Занятие #2

> **Мастер-класс: обучение с подкреплением.** Структура оболочки, обучение через подражание.

Повторение

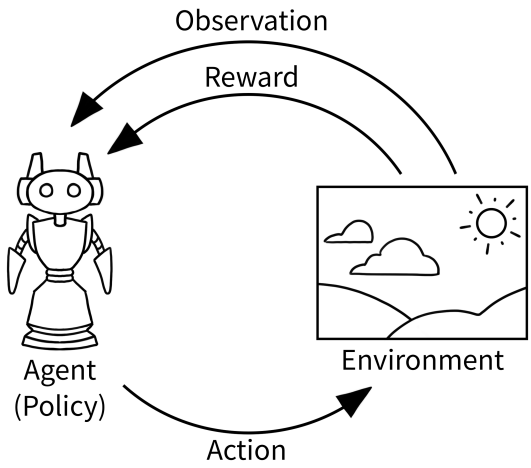
На прошлом занятии мы познакомились с условиями задачи, установили необходимые программы и запустили нашу среду. Давайте еще раз запустим нашу программу и удостоверимся, что пока нас не было её никто не сломал.

Отлично, вроде работает. Но вот только что работает и как пока непонятно. Давайте поближе познакомимся с тем, как устроена оболочка игры.

Оболочка `Gymnasium`

Для любого рода задач машинного обучения важно понимать как устроены данные. В нашем случае данные это приложение или игра, то есть какая-то готовая программа.

На самом деле это не совсем так, то, что мы видим при запуске программы на компьютере называется оболочка. Она запускает интересующее нас приложение, в нашем случае игру, но при этом добавляет дополнительные функции, которых изначально в приложении не было. Например, возможность управлять программой с помощью языка программирования или получать информацию об очках напрямую из памяти игры. Это сильно упрощает коммуникацию между нашей программой и приложением, устанавливает единый язык для общения.



Давайте подробнее разберём каждую составляющую оболочки, и то для чего они нужны.

Окружение	Необходимое нам приложение или игра.
Агент	Наша программа.
Действия	Определенные действия, которые передает программа приложению.
Награда	Вознаграждение за действие в приложении, например, баллы в игре.
Наблюдение	Известные параметры программы, в нашем случае изображение игры.

Для того, чтобы лучше понять эти составляющие, рассмотрим конкретный пример оболочки — `Atari Pacman`.

Действия. Изначально на игровой приставке `Atari 2600` существует 18 различных действий, но конкретно в этой игре нужны всего восемь из них. Давайте на них посмотрим:

Значение	Действие	Значение	Действие	Значение	Действие
`0`	`NOOP`	`1`	`UP`	`2`	`RIGHT`
`3`	`LEFT`	`4`	`DOWN`	`5`	`UPRIGHT`
`6`	`UPLEFT`	`7`	`DOWNRIGHT`	`8`	`DOWNLEFT`

Как несложно заметить действий не восемь, а девять. Всё дело в том, что опция “ничего не делать”, тоже является действием. А остальные действия просто указывают на нажатия кнопок управления на контроллере приставки. Некоторые из них отвечают за одновременное нажатие двух кнопок управления.

```
# Отобразим в консоле пространство действий для текущей среды.
print("Пространство действий —", env.action_space)

>>> Пространство действий — Discrete(9)
```

Во время обучения модели искусственного интеллекта мы будем предсказывать новое действие, то есть на выходе нашей модели мы будем получать число от нуля до восьми.

Наблюдения. Всего существует три вида данных для наблюдения: `"rgb"`, `"grayscale"` и `"ram`

Давайте напишем небольшую программу, которая отобразит как выглядят выходные данные для каждого из типов наблюдений.

```
# Проходим по каждой среде и типу наблюдений.
for obs_type, env in envs:
    # Отобразим в консоле пространство наблюдений для текущей среды.
    print(f'Пространство наблюдений для "{obs_type}" — {env.observation_space}')

>>> Пространство наблюдений для "ram" — Box(0, 255, (128,), uint8)
>>> Пространство наблюдений для "rgb" — Box(0, 255, (210, 160, 3), uint8)
>>> Пространство наблюдений для "grayscale" — Box(0, 255, (210, 160), uint8)
```

Получается, что:

- `"rgb"` — матрица цветных пикселей разрешением 210x160;
- `"ram"` — текущие состояние оперативной памяти консоли;
- `"grayscale"` — матрица черно-белых пикселей разрешением 210x160.

Изображения, которые мы получаем не информативны для стандартных алгоритмов. Зато их хорошо воспринимает человек, а ещё нейросети 🙃.

