

Piter  Py #4

Saint Petersburg

3-4 November

2017

RL for developers

Guillem Duran Ballester



Index

1. Introduction to Reinforcement Learning
2. RL libraries in python
3. RL Algorithms
4. Deep Reinforcement learning
5. Apendix: Useful resources



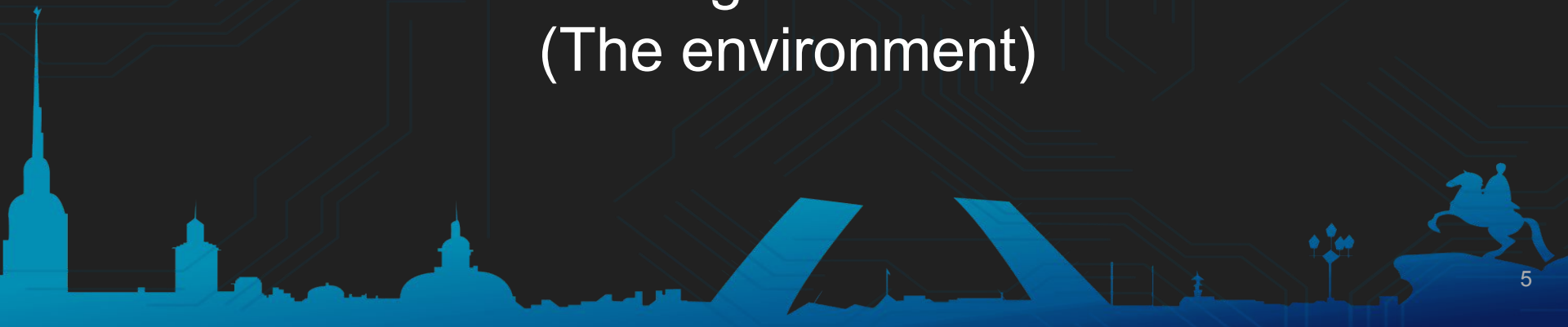
I can explain the math
to you later, but --



Let's go. In and out,
20 minutes adventure.



Video game Rick!! (The environment)



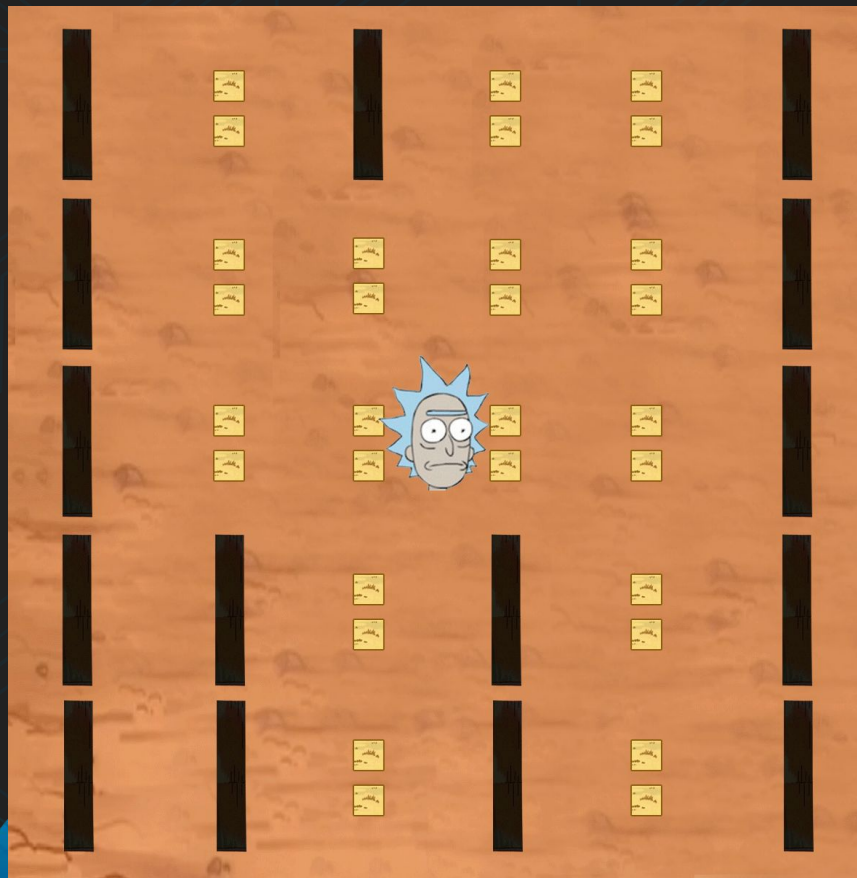
TV
14
DLV



[adult swim]

