人工智能导论

搜索大作业报告

——十滴水

李宏坤 2011011445

Tel：18810699601

Email：lihk11@mails.tsinghua.edu.cn

一、 游戏介绍

“十滴水”是一款有趣的小游戏，我们可以点击向格子中添加小水滴，当格子中水滴成型时再加入水，水滴就会涨破，水滴涨破后，会向上下左右四个方向溅射出四滴小水滴，小水滴碰到其他格子中水滴，又会向其中添加小水滴，从而发生连锁反应，当格子中水全涨破后，即获得游戏胜利。当每有一次连击时，便会奖励玩家一滴水。当玩家的水滴用完时，且大水滴没有全部爆裂，则判定玩家输掉了游戏。

二、 程序介绍

本次作业，我选择了 Mac OS X 的平台，采用 Objective-C 进行编写，使用 Xcode6.0 进行编译。

其中，游戏部分采用了 SpriteKit 游戏引擎，素材提取自 App Store 的安装包，仅用作学习与测试之用。

本程序在 OS X Yosemite（10.10）上调试通过，未针对更低版本的 OS X 进行优化（有可能在控件的显示上存在问题），建议老师使用 OS X Yosemite 作为测试运行平台。

另，随作业附上本程序的**运行视频**，以说明理想的运行状态，如老师电脑上显示有问题，请您联系本人进行调试与演示。

程序 UI 介绍：

（1） 程序初始状态为 AI 模式，即无需用户点击即可进行操作；

（2） 当一个 Level 通关后，点击 Next Level 的标签即可进入下一关；

（3） 点击游戏底部的标签，即可切换 AI Mode 与 Manual Mode；

（4） 当前剩余水滴数显示在背景上；

（5） AI 模式下，在加入一滴水后，当飞行的小水滴全部消失 3 秒后，会自动加入下一滴水，并且将其坐标显示在屏幕上；

（6） 每连击三次，即可获得一滴水的奖励，并显示在屏幕上；

程序运行截图，

三、 算法介绍

在本次作业中，采用了两种算法：

（1） 迭代加深搜索（terative Deepening）简称 ID 算法

（2） 贪心算法

具体算法实现：

（1）迭代加深搜索算法

迭代加深搜索算法拥有宽度搜索与深度搜索的优点。其思想在于限制深度搜索的深度，将从 1 到 MaxDepth 所有的状态枚举一遍，从中选取最优的的一个，作为搜索的结果。其优点在于，空间消耗很小，基本上与深度搜索差不多，不会向宽度搜索一样消耗过大的内存空间，同时也不会像深度搜索一样，对于非常深的子树，ID 算法可以及时停住，而像

宽度搜索一样，一层一层的搜索。

其算法可以描述如下，

对于当前状态，枚举到对（i,j）的泡泡加入一滴水

if 加入后小于等于 4

直接得到新的状态，并对新的状态进行枚举;

else //表示这滴水已经爆裂

然后,模拟此水滴爆裂产生的四个小水滴对其他水滴的影响。

将四个小水滴加入到扫描队列中,根据其坐标位置来判断是否撞裂了大水滴，如果撞裂了新的大水滴，则移除该小水滴，并在队列中加入四个新的小水滴。

当小水滴的坐标超出边界时，也将该小水滴移出队列。

当队列为空时，表示水滴爆裂的影响已经完成。

这时，得到了新的状态，并对新的状态进行枚举。

如果，新得到的状态中，每个节点的大水滴均爆裂，得到一个解;

根据当前解消耗的的水滴数与之前最佳的解进行比较，如果消耗的水滴数较少，则使当前解作为最佳解。

相关的代码如下图，

```
for (i = 1; i <= 6; i++) {
    for (j = 1; j <= 6; j++) {
        CGPoint position = CGPointMake(i, j);
        MyPoint *point = [[MyPoint alloc] initWithX:position.x y:position.y];
        if (position.x == 4 && position.y== 4) {
            point = [[MyPoint alloc] initWithX:4 y:4];

        }
        [currentMethod addObject:point];
        NSTimeInterval st = [NSDate timeIntervalSinceReferenceDate] * 1000;
        NSMutableArray *newState = (NSMutableArray *)[self addDropOnPosition:
            position onArray:currentstate];
        currentRemainDrop = currentRemainDrop - 1;
        NSLog(@"addDropOnPosition:%f", [NSDate timeIntervalSinceReferenceDate]
            * 1000 - st);
        isClear = YES;
        for (k = 0; k < 36; k++) {...}
        if (isClear) {
            //在这里比较方法
            if (currentRemainDrop > formerRemainDrop) {
                formerMethod = [NSMutableArray arrayWithArray:currentMethod];
                formerRemainDrop = currentRemainDrop;
                currentRemainDrop = currentRemainDrop + 1;
                [currentMethod removeLastObject];
                return;
            } else {
                currentRemainDrop = currentRemainDrop + 1;
                [currentMethod removeLastObject];
                return;
            }
        } else {
            int newdepth = depth + 1;
            [self caculateForState:newState depth:newdepth];
            currentRemainDrop = currentRemainDrop + 1;
            [currentMethod removeLastObject];
        }
```

