



Reinforcement Learning

a gentle introduction & industrial application

Dr. Christian Hidber

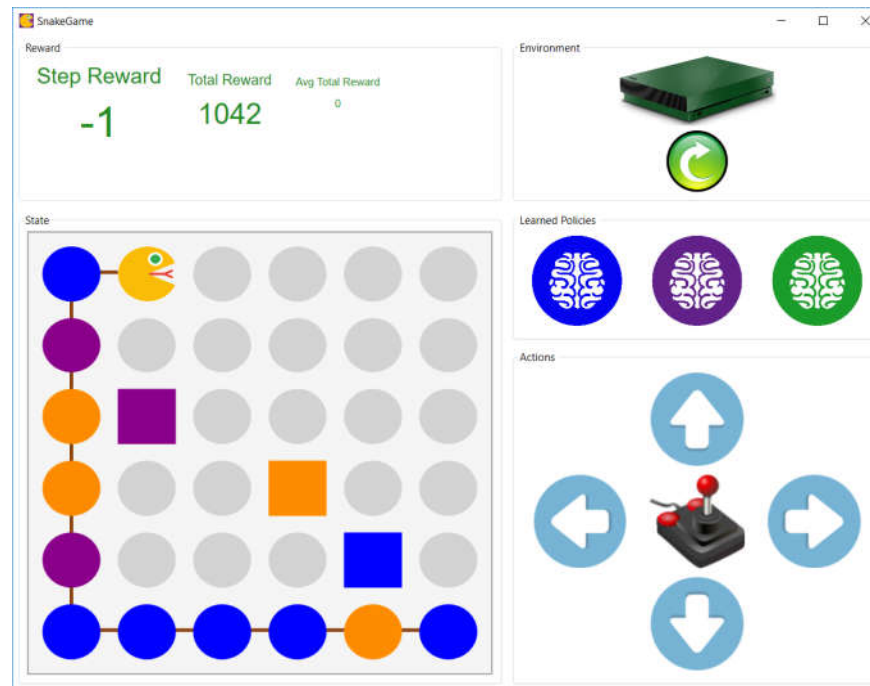
Learning learning from children



FOLIE 2
REINFORCEMENT LEARNING

 **GEBERIT**

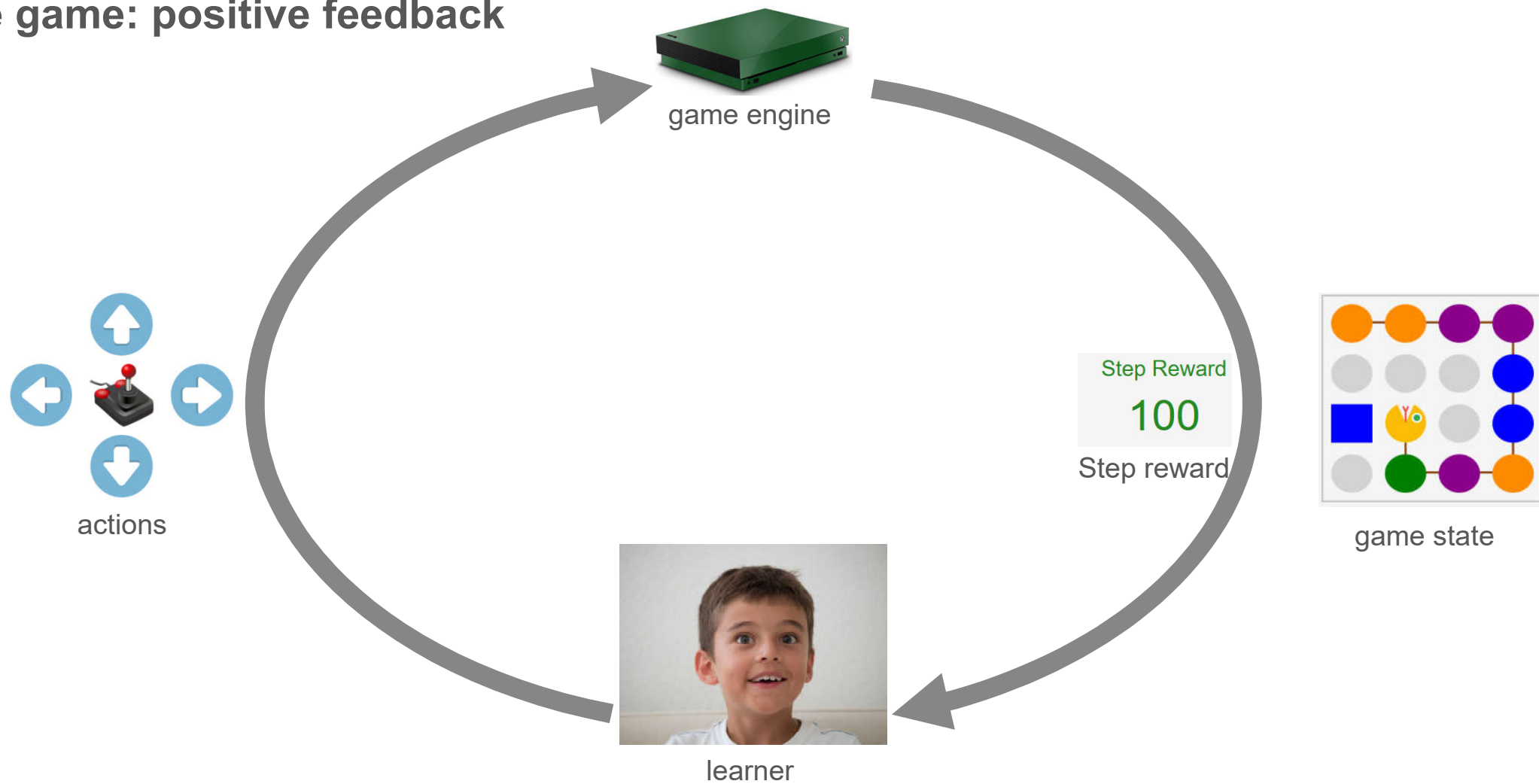
The game: demo



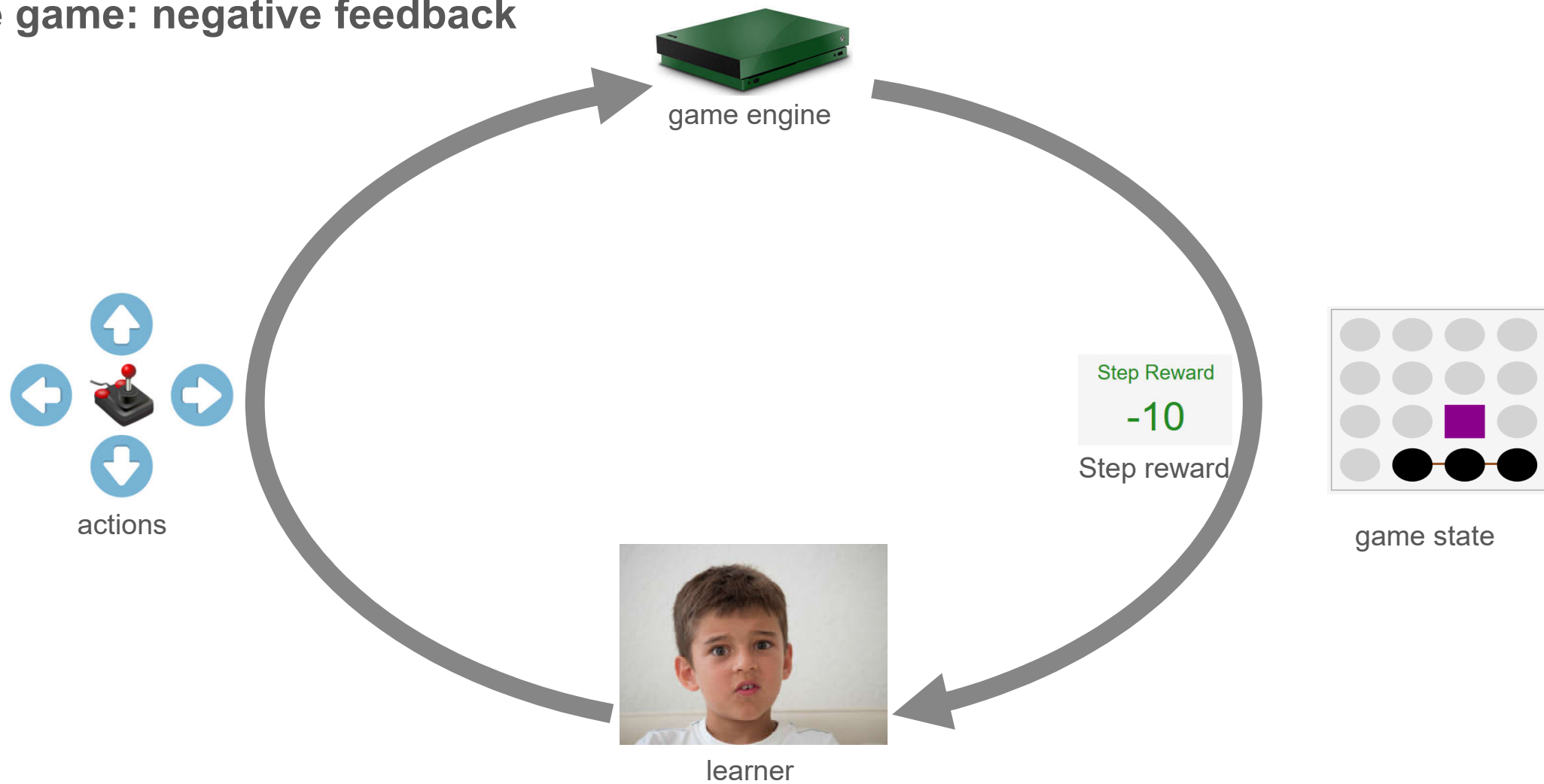
The game: setup



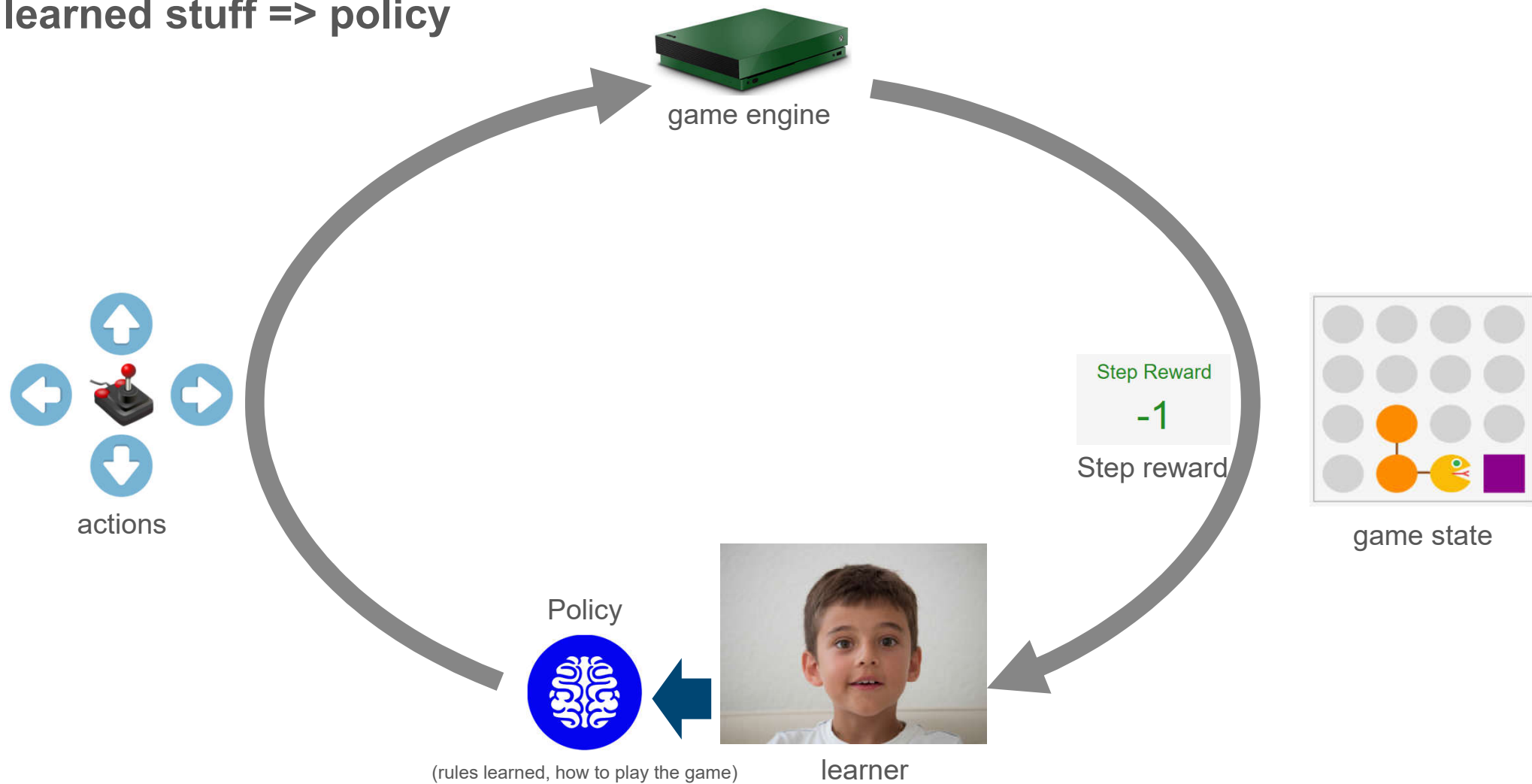
The game: positive feedback



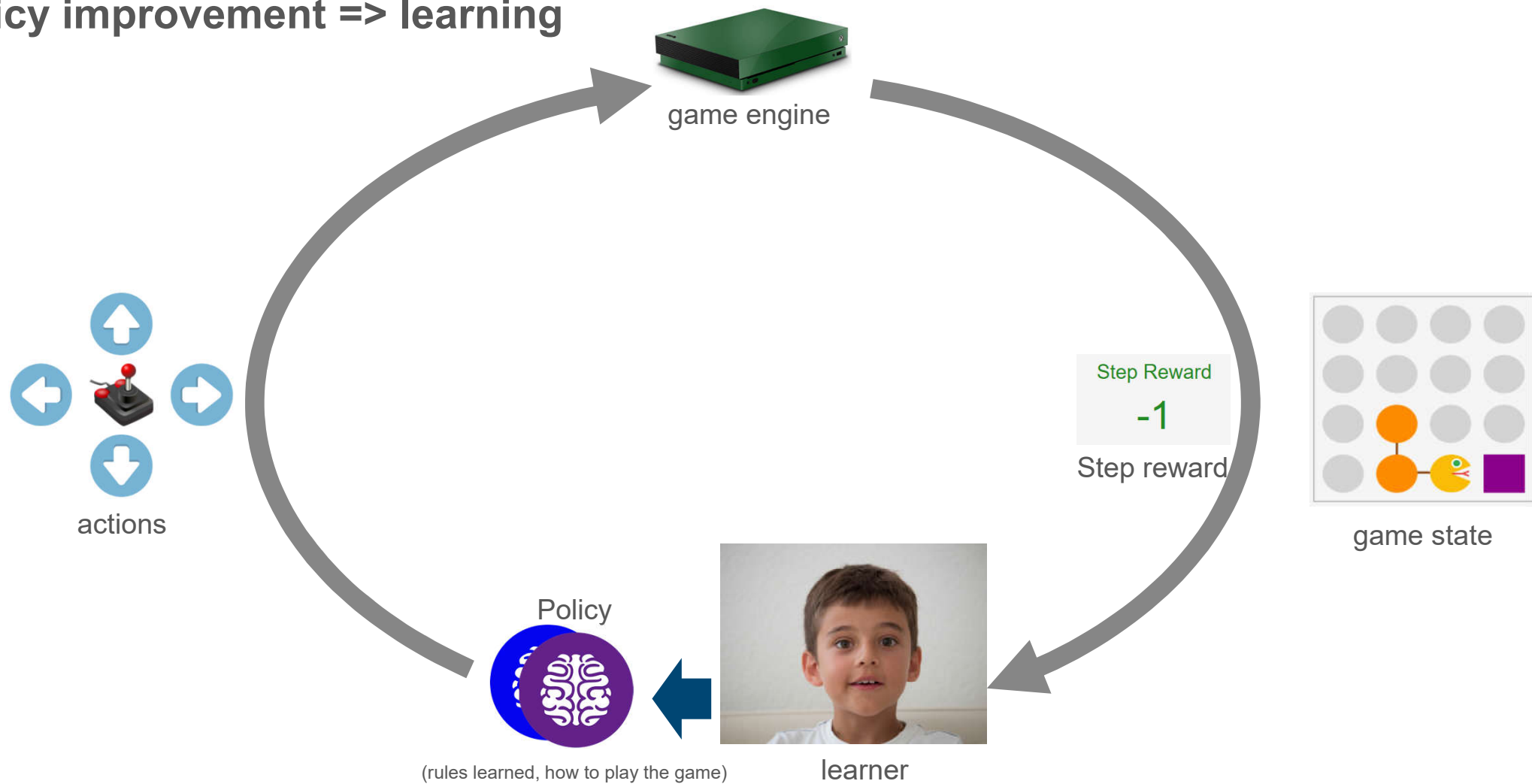
The game: negative feedback



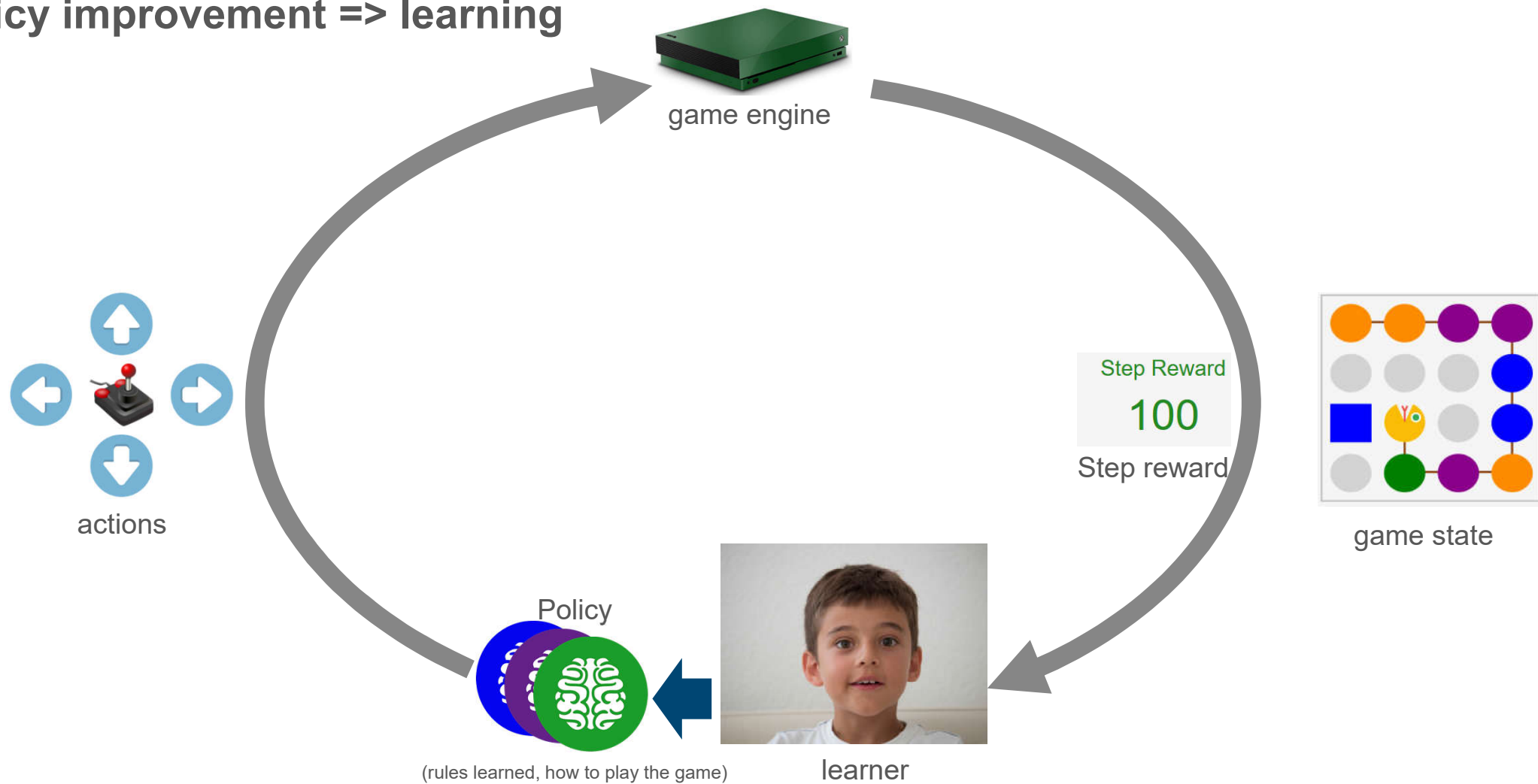
the learned stuff => policy



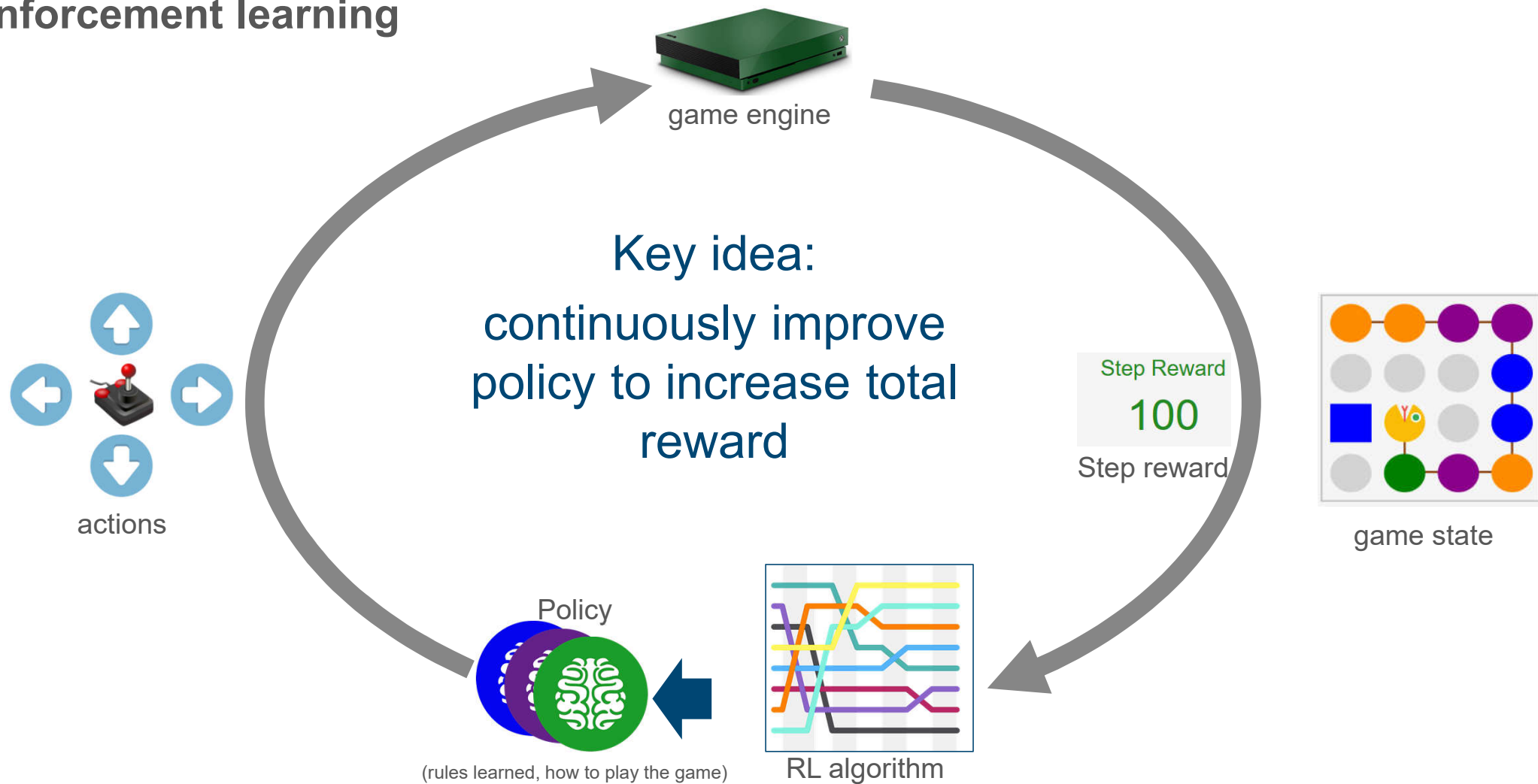