Проверим, что нас не обманывают, выведем матрицу изображения на экран с помощью программы.

```
# Инициализируем начальное состояние среды и получаем начальное наблюдение.
observation, info = env.reset()

# Выводим начальное наблюдение (матрицу изображения экрана) в консоль.
print(observation)
```

На выходе программы мы получим матрицу следующего вида —

$$\begin{bmatrix} (225,50,50) & (228,111,111) & \dots \\ \vdots & \ddots & \\ (55,20,33) & & (100,150,30) \end{bmatrix}$$

Дополнительные наблюдения. Помимо явных наблюдений, таких как информация о пикселях на экране, оболочка дает доступ к дополнительным наблюдениям. Давайте напишем программу для вывода таких дополнительных наблюдений в консоль.

```
# Инициализируем начальное состояние среды и получаем начальное наблюдение.
observation, info = env.reset()

# Выводим начальное наблюдение (матрицу изображения экрана) в консоль.
print(info)

>>> {'lives': 3, 'episode_frame_number': 0, 'frame_number': 0}
```

Таким образом, наша программа ещё получает на вход: `lives`, `episode_frame_number`, `frame_number`. Мы также можем использовать эту информацию для обучения нейронной сети.

Параметры. Дополнительно мы можем настроить некоторые параметры приложения, в нашем случае это `difficulty` и `mode`. Ниже приведена таблица возможных значений и значений по умолчанию.

Доступные режимы	Режим по умолчанию	Доступные сложности	Сложность по умолчанию
`[0, 1, 2, 3]`	`0`	`[0]`	`0`

Награда. Самая важная часть оболочки, награда. Она поступает от приложения в ответ на какое-то действие нашей программы и поощряет наш алгоритм. На основе принципа поощрения и работает метод обучения с подкреплением, который мы будем реализовывать.

Наглядный пример поощрения.

Машинное обучение

В образовательных целях мы рассмотрим несколько алгоритмов машинного обучения для задачи обучения с подкреплением. Какие-то модели нам не подойдут вовсе, какие-то дадут плохой результат, а какие-то окажутся лучше человека.

Подход `Imitation Learning`

Если у вас есть эксперт, профессионал своего дела, который умеет решать задачу лучше всех, то вам несказанно повезло. Ведь вы можете взять его навыки и поместить их в алгоритм. Такой подход называется `Imitation Learning`, в нём алгоритм учится пародировать эксперта. В рамках нашей задачи экспертом будет выступать каждый из вас, вы попробуете обучить алгоритм тому, что знаете и умеете сами.

Установим перед написанием программы все необходимые библиотеки:

```
pip install pynput matplotlib catboost scikit-learn pandas numpy
```

Сначала знания нужно считать, для этого мы напишем отдельную программу. Она будет запускать среду и передавать нам управления. Наша задача показать наилучший результат, а программа будет записывать всё, что мы делаем в файл.

```
# Импортируем необходимые библиотеки и модули.
from pynput import keyboard
from pynput.keyboard import Key, KeyCode
import gymnasium as gym
import json
import os
import datetime

# Отображаемое действие для каждой клавиши.
keys_to_action = {KeyCode.from_char('w'): 1,
                  KeyCode.from_char('a'): 3,
                  KeyCode.from_char('s'): 4,
                  KeyCode.from_char('d'): 2}

# Папка для сохранения данных.
data_directory = 'data'

# Словарь для отслеживания нажатых клавиш.
keys_pressed = {}

# Функция обработки события нажатия клавиши.
def on_press(key):
    keys_pressed[key] = True

# Функция обработки события отпускания клавиши.
def on_release(key):
    keys_pressed[key] = False

# Создаем объект слушателя клавиатуры.
listener = keyboard.Listener(on_press=on_press, on_release=on_release)
listener.start()

# Список для сохранения данных об игре.
saved_data = []

# Проверяем наличие директории для сохранения данных и, если ее нет, создаем.
if not os.path.exists(data_directory):
```

```
  os.makedirs(data_directory)

# Создаем среду для игры.
env = gym.make("ALE/MsPacman-ram-v5", render_mode='human')

# Основной цикл, пока клавиша 'Esc' не нажата.
while not keys_pressed.get(Key.esc):
    observation, info = env.reset()
    observation, reward, terminated, truncated, info = env.step(0)

    # Основной цикл игры.
    while not terminated and not truncated:
        if keys_pressed.get(Key.esc):
            break

        action = 0
        # Определяем действие на основе нажатых клавиш.
        for key in keys_to_action:
            if keys_pressed.get(key):
                action = keys_to_action[key]

        # Выполняем действие и сохраняем данные.
        next_observation, reward, terminated, truncated, info = env.step(
            action)
        saved_data.append(([int(var) for var in observation], int(action)))
        observation = next_observation

# Сохраняем записанные данные в файл JSON.
print("[EXPERT RECORDER] Сохранение...")
filename = f'{data_directory}/expert_{datetime.datetime.now().strftime("%Y-%m-%dT%H-%M-%S")}.txt'
with open(filename, 'w') as file:
    file.write(json.dumps(saved_data))
print(f'[EXPERT RECORDER] Сохранено в {filename}')
```

Запустим программу, попробуем пройти игру или показать хороший результат.

