The

Creation of a VR Game

SPACE ORBS

Realisierung eines VR-Prototypen mit Fokus auf die Übersetzung
von Beschleunigungen aus der Realität in eine Bewegungssteuerung

Hien Quy Tran
Wissmannstraße 47
12049 Berlin

Bachelor Thesis
Game Design
HTW Berlin

Matriculation number: 543800
Berlin, September 2017

Supervisors: Prof. Thomas Bremer
Prof. Susanne Brandhorst

The

Creation of a VR Game

SPACE ORBS

Creation of a VR Game
with emphasis on motion controls

Hien Quy Tran

Abstract

This documentation is part of my bachelor theses: “Creation of a VR Game with emphasis on motion controls”. (original title in German: “Realisierung eines VR-Prototypen mit Fokus auf die Übersetzung von Beschleunigungen aus der Realität in eine Bewegungssteuerung”)

It will give you an insight of what my motivation was and where my idea came from to create this prototype. Furthermore, it will show you my working process and go into more detail of each game entity with all their reiteration steps.

Throughout this document I will not only show you the pitfalls I have fallen into, but also tell you about the lessons I have learned.

Please keep in mind, that this documentation only reflects a fraction of my actual work. Majority of the time was spent in crafting the prototype in the Unreal Engine 4 Editor.

The following documentation will not describe any gameplay of the game. It will focus on describing the creation of the game. ***If you haven’t done so yet, please consider to play the game before reading.***

Table of Contents

Introduction

Motivation · Goals · Technology

Introduction

~ Motivation ~

It was the first time for me, to get in touch with game development, when I started to study game design. Never have I written a single line of code, nor did I know what else to be expected of this profession. I had no clue about, what a game designer is and where to place myself among all the various professions within game development.

It was a long path to walk in my career as a game designer.

Looking back at my studies, I have been learning a lot. In fact, thanks to my education, I have been able to work as a part time game designer for about a year now and was able to gain a lot of complementary work experience.

I see this project as a resemblance of what I have learned and achieved over the course of the last three and a half years and I am aiming to synergize the knowledge I have gained from my studies with the experience I have gained from work.

I am very interested in virtual reality and especially mixed reality. I believe that these technologies will shape the future of the media landscape. The technology is quite new, but leaves us developer a lot of room to explore. I am very interested and curious about its potential and how it can be utilized. I am sure, that we are just seeing the tip of the iceberg and that we are still in the process of comprehending its possibilities.

~ Goals ~

As a student, I was mainly focused on exploring innovative gameplay mechanics, while leaving boundaries aside. The project I am working on as a part time game designer on the other hand is bound to a rigid structures and a clean development. This being said, the code base of my past student projects have been pretty messy in comparison. This would have not allow a for good workflow in a project with a team size of roughly hundred people.

My goal is to create a virtual reality game, which is not only appealing in its design, but is also set up in a way, that the content is modular and easily expandable. For that matter, it is essential to carefully plan and set up a clean foundation for the framework. Each and every aspect of the game has to be implemented as a single game entity. My hope is, that this will result in high stability and great expandability of the game, as well as a solid foundation for further development. I believe, that if the foundation is set up in a correct way, content generation can be very efficient. With this believe, I am aiming to create content which allows for roughly thirty minutes of playtime.

~ Technology ~

As mentioned before, I am very curious about the future of virtual reality and believe, that it will be a huge driving factor for the industry.

I have been playing around with the HTC Vive and the Oculus Rift. While the HTC Vive is superior in image quality and device tracking across a whole room. The Oculus Rift is superior in comfort and setup. Although I really like the touchpad of HTC Vive's motion controllers, I personally prefer the ones from Oculus Rift due to its more ergonomic design.

