# Learning to play Smash Bros. with Deep RL

A Case Study

**Vlad Firiou**, Will Whitney (that's me!), Josh Tenenbaum

# Outline

**Environment** → **Methods** → **Results**

**Methods**

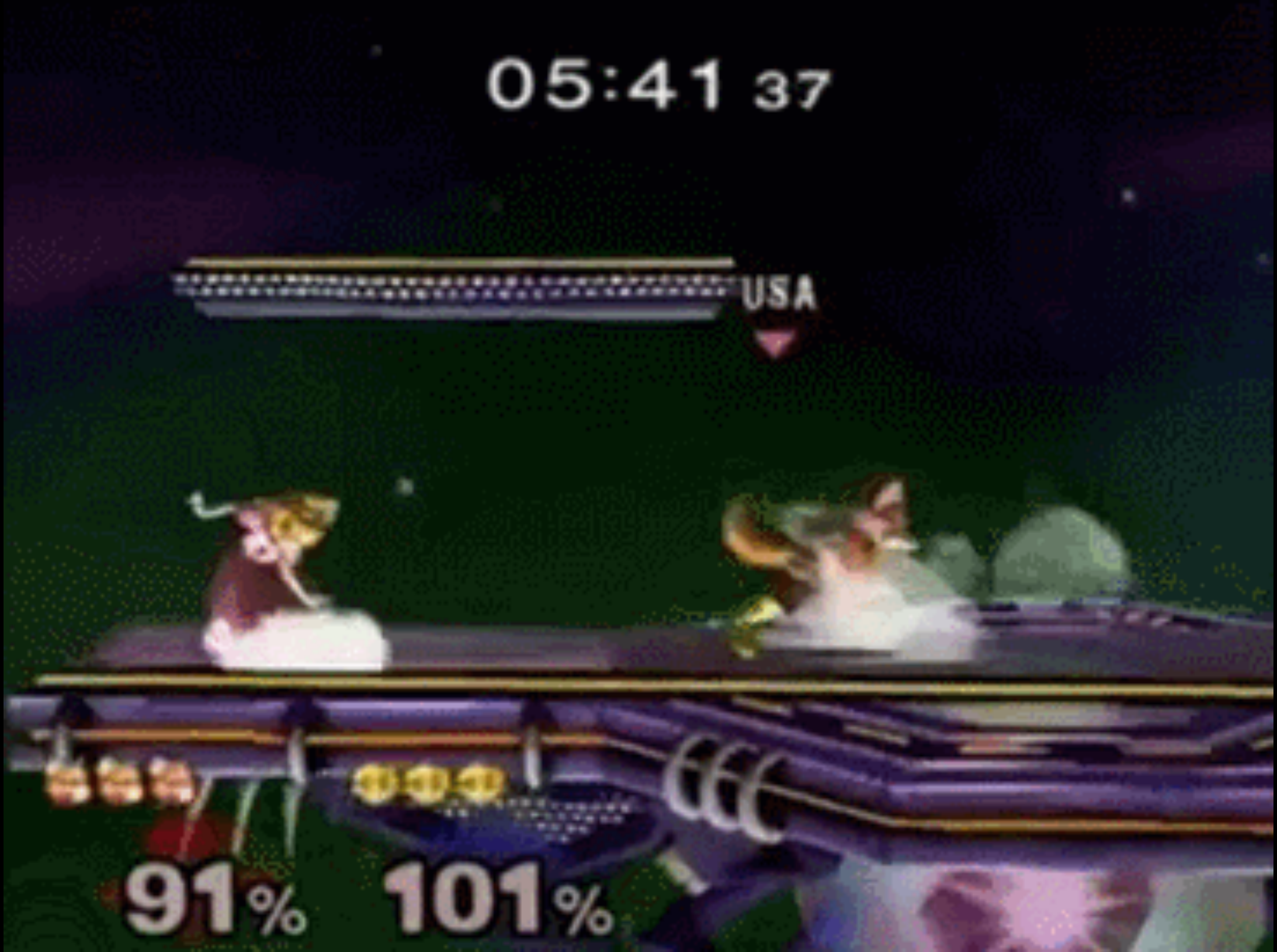
