MicroRts第二次实验报告

09017423 杨彬 09017421 胡世杰

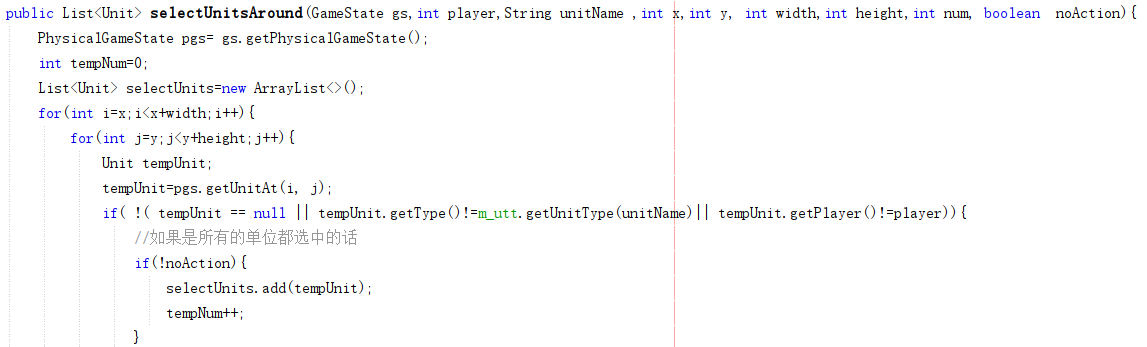
**Part 1 AI开发思路**

**一 AI基础**

至今为止，我们已经大致编写完AI的生产，采矿，造兵营，进攻，防御等基础操作，也在决策方面有了一定的进度，具体内容如下：

基础方面：

三周前我们仅仅是运行了文档里提供的最最基础的AI。在对每个AI有了一定了解并运行观察后，我们发现WorkerRush在8×8的地图中几乎是最强的，但是地图扩张为16×16后，WorkerRush难以在开局就摧毁基地，导致最后失败，所以兵营的意义已经非常大，我们认为是必须建造的。基于WorkerRush有兵就上的进攻方式的失败，我们开始的设想是能否做到像即时战略游戏中那样先集结兵力，再发动总攻。于是写了框选目标区域内指定兵种的函数：



但是之后经过激烈的讨论和实验后，我们发现在这个游戏框架下，或者说是在目前16×16或更小的地图内，集结兵力似乎并无可能且毫无意义：因为即使我方兵力在集结，但由于地图限制，集结方位只能在左上或者右下，而此框架中的寻路算法好像就是先偏向一个角再向敌方的移动方式（应该是，不太确定），所以要么在集结过程中发生遭遇战，要么就是基地被直接攻击。所以我们放弃了这个想法，但是框选函数在后面可能派的上用场所以进行了保留。

