

---

## Arnold

Arnold is a PyTorch implementation of the agent presented in *Playing FPS Games with Deep Reinforcement Learning* (<https://arxiv.org/abs/1609.05521>), and that won the 2017 edition of the *ViZDoom AI Competition*.



### This repository contains:

- The source code to train DOOM agents
- A package with 17 selected maps that can be used for training and evaluation
- 5 pretrained models that you can visualize and play against, including the ones that won the ViZDoom competition

## Installation

**Dependencies** Arnold was tested successfully on Mac OS and Linux distributions. You will need: - Python 2/3 with NumPy and OpenCV - PyTorch - ViZDoom

Follow the instructions on <https://github.com/mwydmuch/ViZDoom> to install ViZDoom. Be sure that you can run `import vizdoom` in Python from any directory. To do so, you can either install the library with `pip`, or compile it, then move it to the `site-packages` directory of your Python installation, as explained here: <https://github.com/mwydmuch/ViZDoom/blob/master/doc/Quickstart.md>.

---

## Code structure

1	.	—	
2	pretrained		# Examples of pretrained models  —
3	resources	—	
4	freedoom2.wad		# DOOM resources file (containing all textures)  —
5	scenarios		# Folder containing all scenarios  —
6	full_deathmatch.wad		# Scenario containing all deathmatch maps  —
7	health_gathering.wad		# Simple test scenario  —
8	...	—	
9	src		# Source files  —
10	doom		# Game interaction / API / scenarios  —
11	model		# DQN / DRQN implementations  —
12	trainer		# Folder containing training scripts  —
13	arnold.py		# Main file  —
14	README.md		

## Scenarios / Maps

### Train a model

There are many parameters you can tune to train a model.

```
1 python arnold.py
2
3 ## General parameters about the game
4 --freedoom "true"           # use freedoom resources
5 --height 60                 # screen height
6 --width 108                 # screen width
7 --gray "false"              # use grayscale screen
8 --use_screen_buffer "true"  # use the screen buffer (what the
    player sees)
9 --use_depth_buffer "false"  # use the depth buffer
10 --labels_mapping ""         # use extra feature maps for specific
    objects
11 --game_features "target,enemy" # game features prediction (auxiliary
    tasks)
12 --render_hud "false"        # render the HUD (status bar in the
    bottom of the screen)
13 --render_crosshair "true"   # render crosshair (targeting aid in
    the center of the screen)
14 --render_weapon "true"      # render weapon
15 --hist_size 4               # history size
16 --frame_skip 4              # frame skip (1 = keep every frame)
```

---

```

17
18 ## Agent allowed actions
19 --action_combinations "attack+move_lr;turn_lr;move_fb" # agent allowed
    actions
20 --freelook "false" # allow the agent to look up and down
21 --speed "on" # make the agent run
22 --crouch "off" # make the agent crouch
23
24 ## Training parameters
25 --batch_size 32 # batch size
26 --replay_memory_size 1000000 # maximum number of frames in the
    replay memory
27 --start_decay 0 # epsilon decay iteration start
28 --stop_decay 1000000 # epsilon decay iteration end
29 --final_decay 0.1 # final epsilon value
30 --gamma 0.99 # discount factor gamma
31 --dueling_network "false" # use a dueling architecture
32 --clip_delta 1.0 # clip the delta loss
33 --update_frequency 4 # DQN update frequency
34 --dropout 0.5 # dropout on CNN output layer
35 --optimizer "rmsprop,lr=0.0002" # network optimizer
36
37 ## Network architecture
38 --network_type "dqn_rnn" # network type (dqn_ff / dqn_rnn)
39 --recurrence "lstm" # recurrent network type (rnn / gru /
    lstm)
40 --n_rec_layers 1 # number of layers in the recurrent
    network
41 --n_rec_updates 5 # number of updates by sample
42 --remember 1 # remember all frames during
    evaluation
43 --use_bn "off" # use BatchNorm when processing the
    screen
44 --variable_dim "32" # game variables embeddings dimension
45 --bucket_size "[10, 1]" # bucket game variables (typically
    health / ammo)
46 --hidden_dim 512 # hidden layers dimension
47
48 ## Scenario parameters (these parameters will differ based on the
    scenario)
49 --scenario "deathmatch" # scenario
50 --wad "full_deathmatch" # WAD file (scenario file)
51 --map_ids_train "2,3,4,5" # maps to train the model
52 --map_ids_test "6,7,8" # maps to test the model
53 --n_bots 8 # number of enemy bots
54 --randomize_textures "true" # randomize walls / floors / ceilings
    textures during training
55 --init_bots_health 20 # reduce initial life of enemy bots (
    helps a lot when using pistol)
56
57 ## Various

