Reinforcement learning for Achtung game

Final project for neural network course made by

Bartosz Rudzki and Krzysztof Bednarek.

Game and training technology

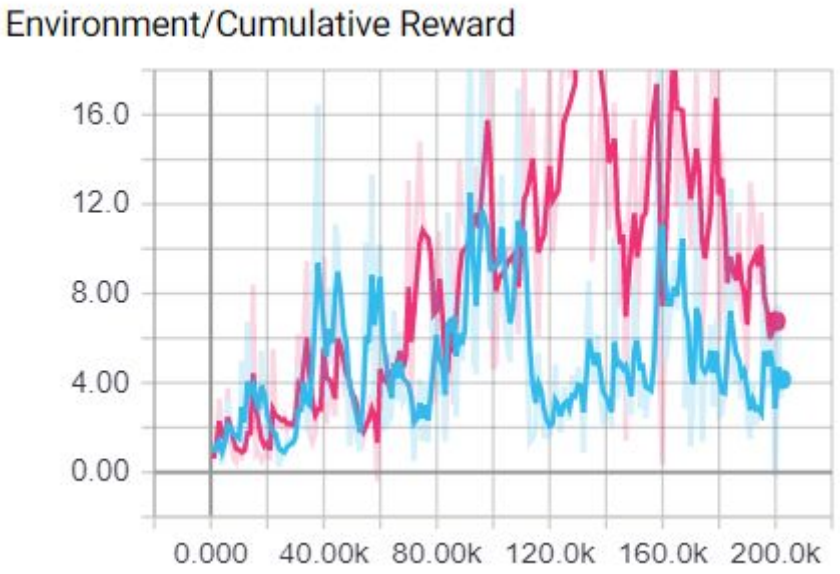
We used our implementation of the game written in Unity. Training was made with use of PPO algorithm included in ML-Agents plugin for Unity (https://github.com/Unity-Technologies/ml-agents).

Research topics

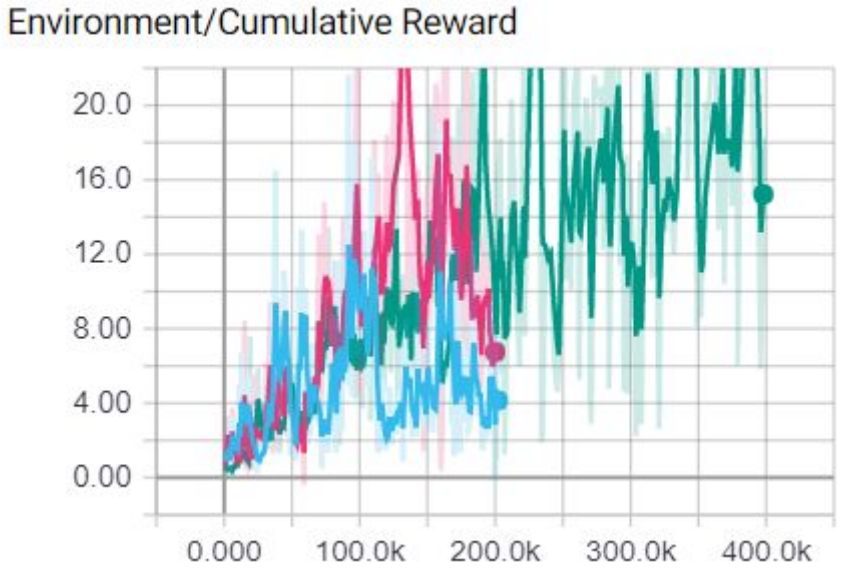
At first we wanted to learn agent play as long as possible without any enemy. We tried to separate approaches: visual learning and distance learning. We checked which parameters and what amount of data is enough for agent to play at decent level. Next we combined two best results to create one combined agent to finally train it to play competitively. We added also map builder to try training agents in specified conditions to enforce some desired behaviours.

Visual learning

This approach as input data takes scene camera images. At first we used one steady camera pinned to the head of the achtung dot. It gave poor result. First big improvement was rotating camera to make agent think that it always rides up. It simplified task for agent and gave a lot better result.

