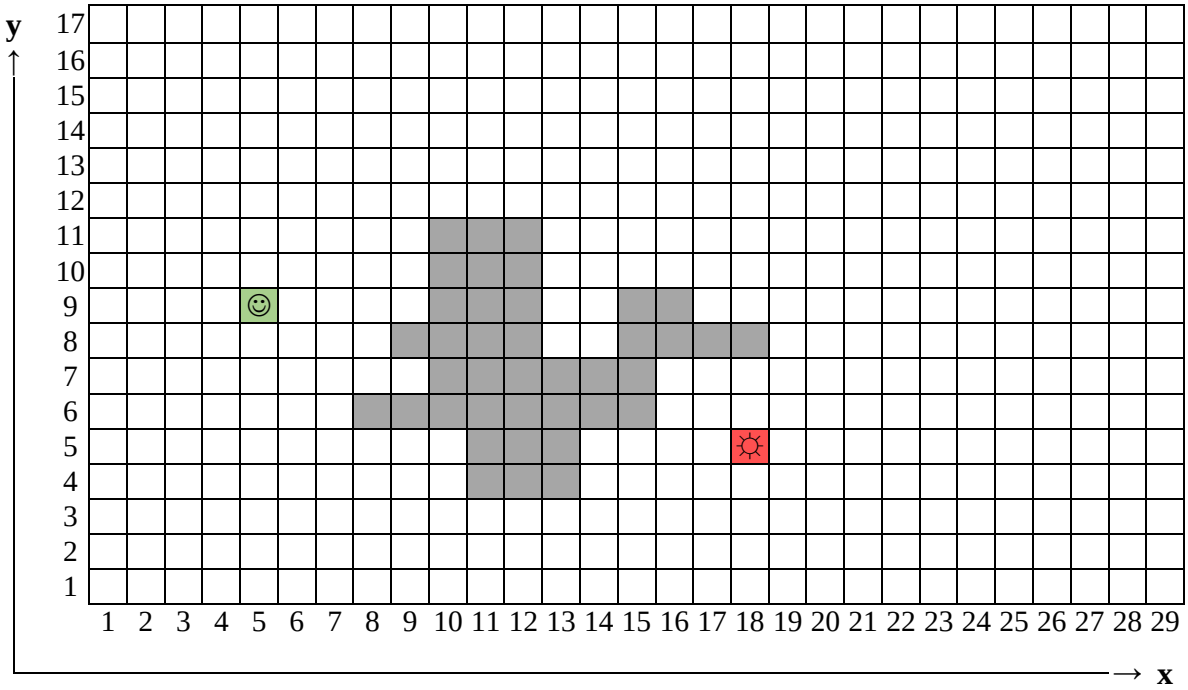
Алгоритми задач пересування на площині. Вступ до роботехніки. Пересування в умовах невизначеності

Будова площини для задач пересування

Для складання алгоритмів пересування площиною треба мати формальну модель площини. Однією з найпростіших моделей є зображення площини прямокутною фігурою, поділеною на квадрати. Наприклад:



Будемо вважати, що площина належить першій чверті декартової площини координат. Тоді координати кожного квадрата є звичними: [1,1], [1,2], [1,3], [2,1], [3,1], [2,2], і так далі. Зауважимо, що нумерація починається від 1, а не від 0, бо мова йде не про відстань від початку координат, а про позицію квадратів.

У випадку моделювання площини програмними методами треба будувати матрицю відповідними операторами програми. Якщо уявляти програмовану матрицю так, як на малюнку, тоді, по-перше, нумерація рядків матриці буде зверху вниз. Нумерація стовпців залишиться зліва направо. По-друге, в програмуванні нумерація рядків і стовпців матриць починається від 0, а не від 1. Але такі зауваження не будуть впливати на загальну будову алгоритмів пересування площиною (матрицею), чи алгоритмів аналізу змісту площини (матриці).

