触屏版推箱子游戏中的走迷宫算法①

殷旭东, 周思林

(常熟理工学院 计算机科学与工程学院, 常熟 215500)

摘 要: 在 Android 等移动平台上的触屏版推箱子游戏中, 游戏主角在触摸操作下的移动路径计算属于走迷宫算法. 提出了一种基于右手法则、足迹标记和捷径优化的迷宫路径搜索算法, 采用直行、沿墙搜索和路径优化三个步骤实现. 经实际项目的应用验证, 该算法具有良好的有效性和高效性, 能够满足游戏操作的实时性要求.

关键词: 走迷宫算法; 推箱子游戏; 触摸屏; 路径搜索; Android

Maze Algorithm in Sokoban Game of Touch-Screen Edition

YIN Xu-Dong, Zhou Si-Lin

(School of Computer Science and Engineering, Changshu Institute of Technology, Changshu 215500, China)

Abstract: In the Sokoban game of touch-screen edition which runs on the mobile platform such as Android, the computing of the game protagonist's walking path under touch operation belongs to maze algorithms. A path searching maze algorithm based on the right-hand rule, footprint marking and shortcut optimizing is proposed. It is implemented through three steps, including walking straight, searching along the wall and optimizing path. After verification of application in the actual project, it shows that the algorithm has good effectiveness and efficiency to meet the real-time requirements of the game operations.

Key words: maze algorithm; Sokoban game; touch-screen; path searching; Android

1 引言

"推箱子"又称"仓库番(Sokoban)", 是一款经典益智游戏, 1982 年由日本 Thinking Rabbit 公司首次发行[1]. 该游戏的基本玩法是在一个二维矩阵组成的仓库内, 通过控制游戏的主角搬运工进行上下左右四个方向移动, 绕过障碍并把所有的箱子全部推到目标位置. 这款游戏已经被广泛移植到到游戏机、PC 机和手机等多种游戏平台. 这些平台一般都带有四个方向控制键或具有相应的控制机构, 可以方便地控制搬运工的移动操作.

近年来带有触摸屏的智能手机和平板电脑大量普及, 但因为没有方向控制按键, 给此款游戏的移植带来了很大困难. 现有的通常解决方案是在屏幕上划出一个专用区域, 放置四个虚拟方向按钮, 玩家通过点击虚拟按钮来操控搬运工的移动. 然而, 这种方式使玩家不能集中精力于游戏本身, 必须分心去观看和点击虚拟按钮, 使游戏的操控性和趣味性大大下降. 显然, 更为自然合理的操作方式应该是直接点击搬运工需要到达的目标位置, 让程序自动控制搬运工的移动. 而自动搜索搬运工的移动路径, 则是一个走迷宫问题.

迷宫问题是数据结构中的一个经典问题[2], 现有的走迷宫算法主要包括传统算法和智能算法等类别. 传统算法包括基于深度优先的右手法则[3]、向心法则[4]等算法, 以及基于广度优先的 Trémaux 算法[3]等. 该类算法效率很低. 智能算法主要包括遗传算法、模糊算法和蚁群算法等[2,5], 这类算法提高了效率, 能够获得近似最优解.

在"推箱子"游戏过程中, 会产生大量的搬运工自动移动操作, 所以对迷宫求解算法有实时性要求. 如果使用已有的这些算法, 则因为时间和(或)空间复杂度较高, 在计算能力有限的移动设备上运行, 会出现游戏操作不流畅、设备耗电过快等问题. 其次, 游戏中搬运工的移动路径是否最短, 则并不重要, 只须让搬运工的移动看起来足够聪明就能满足要求. 本文提出

①收稿时间: 2013-12-12;收到修改稿时间: 2014-01-03

一种适合于移动设备的走迷宫算法.

2 算法描述

2.1 算法主要思想

该算法的核心思想是采用右手法则[2]的沿墙搜索, 结合 Trémaux 算法[2]的足迹标记法, 并在求解过程中连续进行捷径检查优化. 主要分为 3 个步骤, 分别是直行、沿墙搜索以及路径优化.

2.2 初始化

根据迷宫当前状态, 准备好可及矩阵 R[m][n]和足迹矩阵 F[m][n][4]. 其中 R 表示迷宫各单元格能否通行, 即是否可及; F 用以记录各单元格通行时经过的方向; m,n 分别表示迷宫的行数和列数.

2.3 直行步骤

第一步的主要过程是从起始单元格 start 直接往目标方向前进, 直到到达目标或者碰到障碍为止. 首先判断目标单元格 target 处于当前单元格 current(初始化为 start)的方向, 其值为上下左右四个方向之一, 当两单元格不处于同一水平或垂直方向时, 选择垂直方向. 当该方向无障碍时, 前行一步, 并记录此路径. 重复上述步骤直至到达目标或者碰到障碍. 当到达目标时, 算法进入优化步骤 Step 3; 当碰到障碍时, 算法进入沿墙搜索步骤 Step 2.