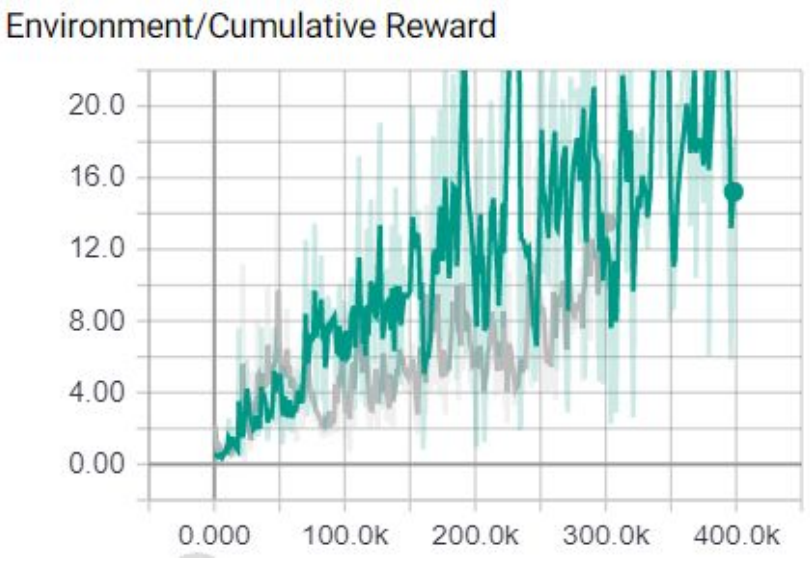


Graph represents how much rewards agent gets during the training process. Agents gets 0.01 reward every step and -1 when it loses.
What pushed this approach further was adjusting hyperparameters responsible for gathering enough amount of data: batch_size and buffer_size. At first we used batch_size: 10, buffer_size: 100 which speeded up the learning process but gave poor agent performance. Increasing those values to batch_size: 512, buffer_size: 4096 resulted in slower but more stable learning process with visible improvement.

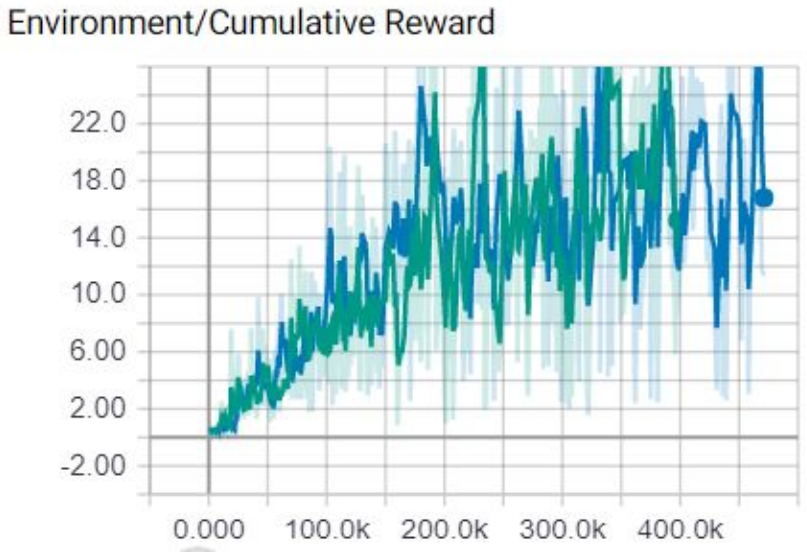


It is good to acknowledge that in the next 200k steps new agent improved his results.
After reaching that point we wanted to learn agent understand whole map. Till this point the agent used one camera with size of 5.

