PyPBE Documentation

Release 0.16

Eric Strong

Sep 01, 2017

Contents

PyPBE is a resource for tabletop gaming which allows Gamemasters (GM) to fairly select which random rolling method is closest to an equivalent Point Buy value. Ideally, all players will determine their character's stats exactly the same way. However, in some cases, players may ask for several different options for generating their character's stats. PyPBE is designed to allow GMs to make this decision in a fair way. By calculating the equivalent Point Buy for a random rolling method, the GM can determine the expected power level of the characters, as well.

PyPBE is designed for Pathfinder, 3e, 3.5e, 4e, and 5e characters. However, the Python API allows the option to supply a custom Point Buy mapping, which means that it is applicable for any system in which the "Point Buy" concept applies. PyPBE also supports any number of attributes, although it was designed for the common 6-attribute system (strength, constitution, dexterity, intelligence, wisdom, charisma).

CHAPTER 1

1. Introduction

PyPBE is a resource for tabletop gaming which allows Gamemasters (GM) to fairly select which random rolling method is closest to an equivalent Point Buy value.

Overview

Some GMs prefer to let their players choose between rolling for their ability scores and letting them use the Point Buy system. However, not all random rolling methods are created equal. Some (4d6, drop lowest) clearly give higher average results than others (3d6). PyPBE is designed to calculate and visualize the distribution of a specified ability score rolling method, which may provide useful information for decision-making.

The stats that PyPBE calculates aren't the "raw" values of the roll (e.g. typically 3 through 18), they're the "Point Buy Equivalent" of 6 rolls using that rolling method. For instance, if you roll 3d6 six times, you might get 10, 12, 8, 13, 7, 9, which has a Point Buy Equivalent of -2 (0+2-2+3-4-1) using the Pathfinder point buy scheme.

PyPBE uses Monte Carlo simulation to obtain its results. If you perform the above process thousands/millions of times, you will get a distribution. The mean of that distribution is the fair Point Buy you should select for that rolling method, and 90% of the time, the random roll PBE will fall between the 5%/95% values. The "Typical Array" gives the most likely stat array using that random rolling method.

Package Organization

PyPBE is organized into a single namespace, called "core". The "PBE" class inside the core namespace is used to simulate a single rolling method (such as the sum of 3d6 for 6 attributes), plot a histogram of the results, and save the results to an array.

For those who wish to contribute or dig deeper into the code, the following folders in the GitHub repository may be of interest:

- pypbe: contains the core functionality of PyPBE
- pypbe-bk: a Bokeh server for visualizing the results from running PyPBE, with a Procfile for running on Heroku

- pypbe-nb: an (older) Jupyter notebook with a basic GUI that shows off some of the functionality of PyPBE
- pypbe-rec: a Bokeh server for recommending a dice rolling method based on a selected point buy (think of this as the inverse of pypbe-bk)
- pypbe-sim: an (older) Windows executable for running PyPBE locally without Python
- tests: a set of unit tests for the core functionality (needs some work)

Systems

PyPBE is designed for Pathfinder, 3.5e, and 5e characters. However, it allows the option to supply a custom Point Buy mapping, which means that it is applicable for any system in which the "Point Buy" concept applies. PyPBE also supports any number of attributes, although it was designed for the common 6-attribute system (strength, constitution, dexterity, intelligence, wisdom, charisma).

CHAPTER 2

2. Getting Started

Installation

If Python is already installed on your computer, PyPBE can be installed using PyPI by opening a command window and typing:

pip install pypbe

Upgrading to a new version of pyedna can be accomplished by:

pip install pypbe –upgrade

The source code of pyedna is hosted on GitHub at:

https://github.com/drericstrong/pypbe

Python Requirements

Required libraries: numpy, seaborn, matplotlib

A requirements.txt document is located in the GitHub repository, and all package requirements can be installed using the following line in a command window:

pip install -r requirements.txt

Numpy is required for the random arrays, and seaborn/matplotlib are required to visualize the histograms. It is very unlikely that these requirements will change in the future.

Python Version Support

Currently, PyPBE only supports Python 3.2+ and is not fully compatible with Python 2. If this is important to you, please make a pull request at:

https://github.com/drericstrong/pypbe

The package maintainer welcomes collaboration.

Importing PyPBE

The main class in PyPBE is usually imported into a script using the following line:

from pypbe import PBE

CHAPTER 3

3. Python API

Basic Examples

Import the PBE class into your program, initializing it using the number and type of dice to roll, then use the "roll_mc" method to investigate the distribution. The results can be visualized using the "plot_histogram" method, and a data row summary can be generated using the "get_results" method.

