Состав команды

- Панчишин Максим
- Балала Олег
- Астапов Илья

Архитектура

Архитектура агента была выбрана по аналогии с семинаром

```
agent = MLPClassifier(
    hidden_layer_sizes=(80, 80),
    activation="logistic",
    warm_start=True,
    max_iter=1,
)
```

Подход к обучению

В качестве интереса был выбран подход основанный на кросс энтропии. То есть были следующие этапы

1. Множественный запуск агента с эффектом случайности

```
for i in range(N_AGENTS):

    states=[]
    actions=[]
    sum_reward = 0
    while True:
        # Получаем предсказание
        proba = agent.predict_proba([collect_parametrs(action_terminal)]).squeeze()
        # Выбираем случайный вариант в соответствии с вероятностями
        action = np.random.choice(N_ACTIONS, p=proba)

        if action == 0:
            input_actions = [1, 0]
        else:
            input_actions = [0, 1]

        states.append(collect_parametrs(action_terminal))
        actions.append(action)

        # if i ==N_AGENTS-1:   You, Yesterday • Uncommitted changes
        #     _, reward, terminal = action_terminal.frame_step(input_actions,30)
        # else:
        _, reward, terminal = action_terminal.frame_step(input_actions, fps: None)

        sum_reward+=reward

        if terminal:
            break

    states_batch.append(states)
    actions_batch.append(actions)
    rewards_batch.append(sum_reward)
```

2. Отбор элитных сессий (набор из пар: состояние, действие), где агент показал себя лучше всего

```
elite_states,elite_actions = select_elites(states_batch,actions_batch,rewards_batch)
```

```
def select_elites(states_batch, actions_batch, rewards_batch, percentile = 70):  1 usage  new *
    '''
        Эта функция отбирает элитные сессии из батча
    '''

    elite_states = []
    elite_actions = []
    reward_threshold = np.percentile(rewards_batch, percentile)
    for session_states, session_actions, session_reward in zip(states_batch, actions_batch, rewards_batch):
        if session_reward < reward_threshold:
            continue
        elite_states.extend(session_states)
        elite_actions.extend(session_actions)

    return elite_states, elite_actions
```

3. Обучение агента на этих элитных сессиях

```
#Тренируем модель на элитных сессиях
agent.fit(elite_states,elite_actions)
```

Входные параметры агента

В качестве входных параметров в модель были выбраны следующие параметры из среды

1. Положение птицы по y
2. Положение передней трубы по x
3. Положение верхней части передней трубы по y
4. Положение нижней части передней трубы по y
5. Скорость персонажа по y

```
parametrs.append(action_terminal.playery)

playerMidPos = action_terminal.playerx + game.PLAYER_WIDTH / 2
for i in range(len(action_terminal.upperPipes)):
    pipeMidPos = action_terminal.upperPipes[i]['x'] + game.PIPE_WIDTH / 2
    if pipeMidPos <= playerMidPos:
        continue
    parametrs.append(action_terminal.upperPipes[i]['x'])
    parametrs.append(action_terminal.upperPipes[i]['y'])
    parametrs.append(action_terminal.upperPipes[i]['y'] + game.PIPEGAPSIZE)
    break

parametrs.append(action_terminal.playerVelY)
return parametrs
```