随后我们准备进行决策方面的工作：

决策即指我们该在什么情况下执行什么策略，而为了能够选择，我们要先实现底层函数

开始时，由自带AI的方式（Rush、Defense）决定将函数决定划分为两部分，由两个函数表示：1、防御策略 2、进攻策略

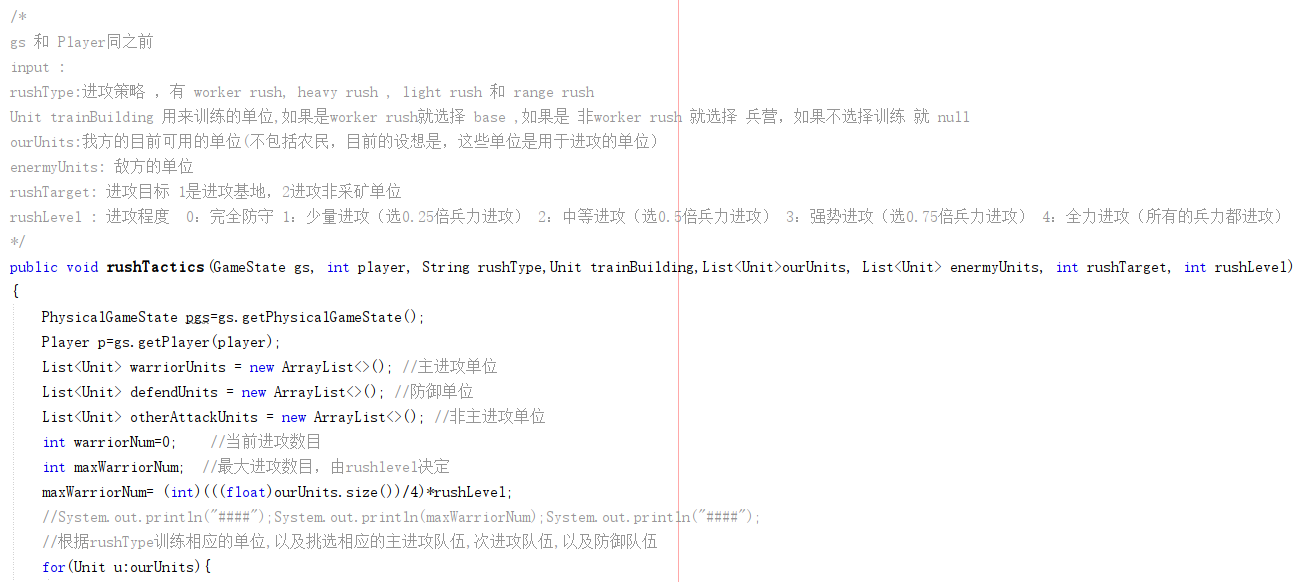
防御策略，就是决定主力兵种类型进行防守

显而易见，防守的策略实际上没有太多的参数以及过于复杂的操作，唯一要做的就是如何控制兵种使其被生产后不会堆积在生产建筑周围（若是堵住生产建筑的四面，则生产建筑无法再进行生产），这实际上与最开始的集结假设差不多，只是这次是为了防守。

进攻策略，就是决定主力兵种类型进行进攻

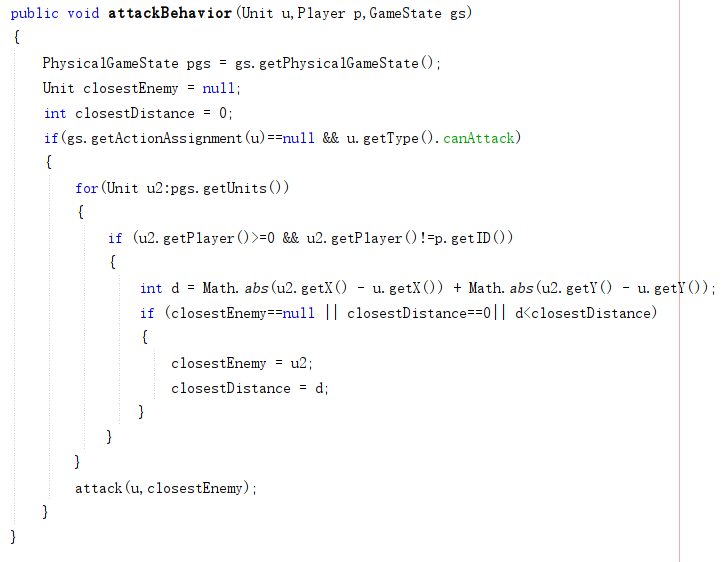
进攻就不存在堵着家门口的情形，所以总的来说有两个方面的因素要考虑，一是对敌方的什么单位进行攻击，二是派遣多少兵力发起进攻。