Q learning

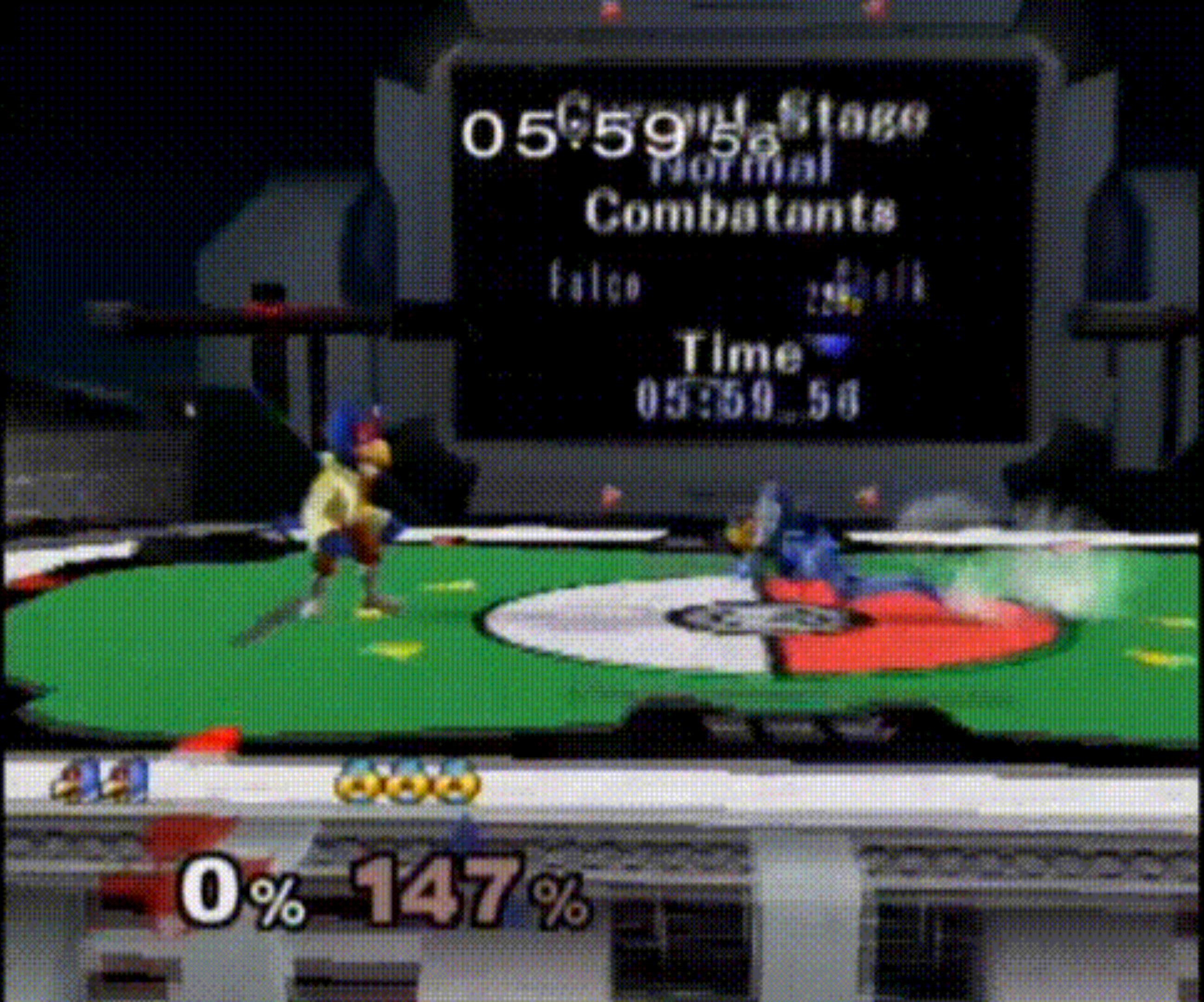
Policy gradient

Training techniques


NINTENDO
GAMECUBE
SUPER
SMASH BROS.
Melee
TEEN
T
Nintendo's Best In
4-Player Action!
Nintendo


