

Let's play!

The only ML presentation you will ever need

Adam Dudczak, twitter/@maneo



About me

Search@Allegro.pl

Java since 2005, Python since 2018

former Poznań JUG and Polish JUG co-leader

one of the brave people behind GeeCON

Where to start?

Where to start?



Shmup!

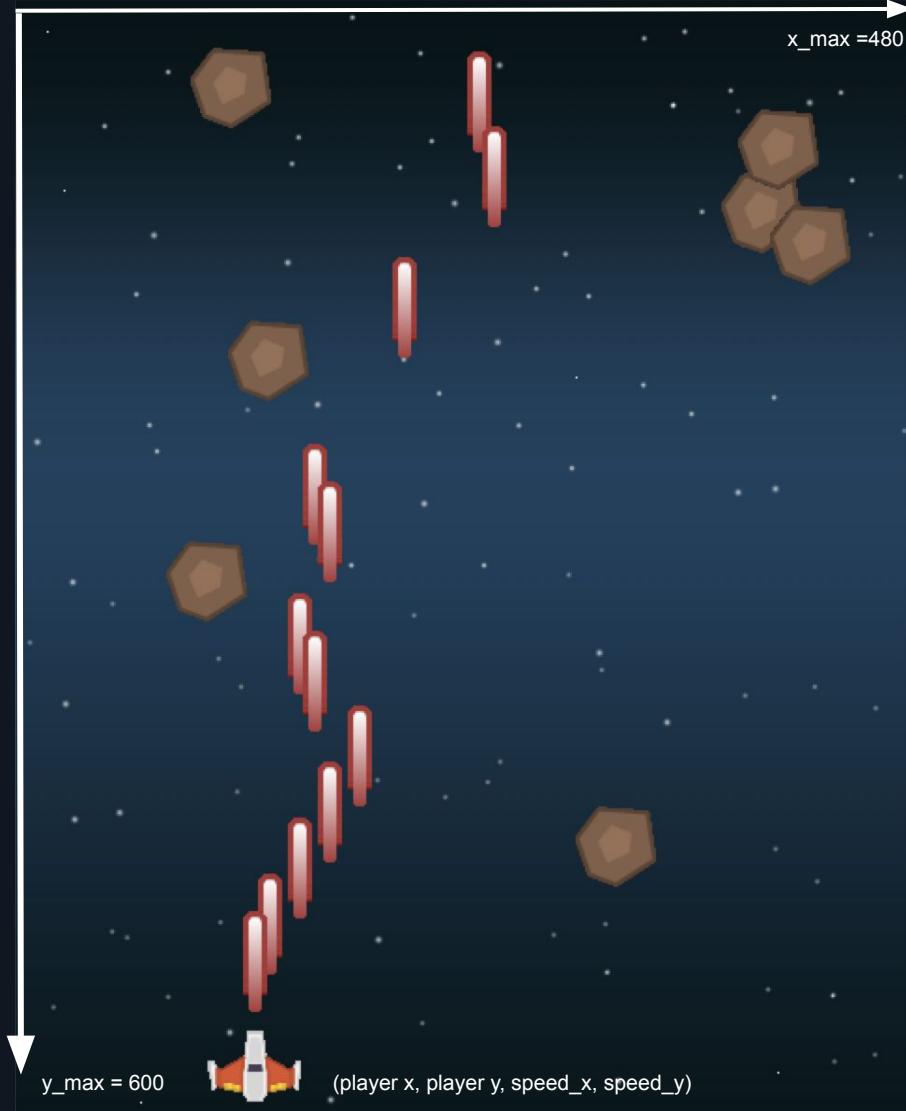
```
14
15 import pygame
16
17 img_dir = path.join(path.dirname(__file__), '../img')
18
19 WIDTH = 480
20 HEIGHT = 600
21 FPS = 60
22 MOBS_SIZE = 8
23
24 # define colors
25 WHITE = (255, 255, 255)
26 BLACK = (0, 0, 0)
27 RED = (255, 0, 0)
28 GREEN = (0, 255, 0)
29 BLUE = (0, 0, 255)
30 YELLOW = (255, 255, 0)
31
32
33 # initialize pygame and create window
34 pygame.init()
35 pygame.mixer.init()
36 screen = pygame.display.set_mode((WIDTH, HEIGHT))
37 pygame.display.set_caption("Shmup!")
38 clock = pygame.time.Clock()
39
```

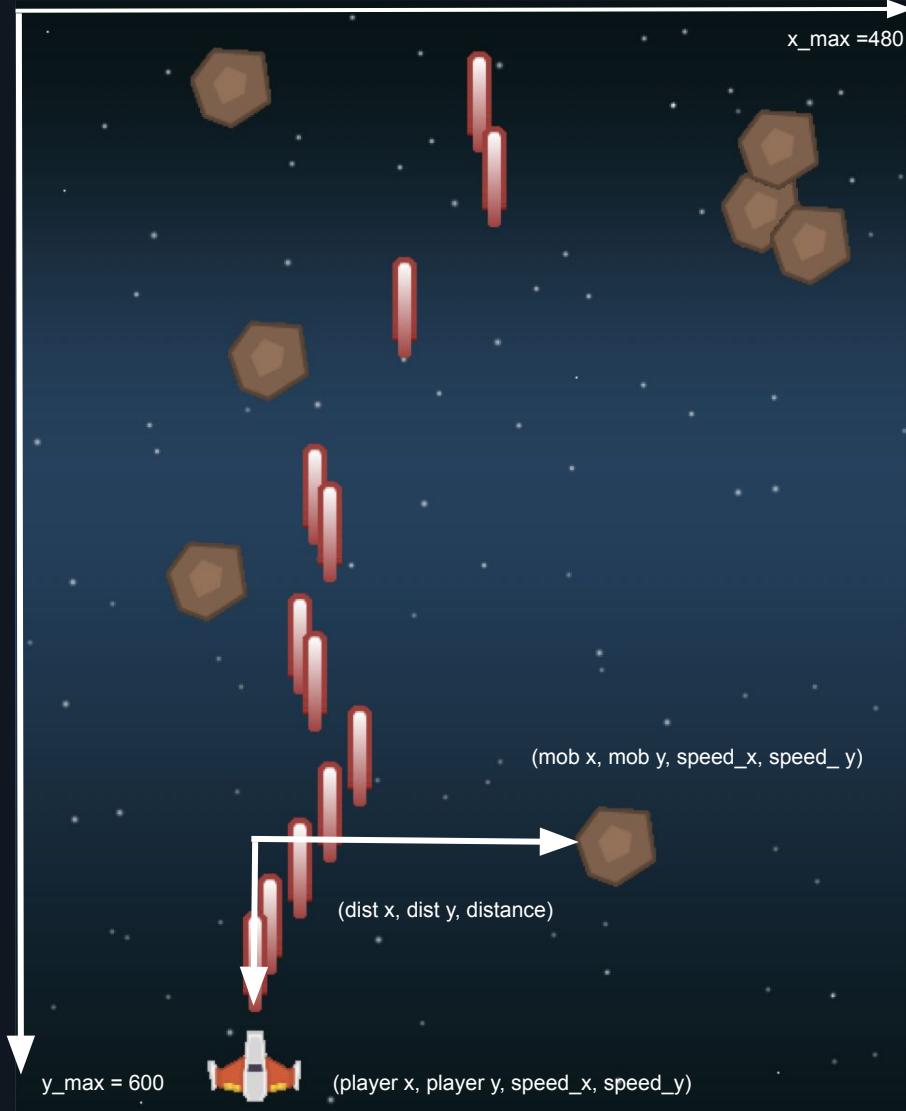
```
147 # Load all game graphics
148 background = pygame.image.load(path.join(img_dir, "starfield.png")).convert()
149 background_rect = background.get_rect()
150 player_img = pygame.image.load(path.join(img_dir, "playerShip1_orange.png")).convert()
151 meteor_img = pygame.image.load(path.join(img_dir, "meteorBrown_med1.png")).convert()
152 bullet_img = pygame.image.load(path.join(img_dir, "laserRed16.png")).convert()
153
154 all_sprites = pygame.sprite.Group()
155 mobs = pygame.sprite.Group()
156 bullets = pygame.sprite.Group()
157 player = Player()
158 all_sprites.add(player)
159 for i in range(MOBS_SIZE):
160     m = Mob()
161     all_sprites.add(m)
162     mobs.add(m)
```

