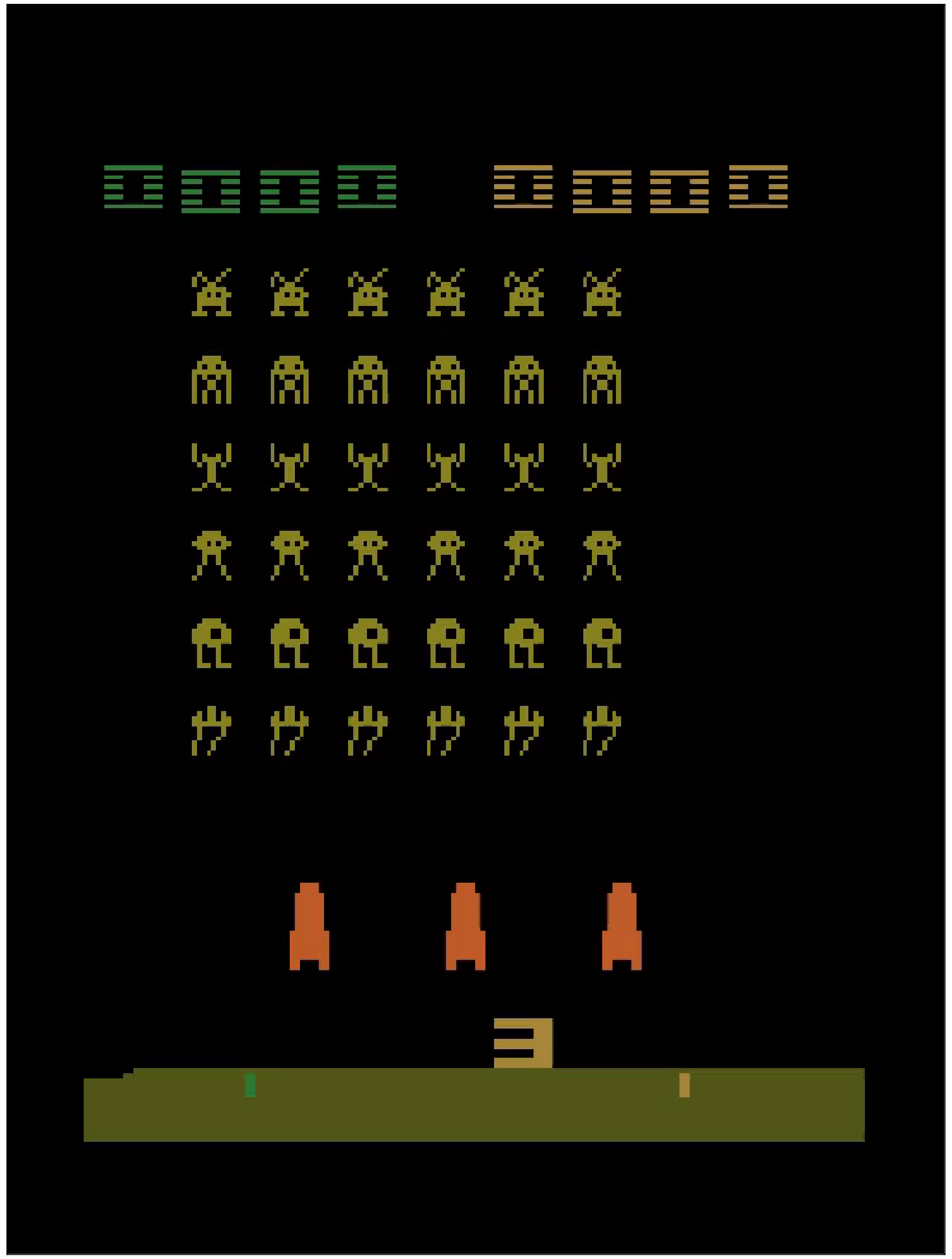
How to teach space invaders to your computer