NINTENDO
GAMECUBE


05:41 37
USA
91%
101%


Combatants
Falco
Time
05:59.58
0%
147%

# The RL Environment

- Environment simulated with the Dolphin emulator
- Game state is read from RAM on each frame
    - Player positions, velocities, action IDs, etc.
    - mostly observable, except for frame delay
- Actions space is big:
    - 2 continuous control sticks, 7 buttons, 2 triggers
    - simplify to (5 control stick positions) x (6 buttons) = 30 actions
- Reward structure:
    - +/- 1 for kills/deaths
    - +/- 0.01 for damage dealt/taken

# Methods: Q - learning

Given state s and action a, predict the expected future reward:

$$Q(s,a) = \mathbb{E}[R_1 + \lambda R_2 + \lambda^2 R_3 + ...]$$

Problem: very **high variance** from Monte Carlo estimate.

more timesteps → more random decisions → higher variance

Bellman's Equation

$$Q_\pi(s_t, a_t) = \mathbb{E}[R_t + \lambda \underbrace{\max_a Q_\pi(S_{t+1}, a)}_{\text{encapsulates future returns}}]$$

**deterministic!**

# Methods: Q - learning

$$Q_\pi(s_t, a_t) = \mathbb{E}[R_t + \lambda \max_a Q_\pi(S_{t+1}, a)]$$

Two problems with this algorithm:

- Always taking the best action. How do you **explore**? Epsilon greedy?
- **Small** change to Q values $\rightarrow$ **large** change in policy

# Methods: Q - learning

Epsilon-greedy exploration (like in DQN):

$$a_t = \begin{cases} \max_a Q_\pi(s_t, a) & \text{with probability } \epsilon \\ a \sim Uniform & \text{with probability } 1-\epsilon \end{cases}$$

Epsilon-greedy exploration is **slow**

Turn values into a distribution instead:

$$a_t \sim Softmax(Q_\pi(s_t, a))$$

Intuition: don't take **bad** actions, take **ambivalent** actions

[1]see e.g. Double DQN, van Hesselt et al.

# Methods: Q - learning

Bellman's Equation can be **unstable** due to **max** operator[1]

$$Q_\pi(s_t, a_t) = \mathbb{E}[R_t + \lambda \max_a Q_\pi(S_{t+1}, a)]$$

always taking best action → small change in Q → take different action → maybe see **different states**

SARSA takes **expectation** over policy

$$Q_\pi(s_t, a_t) = \mathbb{E}[R_t + \lambda Q_\pi(S_{t+1}, \pi(S_{n+1}))]$$

[1]see e.g. Double DQN, van Hesselt et al.

# Methods: Policy Gradient

## Actor-critic model

- similar to Asynchronous Methods for Deep Reinforcement Learning, Mnih et al

## Trust Region Policy Optimization (TRPO)[1]

- second-order method
- **much more stable**
- slightly better results

## Exploration via entropy regularization

- penalize having a sharply spiked (low-entropy) action distribution
- i.e., **encourage randomness**

[1] Trust Region Policy Optimization, Schulman et al.

