Smart Bots for Better Games: Reinforcement Learning in Production

Olivier Delalleau
Data Scientist @ Ubisoft La Forge

GAME DEVELOPERS CONFERENCE
MARCH 18–22, 2019 | #GDC19

Objective

Share **practical** lessons from using
RL-based **bots** in video game **production**

Agenda

1. RL & games
2. Learning from **pixels**
3. Learning from **game state**
4. Learning from **simulation**
5. Epilogue

1. **Reinforcement learning & games**
2. Learning from pixels
3. Learning from game state
4. Learning from simulation
5. Epilogue

[image source]

THE VERGE TECH SCIENCE ENTERTAINMENT MORE

GAMING \ TECH \ ARTIFICIAL INTELLIGENCE

The world's best Dota 2 players just got destroyed by a killer AI from Elon Musk's startup

By T.C. Sottek | Aug 11, 2017, 10:48pm EDT

SHARE

Tonight during Valve's yearly *Dota 2* tournament, a surprise segment introduced
be the best new player in the world -- a bot from Elon Musk-backed startup Open
Engineers from the nonprofit say the bot learned enough to beat *Dota 2* pros in ju
weeks of real-time learning, though in that training period they say it amassed "lif
experience, likely using a neural network judging by the company's prior efforts.
hailing the achievement as the first time artificial intelligence has been able to be
competitive e-sports.

MOST READ

PC GAMER THE GLOBAL AUTHORITY ON PC GAMES US Edition

News Reviews Hardware Indie Best Of Magazine Pro

POPULAR Apex Legends Metro Exodus Anthem Epic Store Best PC Games

How the OpenAI Five tore apart a team of Dota 2 pros

By Morgan Park August 11, 2018

We talked to the players and the OpenAI engineers about what it was like to build and face off against a fearsome Dota bot.

COMMENTS

Last weekend, five very good Dota 2 players gathered in San Francisco to play a competitive match—against a computer. Their opponent was OpenAI Five, five neural networks which have been training a *bit* harder than the average Dota player to learn how to be a competitive team: "OpenAI Five plays 180 years worth of games against itself every day, learning via self-play," says the OpenAI blog. I had no idea who would win, but I wanted to be there in person to find

Advertisement

HARDWARE BUYING GUIDES | LATEST GAME REVIEWS

1 The best 4K TV for gaming PCs

2 PC build for Anthem: the parts you need for 60 fps

3 PC build for Metro Exodus: the parts you need for 60 fps

4 The best gaming motherboards 2019

The Download

What's up in emerging technology

DeepMind's new AI just beat top human pro Starcraft II for the first time

DeepMind, a subsidiary of Alphabet that's focused on
landmark in that grand quest: beating humans at galac

The news: AlphaStar, the company's latest learning a
first time, scoring 10 wins and one loss against the pro

we'll never share your details without your permission.

GDC March 18-22, 2019 San Francisco, CA

ABOUT ATTEND CONFERENCE EXPO FEATURES REGISTER

Search All Content SEARCH

ALL SESSIONS

SPEAKERS

★ MY SCHEDULE

ML Tutorial Day: Smart Bots for Better Games: Reinforcement Learning in Production

Olivier Delalleau (Data scientist, Ubisoft)
Location: Room 303, South Hall
Date: Tuesday, March 19
Time: 4:00pm - 5:00pm
Pass Type: All Access, GDC Conference + Summits, GDC Summits - Get your pass now!
Topic: Programming
Format: Tutorial
Vault Recording: Video
Audience Level: Intermediate

ADD

Tweet Like Share

This talk provides an overview of various reinforcement learning algorithms and how they may help
...ent, the ability to train
...or automated testing and
... brings up many
...cent experiments within
... the integration burden,

GDC March 18-22, 2019 San Francisco, CA

ABOUT ATTEND CONFERENCE EXPO FEATURES REGISTER

Search All Content SEARCH

ALL SESSIONS

SPEAKERS

★ MY SCHEDULE

Reinforcement Learning in Action: Creating Arena Battle AI for 'Blade & Soul'

Jinyun Chung (Team Leader, NCSOFT)
Seungeun Rho (Research Engineer, NCSOFT)
Location: Room 2002, West Hall
Date: Tuesday, March 19
Time: 3:50pm - 4:20pm
Pass Type: All Access, GDC Conference + Summits, GDC Summits - Get your pass now!
Topic: AI Summit
Format: Session
Vault Recording: Video
Audience Level: Intermediate

ADD

Tweet J'aime

The NCSOFT team applied reinforcement learning to create an AI for the arena 1v1 battle in 'Blade & Soul', a global MMORPG. The AI agents participated in the 2018 'Blade & Soul' Tournament World Championship as blind matches and played against three top professional players from across the globe. The AI had 3 wins and 4 loses - an impressive showing against professional players. In this session, the NCSOFT team will will share their experiences on how they built pro-level AI agents.

Takeaway

Attendees will learn how NCSOFT made professional level AIs exclusively with reinforcement learning, as well as hear the NCSOFT team's experiences with solving various issues when applying reinforcement learning to a commercial game.

Intended Audience

Anyone who is interested in applying reinforcement learning to commercial games.

[image sources: #1 #2 #3 #4 #5]

Potential application #1: player-facing AI

Blade & Soul (2016)
SHIPPED

Starcraft II
(AlphaStar, 2019)
RESEARCH

Black & White (2001)
SHIPPED

[image sources: #1 #2 #3 #4]

Dota 2
(OpenAI Five, 2018)
RESEARCH

Potential application #2: testing assistant

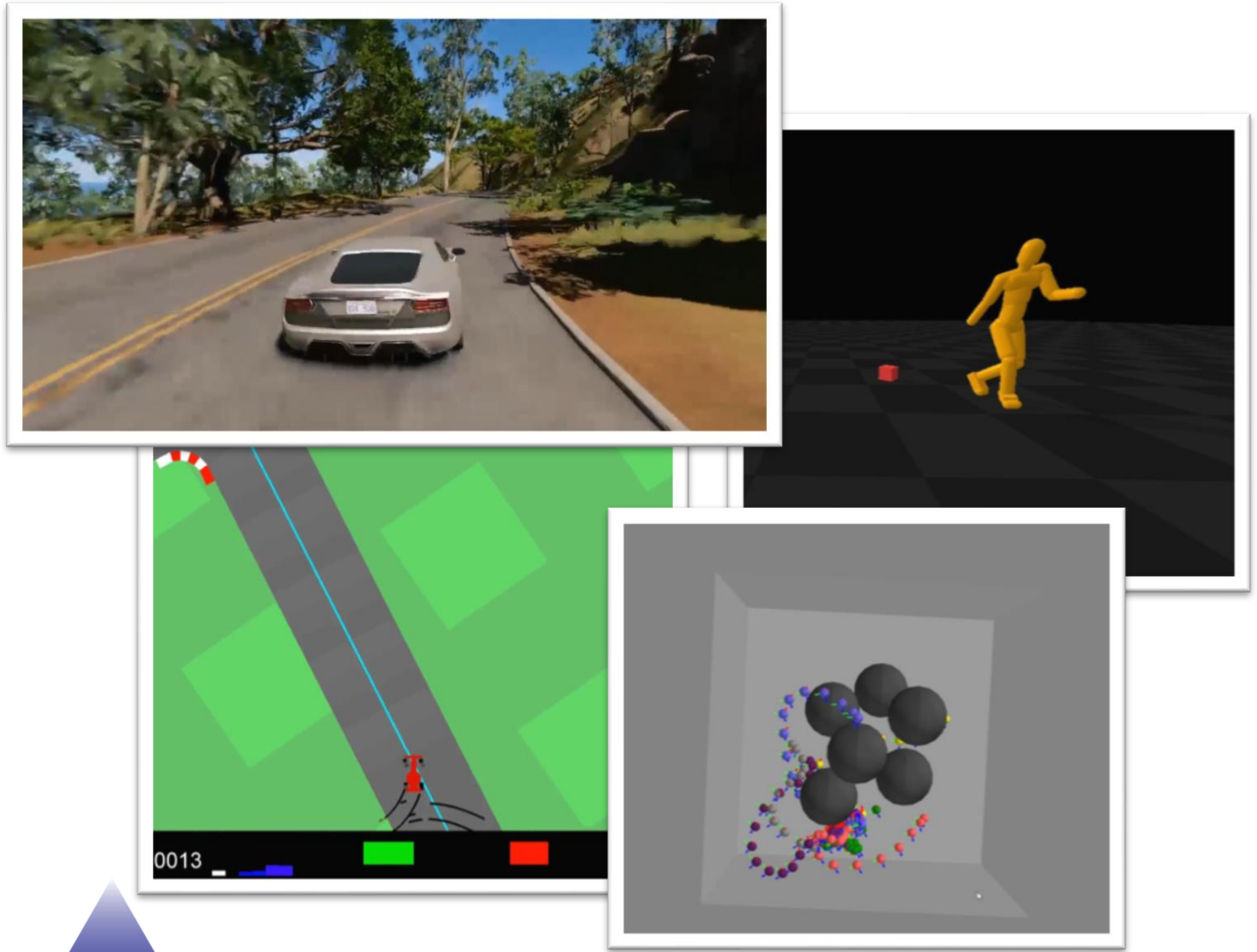
Open World
(Far Cry: New Dawn, 2019)

Live multiplayer
(Tom Clancy’s Rainbow Six Siege, 2015-…)

[image sources: #1 #2]

In this talk: prototypes @Ubisoft

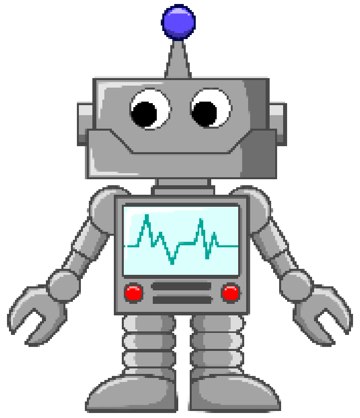
Goal: player-facing AI



Goal: testing assistant

What is reinforcement learning?

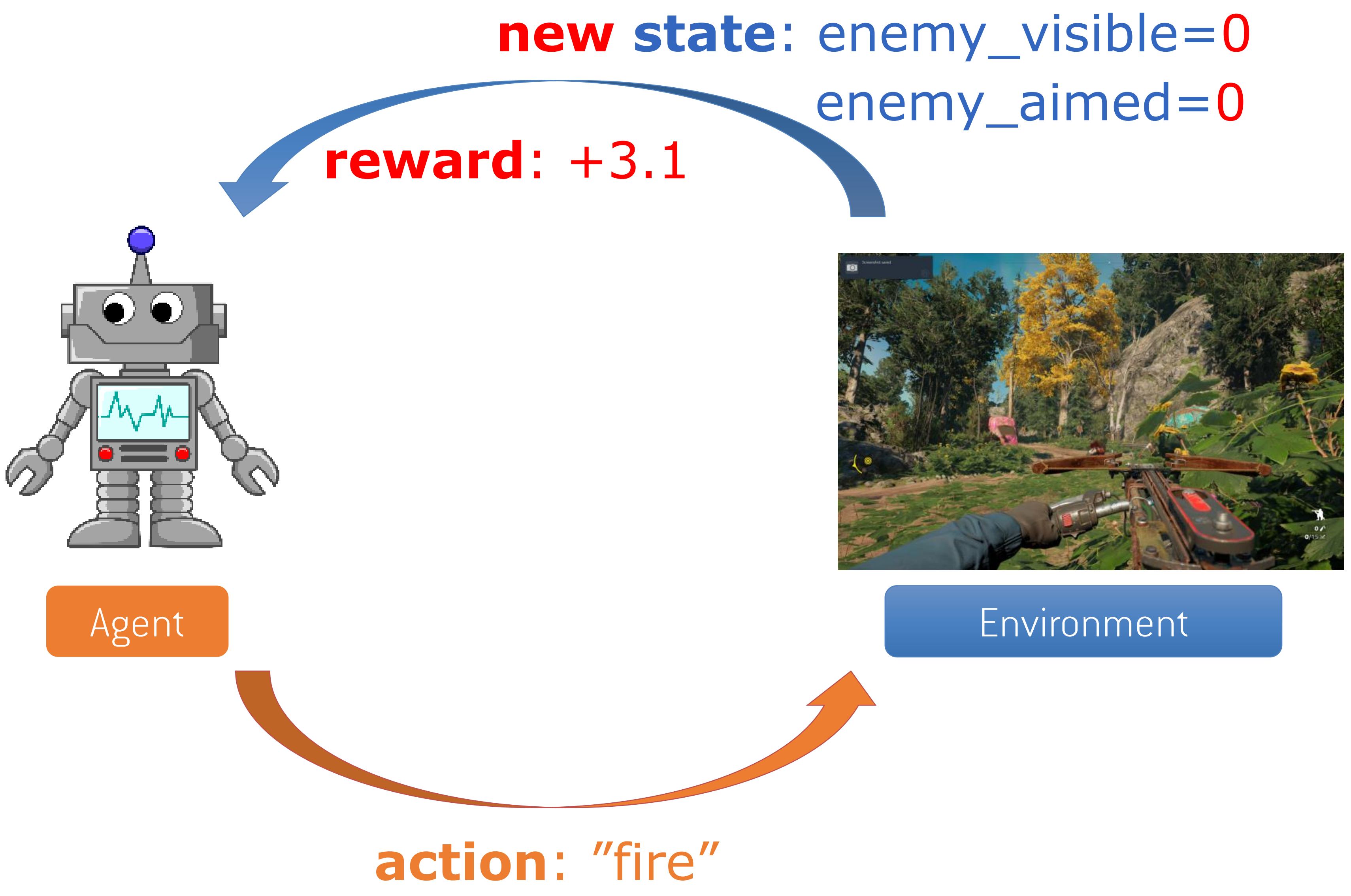
state: enemy_visible=1
enemy_aimed=1

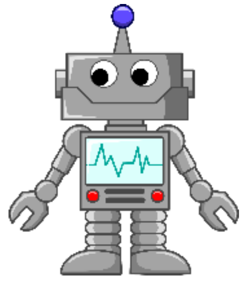


Agent

Environment

What is reinforcement learning?





Objective:

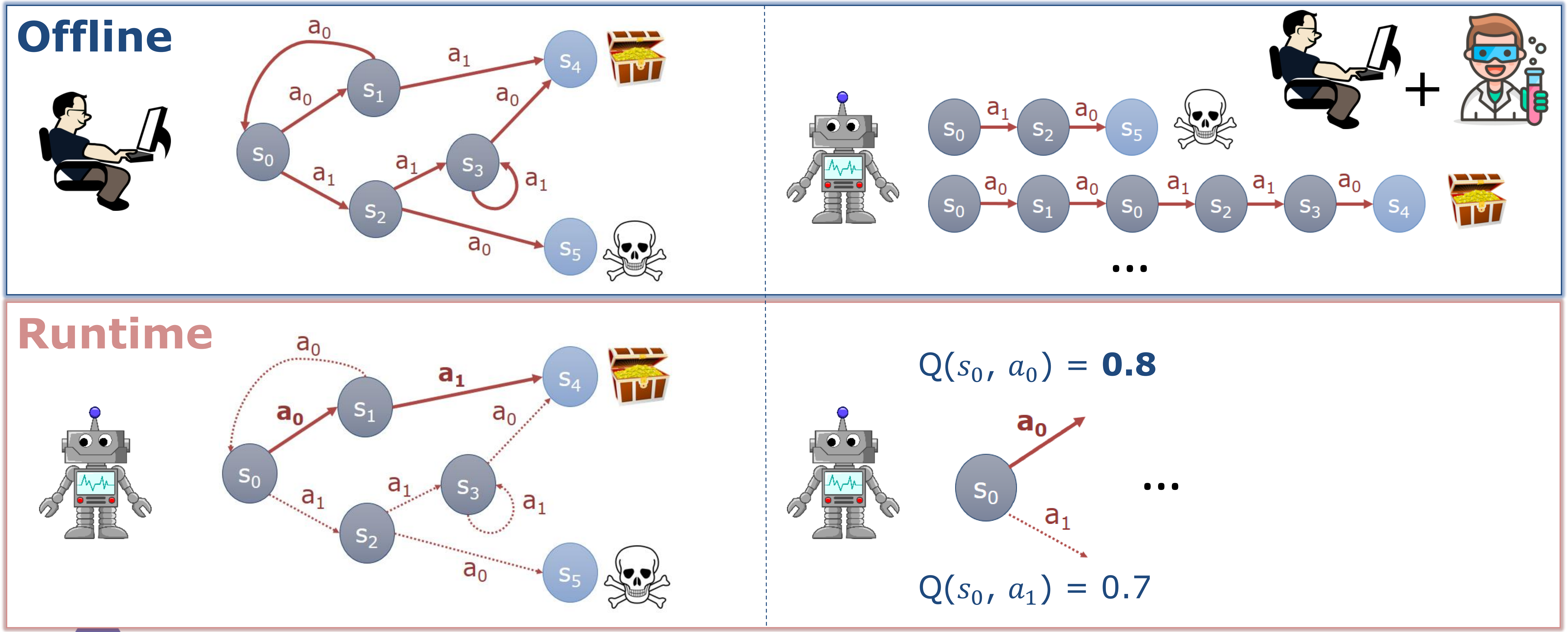
Find **optimal** action in each state to maximize the **sum of rewards**

Q-values:

Q(state, action) = sum of future rewards when taking an action in a given state

Taking the **optimal action = taking** the action with **maximum Q-value**

Search/Planning vs RL



[scientist icon by Vincent Le Moign]

In a nutshell: why reinforcement learning?

- Automated AI generation from reward alone
- "If you can play, you can learn"
- It's not magic (…)

[icons from pngimg.com]

1. Reinforcement learning & games
2. **Learning from pixels**
3. Learning from game state
4. Learning from simulation
5. Epilogue

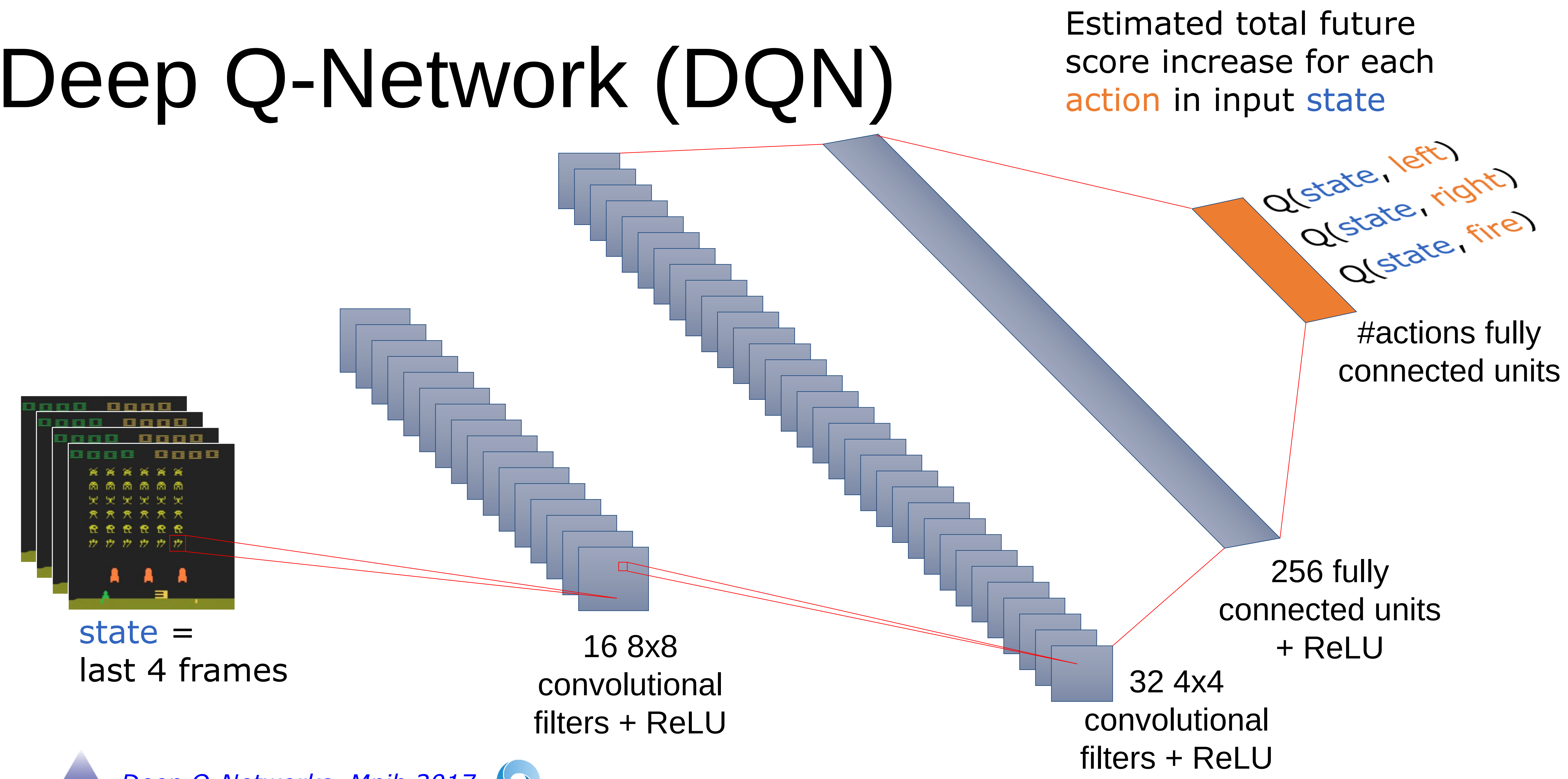
[image source]

Deep Q-Network (DQN) in action

[Human-level control through Deep Reinforcement Learning](#) *(Mnih et al. 2015)*

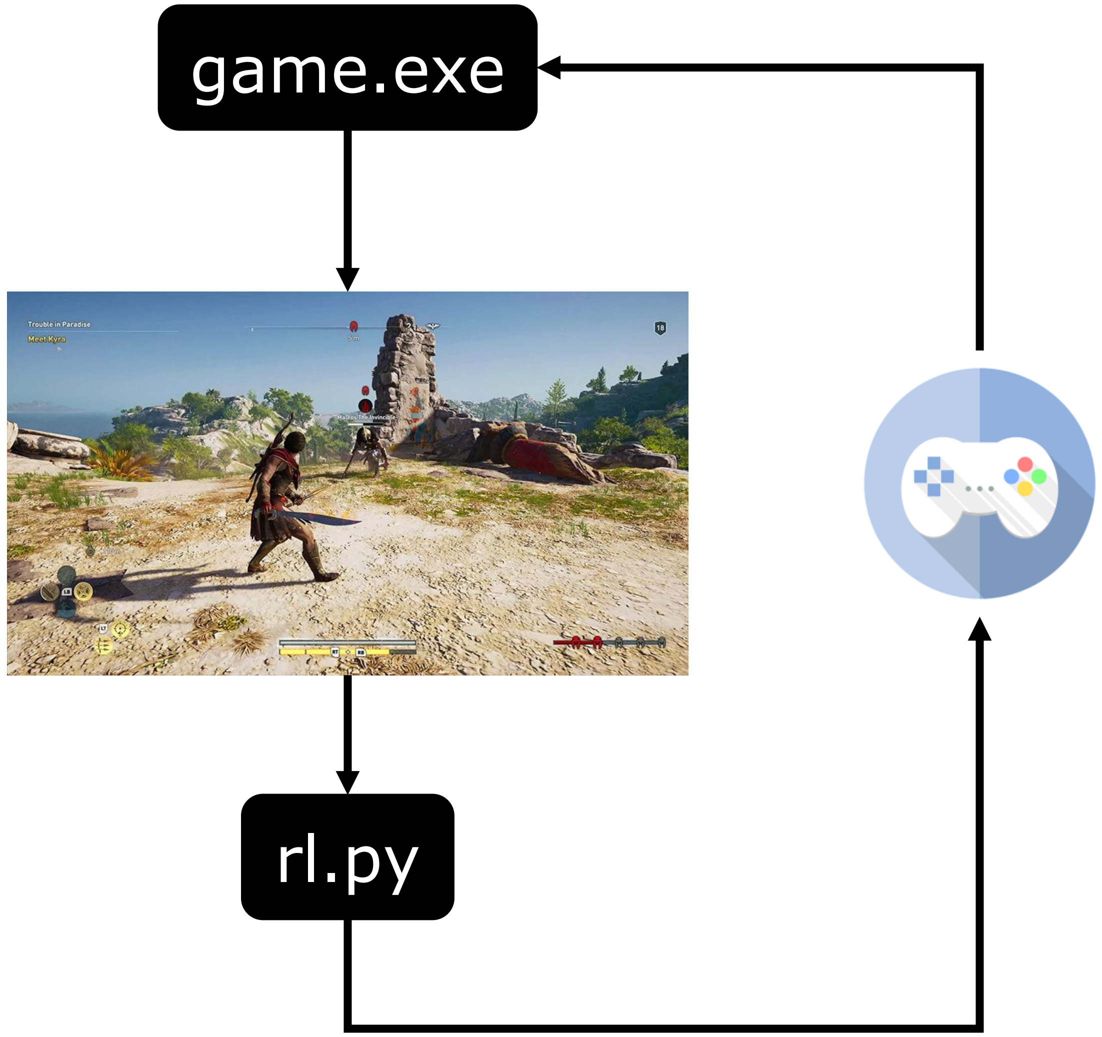
https://www.youtube.com/watch?v=DqzSrEuA2Jw

