贪吃蛇游戏与简易AI蛇

张琦竣 2018级 计算机科学与技术

目　录

基本游戏逻辑的梳理

1.1 贪吃蛇类的分析与设计

有没有想过，当我们面对一个要开发的游戏甚至其他软件的时候，我们应当怎么做呢?
直接打开VS写上#include<iostream>?
或者是先整一个XXX.h出来，想到什么要用的东西就声明出来?

经验表明，这种直接进入代码环节的方式都是不合适的。当我们面对软件需求的时候，先进行“建模”，完全分析透彻之后再开始代码，这才是一种更高效的方式。所谓建模，简单地说就是为软件画一个蓝图。

显然，第一点需要明确的就是，这个软件涉及到哪些东西，这些东西之间有着什么样的关系，也就是说，需要对类进行明确。

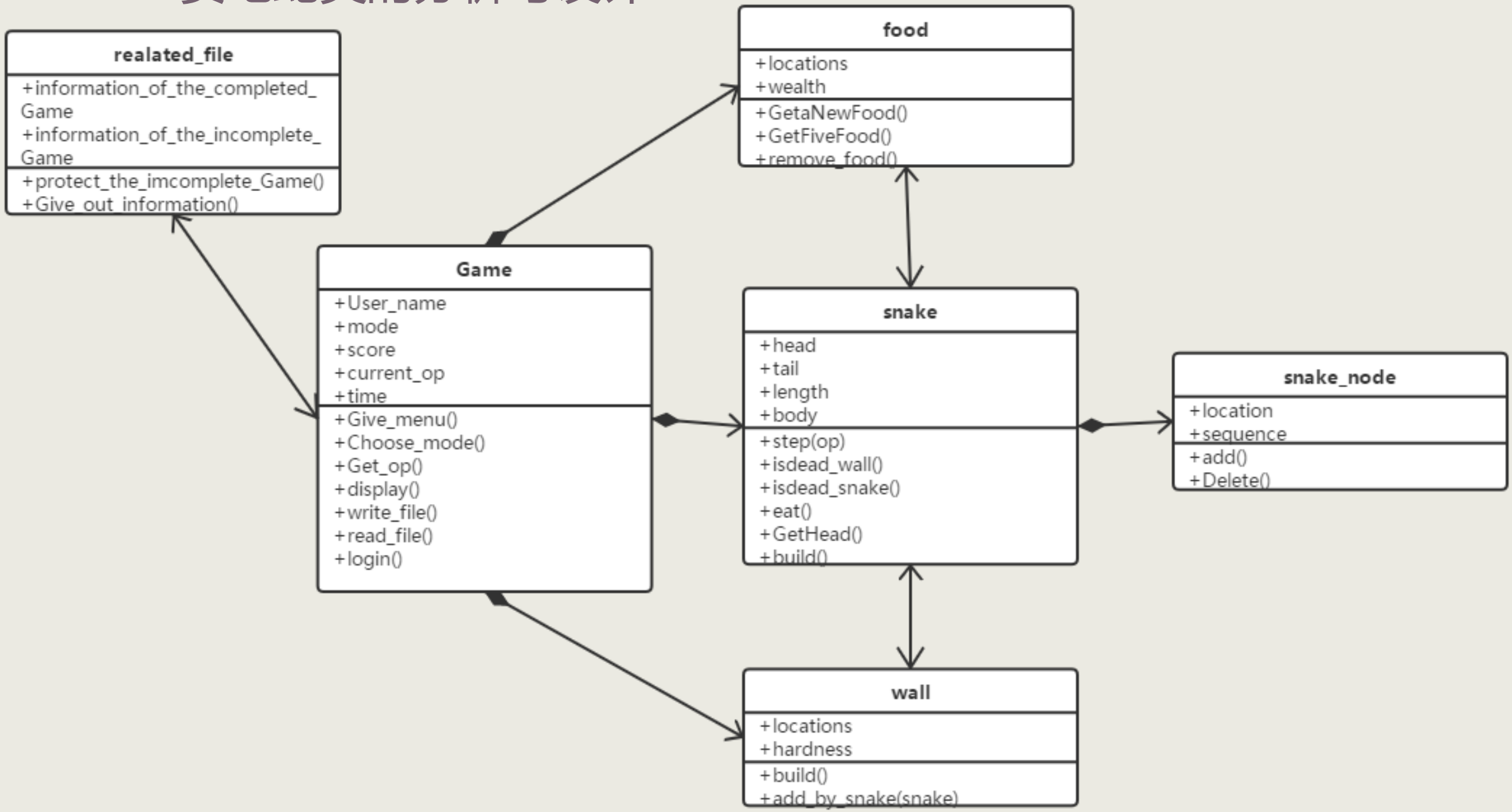
1.1 贪吃蛇类的分析与设计

那么，对于我们的贪吃蛇来讲，涉及到哪些类呢?

他们之间存在什么样的关系呢?

一种叫“类图”的东西便是一个很好的工具
我们通过这种图，来明确以下几点:
1.究竟要涉及哪些类
2.这些类之间存在什么样的关系
3.每个类中，比较要紧的”属性”有哪些
4.每个类中的“方法”（能够进行的操作）有哪些

1.1 贪吃蛇类的分析与设计



1.1 贪吃蛇类的分析与设计

上述图便是关该软件的类图，涉及到6个类：蛇、墙、食物、相关文件、游戏、蛇身结点。