Кожний квадрат площини може належати якомусь об'єкту або бути вільним. Якщо квадрат вільний – можна надати йому значення 0. Якщо квадрат належить якомусь об'єкту, як показано на рисунку, тоді надаємо йому значення відповідно до типу об'єкта. Наприклад, сірі квадрати можуть мати значення 1, зелений – значення 2, а червоний – 3.

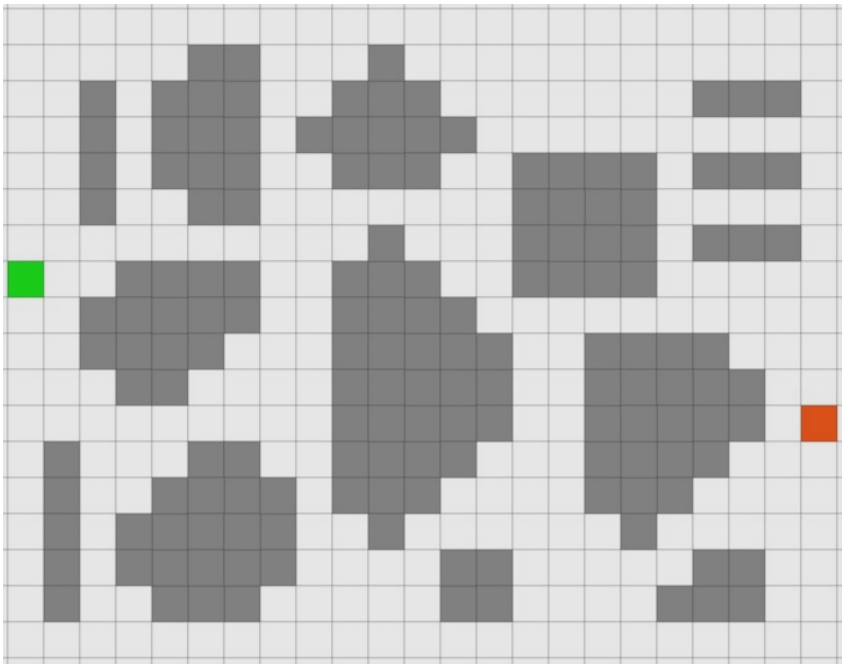
Окремо треба фіксувати позиції квадратів, які є на границі площини – перший чи другий індекс квадрата дорівнює 0, чи sizex-1, чи sizey-1 (в програмуванні). Це потрібно для визначення стану пересування, якщо підійшли до межі площини. Для моделювання алгоритмів пересування вважаємо, що вихід за межі заданої площини не допускається.

Площина з ізольованими опуклими перешкодами. Правосторонній обхід

Задача 1. Деяка ділянка площини прямокутної форми має рівну поверхню, на якій в різних позиціях є так звані перешкоди – ділянки, неможливі для пересування. Розташування перешкод має довільні позиції. Будова кожної перешкоди задовільняє таким вимогам: 1) не сполучена з будь-якою іншою; 2) має замкнений власний периметр; 3) є опуклою за відомим математичним визначенням; якщо сполучити прямою лінією дві сусідні клітинки оболонки, тоді ціла перешкода має бути по одну сторону від лінії; 4) розміри і конфігурація перешкод довільні, але границі перешкод не дотикаються до границь ділянки, завжди існує прохід між межею ділянки і перешкодою.

Потрібно скласти алгоритм переходу площиною від заданої позиції одного краю на задану позицію протилежного краю, оминаючи перешкоди. Алгоритм повинний мати визначену систему команд виконавця (робота), про яку буде далі. Розташування перешкод роботу наперед не відоме, ідентифікувати перешкоду робот може, лише наблизившись до сусідньої клітки від перешкоди, тобто, підійшовши впритул до неї. Проте робот завжди знає координати свого поточного розташування і координати мети.

Приклад площини з перешкодами:



Перешкоди зафарбовані темним кольором. Зелений квадрат ліворуч біля лівого краю позначає стартову позицію переходу. Червоний квадрат біля правого краю позначає фінішну позицію переходу. Перехід площиною треба виконати від зеленого квадрата до червоного зліва направо по осі *x* декартової площини, оминаючи перешкоди і мінімізуючи довжину шляху переміщення. Можна попередньо вважати, що перехід треба виконати від лівого краю до правого, бо кольорові квадрати розташовані на межі.