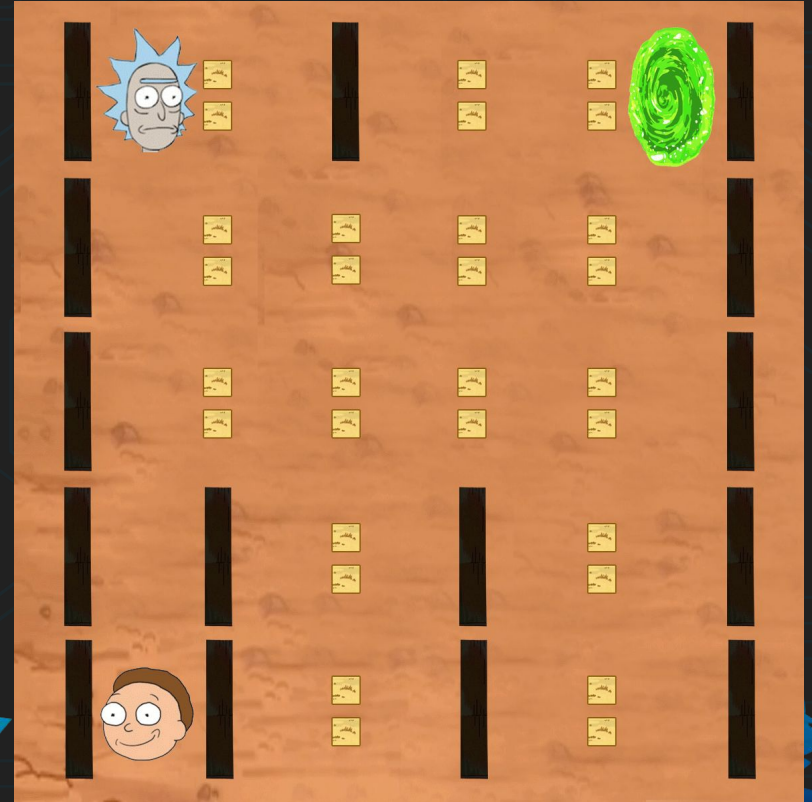
Available actions

1. Move Up
2. Move down
3. Move Right
4. Move left
5. Pickup Morty
6. Portal gun



Videogame Rick!

- Rick starts at a random location
- Walls make rick crash “|”
- He can pick up Morty
- Invalid pickup → Crash
- Invalid portal gun → Everybody dies
- Portal gun + signal → Win