值得注意的是，并不是这6个类就一定全在程序中以class的形式实现出来，具体情况应具体考虑。

接下来便来详细地讨论这6个类，主要讨论是否要在程序中以class的形式实现，对于要实现的类，分析其要紧的属性、并进可能详尽地考虑其可能执行的主要操作。

1.2 各类的设计

1.蛇
{
要紧的是蛇头、蛇尾、蛇身的位置

主要操作:
可以移动
可以判断是否撞墙
可以判断是否吃自己
可以吃食物
可以直接得到蛇头的位置
蛇刚开始的形成
}

2.蛇身
{
要紧的是该节身体的位置，以及其在整个蛇中的位置、（蛇身节点之间有顺序）

主要操作
增加一段蛇身
删除一段蛇身
}

1.2 各类的设计

3.食物
{
要紧的是每个食物的位置，以及其价值

主要操作
生成1个新的食物
生成5个新的食物
移除某个食物
}

4.墙
{
要紧的是哪里有墙，以及墙的硬度

主要操作
开始时生成四周的墙
蛇死亡后生成墙
}

1.3 类设计的反思

Game与realated_file是否写为类

1.Game
1）在该程序中多次出现
2）每次出现均不完全相同，有部分相同点，也有很多的不同点------>继承（后续知识点，暂不考虑）

因此我个人并没有把Game以类的形式实现，是采用了函数封装的形式；但是仍然还是可以封装为类的，比如可以把每个模式都当做是Game类的成员函数，等等，如果大家有更好的方式，欢迎大家积极提出哦