> ⚠️ Программа будет работать бесконечно, завершить её можно, нажав на `esc`.

Но имейте ввиду, что чем больше данных, тем лучше конечный результат. Если у вас что-то не получилось, или вам просто лень собирать достаточное количество данных, то можете воспользоваться готовыми записями эксперта:

data.zip

Готовые данные для обучения, полученные при игре экспертом.

Отлично! Данные собраны, можно приступать к обучению. Мы будем решать задачу классификации табличного ML, предсказывать по текущему состоянию оперативной памяти консоли следующее действие. Да-да, в этой задачи мы немного хитрим и используем

оперативную память, а не изображение с приставки. В качестве модели классификации будет выступать `CatBoostClassifier`.

```
import os
import json
import numpy as np
import pandas as pd
import random
import gymnasium as gym
import matplotlib.pyplot as plt
from matplotlib import animation
from catboost import CatBoostClassifier
from sklearn.model_selection import train_test_split
from IPython import display

data_directory = 'data'
```

Теперь необходимо загрузить наши данные для обучения, для этого воспользуемся структурой `DataFrame` из библиотеки для работы с таблицами, `pandas`.

```
X, y = [], []
for filename in os.listdir(data_directory):
    if not filename.startswith('expert') or not filename.endswith('.txt'):
        continue
    with open(f'{data_directory}/{filename}', 'r') as file:
        content = json.load(file)
        memory, action = zip(*content)
        X.extend(memory)
        y.extend(action)
X, y = pd.DataFrame(X), pd.DataFrame(y)
```

Мы получили два объекта — `X`, данные для обучения, и `y`, необходимые действия. Мы не можем обучаться на всех данных, потому что нам нужно как-то контролировать процесс обучения, чтобы мы случайно не переобучились. Для этого мы будем постоянно валидироваться на тестовой выборке. Воспользуемся функцией `train_test_split`, чтобы её получить.

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, shuffle=True, stratify=y)
```

Приступаем к ключевому этапу, созданию и обучению модели. Обратите внимание на параметры: `iterations`, `depth`, `learning_rate`, `loss_function`. Их можно спокойно менять с целью увеличить качество предсказаний (уменьшить результат функции потерь).

> 💡 Наименьший `loss`, который получилось достичь у автора — `0.51`.

```
model = CatBoostClassifier(iterations=3500,
                           depth=8,
                           learning_rate=4e-2,
                           loss_function='MultiClass',
                           verbose=True)
model.fit(X_train, y_train, eval_set=(X_test, y_test))
```

Воспользуемся обученной моделью и визуально проверим качество обучения.

```
# Создание среды для игры.
env = gym.make("ALE/MsPacman-ram-v5", render_mode="rgb_array")

observations = []
observation, info = env.reset()
observation, reward, terminated, truncated, info = env.step(0)

while not terminated and not truncated:
    action = model.predict(observation)[0]
    observation, reward, terminated, truncated, info = env.step(action)
    observations.append(env.render())
```

Преобразуем сохраненные состояния в видео.

```
video = np.array(observations)

# Создание анимации для визуализации игры.
figure = plt.figure()
images = plt.imshow(video[0, :, :, :])
plt.close()  # это необходимо, чтобы изображение не отображалось на экране


def init():
    images.set_data(video[0, :, :, :])


def animate(i):
    images.set_data(video[i, :, :, :])
    return images

# Преобразование анимации в HTML5 видео и отображение его в блокноте.
html5_video = animation.FuncAnimation(figure, animate, init_func=init, frames=video.shape[0],
                                      interval=50).to_html5_video()
display.HTML(html5_video)
```

https://prod-files-secure.s3.us-west-2.amazonaws.com/b2d4c8bf-712c-403f-aaff-e7b8ee746693/ad028679-12a0-45b5-896e-cd90447c00c3/imler.mp4

Домашнее задание

В качестве домашнего задания к уроку, вам необходимо написать и обучить такого же бота самостоятельно. Попробуйте поменять параметры обучения и сделать прочие мелкие изменения, чтобы улучшить результат.

> ⚠️ Здорово, если вы будете пользоваться этими материалами, как подсказкой, и писать всё на свой лад.

А результатами вашей работы, записями действий эксперта и видеороликами прохождения ботом игры можно (**нужно**) делиться в общем чате.