ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФГАОУ ВО НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа «Прикладная математика и информатика»

Отчет о программном проекте

на тему PathPlanning 2021: Автоматическое планирование траектории

Выполнил:
студент группы БПМИ202 _____ А.С. Афанасьева
Подпись И.О. Фамилия

17.05.2022
Дата

Принял:
руководитель проекта Яковлев Константин Сергеевич
Имя, Отчество, Фамилия

Доцент, Кандидат физико-математических наук
Должность, ученое звание

Факультет компьютерных наук / Базовая кафедра «Интеллектуальные технологии системного анализа и управления» ФИЦ «Информатика и управление» РАН
Место работы (Компания или подразделение НИУ ВШЭ)

Дата проверки 18.05 2022 _____ _____
Оценка Подпись
(по 10-тибалльной шкале)

Москва 2022

Реферат

Этот проект посвящен такой предметной области, как автоматическое планирование траектории, а именно её раздел о безопасно-интервальном планировании в среде с динамическими препятствиями (алгоритм SIPP и его модификации) и применении в задачах многоагентного планирования.

Целью проекта является знакомство с предметной областью планирования траектории и разработка программного обеспечения, содержащего реализацию одного из современных алгоритмов поиска пути в среде с динамическими препятствиями.

В проекте получилось реализовать алгоритмы планирования траектории в среде с динамическими препятствиями Sipp и Anytime Sipp. Они были протестированы на достаточно большой выборке тестов, сгенерированных случайным образом. Написан визуализатор алгоритма. Результаты тестирования оформлены в качестве таблиц и графиков. Реализация алгоритмов, исходные коды генератора, программы, проверяющей корректность планирования пути, и визуализатора, тесты и результаты тестирования лежат в репозитории по ссылке: https://github.com/afnastya/Sipp.

Перечень ключевых слов: PathPlanning, Sipp, Anytime Sipp.

Содержание

Основные термины, определения и сокращения 3
Введение 4
Задачи проекта 5
Обзор статей-источников 6
Описание требований к программному проекту 9
Требования к реализации алгоритма планирования пути 9
Требования к визуализатору 9
Теоретическая часть 10
Постановка задачи 10
Алгоритм Sipp 10
Anytime Sipp 11
Реализация проекта 13
Алгоритмы поиска пути 13
Выбор технологий 13
Описание входных данных программы 13
Описание структуры кода 15
Визуализатор 17
Выбор технологий 17
Описание алгоритма работы визуализатора 17
Тестирование 18
Генерация тестов 18
Описание алгоритма генерации тестов 18
Проверка корректности результата работы программы 18
Тестирование 18
Анализ полученных результатов 19
Алгоритм Sipp 19
Алгоритм Anytime Sipp 21
Заключение 22
Список источников 23

Основные термины, определения и сокращения

Агент — объект, для которого нужно спланировать траекторию движения.

A* — базовый алгоритм планирования траектории, впервые описанный в 1968 году Питером Хартом, Нильсом Нильсоном и Бертрамом Рафаэлем.

Dijkstra — базовый алгоритм планирования траектории, частный случай A*, в котором эвристическая функция тождественно равна 0. Изобретён Эдсгером Дейкстрой в 1959 году.

SIPP (Safe Interval Path Planning) — алгоритм планирования траектории в среде с динамическими препятствиями, предложенный Майком Филлипсом и Максимом Лихачевым на конференции ICRA в 2011 году.

MAPF(Multi-Agent Pathfinding) — задача многоагентного планирования траектории.

ПО — Программное обеспечение

ICRA(International Conference on Robotics and Automation) — международная конференция, специализирующаяся на робототехнике.

IROS(International Conference on Intelligent Robots and Systems) — международная конференция, специализирующаяся на робототехнике.

ICAPS(International Conference on Automated Planning and Scheduling) — международная конференция, специализирующаяся на автономном планировании.

IJCAI(International Joint Conference on Artificial Intelligence) — международное научное сообщество в области искусственного интеллекта.

AAAI (Association for the Advancement of Artificial Intelligence) — международное научное сообщество в области искусственного интеллекта.

XML(eXtensible Markup Language) — язык разметки; расширение файла, содержимое которого написано на этом языке.

Введение

Сейчас люди активно пытаются внедрить в нашу повседневную жизнь различных роботов, беспилотные автомобили и летательные аппараты. Чтобы успешно выполнять поставленные задачи, им нужно уметь самостоятельно строить оптимальный путь, чтобы добраться из начальной точки в их пункт назначения. В отличие от человека, для которого планирование траектории это довольно тривиальное занятие, агенту (роботу, навигатору или персонажу компьютерной игры) автономно построить маршрут не так просто, как кажется. Действительно, ведь нужно формализовать задачу, принимая во внимание большое количество факторов:

- размеры агента
- кинематические ограничения (не каждый робот может мгновенно остановиться перед внезапно появившимся препятствием, поэтому нужно учитывать размер тормозного пути)
- время работы программы (должно быть как можно меньше, чтобы агент мог двигаться непрерывно, а не подолгу останавливаться посреди дороги, для того чтобы проложить маршрут до следующей точки)
- агенту необходимо избегать столкновения не только со статическими препятствиями (деревья, стены зданий), но и с динамическими (проходящие мимо люди или другие роботы, проезжающие автомобили)
- необходимо избегать опасные участки среды, даже если путь через них намного меньше
- бесчисленное количество многих других аспектов

Для всего этого требуются различные сложные алгоритмы, поэтому и существует целая постоянно развивающаяся предметная область, без которой не может существовать современный мир, - **автоматическое планирование траектории**.

В задаче поиска пути на данный момент нет единого эталонного решения (алгоритма), использующегося повсеместно, что и обуславливает актуальность деятельности предметной области. Действительно, сейчас выходит много различных статей, активно проводятся конференции, посвященные этой теме, предлагаются разные варианты моделирования задачи, позволяющие упростить её. Такое количество проблем построения пути для агента породило большое число ответвлений и модификаций задач, которыми занимается область. Например, планирование траектории в среде с динамическими/статическими препятствиями, многоагентное планирование, планирование с учетом кинематических ограничений, планирование с учетом габаритов агента, планирование в непрерывной/дискретной среде и так далее.

Цель проекта — знакомство с предметной областью планирования траектории и разработка ПО, содержащего реализацию одного из современных алгоритмов поиска пути в среде с динамическими препятствиями.