1.3 类设计的反思

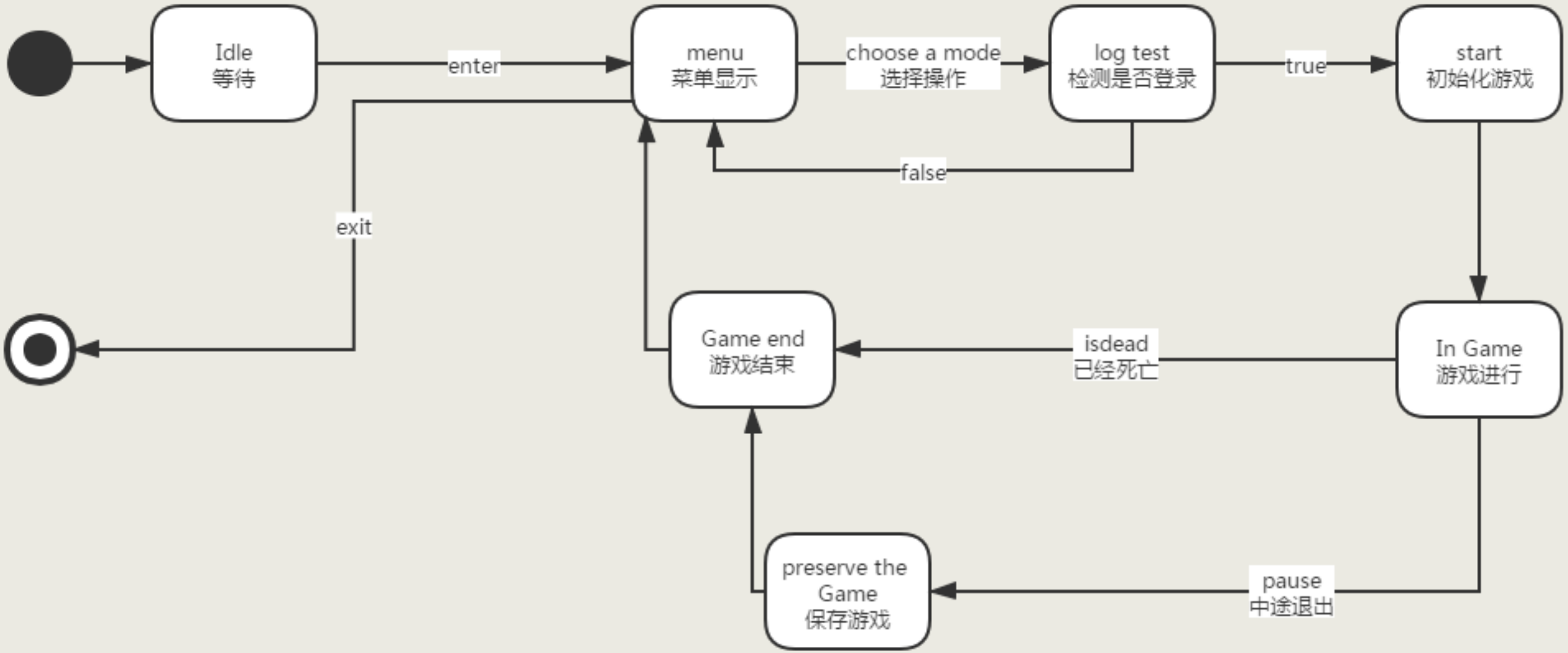
Game与realated_file是否写为类

2.realated_file
1）具有的属性不体现在程序中，而保存在外部文件中
2）仅有行为而没有属性，适合用函数的形式直接实现（个人观点，因为之前看过一个把遗传算法刻意封装成一个class的，调用起来出了不少错）

