NAME

Dice – Dice Rolling Functions

SYNOPSIS

Python Usage:

import SoftwareAlchemy.GMA.Dice

d **= Dice(***qty***=1,** *sides***=20,** *bonus***=0,** *diebonus***=0,** *div***=None,** *description***=None,** *factor***=None,** *generator***=None)**
result **=** *d***.roll(***confirm***=False,** *threat***=None,** *bonus***=None)**
result **=** *d***.value()**
result **=** *d***.maxvalue()**
str **=** *d***.description()**
str **=** *d***.describe_roll()**
structured_results **=** *d***.structured_describe_roll(sf=***None***)**

dr **= DieRoller(***generator***=None)**
*label***,** *result_set* **=** *dr***.do_roll(***dice***='')**
str **=** *dr***.to_text(***structured_results***,** *label***=None,** *formatting***=None)**

main(*argv***=None,** *verbose***=False)**

Go Usage:

```
type Dice struct {
  MinValue int
  MaxValue int
  MultiDice []DieComponent
  Rolled bool
  LastValue int
}

type StructuredDescription struct {
  Type string
  Value string
}

type StructuredResult struct {
  Result int
  Details []StructuredDescription
}

type DieComponent interface {
  ApplyOp(x, y int) (int, error)
  Evaluate(x int) (int, error)
  MaxValue(x int) (int, error)
  LastValue() int
  Description() string
  StructuredDescribeRoll() string
  NaturalRoll() (int, int)
  GetOperator() string
}

type DieConstant struct { // Implements DieComponent
  Operator string
  Value int
  Label string
}

type DieSpec struct { // Implements DieComponent
  Operator string
  Value string
  Numerator int
  Denominator int
  Sides int
  BestReroll bool
  Rerolls int
  DieBonus int
  InitialMax bool
  Label string
  History [][]int
  WasMaximized bool
}
```

```
type DieRoller struct {
  LabelText string
  Confirm bool
  SuccessMessage string
  FailMessage string
  Template string
  Permutations [][]interface{}
  RepeatUntil int
  RepeatFor int
  DoMax bool
  DC int
  PctChance int
  PctLabel string
}
```

Type **Dice**

```
  func NewDice(description string) (*Dice, error)
  func NewDiceBasic(qty, sides, bonus int) (*Dice, error)
  func NewDiceByParameters(qty, sides, bonus, diebonus, div, factor int) (*Dice, error)
  func (d *Dice) Description() string
  func (d *Dice) MaxRoll() (int, error)
  func (d *Dice) MaxRollToConfirm(bonus int) (int, error)
  func (d *Dice) Roll() (int, error)
  func (d *Dice) RollToConfirm(confirm bool, threat, bonus int) (int, error)
  func (d *Dice) StructuredDescribeRoll(sf, successMessage, failureMessage string, rollBonus int)
([]StructuredDescription, error)
```

Type **DieConstant**

```
  func (dc *DieConstant) ApplyOp(x, y int) (int, error)
  func (dc *DieConstant) Description() string
  func (dc *DieConstant) Evaluate(x int) (int, error)
  func (dc *DieConstant) GetOperator() string
  func (dc *DieConstant) LastValue() int
  func (dc *DieConstant) MaxValue(x int) (int, error)
  func (dc *DieConstant) NaturalRoll() (int, int)
  func (dc *DieConstant) StructuredDescribeRoll() []StructuredDescription
```

Type **DieRoller**

```
  func NewDieRoller() (*DieRoller, error)
  func (dr *DieRoller) DoRoll(spec string) (string, []StructuredResult, error)
```

Type **DieSpec**

```
  func (ds *DieSpec) ApplyOp(x, y int) (int, error)
```

func (*ds* *DieSpec) Description() string
func (*ds* *DieSpec) Evaluate(*x* int) (int, error)
func (*ds* *DieSpec) GetOperator() string
func (*ds* *DieSpec) IsMaxRoll() bool
func (*ds* *DieSpec) IsMinRoll() bool
func (*ds* *DieSpec) LastValue() int
func (*ds* *DieSpec) MaxValue(*x* int) (int, error)
func (*ds* *DieSpec) NaturalRoll() (int, int)
func (*ds* *DieSpec) StructuredDescribeRoll() []StructuredDescription

DESCRIPTION

This module provides a generic die-rolling interface.

