Solving Montezuma's Revenge

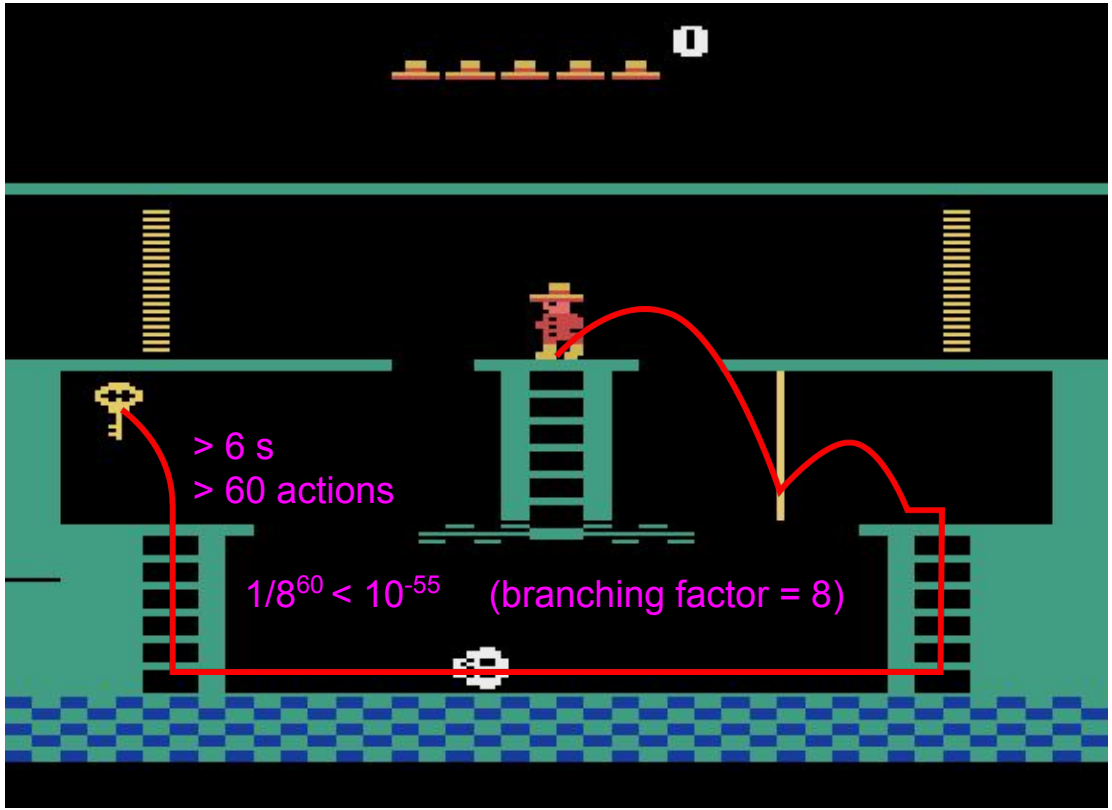
with Planning and Reinforcement Learning

upf. Universi
Pompeu
Barcelor

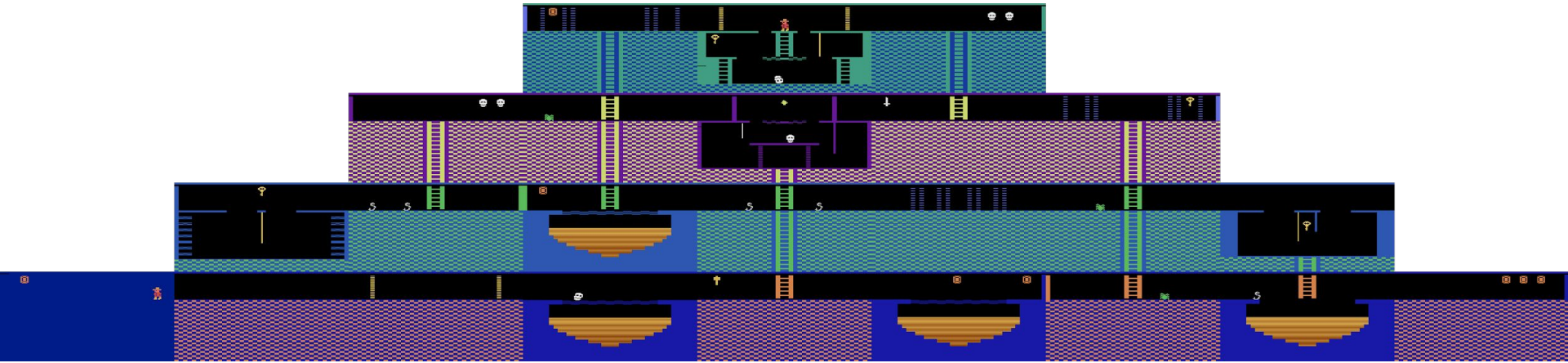
Adrià Garriga
Supervisor: Anders Jonsson

Introduction: Why Montezuma's Revenge?

Sparse rewards.



