GEBERIT

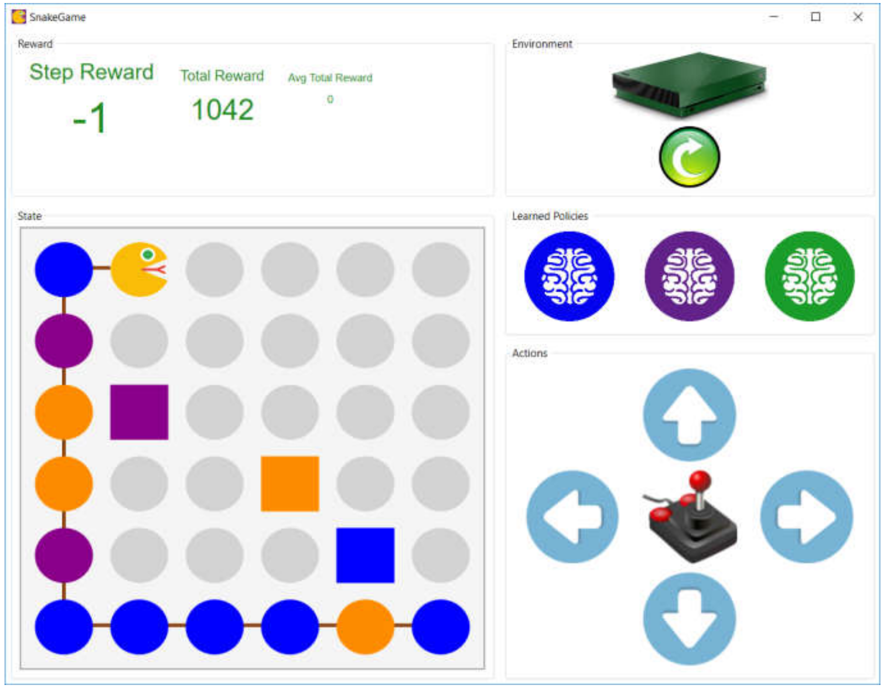
Reinforcement Learning

a gentle introduction & industrial application

Dr. Christian Hidber

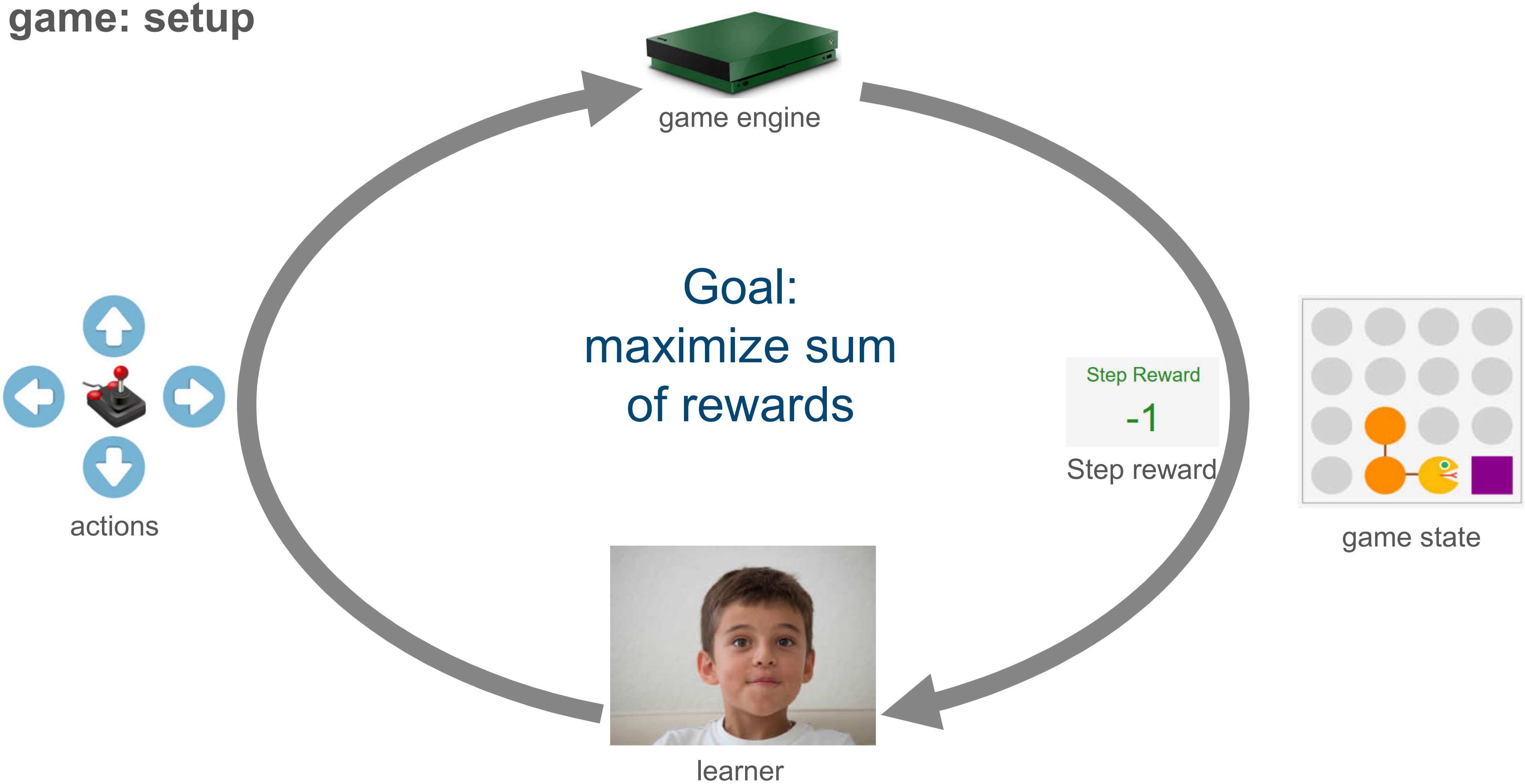
Learning learning from children

The game: demo

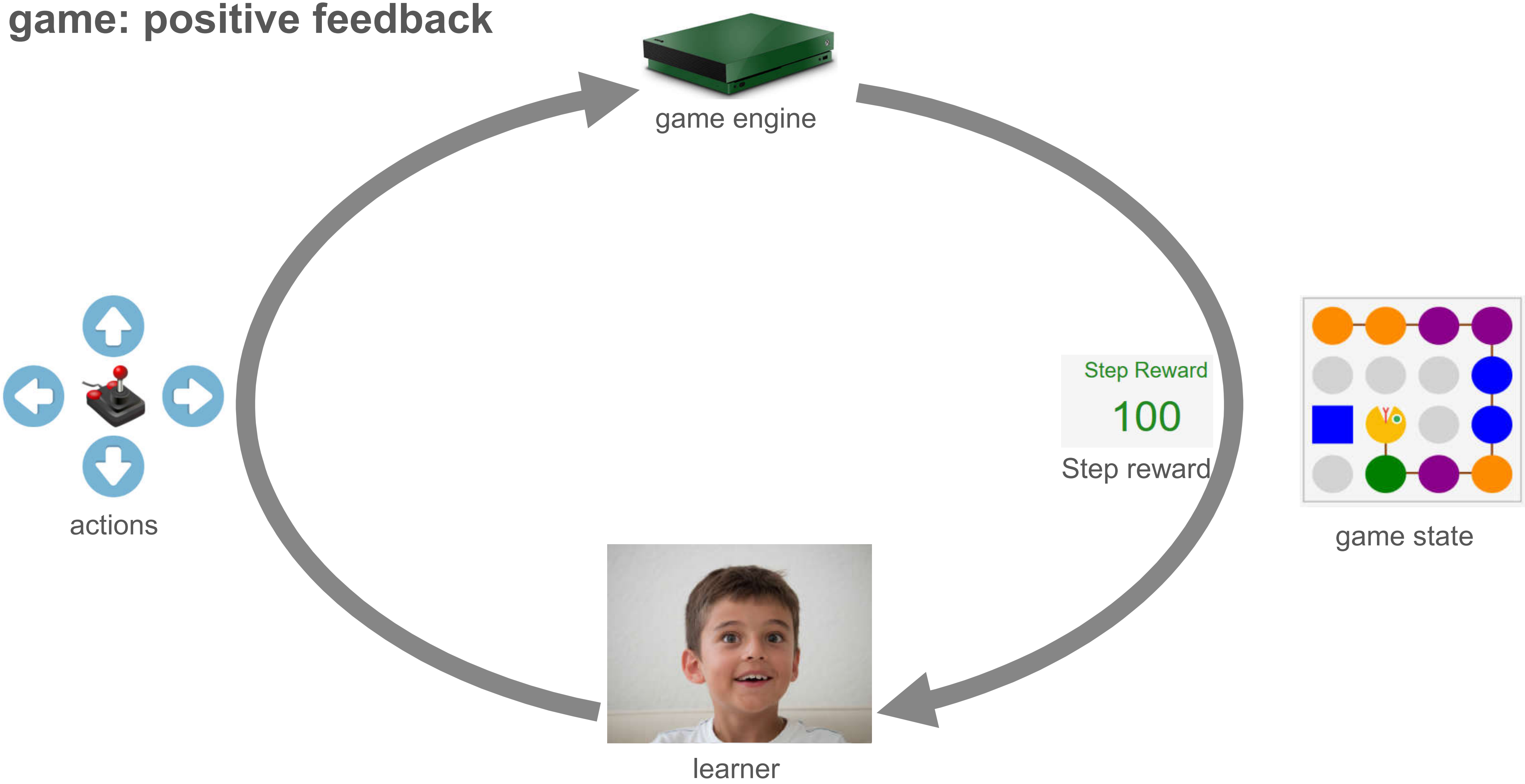


GEBERIT

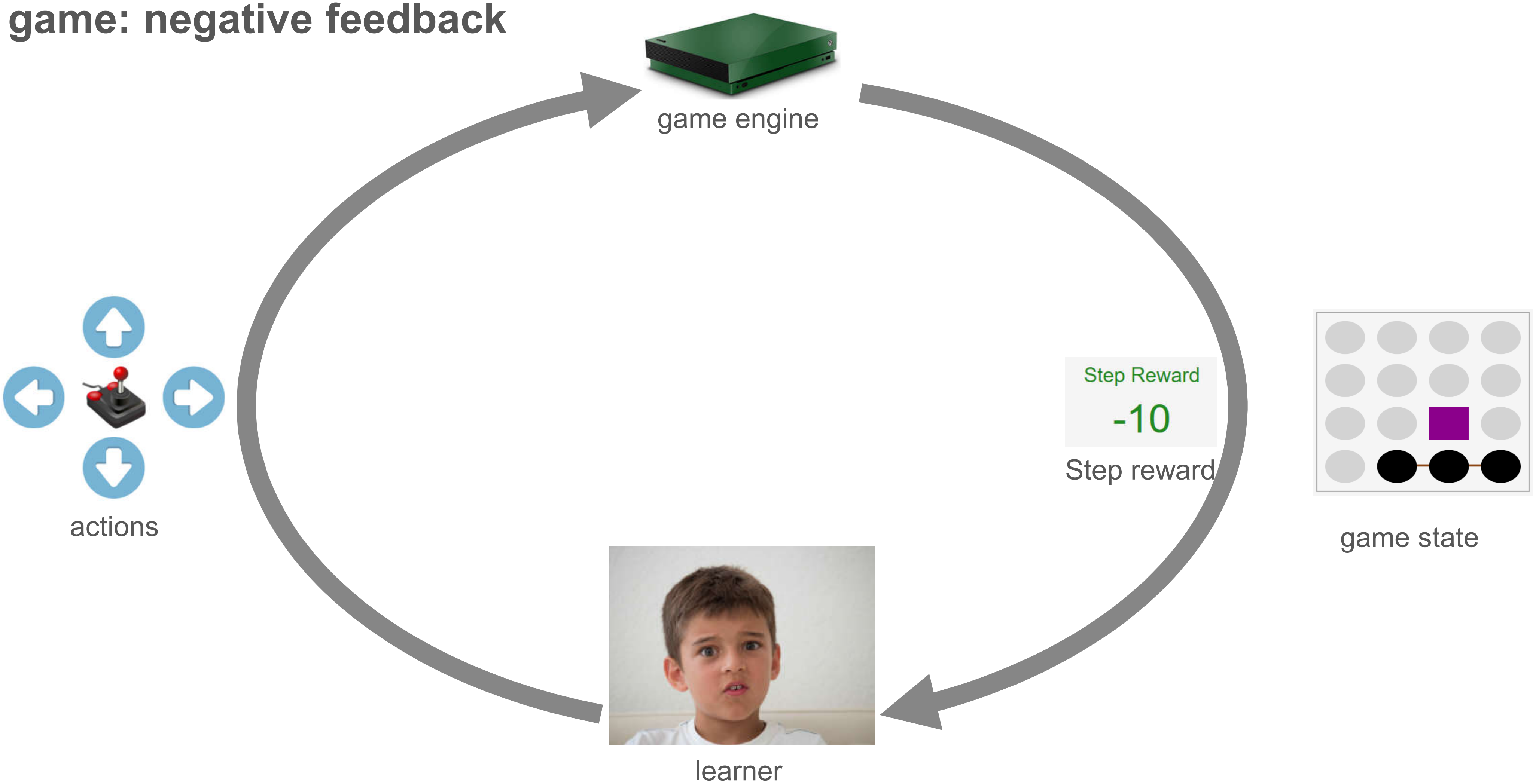
The game: setup



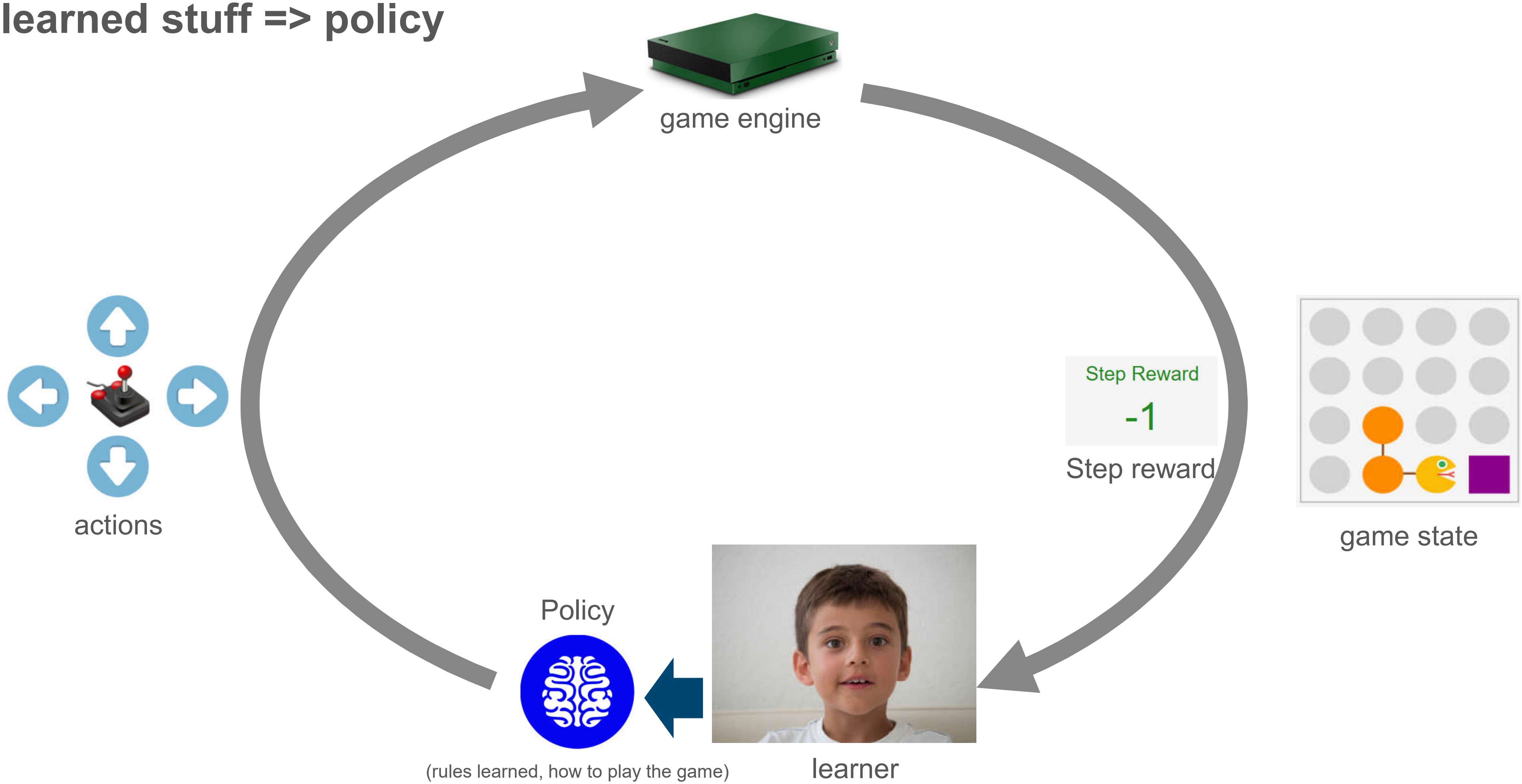