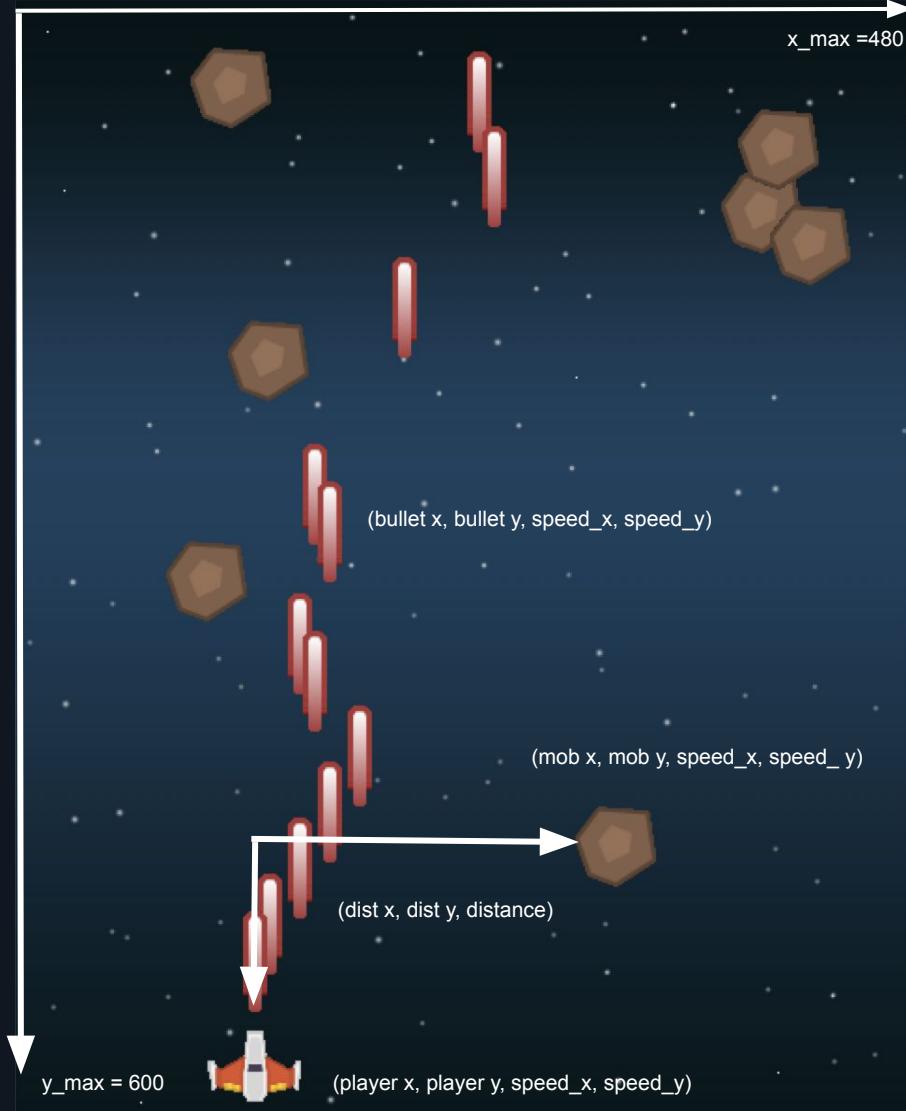
Shmup!

```
163  
164     # Game loop  
165     game_start_time = time.time()  
166  
167     running = True  
168     while running:  
169         # keep loop running at the right speed  
170         clock.tick(FPS)  
171  
172         # Process input (events)  
173         for event in pygame.event.get():  
174             # check for closing window  
175             if event.type == pygame.QUIT:  
176                 running = False  
177             elif event.type == pygame.KEYDOWN:  
178                 if event.key == pygame.K_SPACE:  
179                     player.shoot()  
180  
181         # Update  
182         all_sprites.update()  
183  
184         # check to see if a bullet hit a mob  
185         hits = pygame.sprite.groupcollide(mobs, bullets, True, True)  
186         for hit in hits:  
187             m = Mob()  
188             all_sprites.add(m)  
189             mobs.add(m)  
190  
191         # check to see if a mob hit the player  
192         hits = pygame.sprite.spritecollide(player, mobs, False)  
193         if hits:  
194             running = False  
195  
196         # Draw / render  
197         screen.fill(BLACK)  
198         screen.blit(background, background_rect)  
199         all_sprites.draw(screen)  
200         # *after* drawing everything, flip the display  
201         pygame.display.flip()  
202  
203     end = time.time()  
204     pygame.quit()
```

What data is relevant?







**game state(t) = { player_x, speed_x,
mob₁(x, y, dist_x, dist_y, speed_x, speed_y),
..., mob₈(...) }**

**game state(t) = { player_x, speed_x,
mob₁(x, y, dist_x, dist_y, speed_x, speed_y),
..., mob₈(...) }**

action = { left, right, fire, nothing, left+fire, right+fire }

```
class Player(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.transform.scale(player_img, (50, 38))
        self.image.set_colorkey(BLACK)
        self.rect = self.image.get_rect()
        self.rect.centerx = WIDTH / 2
        self.rect.bottom = HEIGHT - 10
        self.speedx = 0

    state_vector_size = 2

    def dump_state_vector(self):
        return [self.speedx, self.rect.centerx]
```

```
class Mob(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = meteor_img
        self.image.set_colorkey(BLACK)
        self.rect = self.image.get_rect()
        self.rect.x = random.randrange(WIDTH - self.rect.width)
        self.rect.y = random.randrange(-100, -40)
        self.speedy = random.randrange(1, 8)
        self.speedx = random.randrange(-3, 3)

    state_vector_size = 7

    def dump_state(self, player) -> dict:
        state = dict()
        player_x = player.rect.centerx
        player_y = player.rect.centery
        mob_x = self.rect.centerx
        mob_y = self.rect.centery

        state["speedx"] = self.speedx
        state["speedy"] = self.speedy
        state["distance"] = round(sqrt((player_x - mob_x) * (player_x - mob_x)
                                       + (player_y - mob_y) * (player_y - mob_y)))
        state["dist_x"] = player_x - mob_x
        state["dist_y"] = player_y - mob_y

        state['mob_x'] = mob_x
        state['mob_y'] = mob_y

    return state
```

Getting training data

```
164 # Game loop
165 score = 0
166 game_start_time = time.time()
167
168 running = True
169 while running:
170     # keep loop running at the right speed
171     clock.tick(FPS)
172     was_shooting = False
173
174     # Process input (events)
175     for event in pygame.event.get():
176         # check for closing window
177         if event.type == pygame.QUIT:
178             running = False
179         elif event.type == pygame.KEYDOWN:
180             if event.key == pygame.K_SPACE:
181                 player.shoot()
182                 was_shooting = True
183
184     # Update
185     all_sprites.update()
186
187     # check to see if a bullet hit a mob
188     hits = pygame.sprite.groupcollide(mobs, bullets, True, True)
189     for hit in hits:
190         m = Mob()
191         all_sprites.add(m)
192         mobs.add(m)
193         score += 1
194
195     # check to see if a mob hit the player
196     hits = pygame.sprite.spritecollide(player, mobs, False)
197     if hits:
198         running = False
199
200     dump_as_vector(mobs, player, bullets, pygame, was_shooting)
201
202     # Draw / render
203     screen.fill(BLACK)
204     screen.blit(background, background_rect)
205     all_sprites.draw(screen)
206     # *after* drawing everything, flip the display
207     pygame.display.flip()
208
209 end = time. time()
210 print("time: {} sec, score: {}".format(round(end - game_start_time), score))
211
212 pygame.quit()
```

Getting training data

```
python shmup-train.py > training.log
```



```
python shmup-train.py > training.log
```

training.log

```
232,-13,637,5,1,-118,636,5,0,-158,667,7,1,235,638,5,-3,-113,670,1,0,136,657,6,-2,112,667,6,-2,-95,636,7,-2,0  
232,-14,632,5,1,-118,631,5,0,-159,660,7,1,238,633,5,-3,-113,669,1,0,138,651,6,-2,114,661,6,-2,-93,629,7,-2,4  
232,-15,627,5,1,-118,626,5,0,-160,653,7,1,241,628,5,-3,-113,668,1,0,140,645,6,-2,116,655,6,-2,-91,622,7,-2,5  
240,-8,622,5,1,-110,621,5,0,-153,646,7,1,252,623,5,-3,-105,667,1,0,150,639,6,-2,126,649,6,-2,-81,615,7,-2,1  
240,-9,617,5,1,-110,616,5,0,-154,639,7,1,255,618,5,-3,-105,666,1,0,152,633,6,-2,128,643,6,-2,-79,608,7,-2,5  
248,-2,612,5,1,-102,611,5,0,-147,632,7,1,266,613,5,-3,-97,665,1,0,162,627,6,-2,138,637,6,-2,-69,601,7,-2,1  
248,-3,607,5,1,-102,606,5,0,-148,625,7,1,269,608,5,-3,-97,664,1,0,164,621,6,-2,140,631,6,-2,-67,594,7,-2,5
```

game state(t) -> ML model -> action

```
In [27]: features = ["f" + str(i) for i in range(0,26)]
label = ["action"]

headers = label + features

df = pd.read_csv('train.csv',
                  sep = ',',
                  header = None,
                  names = headers)
df
```

Out[27]:

	action	f0	f1	f2	f3	f4	f5	f6	f7	f8	...	f16	f17	f18	f19	f20	f21	f22	f23	f24	f25
0	3	0	240	663	-3	5	631	1	2	656	...	4	677	-2	1	666	-3	3	675	1	4
1	3	0	240	657	-3	5	629	1	2	652	...	4	676	-2	1	663	-3	3	671	1	4
2	3	0	240	652	-3	5	627	1	2	649	...	4	676	-2	1	659	-3	3	667	1	4
3	3	0	240	646	-3	5	625	1	2	646	...	4	675	-2	1	656	-3	3	663	1	4
4	3	0	240	641	-3	5	623	1	2	642	...	4	674	-2	1	652	-3	3	659	1	4
...	
755	3	0	112	551	1	5	628	-2	3	609	...	3	656	0	1	674	-2	2	680	1	1
756	3	0	112	546	1	5	625	-2	3	609	...	3	655	0	1	672	-2	2	679	1	1
757	3	0	112	541	1	5	621	-2	3	608	...	3	654	0	1	670	-2	2	678	1	1
758	3	0	112	537	1	5	618	-2	3	607	...	3	653	0	1	668	-2	2	678	1	1
759	3	0	112	532	1	5	615	-2	3	606	...	3	652	0	1	665	-2	2	677	1	1