算法 Step 1:　goStraight

输入: R 可及矩阵;start 起始单元格;target 目标单元格

输出: path 记录的路径; direction 最后一步前进的方向

算法过程: 图 1 给出了该算法步骤的流程图.

在这个算法步骤中, 每走一步都会判断目标方向并前进, 所以得到的路径并不一定是一条直线. 图 2 展示了一种典型的路径搜索步骤, 其中 S 为起始点(0,0), T 为目标点(2,3), 空白单元格处可通行, 阴影单元格为障碍(墙). 得到的搜索结果 path 为(0,0)-(1,0)-(2,0)-(2,1), 方向 direction 为向右.

2.4 沿墙搜索步骤

进入该步骤时, 当前位置已处于墙边, 且方向指向墙. 首先检测当前位置 current 能否与目标位置 target 经过一次转折连接, 即具有一条捷径. 如果满足条件, 则记录此捷径, 完成该步骤.

无捷径时, 向左转向, 使墙出现在前进方向的右侧, 即按照右手法则前进. 然后判断当前前进方向是否可通行, 如果不可通行, 则左转并检查新的方向; 如果可通行, 则记录此路径和足迹(即标记此单元格处的当前方向已被使用), 到达新位置以后立即右转, 使其右侧靠墙或者指向墙.

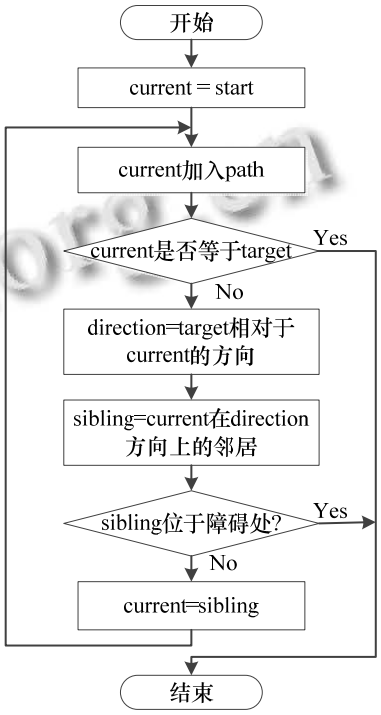

图 1　直行步骤流程图

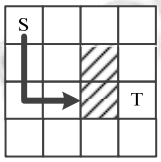

图 2　直行步骤的一种典型结果图

重复进行上述过程, 直到到达目标位置或者找不到未用过的新方向为止. 前者表示成功搜索到一条路径, 此时进入优化步骤 Step 3. 后者表示搜索失败, 不存在一条可达目标位置的路径.

算法 Step 2:　goAlongTheWall

输入: R 可及矩阵; F 足迹矩阵; initDirection 初始方向; path 已搜索路径; target 目标单元格

输出: path 记录的路径(包含输入路径)

算法过程: 图 3 给出了该算法步骤的流程图.

在此步骤中, ①处的处理过程使用足迹矩阵 F 记录检测过的方向, 这能够防止重复走过的路径, 保证算法不会陷入死循环, 能够不断搜索新路径.

图 4 展示了一个典型的沿墙搜索过程. 其中, S 为该算法步骤起始点(2,1), T 为目标单元格(2,3). ①为起始方向向右, 先左转为向上方向②. 因上方单元格(1,1)可通行, 则前进一步, 右转后方向为向右③, 右侧为墙, 故左转为向上方向④. 上方单元格 C 即(0,1)处可通行, 则前进到 C. 因 C 与 T 之间能够通过直线 CR 与 RT 连通, 构成了一条捷径, 故路径搜索成功. 最后得到的搜索结果为 (2,1)-(1,1)-(0,1)-(0,2)-(0,3)-(1,3)-(2,3).

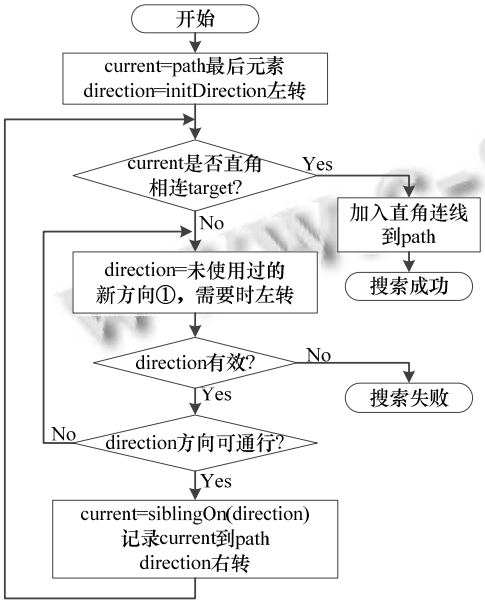

图 3 沿墙搜索步骤流程图

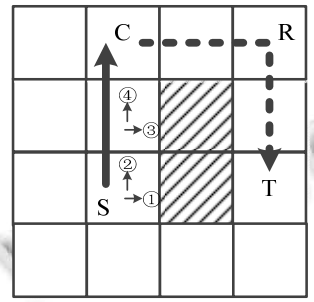

图 4 一个典型的沿墙搜索步骤

2.5 路径优化步骤

当存在至少一条有效路径时, 经过前面 2 个算法步骤, 能够保证找到一条路径. 在这个步骤中对搜索到的路径进行两方面优化: 1)合并重复单元格; 2)捷径优化, 即两个单元格之间可形成一条直线连通或者经过一次转折连接的两条直线连通. 实际上, 1)是 2)的一种特例.