policy improvement => learning



policy improvement => learning



Reinforcement learning



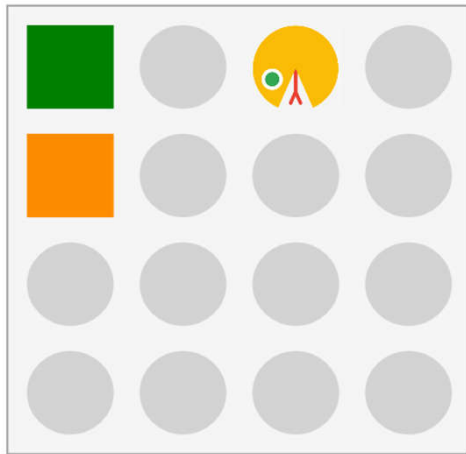
Episode 1 : play with 1st policy (random)



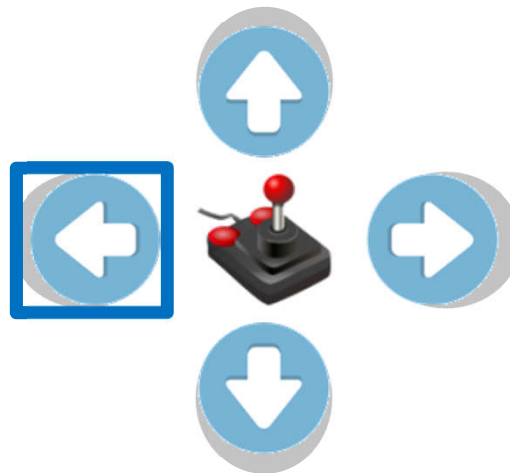
Policy

(rules learned, how to play the game)

State



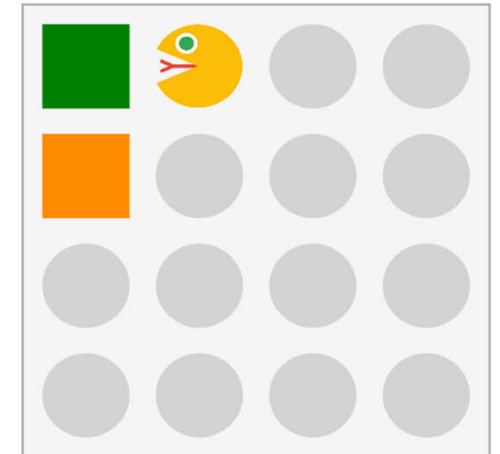
Action from
Policy



Reward

-1

Next State



1

2

3

4

5

6

7 Step #

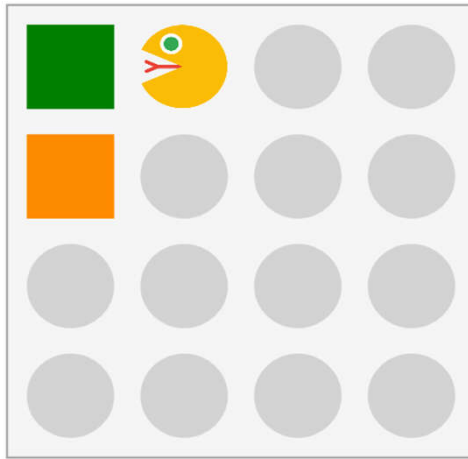
Episode 1 : play with 1st policy (random)



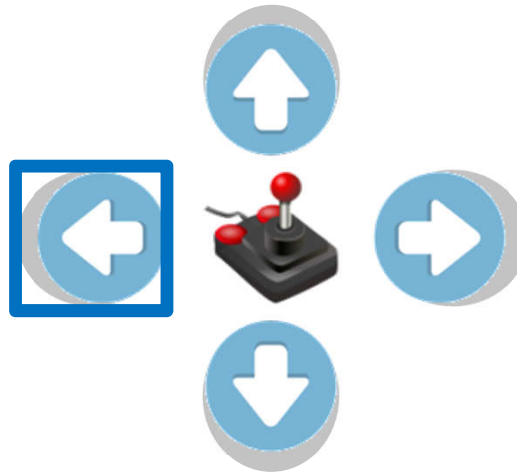
Policy

(rules learned, how to play the game)