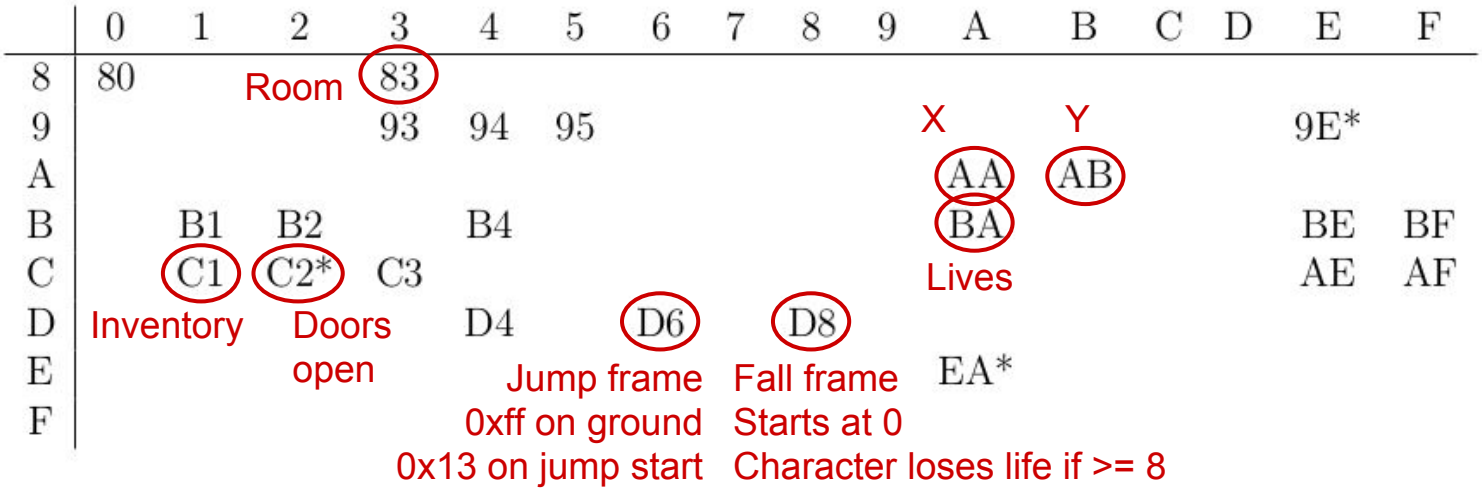
Introduction: The game.



Learning state of the art: 3439 score (Bellemare et. al., 2016)

Planning state of the art: 540 score (Lipovetzky, Ramirez and Geffner, 2015)

Reverse Engineering: known memory layout



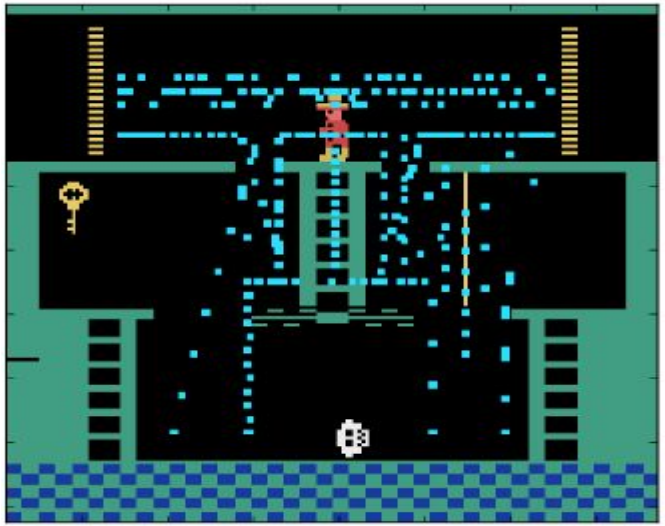
http://problemkaputt.de/2k6specs.htm

Planning: Iterated Width

- IW(*n*): BrFS, pruning when no new tuples of size *n*. That is, *novelty* ≤ n

State	f_1	f_2	f_3	Novel tuples	Novelty
1	F	F	F	$\langle\rangle, \langle f_1\rangle, \langle f_2\rangle, \langle f_3\rangle, \langle f_1, f_2\rangle,$ $\langle f_2, f_3\rangle, \langle f_1, f_3\rangle, \langle f_1, f_2, f_3\rangle$	0
2	F	T	F	$\langle f_2\rangle, \langle f_1, f_2\rangle, \langle f_2, f_3\rangle, \langle f_1, f_2, f_3\rangle$	1
3	T	T	F	$\langle f_1\rangle, \langle f_1, f_2\rangle, \langle f_1, f_3\rangle, \langle f_1, f_2, f_3\rangle$	1
4	T	F	F	$\langle f_1, f_2\rangle, \langle f_1, f_2, f_3\rangle$	2
5	T	F	T	$\langle f_3\rangle, \langle f_1, f_3\rangle, \langle f_2, f_3\rangle, \langle f_1, f_2, f_3\rangle$	1
6	T	F	T	none	$4 = \|F\| + 1$
7	F	F	T	$\langle f_1, f_2, f_3\rangle$	3

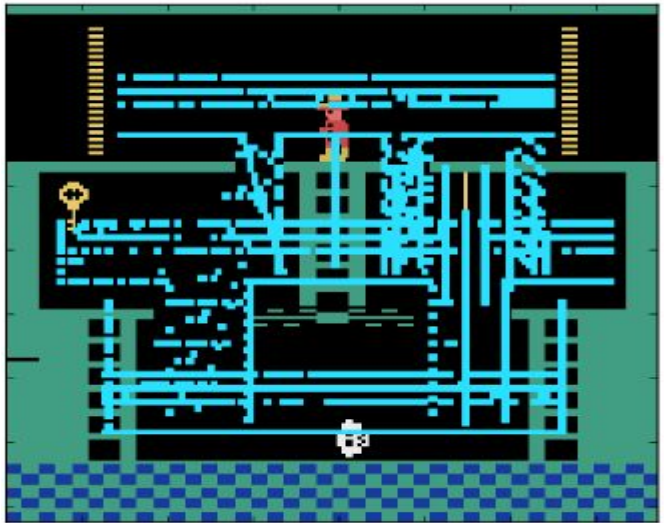
- IW(1) combined with greedy BestFS: 540 score (Lipovetzky, Ramirez and Geffner, 2015)



Visited positions during IW(1) search

Planning: Iterated Width

- Full IW(3):
  - 10^9 possibilities if RAM vector of booleans
  - $3 \cdot 10^{14}$ possibilities if RAM vector of bytes
- Solution:
  - IW(3) only on position (room, X, Y)
  - Give less priority to loss-of-life nodes
  - Obstacle passing
  - 1 reward for visiting new room
  - Randomly prune screens and keep the action sequence with best return



