# Алгоритмы построения пути

Ст. преп. каф. ПМП Пучкин Максим Валентинович

# Ограничения

- Пространство поиска – 2D (на 3D обобщается без особых проблем). Пример – DOOM 1-2;
- Область навигации (сцена) – статична (препятствия, непроходимые области не меняются). Разрушаемость, перемещение объектов, ботов на алгоритмы не влияют;
- Проблемы конфликтов агентов не рассматриваем;
- Поверхность имеет различную стоимость перемещения агентов – в зависимости от типа поверхности (земля, песок и т.д.), опасности для агента (PUBG);
- Известны точные положения агента и цели, есть возможность оценить оставшееся расстояние (евклидово расстояние);
- Пространство поиска может быть представлено сеткой (grid или mesh), с известным расстоянием между узлами.

# Стратегия поиска

path Pathfinder(state st, state fin)

1. Оценить st : st.f = st.g + h(st, fin) (для начальной st.g = 0)
2. foreach node in {Nodes}: node.f = +infinity
3. [OPENED].push(s);
4. while not [OPENED].empty:
    1. node = [OPENED].pop();
    2. if node == fin break;
    3. [CLOSED].push(node);
    4. foreach child in childs(node):
        if child.f > node.f + dist(node, child) + h(child, fin) and not [CLOSED].contains(child):
            if child.f = node.f + dist(node, child) + h(child, fin)
            child.parent = node
            [OPENED].push(child) or [OPENED].update(child)
5. if node == fin:
    while node != st:
        path.push_front(node)
        node = node.parent

$$f(node) = g(node) + h(node)$$

# Вариации алгоритмов

**«Волновой алгоритм»** – простейший вариант, основанный на использовании очереди. Подходит для случаев, когда расстояние между узлами одинаковое, или же принадлежит некоему множеству с относительно небольшим количеством элементов – например, $\{1, \sqrt{2}\}$. Оценочные функции не используются.

**Алгоритм Дейкстры** – учитывает только пройденный путь ($h(node) \equiv 0$), однако не требует введения эвристической функции. Плохая эффективность для полносвязных графов, однако может находить кратчайшие пути до нескольких целевых.

**Алгоритм A*** – требует описания эвристики с определёнными свойствами, при этом становится полным, оптимальным (допустимым), и оптимально эффективным. Учитывает как пройденный путь, так и оценку оставшегося расстояния (эвристическая функция), при этом затраты по памяти всё ещё могут расти экспоненциально. Теоретически считается «идеальным» в качестве алгоритма информированного поиска, на практике всё очень сильно зависит от качества эвристик.

**Алгоритм Iterative-A*** – итеративная версия A*, в которой рассматриваются только вершины, чья оценка не превосходит определённый порог. Если решение не найдено, то принимается следующее значение порога, и поиск начинается заново. При этом повторно рассматриваются уже рассмотренные вершины, однако это, как правило, даёт небольшое ухудшение. Отличается невысокими требованиями к памяти, можно свести к минимальным затратам по памяти (ценой исключения множества [CLOSED] – это может окупаться по времени.

# Допустимость и оптимальность A*

Эвристическая функция *допустима*, если она не «*переоценивает*» минимальную стоимость решения. Пусть C* – стоимость оптимального решения, n – некоторая вершина на оптимальном пути, на котором достигается оптимальное решение, m – неоптимальная целевая.

$$f(m) = g(m) + h(m) = g(m) > C^*$$

$$f(n) = g(n) + h(n) \leq g(n) + H(n) = C^*$$

$$f(n) < f(m),$$

но тогда мы не должны были рассматривать m раньше n.

Доказательство оптимальной эффективности проистекает из разбиения множества вершин на подмножества с одинаковой оценкой. Алгоритм рассматривает эти подмножества последовательно, что гарантирует минимум раскрытых вершин.

При использовании преемственной (монотонной) эвристической функции нет необходимости в пересчете оценок для уже открытых вершин, и оптимальность сохраняется для графов:

$h(n_1) \leq c(n_1,n_2) + h(n_2)$ – требование преемственности

Смысл: реальное положение дел хуже, чем казалось.