The game: positive feedback



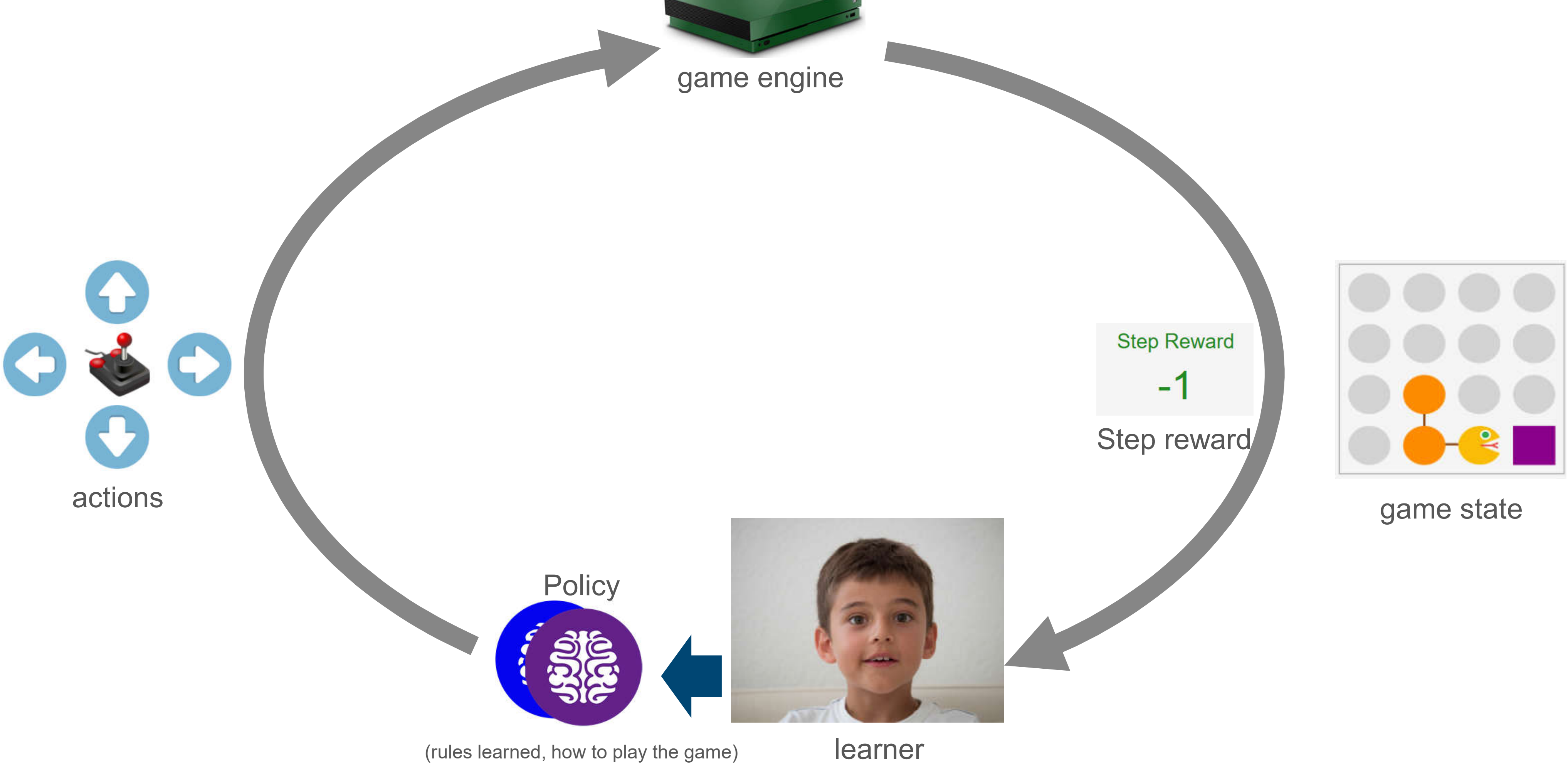
The game: negative feedback



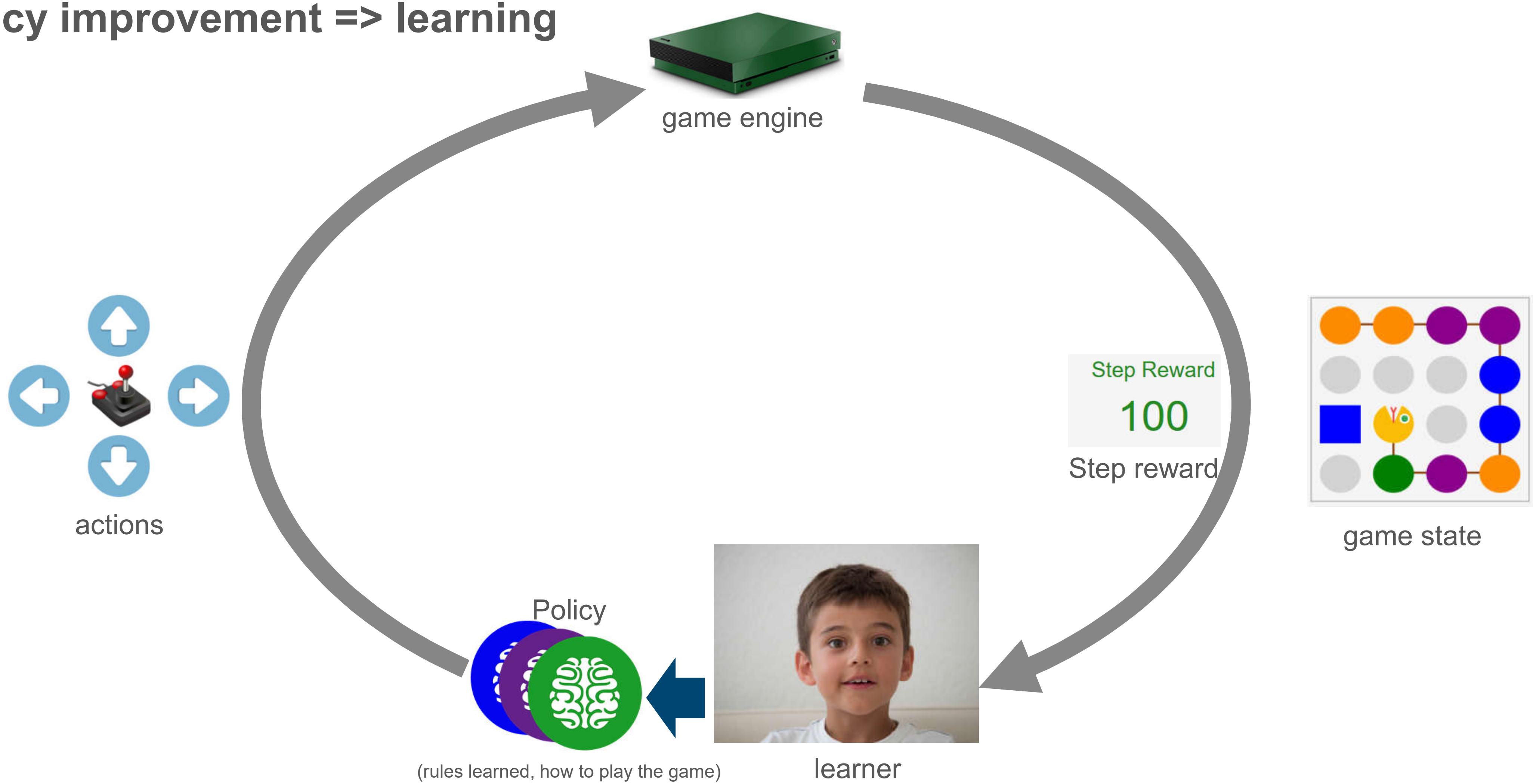
the learned stuff => policy



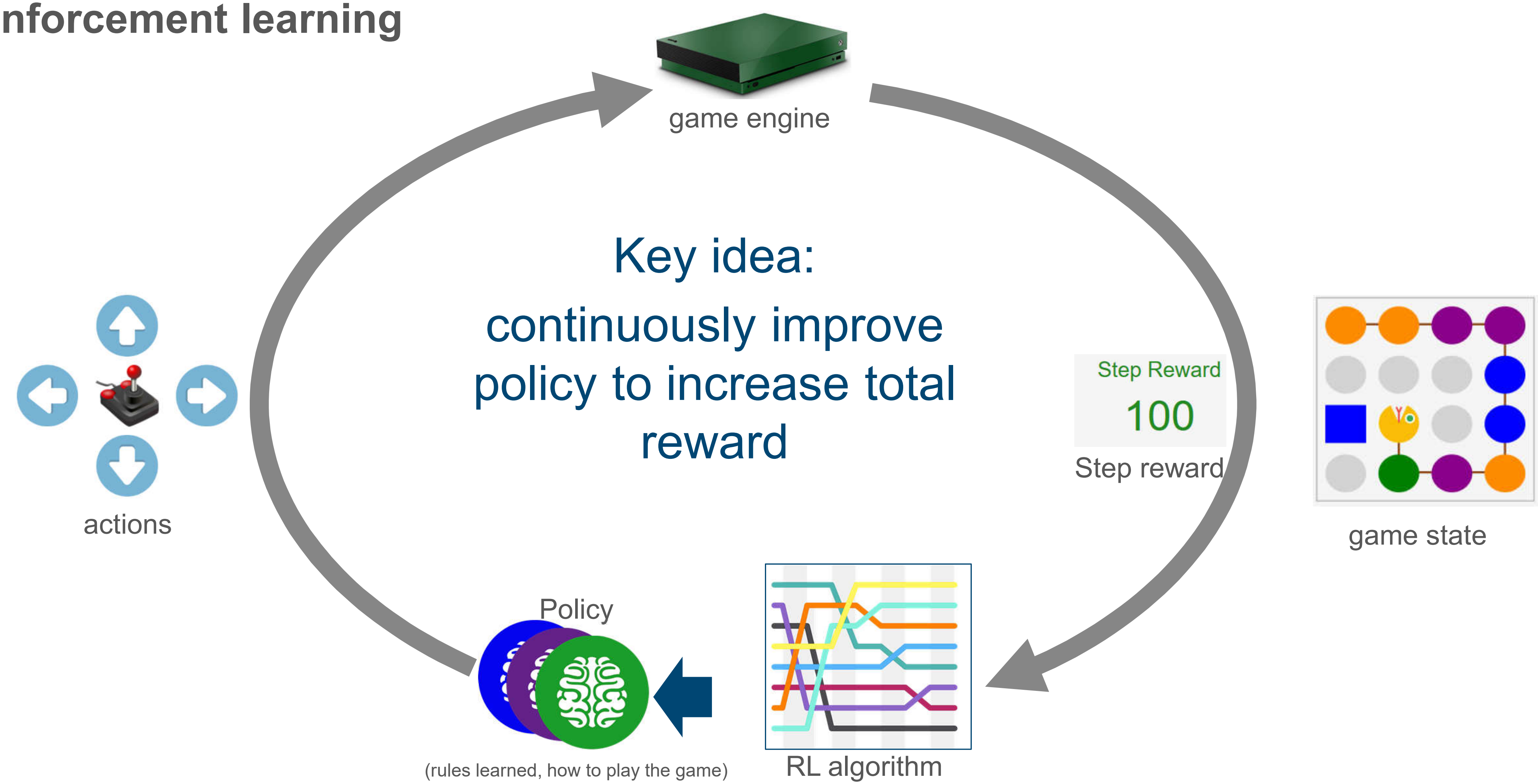
policy improvement => learning



policy improvement => learning



Reinforcement learning



Episode 1 : play with 1st policy (random)

