شرح تفصيلي لكود خوارزمية A*

الكود الكامل لخوارزمية A*

1. فئة Node

```
class Node {
  x: number;           // إحداثي X للعقدة
  y: number;           // إحداثي Y للعقدة
  g: number;           // التكلفة من البداية إلى هذه العقدة
  h: number;           // التكلفة المتوقعة من هذه العقدة إلى الهدف
  f: number;           // f = g + h (التكلفة الإجمالية)
  parent: Node | null; // العقدة الأب (لإعادة بناء المسار)

  constructor(x: number, y: number, g: number, h: number) {
    this.x = x;        // تخزين إحداثي X
    this.y = y;        // تخزين إحداثي Y
    this.g = g;        // تخزين التكلفة من البداية
    this.h = h;        // تخزين التكلفة المتوقعة
    this.f = g + h;    // حساب التكلفة الإجمالية
    this.parent = null; // في البداية لا توجد عقدة أب
  }
}
```

شرح كل سطر:

السطر 1-6: الخصائص

- `x` و `y` : موضع العقدة في الشبكة (مثل: (3 ,5))
- `g` : المسافة الفعلية من نقطة البداية إلى هذه العقدة
  - مثال: إذا بدأنا من (0,0) ووصلنا إلى (3,4)، فإن g = 5
- `h` : تخمين المسافة من هذه العقدة إلى الهدف
  - مثال: إذا الهدف في (10,10) وهذه العقدة في (3,4)، فإن h = 13 (Manhattan)

- `f` : مجموع g و h (هذا ما نستخدمه لاختيار أفضل عقدة)
- `parent` : تخزين العقدة السابقة لإعادة بناء المسار عند الوصول للهدف

السطر 8-18: Constructor

- `constructor(x, y, g, h)` : دالة البناء تأخذ 4 معاملات
- `this.x = x` : تخزين قيمة x
- `this.y = y` : تخزين قيمة y
- `this.g = g` : تخزين قيمة g
- `this.h = h` : تخزين قيمة h
- `this.f = g + h` : حساب f مباشرة عند الإنشاء
- `this.parent = null` : في البداية، لا توجد عقدة أب

مثال على الاستخدام:

```
// إنشاء عقدة جديدة
const node = new Node(5, 3, 4.242, 14);

// الآن:
// node.x = 5
// node.y = 3
// node.g = 4.242 (المسافة الفعلية من البداية)
// node.h = 14 (المسافة المتوقعة للهدف)
// node.f = 18.242 (المجموع)
// node.parent = null
```

---

2. دالة Heuristic (الدالة الإرشادية)

```
function heuristic(
  from: Cell,           // النقطة الحالية
  to: Cell,             // النقطة المقصودة (عادة الهدف)
  type: "manhattan" | "euclidean" = "manhattan"  // نوع الحساب
): number {
  const dx = Math.abs(from.x - to.x);  // الفرق المطلق في X
  const dy = Math.abs(from.y - to.y);  // الفرق المطلق في Y

  if (type === "manhattan") {
    return dx + dy;  // مسافة مانهاتن: مجموع الفروقات
  } else {
    return Math.sqrt(dx * dx + dy * dy);  // المسافة الإقليدية: الجذر التربيعي
  }
}
```

شرح كل سطر:

السطر 1-5: توقيع الدالة

- `function heuristic(...)` : اسم الدالة
- `from: Cell` : النقطة الحالية (مثال: {x: 2, y: 2})
- `to: Cell` : النقطة المقصودة (مثال: {x: 12, y: 12})
- `type: "manhattan" | "euclidean" = "manhattan"` : نوع الحساب (افتراضي: manhattan)

السطر 6-7: حساب الفروقات

- `const dx = Math.abs(from.x - to.x)` : الفرق المطلق بين X في النقطتين
  - مثال: |2 - 12| = 10
- `const dy = Math.abs(from.y - to.y)` : الفرق المطلق بين Y في النقطتين
  - مثال: |2 - 12| = 10

السطر 9-14: حساب المسافة

- `if (type === "manhattan")` : إذا كان النوع manhattan
  - `return dx + dy` : أرجع مجموع الفروقات (10 + 10 = 20)

- `else` : وإلا (نوع euclidean)
  - `return Math.sqrt(dx * dx + dy * dy)` : أرجع الجذر التربيعي
  - مثال: $\sqrt{(10^2 + 10^2)} = \sqrt{200} = 14.14$

