How I won Animal AI Competition

A few words about me:

Denys Makoviichuk

- C++ Software Engineer in Snap Inc.
- Performance Owner in Snapchat AR Engine
- Participated in creating of some most popular Snapchat lenses

Main Challenge problems:

1. **Generalization**

   - You don’t know anything about 99% of the tests
   - It is hard to create different and random tests
   - You need to have one AI to solve different tasks

2. **Exploration and agent confidence**

   - If agent is too confident in its moves it is easy to miss reward
   - Time limit. With 250 frames agent doesn’t have enough time

3. **Sparse rewards**

   - For some levels your agent needs to make sequence of decisions which have very small probability of happening

Algorithm

- My own implementation of the **PPO + LSTM** (Proximal Policy Optimization with vectorized environment)
  - On-policy and model-free algorithm
  - The best way to utilize my home computer (2080TI + AMD Ryzen 3950)
  -
- Worse exploration and sample efficiency compared to **DQN** and other off-policy algorithms
  - No benefit of the experience replay buffer in my case
  - My batch size already took all memory

What is a PPO (Reminder)

- We are trying to maximize reward function for new policy with constraint that KL distance between old and new policy will be small enough. In PPO we solve it using ratio clipping

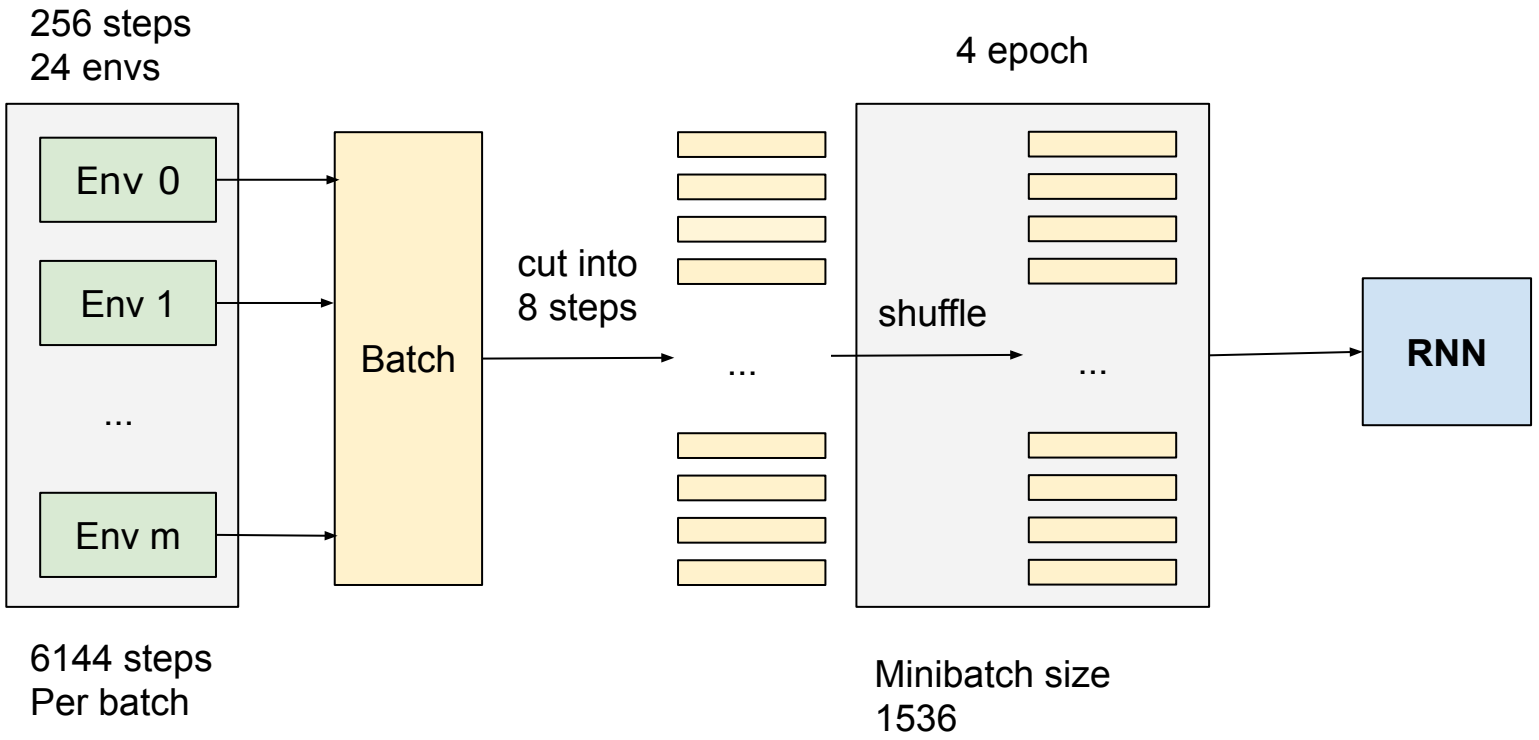
$$r(\Theta) = \frac{\pi_{\Theta}(a|s)}{\pi_{\Theta old}(a|s)}$$