For example, running a simulation for three 6-sided dice (3d6), rolled for 6 attributes, can be accomplished using the following code:

> from pypbe import PBE

> alg = PBE(3,6)

> alg.roll_mc()

> alg.plot_histogram()

> results = alg.get_results()

The "plot_histogram" function will generate a plot that looks like this:

The top part of the plot shows the distribution of each dice roll, ranked from lowest (Roll 1) to highest (usually Roll 6). Think about it this way- the mean value of 6 sets of 3d6, repeated many times, will tend towards 10.5 (each six-sided dice has a mean of 3.5, so 3.5 times 3 equals 10.5). However, if you order the 6 sets from lowest to highest, you'll notice that the lowest roll tends to be lower than 10.5, and the highest roll tends to be higher than 10.5. The plot shows the expected value for each ranked roll, which can be interpreted as the "typical" stat array for this rolling method. In the figure above, the 5th and 95th percentiles are given in brackets. For instance, [5,11] means that 90% of the distribution is between 5 and 11.

The distribution of the Point Buy value is shown in the bottom plot by mapping the results of each dice roll to a Point Buy value. The default mapping is: {3:-16, 4:-12, 5:-9, 6:-6, 7:-4, 8:-2, 9:-1, 10:0, 11:1, 12:2, 13:3, 14:5, 15:7, 16:10, 17:13, 18:17}. The mean, standard deviation, 5th, and 95th percentiles are shown on the figure. You can interpret the mean of the Point Buy distribution as the "Point Buy Equivalent"- the Point Buy value that is most fair to choose as the equivalent for the ability score rolling method.

The *get_results()* and *plot_histogram()* methods can be chained with the *roll_mc()* method, like this:

> hist_plot = alg.roll_mc().plot_histogram()

> res = alg.roll_mc().get_results()

Custom Parameters

More complicated scenarios can be run by adjusting the following user-specified parameters:

- System (*pbe_map* parameter)

Each RPG system has its own point buy scheme, and buying an attribute value may cost different amounts in different systems. For example, in PF a '10' will cost '0' point buy, while in 3e a '10' will cost '2' point buy. It's important that you specify the right system; the results will vary heavily depending on which system you choose. You can try this out for yourself by comparing PF to 3e. Currently recognized values for this parameter include: 'PF', '3e', '4e', and '5e'. However, a custom point buy scheme can be specified using the *custom_pbe_map* mentioned below.

Default: 'PF'.

Example: the rpg group is playing D&D 3rd edition, so the value should be '3e'.

- Number of Dice Per Attribute (*num_dice* parameter)

Each time you roll for an attribute (such as STR), this parameter specifies the total number of dice that you will roll. When using 'XdY+Z' notation, this value is 'X'.

Example: if you are rolling three six-sided dice (3d6) to determine your stats, this value should be '3'.

- Dice Sides (*dice_type* parameter)

This parameter specifies the number of sides per dice. When using 'XdY+Z' notation, this value is 'Y'.

Example: if you are rolling three six-sided dice (3d6) to determine your stats, this value should be '6'.

- Modifier (*add_val* parameter)

Each time you roll for an attribute (such as STR), this parameter specifies the amount that will be added or subtracted. When using 'XdY+Z' notation, this value is 'Z'.

Note- this same modifier is applied to every attribute, up to the total 'Number of Attributes'. For instance, if you roll 6 attributes with a 'Modifier' of -1, the -1 will be applied to each attribute individually. This is not the same thing as racial modifiers.

Default: '0'.

Example: if you are rolling three four-sided dice and adding six (3d4+6) to determine your stats, this value should be '6'.

- Dice to Keep Per Attribute (*keep_dice* parameter)

This value allows you to roll more dice than you need, keeping only the best ones. Keeping less dice than the total rolled dice will increase the average point buy equivalent, since the worst die rolls will be discarded. When using 'XdY+Z' notation, this value affects the number of 'X' that will be kept at the end. Note that it's impossible for the number of dice to keep to be greater than the number of dice that were rolled.

Default: if using the Python API and the *keep_dice* is not specified, the default is to use the same as the 'number of dice per attribute'. If using the Bokeh server, the default is '3'.

Example: if you are rolling four six-sided dice and keeping the best three dice, the 'number of dice per attribute' should be '4' and the 'dice to keep per attribute' should be '3'.