الفرق بين النوعين:

Manhattan Distance (مسافة مانهاتن):

- الصيغة: $h = |x\_goal - x| + |y\_goal - y|$
- الفكرة: تخيل أنك تتحرك في شوارع مدينة (أفقي أو عمودي فقط)
- متى تستخدم؟ عندما تكون الحركة محدودة بـ 4 اتجاهات فقط

Euclidean Distance (المسافة الإقليدية):

- الصيغة: $h = \sqrt{[(x\_goal - x)^2 + (y\_goal - y)^2]}$
- الفكرة: المسافة المستقيمة بين نقطتين (كما تقيسها بالمسطرة)
- متى تستخدم؟ عندما تكون الحركة حرة في 8 اتجاهات (بما فيها الأقطار)

مثال على الاستخدام:

```
const from = { x: 2, y: 2 };
const to = { x: 12, y: 12 };

// استخدام Manhattan
const h_manhattan = heuristic(from, to, "manhattan");
// النتيجة: 20 = 10 + 10

// استخدام Euclidean
const h_euclidean = heuristic(from, to, "euclidean");
// النتيجة: √(10² + 10²) = 14.14
```

---

3. خوارزمية A* الكاملة

```
function aStar(
  start: Cell,            // نقطة البداية
  goal: Cell,             // نقطة الهدف
  walls: Set<string>,     // مجموعة الجدران
  gridSize: number        // حجم الشبكة
): Cell[] {               // إرجاع مصفوفة المسار
```

شرح معاملات الدالة:

- `start` : نقطة البداية (مثال: {x: 2, y: 2})
- `goal` : نقطة الهدف (مثال: {x: 12, y: 12})
- `walls` : مجموعة تحتوي على مواضع الجدران (مثال: {"3,3", "4,4"})
- `gridSize` : حجم الشبكة (مثال: 15 تعني شبكة 15×15)

الخطوة 1: تهيئة المتغيرات

```
const openSet: Node[] = [];             // العقد المرشحة للفحص
const closedSet = new Set<string>(); // العقد المفحوصة بالفعل
const nodeMap = new Map<string, Node>(); // خريطة سريعة للعقد
```

شرح كل متغير:

- `openSet` : مصفوفة تحتوي على العقد التي لم نفحصها بعد
  - نختار منها العقدة بأقل $f(n)$ في كل تكرار
- `closedSet` : مجموعة تحتوي على مواضع العقد التي فحصناها بالفعل
  - نستخدمها لتجنب إعادة فحص نفس العقدة مرتين
- `nodeMap` : خريطة (Map) تربط بين موضع العقدة والعقدة نفسها
  - الفائدة: البحث السريع $O(1)$ بدلاً من $O(n)$

الخطوة 2: إنشاء عقدة البداية

```
const startNode = new Node(
  start.x,
  start.y,
  0,                          // g = 0 (لم نقطع أي مسافة بعد)
  heuristic(start, goal)      // h = المسافة المتوقعة من البداية للهدف
);
openSet.push(startNode);      // إضافة عقدة البداية إلى openSet
nodeMap.set(`${start.x},${start.y}`, startNode); // تخزينها في الخريطة
```

شرح كل سطر:

- `new Node(...)` : إنشاء عقدة جديدة
  - `start.x, start.y` : موضع البداية
  - `0` : $g = 0$ لأننا في البداية ولم نقطع أي مسافة
  - `heuristic(start, goal)` : h = المسافة المتوقعة من البداية للهدف
- `openSet.push(startNode)` : إضافة عقدة البداية إلى قائمة الفحص
- `nodeMap.set(...)` : تخزين العقدة في الخريطة السريعة

الخطوة 3: حلقة البحث الرئيسية

```
while (openSet.length > 0) {  // استمر طالما هناك عقد لم نفحصها
```

الشرح:

- تستمر الحلقة طالما `openSet` يحتوي على عقد
- إذا انتهت `openSet` ولم نجد الهدف = لا يوجد مسار

الخطوة 3.أ: البحث عن أفضل عقدة

```
let current = openSet[0];       // ابدأ بأول عقدة
let currentIndex = 0;           // احفظ موضعها
for (let i = 1; i < openSet.length; i++) {  // ابحث عن الأفضل
  if (openSet[i].f < current.f) {  // أقل f إذا كانت
    current = openSet[i];            // اجعلها الحالية
    currentIndex = i;                // احفظ موضعها
  }
}
```

