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!

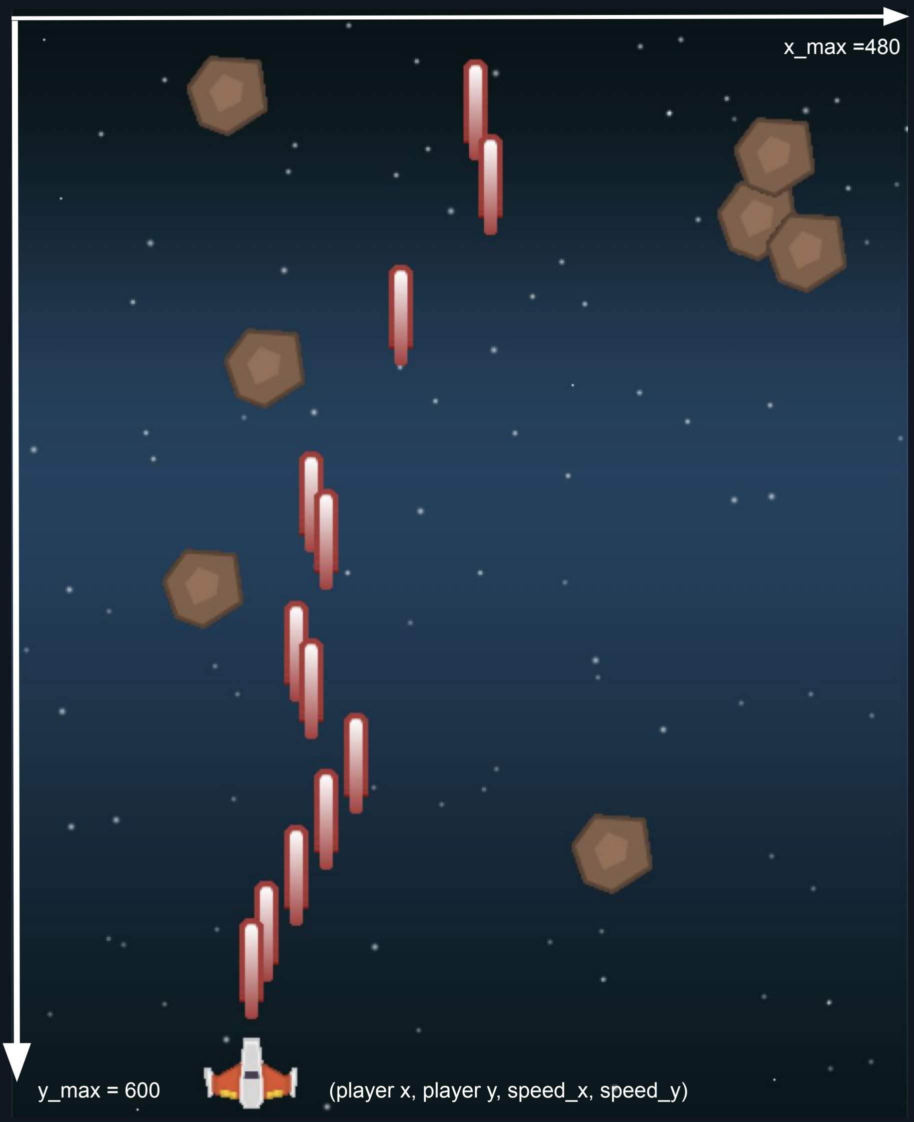
```python
import pygame

img_dir = path.join(path.dirname(__file__), '../img')

WIDTH = 480
HEIGHT = 600
FPS = 60
MOBS_SIZE = 8

# define colors
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
RED = (255, 0, 0)
GREEN = (0, 255, 0)
BLUE = (0, 0, 255)
YELLOW = (255, 255, 0)


# initialize pygame and create window
pygame.init()
pygame.mixer.init()
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Shmup!")
clock = pygame.time.Clock()
```

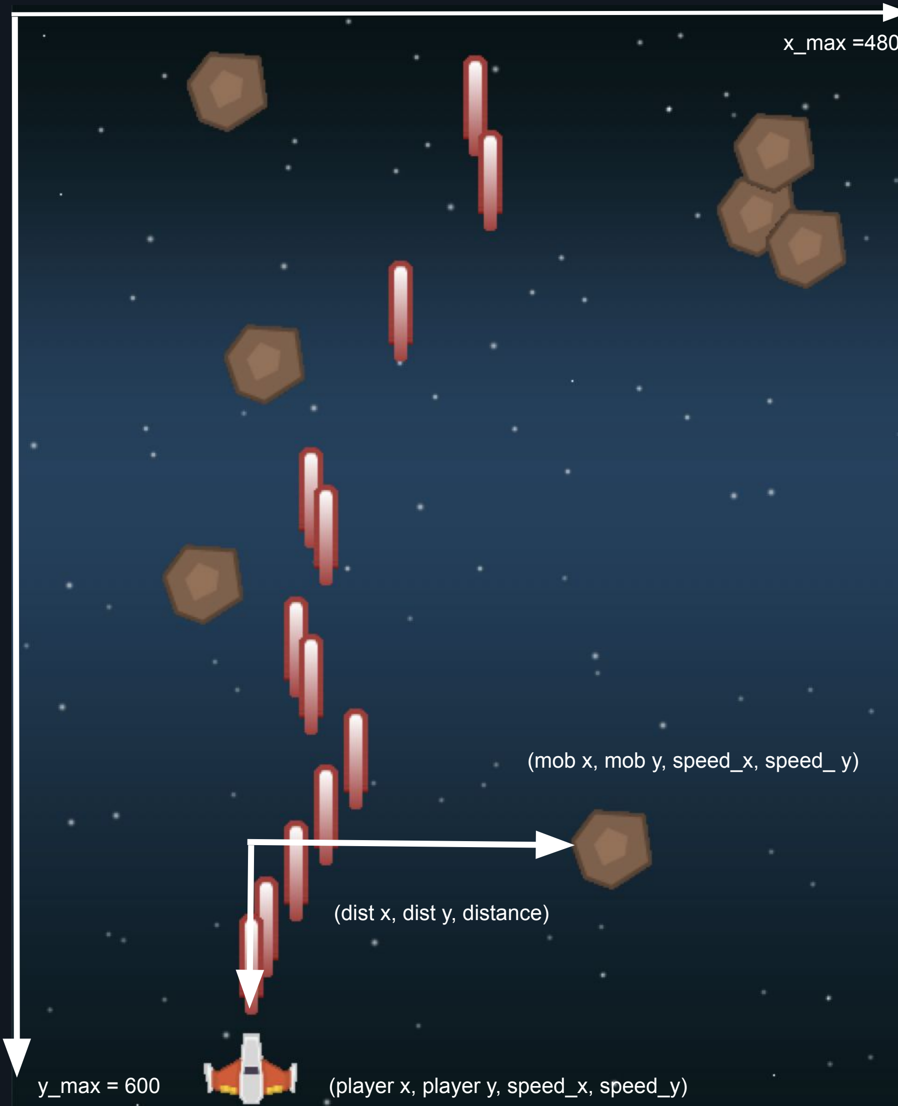
```python
# Load all game graphics
background = pygame.image.load(path.join(img_dir, "starfield.png")).convert()
background_rect = background.get_rect()
player_img = pygame.image.load(path.join(img_dir, "playerShip1_orange.png")).convert()
meteor_img = pygame.image.load(path.join(img_dir, "meteorBrown_med1.png")).convert()
bullet_img = pygame.image.load(path.join(img_dir, "laserRed16.png")).convert()

all_sprites = pygame.sprite.Group()
mobs = pygame.sprite.Group()
bullets = pygame.sprite.Group()
player = Player()
all_sprites.add(player)
for i in range(MOBS_SIZE):
    m = Mob()
    all_sprites.add(m)
    mobs.add(m)
```