Формальное доказательство приведено в [1]

# Iterative deepening A*

Перед каждой итерацией поиска устанавливается bound – ограничение сверху, которое нельзя превышать в процессе поиска.

В идеальных условиях это приводит к тому, что множество дочерних вершин не создаётся на данной итерации поиска.

Если с текущим ограничением цель не найдена, то поиск будет выполнен со следующим возможным ограничением.

Существенное снижение «пустых», не просматриваемых вершин за счёт «нижнего» слоя, который, как правило, содержит максимальное число вершин (существенно превосходящее число вершин на других уровнях).

```
procedure ida_star(root)
  bound := h(root)
  path := [root]
  loop
    t := search(path, 0, bound)
    if t = FOUND then
      return (path, bound)
    if t = ∞ then
      return NOT_FOUND
    bound := t
  end loop
end procedure

function search(path, g, bound)
  node := path.last
  f := g + h(node)
  if f > bound then
    return f
  if is_goal(node) then
    return FOUND
  min := ∞
  for succ in successors(node) do
    if succ not in path then
      path.push(succ)
      t := search(path, g + cost(node, succ), bound)
      if t = FOUND then return FOUND
      if t < min then min := t path.pop()
    end if
  end for
  return min
end function
```

# Сравнение стратегий поиска

| | Стоимость поиска | | | Эффективный коэффициент ветвления | | |
|---|---|---|---|---|---|---|
| $d$ | IDS | A*($h_1$) | A*($h_2$) | IDS | A*($h_1$) | A*($h_2$) |
| 2 | 10 | 6 | 6 | 2,45 | 1,79 | 1,79 |
| 4 | 112 | 13 | 12 | 2,87 | 1,48 | 1,45 |
| 6 | 680 | 20 | 18 | 2,73 | 1,34 | 1,30 |
| 8 | 6384 | 39 | 25 | 2,80 | 1,33 | 1,24 |
| 10 | 47127 | 93 | 39 | 2,79 | 1,38 | 1,22 |
| 12 | 3644035 | 227 | 73 | 2,78 | 1,42 | 1,24 |
| 14 | – | 539 | 113 | – | 1,44 | 1,23 |
| 16 | – | 1301 | 211 | – | 1,45 | 1,25 |
| 18 | – | 3056 | 363 | – | 1,46 | 1,26 |
| 20 | – | 7276 | 676 | – | 1,47 | 1,27 |
| 22 | – | 18094 | 1219 | – | 1,48 | 1,28 |
| 24 | – | 39135 | 1641 | – | 1,48 | 1,26 |

# Требования к игровым алгоритмам

Во многих случаях (за исключением логических игр), можно ослабить требования к алгоритмам поиска пути:

1. Требования оптимальности пути можно заменить на требование нахождения «почти оптимального» пути;
2. Рандомизация – строить всегда один и тот же маршрут не всегда является хорошей идеей, имитация человеческого поведения;
3. Пересчёт пути в динамическом окружении можно производить не на каждом шаге – ухудшение качества, однако некоторая «человечность»;
4. Люди (игроки) не перемещаются по оптимальным маршрутам – например, не движутся вдоль стен. Это можно моделировать разными ограничениями на стоимость перемещения по определённым узлам, варьируя эвристику (что, очевидно, изменит свойства алгоритма);
5. Что можно учитывать пре перемещении, например, в условном «подземелье»?

# Базовые алгоритмы (обзор)

| Algorithm | Properties | Restrictions |
|---|---|---|
| Lee algorithm (wave algorithm, волновой трассировки) | Optimal path, uninformed, static | All edges has same weight |
| Dijkstra algorithm (алгоритм Дейкстры в различных вариациях) | Optimal path, uninformed, static | Memory consumption, complexity |
| A* algorithm | Optimal path, informed, static | Still requires exponential memory, optimal |

# Многоуровневая маршрутизация

- Определяем некоторую иерархию – в идеале необходимо размещение маршрутных точек (waypoints).
- В идеальном случае необходима кластеризация пространства поиска, с точками перехода (соединениями).
- Существует статичная таблица маршрутов между кластерами (доменами - Dirichlet domains), предположительно они не изменяются с течением времени.
- Маршрутизация только до выходных точек, и до целевого кластера.
- Многократное ускорение вычислений, но с некоторой потерей точности. Удачный вариант для переходов/дверей/арок.

