P3：星球大战的行为树

## 简介\_

在此编程作业中，您需要使用 Python 实现一个使用行为树玩《星球大战》的机器人。 《星球大战》是一款即时战略游戏，您必须逐个星球地征服一个星系。每个行星每回合都会产生船只，船只可用于从敌人或中立势力手中接管其他行星（类似于Galcon ）。这些机器人将使用用 Java 实现的游戏版本来实现，这将在下一节中详细描述。您的目标是设计一个具有单一行为树的反应式机器人，该机器人成功地战胜了一系列测试机器人，每个测试机器人都代表着独特的挑战。此外，我们将与班级机器人进行一场小型比赛。我们将使用每对随机选择的三个地图来成对测试提交的机器人。比赛的获胜者将获得额外的学分。

例子

为了以交互方式运行游戏并测试您的机器人，您需要执行以下命令（假设您位于 / src文件夹中）：

*$ python 运行.py*

当您运行此代码时，您应该会看到一个新窗口，其中显示了第一场比赛的初始状态，如图 1 所示。该程序在您的机器人和您必须击败的其他五个预定义机器人之间运行一场比赛。然后它会在界面中重播每场比赛（以便您可以观看！）。要观看比赛，您可以使用窗口上的按钮。比赛也可以在没有图形界面的情况下运行（使用*测试*功能），它运行比赛并在控制台中报告结果。

**如果您在使用 run.py 时遇到问题，请查看其第 11、12、23 和 24 行，并可能将 python 更改为 python3 或其他。**

## image03.pngimage05.pngimage04.png

图1：第一场比赛截图（ *bt\_bot* 与 *easy\_bot* ）处于三种不同的状态：初始状态、5 个刻度后和最终状态（玩家一获胜）。

## 基本代码概述\_

* **运行.py**

这是代码的主要执行点。 *show\_match*函数在两个机器人之间运行匹配，然后在界面中重播匹配。 *test*运行匹配并在终端中报告结果。

* **行为树机器人/bt\_bot.py**

这是您实施玩游戏的总体策略的地方。这里已经有一个基本策略。您可以使用它作为起点，也可以将其完全丢弃并用您自己的替换。

* **behavior\_tree\_bot/behaviors.py**

您将在此处实现操作节点的功能，通常是发布命令。每个函数应该只将游戏状态作为参数。这里已经实现了两个操作作为示例： *attack\_weakest\_enemy\_planet* 和*spread\_to\_weakest\_neutral\_planet 。*

* **行为树机器人/checks.py**

您将在此处实现基于状态的条件检查的函数。与动作一样，每个函数应该只将游戏状态作为参数。这里已经实现了两个条件检查作为示例： *if\_neutral\_planet\_available* 和*have\_largest\_fleet 。*

* **行为树机器人/bt\_nodes.py**

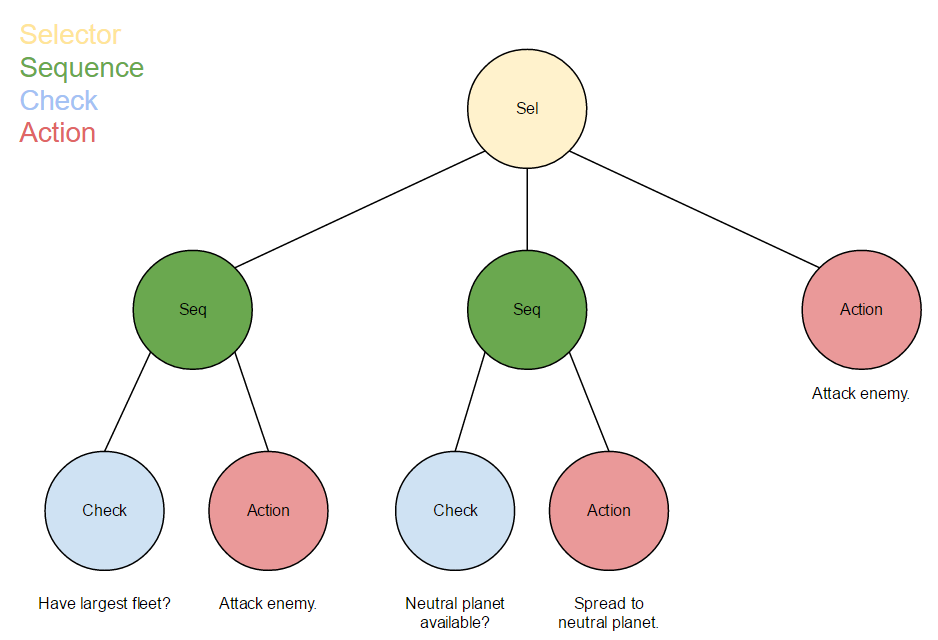
包含构建行为树所需的所有节点类。您将使用 Check、Action、Selector 和 Sequence 类。**不要使用 Node 或 Composite 类。这些是用于其他节点类型的抽象父类。**