- Number of Attributes (*num_attribute* parameter)

This value allows you to adjust the number of character attributes which will be rolled. Most commonly, the number of attributes will be 6 (STR, DEX, CON, INT, WIS, CHA); however, some DMs may wish the characters to have additional attributes, such as 'comeliness'. Furthermore, an rpg system may require more than 6 attributes to be rolled at character creation.

Default: '6'.

Example: The DM wishes the characters to have seven attributes (STR, DEX, CON, INT, WIS, CHA, COM), so the 'number of attributes' should be '7', and the 'attributes to keep' should be '7' as well.

- Attributes to Keep (*keep_attribute* parameter)

This value allows you to roll more attributes than you need, keeping only the best ones. Keeping less attributes than the total generated attributes will increase the average point buy equivalent, since the worst attributes will be discarded. Note that it's impossible for the number of attributes to keep to be greater than the number of attributes that were rolled.

Default: '6'.

Example: if you are rolling ten attributes but only keeping the best six attributes, the 'number of attributes' should be '10', and the 'attributes to keep' should be '6'.

- Rerolls (*reroll* parameter)

Each time you roll a die, you may wish to reroll values that are too low. Any die result that is less than or equal to this parameter will be rerolled. If this value is '0' (the default), no die will be rerolled at all. Values greater than 1 are also inclusive of lower values (i.e. '3' means reroll 1s, 2s, and 3s). Increasing this value will also increase the average point buy equivalent, because the worst die rolls will be rerolled.

Default: '0'.

Examples: if are rolling three six-sided dice but rerolling any 1s that come up, this value should be '1'. If you are rerolling any 1s or 2s, this value should be '2'.

- Number of Arrays (*num_arrays* parameter)

The term 'array' refers to the total number of attributes which were generated using the parameters above. It's the 'final result', in a sense. Each time you generate a full array, it will contain a number of attributes equal to the 'attributes to keep' parameter. For instance, an array with six 'attributes to keep' might look like: [12, 10, 6, 11, 15, 17]. The 'number of arrays' parameter allows you to roll multiple arrays at once, automatically selecting the one with the highest point buy equivalent. Unfortunately, personal preference cannot be considered. For instance, a player might prefer [12, 12, 10, 10, 10, 10] over [18, 8, 8, 8, 8, 8], even though the latter array has a higher point buy equivalent.

Default: '1'.

Example: If you are rolling three arrays and choosing the array with the highest point buy equivalent, this value should be '3'.

- Monte Carlo Histories (*num_hist*)

PyPBE uses Monte Carlo simulation. Behind the scenes, the code is generating thousands (or millions, or more!) of dice rolls and calculating summary statistics from the results. Each of these summaries is called a 'history'. This parameter specifies the number of histories that should be used to determine the statistics for a given rolling method. In general, increasing the number of histories will increase the accuracy, but it will also increase the amount of time and resources that the code will need to complete the calculation. Most common applications of PyPBE will only require 10^5 histories, but more complicated examples may need up to 10^6 or 10^7 histories. Note that in the Python API, the number of Monte Carlo histories is specified when the "roll_mc" function is called, not when the PBE object is initialized.

Default: '10^5'

Example: You are running a very complicated example- rolling 4d6+6, keeping the best 2 dice, generating 12 attributes but only keeping 7, and generating 5 arrays. You notice that the results look noisy, and the histogram is full of 'spiky'

data. So instead of using the default '10^5' histories, you decide to use '10^7' histories, realizing that the code will take much longer to run now.

- Custom Point Buy Mapping (*custom_pbe_map* parameter)

This feature is recommended for advanced users who are proficient in Python and is only available in the Python API. If a system other than PF, 3e, 4e, or 5e is being used, or the DM is using a point buy scheme that needs to go below 3 or above 18, a custom point buy mapping must be specified. This is done using a Python dictionary that looks something like: {3: -16, 4: -12, 5: -9, 6: -6, 7: -4, 8: -2, 9: -1, 10: 0, 11: 1, 12: 2, 13: 3, 14: 5, 15: 7, 16: 10, 17: 13, 18: 17}. The dictionary key is the attribute value, and the dictionary value is the cost for that attribute value. Note that there is no input validation on this parameter.

Example: If you're extending the dictionary beyond 3 or 18, your new dictionary might look like: {2: -20, 3: -16, 4: -12, 5: -9, 6: -6, 7: -4, 8: -2, 9: -1, 10: 0, 11: 1, 12: 2, 13: 3, 14: 5, 15: 7, 16: 10, 17: 13, 18: 17, 19: 21}.

