

人工智能 Java 坦克机器人系列

强化学习



Robo cool
2006 年 7 月 13 日发布

0

系列内容：

+ 此内容是该系列的一部分： 人工智能 **Java** 坦克机器人系列

Robocode 是 IBM 开发的 Java 战斗机器人平台，游戏者可以在平台上设计一个 Java 坦克。每个坦克有个从战场上收集信息的感应器，并且它们还有一个执行动作的传动器。其规则和原理类似于现实中的坦克战斗。其融合了机器学习、物理、数学等知识，是研究人工智能的很好工具。

在 Robocode 坦克程序中，很多爱好者喜欢设计一些策略与移动模式，让自己的坦克机器人能更好的赢得战斗。但是由于 Robocode 环境时刻在变化，手写的代码只能对已知的环境做一些预测，机器人不能根据环境的变化而自我学习和改善。本文中，将用强化学习实现一个机器人。使用强化学习能创建一个自适应的战斗机器人。这个机器人能在战斗中根据环境取得最好的策略，并尽力使战斗行为最佳。并在此过程中不断学习以完善自身不足。

强化学习

强化学习(reinforcement learning)是人工智能中策略学习的一种，是一种重要的机器学习方法，又称再励学习、评价学习. 是从动物学习、参数扰动自适应控制等理论发展而来。

强化学习一词来自于行为心理学，这一理论把行为学习看成是反复试验的过程，从而把动态环境状态映射成相应的动作。它通过不断尝试错误，从环境中得到奖惩的方法来自学习到不同状态下哪些动作具有最大的价值，从而发现或逼近能够得到最大奖励的策略。它类似于传统经验中的“吃一堑长一智”。

原理与模型

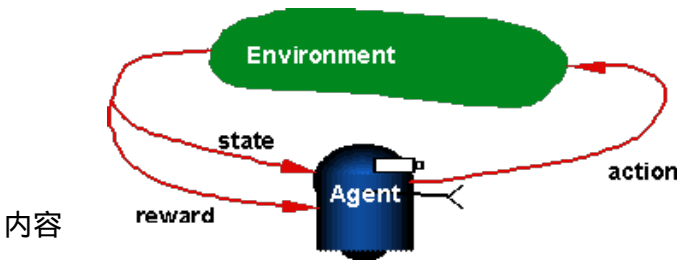
考虑建造一个可学习的机器人，该机器人(或agent)有一些传感器可以观察其环境的状态(state)并能做出一组动作(action)来适应这些状态。比如：一个移动的机器人有摄像头等传感器来感知状态，并可以做"前进", "后退"等动作。学习的任务是获得一个控制策略(policy)，以选择能达到的目的的行为。

强化学习基本原理也是基于上面的思想：如果 Agent 的某个行为策略导致环境正的奖赏(强化信号)，那么 Agent 以后产生这个行为策略的趋势便会加强。Agent 的目标可被定义为一个奖赏或回报函数(reward)，它对 Agent 从不同状态中选取的不同动作赋予一个数字值，即立即支付(immediate payoff)。比如机器人寻找箱子中的回报函数：对能找到的状态-动作赋予正回报，对其他状态动作赋予零或负回报。机器人的任务执行一系列动作，观察结果，再学习控制策略，我们希望的策略是在任何初始离散状态中选择动作，使 Agent 随时间累积中发现最优策略以使期望的折扣奖赏(回报)和最大。

如图描述：Agent选择一个动作(action)用于环境，环境(Enviironment)接受该动作后状态(state)发生变化，同时产生一个强化信号(奖赏reward)反馈给Agent，Agent根据强化信号和环境当前状态再选择下一个动作，选择的原则是使受到正强化(奖)的概率增

大。

强化学习的基本模型



概览

Q 学习(Q-learning)

强化学习

Robocode概述
增强学习要解决的问题：一个能够感知环境的自治 Agent，怎样通过学习选择达到其目标的最优动作。这样一个 Agent 在任意的环境中如何学到最优策略是我们重点考虑的对象，下面介绍的称为 Q 学习的算法，就是其中比较好的一种强化学习算法，它可从有延迟的回报中获取最优控制策略。

动作集 (Action)
Q 学习是强化学习的一种形式，机器人在任意的环境中直接学习最优策略很难，因为训练数据中没有提供<s,a>形式的训练样例。数据中通常学习一个定义在状态和动作上的数值评估函数，然后以此评估函数的形式实现最优策略将会使过程变得容易。