但转念一想，实际上对什么单位进行进攻意义不是很大，而派遣部分兵力进行进攻则必然有部分兵力滞留（实际上很少），所以两个策略最终选择合二为一，写作一个函数：

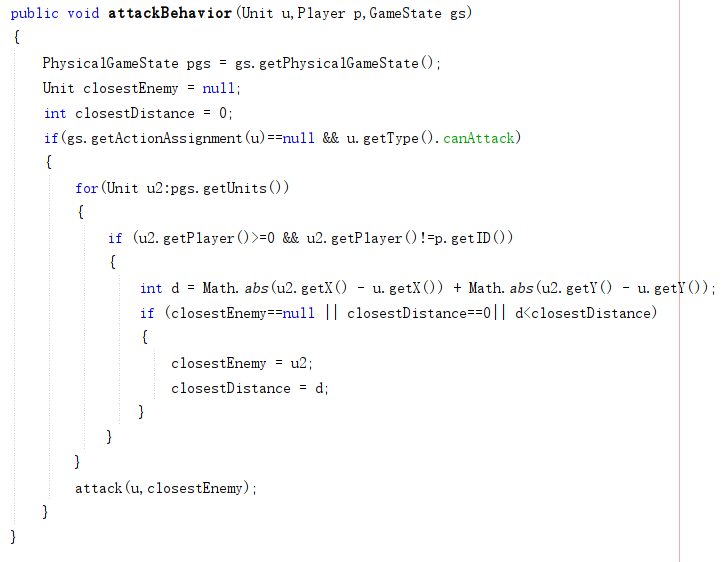


其中进攻和防御的代码分别封装为一个函数在上述函数中进行调用，如下：

进攻函数：

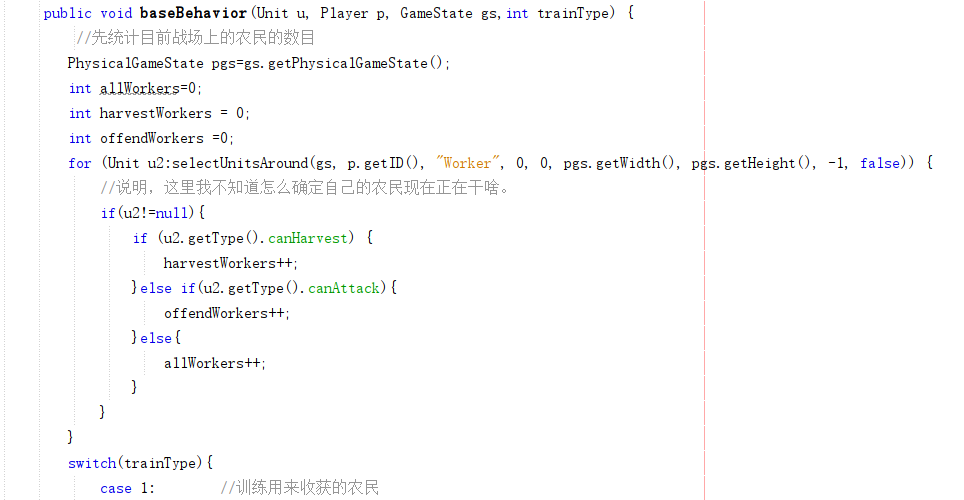


防御函数：



而要进行不同兵种的不同进攻方式，我们则必须要进行兵营的建造，但在原有的代码上进行编写后发现整体结构太过于混乱，并且问题频出难以解决，于是在参考了HeavyRush，LightRush后决定仿照其结构对基地、兵营、工人三者的行为各封装为一个函数为baseBehavior，barrackBehavior和workersBehavior，并进行了相应的改动。

部分截图：







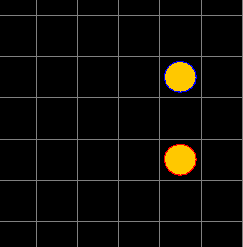
以上就是所有的基础操作，但若只按部就班执行以上命令则显得太过于单调且迟钝。

实际上我们对于这个方面进行过探讨，因为这个游戏虽然是个即时战略游戏，但是在这个框架下：

首先地图的大小限制了许多策略，节奏很快，在相同进攻策略下慢一步可能就会失败，兵营的建造最多只能有一个，没有不行，多了费时费资源。而基地若是被毁，重建基地的机制貌似实现不了，而且即使重建也是输。

其次兵种的属性不太平衡，兵营太脆弱导致worker也能快速破坏，而其他三个兵种之间虽有相互的克制作用，但总体看来Light还是最为强势，而heavy显得又是那么笨重。

第三点是在对战中发现的，我们的进攻都是调用已有的attack函数，而有时或者说经常会出现两个兵对峙（如下图），或是相互上下走位直到第三个士兵到场才有可能打破僵局（或者四人转，六人转）。猜测可能是框架中的寻路算法存在不足。



综上不足，以及对github上前几名的代码进行了观摩，第一名似乎有些复杂，而第二名采用的就是根据地图选择之前存在的所有策略，我们成为“人工智慧”。我们一致认为“人工智慧”确实可能已经可以决策到此框架下的游戏的90%的胜利方式，但是目前我们决定还是采用学习的方式让AI进行自我决策。

而决策的前提则是需要对目前战场形式进行评估，而评估的依据则是将战场的情形进行量化，这也是第一次报告，即三周前所作出的设想，但是行为树应该是用不到的。而量化过程中需要注意的因素很多，我们商讨后决定先考虑：敌我双方兵种组成，敌我兵种与对方基地距离。

（经过试验，敌方采取策略如进攻或防守，敌方主要兵种等因素都可能会造成巨大影响，但是可能性很小，所以先不做考虑）

而量化的过程很关键，错误的量化方式只会得出错误的状态和错误的局势判断，我们给各个兵种间进行了比较，再结合敌我双方兵种组成计算各兵种数量差，再由距离影响比例因子综合后得到最终的数值：