David Wölfle @PyCon.DE 2018



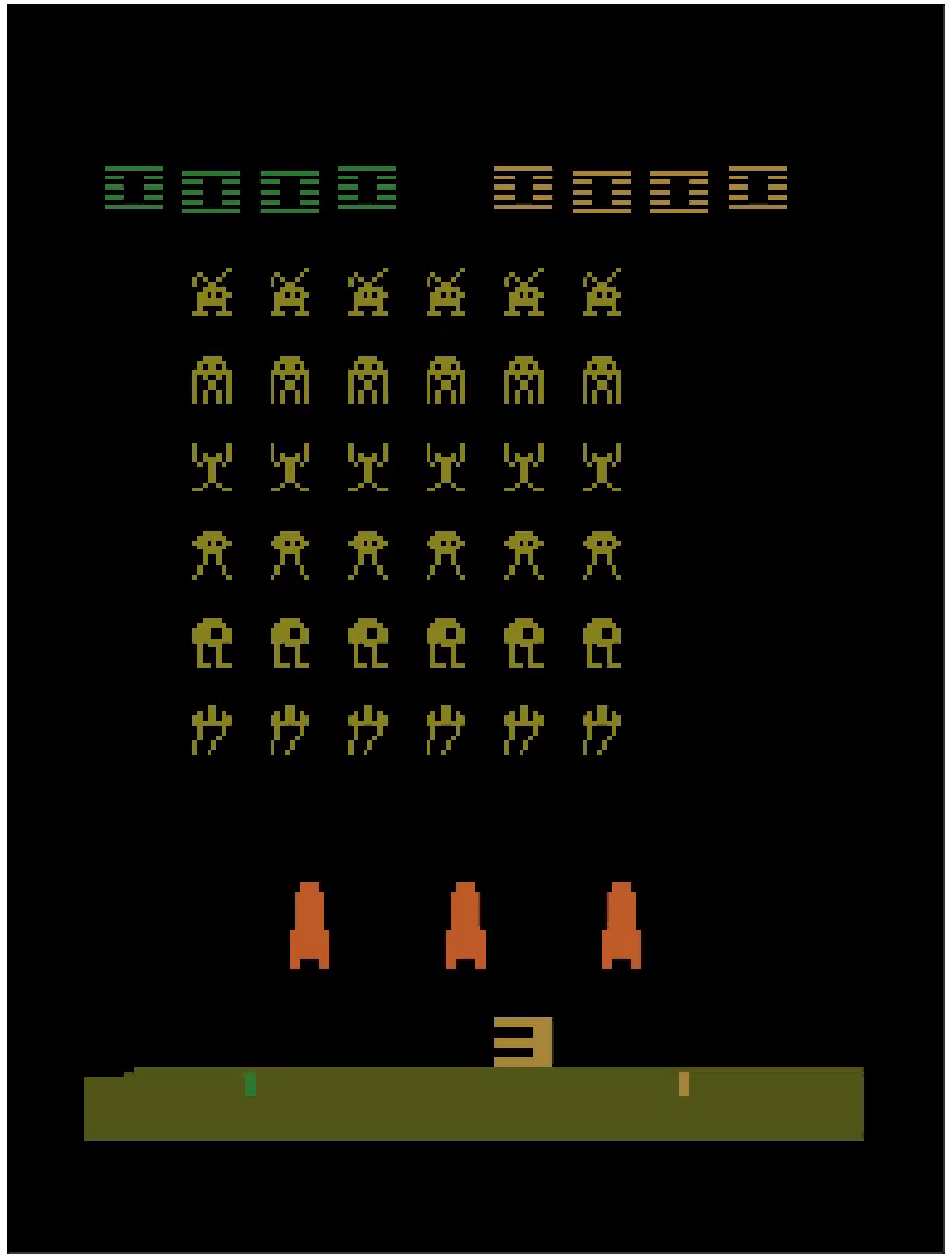
FZI FORSCHUNGSZENTRUM INFORMATIK

How to teach space invaders to your computer

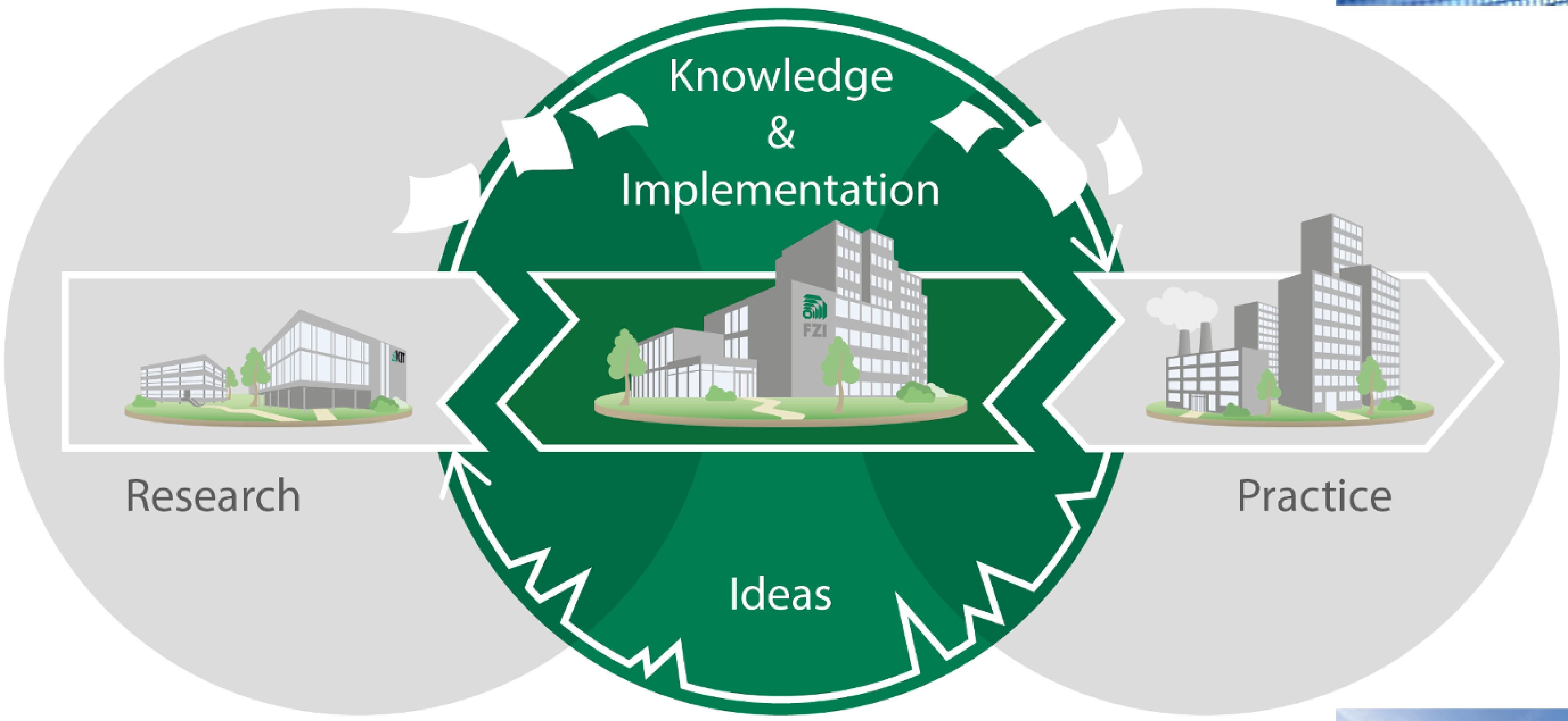
aka: A very brief introduction to reinforcement learning

Content:

1. Motivation or why bother about Space Invaders?
2. A brief introduction to reinforcement learning theory.
3. Example Algorithm: Training space invaders.
4. Results and analysis of training.
5. Summary.



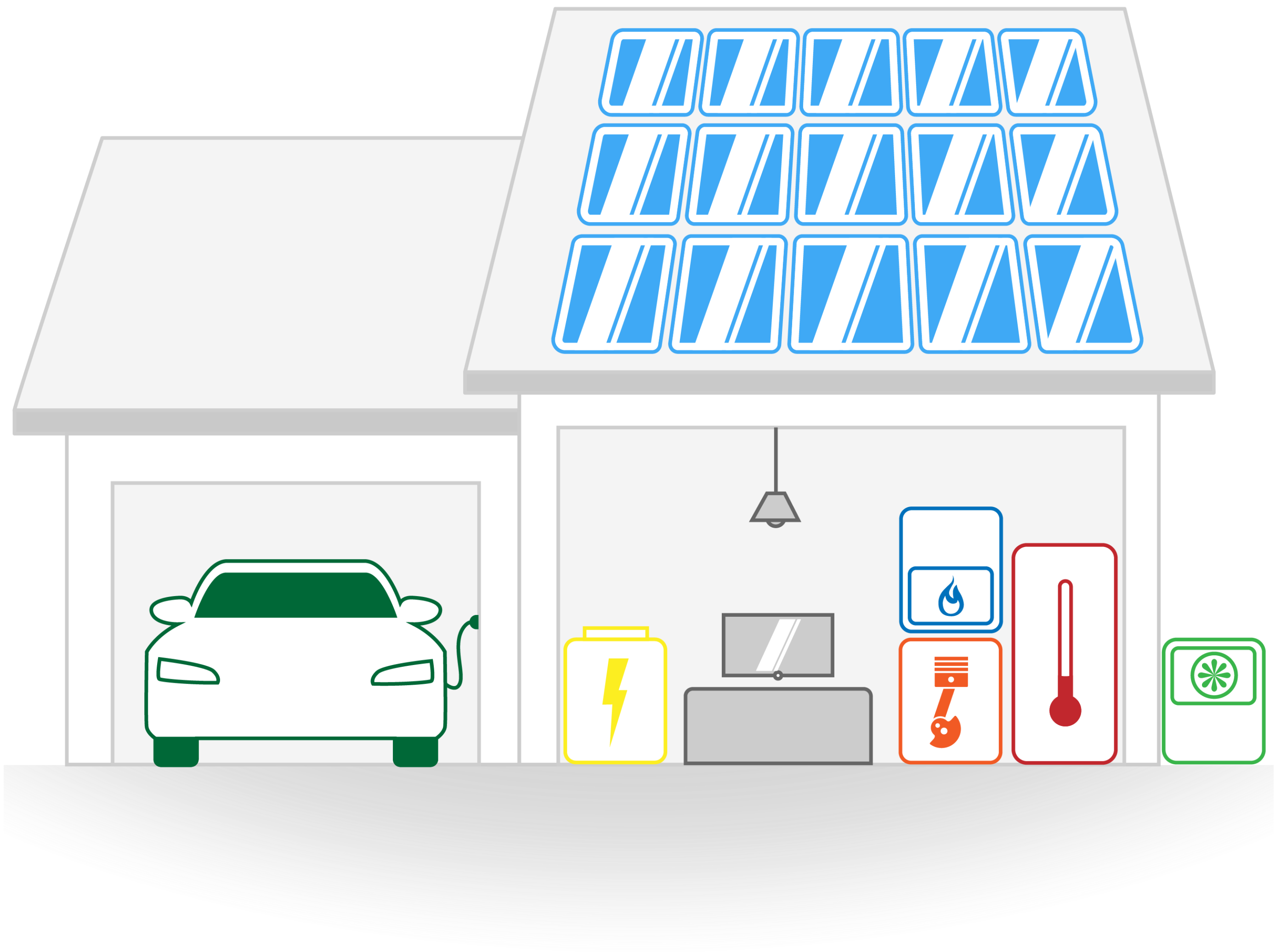
FZI Research Center for Information Technology



Motivation

My research

- Improving the energy efficiency of buildings with smart algorithms.
- Optimizing building energy consumption requires planing and sequential decision making.
- Most research in the field use explicit (i.e. manual) modeling of the system being optimized.
- My approach: Develop an algorithm that learns how to optimize a building through interaction.