Visited positions during position-IW(3) search

Obstacle passing algorithm

for each child $c = f(s, a)\ \forall a \in \mathcal{A}$ **do**
  if CHECK-NOVELTY($seen\_tuples, pruned\_screens, c$) **then**
    UPDATE-NOVELTY($seen\_tuples, visited\_screens, pruned\_screens, c, p_r$)
    if c[0xBA] $<$ s[0xBA], this node loses a life **then**
      if ON-GROUND?(c) $\wedge$ ON-GROUND?(s) $\wedge$ LRUD?(a) **then**
        $obstacle\_child \leftarrow c$
      end if
      $q_l \leftarrow$ QUEUE-INSERT(q_l, c)
    else
      if FALLING?(c) $\wedge$ ON-GROUND?(s) $\wedge$ LRUD?(a) **then**
        $obstacle\_child \leftarrow c$
        if $a =$ DOWN **then** $q \leftarrow$ QUEUE-INSERT(q, c) **end if**
      else
        $q \leftarrow$ QUEUE-INSERT(q, c)
      end if
    end if
  end if
end for
if $obstacle\_child \neq \varnothing$ **then**
  $q, \leftarrow$ OBSTACLE-WAIT($f, obstacle\_child, q, max\_wait, max\_backtrack$)

function OBSTACLE-WAIT($f, obstacle\_child, q, max\_wait, max\_backtrack$)
  $p \leftarrow obstacle\_child$.PARENT, $p_p \leftarrow obstacle\_child$, $l_0 \leftarrow p$[0xBA]
  for $i \in [0, max\_backtrack)$ **do**
    $f^i(s, a) = f(f(\ldots f(s, a)\ldots, a), a)$, totalling $i + 1$ applications of f
    $n \leftarrow f^i(p,$ NOOP$)$
    if ON-GROUND?(n) $\wedge$ n[0xBA] $= l_0$ **then**
      for $i \in [0, max\_wait)$ **do**
        $n_{\text{test}} \leftarrow f^2(n, p_p.$ACTION$)$
        if ON-GROUND?(n_{test}) $\wedge$ n_{test}[0xBA] $= l_0$ **then**
          return QUEUE-INSERT(q, n)
        end if
        $n \leftarrow f(n,$ NOOP$)$
      end for
    end if
    $p_p \leftarrow p$, $p \leftarrow p.$PARENT
  end for
  return q
end function

Search starts

Life is lost after non-jump action

Go back 1

NOOP 1

Go back 2

NOOP 2

Go back 3

NOOP 3

Go back 4

NOOP 4

[...]

Go back 7

NOOP 7

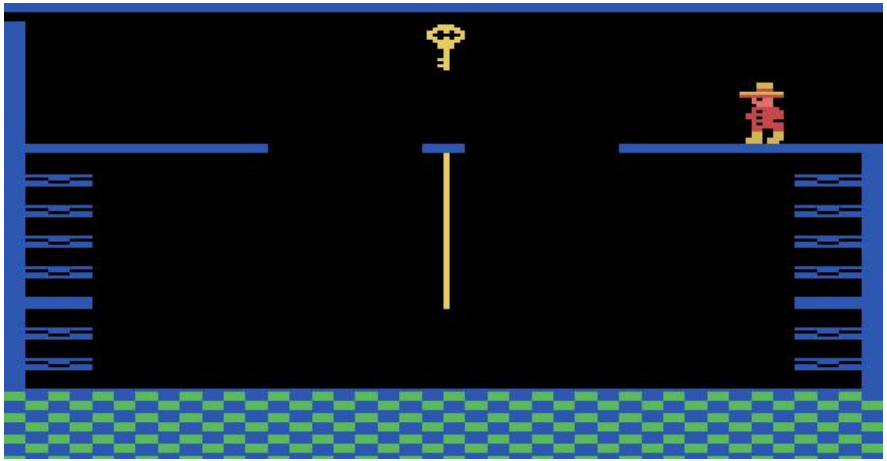
Keep testing if obstacle is passable

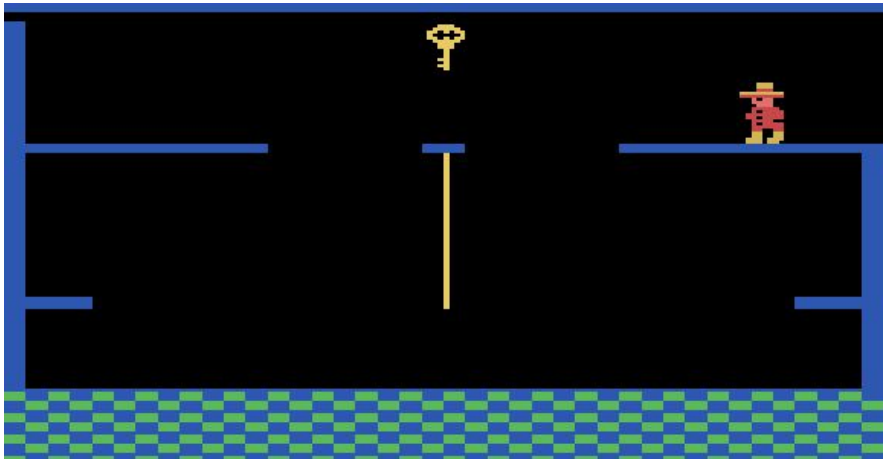
Left. Dead?