奖励的确定 (Reward)
我们在Q 学习中把 Q 表示在状态 s 进行t动作的预期值；s 是状态向量；a 是动作向量；r 是获得的立即回报； γ 为折算因子。则估计函数 $Q(s,a)$ 被定义为：它的值是从状态 s 开始并使用 a 作为第一个动作时可获得的最大期望折算积累回报。也就是说 Q 值是从状态 s 执行动作 a 的立即回报加上遵循最稳定最优策略的值(用 折算)。公式如下：
测试与结果

AI-CODE

$$Q(s,a)=r(s,a)+$$

下载资源

相关主题

$$\max_a Q(s'+a')$$

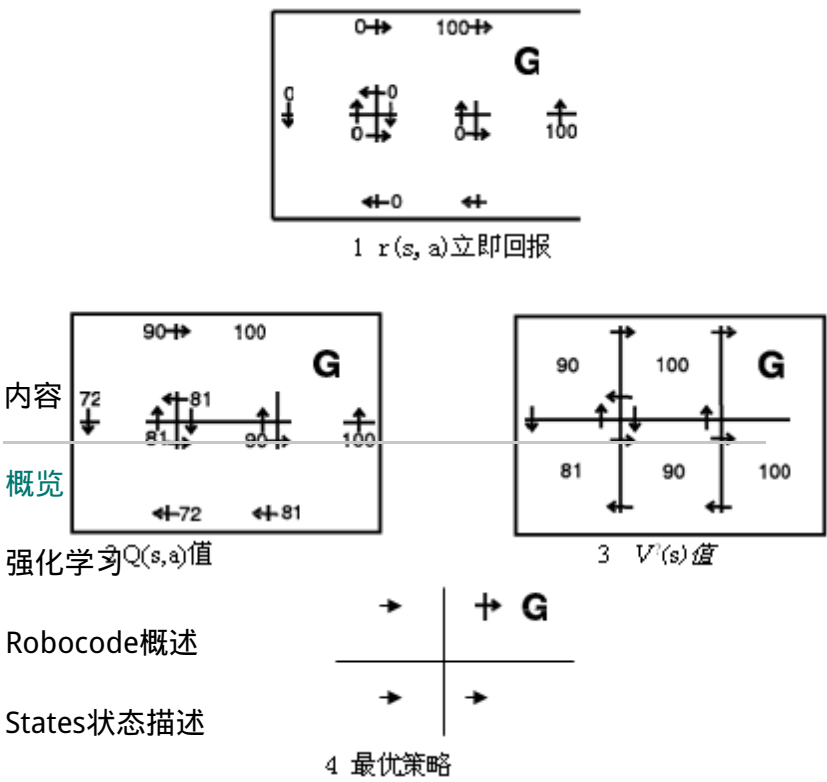
评论

我们用过程来表述 Q 学习算法如下：

- 1.对每个 s,a 初始化表项 Q(s,a)
- 2.观察当前状态 s，一直重复做：
 - a.选择一个动作 a 并执行它
 - b.接收到立即回报 r
 - c.观察新状态 s'
 - d.对 Q(s,a) 按照下式更新表项 $Q(s,a) \leftarrow r(s,a) + \gamma \max_{a'} Q(s'+a')$
 - e. $s \leftarrow s'$

其中 $\gamma(0 \leq \gamma < 1)$ 是折算因子，为一常量。

为了说明这些概念，我们用一些简单的格状确定世界来模拟环境。在这个环境下所有的动作转换除了导向状态G外，都被定义为 0，agent一进入状态G，可选动作只能处在该状态中。图中方格表示agent的6种可能状态或位置，每个箭头代表每个不同的动作。如果agent执行相应状态动作可收到立即回报r(s,a)， $V^*(s)$ 为最优策略的值函数，即从最初状态s到获得的折算积累回报。此处立即回报函数把进入目标状态G的回报赋予100，其他为 0， $V^*(s)$ 和Q(s,a)值来源于r(s,a),以及折算因子 $\gamma=0.9$ 。



定义了状态、动作、立即回报和折算率，我们根据计算就能得出最优策略图4和它的值函数 $V(s)$.该策略把agent以最短路径导向状态G。图3显示了每个状态的 $V(s)$ 值，例如，图3中下方的状态中最优策略使agent向右移动，得到为0的立即回报，然后向上，生成100的立即回报，，此状态的折算回报计算为：

具体学习过程 $0.9 \times 0 + 0.1 \times 100 = 9$ ，即中上方的 $V(s)$ 值为90

测试与结果 Q 学习的优点是即使在学习不具有其动作怎样影响环境的先验知识情况下，此算法仍可应用.