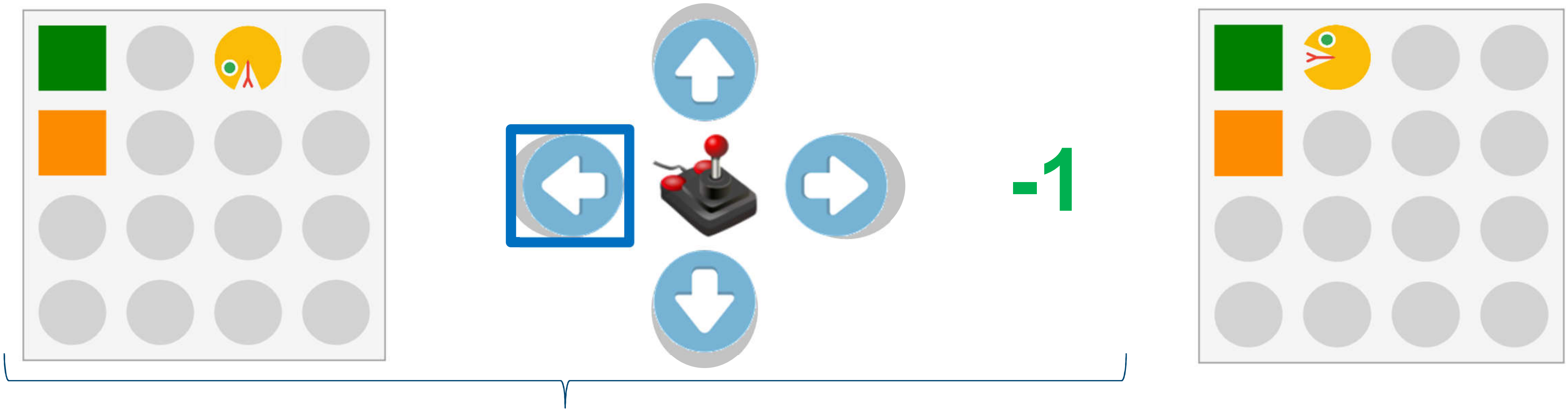
Policy
(rules learned, how to play the game)

State

Action from Policy

Reward

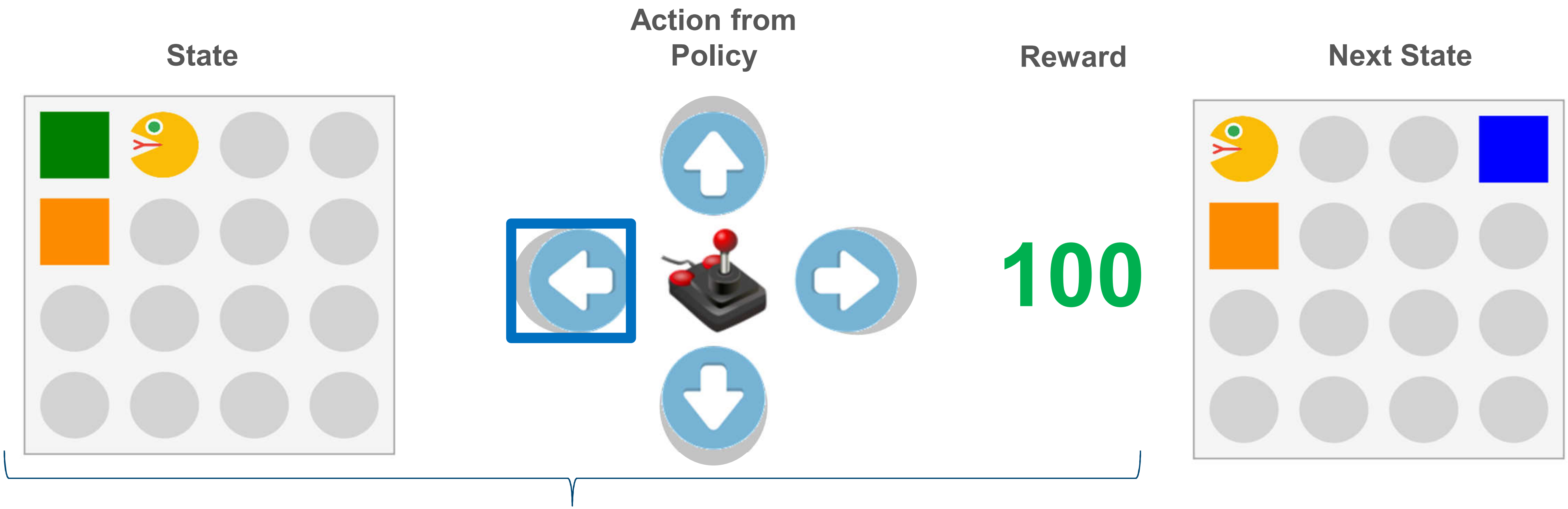
Next State

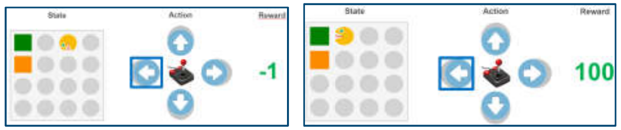


1 2 3 4 5 6 7 Step #

Episode 1 : play with 1st policy (random)

Policy
(rules learned, how to play the game)



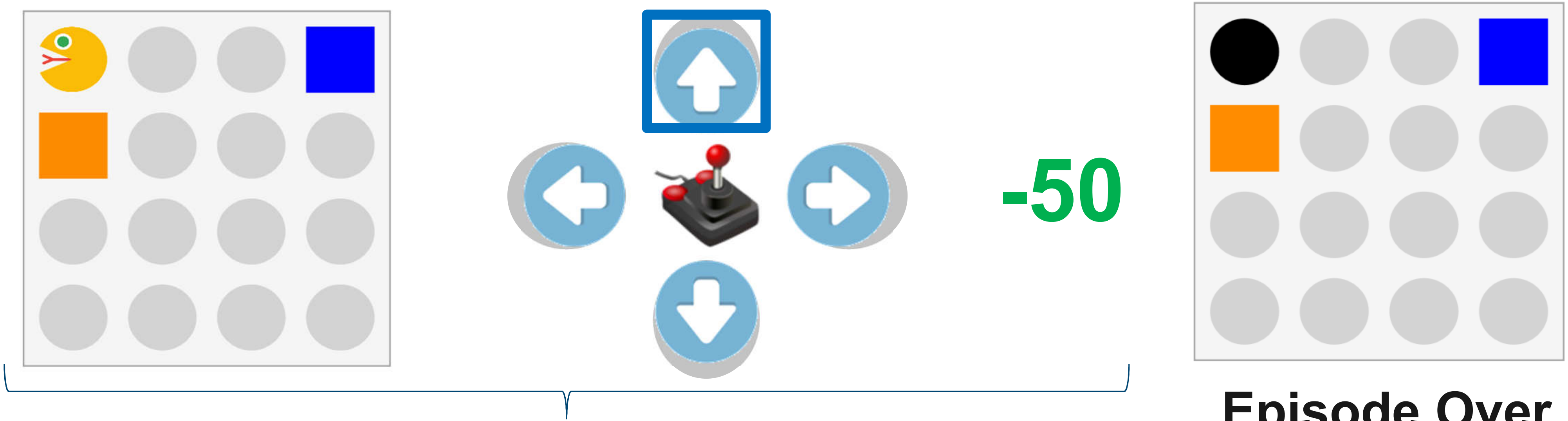


1 2 3 4 5 6 7 Step #

Episode 1 : play with 1st policy (random)

Policy
(rules learned, how to play the game)

State | Action from Policy | Reward | Next State



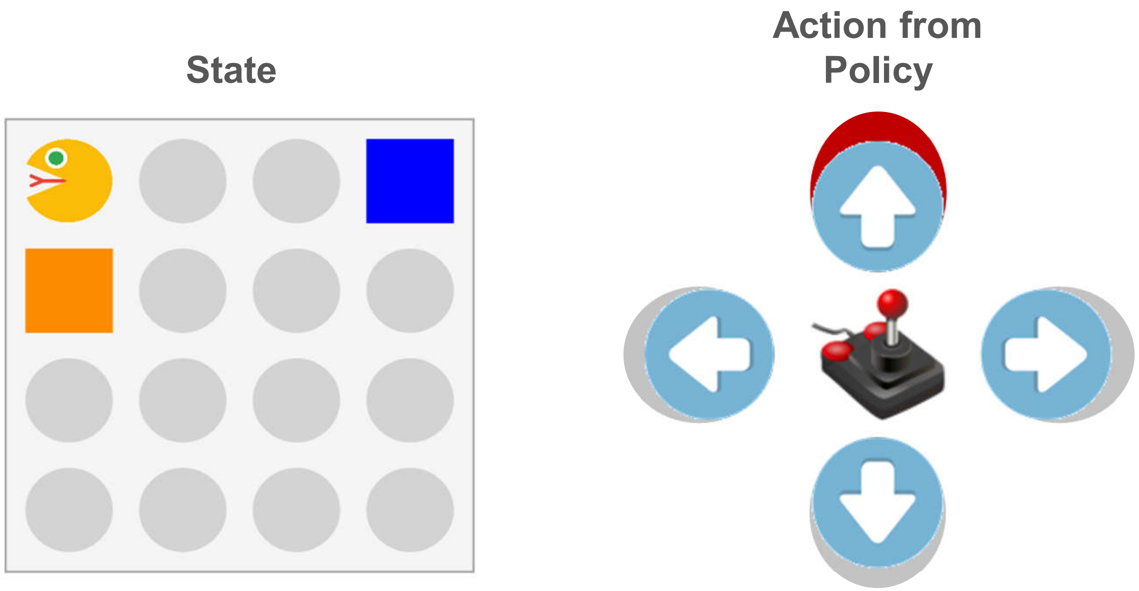
Episode Over

Episode Over

1 2 3 4 5 6 7 Step #

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

Policy
(rules learned, how to play the game)



-50

Episode Over

1 2 3 4 5 6 7 Step #

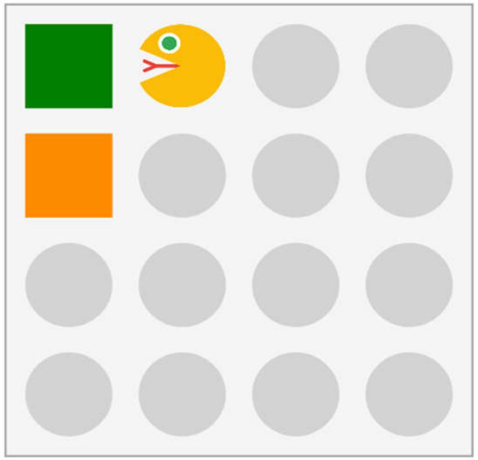
Episode 1 : improve 1st policy for state in step 2

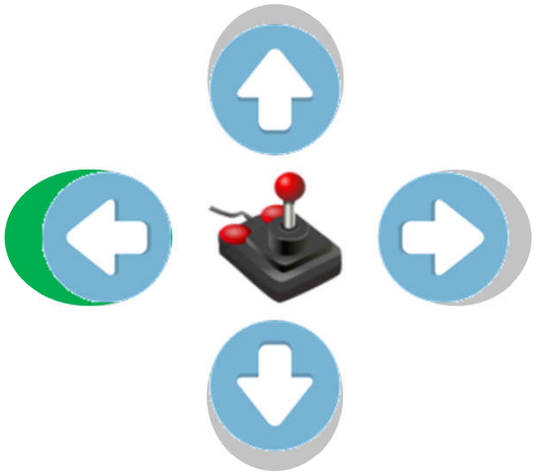
Policy

(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')

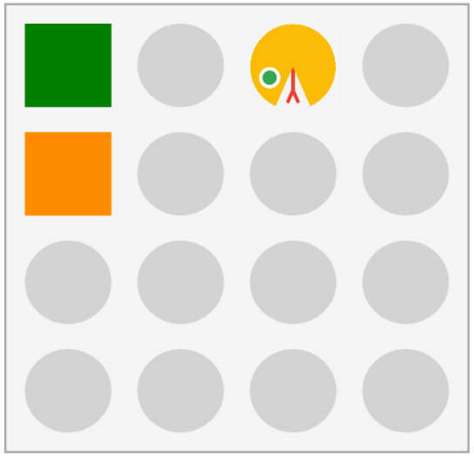
Episode Over

1 2 3 4 5 6 7 Step #

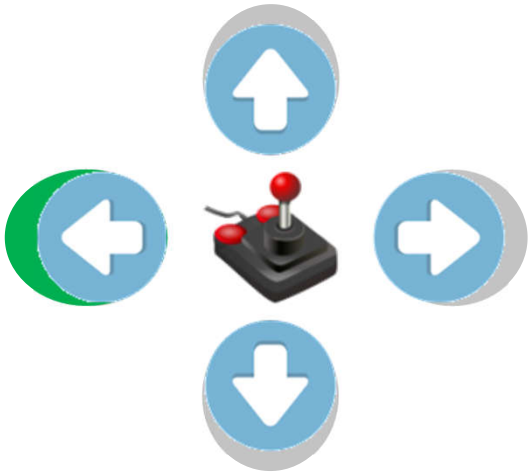
Episode 1 : improve 1st policy for state in step 1

Policy
(rules learned, how to play the game)

State



Action from Policy



49 (=-1 +100 -50)

