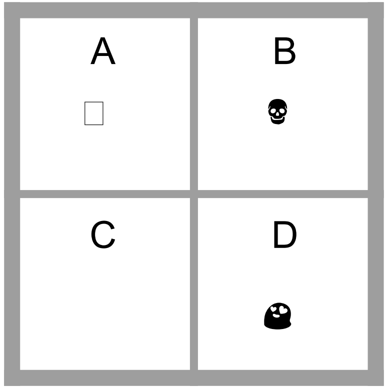
Reinforcement Learning Walk Throughs

These slides as a supplement for lecture 10.

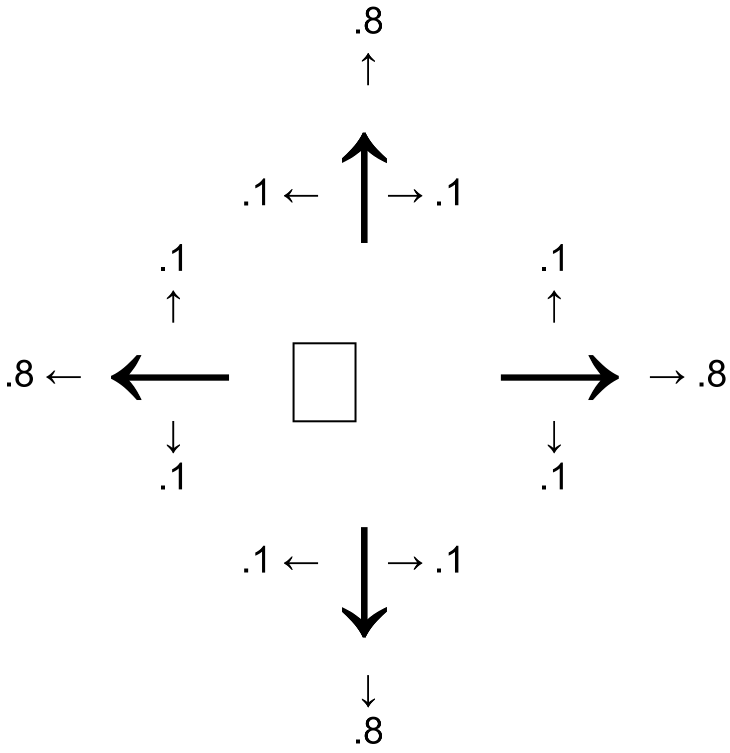
A Very Simple Maze Problem

Wobbie has to get to the winning square without dying.



A,B,C,D	Locations
	Wobbie's Start Position
	Wobbie Dies if he goes here
	Wobbie Wins if he gets here

Problem: Wobbie's Moves are Stochastic



So Wobbies will move 90 degrees to where he aims with .1 probability (in each direction.

If he moves into a wall (an impossible move outside of A,B,C,D) he stays still.

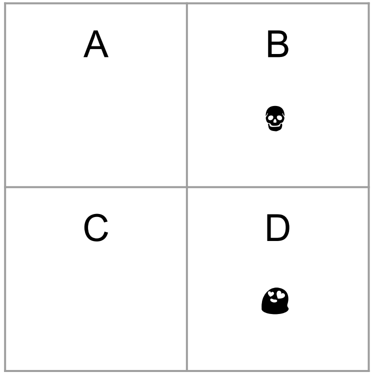
The Transition Function

S_t	Action	$P(S_{t+1})=A$	$P(S_{t+1})=B$	$P(S_{t+1})=C$	$P(S_{t+1})=D$
A	→	.1	.8	.1	0
A	←	.9	0	.1	0
A	↑	.9	.1	0	0
A	↓	.1	.1	.9	0
B	→	0	.9	0	.1
B	←	.8	.1	0	.1
B	↑	.1	.9	0	0
B	↓	.1	.1	0	.8
C	→	.1	0	.1	.8
C	←	.1	0	.9	0
C	↑	.8	0	.1	.1
C	↓	0	0	.9	.1
D	→	0	.1	0	.9
D	←	0	.1	.8	.1
D	↑	0	.8	.1	.1
D	↓	0	0	.1	.9

A	B
C	D

The Transition Function

We can simplify: Wobbie will never be able to make an action in state B or D since the game will be over (he will have died or won respectively).