- Ability Score Lower Limit (*roll_low_limit* parameter)

To ensure that characters are not too weak, you may want to set a lower limit on the possible dice rolls for an ability score. This limit is evaluated **after** the dice rolls have been summed together to get an ability score. The parameter is inclusive: any value **equal to** or **greater** than the ability score lower limit will be kept.

Default: None

Example: The ability score lower limit is 6. You roll 3d6 and get 1, 2, 2. Summing the rolls together, the ability score is 5, which is less than the ability score lower limit, so this ability score is discarded.

Important Note: This option will **discard** ability scores that do not meet the criteria rather than **reroll** them. Hence, the number of Monte Carlo histories will be decreased from the originally-specified amount. The number of histories should be increased to compensate for this effect.

- Ability Score Higher Limit (*roll_high_limit* parameter)

To ensure that characters are not too powerful, you may want to set a higher limit on the possible dice rolls for an ability score. This limit is evaluated **after** the dice rolls have been summed together to get an ability score. The parameter is inclusive: any value **less than** or **equal to** the ability score higher limit will be kept.

Default: None

Example: The ability score higher limit is 16. You roll 3d6 and get 6, 6, 5. Summing the rolls together, the ability score is 17, which is greater than the ability score higher limit, so this ability score is discarded.

Important Note: This option will **discard** ability scores that do not meet the criteria rather than **reroll** them. Hence, the number of Monte Carlo histories will be decreased from the originally-specified amount. The number of histories should be increased to compensate for this effect.

- PBE Lower Limit (*pbe_low_limit* parameter)

To ensure that characters are not too weak, you may want to set a lower limit on the possible point buy equivalent. The parameter is inclusive: any value **equal to** or **greater** than the PBE lower limit will be kept.

In the PyPBE simulator (pypbe-bk), there was no easy way to specify that a lower limit should not be used at all, so the value -21 indicates that there is no lower limit.

Default: None

Example: The pbe lower limit is 5. You roll a stat array of [10, 7, 8, 9, 15, 9], which has a point buy equivalent of 0 (i.e. 0-4-2-1+8-1). This point buy equivalent is less than the PBE lower limit, so the array is discarded.

Important Note: This option will **discard** ability scores that do not meet the criteria rather than **reroll** them. Hence, the number of Monte Carlo histories will be decreased from the originally-specified amount. The number of histories should be increased to compensate for this effect.

- PBE Higher Limit (*pbe_high_limit* parameter)

To ensure that characters are not too powerful, you may want to set a higher limit on the possible point buy equivalent. The parameter is inclusive: any value **less than** or **equal to** the PBE higher limit will be kept.

In the PyPBE simulator (pypbe-bk), there was no easy way to specify that a higher limit should not be used at all, so the value 61 indicates that there is no higher limit.

Default: None

Example: The PBE higher limit is 25. You roll a stat array of [15, 18, 12, 10, 10, 15], which has a point buy equivalent of 33 (i.e. 7+17+2+0+0+7). This point buy equivalent is greater than the PBE higher limit, so the array is discarded.

Important Note: This option will **discard** ability scores that do not meet the criteria rather than **reroll** them. Hence, the number of Monte Carlo histories will be decreased from the originally-specified amount. The number of histories should be increased to compensate for this effect.

Troubleshooting

Most point buy systems cap out at 18 and bottom out at 3, since they are based on rolling 3d6. For example, you can buy an '18' attribute score, but you can't outright buy a '19' attribute score (before racial modifiers). Hence, all possible rolls using PyPBE must fall between 3 and 18, unless a custom point buy mapping is defined. One of the most common problems when using PyPBE is to have a maximum possible value that is higher than the greatest defined point buy, or a minimum possible value that is smaller than the lowest defined point buy.

Using the parameter names from the section above, the maximum possible value is: ('dice to keep per attribute' * 'dice sides' + 'modifier'). The minimum possible value is: ('dice to keep per attribute' + 'modifier'). If the maximum possible value is too high, consider decreasing the 'dice to keep per attribute', the 'dice sides', or the 'modifier'. If the minimum possible value is too low, consider increasing the 'dice to keep per attribute' or the 'modifier'. It may require a subtle balancing act to achieve parameters that meet both specifications.

CHAPTER 4

4. PyPBE Bokeh

A Bokeh server is currently running on Heroku:

https://pypbe.herokuapp.com/pypbe-bk

It contains all the functionality of the Python API in an easy-to-use GUI. For more information about the user controls, please refer to the “Custom Parameters” heading in the Python API section.

