000	08047	06810	06046	09205	05377	10000	06767	10000	10000	05143	10000	00000	-----
00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000	-----
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

Table 2-3. $|\mathscr{R}| = 30$, princess begins in 29, 2 monsters, $\gamma = 0.2$ (top), 0.4 (bottom)

Table 2-4. $|\mathscr{R}| = 30$, princess begins in 29, 2 monsters, $\gamma = 0.6$ (top), 0.8 (bottom)

Appendix B. Code (Princess.java)

```
1  import java.io.*;
2  import java.util.*;
3
4  class Strategy {
5     // Store Q matrix in sparse format
6     private Vector rooms = new Vector();                    // Rooms
7     private Vector I = new Vector();                        // Row indices
8     private Vector J = new Vector();                        // Column indices
9     private Vector Q = new Vector();                        // Q matrix
10    private Vector R = new Vector();                        // R matrix
11    private double gamma;
12    private int penalty = -2000;
13
14    public Strategy(Vector rooms, Vector I, Vector J, Vector Q, Vector R, double
          gamma) {
15       this.rooms = rooms;
16       // Must do deep copy for Q and R
17       for (int k = 0; k < I.size(); k++) {
18          this.I.addElement(I.elementAt(k));
19          this.J.addElement(J.elementAt(k));
20          this.Q.addElement(Q.elementAt(k));
21          this.R.addElement(R.elementAt(k));
22       }
23       this.gamma = gamma;
24    }
25    public void addMonster(int monsterPosition) {
26       for (int k = 0; k < R.size(); k++) {
27          if (Integer.parseInt(I.elementAt(k).toString()) == monsterPosition ||
                Integer.parseInt(J.elementAt(k).toString()) == monsterPosition)
28             setR(penalty, k);
29       }
30    }
31    public void train() {
32       Random rand = new Random();
33       // This is the room we start from
34       int state = rand.nextInt(rooms.size());
35       // Find all possible rooms we can move to
36       Vector possibleJ = new Vector();
37       Vector possibleIndices = new Vector();
38       for (int i = 0; i < I.size(); i++) {
39          if (Integer.parseInt(I.elementAt(i).toString()) == state) {
40             possibleJ.add(J.elementAt(i));
41             possibleIndices.add(i);
42          }
43       }
44       // This is the room we move to
45       int action = Integer.parseInt(possibleJ.elementAt(rand.nextInt(possibleJ.
             size())).toString());
46       int indexOfAction = Integer.parseInt(possibleIndices.elementAt(possibleJ.
             indexOf(action)).toString());
47       possibleJ.removeAllElements();
48       possibleIndices.removeAllElements();
49
50       // Find all possible rooms we can move to once we made the action
51       Vector possibleQ = new Vector();
52       for (int i = 0; i < J.size(); i++) {
53          if (Integer.parseInt(I.elementAt(i).toString()) == action) {
54             possibleJ.add(J.elementAt(i));
55             possibleQ.add(Q.elementAt(i));
56             possibleIndices.add(i);
57          }
58       }
59
60       // Update Q
61       int newQValue = (int)Math.ceil(Integer.parseInt(R.elementAt(indexOfAction).
             toString()) + gamma * Integer.parseInt(Collections.max(possibleQ).
             toString()));
62       Q.setElementAt(newQValue, indexOfAction);
63    }
64    public int getPrincessMove(int princessPosition) {
65       Vector possibleQ = new Vector();
66       Vector possibleIndices = new Vector();
67       for (int i = 0; i < I.size(); i++) {
68          if (Integer.parseInt(I.elementAt(i).toString()) == princessPosition) {
69             possibleQ.add(Integer.parseInt(Q.elementAt(i).toString()));
70             possibleIndices.add(i);
71          }
72       }
73
74       // If there are multiple optimal moves possible, pick one randomly
```

```
75          int maxQ = Integer.parseInt(Collections.max(possibleQ).toString());
76          Vector possibleOptimalMoves = new Vector();
77          for (int i = 0; i < possibleQ.size(); i++) {
78              if (Integer.parseInt(possibleQ.elementAt(i).toString()) == maxQ) {
79                  possibleOptimalMoves.add(Integer.parseInt(possibleIndices.elementAt(i
                        ).toString()));
80              }
81          }
82          Random rand = new Random();
83          return Integer.parseInt(J.elementAt(Integer.parseInt(possibleOptimalMoves.
                elementAt(rand.nextInt(possibleOptimalMoves.size())).toString())).
                toString());
84      }
85      public int getMonsterMove(int monsterPosition) {
86          Vector possibleIndices = new Vector();
87          for (int i = 0; i < I.size(); i++) {
88              // The monster moves to another room to capture the princess
89              if (Integer.parseInt(I.elementAt(i).toString()) == monsterPosition &&
                        Integer.parseInt(J.elementAt(i).toString()) != monsterPosition) {
90                  possibleIndices.add(i);
91              }
92          }
93
94          Random rand = new Random();
95          return Integer.parseInt(J.elementAt(Integer.parseInt(possibleIndices.