760 rows × 27 columns

```
In [14]: X_train, X_test, y_train, y_test = train_test_split(X_train_all, y_train_all, test_size=0.25)
classifier = LogisticRegression(solver='lbfgs', max_iter=20000, multi_class="auto")
classifier.fit(X_train, y_train)
```

```
In [15]: evaluate_model(X_train, y_train, X_test, y_test, classifier)

Training accuracy: 0.78, evaluation accuracy: 0.71
```

```
In [28]: pickle.dump(mlp_model, open("ai_model_logit.pkl", "wb"))
```

Let's plug our model in!

```
ai_model = pickle.load(open("ai_model_logit.pkl", "rb"))
```

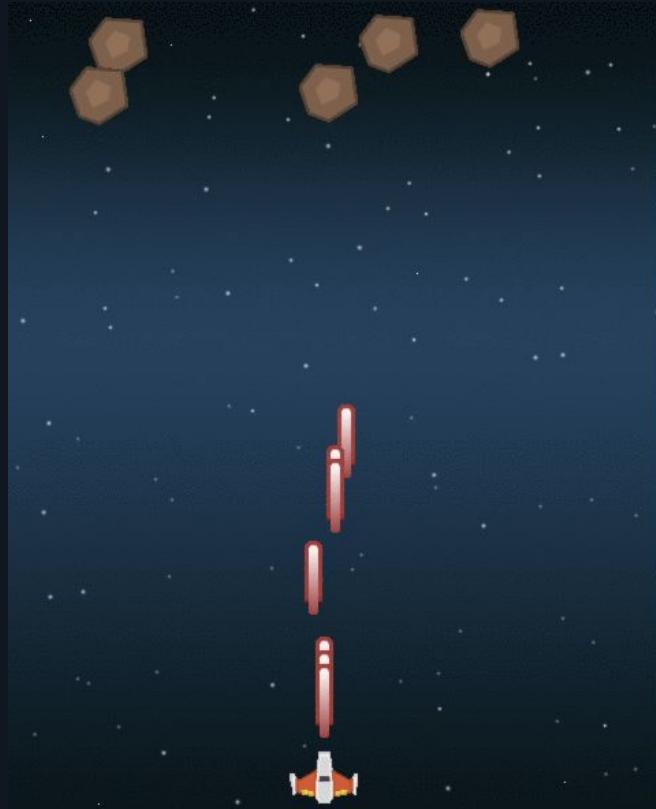
```
def ai(game_state):
    return ai_model.predict(np.array(game_state).reshape(1, -1))
```

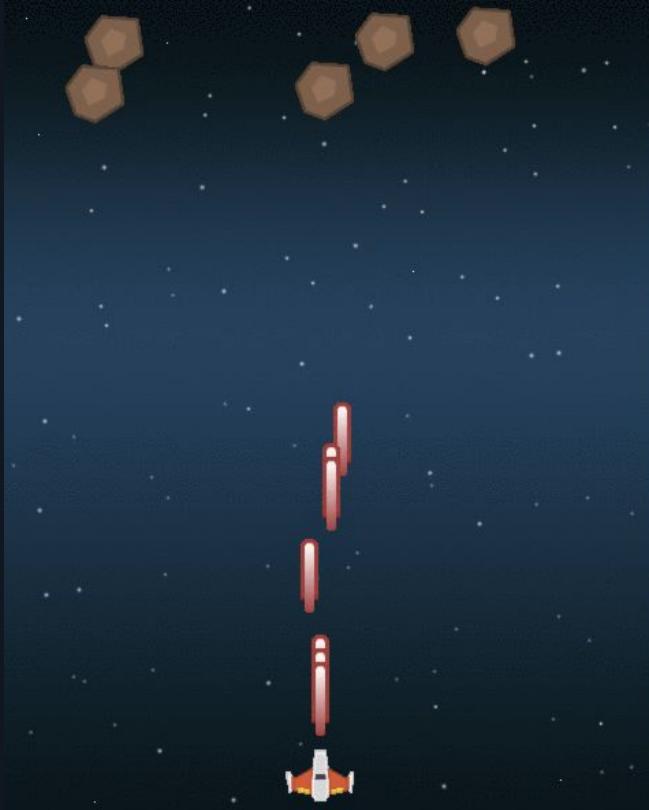
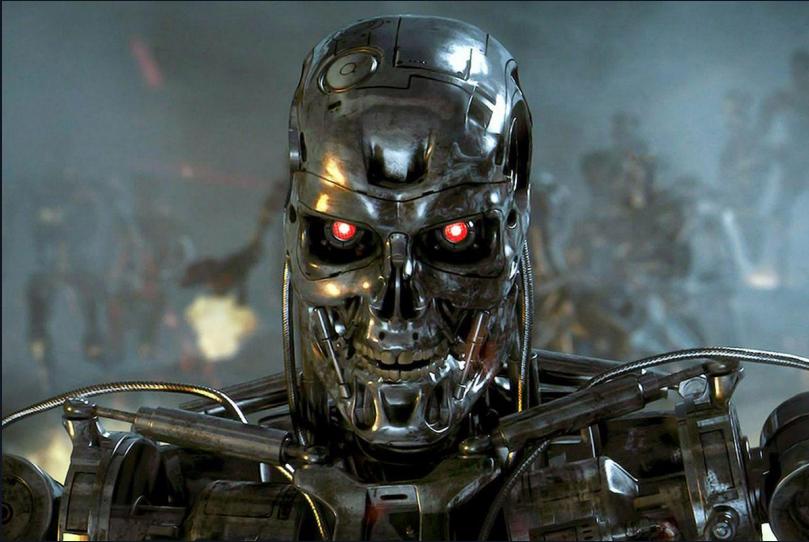
```
# Game loop
running = True
while running:
    # keep loop running at the right speed
    clock.tick(FPS)
```

```
    action = ai(get_game_state(mobs, player, bullets))
```

```
    # Process input (events)
    for event in pygame.event.get():
        # check for closing window
        if event.type == pygame.QUIT:
            running = False
```

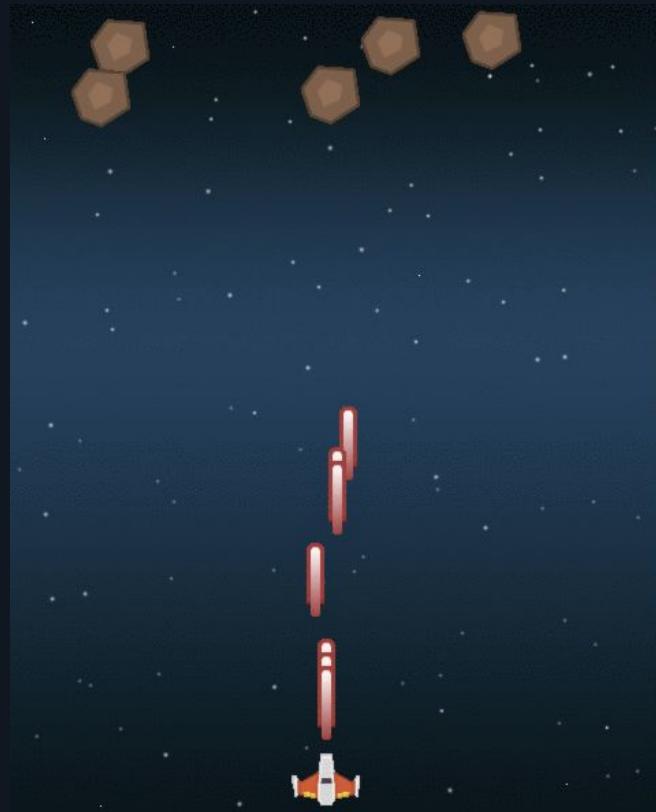
```
    if action == 2 or action == 4 or action == 5:
        player.shoot()
    else:
        player.update_with_action(action)
```







Logistic reg. (10 games, avg score: 3.2)



Random (10 games, avg score: 27)