- Yes: back, NOOP
- No: continue search

Continue search

Not infallible





Random screen pruning

```
for each child c = f(s, a) ∀a ∈ A do
    if CHECK-NOVELTY(seen_tuples, pruned_screens, c) then
        UPDATE-NOVELTY(seen_tuples, visited_screens, pruned_screens, c, p_r)
        if c[0xBA] < s[0xBA], this node loses a life then
            if ON-GROUND?(c) ∧ ON-GROUND?(s) ∧ LRUD?(a) then
                obstacle_child ← c
            end if
            q_l ← QUEUE-INSERT(q_l, c)
        else
            if FALLING?(c) ∧ ON-GROUND?(s) ∧ LRUD?(a) then
                obstacle_child ← c
                if a = DOWN then q ← QUEUE-INSERT(q, c) end if
            else
                q ← QUEUE-INSERT(q, c)
            end if
        end if
    end if
end for
if obstacle_child ≠ ∅ then
    q, ← OBSTACLE-WAIT(f, obstacle_child, q, max_wait, max_backtrack)
```

```
function UPDATE-NOVELTY(seen_tuples, visited_screens, pruned_screens, ram, p_r)
    if ¬visited_screens[ram[0x83]] then
        visited_screens[ram[0x83]] ← true
        With probability p_r: pruned_screens[ram[0x83]] ← true
    end if
    seen_tuples[⟨ram[0xAA], ram[0xAB], ram[0x83]⟩] ← true
end function
function CHECK-NOVELTY(seen_tuples, pruned_screens, ram)
    return ¬(pruned_screens[ram[0x83]] ∨
                seen_tuples[⟨ram[0xAA], ram[0xAB], ram[0x83]⟩])
end function
```