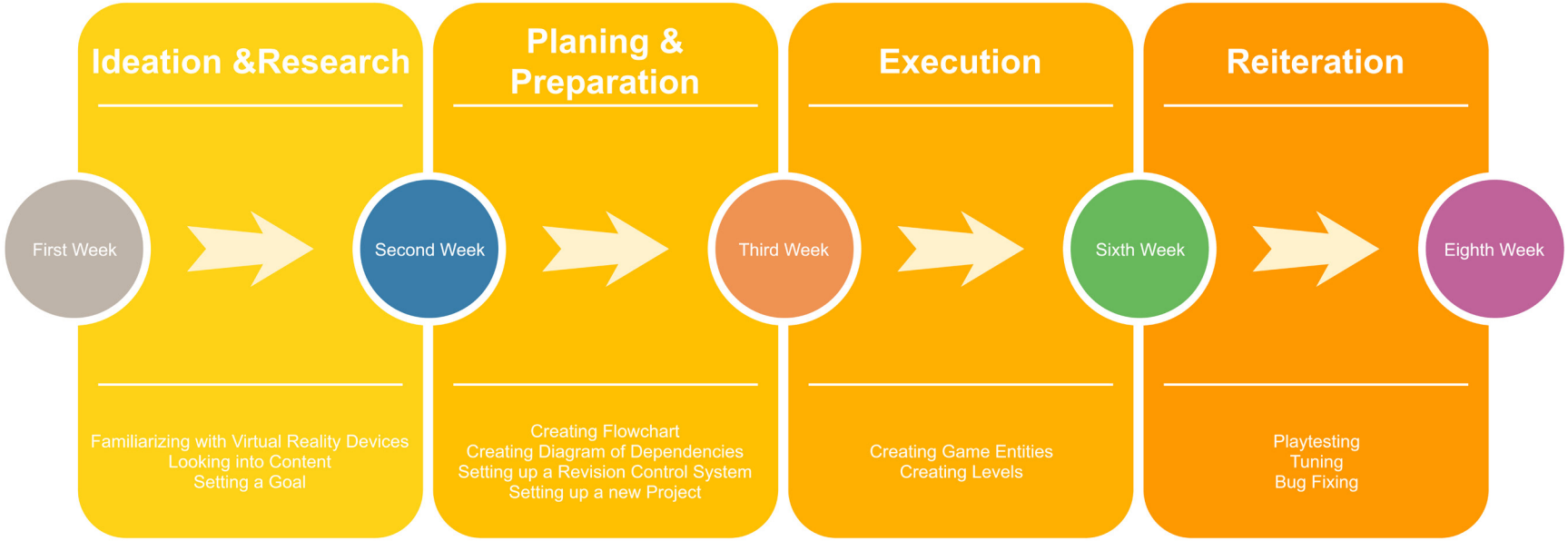
Looking at my game, all advantages of the HTC Vive would not have a great impact and I therefore preferred to choose the one which is sufficiently tailored towards my needs.

During my studies I was mostly developing with the Unity and the Unreal Engine. Both Engines have their own advantages, but because I also work with the Unreal Engine at work, I feel much more familiar with it by now. That is why I chose the Unreal Engine 4 over Unity 5.

Other software aiding me with my project are Autodesk Maya, Adobe Photoshop and Ableton Live.

Work Process

Ideation & Research · Planing & Preparation · Execution


Ideation &Research
Planing & Preparation
Execution
Reiteration
First Week
Second Week
Third Week
Sixth Week
Eighth Week
Familiarizing with Virtual Reality Devices
Looking into Content
Setting a Goal
Creating Flowchart
Creating Diagram of Dependencies
Setting up a Revision Control System
Setting up a new Project
Creating Game Entities
Creating Levels
Playtesting
Tuning
Bug Fixing

Work Process

~ Ideation & Research ~

The Hardware should not be redundant.

Meaning, that the game should not be adaptable into a desktop game. For that, it has to make good use of the freely maneuverable stereoscopic camera and the motion controllers.

To get some inspiration, I took a look into some of the available virtual reality demonstrations. I was especially pleased by the high production value of the Oculus Rift Tutorial and showed it to some of my friends. While most of them had their first virtual reality experience, it became pretty apparent, that even those, who are not playing video games on a regular basis, did not experience any difficulties in interacting with the virtual environment.