$$J^{CLIP}(\Theta) = E[min(r(\Theta)\hat{A}_{\Theta old}(s,a),\ clip(r(\Theta),\ 1 - \varepsilon,\ 1 + \varepsilon)\hat{A}_{\Theta old}(s,a))]$$

$$J^{CLIP'}(\Theta) = E[J^{CLIP}(\Theta) - c_1(V_{\Theta}(s) - V_{target})^2 + c_2 H(s,\ \pi_{\Theta}(.))]$$

J - Reward Function,
A - Advantage (Returns - Predicted Values)
S - observation, a - action
ε - epsilon clipping , π - policy

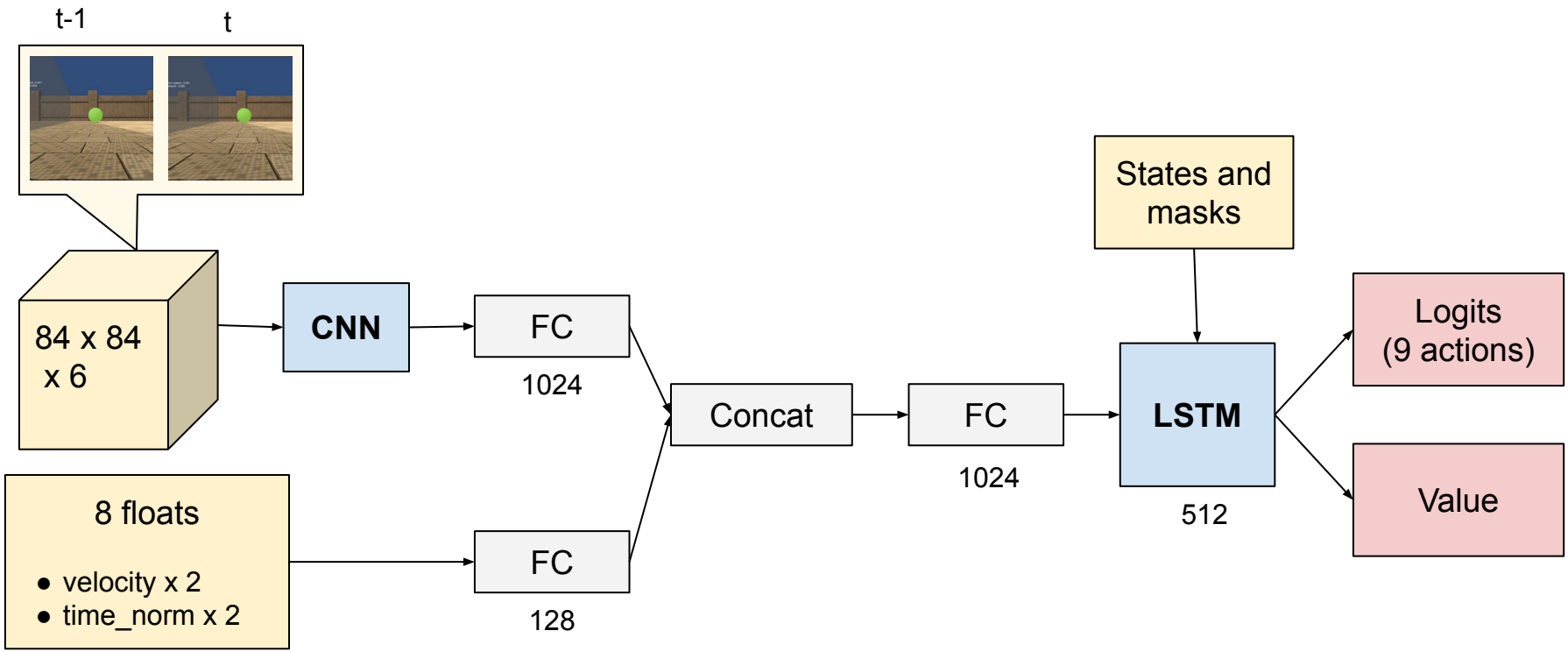
PPO Learning Process



Training Stability Improvements

- Shared network for Value and Policy
  - It improves train performance
  - Especially with LSTM
- Generalized Advantage Estimation
  - TD vs Monte Carlo rollout
- Clipped Value loss
  - It is very useful for shared network
  - Increases stability for LSTM