```

---

---

```
58 --exp_name new_train          # experiment name
59 --dump_freq 200000            # periodically dump the model
60 --gpu_id -1                   # GPU ID (-1 to run on CPU)
```

Once your agent is trained, you can visualize it by running the same command, and using the following extra arguments:

```
1 --visualize 1                  # visualize the model (render the
    screen)
2 --evaluate 1                   # evaluate the agent
3 --manual_control 1            # manually make the agent turn about
    when it gets stuck
4 --reload PATH                  # path where the trained agent was
    saved
```

Here are some examples of training commands for 3 different scenarios:

**Defend the center** In this scenario the agent is in the middle of a circular map. Monsters are regularly appearing on the sides and are walking towards the agent. The agent is given a pistol and limited ammo, and must turn around and kill the monsters before they reach it. The following command trains a standard DQN, that should reach the optimal performance of 56 frags (the number of bullets in the pistol) in about 4 million steps:

```
1 python arnold.py --scenario defend_the_center --action_combinations "
    turn_lr+attack" --frame_skip 2
```

**Health gathering** In this scenario the agent is walking on lava, and is losing health points at each time step. The agent has to move and collect as many health pack as possible in order to survive. The objective is to survive the longest possible time.

```
1 python arnold.py --scenario health_gathering --action_combinations "
    move_fb;turn_lr" --frame_skip 5
```

This scenario is very easy and the model quickly reaches the maximum survival time of 2 minutes ( $35 * 120 = 4200$  frames). The scenario also provides a **supreme** mode, in which the map is more complicated and where the health packs are much harder to collect:

```
1 python arnold.py --scenario health_gathering --action_combinations "
    move_fb;turn_lr" --frame_skip 5 --supreme 1
```

In this scenario, the agent takes about 1.5 million steps to reach the maximum survival time (but often dies before the end).

---

**Deathmatch** In this scenario, the agent is trained to fight against the built-in bots of the game. Here is a command to train the agent using game features prediction (as described in [1]), and a DRQN:

```
1 python arnold.py --scenario deathmatch --wad deathmatch_rocket --
   n_bots 8 \
2 --action_combinations "move_fb;move_lr;turn_lr;attack" --frame_skip 4 \
3 --game_features "enemy" --network_type dqn_rnn --recurrence lstm --
   n_rec_updates 5
```

## Pretrained models

**Defend the center / Health gathering** We provide a pretrained model for each of these scenarios. You can visualize them by running:

```
1 ./run.sh defend_the_center
```

or

```
1 ./run.sh health_gathering
```

**Visual Doom AI Competition 2017** We release the two agents submitted to the first and second tracks of the ViZDoom AI 2017 Competition. You can visualize them playing against the built-in bots using the following commands:

### Track 1 - Arnold vs 10 built-in AI bots

```
1 ./run.sh track1 --n_bots 10
```

### Track 2 - Arnold vs 10 built-in AI bots - Map 2

```
1 ./run.sh track2 --n_bots 10 --map_id 2
```

### Track 2 - 4 Arnold playing against each other - Map 3

```
1 ./run.sh track2 --n_bots 0 --map_id 3 --n_agents 4
```

We also trained an agent on a single map, using a same weapon (the SuperShotgun). This agent is extremely difficult to beat.

### Shotgun - 4 Arnold playing against each other

```
1 ./run.sh shotgun --n_bots 0 --n_agents 4
```

### Shotgun - 3 Arnold playing against each other + 1 human player (to play against the agent)

```
1 ./run.sh shotgun --n_bots 0 --n_agents 3 --human_player 1
```

---

## References

If you found this code useful, please consider citing:

[1] G. Lample\* and D.S. Chaplot\*, *Playing FPS Games with Deep Reinforcement Learning*

```
1 @inproceedings{lample2017playing,  
2   title={Playing FPS Games with Deep Reinforcement Learning.},  
3   author={Lample, Guillaume and Chaplot, Devendra Singh},  
4   booktitle={Proceedings of AAAI},  
5   year={2017}  
6 }
```

[2] D.S. Chaplot\* and G. Lample\*, *Arnold: An Autonomous Agent to Play FPS Games*

```
1 @inproceedings{chaplot2017arnold,  
2   title={Arnold: An Autonomous Agent to Play FPS Games.},  
3   author={Chaplot, Devendra Singh and Lample, Guillaume},  
4   booktitle={Proceedings of AAAI},  
5   year={2017},  
6   Note={Best Demo award}  
7 }
```

## Acknowledgements

We acknowledge the developers of *ViZDoom* for constant help and support during the development of this project. Some of the maps and wad files have been borrowed from the *ViZDoom git repository*. We also thank the members of the *ZDoom* community for their help with the Action Code Scripts (ACS).