# 项目：Hog游戏

## 介绍

在这个项目中，需要为Hog（骰子游戏）开发模拟器和多种策略，实现过程需要结合使用控制和高阶函数。在 Hog游戏中，两名玩家轮流尝试投掷骰子先达到100分的胜出。在每一轮中，当前玩家选择一定数量的骰子来掷骰子，最多10个，一个回合得分是该回合所有投掷骰子结果的总和或者为1。

为了给游戏增添趣味，我们将玩一些特殊的规则：

* Pig out： 如果玩家一轮投掷骰子过程中出现了一次1，则该回合该玩家得分为1
* Free bacon： 如果玩家选择掷零个骰子，则其得分比对手得分中各位上最大的数字多1。例如，如果玩家 1 有 42 分，则玩家 0 通过掷零骰子获得 1 + max(4, 2) = 5 分。如果玩家 1 有 48 分，则玩家 0 获得 1 + max(4, 8) = 9 分。
* Hog wild： 如果两位玩家的总分之和是七的倍数（例如，14、21、35），则当前玩家掷四面骰子而不是通常的六面骰子。
* Swine swap：如果在回合结束时其中一位玩家的总分恰好是另一位玩家总分的两倍，则玩家交换总分。 例1：选手0有20分，选手1有5分；轮到玩家 1 了。她又得了 5 分，总分达到 10 分。玩家交换分数：玩家 0 现在有 10 分，玩家 1 有 20 分。现在轮到玩家 0 了。 示例 2：玩家 0 有 90 分，玩家 1 有 50；轮到玩家 0 了。她又得了 10 分，使她的总分达到 100。玩家交换得分，玩家 1 以 100 比 50 赢得比赛。

该项目包括六个文件，但所有更改都将对第一个文件进行，并且这是唯一需要阅读和理解的文件。下载所有项目代码并解压项目得到下列文件：

hog.py ：Hog 的初始实现。

dice.py ：掷骰子的函数。

ucb.py ：CS 61A 的实用功能。

hog\_gui.py：Hog 的图形用户界面。

hog\_grader.py：测试以检查您的实施的正确性。

autograder.py：用于分级的实用函数。

小组将提交整个项目，仅需要提交的唯一文件是hog.py。该项目价值 20 分，正确性得 14 分，程序的整体可维可读等得6分。

对于我们要求完成的功能，可能会提供一些初始代码。如果不想使用该代码，请随时将其删除并从头开始。还可以根据需要添加新的函数定义。但是，请不要修改任何其他功能。这样做可能会导致您的代码无法通过我们的 autograder 测试。此外，不要更改任何函数签名（名称、参数顺序或参数数量）。

### 1.1 图形用户界面

提供的图形用户界面（简称 GUI）。目前它不起作用，因为还没有实现游戏逻辑。完成问题 4（play函数）后，您就可以玩完全互动版的 Hog！为了渲染图形，请确保您的计算机上安装了 Python 的主要图形库 Tkinter。完成后，您可以从终端运行 GUI：

pythonhog\_gui.py

### 1.2 测试

在整个项目中，您应该测试代码的正确性。经常测试是个好习惯，这样很容易隔离任何问题。

许多测试都包含在hog.py. 附加测试在hog\_grader.py. 要运行所有测试直到发现问题，请运行

pythonhog\_grader.py

上面的命令运行所有测试，直到发生错误，此时它将停止并打印一些错误消息。您还可以针对特定问题运行测试-q：

pythonhog\_grader.py -q 1

在 中hog.py，我们还提供了一种从终端交互调用某些函数的方法：

pythonhog.py -i roll\_dice

## 第一阶段：模拟器

在第一阶段，将为 Hog 游戏开发一个模拟器。

### 2.1问题 1（2 分）

实现roll\_dice 中的函数hog.py，该函数返回多次投掷骰子的总点数或 1。

每次掷骰子的单个结果，通过调用dice()获得。投掷num\_rolls次则调用dice()函数num\_rolls次。实现中需要考虑Pig out规则的实现。

在开发过程可以添加print语句以查看程序中发生了什么。完成后移除它们。

测试实现代码：

pythonhog\_grader.py -q 1

还可以运行交互式测试：

pythonhog.py -i roll\_dice

### 2.2问题 2（1 分）

实现take\_turn 函数，并返回该回合的得分。需要在此处实现Free bacon规则，实现需要调用 roll\_dice.