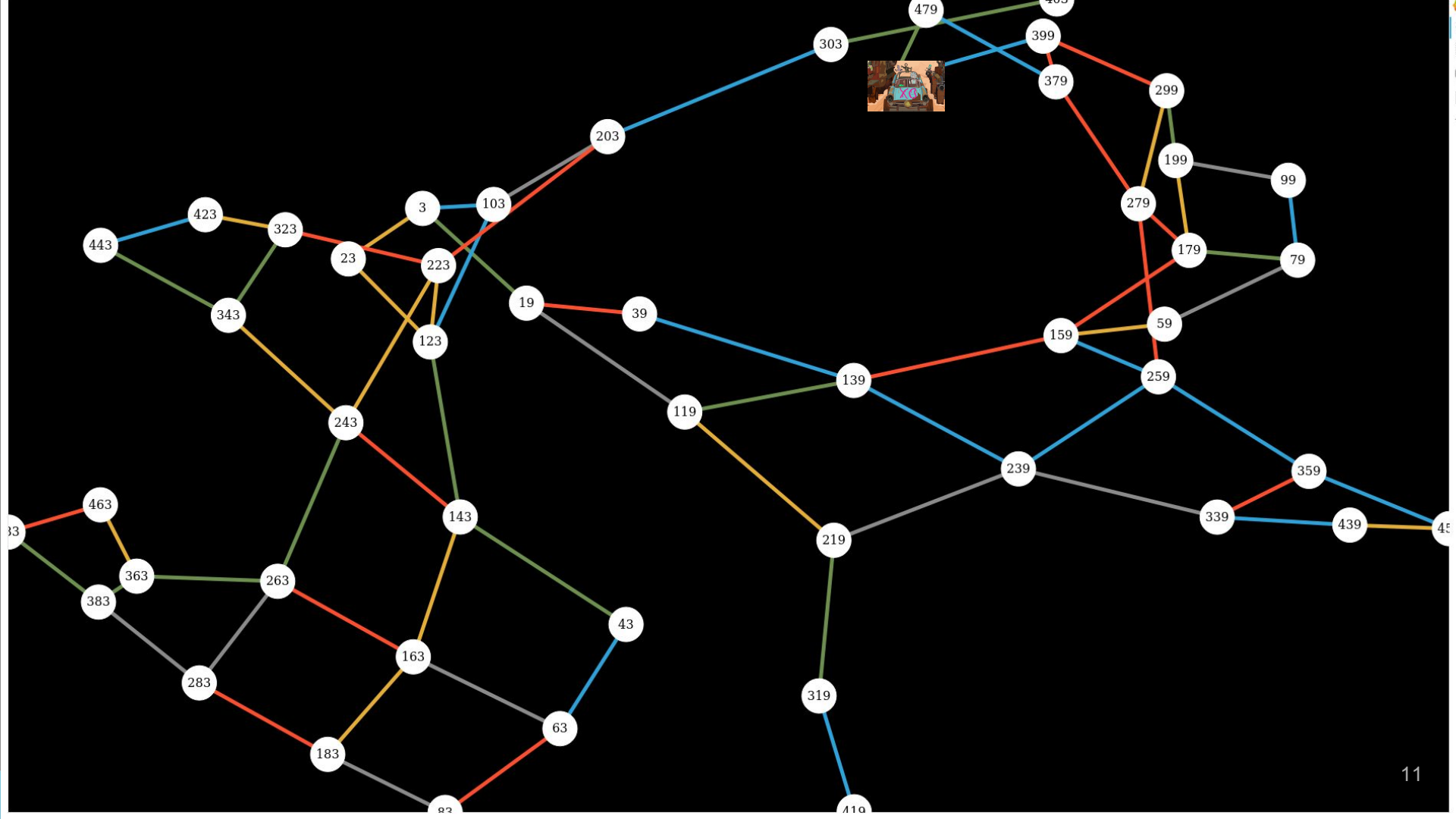


The state space



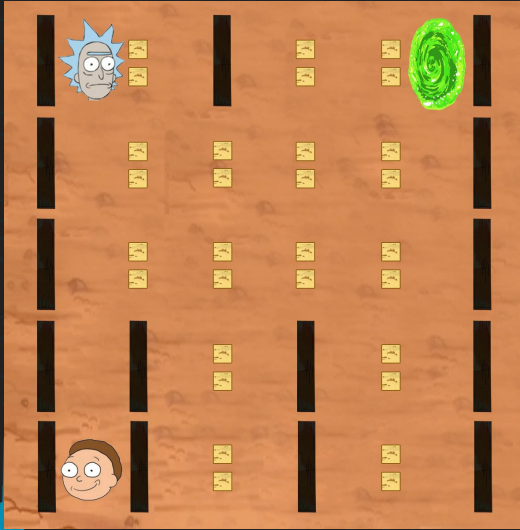
State space



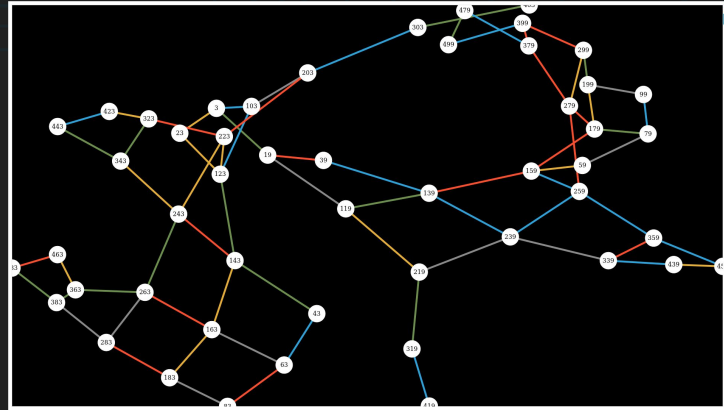


Videogame representations

Visual



Graph



Vector

[4, 4, 1, 4]

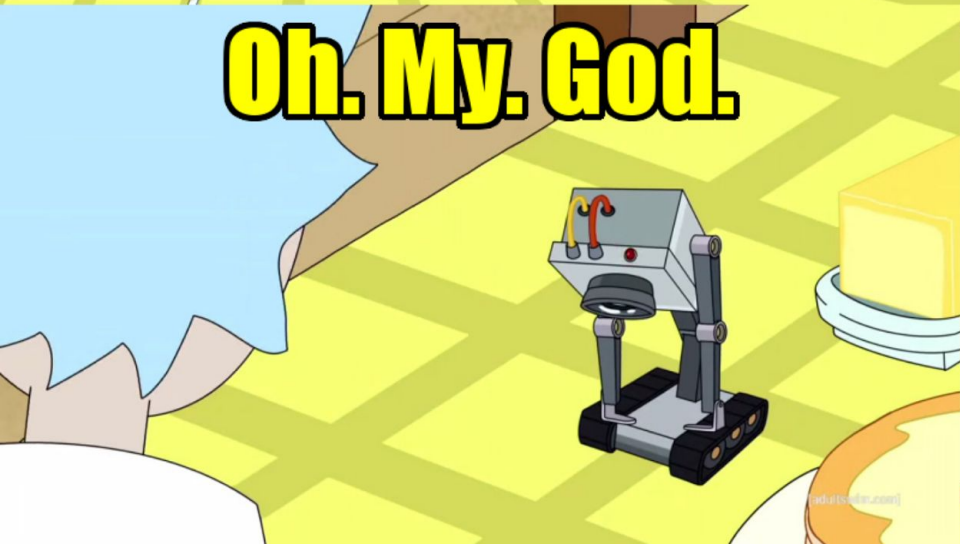
What is my purpose?



You pass butter.