Deep Q-Network (DQN)



[Deep Q-Networks, Mnih 2017](#)

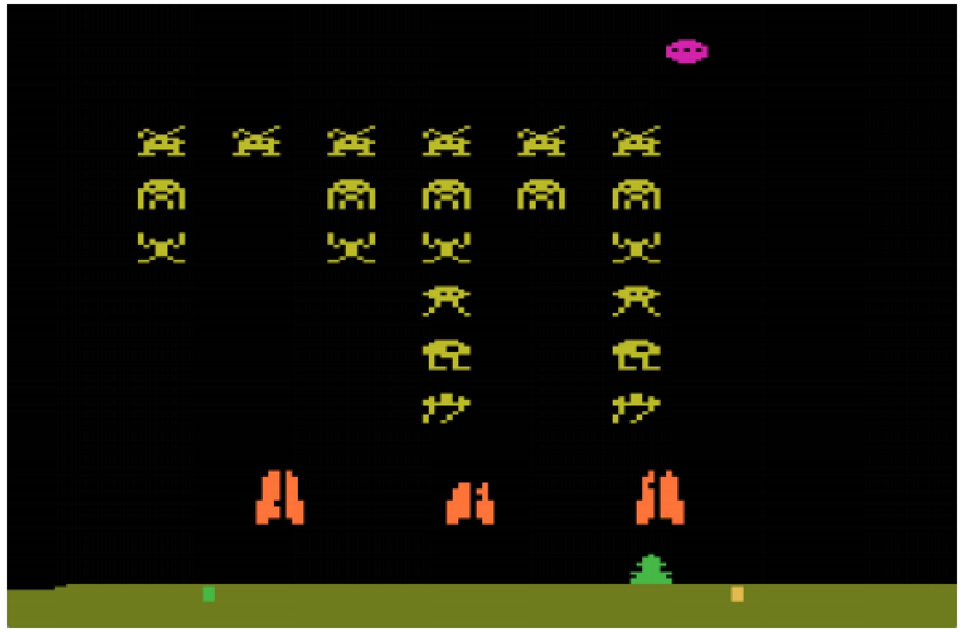
Why learn from pixels?



- Generic
- No need to access code

[screenshot source]

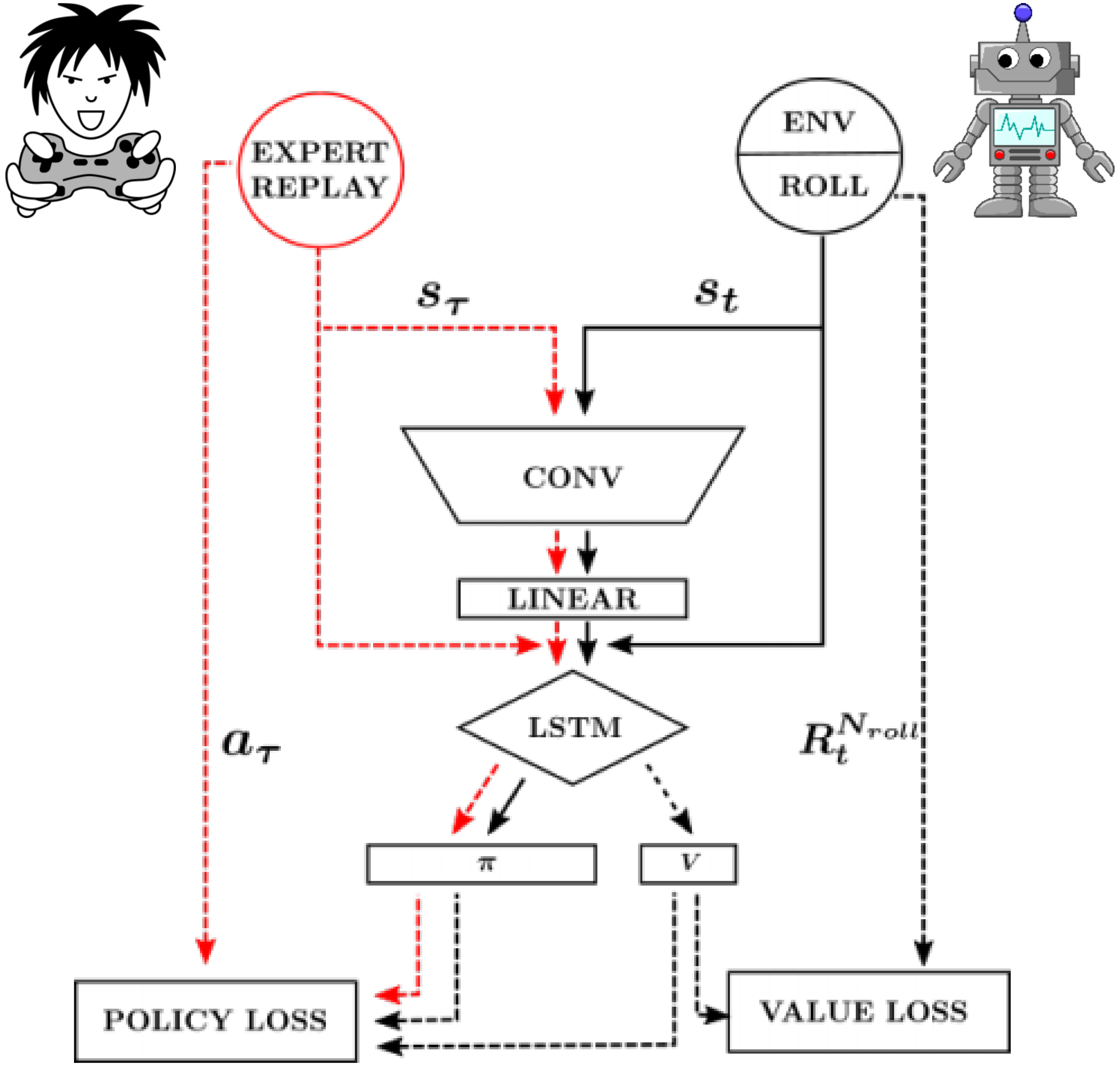
From Atari to AAA games



[image sources: #1 #2 #3 #4]

State: Simplify graphics

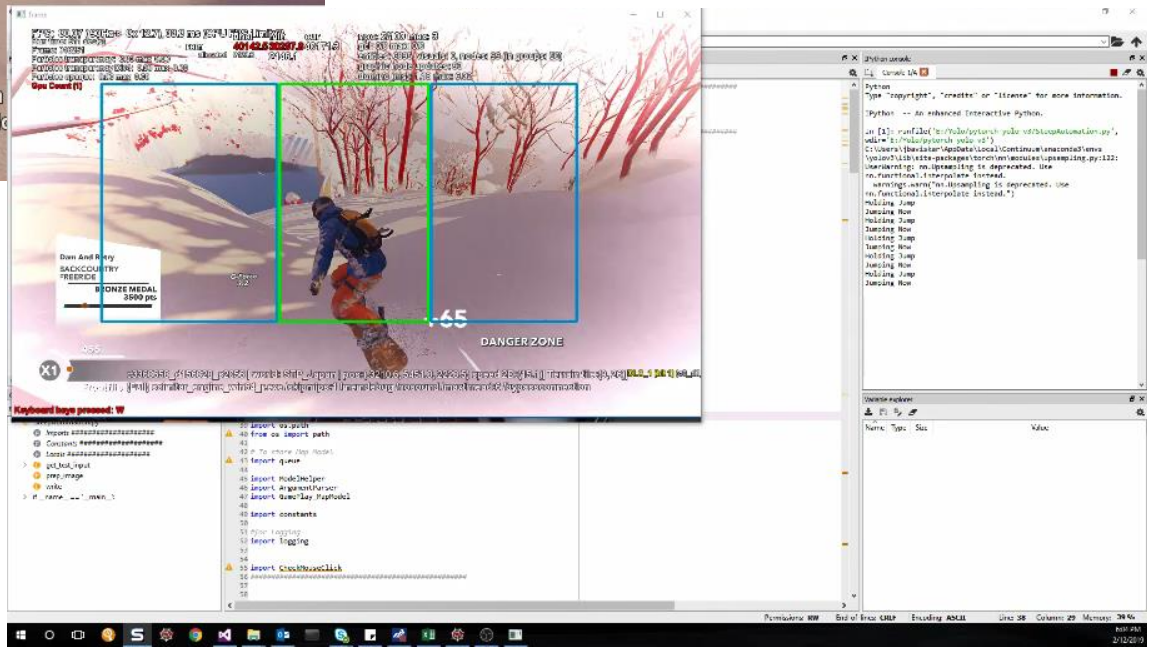
Actions: Imitate humans



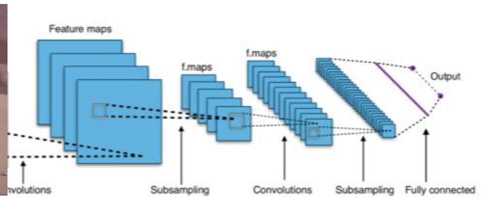
[Imitation Learning with Concurrent Actions in 3D Games](#)
(Harmer et al. 2018)

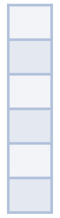
Challenges with pixel-based learning

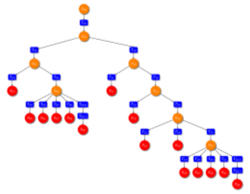
Steep automated testing
(Ubisoft Pune)

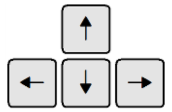


vision | **decision making**







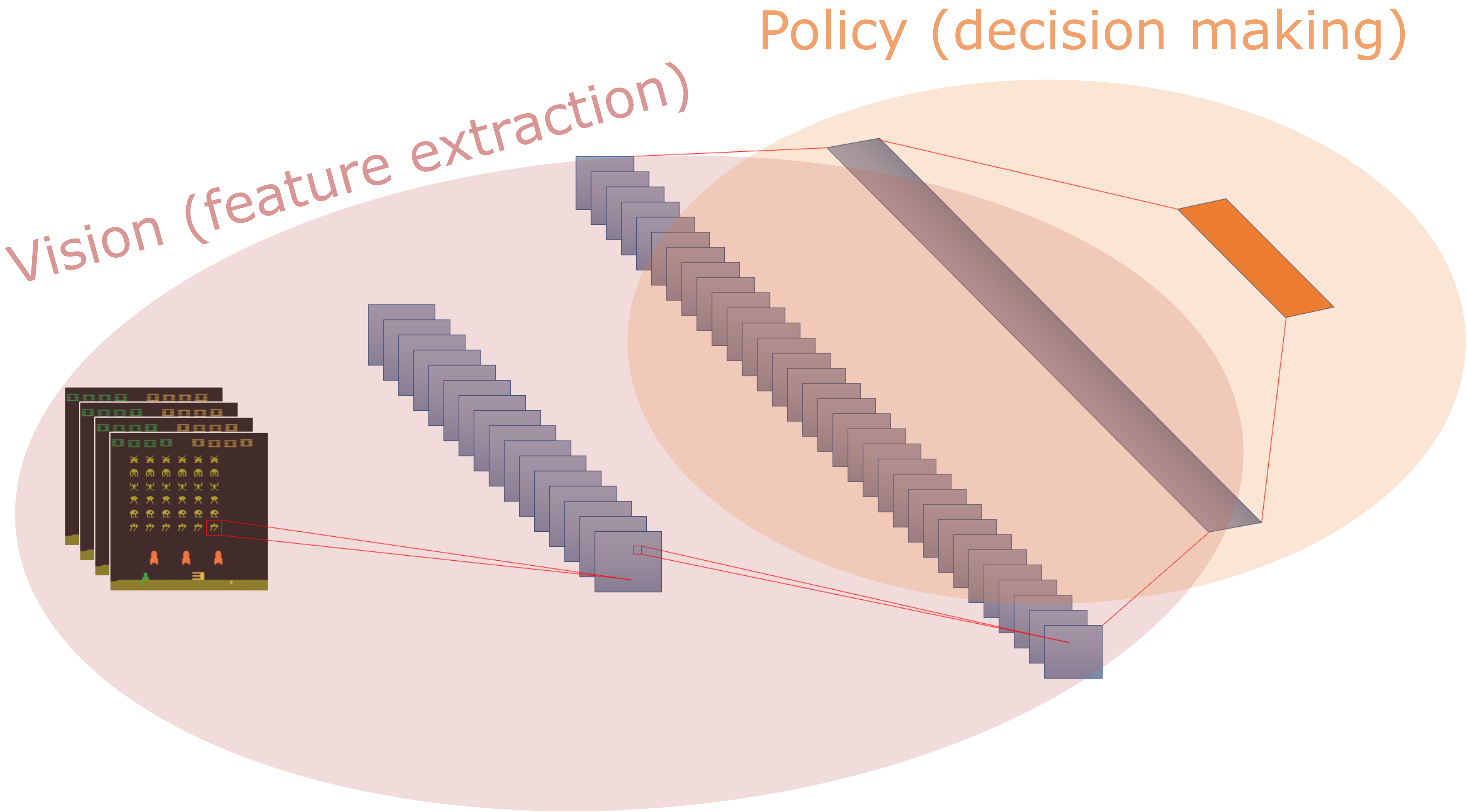


screen CNN objects tree action

[CNN diagram by Aphex34] [tree diagram source]

Why **not** learn from pixels?

- **Complex training**



Why **not** learn from pixels?

- Complex training
- **Large neural network**

Why **not** learn from pixels?

- Complex training
- Large neural network
- **Costly GPU rendering**

Why **not** learn from pixels?

- Complex training
- Large neural network
- Costly GPU rendering
- **Partial observability**

[images source]

Why **not** learn from pixels?

- Complex training
- Large neural network
- Costly GPU rendering
- Partial observability
- **Less than ideal for…**
  - **UI details** 

  - **Sound** 
  - **Reward** 

[headset image source]

In a nutshell: learning from pixels

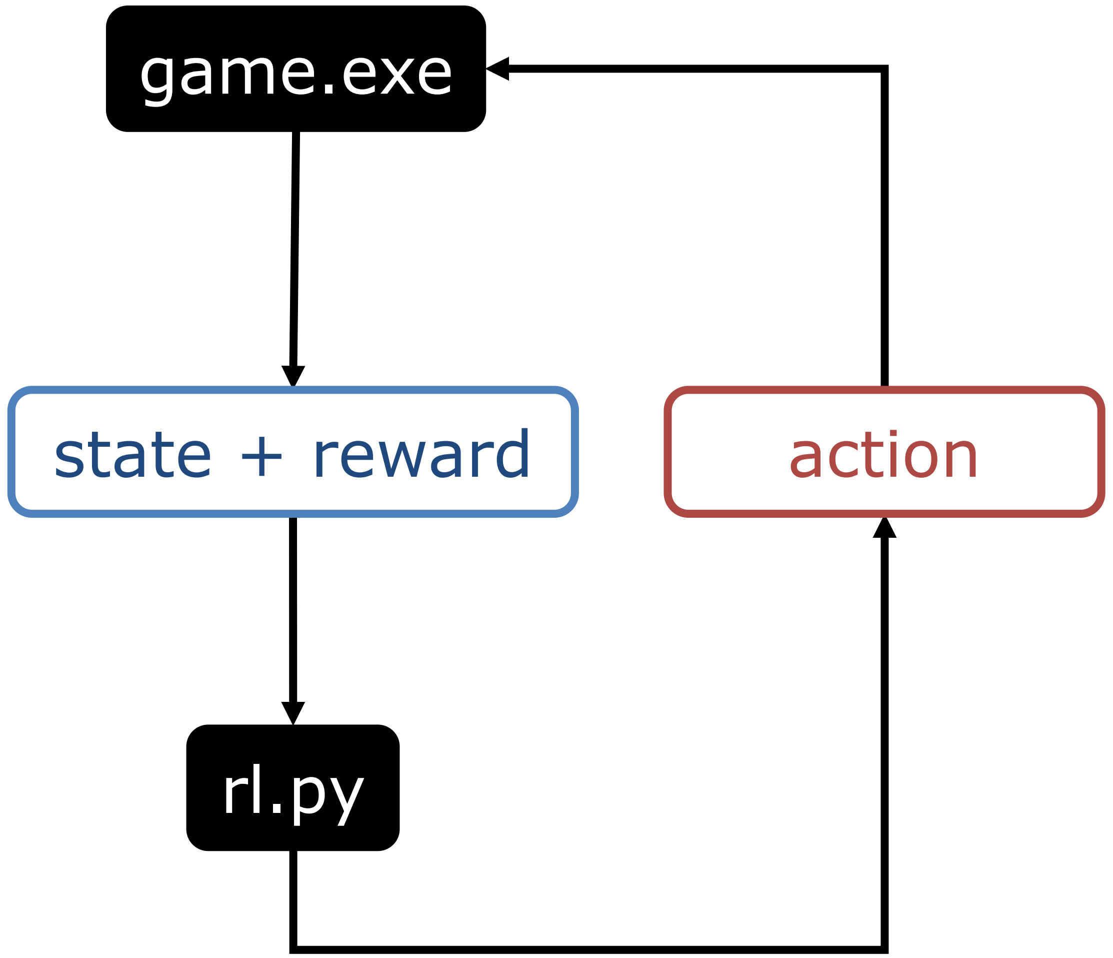
+ **Attractive** due to its genericity…

− … but **trickier** than it seems and computationally **costly**

1. Reinforcement learning & games
2. Learning from pixels
3. **Learning from game state**
4. Learning from simulation
5. Epilogue

[image source]

Learning from game state



- Cheap to compute and process
- Can add unobserved information
- Can inject domain knowledge

Learning from game state

- **AI testing in For Honor**
- RL-based driving in Watch_Dogs 2

AI testing in For Honor

Example: kiting exploit

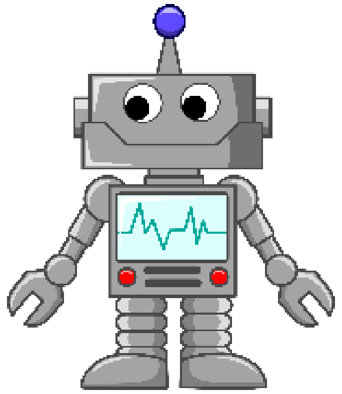
SmartBot
vs
Game AI

RL loop in For Honor

state

distance_to_target=3
self_HP=110
self_stance="top"
self_stamina=60
self_animation_id=4
target_HP = 65
...

reward:
damage_to_opponent – damage_received



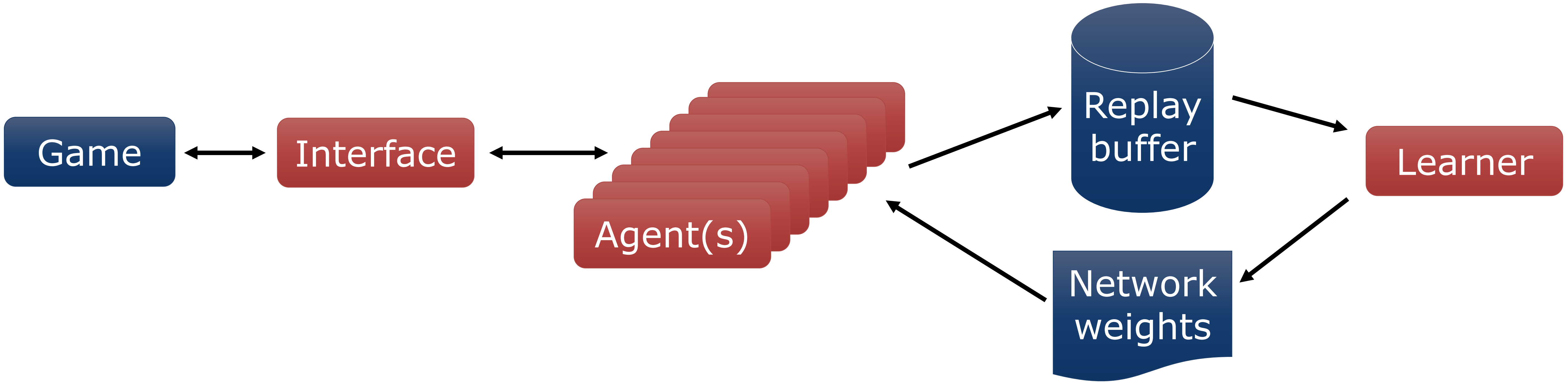
Agent

Environment

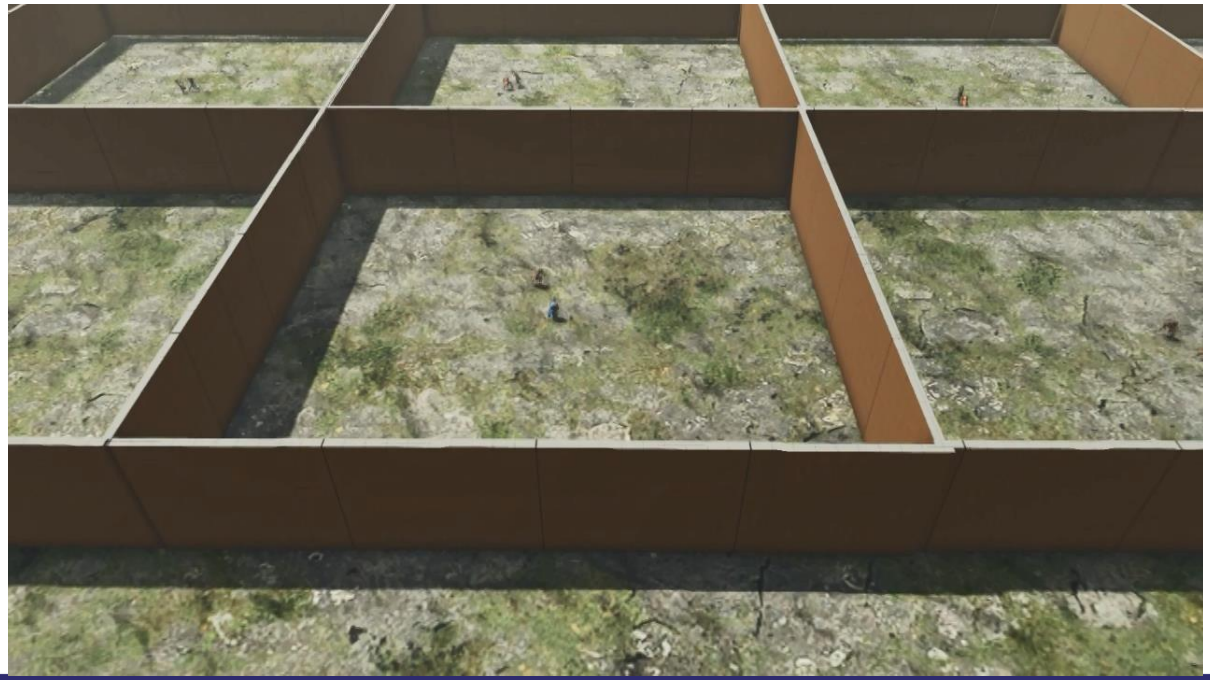
action

"attack_light_left"
"attack_heavy_top"
"block_top"
"dodge_back"
"guard_break"
...

