随机化

游戏充满了基于机会和概率的机制。有些棋盘游戏决定了玩家的动作，有洗牌牌游戏，街机游戏，敌人生物在不可预知的时间出现，角色扮演游戏，每个动作都有机会成功或失败，开放世界游戏，背景人物自然徘徊，还有很多例子。意想不到的惊喜可以让游戏对玩家来说更有趣；随着每次玩法而变化的行为可以为游戏增加重玩价值；模仿自然混乱系统的元素可以让游戏世界更沉浸。

构建一个依赖机会元素的游戏通常涉及使用伪随机数生成器或*随机源*。然而，并非所有随机源都是平等的，使用不当的随机源可能会破坏其目的。要在游戏中构建强大的伪随机行为，您通常需要以下部分或全部特征：

- **随机性。**随机数生成器应该产生不可预测的行为（或至少它的外观）。但是随机性是如何量化的呢？计算机化的伪随机数生成器基于似乎没有顺序或结构的*有限*数字*序列*。最终，序列将重复——源在重复之前生成的随机数确切数量取决于源使用的算法。此外，当生成的数字以二进制方式查看时，根据随机源的算法，数字中的一些位或多或少是可预测的。
- **性能。**更复杂的算法可以以处理时间为代价产生更好的不可预测性。如果您的游戏每帧使用多个随机数，并且需要以每秒60帧的速度运行，那么复杂的随机源可能会太慢。选择随机源可能涉及随机性和性能之间的妥协。
- **确定主义。**高质量的软件需要测试，但真正的随机性使得很难重新创建一组特定的条件进行测试。随机来源应保持游戏玩家的不可预测性，但允许在需要时复制某些结果序列。确定性的随机源对于网络游戏也是必要的——您需要确保随机游戏机制对所有玩家的行为相同。
- **独立。**因为随机源是基于一系列数字，所以生成的下一个数字部分取决于之前的数字。然而，一个游戏通常包含多个随机元素，其行为应该不同。例如，考虑一个随机化游戏元素和美学元素的游戏，例如一个竞争性游戏，每个玩家的每个回合的可用动作都是随机选择的，玩家角色说出随机选择的画外音对话线，为游戏设置增添风味。如果对话系统和可用动作都使用相同的随机化器，一个老练的玩家可能会根据口语对话预测对手的下一步行动。
- **已知分布。**对于游戏中随机化的许多用途，随机数最好遵循*均匀分布*——也就是说，生成特定数字的概率与源生成的数字范围内任何其他数字的概率相同。对于其他用途，随机数最好遵循特定的分布——例如，许多自然过程涉及*正态分布*，其中随机采样通常产生接近平均值的结果，而其他结果离平均值越远的可能性就越小。

GameplayKit包括一套几个随机化类来满足这些目标。

在游戏中使用随机化

GameplayKit中的所有随机化类或随机化器都符合`GKRandom`协议，该协议描述了生成随机数的最小接口。要使用随机化器，您首先需要选择一个适合您任务的：

- In most cases, you need random numbers that are uniformly distributed across a specific range. For this task, use the `GKRandomDistribution` class.
- 要自定义随机化行为，但要保持均匀分布，请选择不同的`GKRandomSource`子类，为`GKRandomDistribution`对象提供基础随机值。
- 要自定义随机数的分布，请使用`GKGaussianDistribution`或`GKShuffledDistribution`类。
- 如果您不需要具有特定范围或分布的随机数，请直接使用`GKRandomSource`子类之一。

  You can also use one of the `GKRandomSource` classes directly to randomize the order of objects in an array; for example, to shuffle a deck of cards. First, choose a random source object with the characteristics you need, then pass the array to the random source's `arrayByShufflingObjectsInArray:` method. This method returns a copy of the original array, but with its contents in a randomized order.

以下各节详细介绍了随机源和随机分布类集之间的差异。

随机来源：随机化的积木

`GKRandomSource`类及其子类提供了生成随机数的算法。要开始构建随机行为，请选择一个具体的`GKRandomSource`子类：

- The `GKARC4RandomSource` class uses the ARC4 algorithm, which is suitable for most gameplay uses. (This algorithm is similar to that used in the C `arc4random` API; however, the `GKARC4RandomSource` class is independent from those functions.)
- `GKLinearCongruentialRandomSource`类使用的算法比`GKARC4RandomSource`类更快，但更少随机。（具体来说，生成数字的低位重复频率高于高位。）当性能比鲁棒不可预测性更重要时，请使用此源。
- `GKMersenneTwisterRandomSource`类使用的算法比`GKARC4RandomSource`类慢，但更随机。当您使用随机数不显示重复模式和性能是不那么令人担心时，请使用此源。

当您创建其中一个类的实例时，结果是一个*独立的*随机源 — — 即一个实例生成的数字序列对另一个实例生成的序列没有影响。

All of these classes implement *deterministic* random number generation. Each class uses a *seed value* that determines its behavior. When you initialize a random source with the `init` initializer, GameplayKit uses a nondeterministic seed value. If you want to re-create the behavior of a specific random source instance, read that value from its `seed` property, then initialize a new source with the `initWithSeed:` initializer to reproduce the sequence of numbers generated by the original source.