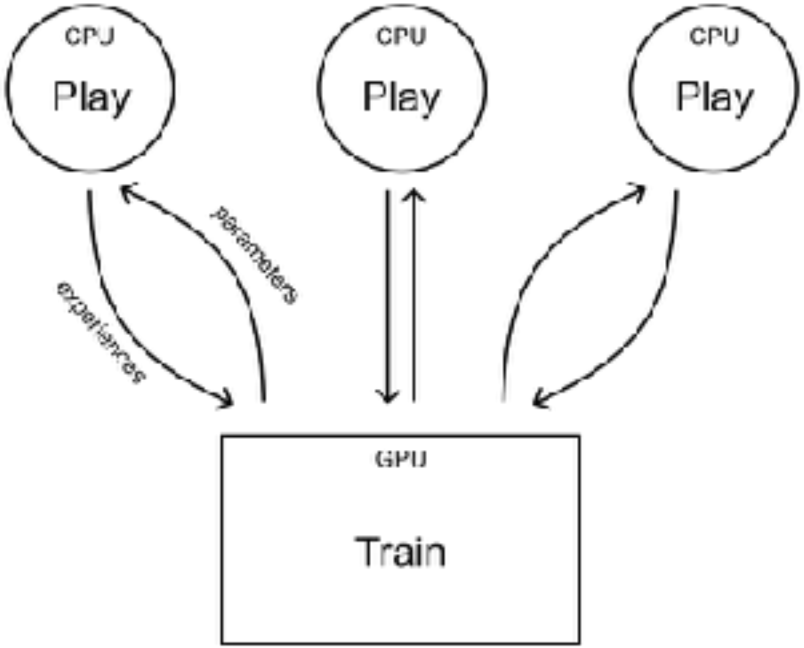
# Asynchronous training

Smash only runs at ~120 fps (2x real)

Need **millions** of frames

Idea:

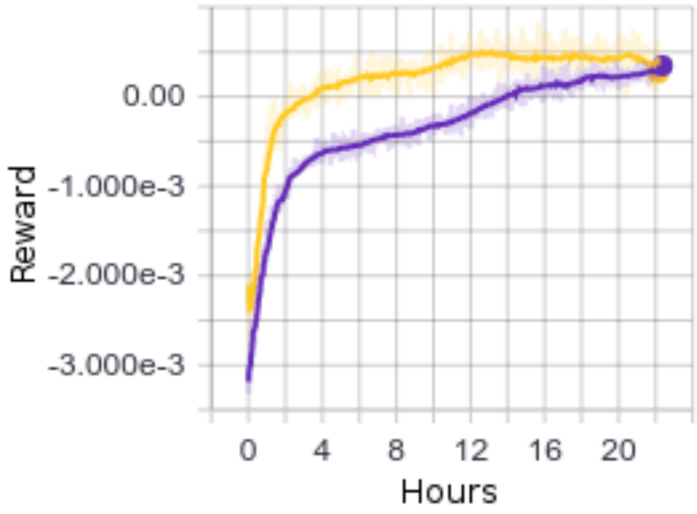
- lots of copies of agent
- lots of copies of emulator
- agents send experiences to trainer
- trainer sends new parameters to agents



# Training

## Start by playing in-game AI on hardest difficulty

- both Actor-Critic and SARSA win after < 1 day
- agents are subjectively OK, but not great




## Self-Play

- play against a population of old versions of the bot (like AlphaGo)
- Actor-Critic beats Vlad or me after a couple of days
- we figured, maybe it's actually really good?

# Vlad Goes to a Tournament

| Opponent | Rank | Kills | Deaths |
|---|---|---|---|
| S2J | 16 | 4 | 2 |
| Zhu | 31 | 4 | 1 |
| Gravy | 41 | 8 | 5 |
| Crush | 49 | 3 | 2 |
| Mafia | 50 | 4 | 3 |
| Slox | 51 | 6 | 4 |
| Redd | 59 | 12 | 8 |
| Darkrain | 61 | 12 | 5 |
| Smuckers | 64 | 8 | 5 |
| Kage | 70 | 4 | 1 |

Table 1: Some results against ranked SSBM players. Rankings from `http://wiki.teamliquid.net/smash/SSBM_Rank`. S2J is considered by some to be the best Captain Falcon player in the world.

# Results

Better than top humans!