Награда агента

В качестве награды были заданы следующие действия/состояния агента

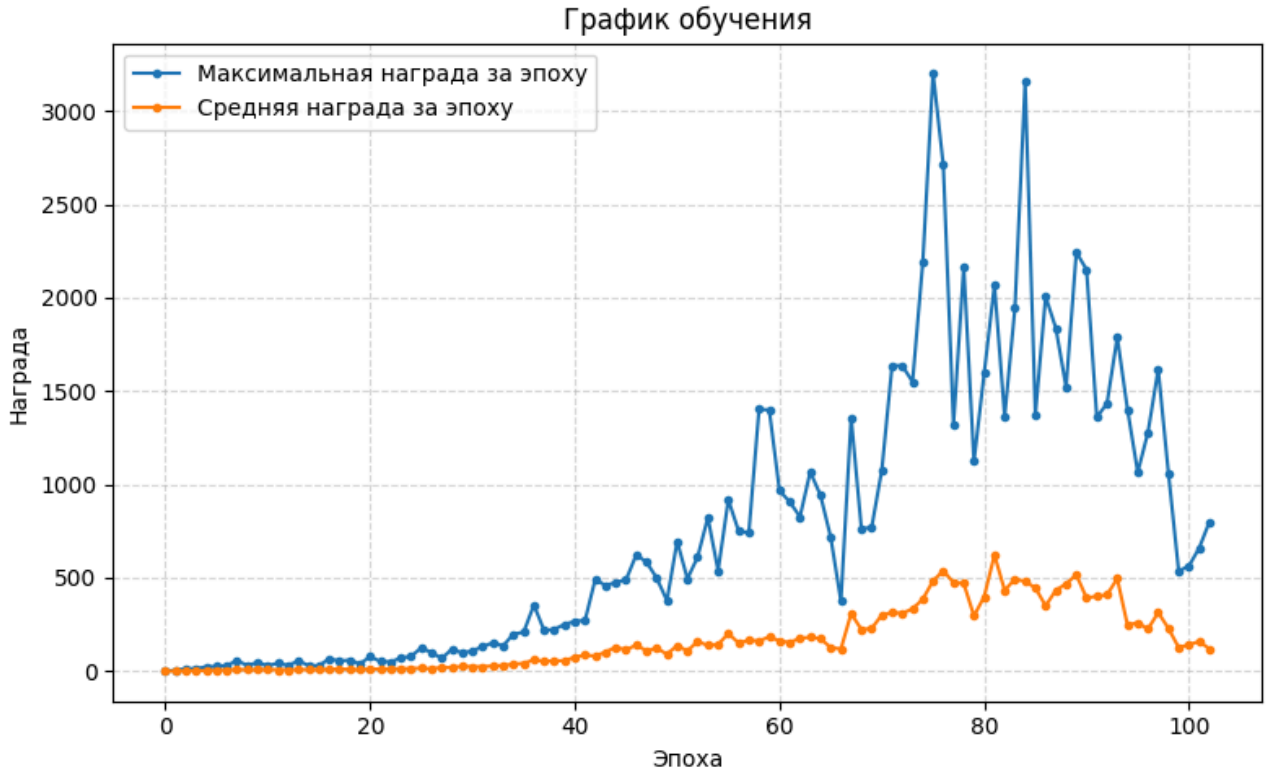
1. Небольшой штраф, если в качестве действия выбран прыжок
```
if input_actions[1] == 1:
    reward = -0.1
```
2. Небольшой награда, если птица находится в Y окрестности пространства промежутка от передней трубы
```
playerMidPos = self.playerx + PLAYER_WIDTH / 2
for i in range(len(self.upperPipes)):
    pipeMidPos = self.upperPipes[i]['x'] + PIPE_WIDTH / 2
    if pipeMidPos <= playerMidPos:
        continue
    if -self.upperPipes[i]['y'] < self.playery < self.lowerPipes[i]['y']:
        reward=0.1
        break
```
3. Большая награда, если птица увеличила score
```
for pipe in self.upperPipes:
    pipeMidPos = pipe['x'] + PIPE_WIDTH / 2
    if pipeMidPos <= playerMidPos < pipeMidPos + 4:
        self.score += 1
        reward = 10
```
4. Умеренное наказание при поражении
```
if isCrash:
    terminal = True
    self.__init__()
    reward = -1
```

Процесс обучения

Перед началом введу дополнительное определение. PIPEGAPSIZE - расстояние между верхней и нижней частями трубы