- Normalized Advantages in a batch before learning
  - Reducing variance in cost of bias
- Gradient Clipping
  - Must have for recurrent networks
  - Improves learning performance

Network Architecture and Inputs



First Steps

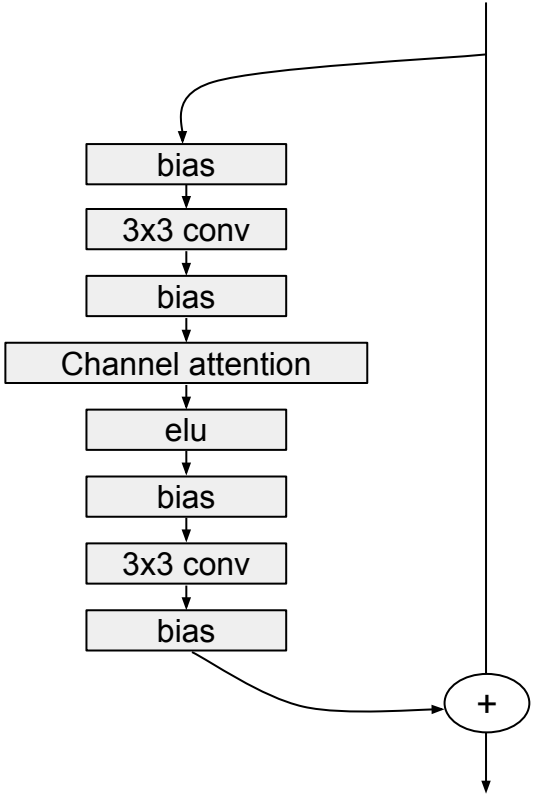
- It was hard to make a network that solves even first category in 100% cases
  - Simple 4 layers CNN couldn’t see small ball on the distance
  - Small reward for small ball, movement was not improving score

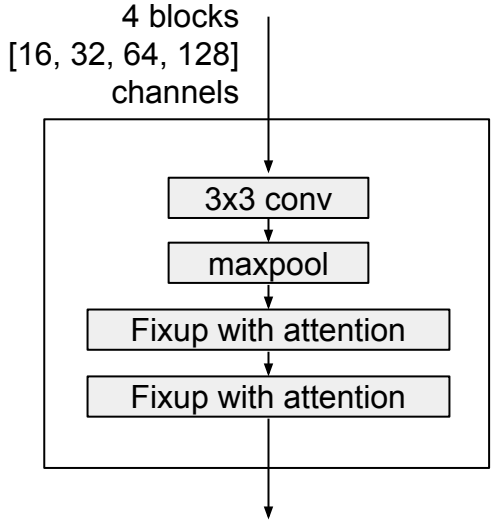
- In a few iterations I improved network increasing its size and adding features
- I ended up with residual network (almost same as described in IMPALA)
  - Used Fixup Initialization
  - Added channel attention

View from the inputs:

My network understands semi-transparent walls

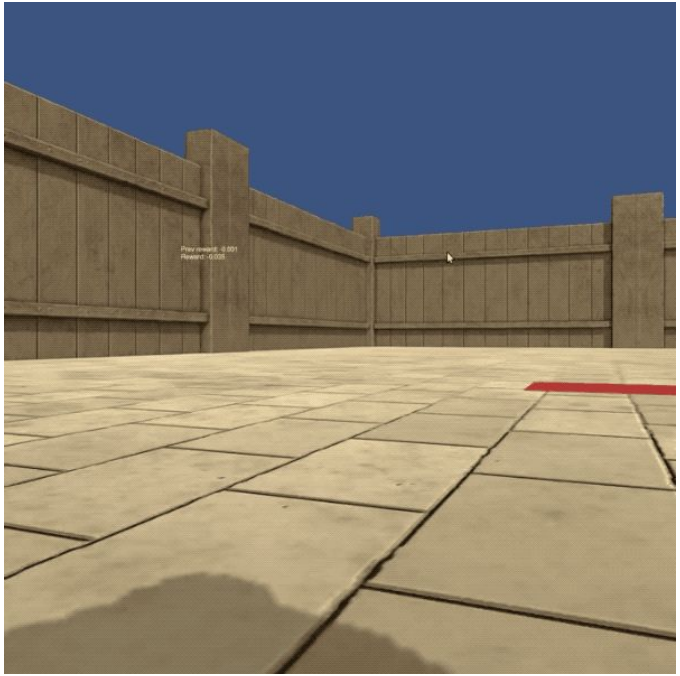
CNN Architecture





Validation

- Made a 100 runs with limits of 250 and 500 frames for simple scenes to cover 6 first categories.
- **Two metrics** were introduced: total score and percent of the solved levels
- **Main idea** was that if network can’t solve simple levels it means that it is not working
    - At the beginning my network was able to solve nearly **60%** of random environments with a few walls in 250 frames limit
    - latest submission had near **98%**

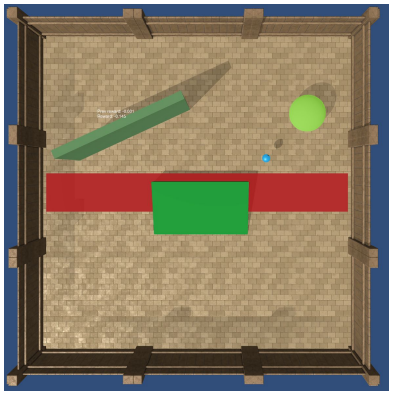
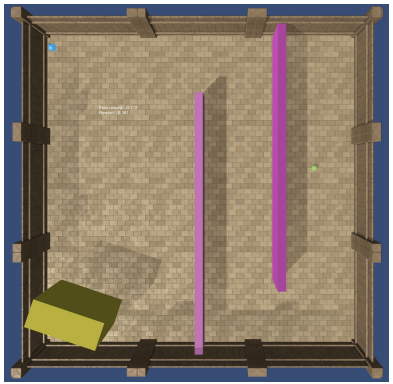
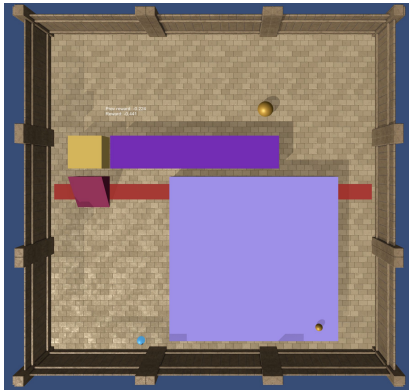
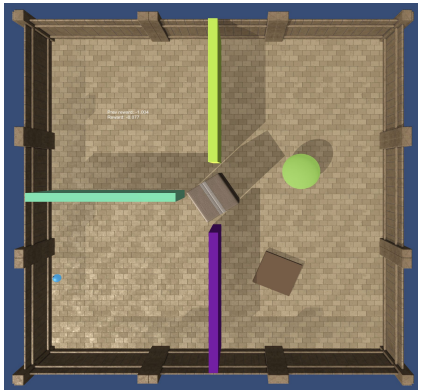
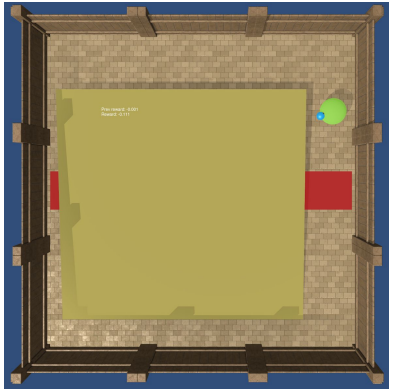
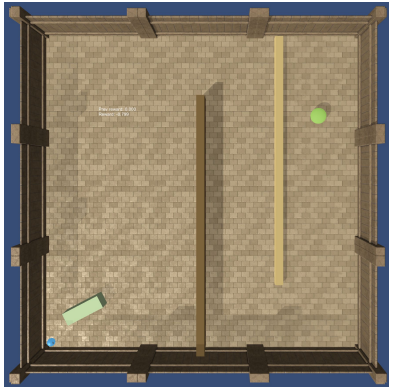
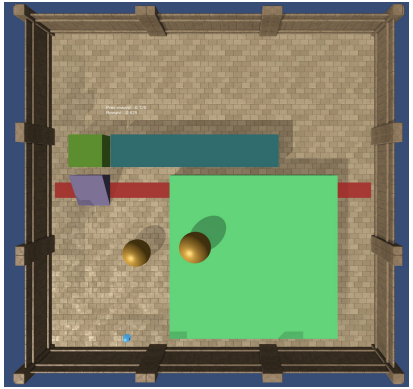
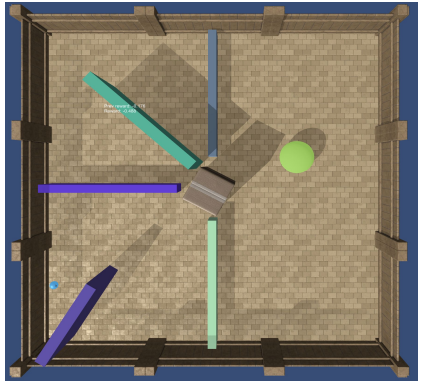