AI-CODE

下载资源
强化学习的应用
相关主题

评论 强化学习主要应用在三个方面：在机器人中的应用， 强化学习最适合、也是应用最多的。 Wnfriedllg采用强化学习来使六足昆虫机器人学会六条腿的协调动作。 Sebastian Thurn采用神经网络结合强化学习方式使机器人通过学习能够到达室内环境中的目标; 在游戏比赛中的应用，在这方面，最早的应用例子是Samuel的下棋程序; 在控制系统中的应用，强化学习在控制中的应用的典型实例，就是倒摆控制系统. 当倒摆保持平衡时，得到奖励，倒摆失败时，得到惩罚，控制器通过自身的学习，最终得到最优的控制动作。

Robocode概述

从上面的强化学习原理中我们知道，要实现强化学习我们必须知道状态、动作以及处理这些状态和动作的Q函数，在对状态和动作的反复实验当中，我们还要给出动作的奖赏、设定学习率、折算率等参数。最后我们还要利用Q 学习算法把上面提到的参数组合进行最优化，最终得到自己想要的值。下面我们就从robocode来分析上面提到的参数和强化学习的实现过程。

States状态描述

Robocode是根据战斗环境模拟而来，所以在此环境中存在很多种状态，不同的状态对机器人会产生不同的影响。这些状态我们都可通过Robocode的函数调用得到，如下表列出了Robocode中能得到的部分状态值。

内容	函数名
	getBattleFieldHeight() getBattleFieldWidth() getEnergy() getHeight() getName() getX() getY() getBearing() getBearingRadians() getDistance() getEnergy() getName() getGunCoolingRate() onHitWall()
概览	

强化学习如此多的状态如果全部放到强化学习中，会耗费强化学习很多时间。而且像getBattleFieldHeight()、getBattleFieldWidth()得到场地高和宽，getGunCoolingRate()炮管冷却率，这些状态值与学习无关联，而有些状态是与不同的动作相结合的，如果使用不当，甚至会达不到应有的效果。所以如何确定和选择状态是很关键的问题。根据战斗经验与测试数据，我们把Robocode机器人的状态分为五个属性，并以类state来封装所有的这五个属性。最后我们还给出了在我们的强化学习中没有应用到但同样重要的一些状态。大家有兴趣可补充进自己的强化学习算法当中。

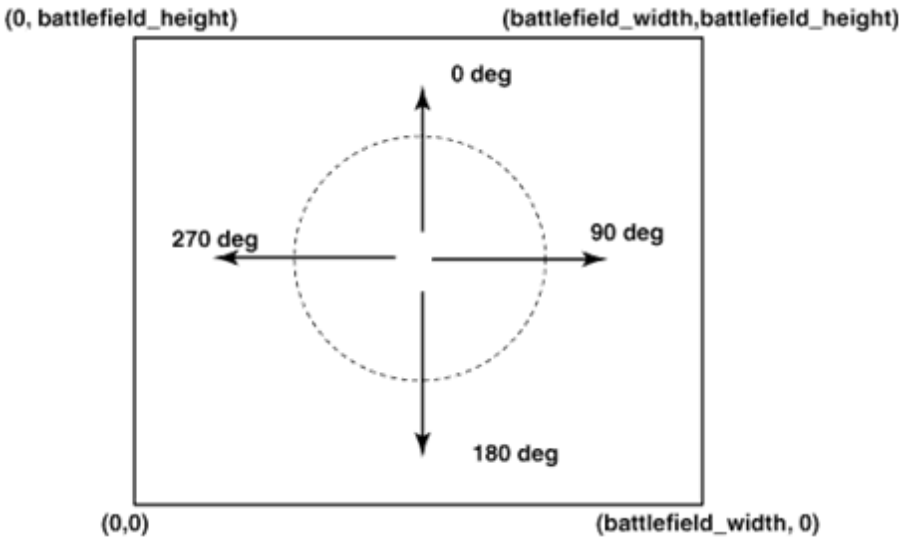
数据与Q表

状态离散化

具体学习方案
强化学习应用到Robocode遇到的最大挑战是：强化学习适合离散空间求解，而Robocode的环境却是连续的。如上表中Robocode的状态中，机器人的方向角（二维矢量），两个机器人的相对角（二维矢量），两个机器人的距离（二维矢量），场地坐标，机器人的x,y坐标等等，输出是一组动作序列，这些都是连续量，若对所有变量进行离散化必然带来维数灾难。所以，我们必须离散化输入状态。以适合强化学习算法的应用。注意Robocode中的角度、距离等状态都有一些特点：角度是分四个方向资源而且是0到360度之间的四个区间值，而两两机器人之间的安全距离一般是维持在30像素左右。根据Robocode这些特点和强化学习原理，我们分别以4，30等区间值来转换状态值为离散点。