/\*

分析战场形势的函数：从两个方面进行评估（1 敌我双方各个单位数目的差值（一定考虑） 2 敌我双方单位和对方基地的距离的插值（可选择考虑的程度）暂时不考虑兵营

output: 一个 int 范围从 0 - 4 0:我方大劣势 1：我方劣势 2：双方均等 3：我方优势 4 我方大优势

input: gs同之前 ,

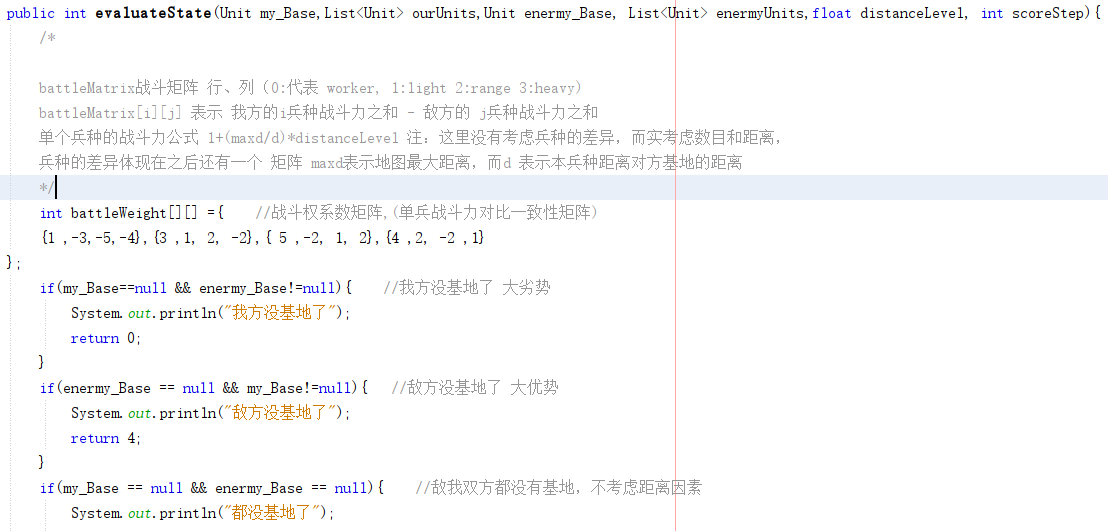
ourUnits(我方单位）和enermyUnits(敌方单位）是评估的对象，

myBase 我方基地 enermyBase 敌方基地

distanceLevel ： 敌我双方距离的影响因子，即是否将敌我双方距离对方基地的距离也纳入评估的范围 范围从 0 到 1 0是不考虑距离，1 考虑距离且程度最大

scoreStep : 评价阈值的步长

\*/



通过这些参数量化从而得到一个数值，再通过阈值判断形成5种情势：我方大劣势，我方劣势，局势均衡，敌方劣势，敌方大劣势。

再统计敌方主要兵力（共十个情形），最终得到50个状态，而学习的目的即保证我方远离大劣势，靠近大优势。

如何让AI学习，我们学习了Q-Learning，以下为我们的学习进展及打算。

**二 应用强化学习 Q-Learning算法进行决策**

（1）决策条件和目标

由我们的AI基础，目前我们已经能够通过 evaluateState函数和 evaluateEnermyMajorUnit 函数评估目前战场的情况，并根据这两个函数的返回结果将战场形势量化成有限（50个左右）state。我们AI有 baseBehavior 、barracksBehavior、workersBehavior、rushTactics等函数，通过控制这些函数的参数，将AI的行为进行编码，获得有限个Action,其中每个action是一个int对应着一串参数，这一串参数是上述函数的输入。而不同的输入对应着AI采取不同的策略（比如说让兵营生产 light，组织进攻等等）通过上述的工作，我们将代表战场形势的state和代表AI具体行动的action进行了量化，而AI决策的目标就是在特定的state下选择最优的action。

（2） Q-learning概述

Q-learning是强化学习领域一个经典的算法。其算法的内容大致为：

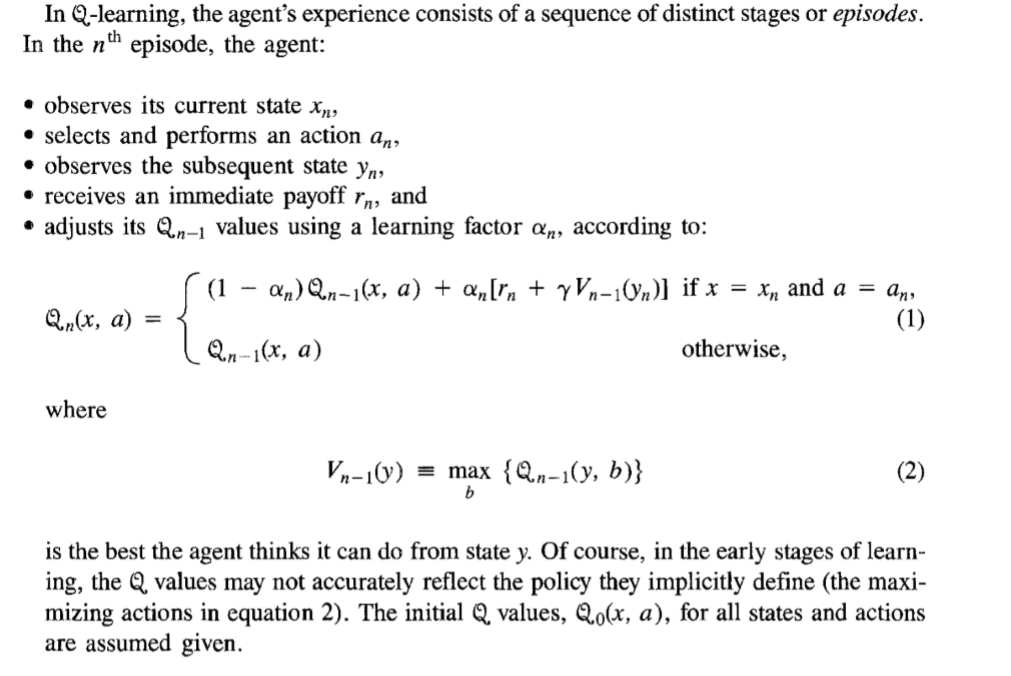
1 以action为行，以state为列，构造一个reward矩阵，reward[i][j]代表在j 这个state下，采用i这种action获得的奖励。Reward矩阵的元素需要认为给定，一种reward矩阵就代表着一种策略。