Задачи проекта

- Изучение научной литературы о безопасно-интервальном планировании в среде с динамическими препятствиями (алгоритм SIPP и его модификации) и применении в задачах многоагентного планирования.
- Реализация одного из изученных алгоритмов планирования траекторий
- Разработка визуализатора
- Тестирование и отладка
- Проведение экспериментальных исследований на представительной выборке данных
- Оформление репозитория с полученным ПО, результатами исследований и необходимой документацией

Обзор статей-источников

- SIPP: Safe Interval Path Planning for Dynamic Environments

  Mike Phillips, Maxim Likhachev, 2011, ICRA

Статья авторов алгоритма SIPP, поэтому она является основополагающей для этого проекта. По сравнению с задачей поиска пути в статической среде здесь добавляется еще одно измерение — время. Важная идея алгоритма, позволяющая избежать дискретизации времени, — для каждой клетки представить новое измерение как конечное число чередующихся промежутков — safe interval (интервал времени, в который эту клетку не пересекает никакое препятствие) и collision interval (интервал времени, в который эту клетку что-то пересекает).

Дополнительные ограничения:

1. Для любой клетки s и её соседа s' верно, что cost(s, s') = time(s, s') (то есть стоимость перехода всегда равна времени перехода). Оптимизируется время.
2. Агент может ждать некоторое время на одном месте.

Сам алгоритм - модификация A*. Необходимые изменения: добавлен предподсчет временных промежутков для сетки; изменена функция getSuccessors(), возвращающая соседей клетки, так, чтобы учитывать время.

Доказывают оптимальность и полноту алгоритма.

- Anytime Safe Interval Path Planning for Dynamic Environments

  Venkatraman Narayanan, Mike Phillips, Maxim Likhachev, 2012, IROS

Статья от авторов SIPP, вышедшая через год. Предлагают Anytime SIPP: модифицируют SIPP так, чтобы он мог использоваться в реальном времени, то есть агент не тратит время в ожидании оптимальной траектории, а как можно скорее начинает движение в нужном направлении, позже получает уже оптимальное решение и подстраивается под него.

Для этого используют идеи ARA* (Anytime Repairing A*): искать путь, запуская алгоритм несколько раз, сначала для быстроты работы с большим весом эвристики, далее каждый раз уменьшаем вес эвристики, получая всё лучшие пути. В запусках после первого также используются имеющиеся вычисления с предыдущих, что уменьшает их время работы.

Доказывают полноту алгоритма и неравенство на конечный результат поиска с весом эвристики: найденный путь при работе алгоритма с весом эвристики w отличается от оптимального пути не более, чем в w раз. Чтобы перенести это неравенство из A* в weighted SIPP, нужно модифицировать SIPP, используя идеи CFDA-A*.

- Using State Dominance for Path Planning in Dynamic Environments with Moving Obstacles

Juan P. Gonzalez, Andrew Dornbush, Maxim Likhachev, 2012, ICRA

До этого в статьях было ограничение на функцию стоимости перехода из клетки в соседнюю. Мотивация авторов этой статьи — убрать это ограничение.

Модифицируют SIPP так, чтобы его можно было использовать на графах, где функция стоимости не равна времени (возможно, более сложна, например, если мы хотим учитывать количество топлива, которое требуется, чтобы добраться из клетки s в соседнюю s').

Вводят бинарное отношение: состояние s2 лучше, чем s1, где состояние — пара из клетки и временного интервала (смысл отношения: однозначно нет пути, проходящего через s1, лучшего для нас, чем оптимальный путь, проходящий через s2). Обобщают функцию стоимости из состояния s1 в s2 как сумму 2 функций (возможно разных): стоимости перехода из клетки состояния s1 в клетку состояния s2 и стоимости ожидания от одного временного интервала до другого. Из этого выводят все результаты и строят свой алгоритм GSIPP (Generalized SIPP).

Доказывают полноту и оптимальность алгоритма, показывают, что он также работает быстрее обычного SIPP.

- Any-Angle Pathfinding for Multiple Agents Based on SIPP Algorithm

Konstantin Yakovlev, Anton Andreychuk, 2017, ICAPS

В статье рассматривается комбинация нескольких ответвлений области планирования траектории - планирование в динамической среде (используется алгоритм SIPP), any-angle планирование и многоагентное планирование.

Представляется алгоритм AA-SIPP — это SIPP, модифицированный так, что убраны ограничения на угол движения агента (ранее можно было двигаться по горизонтали, вертикали и по диагонали т.е. всего 8 вариантов). Далее алгоритм применен для приоритизированного многоагентного планирования. Это один из подходов к решению задачи для нескольких агентов: агенты как-то сортируются и для каждого последовательно ищут свою траекторию движения, причем те, для кого уже нашли, добавляются в параметры среды, как будто бы они являются динамическими препятствиями. Конечно, в таком варианте нет гарантированной оптимальности суммы всех путей, но получается хоть какое-то решение задачи. Доказана полнота алгоритма.

- Multi-Agent Pathfinding with Continuous Time

Anton Andreychuk, Konstantin Yakovlev , Dor Atzmon, Roni Stern, 2019, IJCAI

Авторы статьи интегрируют идеи планирования в динамической среде с помощью алгоритма SIPP в решение задачи многоагентного планирования.

Представляют алгоритм CCBS: смесь модификации SIPP, учитывающей ограничения накладываемые алгоритмом CBS (Conflict-Based Search) и модификации CBS.

Доказывают полноту и оптимальность алгоритма, акцентируя внимание, что это первый алгоритм, решающий задачу MAPF с непрерывным временем, для которого доказана оптимальность.

- Lifelong Path Planning with Kinematic Constraints for Multi-Agent Pickup and Delivery

Hang Ma, Wolfgang Hönig, T. K. Satish Kumar, Nora Ayanian, Sven Koenig, 2019, AAAI

Статья посвящена комбинированию нескольких областей: многоагентное планирование, учет кинематических ограничений, задача MAPD(Multi-Agent Pickup and Delivery), смысл которой — непрерывно оптимально раздавать агентам цели, куда нужно добраться (например, чтобы доставить заказы клиентам), и строить им пути. Это сильно отличается от предыдущих рассмотренных мной задач. До этого речь шла о единичном случае построения траектории.

Основная цель этой статьи — усовершенствовать существующий алгоритм для задачи MAPD, называющийся TP (Token Passing). Для этого они используют модификацию SIPP — SIPPwRT(Safe Interval Path Planning with Reservation Table). Этот алгоритм учитывает кинематические ограничения агентов(скорости агентов). Далее они интегрируют его в TP, получая алгоритм TP-SIPPwRT.

