# AutomatedDices - Code Documentation

**Made and documented by Lazare**
**Current version : 1.0**
**Last update : 10th of January 2023**

# Before starting the script

First of all, thank you for downloading, testing, and using this piece of code. I know a lot of other people tried to make an algorithm to create such dice, but this one is free, and anybody can use it.

I know not everybody will read this documentation, but RTFM will be needed if you wish to properly use the main code.

To make the project work, simply download the archive of the project, unzip it and run the Blender file called "script". Please refer to the Enabling function section a bit later in this documentation to have everything ready. Everything goes at the end of the code, on the main function.

This project has been tested and developed on a version 3 of Blender. Any previous versions might not work properly. Any retrofit of the version will not be done. Please, update your Blender version before trying the project.

## What is the purpose of this Blender Python Script ?

As these are not the first dice I make on Blender. I found this YouTube video by Cly Faker that is a gold mine to start your first dice modeling :
▶ Design Your Own Custom Dice for Free | Blender Tutorial | DIY with Cly Ep. 21
Big shootout to him, even if I never dared contact him.

At first, I wanted to try Python, as I never coded in this language. This code is not optimized, neither is the Blender integration, but it works quite correctly. Some further automations are planned, and you will be noticed if you follow me on Twitter. They are explained in the next point.

## To-do list

This project is not finished yet, there are several new functions that need to be added such as :
- All scaling in millimeters (as much as possible)
- Full symbol dice (replacing all numbers by symbols)

No Blender incorporation in the UI is provided nor will be provided at this point in time. Too much work for something hypothetical.

Not all functions are explained yet. This documentation will be continued as soon as I finish the project. Might take some time anyway 🙂

# Enabling functions

Before doing anything, enable some functions in Blender to get Platonic shapes access :
- Go to Edit > Preferences > Addons
- Type : "Extra" in the search bar
- Check : "Add Mesh : Extra Objects"

Do not forget to Apply the changes and you are ready to go !

# Sizing the outputs

This project is based on the hypothesis of dice functionalities. They have not been all tested as I do not own a 3D printer or I need to find a cheaper way than having to go with big 3D printing companies.

Also, Blender tends to use meters as a standard unit. With the STL file conversion, every measurement is back in millimeters.

Any feedback is appreciated on the standard scaling of the dice. They are a bit bigger than the standard dice you may find with Chessex or any other dice-making companies. Changing the scale to the appropriate size may be helpful.

# Possible legal issues

If you plan to use this code to generate and sell dice, feel free to contact me, it's a great pleasure to know that this work is in good hands. But I must inform you that using protected TrueType Fonts may give you problems if you plan to sell these dice. Shapes are not a problem, but you can face a legal procedure if you do not pay attention to the license of the font.

Also, there are some DIY molds out there that use some special fonts. Please, do not use this algorithm to generate an already made dice.

# Generation time and STL size

It appeared to me that some fonts are heavily detailed. The more vectors the font has, the more time Blender will need to generate all dice. It can be really significant if you have a small GPU. Do not be scared that Blender seems to crash, it is thinking.

If you want to trace where the algorithm is, you can go in Blender, go in Window > Toggle System Console. I use this text output to know where the algorithm is. It does not have a chronometer though. Not useful but can be fun !

Also, time and STL size are linked. More detailed fonts generated more detailed STL files that can go up to 30MB. There is no optimization on the STL file yet, and if you have some thoughts about it, again, feel free to contact me, we will discuss them !

# Versions & Updates

0.1 - July 23rd 2022
- Initial commit
- D4, D6, D8 & D10 (unit & tens) available

0.2 - July 23rd 2022
- Added D12 & D20

0.3 - July 24th 2022
- Prism dice available

0.4 - July 25th 2022
- Antiprism dice available

0.5 - August 12th 2022
- Automatic dice number sequence added to the algorithm
- Trapezoid dice available
- Text centered on dice

0.5.1 - August 13th 2022
- Minor updates on trapezoid dice text placement

0.6 - August 15th 2022
- All text meshes are centered using the "0" as reference point
- Trapezoid and Antiprism text position has been modified to better suit the dice
- Text imprinting size is now available on parameter
- Text size and scale on Platonic dice is now parametrable
- Mesh alteration is possible (only Bevel for the moment)
- Face merging algorithm for D12
- Initialization sequence starts by clearing all meshed on the Blender scene
- Some function and variables had been renamed