2 Q-table,同样的以action为行，以state为列，构造一个Q矩阵，Q[i][j]代表在j这个state下，采用i这种action最终能获得的最大reward值的期望。Q-table就相当与AI的大脑，训练的最终目标就是填满Q-table并让其收敛。

3 Q-learning算法的决策过程

获取当前的state ，在Q-table中，选择 i 使得Q-table表格中第 列Q[i][]最大，AI采用行动i, 换言之通过Q-table AI选择了在特定 state下的最佳的动作。随着游戏的进行，AI不断通过上述方式进行决策。

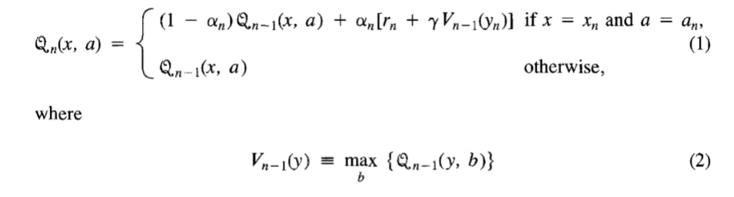
4 Q-learning算法的学习过程



在第n次 episode中：

1首先获取当前的state ，然后通过3中的决策过程选择一个action ,（如果处于学习初期可以采用生成随机数的方法，随机选择一个action）

2决策之后，获取决策后的state , 根据reward矩阵以及在state 下选择的action 获取我们上一步决策的奖励 。然后根据 Bellman方程更新 Q[,公式为：



其中 表示学习速率，值越高，越快放弃之前学习到的内容，γ系数表示对未来情况的重视程度，其值越高对未来越重视。

3经过不断的迭代，Q-table最终能够收敛，我们也就训练好了AI

（证明：Watkins-Dayan1992\_Article\_Q-learning）

（3）应用Q-learning

由（2）Q-learning的概述，以及（1）中我们对战场形势和决策的量化，我们发现我们能够应用Q-learning满足我们的决策需求，具体的方法是：

1将我们量化的表示战场形势的state作为 Q-learning算法的state的，将动作编码action 作为Q-learning算法的action。

2认为根据经验，以及之前的观看和microRts提供的自带AI（lightRush,HeavyRush,lightDefence,HeavyDefence等等）和自己人工决策AI的对战情况，手动地填写reward表格。

3 根据游戏进行的时间，每隔一段时间（帧数）n（n作为参数可调），进行一次决策以及训练更新Q-table，这样我们预想通过不断地和AI对战，不断地更新Q-table我们不断训练自己的AI，最终让AI做出更加科学的决策（未实现）。