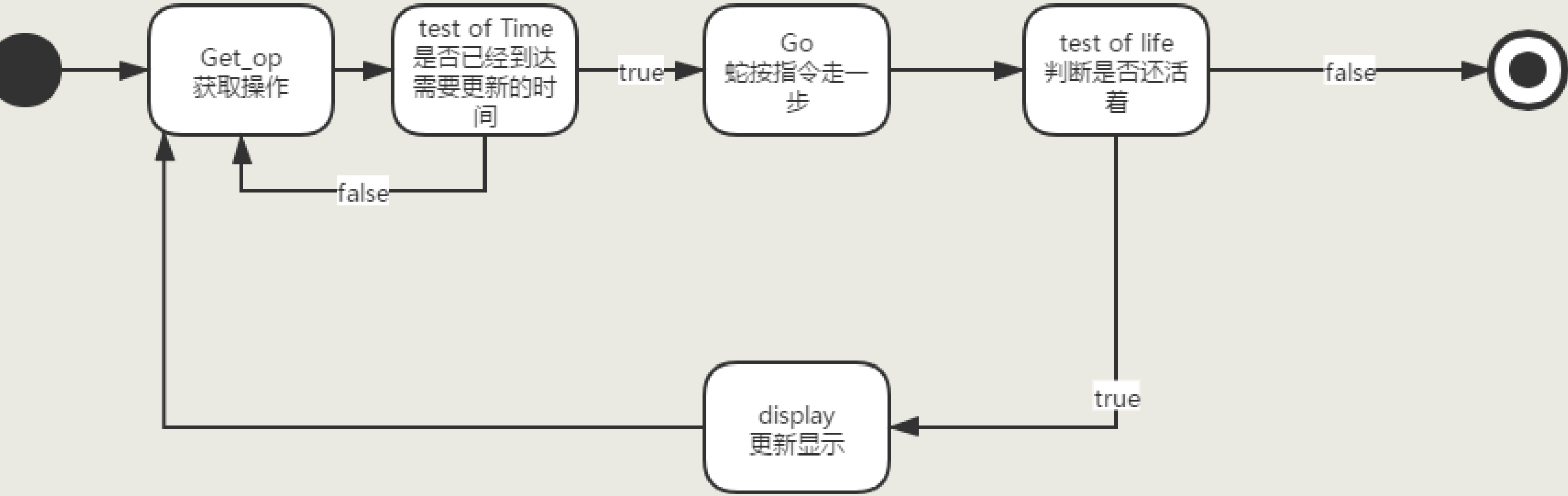
1.4 仔细考虑状态转化非常有助于软件开发（软件整体状态图）

我们已经明确了需要什么东西，当然，下一步就是要明确整个游戏或者说软件应该是如何运行的，一个叫状态图的东西便是很好的工具

1.4 仔细考虑状态转化非常有助于软件开发（软件整体状态图）



1.4 仔细考虑状态转化非常有助于软件开发（In Game内部状态图）



1.4 蛇的移动（Go）

1.头增尾减
蛇每次移动（不吃墙、不吃食物、不吃自己的情况下），都相当于之前的头部结点之前又多了一个结点，而尾部的最后一个结点则应该被删除，且显然，蛇的身体的每个结点之间是存在先后顺序的。
根据这种行动特性，因此选择采取链式队列的数据结构来存储蛇（说白了就是链表，每次只操作第一个结点和最后一个结点，选择的原因是因为链表的增删时间复杂度是O(1)，且增删操作较简单）

2.吃食物

如果蛇的该步吃到了食物，那蛇会增长一节，也就是说，尾巴本来应该删掉的，如果吃到了食物就不删了，这也是为什么处理头部，再处理尾部的原因。

3.尾巴缩短会导致的问题

考虑一种极端情况，某一步蛇头走到了蛇尾的位置，这时蛇尾还没有被删除，会判断出蛇吃到了自己，但是显然，这并没有吃到自己，因此这就是为什么在蛇的类中要记录尾巴的位置。

1.4 蛇的移动（Go）

4.判断死亡

因此，判断蛇如何算死亡的方法如下：

1）若该步使蛇的头走到了墙的位置，则蛇死亡

2）若该步使蛇的头走到了蛇的位置，且该位置不是本蛇尾巴的位置，则蛇死亡

旧版本的加载与多人游戏

2.1 旧版本的加载

1.存什么?
1）蛇----->怎么存蛇
2）墙
3）食物
4）时间、分数

2.怎么接?
1）续上时间、分数
2）墙、食物
3）蛇----->怎么再现

2.1 旧版本的加载

1.存蛇的方式
由于蛇是链式存储的，无法直接通过write(addr,size)的方式直接把蛇存到文件中，因此应把蛇的每个身体结点的位置坐标按顺序复制到一个数组中，再一次性写入文件（注意应先存入蛇的长度）
2.再次打开文件时复现蛇的方式
先读入长度，之后无论是逐个读入并随时建立链表也好，还是直接开个数组一次性复制过来也好，只要长度有了，就知道要读取多长的内容了。