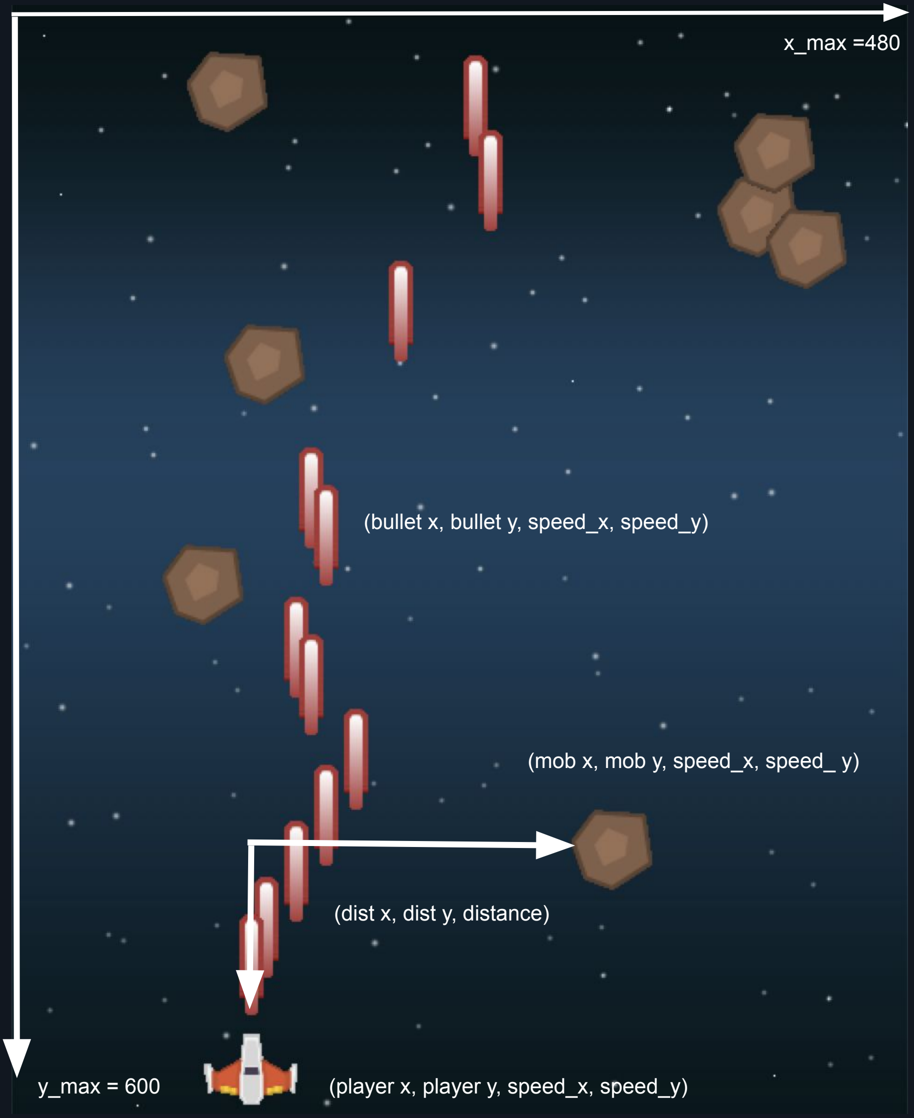
```python
# Game loop
game_start_time = time.time()

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

    # Process input (events)
    for event in pygame.event.get():
        # check for closing window
        if event.type == pygame.QUIT:
            running = False
        elif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_SPACE:
                player.shoot()

    # Update
    all_sprites.update()

    # check to see if a bullet hit a mob
    hits = pygame.sprite.groupcollide(mobs, bullets, True, True)
    for hit in hits:
        m = Mob()
        all_sprites.add(m)
        mobs.add(m)

    # check to see if a mob hit the player
    hits = pygame.sprite.spritecollide(player, mobs, False)
    if hits:
        running = False

    # Draw / render
    screen.fill(BLACK)
    screen.blit(background, background_rect)
    all_sprites.draw(screen)
    # *after* drawing everything, flip the display
    pygame.display.flip()

end = time. time()
pygame.quit()
```

Shmup!

What data is relevant?


x_max =480
y_max = 600
(player x, player y, speed_x, speed_y)


x_max =480
(mob x, mob y, speed_x, speed_ y)
(dist x, dist y, distance)
y_max = 600
(player x, player y, speed_x, speed_y)


x_max =480
(bullet x, bullet y, speed_x, speed_y)
(mob x, mob y, speed_x, speed_ y)
(dist x, dist y, distance)
y_max = 600
(player x, player y, speed_x, speed_y)

$$
\begin{aligned}
\text{game state}(t) = \{ & \text{player\_x},\ \text{speed\_x}, \\
& \text{mob}_1(\text{x, y, dist\_x, dist\_y, speed\_x, speed\_y}), \\
& \ldots, \text{mob}_8(\ldots) \}
\end{aligned}
$$

$$
\begin{aligned}
\text{game state}(t) = \{\ & \text{player\_x},\ \ \text{speed\_x}, \\
& \text{mob}_1(\text{x, y, dist\_x, dist\_y, speed\_x, speed\_y}), \\
& \ldots,\ \text{mob}_8(\ldots)\ \}
\end{aligned}
$$

$$
\text{action} = \{\ \text{left, right, fire, nothing, left+fire, right+fire}\ \}
$$

```python
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]
```

```python
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

Getting training data

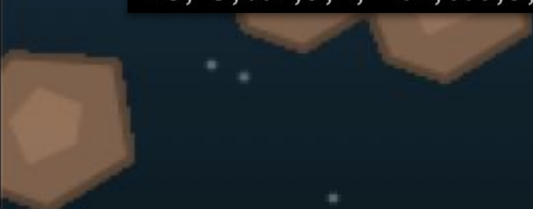
```python
# Game loop
score = 0
game_start_time = time.time()

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

    # Process input (events)
    for event in pygame.event.get():
        # check for closing window
        if event.type == pygame.QUIT:
            running = False
        elif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_SPACE:
                player.shoot()
                was_shooting = True

    # Update
    all_sprites.update()

    # check to see if a bullet hit a mob
    hits = pygame.sprite.groupcollide(mobs, bullets, True, True)
    for hit in hits:
        m = Mob()
        all_sprites.add(m)
        mobs.add(m)
        score += 1