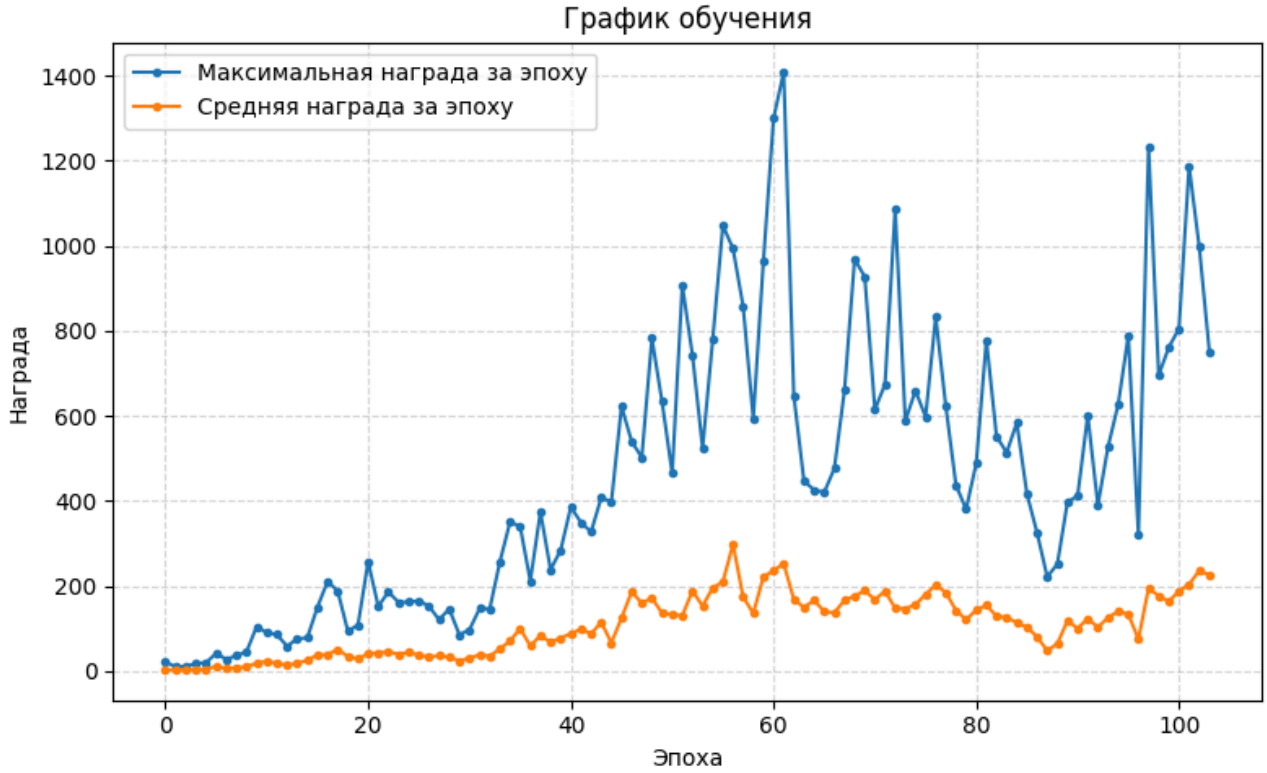
Изначально были попытки обучения сразу на ключевую задачу, т.е. пролёт агента при целевом параметру PIPEGAPSIZE = 100. Но агенту было тяжело начать обучение
Из-за возникших сложностей было принято решение начать обучение с больших значений PIPEGAPSIZE. А в дальнейшем уже использовать эту предобученную модель для обучения на PIPEGAPSIZE = 150, получения новых весов и обучения их на PIPEGAPSIZE = 100

Всё обучение производилось при помощи файла `study.py` и изменённого параметра PIPEGAPSIZE в файле `wrapped_flappy_bird.py`
Были получены следующие результаты:



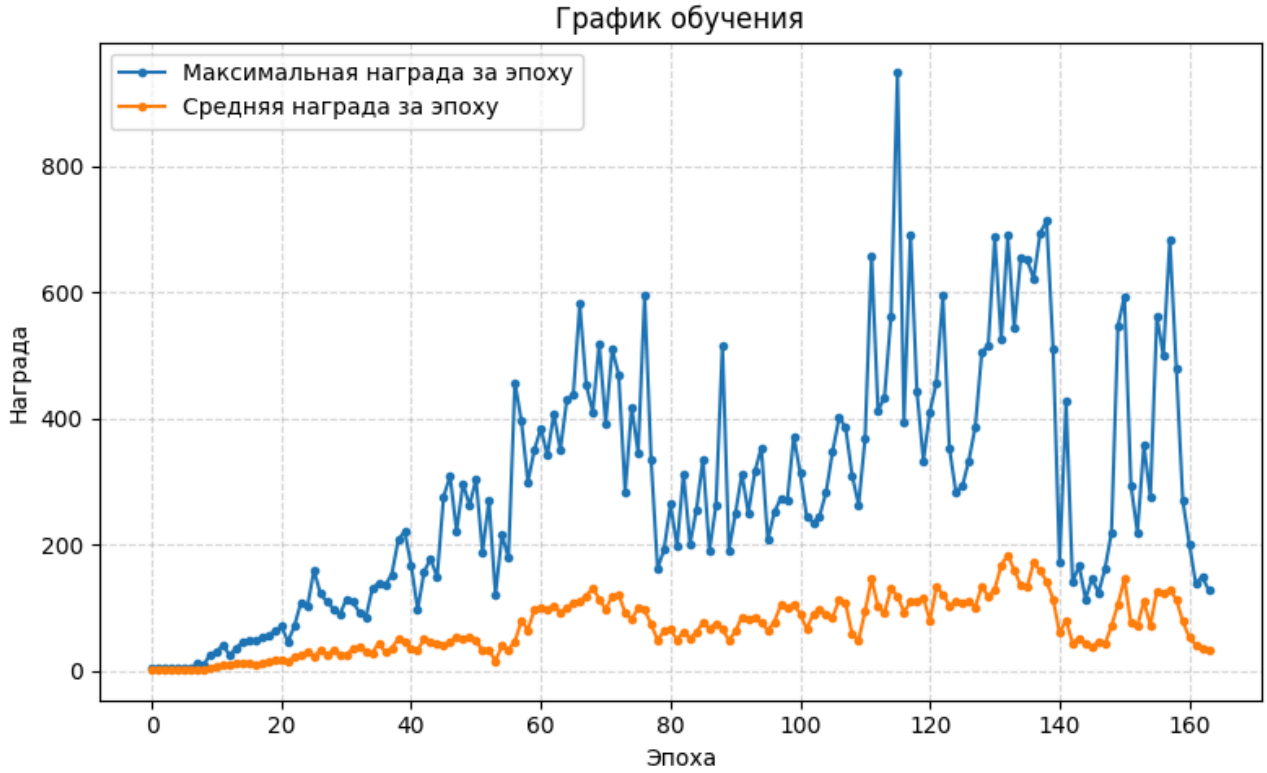
1.
2. Файл с весами модели, который находится по пути `weights/mlp_agent_200.pkl`
3. Видео, где показано, что агент способен пролетать 100 труб. Видео находится в файле `hw_6_200.mp4`
   Важное замечание: игра происходила с частотой в 60 кадров (FPS=60). Запись видео производилась с частотой кадров равной 30

После обучения данной агента был осуществлён переход к трубе с PIPEGAPSIZE = 150 и начальным весам из предыдущего пункта
Были получены следующие результаты:



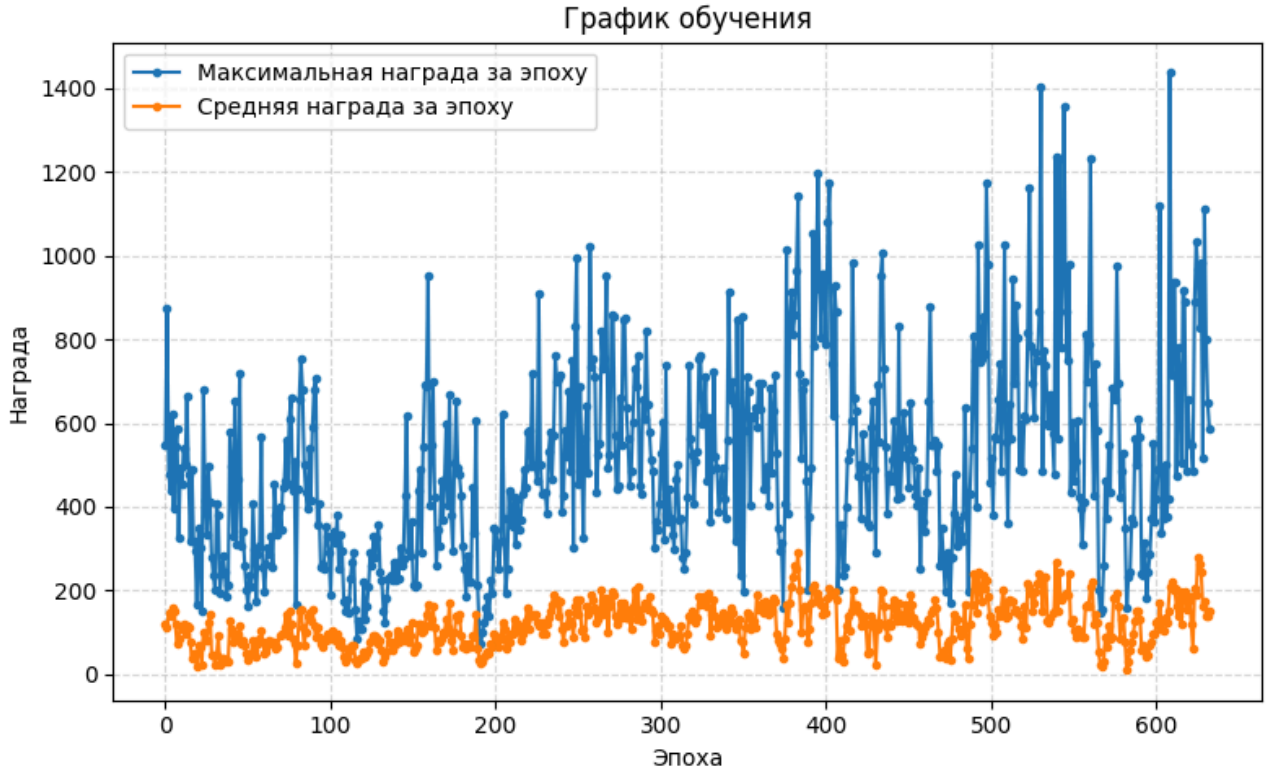
1.
2. Файл с весами модели, который находится по пути `weights/mlp_agent_150.pkl`

Затем производилось обучение агента с параметром PIPEGAPSIZE = 100 и весами из прошлого пункта
Оно осуществлялось в несколько подходов: сначала обучалась модель на 160 эпохах, а затем на 630.
Результаты:



1.
2. Файл с весами модели, который находится по пути `weights/mlp_agent_100_prom_save.pkl`

Затем на основе этих весов было совершено дообучение на 630 эпохах
Результаты:



1.