Loses to certain silly tactics

- fixed by having more diversity in self-play population

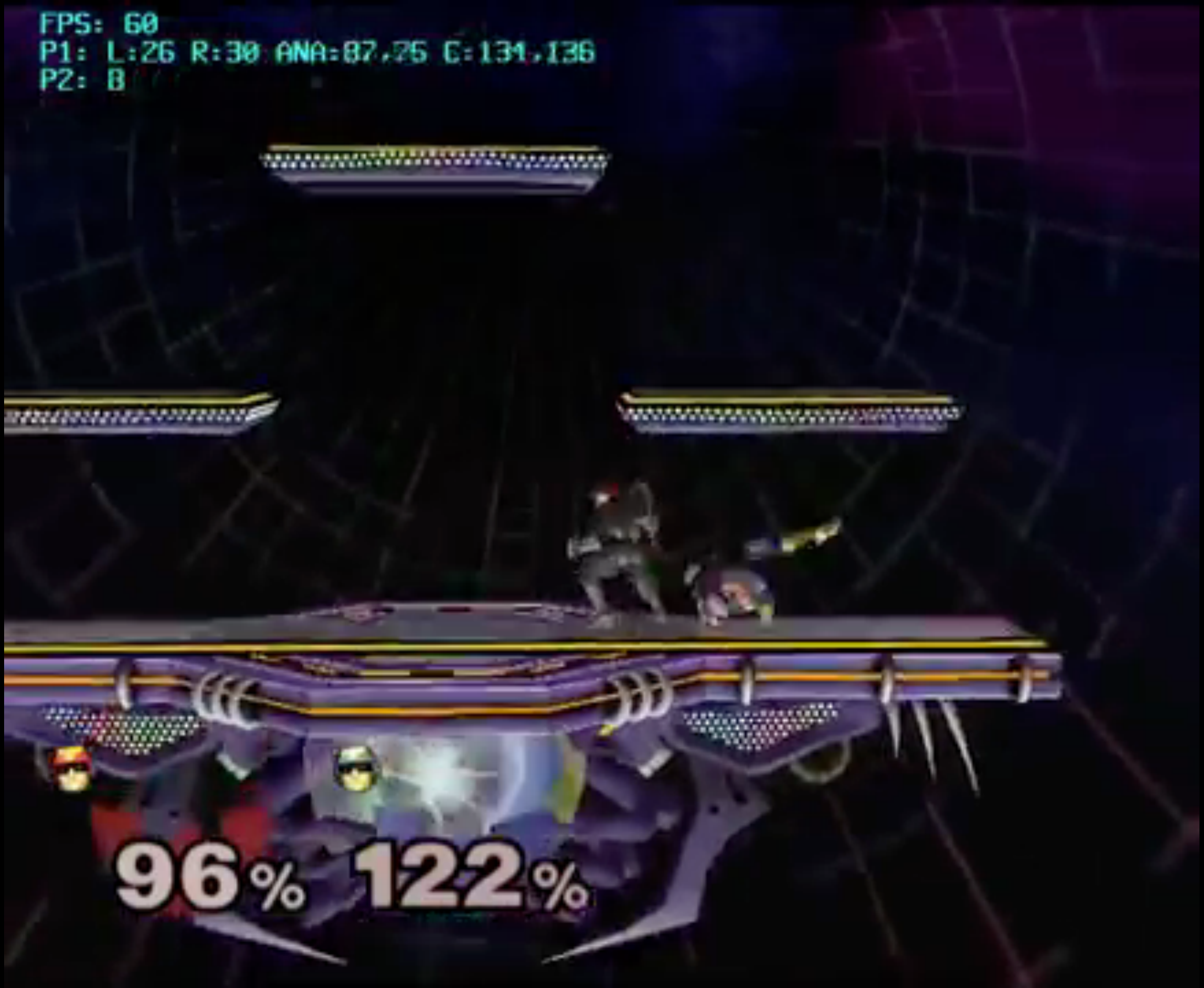
Q-learning is not appropriate for self play

- if the distributions are not stationary, it won't converge

Playing with delay is hard

- human reaction time is about 15-20 frames
- bot performance drops off after ~6 frames of delay


P1
Ready
2 96


FPS: 60
P1: L:26 R:30 ANA:87,25 C:134,136
P2: B
96%
122%