Table of Contents

[Introduction 2](#_Toc7790532)

[Goals and concepts 2](#_Toc7790533)

[Goal 2](#_Toc7790534)

[Games and Programs with Creature generation 2](#_Toc7790535)

[No Mans Sky 2](#_Toc7790536)

[Spore 3](#_Toc7790537)

[Solutions 4](#_Toc7790538)

[Poisson disc sampling: 4](#_Toc7790539)

[Superformula: 5](#_Toc7790540)

[Divago system: 6](#_Toc7790541)

[Variational auto encoder: 9](#_Toc7790542)

[Lindenmayer system: 9](#_Toc7790543)

[Metaballs: 10](#_Toc7790544)

[Rigblocks 10](#_Toc7790545)

[My approach 11](#_Toc7790546)

[Project result 12](#_Toc7790547)

[How to use 12](#_Toc7790548)

[Overall flow 12](#_Toc7790549)

[Lindenmayer System 12](#_Toc7790550)

[Command executer 14](#_Toc7790551)

[Metaballs 14](#_Toc7790552)

[GUI 15](#_Toc7790553)

[Post mortem 15](#_Toc7790554)

[Problems 15](#_Toc7790555)

[Changes from plan 17](#_Toc7790556)

[Was the approach good? 20](#_Toc7790557)

[Ideas and improvements for the future 20](#_Toc7790558)

[Conclusion 20](#_Toc7790559)

[References 21](#_Toc7790560)

[Github 21](#_Toc7790561)

[Text 21](#_Toc7790562)

[Images 22](#_Toc7790563)

[Model for GUI prototype 23](#_Toc7790564)

# Introduction

# Goals and concepts

## Goal

The minimum goal of this project is that the user can set the settings and then a monster, with basic body parts, as heads, arms and legs merged into a single mesh, will be generated. Furthermore, the user will have the ability to export the monster model so that they can use it outside of this project, for example in a 3D modelling software. When I reached the basic goal and still have time remaining, I will consider implementing other features like additional body parts or further control for the user on the creature generation inside the editor.

Procedural generation takes a big part of the video game industry as seen in their big role in genres like roguelike or endless runners. While the generation of worlds and levels are the main purpose of procedural generation in games, the generation of creatures are rarely seen. Only a few games like No Man’s Sky tackle this issue. My intention behind this project is to explore possibilities in the creation of creatures as I can see many benefits it can have for the video game industry.

One advantage is that artists can use these generated models as a start to create unique monsters or even inspire them to create ones on their own. Moreover, it can help to reduce the time and cost behind developing the creatures as they can be generated instead of being modelled from the ground. The procedural generation of creatures can be also interesting for the players as it can motivate them to explore the world as they can see more of these unique creatures. One example for the last advantage can be seen in the mentioned game No Man’s Sky as some players like to spend more time exploring due to these vastly different creatures.

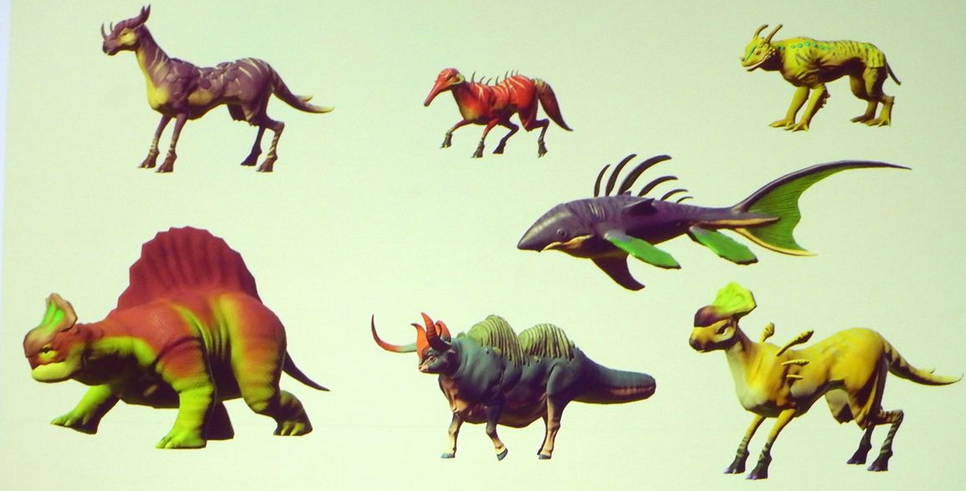
## Games and Programs with Creature generation

### No Mans Sky

No Mans Sky was developed by Hello Games and was released in the year 2016. The core aspect of this game is to explore different planets with their unique floras and faunas. The whole universe together with the planets and their creatures and plants are procedurally generated and consists of over 18 quintillion planets [Wikipedia].

It was rumored that No Mans Sky was using the patented Superformula algorithm by Johan Gielis without the consent of the owner of the patent, which was declared as false by Sean Murray, the founder of Hello Games [ <https://www.forbes.com/sites/jvchamary/2016/07/27/no-mans-sky-superformula/> ].

Other than this rumor I unfortunately could not find out about their algorithms to accomplish the procedural generation of the creatures.

 [56]

*A collection of creatures generated in No Mans Sky*

### Spore

Spore is a videogame created by the studio Maxis and published by EA Games in the year 2008 for the PC and can be assigned to the “God game” genre. This game allows the player to control the development of a species from its beginnings as a microscopic organism, through development as an intelligent and social creature, to interstellar exploration as a spacefaring culture. [Wikipedia]  
Although there is no generation of creatures as the creatures not controlled by the player were created beforehand, there are many interesting concepts inside their creature editor which can be used for the generation of such creatures.