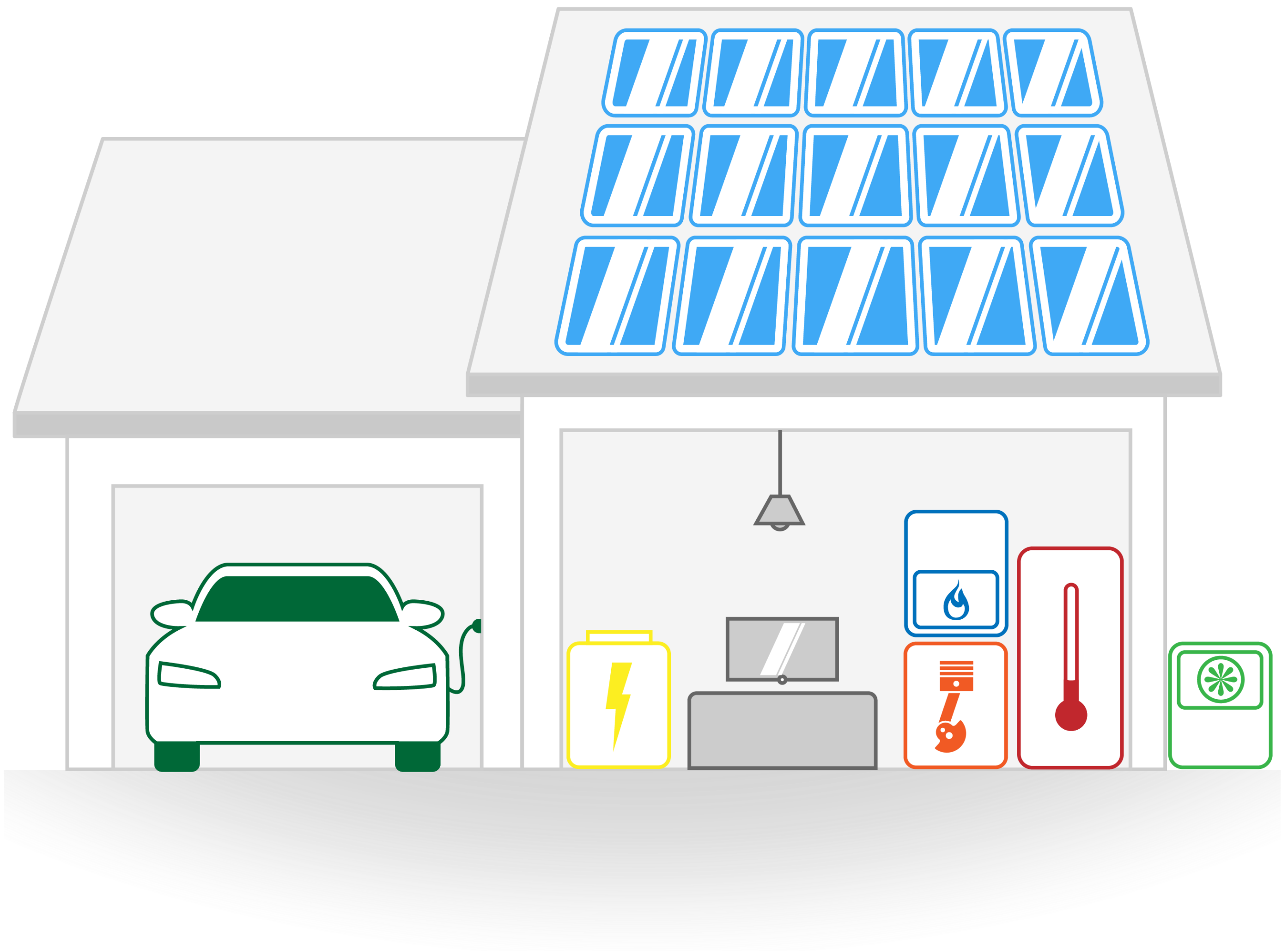
FZI FORSCHUNGSZENTRUM INFORMATIK

Motivation

FZI

My research

- Improving the energy efficiency of buildings with smart algorithms.
- Optimizing building energy consumption requires planing and **sequential decision making**.
- Most research in the field use explicit (i.e. manual) modeling of the system being optimized.
- My approach: Develop an algorithm that learns how to optimize a building through **interaction**.
- ⇒ **This is a Reinforcement Learning problem!**



FZI FORSCHUNGSZENTRUM INFORMATIK

Motivation

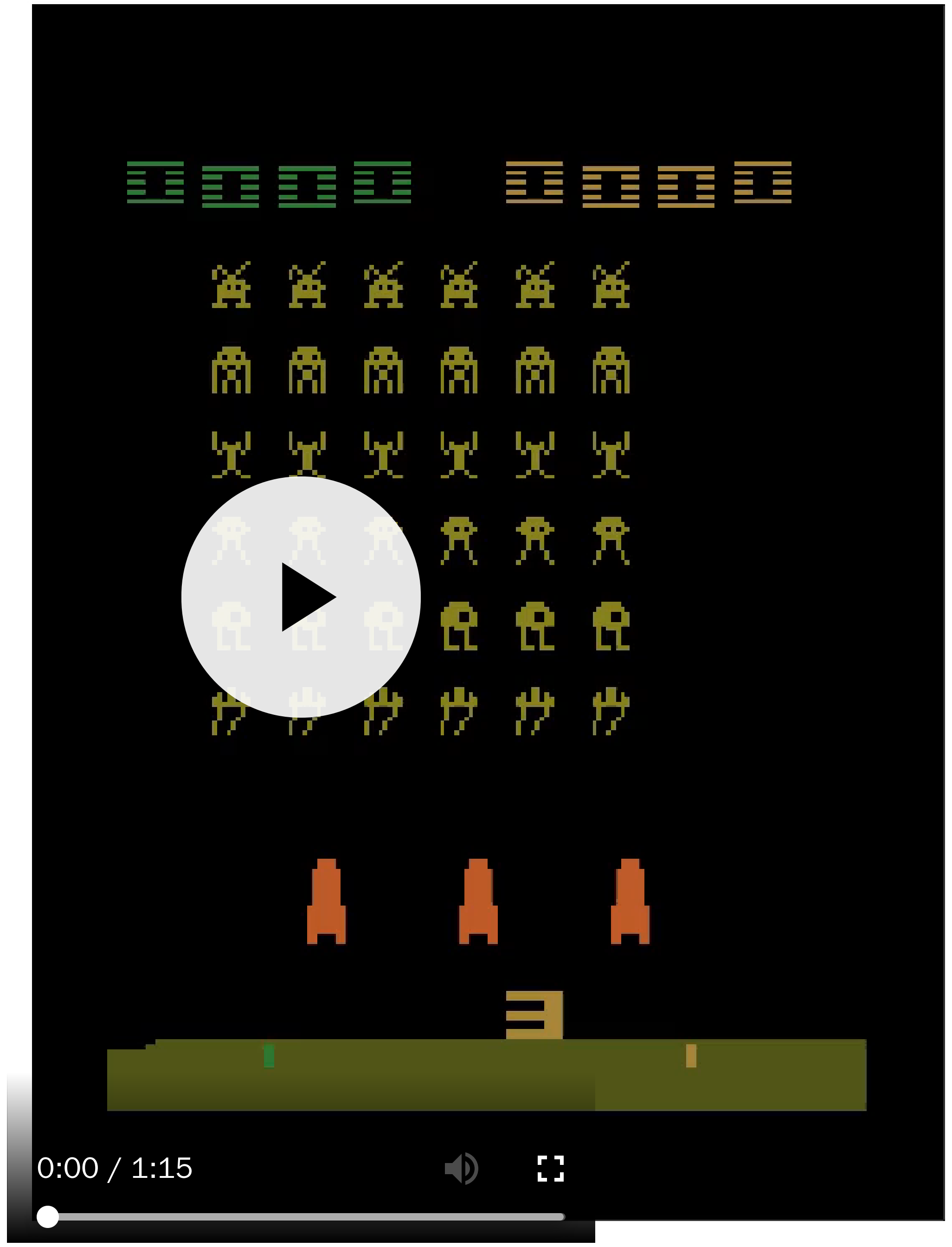
Why Space Invaders?

Atari games are a benchmark problem in Reinforcement Learning.

New algorithms are routinely tested on those, e.g:

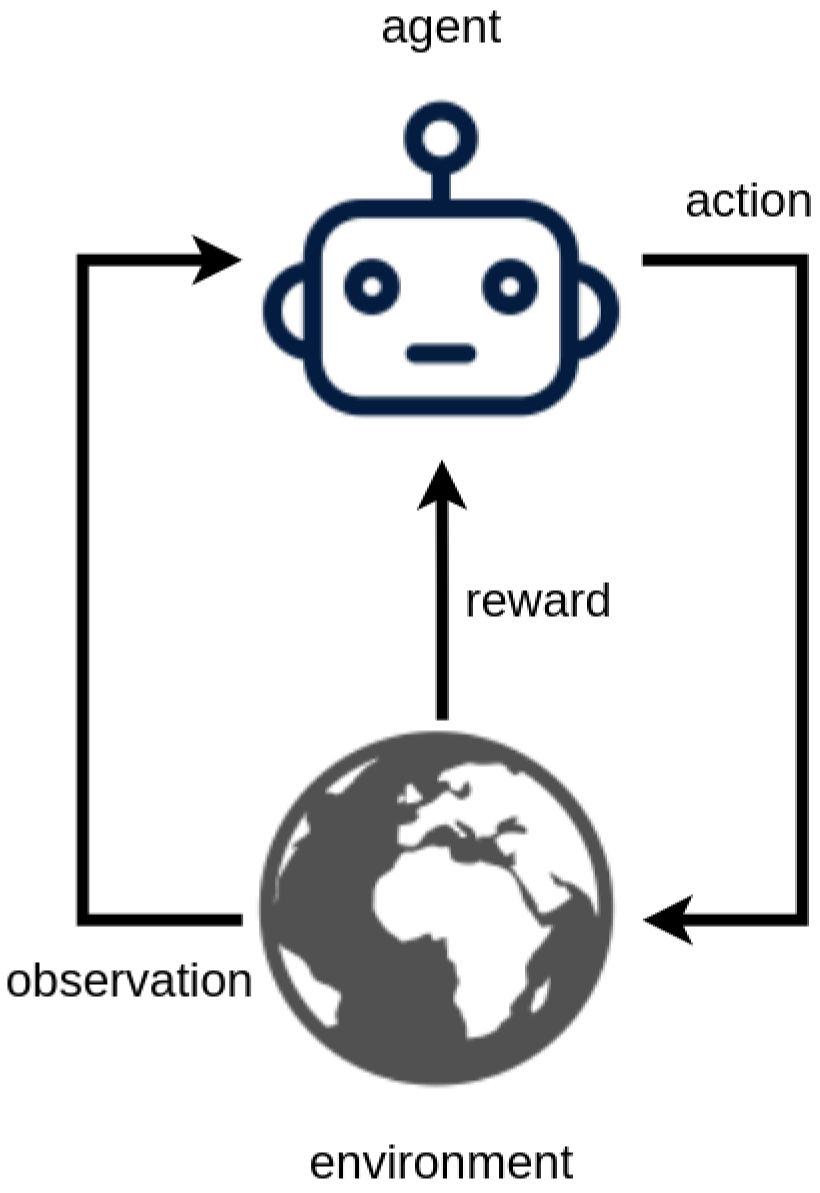
- Mnih et al. - DQN
  https://www.nature.com/articles/nature14236
- Mnih et al. - A3C
  https://arxiv.org/abs/1602.01783
- Schulman et al. - PPO
  https://arxiv.org/abs/1707.06347