CHAPTER 5

5. PyPBE Recommender

This feature is under development.

CHAPTER 6

6. PyPBE Notebook

The PBE Jupyter Notebook allows the user to run PyPBE interactively in a Jupyter notebook. Navigate to this link and follow the instructions in the notebook:

https://github.com/drericstrong/pypbe/blob/master/pypbe-nb/PBENotebook.ipynb

Running this notebook interactively requires you to install Python 3, the latest version of which can be found here:

https://www.python.org/downloads/

Alternatively, if you're having trouble getting these instructions to work, you may want to download the scientific Anaconda package (preferred, but larger size):

https://www.continuum.io/downloads

Once Python 3 is installed, the following Python modules are also required:

- ipywidgets
- jupyter
- numpy
- matplotlib
- seaborn
- pypbe

The "install.bat" file in this base directory can be used to install these modules automatically using pip (assuming that you are on Windows). Just double-click on the batch file to run it. If you installed Anaconda, most of these are available by default. Otherwise, you can install these modules by opening a command terminal and typing (for each of the six MODULEs above):

pip install MODULE

After those steps are complete, download this notebook to your computer. You can run a Jupyter notebook by typing:

jupyter notebook

in a command terminal, or if you installed Anaconda, open the "Anaconda Navigator" program and click on the "Jupyter" icon. Once the Jupyter notebook interface is open (it should appear in a web browser), navigate to where you downloaded this file, and click on it.

Run the following code by clicking the next "code" cell, then clicking the green arrow ("run") above, and a series of sliders and a button should appear, which allows you to investigate the results from any random character attribute rolling method you desire.

The plots will keep generating every time you click the button (so you can compare results), but you can re-run the cell using the green arrow in order to clear the current plots.

CHAPTER 7

7. PyPBE Simulator

This executable was developed using an older version of PyPBE and may not work correctly at the current time.

The torrent file for a compiled, Windows executable can be found in the base directory of this repository:

https://github.com/drericstrong/pypbe/blob/master/pypbe-sim/pbeSimRun.torrent

Warning- if you're the type of person to download and run an executable from an unknown, random website, you need to take a long, hard look at your life and the various (bad) choices that have led you to this point. I cannot emphasize strongly enough that you should **not** directly download the executable from the above link. Instead, take a look at the following instructions for compiling it from source. If you can't get them to work, try searching for the answers yourself. Python is a very enjoyable language, and it may be good for your personal development to learn. In fact, you might find it fun.

The maintainer apologizes that only a Windows executable is available in this section. However, if you are running another OS, the instructions below should allow you to compile a version that works for your specific OS. Check pyinstaller documentation for compatability with your OS:

http://www.pyinstaller.org/

Compiling from Source

Unfortunately, this section does assume some knowledge of Python and programming in general. Also, I (obviously) do not personally support any of the software required for these steps, and so any major changes in the modules/libraries in this section may invalidate these instructions. These instructions were last updated May 2017.

First, install Python. Since PyPBE is written in Python, this is an absolutely necessary first step. I suggest version 3.5.3, since pyinstaller does not (currently) support 3.6. Instructions and binaries for Python 3.5.3 can be found here:

https://www.python.org/downloads/release/python-353/

Next, install the Qt framework. Qt is used for the PyPBE simulator graphical user interface. You should select the free, open-source version from this link:

https://info.qt.io/download-qt-for-application-development

Then, install the required Python modules. There's a requirements.txt file in the main directory of this repository. Download the pypbe-sim repository to your computer, navigate to the base directory (the one that "requirements.txt" is in) and then run the following command:

pip install -r requirements.txt

Finally, navigate to the "src" directory of the repository, and then run 3create.bat, which should contain the following line of code:

pyinstaller pbeSimRun.py –onefile

Once this is complete, your executable will appear in the "dist" folder of the repository!

CHAPTER 8

Change Log

Version 0.11

- Initial build

Version 0.12

- Updated plot titles
- Unit tests
- Documentation
- New logo

Version 0.13

- Support for 4e

Version 0.14

- Added a Jupyter notebook simulator (pypbe-nb)
- Added a executable simulator (pypbe-sim)

Version 0.15

- Added more unit tests

- Configuration error-checking during initialization of PBE object
- Better commenting of the core module
- Added a Bokeh server simulator (pypbe-bk)

Version 0.16

- Added the ability to specify lower and upper limits for ability scores and point buy equivalents
- Updated pypbe-bk with the limits mentioned above
- Updated documentation

CHAPTER 9

Indices and tables

- genindex
- modindex
- search