3.墙、食物、时间、分数等信息，可以逐个直接存入文件，但是应注意的是，读出与写入的顺序应当相同。

2.2 多人游戏中蛇的移动

1.头增尾减

2.尾减的影响

3.增强复用性，解耦

4.死亡判断（吃墙、吃自己、吃别人）

2.2 多人游戏中蛇的移动

1.头增尾减
蛇的移动仍先采取单个蛇移动的方式。

2.尾减的影响
但是应注意的是，为了操作简单，虽然在该时刻多个蛇都需要动，但是仍是采取的逐个移动的方式，前一个动完紧接着下一个再动，正是由于这种方式，导致在判断“不同的蛇之间是否相撞”的时候，如果当前蛇的蛇头触碰到了已经移动完的蛇，则直接判断死亡，如果碰到了未移动的蛇，则还需要判断碰到的是不是尾巴。

3.增强复用性，解耦

显然，2中描述的这种判断方式使得不同的蛇在移动的时候不能执行同样的操作，导致代码耦合程度较高，复用性不强，因此，应当换一种新的判断死亡的方式。

4.死亡判断（吃墙、吃自己、吃别人）

仍旧采用单蛇的时候判断死亡的方法，当某个蛇移动的时候就能判断出他是否吃了自己或吃了墙，而对于是不是吃了别人，最后通过一个专用的函数判断：在所有蛇移动一波后，如果某个蛇的头部在另一个蛇的身体中出现，则判定蛇死亡。

贪吃蛇AI与人机博弈

3.1 两种常用的决策逻辑

1.对接下来要进行的所有行为进行评估，贪心地选出最佳行为

2.通过搜索的方式对接下来可能的几步进行事先模拟，从而择出最优解

3.2 贪吃蛇AI

研究贪吃蛇的行为方式，看看如何评估接下来的行动

1.喜欢走直线
2.“贪”----贪心策略----以最简单的方式吃食物

评估因素：
1.正面因素：食物与蛇头的位置关系
2.负面因素：障碍与蛇头的位置关系

3.2 贪吃蛇AI

评估方式：

1.如果蛇头的某一侧直线看过去就能看到食物（中间没有障碍），则蛇往该方向走

2.否则，判断蛇头的哪个方向离障碍物最远，蛇往该方向走。

3.3 贪的问题

贪------->抢

问题：
1.多条蛇只顾当前利益，去抢同一个食物，导致撞死
2.在追食物的过程中绕到一起

解决方案：别太贪，多观望一下
当某个蛇朝着食物走的时候，判断左前方和右前方的两个位置是不是有障碍，如果有，则不再朝着食物的方向走

3.4 经典问题求解算法

经典的问题求解搜索算法可以分为2大类

一种是着重于单智能体的求解算法（以A*为代表）

一种是着重与多智能体的博弈搜索（以极小极大算法为代表）

3.5 A*算法

A*算法

A*搜索算法是一种有信息搜索算法，通过评估函数f(n)=g(n)+h(n)的信息来进行搜索，每次选择扩展结点时，均选择开结点表中f(n)最小的结点来进行扩展，当扩展目标结点时，搜索算法完成（说白了还是贪，但又不太贪）。

其中重要的是，保证【最优性】的条件为：【可采纳性】和【一致性】。【可采纳性】是指h(n)应是一个可采纳启发式，应当从不会过高估计到达目标的代价。【一致性】是指，如果对于每个结点n和经过行动a后生成每个后继结点n’，从结点n到达目标的估计代价不大于从n到n’的单步代价与从结点n’到达目标的估计代价之和。