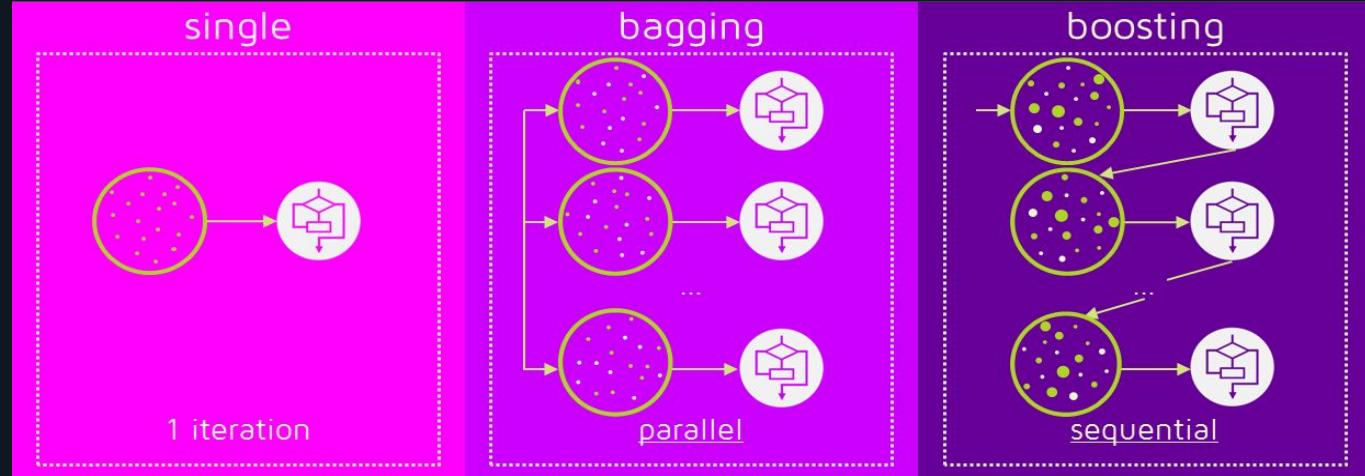
```
In [18]: from sklearn.ensemble import RandomForestClassifier
forest_model = RandomForestClassifier(random_state=1, n_estimators=1000)
forest_model.fit(X_train, y_train)

Out[18]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                                 max_depth=None, max_features='auto', max_leaf_nodes=None,
                                 min_impurity_decrease=0.0, min_impurity_split=None,
                                 min_samples_leaf=1, min_samples_split=2,
                                 min_weight_fraction_leaf=0.0, n_estimators=1000,
                                 n_jobs=None, oob_score=False, random_state=1, verbose=0,
                                 warm_start=False)

In [20]: evaluate_model(X_train, y_train, X_test, y_test, forest_model)

Training accuracy: 1.00, evaluation accuracy: 1.00

In [21]: pickle.dump(forest_model, open("ai_model_forest.pkl", "wb"))
```



source: <https://quantdare.com/what-is-the-difference-between-bagging-and-boosting/>

```
In [18]: from sklearn.ensemble import RandomForestClassifier  
forest_model = RandomForestClassifier(random_state=1, n_estimators=1000)  
forest_model.fit(X_train, y_train)
```

```
Out[18]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',  
                                 max_depth=None, max_features='auto', max_leaf_nodes=None,  
                                 min_impurity_decrease=0.0, min_impurity_split=None,  
                                 min_samples_leaf=1, min_samples_split=2,  
                                 min_weight_fraction_leaf=0.0, n_estimators=1000,  
                                 n_jobs=None, oob_score=False, random_state=1, verbose=0,  
                                 warm_start=False)
```

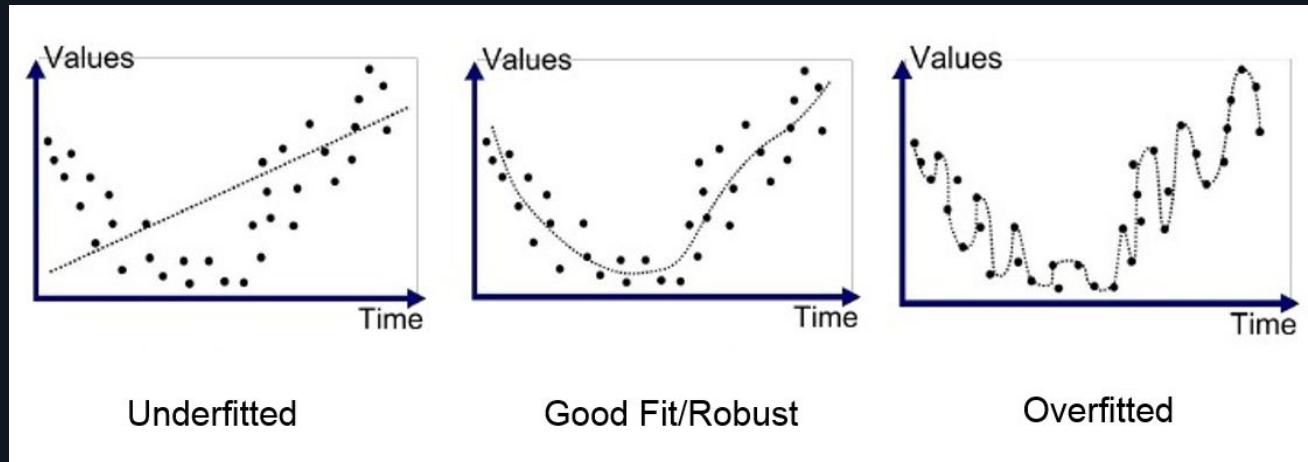
```
In [20]: evaluate_model(X_train, y_train, X_test, y_test, forest_model)
```

Training accuracy: 1.00, evaluation accuracy: 1.00

```
In [21]: pickle.dump(forest_model, open("ai_model_forest.pkl", "wb"))
```







SOURCE: <https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76>

**#1. Simple models are not enough for this representation
(and possibly this problem)**

**#1. Simple models are not enough for this representation
(and possibly this problem)**

#2. Too few data for more complex models

**#1. Simple models are not enough for this representation
(and possibly this problem)**

#2. Too few data for more complex models

#3. Model accuracy may not be very helpful here

Let's get more.... data

**game state(t) = { game state(t-1), player_x, speed_x,
mob₁(x, y, dist_x, dist_y, speed_x, speed_y),
..., mob₈(...) }**

action = { left, right, fire, nothing, left+fire, right+fire }

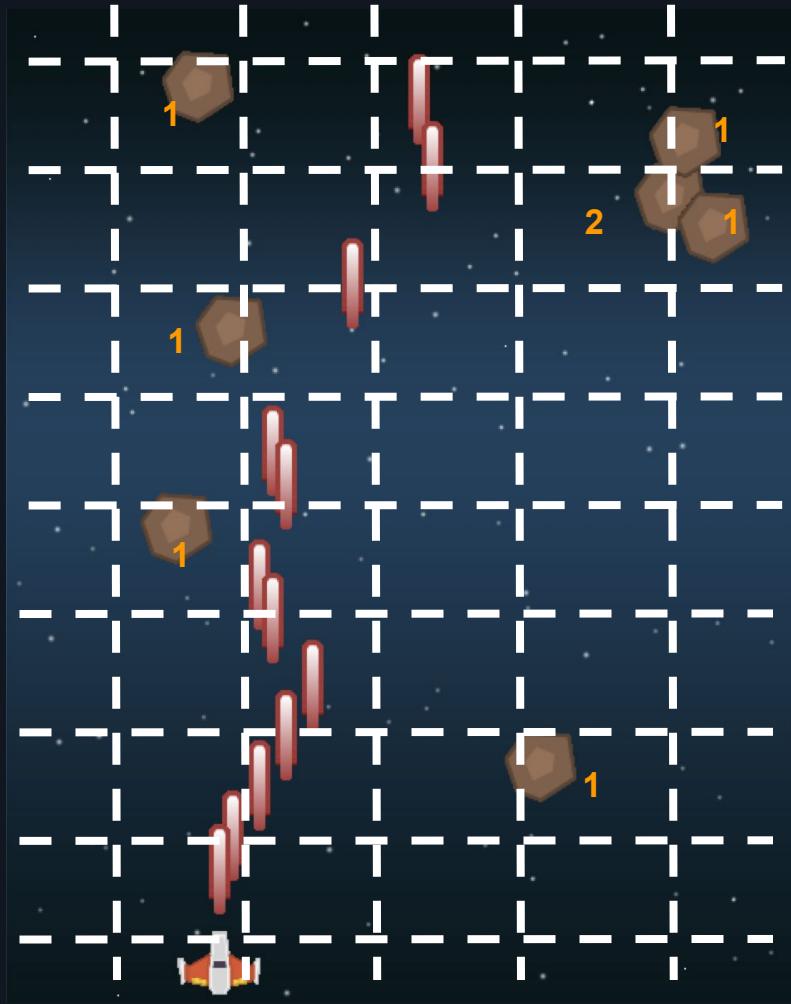
Model	Score (avg, 10 attempts)
XGBClassifier	0.65
MLPClassifier (neural network)	1.83
Logistic Regression	13