Conventional video games requires the player to cognitively map specific action options to corresponding buttons or even button combinations. The use of motion controllers allows for an input option, that is more similar to the actions of every day life. Simple tasks like reaching out for objects, grabbing and dropping them are very natural to perform. The simplicity and easy adaptation of the controls made virtual reality an enjoyable experience for everyone. It was important to me, to maintain this condition and to not end up with a hot wire game.

The execution of the interaction should feel natural.

Having this in mind and while observing my friends, I noticed, that throwing stuff around was a popular thing to do in the virtual reality. Furthermore, it was often the first item on the to-do list of things that have never been asked for.

Throwing objects is interesting in many ways. In its core, it is a seemingly easy action, but it gets hard real quick, if asked to throw an object in a consistent manner. I can throw a basket ball, but can I hit the basket? Too many variables are contributing towards the end result, so that even professional basket ball players are unable to predict their shot with 100% accuracy. Most of those variables, including angle and force, are also replicable in virtual reality. But I did not want to create a hit the basket simulator.

It should be about something, that is almost impossible to recreate in real life.

One of the great features of virtual reality is the possibility to set up conditions, which are difficult to be set up in the real world and let's face it: gravity was always an annoyance in sports class.

I have played a few games, in which gravity is key element, but there is one in particular, which is standing out:

'Sagittarius' by George Prosser

It is a turn-based competitive 2D cosmic archery game, which I personally like, because of its minimalistic but strong game design. I also looked into other more popular games, like for example: 'Angry Birds Space' by Rovio.

Inspired by those games, I started to sketch out my game.

- It is a single player experience.

- The players main activity is trying to hit a specific target by throwing object towards it.

- There are numerous of different objects to throw and they all behave differently.

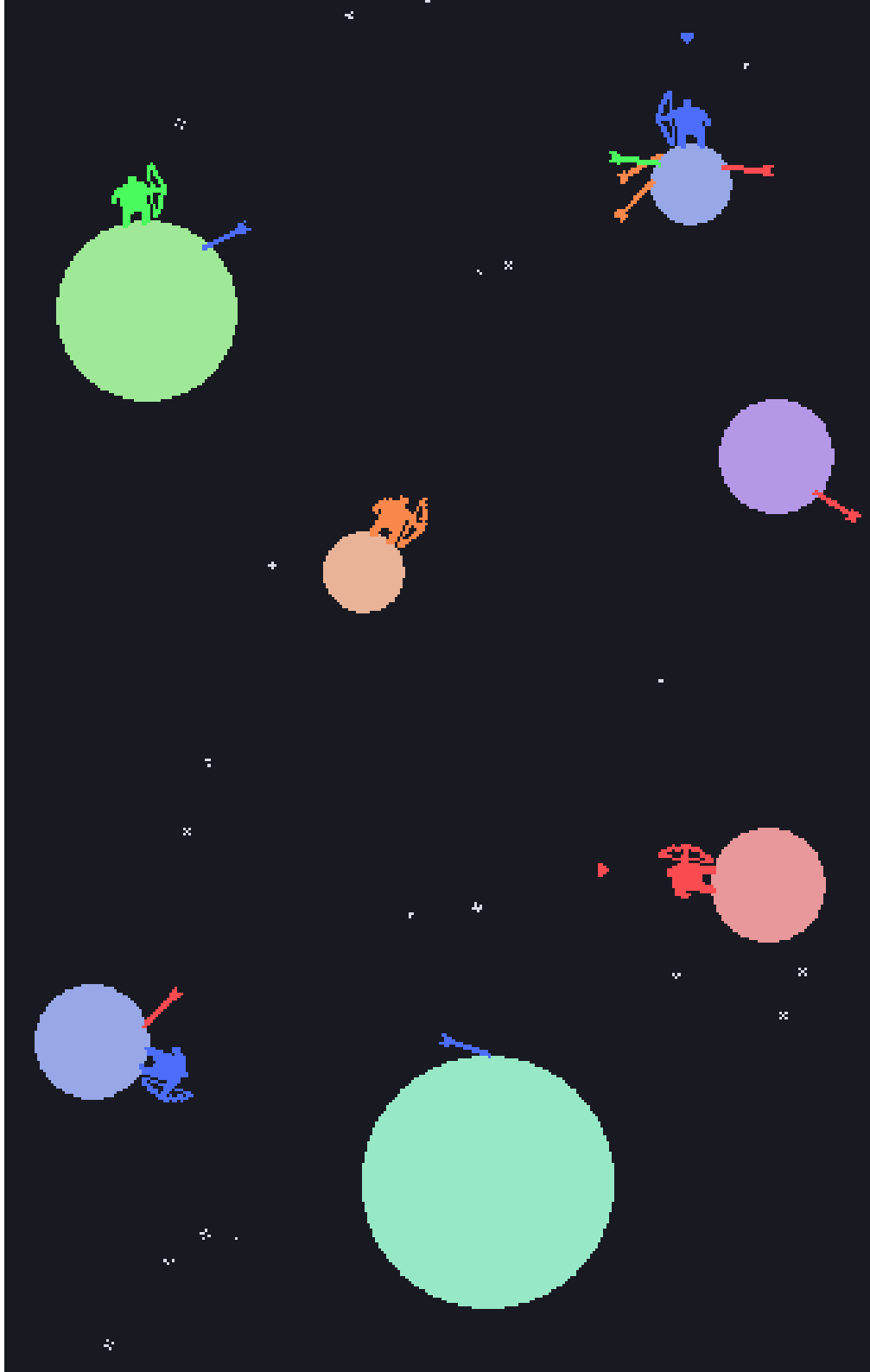
- To hit the target, the player needs to understand and solve the puzzle like level.