 [33]

*A collection of user created creatures*

Inside the creature generator the player starts with the body of the creature. Spore uses metaballs to display the body of the creature resulting in an easy way to modify the body. If you want to extend the body, you can add a metaball and because of their merging nature you do not have to worry about the combination of the meshes.

 [34]

*The body of a creature*

After you are happy with the body of the creature you can choose between multiple body parts and add them to the creature. The body parts are Rigblocks which means that the user can change the properties of the model by playing with the handles. This results in an intuitive method to change the body parts and therefore have multiple variations of the same body part.

 [35]

*On the left are the available body parts. On the right you can see the handles of the body part*

## Solutions

### Poisson disc sampling:

Poisson disc sampling is an algorithm to generate multiple objects which have a minimal distance between each other, so that they do not collide.

Instead of just spawning the object randomly and then going through every existing object to check if they are colliding, we implement a grid system. The objects now have the length of a cell’s diagonal as their radius so that it is guaranteed that they fill the cell despite their position inside the cell.

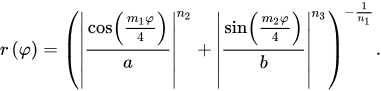
Furthermore, because of their radius size we only have to consider a 5x5 area with the cell as its center, which helps vastly to improve the efficiency (Sebastian Lague 2019).

This easily implemented algorithm can be used for the project to place the body parts on the body without fearing that they collide with each other.

On the other side this algorithm does not help with the problems of creating the body parts and the combination of these into a single mesh. Moreover, we have to constraint the places where the body parts can spawn as this can result into strange creatures, for example the feet could be spawned on the upper part of the body where they would be useless.

### Superformula:

The superformula is an algorithm, proposed by Johan Gielis around 2000. This formula is a combination of the equations for spheres, eclipses and the superellipse and can be used to create different shapes.



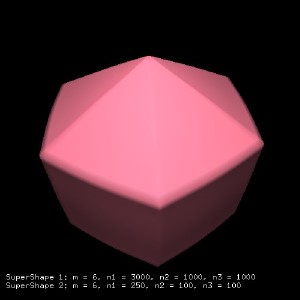
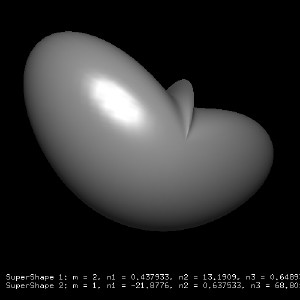
The values of the parameters a, b, m1, m2, n1 and n2 can be changed to achieve different results. The m parameter is responsible for the rotational symmetry while a and b control the size. The other three parameters control the curvature of the sides [9]. As this equation can only have two dimensional results, we have to multiply two different superformulas as following:

https://lh6.googleusercontent.com/E3NqWr5N5_BkHinwp3w2GyZdWQoppK_JrTIFerCeEskcosByyP_KaLJrR6OZyxRrVSg8EZK3VDSlqWOYXoCMIW8pPt3uW9K8AiAnC9Las8J-8l1dirZF34LW8oYBKwRWClh5SnLQ

https://lh5.googleusercontent.com/7OuOEsbnZYKZORiA_0ixXMmsqz12HXkkAuW-JAjNKMwRAYg63lUKtU_vhffcB_0JoT70V-9xKeUBEvPuwt3DuG7WI79yqv0S62LTpM2RzvsAvjxtkZrv4ovUAVZa78hI662b1-aV

https://lh4.googleusercontent.com/2o1F4hsFcBpSPUL6Uf-82np71QJioRwLA5YFsrA-rFdIQ_auA_ZpDIdr8hoS98wbKgZNEKrwuiv5xhg80Mlzb9cxAssgb0hpbIn9nL8cwEbqKxdA-Fxf9Dq9jAHBL6lVc7tM6dJJ

The https://lh6.googleusercontent.com/bqkLQ0L0-MMBGufaJ1TZnIaHhbDSPmwuWMnT_0bS5j5H0nIzwzgFdmP8wxDc5njYOtQWyxocTq_TfA-klfOF3rRs3AtTEwHZ6OVLzFa-Xl8Ear6yXw1jyU-fGqaaR3UvR6Q9EXvq can only have values between -*π*/2 and *π*/2 while *θ* stays between -*π* and *π* (Wikipedia 2019)*.*

 [10]  [11]

*Examples of Supershapes in 3D.*

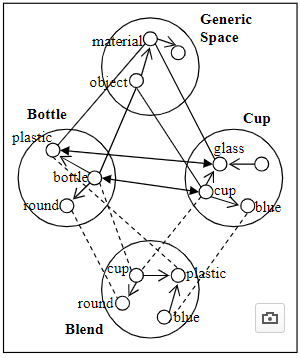
This formula could generate different shapes to be used as the body or for the body parts. Also, it is easy to use because you only have to change the values to have different shapes.

On the contrary it could generate shapes which are not usable to use, and it could be hard to find the usable shapes. Furthermore, there is an active patent on this formula so that I cannot use it for this project.

### Divago system:

The goal of the Divago system is based on conceptual blending and is used to create a unique object by blending two different objects using knowledge gained by previous blends.

Conceptual blending uses mental spaces which are knowledge structures regarding an object. We can bend two mental spaces by mapping different elements of the mental spaces with each other. Following we have an example of the mental spaces and how the mappings can look like.