0.7 - August 28th 2022
- Added Rounded Cubes for D2 and D4
- Added Rhombic D12
- Added Kiscube D24

0.7.1 - September 17th 2022
- Added D30, D60, D48
- Example dice resizing
- Exports are now available

0.8 - September 21st 2022
- Added shard dice
- Added dot, underscore methods for 6 and 9 distinction

0.8.1 - December 4th 2022
- File organization modified to match the project use

0.8.2 - December 29th 2022
- Added documentation
- Minor typo corrections on code

1.0 - January 10th 2023
- Documentation updated
- Antiprism & Trapezoid dice text position corrections
- Text automatic resizing
- Text orientation available on parameter dice
- Symbol can be added on one side on any dice
- Better bevel method
- Corrections on Rounded Cube & Pseudo Antiprism
- Examples update

# Main Functions

All the functions explained here are in use in the main function of the program, which you can find at the bottom of the code.

If you want detailed information on how the functions work, please refer to the code comments that are more explanatory than this simple document.

## Initialization functions

These functions are made to manipulate the global parameters of the Blender scene, and assign parameters for the numbers generation.

### initializeBlenderSceneObjects()

Cleans the Blender scene. Does nothing else. Please leave it.

### importLogoFromSTLFile(STLFileName)

Allow the code to use a STL file as a logo to be used on faces. STLFileName is the name of the file (without the extension) of the selected logo. The file must be placed in the /logo folder on the project.
The logo is resized within this function to match the algorithm.

### setFont(Font)

Allow the user to load a font for the project. Font is the name of the file you want to use with the numbers printed on dice. This file must be placed in the /font folder. The extension must be added to the file name.

As far as I tested, .ttf and .otf files are accepted by Blender to be used.

### setDotSign(dotPosition, dotSign)

Change the 6 and 9 marker symbols for dice that have more than 8 faces.
dotPosition allows the position of the dot/underscore sign under the number.
-   -0.20 is correct for the dot sign
-   -0.05 is correct for the underscore sign
3 signs are admitted for the parameter : dotSign.
-   "" (empty string) : 6 and 9 are set as 6. and 9. like the classic Chessex method
-   "." (dot) : put the dot sign under the 6 and 9 as marker
-   "_" (underscore) : put an underscore sign under the 6 and 9 as marker

Warning : The dot and underscore still have problems on this version of the project. When changing the font, the dot or underscore can be badly placed, causing these signs to be too close or far from the number.

## generateAllNumbersMeshes(maxNumber)

Generate all meshes for the numbers to be used by the code, from 0 to maxNumber. It is set as 100 in the code, but it depends on what is the maximum side you need. Floating point numbers are not allowed by the function.
You do not need to handle everything here, let the code drive you.


## meshBevelGlobalParameters(activation, amount, segments, shape, spread, method)

This function allows the user to set the Bevel method on the dice. Text imprint is not modified by the bevel.
- activation (boolean) : activate the Bevel method (True/False)
- amount : size of the bevel on the edges
- segments : how many sub-polygons the bevel method generates
- shape : percentage (between 0.5 and 1) of the bevel angle
- spread : percentage (beween 0 and 1) of the bevel arc between vertices and edges
- method : method of Bevel generation
    - 'EDGES' : Bevel on edges
    - 'VERTICES' : Bevel on vertices

# Standard/Platonic Dice Generators

## dice4Generator(scale, textSize, replaceLogo, textDepth)

This function generates a classic tetrahedron D4 with numbers read on the top vertex.
- scale : size of the tetrahedron. Scale at 15 creates a 20mm tall, 24.5mm sided tetrahedron
- textSize : size of the text used for numbers
- replaceLogo : logo replaces the side on replaceLogo. Put 0 or 99 if you want no logo replacement
- textDepth : depth of the text imprint. 0.5 is recommended.

Output : Object D4 in Blender scene

## dice6Generator(scale, textSize, replaceLogo, textDepth)

This function generates a classic cube D6.
- scale : size of the cube. Scale at 8 creates a 16mm cube.
- textSize : size of the text used for numbers
- replaceLogo : logo replaces the side on replaceLogo. Put 0 or 99 if you want no logo replacement
- textDepth : depth of the text imprint. 0.5 is recommended.