State



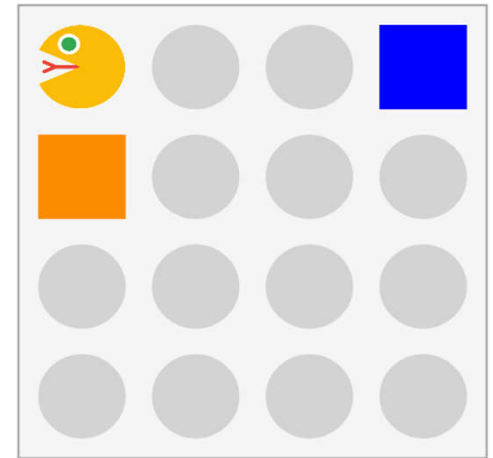
Action from
Policy



Reward

100

Next State



1

2

3

4

5

6

7 Step #

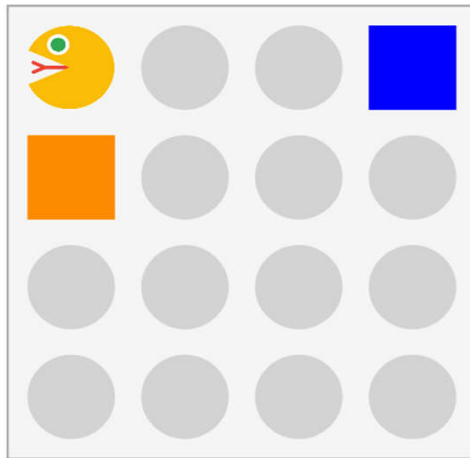
Episode 1 : play with 1st policy (random)



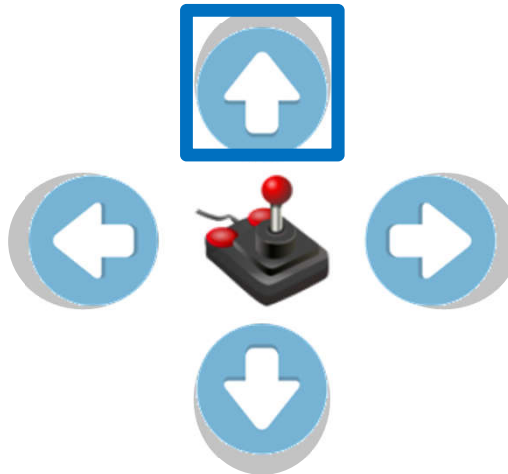
Policy

(rules learned, how to play the game)

State



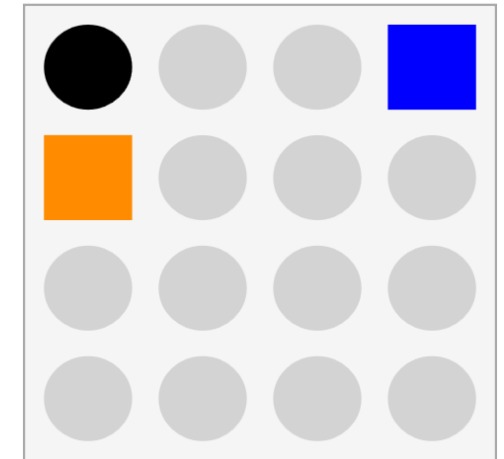
Action from
Policy



Reward

-50

Next State



Episode Over

Episode Over

1

2

3

4

5

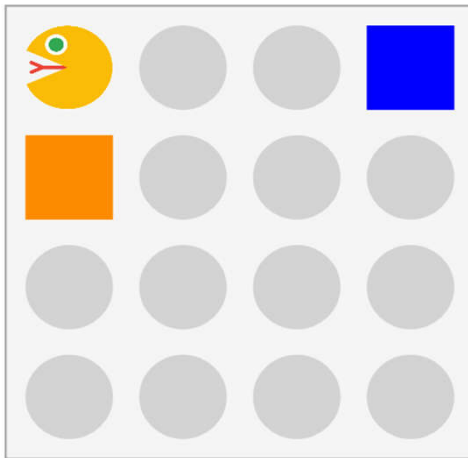
6

7

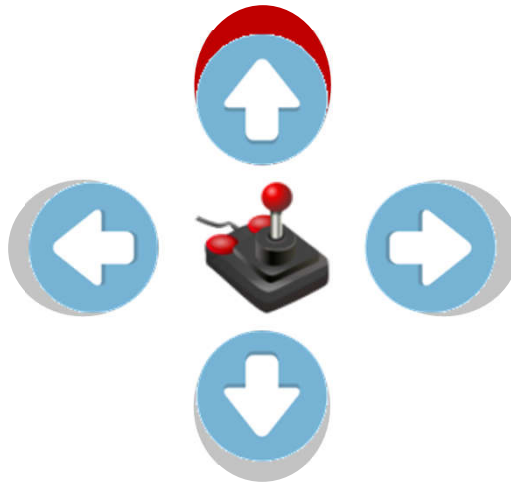
Step #

Episode 1 : improve 1st policy for state in step 3

State



Action from
Policy



-50

Episode Over



1

2

3

4

5

6

7 Step #

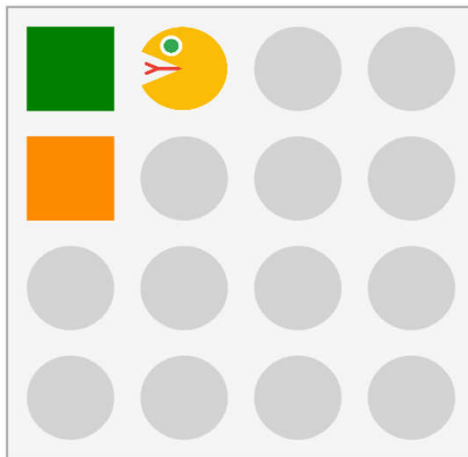
Episode 1 : improve 1st policy for state in step 2



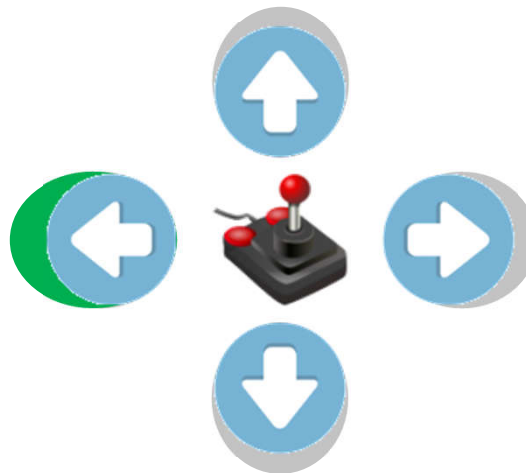
Policy

(rules learned, how to play the game)

State



Action from
Policy



50 (=+100 -50)

Future Reward

(sum of all rewards from current state until
'game over')

Episode Over



1

2

3

4

5

6

7 Step #

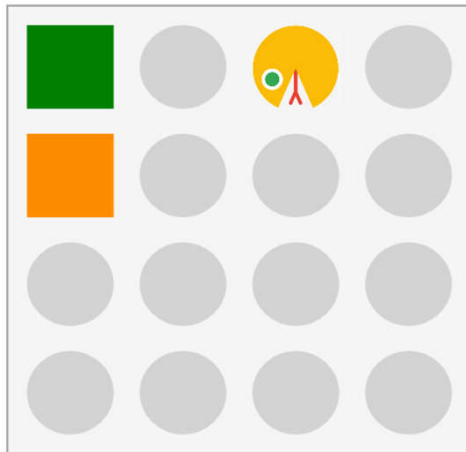
Episode 1 : improve 1st policy for state in step 1



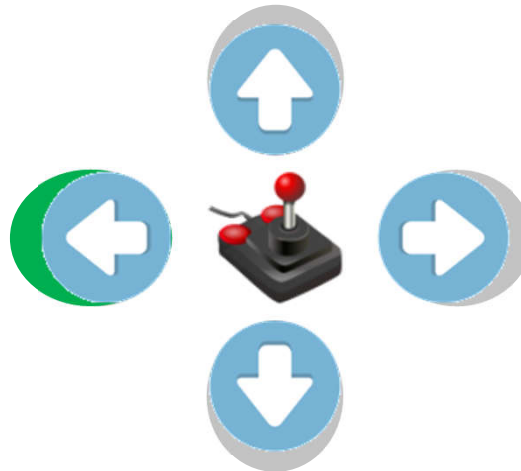
Policy

(rules learned, how to play the game)

State



Action from
Policy



49 (=-1 +100 -50)

Future Reward

(sum of all rewards from current state until
'game over')



Episode Over

1

2

3

4

5

6

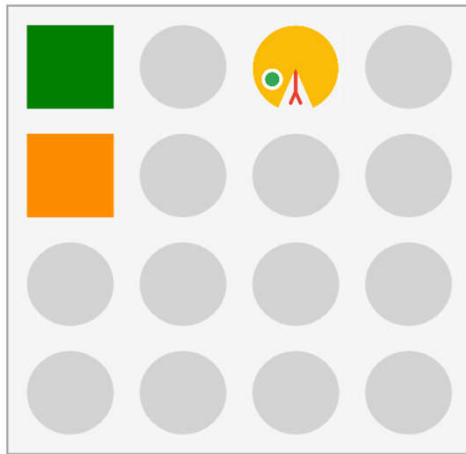
7 Step #

Episode 2 : play with 2nd policy

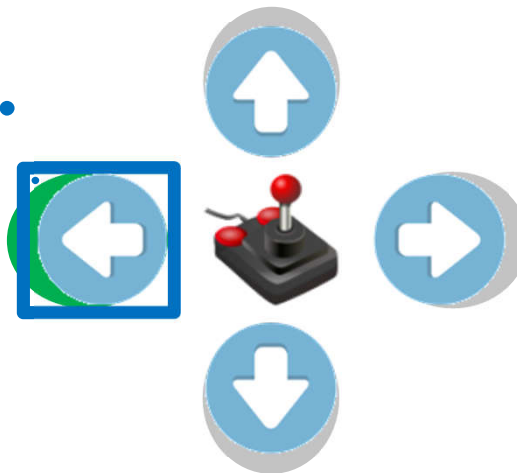

Policy
(rules learned, how to play the game)

Already learned:
go left is ok

State



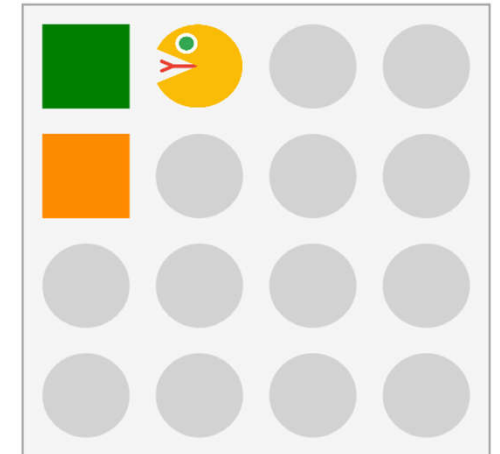
Action from
Policy



Reward

-1

Next State



1

2

3

4

5

6

7 Step #

Episode 2 : play with 2nd policy

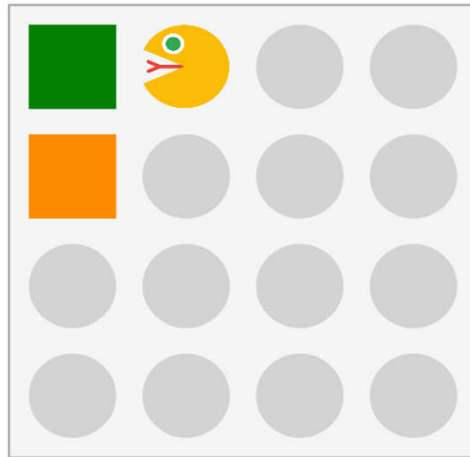


Policy

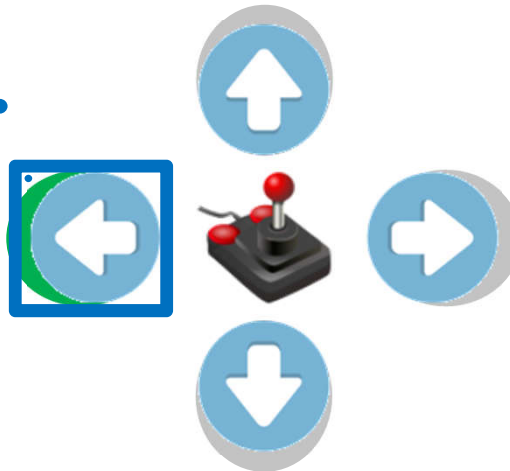
(rules learned, how to play the game)

Already learned:
go left is ok

State



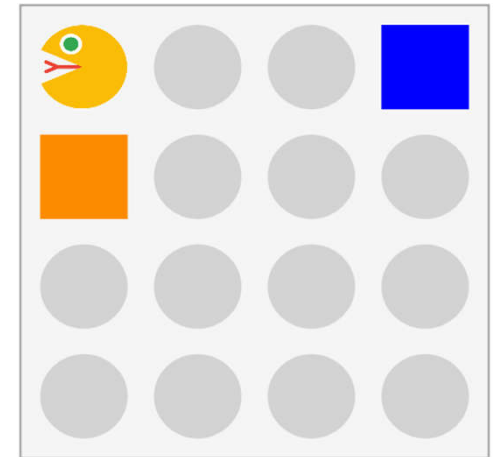
Action from
Policy



Reward

100

Next State



1

2

3

4

5

6

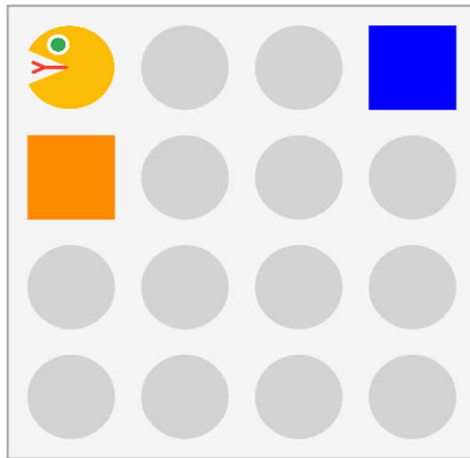
7 Step #

Episode 2 : play with 2nd policy

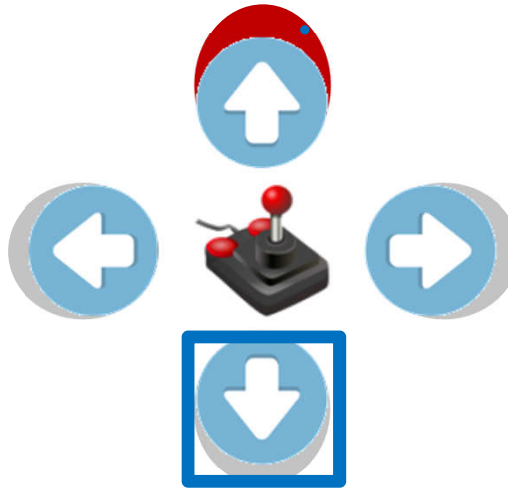
Already learned:
don't go up


Policy
(rules learned, how to play the game)

