Life of Blob

Interim Report - 20.04.2015

Andreas Halter
Diego Martinez
Manuel Braunschweiler

1. Project Status

Currently we are into layer 3 with a few tasks from layer 2 that still need to be implemented in a satisfactory way. For example the procedural generation of enemies is at the time of writing simply a random generator that is not connected to the player's abilities or the special properties of the level environment. Meaning: enemies are getting their abilities at random and are therefore often easy prey for the player. Secondly it is harder for the player to obtain the abilities he/she needs in order succeed in the level - but since the levels are not changing yet, this doesn't impact gameplay a lot. Another major backlog we're having is in the visual part of the game. Things almost look the way they did during the first presentation.

2. Implementation Details

We decided to generate the levels with Perlin noise. This allows for having infinite levels but comes with a major performance impact. This is partly due to the fact that the level is basically regenerated in every frame. But this fact would also allow to change the level's look and feel smoothly over time. How to accomplish this without the level going nuts is one of the things we have to test in the coming days. That the Perlin noise calculation is costly showed already early during developement and forced us to parallelize the main loop of the Perlin noise calculation. After dealing with some weird loop optimization problems, the parallelization was eventually working and pleased us with a nice gain in framerate.
Regenerating the level in every frame also means to regenerate (or lets say: reposition) environmental obstacles in every frame. Unsurprisingly this turned out to be even costlier than the pure Perlin noise generation. While it is horribly slow in the Unity editor, it works quite fine as a compiled exe program. Currently it doesn't look nice at all, because the obstacles are flickering on the screen at each movement. But since these will only be the colliders, they should not be seen in the final game.

While the basic enemy AI worked quite fine for empty levels, we needed to improve it in a way, such that the AI gets aware of the environmental obstacles and avoids them if the AI blobs don't have the necessary ability to cross that environment. Avoiding these objects works in some cases but isn't flawless by any means. Also, this behaviour needs to be tweaked for

every possible state of the enemy: is the enemy running away from the player? Is it hunting the player? Is it just doing some idle behaviour? So the AI should seek for some protective environment if the AI blob is running away from the player. At the same time, the AI should stop in front of these obstacles if it doesn't have the ability to enter it and should therefore no longer pursue the player, who found shelter in there. Making the AI aware of an ever changing environment certainly isn't easy.
There also needs to exist some algorithm that decides on what ability to use next - according to the current situation (i.e. environment, player actions, etc).

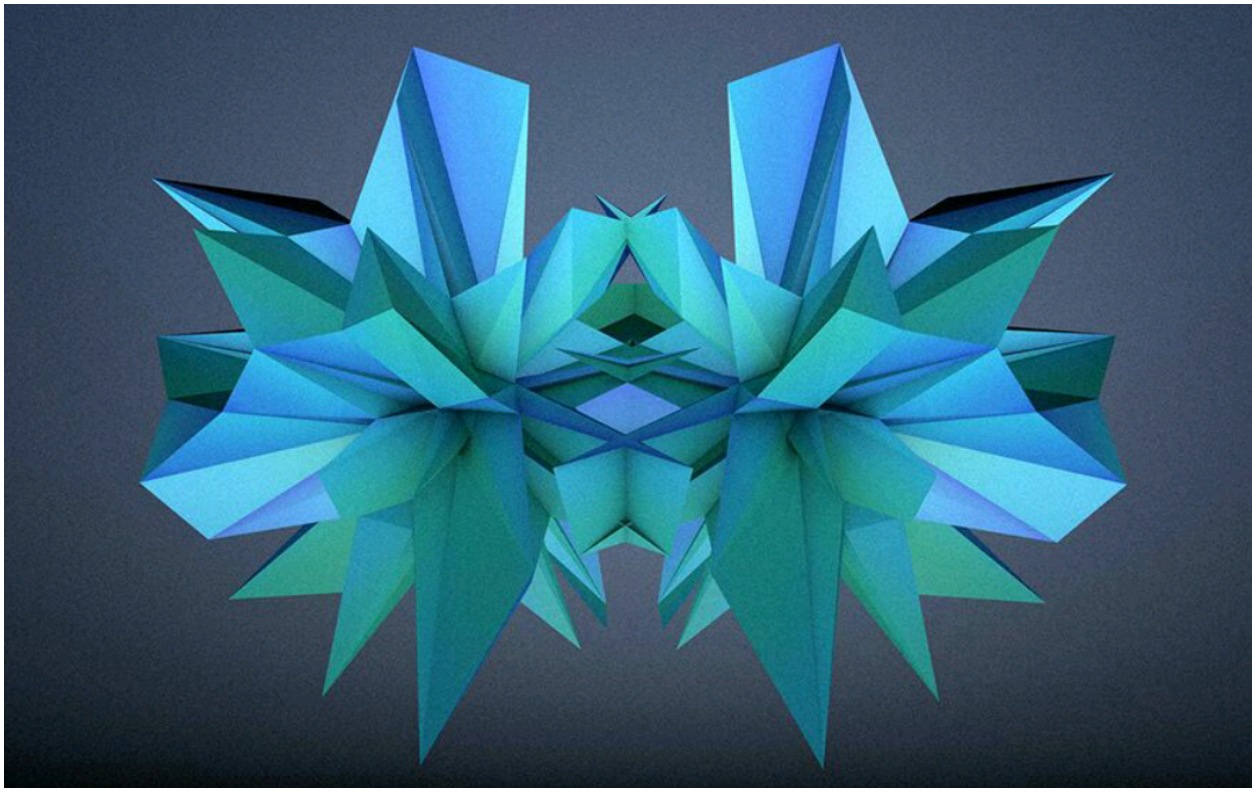
We have implemented some basic abilities, which allow to attack, be faster, view further or enter hazardous environments. The ones implemented work fine most of the time but can lead to some problems. For example the ram ability (which allows the player to quickly rotate into a selected direction and then charging with high speed into this direction, smashing (almost) everything on the way) takes control of the player's actions for a short time. The player should not be allowed to move freely during that time. Disabling the controls is easy but stopping the animation in case something unforseen happens (e.g. player dies, player gets stunned by some different attack, etc) turns out to be difficult, as not every part of the code has access to the controls of the ram animation. We belive to have covered most of the cases for the ram ability but for some other abilities coming in the future, we may have to face similar problems, solving which will add more complexity to the code.