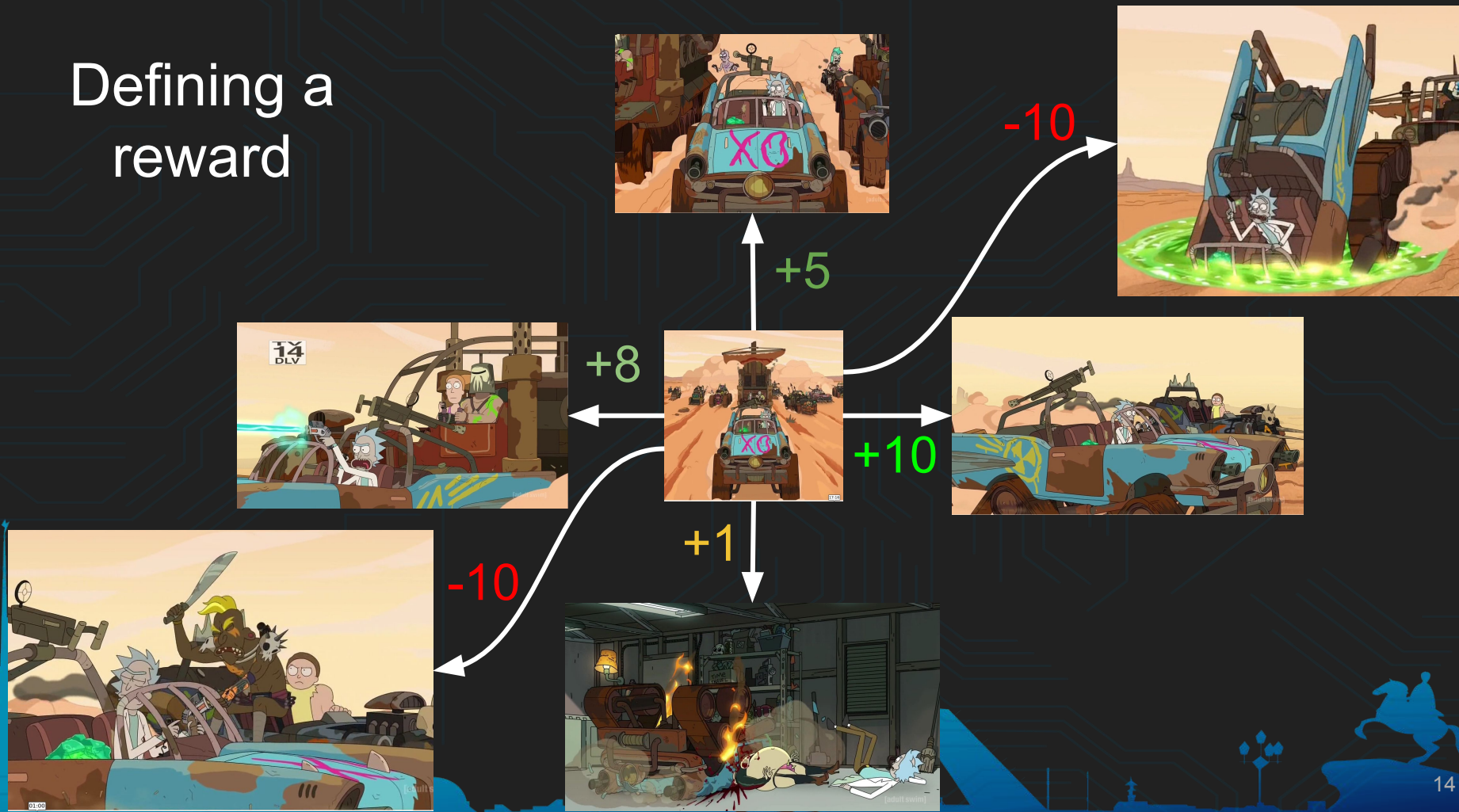
Oh. My. God.



Yeah, welcome to the Club, pal.



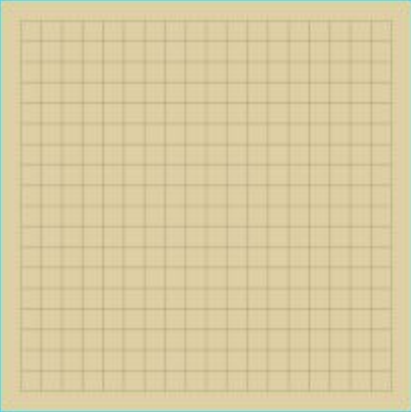
Defining a reward



Real world reward sucks: Vary widely in scale



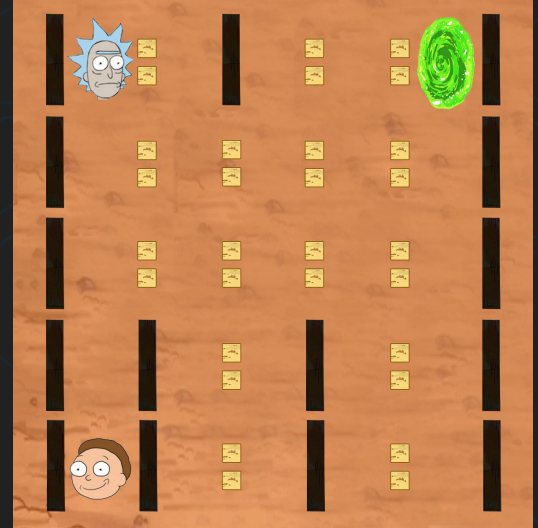
Real world reward sucks: They are sparse



The game of Go has over

1,000,000,000,000,000,000,
000,000,000,000,000,000,
000,000,000,000,000,000,000,
000,000,000,000,000,000,000,
000,000,000,000,000,000,000,
000,000,000,000,000,000,000,
000,000,000,000,000,000,000,
000,000,000,000,000,000,000,
000,000,000

possible positions.



+1 Win / -1 Loss

+20 Win

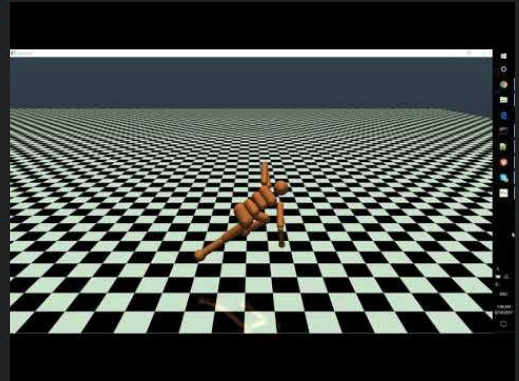
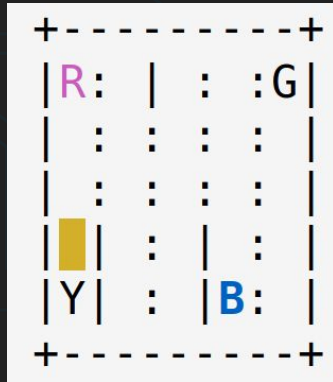
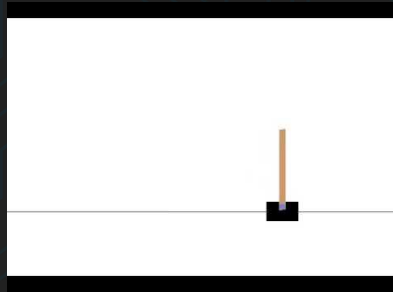
-1 Otherwise