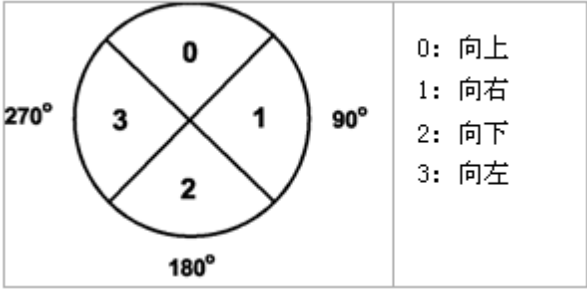
1. 机器人的绝对方向(Heading)

知已知彼，百战不怠，在开始战斗前知道敌人和自己的方向很重要。在Robocode中要想得到敌人的方向，我们首先要知道Robocode的坐标系统。Robocode坐标系统是一个标准的笛卡尔坐标，战场地图的左下角坐标为（0，0），右下角为（地图宽，0），左上角为（0，地图高），右上角为（地图宽，地图高），如下图所示：



从图上我们知道了如果机器人处于场地中央则面向场地水平向上为0度方向，按顺时针转动，水平向右为90度方向，水平向左为270度方向，水平向下为360度方向。也就是说0 <= heading < 360。此角度值是一个绝对值。在Robocode中我们通过调用getHeight()函数能得到想要的机器人当前方向。机器人处于场地什么位置，就可对照下图得到其对应的角度。

为了保存角度到Q表中，我们把得到的连续值转换为离散的值。按照角的大小和方位我们把heading分成范围为0-3的四份，如下图所示：



在代码我们直接用360除以4得到不同的离散方向值。

内容

1 public static final int NumHeading = 4;
2 ...
3 double angle = 360 / NumHeading;

强化学习

Robocode概述

然后直接通过下面的表达式求得新的方向角

States状态描述

1 double newHeading = heading + angle / 2;
2 return (int)(newHeading / angle);

数据与Q表

奖励的确定的相对角(Bearing)

具体学习方案

上面我们知道了机器人的绝对度，但是战斗都是存在于两个机器人之间，所以这里我们要了解Robocode的第二个状态值相对角，顾名思义，它就是某一机器人相对另一机器人的角度，是针对两个机器人而言的。如下图所示：

测试与结果

AI-CODE

下载资源

相关主题

评论

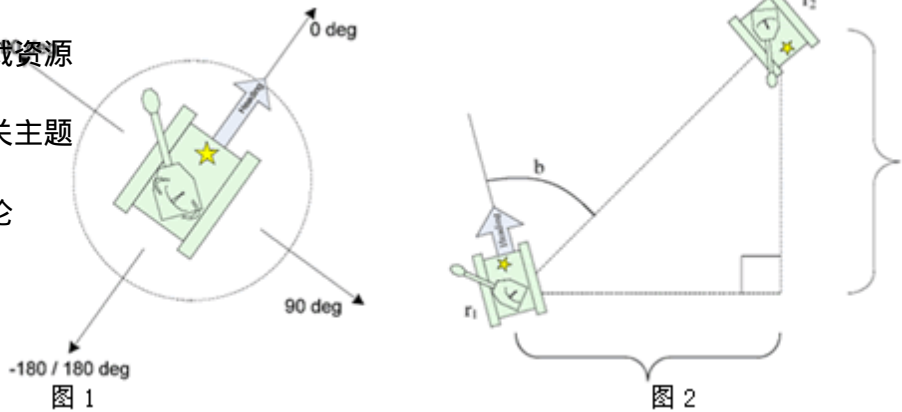


图1显示了机器人相对于自己的bearing角度。而图2显示了机器人r2相对于r1的b = 60度。由此可知，-180 < bearing <= 180。如图相对度在机器人左边为正，右边为负。在Robocode中我们可以通过函数getBearing()得到相对角。相对角是个很有用的函数，通过确定敌人相对度，我们能得到最佳的移动位置。

和上面同样的道理我们把bearing分成0-3的四份，在代码我们直接用PI值除以4得到不同的离散相对角值

1 public static final int NumTargetBearing = 4;
2 ...
3 double angle = PIx2 / NumTargetBearing;

然后直接通过下面的表达式求得新的相对角

1 double newBearing = bearing + angle / 2;
2 return (int)(newBearing / angle);