There is a low-level **Dice** object which models a set of dice and handles most general random number generation TTRPG games require. A **Dice** object may be constructed using discrete parameters which together describe a set of dice to be rolled:

qty	The number of dice to roll. Their values will be added together.
sides	Number of sides each die has.
bonus	Additional value to add to the sum of all dice rolled.
diebonus	Additional value added to each die individually.
div	If specified, each die result is divided by *div* (fractions truncated).
factor	If specified, this value is multiplied to the final result.

Alternatively, a single string *description* may be passed which describes a more complex range of die roll arrangements. This string may contain any number of nonnegative integer values and/or die-roll expressions separated by the basic math operators "**+**", "**–**", "*****", and "**//**" which respectively add, subtract, multiply, and divide the total value so far with the value that follows the operator. Division performed with the "**//**" operator is integer-only (results are immediately truncated by discarding any fractional part). There is no order of operations or parentheses supported. The expressions are simply evaluated left-to-right as they appear. Generally speaking, whitespace is insignificant in the *description* string.

Each die-roll expression has the general form
[**>**] [*n*[**/***div*]] **d** *sides* [**best**|**worst of** *r*] [*label*]

This calls for *n* dice with the given number of *sides* (which may be a number or the character "**%**" which means percentile dice or d100). The optional *div* part of the expression allows a fractional number of dice: the expression "**1/2d20**" rolls half of a d20 (in other words, it rolls 1d20 and divides the result by 2, truncating down). The optional qualifier "**best of** *r*" will cause the dice to be rolled *r* times, keeping the best result. (You may also use the word **worst** in place of **best** to take the lowest of the rolls.)

Arbitrary text (*label*) may appear at the end of the expression. It is simply reported back in the result as a label to describe that value (e.g. "**1d10 + 1d6 fire + 2d6 sneak**".) If the expression begins with the character "**>**", then the first die in the set is maximized: in the expression "**>3d6**", the first d6 is assumed to have the maximum value (6), and the remaining two dice are rolled to produce random values.

The entire die roll expression may be followed by one or both of the following global modifiers, separated from the expression and each other by vertical bars ("**|**"): "**min** *a*" or "**max** *b*".

These force the final result to be no smaller than *a* and/or no larger than *b*, where *a* and *b* are integer values. For example:
2d6 + 1d4 | min 6
which rolls 2d6 and 1d4 to get a random value between 3 and 16, but if the result is less than 6, it will return 6 anyway.

Regardless of the method used to specify the dice represented by the **Dice** object, it will use a default random number generator such as **random.Random** or the function passed to the **Dice** constructor in the *generator* parameter. This function must have a **randint()** method with the same semantics as the

random.Random generator.

Once the **Dice** object is thus constructed, the following methods may be called on it to roll and examine the dice:

roll(*confirm***=False,** *threat***=None,** *bonus***=None)**
: Roll the dice which this **Dice** instance represents. The result is returned as an integer value. Each time this is called, the dice are rerolled to get a new result. The **Dice** object's internal state reflects the last call to this method.

  If the *confirm* parameter is **True**, then this roll is for the purpose of confirming a critical roll based on the previous roll, which must have involved only a single die. If the previous roll was the maximum value of the die before other modifiers are applied (or was at least the value of *threat* if that was supplied), then another roll is performed and the new result returned. If *bonus* is supplied, it is added to the result as well. If there was no previous roll, an exception is raised.