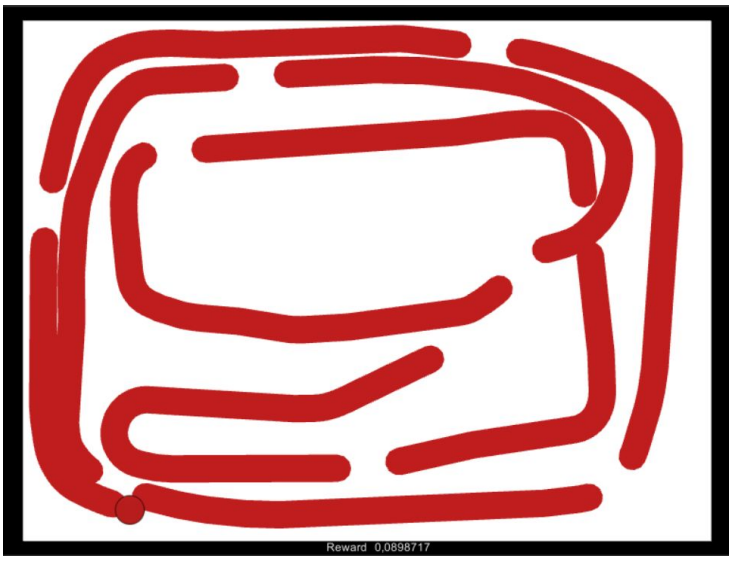
An agent brain used this image with resolution of 84 x 84 pixels as input data. What is more the input was grayed to simplify the task as border (black) and actual dot trace (red) are both killable obstacles for an agent.
Next we were adding more cameras with different sizes to check if it helps. For example for one camera with resolution 256x256 and size 10 the results were quite bad.



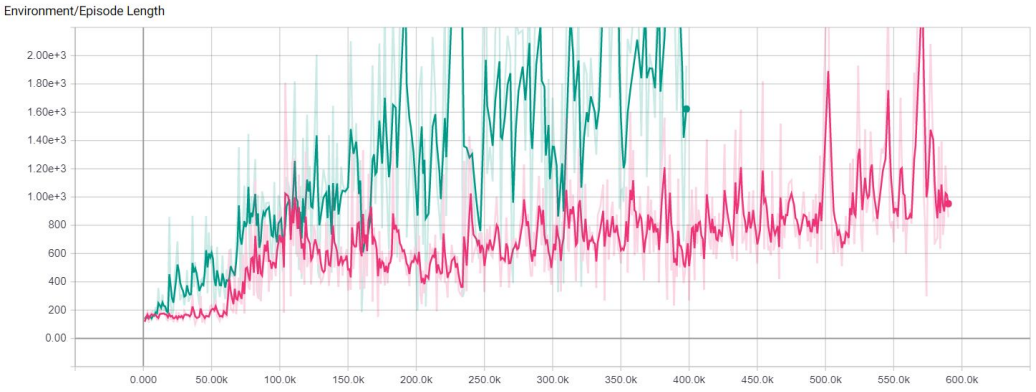
The agent was learning worse and the training process was 5 times slower (gradient propagation etc)!
We tried adding one more camera for our current “champion” (green one). It was camera of size 20 that covers whole board. We hoped that agent will detect open spaces to better fill the board. Unfortunately the final result was no better than green one.



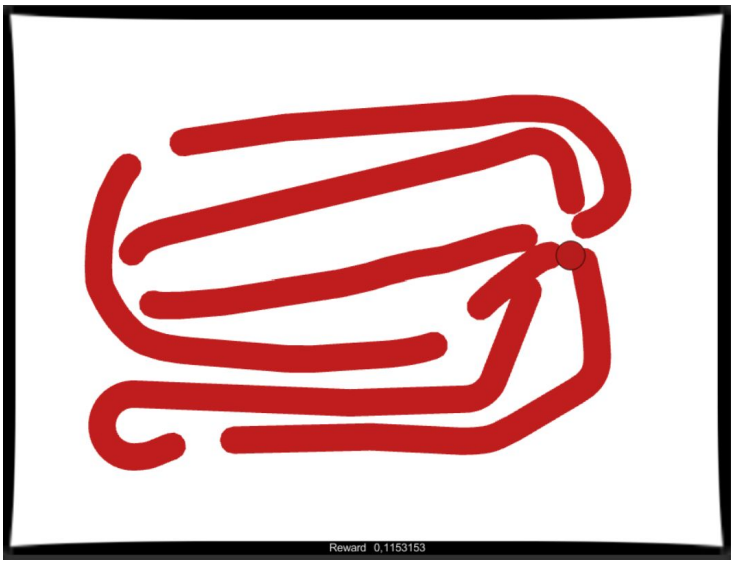
Four cameras (with sizes of 2, 4, 8 and 16) did not work as well. Adding more cameras resulted only in increased needed time for making desired amount of steps.
Best agent preferred long spiral rides leaving some place in the middle of the board.