3. 目标距离(distance)

distance即两个机器人之间长度，即自己机器人中心点到敌人机器人中心点连线长。距离是以像素(Pixels)点为单位，如何确定其离散值的大小以方便保存到Q表中有一定的难度。在此我们根据机器人本身的大小(30为安全距离)以及场地大小设置，我们把距离分为0-19的20个离散值。每个离散值分为30的倍数，如下表：

距离 d	
连续值	离散点
d < 30	0
30<=d < 60	1
60 <= d < 90	2
...
540 <=d < 570	4
570>= d	19

内容

在代码我们直接用距离值除以30得到不同的离散值

概览

强化学习

```
1 public static final int NumTargetDistance = 20;
2 ...
Robocode概述distance = (int)(value / 30.0);
```

States状态描述

然后直接通过下面的表达式求得新的距离

动作集 (Action)

数据与Q表

```
1 if (distance > NumTargetDistance - 1)
2     distance = NumTargetDistance - 1;
奖励的确定 (Reward)let reward = distance;
```

具体学习方案

测试与结果

AI-CP当机器人撞击墙时，能量会发生改变，这时"Hit wall"事件会触发，我们就此状态定义为"Hit wall"，而机器人根据这个事件能做相应的动作。撞墙状态比较简单，只有撞也没有撞，所以我们在此用两个离散值来表示这一状态，0表示没有撞墙，状态没有发生。1表示撞墙状态发生。通过Robocode的ohHitWall事件我们能得到这个状态。

相关主题

5.Hit by Bullet （子弹相撞）

评论

当机器人被子弹击中时，能量会发生改变，这时"Hit by Bullet"事件会触发，我们就此状态定义为"Hit by Bullet"，而机器人根据这个事件也能做出相应的动作。同样的我们在此用两个离散值来表示这一状态，0表示没有撞墙，状态没有发生。1表示撞墙状态发生。通过onHitByBullet()事件我们能得到这个状态。

因为Hit wall和Hit by bullet本身都是Robocode的事件函数，所以在直接把离散值写入Robocode本身的事件当中。

根据上面五个状态的组合，我们可得到共1280 (4 x 20 x 4 x 2 x 2) 可能存在的状态。在robocode中还有其他一些状态：比如两个机器人相撞的HitRobotEvent事件状态，团队中的消息接收MessageEvent事件状态，还有机器人的x,y坐标状态，机器人在战地中间的坐标和战地边缘的坐标都会对状态产生影响。如果有兴趣大家可以试一试别的状态处理。

动作集（Action）

Robocode的动作相对是比较复杂的，而且涉及到炮管、雷达和机器人本身的移动。为了简化操作，在此处我们只定义了机器人自身的移动动作集：移动和转动。在Robocode中最基本的移动就是前进和后退，转动就是向左或向右。同状态一样，我们把Action中的值进化离散化以保存到Q表中。如下代码：

```
1 public static final int RobotAhead = 0;
2 public static final int RobotBack = 1;
3 public static final int RobotAheadTurnLeft = 2;
4 public static final int RobotAheadTurnRight= 3;
5 public static final int RobotBackTurnLeft = 4;
6 public static final int RobotBackTurnRight= 5;
```

动作选择

我们知道了状态、动作，就要面临如何根据状态来选择机器人的动作。选择的动作不仅影响立即强化值，而且影响环境下一时刻的状态及最终的强化值。所以好的选择方法很重要。在强化学习中通常使用概率来选择动作，对于状态S，做不同Q值的动作时赋予不同的概率，高值得到高概率，低值得到低概率，所有动作的概率都非0。如下方法：

1

|

$$P(a | s) = e^{Q(s, a)} / \sum (e^{Q(s, a_i)})$$

其中P (a | s)为机器人在状态s时选择的动作a的概率，e为一常量大于0。在代码中我们通过getQValue得到当前Q(s,a)值，并利用内容上面的公式在所有的Q值中选择出最优的动作。如下代码，其中ExploitationRate设定为1.