 [11]

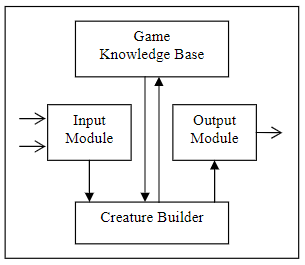
*Diagram for Conceptual blending*

Divago system works with two input domains and a generic space domain similar to conceptual blending which will contain the knowledge gained by previous blends.

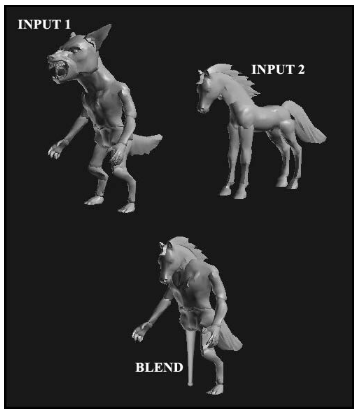
There are four phases in the divago system: mapping phase, projection phase, elaboration phase and the validation phase.

The mapping phase will start the mapping between the two inputs with the help of the domains. After the mappings in the projection phase the blending will begin by searching for the optimal blend bases on the mappings. The next step is the elaboration phase where the blend is refined and altered with the knowledge of the domains. At last the blend will be tested in the validation phase if it does not violate any rules.

Ribeiro et al(2003) uses following architecture for their creature generator described in their article:

[12]

*Architecture of a creture generator using divago system*

[13]

*Example output*

One disadvantage of this approach is that you need a few models beforehand as without the required inputs this system does not work. As many learning algorithms it can take many data example before getting good results. Furthermore, the variations in the resulted blend can heavily be influenced by the differences between the inputs, which means that the data examples should have many differences to have varying results.

On the other hand, once this system is trained the results could be very promising as it learns to understand how the creature should be generated.

### Variational auto encoder:

Variational auto encoders are being used to generate an image based on the knowledge of previous images.

This system consists of the encoder, the decoder and the loss function. The encoder is a neural network where the input will be encoded to a result with lower dimensions than the input, which is often referred as the bottleneck. To achieve this result the encoder has to learn to efficiently compress the input so that the output can be used for the decoder.

Like the encoder the decoder is also a neural network. Inside the decoder the results given by the encoder is being used to reconstruct the original input. As it goes from a smaller to a bigger dimension again there will be a loss of information.

To learn with every iteration there is also a loss function to calculate the difference between the input and the result given by the decoder. The formula for this calculation is the following:



The first term is called the reconstruction loss where we calculate if the decoder reconstructed the image well enough. The second part is a regularizer using Kullback-Leibler divergence to calculate the loss between input given and the result from the encoder. To improve the next iteration, we use the data gained from the loss function to update the parameters of the encoder and decoder(Altosaar, 2019) .

As with many learning algorithms the problem of this approach is that we need a large dataset to guarantee a satisfying result. Also, it needs an input to create a result which is not fitting for this project since we want to generate a model without the need of a model beforehand.

### Lindenmayer system:

The Lindenmayer system or also known as L-System is a string rewriting algorithm developed by Aristid Lindenmayer in 1968.

You start with a base string and with each recursion you replace the chars of the string based on the rules you set beforehand.

Following we have an example of this algorithm. We start with the string “E” and the rules are “E = ER” and “R = E” which means that with every recursion we replace the all E’s in the string with “ER” and all R’s will become E’s.

n=0:         E   
            / \  
n=1:       E R

         /| \  
n=2:     E R E          
        /| |   | \  
n=3:   E R E   E R

    / | | | \   |\ \  
n=4: E R E E R  E R E

So, after four iterations our initial string of “E” have become the string “EREERERE”. At the end we will execute operations based on the letter. In this example “E” means “Extrude” and ”R” is “Rotate”.

For our cause a stochastic grammar would be more useful as it can have different results. As an example we can change the rules from the previous example by saying there is a 50% chance of “B = B” and the remaining 50% will result in “B = F”.

An advantage of this algorithm is that it is easy to understand and to implement. Additionally you can easily modify the rules if the results are not satisfying.

The biggest problem with this approach is to find a correct grammar to generate acceptable creatures.

### Metaballs:

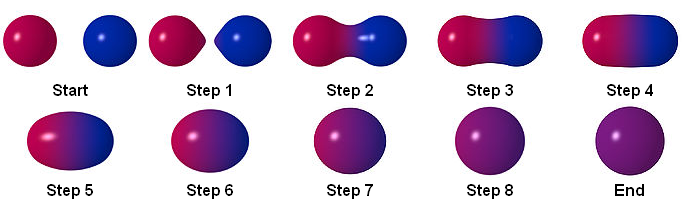
Metaballs are organic objects which will merge with each other if they are nearby.

The metaball can be seen as a particle surrounded by a density field, where the density depends on the distance to the particle position (Kenwright, 2019). Each meatball is defined as a function with a thresholding value to define the solid volume.



To render the metaballs we use the marching cube algorithm, which will generate an approximation of the contour line for the metaballs. By taking eight neighbor locations and therefore creating an imaginary cube we can detect the parts the metaball will fill in the cube. After checking all 8 points with the values of the metaball we can assume the shape of the metaball.

The game Spore uses metaballs inside the creature generator to build the body of the creature, which enables an intuitive way to create the creatures.



*Example of the interaction between two metaballs*

### Rigblocks

Rigblocks are models with components, like the width of the nostrils, which can be modified by the user.