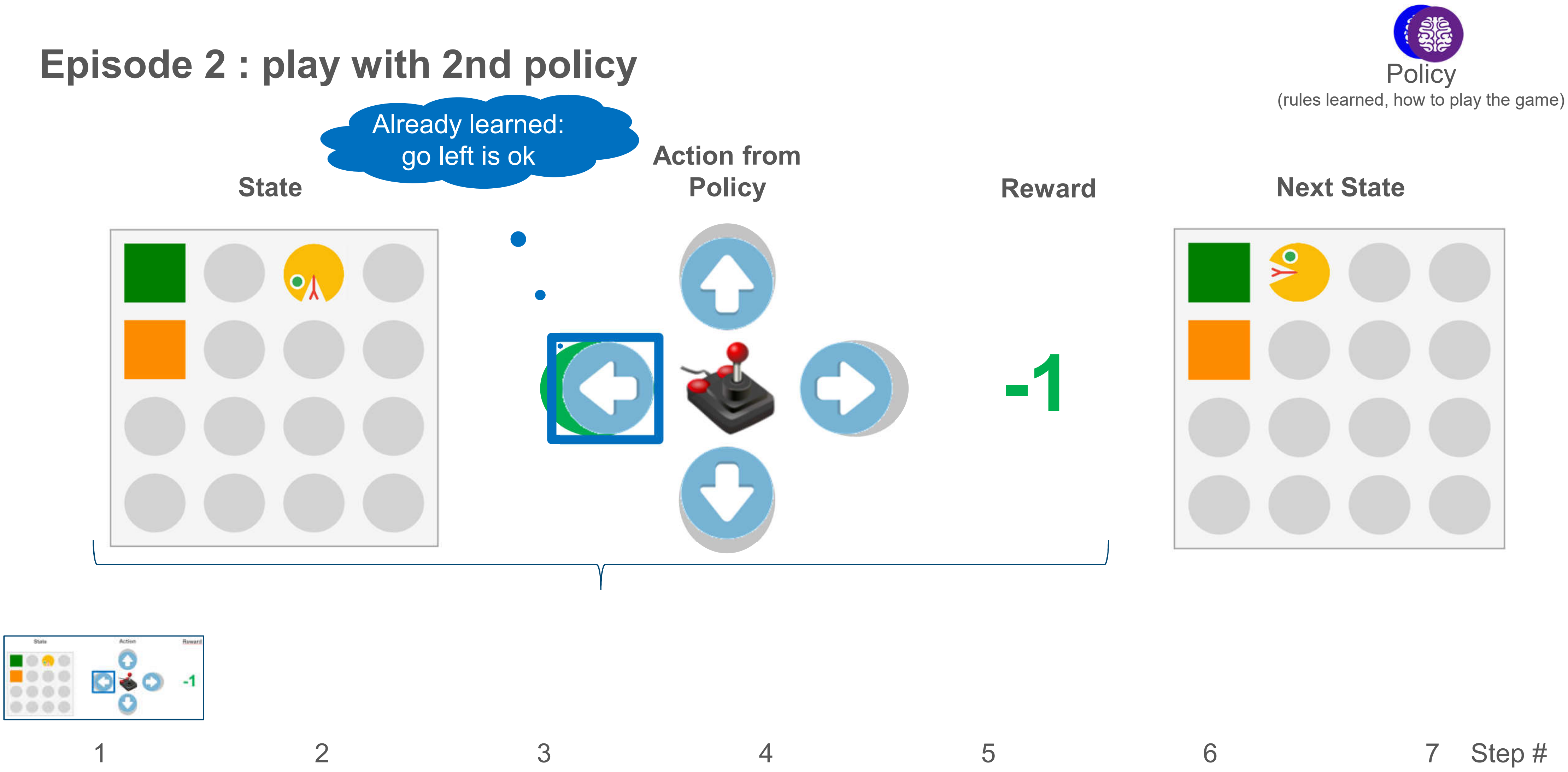
Future Reward

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

Episode Over

1 2 3 4 5 6 7 Step #

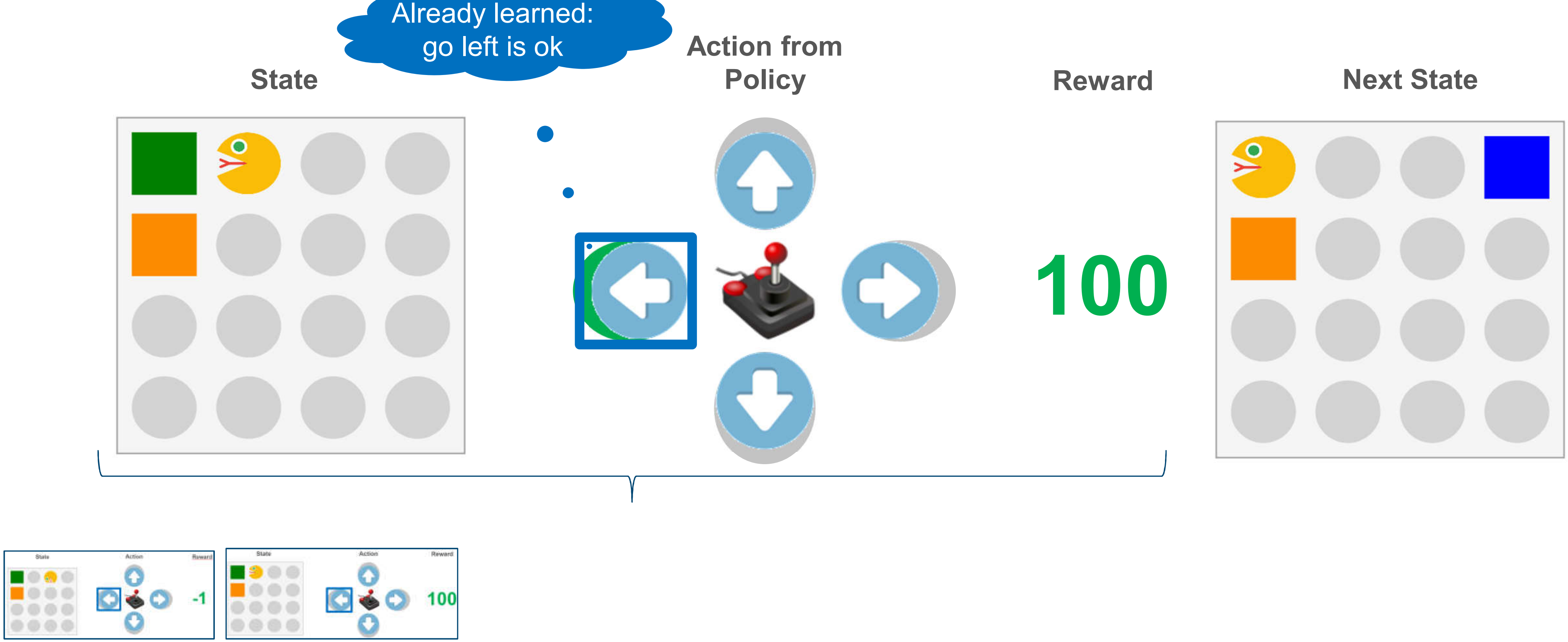
Episode 2 : play with 2nd policy



GEBERIT

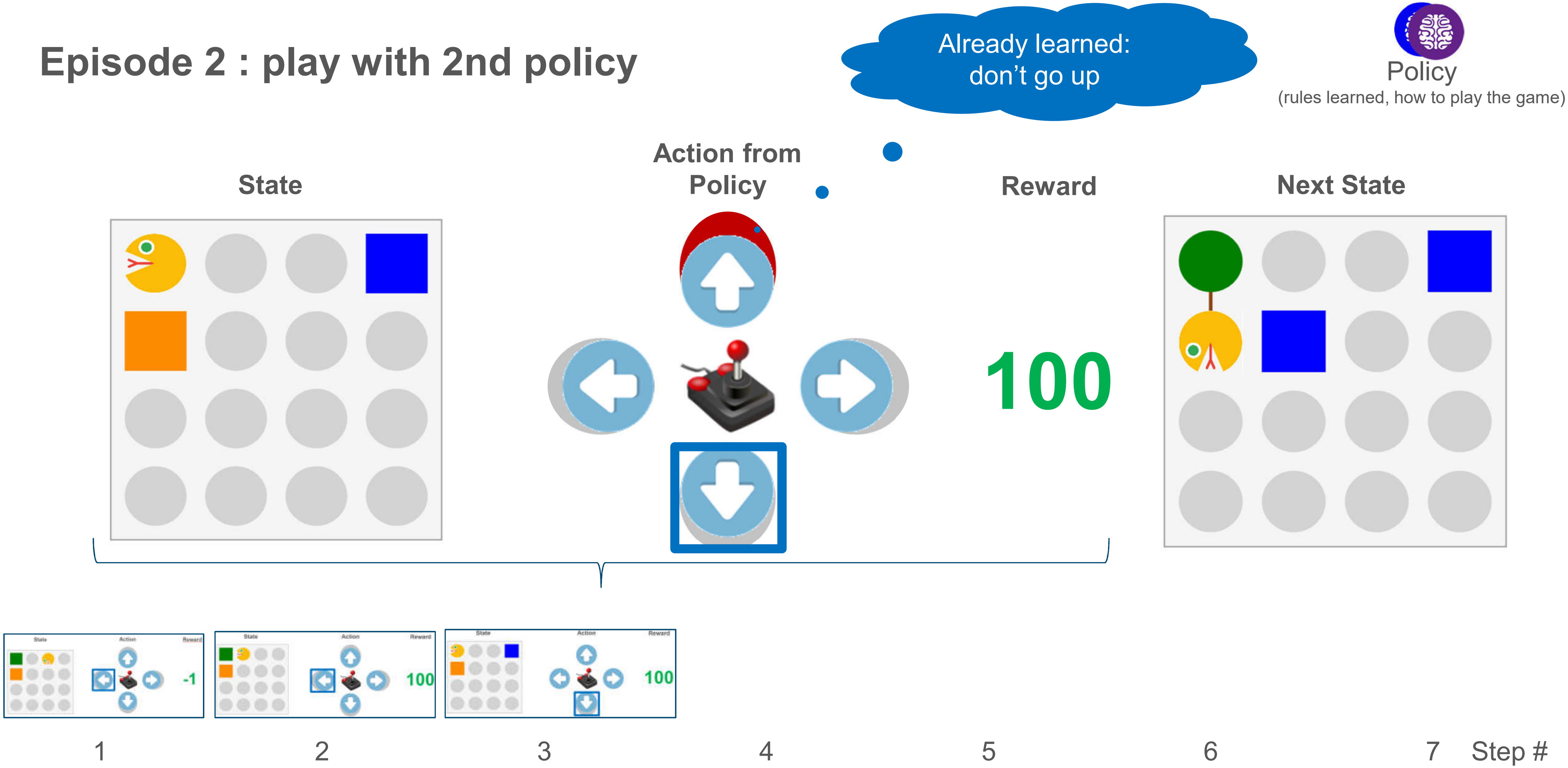
Episode 2 : play with 2nd policy

Policy
(rules learned, how to play the game)

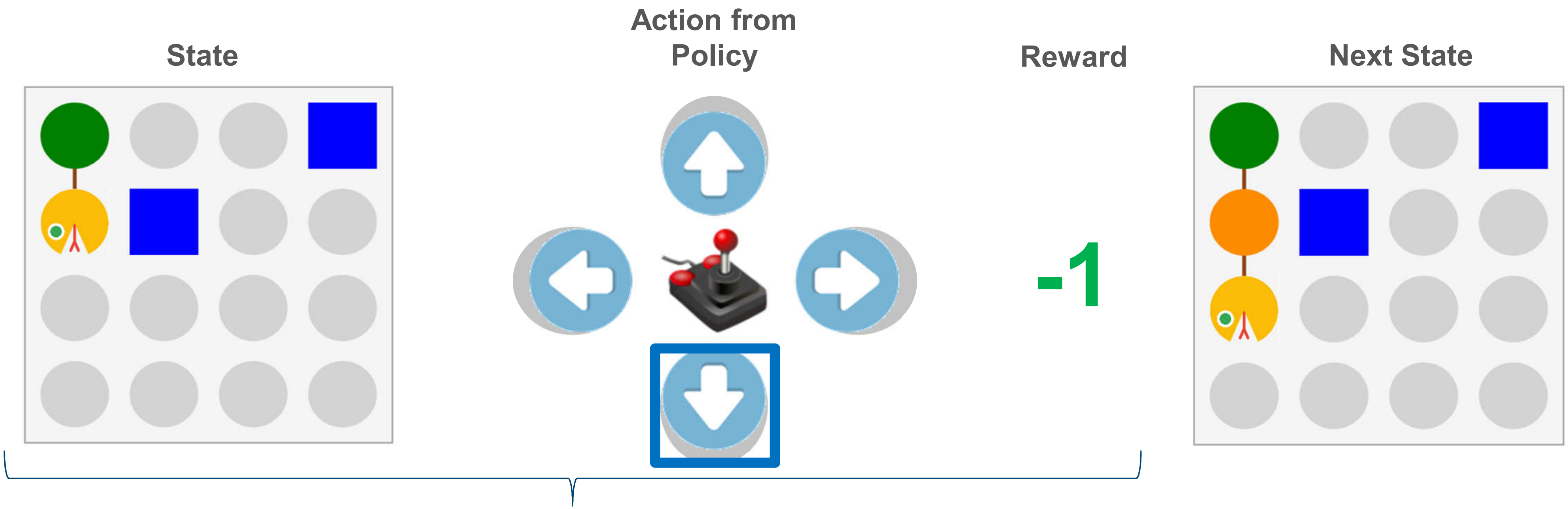


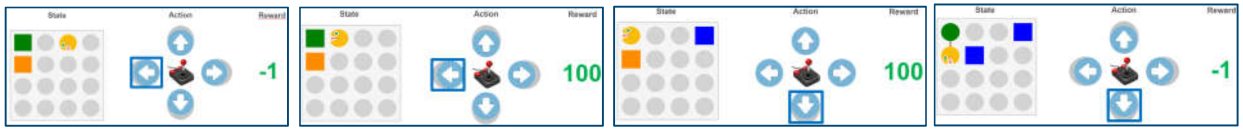
1 2 3 4 5 6 7 Step #

Episode 2 : play with 2nd policy



Episode 2 : play with 2nd policy

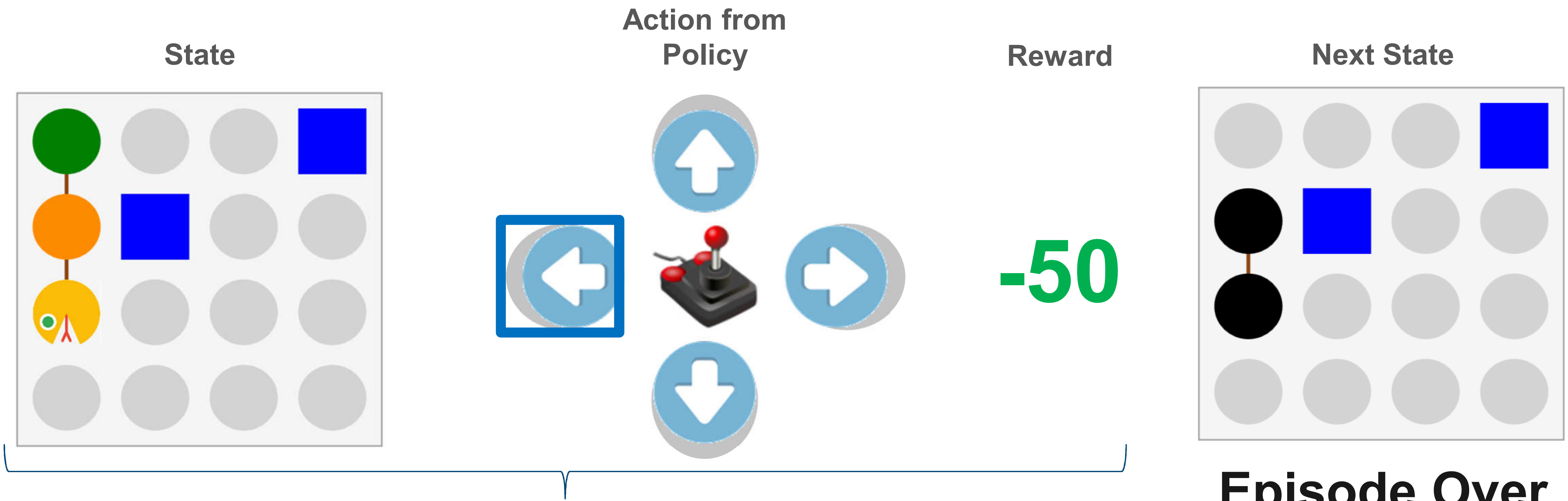




1 2 3 4 5 6 7 Step #

Episode 2 : play with 2nd policy

Policy
(rules learned, how to play the game)