Зауважимо, що при виконанні перерахованих вимог до будови перешкод завжди існує шлях від зеленого квадрата до червоного.

Кожний квадрат зображеної площини можна позначити числами, наприклад так, як було сказано вище:

0 – квадрат вільний;

1 – квадрат належить перешкоді;

2 – стартова позиція для переходу (зелений квадрат) – на лівому краю;

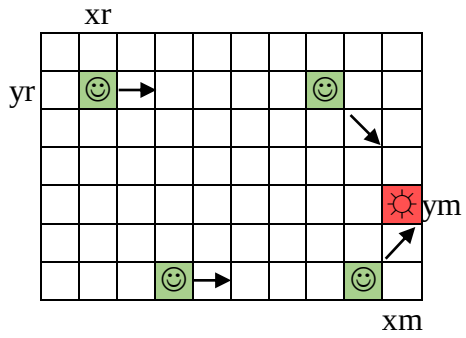
3 – фінішна позиція переходу (червоний квадрат) – на правому краю.

Алгоритм. Цей алгоритм є евристичним, побудований на апріорно визначених правилах руху. Як кожний евристичний алгоритм, він не гарантує оптимального шляху.

Визначимо допустимі напрямки руху горизонтально, вертикально і діагонально:

- xplus горизонтально направо;
- xplusyminus вниз направо діагонально;
- yminus вертикально вниз;
- xminusyminus вниз наліво діагонально
- xminus горизонтально наліво
- xminusyplus вверх наліво діагонально
- yplus вертикально вверх
- xplusyplus вверх направо діагонально

Приймемо за основу правило пересування, за яким треба з останньої поточної клітки завжди переходити до клітки в напрямку вектора позиції клітки мети, якщо така клітка вільна і робот не є біля правого краю площини $xr \neq xm$. На рисунку нижче (xr,yr) – поточні координати робота, (xm,ym) – координати клітки мети, $xr<xm$. Вектор напрямку *direction* треба привести до ближчого кута, кратного 45 градусів:




```
# для випадків, коли є вільна клітка
# xr != xm, інакше вже на краю
if (xm-xr) >= abs(ym-yr):
  direction = xplus
elif ym < yr :
  direction = xplusyminus
else:  # ym > yr
  direction = xplusyplus
```

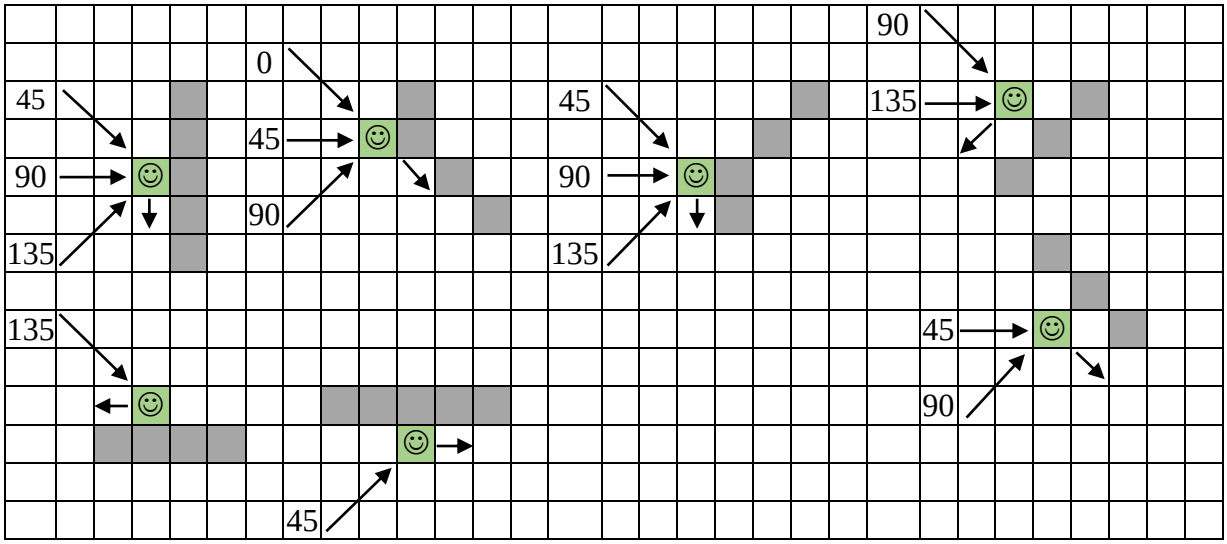
На кожному кроці потрібно пам'ятати напрямок руху.