概览

1 for (int i = 0; i < value.length; i++)

2 {

3 qValue = table.getQValue(state, i);

4 value[i] = Math.exp(ExploitationRate * qValue);

5 sum += value[i];

6 }

Robocode概述

States状态描述

动作集 (Action)

数据与Q表

奖赏的确定 (Reward)

上面我们定义了状态/动作值，而且从强化学习概念我们知道状态/动作值都是以对的方式存在。我们把这种值叫Q(s,a)值。学习都是在前面一个Q值的基础上对新的值进行判断和完善。所以在强化学习中定义了Q表用以保存所有收集的Q值。由于状态和动作是成对存在，二维数组是保存Q值的最佳工具，如下代码。

AI-CODE

1 private double[][] table;

2 table = new double[State.NumStates][Action.NumRobotActions];

下载资源

由代码可知表值大小决定于状态和动作的数量，这两者数量越多，表越大。Robocode是以回合制的方式进行战斗，要想在每个回合都能利用到原始Q值，文件是最好的通讯工具，通过java文件流我们定义输入 (input) 和输出(output)函数来保存数据到文本文件。这样在每个回合中，我们都能对原始数据进行分析处理，并不断把新的数据写入表中。

Q表中在强化学习中是很重要的概念，它不仅保存了所有Q值，同时也定义了对这些Q值进行操作的方法。通过调用这些方法，我们能直接得到最大的状态值及最优化的动作。

1 public double getMaxQValue(int state)

2 {

3 double maxinum = Double.NEGATIVE_INFINITY;

4 for (int i = 0; i < table[state].length; i++)

5 {

6 if (table[state][i] > maxinum)

7 maxinum = table[state][i];

8 }

9 return maxinum;

10 }

遍历所有状态值，从中找出最大化的状态。

1 public int getBestAction(int state)

2 {

3 double maxinum = Double.NEGATIVE_INFINITY;

4 int bestAction = 0;

5 for (int i = 0; i < table[state].length; i++)

6 {

7 double qValue = table[state][i];

8 if (table[state][i] > maxinum) {

9 maxinum = table[state][i];

10 bestAction = i;

11 }

12 }

13 return bestAction;

14 }

遍历所有状态值，从中找出最大化的状态。根据状态值得到当时最佳动作。

奖赏的确定（Reward）

经过上述状态和动作离散化，机器人的移动的学习问题已经转化为一个离散的强化学习问题，现在我们只要选择Q 学习（Q-Learning）算法，直接以Q值作为状态-动作对的评价值，进行Q(s,a)的强化学习。在开始之前，我们还需要设计一套奖赏规则。通过观察，在上面动作和状态发生改变时，特别是机器人本身的状态发生改变时，机器人的能量都会或增或减。

Robocode中一场战斗开始，每一个机器人都能得到100的能量，当在不同的状态下，如撞墙，撞到机器人，打中敌人和被敌人打中时，机器人的能量都会发生改变，而且不同的状态都有不同的能量转换规则：

1.发射子弹能量大小:我们的机器人在开始时能以不同的能量发射子弹，子弹能量在0.1到3之间。通过getPower()函数我们能得到我们的子弹能量。

2.当机器人撞墙时:能量损伤度=Math.abs(velocity) * 0.5 -1，此处的velocity即撞墙时机器人的速度

3.当机器人被敌人子弹打中时:能量损伤度= 4 * power，如果敌人子弹能量大于1，则能量损伤度 += 2 * (power-1)

4.我们每发射一颗子弹我们的生命能量就会减1

5.有失必有得，如果我们的子弹打中别的机器人，我们可以从子弹那获得3*power的能量 在robocode中我们能通过函数getEnergy()得到自身的能量值。由于能量的这种随动作和状态改变的特殊性，我们就以它来定义奖赏，根据上面的规则，实现如下：

当机器人撞墙时，奖赏为负能量(Math.abs(getVelocity()) * 0.5 - 1)，如下代码：

```
public void onHitWall(HitWallEvent e)
{
    double change = -(Math.abs(getVelocity()) * 0.5 - 1);
    reinforcement += change;
    isHitWall = 1;
}
```

2.当机器人相撞时，奖赏为负能量6，如下代码：

```
public void onHitRobot(HitRobotEvent e)
{
    double change = -6.0;
    reinforcement += change;
}
```

3.当机器人被子弹打中时，奖赏减少能量(4 * power + 2 * (power - 1))

```
public void onHitByBullet(HitByBulletEvent e)
{
    double power = e.getBullet().getPower();
    double change = -(4 * power + 2 * (power - 1));
    reinforcement += change;
}
```

4.当机器人打中敌人时，奖赏增加e.getBullet().getPower() * 3;


```
1 public void onBulletHit(BulletHitEvent e)
2 {
3     double change = e.getBullet().getPower() * 3;
4     reinforcement += change;
5 }
```