* *检查*- 包含检查函数的叶节点，即检查状态内条件的函数。**这些函数调用不应发出命令。**
* *动作*- 包含动作函数的叶节点，通常发出一个或多个命令。
* *选择器*- 包含子节点有序列表的分支节点。当执行选择器节点时，它将尝试*按顺序执行其子节点*，直到返回**成功**（True）。一旦子节点返回成功，它就会跳过剩余子节点的执行。
* *序列*- 包含子节点有序列表的分支节点。当执行序列节点时，它将尝试按顺序执行其子节点，*直到*返回**失败**（False）。一旦子进程返回失败，序列就会中止。
* ***可选***：使用已建立的类，实现装饰器类型（例如 Inverter/ LoopUntilFailed / AlwaysSucceed /等）和/或随机选择器的行为树节点。
* **星球大战.py**

包含行星、舰队和游戏状态 ( PlanetWars ) 的类，以及两个函数 ( issue\_order和finish\_turn )。

* *PlanetWars*包含游戏的所有相关信息以及用于访问信息的几种方便的界面方法：
  + my\_planets ( ) - 您的机器人拥有的所有行星的列表
  + neutral\_planets ( ) - 所有未被拥有的行星的列表
  + 敌人\_行星( ) - 敌人拥有的所有行星的列表
  + not\_my\_planets ( ) - 不属于您的机器人的所有行星的列表
  + my\_fleets ( ) - 您的机器人舰队列表
  + 敌方舰队( ) - 敌方舰队列表
  + distance( source\_planet , destination\_planet ) - 两个行星之间的距离
  + is\_alive ( player\_id ) - 如果玩家拥有任何舰队或行星，则返回 True，否则返回 False
* *Planet*包含描述行星的信息：
  + PlanetWars.planets中的索引）
  + x, y - 行星的坐标
  + 所有者 - 所有者的 ID（0 - 中立，1 - 你，2 - 对手）
  + num\_ships - 船舶数量
  + Growth\_rate - 如果不是中性的话，每回合添加多少艘船
* *舰队*包含描述舰队的信息：
  + 所有者-所有者的 ID（0 - 中立，1 - 你，2 - 对手）
  + num\_ships - 船舶数量
  + source\_planet - 舰队起源地
  + destination\_planet - 舰队前往的地方
  + Total\_trip\_length - 两颗行星之间的距离
  + turns\_remaining - 舰队到达之前还剩多少回合
* *问题订单（状态、源行星、目的地行星、舰队编号）*
  + 如果源星球拥有足够的船只，就会创建一支新的舰队前往目的地星球。源星球的飞船数量已更新。
  + 如果源星球不属于机器人，或者没有足够的船只用于订单，则该函数返回 False（失败）
* *完成\_转( )*
  + 将回合传给对方球员

行为树

图 2 显示了*bt\_bot.py*中已实现的策略的行为树。构造此树的代码在*setup\_behavior\_tree*函数中定义。

*选择器*：策略的高级排序

|*顺序*：进攻策略

| |*检查*： have\_largest\_fleet

| |*行动*： attack\_weakest\_enemy\_planet

|*顺序*：传播策略

| |*检查*： if\_neutral\_planet\_available

| |*行动*： spread\_to\_weakest\_neutral\_planet

|*行动*： attack\_weakest\_enemy\_planet

以文本和图形格式描述的行为树示例。

该树按以下顺序执行：

* *选择器*->开始执行子节点
  + *序列*-> 开始执行子节点
    - *检查（最大的机队？）* -> 错误。返回失败（False）。
  + *序列*-> 收到失败。中止执行；返回失败。
* *选择器*-> 收到失败。尝试下一个孩子。
  + *序列*-> 开始执行子节点。
    - *检查（中性行星可用？）* -> 正确。返回成功（True）。
  + *序列*-> 收到成功。执行下一个孩子。
    - *行动（传播到中立星球）* -> 发布命令。返回成功。
  + *序列*-> 收到成功。子节点已完成。返回成功。
* *选择器*-> 收到成功。中止执行剩余的子项。