Будова алгоритму є така:

1) якщо дійшли до правого краю площини – перейти до кроку 5;

2) якщо є вільна клітка в напрямку вектора позиції клітки мети (*xm*,*ym*), то перейти до неї; напрямок щоразу обчислювати за схемою, показаною вище;

3) якщо клітка попереду напряму руху належить перешкоді, тоді перейти до процедури *правостороннього* обходу перешкоди. Використаємо для цього відоме "правило лівої руки", яке застосовують в задачах пересування лабіринтом. Для цього, підійшовши до перешкоди, повертаємо направо за годинниковою стрілкою на 45, 90 або 135 градусів:

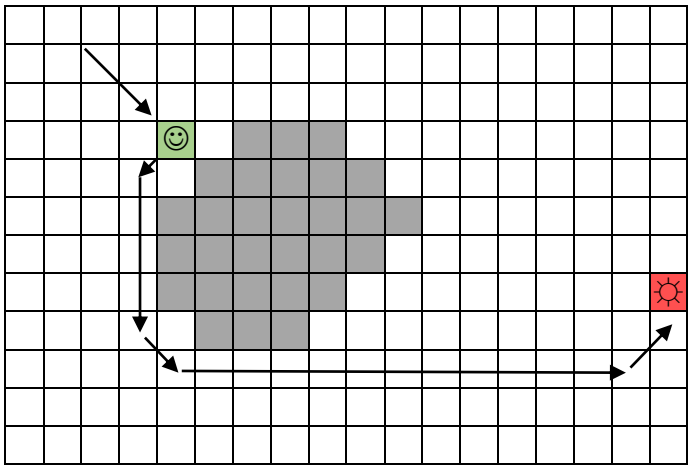


В таблиці показані основні конфігурації для випадків, коли робот підходить до краю перешкоди. Стрілками показані напрями, якими потрапили до краю перешкоди і яким напрямком треба продовжити рух. [Напрями продовження можуть мати варіанти, залежно від точної конфігурації границі перешкоди і від наших правил руху.] Числами позначені необхідні повороти в градусах за годинниковою стрілкою. Нуль означає, що можна продовжити рух в тому самому напрямку. Можна виконати додатковий аналіз можливих випадків наближення робота до стіни.

Щоб встановити варіант наближення робота до стіни, треба виконати перевірку конфігурації сусідніх граничних кліток перешкоди.

Тримаючись умовно лівою рукою стіни перешкоди (після повороту направо перешкода буде ліворуч), продовжуємо рух кроками по одній клітці вздовж границі перешкоди, повторюючи її контур і виконуючи відповідні повороти. Цю частину руху моделювати і визначити окремо.

Приклад правостороннього обходу перешкоди:



4) рух вздовж стіни продовжуємо доти, поки рух не відновиться в напрямку вектора позиції клітки мети (xm,ym); напрям треба обчислювати заново після кожного кроку; тоді повернутись до кроку 1.

5) підійшовши до правого краю площини, рухатись вздовж неї зверху вниз чи знизу вверх, залежно від позиції робота, до позиції червоного квадрата.

Система команд виконавця

Прийнявши до уваги викладені вище правила руху на площині з перешкодами, треба формалізувати систему команд виконавця, за якою можна реалізувати правила руху. Система команд – це окремі елементи алгоритму виконавця (робота). Систему команд можна будувати різними способами. Далі наведено один з можливих варіантів будови системи команд. Важливим є лиш те, щоб система команд була функціонально повною, тобто, надавала можливість будови шляху будь-якої конфігурації.