The game „Spore“ uses this system inside their creature editor so that the player can deform and modify the body parts to their liking [3].The developers behind spore used sliders to change the components because they are intuitive for the player to use. Additionally, these models also contain animations, which are being played when the user deforms the model so that there is a fluent transition between the old and new state [8].

[14]

*On the left we see the model with its handles, the following images show possible deformations*

As seen in Spore this system is really helpful in creating quality body parts, which can be deformed and therefore have many variations. Furthermore, this could help to implement features similar to Spore’s creature editor where the player can adjust the creature inside the editor after the generation.

The problem for this system to this project is that the integration of these body parts with the body could be difficult because we have to find a solution to combine the different meshes into a single mesh without strange deformations of the body or the body parts.

## 

## My approach

My approach to that project is a combination of the Metaballs and the Lindenmayer system. I chose the combination of both because both harmonize well with each other.

With the help of the Lindenmayer system I can place the body parts based on the grammar and then refine these body parts with each iteration. Furthermore, this system is also really cost efficient as it only must rewrite the string with each iteration instead of doing complex calculations. The base rules of my grammar would consist of changing the position and rotation and in spawning different metaballs. As mentioned in the metaballs explanation we will use stochastic grammar so that we have varying results.

The metaballs on the other hand can be easily generated and because of the merging of the metaballs with each other I do not have to worry about the connection between the body parts. The shape of the metaballs can also be easily modified by using different formulas and threshold values, which helps greatly in creating different creatures. As seen in the creature editor inside of the game Spore, metaballs are useful to generate and display creatures.

I have decided against the divago system and the variational auto encoder as they have to be trained to give satisfying results and I do not have the required amount of data nor the time needed to create or collect these data. Poisson disc sampling was considered but because of their lack of integration of the body parts not fitting for this project. I would have used the superformula to create unique shapes to use as body or body parts but because of the active patent I am not allowed to use this formula. Similar to poisson disc sampling I will not use rigblocks as I have to worry how to integrate them.

# Project result

## How to use

To use this project you either import the unity package called CreatureGenerator to your project and use the generator and gui prefabs or use my unity project instead.

This project was created in Unity 2018.2 but it should also work with older versions as this project does not make use of new content of unity.

## Overall flow

The generation of the creature can be divided into four different phases. The first phase is the preparation phase where the user can set the number of arms, legs and heads they wish for the creature with the help of the GUI. When the user is satisfied with their settings, they can press the generate button and start with it the second phase of the generation.

The core part of the second phase is the Lindenmayer System. Inside this phase we create a ruleset based on the settings used by the player and then create a command string based on these results by replacing letters in each iteration of the system. Afterwards if the string was generated, we start with the third phase.

In the third phase we concentrate on the command executer. This executer is being fed with string generated by the Lindenmayer system. We will go through every letter of this string and execute a command based on the letter. These commands modify the current position and spawns the metaballs for the next phase.

The last phase has the metaballs as their focus point. We calculate the shapes of the metaballs and finally render the combined mesh of them with the help of the marching cube algorithm. With these four phases completed the user can see the generated creature inside the editor.

After the generation the user can fine tune the shape of the creature by changing the iso level and if the user wants to export the model they can press on the export button so that the mesh will be exported as an obj file so that you can use it outside of the unity project, for example you can edit it afterwards in a 3D Modelling software or share it with other persons.

## Lindenmayer System

The Lindenmayer System (also known as L-System) is a string rewriting algorithm which replaces with every iteration the characters in the string based on rules set beforehand.

This system is the most important for the creature generation as the result of this system defines the shape of the creature.

In my implementation of the Lindenmayer system you can use rules which have constantly the same replacement output or you can use stochastic rules where it can have different outputs with different percentages. Additionally, both sort of rules have an end output which will mark the end of the specific body part and can be randomly appear or used when the maximum iteration count of the body part is reached.

I have used mostly stochastic rules as they help to accomplish different variations for the body part and therefore also different creatures. I have only used constant rules for the body and for the heads. For the body I used a constant rule so that the body is consistent and therefore does not complicate the placement of the arms and heads further.

To store all rules and to access them easily I used a dictionary where every rule is assigned to a character. The following table shows all rules and their outputs:

**Ruleset**

|  |  |  |  |
| --- | --- | --- | --- |
| **Body part** | **Character** | **Output** | **Percentage** |
| Leg | ‘L’ | UML | 30 |
| “” | “” | UMFMNNMFL | 30 |
| “” | “” | UFMNPMRNMFML | 30 |
| “” | “” | MUB | 15 (Endresult) |
| Body | ‘B’ | MFMRMNMNMPMPMFMFMRNUB | 100 |
| “” | “” | MFMRMNMNMPMPMFMFMRNQ | Endresult |
| Arm left side | ‘A’ | MRA | 20 |
| “” | “” | MUMRMRA | 20 |
| “” | “” | MDMRMRA | 20 |
| “” | “” | M | Endresult |
| Arm right side | ‘V’ | MPV | 20 |
| “” | “” | MUMPMPV | 20 |
| “” | “” | MDMPMPV | 20 |
| “” | “” | M | Endresult |
| Head | ‘H’ | MRMUPH | 100 |
| “” | “” | MRMP | Endresult |

I added an additional variable for the rules to store the end result of the body part so that I can use this to end the body part if they reached their maximum number of iterations.

First, we generate the start string which will add the legs in pairs, one on the left and one on the right side. If the number of legs is odd the final leg will be placed separately in the middle. Each leg can have different heights as they can result earlier into the body, otherwise the leg will end after a certain number of iterations into the body. To equalize the final height of the body the maximum iteration of the body area is depending on the iteration count of the leg it resulted from.