                elementAt(rand.nextInt(possibleIndices.size())).toString())).toString()
                );
96      }
97      // Represent the vectors as a full matrix
98      public void full(boolean displayQ, boolean displayR) {
99          int[][] QFull = new int[rooms.size()][rooms.size()], RFull = new int[rooms.
                size()][rooms.size()];
100
101         for (int k = 0; k < I.size(); k++) {
102             int i = Integer.parseInt(I.elementAt(k).toString()),
103                 j = Integer.parseInt(J.elementAt(k).toString()),
104                 q = Integer.parseInt(Q.elementAt(k).toString()),
105                 r = Integer.parseInt(R.elementAt(k).toString());
106             QFull[i][j] = q;
107             RFull[i][j] = r;
108         }
109
110         if (displayQ) {
111             System.out.println("Q: ");
112             for (int i = 0; i < rooms.size(); i++) {
113                 for (int j = 0; j < rooms.size(); j++)
114                     System.out.print(QFull[i][j] + "\t");
115                 System.out.print("\n");
116             }
117             System.out.println();
118         }
119         if (displayR) {
120             System.out.println("R: ");
121             for (int i = 0; i < rooms.size(); i++) {
122                 for (int j = 0; j < rooms.size(); j++)
123                     System.out.print(RFull[i][j] + "\t");
124                 System.out.print("\n");
125             }
126             System.out.println();
127         }
128     }
129     private void setQ(int value, int index) { Q.setElementAt(value, index); }
130     private void setR(int value, int index) { R.setElementAt(value, index); }
131 }
132
133 public class Princess {
134     private static Vector<Strategy> strategies = new Vector<Strategy>();
135     private static Vector rooms = new Vector(), I = new Vector(), J = new Vector()
            , Q = new Vector(), R = new Vector();
136     private static int reward = 400;
137     private static double gamma = .8;
138     private static int trainingSize = 20000, trialSize = 10000;
139
140     private static void add(String line) {
141         int from = Integer.parseInt(line.substring(0, line.indexOf(" "))),
142             to = Integer.parseInt(line.substring(line.indexOf(" ") + 1, line.length
                    ()));
143         // Create edge (from, to)
144         I.add(from);
145         J.add(to);
146         Q.add(0);
147         R.add((to == 0) ? reward : 0);
148         // Create edge (to, from)
```

```
149         I.add(to);
150         J.add(from);
151         Q.add(0);
152         R.add((from == 0) ? reward : 0);
153         // Create a loop (from, from)
154         if (rooms.indexOf(from) == -1) {
155             rooms.add(from);
156             I.add(from);
157             J.add(from);
158             Q.add(0);
159             R.add((from == 0) ? reward: 0);
160         }
161         // Create a loop (to, to)
162         if (rooms.indexOf(to) == -1) {
163             rooms.add(to);
164             I.add(to);
165             J.add(to);
166             Q.add(0);
167             R.add((to == 0) ? reward : 0);
168         }
169     }
170 
171     public static void main(String[] args) {
172         // Read data file
173         try {
174             BufferedReader reader = new BufferedReader(new FileReader("Data2.txt"));
175             add(reader.readLine());
176             String line;
177             while ((line = reader.readLine()) != null)
178                 add(line);
179             System.out.println(I.toString());
180             System.out.println(J.toString());
181             System.out.println(Q.toString());
182             System.out.println(R.toString());
183             reader.close();
184         } catch (FileNotFoundException e) {
185             e.printStackTrace();
186         } catch (IOException e) {
187             e.printStackTrace();
188         }
189 
190         // Instantiate
191         for (int k = 0; k < rooms.size(); k++) {
192             strategies.add(new Strategy(rooms, I, J, Q, R, gamma));
193             // Add a monster to every room except the exit room
194             if (k > 0)
195                 strategies.elementAt(k).addMonster(k);
196         }
197 
198         // Train
199         for (int k = 0; k < rooms.size(); k++) {
200             System.out.println("Monster at: " + k);
201             for (int training = 1; training <= trainingSize; training++) {
202                 strategies.elementAt(k).train();
203             }
204             // Display strategy
205             strategies.elementAt(k).full(true, false);
206         }
207 
208         // Simulate - I/O
209 /*      int princessPosition = 0, monsterPosition = 0;
210         System.out.print("Enter the princess' starting position: ");
211         try {
212             princessPosition = Integer.parseInt(new BufferedReader(new
                    InputStreamReader(System.in)).readLine());
213         } catch (IOException e) {
214             e.printStackTrace();
215         }
216         System.out.print("Enter the monster's starting position: ");
217         try {
218             monsterPosition = Integer.parseInt(new BufferedReader(new
                    InputStreamReader(System.in)).readLine());
219         } catch (IOException e) {
220             e.printStackTrace();
221         }
222         System.out.println("The princess is in room " + princessPosition + ". The
                monster is in room " + monsterPosition + ".");
223         while (true) {
224             // If the princess escapes (room 0) or is captured, the round ends
225             if (princessPosition == 0) {
226                 System.out.println("The princess has escaped from the fortress.");
227                 break;
228             } else if (princessPosition == monsterPosition) {
```

```
                System.out.println("The princess has been captured by the monster.");
                break;
            }
            princessPosition = strategies.elementAt(monsterPosition).getOptimalMove(
                princessPosition);
            monsterPosition = strategies.elementAt(monsterPosition).getMonsterMove(
                monsterPosition);
            System.out.println("The princess is in room " + princessPosition + ".