Для випадку виконавця-робота систему команд можна розділити на три частини: 1) команди руху; 2) команди аналізу стану виконавця і руху; 3) аналітичні команди.

Команди руху:

`setpos(x,y)` перейти до клітки з координатами `x,y`; робот стає в задану позицію, напрям руху не визначений;

`setaim(x,y)` встановити позицію `x,y` для клітки мети;

`setdir(direction)` встановити напрям руху (вектор) `direction`; допустимі напрямки руху визначені вище;

`step1()` перейти на сусідню клітку в напрямку руху, напрям зберігається;

`turnright(angle)` змінити напрям руху поворотом направо від поточного напрямку за годинниковою стрілкою на кут `angle`; кут повороту має бути кратний до 45 градусів;

`turnleft(angle)` змінити напрям руху поворотом наліво від поточного напрямку проти годинникової стрілки на кут `angle`; кут повороту має бути кратний до 45 градусів;

Команди перевірки стану виконавця:

`IstheCellinFrontFree()` чи вільна клітка в напрямку руху – true/false;

`IsCellxyFree(x,y)` чи вільна клітка з координатами `x,y` – true/false;

`getposxy()` отримати поточну позицію розташування – `x,y`;

`isborderleft(), isborderright(), isborderbelow(), isbordertop()` чи поточна позиція є біля лівого боку площини, правого, нижнього, верхнього відповідно – true/false;

`gettargetposition()` отримати позицію `x,y` клітки мети – кінцевого пункту руху;

Аналітичні команди: (комплексний аналіз)

`CalcNextDirRight()` обчислити напрям руху наступного кроку, застосувавши метод *правостороннього* обходу; це є обчислювальна процедура, яку виконують тоді, коли робот підійшов до краю перешкоди; процедура обчислює конфігурацію граничної ділянки перешкоди в околі позиції робота і приймає рішення про напрям наступного кроку; по суті – це основний алгоритм правостороннього обходу;

`CalcNextDirLeft()` те саме, що й `CalcNextDirRight()`, лише для методу *лівостороннього* обходу.

Висновки

Зазначені команди можна програмно реалізувати як бібліотеку функцій для задач будови шляхів пересування виконавця-робота. Список команд можна доповнити за потреби іншими командами. Програмування задач пересування площиною з перешкодами має спиратись на таку бібліотеку функцій. Самі функції можна в майбутньому вдосконалити, наприклад, придумати кращий алгоритм для аналітичних команд правостороннього чи лівостороннього обходу. При цьому буде збережений основний алгоритм загального обчислення шляху.

Насамкінець нагадаємо, що описані правила руху і сам алгоритм пересування є евристичними. Можливість експериментів з правилами руху і алгоритмом пересування залишаєть відкритою. Так само відкритою залишається оцінка ефективності подібних алгоритмів.

Лівосторонній обхід

Задача 2. Зберігаємо повністю задачу 1 і всі наведені вище виклади. Міняємо лише спосіб обходу перешкоди – замість правостороннього використаємо *лівосторонній*.

Покажемо зміни, які відбуваються в цьому випадку.