4由于一种reward代表着一种策略，根据不同的地图大小，不同的出生点位置，我们可以给出不同的reward矩阵，并且巡礼不同的Q-table

（4）可能存在的问题以及缺陷

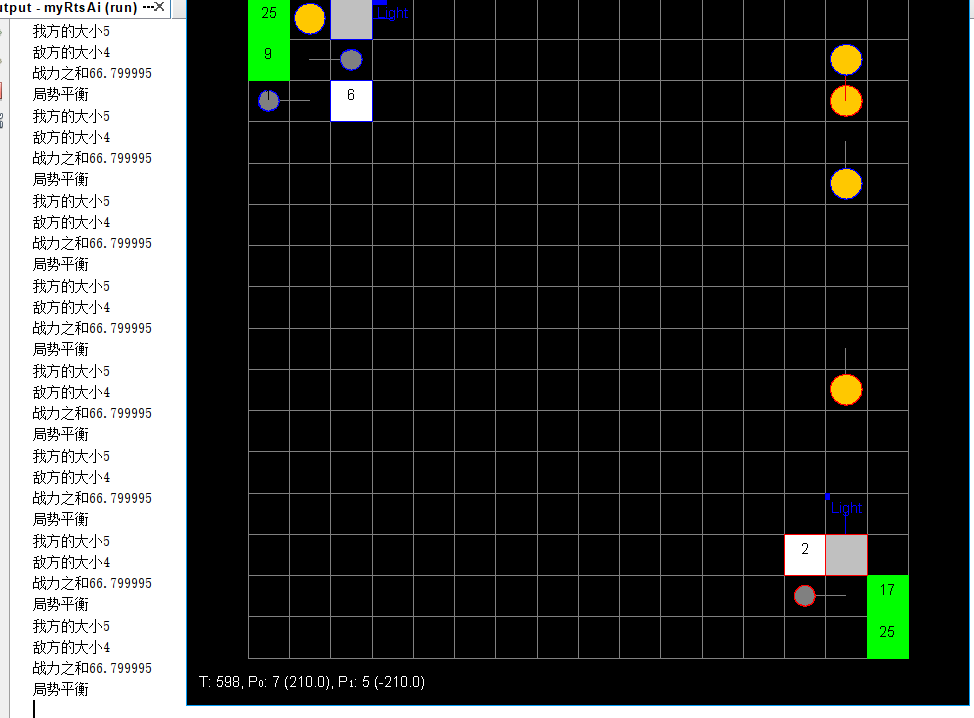
1首先通过研究Q-learning算法，我们发现，应用Q-learning算法进行决策能够达到较好的效果，取决于两个主要的条件，reward矩阵，和state action状态的划分

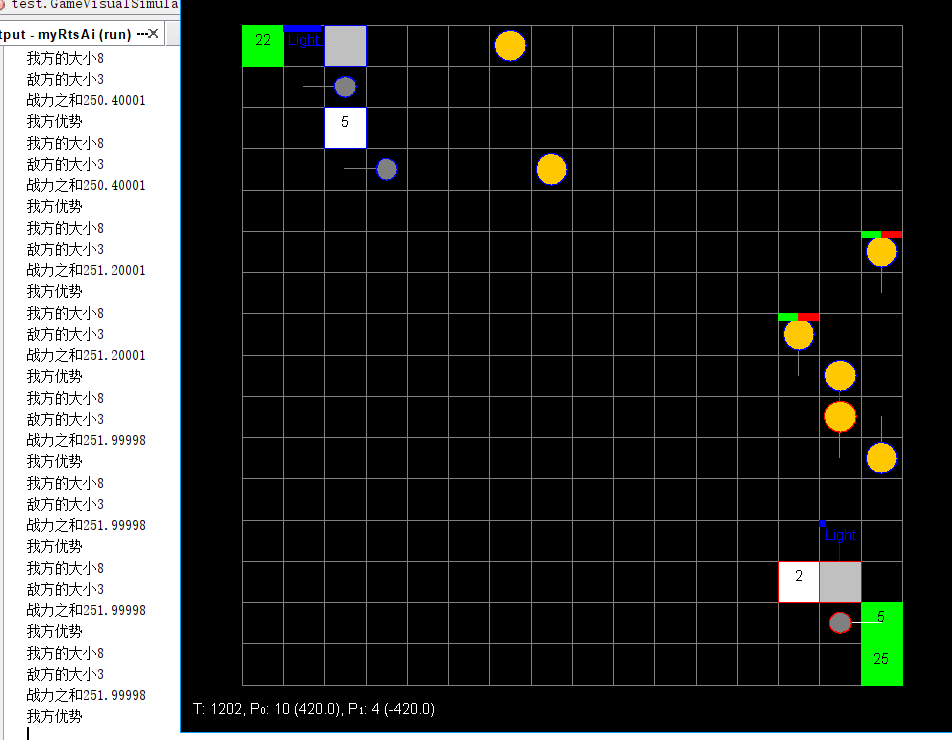
2应用到rtsai上来，首先战场的形势与其它游戏相比较显得更为复杂，很多时候对战场的判断并不能特别的全面、准确，这就影响了Q-learning算法的有效性，其次rts游戏的动作更加复杂多样化，存在很多复杂的战略战术，而Q-learning适用于有限的action和state，因此采用Q-learning进行RtsAI的决策显然是有一定的局限性的。但是考虑到microRts的兵种简单，游戏节奏较快，战场形势比较简单，操作也比较单一,还是能够比较清楚地通过某种标准评估战场形势以及采用有限种策略，因此我们认为Q-learning还是能够满足microRtsAI的决策需求的，当然这只是我们根据游戏的特点以及Q-learning算法本身的特点做出的猜想，还未通过实验进行验证（这是我们下一步的工作）