شرح كل سطر:

- `let current = openSet[0]` : ابدأ بأول عقدة في openSet
- `let currentIndex = 0` : احفظ موضعها (سنحتاجه لاحقاً)
- `for (let i = 1; i < openSet.length; i++)` : ابحث عن باقي العقد
- `if (openSet[i].f < current.f)` : إذا كانت f الحالية أقل من f السابقة
  - `current = openSet[i]` : اجعلها الحالية
  - `currentIndex = i` : احفظ موضعها

لماذا نختار أقل f؟

- $f = g + h$
- أقل f تعني أن العقدة الأكثر واعدة للوصول للهدف بسرعة
- g: المسافة المقطوعة (معروفة)
- h: المسافة المتبقية (تخمين)
- الخوارزمية توازن بين المسافة المقطوعة والمسافة المتبقية

الخطوة 3.ب: التحقق من الوصول للهدف

```
if (current.x === goal.x && current.y === goal.y) {
  // وصلنا للهدف! الآن نعيد بناء المسار
  const path: Cell[] = [];
  let node: Node | null = current;
  while (node) {
    path.unshift({ x: node.x, y: node.y });  // إضافة في البداية
    node = node.parent;                       // الانتقال للعقدة الأب
  }
  return path;  // إرجاع المسار
}
```

شرح كل سطر:

- `if (current.x === goal.x && current.y === goal.y)` : هل وصلنا للهدف؟
- `const path: Cell[] = []` : إنشاء مصفوفة فارغة للمسار
- `let node: Node | null = current` : ابدأ من الهدف
- `while (node)` : استمر طالما هناك عقدة
  - `path.unshift(...)` : إضافة العقدة في البداية (لأننا نعود للخلف)
  - `node = node.parent` : الانتقال للعقدة الأب
- `return path` : إرجاع المسار الكامل

مثال على إعادة البناء:

```
current = Node(12,12, parent=Node(11,11, parent=Node(10,10,
parent=Node(2,2, parent=null))))

المسار المبني:
[
  {x:2, y:2},
  {x:10, y:10},
  {x:11, y:11},
  {x:12, y:12}
]
```

الخطوة 3.ج: نقل العقدة للمفحوصة

```
openSet.splice(currentIndex, 1);       // إزالة من openSet
closedSet.add(`${current.x},${current.y}`);  // إضافة إلى closedSet
```

شرح كل سطر:

- `openSet.splice(currentIndex, 1)` : إزالة العقدة من openSet
  - `splice(موضع, عدد العناصر)` : إزالة عنصر واحد من الموضع
- `closedSet.add(...)` : إضافة موضع العقدة إلى closedSet
  - نستخدم "x,y string" لتسهيل المقارنة

لماذا نفعل هذا؟

- نريد تجنب فحص نفس العقدة مرتين
- بمجرد فحص عقدة، نضيفها إلى closedSet
- إذا رأينا نفس الموضع مرة أخرى، نتخطاه

الخطوة 3.د: فحص الجيران

```
const neighbors = [
  { x: 0, y: -1 },   // أعلى
  { x: 1, y: 0 },    // يمين
  { x: 0, y: 1 },    // أسفل
  { x: -1, y: 0 },   // يسار
  { x: 1, y: -1 },   // أعلى يمين
  { x: 1, y: 1 },    // أسفل يمين
  { x: -1, y: 1 },   // أسفل يسار
  { x: -1, y: -1 }   // أعلى يسار
];
```

الشرح:

- 8 اتجاهات حركة ممكنة
- كل جار هو إزاحة من الموضع الحالي
- مثال: إذا current = (5, 5)، الجيران:
  - (4 ,5) أعلى

- (5 ,6) يمين
- (6 ,5) أسفل
- (5 ,4) يسار
- (4 ,6) أعلى يمين
- (6 ,6) أسفل يمين
- (6 ,4) أسفل يسار
- (4 ,4) أعلى يسار

الخطوة 3.هـ: حلقة فحص الجيران

```
for (const neighbor of neighbors) {
  const newX = current.x + neighbor.x;  // حساب X الجديد
  const newY = current.y + neighbor.y;  // حساب Y الجديد

  // التحقق من الحدود
  if (newX < 0 || newX >= gridSize || newY < 0 || newY >= gridSize)
continue;

  // التحقق من الجدران
  if (walls.has(`${newX},${newY}`)) continue;

  // التحقق من closedSet
  if (closedSet.has(`${newX},${newY}`)) continue;
```

