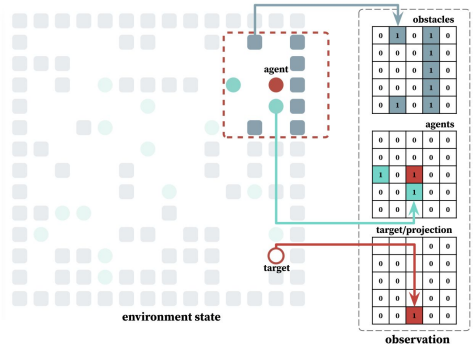
ДАННЫЕ

Partially observable multi-agent pathfinding (PO-MAPF)

На каждом временном шаге агент получает (локальное) наблюдение за окружающей средой и решает, какое действие предпринять. Конечная цель агентов — достичь своих целей, избегая столкновений друг с другом и статическими препятствиями.

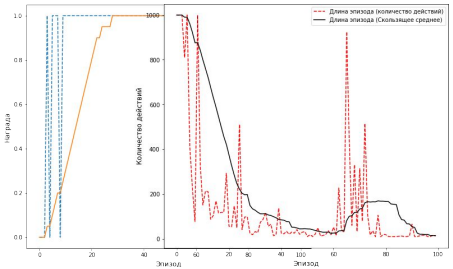


ИССЛЕДУЕМЫЕ ПОДХОДЫ

- DQ_learning
- Policy Gradient
- REINFORCE
- Baseline

REINFORCE:
Input (2, 3, 11, 11) - 2 состояния (текущий и предыдущий шаг)
Action - случайный выбор с учетом плотности вероятности output (5)
Метрика: финальную награду обратной итерацией распределяем на эпизоды с коэффициентом 0.98

Количество кадров около
1000 (random) → **10-12** (model)



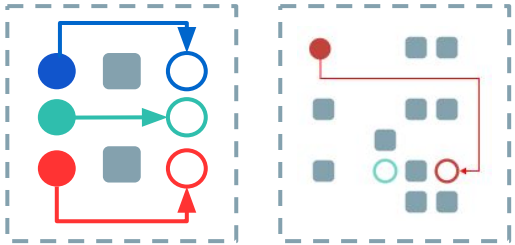
ФИНАЛЬНОЕ РЕШЕНИЕ

За основу взят предоставленный baseline. Модель доработана таким образом, чтобы агенты строили свой маршрут с учетом других агентов, что позволяет агентам избегать заторов и столкновений. С некоторой вероятностью агент определяет, обойти другого агента или остаться на месте.

- **Высокая скорость работы в сравнении с нейросетью**
- **Не требует обучения в отличие от нейросети**
- **Агенты не пересекаются при перемещении**
- **Агенты находят кратчайший путь до цели**
- **Агенты не зацикливаются в действиях**

CSR

0.2685714286 → 0.5514285714



ИССЛЕДУЕМЫЕ ПОДХОДЫ: АЛГОРИТМ REINFORCE

У нас есть состояние среды: obs[agent] = ['препятствия', 'положения агентов', 'цели агентов']

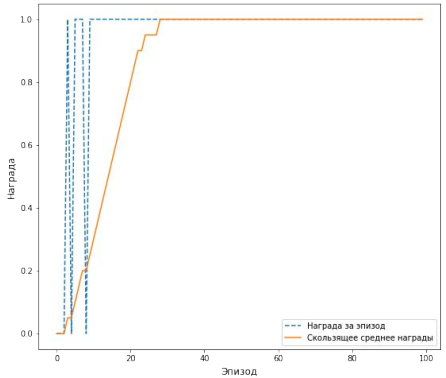
Разница между текущей средой и средой за предыдущий шаг - это как раз действия за кадр. Подадим в нейросеть на вход вектор из двух состояний - предыдущего и текущего, т.е. 2 x (3, 11, 11) -> (2, 3, 11, 11) В нейросети этот вектор пропустим через Flatten() слой и далее обработаем Dense слоями. Эти данные у нас будут на входе нейросети.

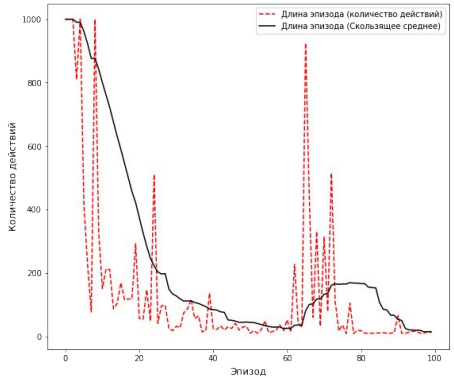
На выходе нейросети мы получаем распределение вероятности для действия агента. Мы берём не argmax, а случайную величину с вероятностью, пропорциональной плотности вероятности для данного действия - так система будет гибче к исследованию среды.