If we have reached the maximum body height, we can start adding the arms and heads to the body. Before we do this, we calculate if there is enough space available on the body. If not, we will add additional body parts so that there is enough space for the remaining body parts.

I have chosen the back side of the creature (negative z direction) as the place for the heads as it will not disturb the single leg as it is on the other side. For the arms I have decided to put them on the sides as pairs on every side each. I wanted to have the arms in pairs as most animals have their arms in pairs and rarely just single arms.

## Command executer

To execute commands, I have used the command pattern as enables the easy use of different commands and helps to expand it later if further commands are needed. Currently the commands change the current saved position or spawn the metaball at the current position. In the future this can be expanded by adding commands to change the rotation, which is unnecessary at the moment as it does not have much impact. Another command idea is to change the size of the metaball which can be used to make more details like eyes on the head.

The command executer has a dictionary where the key is a char and the value is the action assigned to the character. The following table shows the content of this dictionary:

**Command dictionary**

|  |  |
| --- | --- |
| **Command character** | **Action** |
| ‘M’ | Spawn metaball at current position |
| ‘P’ | Move left (-x direction) |
| ‘R’ | Move right (x direction) |
| ‘F’ | Move forward (z direction) |
| ‘N’ | Move backwards (-z direction) |
| ‘U’ | Move up (y direction) |
| ‘D’ | Move down (-y direction) |
| ‘X’ | Reset X Position (0) |
| ‘Y’ | Reset Y Position (0) |
| ‘Z’ | Reset Z Position (0) |
| ‘Q’ | Reset Position (0,0,0) |

The function of the command executer is to go through every letter from the string generated by the Lindenmayer system to execute commands for every letter.

## Metaballs

Metaballs are shapes defined by a function and merge with other metaballs nearby.

I have used the structure of the implementation of a metaball system by Brian R. Cowan [https://wiki.unity3d.com/index.php/MetaBalls] as the base for my own implementation.

In the beginning of my implementation we first find and store all metaballs which were spawned by the command executer inside the editor. Afterwards we create the grid which is necessary for the rendering of the metaball mesh. After the creation of the grid we go through every cube consisting of eight points inside the grid calculate for every point the intensity value which is depending on the position difference with each metaball. If the value is above the threshold which the user can influence the point will be part of the mesh. When we are done with every point of the cube, we will generate the shape with the help of the marching cube algorithm.

While checking every point we also create a 8 bit value to be used as an index for both edge and triangle table. These tables store the vertices and triangles we need to recreate the shapes.

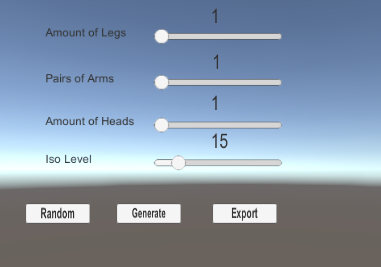
## GUI

Due to problems I had with the implementation of the metaballs I only created a basic and simple GUI so that I can use the remaining time which I assigned to the development of the GUI to work on the metaballs, which had a higher priority for this project. Nonetheless the GUI is fully functional and has everything necessary to use this project.

The GUI consists of four different sliders and three buttons. With the first three sliders you can set before the generation of the creature the number of arms, legs and heads which you want to have for the creature. The last slider decides the iso level and can be also used after the generation of the creature and influences the shape of the creature as the iso level is the threshold for a point being considered as part of the mesh.

If you want to surprise yourself with the result you can press on the random button which results in a random value for each slider. By pressing the generate button you start the generation of the creature based on the values you set beforehand with the sliders. When you press the button while there is already an existing creature, the previous creature will be replaced by the new creature.

Finally, when you press the export button after you generated the creature you can export the mesh of the creature to an obj for further uses outside the project.



*Image of the GUI*

# Post mortem

## Problems

The problem which took a huge part of my time was that the metaballs were not rendered. This was due many different logical errors.

Some of these problems was in the creation of the lattice. There I had many small errors which lead to missing points or objects with wrong values were created. Another issue I found was that I placed the calculation of point intensity in the wrong brackets resulting that it was overridden again with a zero value and therefore would not be accounted as part of the mesh.