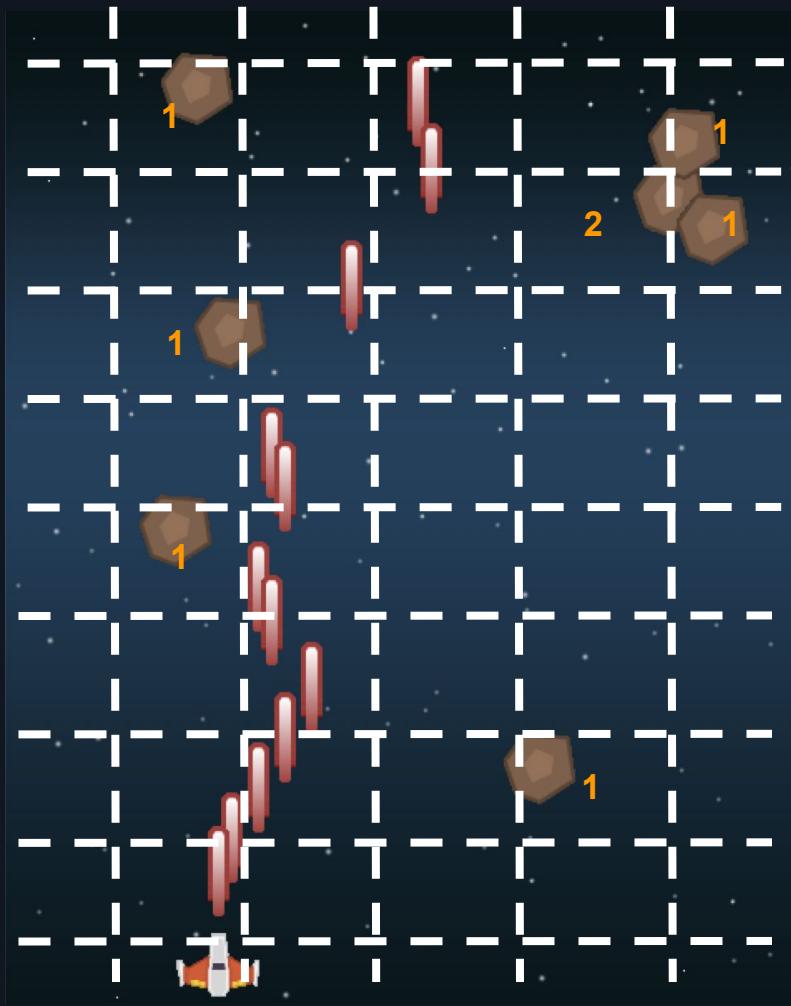


**#4. Increasing complexity of state representation may
not be the best idea**

#5. Premature optimization is the root of all ML evil







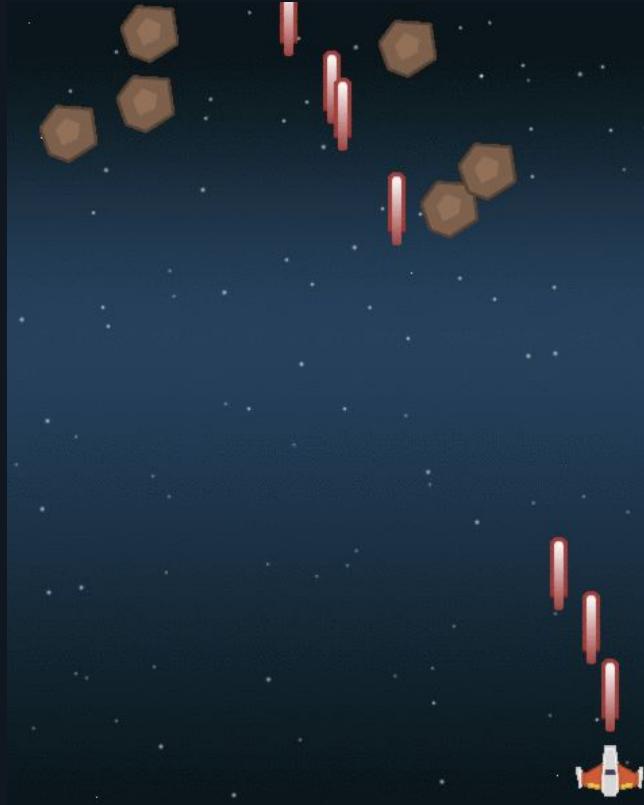
```
state.append(player.dump_state_vector()[1] % 6)

mob_positions = [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]

# mob_vector_size = self.mob_vector_size
for mob in mobs.sprites():
    mob_state = mob.dump_state(player)
    pos_x = mob_state['mob_x'] % 6
    pos_y = mob_state['mob_y'] % 10
    mob_positions[pos_x][pos_y] = mob_positions[pos_x][pos_y] + 1

for column in mob_positions:
    state.extend(column)
```

Model	Score (avg 10 attempts)
XGBClassifier	13.7
MLPClassifier (neural network)	43.3
Logistic Regression	16
RandomForest	0.1



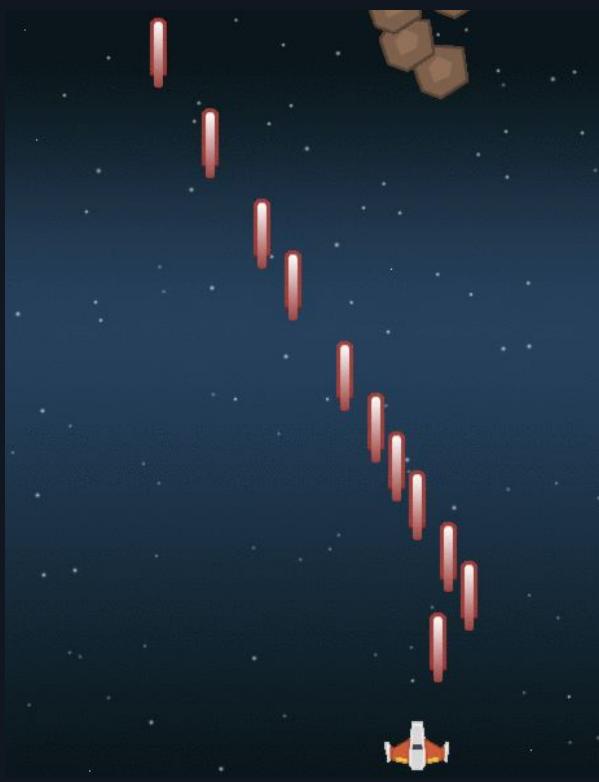
#6. Good representations + good model should work well
even
on small amount of training data

Ok, what's now?



3,0	4,26	5,51
3,1	0,27	1,52
3,2	0,28	1,53
3,3	0,29	1,54
3,4	0,30	5,55
3,5	0,31	1,56
0,6	4,32	1,57
0,7	0,33	5,58
4,8	0,34	1,59
0,9	3,35	1,60
0,10	5,36	1,61
0,11	1,37	5,62
4,12	1,38	1,63
0,13	1,39	1,64
0,14	1,40	5,65
0,15	5,41	1,66
4,16	1,42	1,67
0,17	1,43	1,68
0,18	5,44	5,69
4,19	1,45	1,70
0,20	1,46	1,71
0,21	5,47	1,72
4,22	1,48	5,73
0,23	1,49	1,74
0,24	1,50	1,75
0,25		

3,0		
3,1	4,26	5,51
3,2	0,27	1,52
3,3	0,28	1,53
3,4	0,29	1,54
3,5	0,30	5,55
0,6	0,31	1,56
0,7	4,32	1,57
4,8	0,33	5,58
0,9	0,34	1,59
0,10	3,35	1,60
0,11	5,36	1,61
0,12	1,37	5,62
4,12	1,38	1,63
0,13	1,39	1,64
0,14	1,40	5,65
0,15	5,41	1,66
4,16	1,42	1,67
0,17	1,43	1,68
0,18	5,44	5,69
4,19	1,45	1,70
0,20	1,46	1,71
0,21	5,47	1,72
4,22	1,48	5,73
0,23	1,49	1,74
0,24	1,50	1,75
0,25		



Only time (synth 126): avg time: 193.7 and avg score: 296

**#7. If there is known (only one) solution
you most likely don't need Machine Learning**

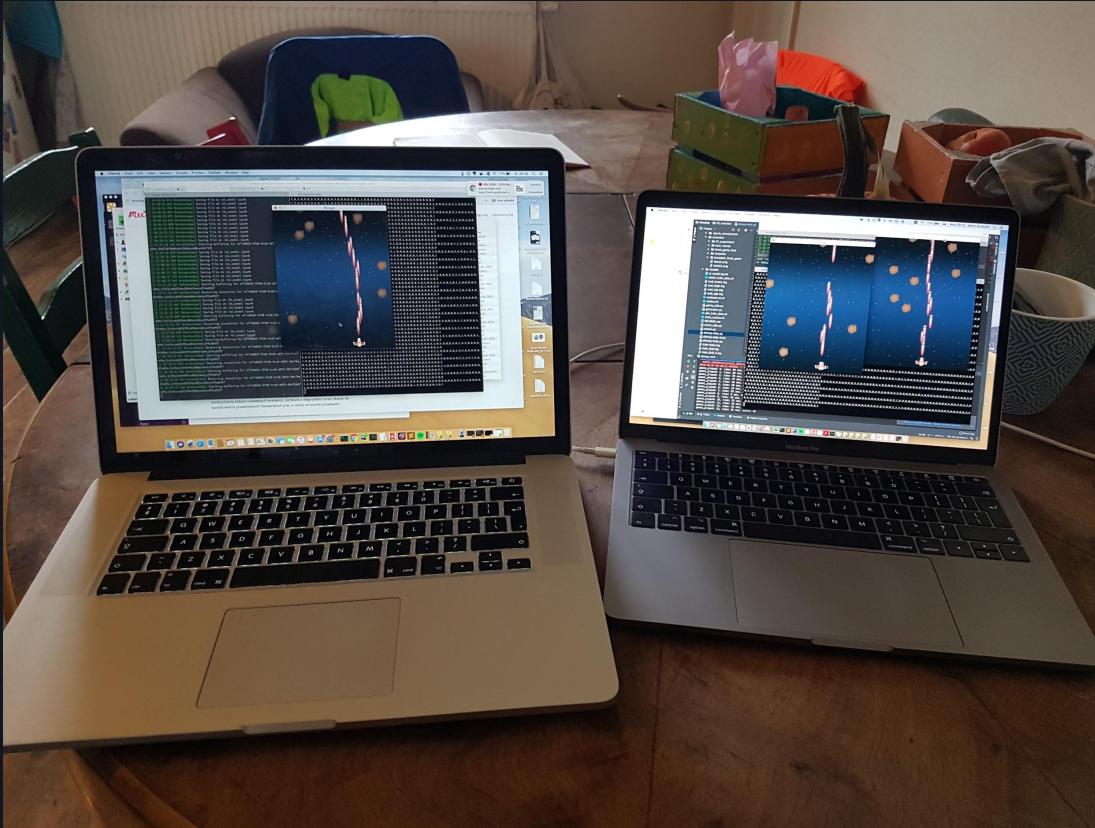
What about other games? What about GAI?