- Multi-objective Conflict-based Search Using Safe-interval Path Planning

Zhongqiang Ren, Sivakumar Rathinam and Howie Choset, 2021

Комбинация нескольких ответвлений области планирования траектории - планирование в динамической среде (используется алгоритм SIPP) , многоагентное планирование (используется алгоритм CBS), а также многокритериальная оптимизация.

Представляют свои алгоритмы MO-CBS(Multi-objective CBS) и MO-SIPP(Multi-objective SIPP) — модификации CBS и SIPP, оптимизирующие траектории по многим параметрам одновременно (например, время и риск), интегрируют второй в первый, получая алгоритм MO-CBS-ts.

В этом проекте были выбраны для реализации алгоритмы Sipp и Anytime Sipp. Далее речь будет идти только о них.

Описание требований к программному проекту

- Кроссплатформенность: ПО должно корректно работать на Linux, Mac и Windows
- Надежность: программа должна корректно завершаться при любых входных данных. При входных данных неправильного формата должно быть выведено в консоль соответствующее сообщение и произведено корректное завершение программы.

Требования к реализации алгоритма планирования пути

- Языке программирования: C++
- Версии компилятора:
  Linux: GCC 10 или выше
  Mac: Apple LLVM version 11.0.3 или выше
  Windows: MinGW-w64 5.0.3 или выше
- Для сборки используется CMake 3.2 или выше
- На вход программе подаются XML файлы, содержащие следующую информацию:
  1. размеры карты
  2. начальная и конечная точка агента
  3. опции движения: можно ли ходить по диагонали или только по вертикали и горизонтали, можно ли срезать углы
  4. список препятствий, для каждого из которых дан список пар (координаты, время) , единственным образом восстанавливающих траекторию движения препятствия. Например:
- На выходе также должен получиться XML файл, содержащий следующую информацию:
  1. найден ли путь — поле может иметь значение «path found» или «path not found»
  2. если путь существует, то основные показатели:
     a. длина пути
     b. время работы программы
     c. количество итераций алгоритма
  3. путь агента - список из пар (координаты, время), единственным образом восстанавливающих траекторию движения. Например:

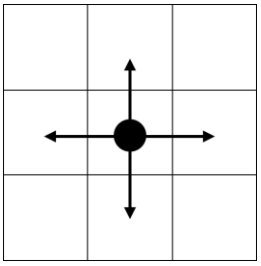
Требования к визуализатору

- Язык программирования: Python
- Версия: 3.8 и выше
- Используются встроенные модули Python и общедоступные библиотеки, которые могут быть установлены с помощью системы управления пакетами pip3.
- На вход подаются XML файлы, полученные в результате работы алгоритма (описаны выше)
- На выходе получается файл формата mp4, содержащий карту с движущимися точками-препятствиями и выделенным среди них агентом, для которого планировалась траектория.

Теоретическая часть

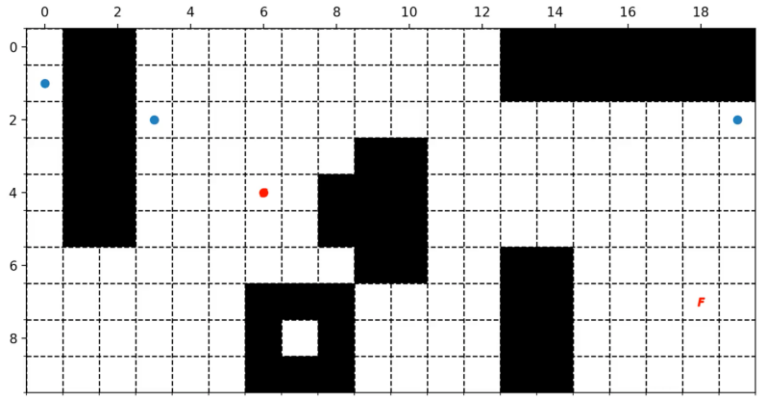
Постановка задачи

Пусть есть некоторая среда – двумерное поле со свободными и занятыми клетками, где занятые клетки – статические препятствия. На этом поле также есть динамические препятствия. У них нет размеров, поэтому будет считать, что это точки. Динамические препятствия двигаются равномерно и прямолинейно между клетками с общей стороной, причем из любой клетки они могут переходить только в 4 направлениях: вверх, вниз, вправо, влево. Переход между клетками занимает 1 единицу времени. При целом числе единиц времени любое динамическое препятствие должно находиться в центре какой-нибудь клетки.



Есть агент, для которого нужно найти путь из данной клетки S (start) в данную клетку F (finish). У агента нет размеров, поэтому тоже считаем, что агент – точка. Агент двигается равномерно и прямолинейно только по свободным клеткам, и так же, как и динамические препятствия, может переходить между клетками с общей стороной только в тех же 4 направлениях. Аналогично динамическим препятствиям, агент перемещается в соседнюю клетку за 1 единицу времени, и при целом числе времени должен находиться в центре какой-нибудь клетки.

Задача: для агента нужно найти путь из клетки S в клетку F, проходящий только по свободным клеткам, так что в любой момент времени (необязательно целый) координаты агента (необязательно целые) не совпадают ни с каким динамическим препятствием.

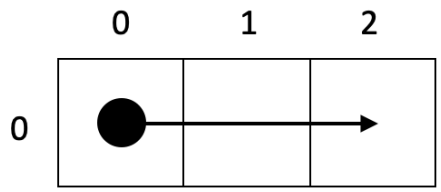


На рисунке слева пример поля на старте, белые клетки – свободные, черные – занятые. Красная точка – искомый агент, стоит на старте. Клетка, помеченная буквой F, – финиш, куда нужно прийти агенту. 3 синие точки – динамические препятствия на своих стартовых позициях.

Алгоритм Sipp

Как уже говорилось выше, временное измерение представляется как конечное число чередующихся промежутков — safe interval (интервал времени, в который эту клетку не пересекает никакое препятствие) и collision interval (интервал времени, в который эту клетку что-то пересекает).