  If the previous roll did not meet the qualifications above (e.g., it was not a natural 20 on a d20), then no roll is performed. In this case, **None** is returned.

value() Return the value of the last **roll()** of the dice. This is the same value originally returned by **roll()** or **None** if there was no previous roll.

maxvalue()
: Return the maximum possible roll that the dice could produce. This counts as a roll of the dice, so subsequent calls to **value()**, **describe_roll()**, and **structured_describe_roll()** will return or describe the maximized value.

description()
: This returns a string that describes, in plain text, the dice rolling expression the object was constructed to represent. It is close to, but not identical to, the format of the *description* parameter to the constructor.

describe_roll()
: This returns a string that describes, in plain text, the results of the most recent **roll()**.

structured_describe_roll(sf=*None***)**
: This returns a detailed accounting of the results of the most recent **roll()**. If **sf** is provided, it is a sequence of two values. If the die-roll involved a single die and that die came up as a natural 1, the reported description includes **('fail',***sf***[1])** to indicate automatic failure. If the die was naturally the maximum value possible for the die type, **('success',***sf***[0])** to show automatic success. The return value is a *structured_results* list. Each element of the list describes a discrete part of the results as a tuple of **(***type***,** *string***)** where *type* is a word that describes what the *string* value means. It can be used to select an appropriate formatting for the *string*. The possible values for *type* include "**best**", "**bonus**", "**constant**", "**diebonus**", "**diespec**", "**discarded**", "**fail**", "**label**", "**max**", "**maximized**", "**maxroll**", "**min**", "**moddelim**", "**operator**", "**result**", "**roll**", "**separator**", "**success**", and "**worst**". If no roll has yet been made, it returns **None** instead.

Higher-Level Interface

A more comprehensive die-rolling interface is provided by the **DieRoller** class. Once instantiated, a new random number is generated by calling the following method:

do_roll(*dice***='')**

If specified, the *dice* parameter is a string that describes the die-rolling expression in the form:

[*title***=**] *expression* [**|***options*]

or

[*title***=**] *chance***%** [*success*[**/** *fail*]] [**|***options*]

where *expression* is anything that can be given as the *description* parameter to the **Dice** constructor as described above, and *options* is a vertical-bar-delimited set of global options that control how the entire die roll is handled.

A *title* may be given to the roll to document what the roll is for. This is simply reported back with the results and has no other effect.

The global *options* which may be added to the end are separated from each other and from the die roll *expression* with vertical bar ("**|**") characters. They may be any of the following:

min *n*	Result will be at least *n*.
max *n*	Result will be no more than *n*.
c[*t*[±*b*]]	This indicates that the roll may need a critical confirmation roll to follow it. After making the initial (internal) **Dice.roll()** call, another one with *confirm*=**True** is made and, if successful, added to the result returned by this method. By default, the critical threat range is assumed to be the maximum value for the die rolled (e.g., a natural 20 on a d20). If the *t* parameter is given, a natural roll equal to or greater than *t* is assumed to be a threat. If a plus or minus sign followed by a number *b* is given, that value is added to the confirmation die roll. This option automatically implies a "**|sf hit/miss**" option.
DC *n*	This is a roll against a known difficulty class (DC) of *n*. Along with the roll result, it will report whether the DC was met, and if not exactly, the amount by which the roll exceeded or fell short of the DC. (The word "DC" may be in upper- or lowercase.)
sf [*success*[**/***fail*]]	Auto-success/fail: the roll, which must involve only a single die, is automatically considered to succeed if the die rolled to its maximum value (before modifiers are applied), and a failure if it rolled a natural 1. In this case, rather than reporting the result numerically, the string *success* is reported (or "**success**" by default) if automatic success was indicated, or "**failed**" by default) in the event of automatic failure.
until *n*	Roll repeatedly until the result is at least *n*. This can be used for repeated skill checks where you need to know how many times the check failed (and by how much) along the way to success.
repeat *n*	Roll *n* times, reporting each result.
maximized	The maximum possible result of the die roll is given rather than a random number. An exclamation mark ("**!**") may be used in place of the word **maximized**. This option is not guaranteed to work with the **c** or **sf** options.