S_t	Action	$P(S_{t+1})$=A	$P(S_{t+1})$=B	$P(S_{t+1})$=C	$P(S_{t+1})$=D
A	→	.1	.8	.1	0
A	←	.9	0	.1	0
A	↑	.9	.1	0	0
A	↓	.1	.1	.8	0
C	→	.1	0	.1	.8
C	←	.1	0	.9	0
C	↑	.8	0	.1	.1
C	↓	0	0	.9	.1

The Reward Function

Each non-terminal turn costs 1 (reward is -1). If Wobbie dies, reward is -100. If he wins it is 100.

State	Reward
A	-1
B	-100
C	-1
D	100

Value Iteration

Start with a utility function from state/action pairs to utility values, where all are 0.

(Utility = Expected Cumulative Discounted Reward given Optimal Policy)

State	Action	Expected Utility
A	→	0
A	←	0
A	↑	0
A	↓	0
B	Terminal	-100
C	→	0
C	←	0
C	↑	0
C	↓	0
D	Terminal	100

Value Iteration

State	Action	Expected Utility
A	→	0
A	←	0
A	↑	0
A	↓	0
C	→	0
C	←	0
C	↑	0
C	↓	0

Each iteration:

- Recalculate the utility value for each state/action pair based on the Bellman equation:

  $U(A,\rightarrow) = \Sigma_{s'} P(s'|s,\rightarrow)\,\big(R(s')+\gamma\, \max_{act} U(s',act)\big)$

  We'll assume $\gamma=1$.

Value Iteration: First Iteration, First Value

State	Action	Expected Utility
A	→	0
A	←	0
A	↑	0
A	↓	0
B	Terminal	-100
C	→	0
C	←	0
C	↑	0
C	↓	0
D	Terminal	100

$$U(A,\rightarrow) = \Sigma_{s'} P(s'|s,\rightarrow)\ (R(s')+\gamma\ \max_{act} U(s',act))$$

State	Reward
A	-1
B	-100
C	-1
D	100

S_t	Action	$P(S_{t+1})$=A	$P(S_{t+1})$=B	$P(S_{t+1})$=C	$P(S_{t+1})$=D
A	→	.1	.8	.1	0
A	←	.9	0	.1	0
A	↑	.9	.1	0	0
A	↓	.1	.1	.8	0
C	→	.1	0	.1	.8
C	←	.1	0	.9	0
C	↑	.8	0	.1	.1
C	↓	0	0	.9	.1

Value Iteration: First Iteration, First Value

State	Action	Expected Utility
A	→	0
A	←	0
A	↑	0
A	↓	0
B	Terminal	-100
C	→	0
C	←	0
C	↑	0
C	↓	0
D	Terminal	100

$U(A,\rightarrow) = \Sigma_{s'} P(s'|s,\rightarrow)\ (R(s')+\gamma\ \max_{act}\ U(s',act))$

$= (.1)(-1 + \gamma \cdot 0) +$
$(.8)(-100) +$
$(.1)(-1 + \gamma \cdot 0)$

$= -80.2$

State	Reward
A	-1
B	-100
C	-1
D	100

S_t	Action	$P(S_{t+1})=A$	$P(S_{t+1})=B$	$P(S_{t+1})=C$	$P(S_{t+1})=D$
A	→	.1	.8	.1	0
A	←	.9	0	.1	0
A	↑	.9	.1	0	0
A	↓	.1	.1	.8	0
C	→	.1	0	.1	.8
C	←	.1	0	.9	0
C	↑	.8	0	.1	.1
C	↓	0	0	.9	.1

Remember: $\gamma=1$
(Our future discount rate)

Value Iteration: First Iteration, First Value

State	Action	Expected Utility
A	→	-80.2
A	←	0
A	↑	0
A	↓	0
B	Terminal	-100
C	→	0
C	←	0
C	↑	0
C	↓	0
D	Terminal	100

Note that all expected utilites would be updated simultaneously: The 80.2 value would play no role in other calcuations.

State	Reward
A	-1
B	-100
C	-1
D	100

S_t	Action	$P(S_{t+1})$=A	$P(S_{t+1})$=B	$P(S_{t+1})$=C	$P(S_{t+1})$=D
A	→	.1	.8	.1	0
A	←	.9	0	.1	0
A	↑	.9	.1	0	0
A	↓	.1	.1	.8	0
C	→	.1	0	.1	.8
C	←	.1	0	.9	0
C	↑	.8	0	.1	.1
C	↓	0	0	.9	.1

Value Iteration: First Iteration

State	Action	Expected Utility
A	→	-80.2
A	←	-1
A	↑	-10.2
A	↓	-10.2
B	Terminal	-100
C	→	79.8
C	←	-1
C	↑	9.8
C	↓	9.8
D	Terminal	100

After the first iteration we have iterated back the effect of making one move from a state (acting optimally according to our utility function).