My agent is avoiding red zones

Training Process

- 90% of my levels were very similar to the one we had in demo
- I created different levels with a few of the available objects
  For example:
  - Wall + Lava + Moving balls
  - Empty level with a lot of random balls
  - Level with lot of ramps, and etc.
- Custom levels didn’t work without randomization
- For every new level I randomized maximum number of steps in range of 200-1100
- At every step there was a slight chance (about 0.0001) to blackout for a few frames

4 Types of The Custom Levels

Final solution

- Learning performance was 600 frames per second
- Near 40 million of frames per day
- Final solution was near **70 billion** of frames. Which is pretty low!
- Less than **2 days** to learn
- It learned **from scratch**. No imitation learning
- I got the best score in open tests near a week before ending. I had nothing to do and decided to train a fat network with twice bigger fully connected and lstm layers
- After 50 billion of steps I reduced **entropy** to **0.001**,

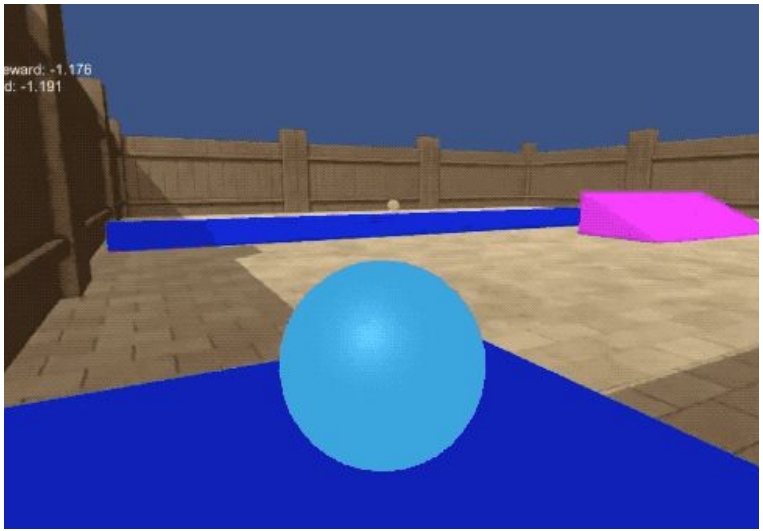
learning rate to the **5 * 10e-5** and **epsilon clip** to the **0.1**

That's all!

Ramps

- Agent didn’t want to explore the ramps
- It only worked when he saw the ball

Solution: Add reward for the vertical component of velocity



My agent solves two hidden tests

Reward Shaping

- Environment returns -1.0/frames every step plus reward for balls proportional to their size
- I added 0.5 to all positive rewards, it helped with small balls
- I encouraged agent to use ramps for exploration
- Also added small penalty for moving backwards

```
def calc_rewards_v2(reward, vel, penalize_back = hps.BACK_MOVE_PENALTY, reward_up = hps.REWARD_RAMPS):
    if reward > 0.1:
        reward += 0.5
    if reward_up and vel[1] > 0.01:
        reward += vel[1] * hps.RAMPS_COEF
    if penalize_back and vel[2] < 0:
        reward += vel[2] * hps.BACK_MOVE_COEF

    return reward
```

Learning Config

GAMMA	0.99
LAMBDA	0.9
LEARNING_RATE	1e-4
GRAD_NORM	0.5
ENTROPY_COEF	0.01
E_CLIP	0.2
NUM_ACTORS	24
STEPS_NUM	32 * 8
MINIBATCH_SIZE	384 * 4
MINI_EPOCHS	4
CRITIC_COEF	1.0
SEQ_LEN	8

- For every batch was generated 32 * 8 * 24 = **6144** steps
- After **50** million of steps I reduced entropy coefficient, e_clip and learning rate and increased blackout chance
- **600** Frames Per Second
- Total train time: **40** hours
- Total **70** million of steps

Why it worked?