具体学习方案

内容

现在我们得到了状态、动作、Q值、、奖赏等Q 学习算法中所有参数，下面我们就以这些参数来实现我们的学习方案。

1. 确定环境和行动：

Robocode概述
状态集 $S = \text{Mapping}\{s_1\}$

States状态描述
S1中状态分别为机器人的heading、bearing、distance、hit wall、hit by bullet状态

动作集（Action）

2. 确定Q值, Q函数 $Q(s,a)$ 用于映射 state/action对到Q值中

3. 确定参数：在此根据经验我们选取如下参数，这些参数值可在实验中调整(hang原值折现率为0.7，学习率为0.05)：

折现率 $\gamma = 0.9$ ； $\gamma(0 \leq \gamma < 1)$ 为常量

测试与结果
学习率： $\alpha = 0.1$ ；

AI-CODE
 $Q_{t+1}(s_t, a_t)$: 根据行为" a_t " 和状态 " S_t "得到的新Q

下载资源

$R_t(s_t, a_t)$: 根据状态和行为而得到的奖赏

评论
 $\max Q(s_{t+1}, a_i)$: 根据行为" a_t " 和状态 " S_t "得到的最大Q值

3. 计算行动选择概率

4.迭代公式：

$$Q(s, a) \leftarrow -Q(s, a) + \gamma s + \gamma (\max Q(s', a') - Q(s, a))$$

5. 更新Q值

$$Q_{t+1}(s_t, a_t) = (1 - \alpha) Q_t(s_t, a_t) + [\alpha R_t(s_t, a_t) + \max Q(s_{t+1}, a_i)]$$

```
1 public static final double LearningRate = 0.05;
2 public static final double DiscountRate = 0.7;
3 ...
4 double oldQValue = table.getQValue(lastState, lastAction);
5 double newQValue = (1 - LearningRate) * oldQValue + LearningRate *
6 (reinforcement + DiscountRate * table.getMaxQValue(state));
7 table.setQValue(lastState, lastAction, newQValue);
```

Agent在经过一定的状态和执行多个动作后获得了最终奖赏，这时就会对这个状态-动作序列分配奖赏。Q 学习算法的核心就是一个状态和动作的组合都拥有一个Q值，每次获得最终回报后通过更新等式更新这个Q值。其实这是一个典型的马尔科夫决策过程（Markov decision process, MDP）。

马尔科夫决策过程（Markov decision process, MDP）：Agent可感知到其环境的不同状态集合，并且有它可执行的动作集合。在每个离散时间步t，Agent感知到当前状态st，选择当前动作at并执行它。环境响应此Agent，给出回报Rt=Q(st, at)，并产生一个后继状态St+1=a (s t, a t)。在MDP中，其中函数Q(st, at)称之为动作评估函数（价值函数），a (s t, a t)称之为状态转换函数；其中Q(st, at)和a (s t, a t)只依赖于当前状态和动作，而不依赖于以前的状态和动作。

实现步骤

- 1.随机初始化Q 值，机器人收集状态信息，转换这些状态信息为离散值
- 2.根据行动选择概率公式选择一个动作执行执行选择动作操作
- 3.由行动结果中即机器人的能量改变中得到奖赏
- 4.从环境中确定新的状态，迭代修改Q 值

内容把奖赏和最佳的Q值用于新的状态中

概览6.转2

强化学习

Robot描述

测试与结果

States状态描述

强化学习同遗传算法一样，也要对把机器人同不同对手进行训练，让其不断的自我学习。下面我们以Hang的QLearningBot机器人与不同的例子机器人进行训练，得到不同的测试结果比较。

数据与Q表

奖赏的确定 (Reward)

实验结果

具体学习方案

为了检验训练的效果，测试实验设置了多个场景比如：不同的地图大小，不同的子弹冷却度等，让学习机器人完全依靠学习Q表进行自主决策。观察学习机器人的决策过程，学习结果令我们很满意：学习机器人能够躲避敌人的威胁，并找到战场的最佳位置。AI-Code找到敌人并及时与有效的打击。

下载资源100个回合与fire机器人的战斗结果

相关主题

Robot Name	Total Score	Bullet Damage	Ram Damage	Surviva	Bonus
1st:	14104	9350	116	58	69
QLearningBot	7895	4656	4	43	0

评论

500个回合与fire机器人的战斗结果

Robot Name	Total Score	Bullet Damage	Ram Damage	Surviva	Bonus
1st:	62601	43146	316	11700	102
QLearningBot	45843	25840	15	1330	0

下面是最初的Q值、选择动作率、新值、新值与原始值的比较，通过对Q 表的数据分析，证明了Q表在逐渐收敛，并最终使Q表收敛到一个稳态，第498回合和499回合先后两表中各项差的平方和趋近于零。