    # check to see if a mob hit the player
    hits = pygame.sprite.spritecollide(player, mobs, False)
    if hits:
        running = False

    dump_as_vector(mobs, player, bullets, pygame, was_shooting)

    # Draw / render
    screen.fill(BLACK)
    screen.blit(background, background_rect)
    all_sprites.draw(screen)
    # *after* drawing everything, flip the display
    pygame.display.flip()

end = time. time()
print("time: {} sec, score: {}".format(round(end - game_start_time), sco

pygame.quit()
```


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)
```

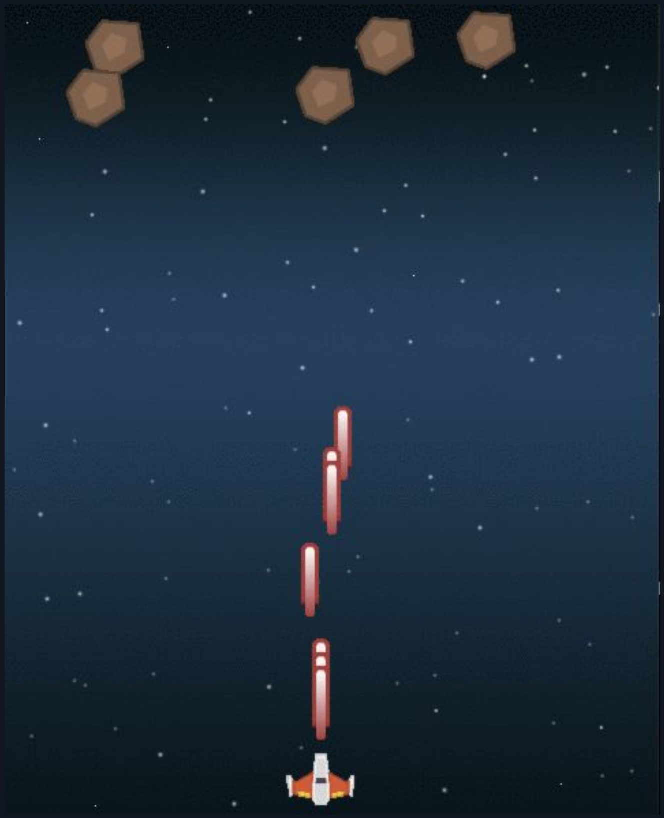
```
Training accuracy: 0.78, evaluation accuracy: 0.71
```

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

Let’s plug our model in!

```python
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))
```

```python
# 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)
```