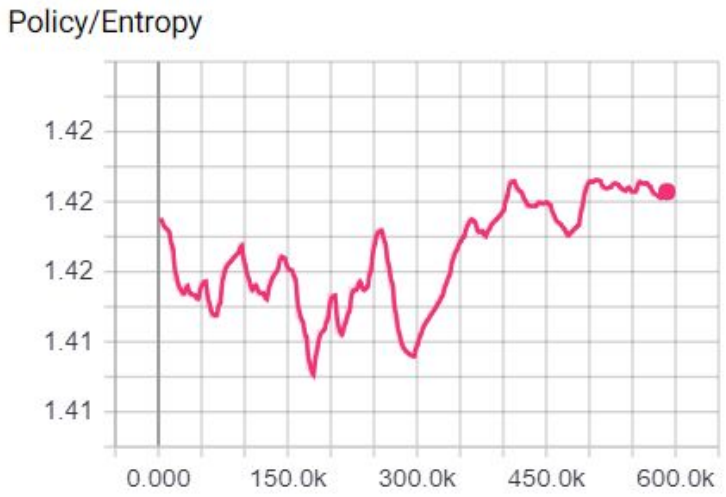
We strive to enforce better space management. One possible solution was changing rewards. Instead of constant reward every step we were adding normalised sum of distances to closest obstacles. We measured those values every 33 degrees from -165 to 165 around the head. We trained next agent for our current best (green one) parameters but with this new reward system.



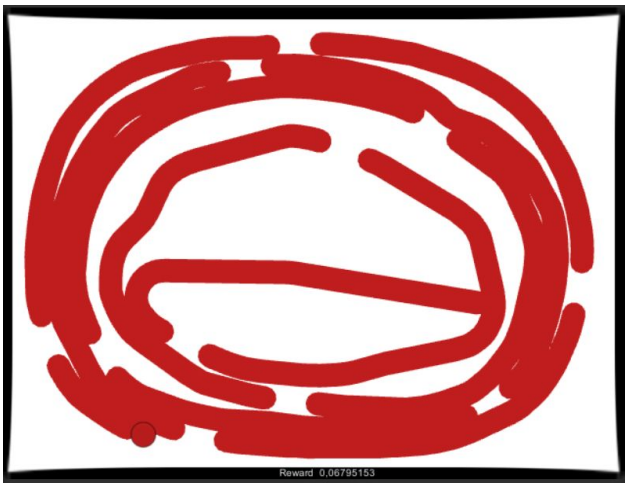
Results where worse. Agent had tendency to make interesting shapes but that was not effective in the end.



We reviewed the problem and it turned out that new agent had problem with entropy.