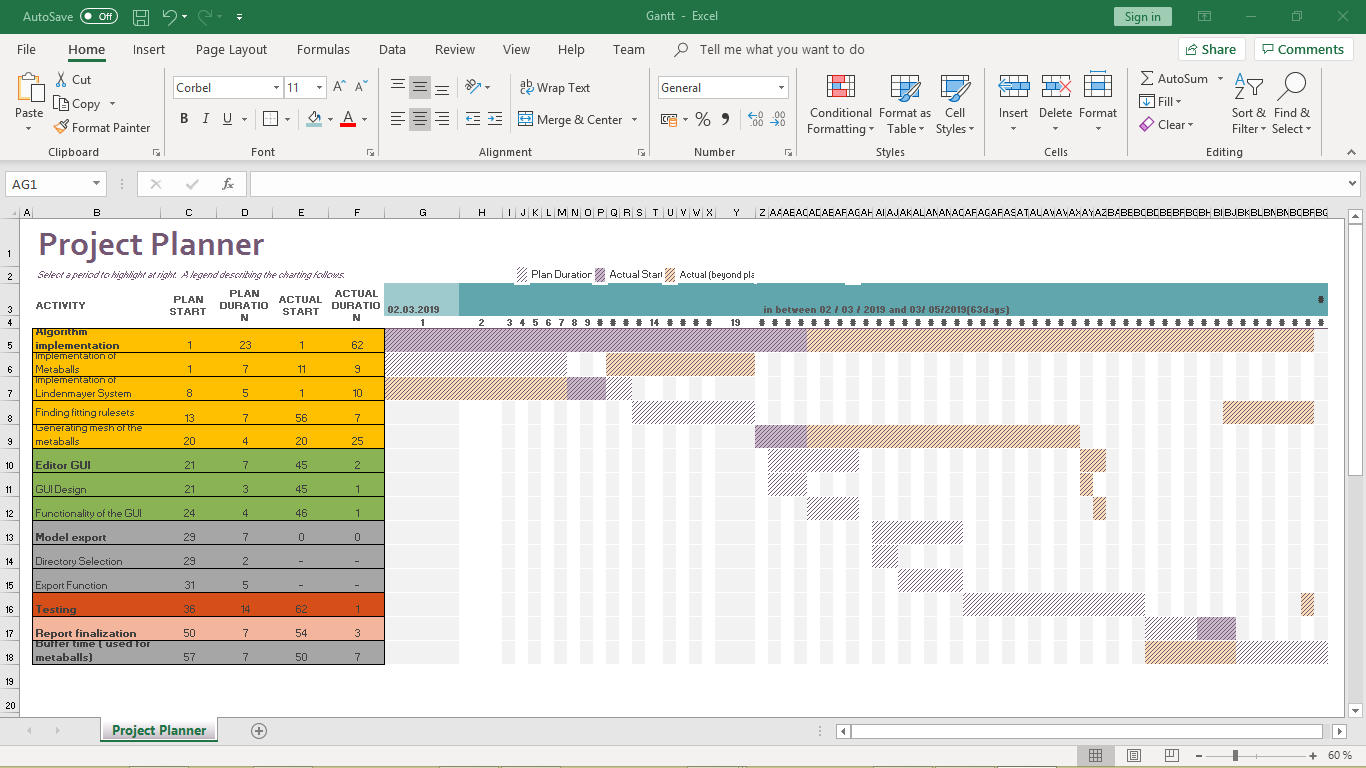
Another small problem was that I assigned the wrong values to the uvs and normal which also lead to the problem of the metaballs not rendering.

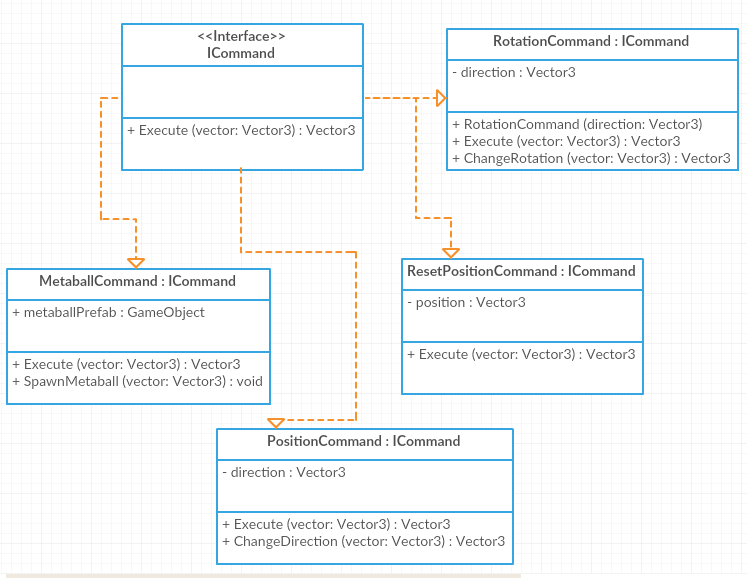
As I have solved these problems, I encountered another obstacle in the form of the metaballs not correctly merging with each other. This problem was not solved and therefore I changed to simple spheres.

Furthermore, I could not managed to show the result properly in the game screen as I had not enough time to do this.

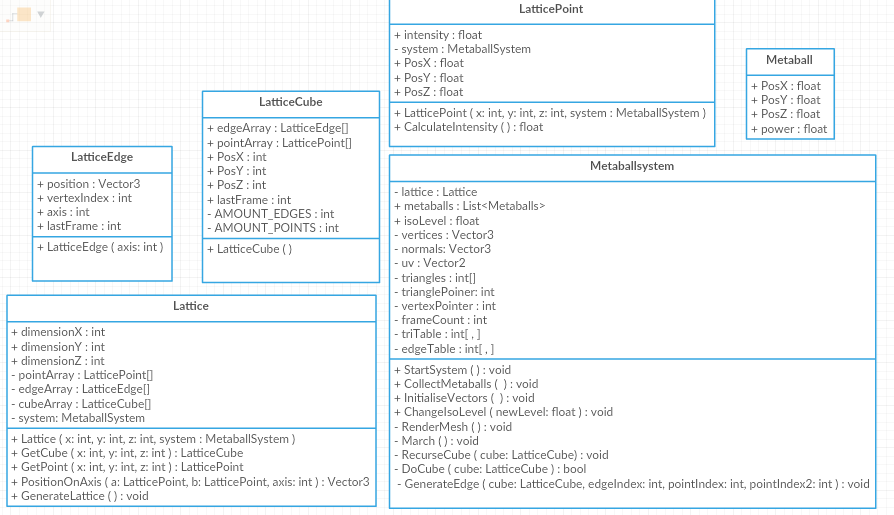
I also encountered some problem with the ruleset of the L-System, especially with the placement of the body parts. First it is not easy for me to visualize how I have to change the positions to accomplish the correct placements. Second, I also had to fight against small displacements or incorrect position resets which will sum up and ending with body parts not being in the correct place. This was also not resolved.