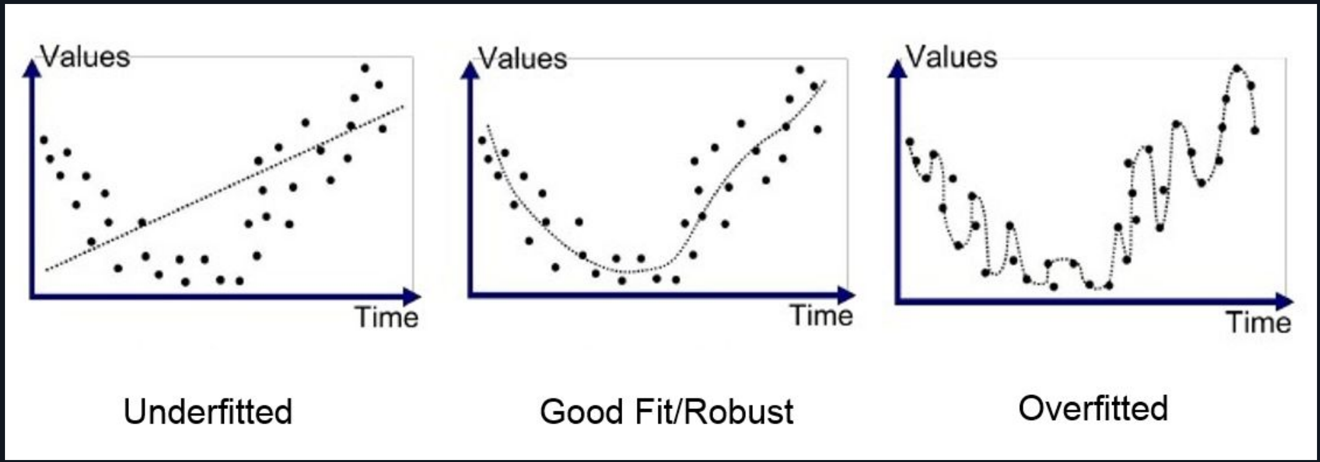
Out[18]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       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"))
```


ARTIFICIAL INTELLIGENCE
HAS COME SO FAR.



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

$$\text{game state}(t) = \{\ \text{game state}(t-1),\ \text{player\_x},\ \ \text{speed\_x},$$
$$\text{mob}_1(x,\ y,\ \text{dist\_x},\ \text{dist\_y},\ \text{speed\_x},\ \text{speed\_y}),$$
$$\ldots,\ \text{mob}_8(\ldots)\ \}$$

$$\text{action} = \{\ \text{left, right, fire, nothing, left+fire, right+fire}\ \}$$

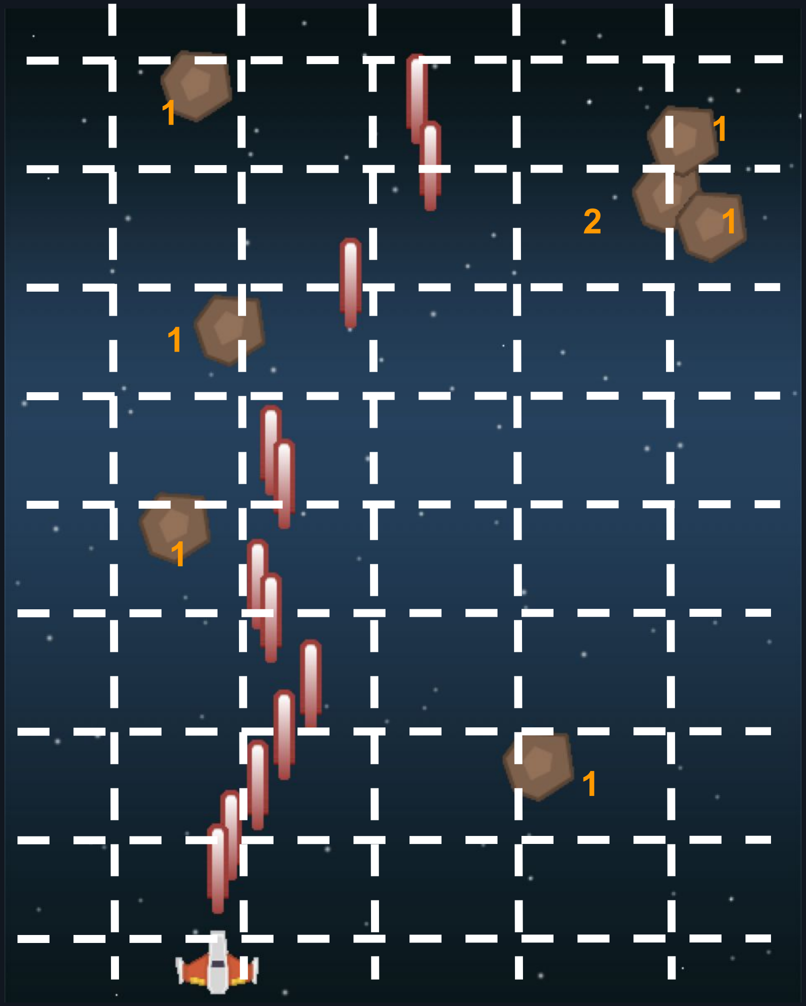
Model	Score (avg, 10 attempts)
XGBClassifier	0.65
MLPClassifier (neural network)	1.83
Logistic Regression	13

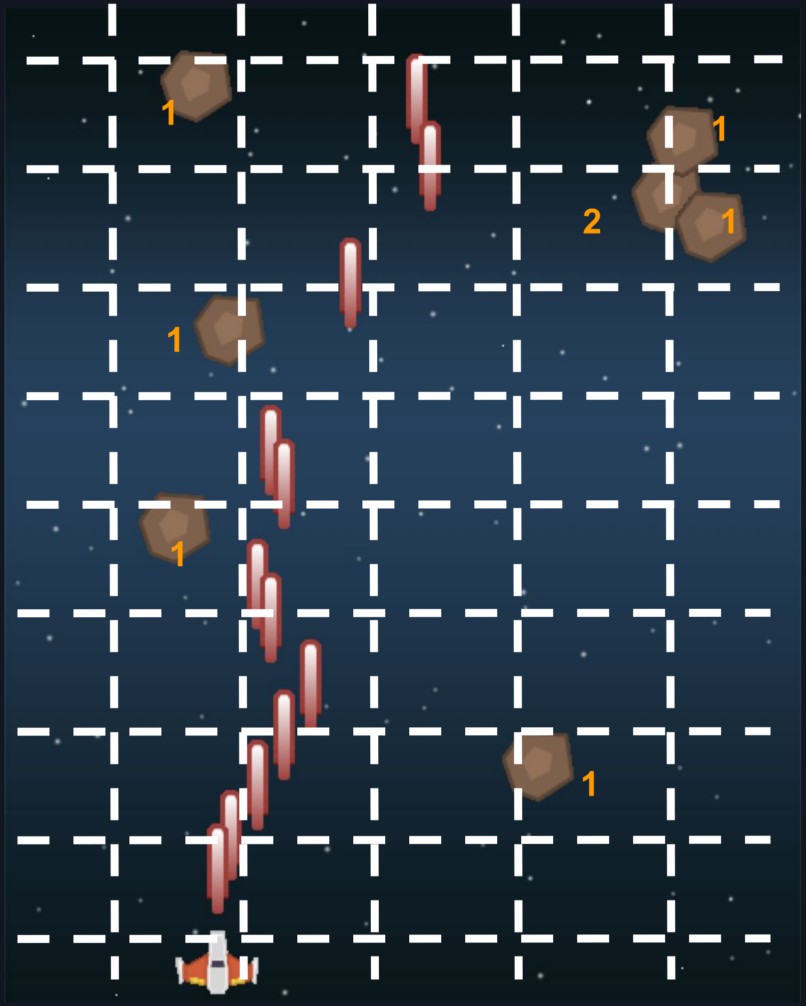
#4. Increasing complexity of state representation may not be the best idea

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


THULE

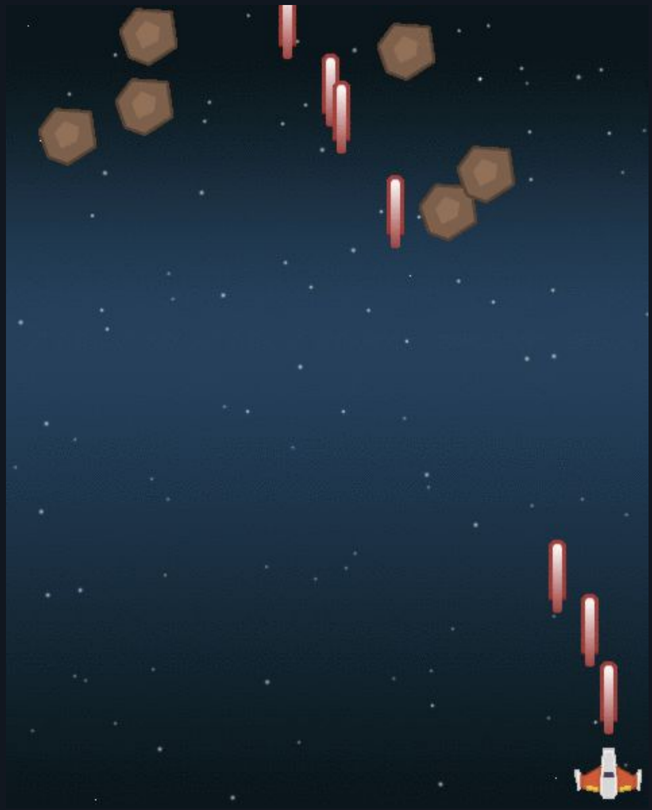
```python
import os
os.environ['SDL_VIDEODRIVER'] = 'dummy'

import pygame
```


1
1
2
1
1
1
1



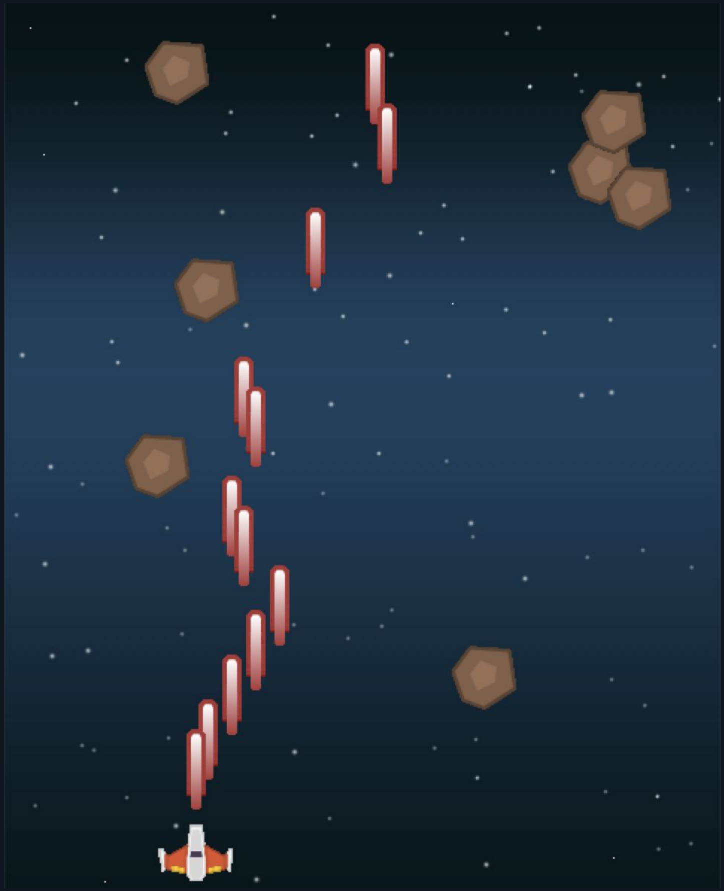

```
state.append(player.dump_state_vector()[1] % 6)