Evolution in Long.MAX_VALUE easy steps

Are you really into gaming?

What can be better than that?





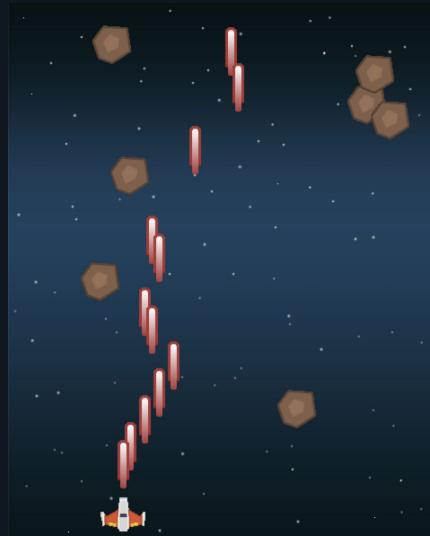


Work habits die hard ;-)

```
import os  
os.environ['SDL_VIDEODRIVER'] = 'dummy'  
import pygame
```

Sequence 1	3	0	2	5	4	3	5	4
Sequence 2	3	0	1	5	4	3	3	5
Sequence 3	3	0	5	5	4	2	3	4
Sequence 4	3	0	4	5	4	3	3	5
Sequence 5	3	1	2	4	4	2	3	4
Sequence 6	3	1	2	5	5	3	3	5

Sequence 1	3	0	2	5	4	3	5	4
Sequence 2	3	0	1	5	4	3	3	5
Sequence 3	3	0	5	5	4	2	3	4
Sequence 4	3	0	4	5	4	3	3	5
Sequence 5	3	1	2	4	4	2	3	4
Sequence 6	3	1	2	5	5	3	3	5



Sequence 4

Sequence 2

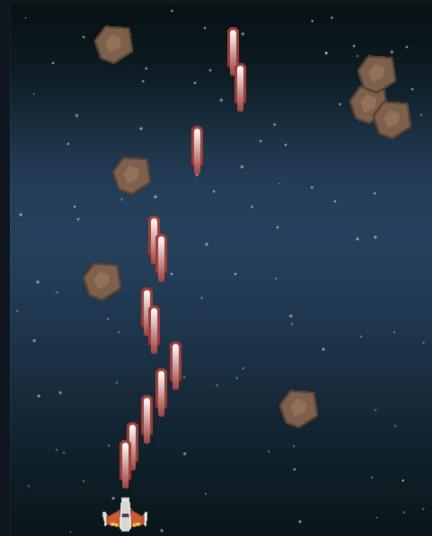
Sequence 1

Sequence 3

Sequence 5

Sequence 6

Sequence 1	3	0	2	5	4	3	5	4
Sequence 2	3	0	1	5	4	3	3	5
Sequence 3	3	0	5	5	4	2	3	4
Sequence 4	3	0	4	5	4	3	3	5
Sequence 5	3	1	2	4	4	2	3	4
Sequence 6	3	1	2	5	5	3	3	5



by score (avg, 4 runs)

Sequence 4

Sequence 2

Sequence 1

Sequence 3

Sequence 5

Sequence 6

Sequence 1	3	0	2	5	4	3	5	4
Sequence 2	3	0	1	5	4	3	3	5
Sequence 3	3	0	5	5	4	2	3	4
Sequence 4	3	0	4	5	4	3	3	5
Sequence 5	3	1	2	1	1	2	3	4
Sequence 6	3	1	2	5	5	3	3	5

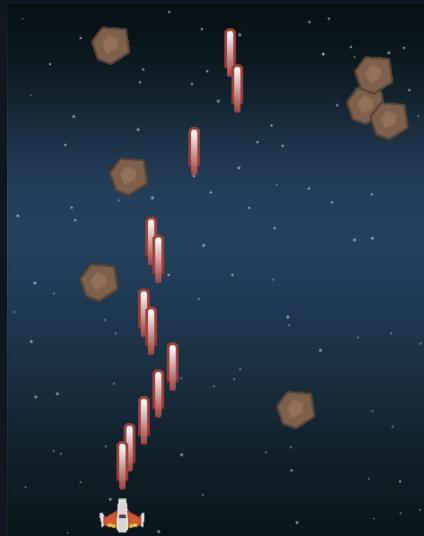
crossover

Sequence 1_2
Sequence 1_3
Sequence 1_4
Sequence 2_3
Sequence 2_4
Sequence 4

Sequence 1_2m	3	0	1	5	4	4	5	5
Sequence 1_3m	3	0	1	5	4	3	0	3
Sequence 1_4m	3	0	2	5	4	2	3	3
Sequence 2_3m	3	0	4	5	4	0	3	3
Sequence 2_4m	3	4	2	4	4	2	3	3
Sequence 4m	3	1	2	4	4	3	3	3

mutations

Sequence 1_2m	3	0	1	5	4	4	5	5
Sequence 1_3m	3	0	1	5	4	3	0	3
Sequence 1_4m	3	0	2	5	4	2	3	3
Sequence 2_3m	3	0	4	5	4	0	3	3
Sequence 2_4m	3	4	2	4	4	2	3	3
Sequence 4m	3	1	2	4	4	3	3	3



by score (avg, 4 runs)