⇒ **Atari games are a good starting point to learn about Reinforcement Learning!**



FZI FORSCHUNGSZENTRUM INFORMATIK

Reinforcement Learning terminology (the basics)



Consider Space Invaders:

environment:	Game of Space Invaders proceeding in discrete timesteps.
observation:	The frame (screenshot) of gameplay after each timestep.
reward:	The score (for shooting alien ships) received during the last timestep.
agent:	Program that receives observation and reward and computes next action.
action:	Choice made by agent (E.g. left, right, fire, do nothing).
episode:	One round of space invaders played by the agent until game over.

How to approach Reinforcement Learning: "Classic Method"

Common approach (simplified):

- Abstract states from observations. (Where am I absolutely in my search space?)
- Estimate how good it is to be in any particular state.
- Pick actions by selecting those that lead to most valuable next state.

Deep dive into the beautiful Reinforcement Learning theory:

- Textbook: Sutton & Barto - Reinforcement Learning: An Introduction
  http://incompleteideas.net/book/bookdraft2017nov5.pdf
- Lecture: Silver - Introduction to reinforcement learning
  https://www.youtube.com/playlist?list=PLqYmG7hTraZDM-OYHWgPebj2MfCFzFObQ
- Python exercises: Britz - reinforcement-learning GitHub repository
  https://github.com/dennybritz/reinforcement-learning

Extensive theory involved $\Rightarrow$ Out of scope for this talk.

How to approach Reinforcement Learning: "Evolutionary Method"

Common approach (simplified):

- Use black-box optimization principle.
- Direct mapping of observation to action.
- Optimizer only uses the accumulated reward. (i.e. no states etc.)
- Optimizer modifies parameters how agents computes actions given a observation.

Next slides

- Follow concept of "World Models" by Ha & Schmidhuber https://arxiv.org/abs/1803.10122
- Paper covers Carracing and ViZDoom environments.
- Videos credit: https://worldmodels.github.io/

CMA-ES

Key points:

Name: Covariance Matrix Adaptation Evolution Strategy

Considered a reliable choice if dimension $\lesssim 1000$.

Reference: Hansen - The CMA Evolution Strategy: A Tutorial
https://arxiv.org/abs/1604.00772

Uses a multivariate Gaussian distribution to generate candidates.

Hyperparameters:

- Initial mean
- Initial standard deviation
- Population size

Python package: https://github.com/CMA-ES/pycma

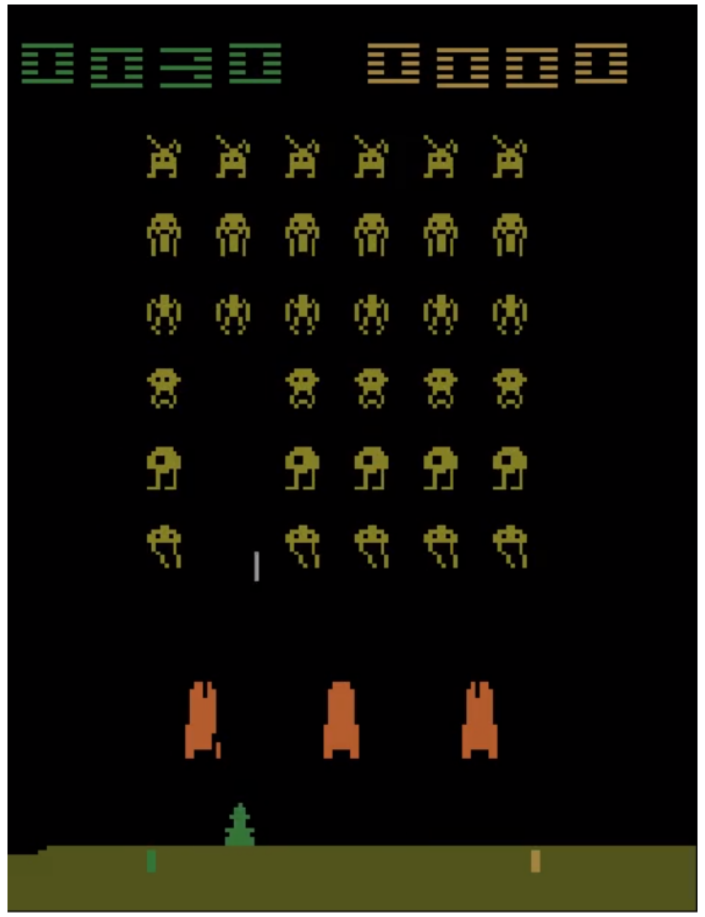
```
In [1]: import cma
        inital_mean = 12 * [0]
        initial_std = 0.5
        population_size = 25

        # Use the Rosenbrock function for illustration.
        reward_function = cma.ff.rosen
```

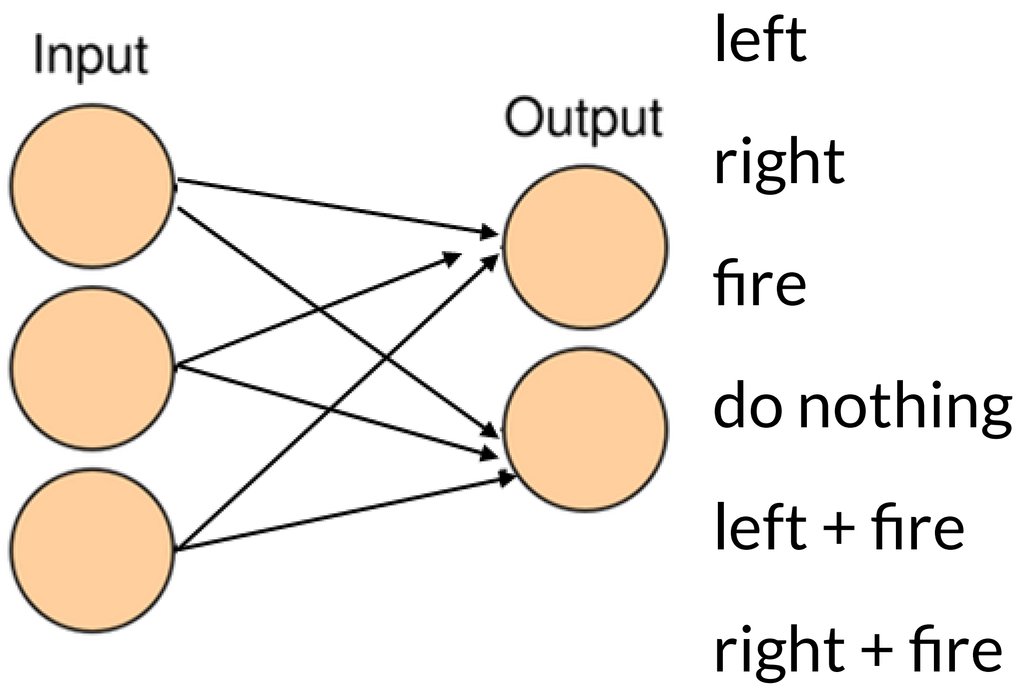
```
In [2]: es = cma.CMAEvolutionStrategy(inital_mean,
                                      initial_std,
                                      {'popsize': population_size})

        while not es.stop():
            # Sample from multivariate Gaussian.
            canidates = es.ask()

            # Compute the reward (aka the fitness) of each canidate.
            rewards = [reward_function(c) for c in canidates]

            # Update mean and covariance matrix.
            es.tell(canidates, rewards)

        print('\nBest solution found:\n', es.best.x)
```

```
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 12 (seed=682612, Mon Oc
t 15 15:16:39 2018)

Best solution found:
 [1.         0.99999999 1.         1.         1.         0.99999999
 1.00000001 1.00000001 1.00000001 1.00000004 1.00000007 1.00000013]
```

Applying CMA-ES to Reinforcement Learning Problems

Approach: Use CMA-ES to train weights of neural network.