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]]

# 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
3,1
3,2
3,3
3,4
3,5
0,6
0,7
4,8
0,9
0,10
0,11
4,12
0,13
0,14
0,15
4,16
0,17
0,18
4,19
0,20
0,21
4,22
0,23
0,24
0,25
```

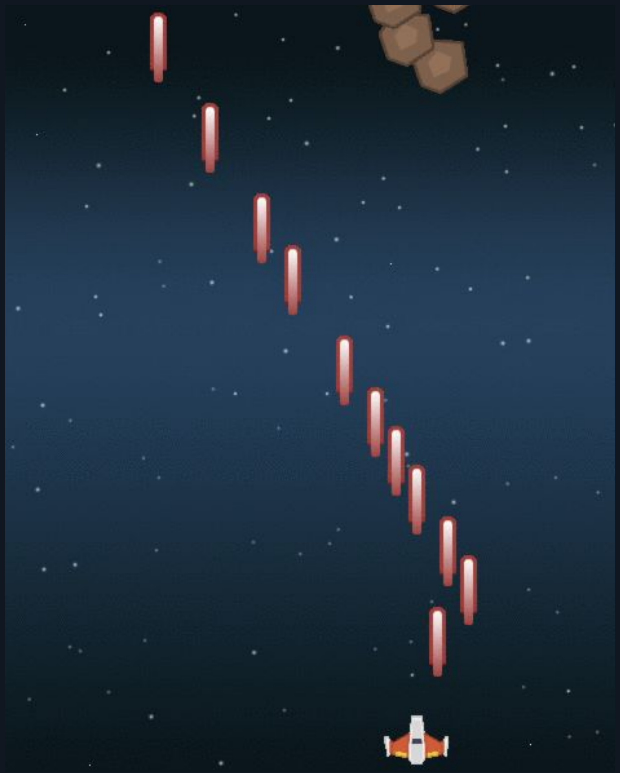
```
4,26
0,27
0,28
0,29
0,30
0,31
4,32
0,33
0,34
3,35
5,36
1,37
1,38
1,39
1,40
5,41
1,42
1,43
5,44
1,45
1,46
5,47
1,48
1,49
1,50
```

```
5,51
1,52
1,53
1,54
5,55
1,56
1,57
5,58
1,59
1,60
1,61
5,62
1,63
1,64
5,65
1,66
1,67
1,68
5,69
1,70
1,71
1,72
5,73
1,74
1,75
```

```
3,0
3,1
3,2
3,3
3,4
3,5
0,6
0,7
4,8
0,9
0,10
0,11
4,12
0,13
0,14
0,15
4,16
0,17
0,18
4,19
0,20
0,21
4,22
0,23
0,24
0,25
```

```
4,26
0,27
0,28
0,29
0,30
0,31
4,32
0,33
0,34
3,35
5,36
1,37
1,38
1,39
1,40
5,41
1,42
1,43
5,44
1,45
1,46
5,47
1,48
1,49
1,50
```