State



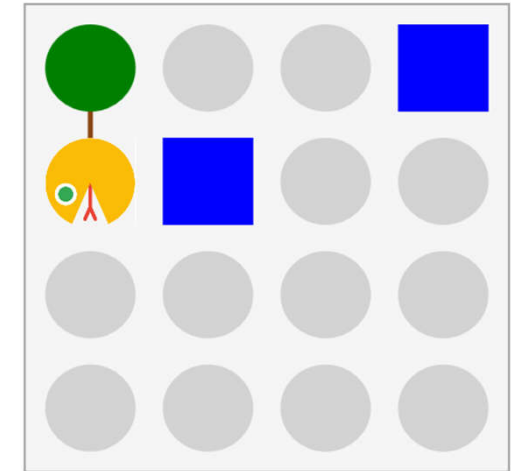
Action from
Policy



Reward

100

Next State



1

2

3

4

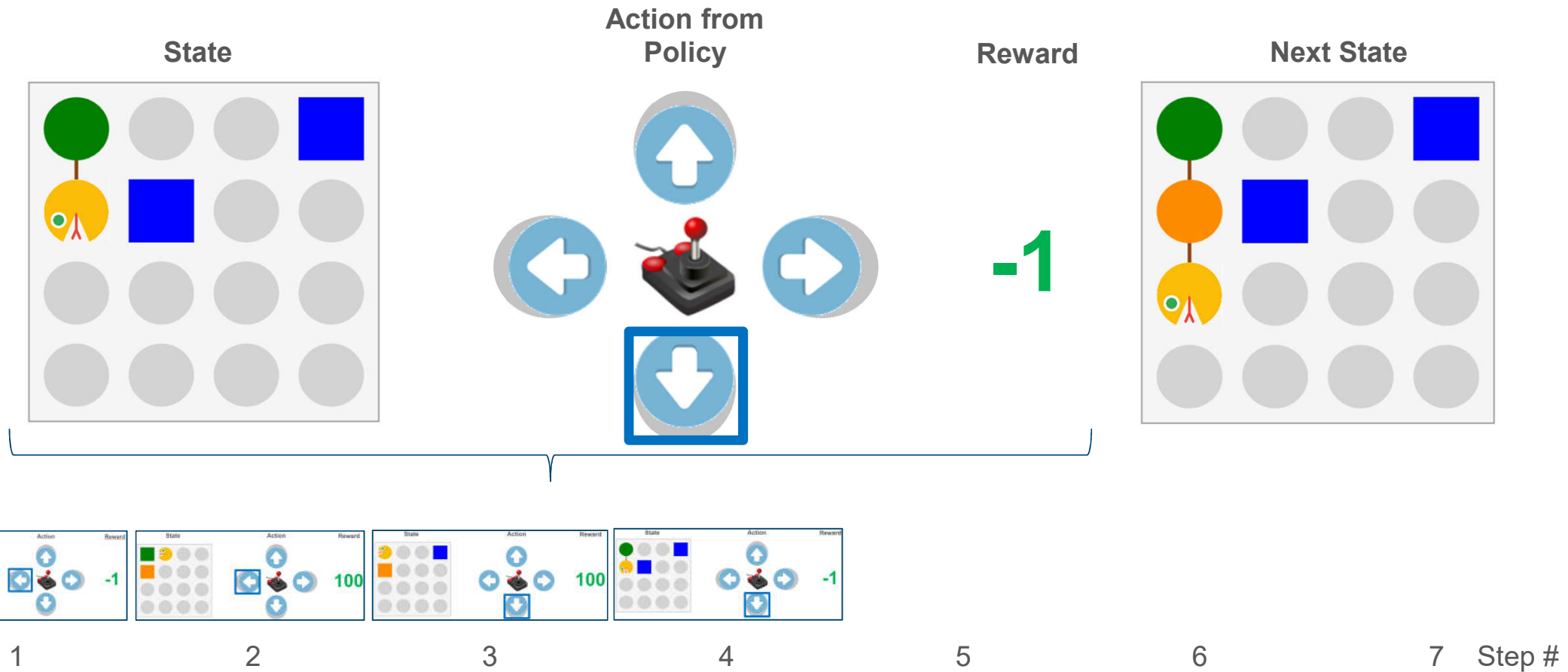
5

6

7 Step #

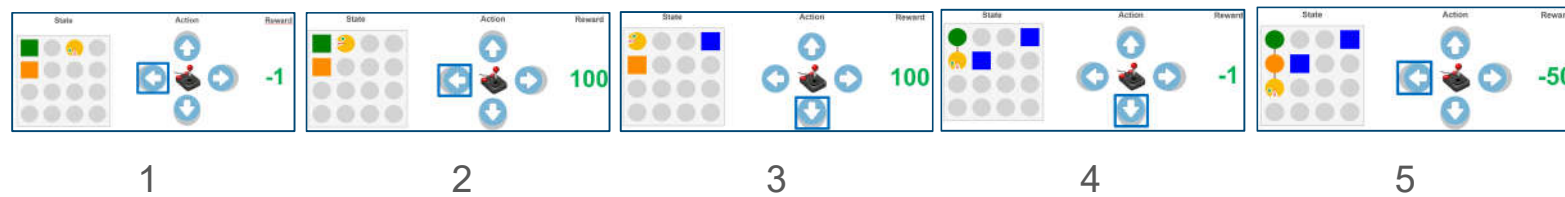
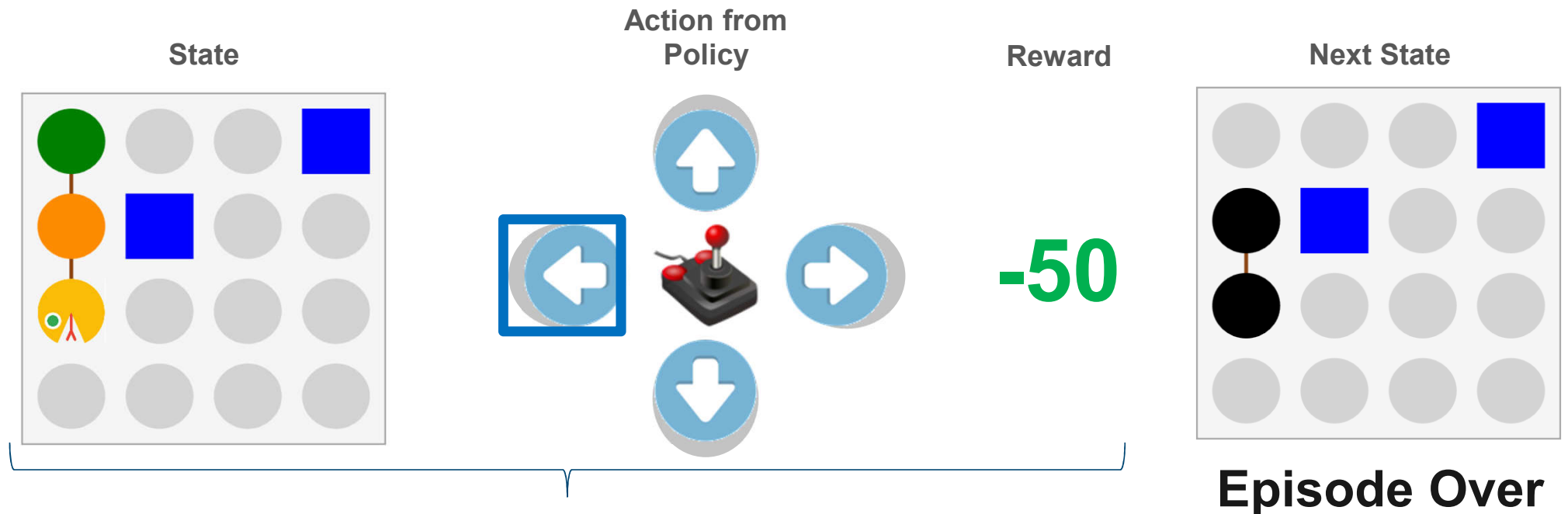
Episode 2 : play with 2nd policy


Policy
(rules learned, how to play the game)



Episode 2 : play with 2nd policy


Policy
(rules learned, how to play the game)

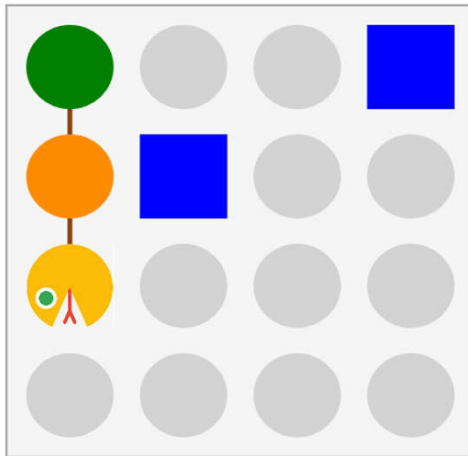


Episode Over

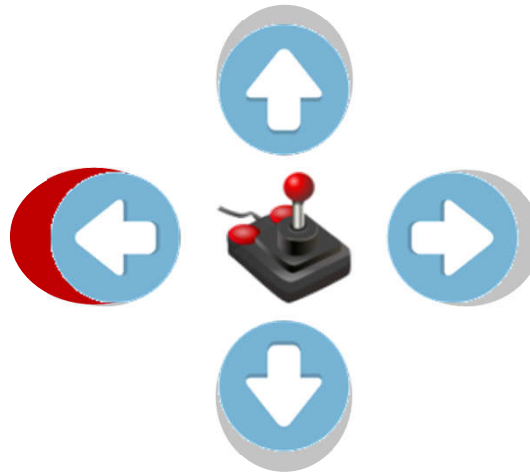
6 7 Step #

Episode 2 : improve 2nd policy for state in step 5

State



Action from
Policy

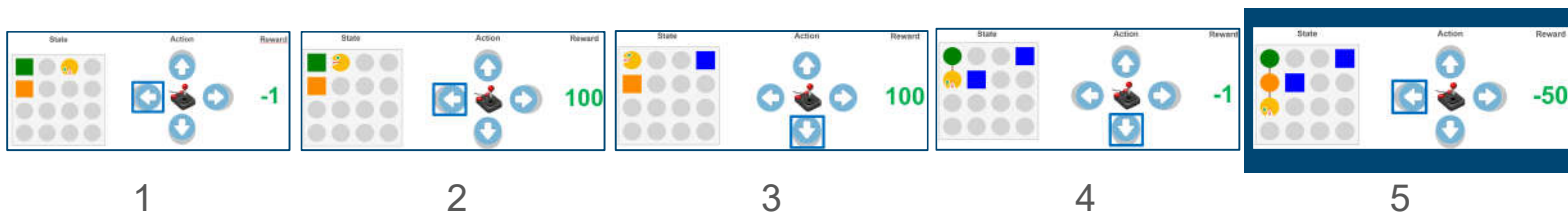


-50 (= -50)

Future Reward

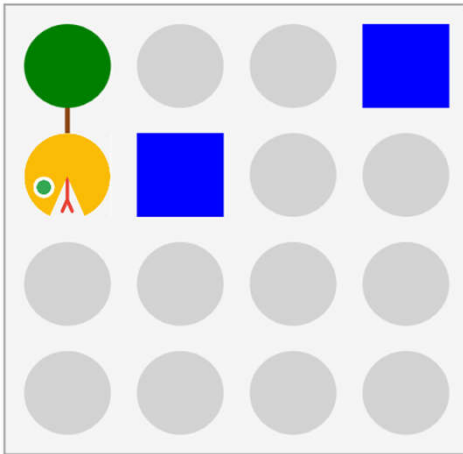
(sum of all rewards from current state until
'game over')

Episode Over

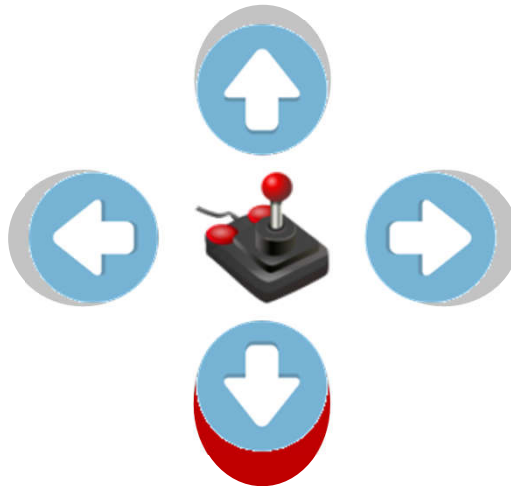


Episode 2 : improve 2nd policy for state in step 4

State



Action from
Policy



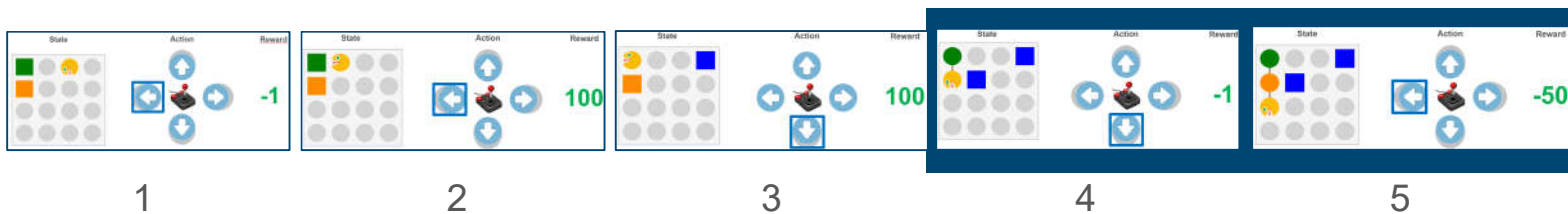
-51 (= -1 -50)

Future Reward

(sum of all rewards from current state until
'game over')

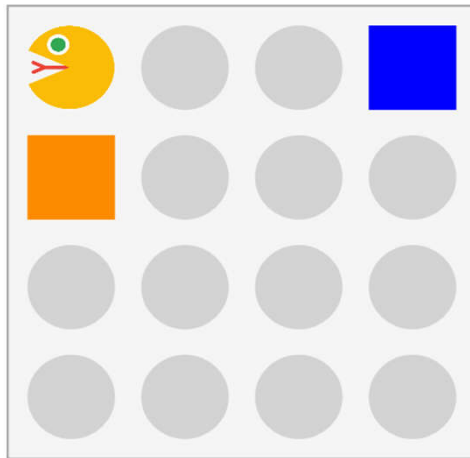
Episode Over

7 Step #

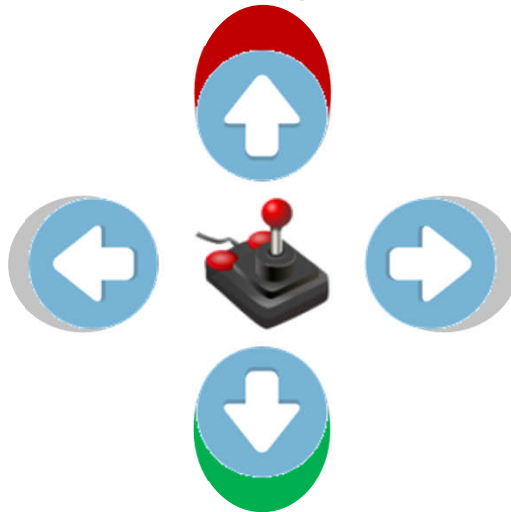


Episode 2 : improve 2nd policy for state in step 3

State



Action from
Policy



49 $(=+100-1-50)$

Future Reward

(sum of all rewards from current state until
'game over')