To prevent getting caught in an infinite loop, a maximum of 100 rolls will be made regardless of **repeat** and **until** options.

Anywhere in the string you may introduce a combination specifier in curly braces as "**{***a***/***b***/***c***/...}**". This will repeat the overall die roll expression once for each of the values *a*, *b*, *c*, etc., substituting each in turn for the braced list. If multiple specifiers appear, they'll all repeat so you get the cartesian product of all the sets of values. This allows, for example, multiple attack rolls in a single click. For example, "**Attack=d20+{17/12/7}**" would roll three attack rolls: d20+17, d20+12, and d20+7.

In the second form, *chance* gives the percentage chance of something occurring, causing percentile dice to be rolled. The result will be a boolean value that is **True** if the d100 roll was less than or equal to *chance*. By default, the result is reported in the detailed description as "success" or "fail". If a *success* label is given in the die-roll string, that is reported in case of a successful roll, and "did not *success*" otherwise. If an explicit *fail* label is also given, that is used for unsuccessful rolls instead. As a special case, if *success* is "miss" then *fail* is assumed to be "hit" and vice versa.

For percentile rolls, only the **until**, **repeat**, and **maximized** global options may be used. Permutations ("**{...}**") are also disallowed with percentile rolls.

If the *dice* string is empty, **do_roll()** re-rolls the previously-specified die-roll expression. If no *dice* string was ever given, the **DieRoller** defaults to rolling 1d20.

This method returns a tuple of values (*title***,** *result_set*) representing the results of rolling the dice. The *title* is the title specified in the *dice* string, or **None** if one was not given. The *result_set* is a list of results, one for each roll of the dice that was performed. Each element of *result_set* is a tuple (*result*, *structured_results*) which give the integer result of that die roll, and a structured description of it as described for the **Dice.structured_describe_roll()** method, with the additional *type* values of "**critlabel**", "**critspec**", "**dc**", "**exceeded**", "**fail**", "**fullmax**", "**iteration**", "**met**", "**repeat**", "**short**", "**sf**", "**success**", and "**until**".

to_text(*structured_results***,** *label***=None,** *formatting***=None)**

This renders the *structured_results* list returned by the **Dice.structured_describe_roll()** and **DieRoller.do_roll()** methods into a string according to the formatting rules given by *formatting* (or simply producing a simple plain text version if no *formatting* value is given).
If *label* is given, it is the overall label (called title elsewhere in this document) for the die roll.

The *formatting* dictionary maps *type* names as returned by the above-named methods to format strings such as would be accepted by the **str.format()** method, with the corresponding *value* provided as the only positional parameter. Thus, the value in *formatting***[***type***]** is a string which describes a *value* of that *type*, with a "**{}**" or "**{0}**" at the place(s) in the string where the actual *value* string should go (see **str.format()** for details).

Interactive Usage

The module's **main()** function may be called to enter an interactive loop. The user is prompted for a die roll, that is fed into **DieRoller.do_roll()**, and the process repeats until the user enters an EOF, or one of the strings "**exit**", "**quit**", or "**q**". If command-line parameters are found in *argv*, it rolls each dice string in that list instead. If *verbose* is **True**, then the raw data structures returned by the underlying API functions are printed as well.

GO API VARIANT

While the GMA system is, generally, implemented in Python, and the majority of this document describes the Python API for the die-rolling facility within GMA, there also exists a Go language port of the die-rolling code, created to support a Go version of the server.

This section documents the Go interface so far as it differs from what is described elsewhere in this document.

As with the Python version, there is a high-level interface via the **DieRoller** type, which accepts a die-roll specification string in the same format as the Python API, which should be compatible with the way humans describe dice rolls when playing games. A new **DieRoller** value is constructed by calling **NewDieRoller()** and then its **DoRoll()** method each time a new roll is required. The **DoRoll()** method defaults initially to "1d20" until a non-empty *spec* string is passed to it. If *spec* is an empty string, it will roll again the last-known die-roll specification.