Пример построения интервалов:



Пусть динамическое препятствие:

t = 0: в клетке (0, 0) (t – время)
t = 1: в клетке (1, 0)
t = 2: в клетке (2, 0)

Пусть также клетку (1, 0) больше никто никогда не пересекает. Тогда у клетки (1, 0) будут следующие временные интервалы:

[0, 0] – safe interval
(0, 2) – collision interval
[2, +∞) – safe interval

В алгоритмах планирования пути обычно на основе конфигурации клеток создаются состояния (node), в которых хранятся их координаты, **g-value**, **h-value** и **f-value** (определения даны ниже). Когда состояние только создаётся, попадает в контейнер OPEN, далее на каждой итерации из этого контейнера достается какое-то состояние (минимизируется f-value), “раскрывается” и перемещается в контейнер CLOSE. При “раскрытии” (node expansion) ищутся все состояния соседних клеток, в которые можно попасть из этого. Они создаются и добавляются в OPEN, если их еще не было. Проверяется, получается ли их g-value меньше, если перейти в них из раскрывающегося состояния.

g-value – текущее минимальное расстояние от старта до этого состояния

h-value – эвристика – положительное значение, оценивающее расстояние от этого состояния до финиша. Эвристика должна быть **допустимой (admissible)**, то есть обязательно меньше или равна оптимальному путь (другими словами, оценка снизу). Чтобы не нужно было “перераскрывать” состояния, эвристика также должна быть **монотонной** (**consistent**), то есть удовлетворять неравенству треугольника: $\forall$ s, s’ (s’ – наследник s) $h(s) \leqslant h(s') + cost(s, s')$. В этом проекте всегда будет использоваться манхэттенская эвристическая функция.

f-value – функция, минимизирующаяся при выборе состояния из OPEN на каждой итерации. $f(s) = g(s) + w * h(s)$, w – вес эвристики. В алгоритме Sipp вес эвристики всегда равен единице.

В алгоритме Sipp состояние идентифицируются координатами и номером safe interval. Для клетки вычисляются временные интервалы только при достижении этой клетки в ходе алгоритма, а не для всех перед началом работы алгоритма.

Anytime Sipp

Как говорилось выше, на выходе этого алгоритма получается не один, а несколько путей: с каждым путём расстояние найденного пути уменьшается, приходя в итоге к оптимальному. Для каждого пути вычисляется suboptimality bound – во сколько максимально длина этого пути больше длины оптимального пути. Созданные состояния используются в поисках следующих путей. Поиск следующего пути происходит в функции ImprovePath (названа так в статье), очень похожей на основную функцию алгоритма Sipp.

Еще одно весомое отличие от Sipp – в этом алгоритме состояния идентифицируются не только координатами клетки и номером интервала, но и булевым флагом, показывающем оптимальное состояние и или неоптимальное. Неоптимальные состояния отличаются от оптимальных функцией f.

Для оптимальных: $f(s) = g(s) + w * h(s)$

Для неоптимальных: $f(s) = w * (g(s) + h(s))$

Также для оптимальных состояний ищутся только оптимальные наследники, а для неоптимальных любые.

В алгоритме Sipp наследники, находящиеся в CLOSE пропускались, а здесь добавляется новый контейнер INCONS. В него попадают все состояния, для которых получилось найти меньшее g-value, несмотря на то, что они уже находились в CLOSE. Контейнер INCONS хранит такие состояния, они перемещаются в OPEN перед следующим вызовом ImprovePath.

Реализация проекта

Ссылка на репозиторий, где хранятся все материалы: https://github.com/afnastya/Sipp

Алгоритмы поиска пути

Выбор технологий

Реализация этой части проекта выполнена на языке C++. Дополнительно к проекту подключены библиотеки:

- **Tinyxml2** — парсер XML файлов на C++
- **Boost** — библиотека, состоящая из широкого спектра полезных дополнений к стандартной библиотеке. Был использован модуль **program options**, позволяющий парсить аргументы командной строки, и **boost::hash** (нужен был, чтобы вычислять хэш от std::tuple, так как в стандартной библиотеке такой функции нет)

Описание входных данных программы

На вход программе подается XML файл следующего вида:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<root>
   <map>
       <width>4</width>
       <height>3</height>
       <startx>1</startx>
       <starty>3</starty>
       <finishx>4</finishx>
       <finishy>1</finishy>
       <grid>
           <row>0 1 1 0</row>
           <row>0 0 0 0</row>
           <row>0 1 1 0</row>
       </grid>
       <dynamicobstacles>
           <obstacle id="1">
               <point x="4" y="3" time="0"/>
               <point x="4" y="2" time="1"/>
               <point x="1" y="2" time="4"/>
               <point x="1" y="1" time="5"/>
           </obstacle>
       </dynamicobstacles>
   </map>
</root>
```

<map> – тег открывает секцию, описывающую карту
<height> и **<width>** – высота и ширина карты соответственно.
<startx>, **<starty>** – координаты старта агента, для которого будет строиться путь
<finishx>, **<finishy>** – координаты финиша агента
<grid> – описание свободных и занятых статическими препятствиями клеток карты.

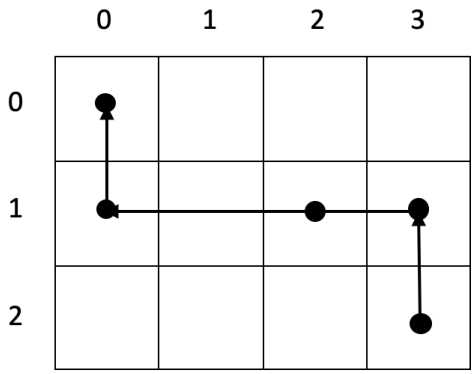
<row> – одна строчка карты, значение клетки может быть равным 1, если в клетке статическое препятствие, или 0, если клетка свободна

<dynamicobstacles> - описание траекторий движения динамических препятствий

<obstacle> – внутри этого тега описание пути препятствия, таких тегов может быть несколько в теге <dynamicobstacles>

Траектория пути представляется в виде последовательности троек из целых неотрицательных чисел (x, y, time) – координаты клетки и время, в которое препятствие или агент там находится. Эта тройка записывается в XML файле как атрибуты тега <**point**>. Любые 2 соседние тройки должны образовывать либо прямолинейный отрезок пути так, что время движения между этими координатами должно быть в точности равно разнице координат времени в этих тройках, либо координаты клетки в соседних тройках должны быть равны, а координаты времени отличаются на целое число больше 0.

Пример представление пути:

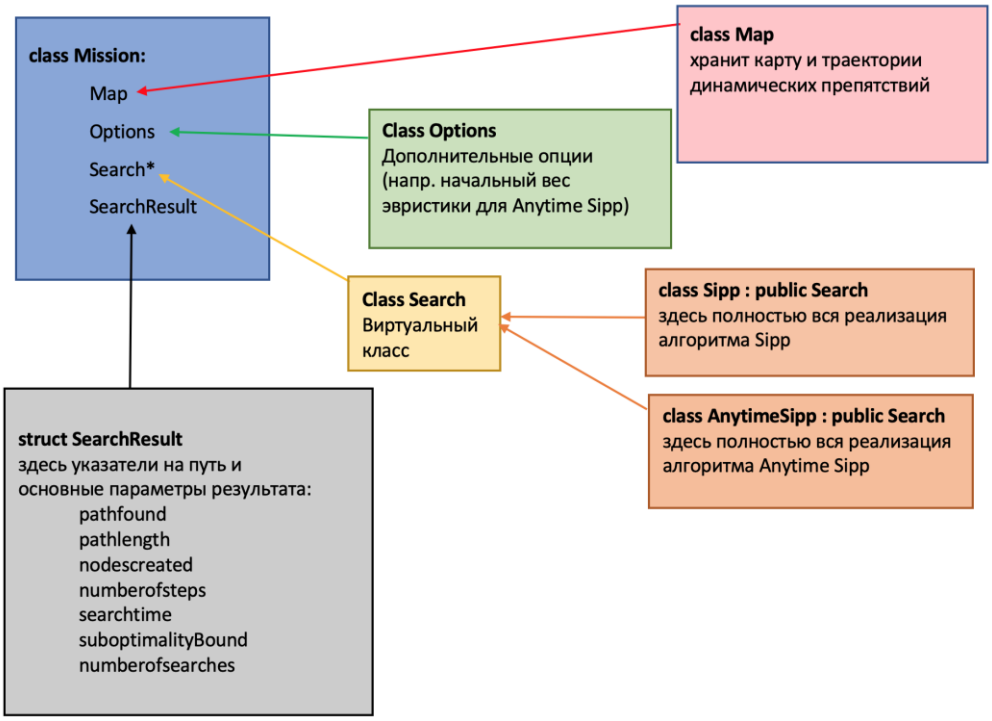


Пусть на рисунке траектория движения препятствия (или агента), начинающаяся в клетке (3, 2), конец – в клетке (0, 0). Пусть также в клетке (2, 1), он ждал 2 единицы времени. Тогда будет записано так:

```
<obstacle id="1">
    <point x="3" y="2" time="0"/>
    <point x="3" y="1" time="1"/>
    <point x="2" y="1" time="3"/>
    <point x="2" y="1" time="5"/>
    <point x="0" y="1" time="7"/>
    <point x="0" y="0" time="8"/>
</obstacle>
```

Описание структуры кода

Весь код этого раздела проекта хранится в папке Src/PathPlanning репозитория.



На данном изображении диаграмма основных классов.

- **Mission** – класс-интерфейс, через который ведется всё взаимодействие с программой. У этого класса есть методы такие, как загрузить задание, начать поиск пути, вывести результаты в консоль или в XML файл с результатом и т.д.
- **Map** – класс, владеющий информацией о карте и динамических препятствиях. Карта хранится как матрица, каждый элемент которой с индексами (i, j) – вектор из безопасных интервалов клетки (i, j). Интервал - структура из времени начала и конца, а также булевого флага, показывающего безопасен ли он. Для свободных клеток хранятся только безопасные интервалы, а для занятых статическими препятствиями хранится 1 collision interval со значениями начала и конца времени – 0 и бесконечность соответственно
- **Sipp** и **AnytimeSipp –** основные классы проекта, хранящие реализацию алгоритмов Sipp и Anytime Sipp соответственно.
  Для контейнеров OPEN и INCONS использовалось упорядоченное множество (std::set). Контейнер CLOSE – хэш-таблица (std::unordered_set).

Основные параметры результата (термины понадобятся при анализе результатов тестирования):

1. pathfound – значение true, если путь найден, false, если нет
2. pathlength – длина найденного пути или 0, если не найден
3. nodescreated – количество созданных состояний (node)
4. numberofsteps – количество итераций алгоритма, то есть число раз, когда из контейнера OPEN вынимается node с минимальным значением F и раскрывается
5. searchtime – время работы программы

6. suboptimalityBound – параметр нужен для Anytime Sipp, показывает во сколько максимально отличается длина найденного пути от длины оптимального пути
7. numberofsearches – параметр нужен для Anytime Sipp, количество вызовов функции ImprovePath

Визуализатор

Выбор технологий

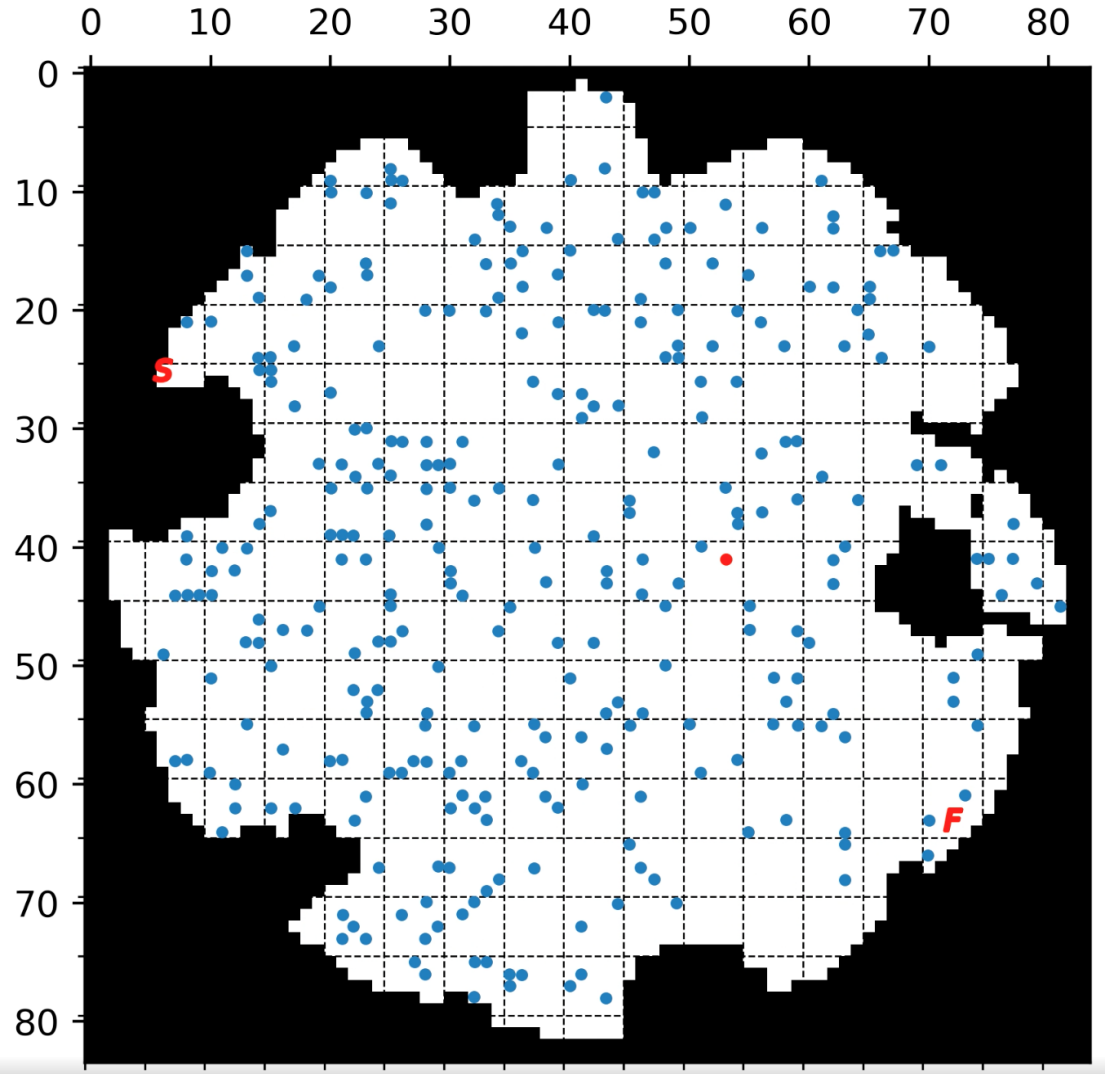
Программа написана на языке Python. Помимо встроенных модулей использованы библиотеки:

- **BeautifulSoup** — парсер XML файлов
- **MatPlotLib** — библиотека, позволяющая получить графики и анимировать их
- **FFMPEG** — библиотека позволяет преобразовать анимацию, сделанную с помощью matplotlib, в видео формата mp4.

Описание алгоритма работы визуализатора

Программа принимает на вход XML файл, полученный на выходе работы программы, планирующей путь. Парсит этот файл, заполняя поля класса Map. Далее с помощью matplotlib делает анимацию и с помощью FFMPEG сохраняет её в качестве видео.

Изображение ниже – скриншот полученного видео. Больше примеров можно найти в папке Examples/videos. Белые клетки – свободные клетки, синие точки – динамические препятствия, красная – агент, буквы S и F стоят в клетках старта и финиша агента соответственно.



Тестирование

Генерация тестов

Для тестирования были взяты городские карты разных размеров с ресурса: https://movingai.com/benchmarks/street/index.html. На этих картах изображены небольшие фрагменты известных городов (Москва, Лондон, Париж и т.д.), на картах есть только статические препятствия (обычно здания). Для перевода этих карт в формат входных данных, описанный в пункте о входных данных алгоритма, и для добавления траекторий динамических препятствий был написан специальный генератор. Все траектории не пересекаются между собой, а начальная и конечная точки достаточно удалены (в манхэттенской метрике больше стороны карты). Код хранится в папке Tests.