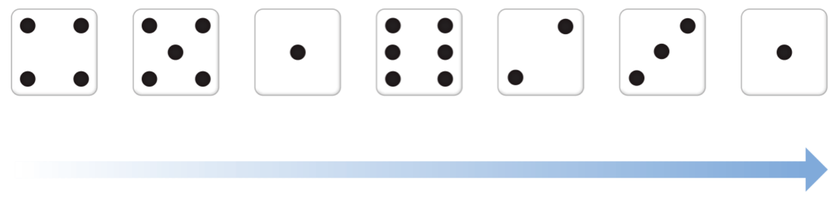
随机源存档是保存其状态的另一种方式。例如，如果您使用`NSKeyedArchiver`对象对`GKRandomSource`（或子类）实例进行编码，则从同一归档解码的任何两个随机源将产生相同的随机数序列。

随机分布产生专门的随机行为

随机源提供了基本的随机数生成算法。构建随机博弈机制通常需要进一步的专业化，例如在许多随机采样中出现的生成数字或一致模式（或*分布*）的范围。要控制这些参数，请使用`GKRandomDistribution`类或其子类之一：

- `GKRandomDistribution`类本身在指定范围内产生均匀分布——也就是说，指定最小值和最大值之间的每个值都同样可能发生。每当您需要属于特定数字范围内的随机数时，请使用此类。骰子是具有特定范围的均匀随机分布的常见示例，如图2-1所示。

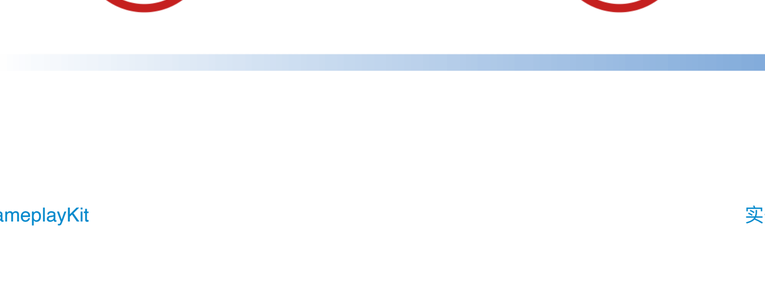
图2-1 A均匀随机分布模拟单模卷



  For example, a fair six-sided die has the same probability of landing on any of the numbers one through six. To model commonly used kinds of dice, use the `d6` and `d20` convenience initializers, or use the `distributionForDieWithSideCount:` initializer to model a die with any number of sides.

- The `GKGaussianDistribution` class models a Gaussian distribution, also known as a *normal distribution*, shown in Figure 2-2. In a normal distribution, a certain result—the *mean*—has the highest probability of occurring, and higher- or lower-valued results are less likely. This distribution is symmetric: results that are the same distance from the mean have the same probability of occurring, whether they are above or below the mean.

图2-2 A高斯随机分布相当于一卷多骰子



  高斯分布出现在许多现实世界的现象中，你可能会在游戏中建模。例如：

  - **一卷多掷骰子。**如果角色扮演游戏涉及投掷三个六边骰子（通常缩写为*3d6*）来计算命中的伤害，您可以平衡游戏中的其他变量，假设命中通常会导致9到13点的伤害。
  - **投掷飞镖的玩家。**每个飞镖命中的x和y坐标可以随机化在这个分布上，导致飞镖聚集在飞镖板中心附近。
  - **随机生成的字符变体。**非玩家角色的高度、重量和其他身体特征可以正常分布，为您的游戏世界提供一个现实的人口，大多数人的平均身高，非常高或非常矮的人很少。

- `GKShuffledDistribution`类建模的分布通常随着时间的推移是一致的，但这也防止了类似值的集群按顺序出现，如图2-3所示。一个真正随机的序列通常可以包括一系列连续的“好”或“坏”值，因此洗牌分布可以通过避免此类行为来帮助您的游戏看起来更公平。这种发行版在任何游戏机制中都很有用，但特别适用于随机行为是游戏的关键，用户高度可见的情况，例如屏幕上显示的骰子卷或洗牌供玩的牌。

图2-3 A 洗牌随机分布避免重复相同值