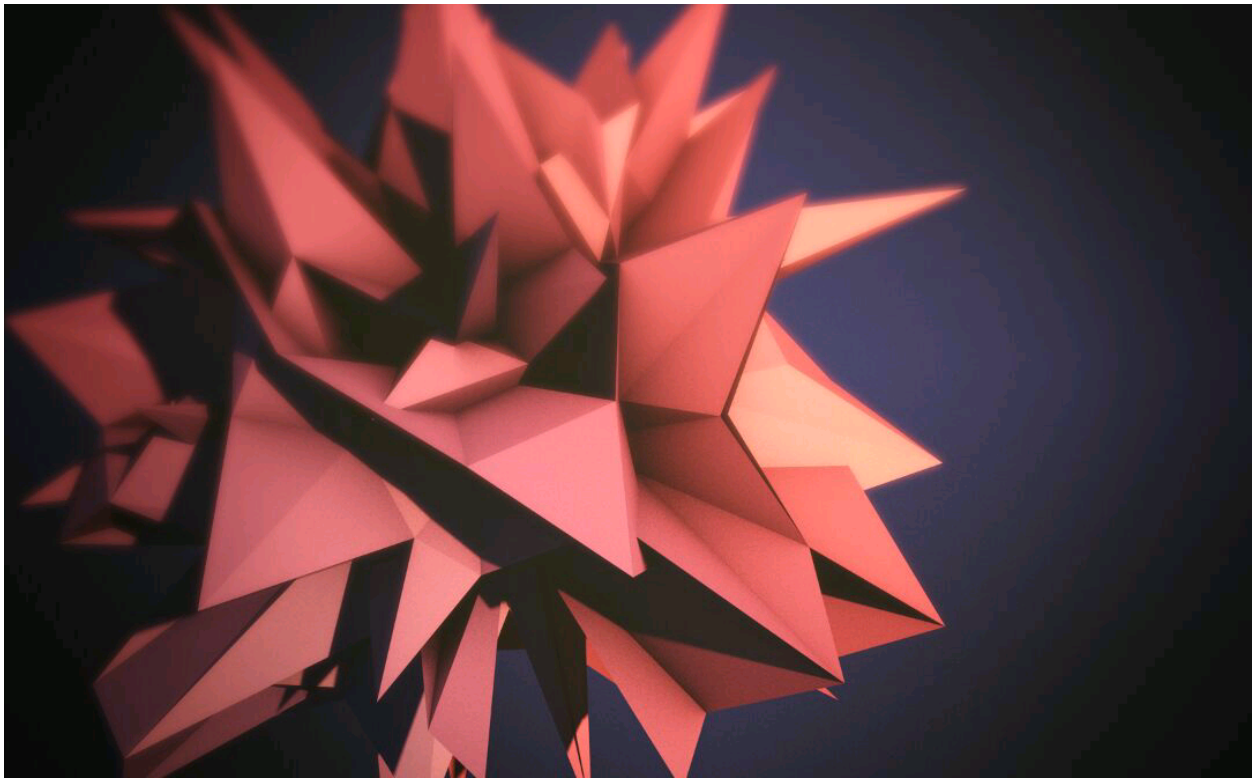
As we already mentioned, abilities are distributed by random on the enemy blobs. Once an enemy blob with an ability is defeated, one of its abilities is chosen at random, to be transfered to the player. If it is a passive ability (like speed or viewing range), the ability will simply be added to the player. If it is an active ability however, the frame freezes and a short dialogue is shown, that asks the player to place the new ability on one of the available buttons for future use. While this works fine, it is a bit problematic, that the game always pauses for this occasion and therefore kind of breaks the flow of the game. Hopefully we'll find a way to show this dialogue without pausing the game.

Something else we're currently testing is the way enemy blobs can hurt the player without the use of any abilities. Should bigger blobs be able to hurt the player without ability? Should smaller blobs be able to hurt the player or should they always be running away? This again will eventually lead into the realm of enemy AI and the use of the different shield abilities. These shield abilities are not just to enter certain environments but can also protect from certain attacks or damage enemies upon contact. The exact way these shields work also needs some more concideration: Should they always be active? Should they only be active upon button press? For how long? We currently tend to make them only active upon button press, in order to make it more involving for the player, as he/she has to actively block attacks or surprise enemies shortly before contact with a spiky skin, etc.

Graphics

We are currently creating and implementing a graphic style that fits the theme of procedural enemy and level creation. It will be mainly abstract with low poly 3D models that are procedurally distorted to reflect the status of the being. For example, more aggressive enemies will have a more distorted structure that is reminding of spikes while more peaceful enemies will have smoother surfaces.
The graphic style should mainly serve as a gameplay indicator. Colors will be chosen and distributed to show the player which enemies are prone to attack them and which ones are easy prey. Fields of a certain color that can not be passed through require an ability that is possessed by enemies of the same color, and so on.





Some inspirations