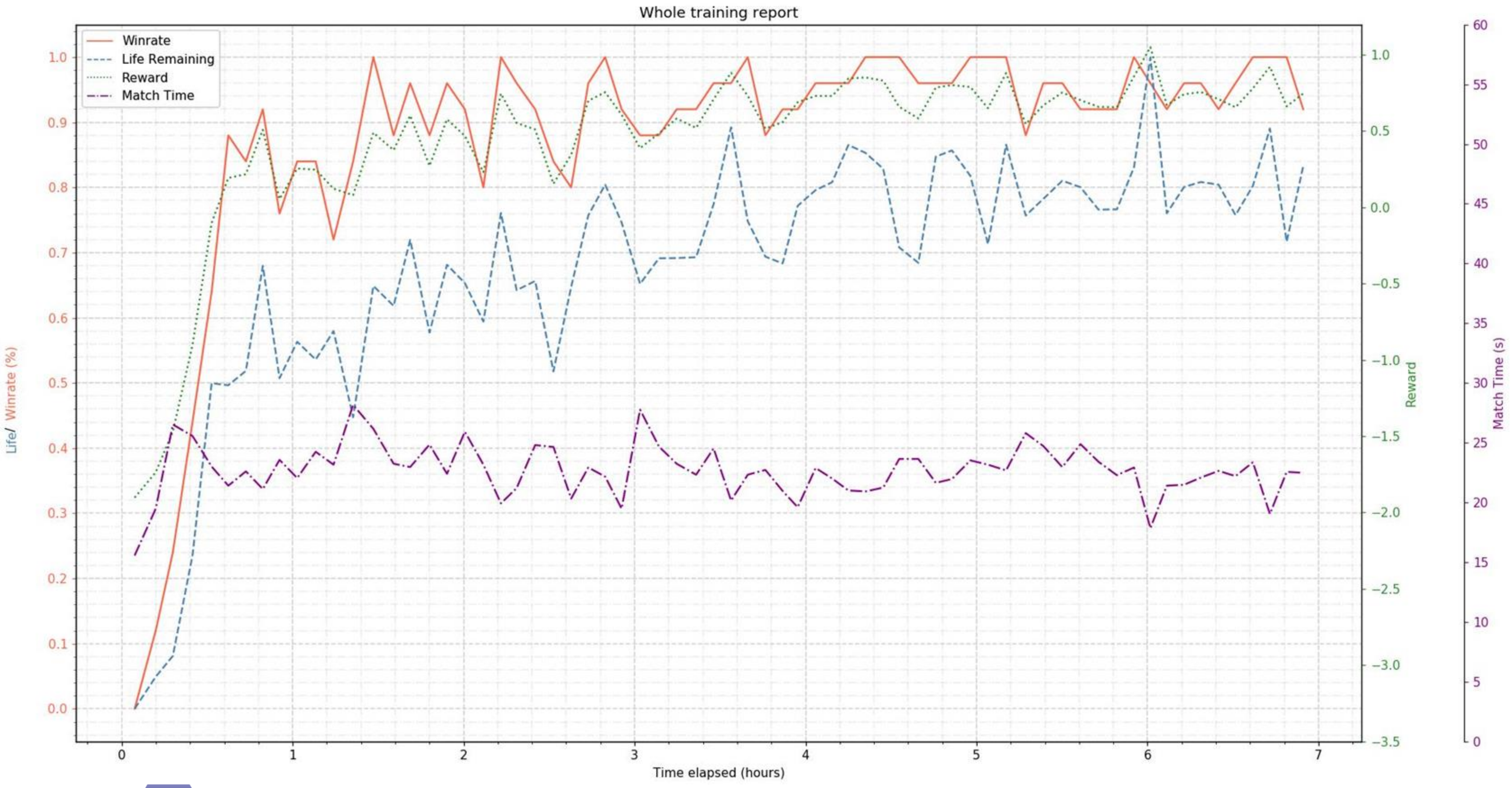
DQN with distributed experience replay



[*Distributed Prioritized Experience Replay*](#)
(Horgan et al. 2018)



Example: poor defense vs zone attack



Example: poor defense vs zone attack

Action Info: Distribution of the actions

	1st	2nd	3rd	4th
All actions distribution	Dodge_Back(23.5%)	Attack_AoE_Special_Strike_Part1(17.9%)	Dodge_Left(11.9%)	Do
% damage done	Attack_AoE_Special_Strike_Part1(68.4%)	Attack_AoE_Special_Strike_Part2_(16.0%)	Attack_SideHeavy_Strike_Left(11.6%)	Att

Example: poor defense vs zone attack

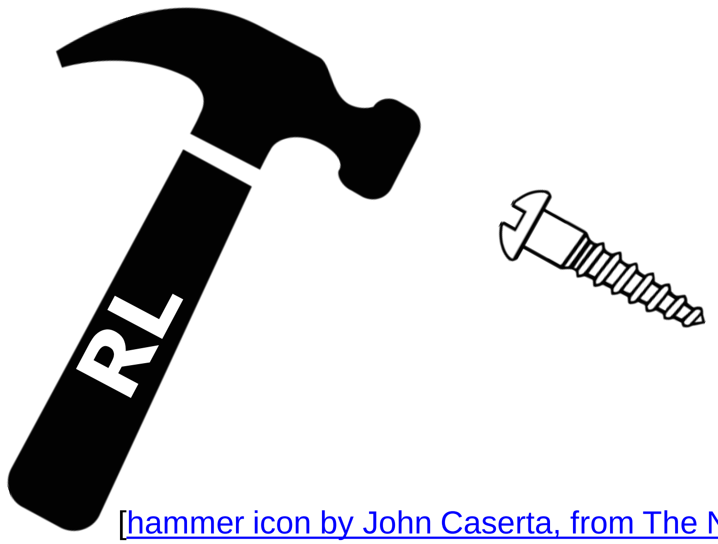
SmartBot
vs
Game AI

Some lessons we learned the hard way

- The RL loop (state → action → reward) must be bulletproof
  - Complex gameplay logic was modifying actions chosen by our agent

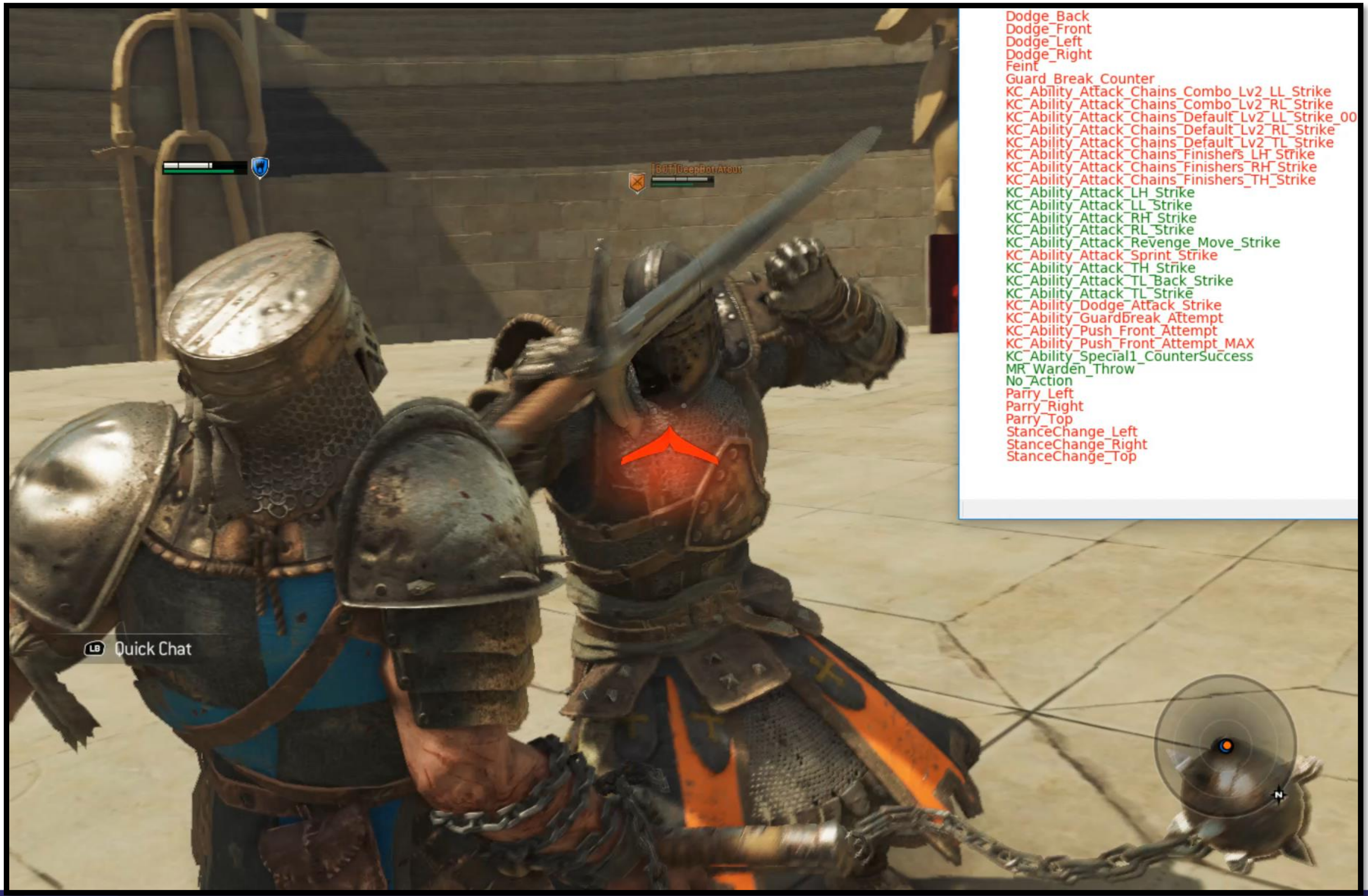
- Ability to reset state & replay matches can help a lot
  - We did not collect enough data in rare-but-important situations
  - Debugging from logs only was painful

- RL is not always the best solution
  - Was found inefficient on some tasks



[hammer icon by John Caserta, from The Noun Project]

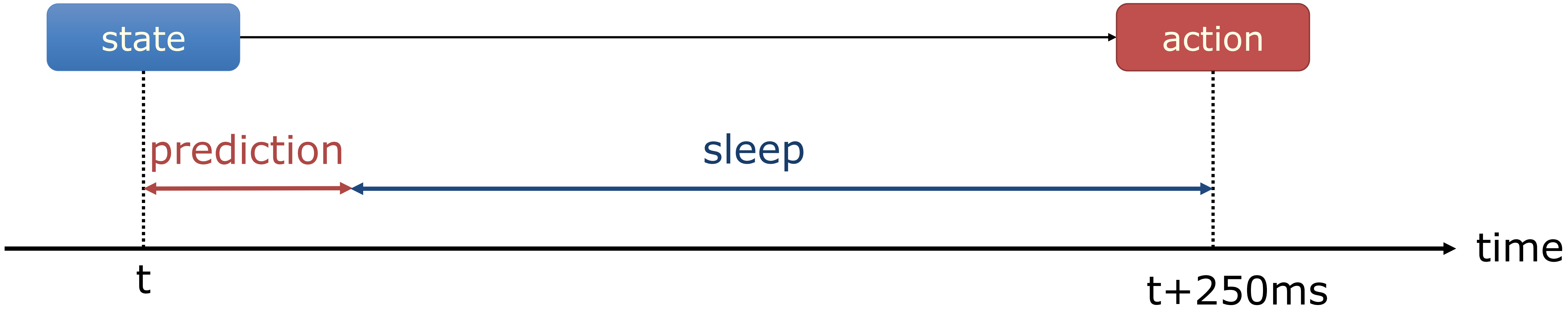
Action mask



Alternatively: penalize incorrect actions
(ex: Borovikov et al. 2019)

Human-like reaction time

Ex: mimic 250ms reactions



Reward shaping: bonus for guard break

SadisticBot
~~SmartBot~~
vs
Game AI

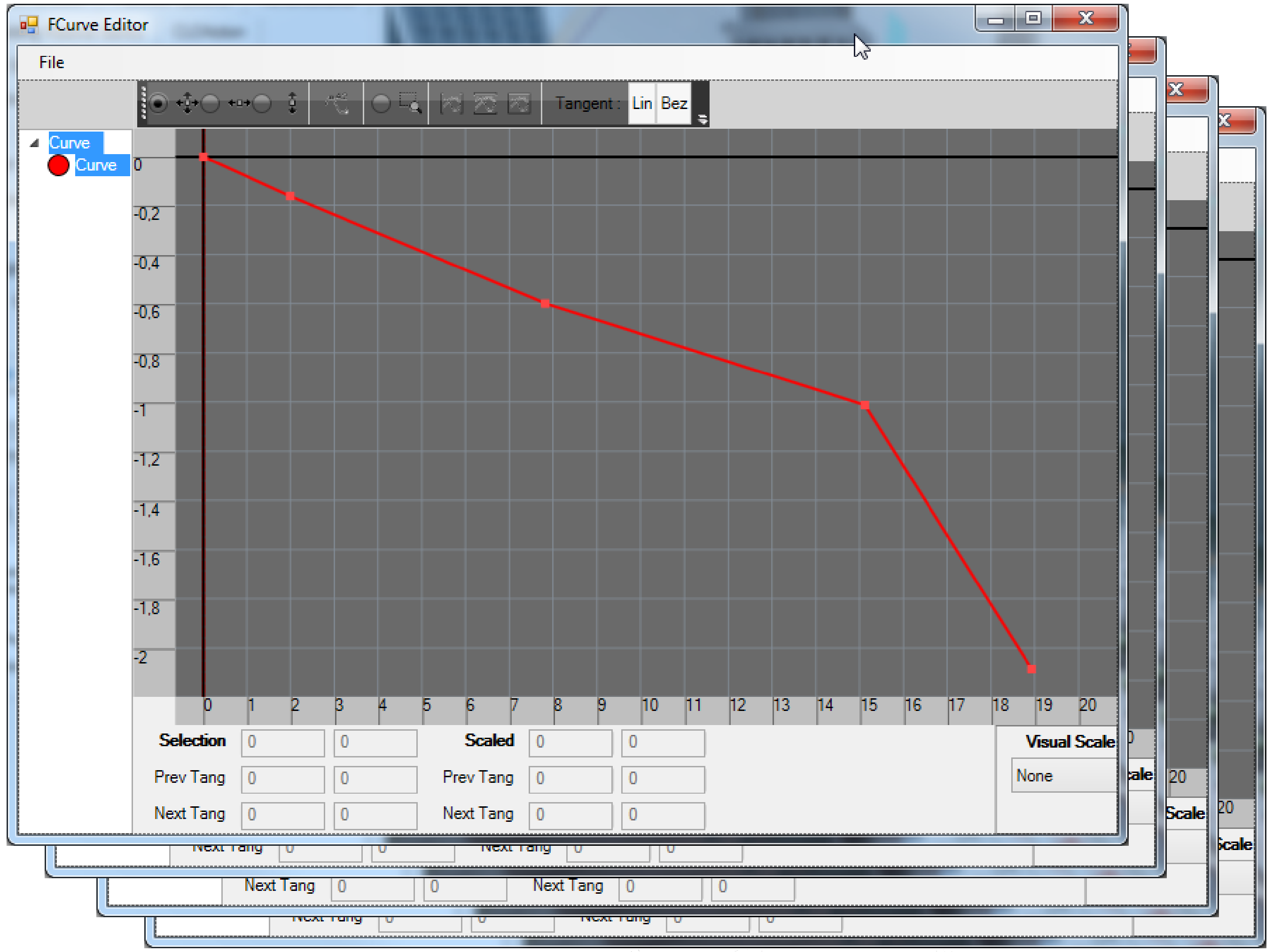
Learning from game state

- AI testing in For Honor
- **RL-based driving in Watch_Dogs 2**

Hand-tuning driving behavior

- Classical driving logic:
  PID controller

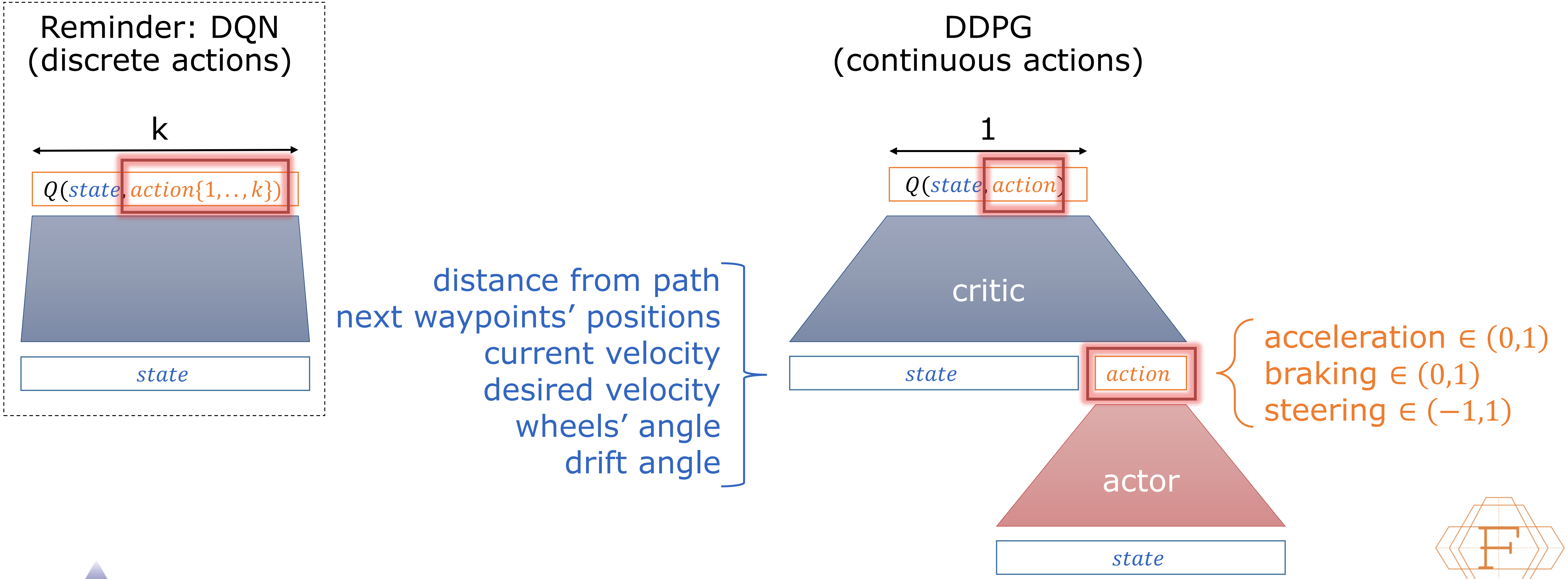
- Costly to tune across many different vehicles



[images source]