（2） 贪心算法

对于初始状态，假设有 n 个大水滴，则对这 n 滴水分别加一滴水，选出其中最好的，作为下次搜索的初始状态，直到得到一个解为止。

评判一个状态是否是好状态的标准：

对每个大水滴拥有的 1~4 的四个种类，对该其求算数平均数，对于平均值较大的状态成为比较好的状态。

```
currentValue = sum / arraySpriteCount;
```

其具体算法可以参考下图，最大的搜索深度设为了 20；

```
-(NSMutableArray*)solveWithEstimation{
    //先计算当前所有加一滴水，选出其中个最好的，（如果没有clear）分别再加一滴水
    //找出当中最好的，作为下次的结果
    NSMutableArray *currentState = [NSMutableArray arrayWithCapacity:
        originalMapArray.count];
    int i;
    for ( i = 0; i<originalMapArray.count; i++) {
        NSString *number_string = [originalMapArray objectAtIndex:i];
        NSNumber *number = [NSNumber numberWithInt:[number_string intValue]];
        [currentState addObject:number];
    }
    for (i=0; i<20; i++) {
        NSDictionary *resultDictionary = [self solveWithEstimationWithArray:
            currentState];
        NSNumber *position_x = [resultDictionary objectForKey:@"x"];
        NSNumber *position_y = [resultDictionary objectForKey:@"y"];
        NSNumber *maxValue = [resultDictionary objectForKey:@"maxValue"];
        currentState = [resultDictionary objectForKey:@"map"];
        MyPoint *position = [[MyPoint alloc] initWithX:[position_x intValue] y:
            [position_y intValue]];
        [currentMethod addObject:position];
        if ([maxValue floatValue] == 0) {
            return currentMethod;
        }
    }
    return currentMethod;
}
```

四、 算法分析

（1）迭代加深搜索算法

每加入新的一滴水，则会产生一种状态，每个状态又可以生成 36 个子状态，计算可以得到，

dep =4 时，状态数是 36^4 ,约为 168 万个状态，

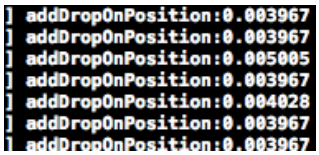
dep = 5 时，约为 6000 万个状态，

dep= 6 时，约为 21 亿个状态。

当 MaxDepth 设置的比较小时，迭代加深搜索算法可以很快的得到结果，但是当 MaxDepth 稍大时，计算出最优解已经需要相当长的时间了。

在本人的 MacBook Air 上，对每次状态的枚举消耗的时

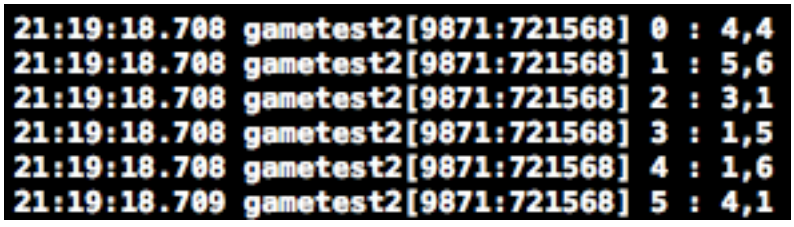
间为 0.003~0.2 毫秒（如图），由于超极本的性能比较低，Maxdepth = 4 已经需要约 2~3 分钟，当 Maxdepth = 5 时，已经要 1~2 个小时，接近于不可解。但是使用此算法可以解出在当前深度的最优解（如果有解的话）。



（2）贪心算法

贪心算法虽然不一定总能得到最优解，但是每关所消耗的水滴数一般小于 5 滴，多一些的 10 滴已经能够满足，极少出现 10 滴以上的水滴。对于贪心算法，在几步之内便可得到结果，并且结果也不会太差，大多数情形就是最优解，或者接近最优解。

譬如，对于其中的一关，其所消耗的水滴数为 6，由贪心算法所解得的结果为，图中 4，4 代表水滴所在的格子的坐标。



两种算法相比较：

对于迭代加深搜索算法，可以用来解最优解，但是时间开销与搜索深度成指数变化的关系（$t \sim 36^n$），不适合作为程序的动态演示。

对于贪心算法，虽然不一定能够得到最优解，但其给出的解已经接

近或者就是最优解了，而且从时间开销上来看是基本是固定的，仅需不超过 100 次的运算便可以得到比较好的解。比较适合作为程序的动态演示。

图为 AI 模式下，自动运行并显示加入水滴坐标的效果图，

五、心得体会

1、 关于游戏的编写

在本次作业中，自己使用 Apple 公司出品的游戏引擎 SpriteKit，初步了解的游戏开发的基本流程与基本思想，同时也实现了比较好的视觉效果。

2、 关于搜索

对于搜索，本次作业采用了两种算法：迭代加深搜索算法和贪心算法。其中迭代加深搜索算法吸取了宽度优先与深度优先的长处，加深我了对于宽度优先与深度优先的理解。而贪心算法则从启发函数的角度来解决问题，体现了具体问题具体分析的思想。