-10 Illegal action

2. RL in Python

OpenAI gym

- Contains different agents for testing RL algorithms
- Common interface in all the environments
- Wide range of different environments
- De facto standard



OpenAI gym usage


```
import gym
env = gym.make('Taxi-v2') # Create an environment
observation = env.reset() # Reset the env before stepping it
done = False

while not done:
    env.render() # Display the environment
    random_action = env.action_space.sample() # Sample an action
    observation, reward, done, info = env.step(random_action)
```


OpenAI Roboschool



OpenAI Universe



go-vncdriver: TigerVNC (flashgames-4tkn2fue)

localhost/flashga... x

localhost/flashgames.DuskDrive-v0/

00 : 15 : 10
Score 236460


TURBO X

MPH

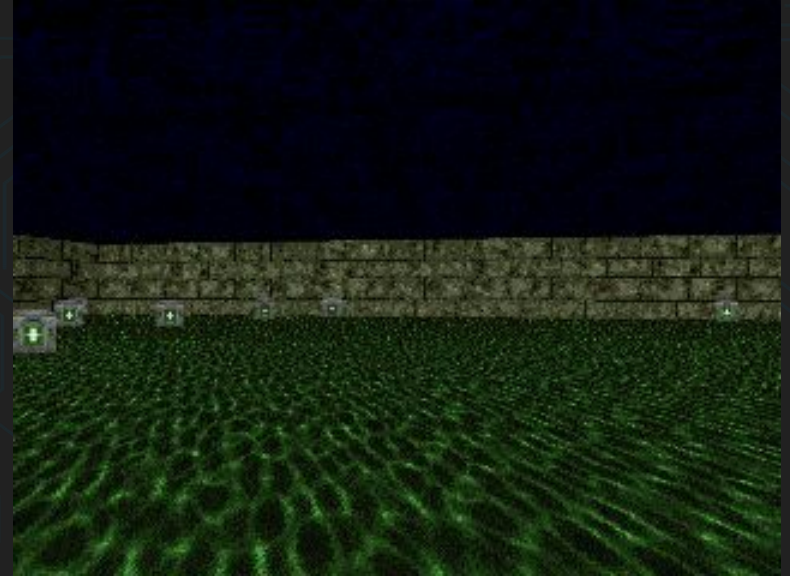
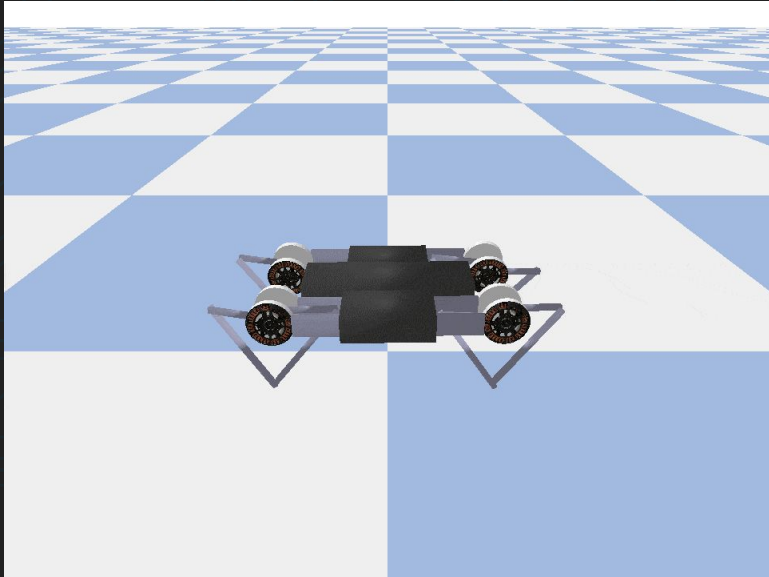
flashgames.DuskDrive-v0

This game is being played by an AI. This browser is not just for you: it's what the AI sees too. You can play the original game here:
<http://www.kongregate.com/games/LongAnimals/dusk-drive>