Watch_Dogs 2 as RL driving playground

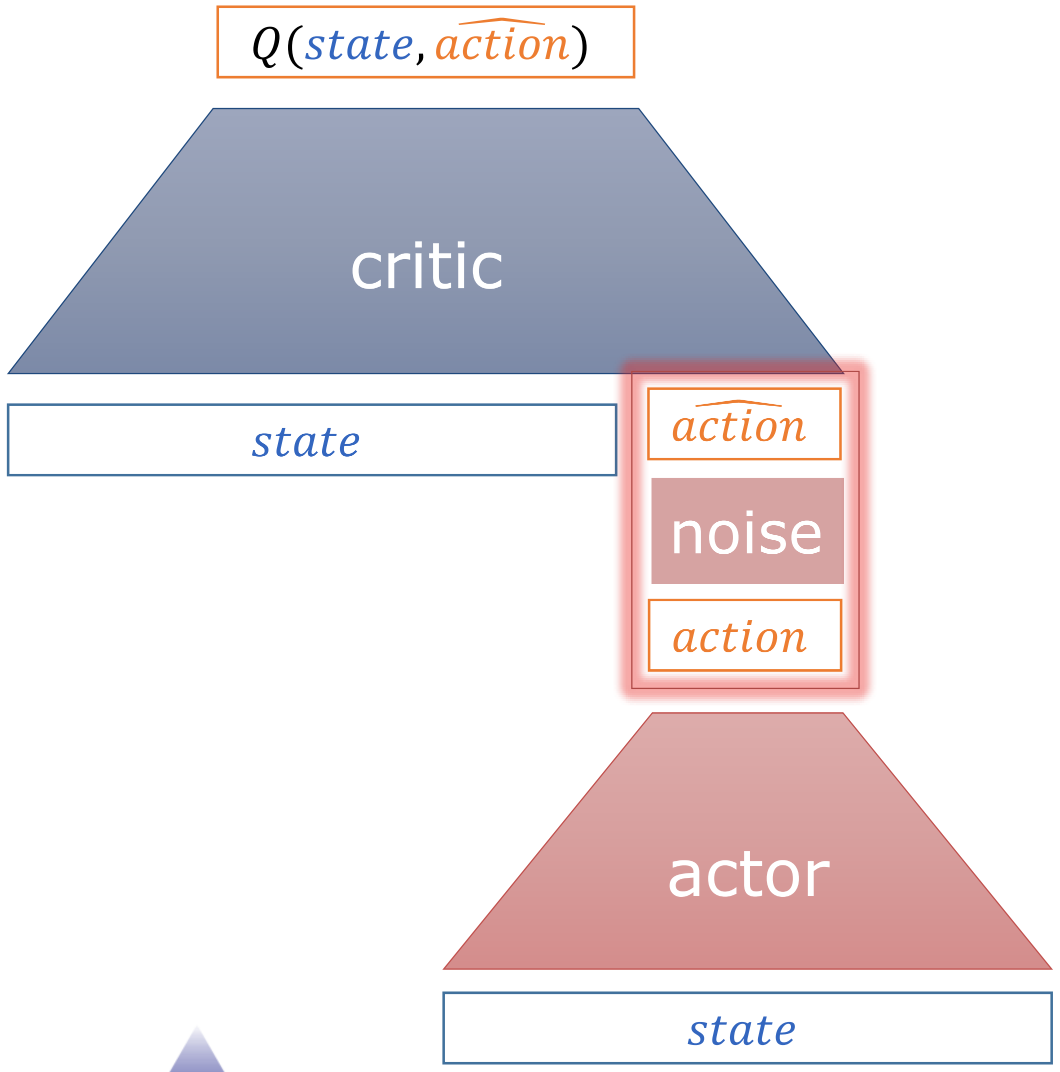
Deep Deterministic Policy Gradient



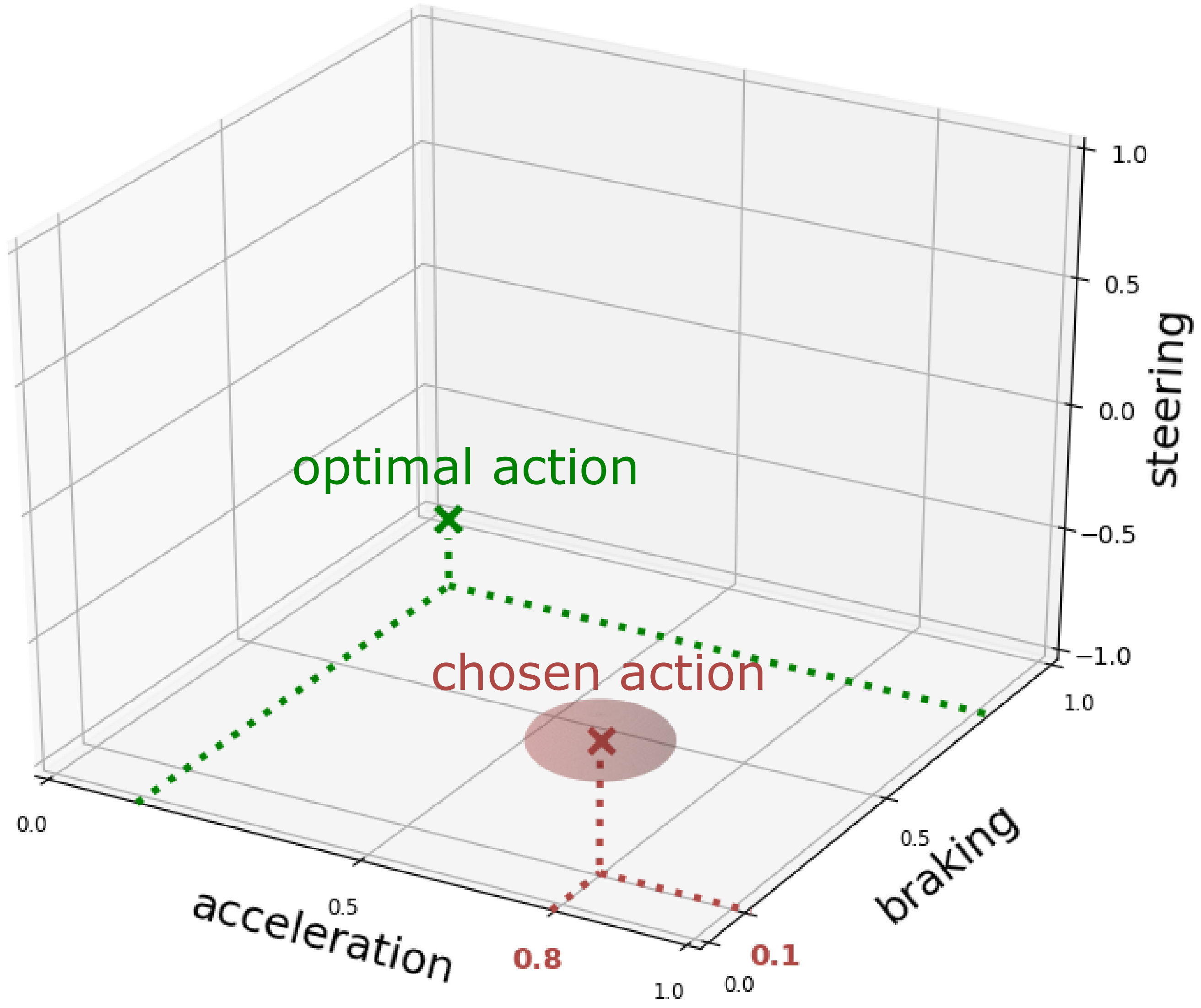
Learning to brake at high speed (or not)

Problem: local exploration

Exploration noise is added during training

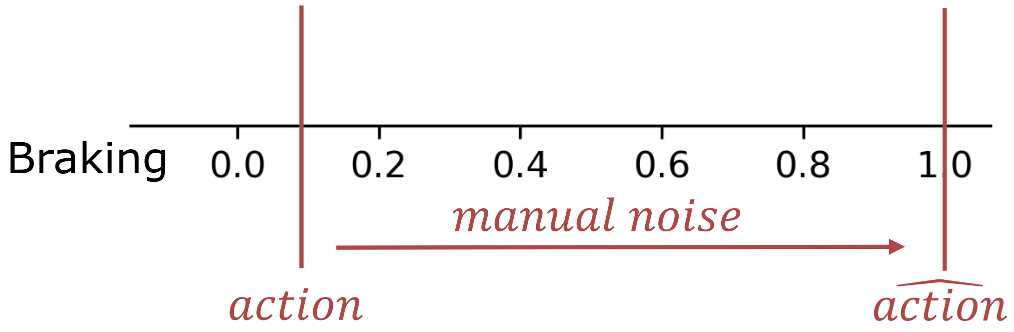


Typically: small amount of noise ➔ fails to **discover** braking benefits



Solution: expert-driven exploration

Here: randomly force the "braking" action dimension to 1 during training



➔ use domain knowledge to guide exploration

In a nutshell: learning from game state

- More efficient & flexible than from pixels…
- … but requires a bug-free training interface and may remain costly

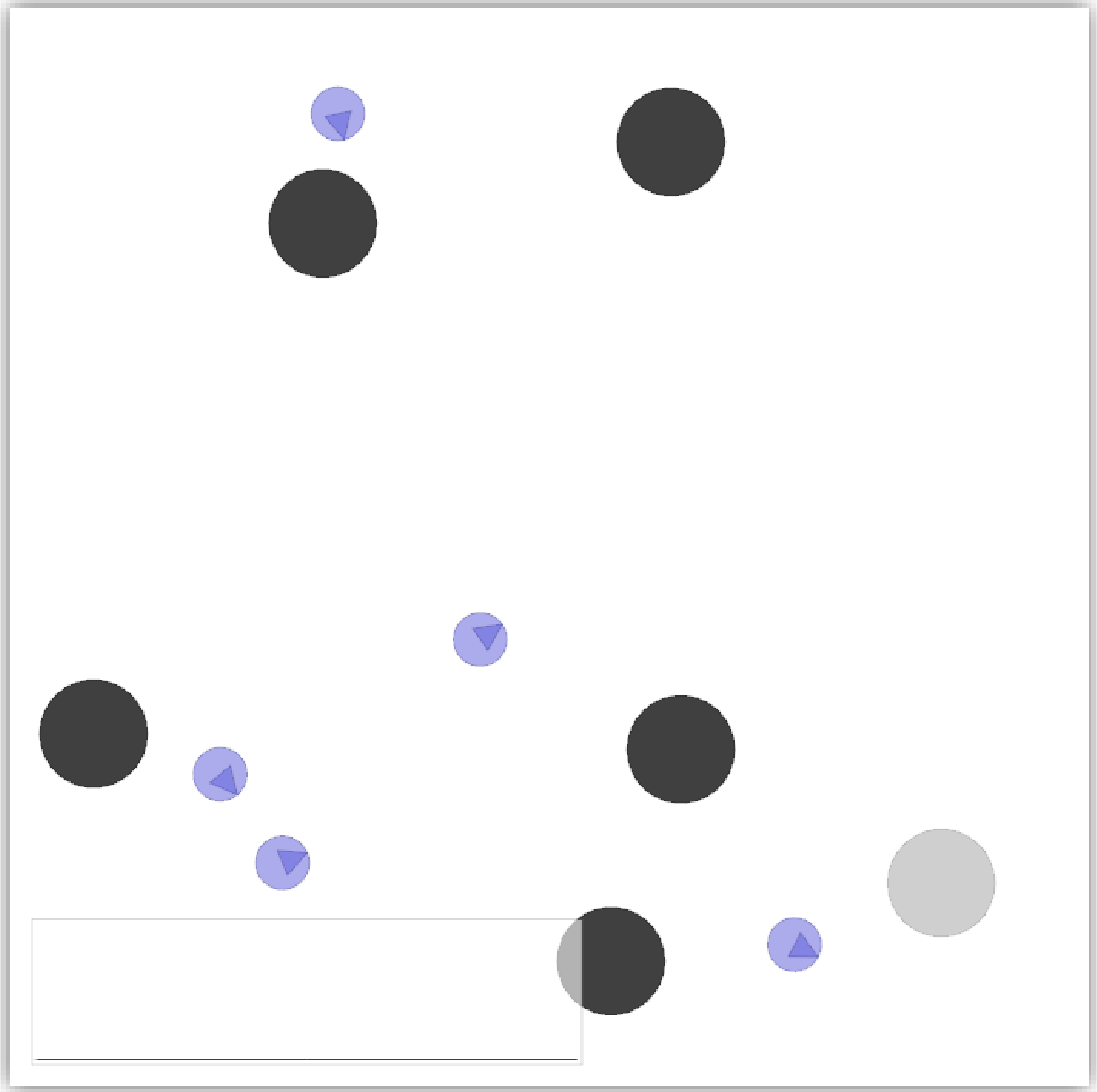
1. Reinforcement learning & games
2. Learning from pixels
3. Learning from game state
4. **Learning from simulation**
5. Epilogue

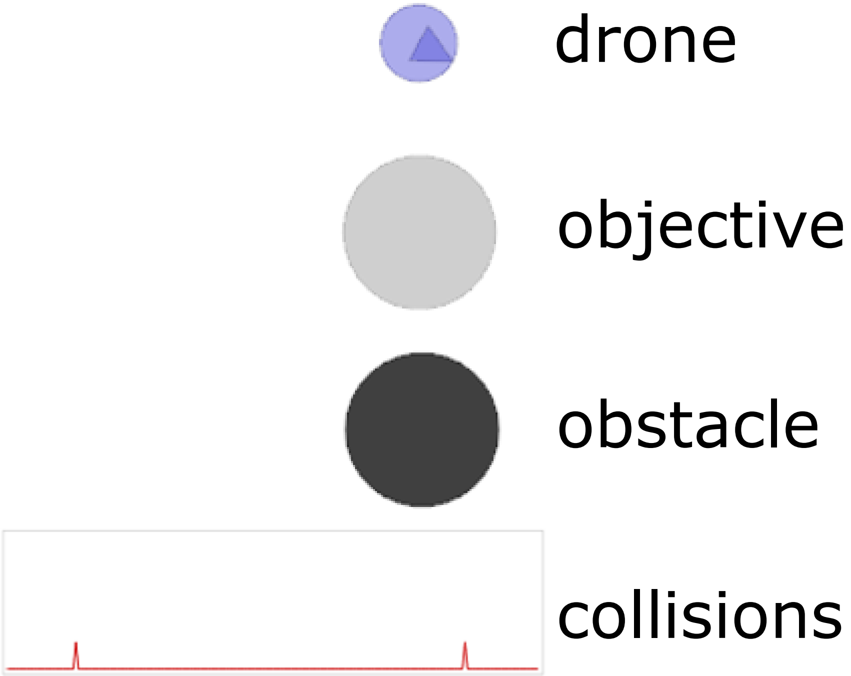
[image source]

Learning from simulation

- **Direct transfer from simulation to game**
- Prototype in simulation, re-train in game
- Pre-train in simulation, fine-tune in game

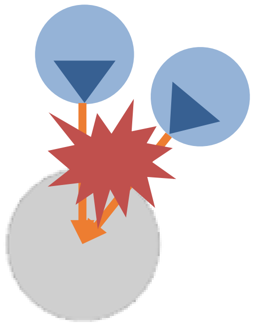
Drone Swarms

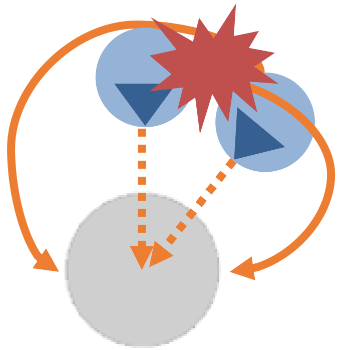




Multi-agent & non-stationarity

Each agent's environment changes as other agents learn!

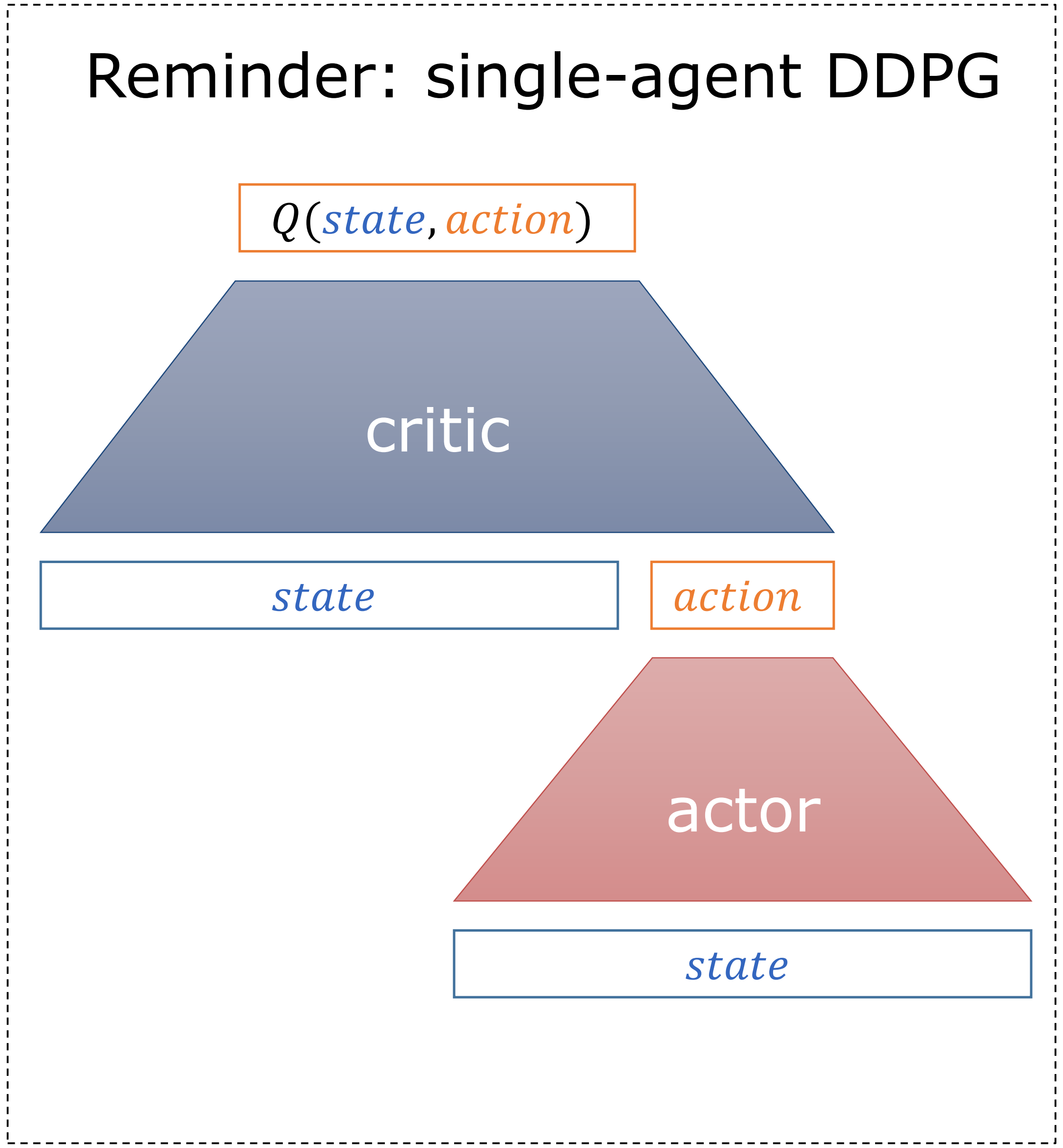


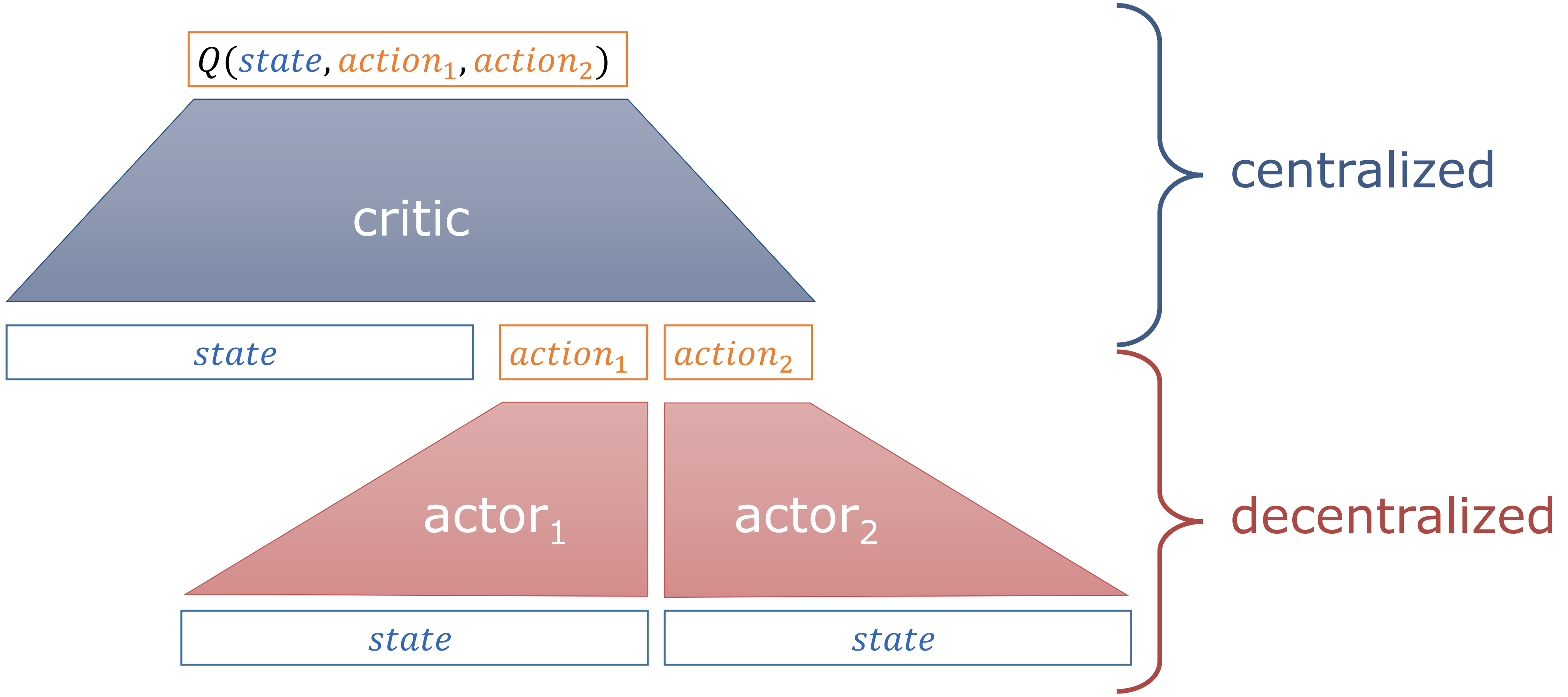


...

➔ For **some** tasks training may get unstable

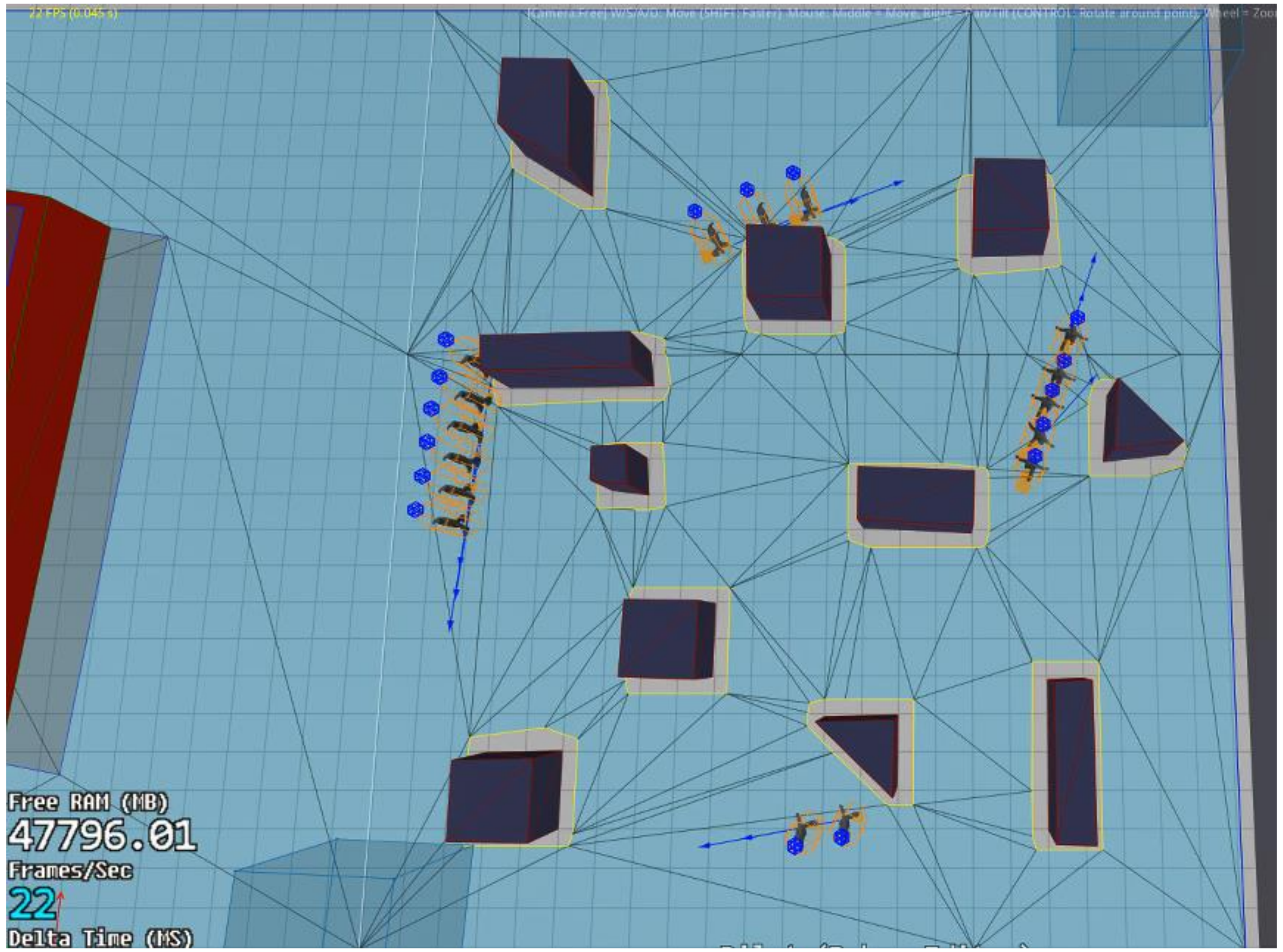
Multi-Agent DDPG



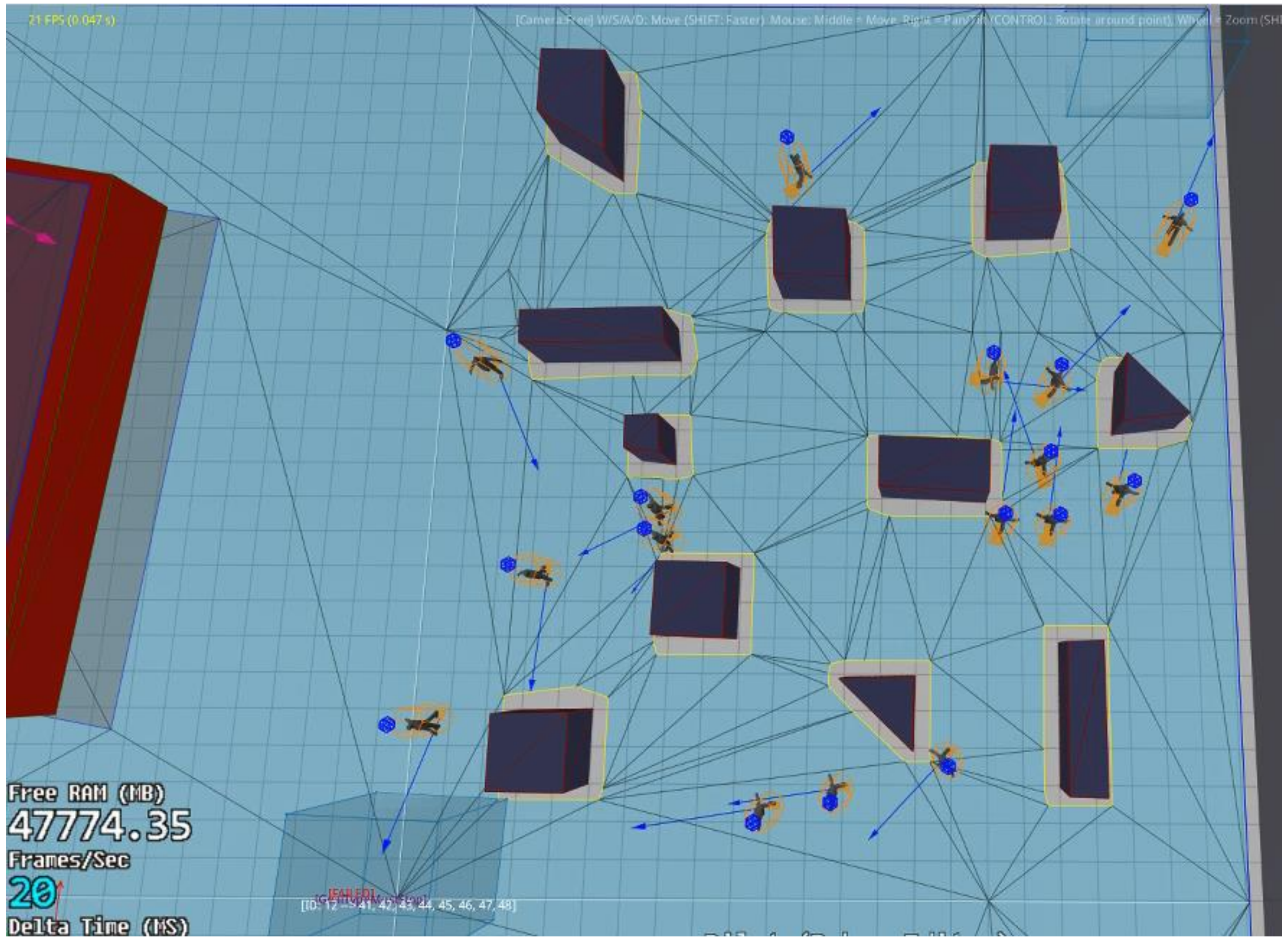


- One actor per agent: MADDPG (Lowe et al. 2017)
- One actor for all agents: "Clone-Ensemble DDPG"

