Robot Path Planning Game - Documentation

جدول المحتويات

مقدمة عن اللعبة

Robot Path Planning Game هي لعبة تعليمية تفاعلية تهدف إلى تصور وفهم خوارزمية البحث **A*** **(A Star)** بشكل عملي وسهل.

المميزات الرئيسية:

- شبكة (Grid) بحجم 18×18 خلية
- روبوت يبدأ من نقطة البداية (أخضر)
- هدف يتغير عشوائياً (أحمر)
- عوائق (جدران) سوداء
- تحكم يدوي بالروبوت
- حل آلي باستخدام خوارزمية A*
- نظام تلميحات (Hint)
- حركة متدرجة للمسار (Animated Path)

شرح خوارزمية A*

ما هي خوارزمية A*؟

خوارزمية A* هي خوارزمية بحث عن المسار الأمثل (Optimal Path Finding Algorithm) تجمع بين مزايا:

- **خوارزمية Dijkstra**: البحث عن أقصر مسار
- **خوارزمية Greedy Best-First Search**: الاستخدام الفعال للمعلومات الإضافية

الصيغة الرياضية:

```
f(n) = g(n) + h(n)
```

حيث:

- **f(n)**: التكلفة الإجمالية المتوقعة للعقدة n
- **g(n)**: التكلفة الفعلية من نقطة البداية إلى العقدة n
- **h(n)**: التكلفة المتوقعة (Heuristic) من العقدة n إلى الهدف

خصائص الخوارزمية:

الخاصية	الوصف
شمولية (Complete)	تجد حلاً إذا كان موجوداً
الأمثلية (Optimal)	تجد أقصر مسار ممكن
الكفاءة	أسرع من Dijkstra في معظم الحالات
التعقيد الزمني	O(b^d) حيث b عامل التفرع و d العمق

الدوال الإرشادية (Heuristics):

1. Manhattan Distance (مسافة مانهاتن)

```
h(n) = |x_goal - x_n| + |y_goal - y_n|
```

- تُستخدم للحركة الأفقية والعمودية فقط
- أكثر دقة عندما تكون الحركة محدودة بـ 4 اتجاهات

2. Euclidean Distance (المسافة الإقليدية)

```
h(n) = √[(x_goal - x_n)² + (y_goal - y_n)²]
```

- تُستخدم للحركة في 8 اتجاهات (بما فيها الأقطار)
- أكثر دقة عندما تكون الحركة حرة في جميع الاتجاهات

طريقة التكويد والتطبيق

1. تعريف فئة Node

```
class Node {
  x: number;           // الموضع X
  y: number;           // الموضع Y
  g: number;           // التكلفة من البداية
  h: number;           // التكلفة المتوقعة للهدف
  f: number;           // f(n) = g(n) + h(n)
  parent: Node | null; // العقدة الأب (للتتبع)

  constructor(x: number, y: number, g: number, h: number) {
    this.x = x;
    this.y = y;
    this.g = g;
    this.h = h;
    this.f = g + h;
    this.parent = null;
  }
}
```

2. دالة الحساب الإرشادي

```
function heuristic(
  from: Cell,
  to: Cell,
  type: "manhattan" | "euclidean" = "manhattan"
): number {
  const dx = Math.abs(from.x - to.x);
  const dy = Math.abs(from.y - to.y);

  if (type === "manhattan") {
    return dx + dy;
  } else {
    return Math.sqrt(dx * dx + dy * dy);
  }
}
```

3. خوارزمية A* الكاملة

```
function aStar(
  start: Cell,
  goal: Cell,
  walls: Set<string>,
  gridSize: number
): Cell[] {
  // 1. تهيئة المجموعات
  const openSet: Node[] = [];       // العقد المراد فحصها
  const closedSet = new Set<string>(); // العقد المفحوصة
  const nodeMap = new Map<string, Node>(); // خريطة العقد

  // 2. إضافة عقدة البداية
  const startNode = new Node(
    start.x,
    start.y,
    0,
    heuristic(start, goal)
  );
  openSet.push(startNode);
  nodeMap.set(`${start.x},${start.y}`, startNode);

  // 3. حلقة البحث الرئيسية
  while (openSet.length > 0) {
    // أ. البحث عن العقدة بأقل f(n)
    let current = openSet[0];
    let currentIndex = 0;
    for (let i = 1; i < openSet.length; i++) {
      if (openSet[i].f < current.f) {
        current = openSet[i];
        currentIndex = i;
      }
    }

    // ب. التحقق من الوصول للهدف
    if (current.x === goal.x && current.y === goal.y) {
      // إعادة بناء المسار
      const path: Cell[] = [];
      let node: Node | null = current;
      while (node) {
        path.unshift({ x: node.x, y: node.y });
        node = node.parent;
      }
      return path;
```

```
    }

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

    // د. فحص الجيران (8 اتجاهات)
    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
    ];

    for (const neighbor of neighbors) {
      const newX = current.x + neighbor.x;
      const newY = current.y + neighbor.y;

      // 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;

      // 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;

      // التحقق من وجود مسار أفضل
      const existingNode = nodeMap.get(`${newX},${newY}`);
      if (existingNode && existingNode.g <= newG) continue;

      // إنشاء عقدة جديدة
      const newNode = new Node(newX, newY, newG, newH);
      newNode.parent = current;
      openSet.push(newNode);
```

```
        nodeMap.set(`${newX},${newY}`, newNode);
      }
    }

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

4. خطوات الخوارزمية بالتفصيل:

الخطوة 1: التهيئة

- إنشاء مجموعة openSet تحتوي على عقدة البداية
- إنشاء مجموعة closedSet فارغة
- تعيين $g(start) = 0$ و $h(start)$ = المسافة للهدف

الخطوة 2: حلقة البحث

- اختيار العقدة بأقل $f(n)$ من openSet
- إذا كانت الهدف، إعادة بناء المسار والخروج
- نقل العقدة إلى closedSet

الخطوة 3: فحص الجيران

- لكل جار لم يتم فحصه:
  - حساب g الجديدة
  - حساب h باستخدام الدالة الإرشادية
  - إضافة الجار إلى openSet إذا كان أفضل من المسار السابق

الخطوة 4: التكرار

- تكرار الخطوات 2-3 حتى إيجاد الهدف أو انتهاء openSet

الفوائد المضافة

1. التعليم التفاعلي

- **فهم عملي**: رؤية الخوارزمية تعمل بشكل مباشر
- **تصور بصري**: رسم المسار خطوة بخطوة
- **تجربة فورية**: تغيير الجدران والأهداف والتحكم اليدوي

2. الحركة المتدرجة (Animated Path)