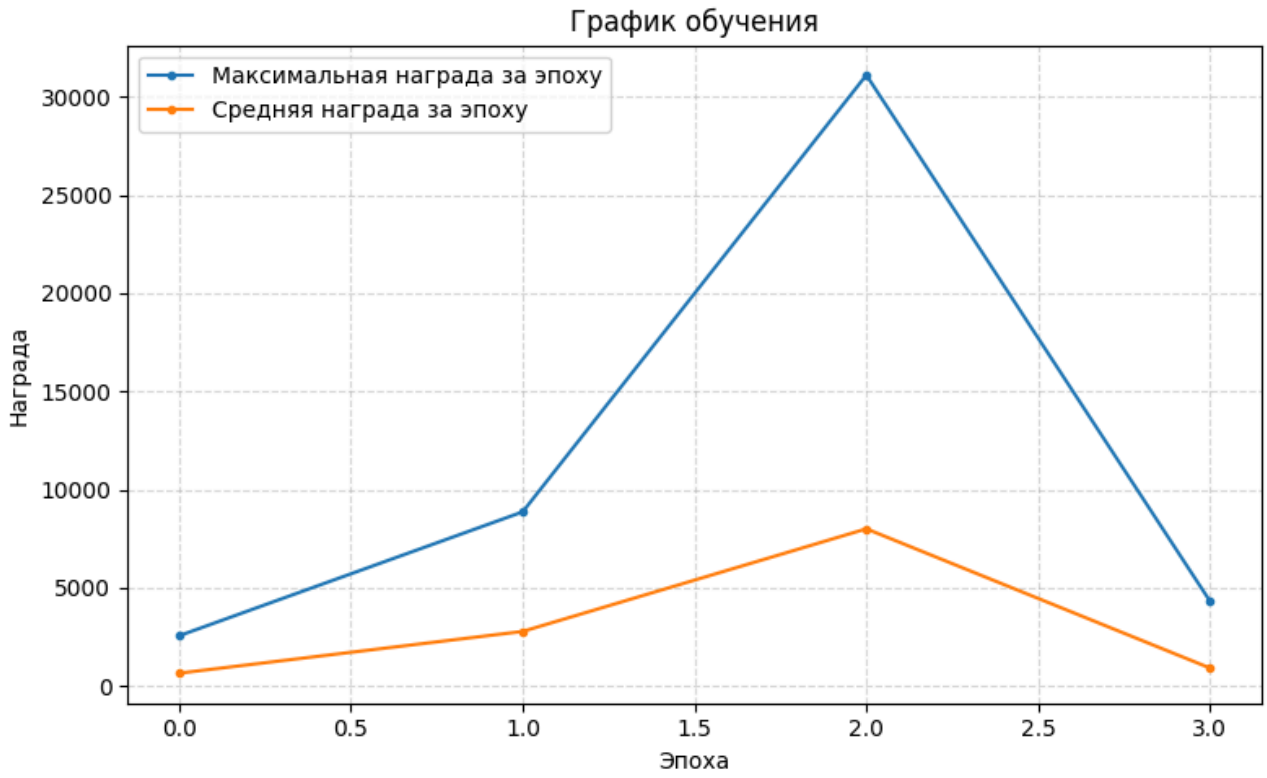
2. Файл с весами модели, который находится по пути `weights/mlp_agent_100_prom_save_2.pkl`

Полученный алгоритм был неплох (часто доходил до 20), но он был нестабильным, поэтому было принято решение дообучить его, но уже без случайного выбора ответа, а выбирать именно тот ответ, что он предсказывает скорее всего

До	После

`# Получаем предсказание`
`proba = agent.predict_proba([collect_parametrs(action_terminal)]).squeeze()`
`# Выбираем случайный вариант в соответствии с вероятностями`
`action = np.random.choice(N_ACTIONS, p=proba)`
	`# Получаем предсказание`
	`proba = agent.predict_proba([collect_parametrs(action_terminal)]).squeeze()`
	`# Выбираем вариант предсказанный моделью`
	`action = np.argmax(proba)`

Финальное дообучение агента происходило при помощи файла `study_without_random.py`
Такое обучение дало большие плоды и уже за пару эпох были получены веса, которые решают ключевую задачу (пролёт 100 труб)
Финальные результаты:



1.
2. Файл с весами модели, который находится по пути `weights/mlp_agent_100_final.pkl`
3. Видео, где показывается, что агент способен проходить 100 труб (в видео проходит полёт через 1100 труб). Видео находится в файле `hw6_100.mp4`
   Важное замечание: вся игра происходила с частотой в 60 кадров (FPS=60). Запись

видео производилась с частотой кадров равной 30. После достижения 107 трубы остаток видео был ускорен в 8 раз, чтобы сократить общую продолжительность видео