## Changes from plan

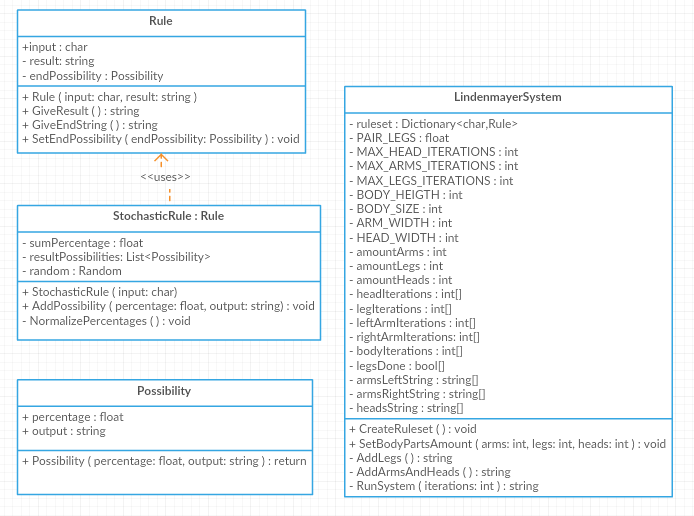




*Command Part of class diagram*



*Metaball Part of class diagram*



*Lindenmayer Part of class diagram*

The biggest change between my project plans and the actual result was the work behind the metaballs, both in complexity as well as in time management. I completely underestimated the amount of time needed and the complexity of this topic. As you can see in the Gantt chart, I assumed that I only need about two weeks for the implementation of the metaballs but eventually it took me more than a month. Another proof of my underestimation is the comparison of my planned class diagram and the final diagram. While in the first diagram the metaball consist of only one class with a few variables and methods, in the result it consists of multiple classes with more variables and functions.

I have replaced the metaballs in the final result with simple spheres as the merging of the metaballs were not correctly and therefore the simple spheres gave the shape of the creature better away.

Another change was the removal of the creature class as it was unnecessary. The designed function of the creature was basically just to store the vertices of the mesh. I replaced the creature class by using the mesh component of unity as it can store vertices and other useful information such as uv’s und normal. Furthermore, it allows for easy rendering of the mesh as it is automatically rendered if you give it the necessary information.

Due to missing time I could not manage to implement the function to export the generated mesh as an obj file. Further researches in this area also let me think that my previous thoughts on the time needed for the file export was most likely more than I would have needed. To export the mesh, you only must write the information of the mesh in a specific form in a string and save as a file.

A small addition to the rule classes of the L-System that I implemented was that they can have a string which can used to close the rule. This enabled me the ability to finish off body parts when they experienced a certain number of iterations.

Another noticeable difference between the plan and the result is that the LindenmayerSystem class now have more variables. These variables were needed to keep track of the different body parts and the limitations needed to accomplish the generation of a creature.

Otherwise there were no changes to the general structure of the process except the exclusion of the export function.

## Was the approach good?

The Lindenmayer system was a great idea as it allowed relatively easy to define the shape of the creature. Although I should have started earlier on that and wasted less time on the metaballs as they were the most important part of this project.

I should have understood the metaballs better beforehand as they overwhelmed me with their complexity. Also I should have moved on earlier away from the metaballs as they slowed down my productiveness.

## Ideas and improvements for the future

In the future the ruleset can be improved and expanded so that there will be more variants of creatures types and more diverse results for each body part.

One feature I tried to include but could not because of my problems with the metaballs was to also use other shapes like cubes instead of only using spheres. This should not be a big problem to include as the shape is basically only mentioned in the “CalculateIntensity” function inside the LatticePoint class and therefore to create other shapes you should only have to use different algorithms to calculate the intensity of a point based on the shape you want.

Another feature which could be implemented in the future is to let the user add, delete or edit metaballs so that they can do quick adjustments inside the editor without the need of extern 3D Modelling software. Alternatively, you could let the user edit the vertices of the mesh. The problem with vertices control is that it is more complicated to implement and is far more advanced than the other features and would contradict my goal of a program which is easy and intuitive to use.

Furthermore, I have the idea that the user can also modify and create their own rulesets for the generation of the creature. Similarly, to the vertices control the problem of this feature is that it would be against my intention of an easy to use program.

Additionally, one small feature which can be included to improve the usability is the addition of a camera system where the user can rotate the creature to see them from different angles.

## Conclusion

There were many issues I encountered during the development of this project. The biggest two problems were the implementation of the metaballs and motivation problems which were partly caused by the metaballs.

For future projects I will try to understand complicated systems like metaball system better during the planning phase instead during the development phase. This way I can plan better and possible see that some options are too complicated and therefore should seek for another solution.

On the other hand, I do now understand the concept behind metaballs which are interesting and can be also used in different situations, for example the simulation of fluids like water.

Considering the motivation problems my goal for future projects is to set more and smaller goals. This will help to handle the tasks better and I will have more little successes which helps to boost the motivation enormously. Furthermore, when I am stuck at a problem, I can try on another task so that I will not be this easily frustrated and can later come back to the problem with a fresh mind.

Another idea to fight against motivation issues next time is to have more regular working times as that allows me to be more productive and have less problems to start working.

An additional problem was also to set priorities better beforehand. In this project I should have worked earlier on the ruleset for the creature as it is more important than the metaballs.