```
5,51
1,52
1,53
1,54
5,55
1,56
1,57
5,58
1,59
1,60
1,61
5,62
1,63
1,64
5,65
1,66
1,67
1,68
5,69
1,70
1,71
1,72
5,73
1,74
1,75
```

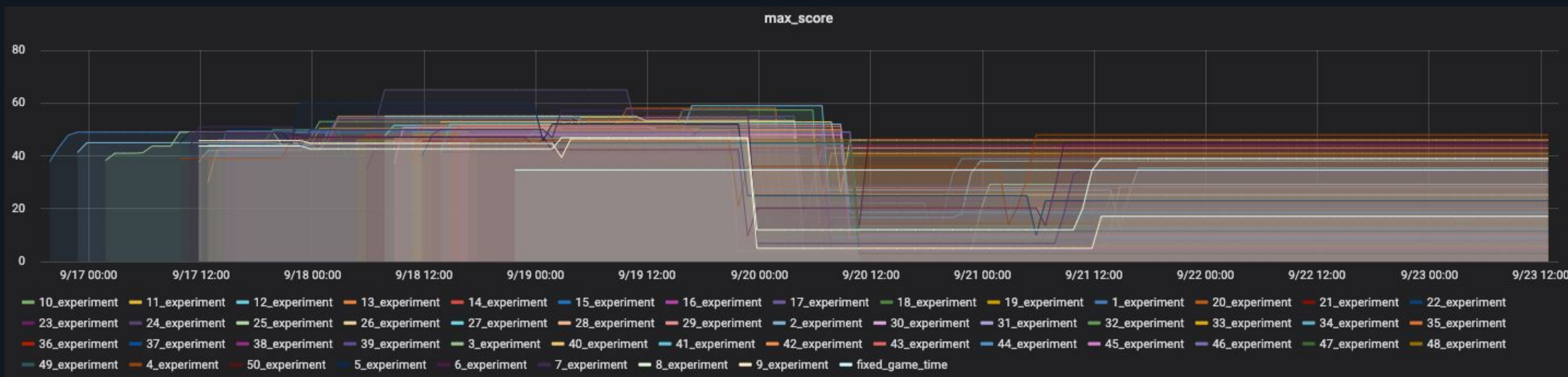


Only time (synth 126): avg time: 193.7 and avg score: 296

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

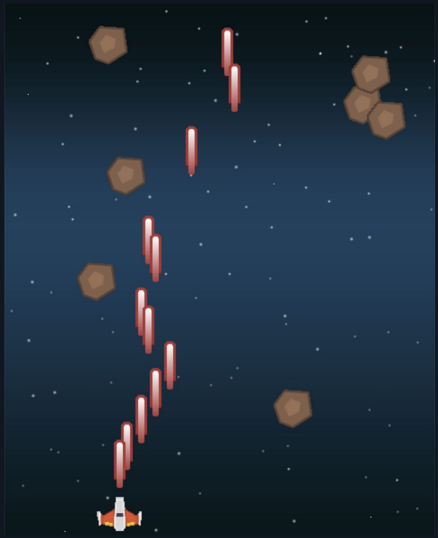
What about other games? What about GAI?

Evolution in Long.MAX_VALUE easy steps



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



by score (avg, 4 runs)

Sequence 4

Sequence 2

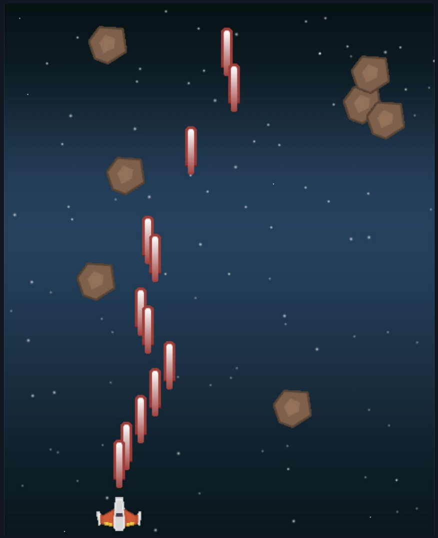
Sequence 1

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



by score (avg, 4 runs)

Sequence 4

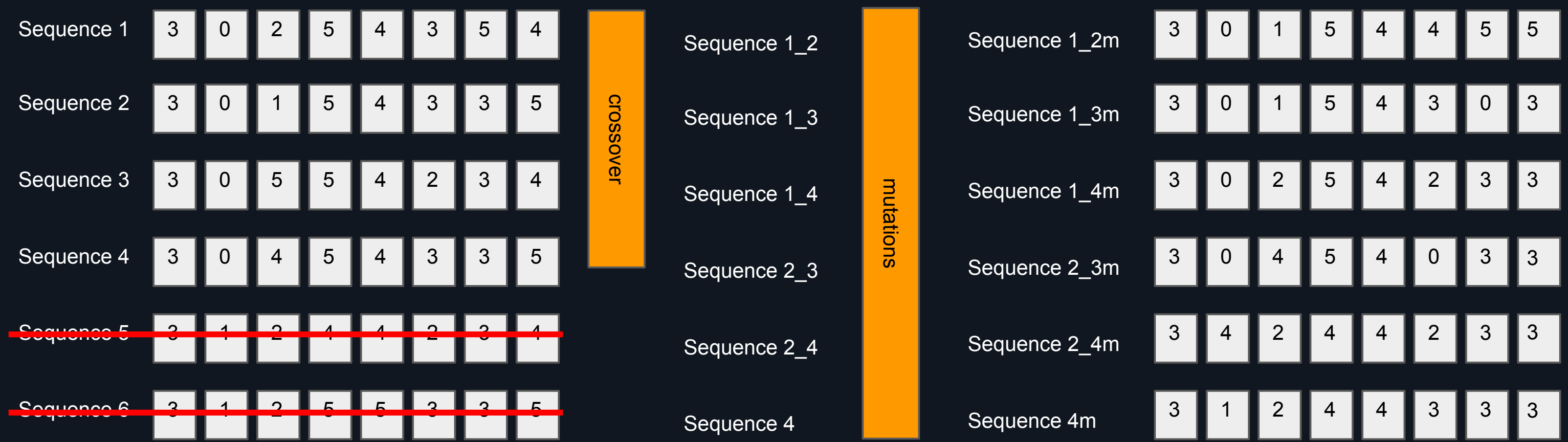
Sequence 2

Sequence 1

Sequence 3

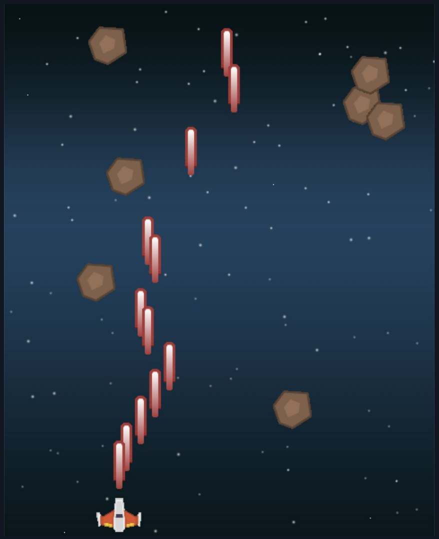
Sequence 5

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 4 4 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
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

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

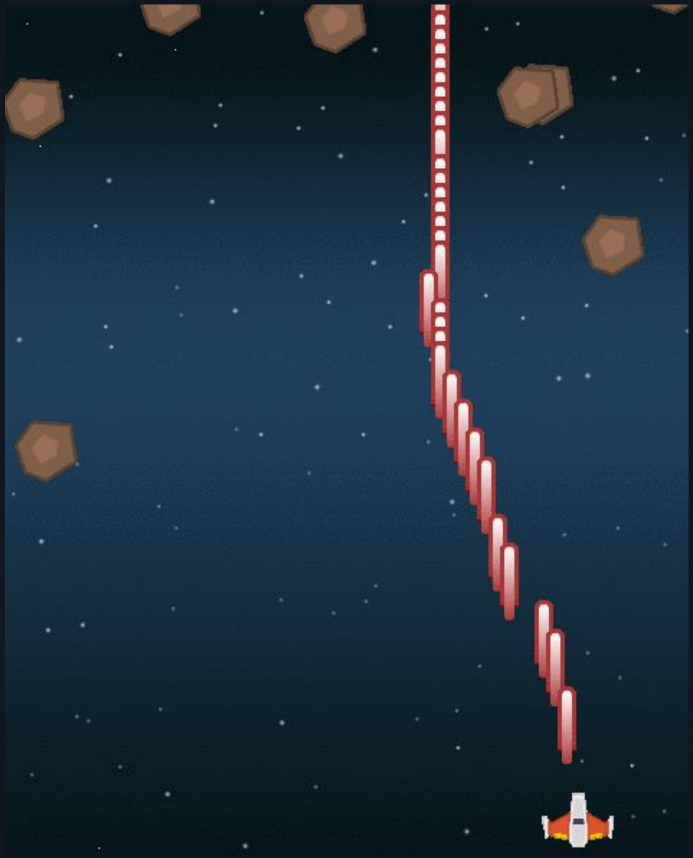
Sequence 4m