一些注意事项：

* 叶节点（操作和检查）包含执行函数
* 复合节点（选择器和序列）包含子节点。
* root.tree \_to\_string ())检查树时清晰起见
* 复合节点可以包含任意数量、任意种类的子节点。
* **有用：**每个节点类型都包含一个copy( ) 方法，该方法返回节点的副本。如果该节点是复合节点，它将递归地复制整个子树以供重用。

要求

* 提交一个基于行为树的机器人（ behavior\_tree\_bot目录中的*bt\_bot.py* ），该机器人成功战胜了所有提供的五个机器人。我们将运行它来验证它是否获胜。
* 提交包含机器人使用的原始操作和检查的behaviors.py 和checks.py 文件。
* 确保您的机器人在游戏要求的时间内运行（每回合 1 秒）。
* 提交显示您的行为树的文本文件（使用 print( root.tree \_to\_string ()）。

分级标准

* 您的机器人在每回合 1 秒的时间限制内运行。
* 您的机器人完成每个测试用例（每个测试用例的单独点）。
* 比赛的获胜者将获得额外的学分。

提交说明

提交以“Lastname1-Lastname2-P3.zip”形式命名的 zip 文件，其中包含：

* zip 文件应包含您的behavior\_tree\_bot文件夹，包括其所有文件。**请包含文件夹本身，而不仅仅是文件。这将有助于全班比赛的自动化。**
* 从行为树的根调用的tree\_to\_ string ( )的输出。如果您没有更改原始bt\_bot中的特定代码行，它应该是构建行为树后记录的第一个项目。您可以将其从那里复制到新的文本文件。

尖端

我们提供了一个名为*do\_nothing\_bot 的对手机器人*，它不执行任何操作。如果您想观察您的机器人，而又不想让活跃的对手变得复杂，请尝试一下。

***无法使用 print 语句来调试您的机器人。***游戏（Java 程序）通过stdout与机器人进行通信。幸运的是，Python 在其标准库中有一个方便的日志记录模块，可用于成熟的调试实践。概述如下：

* 在每个机器人进程开始时，记录器都会通过以下方式创建一个同名的日志文件

logging.basicConfig (文件名=\_\_file\_\_[:- 3 ] + **'.log'** , filemode = **'w'** , level = logging.DEBUG )

* 如果您的机器人崩溃，记录器将通过机器人代码的try-> except 部分中的logging.exception () 调用来记录跟踪。
* 当进程运行时，*任何模块中的任何日志记录调用*都将写入此文件。这些由进行的日志记录调用类型标记（信息、警告、调试、错误、异常等--- 请参阅[https://docs.python.org/3/library/logging.html](https://docs.python.org/2/library/logging.html) ）例如，我们有bt\_bot设置为在构造行为树时，将其记录为

日志记录.info( **' \n '** + root.tree\_to\_string ())

* 我们已经将行为树的执行调用包装在logging.debug ()调用中，默认情况下您应该在日志中看到执行跟踪。
* 您可能还想在行为和检查文件中添加自己的调试调用。如果您得到不正确的逻辑或值，请在收到无效行为时尝试logging.error ()。

参考

* 行为树概述： [http://gamasutra.com/blogs/Ch r isSimpson/20140717/221339/Behavior\_trees\_for\_AI\_How\_they\_work.php](http://gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php)
* 装饰器 - <http://aigamedev.com/open/article/decorator/>
* 游戏说明： [http : //franz.com/services/conferences\_seminars/webinar\_1-20-11.gm.pdf](http://franz.com/services/conferences_seminars/webinar_1-20-11.gm.pdf)