- The trajectory with highest return is kept after each search

Visited spots every search step

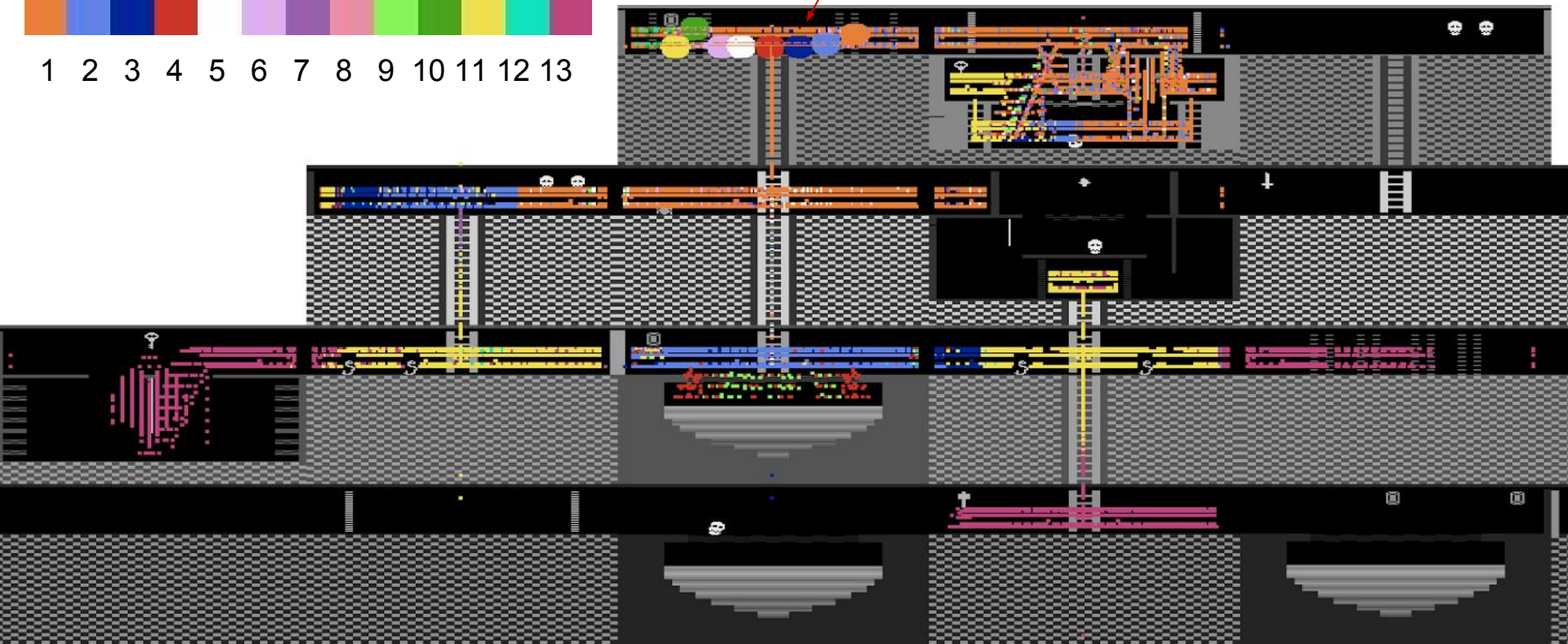
Score = 11200

Iteration number

1 2 3 4 5 6 7 8 9 10 11 12 13

Search start locations

$p_r = 0$



Visited spots every search step

Score = 14900

Iteration number

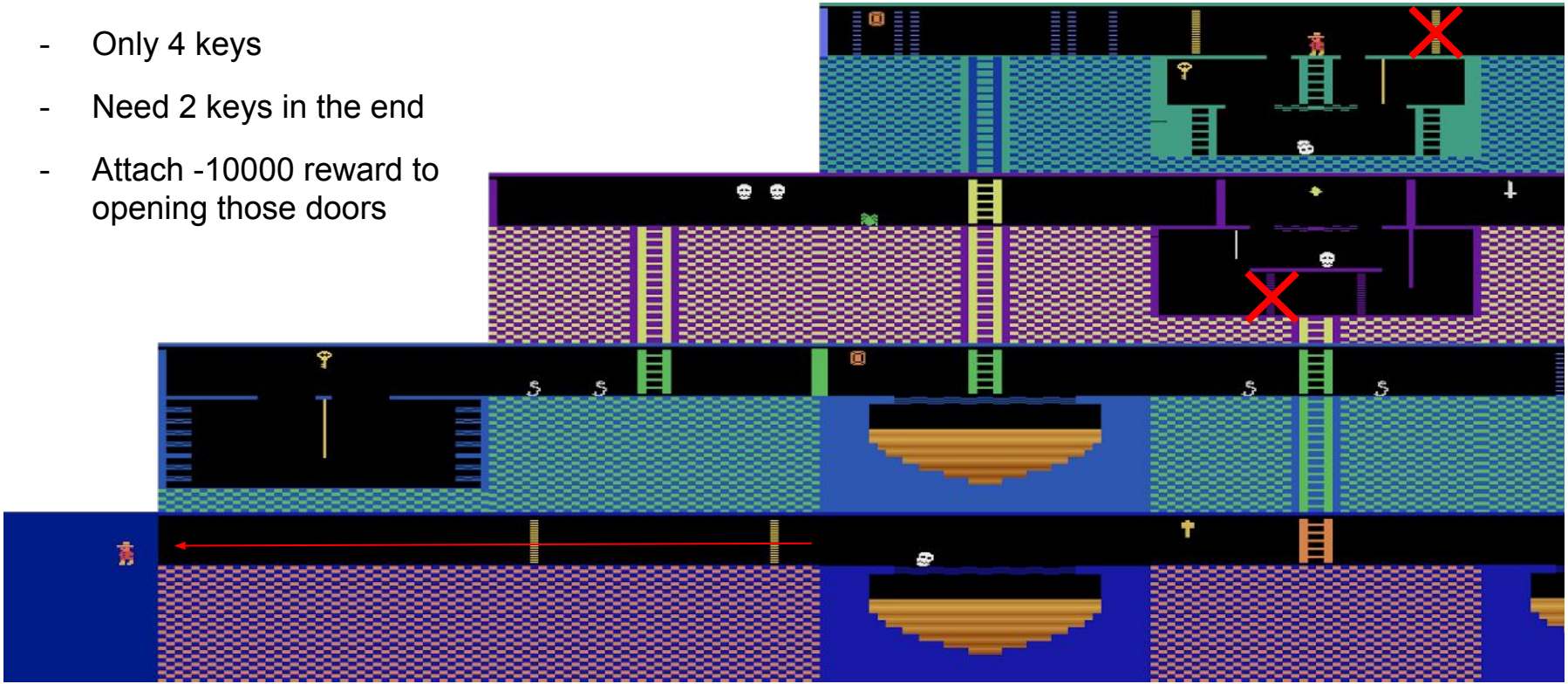
1 2 3 4 5 6 7 8 9 10 11 12 13

Search start locations

$p_r = 0.2$

The forbidden portals

- Only 4 keys
- Need 2 keys in the end
- Attach -10000 reward to opening those doors



Planning video

Learning: Tabular Sarsa

- Skull position: $v_1 = \text{RAM}(\text{0xAF}) - 0x16$, $v_1 \in [0, 51)$.
- Skull direction: $v_2 = 1$ initially, set to 0 if $v_1 = 0$, set to 1 if $v_1 = 50$, otherwise keep the same value as the last state.
- Joe's X: $v_3 = \text{RAM}(\text{0xAA})$, $v_3 \in [0, 256)$.
- Joe's Y: $v_4 = \text{RAM}(\text{0xAB})$, $v_4 \in [0, 256)$.
- Whether Joe has the key: $v_5 = \begin{cases} 0 & \text{if } \textsc{Bitwise-And}(\text{RAM}(\text{0xC1}), 0x1E) = 0 \\ 1 & \text{otherwise} \end{cases}$
- Whether Joe will lose a life upon touching the ground, or has already done so:
  $v_6 = \begin{cases} 0 & \text{if } \text{RAM}(\text{0xD8}) \geq 8 \vee \text{RAM}(\text{0xBA}) < 5 \\ 1 & \text{otherwise} \end{cases}$

$51 \cdot 2 \cdot 256 \cdot 256 \cdot 2 \cdot 2 =$

26 738 688 possible states

$26\,738\,688 \cdot 8 =$

213 909 504 action-state pairs

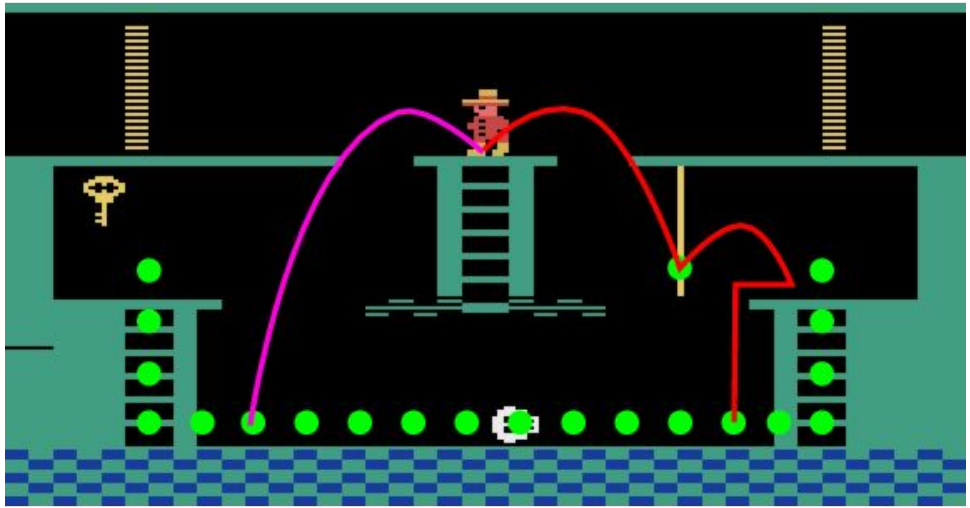
Q with 32-bit float:

$213\,909\,504 \cdot 4 / 10^6 \approx$ 855 MB

Shaping: Alleviate sparse rewards

- Add rewards to guide the agent in learning
- But shaping may make the agent follow a policy that is not optimal in the original setting
- Solution: potential-based shaping function (Ng, Harada, and Russell, 1999).