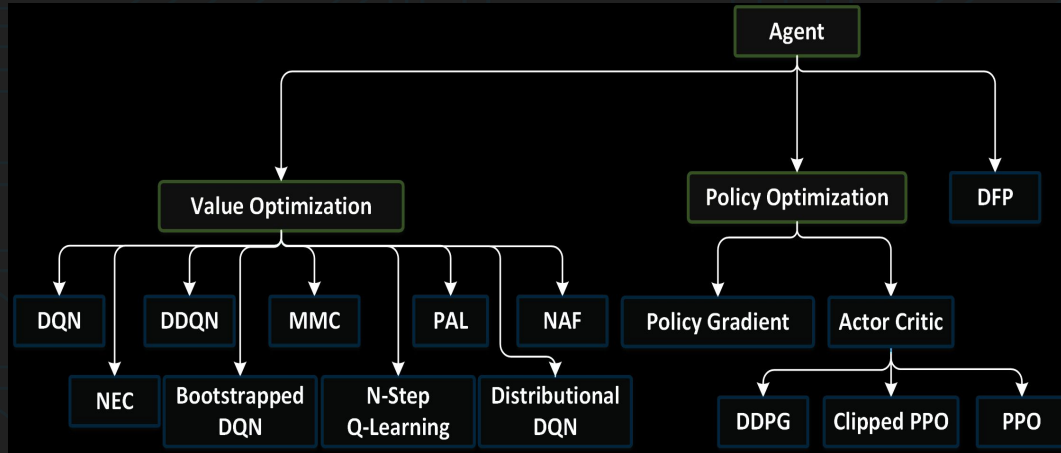
Pressed keys: ArrowUp Mouse: x=0 y=0



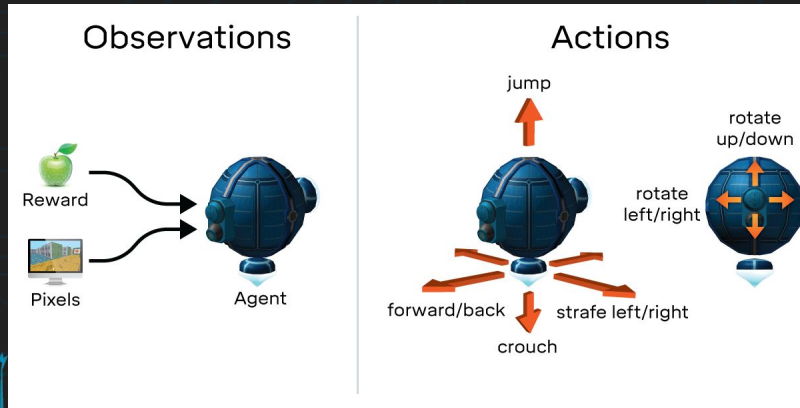
Intel Nervana Coach



Intel Nervana Coach



DeepMind Lab



Deepind Pysc2: - StarCraft II Learning Environment



3. RL Algorithms

Value Function

- Rewards for nodes
- Expected reward achievable from a given state, if we follow the given policy
- Encodes knowledge about future

The Value Iteration algorithm

- Algorithm for calculating a value function
- It consists in propagating the rewards backwards
- Assumes full knowledge of the state space
- Assumes the state space is iterable



```
def value_iteration(transitions, n_iters: int=1000, n_states: int=500, n_actions: int=6):  
    """Action value function using value iteration. It propagates backwards the expected  
    rewards by iterating several times over all the available (state, action) pairs.  
    """  
    value_function = np.zeros(n_states)  
    # this will loop for n_iters * n_states * n_actions = 3,000,000 times  
    for i in range(n_iters):  
        for state in range(n_states):  
            new_value = 0  
            # Get maximum expected reward for all the available actions  
            # value_function[state] = max(value_function[next state] + reward)  
            for action in range(n_actions):  
                next_state, reward= transitions[state, action]  
                action_value = value_function[next_state] + reward  
                new_value = max(new_value, action_value)  
            # This can only increase the old value_function  
            value_function[state] = new_value  
    return value_function
```

Value iteration animation: Absolute value function



Value iteration animation: Relative value function



Value iteration animation: Relative value function



Value iteration: Pros & Cons

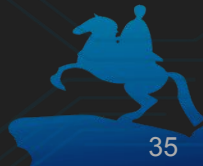
Pros

- Choose state with max value function → Perfect game
- Complies with the definition of a good reward

Cons

- Requires knowledge of every future state of the system
- We need to compute the values for every state

The fog of war effect



Q learning

- Algorithm meant to overcome the problems of value iteration
- No need to search the entire state space
- Same idea of propagating the score backwards

Q learning

```
for i in range(n_iters):  
    while not terminal:  
        action = sample_from_exploration_policy(current_state)  
        # Q value update  
        next_state, reward, terminal, info = videogame_rick.step(action)  
        q_table = update_q(current_state, action, videogame_rick.transitions, alpha, gamma)  
  
        current_state = update_current_state(next_state, terminal)
```


Q value update

$$Q(s_t, a_t) \leftarrow (1 - \alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)$$

The equation shows the Q value update formula. The term $(1 - \alpha) \cdot Q(s_t, a_t)$ is labeled as 'old value'. The term α is labeled as 'learning rate'. The term r_t is labeled as 'reward'. The term γ is labeled as 'discount factor'. The term $\max_a Q(s_{t+1}, a)$ is labeled as 'estimate of optimal future value'. The entire term $r_t + \gamma \cdot \max_a Q(s_{t+1}, a)$ is labeled as 'learned value'.

```
def update_q(q_table, transitions, state, action, alpha, gamma):
    reward, next_state = transitions[state][action][1:3]
    # q_table is a n_states * num_actions array. It stores the q val for each (state,
    # action)
    old_q_sa = q_table[state, action]
    max_future_q = max(q_table[next_state, :]) - old_q_sa # Estimated future reward
    # Formula above
    q_table[state, action] = (1 - alpha) * old_q_sa + alpha * (reward + gamma *
    max_future_q)
    return q_table
```

Q Learning: Pros & Cons

Pros

- Choose action with max Q value → Perfect game
- Complies with the definition of a good reward
- No need to use a perfect transitions matrix
- Proven to converge in the limit

Cons

- Requires an exploration policy → Only learns from memory
- “The limit” can be infeasible in practice
- The Q table can be extremely large

Deep RL parameter hell

- Exploration policy params
- Optimizer
- Advantage
- Architecture
- Opt params: LR, momentum...
- Batch size
- Gradient update freq.
- Loss clipping
- Gradient clipping
- Random seed!?