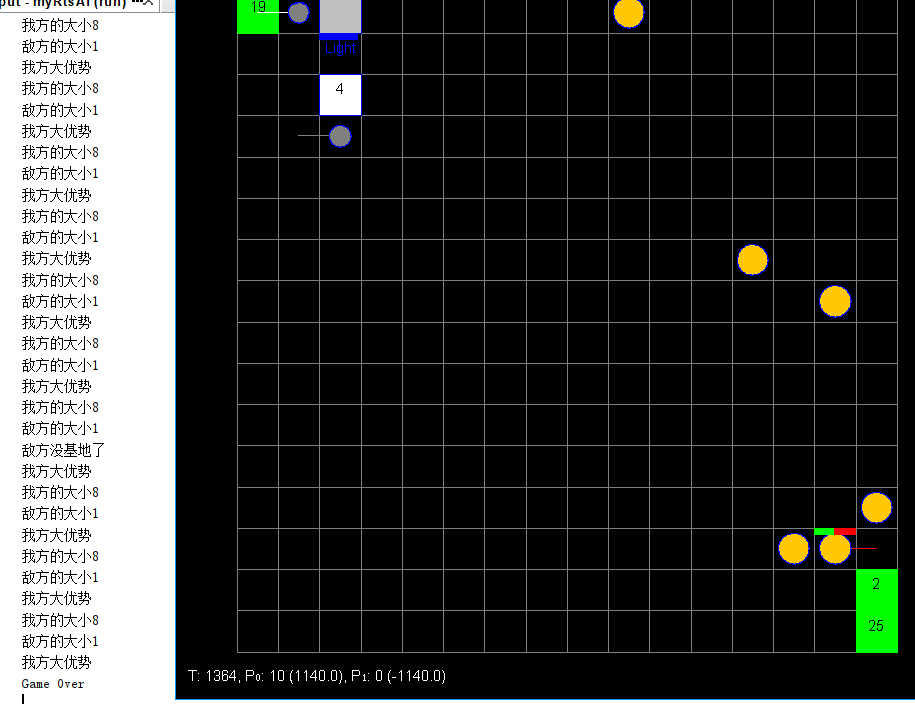
3 由（2）中的描述,一种reward矩阵代表着一种策略，但是reward矩阵是需要我们自己根据之前的对战结果人为给定的，而且reward矩阵极大影响Q-learning的学习结果，而通过人为获得一个比较合理的Q-learning矩阵是比较困难的。这也是采用Q-learning算法的一个主要的而且是比较大的局限。