Описание алгоритма генерации тестов

Генератор принимает на вход карту или директорию из карт и необходимое количество динамических препятствий. Преобразует карту в формат, как в теге <grid>, описанный в пункте.. Далее случайно выбирает начальную и конечную точку траектории агента, для которого нужно найти путь в этом тесте. Точки выбираются так, что расстояние между ними достаточно большое (в манхэттенской метрике больше стороны карты). Затем генерируем траектории динамических препятствий. Для каждого препятствия случайно выбираем начальную точку, далее, пока время его пути меньше удвоенной стороны карты, определяем свободные клетки куда оно может прийти по прямой, случайно выбираем клетку, проверяем, что новый отрезок пути не пересекается ни с какой из уже построенных траекторией препятствий.

Проверка корректности результата работы программы

Код лежит в папке Tests. На вход принимается xml файл, полученный на выходе работы программы, планирующей путь. Проверяется, что найденный путь имеет корректное представление, не пересекается со статическими и динамическими препятствиями.

Тестирование

Всего с указанного выше ресурса были взяты по 30 карт размеров 256x256, 512x512 и 1024x1024, для каждой из карт было сгенерировано задание с 10, 20, 50, 100, 200, 500, 1000 динамических препятствий (по 5 тестов для каждой конфигурации). Итого получилось 3150 тестов. Все задания лежат в репозитории в папке TestsData. Тестирование Sipp производилось на всех 3150 тестов, а тестирование Anytime Sipp на каждом из этих тестов запускалось по 3 раза (с разными начальными весами эвристики: 2, 5 и 10).

Анализ полученных результатов

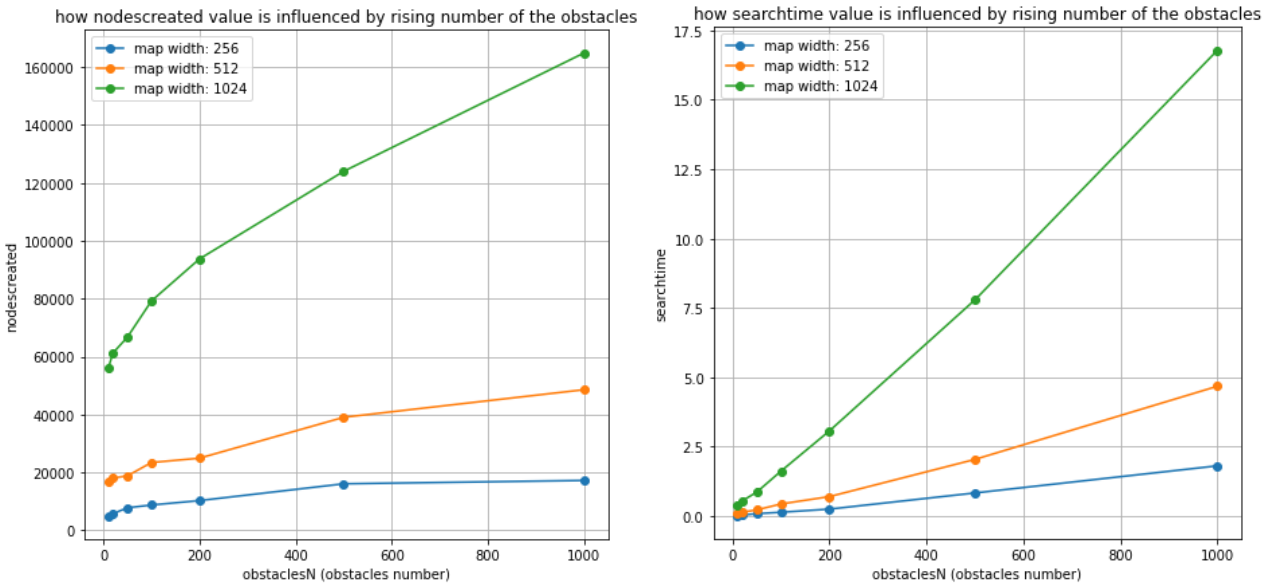
Алгоритм Sipp

Предварительно выбрасываем все результаты, в которых не удалось найти путь. В таких заданиях пути, действительно, нет, так как начальная и конечная точка находятся в разных компонентах связности.

В следующей таблице представлены данные, сгруппированные по размеру карты и количеству динамических препятствий и далее усредненные по каждому из параметров (found - был ли найден путь, length - длина пути, nodescreated - количество созданных состояний, numberofsteps - количество итераций алгоритма, searchtime - время работы алгоритма).

		found	length	nodescreated	numberofsteps	searchtime
width	**obstaclesN**					
256	10	1.0	309.924658	5041.191781	4172.493151	0.029319
	20	1.0	320.979730	5763.925676	4559.763514	0.041429
	50	1.0	321.653061	7780.156463	5491.659864	0.089862
	100	1.0	313.527027	8751.777027	5020.682432	0.142150
	200	1.0	314.539568	10277.539568	4902.877698	0.255010
	500	1.0	313.360544	16076.802721	7201.047619	0.837235
	1000	1.0	315.751773	17234.936170	7438.815603	1.813978
512	10	1.0	636.791667	16598.701389	14896.201389	0.107289
	20	1.0	621.184932	18056.958904	15825.623288	0.143465
	50	1.0	626.699301	18814.153846	14792.741259	0.227395
	100	1.0	629.551020	23431.755102	16452.727891	0.440166
	200	1.0	621.554795	24928.356164	14751.465753	0.705970
	500	1.0	625.678082	39054.356164	18599.034247	2.046293
	1000	1.0	638.027211	48585.319728	20922.272109	4.681575
1024	10	1.0	1263.287671	55955.917808	52841.013699	0.413633
	20	1.0	1235.306122	61244.884354	54993.360544	0.551332
	50	1.0	1270.109589	66732.082192	57155.801370	0.867240
	100	1.0	1256.061224	79222.265306	65103.034014	1.623834
	200	1.0	1256.172414	93757.689655	67844.703448	3.075061
	500	1.0	1270.500000	124018.694444	71743.159722	7.794057
	1000	1.0	1260.601351	164900.702703	80169.858108	16.785277