Episode Over

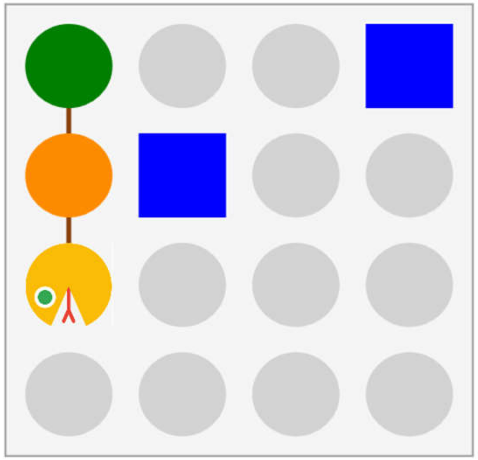
Episode Over

1 2 3 4 5 6 7 Step #

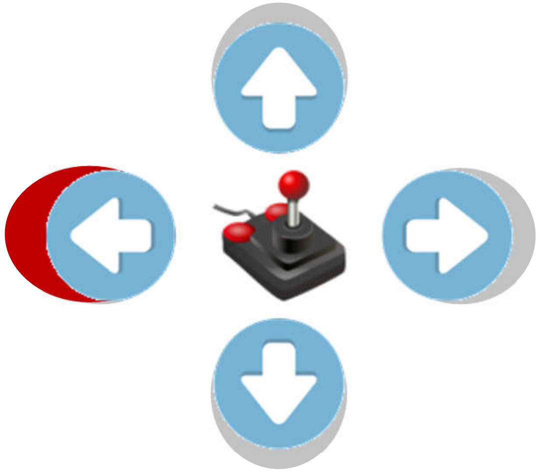
Episode 2 : improve 2nd policy for state in step 5

Policy
(rules learned, how to play the game)

State



Action from Policy



-50 (=-50)

Future Reward

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

Episode Over

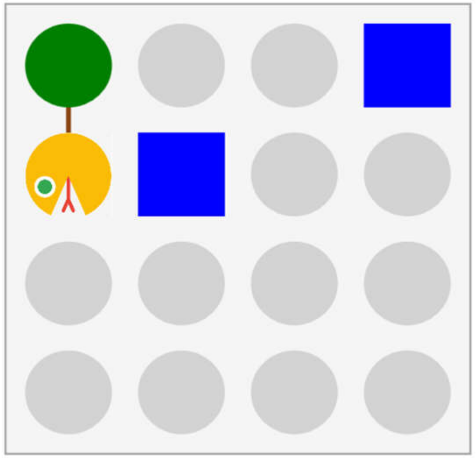
1 2 3 4 5 6 7 Step #

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

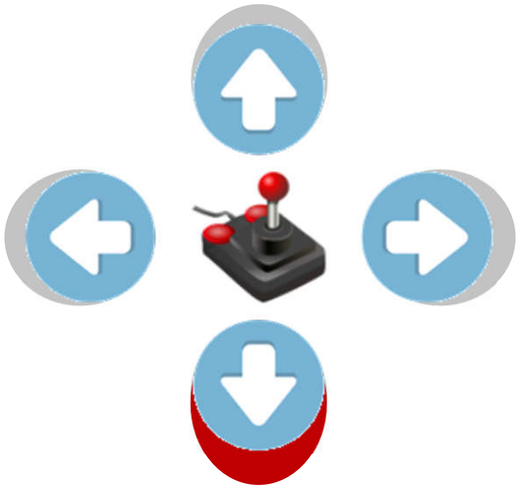
Policy

(rules learned, how to play the game)

State



Action from Policy



-51 (=-1-50)

Future Reward

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

Episode Over

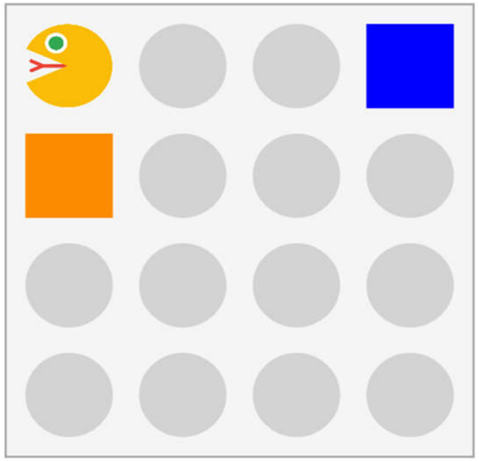
1 2 3 4 5 6 7 Step #

GEBERIT

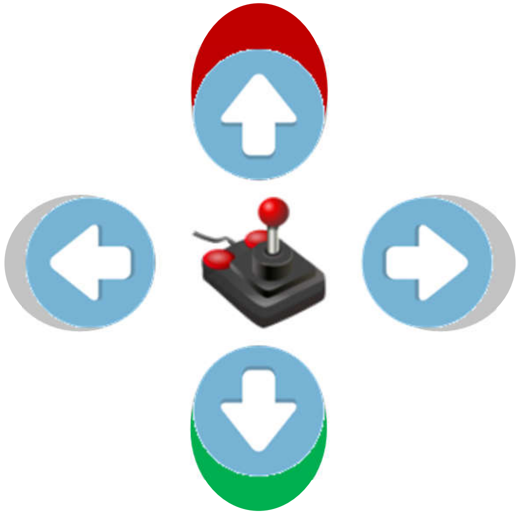
Episode 2 : improve 2nd policy for state in step 3

Policy
(rules learned, how to play the game)

State



Action from Policy



49 (=+100-1-50)

Future Reward

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

Episode Over

1 2 3 4 5 6 7 Step #

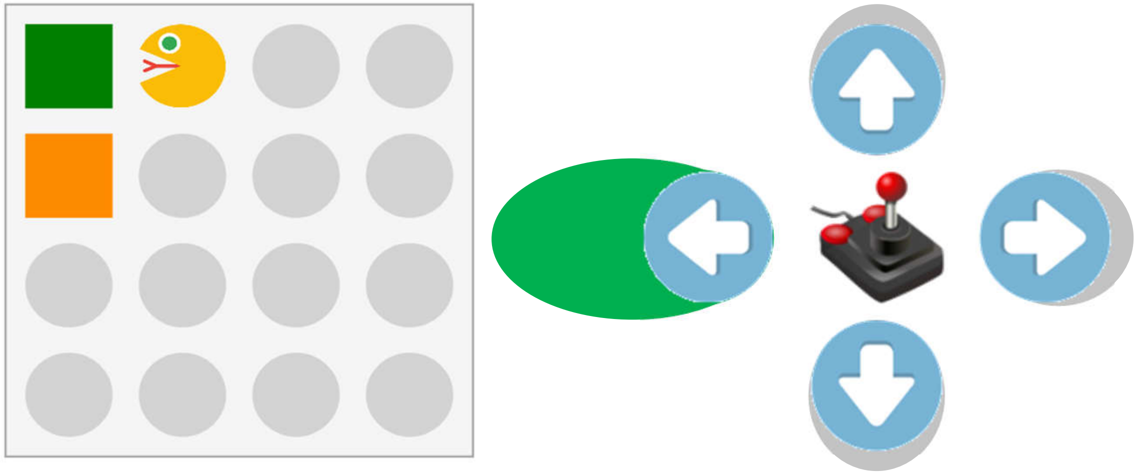
GEBERIT

Policy

(rules learned, how to play the game)

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

State | **Action from Policy**



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

Future Reward

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

Episode Over

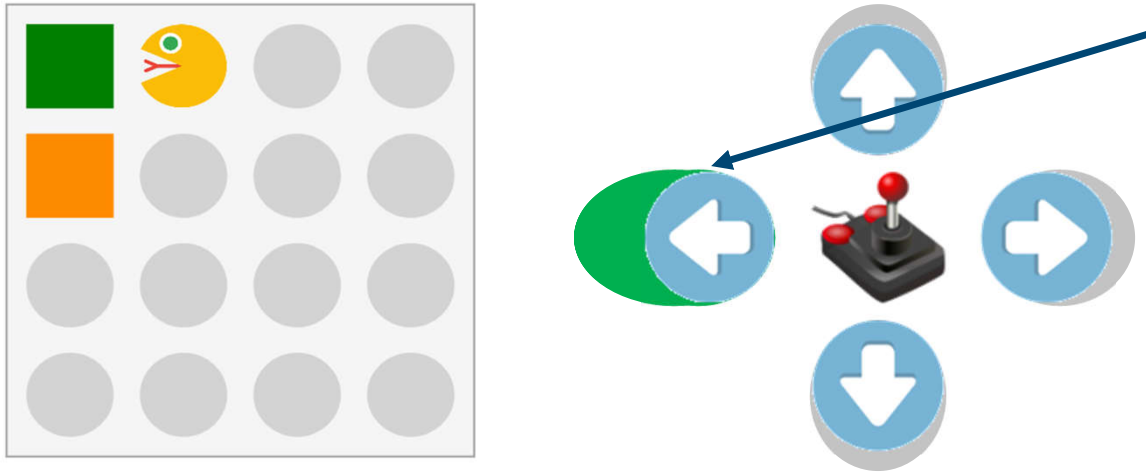
1 2 3 4 5 6 7 Step #

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

Policy
(rules learned, how to play the game)

State

Action from Policy



= 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')

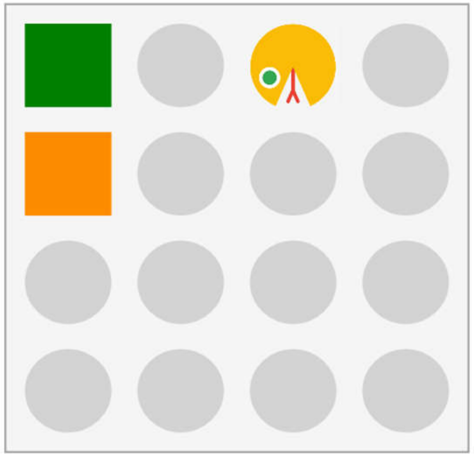
Episode Over

1 2 3 4 5 6 7 Step #

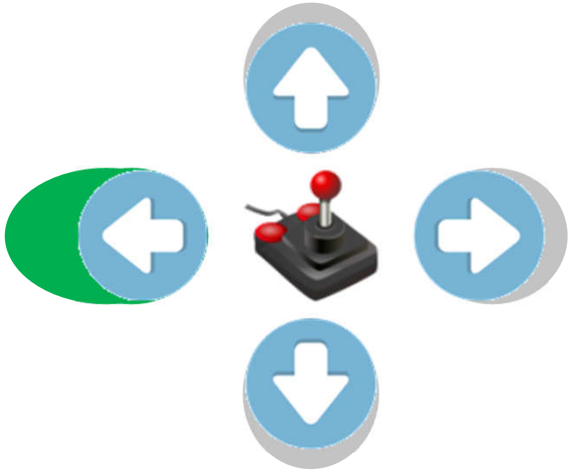
Episode 2 : improve 2nd policy for state in step 1

Policy
(rules learned, how to play the game)

State



Action from
Policy



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

Future Reward

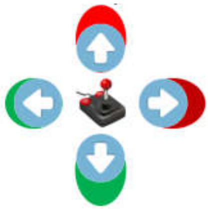
(sum of all rewards from current state until 'game over')

Episode Over

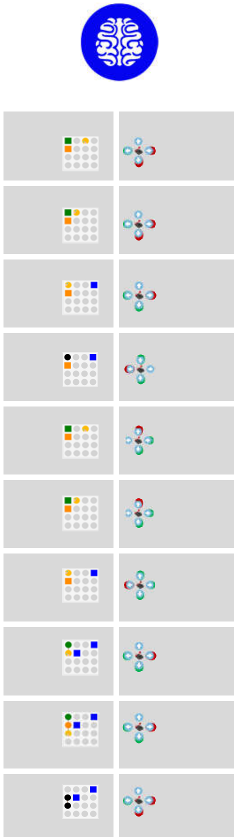
1 2 3 4 5 6 7 Step #

GEBERIT

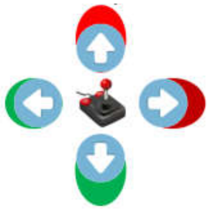
So far …..