State	Reward
A	-1
B	-100
C	-1
D	100

S_t	Action	$P(S_{t+1})=A$	$P(S_{t+1})=B$	$P(S_{t+1})=C$	$P(S_{t+1})=D$
A	→	.1	.8	.1	0
A	←	.9	0	.1	0
A	↑	.9	.1	0	0
A	↓	.1	.1	.8	0
C	→	.1	0	.1	.8
C	←	.1	0	.9	0
C	↑	.8	0	.1	.1
C	↓	0	0	.9	.1

Value Iteration: Second Iteration, First Value

State	Action	Expected Utility
A	→	-80.2
A	←	-1
A	↑	-10.2
A	↓	-10.2
B	Terminal	-100
C	→	79.8
C	←	-1
C	↑	9.8
C	↓	9.8
D	Terminal	100

$$U(A,\rightarrow) = \Sigma_{s'} P(s'|s,\rightarrow)\ (R(s')+\gamma \max_{act} U(s',act))$$

$$= (.1)(-1 + \gamma \cdot -1) + (.8)(-100) + (.1)(-1 + \gamma \cdot 79.8)$$

$$= -72.33$$

State	Reward
A	-1
B	-100
C	-1
D	100

S_t	Action	$P(S_{t+1})$=A	$P(S_{t+1})$=B	$P(S_{t+1})$=C	$P(S_{t+1})$=D
A	→	.1	.8	.1	0
A	←	.9	0	.1	0
A	↑	.9	.1	0	0
A	↓	.1	.1	.8	0
C	→	.1	0	.1	.8
C	←	.1	0	.9	0
C	↑	.8	0	.1	.1
C	↓	0	0	.9	.1

Remember: γ=1
(Our future discount rate)

Value Iteration: Second Iteration

State	Action	Expected Utility
A	→	-80.24
A	←	-65.20
A	↑	-83.08
A	↓	44.92
B	Terminal	-100
C	→	79.76
C	←	62.80
C	↑	-45.08
C	↓	80.92
D	Terminal	100

After the second iteration we have iterated back the effect of making two moves from a state (acting optimally according to our utility function).

State	Reward
A	-1
B	-100
C	-1
D	100

S_t	Action	$P(S_{t+1})=A$	$P(S_{t+1})=B$	$P(S_{t+1})=C$	$P(S_{t+1})=D$
A	→	.1	.8	.1	0
A	←	.9	0	.1	0
A	↑	.9	.1	0	0
A	↓	.1	.1	.8	0
C	→	.1	0	.1	.8
C	←	.1	0	.9	0
C	↑	.8	0	.1	.1
C	↓	0	0	.9	.1

Episodic Q-Learning

Using the implied policy of our Q-model, perform whole 'runs' and update Q-model based on observed sequences of rewards.

The 'implied policy' is that we do whatever action has the highest Q-value for the current state.

State **S**	**Action** **A**	**Reward** **R**	**New State** **S´**
Game One			
A	↓	-1	C
C	→	100 (T)	D
Game Two			
A	→	-100 (T)	B
Game Three			
A	←	-1	C
C	↑	-1	A
A	↓	-1	C
C	→	100 (T)	D

Episodic Q-Learning

State	Action	Q (Expected Utility)
A	→	0
A	←	0
A	↑	0
A	↓	0
B	Terminal	-100
C	→	0
C	←	0
C	↑	0
C	↓	0
D	Terminal	100

1. **Start with arbitrary Q-model.**
   The table to the left.

Repeat for each run:

2. Calculate cumulative discounted rewards for each move.
3. Adjust Q-model.

Episodic Q-Learning

State	Action	Q (Expected Utility)
A	→	0
A	←	0
A	↑	0
A	↓	0
B	Terminal	-100
C	→	0
C	←	0
C	↑	0
C	↓	0
D	Terminal	100

1. Start with arbitrary Q-model.

Repeat for each run:

2. **Calculate cumulative discounted rewards for each move.**
3. Adjust Q-model.