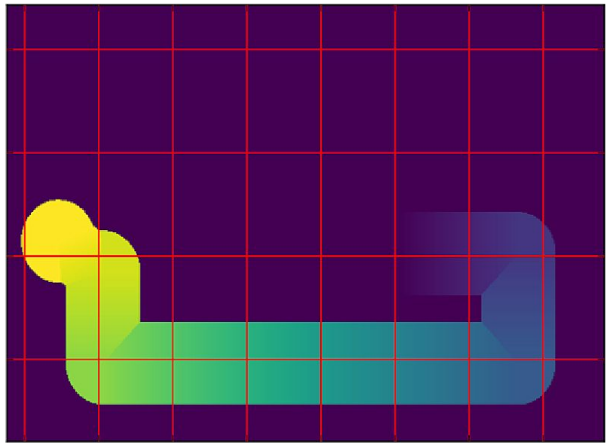
$$F(s, a, s') = \gamma \cdot \varphi(s') - \varphi(s)$$



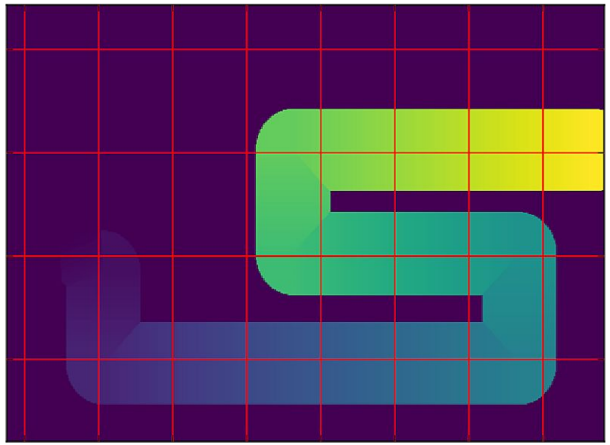
Undesirable shortcuts learned when the shaping is collectable reward “pills”. The agent collects all previous pills when touching one.

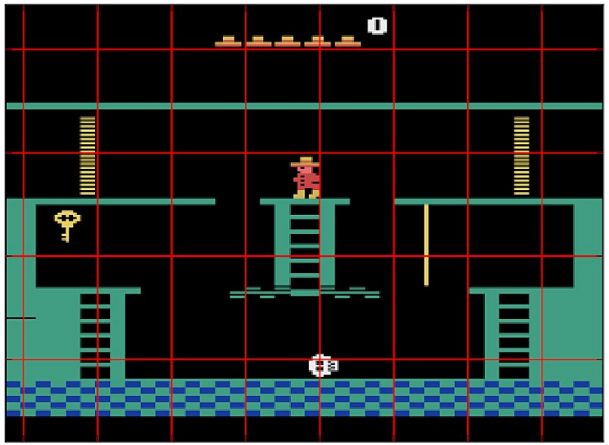
Shaping: Alleviate sparse rewards

Before having the key



After having the key





The φ function for the 1st (and only) screen. Yellow is 2, deep purple is 1.

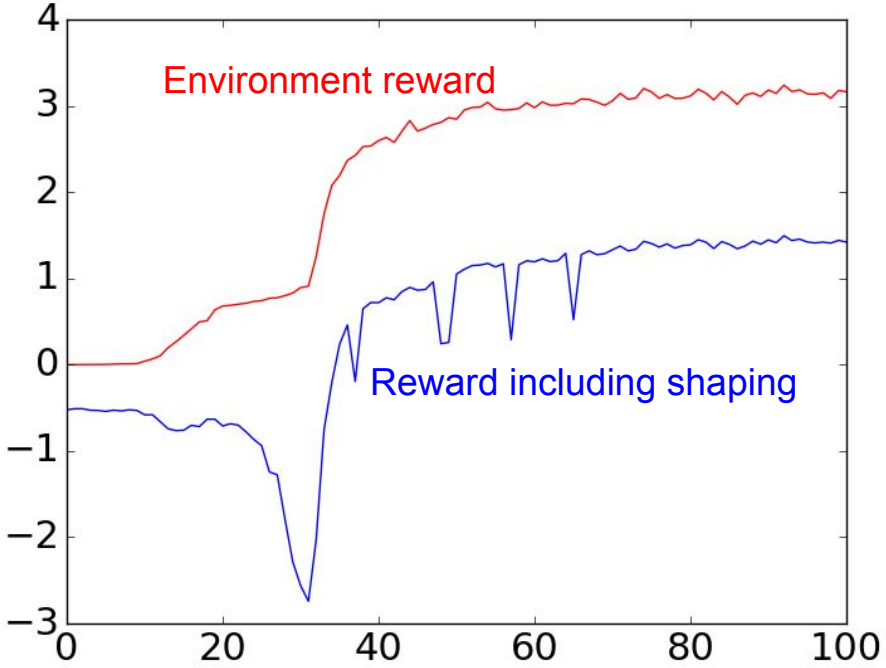
The function is 1 when falling to lose a life, RAM(0xD8) >= 8

If all φ is positive, reward for staying still is negative: $\gamma \cdot \varphi(s) - \varphi(s) < 0$ iff $\gamma < 1, \varphi(s) > 0$