شرح كل سطر:

- `for (const neighbor of neighbors)` : لكل جار
- `const newX = current.x + neighbor.x` : حساب X الجديد
  - مثال: 5 + 1 = 6
- `const newY = current.y + neighbor.y` : حساب Y الجديد
  - مثال: 5 + 0 = 5
- `if (newX < 0 || newX >= gridSize || newY < 0 || newY >= gridSize) continue`:
  - هل الموضع الجديد خارج الشبكة؟
  - إذا نعم، تخطَّ هذا الجار

- `if (walls.has(newX,{newY})) continue`:
  - هل هناك جدار في هذا الموضع؟
  - إذا نعم، تخطَّ هذا الجار
- `if (closedSet.has(newX,{newY})) continue`:
  - هل فحصنا هذا الموضع بالفعل؟
  - إذا نعم، تخطَّ هذا الجار

الخطوة 3.و: حساب التكلفة الجديدة

```
const newG = current.g + (Math.abs(neighbor.x) + Math.abs(neighbor.y) === 2
? 1.414 : 1);
const newH = heuristic({ x: newX, y: newY }, goal);
const newF = newG + newH;
```

شرح كل سطر:

- `const newG = current.g + (...)`:
  - `current.g`: التكلفة من البداية إلى العقدة الحالية
  - `(Math.abs(neighbor.x) + Math.abs(neighbor.y) === 2 ? 1.414 : 1)`:
    - إذا كانت الحركة قطرية ($|x| + |y| = 2$) = 1.414
    - وإلا (حركة أفقية أو عمودية) = 1
  - مثال: إذا current.g = 5 وحركة قطرية، newG = 5 + 1.414 = 6.414
- `const newH = heuristic({ x: newX, y: newY }, goal)`:
  - حساب المسافة المتوقعة من الجار الجديد للهدف
- `const newF = newG + newH`:
  - مجموع التكلفة الإجمالية

الخطوة 3.ز: التحقق من وجود مسار أفضل

```
const existingNode = nodeMap.get(`${newX},${newY}`);
if (existingNode && existingNode.g <= newG) continue;
```

الشرح:

- `const existingNode = nodeMap.get(...)`:
  - هل فحصنا هذا الموضع من قبل؟
  - إذا نعم، احصل على العقدة القديمة

- `if (existingNode && existingNode.g <= newG) continue`:
  - إذا كانت العقدة موجودة و g القديم أقل أو يساوي g الجديد
  - تخطَّ هذا الجار (المسار السابق أفضل أو متساوي)

الخطوة 3.ح: إضافة الجار الجديد

```
const newNode = new Node(newX, newY, newG, newH);
newNode.parent = current;
openSet.push(newNode);
nodeMap.set(`${newX},${newY}`, newNode);
```

شرح كل سطر:

- `const newNode = new Node(newX, newY, newG, newH)`:
  - إنشاء عقدة جديدة للجار

- `newNode.parent = current`:
  - تعيين العقدة الحالية كأب (لإعادة بناء المسار لاحقاً)

- `openSet.push(newNode)`:
  - إضافة العقدة الجديدة إلى openSet (للفحص لاحقاً)

- `nodeMap.set(...)`:
  - تخزين العقدة في الخريطة السريعة

الخطوة 4: إذا لم يوجد مسار

```
return []; // لم يتم العثور على مسار
```

الشرح:

- إذا انتهت حلقة while ولم نجد الهدف
- نرجع مصفوفة فارغة []

- هذا يعني لا يوجد مسار من البداية للهدف

ملخص خطوات الخوارزمية

1. **التهيئة**: إنشاء openSet و closedSet و nodeMap
2. **إضافة البداية**: إضافة عقدة البداية إلى openSet
3. **حلقة البحث**:
   - اختيار العقدة بأقل f من openSet
   - التحقق من الوصول للهدف
   - نقل العقدة إلى closedSet
   - فحص الجيران
   - إضافة الجيران الجدد إلى openSet
4. **إرجاع المسار**: إعادة بناء المسار من الهدف للبداية

أمثلة عملية

مثال 1: حساب f

```
(2 ,2) :البداية
(12 ,12) :الهدف
(5 ,5) :العقدة الحالية

g(5,5) = (5,5) إلى (2,2) من المسافة الفعلية
       = 3 + 3 = 6 (حركة قطرية 3 مرات)
       = 3 * 1.414 = 4.242

h(5,5) = Manhattan: |12-5| + |12-5| = 7 + 7 = 14

f(5,5) = 4.242 + 14 = 18.242
```