Output : Object D6 in Blender scene

## dice8Generator(scale, textSize, replaceLogo, textDepth)

This function generates a classic octahedron D8.
- scale : size of the octahedron. Scale at 15 creates a 21.2mm base octahedron with a 30mm height.
- textSize : size of the text used for numbers
- replaceLogo : logo replaces the side on replaceLogo. Put 0 or 99 if you want no logo replacement
- textDepth : depth of the text imprint. 0.5 is recommended.

Output : Object D8 in Blender scene

## dice10UnitGenerator(scale, textSize, replaceLogo, textDepth)

This function generates a classic pentagonal trapezohedron D10. This dice is used for units in a double d10 roll (see dice10DecimalGenerator function for the 10x version)
- scale : size of the trapezohedron. Scale at 13 creates a 13.6mm x 19.1mm base face for the trapezohedron. Total height of the shape is 29.3mm.
- textSize : size of the text used for numbers
- replaceLogo : logo replaces the side on replaceLogo. Put 0 or 99 if you want no logo replacement
- textDepth : depth of the text imprint. 0.5 is recommended.

Output : Object D10u in Blender scene

## dice10DecimalGenerator(scale, textSize, replaceLogo, textDepth)

This function generates a classic pentagonal trapezohedron D10. This dice is used for tens in a double d10 roll. Numbers are rotated by 90° for readability.

- scale : size of the trapezohedron. Scale at 13 creates a 13.6mm x 19.1mm base face for the trapezohedron. Total height of the shape is 29.3mm.
- textSize : size of the text used for numbers
- replaceLogo : logo replaces the side on replaceLogo. Put 0 or 99 if you want no logo replacement
- textDepth : depth of the text imprint. 0.5 is recommended.

Output : Object D10d in Blender scene

## dice12Generator(scale, textSize, replaceLogo, textDepth)

This function generates a classic dodecahedron D12.

- scale : size of the dodecahedron. Size at 12 creates a 8.56mm faced pentagon to create the dodecahedron. Total height of the dodecahedron is 19.1mm.
- textSize : size of the text used for numbers
- replaceLogo : logo replaces the side on replaceLogo. Put 0 or 99 if you want no logo replacement
- textDepth : depth of the text imprint. 0.5 is recommended.

Output : Object D12 in Blender scene

## dice20Generator(scale, textSize, replaceLogo, textDepth)

This function generated a classic icosahedron D20.

- scale : size of the icosahedron. Size at 14 creates 14.72mm equilateral triangles for the icosahedron face. Total height is 22.3mm.
- textSize : size of the text used for numbers
- replaceLogo : logo replaces the side on replaceLogo. Put 0 or 99 if you want no logo replacement
- textDepth : depth of the text imprint. 0.5 is recommended.

Output : Object D20 in Blender scene

# Parametrable Dice Generators

diceEvenPrismaticGenerator(faces, radius, depth, extrude, scale, textSize, replaceLogo, textDepth)

This function generates a prism-based even-sided polyhedron. Each face is imprinted with the number. Number sequence is automatically generated by the algorithm and can create any number of sides for the dice.

- faces : the number of faces of the polyhedron.
- radius : ratio of the size of the polygon composing the polyhedron
- depth : ratio of the center of the prism
- extrude : ratio of the side extrusions
- scale : size of the prism-based dice
- textSize : size of the text used for numbers
- replaceLogo : logo replaces the side on replaceLogo. Put 0 or 99 if you want no logo replacement
- textDepth : depth of the text imprint

Output : Object D*faces*Prism in Blender scene. *faces* is a numeric value, entered in the parameter faces.

Notices :
- An odd number in faces will result in a function exiting before making the dice. No output will be given.
- Radius, depth and extrude parameters are ratios, expressed as a floating number. 1.0 is equivalent to 100%. You can put any number you want in these parameters. You can use the scale parameter for further adjustments.
- Only D4 is made with this function because antiprism polyhedron works better for even-number prism-based dice. And D4 can not work as antiprism.

## diceOddPrismaticGenerator(faces, radius, depth, extrude, scale, textSize, replaceLogo, textDepth)

This function generates a prism-based odd-sided polyhedron. Each face is imprinted with the number, and read on the upper edge of the dice. Number sequence is automatically generated by the algorithm and can create any number of sides for the dice.