Episodic Q-Learning

Game One:

- $CDR(A,\downarrow) = -1 + \gamma \cdot 100$
- $CDR(C,\rightarrow) = 100$

If $\gamma=1$:

- $CDR(A,\downarrow) = -1 + 100 = 99$
- $CDR(C,\rightarrow) = 100$

(If state/action pairs occur more than once, calculate the CDR for each occurance and average.)

State S	**Action A**	**Reward R**	**New State S´**
Game One			
A	↓	-1	C
C	→	100 (T)	D
Game Two			
A	→	-100 (T)	B
Game Three			
A	←	-1	C
C	↑	-1	A
A	↓	-1	C
C	→	100 (T)	D

Episodic Q-Learning

State	Action	Q (Expected Utility)
A	→	0
A	←	0
A	↑	0
A	↓	0
B	Terminal	-100
C	→	0
C	←	0
C	↑	0
C	↓	0
D	Terminal	100

1. Start with arbitrary Q-model.

Repeat for each run:

2. **Calculate cumulative discounted rewards for each move.**

   For game one we have:

   - **CDR(A,↓) = 99**
   - **CDR(C,→) = 100**

3. Adjust Q-model.

Episodic Q-Learning

State	Action	Q (Expected Utility)
A	→	0
A	←	0
A	↑	0
A	↓	0
B	Terminal	-100
C	→	0
C	←	0
C	↑	0
C	↓	0
D	Terminal	100

1. Start with arbitrary Q-model.

Repeat for each run:

2. Calculate cumulative discounted rewards for each move.

   For game one we have:

   - CDR(A,↓) = 99
   - CDR(C,→) = 100

3. **Adjust Q-model.**

   Simplest to use a learning rate, λ:

   Q(S,A)=Q(S,A)-λ(Q(S,A)-CDR(S,A))

Episodic Q-Learning

State	Action	Q (Expected Utility)
A	→	0
A	←	0
A	↑	0
A	↓	.99
B	Terminal	-100
C	→	1
C	←	0
C	↑	0
C	↓	0
D	Terminal	100

1. Start with arbitrary Q-model.

Repeat for each run:

2. Calculate cumulative discounted rewards for each move.

   For game one we have:

   - CDR(A,↓) = 99
   - CDR(C,→) = 100

3. **Adjust Q-model.**
   - **Q(A,↓)=0 - .01(0-99)= .99**
   - **Q(C,→)=0 - .01(0-100) = 1**

Episodic Q-Learning

Three issues:

- By using the implied policy of the Q-model we quickly hit low-quality local optima: We need to do more ‘exploration’!
- We don’t really want to run whole episodes before tuning our Q-model. This is very slow and in real problems normally impractical.
- We don’t take into account the chained nature of the Q values given by the Bellman equations.
    - Well… we do in the sense that they rewards obtained in the episodes run are so chained. But we don’t make use of this when updating the Q-model.

ε-Q-Learning

Issue One: By using the implied policy of the Q-model we quickly hit low-quality local optima: We need to do more ‘exploration’!

To add exploration into our Q-Learning, we can introduce randomness: When we make an action we *either*:

- Act according the implied policy (with probability 1-ε).
- Act randomly (with probability ε).

To ensure convergence to an optimal policy in the long run, ε should approach 0 as the number of actions taken approaches infinity. So we specify some schedule for ε. Some examples (more sophisticated alternatives are possible):

- Start at .1 and decrease by .001 every 10 actions.
- Start at .6 and decrease by .01 every episode.

SARS´ ε-Q-Learning

Issue Two: We don’t really want to run whole episodes before tuning our Q-model. This is very slow and in real problems normally impractical.

Issue Three: We don’t take into account the chained nature of the Q values given by the Bellman equations.

We can deal with both of these issues by moving to SARS´ (ε-)Q-learning.

SARS´=S(tate) A(ction) R(eward) (Next) S(tate).

> SARS is an example of an ‘off-policy temporal difference’ Q-Learning algorithm. As this suggests there are other options for solving these issues, but SARS is the most used.

SARS´ ε-Q-Learning

Using ε-Q-learning to decide actions to perform, perform ‘runs’ and update Q-model based on observed SARS´ quadtuples.