Furthermore, I should have decided earlier on an approach during the planning phase as it would have allowed to have more time on planning the project and therefore could have had an easier time during the development.

One last point I can improve on is to write and test in smaller chunks. I find myself repeatedly in the situation where I wrote many lines of code and then have my problems finding out where exactly the problem is.

# References

## Github

<https://github.com/Markus-Krugel/Creature_Generator>

## Text

1. Ribeiro, P., Pereira, F., Marques, B., Leitão, B. and Cardoso, A. (2019). A Model for Creativity in Creature Generation. In: *4th International Conference on Intelligent Games and Simulation {(GAME-ON}*. [online] London: EUROSIS, p.5. Available at: <https://www.researchgate.net/publication/221024396_A_Model_for_Creativity_in_Creature_Generation> [Accessed 27 Feb. 2019].
2. [Unity] Procedural Object Placement (E01: poisson disc sampling) (2019) YouTube video, added by Sebastian Lague [Online]. Available at <https://www.youtube.com/watch?v=7WcmyxyFO7o> [Accessed 27 February 2019].
3. Hecker, C. (2014). *My liner notes for spore*. [online] Chrishecker.com. Available at: <http://chrishecker.com/My_liner_notes_for_spore> [Accessed 27 Feb. 2019].
4. ‘Metaballs’ (2019) *Wikipedia.*Available at <https://en.wikipedia.org/wiki/Metaballs> (Accessed: 27 February 2019).
5. Wong, J. (2014). *Metaballs and Marching Squares*. [online] Jamie Wong. Available at: <http://jamie-wong.com/2014/08/19/metaballs-and-marching-squares/> [Accessed 27 Feb. 2019].
6. ‘Superformula (2018) *Wikipedia.*Available at <https://en.wikipedia.org/wiki/Superformula> (Accessed: 27 February 2019).
7. Velho, L., Perlin, K., Biermann, H. and Ying, L. (2002). *Algorithmic shape modeling with subdivision surfaces*. [ebook] New York: IMPA—Instituto de Matematica Pura e Aplicada, pp.865 - 875. Available at: <https://web.stanford.edu/~lexing/asmss.pdf> [Accessed 27 Feb. 2019].
8. Choy, L., Ingram, R., Quigley, O., Sharp, B. and Willmott, A. (2007). *Rigblocks: Player-deformable Objects*. [ebook] Electronic Arts. Available at: <http://www.cs.cmu.edu/~ajw/s2007/0248-Rigblocks.pdf> [Accessed 27 Feb. 2019].
9. Norato, J.A. Struct Multidisc Optim (2018) 58: 415. <https://doi.org/10.1007/s00158-018-2034-z>
10. Weber, S. and Weber, S. (2019). *Poor Man’s Sky: Experiments with Procedural Generation of Characters*.[online]Making Games. Available at: <http://www.makinggames.biz/feature/procedural-generation-characters,12342.html> [Accessed 28 Feb. 2019].
11. Altosaar, J. (2019). *Tutorial - What is a variational autoencoder?*. [online] Available at: <https://jaan.io/what-is-variational-autoencoder-vae-tutorial/> [Accessed 1 Mar. 2019].
12. Kenwright, B. (2019). *Metaballs & Marching Cubes*. [ebook] Available at: <http://www.xbdev.net/misc_demos/demos/marching_cubes/paper.pdf> [Accessed 1 Mar. 2019].

## Images

1. Bourke, P. (2002). *Example 1 of Supershape in 3D*. [image] Available at: <http://paulbourke.net/geometry/supershape/C_0009s.jpg> [Accessed 27 Feb. 2019].
2. Bourke, P. (2002). *Example 2 of Supershape in 3D*. [image] Available at: <http://paulbourke.net/geometry/supershape/C_0022s.jpg> [Accessed 27 Feb. 2019].
3. Cardoso, A. (2003). *Conceptual blending figure*. [image] Available at: <https://www.researchgate.net/publication/221024396_A_Model_for_Creativity_in_Creature_Generation> [Accessed 27 Feb. 2019].
4. Cardoso, A. (2003). *Divago system figure*. [image] Available at: <https://www.researchgate.net/publication/221024396_A_Model_for_Creativity_in_Creature_Generation> [Accessed 27 Feb. 2019].
5. Cardoso, A. (2003). *Outcome of the Divago system*. [image] Available at: <https://www.researchgate.net/publication/221024396_A_Model_for_Creativity_in_Creature_Generation> [Accessed 27 Feb. 2019].
6. Choy, L., Ingram, R., Quigley, O., Sharp, B. and Willmott, A. (2007). Example of a rigblock. [image] Available at: <http://www.cs.cmu.edu/~ajw/s2007/0248-Rigblocks.pdf> [Accessed 28 Feb. 2019].
7. Author unknown (2019). Example of metaball interaction [image] Available at: <https://upload.wikimedia.org/wikipedia/commons/6/6d/Metaball_contact_sheet.png> [Accessed 1 March 209].

33 <https://www.deviantart.com/bernoully/art/Spore-Creations-Showcase-13-102078210>

34 <https://images.sftcdn.net/images/t_app-cover-l,f_auto/p/7ee75bde-96d1-11e6-b796-00163ec9f5fa/966625763/spore-creature-editor-screenshot.jpg>

35 <https://i.ytimg.com/vi/WpUbnz2XHZY/maxresdefault.jpg>

56 <https://www.pinterest.com/pin/401453754264993498/>

## Model for GUI prototype

<https://free3d.com/3d-model/murloc-93031.html>