- **Simple and Fast**
  - C++ Experience helped me
- No assumption from my side about hidden levels
  - Didn’t try to guess
  - Because of the random levels I didn’t need curriculum learning
- High training performance and sample efficiency
  - Invested more time into network architecture
  - Iterated pretty fast over different approaches
  - Used short sequence length for LSTM
- **Randomized** everything I could

My Results

Team	Total	1. Food	2. Preferences	3. Obstacles	4. Avoidance	5. Spatial Reasoning	6. Generalisation	7. Internal Models	8. Object Permanence	9. Numerosity	10. Causal Reasoning
Trrrrr	43.7	87.8	72.2	30.0	44.4	42.2	54.4	57.8	10.0	32.2	5.6
ironbar	43.6	87.8	63.3	40.0	48.9	40.0	34.4	51.1	25.6	43.3	1.1
sirius	38.7	92.2	66.7	23.3	26.7	34.4	36.7	50.0	6.7	48.9	1.1
BronzeBlood	35.4	73.3	81.1	17.8	21.1	30.0	40.0	38.9	2.2	50.0	0.0
Oltau.ai	35.0	71.1	73.3	21.1	24.4	24.4	33.3	46.7	5.6	50.0	0.0
sungbinchoi	34.3	84.4	56.7	23.3	27.8	31.1	42.2	28.9	1.1	36.7	11.1
DeepFox	33.2	82.2	45.6	24.4	34.4	24.4	27.8	45.6	12.2	34.4	1.1
ARF-RL	32.7	81.1	52.2	14.4	31.1	25.6	27.8	44.4	10.0	40.0	0.0
UniboTeam	32.6	73.3	75.6	13.3	21.1	23.3	32.2	36.7	8.9	40.0	1.1
Juramaia	32.0	84.4	71.1	12.2	31.1	13.3	33.3	30.0	10.0	32.2	2.2

- Best Score In:
  - Spatial Reasoning
  - Generalisation
  - Internal Models

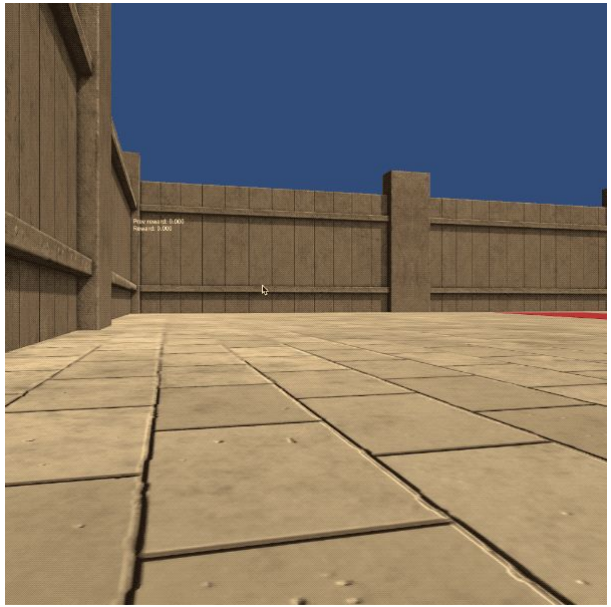
Agent Fails

Agent is running around the same places

- LSTM and time left in inputs helped to fix this issue

My favorite fail - moving backwards

- More train time plus penalty fixed it



What If I Had More Time

- Improve test levels, my levels were really bad
  - On some random colours agent worked worse
  - Unexpected but it learned to recognize green or red walls pretty easily
- Experiment more with inputs
- Add more LSTM layers
- Train it for 200 millions of steps or more
- Imitation learning could help for hard cases

Questions?

email: trrrrr97@gmail.com
denys.makoviichuk@snap.com

twitter: https://twitter.com/DenysM88

Github link:
https://github.com/Denys88/rl_animal