Episode Over

7 Step #

1

2

3

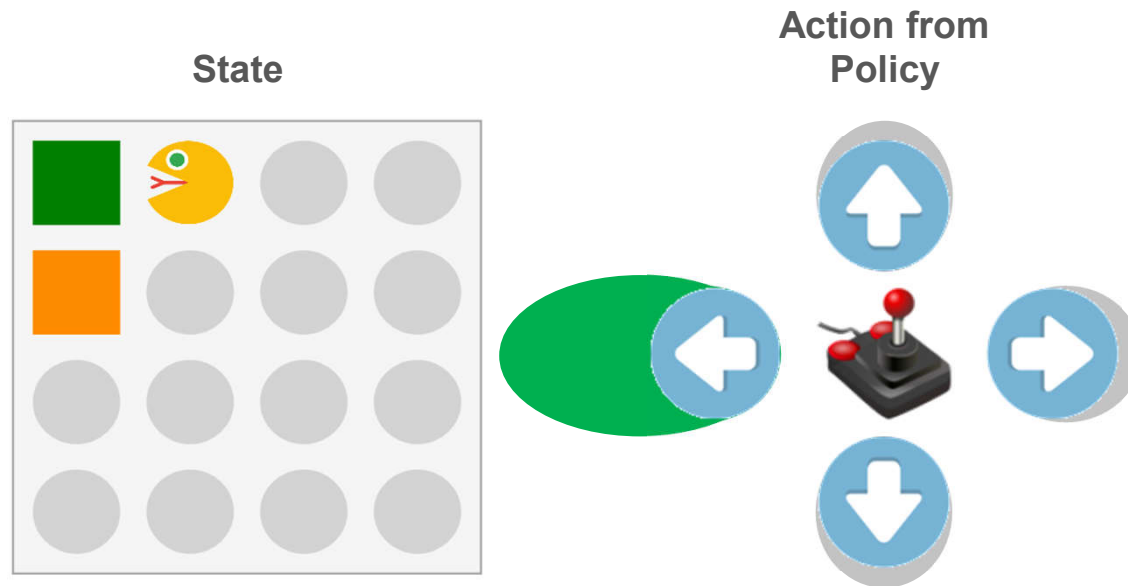
4

5

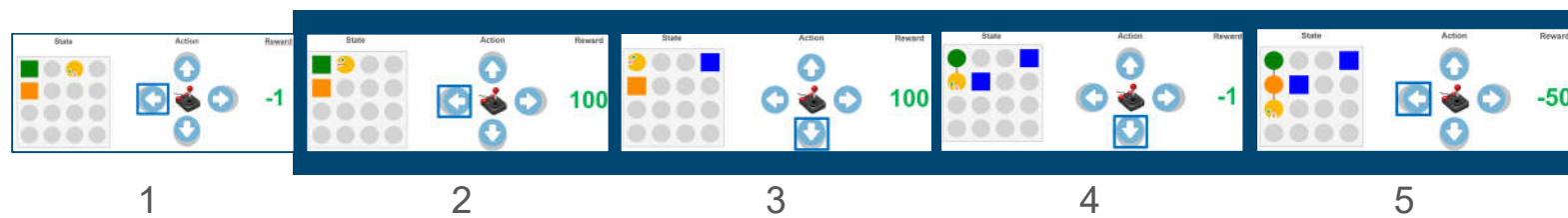
6



Episode 2 : improve 2nd policy for state in step 2

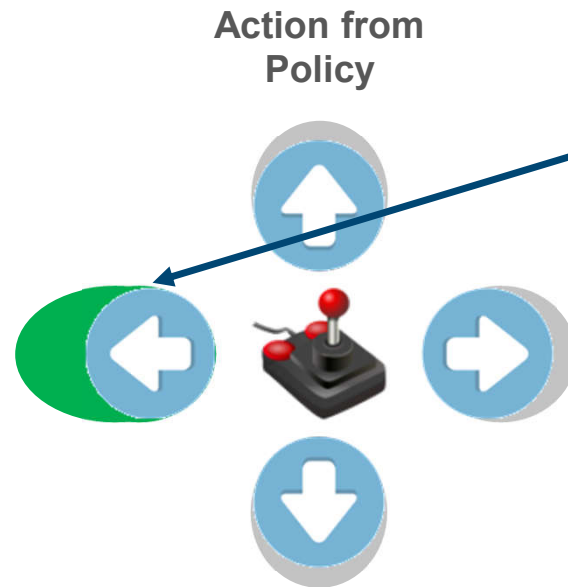
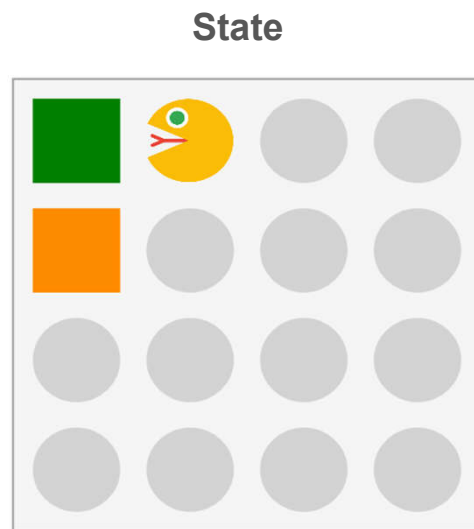


149 $(=+100+100-1-50)$
Future Reward
(sum of all rewards from current state until 'game over')



Episode Over

Episode 2 : improve 2nd policy for state in step 2

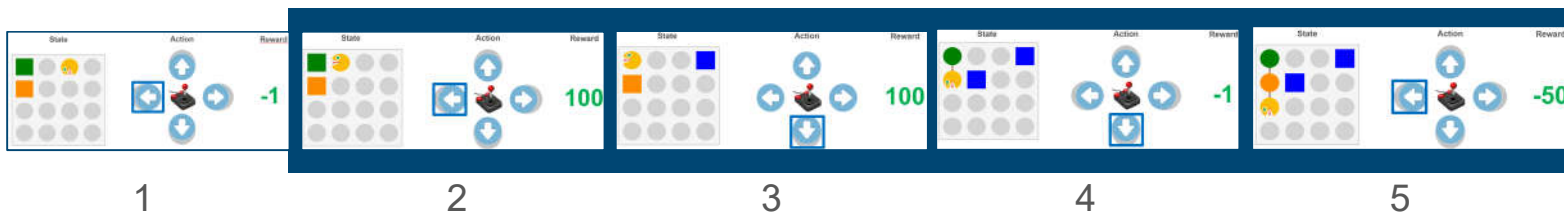


= some running average
of old and new value

149 $(=+100+100-1-50)$

Future Reward

(sum of all rewards from current state until
'game over')

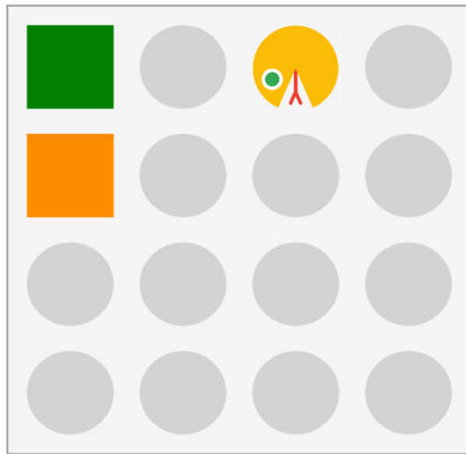


Episode Over

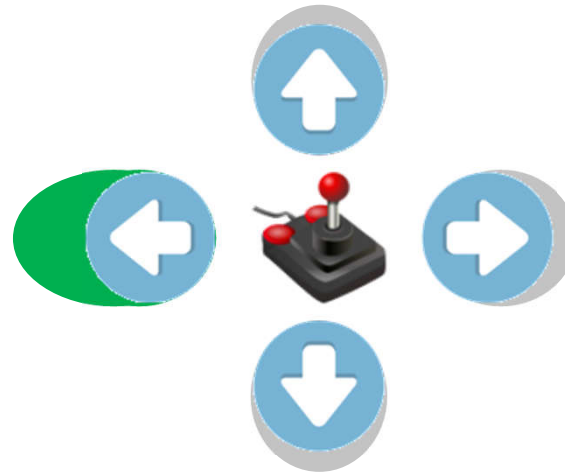
7 Step #

Episode 2 : improve 2nd policy for state in step 1

State



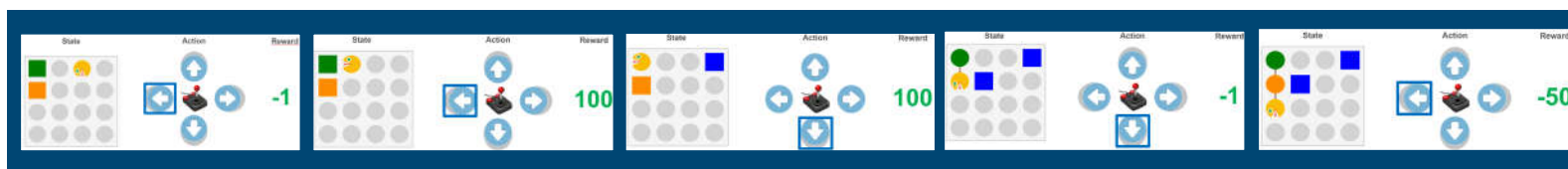
Action from
Policy



148 $(=-1+100+100-1-50)$

Future Reward

(sum of all rewards from current state until
'game over')



1

2

3

4

5

6

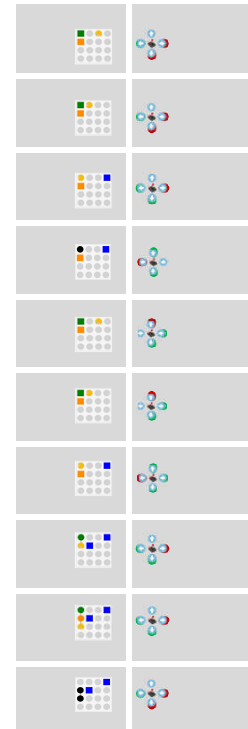
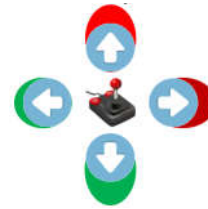
7 Step #

Episode Over

So far



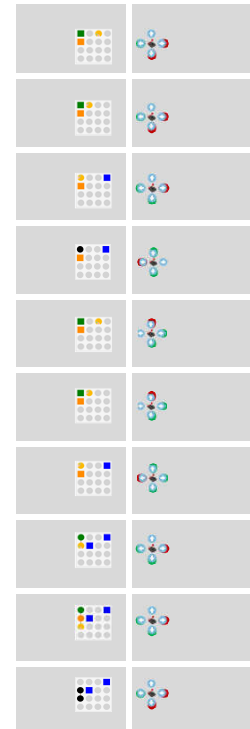
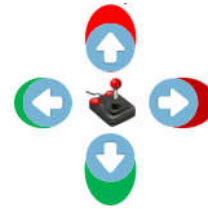
a policy is a map from states to action probabilities



...updated by the reinforcement learning algorithm



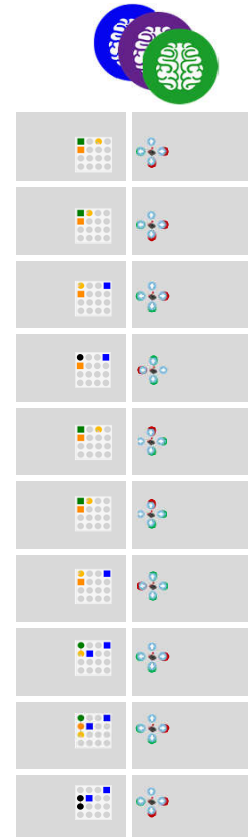
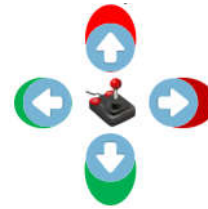
a policy is a map from states to action probabilities



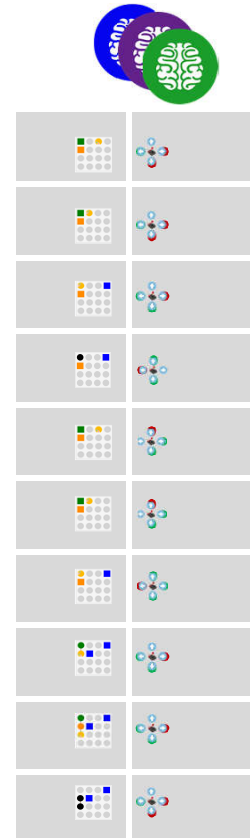
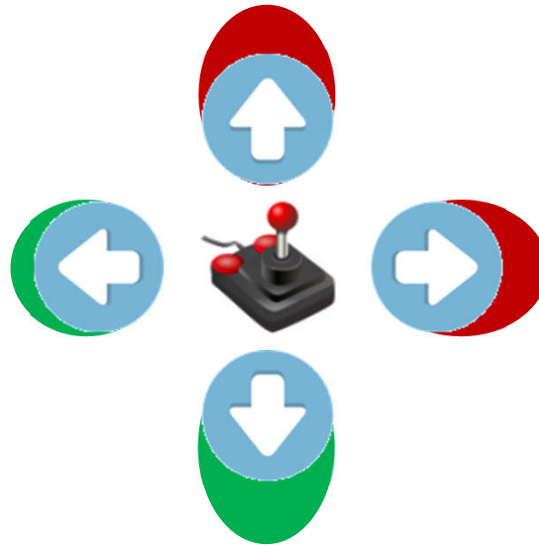
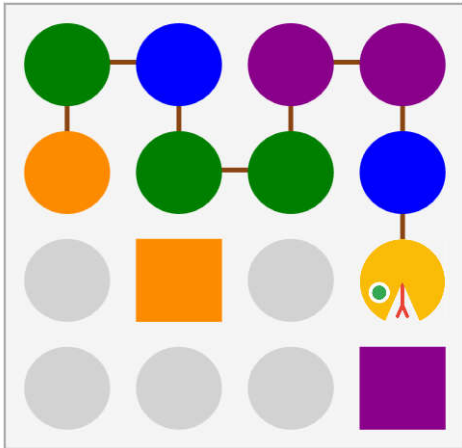
...updated by the reinforcement learning algorithm



a policy is a map from states to action probabilities



After many, many episodes, for each state...