Метрика - **предобработанное вознаграждение** за эпизод. Если в конце эпизода мы получаем вознаграждение - то пробрасываем его обратной итерацией в начальные кадры с коэффициентом сохранения 0.98 (затухание 2%) и таким образом получаем награду за каждое действие.

Layer (type)	Output Shape	Param #
input_3 (InputLayer)	[(None, 2, 3, 11, 11)]	0
flatten_2 (Flatten)	(None, 726)	0
dense_6 (Dense)	(None, 1024)	744448
dense_7 (Dense)	(None, 128)	131200
dense_8 (Dense)	(None, 5)	645

Total params: 876,293
Trainable params: 876,293
Non-trainable params: 0





ИССЛЕДУЕМЫЕ ПОДХОДЫ: BASELINE

Алгоритм Policy Gradient с нейросетью очень долго обучается и по сравнению с бейзлайном вряд ли смог бы показать лучший результат. Поэтому предпочли сфокусироваться на улучшении бейзлайна. Вместо попыток улучшить алгоритм поиска цели, который и так довольно хороший, мы решили сфокусироваться на проблеме столкновений агентов друг с другом.

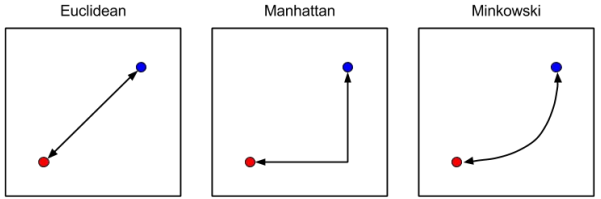
Мы пробовали разные способы для решения проблемы столкновения агентов:

- добавлять случайное действие вне зависимости от обстоятельств;
- определять наличие других агентов через состоянии среды и добавлять реакцию на это;
- смотреть на агентов только в непосредственной близости или в радиусе обзора;
- **и наконец тот способ, который сейчас (используем, оптимальную вероятность для расхождение агентов при встрече (эксперименты) и берём лучший путь от сочетания евклидова расстояния и минимальности шагов.)**

```
while len(self.OPEN) > 0 and steps < self.max_steps and (u.i, u.j) != self.goal:
    u = heappop(self.OPEN)
    steps += 1

    for d in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
        n = (u.i+d[0], u.j + d[1])

        if n in self.other_agents and np.random.random()<0.6:
            n = (u.i, u.j)

        if n not in self.obstacles and n not in self.CLOSED:

            h = (abs(n[0] - self.goal[0]) + abs(n[1] - self.goal[1])*steps)
            heappush(self.OPEN, Node(n, u.g + 1, h))
```

Меры расстояния

Мы пробовали разные меры расстояния, помимо расстояния манхеттена, в т.ч. квадратичное евклидово и минковски. Столкновения были при использовании метрики манхеттана, а вот при использовании евклидовой метрики агенты смогли разойтись. это без каких либо дополнительных изменений. но при этом манхеттен дал огромный прирост точности при комбирнации с методами обхода других агентов, а евклид не дал такого прироста.



РЕШЕНИЕ

Создан алгоритм определения расстояния до цели агента и агента - используем, оптимальную вероятность для расхождения агентов при встрече (эксперименты) и берём лучший путь от сочетания евклидова расстояния и минимальности шагов.

```
h = np.sqrt(((n[0] - self.goal[0])**2 + (n[1] - self.goal[1])**2)*steps)
```

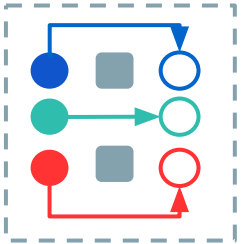
Были проделаны многочисленные тесты в Colab с усреднением для поиска лучшего сочетания. **Получили лучший результат при вероятности 0.6-0.62**

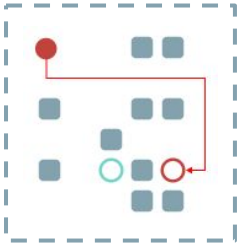
Алгоритм показал высокую точность и скорость работы, это можно увидеть в таблице ниже. Результат после доработки в сравнении начальным результатом baseline.

	CSR	ISR	FPS	makespan
после	0.5514285714	0.9137797619	222.9373473672	199.3852380952
до	0.2685714286	0.7675855655	177.9205560865	217.2409523810

Итог:

- ✅ **Высокая скорость работы в сравнении с нейросетью**
- ✅ **Не требует обучения в отличие от нейросети**
- ✅ **Агенты не пересекаются при перемещении**
- ✅ **Агенты находят кратчайший путь до цели**
- ✅ **Агенты не зацикливаются в действиях**





НАША КОМАНДА

Качалкин Артём

Data scientist

@anarakinson

Пузицкий Михаил

Data scientist

@MikePuzitsky

Хуторной Борис

Data scientist

@boris_khutornoi

Домненко Алексей

Data scientist
Full-stack

@domnenko_a_n