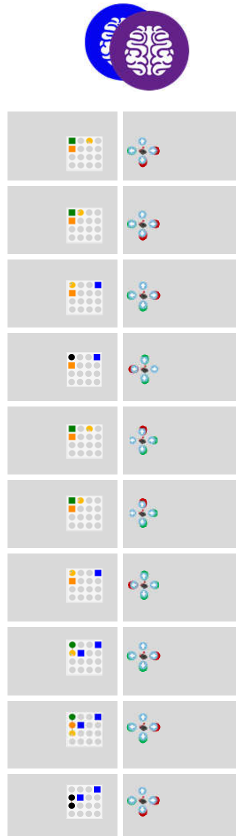


a policy is a map from states to action probabilities



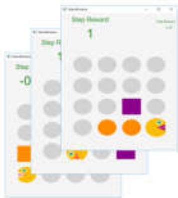
…updated by the reinforcement learning algorithm

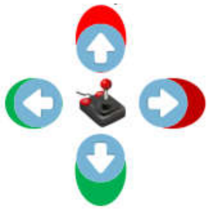


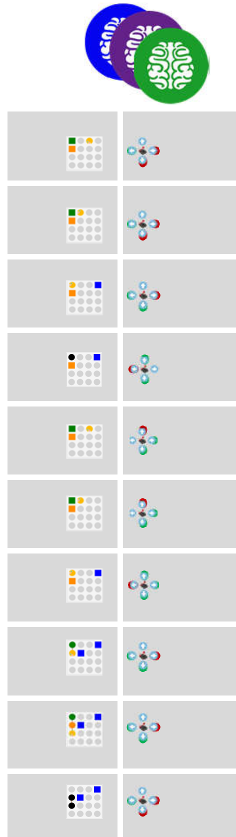


a policy is a map from states to action probabilities

…updated by the reinforcement learning algorithm

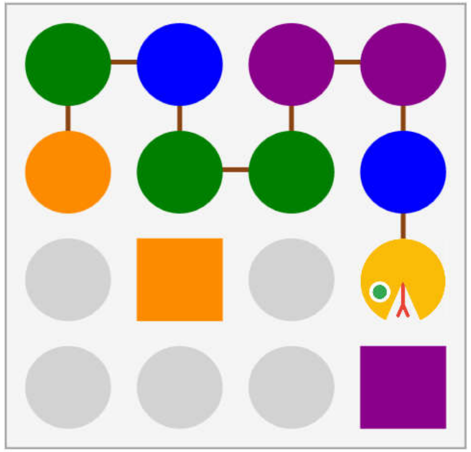


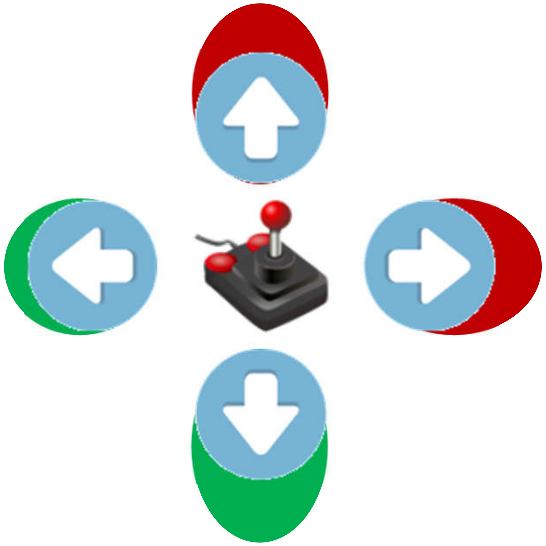


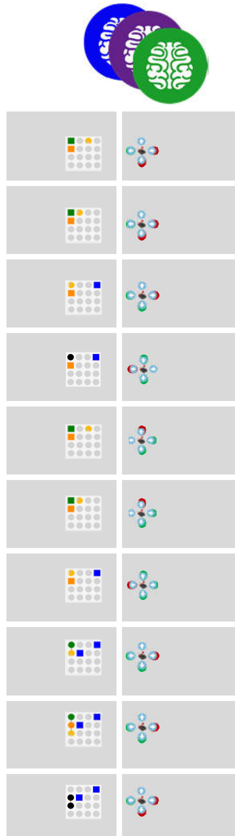


a policy is a map from states to action probabilities

After many, many episodes, for each state…







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 ($state_1$,$action_1$,$reward_1$),($state_2$,$action_2$,$reward_2$),…. for episode

For each step i

compute $FutureReward_i$ = $reward_i$ + $reward_{i+1}$ +…

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 ($state_1$,$action_1$,$reward_1$),($state_2$,$action_2$,$reward_2$),…. for episode

For each step i

compute $FutureReward_i$ = $reward_i$ + $reward_{i+1}$ +…

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”

GEBERIT

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 ($state_1$,$action_1$,$reward_1$),($state_2$,$action_2$,$reward_2$),…. for episode

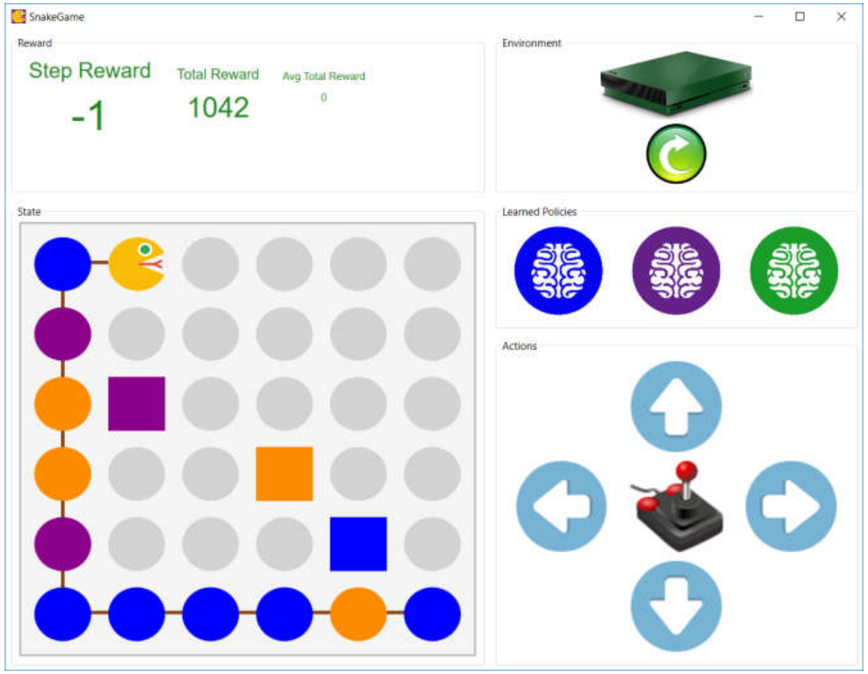
For each step i

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

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"

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

Table[state] = action probabilities

Do

NeuralNet(state) ~ action probablities

```
Initialize table with action probabilities for each state
Repeat
    play episode with policy given by table
    Record (state_1,action_1,reward_1),(state_2,action_2,reward_2),…. for episode
    For each step i
        compute FutureReward_i = reward_i + reward_{i+1} + …
        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"
```

Change to "play episode with policy given by NeuralNet"

Change to "update weights of NeuralNet"

Neural nets to the rescue

Policy
(rules learned, how to play the game)

Idea:

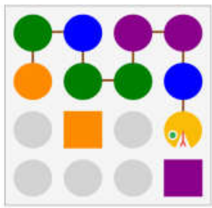
Replace lookup table with a neural network that approximates the action probabilities contained in the table

Instead of

Table[state] = action probabilities

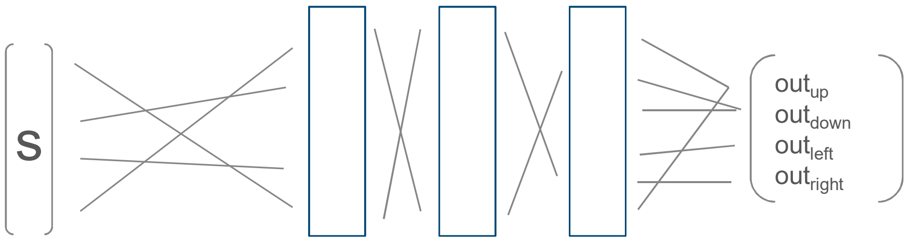
Do

NeuralNet(state) ~ action probablities



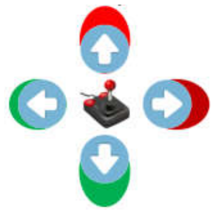
Encode state as vector

~



Apply neural network with “the right” weights

~



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

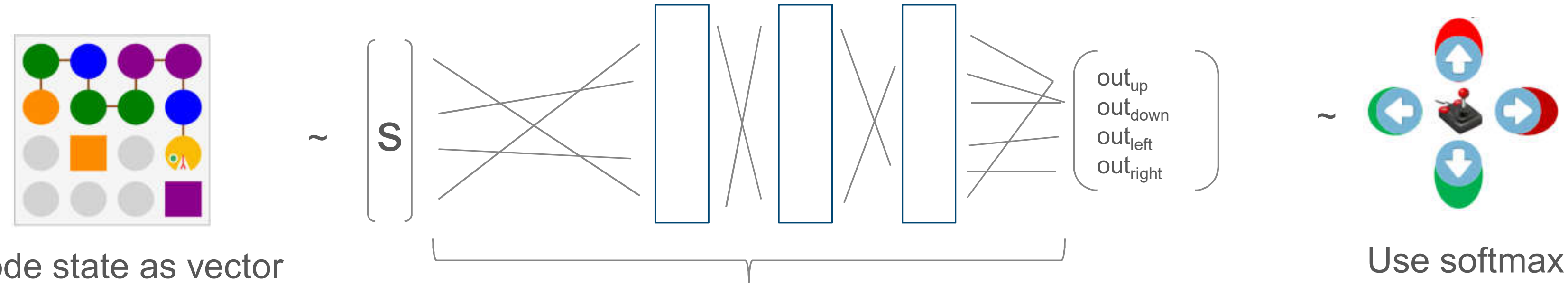
  Record $(state_1, action_1, reward_1), (state_2, action_2, reward_2), \ldots$ for episode(s)

  For each step i

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

    Update weights W

      W = W +   ????



GEBERIT

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

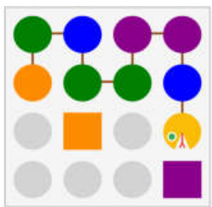
  Record ($state_1$,$action_1$,$reward_1$),($state_2$,$action_2$,$reward_2$),…. for episode(s)

  For each step i

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

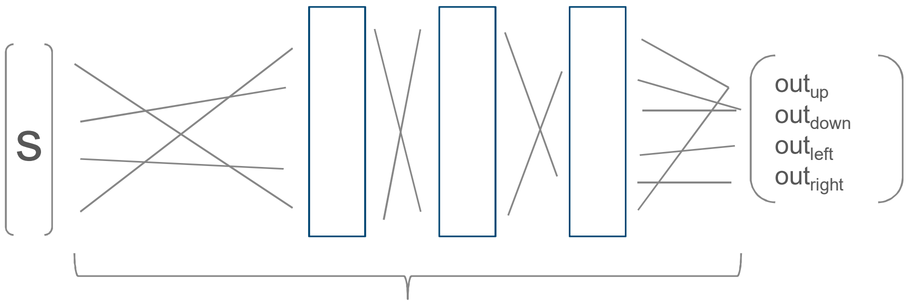
    Update weights W

      W = W + ????



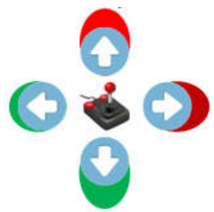
Encode state as vector

~



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

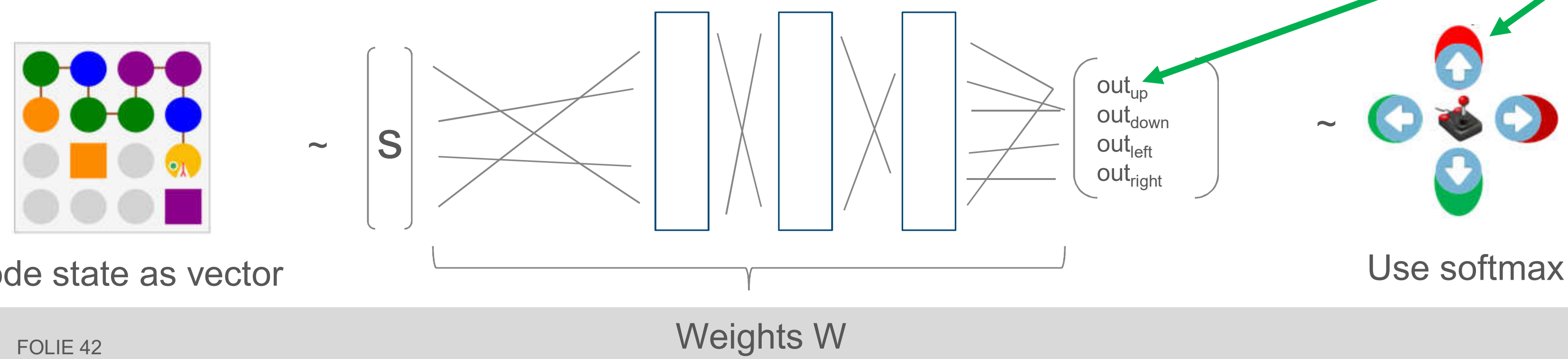
Record ($state_1$,$action_1$,$reward_1$),($state_2$,$action_2$,$reward_2$),…. for episode(s)

For each step i

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

Update weights W

$$W = W + \underbrace{????}_{\text{Increases } out_i}$$



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

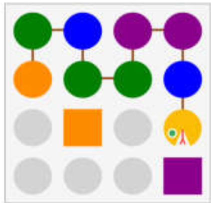
Record $(state_1, action_1, reward_1), (state_2, action_2, reward_2), \ldots$ for episode(s)

For each step i

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

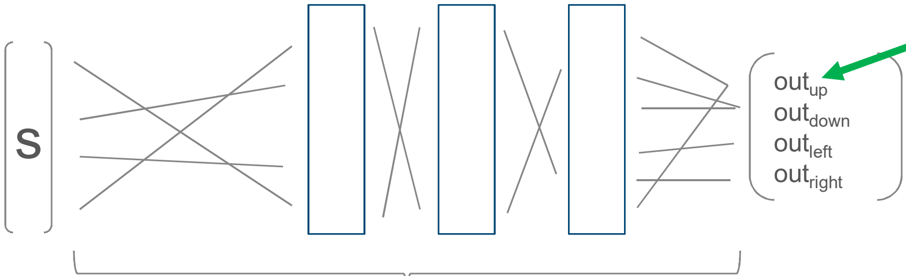
Update weights W

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



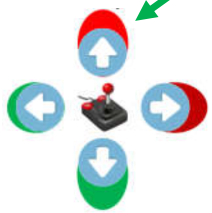
Encode state as vector

~



Weights W

~



Use softmax

Policy Gradient

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

= f(W)

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

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

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

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

Policy Gradient

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

= f(W)

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

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

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

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

$$\nabla_W E_{\tau\sim p_W}[R(\tau)] = \nabla_W \int_\tau p_W(\tau)\cdot R(\tau) = \int_\tau \nabla_W p_W(\tau)\cdot R(\tau)$$

$$= \int_\tau p_W(\tau)\cdot \frac{\nabla_W p_W(\tau)}{p_W(\tau)} \cdot R(\tau) = \int_\tau p_W(\tau) \cdot \nabla_W \log p_W(\tau)\cdot R(\tau)$$

$$= \underbrace{E_{\tau\sim p_W}[\nabla_W \log p_W(\tau)\cdot R(\tau)]}_{\text{good news}}$$

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

Policy Gradient

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

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

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

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

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

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

W = W + alpha * Gradient_W (neuralNet_W(state_i, action_i)) * FutureReward_i

What for ?

no feasible, deterministic algorithm

What for ?