Policy

(rules learned, how to play the game)

Algorithm sketch

Initialize **table** with random action probabilities for each **state**

Repeat

play episode with policy given by **table**

Record $(\text{state}_1, \text{action}_1, \text{reward}_1), (\text{state}_2, \text{action}_2, \text{reward}_2), \dots$ for episode

For each step i

compute $\text{FutureReward}_i = \text{reward}_i + \text{reward}_{i+1} + \dots$

update **table**[state_i] s.t.

- **action** _{i} becomes for state_i more likely if FutureReward_i is “high”
- **action** _{i} becomes for state_i less likely if FutureReward_i is “low”



Algorithm sketch



Policy

(rules learned, how to play the game)

Initialize **table** with random action probabilities for each **state**

Repeat

play episode with policy given by **table**

Record $(\text{state}_1, \text{action}_1, \text{reward}_1), (\text{state}_2, \text{action}_2, \text{reward}_2), \dots$ for episode

For each step i

compute $\text{FutureReward}_i = \text{reward}_i + \text{reward}_{i+1} + \dots$

update **table**[**state** _{i}] s.t.

- **action** _{i} becomes for **state** _{i} more likely if FutureReward_i is “high”
- **action** _{i} becomes for **state** _{i} less likely if FutureReward_i is “low”



Algorithm sketch



Policy

(rules learned, how to play the game)

Initialize **table** with random action probabilities for each **state**

Repeat

play episode with policy given by **table**

Record $(\text{state}_1, \text{action}_1, \text{reward}_1), (\text{state}_2, \text{action}_2, \text{reward}_2), \dots$ for episode

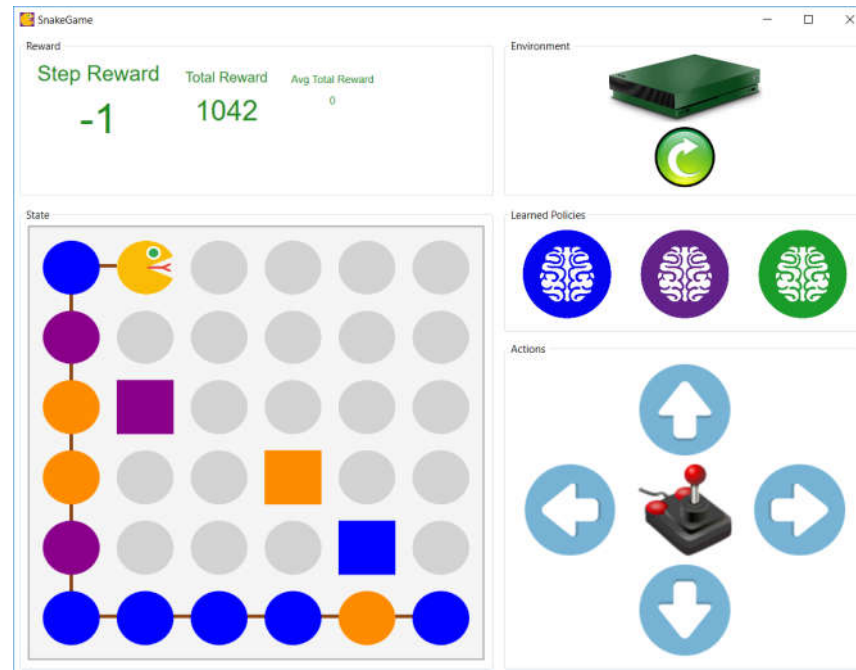
For each step i

compute $\text{FutureReward}_i = \text{reward}_i + \text{reward}_{i+1} + \dots$

update $\text{table}[\text{state}_i]$ s.t.

- action_i becomes for state_i more likely if FutureReward_i is “high”
- action_i becomes for state_i less likely if FutureReward_i is “low”

The game: demo



The bad news: nice idea, but...



The bad news: nice idea, but...

too many states... too many actions

- Too much memory needed
- Too much time



The solution



Policy

(rules learned, how to play the game)

Idea:

Replace lookup table with a **neural network** that approximates the action probabilities contained in the table

Instead of

$\text{Table}[\text{state}] = \text{action probabilities}$

Do

$\text{NeuralNet}(\text{state}) \sim \text{action probabilities}$

```
Initialize table with action probabilities for each state
Repeat
  play episode with policy given by table
  Record (state1,action1,reward1),(state2,action2,reward2),... for episode
  For each step i
    compute FutureRewardi = rewardi + rewardi+1 + ...
    update table[statei] st.
      • actioni becomes for statei more likely if FutureRewardi is "high"
      • actioni becomes for statei less likely if FutureRewardi is "low"
```

Change to "play episode with policy given by **NeuralNet**"

Change to "update **weights** of **NeuralNet**"

Neural nets to the rescue

Idea:

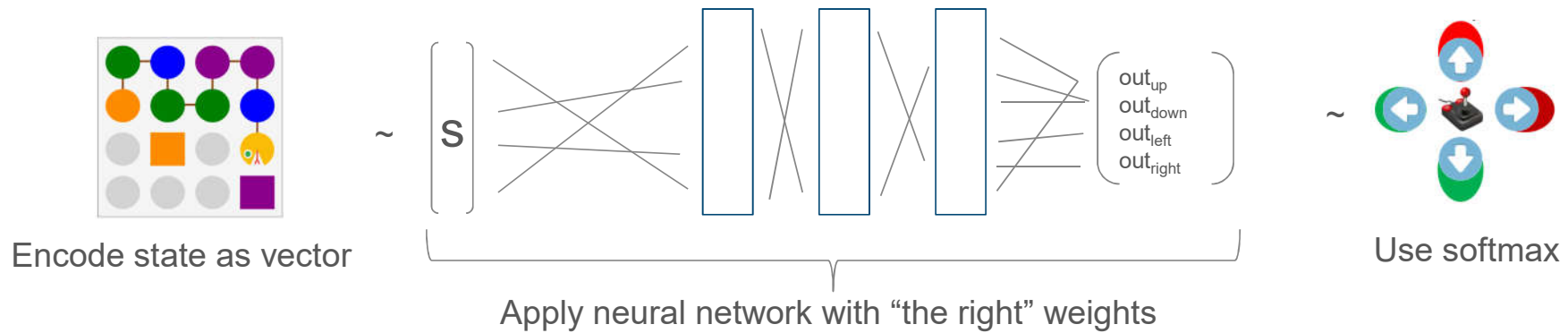
Replace lookup table with a neural network that approximates the action probabilities contained in the table

Instead of

$\text{Table}[\text{state}] = \text{action probabilities}$

Do

$\text{NeuralNet}(\text{state}) \sim \text{action probabilities}$



Policy Gradient Algorithm sketch



Policy

(rules learned, how to play the game)

Initialize **neuralNet** with random **weights W**

Repeat

play episode(s) with policy given by **weights W**

Record $(state_1, action_1, reward_1), (state_2, action_2, reward_2), \dots$ for episode(s)

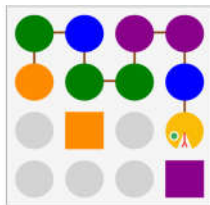
For each step i

compute $FutureReward_i = reward_i + reward_{i+1} + \dots$

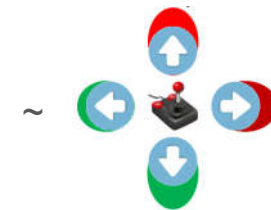
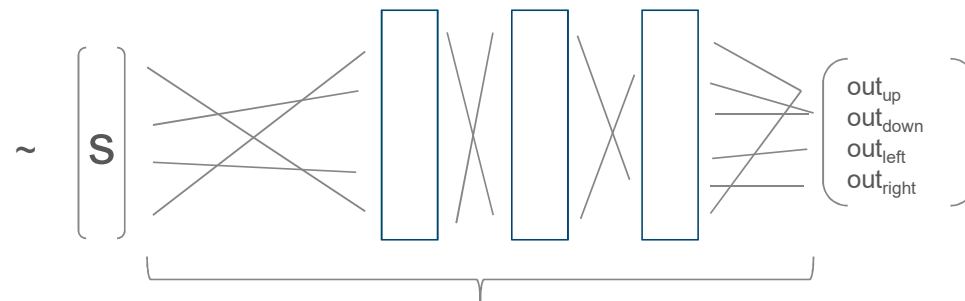
Update **weights W**

$W = W +$

????



Encode state as vector



Use softmax

Policy Gradient Algorithm sketch



Policy

(rules learned, how to play the game)

Initialize **neuralNet** with random **weights W**

Repeat

play episode(s) with policy given by **weights W**

Record $(state_1, action_1, reward_1), (state_2, action_2, reward_2), \dots$ for episode(s)

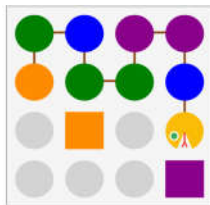
For each step i

compute $FutureReward_i = reward_i + reward_{i+1} + \dots$

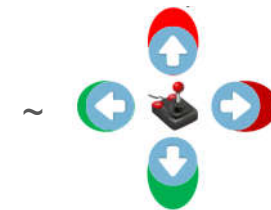
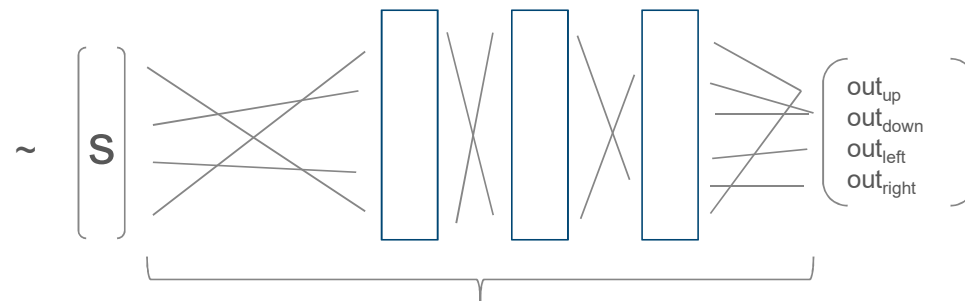
Update **weights W**

$W = W +$

????



Encode state as vector



Use softmax

Policy Gradient Algorithm sketch



Policy

(rules learned, how to play the game)

Initialize **neuralNet** with random **weights W**

Repeat

play episode(s) with policy given by **weights W**

Record $(state_1, action_1, reward_1), (state_2, action_2, reward_2), \dots$ for episode(s)

For each step i

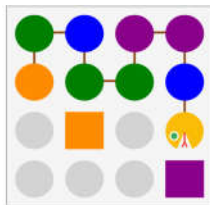
compute $FutureReward_i = reward_i + reward_{i+1} + \dots$

Update **weights W**

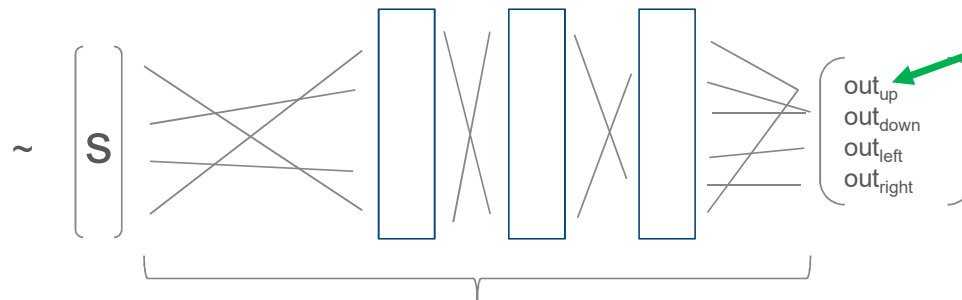
$W = W +$

????

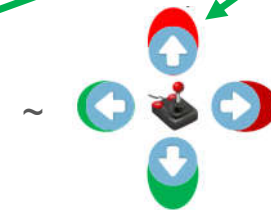
Increases out_i



Encode state as vector



Weights W



Use softmax

Policy Gradient Algorithm sketch



Policy

(rules learned, how to play the game)

Initialize **neuralNet** with random **weights W**

Repeat

play episode(s) with policy given by **weights W**

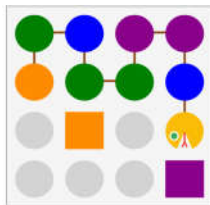
Record $(state_1, action_1, reward_1), (state_2, action_2, reward_2), \dots$ for episode(s)

For each step i

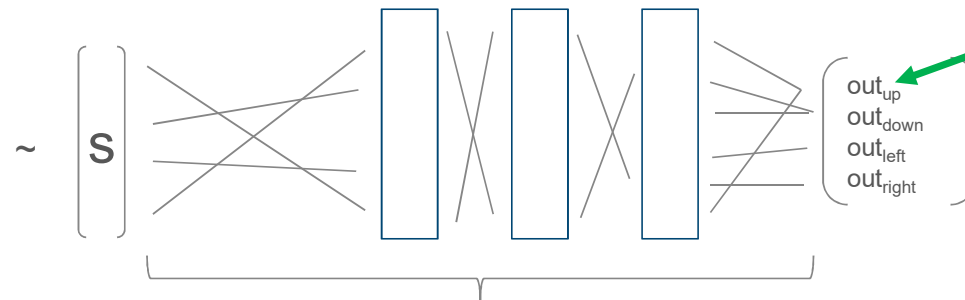
compute $FutureReward_i = reward_i + reward_{i+1} + \dots$

Update **weights W**

$$W = W + \underbrace{\alpha}_{\text{Learning rate}} * FutureReward_i * \underbrace{Gradient_W (neuralNet_W (state_i, action_i))}_{\text{Increases out}_i}$$



Encode state as vector



Weights W



Use softmax

Policy Gradient

$$W = \underbrace{\arg \max_W}_{\text{this is trouble}} \underbrace{E_{\tau \sim p_W}[R(\tau)]}_{\text{expected total reward playing with } W = f(W)}$$

$$W_{k+1} = W_k + \alpha \cdot \nabla_W f(W_k)$$

$$\tau = (s_1, a_1, r_1), (s_2, a_2, r_2), \dots$$

$$R(\tau) = \sum_i r_i$$

p_W = episode probability given the policy defined by the NeuralNet with weights W



Policy Gradient

$$W = \underbrace{\arg \max_W}_{\text{this is trouble}} \underbrace{E_{\tau \sim p_W}[R(\tau)]}_{\substack{\text{expected total reward playing with } W \\ = f(W)}}$$

$$\tau = (s_1, a_1, r_1), (s_2, a_2, r_2), \dots$$

$$R(\tau) = \sum_i r_i$$

p_W = episode probability given the policy defined by the NeuralNet with weights W

$$W_{k+1} = W_k + \alpha \cdot \underbrace{\nabla_W E_{\tau \sim p_W}[R(\tau)]}_{\text{this is the new trouble}}$$

$$\begin{aligned} \nabla_W E_{\tau \sim p_W}[R(\tau)] &= \nabla_W \int_{\tau} p_W(\tau) \cdot R(\tau) &&= \int_{\tau} \nabla_W p_W(\tau) \cdot R(\tau) \\ &= \int_{\tau} p_W(\tau) \cdot \underbrace{\frac{\nabla_W p_W(\tau)}{p_W(\tau)}} \cdot R(\tau) &&= \int_{\tau} p_W(\tau) \cdot \underbrace{\nabla_W \log p_W(\tau)} \cdot R(\tau) \\ &= \underbrace{E_{\tau \sim p_W}[\nabla_W \log p_W(\tau) \cdot R(\tau)]}_{\text{good news}} \end{aligned}$$

$$\frac{\nabla_x f(x)}{f(x)} = \nabla_x \log f(x)$$

Policy Gradient

$$W = \underbrace{\arg \max_W}_{\text{this is trouble}} \underbrace{E_{\tau \sim p_W}[R(\tau)]}_{\text{"average" total reward playing with W}}$$

$$\tau = (s_1, a_1, r_1), (s_2, a_2, r_2), \dots$$

$$R(\tau) = \sum_i r_i$$

p_W = episode probability given the policy defined by the NeuralNet with weights W

$$W_{k+1} = W_k + \alpha \cdot \underbrace{\nabla_W E_{\tau \sim p_W}[R(\tau)]}_{\text{this is the new trouble}}$$

$$W_{k+1} = W_k + \alpha \cdot E_{\tau \sim p_W} \left[\underbrace{\nabla_W \log p_W(\tau)}_{\sum_i \nabla_W \text{NeuralNet}_W(s_i, a_i)} \cdot \underbrace{R(\tau)}_{\text{FutureReward}_i} \right]$$

$$W = W + \text{alpha} * \text{Gradient}_W (\text{neuralNet}_W(\text{state}_i, \text{action}_i)) * \text{FutureReward}_i$$

What for ?