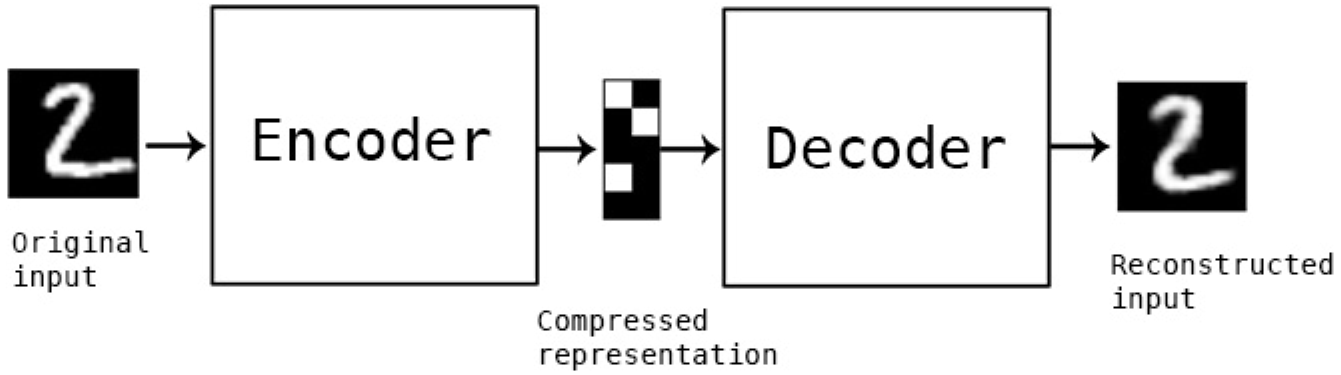


Issue: For Space Invaders that means training ~ 600k weights.
(210x160x3 color values per frame x 6 outputs)

Solution in World Models paper: Use auto encoder for dimensionality reduction.

Autoencoders

Basics:



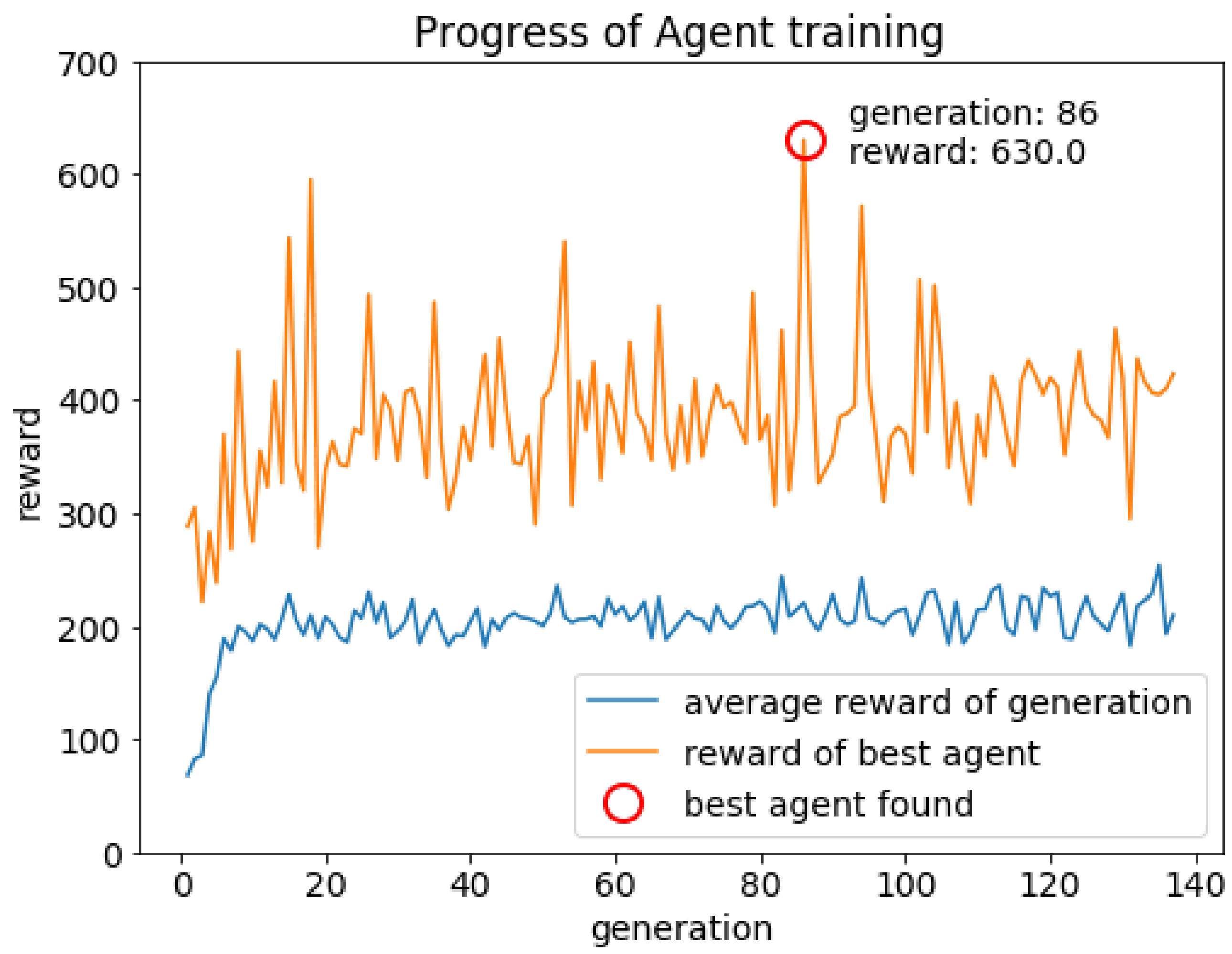
- Consist of a encoder, compressed representation (bottleneck) and decoder.
- Usually implemented as neural network.
- The compressed representation is considered to be a high level description of the input.
- Trained in semi supervised fashion with input and expected output being the same.
- Reconstruction loss of input (often MSE) is used for training.
- For more information see e.g. https://blog.keras.io/building-autoencoders-in-keras.html

Training the agent

Algorithm:

- Build and train the autoencoder with episodes generated following a random policy.
- Build agent model (neural network): 64 in, 6 out, fully connected, no hidden layers, tanh activation
- Initialize CMA ES with initial_mean=0 and initial_std=1.0
- Repeat until stopping criterion is met:
  - Sample 32 candidates from CMA-ES.
  - Compute the average reward over three episodes for every candidate.
  - Trigger optimization step (tell rewards to CMA-ES).

Training result:

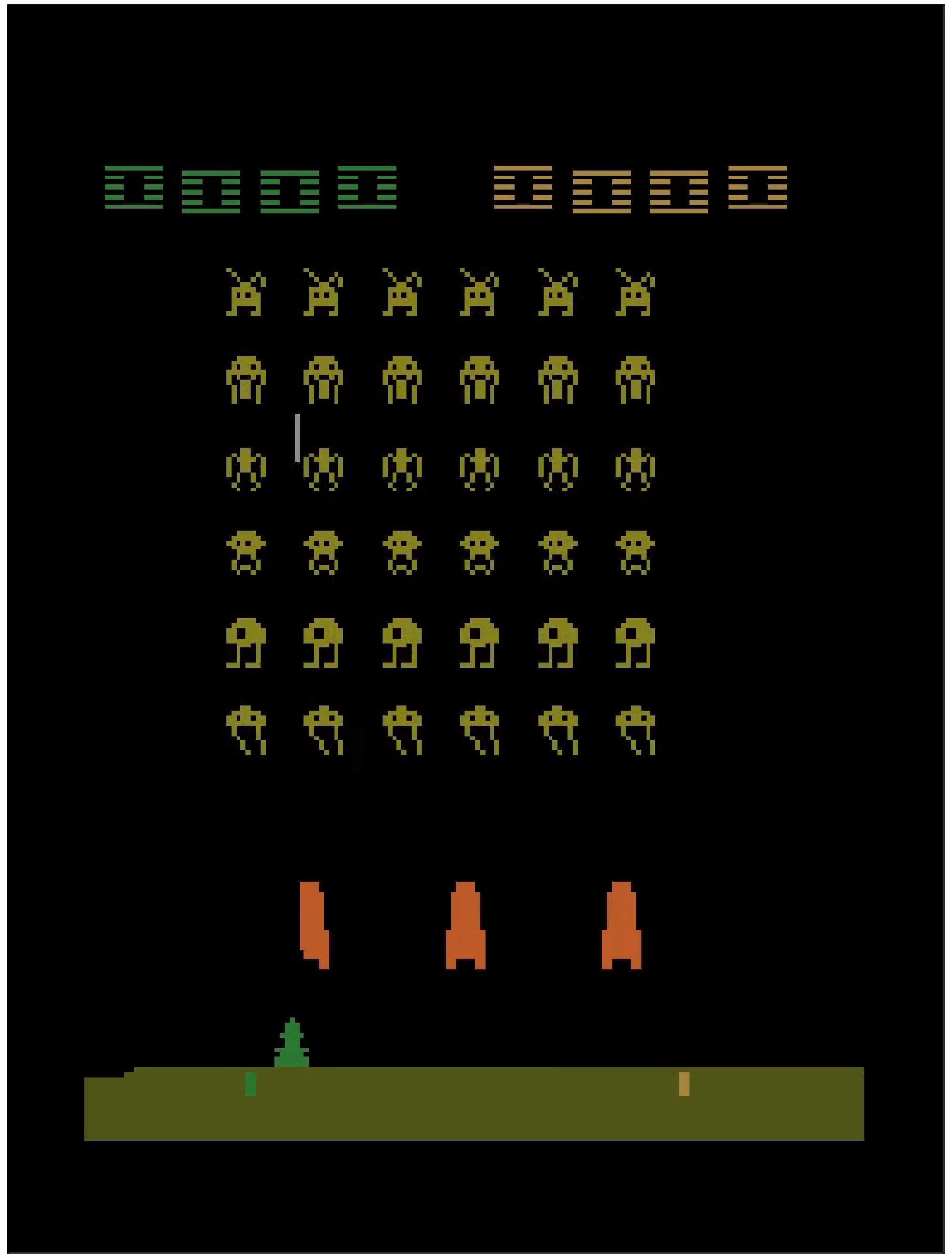


Average reward of best agent (50 episodes):
- 252

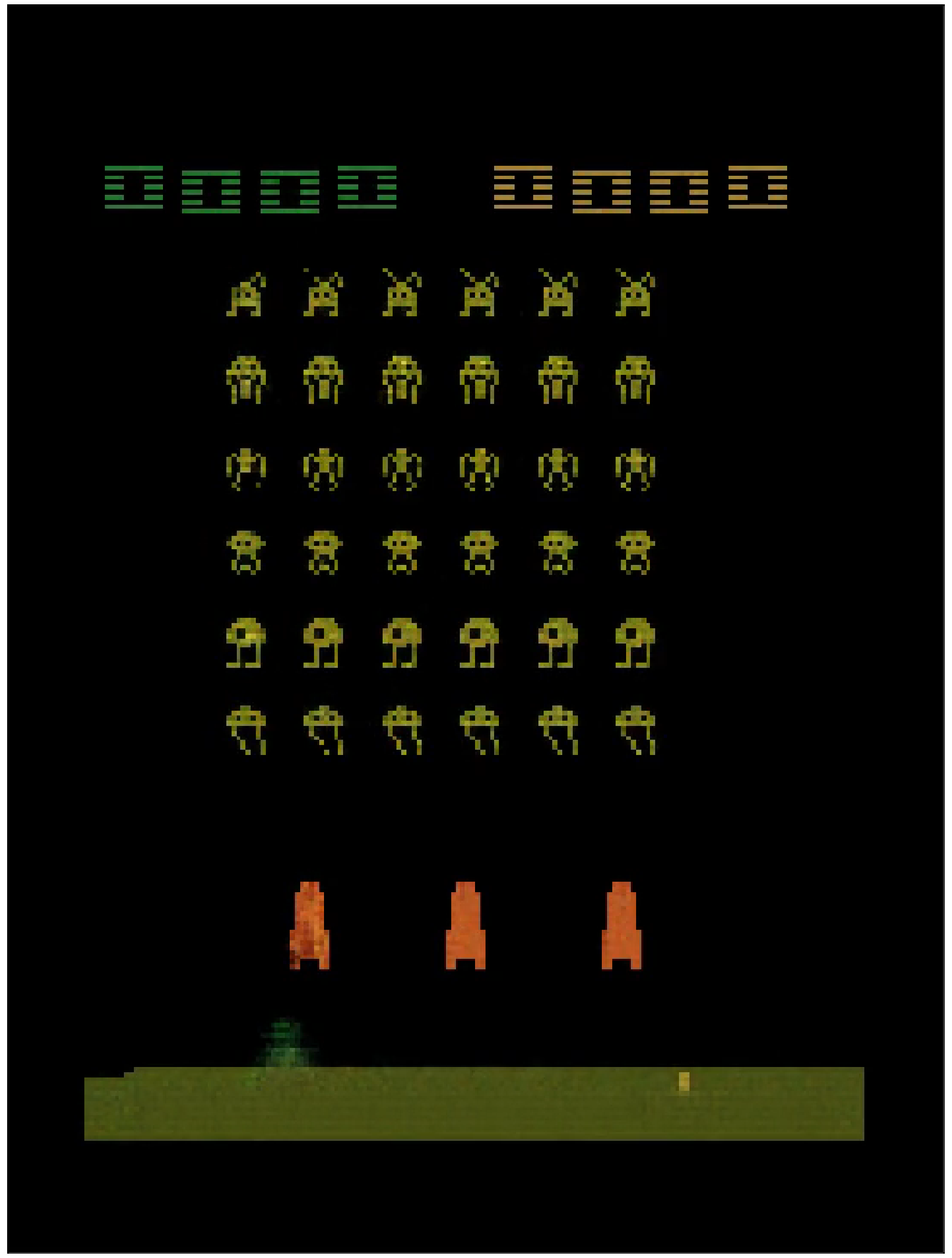
Average reward of random agent (1000 episodes):
- 155

FZI FORSCHUNGSZENTRUM INFORMATIK

Analysis of training results



Average episode of best agent.

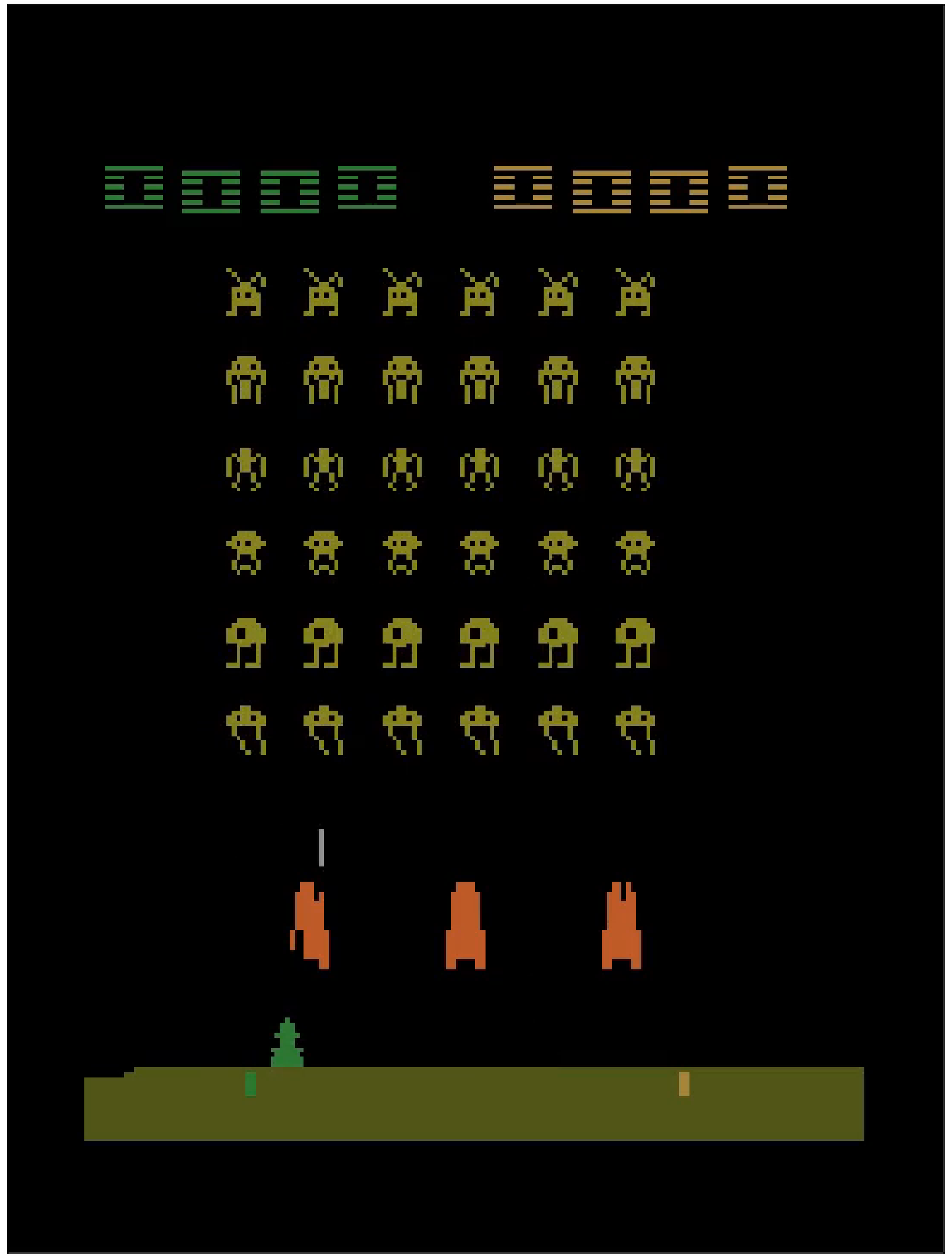


Same episode after autoencoder.