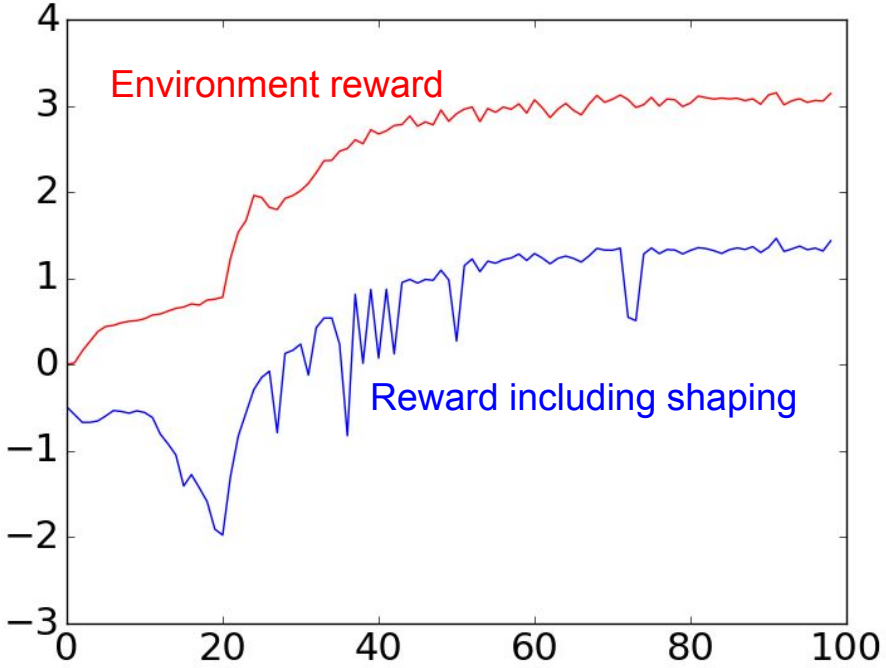
Learning: without options

γ = 0.995 frame_skip = 4

Annealed ε = 0.7 to 0.1 over 20k episodes



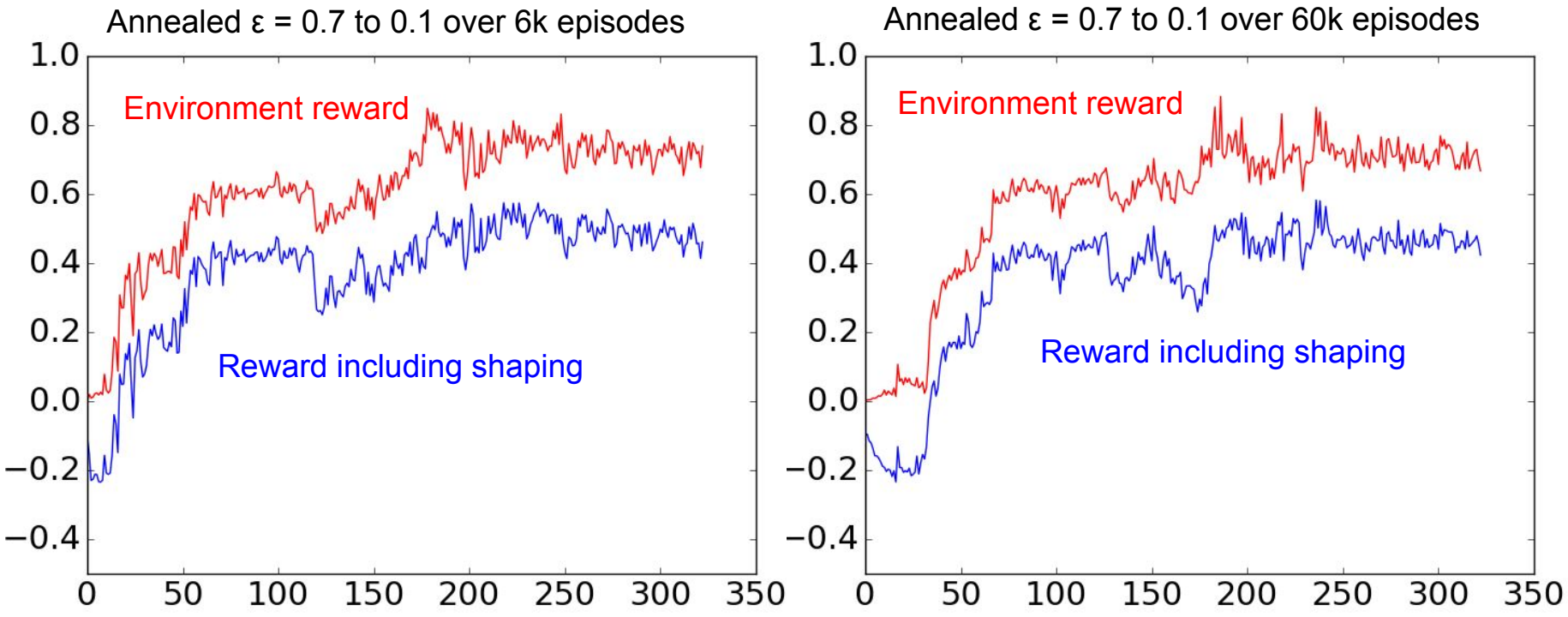
ε = 0.1



x: Thousands of episodes y: Accumulated reward at the end of each episode, mean every 1000 episodes

Learning: with options

γ = 0.9995 frame_skip = 1



x: Thousands of episodes y: Accumulated reward at the end of each episode, mean every 1000 episodes

Learning video (without options)

Conclusions

- Planning
  - Domain knowledge is very useful
  - Make sure to explore the world enough
  - Our approach is too specific to really be useful
- Learning
  - Reward shaping helps with sparse rewards
  - Options may do more harm than help.