Traditional Heuristics

Classic Machine Learning

Reinforcement Learning

Automatic solution found in 93.4%

The challenges

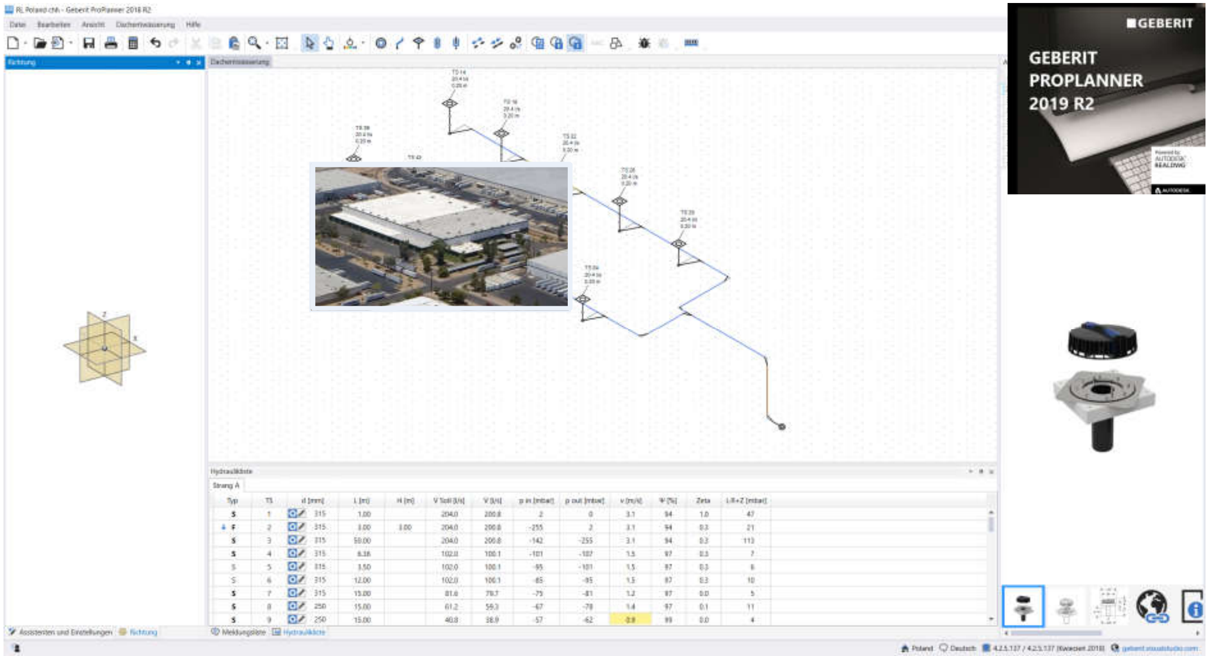
manage the water level on the roof
control & steer the water flow
find the right dimensions
save & reliable

GEBERIT


GEBERIT

Finding the right dimensions

Finding the „right“ dimensions: demo



GEBERIT

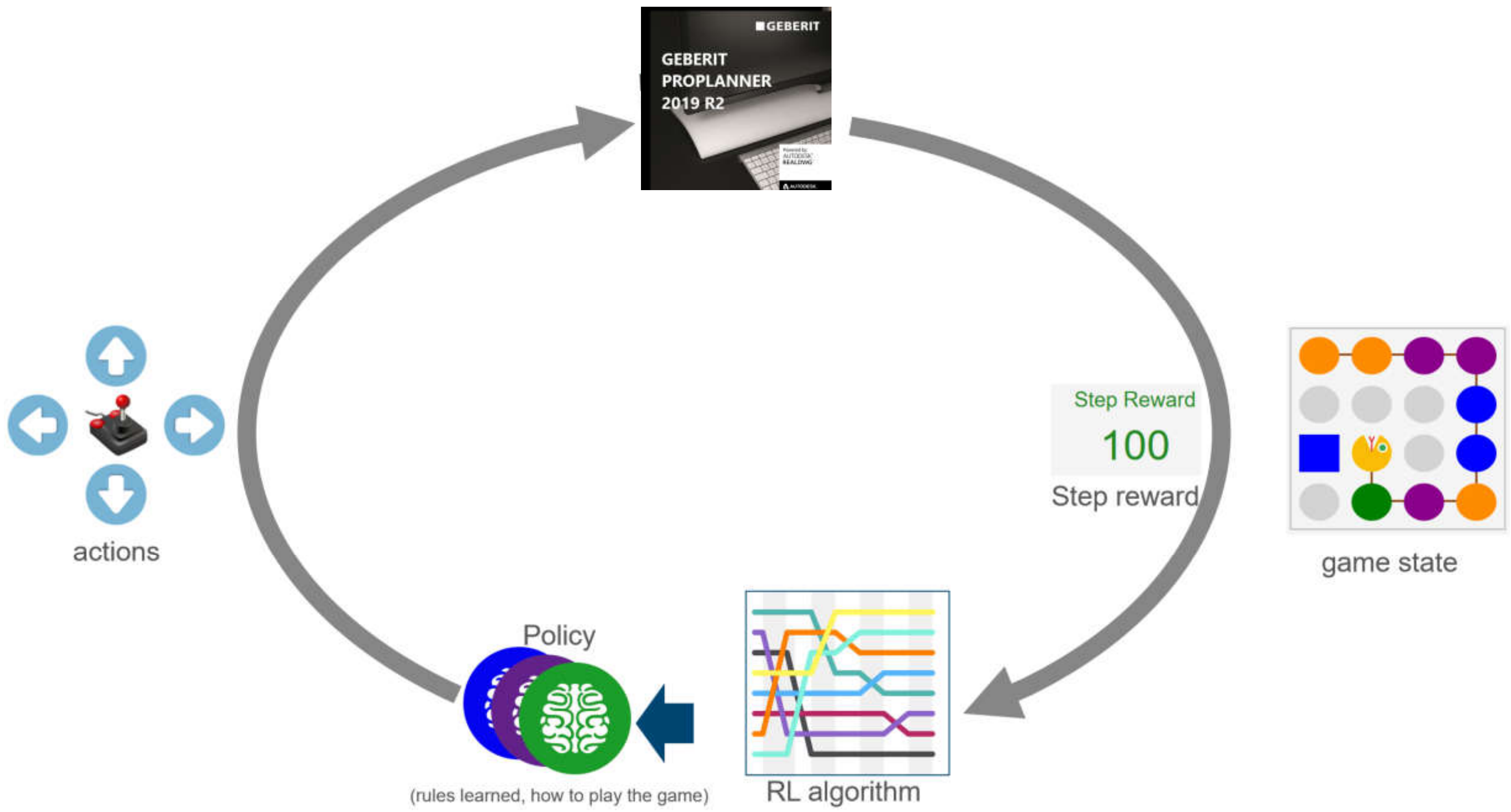
What if…

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

Figure 1: Magic Mart Storefront, Credit: Mike Kalasnik, Flickr. Public Use. No Changes Made.

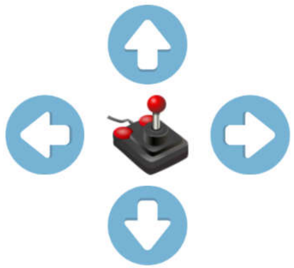
Figure 4: Magic Mart scene at the time of SGH arriving to the scene. Courtesy, SGH. Inc.

Turning the problem into a game

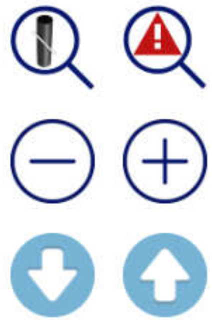


Designing the Action-Space

Snake game



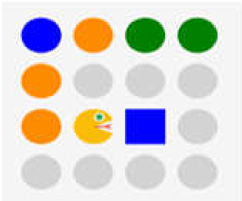
Roof drainage systems



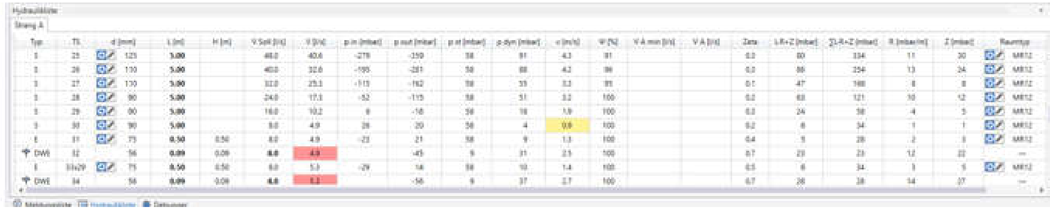
- What actions would a human expert like to have ?
- Are theses actions sufficient ?
- Would more / other actions be helpful ?
- Can we drop any actions ?