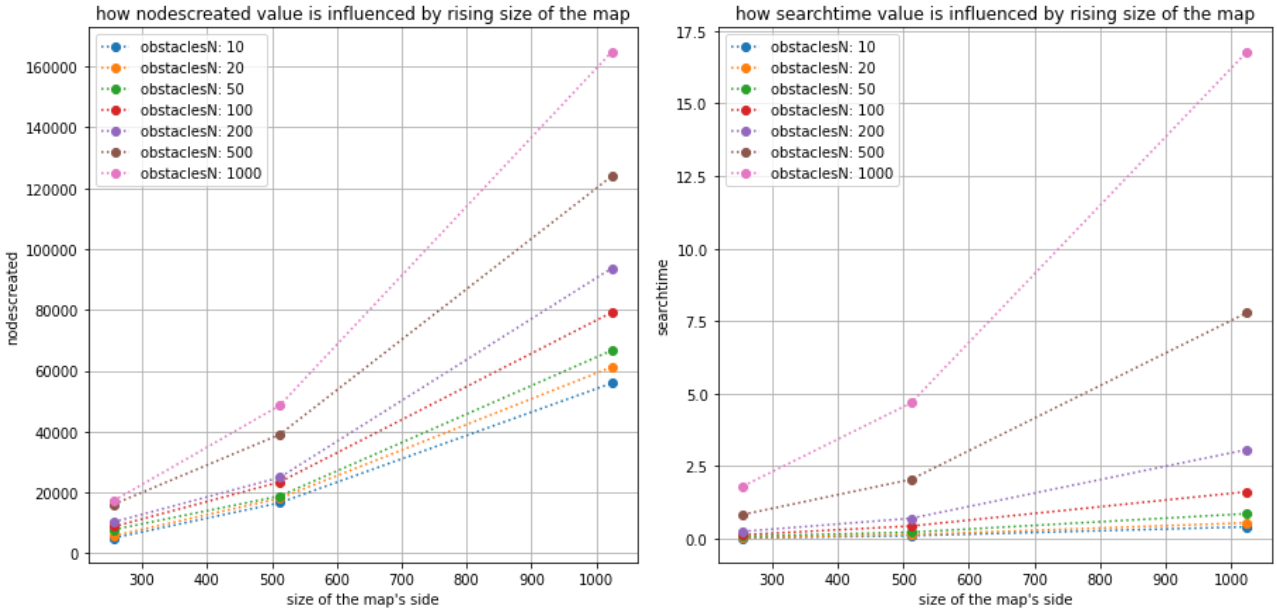
Таким образом, путь в заданиях с любой конфигурацией ищется в среднем менее, чем за 17 секунд. А для карт 256x256 алгоритм работает в среднем менее 2 секунд для любого количества препятствий до 1000.



Чтобы проанализировать, как зависит количество создающихся состояний (nodescreated) и время работы алгоритма от количества динамических препятствий, построим графики. Здесь на левом графике каждая кривая показывает зависимость усредненного числа созданных при работе алгоритма состояний среди тестов с картами одного размера от числа динамических препятствий на них. На правом - зависимость усредненного времени работы алгоритма среди тестов с картами одного размера от числа динамических препятствий.

Из этих графиков видно, что с ростом количества динамических препятствий, время работы алгоритма и количество создающихся состояний растет линейно.

Далее посмотрим, как зависит количество создающихся состояний (nodescreated) и время работы алгоритма от размера стороны карты. Здесь на левом графике каждая кривая показывает зависимость усредненного числа созданных при работе алгоритма состояний среди тестов с одним количеством динамических препятствий от размера стороны карты. На правом - зависимость усредненного времени работы алгоритма среди тестов с одним количеством динамических препятствий от размера стороны карты.



Конечно, по трем точкам мы не можем делать явные выводы о зависимости, но по этим графикам, можно предположить, что при больших числах динамических препятствий, эти зависимости всё более похожи на экспоненциальные. К сожалению, в этом датасете городских карт нет других размеров – ни размеров более 1024x1024, ни промежуточных размеров карт, ни карт меньше 256x256, поэтому проверить эту гипотезу не удалось.

Алгоритм Anytime Sipp

Построим аналогичную таблицу как для алгоритма Sipp.

		found	length	nodescreated	numberofsteps	searchtime	firstSearchtime
width	obstaclesN						
256	10	1.0	309.924658	7922.753425	8938.589041	0.078624	0.008304
	20	1.0	320.979730	8918.335586	10407.837838	0.101844	0.012746
	50	1.0	321.653061	12645.807256	13229.741497	0.177772	0.024667
	100	1.0	313.527027	14035.297297	10870.153153	0.232068	0.035828
	200	1.0	314.539568	16224.314149	10157.541966	0.361057	0.063534
	500	1.0	313.360544	25733.473923	15707.079365	1.045522	0.193743
	1000	1.0	315.751773	27887.321513	15240.763593	2.106191	0.365626
512	10	1.0	636.791667	25707.238426	31456.435185	0.294194	0.024357
	20	1.0	621.184932	28655.121005	37115.493151	0.372154	0.032240
	50	1.0	626.699301	29717.223776	30528.139860	0.442043	0.049478
	100	1.0	629.551020	37313.292517	38066.727891	0.750016	0.094731
	200	1.0	621.554795	40002.518265	30978.317352	1.025645	0.143562
	500	1.0	625.678082	61434.536530	37876.924658	2.589699	0.387165
	1000	1.0	638.027211	76121.176871	41781.895692	5.485540	0.776011
1024	10	1.0	1263.287671	102685.360731	147660.867580	1.606300	0.105296
	20	1.0	1235.306122	114494.959184	149624.712018	1.745716	0.113211
	50	1.0	1270.109589	119077.057078	148127.255708	1.816613	0.174891
	100	1.0	1256.061224	142695.335601	161674.786848	3.279159	0.344021
	200	1.0	1256.172414	153935.133333	165024.105747	4.971380	0.657914
	500	1.0	1270.500000	204245.481481	171083.708333	10.954093	1.443444
	1000	1.0	1260.601351	258925.504505	168536.632883	21.091985	2.439664

Таким образом, путь в заданиях с любой конфигурацией ищется в среднем менее, чем за 22 секунды. А для карт 256x256 алгоритм работает в среднем менее 2.5 секунд для любого количества препятствий до 1000.