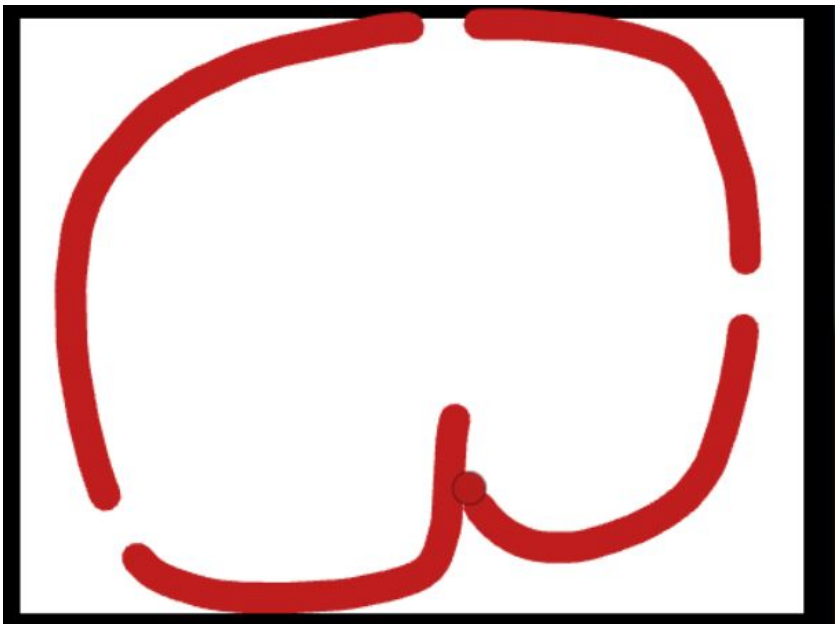


It should slowly decrease during the learning process but it was increasing instead. To adjust it we lowered the beta parameter (which is responsible for making random decisions) from 5e-3 to 1e-4 and it helped. Unfortunately agent learned that way was as good as our previous best one. New approach did not fixed the problem with wasted middle space.

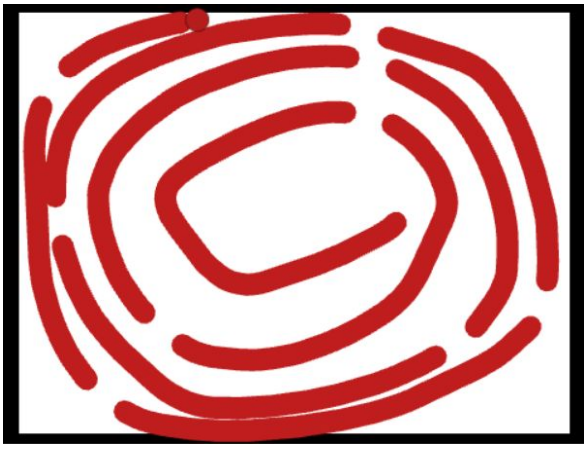


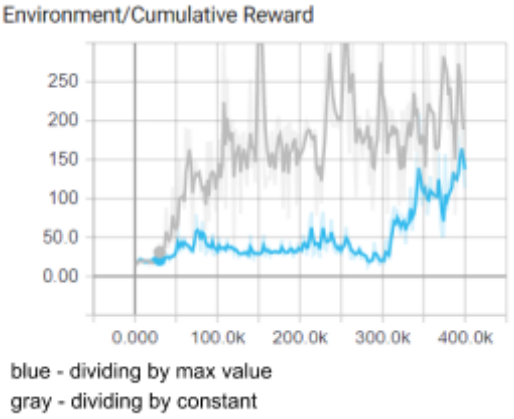
Distance learning

Other approach was to give distances to objects in front of agent. We used 101 samples every 2 degrees. But plugin should receive numbers in range 0 to +1. So firstly we tried to deal with too big numbers in our input.
In this part agent gets 0.1 reward every step he lived.
Our first try was dividing each distance by biggest measured in that step.

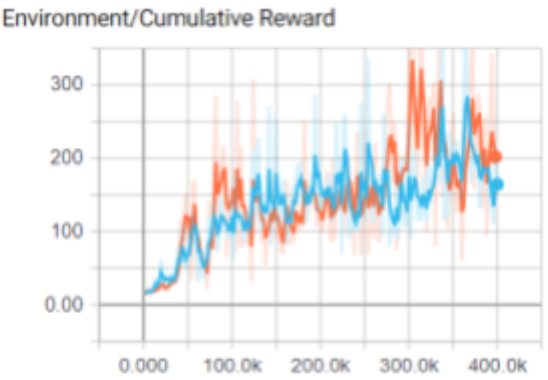


It turned to not work very well. Agent had problem with evaluating how far exactly he is and when he need to turn.
Then we tried dividing each distance by number bigger than maximum possible distance.