4. Deep Reinforcement learning

I'M MR. MEESEKKS



LOOK AT ME!



Deep Reinforcement learning

- Allows traditional algorithms to scale with increasing state space complexity
- Replaces the Q value / Value function tables with a DNN
- Defines an objective function to train the network

Deep RL paradigm shift

It is all about minimizing a loss function

Replaces “Tables” with “black boxes”

A bunch of new metaparameters

Need to stabilize gradients



Deep Q learning

$Q(S | t=0)$



$Q(S | t=1)$



error = Hubber_loss($Q(S | t=0)$ - (reward + gamma * $Q(S | t=1)$))

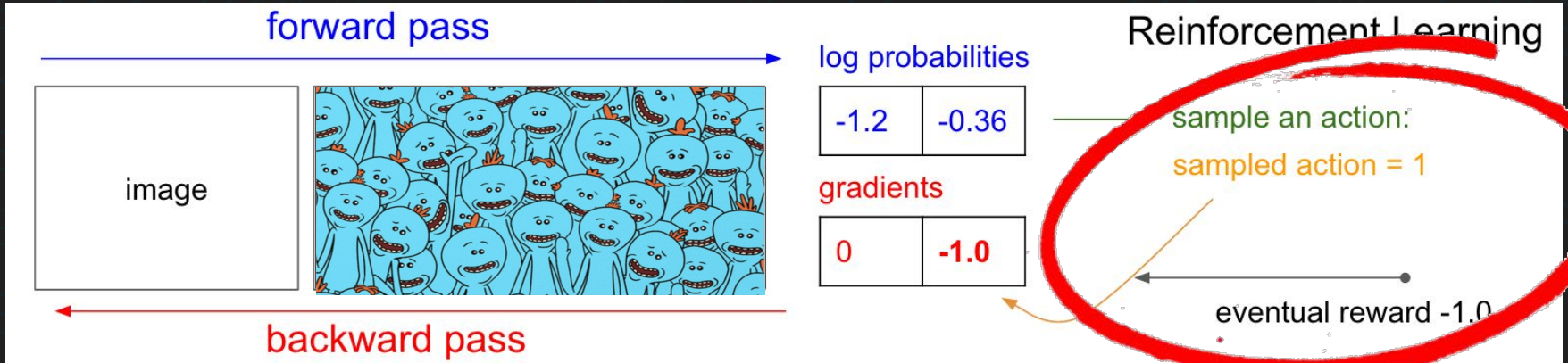
- Tensors as input → DNN train on batches
- Requires applying masks
- A lot of variants to improve stability
- Extremely dependent on metaparameters



Policy Gradient

- Change the probability of the network of choosing a given action
- Use a Q like function (Advantage) to influence the decisions of the network when defining the loss
- Let backpropagation take care of the rest