Swarm in action

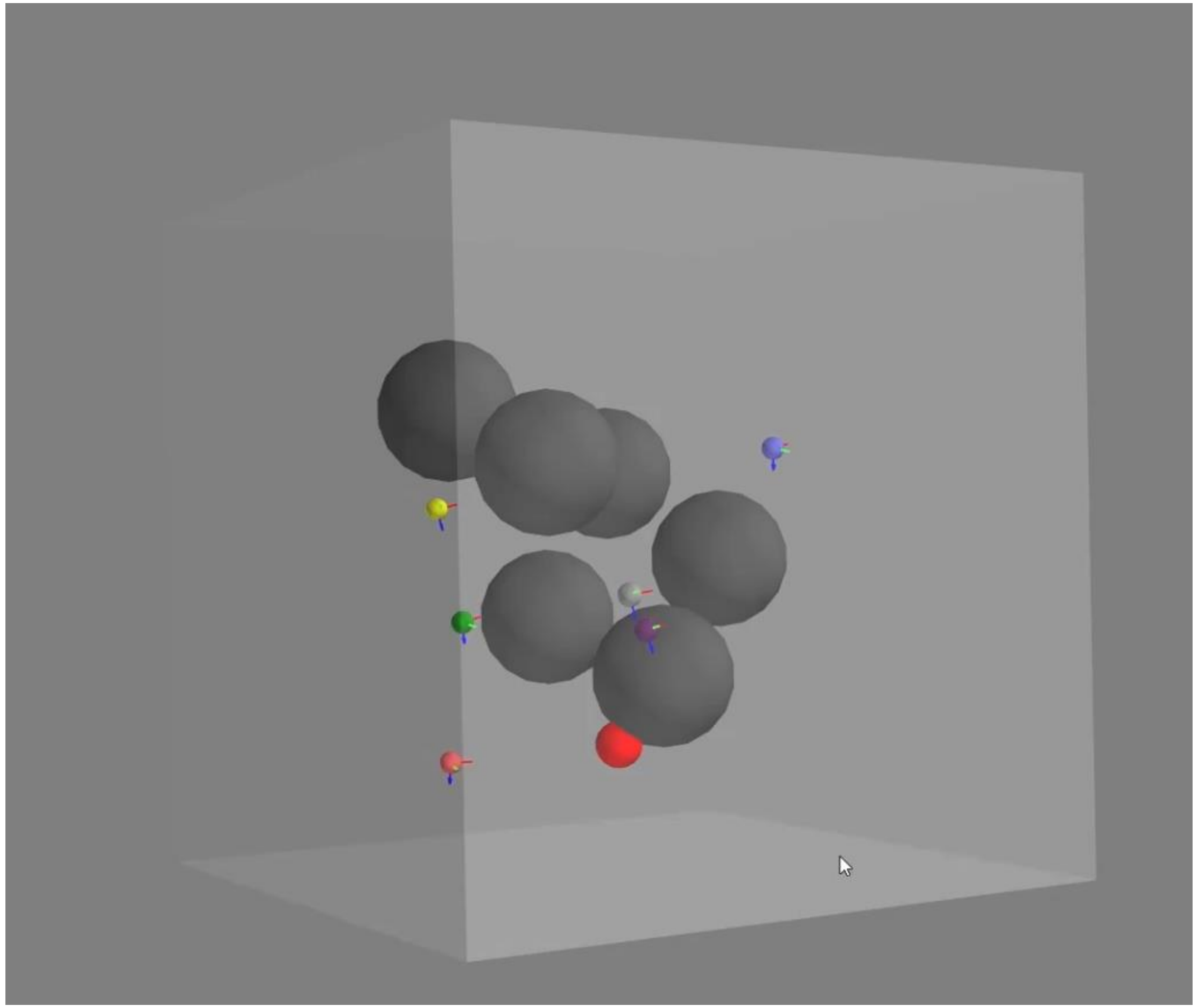


ORCA (traditional)

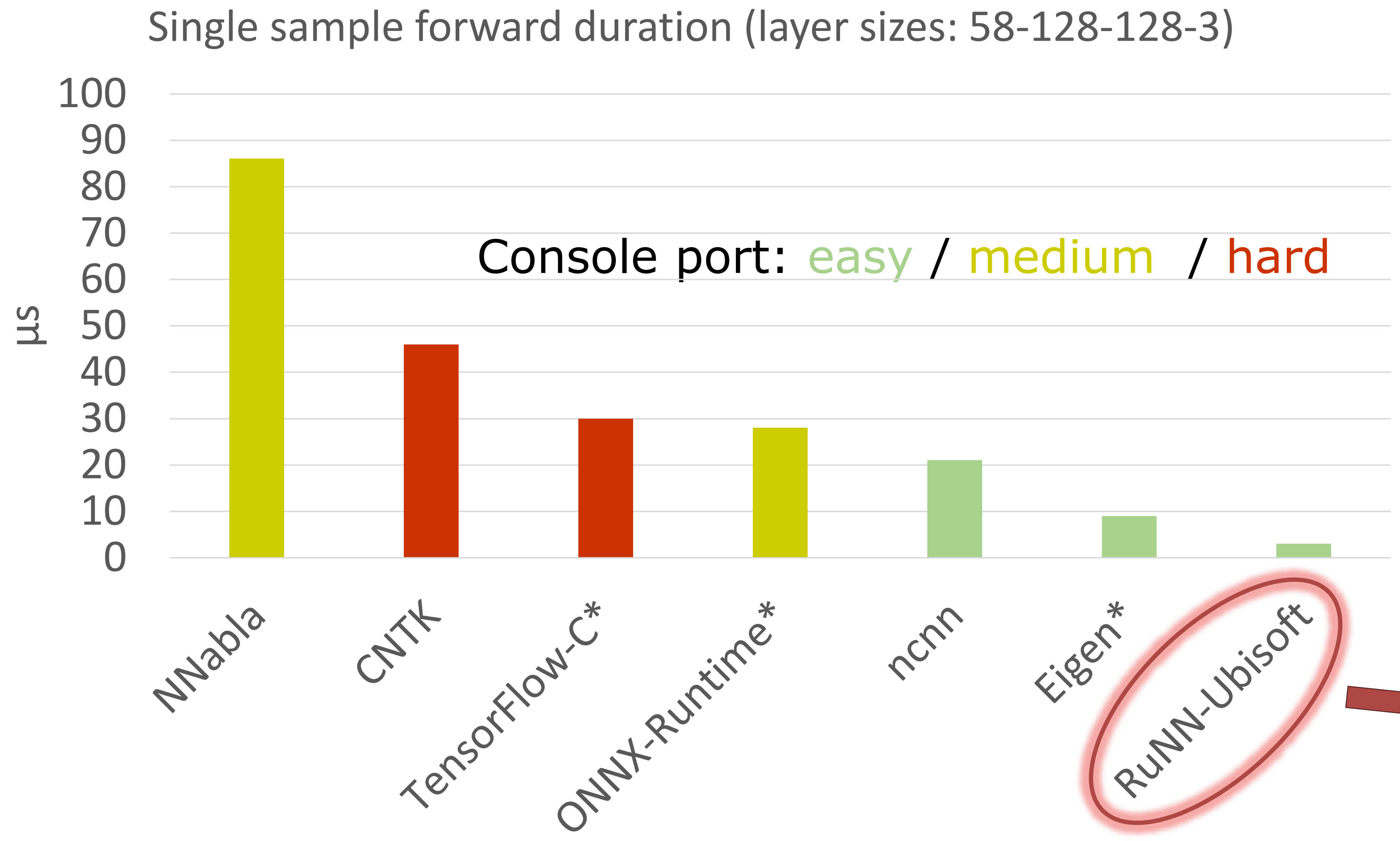


Swarm (RL-based)

From 2D to 3D



Neural Net performance (single-thread PC CPU)



Caveats:

- Small network
- No input batching

Tip: use C++ keyword __restrict

* Untuned

Learning from simulation

- Direct transfer from simulation to game
- **Prototype in simulation, re-train in game**
- Pre-train in simulation, fine-tune in game

Driving++: handling maneuvers & obstacles

- **Prototyping** in toy 2D driving environment with simple physics

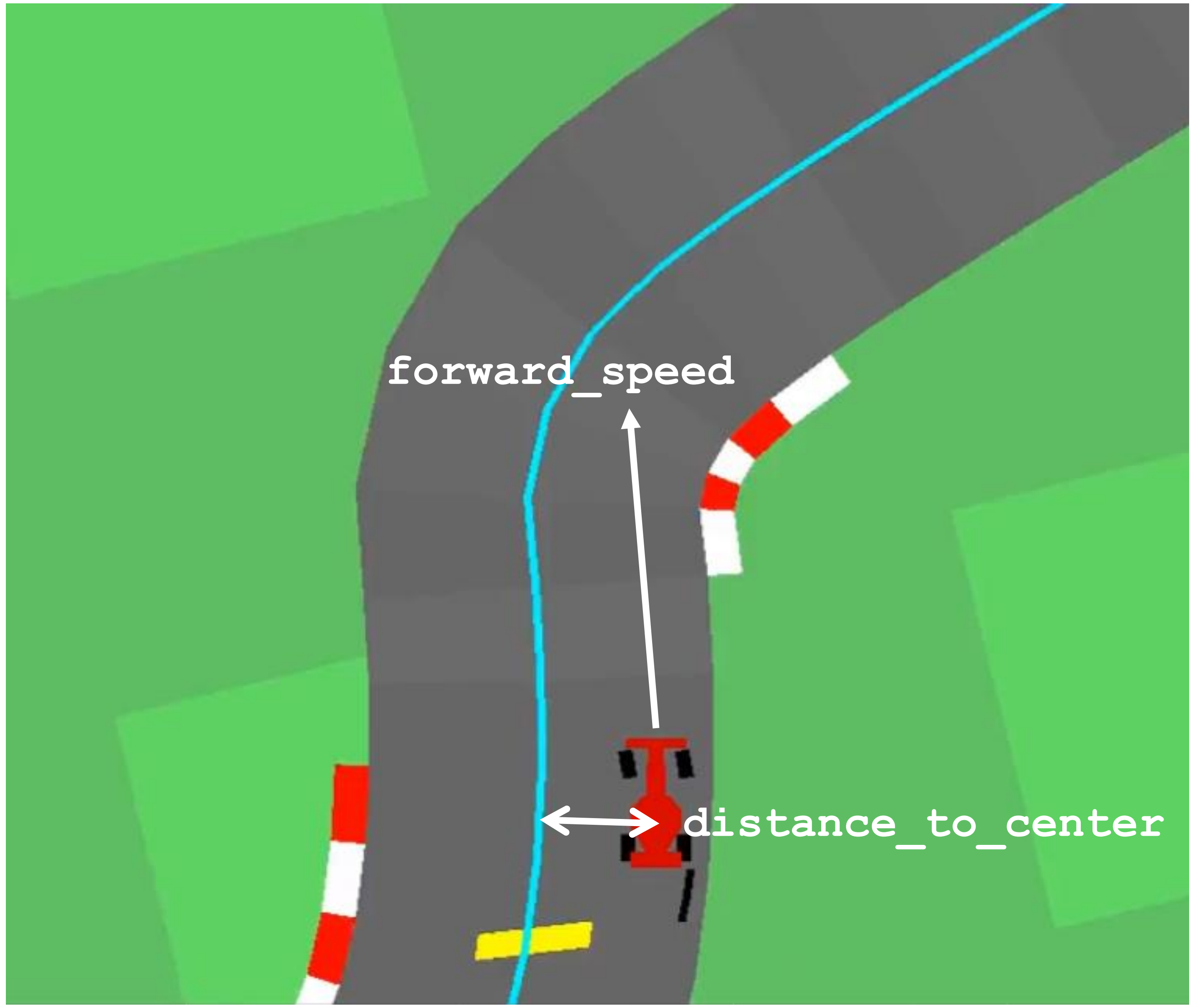
- Soft Actor-Critic algorithm for improved **exploration**

[custom version of CarRacing-v0]

Reward shaping: distance to center penalty

Reward:

`forward_speed * (0.5 - distance_to_center)`

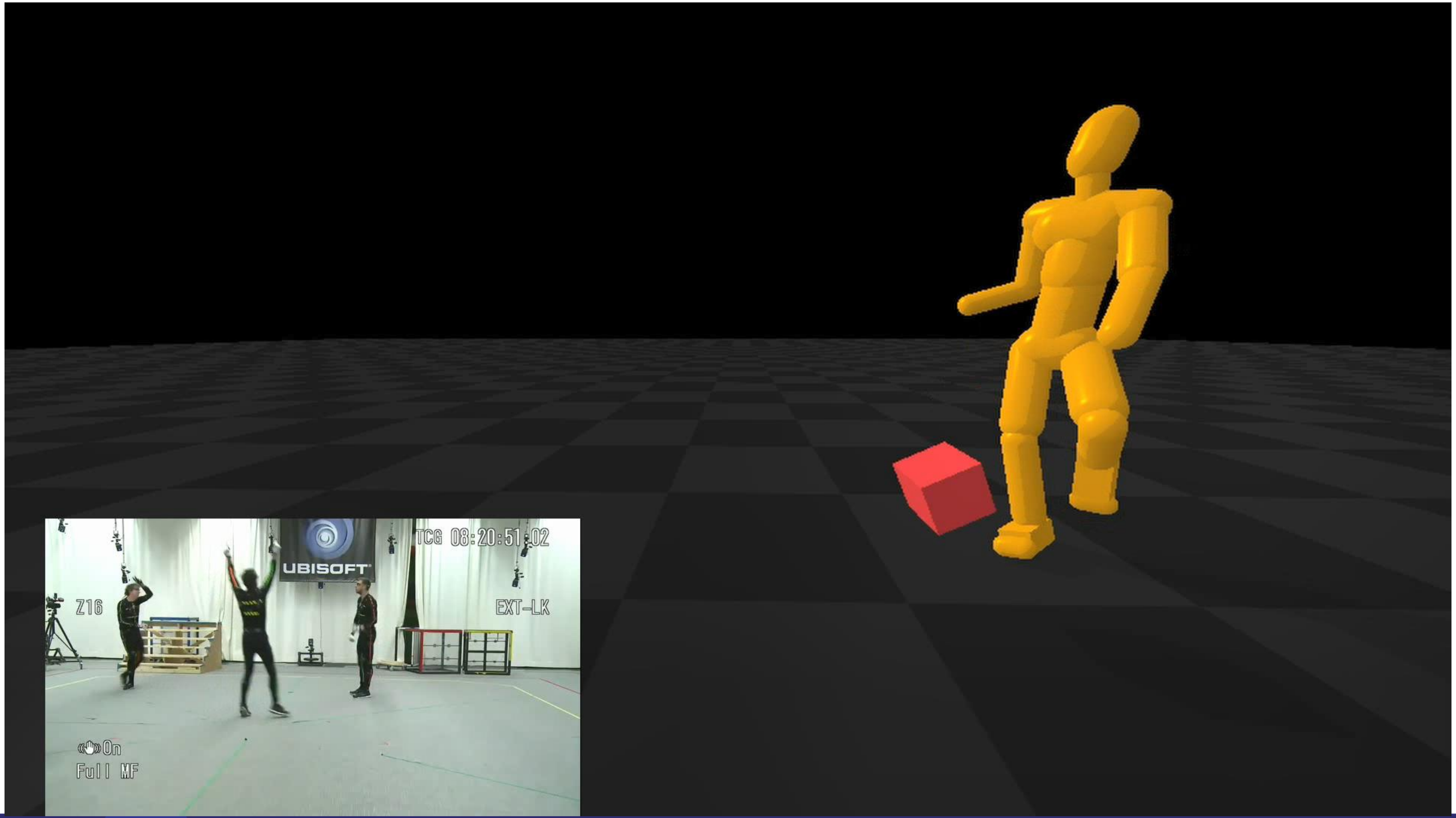


Reward shaping: distance to center penalty

Reward:

```
forward_speed * (0.5 - distance_to_center)
        -0.8 * (0.5 - 0.9) = 0.32
```

Physically simulated ragdoll from MoCap



- **Prototyping** in toy 3D environment with Bullet physics engine
- [Proximal Policy Optimization]() algorithm with a reward for **matching** the desired pose

Learning from simulation

- Direct transfer from simulation to game
- Prototype in simulation, re-train in game
- **Pre-train in simulation, fine-tune in game**

1. Reinforcement learning & games
2. Learning from pixels
3. Learning from game state
4. Learning from simulation
5. **Epilogue**

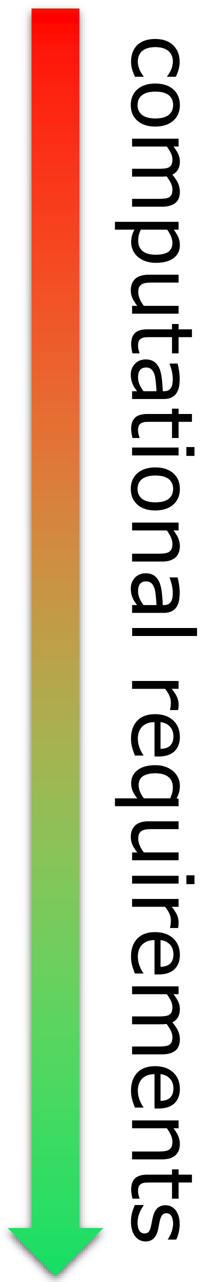
[image source]

In a nutshell: personal advice

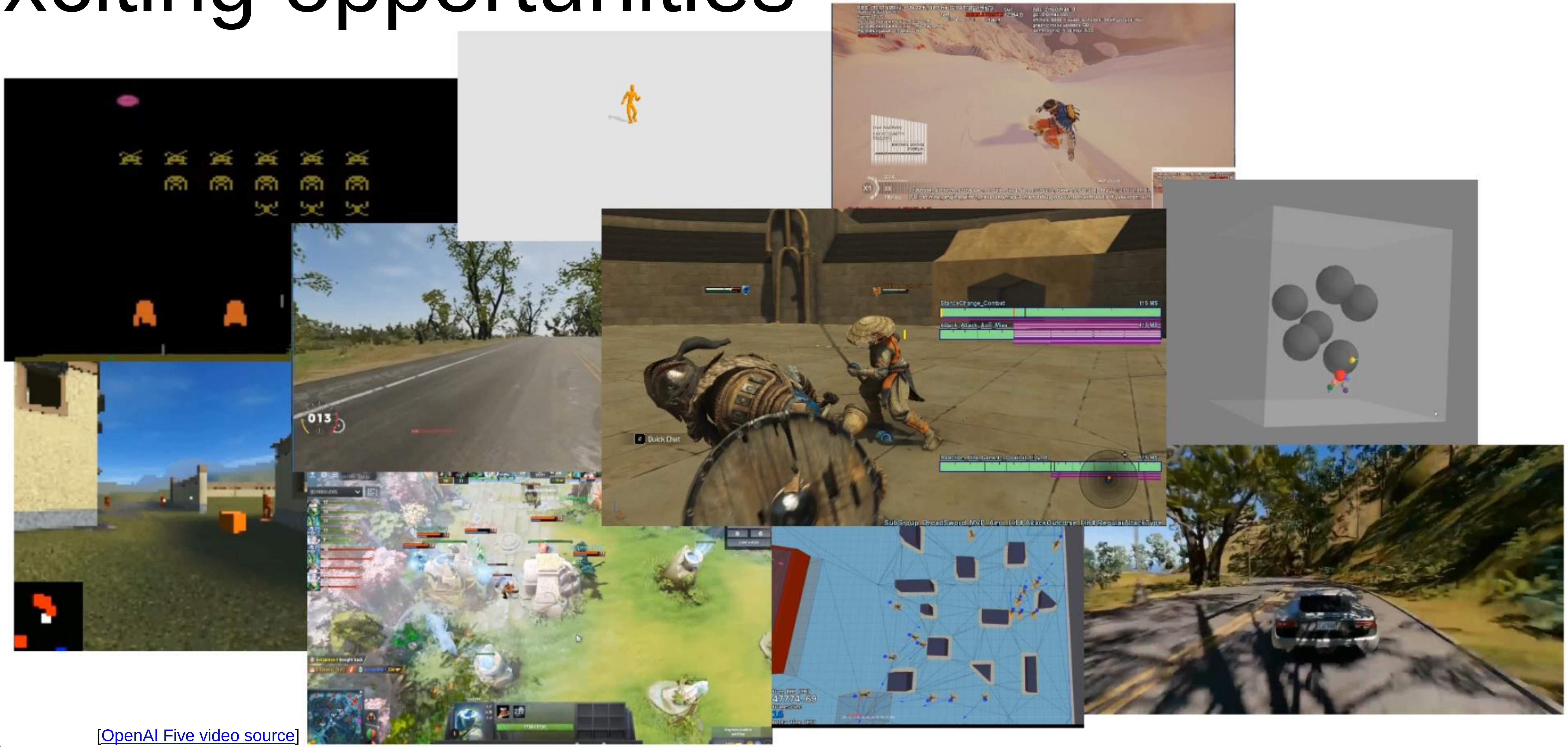
- Learning from **pixels**
  - **Avoid** it if you can