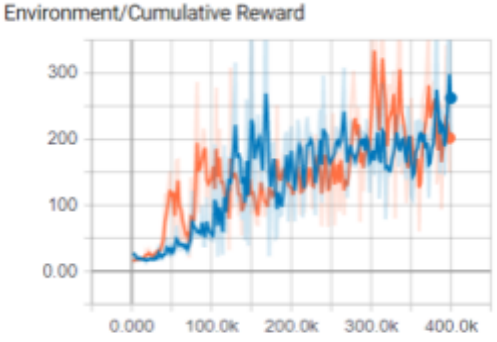


As we can see on the graph above, it worked much better, but we wanted to try other possible ways to solve this problem. Our next try used hyperbolic tangent. Because this function for numbers around 6 already return almost 1, we needed to divide each input by some constant. it worked very similarly to dividing by constant.
Our next approach was to for each distance get its inverse and ignore each that originally was smaller than 1 making it 1 instead. It was first distance based agent that tried going inside. But as we can see on the graph below it was little worse than previous one.



blue - reverse and ignore small distances
orange - tanh

Lastly, we tried giving agent combination of reversed and not reversed input: we gave it one vector of reversed values with 1 in place of each value smaller than 1 and second with normal values with 1 in place every value bigger than 1. This time agent also tried to go inside, but there was less space between traces and it turned to be better than previous approaches.

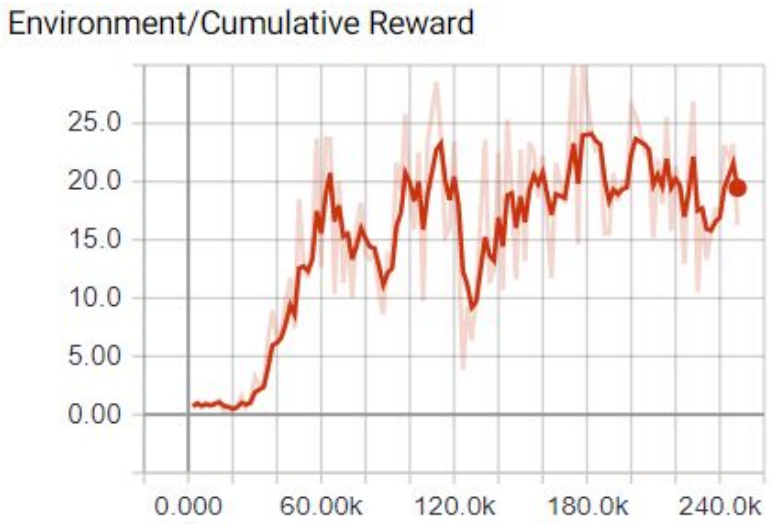


blue - reversed and nor reversed
orange - tanh

Based on this experiments, we decided to use combination of reversed and not reversed distances.

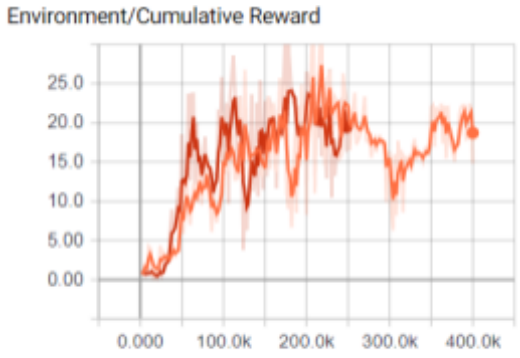
Combined learning

We decided to combine two approaches, and give agent distances as input for local knowledge and camera for global knowledge. For distances we used combination of reversed and not reversed distances described in previous paragraph. We chose to give one, big camera of size 16.
Next, we tried to find better hyperparameters for our neural network.
We started experiments with learning rate = 6e-3, gamma = 0.99 (discount factor for future rewards) and hidden units = 512. For reward agent gets 0.01 for each step and loose 1 for death.
Results was weaker than using only distances, but similar or even better than using only cameras. Unfortunately, in some runs agent decided to constantly turn and ride in circles.