第一回合

1

Q-value: -2.6121109205097683

2

Q-value: -0.9225833518300154

3

Q-value: -1.9101180820403036

4

Q-value: -1.5324221662755257

5

Q-value: -1.0267866181848935

6

Q-value: -1.3407928359453818

7

P(a|s): 0.050441674598822324

8

P(a|s): 0.27323856526964685

9

P(a|s): 0.10177968729898158

10

P(a|s): 0.14848834415120124

11

P(a|s): 0.2461994580991779

12

P(a|s): 0.1798522705821701

13

Random Number: 0.46314301126269086

14

Action selected: 3

15

Reinforcement: -3.0

16

Old Q-Value: 0.26091702848069354, New Q-Value: -0.1482071760320772,

17

Different: -0.40912420451277076

第498回合

1

Q-value: 0.0

2

Q-value: 0.0

内容概览

```
3 | Q-value: 0.0
4 | Q-value: 0.2457376170002843
5 | Q-value: 0.0
6 | Q-value: 0.0
7 | P(a|s): 0.15927208690906397
8 | P(a|s): 0.15927208690906397
9 | P(a|s): 0.15927208690906397
10 | P(a|s): 0.20363956545468018
11 | P(a|s): 0.15927208690906397
12 | P(a|s): 0.15927208690906397
13 | Random Number: 0.49301116788706556
14 | Action selected: 3
15 | Reinforcement: 0.0
16 | Old Q Value: 0.0, New Q Value: 0.022116385530025588,
17 | Different: 0.022116385530025588
```

强化学习

第498回合
Robocode概述

States状态描述

动作集(Actions)

数据与Q表

奖赏的确定(Reward)

具体学习方案

测试与结果

AI-CODE

下载资源

```
1 | Q-value: 0.0
2 | Q-value: 0.0
3 | Q-value: 0.0
4 | Q-value: 0.2457376170002843
5 | Q-value: 0.0
6 | Q-value: 0.0
7 | P(a|s): 0.15927208690906397
8 | P(a|s): 0.15927208690906397
9 | P(a|s): 0.15927208690906397
10 | P(a|s): 0.20363956545468018
11 | P(a|s): 0.15927208690906397
12 | P(a|s): 0.15927208690906397
13 | Random Number: 0.5797587942809831
14 | Action selected: 3
15 | Reinforcement: 0.0
16 | Old Q-Value: 0.2457376170002843, New Q-Value: 0.24328024083028146,
17 | Different: -0.0024573761700028285
```


相关主题

评论

AI-CODE

近来在研究人工智能过程和坦克机器人时，发现国内也开发出了一个类似于 Robocode 仿真器的平台 AI-CODE，其思想延用 Robocode，但在 Robocode 基础上做了很多的改进，封装了一些函数模块，让开发者更侧重于算法和程序设计的学习。最有意思的这个平台能同时支持 Java，C，C++，C# 语言，从理论上看它支持任何语言。美中不足的是国内应用的例子还不是很多，远没有 Robocode 那么多可参考的例子。如果大家有兴趣可尝试在 AI-CODE 平台上用不同语言做一些遗传算法的测试。我想能帮助更多人工智能爱好者。其相关网站大家可上：<http://www.ai-code.org>上去了解。

下载资源

 [Source code](#) (kevin.zip | 16KB)

相关主题

[人工智能 Java 坦克机器人系列: 遗传算法](#)：介绍了近几年发展起来的一种崭新的全局优化算法 —— 遗传算法。

[专栏：Robocode 技巧精粹](#)：这里包含广大 Robocode 爱好者成功经验的总结。

Tsang Hin Hang 的[Reinforcement Learning in Robocode](#)。

M.C.M. Urlings的文章[Comparing Reinforcement Learning with Genetic Algorithms in RoboCode](#)。

[AI-CODE 技术支持网站](#)：关于 AI-CODE 高级应用。

大量测试机器人下载 [Robcode 仓库](#)

评论

添加或订阅评论，请先[登录](#)或[注册](#)。

☐ 有新评论时提醒我

developerWorks

站点反馈

我要投稿

投稿指南

报告滥用

第三方提示

关注微博

加入

ISV 资源 (英语)

选择语言

English

中文

日本語

Русский

Português (Brasil)

Español

한글

技术文档库

订阅源

社区

dW 中国时事通讯

软件下载

[联系 IBM](#) [隐私条约](#) [使用条款](#) [信息无障碍选项](#) [反馈](#) [Cookie 首选项](#)