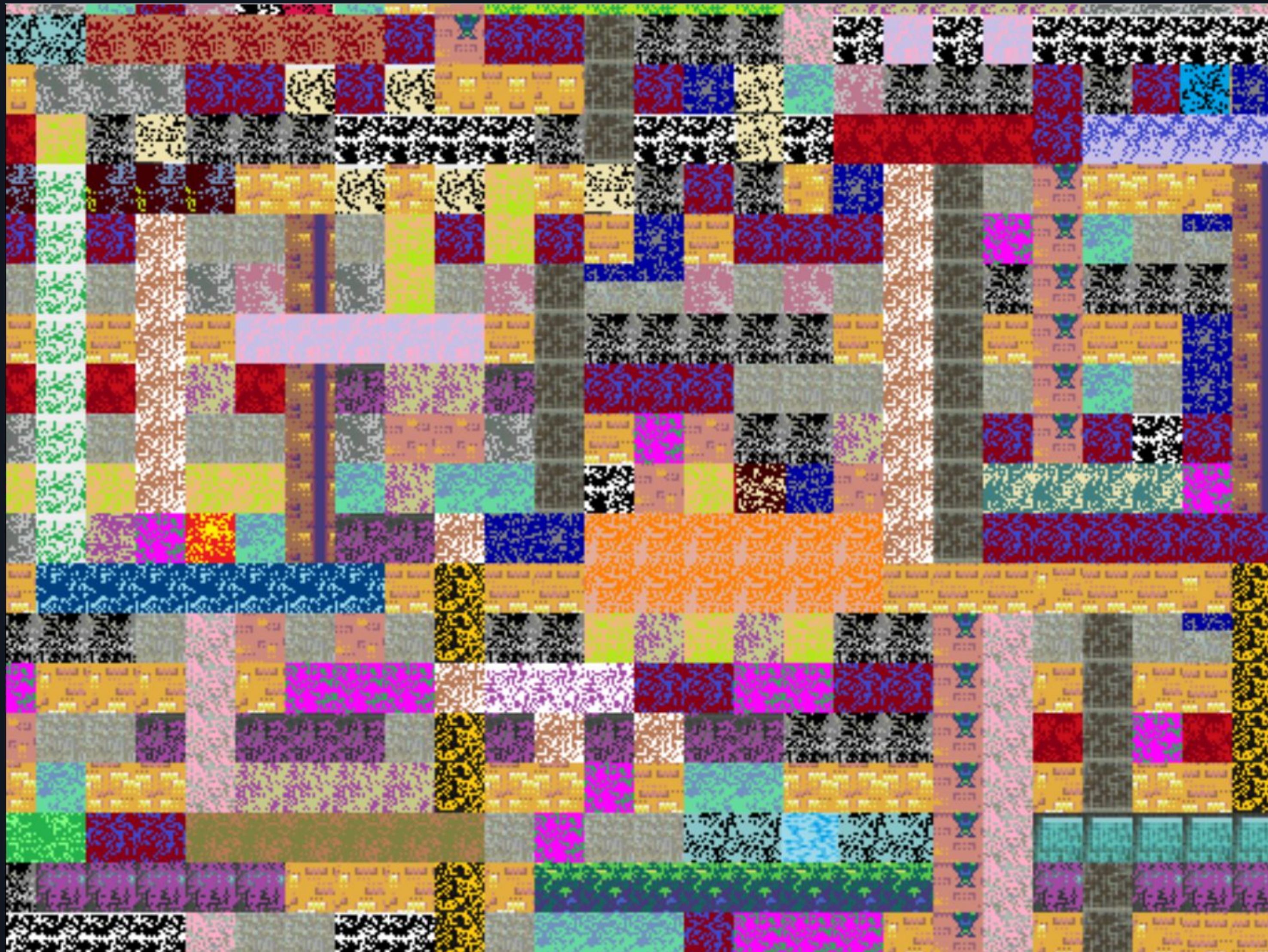
Sequence 1_4m

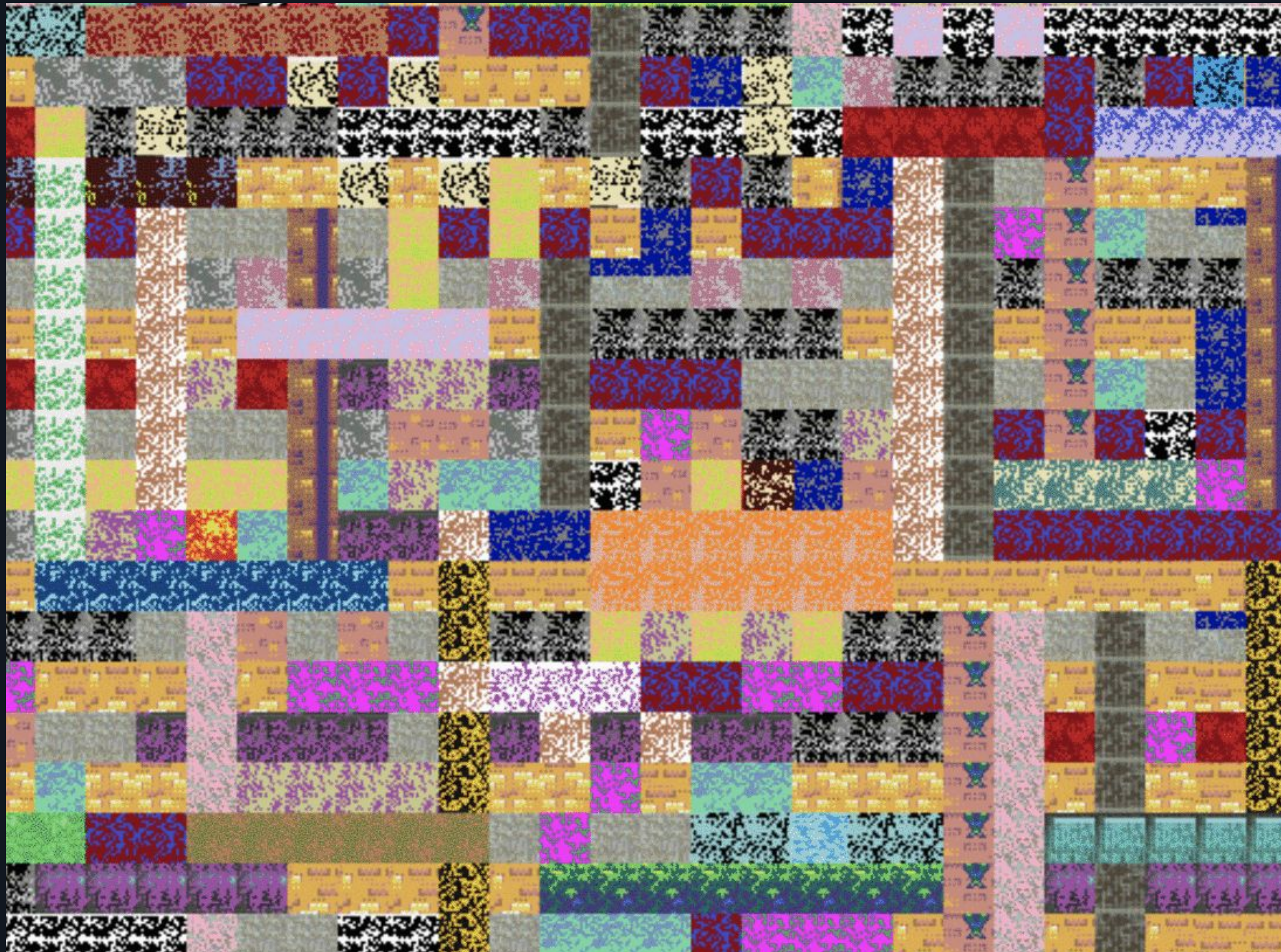
Sequence 2_3m

Sequence 2_4m

Sequence 1_2m







(Deep) Reinforcement Learning

Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih **Koray Kavukcuoglu** **David Silver** **Alex Graves** **Ioannis Antonoglou**
Daan Wierstra **Martin Riedmiller**
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 Pohlen[1], **Bilal Piot**[1], **Todd Hester**[1], **Mohammad Gheshlaghi Azar**[1], **Dan Horgan**[1], **David Budden**[1], **Gabriel Barth-Maron**[1], **Hado van Hasselt**[1], **John Quan**[1], **Mel Večerík**[1], **Matteo Hessel**[1], **Rémi Munos**[1], and **Olivier Pietquin**[2]

[1]DeepMind, {pohlen, piot, toddhester, mazar, horgan, budden, gabirelbm, hado, johnquan, vec, mtthss, munos}@google.com
[2]Google Brain, pietquin@google.com

Abstract

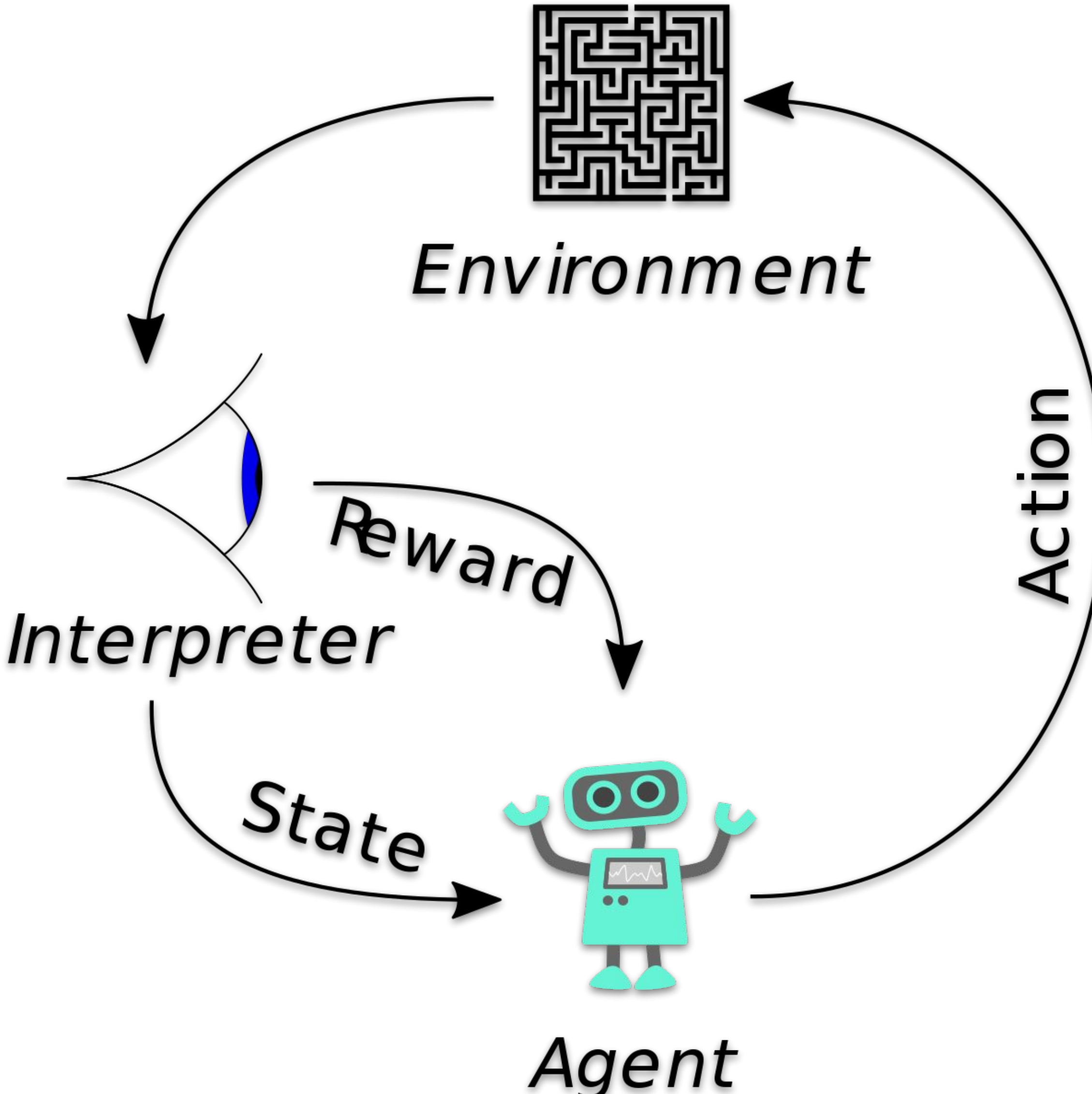
Despite significant advances in the field of deep Reinforcement Learning (RL), today's algorithms still fail to learn human-level policies consistently over a set of diverse tasks such as Atari 2600 