مثال 2: اختيار أفضل عقدة

```
openSet = [
  Node(5,5, f=18),
  Node(3,3, f=15),  ← الأقل f
  Node(7,7, f=20)
]

نختار: Node(3,3, f=15)
```

مثال 3: فحص الجيران

```
current = (5, 5)

الجيران:
- (5, 4): أعلى ✓
- (6, 5): يمين ✓
- (5, 6): أسفل ✓
- (4, 5): يسار ✓
- (6, 4): أعلى يمين ✓
- (6, 6): أسفل يمين ✓
- (4, 6): أسفل يسار ✓
- (4, 4): أعلى يسار ✓

إذا كان (6, 5) جدار:
- نتخطاه ولا نضيفه إلى openSet
```

أسئلة المقابلة المتوقعة

السؤال 1: اشرح الصيغة $f(n) = g(n) + h(n)$

الإجابة:

- $f(n)$: التكلفة الإجمالية المتوقعة للمسار عبر العقدة n
- $g(n)$: التكلفة الفعلية من البداية إلى العقدة n (معروفة)
- $h(n)$: التكلفة المتوقعة من العقدة n إلى الهدف (تخمين)

السؤال 2: لماذا نختار أقل f؟

الإجابة: لأن أقل f تعني أن العقدة الأكثر واعدة للوصول للهدف بسرعة. نحن نوازن بين:

- المسافة المقطوعة (g): لا نريد أن نذهب بعيداً جداً
- المسافة المتبقية (h): نريد أن نقترب من الهدف

السؤال 3: ما الفرق بين openSet و closedSet؟

الإجابة:

- openSet: العقد المرشحة للفحص (لم نفحصها بعد)
- closedSet: العقد المفحوصة بالفعل (لا نريد فحصها مرة أخرى)

السؤال 4: كيف نعيد بناء المسار؟

الإجابة: عندما نصل للهدف، نتتبع الآباء للخلف:

```
current = goal
path = []
while (current != null) {
  path.add(current)
  current = current.parent
}
```

السؤال 5: ما الفرق بين Manhattan و Euclidean؟

الإجابة:

- Manhattan: h = |x_goal - x| + |y_goal - y| (للحركة بـ 4 اتجاهات)
- Euclidean: h = √(x_goal - x)² + (y_goal - y)²

السؤال 6: ما التعقيد الزمني؟

الإجابة: O(b^d) حيث:

- b: عامل التفرع (عدد الجيران المحتملين)
- d: عمق الحل (عدد الخطوات)

السؤال 7: هل الخوارزمية تجد أقصر مسار؟

الإجابة: نعم، إذا كانت الدالة الإرشادية admissible (لا تبالغ في التقدير). Manhattan و Euclidean كلاهما admissible.

السؤال 8: ماذا يحدث إذا لم يوجد مسار؟

الإجابة: إذا انتهت openSet ولم نجد الهدف، نرجع مصفوفة فارغة [].

السؤال 9: لماذا نستخدم nodeMap؟

الإجابة: للبحث السريع $O(1)$ بدلاً من البحث في Array الذي يأخذ $O(n)$.

السؤال 10: كيف تتعامل مع الجدران؟

الإجابة: قبل إضافة جار إلى openSet، نتحقق:

```
if (walls.has(`${newX},${newY}`)) continue;
```

إذا كان هناك جدار، نتخطاه.

ملخص سريع

المفهوم	الشرح
f(n)	التكلفة الإجمالية = $g(n) + h(n)$
g(n)	التكلفة الفعلية من البداية
h(n)	التكلفة المتوقعة للهدف
openSet	العقد المرشحة للفحص
closedSet	العقد المفحوصة
parent	العقدة السابقة
Manhattan	$\|h = \|x\_goal - x\| + \|y\_goal - y$
Euclidean	$h = \sqrt{[(x\_goal - x)^2 + (y\_goal - y)^2]}$
nodeMap	خريطة سريعة للعقد $O(1)$
neighbors	8 اتجاهات حركة

تم الإنشاء: ديسمبر 2025 **الهدف**: شرح تفصيلي ممل لكود خوارزمية A* فقط **المستوى**: شرح كل سطر بالتفصيل