We started with changing gamma. When we reduced it to less than 0.98 agent always ridden in circles. Changing it to 0.98 gaved worse results and bigger entropy.
Changing it to bigger gaved smaller entropy and little better result.

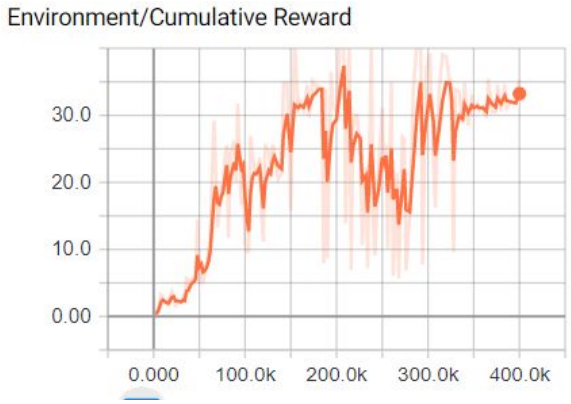
Afterwards, we tried changing learning rate and gamma at the same time, but it gave very similar results.
Until now, we used continuous action space - neural net gaved us one number from -1 to 1, and if it was smaller than -0.33, we interpreted it as left turn and result bigger than 0.33 we interpreted as right turn. Now we decided to try discrete space actions. With similar hyperparameters results was similar what we can see on the left graph below.



orange - discrete
dark red - continuous

orange - learning rate = 6e-3
pink - learning rate = 7e-3
blue - learning rate = 8e-3

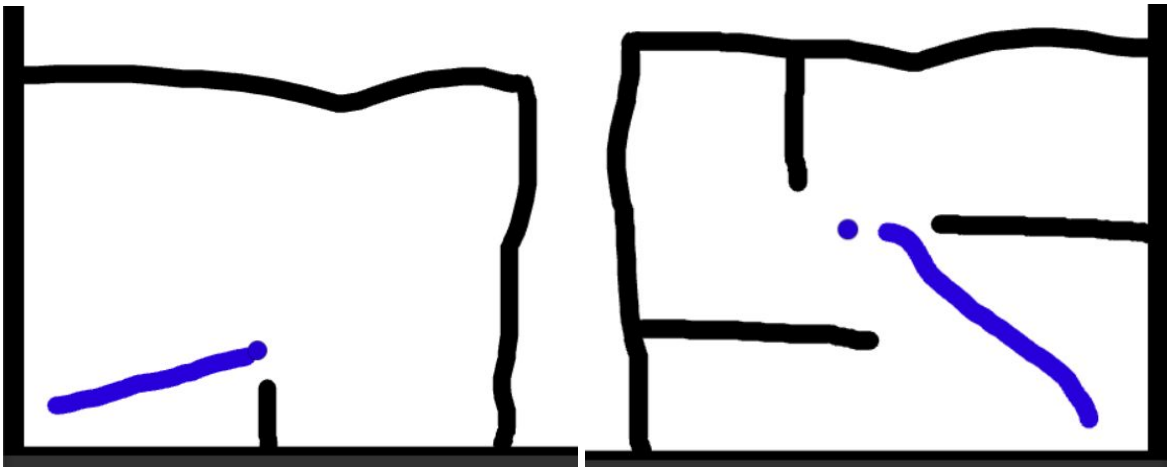
Next, we tried combine discrete actions and higher learning rate. Results are on the right graph above. As we can see, learning rate equal to 7e-3 gaved much better results than previous setting, but it was on the edge of stable learning parameters, because with higher learning rate agent wasn't doing anything.
With those parameters we got best agent of our singleplayer experiments. We can see it's learning on the graph below.