- faces : the number of faces of the polyhedron.
- radius : ratio of the size of the polygon composing the polyhedron
- depth : ratio of the center of the prism
- extrude : ratio of the side extrusions
- scale : size of the prism-based dice
- textSize : size of the text used for numbers
- replaceLogo : logo replaces the side on replaceLogo. Put 0 or 99 if you want no logo replacement
- textDepth : depth of the text imprint

Output : Object D*faces*Prism in Blender scene. *faces* is a numeric value, entered in the parameter faces.

Notices :
- An even number in faces will result in a function exiting before making the dice. No output will be given.
- Radius, depth and extrude parameters are ratios, expressed as a floating number. 1.0 is equivalent to 100%. You can put any number you want in these parameters. You can use the scale parameter for further adjustments.
- Only D3 is made in the example because other dice are unreadable. Moreover, they are ugly as f.

## diceAntiprismaticGenerator(faces, radius, depth, extrude, scale, textSize, textRotation, replaceLogo, textDepth)

This function generates an antiprism-based even-sided polyhedron. Each face is imprinted with the number. Number sequence is automatically generated by the algorithm and can create any number of sides for the dice.

- faces : the number of faces of the polyhedron.
- radius : ratio of the size of the polygon composing the polyhedron
- depth : ratio of the center of the prism
- extrude : ratio of the side extrusions
- scale : size of the prism-based dice
- textSize : size of the text used for numbers
- textRotation : angle of rotation (in degree) of the text
- replaceLogo : logo replaces the side on replaceLogo. Put 0 or 99 if you want no logo replacement
- textDepth : depth of the text imprint

Output : Object D*faces*Antiprism in Blender scene. *faces* is a numeric value, entered in the parameter faces.

Notices :
- An odd number in faces will result in a function exiting before making the dice. No output will be given.
- Radius, depth and extrude parameters are ratios, expressed as a floating number. 1.0 is equivalent to 100%. You can put any number you want in these parameters. You can use the scale parameter for further adjustments.
- D4 can not be created with this method, see dice4PseudoAntiprismaticGenerator.

diceBipyramidGenerator(faces, radius, depth, scale, textSize, textRotation, replaceLogo, textDepth)

This function generates a n-gonal trapezohedron dice. This polyhedron is even-sided. Each face is imprinted with the number. Number sequence is automatically generated by the algorithm and can create any number of sides for the dice.

- faces : the number of faces of the polyhedron.
- radius : ratio of the size of the polygon composing the center polyhedron
- depth : ratio of the center of the prism
- scale : size of the prism-based dice
- textSize : size of the text used for numbers
- textRotation : angle of rotation (in degree) of the text
- replaceLogo : logo replaces the side on replaceLogo. Put 0 or 99 if you want no logo replacement
- textDepth : depth of the text imprint

Output : Object D*faces*Bipyramid in Blender scene. *faces* is a numeric value, entered in the parameter faces.

Notices :
- An odd number in faces will result in a function exiting before making the dice. No output will be given.
- Radius and depth parameters are ratios, expressed as a floating number. 1.0 is equivalent to 100%. You can put any number you want in these parameters. You can use the scale parameter for further adjustments.
- D4 can not be created with this method.
- D8 and D10 are very similar to the classic dice.

## diceShardGenerator(faces, radius, depth, depthTopRatio, scale, textSize, replaceLogo, textDepth)

This function generates a n-gonal asymmetrical trapezohedron dice. Only the upper faces are imprinted with the number sequence. Number sequence is automatically generated by the algorithm and can create any number of sides for the dice.

- faces : the number of faces of the polyhedron.
- radius : ratio of the size of the polygon composing the center polyhedron
- depth : ratio of the center of the prism
- depthTopRatio : ratio between the global depth of the polyhedron. depthTopRatio should be inferior to 1.0 to work better.
- scale : size of the prism-based dice
- textSize : size of the text used for numbers
- replaceLogo : logo replaces the side on replaceLogo. Put 0 or 99 if you want no logo replacement
- textDepth : depth of the text imprint

Output : Object D*faces*Prism in Blender scene. *faces* is a numeric value, entered in the parameter faces.

Notices :
- Any number of faces will work
- Radius and depth parameters are ratios, expressed as a floating number. 1.0 is equivalent to 100%. You can put any number you want in these parameters. You can use the scale parameter for further adjustments.
- D2 and D3 can not be created with this method.