```
// عرض المسار خطوة بخطوة بدلاً من الحل الفوري
let step = 0;
const animationInterval = setInterval(() => {
  step++;
  setGameState((prev) => ({
    ...prev,
    animatedPath: path.slice(0, step + 1)
  }));
}, 150); // 150ms لكل خطوة
```

الفائدة:

- فهم أعمق لكيفية تحرك الروبوت
- رؤية المسار الذي تتبعه الخوارزمية
- تعليم أكثر فعالية

3. الأهداف العشوائية (Dynamic Goals)

```
function generateRandomGoal(walls: Set<string>, gridSize: number, robot:
Cell): Cell {
  let goal: Cell;
  do {
    goal = {
      x: Math.floor(Math.random() * gridSize),
      y: Math.floor(Math.random() * gridSize)
    };
  } while (walls.has(`${goal.x},${goal.y}`) || (goal.x === robot.x &&
goal.y === robot.y));
  return goal;
}
```

الفائدة:

- تحديات متنوعة
- إعادة لعب غير محدودة
- اختبار الخوارزمية في حالات مختلفة

4. نظام التلميحات (Hint System)

```
const handleHint = () => {
  const path = aStar(gameState.robot, gameState.goal, gameState.walls,
GRID_SIZE);
  if (path.length > 1) {
    setGameState((prev) => ({
      ...prev,
      nextHint: path[1] // الخطوة التالية فقط
    }));
  }
};
```

الفائدة:

- مساعدة المستخدم دون حل المشكلة بالكامل
- تعليم تدريجي

- تحفيز على التفكير

5. تحرير الجدران (Edit Walls)

- إنشاء متاهات مخصصة
- اختبار الخوارزمية في حالات معقدة
- فهم تأثير العوائق على المسار

6. دعم Heuristics المتعددة

- **Manhattan Distance**: للحركة المحدودة
- **Euclidean Distance**: للحركة الحرة
- مقارنة الأداء والنتائج

المعلومات التقنية

معايير الأداء

المعيار	القيمة
حجم الشبكة	18×18 خلية
عدد الاتجاهات	8 (أفقي، عمودي، قطري)
وقت الحركة	150ms لكل خطوة
الحد الأقصى للخطوات	324 (18×18)

التعقيد الحسابي

```
Time Complexity: O(b^d)
Space Complexity: O(b^d)

حيث:
b = (عامل التفرع (8 في هذه اللعبة
d = (عمق الحل (عدد الخطوات
```

مثال عملي:

السيناريو: روبوت في (2,2) والهدف في (12,12)

1. **البداية**:
   - openSet = [Node(2,2, g=0, h=20)]
   - closedSet = []
2. **التكرار الأول**:
   - اختيار Node(2,2)
   - فحص 8 جيران
   - إضافة الجيران إلى openSet
3. **التكرارات التالية**:
   - اختيار العقدة بأقل f(n)
   - الاستمرار حتى الوصول للهدف
4. **النتيجة**:
   - مسار بـ 14-21 خطوة (حسب الجدران)
   - أقصر مسار ممكن

كيفية الاستخدام

التحكم اليدوي

- **أسهم لوحة المفاتيح**: تحريك الروبوت
- **WASD**: تحريك بديل

الأزرار

- **SOLVE**: تشغيل خوارزمية A* وعرض المسار
- **HINT**: عرض الخطوة التالية الموصى بها
- **RESET**: إعادة اللعبة للوضع الابتدائي

الإعدادات

- **Edit Walls**: تفعيل وضع تحرير الجدران
- **Heuristic**: اختيار نوع الدالة الإرشادية

الخلاصة

لعبة **Robot Path Planning** توفر:

- ✓ فهم عملي لخوارزمية A*
- ✓ تصور بصري للمسار الأمثل
- ✓ تجربة تفاعلية وممتعة
- ✓ أداة تعليمية قوية
- ✓ واجهة احترافية وسهلة الاستخدام

تاريخ الإنشاء: ديسمبر 2025 **الإصدار**: 1.0 **المطور**: Karam Ghanem