The real world



no feasible, deterministic algorithm



What for ?



Traditional
Heuristics

Classic
Machine
Learning

Reinforcement
Learning

Automatic solution found in 93.4%

The challenges



manage the water level on the roof
control & steer the water flow
find the right dimensions
save & reliable





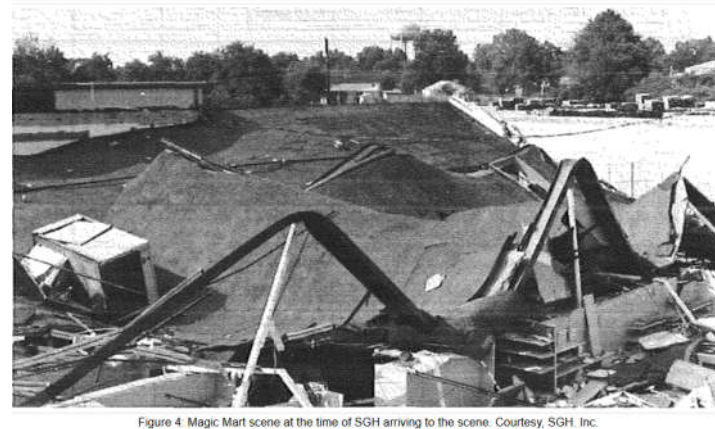
Finding the right dimensions





What if...

- Collapsing pipes
- Collapsing roofs
- Clogged pipes
- Façade damages

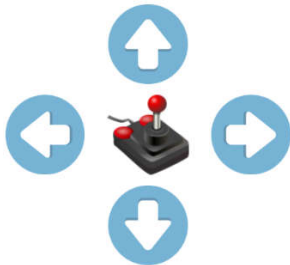


Turning the problem into a game

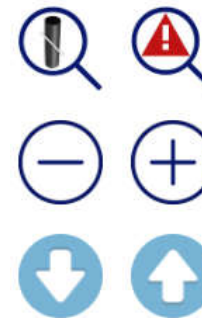


Designing the Action-Space

Snake game



Roof drainage systems

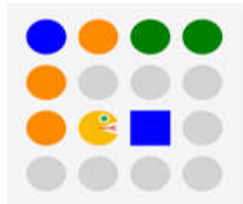


- What actions would a human expert like to have ?
- Are these actions sufficient ?
- Would more / other actions be helpful ?
- Can we drop any actions ?



Designing the State-Space

Snake game



Roof drainage systems

Typ	TS	d [mm]	L [m]	H [m]	V Set [l/s]	V [l/s]	p in [mbar]	p out [mbar]	p at [mbar]	p dyn [mbar]	v [m/s]	W [N]	V A min [N]	V A [N]	Zeta	L R+Z [mbar]	L R+Z [mbar]	R [mbar]	Z [mbar]	Reumtyp
1	25	125	5.00		40.0	40.6	-278	-250	58	81	4.3	81			0.0	80	254	11	30	MR12
1	26	110	5.00		40.0	32.0	-195	-281	58	88	4.2	88			0.0	88	254	13	34	MR12
1	27	110	5.00		32.0	25.3	-115	-182	58	55	3.2	85			0.1	47	188	8	8	MR12
1	28	90	5.00		24.0	17.3	-52	-115	58	51	3.2	100			0.2	63	121	10	12	MR12
1	29	90	5.00		18.0	10.2	8	-18	28	18	1.9	100			0.3	24	58	4	5	MR12
1	30	90	5.00		8.0	4.0	28	20	28	4	0.8	100			0.2	8	34	1	1	MR12
1	31	75	0.50	0.50	8.0	4.0	-23	21	58	9	1.9	100			0.4	5	28	2	9	MR12
1	32	56	0.09	0.50	8.0	4.0	-45	9	31	2.5	100				0.7	23	23	12	22	---
1	33/29	75	0.50	0.50	8.0	5.3	-29	18	58	10	1.4	100			0.5	6	34	3	5	MR12
1	34	56	0.09	0.08	8.0	8.2	-58	9	37	2.7	100				0.7	28	38	14	37	---

- What does a human expert look at ?
- Can you switch the experts between 2 steps ?
- Full state vs partial state
- Designing Features

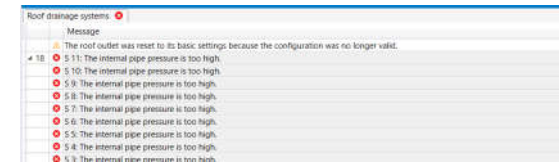
Designing the Reward Function

Snake game

Step Reward
100

Fruit	100
Death	-50
Success	1000
Step	-1

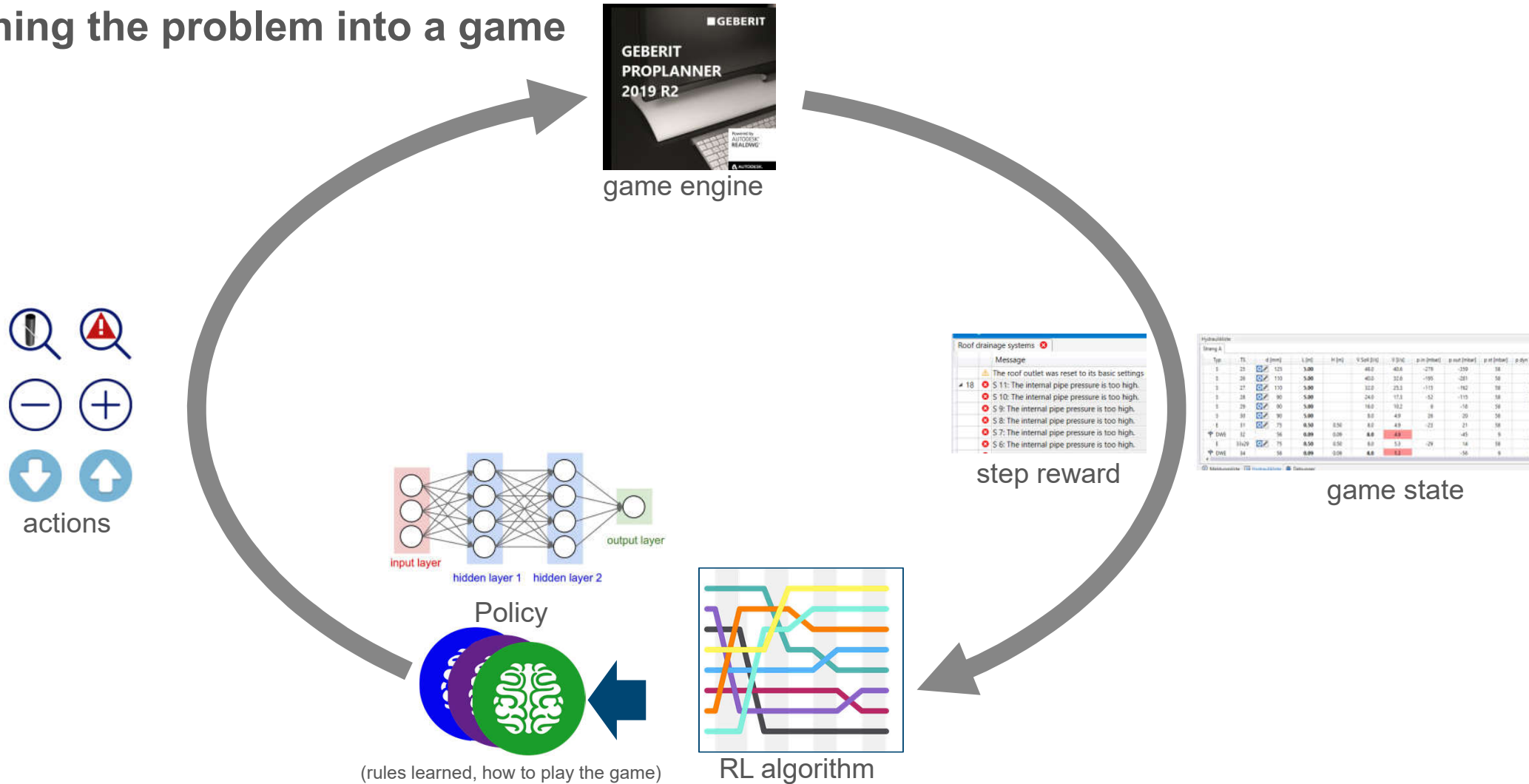
Roof drainage systems



Change Error Count	+/- 1 per Error
Success	100
Step	-0.01

- How would you rate the result of an expert ?
- As simple as possible
- Positive feedback during the game
- Beware of “surprising policies”
- Game over if TotalReward too low

Turning the problem into a game



Finding the dimensions with reinforcement learning: demo

The screenshot displays the GEBERIT PROPLANNER 2019 R2 software interface. The main window shows a 3D hydraulic network diagram with various pipe segments and fittings. The interface includes a top menu bar, a toolbar, and a sidebar with various views and settings.

On the left side, there is a 'Debugger' panel with a 'Reward' section showing 'Step Reward 100.99' and 'Total Reward 114.59'. Below this, there are 'Actions' represented by icons: a magnifying glass, a warning triangle, a minus sign, a plus sign, a down arrow, and an up arrow.

The bottom right corner features a 'Hydraulikliste' (Hydraulic List) table for 'Strang A'. The table contains the following data:

Typ	TS	d [mm]	L [m]	H [m]	V Sob [l/s]	V [l/s]	p in [mbar]	p out [mbar]	v [m/s]	Ψ [%]	Zeta	L-R-Z [mbar]
S	1	315	1.00		204.0	200.8	2	0	3.1	94	1.0	47
F	2	315	3.00	3.00	204.0	200.8	-255	2	3.1	94	0.3	21
S	3	315	50.00		204.0	200.8	-142	-235	3.1	94	0.3	113
F	4	115	6.36		102.0	100.1	-101	-107	1.5	97	0.3	7
S	5	315	3.50		102.0	100.1	-95	-101	1.5	97	0.3	6
S	6	315	12.00		102.0	100.1	-85	-95	1.5	97	0.3	10
S	7	315	15.00		81.6	79.7	-75	-81	1.2	97	0.0	5
S	8	230	15.00		61.2	59.3	-67	-78	1.4	97	0.1	11
S	9	230	15.00		40.8	38.9	-57	-62	0.9	99	0.0	4
S	10	160	15.00		20.4	20.4	-45	-51	1.2	99	0.4	15
S	11	110	0.71		20.4	20.4	-49	-70	2.6	99	0.5	21
E	12	110	0.25	0.25	20.4	20.4	-63	-49	2.6	99	0.3	11
E	13	110	0.20	0.20	20.4	20.4	-81	-63	2.6	99	0.0	1
PKS	14	90	0.20	0.20	20.4	20.4	-121	-121	1.8	99	0.6	68

The bottom right corner of the interface shows the status bar with 'Poland', 'Deutsch', and version information '4.2.5.163 / 4.2.5.150 (Kwadrat 2018)'.