Sequence 1_3m

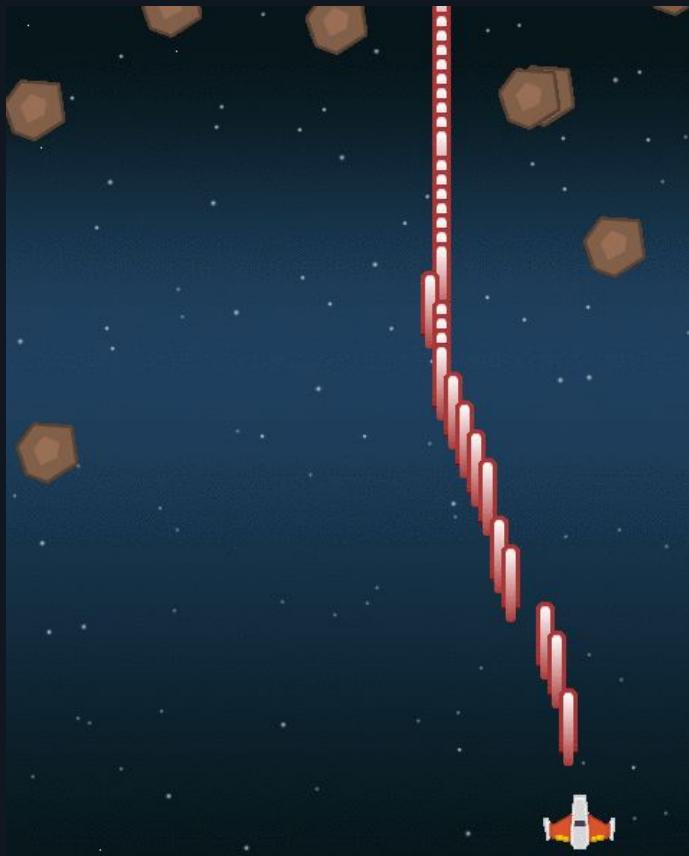
Sequence 4m

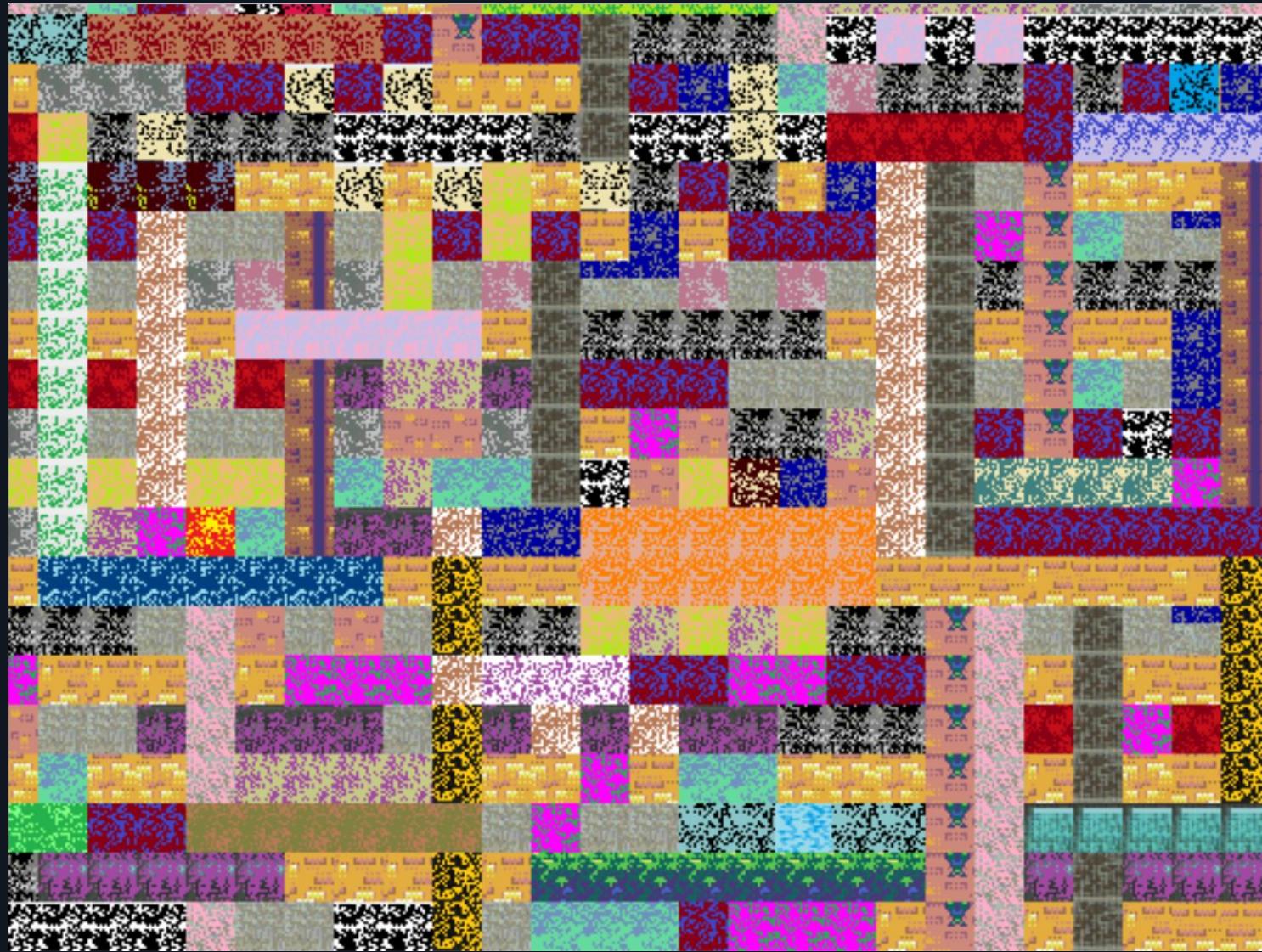
Sequence 1_4m

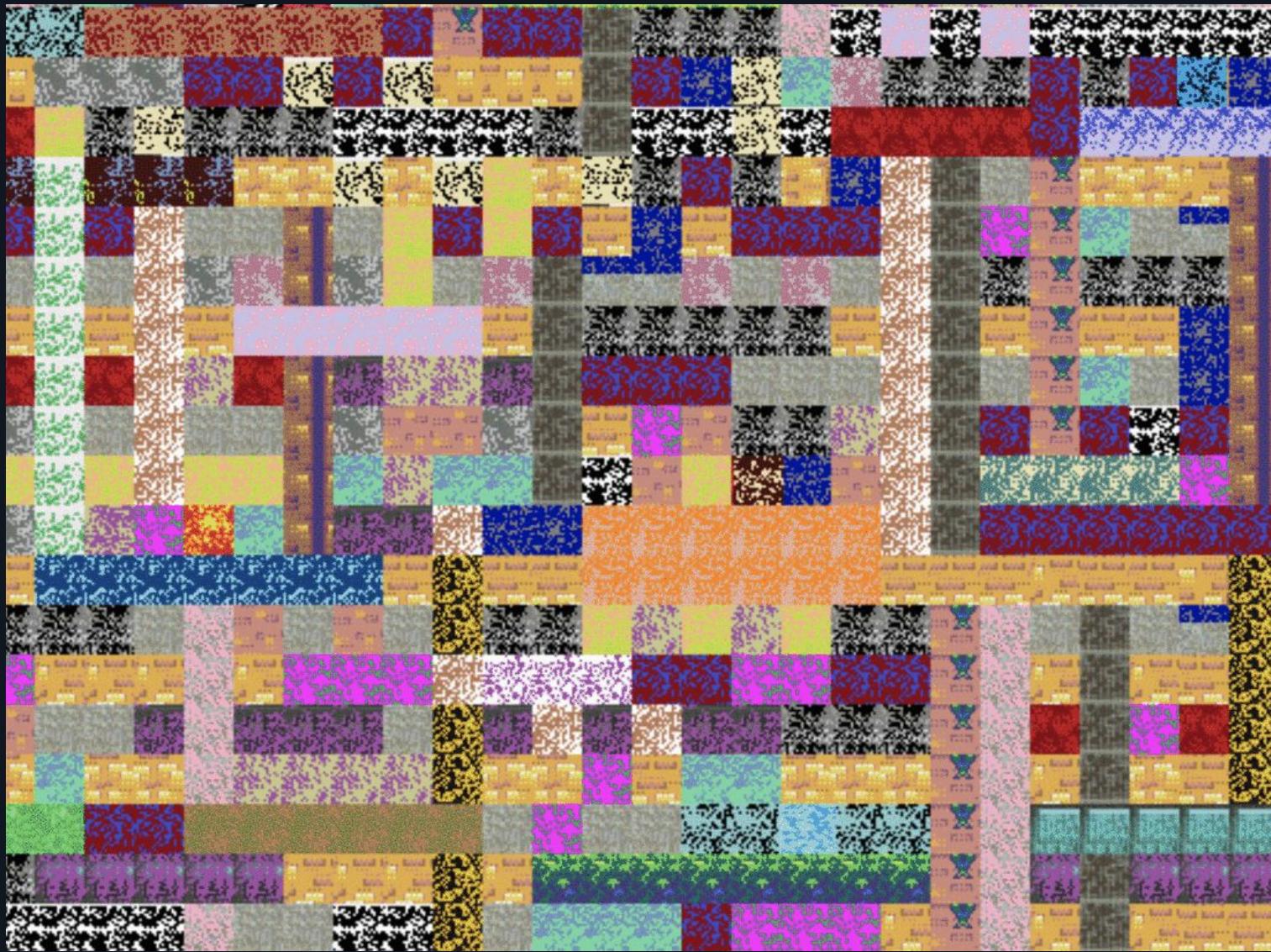
Sequence 2_3m

Sequence 2_4m

Sequence 1_2m







(Deep) Reinforcement Learning

Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih¹ Koray Kavukcuoglu² David Silver¹ Alex Graves¹ Ioannis Antonoglou¹

Daan Wierstra¹ Martin Riedmiller¹

DeepMind Technologies

{vlad,koray,david,alex.graves,ioannis,daan,martin.riedmiller}@deepmind.com

Abstract

We present the first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning. The model is a convolutional neural network, trained with a variant of Q-learning, whose input is raw pixels and whose output is a value function estimating future rewards. We apply our method to seven Atari 2600 games from the Arcade Learning Environment, with no adjustment of the architecture or learning algorithm. We find that it outperforms all previous approaches on six of the games and surpasses a human expert on three of them.

Observe and Look Further: Achieving Consistent Performance on Atari

Tobias Pohlen¹, Bilal Piot¹, Todd Hester¹, Mohammad Gheshlaghi Azar¹, Dan Horgan¹, David Budden¹, Gabriel Barth-Maron¹, Hado van Hasselt¹, John Quan¹, Mel Vecerik¹, Matteo Hessel¹, Rémi Munos¹, and Olivier Pietquin²

¹DeepMind, {pohlen,piot,todhester,mazar,horgan,budden,gabirelhm,hado,johnquan,vec,mthass,monos}@google.com
²Google Brain, pietquin@google.com

Abstract

Despite significant advances in the field of deep Reinforcement Learning (RL), today's algorithms still fail to learn human-level policies consistently over a set of diverse tasks such as Atari 2600 games. We identify three key challenges that any algorithm needs to master in order to perform well on all games: processing diverse reward distributions, reasoning over long time horizons, and exploring efficiently. In this paper, we propose an algorithm that addresses each of these challenges and is able to learn human-level policies on nearly all Atari games. A new transformed Bellman operator allows our algorithm to process rewards of varying densities and scales; an auxiliary *temporal consistency loss* allows us to train stably using a discount factor of $\gamma = 0.999$ (instead of $\gamma = 0.99$) extending the effective planning horizon by an order of magnitude; and we ease the exploration problem by using human demonstrations that guide the agent towards rewarding states. When tested on a set of 42 Atari games, our algorithm exceeds the performance of an average human on 40 games using a common set of hyper parameters. Furthermore, it is the first deep RL algorithm to solve the first level of MONTEZUMA'S REVENGE.