Policy Gradient

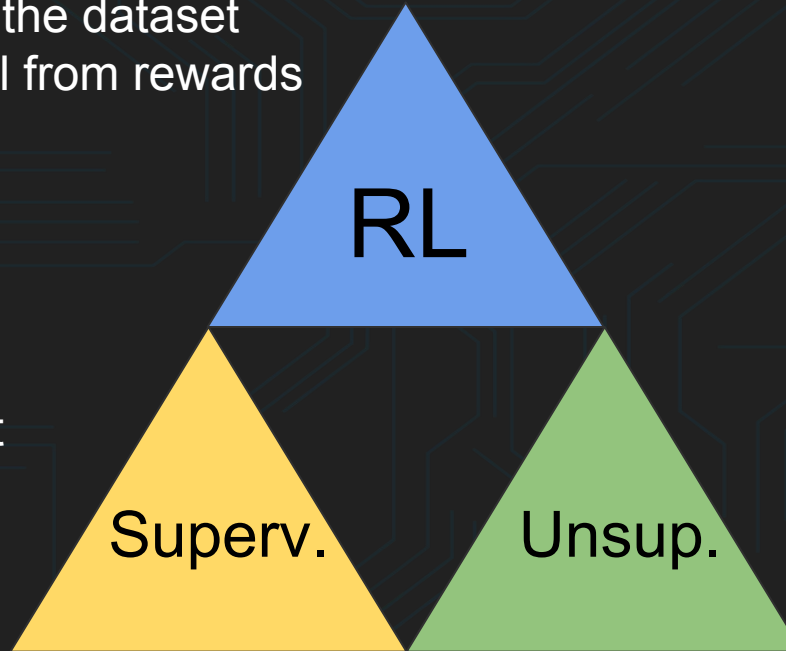






Machine Learning

- Generate the dataset
- Build label from rewards

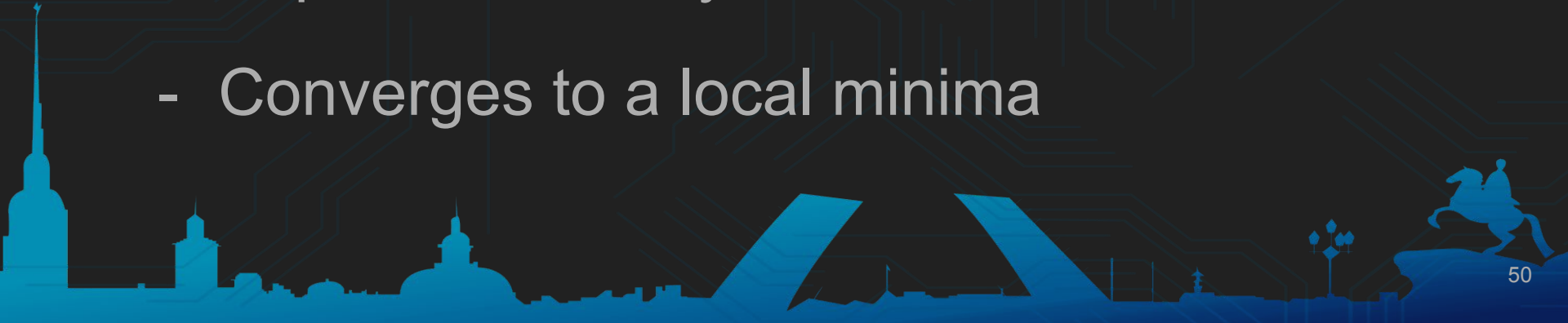


- Training / Test dataset
- Labeled data

- A big enough dataset
- No Labels

Useful things to remember

- No control over how features are learned
- Only learning from memory
- Exploration is key
- Converges to a local minima



Dirty tricks that happen to work

- Reward & lives clipping
- Gradient & loss clipping
- Use a memory to decorrelate samples
- Batch normalization & Regularization
- Computer vision tricks are welcome

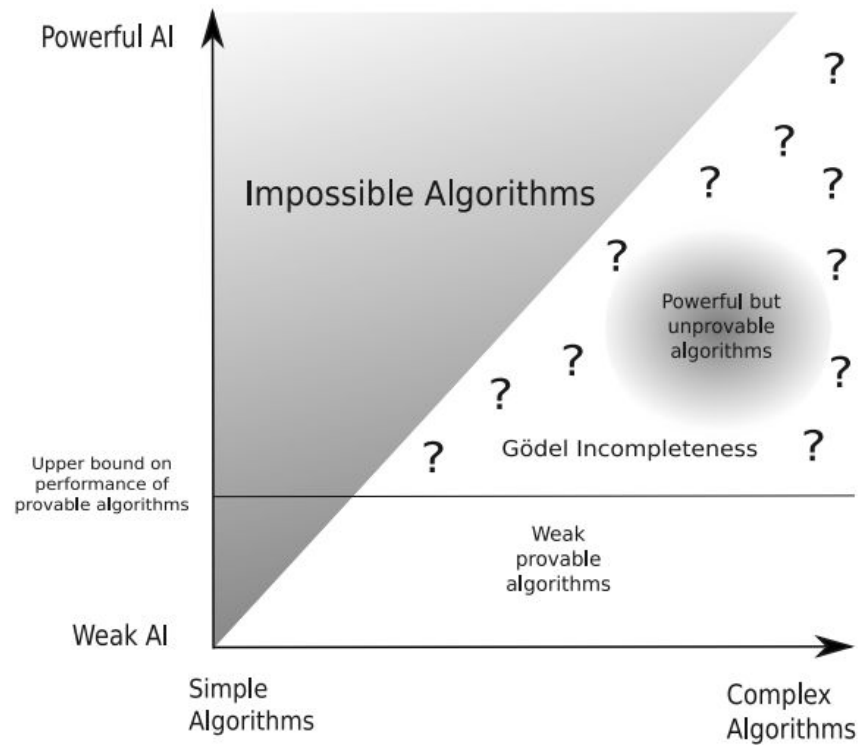


Figure 5.1.: Theorem 5.3.3 rules out simple but powerful artificial intelligence algorithms, as indicated by the greyed out region in the upper left. Theorem 5.6.1 upper bounds how powerful an algorithm can be before it can no longer be proven to be a powerful algorithm. This is indicated by the horizontal line separating the region of provable algorithms from the region of Gödel incompleteness.



Mandatory slide about me

Guillem Duran Ballester

- Telecommunications engineer
- Data Science enthusiast & PyData Mallorca organiser
- Worked as an AI engineer at source{d}
- 4 years learning AI for the pleasure of finding things out
- Learning to be a developer

A scene from the animated series Rick and Morty. Rick Sanchez is in the driver's seat of a futuristic car, looking stressed with wide, red-rimmed eyes and a grimace. Morty Smith is in the passenger seat, covering his face with his hands and crying. A large black rectangular box with the white text "Thank you!" is superimposed over the center of the image. The car's interior is green and futuristic, with a large circular window behind the seats.

Thank you!

@Miau_DB

Guillem-db⁵⁵

5. Appendix



RL fundamentals

- Notebooks on AI: State space structure
- [A Karpathy blog](#): Playing Pong with a DQN
- [Let's make a DQN](#)
- David Silver [UCL course](#) on RL & [videos](#)
- [Nando de Freitas](#) course, mostly DL

Environment libraries

- OpenAI [Roboschool](#): Robot simulation
- OpenAI [Universe](#): gym on VNC servers inside docker containers
- Intel Nervana [Coach](#): Environments + DLR algorithms
- Deepmind [Lab](#):
- Deepmind [Pysc2](#): Star Craft 2 - like environments

Deep RL libraries

- [Keras-rl](#)
- [Tensorforce](#)
- Nervana Coach
- [Tensorflow Agents](#)
- OpenAI [baselines](#)
- [PyTorch-rl](#)

Papers

- Agents on Starcraft 2
- Imagination Augmented Agents
- Alphago zero
- Evolutionary strategies

The reward

- Scalar that measures “how good an action is”
- Associated to an (state, action) tuple
- The better the action, the higher the reward
- It should have high sensibility → Change frequently