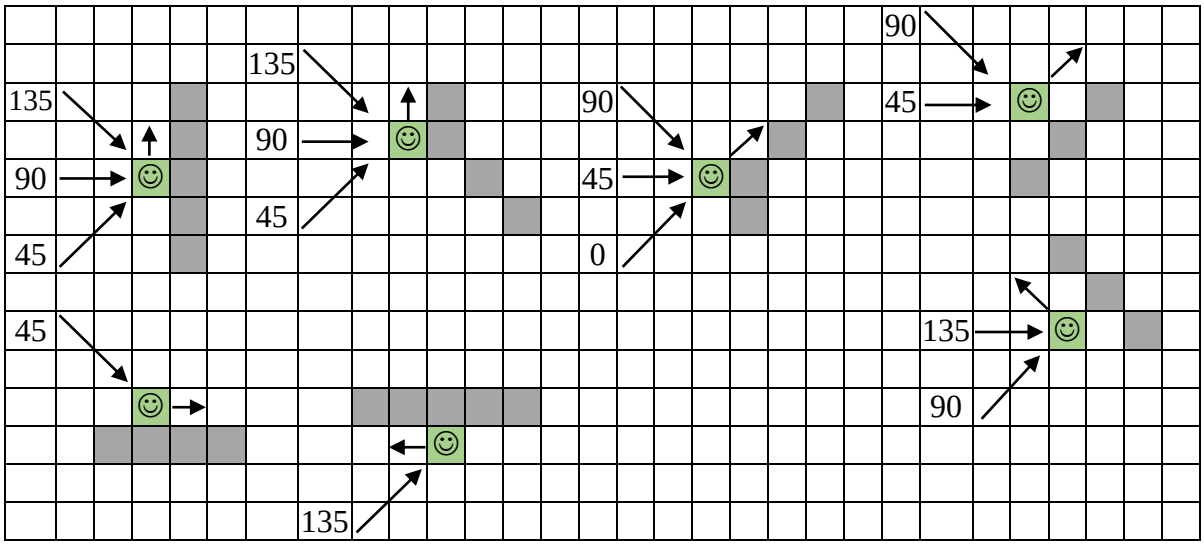
Позначення квадратів площини – без змін.

Допустимі напрямки руху – без змін.

Основне правило пересування – без змін.

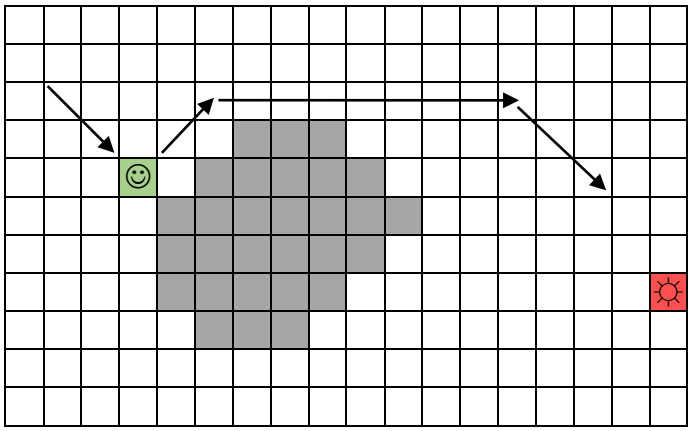
Система команд виконавця – без змін.

В будові алгоритму пересування міняємо пункт 3 – замість правостороннього обходу визначаємо лівосторонній. Використаємо для цього "правило правої руки", яке застосовують в задачах пересування лабіринтом. Для цього, підійшовши до перешкоди, повертаємо наліво за годинниковою стрілкою на 45, 90 або 135 градусів:



В таблиці показані основні конфігурації для випадків, коли робот підходить до краю перешкоди. Стрілками показані напрями, якими потрапили до краю перешкоди і яким напрямком треба продовжити рух. Числами позначені необхідні повороти в градусах проти годинникової стрілки. Тримаючись умовно правою рукою стіни перешкоди (після повороту наліво перешкода буде праворуч), продовжуємо рух кроками по одній клітці вздовж границі перешкоди, повторюючи її контур і виконуючи відповідні повороти.

Приклад лівостороннього обходу перешкоди:

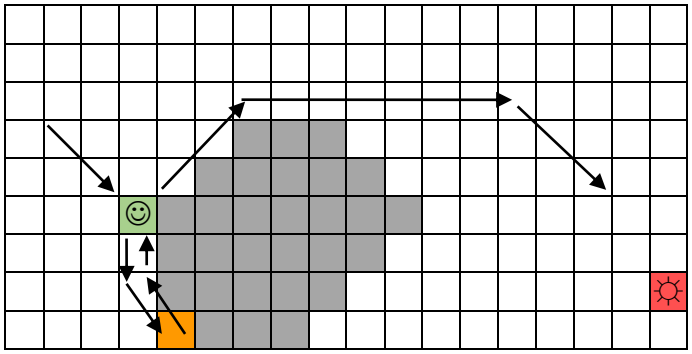


Опуклі перешкоди дотичні до границь

Задача 3. Залишаємо в силі умови задач 1 або 2. Додаємо нову умову: перешкода може дотикатись до одного краю (лише одного) ділянки пересування. В цьому разі неможливий перехід через перешкоду з певного боку.