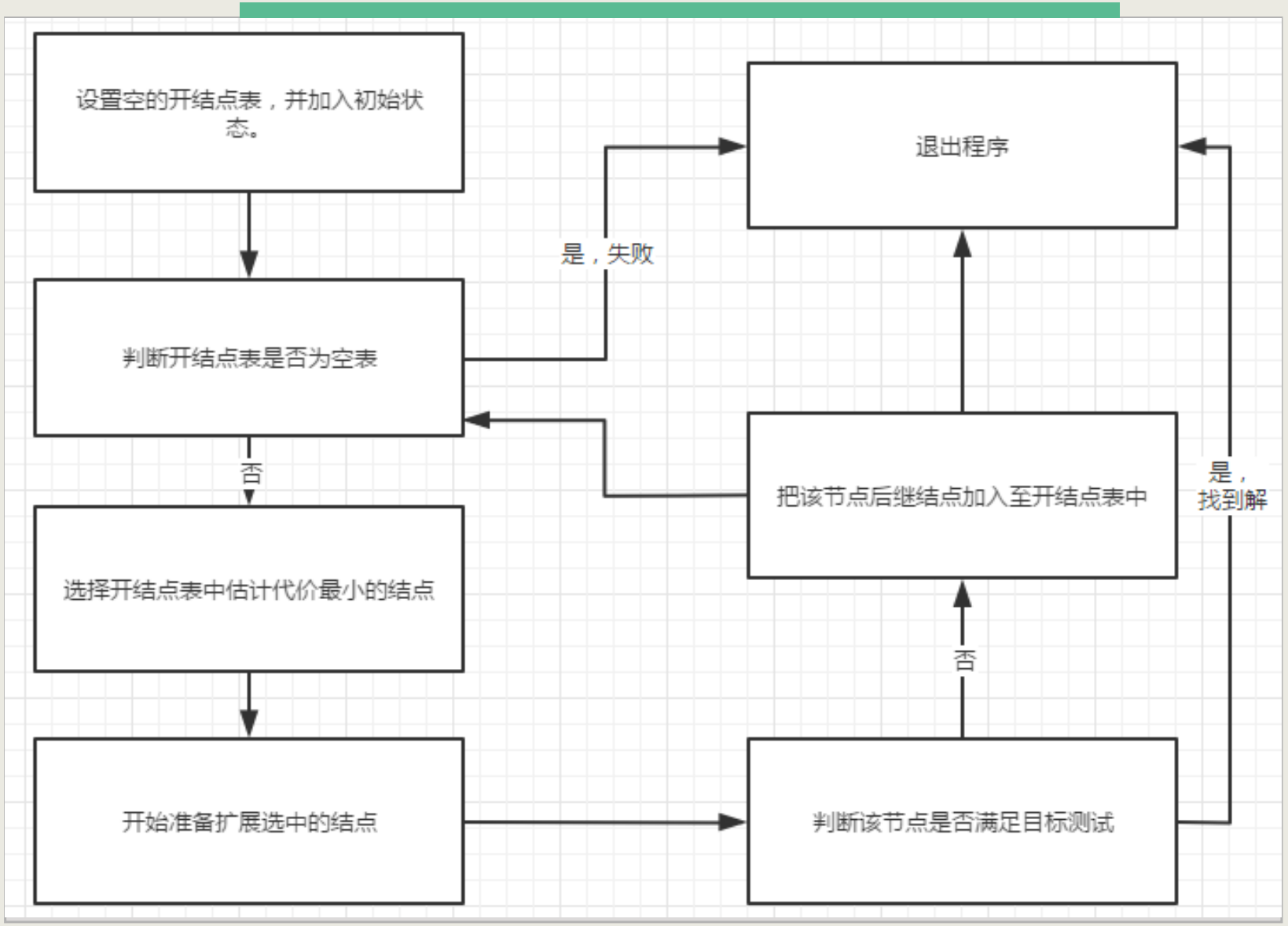
3.5 A*算法

算法详细步骤

- 设置空的开结点表，并加入初始状态。
- 如果开结点表为空，则退出搜索，失败。
- 选择开结点表中估计代价最小的结点，对其扩展。
- 若该结点满足目标测试，则成功得到问题的解，退出。
- 把这个的后继结点加入至开结点表中。
- 返回第2步。