**Part 2 当前进展**

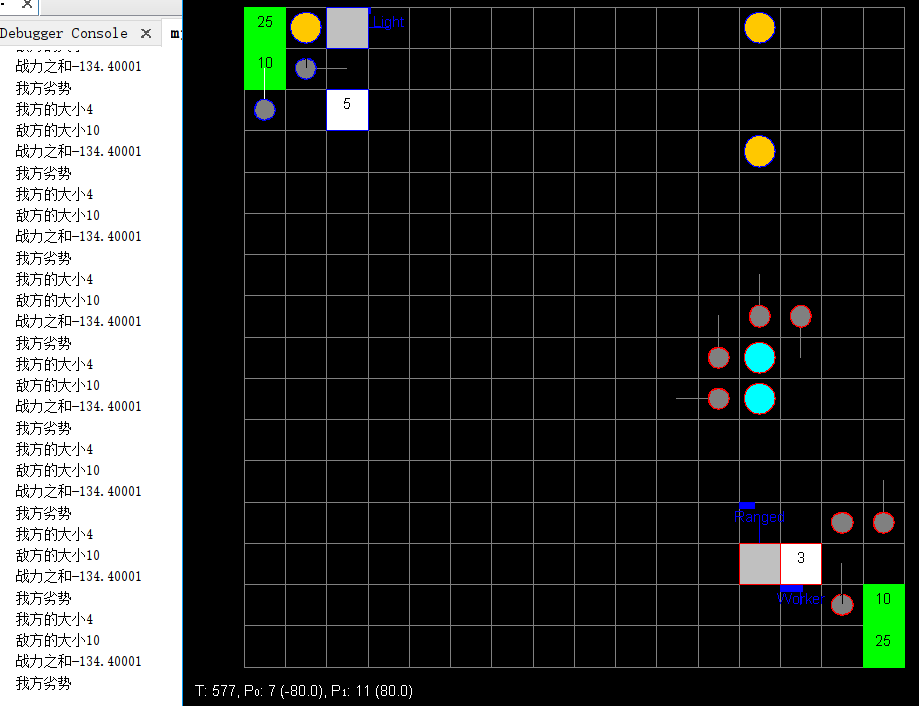
对阵LightRush



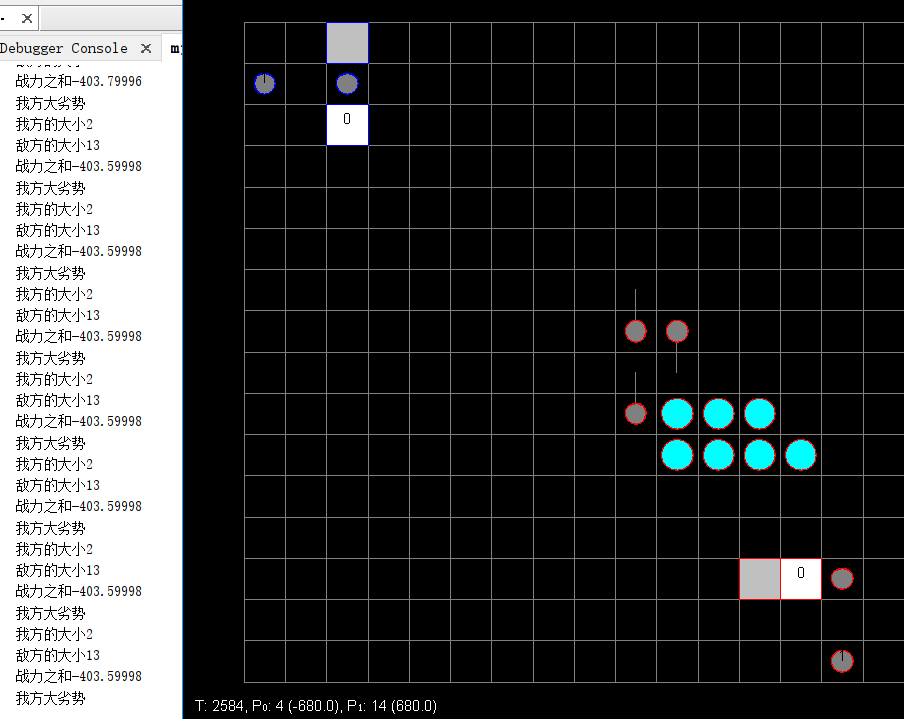
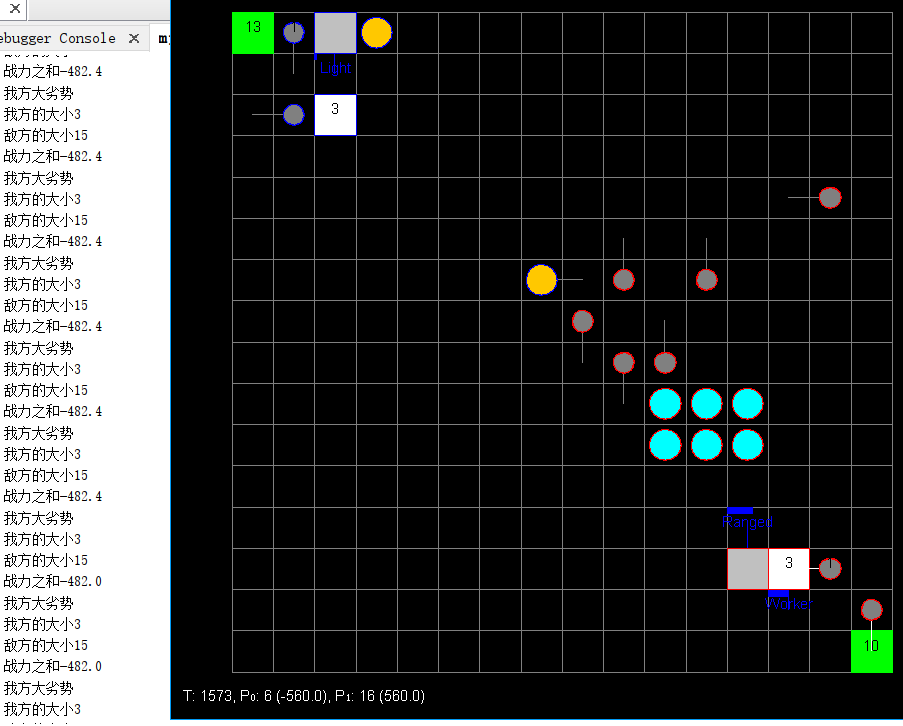




对阵CRush\_V2







以上为两种情形，目前我们的AI在我们做决策下能够打败几乎所有的自带AI，局势的判断较为正确，但是自我决策还未完善，以及细枝末节上仍有许多小bug，这方面也是我们接下来几周努力的方向。