# Fixed/Alternative Dice Generators

All the dice of the section are alternate versions or unusual dice. Even if most of them are correct for the shape and number placement, scaling has not been tested yet.

## diceRoundedCubeGenerator(faces, roundVertices, scale, textSize, replaceLogo, textDepth)

This function generates a "rounded" cube dice. The generation is based on an hypothetical shape with 4 faces and 2 opposite sides.

- faces : Only 2 or 4. Number of maximum faces on the dice.
    - 2 : Opposite sides will have the same number
    - 4 : Each remaining sides will be composed as a D4
- roundVertices : Number of sub vertices on the rounded side
- scale : size of the prism-based dice
- textSize : size of the text used for numbers
- replaceLogo : logo replaces the side on replaceLogo. Put 0 or 99 if you want no logo replacement
- textDepth : depth of the text imprint

Output : Object D*faces*RoundedCube in Blender scene. *faces* is a numeric value, entered in the parameter faces.

Notice : This idea must be tested to check the equiprobability of the dice.

## dice4PseudoAntiprismaticGenerator(length, minWidth, maxWidth, scale, textSize, replaceLogo, textDepth)

This function generates a pseudo antiprismatic D4 dice. As this shape is not possible with the main antiprism method, this allows the user to have a correct version.

- length : Length ratio of the dice
- minWidth : ratio size of the rectangle minimum width
- maxWidth : ratio size of the rectangle maximum width
- scale : size of the prism-based dice
- textSize : size of the text used for numbers
- replaceLogo : logo replaces the side on replaceLogo. Put 0 or 99 if you want no logo replacement
- textDepth : depth of the text imprint

Output : Object D4PseudoAntiprism in Blender scene.

## dice12RhombicGenerator(scale, textSize, replaceLogo, textDepth)

This function generates a rhombic D12 dice. This is an alternative version of the dodecahedron. Numbers are generated using an existing rhombic D12.

- scale : size of the prism-based dice
- textSize : size of the text used for numbers
- replaceLogo : logo replaces the side on replaceLogo. Put 0 or 99 if you want no logo replacement
- textDepth : depth of the text imprint

Output : Object D12Rhombic in Blender scene.

## dice24TetrakisGenerator(scale, textSize, replaceLogo, textDepth)

This function generates a tetrakis hexahedron D24 dice. This is an alternative version of the dodecahedron. Numbers are generated using an existing kiscube D24.

- scale : size of the prism-based dice
- textSize : size of the text used for numbers
- replaceLogo : logo replaces the side on replaceLogo. Put 0 or 99 if you want no logo replacement
- textDepth : depth of the text imprint

Output : Object D24Tetrakis in Blender scene

## dice30RhombicGenerator(scale, textSize, replaceLogo, textDepth)

This function generates a rhombic triacontahedron D30 dice. Numbers are generated using an existing D30.

- scale : size of the prism-based dice
- textSize : size of the text used for numbers
- replaceLogo : logo replaces the side on replaceLogo. Put 0 or 99 if you want no logo replacement
- textDepth : depth of the text imprint

Output : Object D30Rhombic on Blender scene

## dice60PentakisGenerator(scale, textSize, replaceLogo, textDepth)

This function generates a pentakis dodecahedron D60 dice. Numbers are generated using an existing D60.

- scale : size of the prism-based dice
- textSize : size of the text used for numbers
- replaceLogo : logo replaces the side on replaceLogo. Put 0 or 99 if you want no logo replacement
- textDepth : depth of the text imprint

Output : Object D60Pentakis on Blender scene

## dice48DisdyakisDodecahedron(scale, textSize, replaceLogo, textDepth)

This function generates a disdyakis dodecahedron D48 dice. Numbers are generated using an existing D48.

- scale : size of the prism-based dice
- textSize : size of the text used for numbers
- replaceLogo : logo replaces the side on replaceLogo. Put 0 or 99 if you want no logo replacement
- textDepth : depth of the text imprint

Output : Object D48Disdyakis on Blender scene

## Finalizing functions

### clearAllNumbersMeshes()

After the dice generation, this method clears all generated text symbols used by the program to make the imprint.

### exportDiceToSTLFile()

Exports all generated dice into STL files. These exports will be found on the /export folder.