Designing the State-Space

Snake game



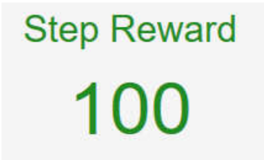
Roof drainage systems



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

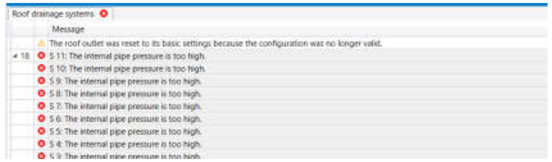
Designing the Reward Function

Snake game



Fruit	100
Death	-50
Success	1000
Step	-1

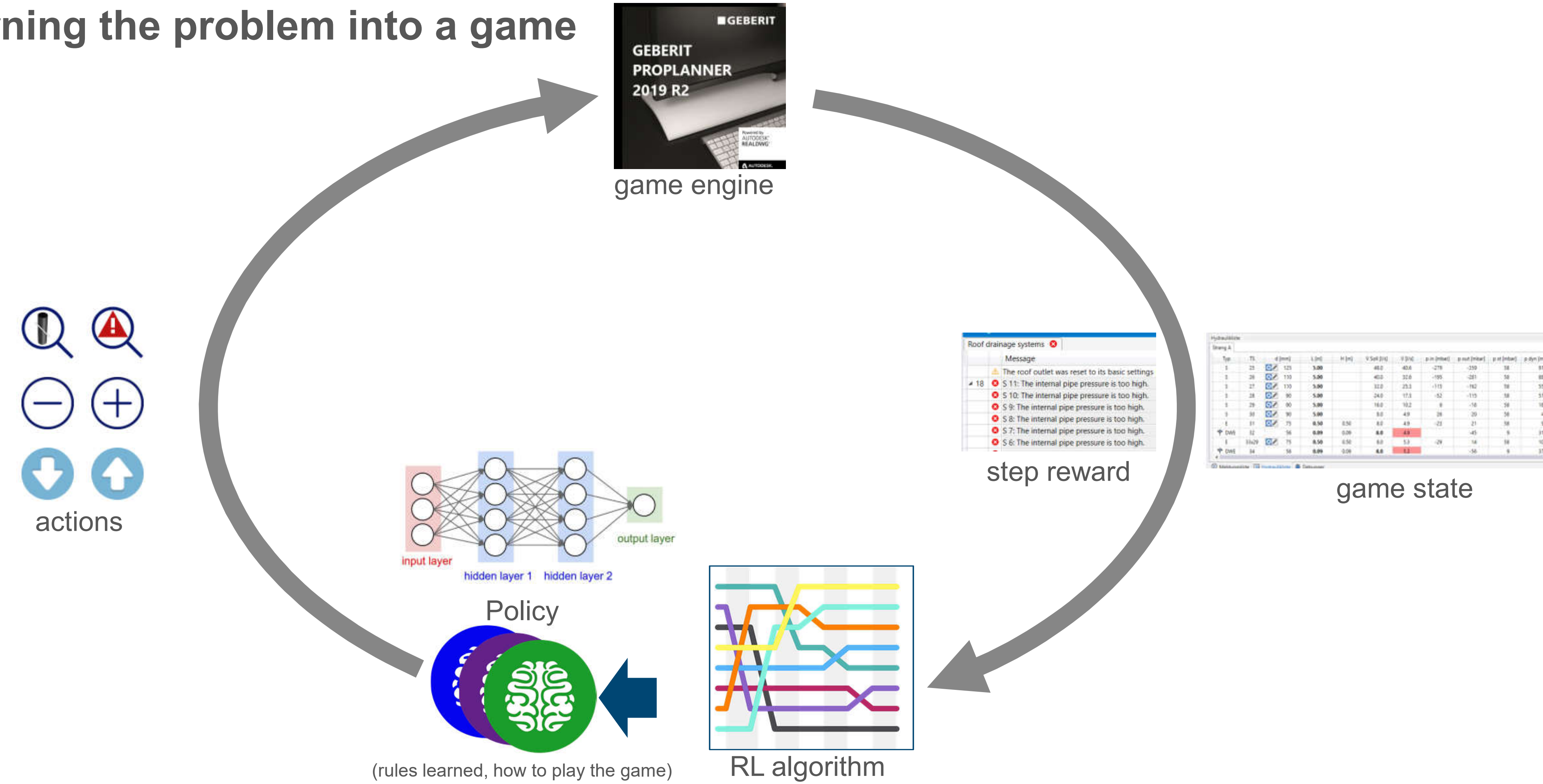
Roof drainage systems



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

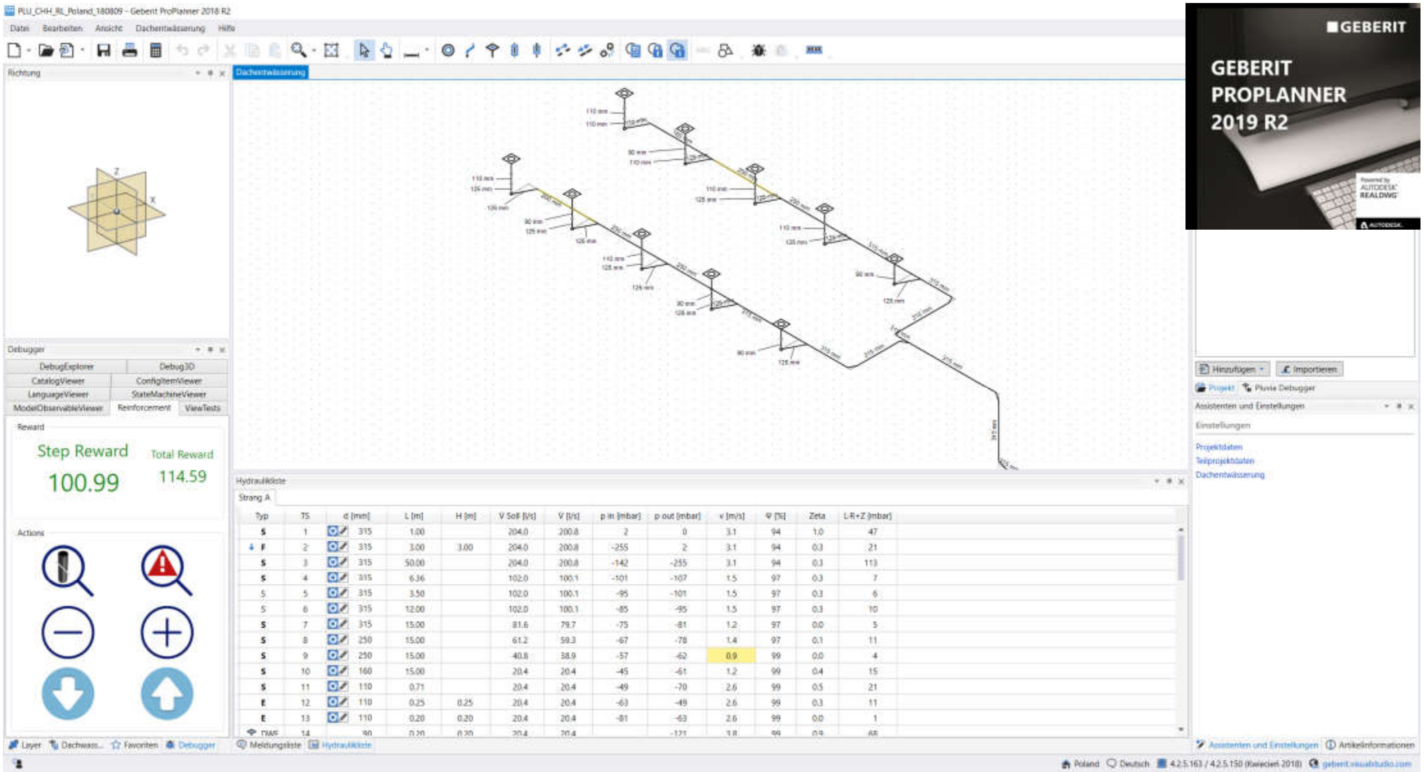
- How would you rate the result of an expert ?
- As simple as possible
- Positive feedback during the game
- Beware of “surprising policies”
- Game over if TotalReward too low

Turning the problem into a game



GEBERIT

Finding the dimensions with reinforcement learning: demo



Hydraulics Calculation Pipeline

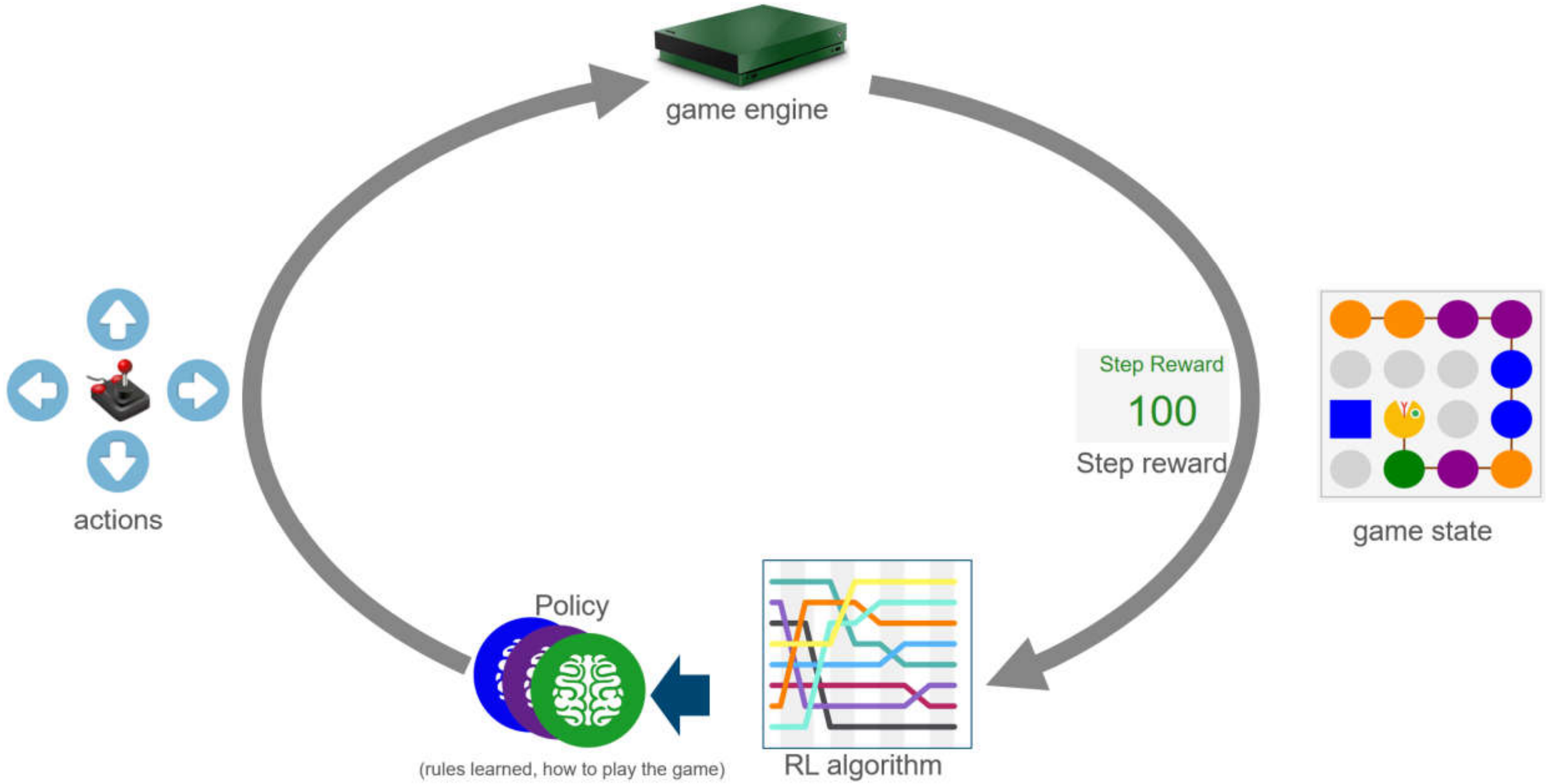
Traditional Heuristics → Classic Machine Learning → Reinforcement Learning

Automatic solution found in 93.4%

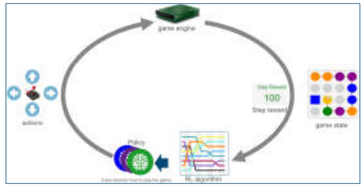
Finds a solution in 70.7% of the remaining 6.6%

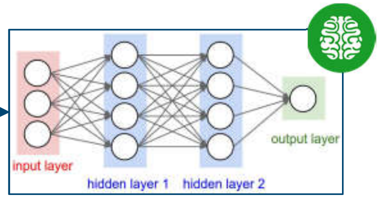
Automatic solution found in 98.1%

Implementing Reinforcement Learning: Training



Implementing Reinforcement Learning: Training



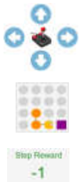


Main

RL-Algorithm
(eg OpenAI baselines)

Gym Framework
(OpenAI)

$(action_{space}, state_{space}) =$ Init
(reward, state, done) = Step(action)
Reset



Environment
(Game Engine)

ProPlanner
.Net

Libraries

Tensorflow

GEBERIT

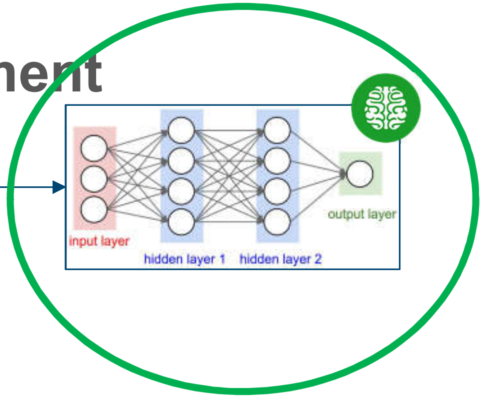
Implementing Reinforcement Learning: Deployment

Main

RL-Algorithm
(eg OpenAI baselines)

Gym Framework
(OpenAI)

Environment
(Game Engine)

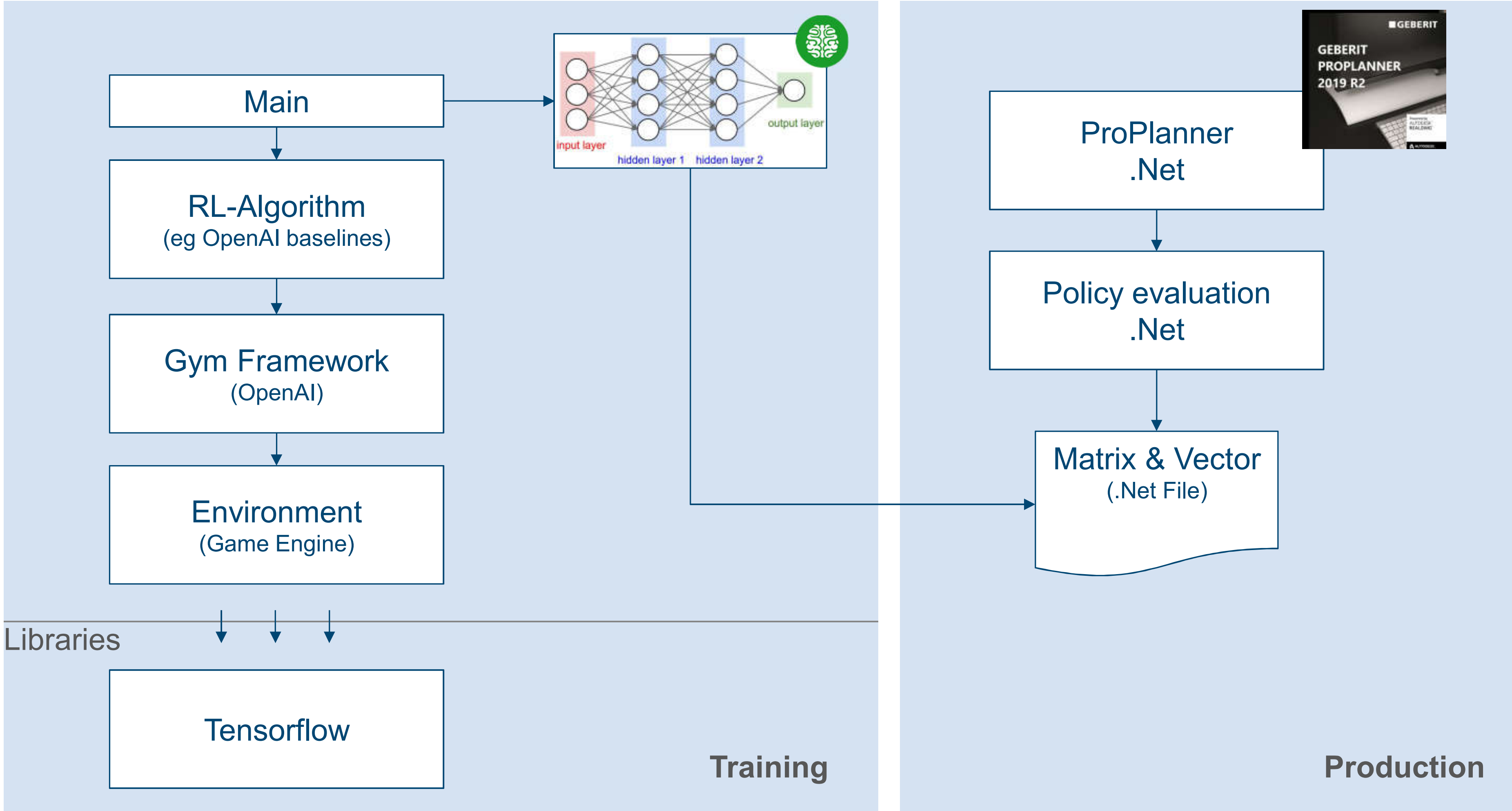


ProPlanner
.Net

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!

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.

Christian.Hidber@bSquare.ch

W +41 44 260 54 00

M +41 76 558 41 48

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

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/