3.5 A*算法

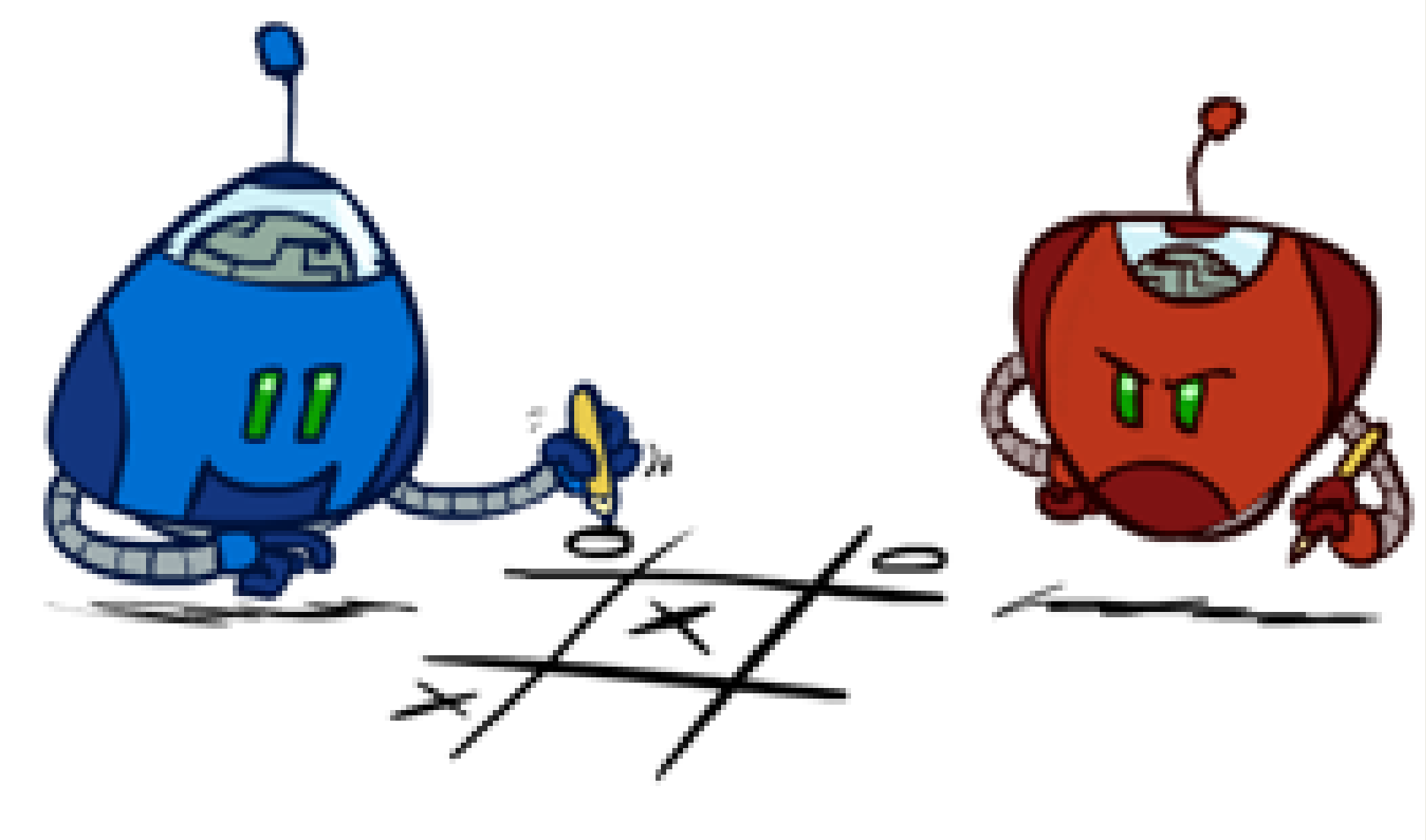
算法详细步骤



3.6 小应用

如果我们的蛇只有一条，且这个蛇只有头，食物是有限个，吃完后不再生成，障碍物静止地分布在场上，如何才能走最短的路径去吃掉所有食物?