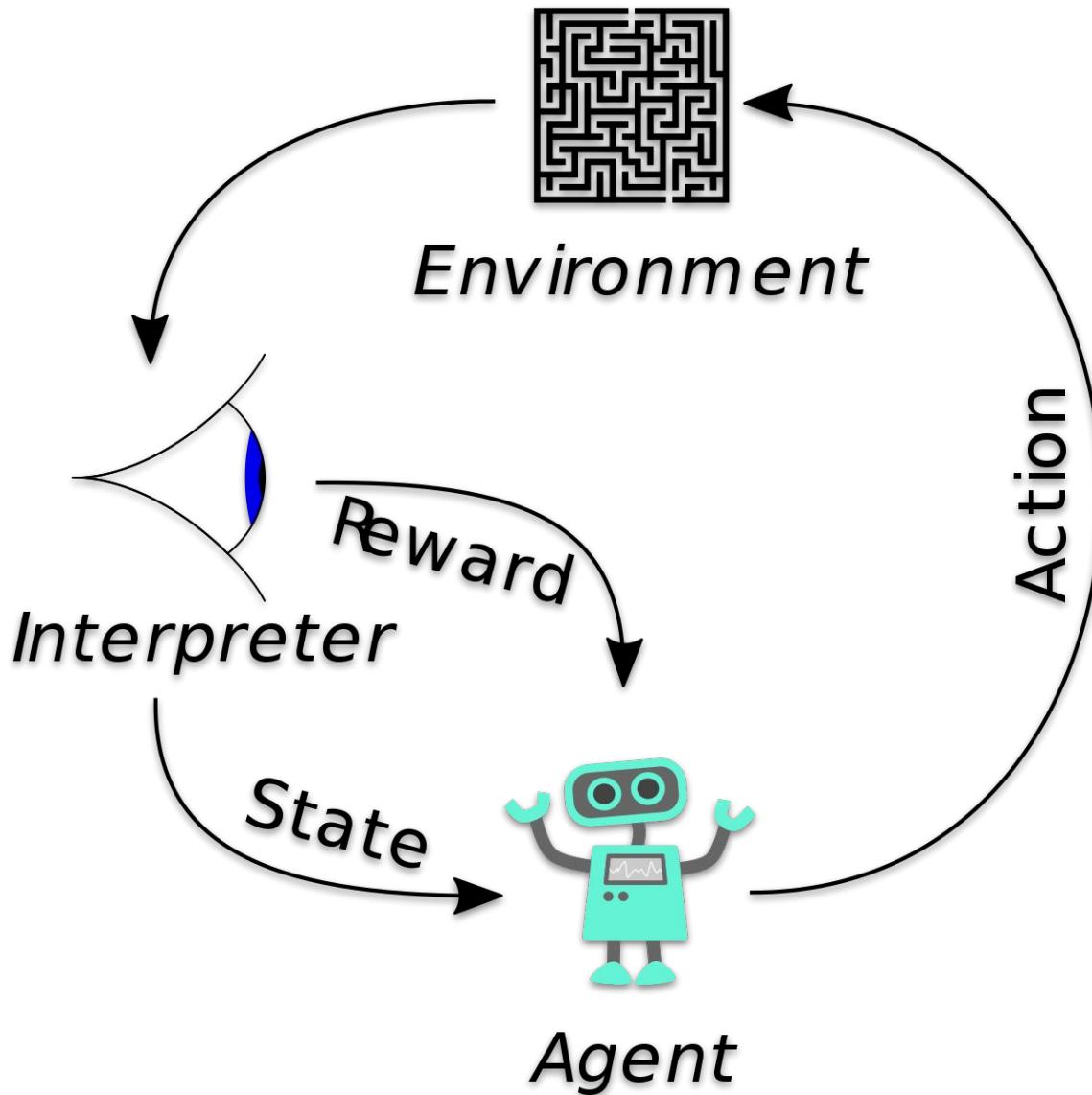
Playing hard exploration games by watching YouTube

Yusuf Aytar*, Tobias Pfaff*, David Budden, Tom Le Paine, Ziyu Wang, Nando de Freitas

DeepMind, London, UK
{yusufaytar,tpfaff,budden,tpaine,ziyu,nandodefreatas}@google.com

Abstract

Deep reinforcement learning methods traditionally struggle with tasks where environmental rewards are particularly sparse. One successful method of guiding exploration in these domains is to imitate trajectories provided by a human demonstrator. However, these demonstrations are typically collected under artificial conditions, i.e. with access to the agent's exact environment setup and the demonstrator's exact reward trajectories. Here we propose a framework that allows to collect these demonstrations by relying on publicly available footages without access to such data. First, we learn to map unaligned videos from multiple sources to a common representation using self-supervised objectives constructed over both time and modality (i.e. vision and sound). Second, we embed a single YouTube video in this representation to construct a reward function that encourages an agent to convincingly exceed human-level performance on the infamously hard exploration games MONTEZUMA'S REVENGE, PITFALL!, and PRIVATE EYE for the first time, even if the agent is not presented with any environment rewards.



(Deep) Reinforcement Learning

Q-function - estimates reward for given state and action

State = { player_x, player_y..... }

Rewards = {
 scores = 1,
 stay_alive = 0.2,
 dead = 0
}

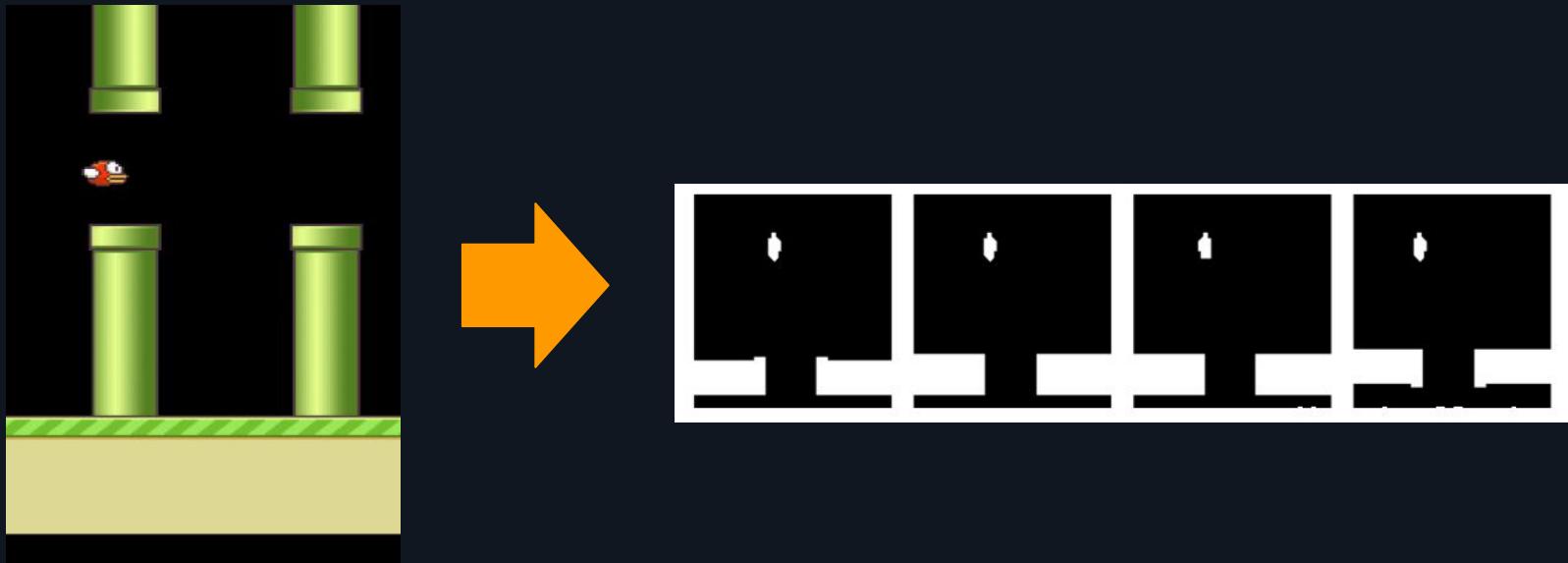
	Action 1	Action 2	Action 3
State 1	0.1	0.4	0.8
State 2	0.4	0.2	0.6
...
State n	1	0.6	0.6

Reinforcement Learning

Initialize Q-table

```
while alive:  
    state = get_game_state()  
    action = Q-table.get_best_action(state)  
    reward = play_game(action)  
    Q-table.update(state, action, reward)  
  
Q-table.get_best_action(state):  
    if state not in seen_states:  
        seen_states[state] = initialize_state(0)  
    return seen_states[state].get_best_action()
```

Pixels as state representation



Convolutional networks

Layer	Input	Filter size	Stride	Num filters	Activation	Output
conv1	84x84x4	8x8	4	32	ReLU	20x20x32
conv2	20x20x32	4x4	2	64	ReLU	9x9x64
conv3	9x9x64	3x3	1	64	ReLU	7x7x64
fc4	7x7x64			512	ReLU	512
fc5	512			18	Linear	18

Source: <https://www.intel.ai/demystifying-deep-reinforcement-learning/#gs.4ee7yr>

After 240 minutes of training

**This is where the magic happens:
it realizes that digging a tunnel through the
wall is the most effective technique to beat
the game.**



<https://gym.openai.com/envs/#atari>
<https://github.com/openai/gym>

problem representation, training data, model
hyperparameters, test in production
fun / patience

Links and source code:

<https://github.com/maneo/letsplay>

Questions?

name.surname@allegro.pl

twitter: @maneo