- Learning from game **state**
  - Ensure the RL loop is **bug-free & efficient**

- Learning from **simulation**
  - **Trade-off** fidelity vs computational+implementation cost

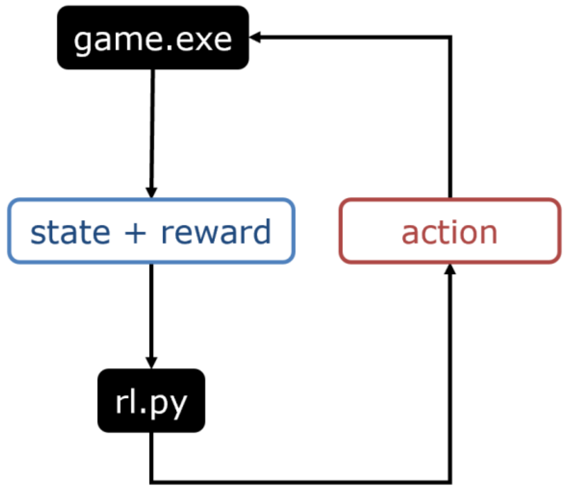


Exciting opportunities



[OpenAI Five video source]

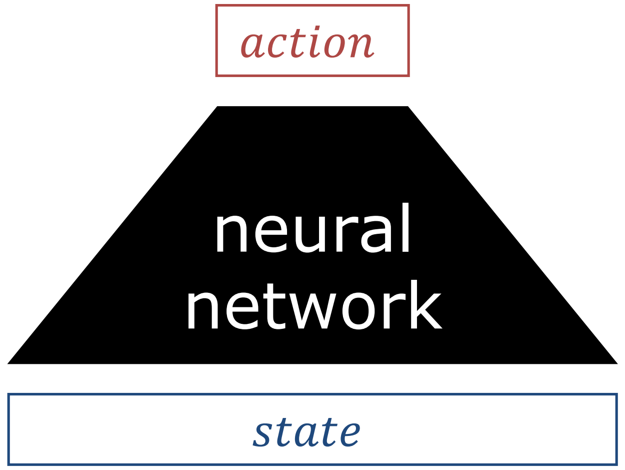
Production challenges



- Building an “RL-friendly” engine

- Managing heavy computations (task & algo-dependent)

- Controlling RL agents



Reward shaping is tricky: *[Specification gaming examples in AI](#)* *(Krakovna, 2018)*

The End

Olivier Delalleau
http://laforge.ubisoft.com
laforge@ubisoft.com

Bonus content: keep reading for more tips / references / examples


ARE YOU READY TO
CREATE THE UNKNOWN?
jobs.ubisoft.com
LEARN MORE AT OUR BOOTH!
WEST HALL - FLOOR 2
UBISOFT

Bonus content

[photo by Rich Grundy]

RL tips for game developers

- Only use RL when it's the right tool for your task (consider also decision trees / search / planning / imitation / evolution / …)
- Avoid learning from pixels if you can (and if you can't, try and help your agent with custom object recognition)
- Use human data to speed-up learning if available
- RNNs can help but are trickier to work with, try to fake memory through feature engineering first
- (Ab)use domain knowledge for feature engineering, reward shaping, network architecture and exploration strategy
- Estimate computational requirements before investing too much effort
- Don't rush the game RL loop implementation (state-action-reward) – bugs will haunt you later
- Define benchmarks to compare algorithms – while ensuring statistical significance
- Mask invalid actions
- Fake human-like reaction time when it matters (but ask yourself whether some state features should *not* be delayed)
- Practice trumps theory: try simple techniques first, even in settings where you suspect they may not work
- Simulate your game (even imperfectly) if it is slow and / or complex
- Don't aim straight for a generic / efficient / cross-platform / perfect RL framework for all your games – hacks get things done
- Re-use open source implementations of established algorithms, understand them and customize them
- Ideal RL-friendly engine should (easily) allow:
    - Two-way communication with Python (e.g. through sockets) – with events, callbacks & RPC
    - Parallelization (multiple agents within a single game instance / across multiple instances on one computer / across multiple computers)
    - Efficient execution of core gameplay code without the "cosmetics" (e.g. without graphics, vfx, animations, …)
    - Ability to run neural networks directly in-engine (also useful during training to properly synchronize agent decisions with the game update loop)
    - Direct access to game data from Python
    - Save / reset state
    - Replay
    - Player data collection (same state-action-reward format as RL agent)
    - Linux compatibility

RL pointers – My top 3's

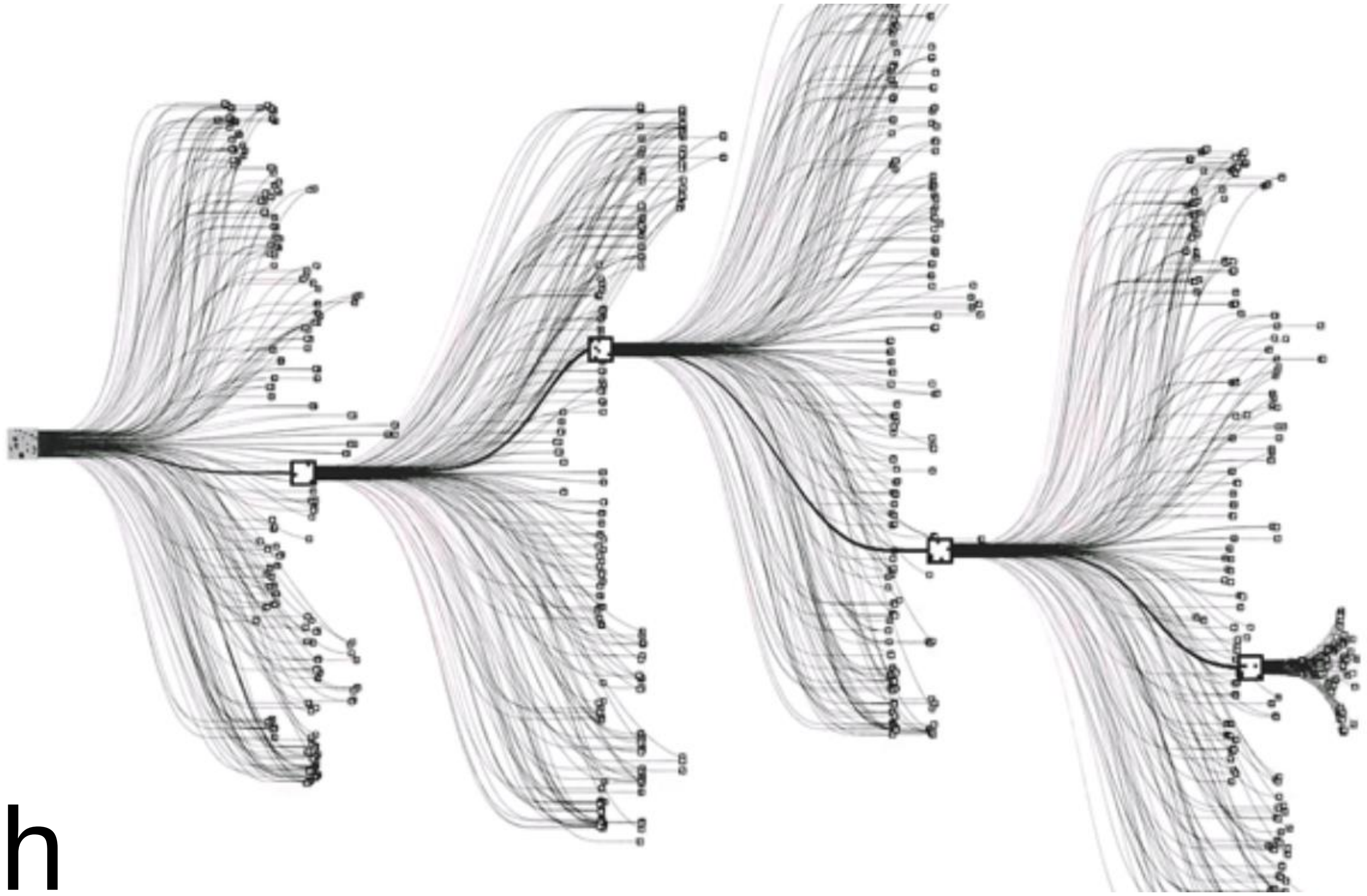
- Theory
  - [Deep RL Bootcamp]
  - [Reinforcement Learning: An Introduction]
  - [David Silver's UCL Course on RL]
- Hands-on
  - [OpenAI Spinning Up in Deep RL]
  - [Simple Reinforcement Learning with Tensorflow]
  - [A Free course in Deep Reinforcement Learning from beginner to expert]
- Software
  - [Stable Baselines]
  - [RLgraph]
  - [RLlib (linux-only ☹)]

Warning!

The following slides do not tell a coherent story!
Consider them as “deleted scenes” on a DVD ☺

Going further…

- **Learning from search**
  - Ex: [AlphaGo [Zero]]
  - Combines RL with Monte-Carlo Tree Search
  - Costly!



[image source]

> What was really heartbreaking was that we could see the improvement that the network was making. We could see the improvement over time. But the rate of improvement was just too slow for the amount of money we were spending. It was a very difficult decision, but we've decided that

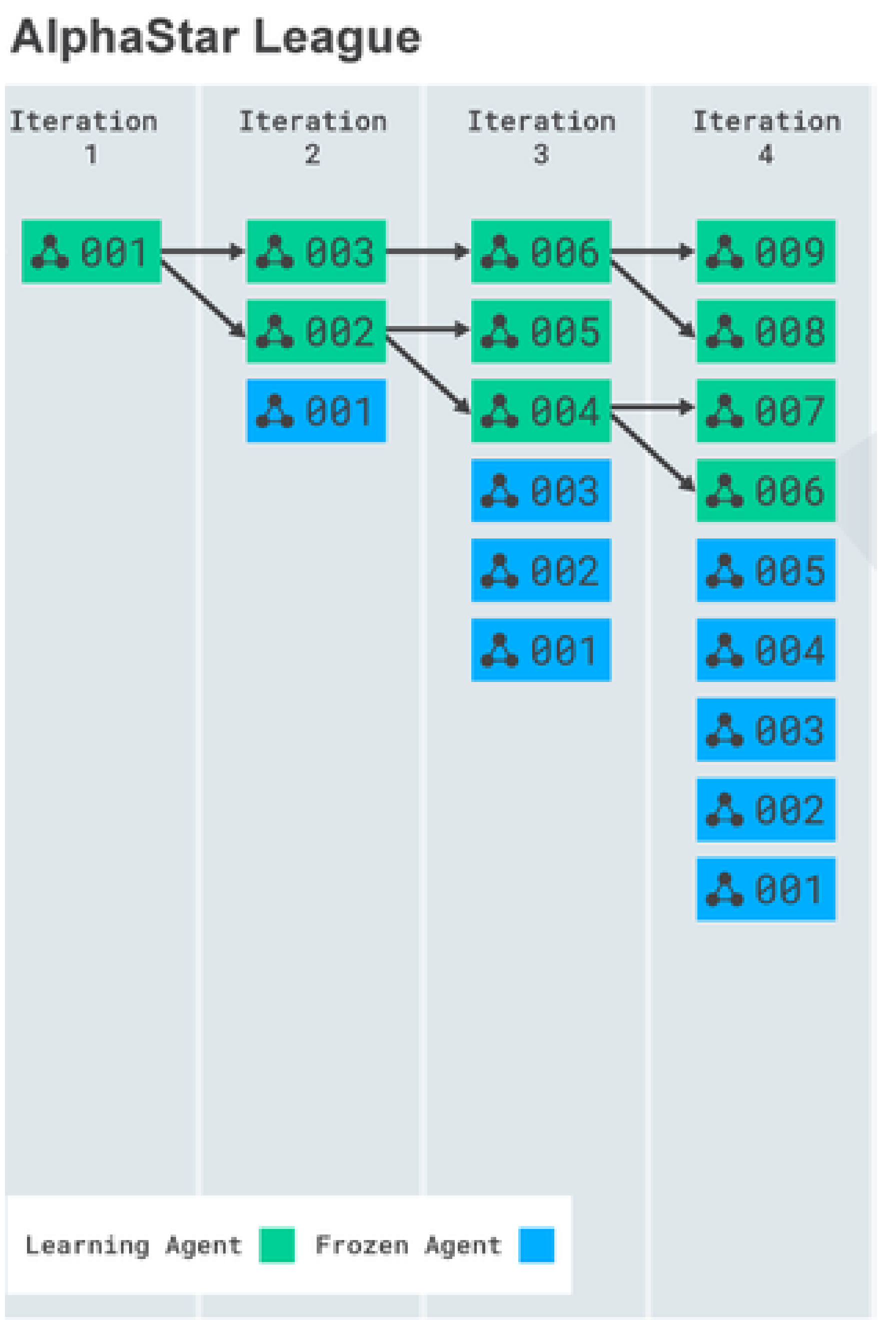
Designer Diary: The Search for AlphaMystica

Going further…

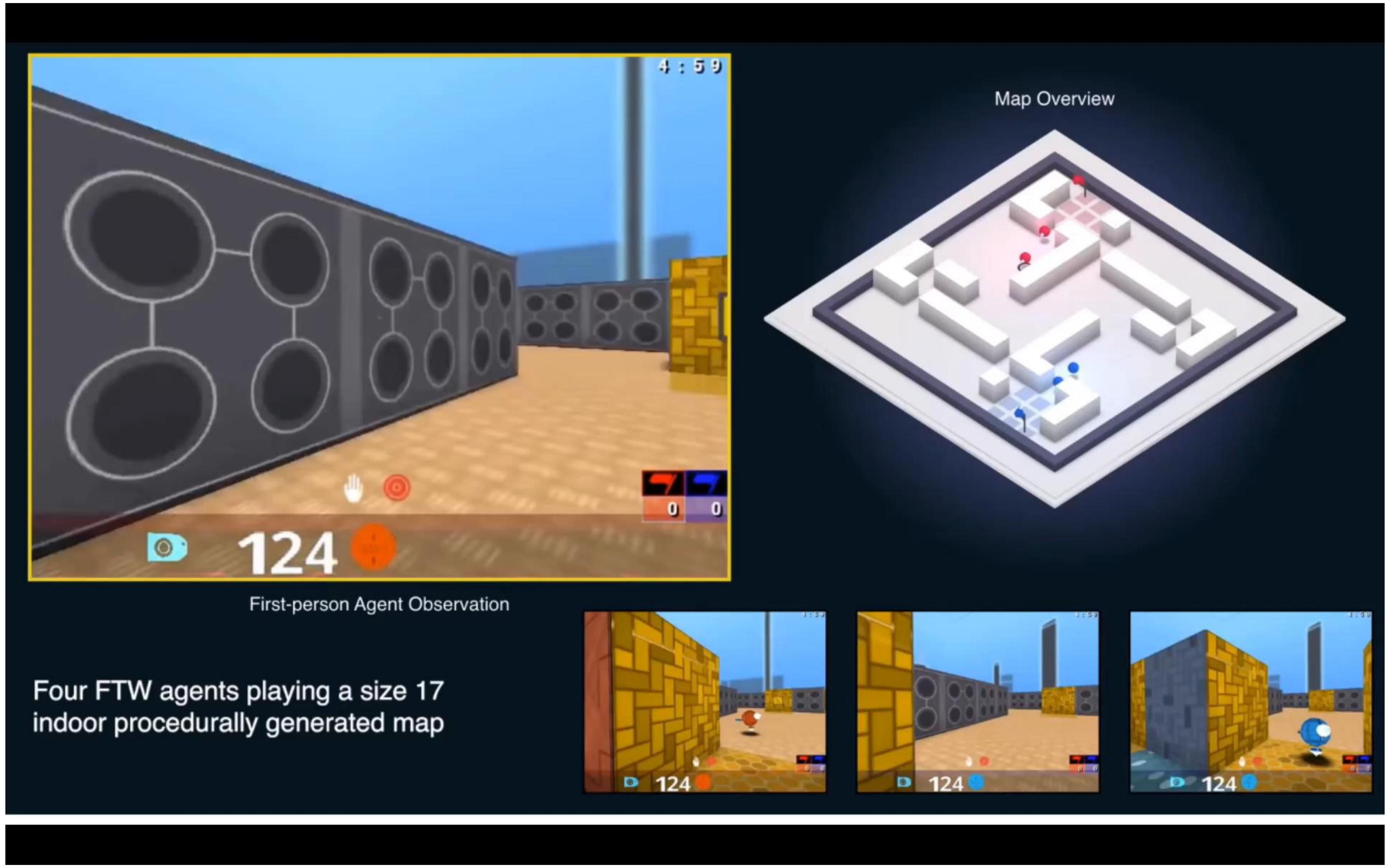
- Learning from search
- **Learning from player data**
  - Ex: [AlphaStar]
  - Combines RL with imitation learning from replays

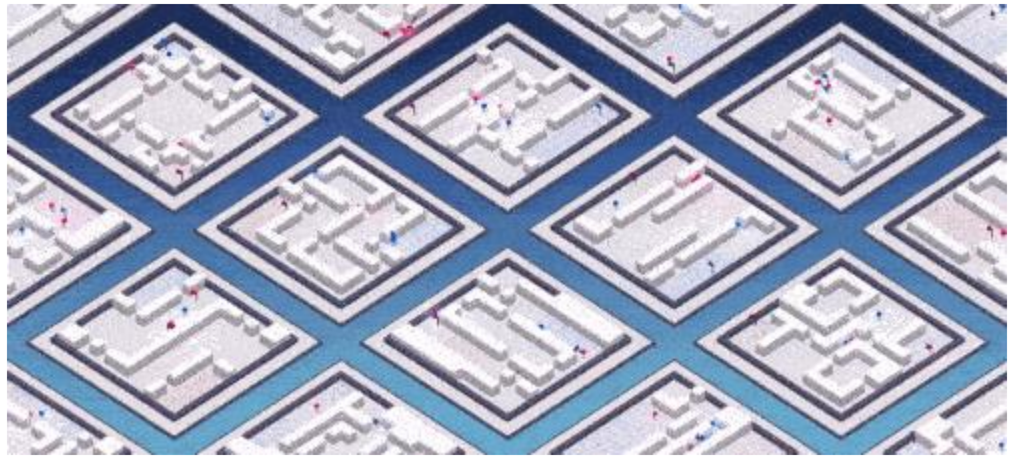
Going further…

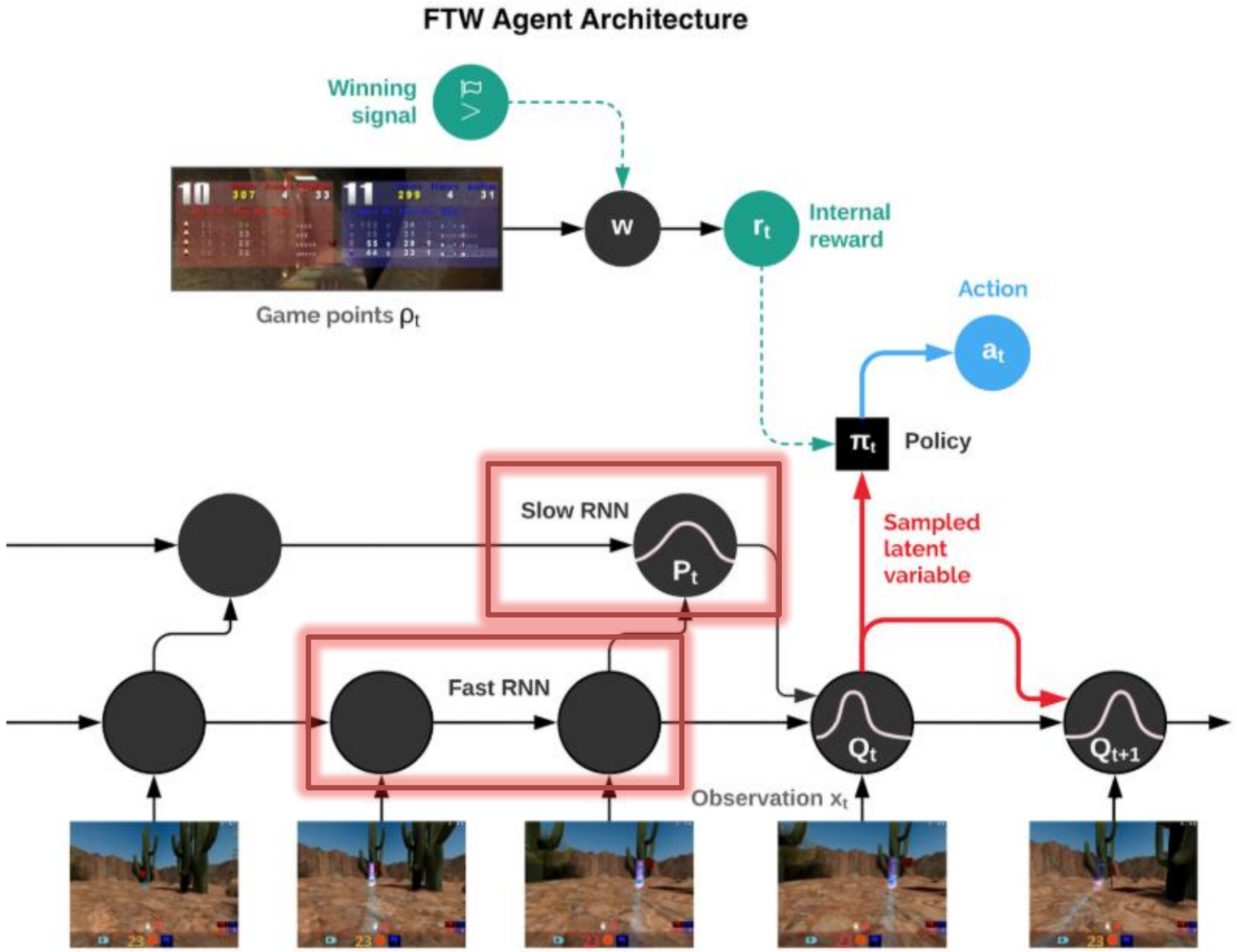
- Learning from search
- Learning from player data
- **Learning from evolution**
  - Ex: AlphaStar (again)
  - Combines RL with evolution of a population of agents to obtain a variety of behaviors