3.7 极小极大算法



3.7 极小极大算法

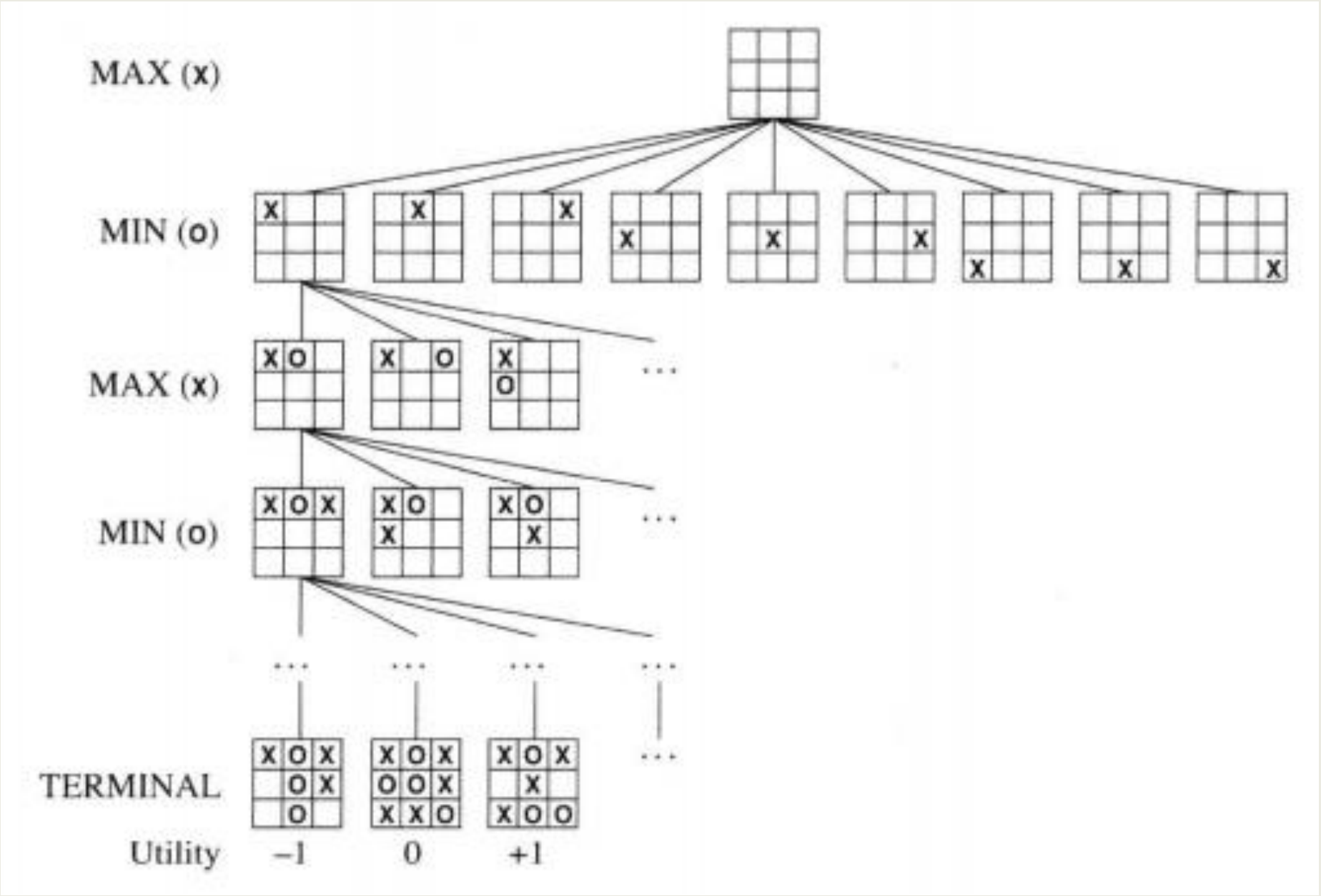
我想要结局尽可能的小：

1）我得选择最有利于我的步骤

2）由于下一步可能不是结局，我可能无法直接选择对我最有利的步骤

3）如果这样的话，我得猜猜对面的家伙他可能得怎么害我

3.7 极小极大算法



Copyright 《artificial intelligence : a modern approach》

感谢大家的批评指正

THANK YOU FOR WATCHING