В целом, время работы Anytime Sipp, хоть и незначительно, но всё-таки больше, чем время работы Sipp. Но, это компенсируется тем, что первый путь, полученный Anytime Sipp, находится в разы быстрее работы Sipp. А следовательно, видно большое преимущество Anytime версии алгоритма: агент может сразу начать двигаться, а не ждать на месте, пока закончится поиск оптимального пути. Особенно, это важно на больших тестах, например, на картах 1024x1024 с 1000 препятствиями, где оптимальный путь ищется в среднем более 20 секунд.

Заключение

Таким образом, в проекте были успешно реализованы алгоритмы планирования траектории в среде с динамическими препятствиями Sipp и Anytime Sipp. Алгоритмы протестированы на достаточно большой выборке тестов, сгенерированных случайным образом. Написан визуализатор алгоритма. Результаты тестирования оформлены в качестве таблиц и графиков. Всё это лежит в репозитории по ссылке: https://github.com/afnastya/Sipp.

В дальнейшем, есть множество вариантов, как можно было бы продолжить развивать проект.

В первую очередь, можно было бы дополнить уже реализованные алгоритмы некоторыми опциями. Сейчас у агентов и препятствий всего 4 варианта направления движения. Предлагается увеличить это число до 8, добавив диагональные ходы. Тогда можно будет добавить выбор разных опций ходов – например, может ли агент срезать углы. А также можно будет выбирать разные эвристики: диагональную, евклидову и т.п.

В общем, можно продолжить реализовывать алгоритмы из списка, описанного в пункте “Обзор статей-источников” этого отчета, то есть добавить в проект другие модификации Sipp. Например, комбинации Sipp с алгоритмами решения задачи многоагентного планирования, описанные в последних трех статьях этого раздела.

Также можно развить проект в другом направлении, отбросив условие о том, что у препятствий и агента нет размеров, и добавив в программу учет этих параметров для агентов и динамических препятствий.

Список источников

1. M. Phillips. Sipp: Safe interval path planning for dynamic environments / M. Phillips, M. Likhachev // IEEE International Conference on Robotics and Automation (ICRA). IEEE – 2011.
2. V. Narayanan. Anytime safe interval pathplanning for dynamic environments / V. Narayanan, M. Phillips, M. Likhachev // RSJ International Conference on Intelligent Robots and Systems, IEEE – 2012 – c. 4708–4715.
3. J. P. Gonzalez. Using state dominance for path planning in dynamic environments with moving obstacles / J. P. Gonzalez, A. Dornbush, and M. Likhachev // ICRA, IEEE – 2012.
4. M. Phillips. Planning in domains with cost function dependent actions / M. Phillips, M. Likhachev // AAAI Conference on Artificial Intelligence – 2011
5. K. Yakovlev. Any-angle pathfinding for multiple agents based on SIPP algorithm / K. Yakovlev, A. Andreychuk // International Conference on Automated Planning and Scheduling (ICAPS) – 2017 – c. 586.
6. A. Andreychuk. Multi-agent pathfinding with continuous time / A. Andreychuk, K. Yakovlev, D. Atzmon, R. Stern // arXiv:1901.05506v3 [cs.AI] – 2019
7. H. Ma. Lifelong path planning with kinematic constraints for multi-agent pickup and delivery / Hang Ma, Wolfgang Hönig, T. K. Satish Kumar, Nora Ayanian, Sven Koenig / AAAI Conference on Artificial Intelligence – 2019
8. Z. Ren. Multi-objective Conflict-based Search Using Safe-interval Path Planning / Zhongqiang Ren, Sivakumar Rathinam, Howie Choset // arXiv:2108.00745 – 2021.
9. Sturtevant, N. R. Benchmarks for grid-based pathfinding // IEEE Transactions on Computational Intelligence and AI in Games, vol.4,n.*2* – 2012 – c.144–148