测试实现代码：

pythonhog\_grader.py -q 2

还可以交互式运行take\_turn。

pythonhog.py -i take\_turn

**问题 3（1 分）**

实现一个辅助函数select\_dice，将简化（下一个问题）play的实现。该函数select\_dice有助于执行Hog wild。此函数有两个参数：当前和对方玩家的得分。

测试实现代码：

pythonhog\_grader.py -q 3

**问题 4（3 分）**

实现play 模拟一整场Hog游戏的功能。玩家轮流轮流使用最初提供的策略，直到其中一名玩家达到goal分数（缺省为100）。当游戏结束时，play返回两个玩家的最终总分，首先是玩家 0 的分数，然后是玩家 1 的分数。

这里有一些提示：

请记住执行所有特殊规则！您应该在这里执行Hog wild特殊规则（通过使用 select\_dice），以及Swine swap特殊规则。

需要调用take\_turn函数。

可以通过调用提供的函数来获取其他玩家的值（0 或 1）other。例如，other(0)为 1。

策略是一种函数，它根据两个玩家的得分来确定玩家想要掷多少次骰子。策略函数（例如strategy0和strategy1）有两个参数：当前玩家和对方玩家的分数。一个策略函数返回当前玩家想要在回合中掷的骰子数。不要担心实施策略的细节，将在第 2 阶段开发它们。

测试实现代码：

python hog\_grader.py -q 4

交互式测试，在这里您可以选择为两个玩家掷多少骰子。你会想要添加print语句来显示玩游戏的结果，但一定要在进入第 2 阶段之前删除它们。

python hog.py -i play

测试通过后将能够玩游戏的图形版本。我们提供了一个名为的文件hog\_gui.py，您可以从终端运行该文件：

python hog\_gui.py

如果您还没有安装 Tkinter（Python 的图形库），则需要先安装它，然后才能运行 GUI。

GUI 依赖于您的实现，因此如果您的代码中有任何错误，它们将反映在 GUI 中。这意味着您还可以将 GUI 用作调试工具；但是，最好先运行测试。

恭喜！你已经完成了这个项目的第一阶段！

## 第二阶段：策略

在第二阶段，您将尝试改进始终掷固定数量骰子的基本策略的方法。首先，您需要开发一些工具来评估策略。

**问题 5（2 分）。**

实现make\_averaged 功能。这个高阶函数以一个函数fn作为参数。它返回另一个函数，该函数采用与原始函数相同数量的参数。fn此返回函数与输入函数的不同之处在于，它返回重复调用相同参数的平均值。此函数功能为调用fn总次数num\_samples次并返回结果的平均值。

注意：如果输入函数fn是非纯函数（例如函数random），那么make\_averaged 也将是非纯函数。要实现这个功能，你需要一段新的 Python 语法！您必须编写一个接受任意数量参数的函数，然后使用这些参数调用另一个函数。这是它的工作原理。

我们不列出函数的形式参数，而是编写 \*args. 要使用这些参数调用另一个函数，我们再次调用它\*args。例如，

仔细阅读文档字符串make\_averaged以了解它的工作原理。

测试实现代码：

python hog\_grader.py -q 5

**问题 6（2 分）**

实现 max\_scoring\_num\_rolls函数，它运行一个实验来确定给出一个回合的最大平均得分的掷骰数（从 1 到 10）。需要使用make\_averaged和 roll\_dice。它需要打印出所有可能掷骰数的平均值，就像在 doctest 中一样max\_scoring\_num\_rolls。

测试实现代码：

python hog\_grader.py -q 6

要在随机骰子上运行此实验，请run\_experiments 使用以下-r选项调用：

python hog.py -r

运行实验

run\_experiments对于本项目的其余部分，您可以根据需要更改 的实现。通过调用average\_win\_rate，您可以评估各种 Hog 策略。例如，将第一个更改if False:为if True:，以便always\_roll(8)根据 的基线策略进行评估always\_roll(5)。您应该会发现它输的次数多于赢的次数，因此胜率低于 0.5。

某些实验可能需要长达一分钟的时间才能运行。您始终可以减少样本数量以make\_averaged加快实验速度。

**问题 7（1 分）**

策略可以通过在最有利的时候选择投掷0次骰子(利用 Free bacon), 实现 bacon\_strategy，如果投掷0次得分不小于BACON\_MARGIN则返回0，否则返回 BASELINE\_NUM\_ROLLS（这两个全局变量位于always\_roll函数的正上方）。

测试实现代码：

pythonhog\_grader.py -q 7

**问题 8（2 分）**

策略也可以利用Swine swap。实施swap\_strategy，其中

如果投掷0次有益的Swine swap， 则投掷0次。

如果投掷0 会导致有害交换并失去分数，则掷 BASELINE\_NUM\_ROLLS次。

如果投掷0次不会导致交换，那么它至少能获得BACON\_MARGIN点数，否则投掷BASELINE\_NUM\_ROLLS次 。

测试实现代码：

python hog\_grader.py -q 8

整体运行测试。

python hog\_grader.py

恭喜，您的第一个项目已经结束！