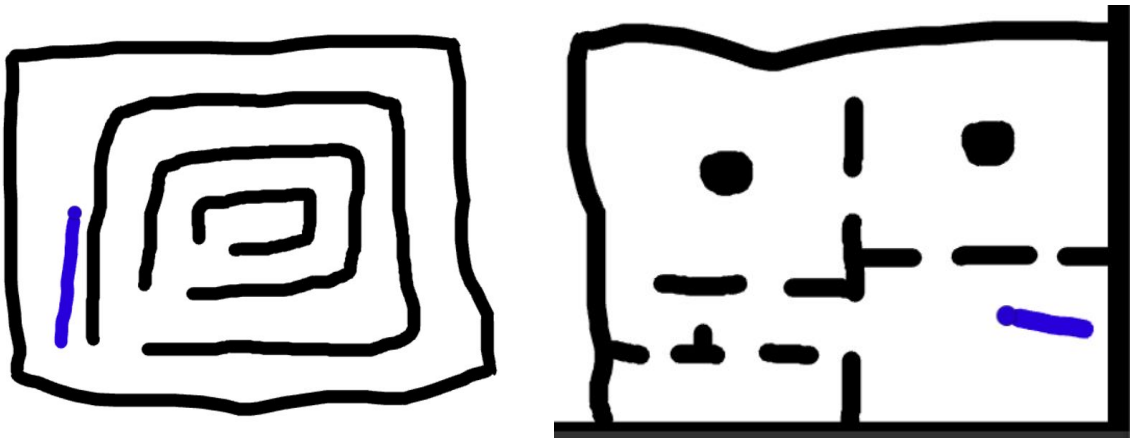


Our next try was to change gamma with discrete actions. First combination - gamma = 0.995 and learning rate = 7e-3 gaved very bad and unstable results. Our other try for gamma = 0.995 and learning rate = 6e-3 gaved very similar result to average result for gamma = 0.99 and learning rate = 7e-3.
Last part of experiments with hyperparameters was attempts with more hidden units. Unfortunately, all of our experiments with 512 units per hidden layer gave worse results than with 256 units per hidden layer.
Summarizing, we got best results with discrete action space, learning rate = 7e-3, gamma = 0.99 and 256 units per hidden layer.

Custom maps

With best possible agent so far received from combined learning approach we decided to try enforce some desired behaviours. We made 4 maps. During the learning process agent learned playing on every map for 50k steps. There was 50% chance that map became symmetric along Y axis to improve learning process (so in the end there were 8 maps). Each map had established starting point .

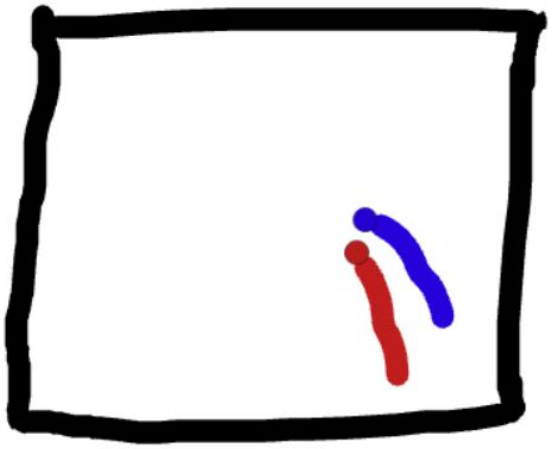




Such training resulted in agent that actually was capable to fill board quite well.

Competitive learning

We loaded agents trained on custom maps to learn to fight enemies. The agents was not able to learn strategy of killing other. It preferred just riding as long as possible. To force killing we added extra rewards if enemy was killed by our newest part (+1) or second one (+0.5). Furthermore we trained agents on small maps first. It did not bring expected result. We tried also making asymmetric map where one agent was further than the other giving him opportunity to kill the enemy.



Sometimes it were using given opportunity but agent did not perfected that and prefered more safe play. Some agents where using grayed colors other RGB ones. Results where indistinguishable. Final trained agents where weak and no match for human players.

Conclusion

ML-Agents library combined with PPO algorithm gives satisfying results when it comes to simple non competitive tasks. In our case, good results were achieved very fast after some obvious improvements. It shows that it can be really great tool for learning simple agents in various games with no much time needed. When it comes to harder tasks ideal solution would be combining imitation learning with reinforcement learning. Sadly plugin we used did not support such option. Behaviours we tried to enforce could be easily and fast learned from human player. With agents learned that way possible further improvements could be done with reinforcement learning.