Як будувати шлях пересування?

Приклад дотичної перешкоди до нижнього краю ділянки і спосіб вирішення проблеми для випадку *правостороннього* обходу:



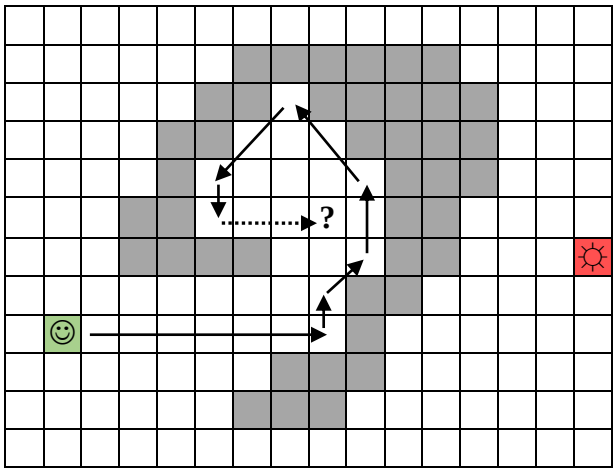
Розпочавши правосторонній обхід від зеленої клітки, потрапляємо до клітки нижнього боку ділянки (зафарбована жовтим кольором), яка є на границі. Дальший рух за правилом правостороннього обходу неможливий. Міняємо правило руху на протилежне – переходимо до *лівостороннього* обходу. Оскільки перешкода за умовою немає точок дотику до інших країв ділянки пересування, то обійти перешкоду можна за протилежним правилом.

Аналогічно можна вирішити проблему у випадку, коли за лівостороннім обходом потрапляємо до граничної клітки верхнього боку ділянки, - треба перейти до правостороннього обходу.

Такі самі міркування є справедливі у випадках дотичності перешкоди до лівого чи правого боку ділянки.

Неопуклі перешкоди

Якщо конфігурація перешкоди не задовільняє умові опуклості, тоді викладені вище правила пересування можуть приводити до ситуацій неможливості руху. Може бути багато різновидів таких ситуацій. Для прикладу наведемо лише одну з них, коли основне правило переходу з останньої поточної клітки в напрямку вектора позиції клітки мети може будувати безконечний цикл, якщо виконувати лівосторонній обхід:



Пунктирною лінією показаний розрахунковий наступний напрям руху відповідно до раніше визначених правил обчислення вектора напрямку direction. Робот потрапляє в "пастку", з якої неможливо вийти, користуючись попередніми правилами.

Задача 4. Обгрунтувати і запропонувати рішення конкретної проблеми руху, показаної на малюнку.

Задача 5. Як треба змінити правила руху робота площиною, щоб долати неопуклі перешкоди? Чи треба змінити основне правило пересування?

Покликання на матеріали, дотичні за темою лекції

Алгоритм розгадування лабіринту:
https://en.wikipedia.org/wiki/Maze-solving_algorithm

Solving a Reconfigurable Maze using Hybrid Wall Follower Algorithm:
https://research.ijcaonline.org/volume82/number3/pxc3892114.pdf

[додаток Quora] Яка логіка програмування для навігації лабіринтом за правилом правої руки?
https://www.quora.com/What-is-the-programming-logic-for-navigating-a-maze-with-the-right-hand-rule

[ResearchGate] Алгоритми відображення лабіринту (наукові публікації):
https://www.researchgate.net/figure/Maze-mapping-algorithm-a-Right-hand-rule-b-Left-hand-rule_fig2_332434440

[Наукова спільнота. Енциклопедія] Алгоритм вирішення лабіринту:
https://encyclopedia.pub/entry/33079