DeepMind – Quake CTF

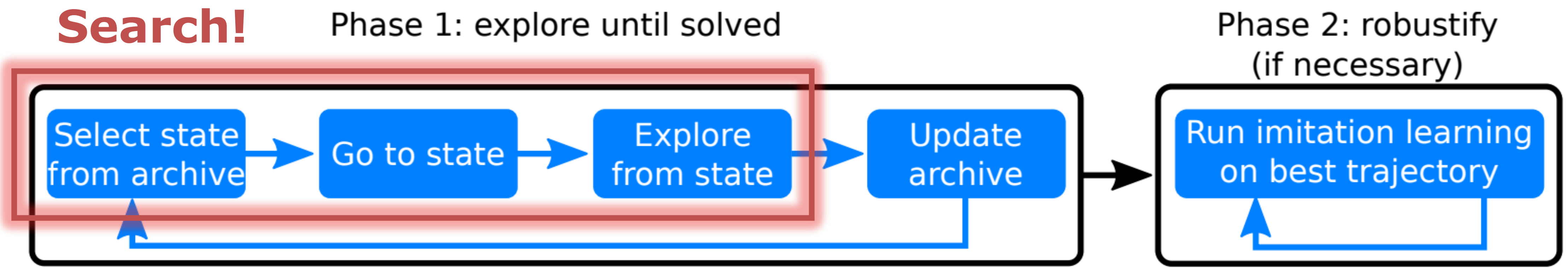


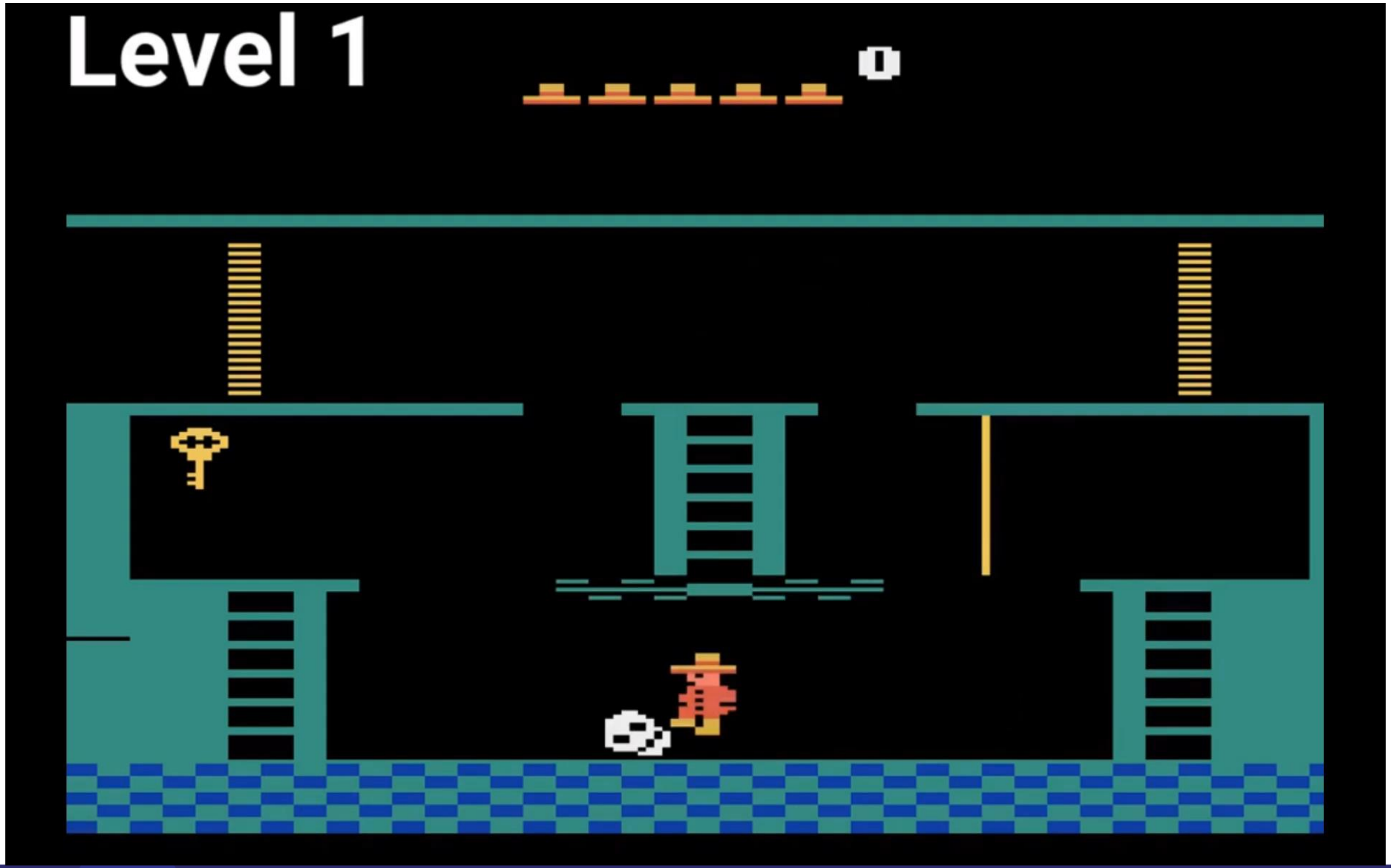


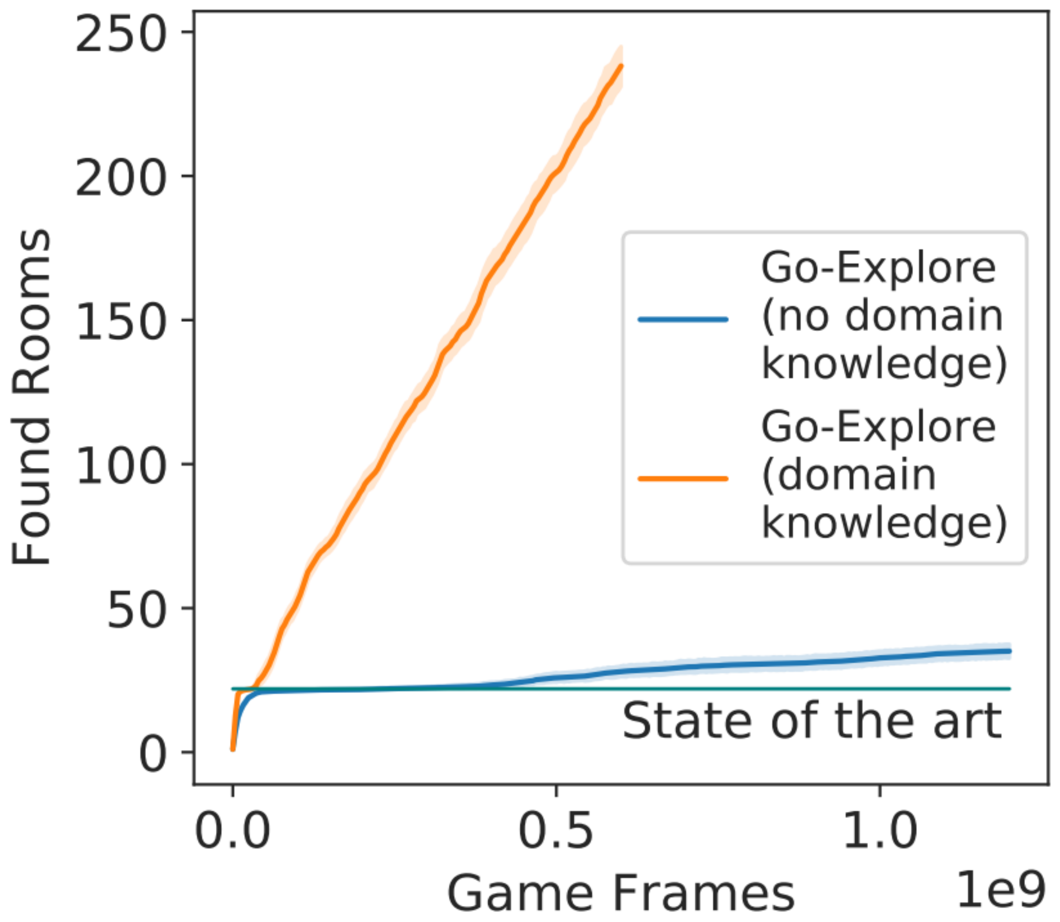


[source]

Go-Explore

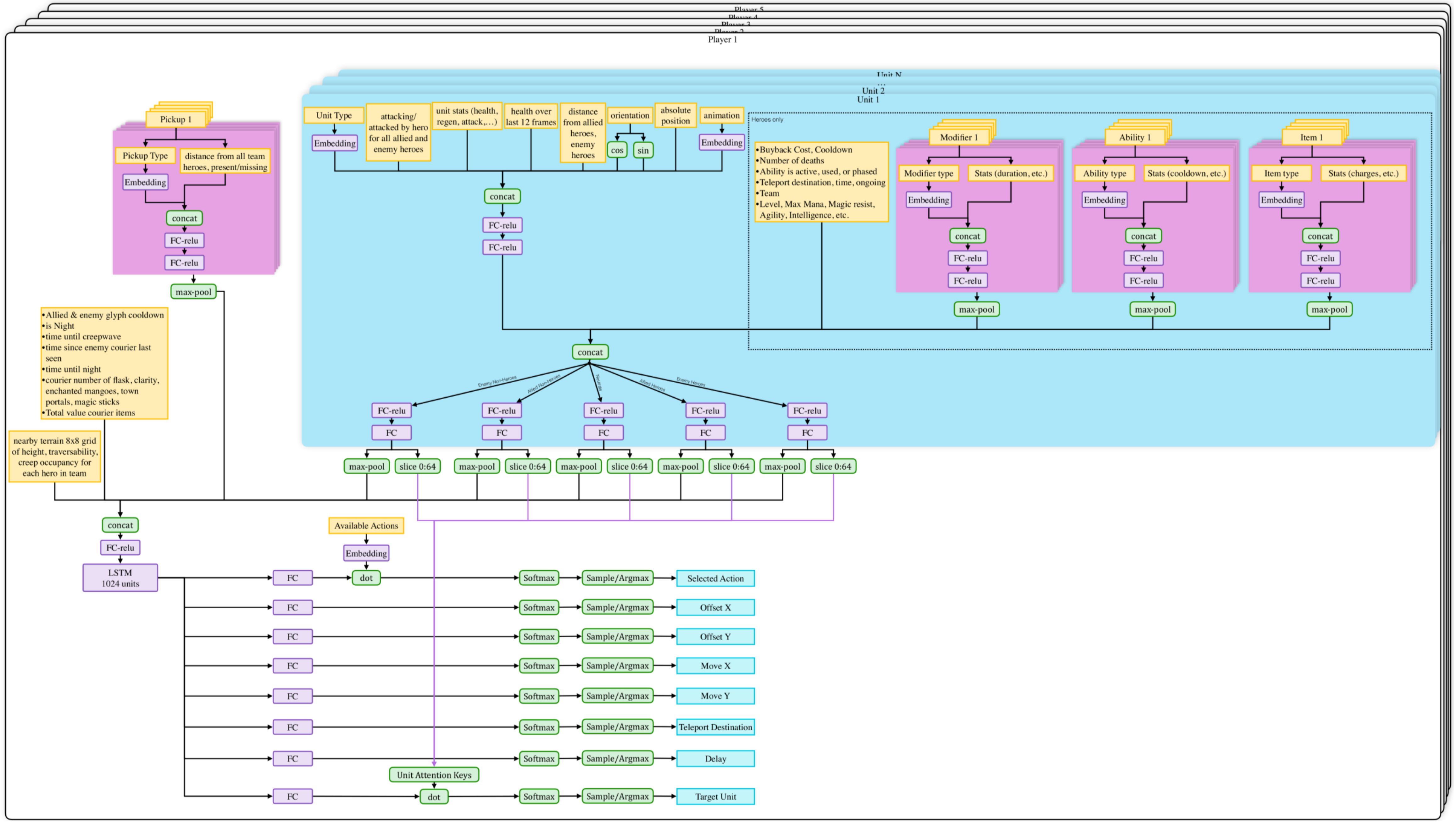




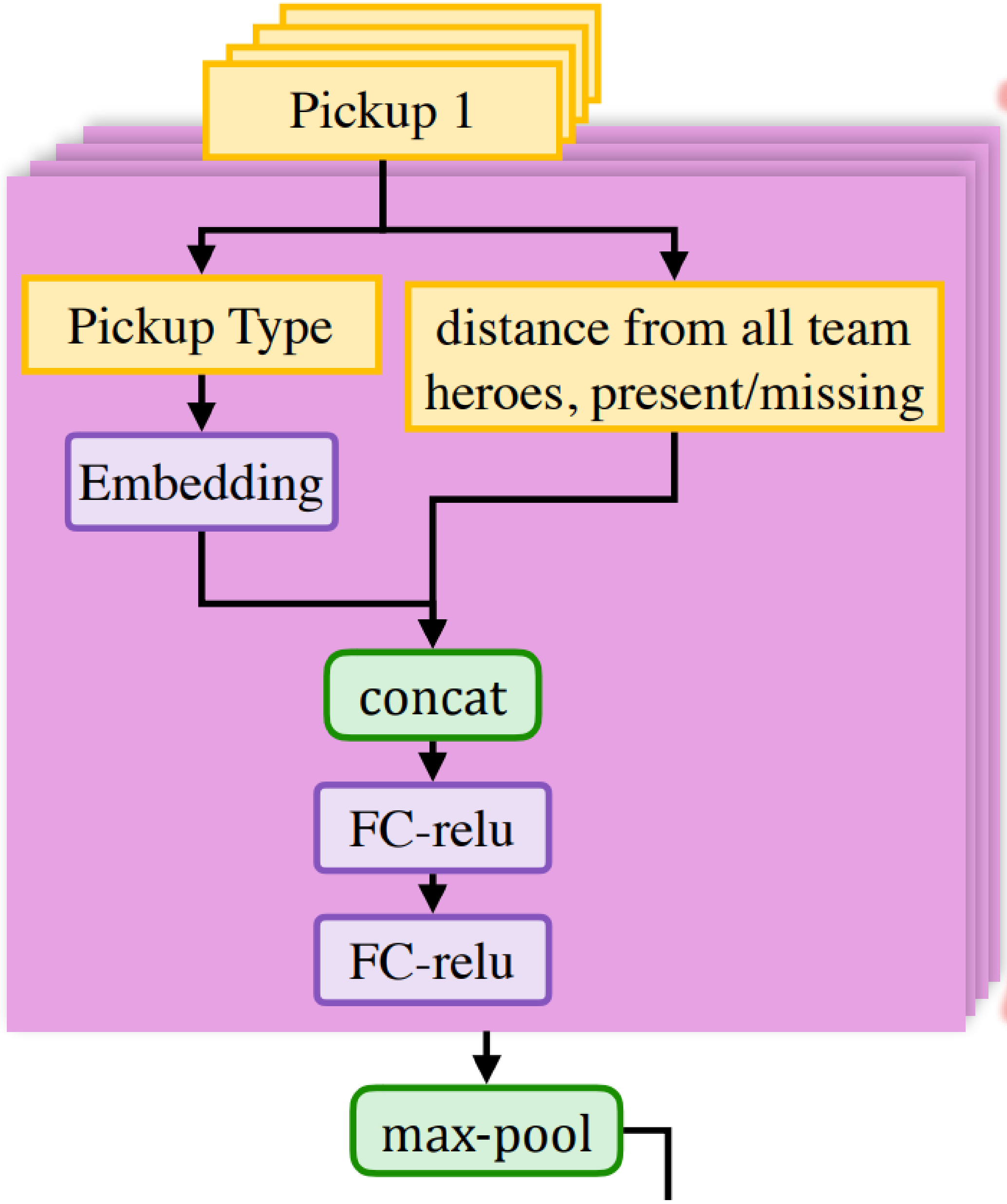


[source]

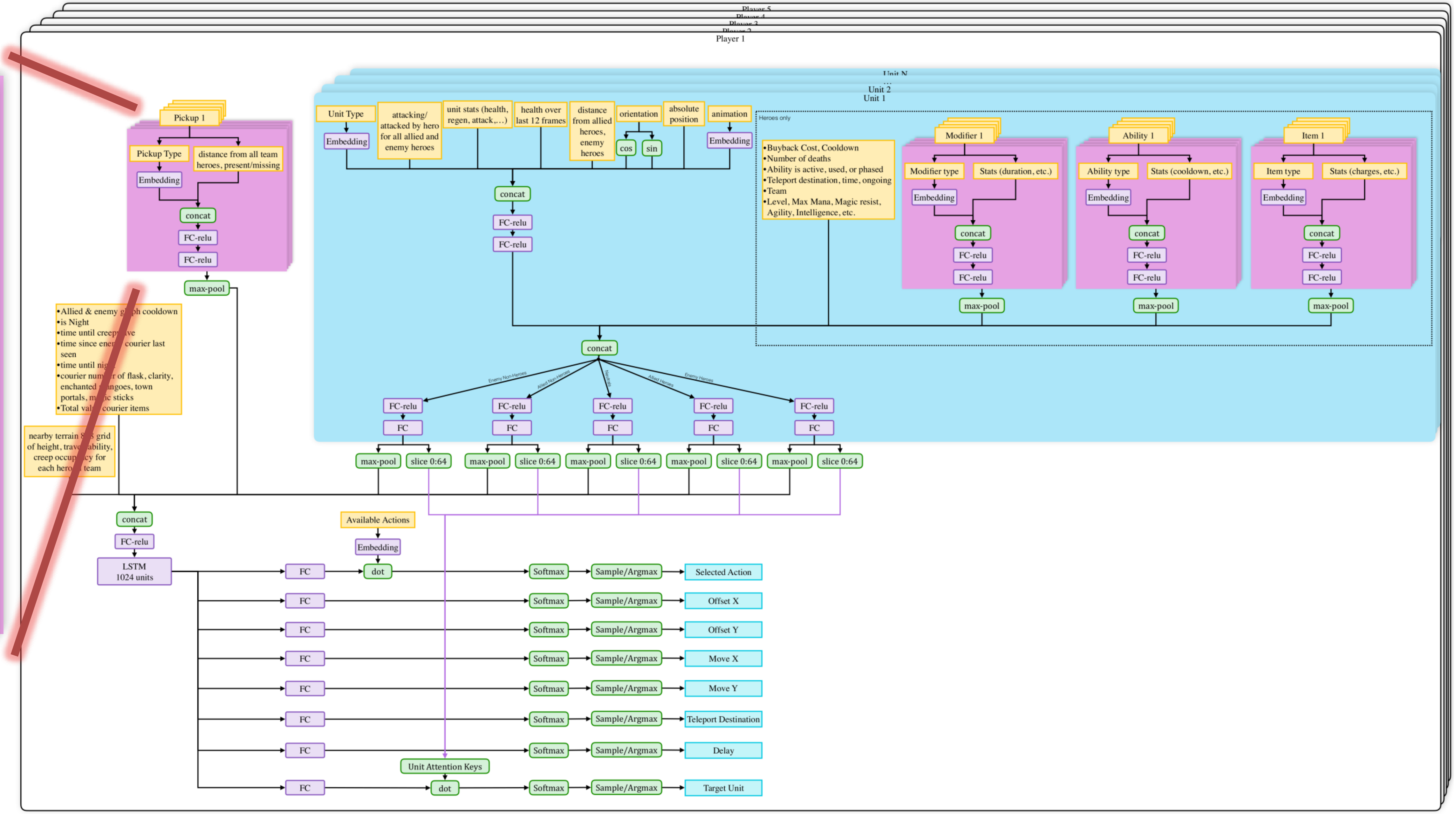
OpenAI Five Lesson #1: specialize your network



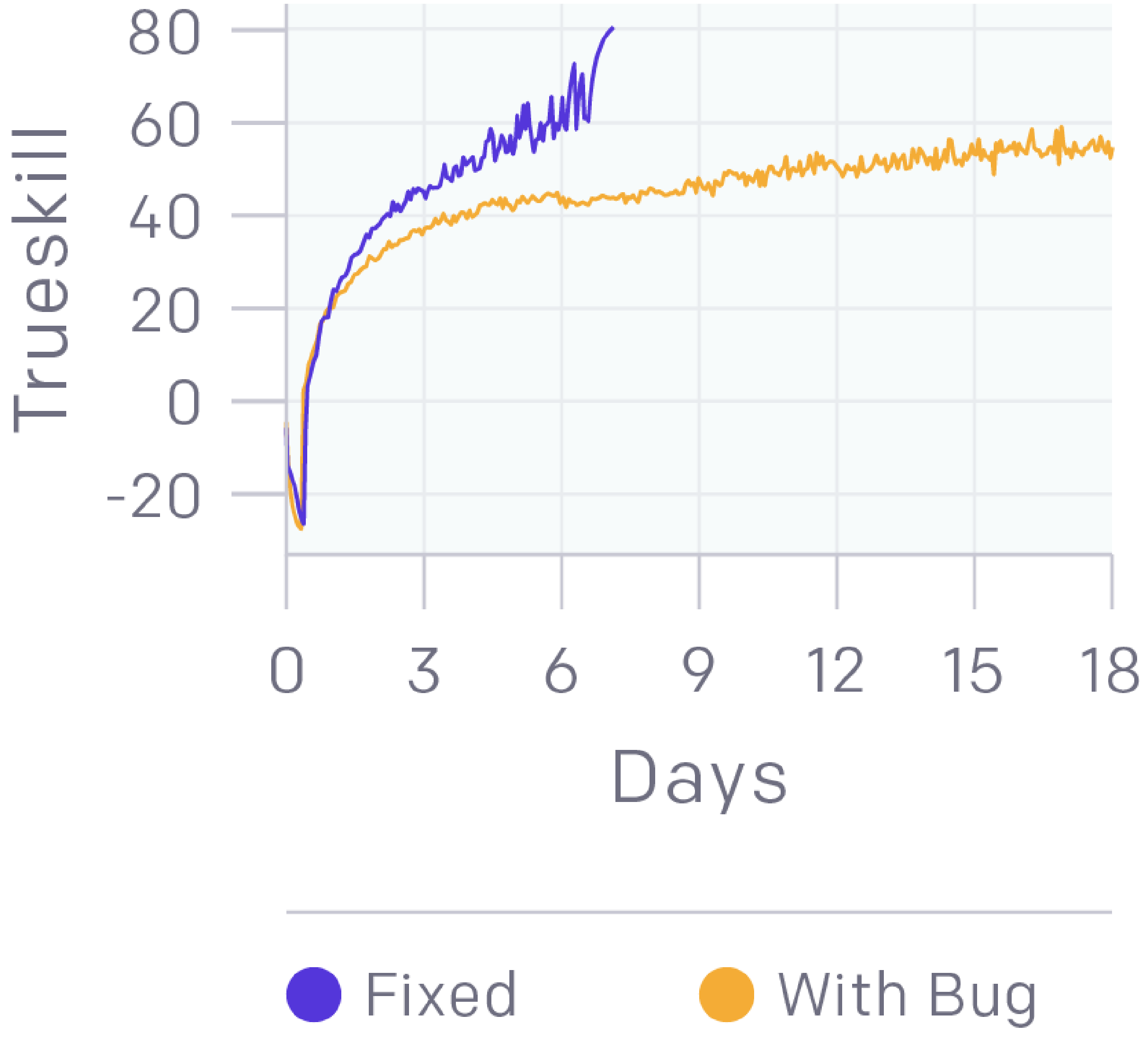
Input preprocessing



OpenAI Five Model Architecture
(06/06/2018)



OpenAI Five Lesson #2: build a robust RL pipeline



OpenAI Five (2018)

Watch for bugs in **state definition** and **action execution**!