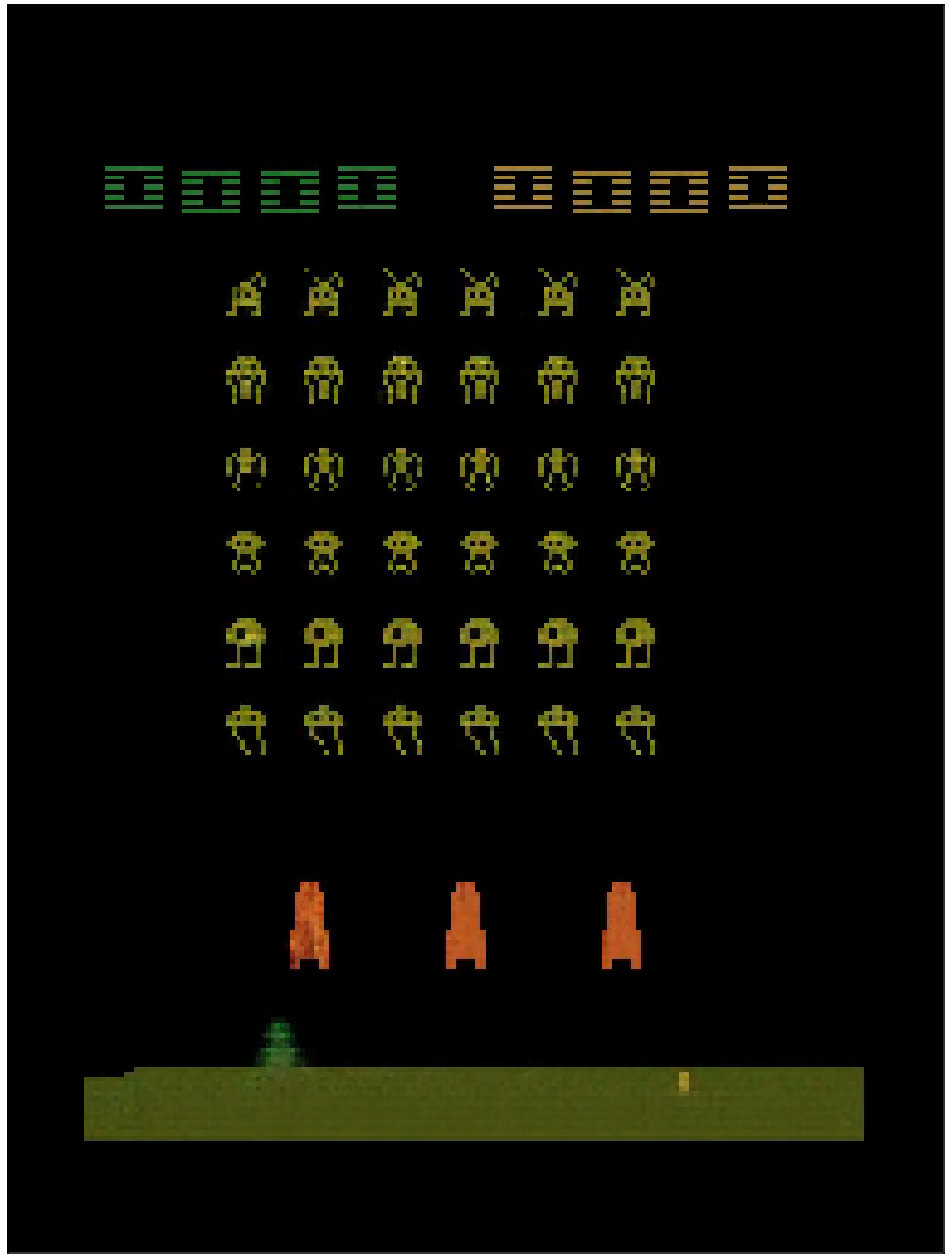
Findings:

- Agent can't see lasers or mothership ⇒ Details are poorly represented by autoencoder.
- Player ship and aliens are not represented correctly by the autencoder for the right side of the game.

Analysis of training results



Average episode of random agent.



Same episode after autoencoder.

Findings:

- Autoencoder is trained with episodes of random actions. ⇒ Player ship rarley enters the right side of the game.
- Representation of player ship and aliens on right side of game fail as such situations have not been part of the training.

Now, how to teach space invaders to your computer?

	"Classic" Reinforcement Learning method	"Evolutionary" Reinforcement Learning method
Possible strategy to solve Space Invaders:	• Pick an algorithm from scientific publications • Implement and tune the algorithm. Some open source implementations: https://github.com/reinforceio/tensorforce https://github.com/openai/baselines	• As shown in this presentation. • Add loop to train autoencoder with episodes generated by agent. • Support learning of laser shots by e.g. modifying the loss functions or increasing visibility of laser shots.
Required domain knowledge:	• "Classic" Reinforcement Learning theory • Deep supervised learning	• Deep supervised learning • Auto encoders & computer vision
Chance of success:	• High	• Uncertain
Generalization:	• Uncertain	• Might translate nicely to other problems.

Summary

Reinforcement Learning is beautiful and has interesting applications, ...
... but is also a challenging field.

To solve Reinforcement Learning problems you may wish to study the extensive theory, ...
... or try to apply a evolutionary method.

CMA-ES is often a reliable choice for black-box optimization, ...
... but only if the dimensionality of the problem is within limits ($\lesssim$1000)

Autoencoders can be used for dimensionality reduction of Reinforcement Learning problems, ...
... but only if the solution does not require fine details, ...
... and only if all possible observations have been included in the training set.

Alternatively, many common Reinforcement Learning algorithms are available and free to use, ...
... but you will probably need to understand theory to use them properly.

Find code and slides on:
https://github.com/david-woelfle/How_to_teach_space_invaders_to_your_computer

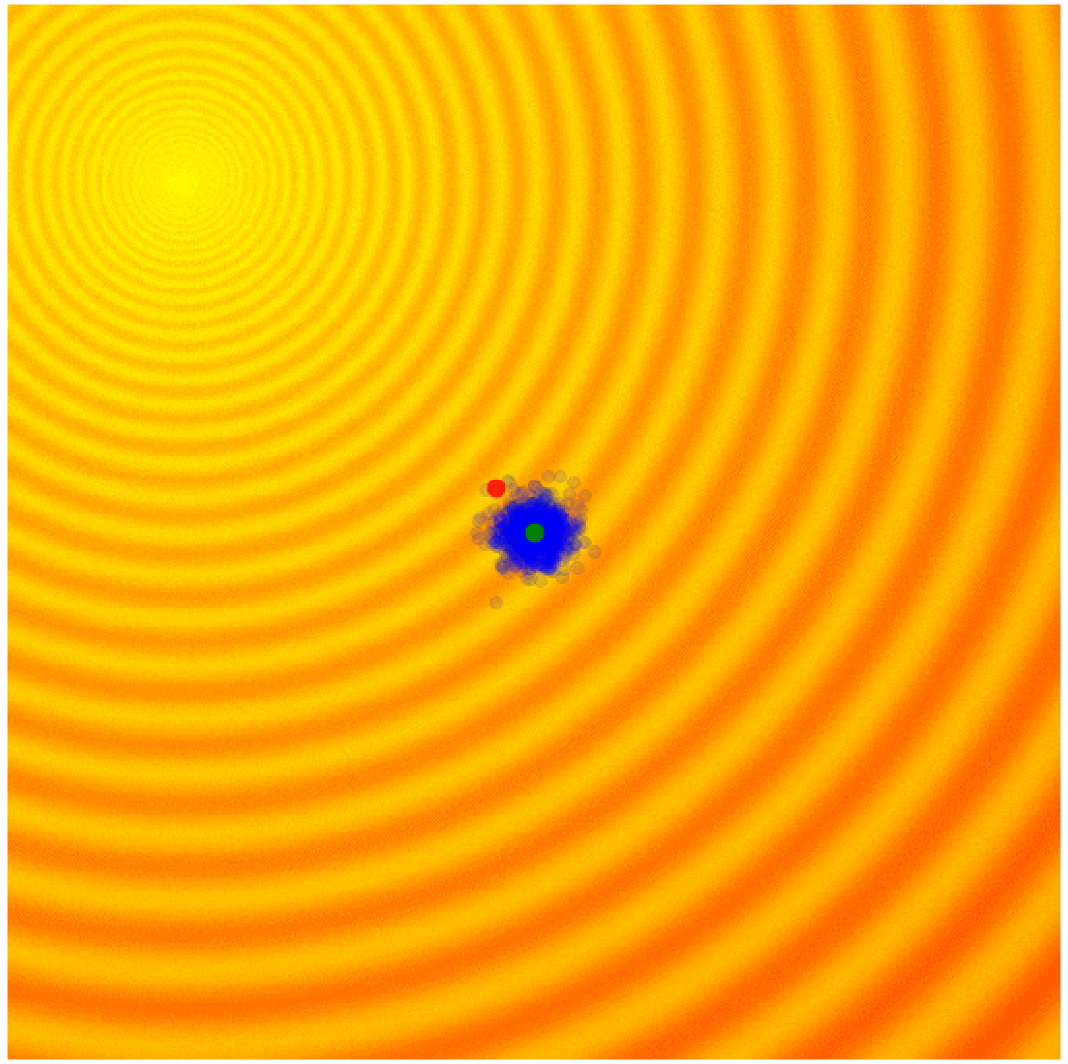
Contact me:
woelfle@fzi.de; https://www.linkedin.com/in/david-woelfle/