                The monster is in room " + monsterPosition + ".");
        }*/

        // Simulate - Auto
        int princessPosition, monsterPosition;
        int[][] numEscapes = new int[rooms.size()][rooms.size()];
        for (int i = 1; i < rooms.size(); i++) {
            for (int j = 0; j < rooms.size(); j++) {
                // The princess and the monster start off in different rooms
                if (i != j) {
                    System.out.println("The princess is in room " + i + ". The monster
                        is in room " + j + ".");
                    for (int trial = 1; trial <= trialSize; trial++) {
                        princessPosition = i; monsterPosition = j;
                        while (true) {
                            // If the princess escapes (room 0) or is captured, the
                                round ends
                            if (princessPosition == 0) {
                                numEscapes[i][j]++;
                                break;
                            } else if (princessPosition == monsterPosition) {
                                break;
                            }
                            princessPosition = strategies.elementAt(monsterPosition).
                                getPrincessMove(princessPosition);
                            monsterPosition = strategies.elementAt(monsterPosition).
                                getMonsterMove(monsterPosition);
                        }
                    }
                }
            }
        }

        // Display number of escapes made
        for (int i = 0; i < rooms.size(); i++) {
            for (int j = 0; j < rooms.size(); j++) {
                if (i != 0 && i != j)
                    System.out.format("%05d ", numEscapes[i][j]);
                else
                    System.out.print("----- ");
            }
            System.out.println();
        }
    }
}
```

```
// INSTRUCTIONS: To run the game with two monsters, copy-paste the following four
    blocks into Princess.java and replace the original ones.
        // Instantiate
        for (int k = 0; k < rooms.size(); k++) {
            for (int l = 0; l < rooms.size(); l++) {
                strategies.add(new Strategy(rooms, I, J, Q, R, gamma));
                // Add a monster to every room except the exit room
                if (k > 0 && l > 0) {
                    strategies.elementAt(k * rooms.size() + l).addMonster(k);
                    strategies.elementAt(k * rooms.size() + l).addMonster(l);
                    System.out.println(k * rooms.size() + l);
                } else continue;
            }
        }

        // Train
        for (int k = 0; k < rooms.size(); k++) {
            for (int l = 0; l < rooms.size(); l++) {
                System.out.println("Monsters at: " + k + ", " + l);
                for (int training = 1; training <= trainingSize; training++) {
                    strategies.elementAt(k * rooms.size() + l).train();
                }
                // Display strategy
                strategies.elementAt(k * rooms.size() + l).full(true, false);
            }
        }
```

```
        // Simulate - Auto
        int princessPosition = 29, monsterPosition, monster2Position;
        int[][] numEscapes = new int[rooms.size()][rooms.size()];
        for (int i = 0; i < rooms.size(); i++) {
            for (int j = 0; j < rooms.size(); j++) {
                princessPosition = 29;
                // The princess and the monster start off in different rooms
                if (i != princessPosition && j != princessPosition) {
                    System.out.println("The princess is in room " + princessPosition +
                            ". The monsters are in rooms " + i + ", " + j + ".");
                    for (int trial = 1; trial <= trialSize; trial++) {
                        princessPosition = 29; monsterPosition = i; monster2Position =
                            j;
                        while (true) {
                            // If the princess escapes (room 0) or is captured, the
                                round ends
                            if (princessPosition == 0) {
                                numEscapes[j][k]++;
                                break;
                            } else if (princessPosition == monsterPosition ||
                                princessPosition == monster2Position) {
                                break;
                            }
                            princessPosition = strategies.elementAt(monsterPosition).
                                getPrincessMove(princessPosition);
                            monsterPosition = strategies.elementAt(monsterPosition).
                                getMonsterMove(monsterPosition);
                            monster2Position = strategies.elementAt(monsterPosition).
                                getMonsterMove(monster2Position);
                        }
                    }
                }
            }
        }

        // Display number of escapes made
        for (int i = 0; i < rooms.size(); i++) {
            for (int j = 0; j < rooms.size(); j++) {
                if (i != 29 && j != 29)
                    System.out.format("%05d ", numEscapes[i][j]);
                else
                    System.out.print("----- ");
            }
            System.out.println();
        }
```