See also related Lessons from AlphaZero:

While we were implementing AlphaZero, it took us some time to realize just how finicky the algorithm can be, because even when you have all the hyperparameters way off, it still can learn, albeit slowly. Further, there are

OpenAI Five Lesson #3: watch your wallet

OpenAI Five plays 180 years worth of games against itself every day, learning via self-play. It trains using a scaled-up version of Proximal Policy Optimization running on 256 GPUs and 128,000 CPU cores — a

OpenAI Five (2018)

Learning from game state: compute cost

Computational requirements can remain prohibitive:

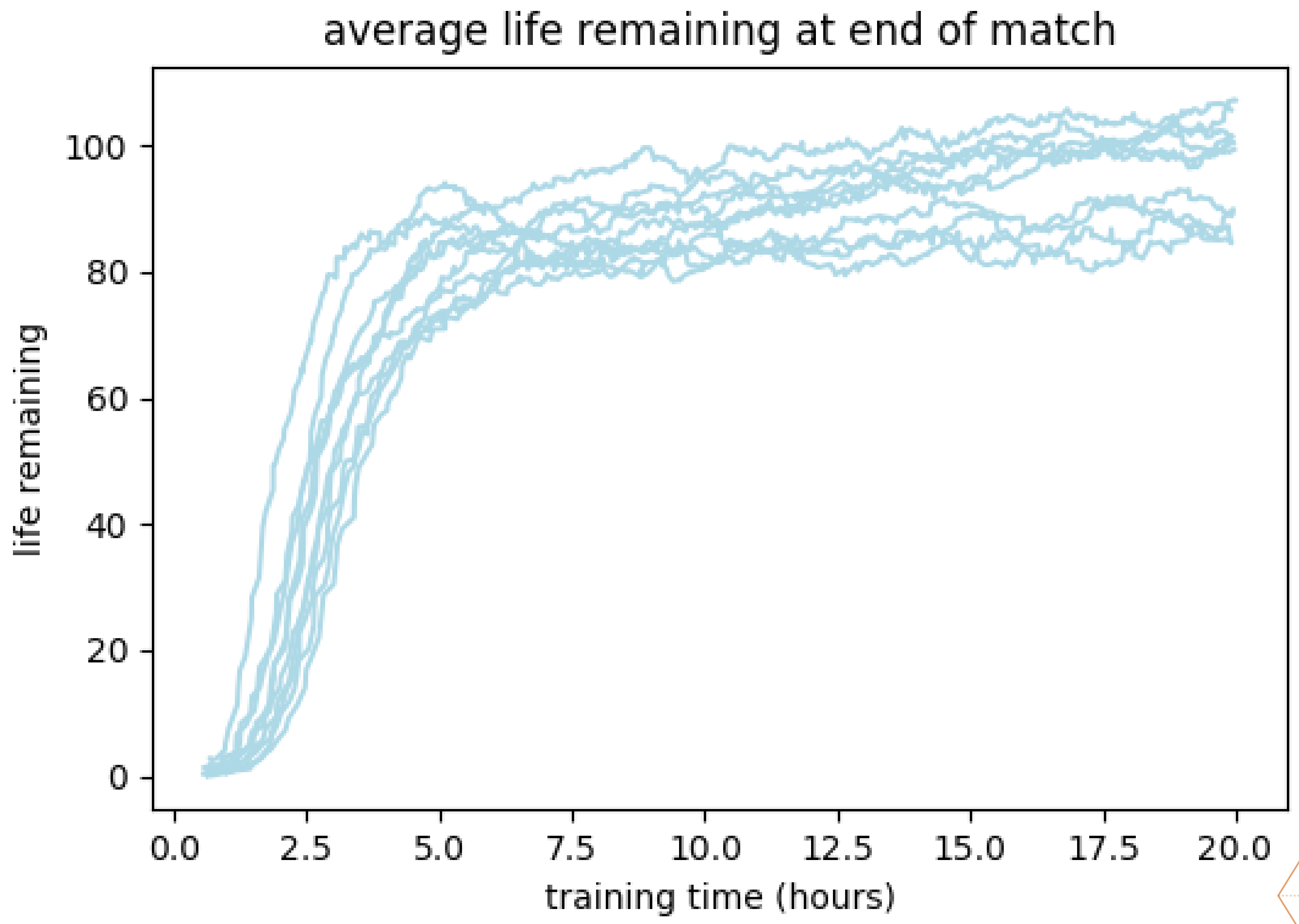
- Complex gameplay
  - Many states and actions
  - Long-term strategy
  - Varied playstyles
  - Multiplayer interactions
- CPU & GPU-intensive game

In order to train AlphaStar, we built a highly scalable distributed training setup using Google's v3 TPUs that supports a population of agents learning from many thousands of parallel instances of StarCraft II. The AlphaStar league was run for 14 days, using 16 TPUs for each agent. During training, each agent experienced up to 200 years of real-time StarCraft play. The final AlphaStar agent consists of the components of the Nash

[source]

For Honor prototype's main limitations

- **Training instability**



For Honor prototype's main limitations

- Training instability
- **Predictability**

> *FIGHTING GAMES ARE, AT THEIR MOST BASIC LEVEL, REALLY FANCY VERSIONS OF ROCK, PAPER, SCISSORS*

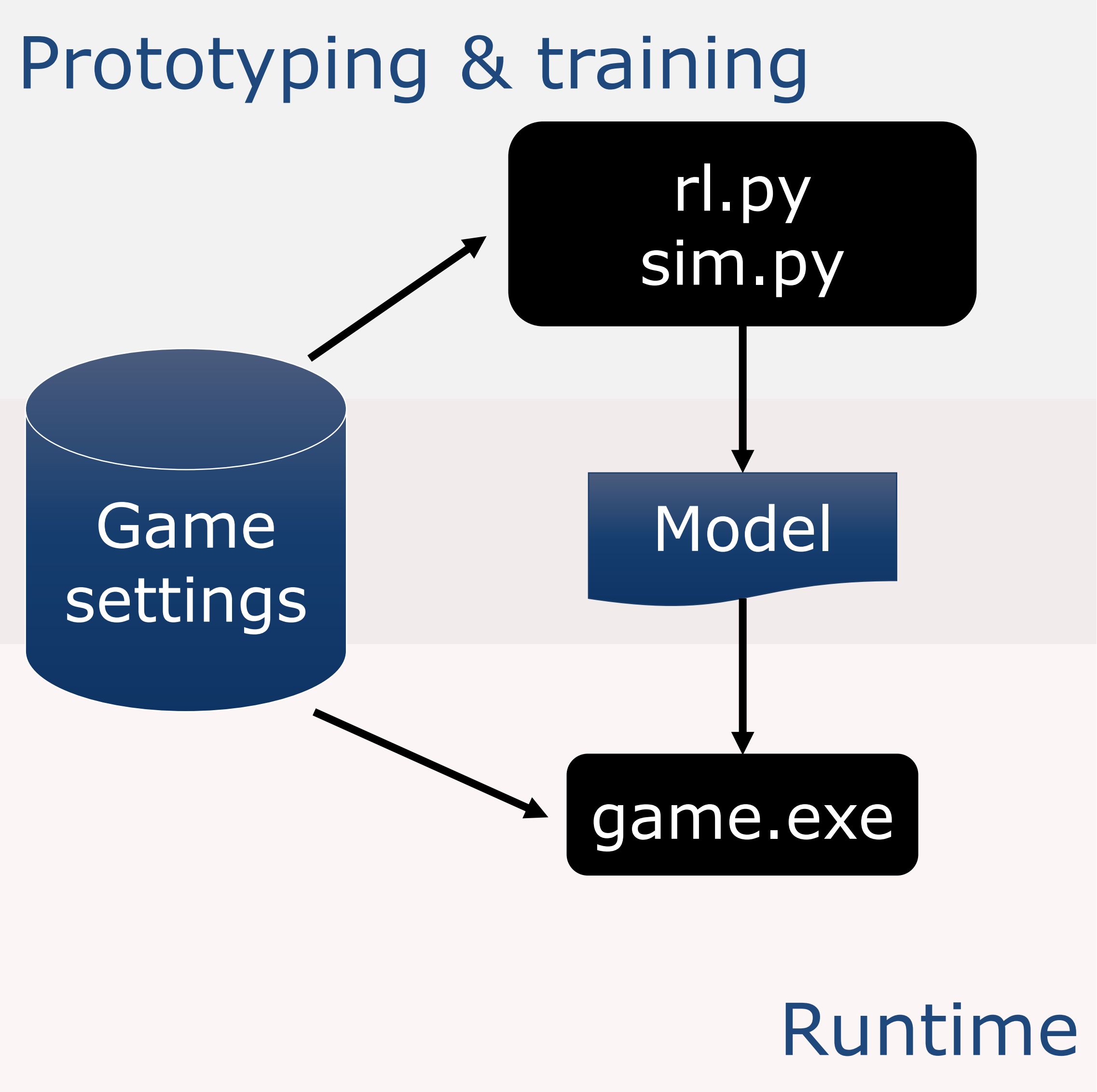
[source]

➔ Stochastic policies

➔ Game theory

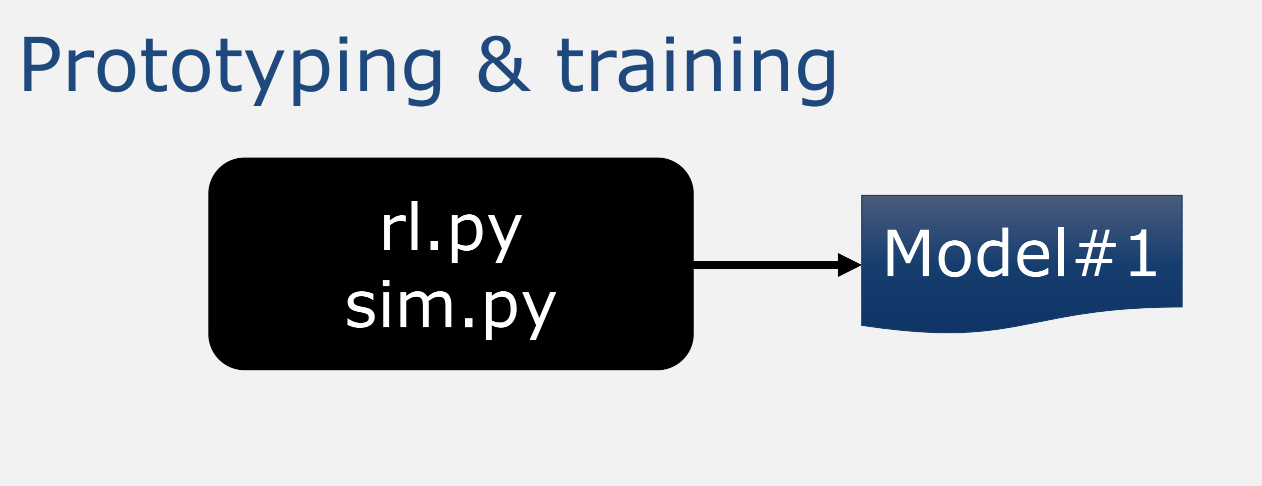
➔ Opponent modeling ("mind games")

Direct transfer from simulation to game

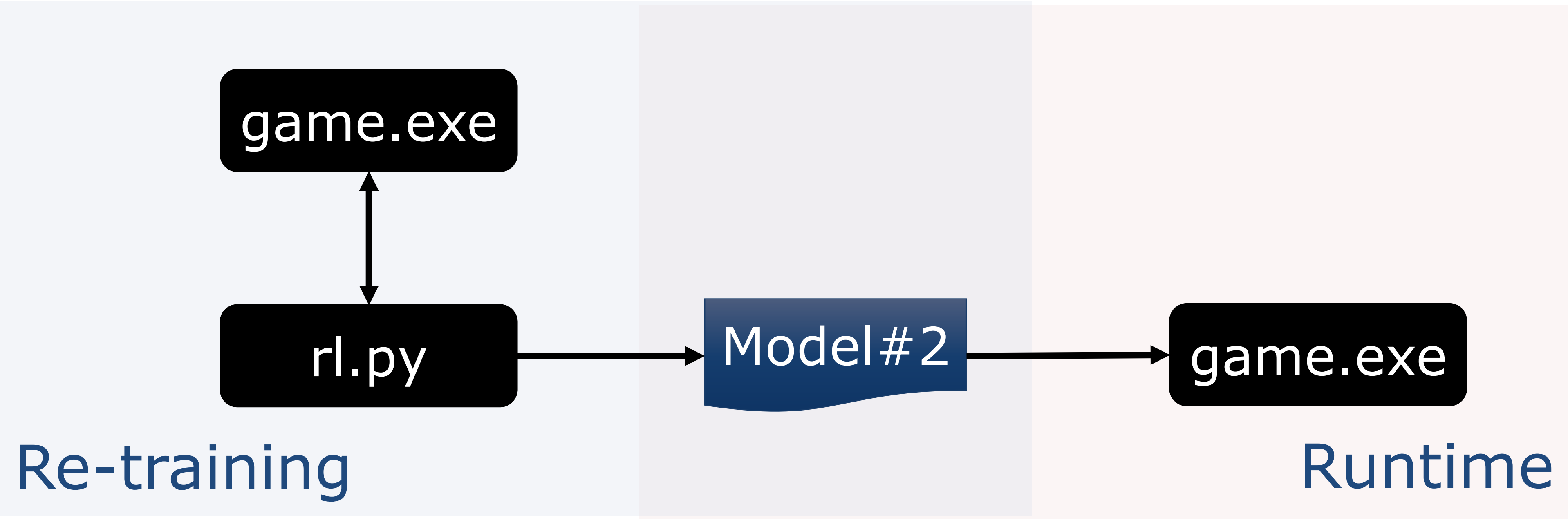


- ➕ Efficient prototyping loop
- ➕ Fast model training
- ➖ Simulator must perfectly match game behavior

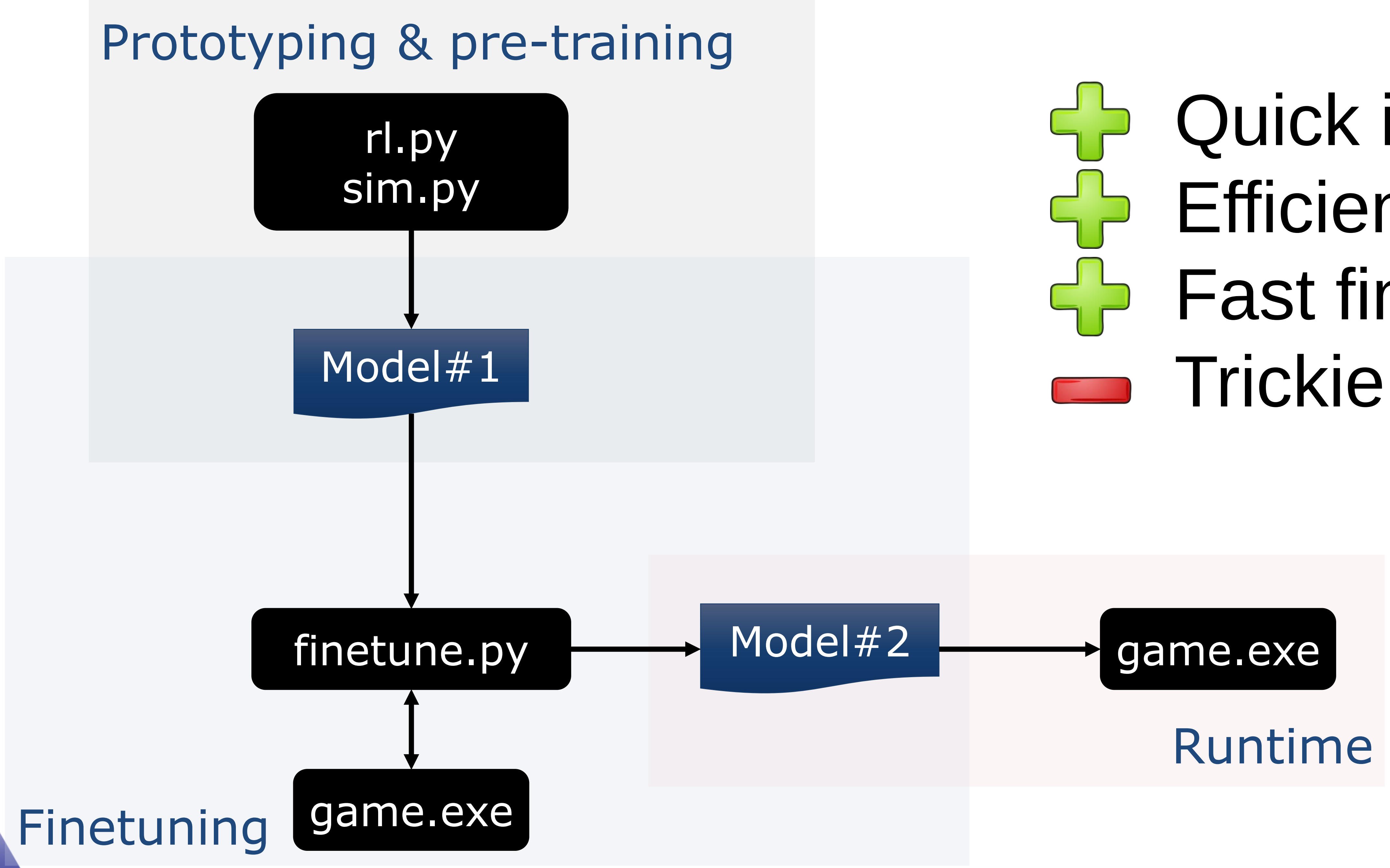
Prototype in simulation, re-train in game



- ➕ Quick prototyping loop
- ➖ Re-training may be slow
- ➖ Re-training may need tweaks



Pre-train in simulation, fine-tune in game



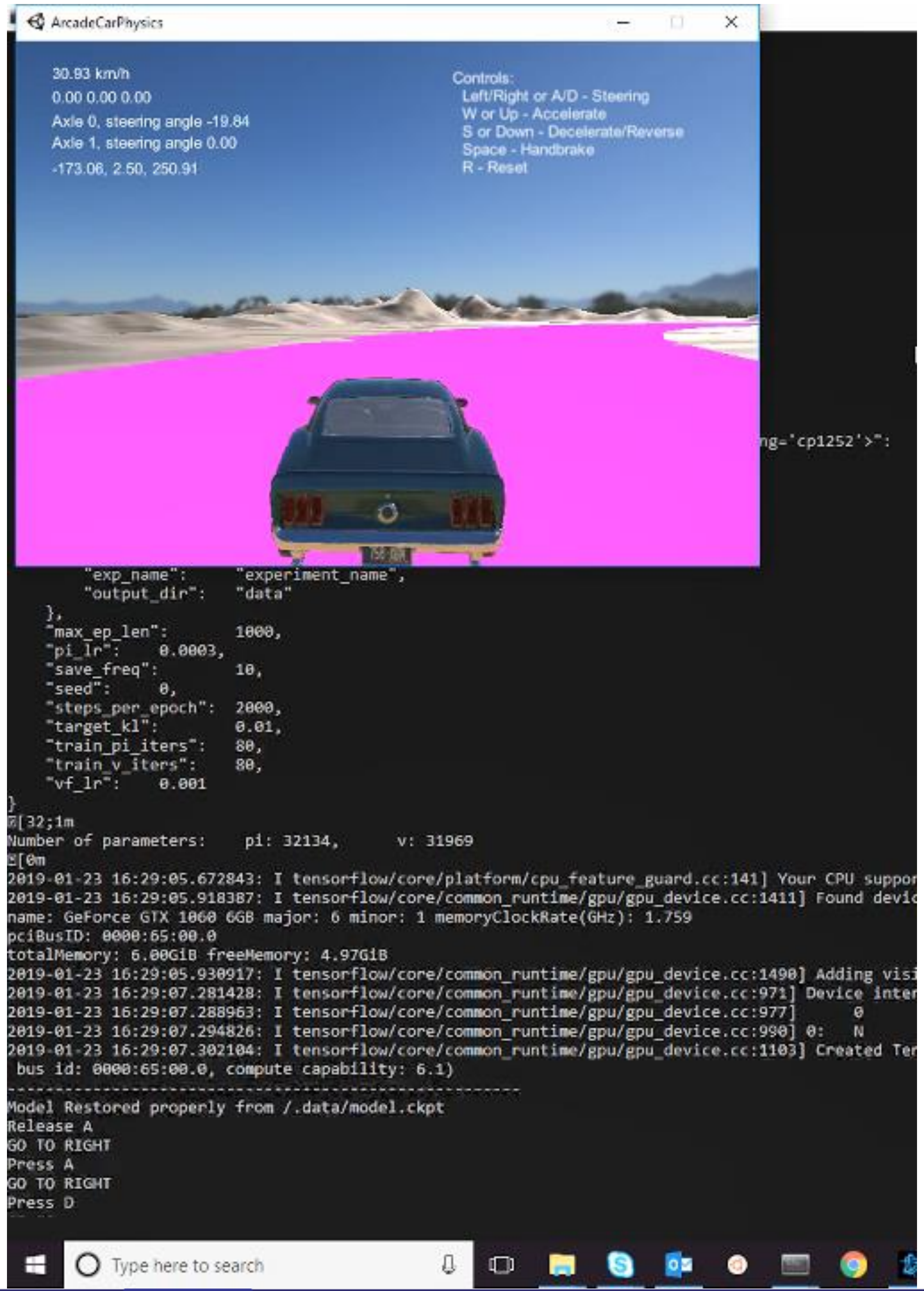
- ➕ Quick iteration loop
- ➕ Efficient pre-training
- ➕ Fast fine-tuning
- ➖ Trickier to implement

Sim2Real

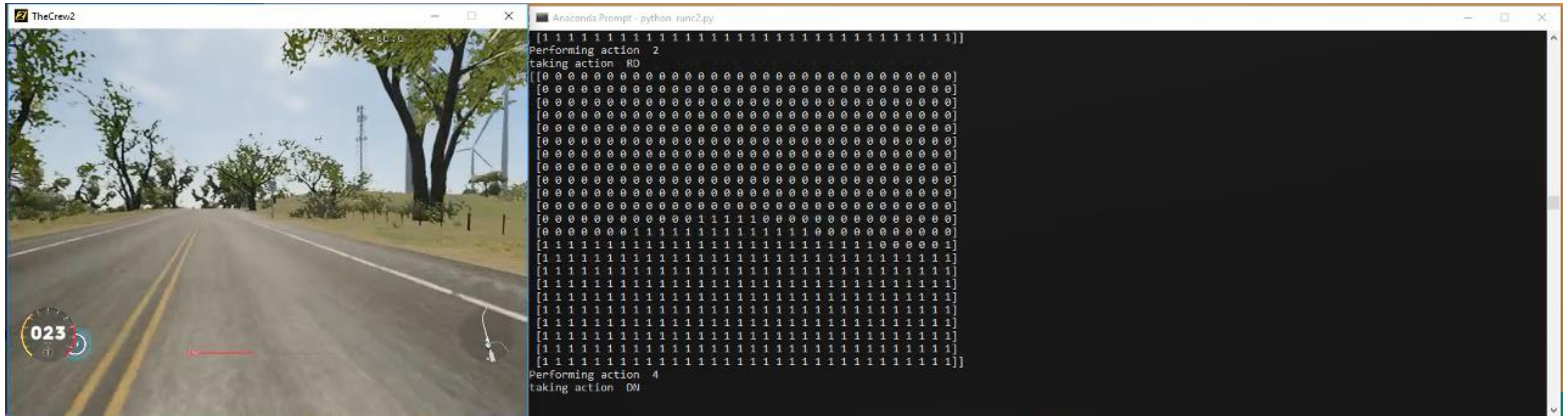
[Sim2Real View Invariant Visual Servoing by Recurrent Control]() (Sadeghi et al. 2017)

Sim2Game

From Sim ([Arcade Car Physics]() – Vehicle Simulation for Unity3D)…



… to AAA Game
(WIP: automated tests in The Crew 2)



Early prototype not using Sim2Game transfer yet

PUNE