在这个步骤中, 从路径中首末两端开始选择 2 个单元格, 进行上述优化处理, 并逐步向中间合拢.

算法 Step 3: optimize

输入: R 可及矩阵; path 优化前的路径

输出: path 优化后的路径

算法过程: 图 5 给出了该算法步骤的流程图.

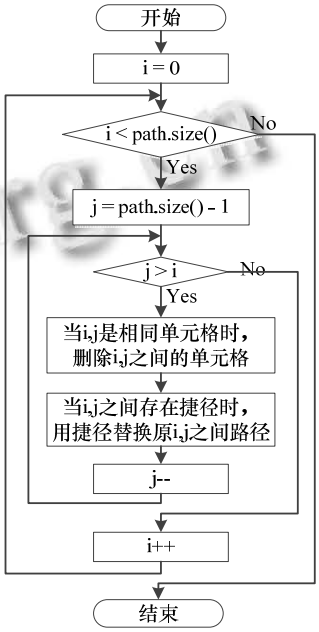

图 5 优化步骤流程图

图 6 与图 7 分别展示了该算法步骤中两种优化的前后效果对比. 其中 S 为起始点, T 为目标点.

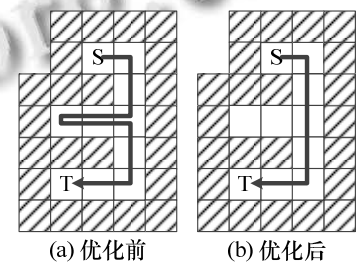

图 6 合并重复单元格优化效果对比图

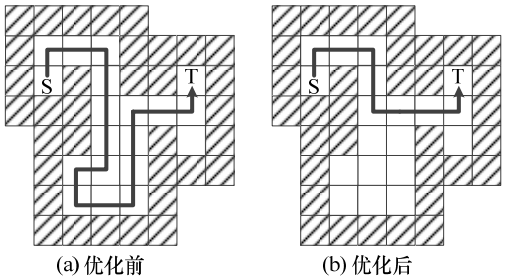

图 7 捷径优化效果对比图

3 算法指标与应用

3.1 算法指标

由算法描述和实现过程可知, 该算法的时间复杂度为 O(n1.5), 其中 n 为迷宫单元格数量. 求解时分配的空间和 n 成正比, 所以空间复杂度为 O(n). 此外, 该算法获得的不是最优解或者相对最优解, 仅为一个较优解.

3.2 算法应用

该算法用 Java 语言实现, 类名 MazePathResolver, 使用构造器注入游戏场景的二维矩阵信息 CellMatrix. CellMatrix 定义为接口, 是对 “推箱子”游戏场景的一个抽象, 提供仓库矩阵的尺寸和障碍物位置等信息. MazePathResolver 的公有方法 findOptimizedPath()为算法求解入口, 根据起始单元格 start 和目标单元格 targer 求解搜索路径 path.

在 Android 平台上开发了一款 “吉豆推箱子”游戏, 使用本算法实现了对搬运工的触摸控制操作. 图 8 为该游戏在模拟器上的操作界面.

图 8 “吉豆推箱子”游戏界面

4 结语

在“推箱子”游戏等场合, 走迷宫算法要求具有实时性好效率高的特点, 即需要一种快速有效直观但不需要最优解的算法. 本文提出并实现了基于右手法则、足迹标记和捷径优化的算法, 并在实际游戏项目中得到了应用. 在 ZTE V880 等多型号的 Android 手机上, 游戏操控灵敏、实时性好, 并且搜索路径显得较为聪明, 符合游戏玩家的直观思维. 因此, 本算法特别适合于智能移动平台.

参考文献

1 Sokoban official site. http: //www.sokoban.jp/.
2 徐守江.迷宫算法综述.信息与电脑,2009,(10):91–92.
3 Poundstone W,李大强.推理的迷宫:悖论,谜题,及知识的脆弱性.北京:北京理工大学出版社,2005:195–198.
4 贺少波,孙克辉.基于向心法则的电脑鼠走迷宫算法设计与优化.计算机系统应用,2012,21(9):79–82.
5 张捍东,郑睿,岑豫皖.移动机器人路径规划技术的现状与展望.系统仿真学报,2005,17(2):439–443.