This method returns three values: *title*, *results*, and *error*. If *error* is non-**nil**, something went wrong and none of the other values are defined. Otherwise, *title* gives the user-supplied title for the die roll (if any), and *results* is a list of results of the dice rolled. There may be more than one result if the *spec* called for multiple rolls (e.g. if there are permutations in the specification or if a critical roll confirmation was attempted). Each result is of type **StructuredResult** which contains an integer **Result** value giving the total result of the roll. In the case of percentile check rolls, this will be 1 if the roll indicated success or 0 if it indicated failure. The other value in that structure is **Details**, which is a slice of **StructuredDescription** values. Each of these gives one detail about what went into the result, as documented below.

There is also, as with the Python API, a lower-level representation of a dice roll in the form of the **Dice** value type. A **Dice** value may be constructed using a *description* string (as with the Python version, this is a simplified subset of the **DieRoller**'s *spec* string), as in:

dice, err := NewDice("2d6+5")

There are also parameterized constructors available. The simpler is

dice, err := NewDiceBasic(*qty***,** *sides***,** *bonus***)**

where *qty* is the number of dice to roll (and add together), *sides* is the number of sides on each die, and *bonus* is an amount to add to the results of the dice. Thus, the previous constructor example is equivalent to calling **NewDiceBasic(2, 6, 5)**.

For greater control over the dice to be rolled, the fully parameterized constructor is

dice, err := NewDiceByParameters(*qty***,** *sides***,** *bonus***,** *diebonus***,** *div***,** *factor***)**

This allows the following additional parameters beyond those supported by **NewDiceBasic**:

diebonus	An additional amount to add to each individual die rolled.
div	If nonzero, a value by which to divide the result of each die roll.
factor	If nonzero, a value by which to multiple the final result.

Once constructed, calling *dice***.Roll()** rolls the dice and returns the integer result of the roll, or an error if something went wrong. Alternatively, calling *dice***.MaxRoll()** returns the maximum possible result, just as though all dice happened to turn up with their maximum face values.

Calling *dice***.Description()** returns a text string describing in human-readable terms what dice will be rolled, in a manner similar to (but not guaranteed to be exactly matching) the *description* parameter that may be passed to the **NewDice()** constructor.

If a critical roll is threatened, a confirmation roll is needed, which may have its own bonus added to (or subtracted from) the total. This may be performed by calling *dice***.RollToConfirm(true,** *threat***,** *bonus)* after the initial (possibly critical) roll was made with a previous call to *dice***.Roll()**. (Calling *dice***.RollToConfirm()** with a **false** initial parameter is equivalent to calling *dice***.Roll()**.) This will check to see if the previous roll qualified as a critical threat based on the *threat* value. If not, 0 is returned as the result. Otherwise, a new roll will be made just as if with the *dice***.Roll()** method, with the additional *bonus* value added to this roll.

There is a corresponding *dice***.MaxRollToConfirm(***bonus***)** call which works identically to *dice***.RollToConfirm()** but assumes all dice rolled their maximum values (as with the *dice***.MaxRoll()** method). This also implies that no *threat* parameter is needed since the previous roll was maximized and by definition would threaten a critical roll.

Once a roll has been made by one of the above methods, calling *dice***.StructuredDescribeRoll()** returns a slice of **StructuredDescription** values, together describing all of the details behind how the total result was obtained.