Hydraulics Calculation Pipeline



Traditional
Heuristics

Classic
Machine
Learning

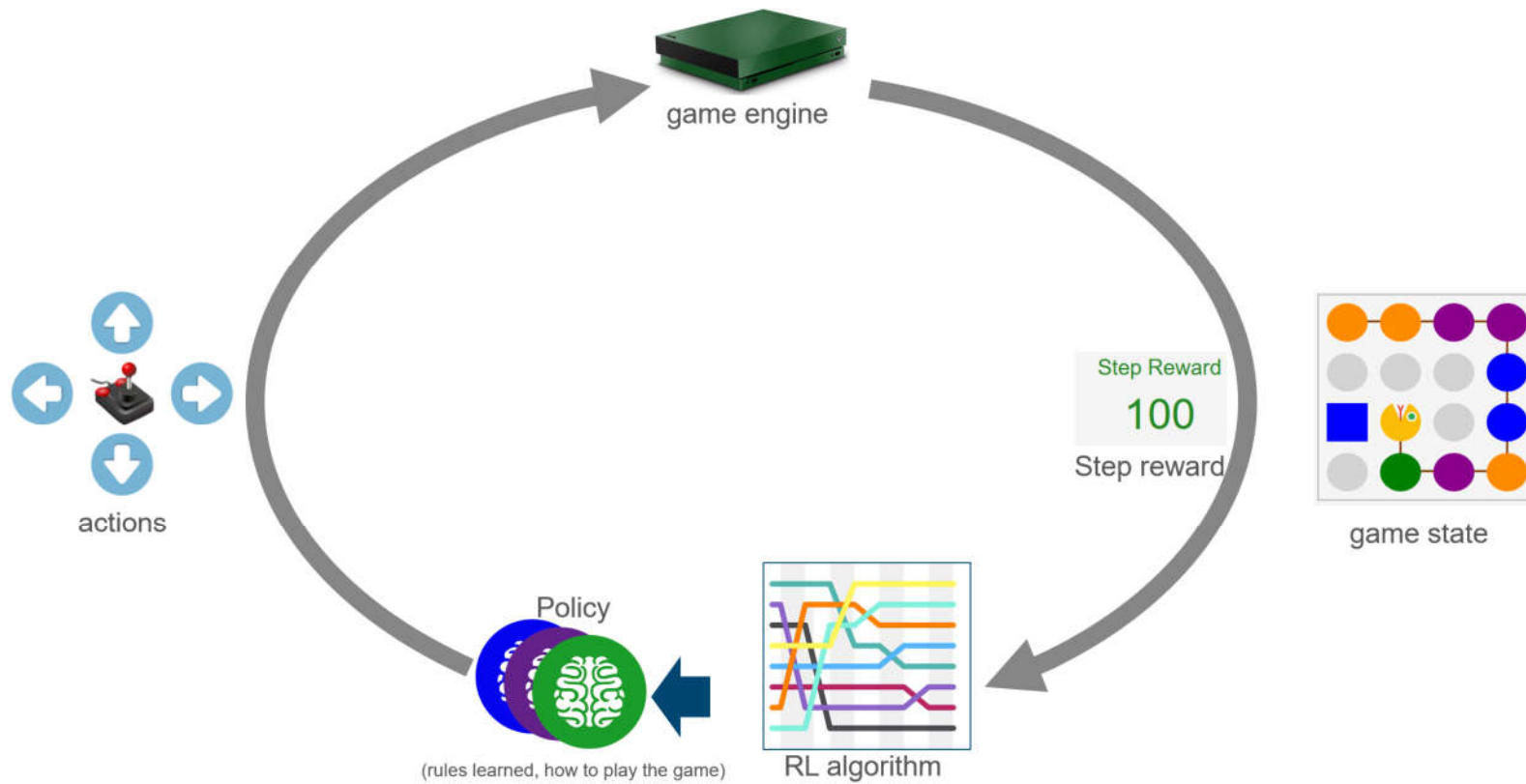
Reinforcement
Learning

Automatic solution found in 93.4%

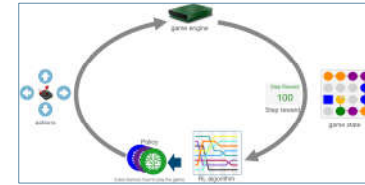
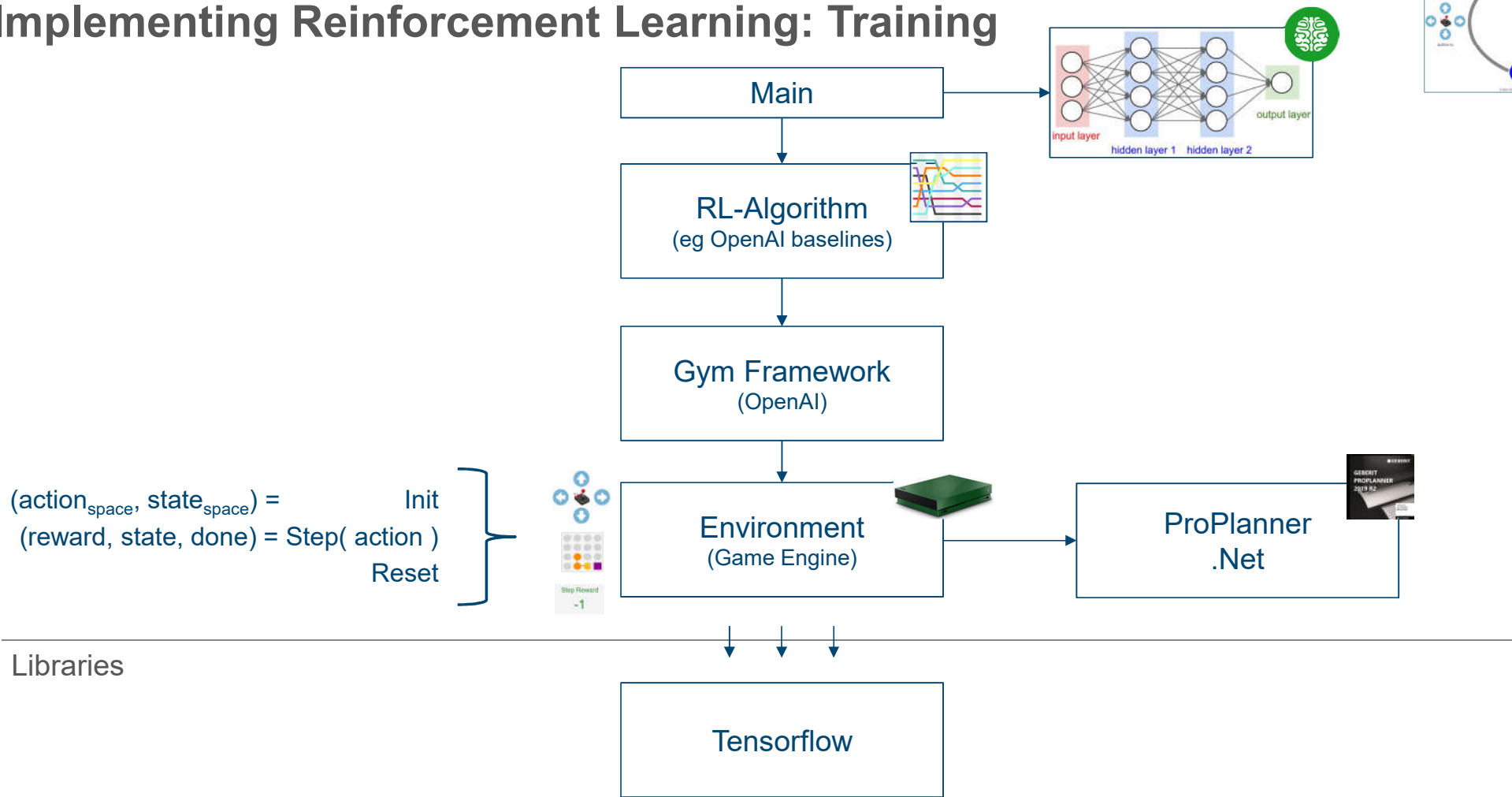
Finds a solution in **70.7%**
of the remaining 6.6%

Automatic solution found in **98.1%**

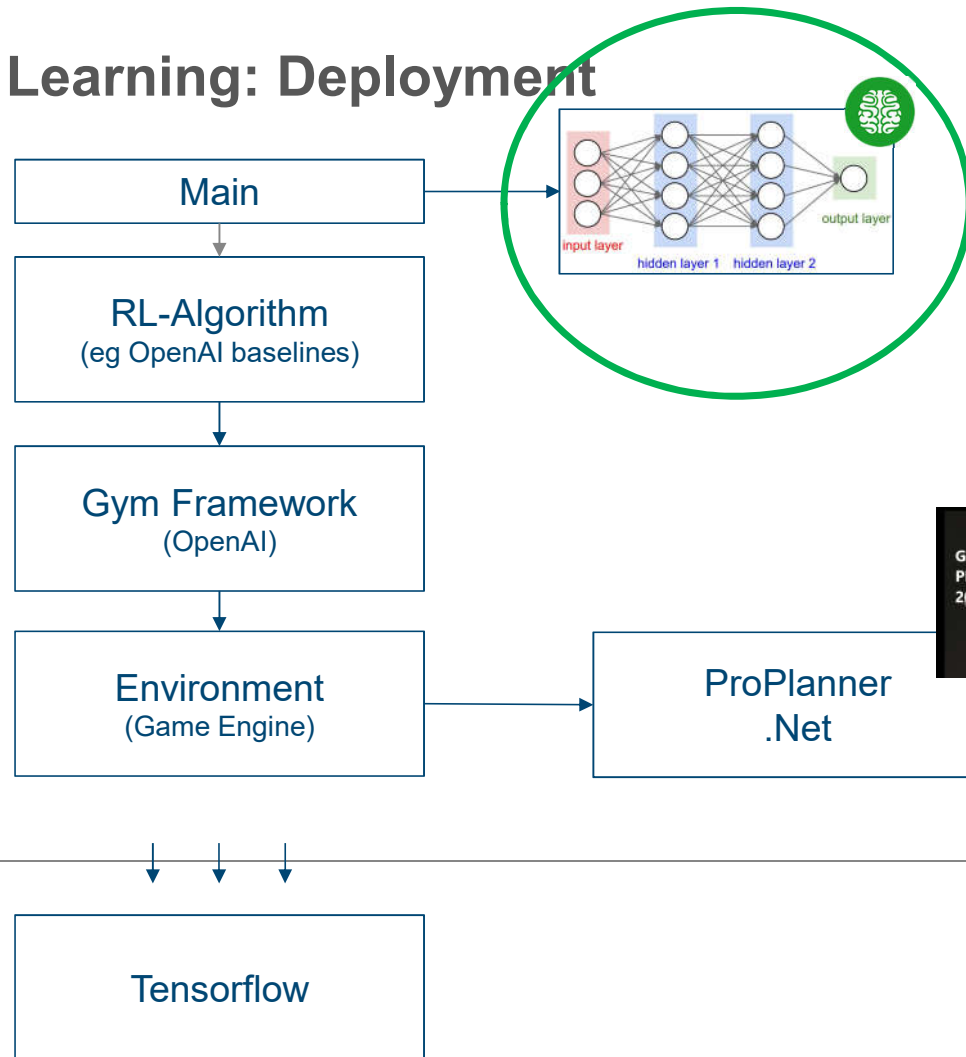
Implementing Reinforcement Learning: Training



Implementing Reinforcement Learning: Training



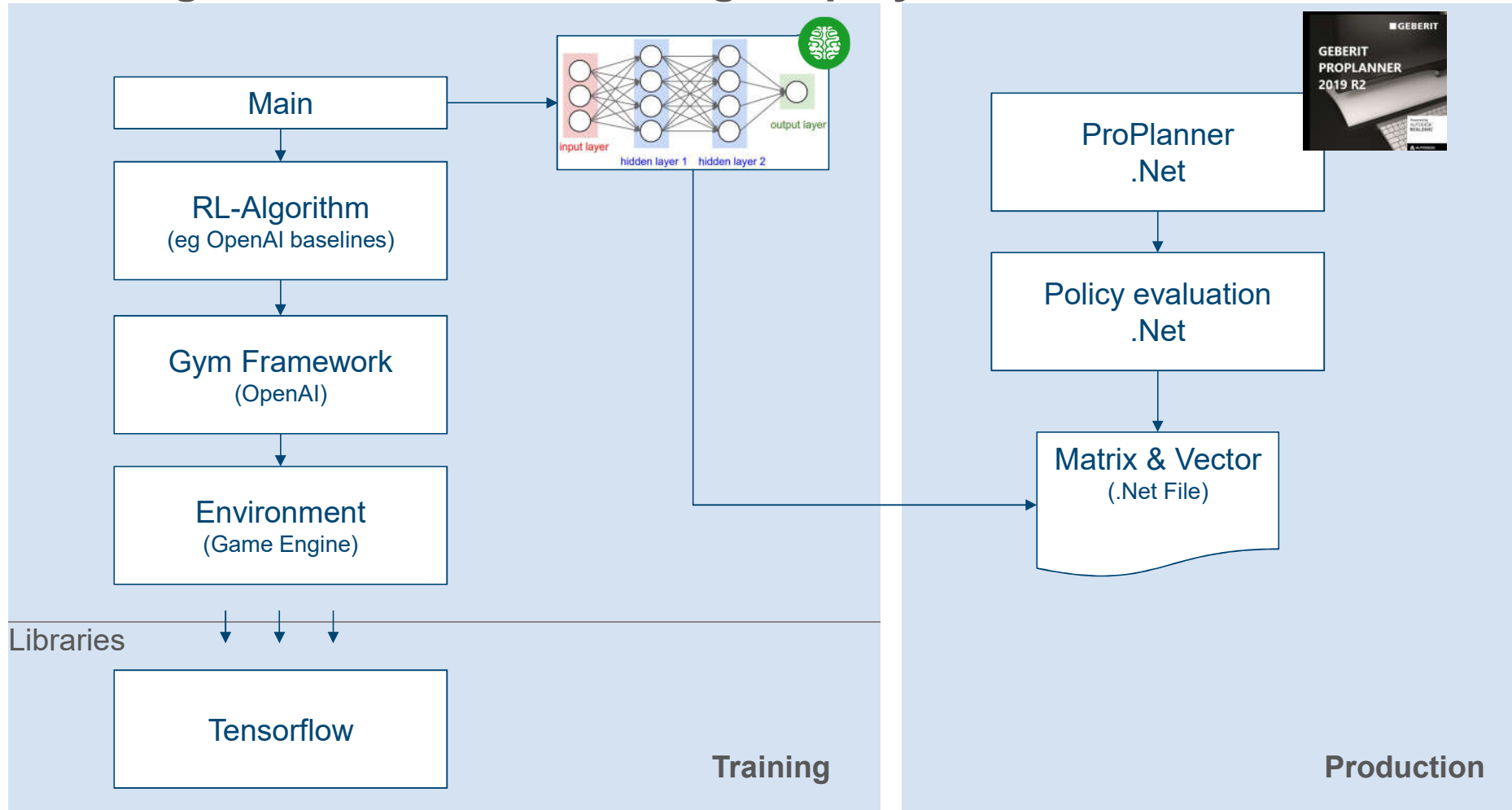
Implementing Reinforcement Learning: Deployment



Libraries

Tensorflow

Implementing Reinforcement Learning: Deployment



Wrap Up

- Turning the problem into a game
- Continuous policy improvement
- No training dataset
- Complements supervised learning



Thank you!

Christian.Hidber@bSquare.ch

W +41 44 260 54 00

M +41 76 558 41 48

<https://www.linkedin.com/in/christian-hidber/>



About Geberit

The globally operating Geberit Group is a European leader in the field of sanitary products. Geberit operates with a strong local presence in most European countries, providing unique added value when it comes to sanitary technology and bathroom ceramics.

The production network encompasses 30 production facilities, of which 6 are located overseas. The Group is headquartered in Rapperswil-Jona, Switzerland. With around 12,000 employees in around 50 countries, Geberit generated net sales of CHF 2.9 billion in 2017. The Geberit shares are listed on the SIX Swiss Exchange and have been included in the SMI (Swiss Market Index) since 2012.



FOLIE 66
REINFORCEMENT LEARNING

 **GEBERIT**

Resources

- Sutton & Barto: Reinforcement Learning, an introduction, 2nd edition, 2018:
https://drive.google.com/file/d/1opPSz5AZ_kVa1uWOdOiveNiBFiEOHjkG/view
- http://rll.berkeley.edu/deeprlcourse/f17docs/lecture_4_policy_gradient.pdf
- <http://karpathy.github.io/2016/05/31/rl/>
- <https://papers.nips.cc/paper/1713-policy-gradient-methods-for-reinforcement-learning-with-function-approximation.pdf>
- <https://arxiv.org/pdf/1707.06347.pdf>
- <https://openai.com/>