# Сглаживание пути

- Позволяет избавиться от проблем дискретизации путей в tile-based map representation.
- Предполагаем, что существует путь с нормальной достижимостью по узлам – каждая пара соседних точек достижима, проверяется аналогом raycasting.
- Алгоритм:

1. Взять начальную точку пути в качестве текущей.
2. Пока текущая не целевая:
    1. Искать точку, которая не является достижимой;
    2. Если дошли до конечной точки – выход;
    3. Взять в качестве текущей предыдущую точку, поместить её в путь.

# Избегание препятствий

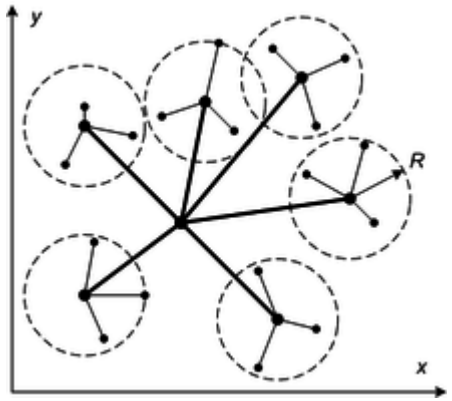
- Необходимо строить маршруты таким образом, чтобы не проходить вплотную к углам – это неестественно.
- «Отражение» от препятствий – некий аналог сглаживания пути, однако требует использования raycasting, или похожих методов.
- Можно добиться увеличением стоимости узлов около препятствий – будет некий аналог разумной навигации, однако возможны и артефакты в маршрутах.
- Можно использовать различные веса для реализации различного поведения.

# Если мы – банда?

- Какие проблемы решаем?
- Оптимизация перемещения множества агентов – алгоритмы, соответствующие максимальному потоку на графах (Форд-Фолкерсон и другие).
- Кооперированное перемещение – поддержание формации среди юнитов.
- Патрулирование (roaming).
- Сбор ресурсов.

# Алгоритм муравьиной колонии

- Ant colony optimization – поиск оптимальных путей на графах, используется для различных целей, однако изначально – сбор ресурсов.
- $p_{i,j} = \frac{\tau_{i,j}^{\alpha}\eta_{i,j}^{\beta}}{\sum_{j \in Adj(i)} \tau_{i,j}^{\alpha}\eta_{i,j}^{\beta}}$ – вероятность перехода по ребру (i,j)
- $\tau_{i,j} = (1-\rho)\tau_{i,j} + \Delta\tau_{i,j}$ – обновление феромона на ребре. Изменение обычно обратно пропорционально стоимости пути для прошедшего по нему муравья.
- Аналогично – алгоритм пчелиной колонии



# Требования к pathfinder

Если реализуется объектом, то:

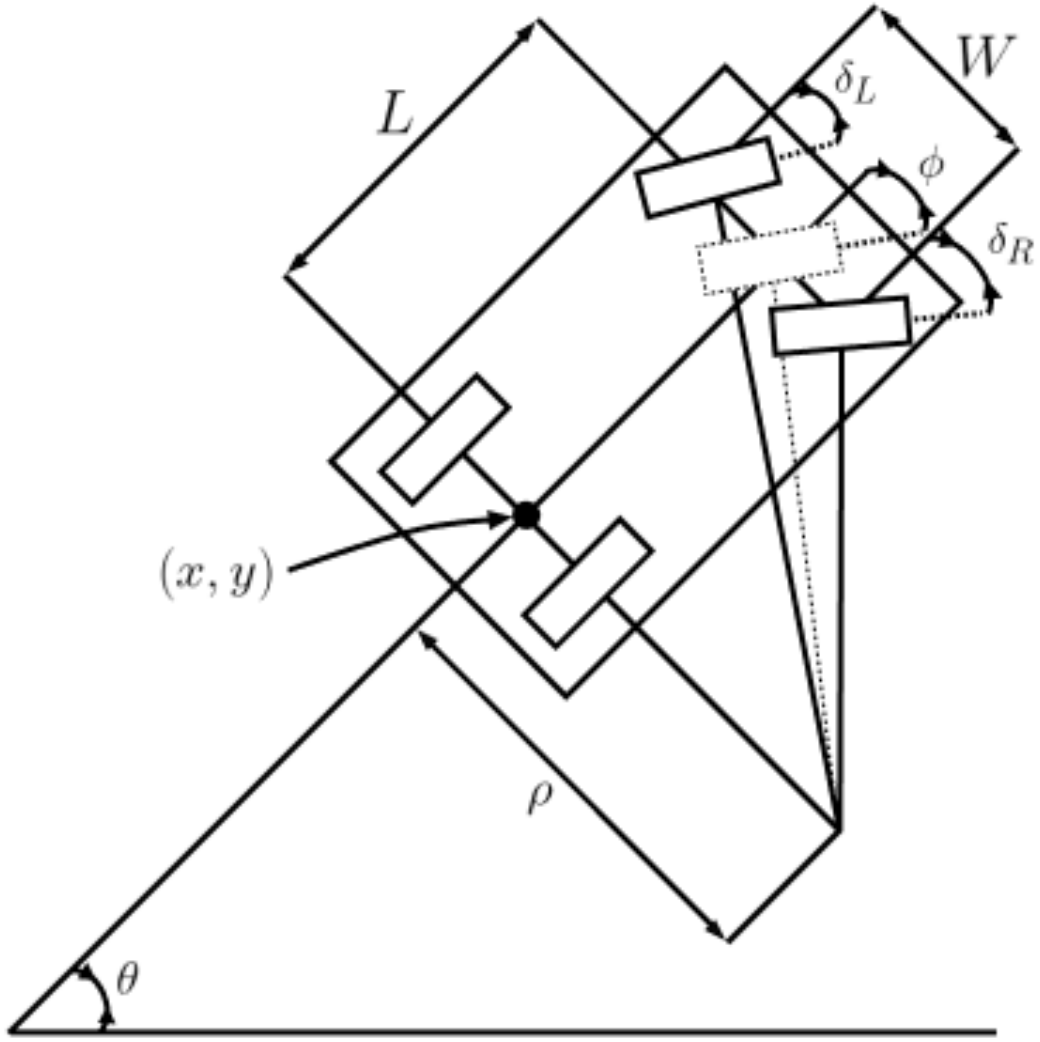
1. Трансляция точек пространства в узлы графа;
2. Работа в отдельном потоке;
3. Контроль времени работы и возможность прерывания алгоритма в заданный момент времени;
4. Генерация путей (или их части), либо точек для локальной маршрутизации;
5. Построение локальных маршрутов в случае прерывания;
6. Обслуживание множественных запросов на маршрутизацию;
7. Сохранение построенных путей для учета множества агентов;
8. Возможность изучения области маршрутизации.

# Сложные объекты

- Могут обладать объёмом, направлением, инерцией, другими параметрами.
- Не транслируются напрямую в узлы сетки/графа.
- Имеют коллайдеры – механизмы определения допустимых перемещений.
- Смещение по времени, коррекция геометрии.
- Имеет значение время.

# Автомобиль

Геометрия движения Аккермана. Согласно этой модели, углы $\delta L$ и $\delta R$ рассчитываются таким образом, чтобы центры вращения каждого из колес совпали с центром вращения мнимого колеса расположенного в центре передней колесной базы и имеющего угол поворота $\varphi$