STRUCTURED RESULTS

Some methods return a structured result list to describe the outcome of the die roll. This is a human-readable text description of the rolls made, broken into pieces to facilitate easier formatting. Each element of the structured results list is a **(***type***,** *value***)** tuple. The *type* explains how to format the corresponding *value*, and may be one of the following:

best	The specifier to reroll and take the best of the dice rolled. The value is just the number of die rolls to perform.
bonus	Overall bonus applied to the entire roll but not included in the expression (e.g., the bonus added to confirm critical rolls).
constant	A constant value being added (or whatever) to the overall total.
critlabel	Label automatically given to confirmation rolls.
critspec	Specifier for confirming critical rolls.
dc	The difficulty class for this roll.
diebonus	Bonus value added to each die in an individual diespec (not the overall total).
diespec	The specification of a single die roll component, such as "**12d6**".
discarded	Results of a die roll that was made but discarded due to another rule, as a comma-separated list of numbers that were rolled.
exceeded	The amount by which the roll exceeded the DC.
fail	The result of a rolled percentile check roll that indicated failure.

fullmax	The option "**maximized**" which requests all dice to be maximized.
iteration	The roll number of the current iteration of rolls for repeating sets of rolls.
label	Arbitrary text label attached to a die expression (e.g., the "**fire**" in "**1d6 fire**").
max	The maximum value allowed for the die roll.
maximized	The prefix "**>**" which requests a maximized first die in the roll.
maxroll	Same as **roll**, but all die rolls were forced to be maximized.
met	An indicator that the DC was met exactly.
min	The minimum value for the die roll.
moddelim	Delimiters which separate global modifiers from the other parts of the expression.
operator	One of the operators ("**+**", "**−**", etc.) which separates each die roll expression.
repeat	The number of times the roll is being repeated.
result	The full result of the die roll.
roll	The results of rolling a single diespec, as a comma-separated list of numbers representing the side that rolled for each die in the diespec.
separator	Separators such as the "**=**" between the title and the rest of the expression.
short	The amount by which the roll fell short of the DC.
success	The result of a rolled percentile check roll that indicated success.
until	The value we are rolling repeatedly until we reach.
worst	The specifier to reroll and take the worst of the dice rolled. The value is the number of times to try rolling.

For example, calling **DieRoller.do_roll('1d20+3 | min 5 | c')** might result in an initial natural 20 (giving an overall result of 23). But since that was a natural 20, a second confirmation roll is made (due to the "**c**" at the end of the expression), with a result of 13. In this case, the value returned from **do_roll()** is:

```
(None, [
        (23, [
                ('result', '23'),
                ('success', 'HIT'),
                ('separator', '='),
                ('diespec', '1d20'),
                ('roll', '20'),
                ('operator', '+'),
                ('constant', '3'),
                ('moddelim', '|'),
                ('min', '5'),
                ('moddelim', '|'),
                ('critspec', 'c')
        ]),
        (13, [
                ('critlabel', 'Confirm:'),
                ('result', '13'),
                ('separator', '='),
                ('diespec', '1d20'),
                ('roll', '10'),
                ('operator', '+'),
                ('constant', '3'),
                ('moddelim', '|'),
                ('min', '5')
        ])
    ])
```

Likewise, calling **DieRoller.do_roll('Attack=d20+{17/12}+3')** might return:

```
    ('Attack', [
        (38, [
            ('result', '38'),
            ('separator', '='),
            ('diespec', '1d20'),
            ('roll', '18'),
            ('operator', '+'),
            ('constant', '17'),
            ('operator', '+'),
            ('constant', '3')
        ]),
        (35, [
            ('result', '35'),
            ('separator', '='),
            ('diespec', '1d20'),
            ('roll', '20'),
            ('operator', '+'),
            ('constant', '12'),
            ('operator', '+'),
            ('constant', '3')
        ])
    ]
```

DIAGNOSTICS

These functions may raise the following exceptions in addition to the normal set that may occur otherwise:

InvalidDiceDescription
The specification of the dice to roll was not understandable as written.

InvalidConfirmUsage
A confirmation was requested for a die roll specification that could not apply to.

SEE ALSO

AUTHOR

Steve Willoughby / steve@alchemy.com.

HISTORY

The Go port appeared in November, 2020, in GMA version 4.2.2.

BUGS

COPYRGHT