games. We identify three key challenges that any algorithm needs to master in order to perform well on 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,nandodefreitas}@google.com

Abstract

Deep reinforcement learning methods traditionally struggle with tasks where environment 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 action and reward trajectories. Here we propose a two-stage method that overcomes these limitations by relying on noisy, unaligned footage 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 imitate human gameplay. This method of one-shot imitation allows our 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.



Source: Wikipedia

(Deep) Reinforcement Learning

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

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

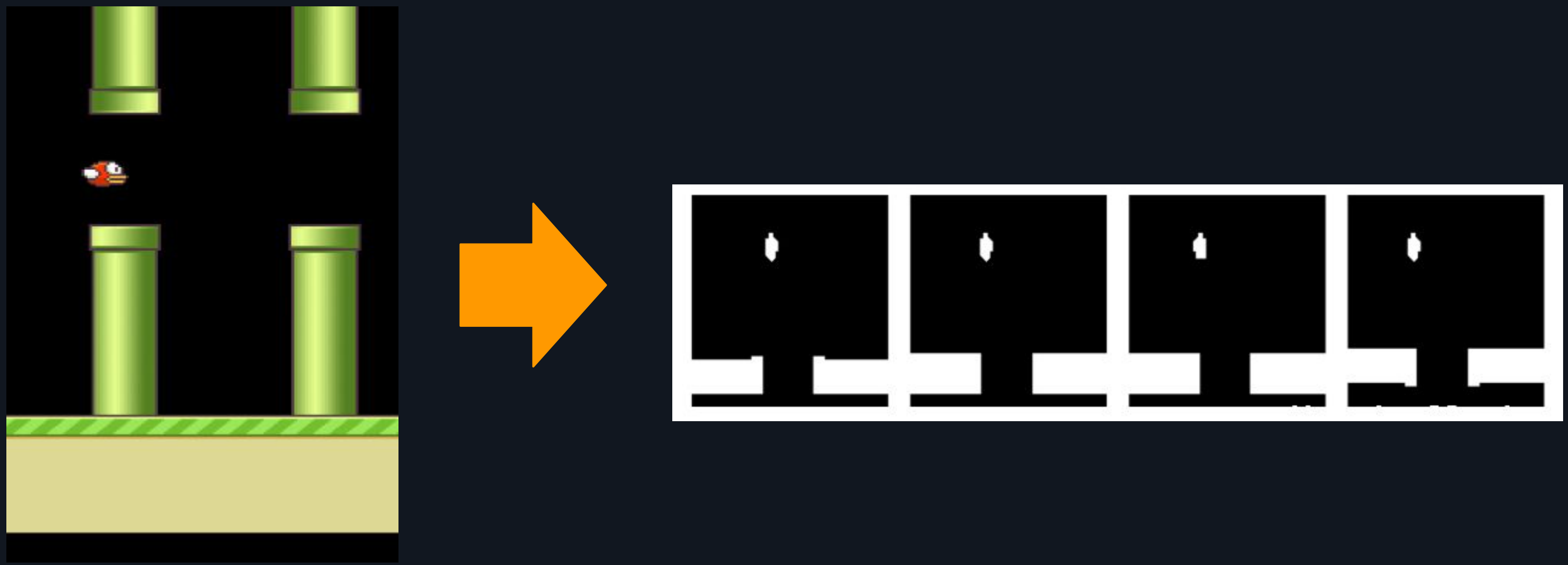
Q-function - estimates reward for given state and action

	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.

Links and source code:

https://github.com/maneo/letsplay

Questions?

name.surname@allegro.pl

twitter: @maneo