State **S**	**Action** **A**	**Reward** **R**	**New State** **S´**
	Game One		
A	↓	-1	C
C	→	100 (T)	D
	Game Two		
A	→	-100 (T)	B
	Game Three		
A	←	-1	C
C	↑	-1	A
A	↓	-1	C
C	→	100 (T)	D

SARS´ ε-Q-Learning

State	Action	Q (Expected Utility)
A	→	0
A	←	0
A	↑	0
A	↓	0
B	Terminal	-100
C	→	0
C	←	0
C	↑	0
C	↓	0
D	Terminal	100

1. **Start with arbitrary Q-model.**

The table to the left.

Repeat for each action in each run:

2. Estimate error of Q-model Q-estimation.
3. Adjust Q-model.

SARS´ ε-Q-Learning

State	Action	Q (Expected Utility)
A	→	0
A	←	0
A	↑	0
A	↓	0
B	Terminal	-100
C	→	0
C	←	0
C	↑	0
C	↓	0
D	Terminal	100

1. Start with arbitrary Q-model.

Repeat for each action in each run:

2. **Estimate error of Q-model Q-estimation.**
3. Adjust Q-model.

SARS´ ε-Q-Learning

State S	Action A	Reward R	New State S´
Game One			
A	↓	-1	C
C	→	100 (T)	D

Game One, Move One:

Compare the following:

1. $Q(A,\downarrow) = 0$
2. $Q´(A,\downarrow) = -1 + \gamma \cdot \max_{act} Q(C,act)$

(1) is our current estimate of $Q(A,\downarrow)$. (2) gives $Q´(A,\downarrow)$ which is our estimate of the expected cumulated discounted reward we will get in this game from now performing ↓ in state A, since we end up in state C.

- (2) exploits the Bellman equation for Q-values
- (2) makes use of the Q-model in its estimation

Episodic Q-Learning

State S	Action A	Reward R	New State S´
Game One			
A	↓	-1	C
C	→	100 (T)	D

Game One, Move One:

Compare the following:

1. $Q(A,\downarrow) = 0$
2. $Q´(A,\downarrow) = -1 + \gamma \cdot \max_{act} Q(C,act)$

We can update our estimate of $Q(A,\downarrow)$ based on $Q´(A,\downarrow)$.

Although we make use of the Q-model in calculating $Q´(A,\downarrow)$, objective information is also used:

- When we made move ↓ in state A, we did end up in state C.
- When we made move ↓ in state A, we did get an immediate reward of -1.

SARS´ ε-Q-Learning

State	Action	Q (Expected Utility)
A	→	0
A	←	0
A	↑	0
A	↓	0
B	Terminal	-100
C	→	0
C	←	0
C	↑	0
C	↓	0
D	Terminal	100

1. Start with arbitrary Q-model.

Repeat for each action in each run:

2. Estimate error of Q-model Q-estimation.
3. **Adjust Q-model.**

   Simplest to use a learning rate, λ:
   Q(S,A)=Q(S,A)-λ(Q(S,A)-Q´(S,A))

Episodic Q-Learning

Game One, Move One:

Compare the following:

1. $Q(A,\downarrow) = 0$
2. $Q'(A,\downarrow) = -1 + \gamma \cdot \max_{act} Q(C,act) = -1$

We are assuming $\gamma=1$.

State S	Action A	Reward R	New State S´
Game One			
A	↓	-1	C
C	→	100 (T)	D

State	Action	Q (Expected Utility)
A	→	0
A	←	0
A	↑	0
A	↓	0
B	Terminal	-100
C	→	0
C	←	0
C	↑	0
C	↓	0
D	Terminal	100

SARS´ ε-Q-Learning

State	Action	Q (Expected Utility)
A	→	0
A	←	0
A	↑	0
A	↓	-.01
B	Terminal	-100
C	→	0
C	←	0
C	↑	0
C	↓	0
D	Terminal	100

1. Start with arbitrary Q-model.

Repeat for each action in each run:

2. Estimate error of Q-model Q-estimation.
3. **Adjust Q-model.**

   Simplest to use a learning rate, λ:
   Q(A,↓)=0-λ(0 - -1)=-0.01

   We assume λ=.01

Function (non-tabular) SARS´ ε-Q-Learning

If our Q-model is a parameterized (non-tabular) function we just do a single step of gradient descent.

(Actually that is what we are doing with the parameters of the tabular function too.)

If our Q-model is a deep neural network, we are doing deep reinforcement learning.

You should perform SARS´ ε-Q-Learning using a table in project 5. 😊