Future Work

- Planning
  - Generalise approach to similar domains (such as Private Eye)
    - Use trial and error, or controllable pixels, number of pixels changed, … to figure out the “chosen tuple”
  - Use autoencoder, dimensionality reduction to calculate novelty (Oh et. al., 2015)
  - Plan using predicted future frames (Oh et. al., 2015)
  - Learn a high-level representation of the screen map, with synthetic maps
- Learning
  - Use training frames gathered while planning to learn (Guo et. al, 2014)

References

Bellemare, M. G. et al. (2013). “The Arcade Learning Environment: An Evaluation Platform for General Agents”. In: Journal of Artificial Intelligence Research 47, pp. 253–279.

Bellemare, Marc G et al. (2016). “Unifying Count-Based Exploration and Intrinsic Motivation”. In: arXiv preprint arXiv:1606.01868.

Guo X, Singh S, Lee H, Lewis RL, Wang X (2014). “Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning”. In: Advances in Neural Information Processing Systems, pp. 3338–3346.

Lipovetzky, Nir, Miquel Ramirez, and Hector Geffner (2015). “Classical planning with simulators: results on the Atari video games”. In: Proc. International Joint Conference on Artificial Intelligence (IJCAI-15).

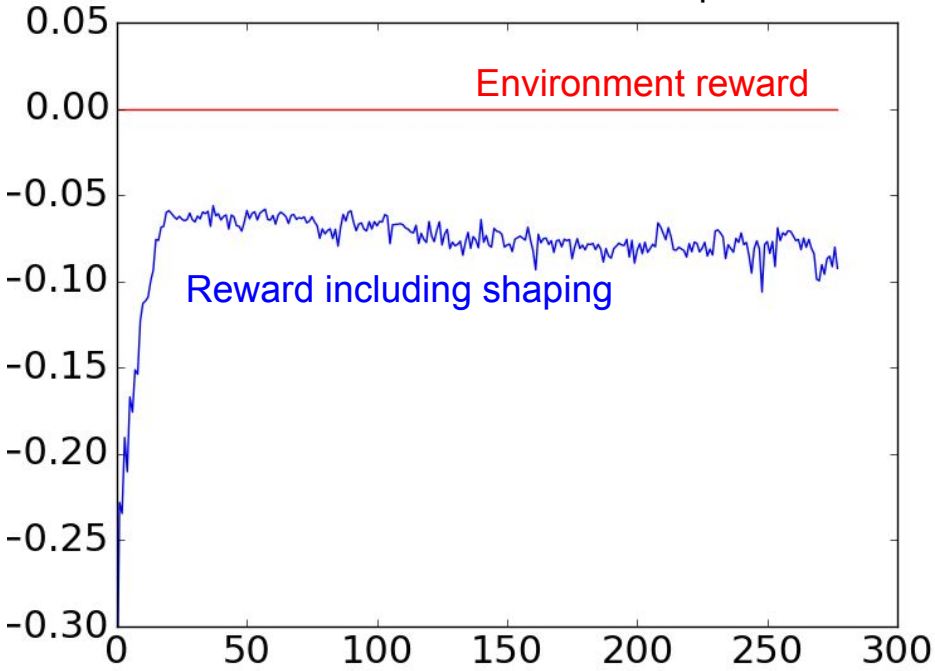
Ng, Andrew Y, Daishi Harada, and Stuart Russell (1999). “Policy invariance under reward transformations: Theory and application to reward shaping”. In: ICML. Vol. 99, pp. 278–287.

Oh, Junhyuk et al. (2015). “Action-conditional video prediction using deep networks in atari games”. In: Advances in Neural Information Processing Systems, pp. 2863–2871.

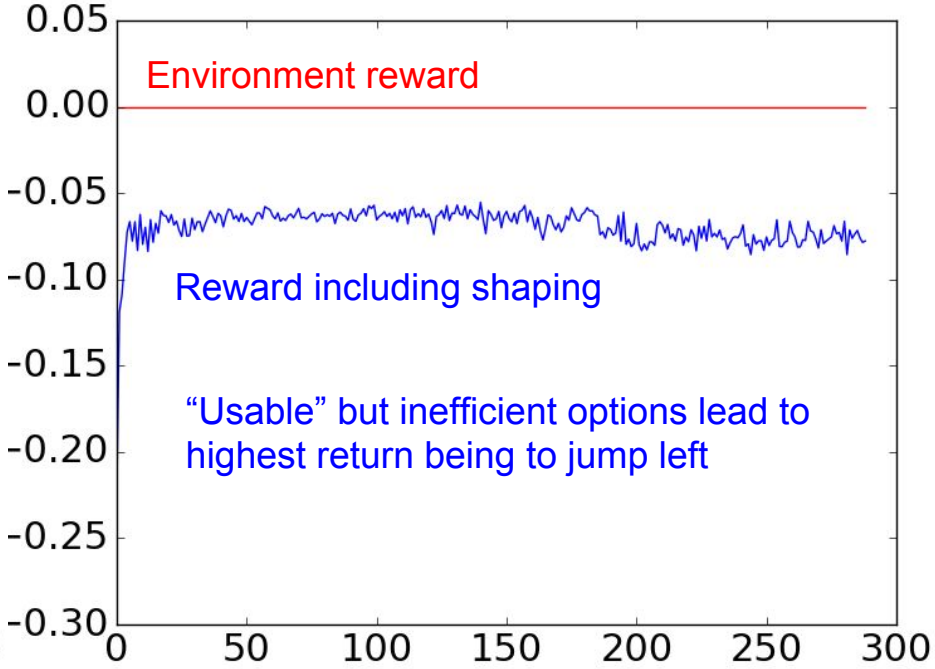
Learning: with options

γ = 0.995 frame_skip = 1

Annealed ε = 0.7 to 0.1 over 20k episodes



ε = 0.1



x: Thousands of episodes y: Accumulated reward at the end of each episode, mean every 1000 episodes