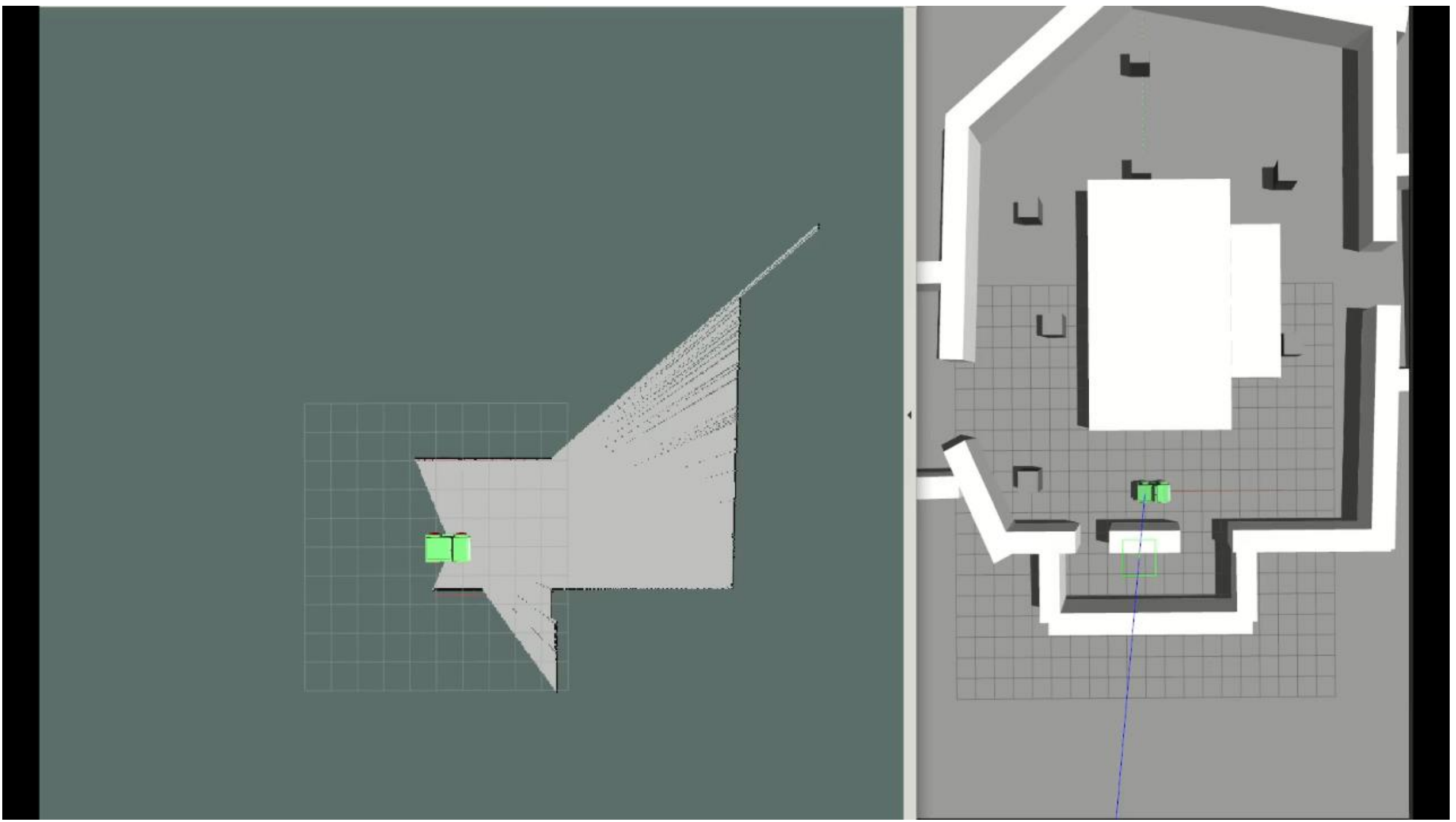
# Общие идеи

Большое количество алгоритмов, использующих методы частиц – по сути, стохастическая оптимизация.

Обычно используют дискретизацию пространства управляющих воздействий (поворот колёс, ускорение/торможение), при этом область можно рассматривать как непрерывное пространство.

Зачастую используется многоуровневая маршрутизация – строится маршрут, после чего по нему работает локальный планировщик, используя точки маршрута в качестве опорных.

Методы обладают высокой трудоёмкостью, однако при этом в реальных задачах требуется нахождение только локальных точек маршрута.



# Прыжки и перемещающиеся препятствия

- При tile-based возможно использование дискретизации по времени.
- Глобальный планировщик – догнать и прыгнуть, остальное на усмотрение локального.
- Предсказание положений препятствий – опрос препятствий и откат назад.
- Триггеры на основе точек прыжков.