FZI FORSCHUNGSZENTRUM INFORMATIK

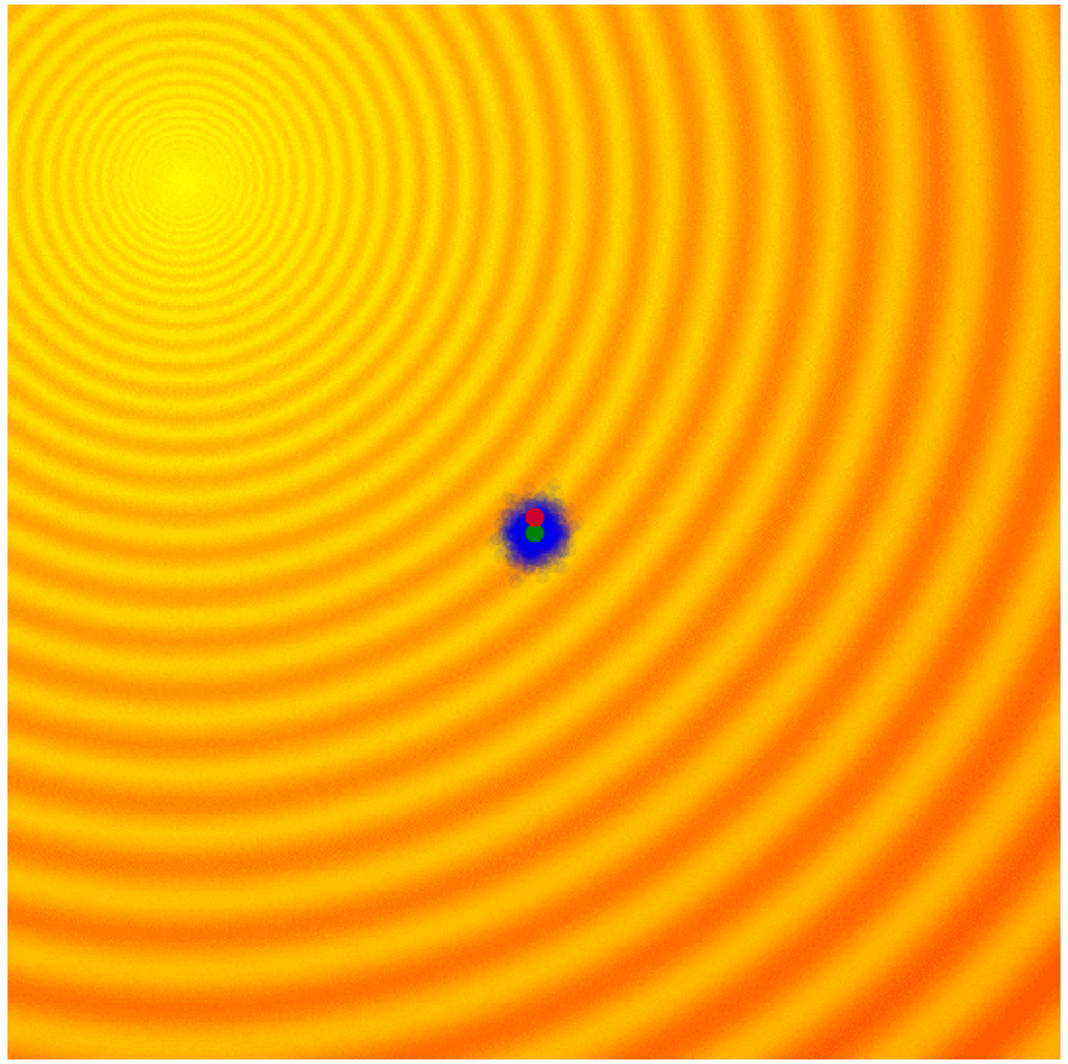
Backup

CMA-ES

Adaptive standard deviation (covariance matrix) of CMA-ES



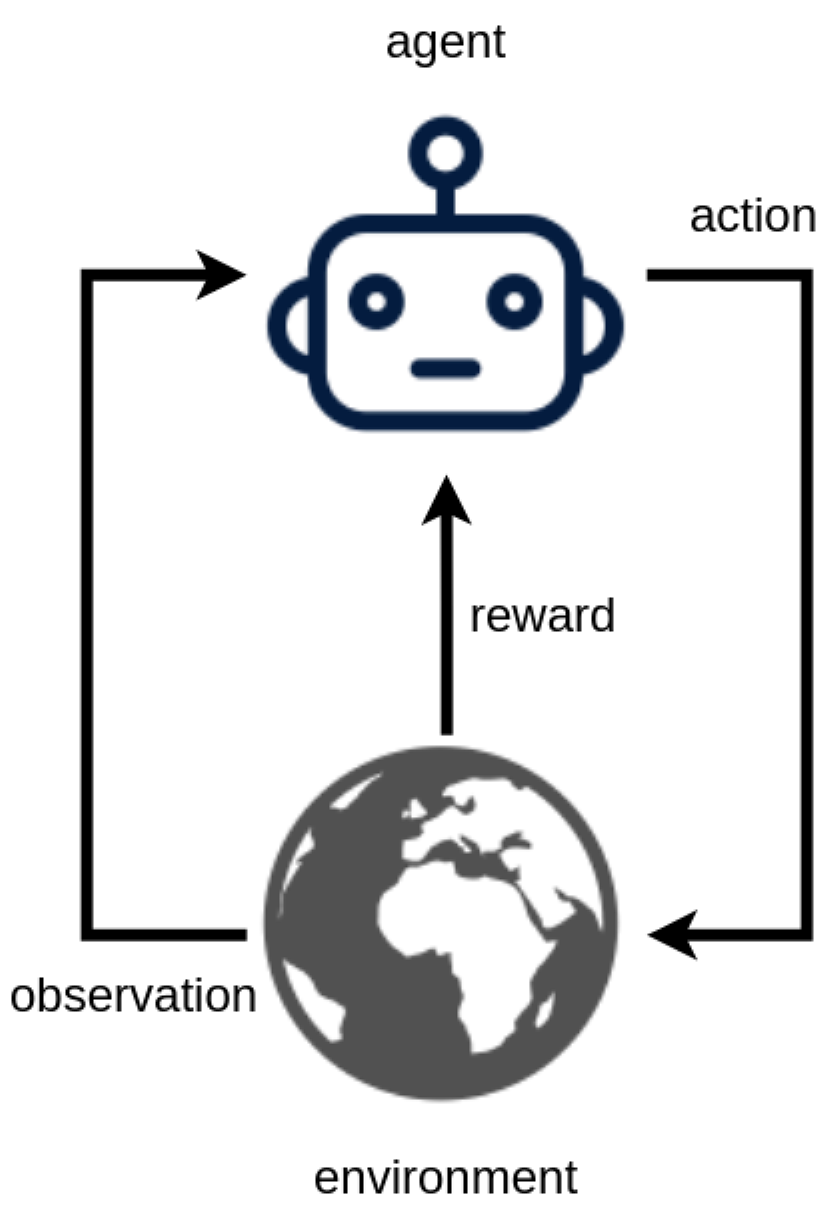
CMA-ES



Simple Evolution Strategy

Taken from http://blog.otoro.net/2017/10/29/visual-evolution-strategies/

Black-box Optimization for Reinforcement Learning



A black-box method would be:

Consider an agent for which a parameter vector **X** determines how the agent computes an action given an observation. (And optionally a reward.)

- Initialize a black-box optimization algorithm.
- Repeat until stopping criterion is met:
  - Sample one or more candidates of **X** from black-box optimization algorithm
  - Compute the expected total reward corresponding to each **X** by unrolling (that is letting the agent play) one or more episodes utilizing the agent loaded with the **X** under review.
  - Trigger optimization step (i.e update) of algorithm with respect to the canidates **X** and the corresponding estimated total rewards.

See also: https://en.wikipedia.org/wiki/Random_optimization