- The player progresses through levels, but can also fail and end the game early.

- It has a system, that allows for competitive play by setting high scores.

- Planing & Preparation

To give me a head start on this time limited project, I decided to build my game on top of the Unreal Engine's Virtual Reality Template.

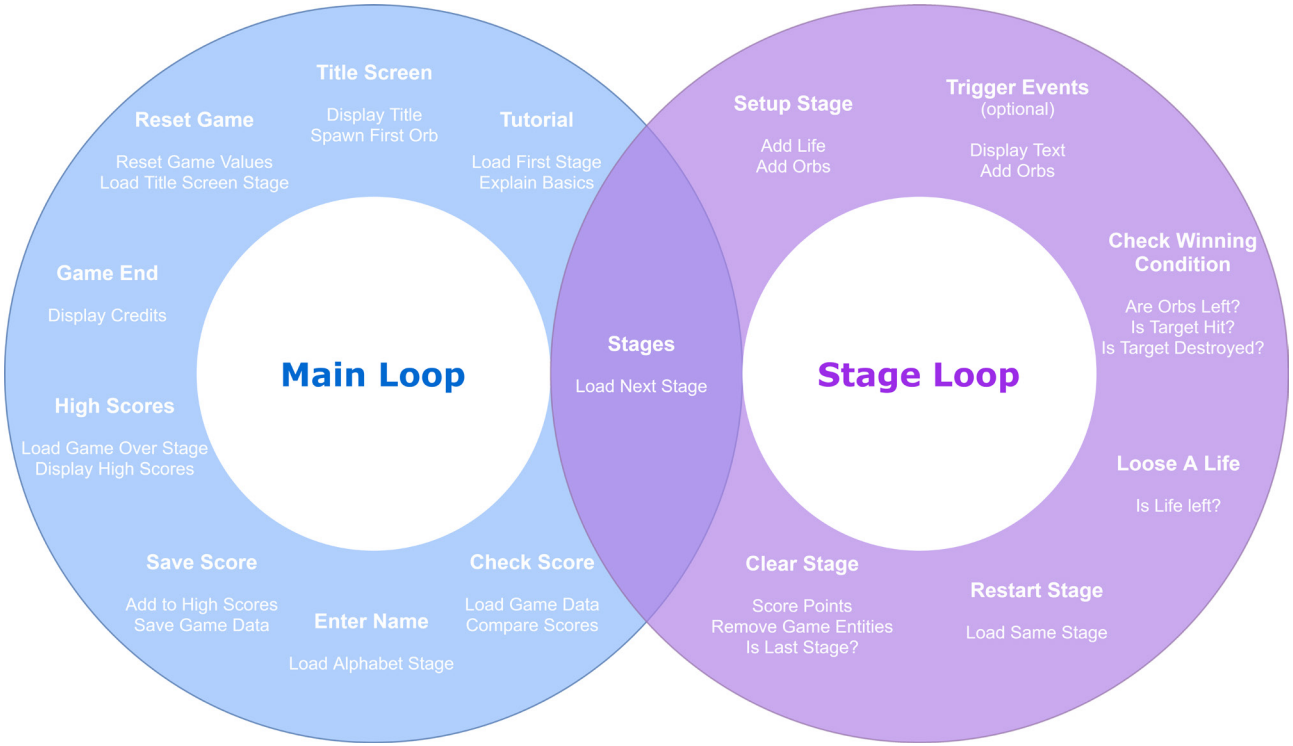


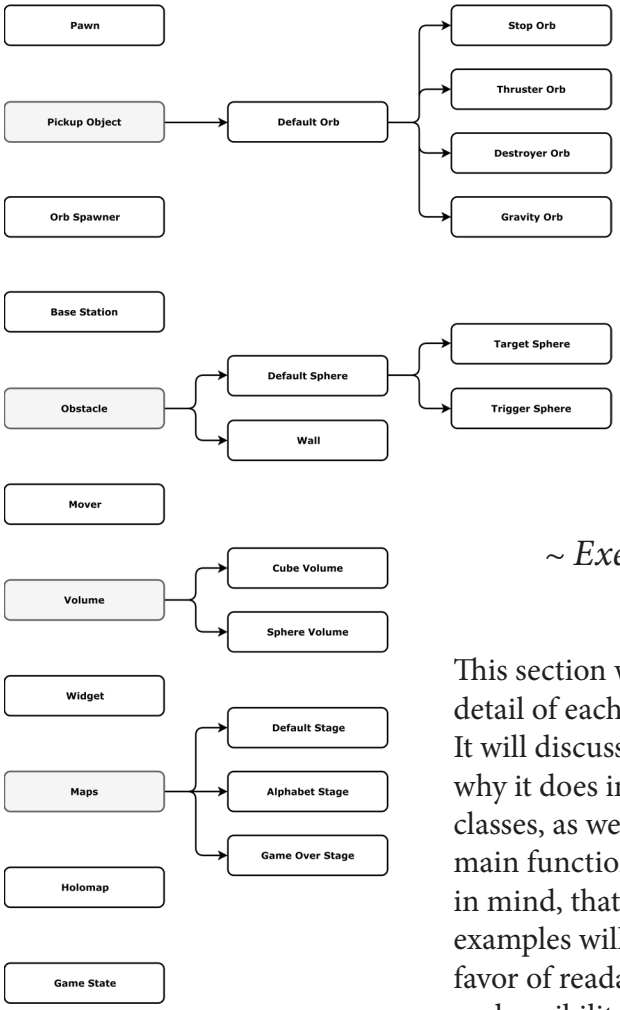
This template already includes a pawn with functions to grab objects, release objects and teleport itself. That came handy, because I only needed a pawn with the function to grab and release objects. I remapped the buttons to my preference and adjusted some of the force feedback effects. I then deactivated the teleport function, since it was not needed. Moreover, I was able to find little squared boxes in the template, which could be grabbed and released. I was about to create objects which share exactly this trait. Instead, I was able to use these boxes as the parent pickup object class. Everything, that can be picked up will simply inhered exactly this trait from this class. I then sketched out the overall game loop and listed all required game entities and their dependencies.

Before I continued to create all the game entities I was setting up a revision control system. I cannot stress out enough, how important this step is.

It usually does not take more than one hour to set it up and can spare a lot of time in further development. To be able to jump back to any previous revision can save a lot of work.

It also helps the user to document all the changes made. I chose Perforce as the revision control system of this project, because I am fairly familiar with it and is greatly supported by the Unreal Engine 4.





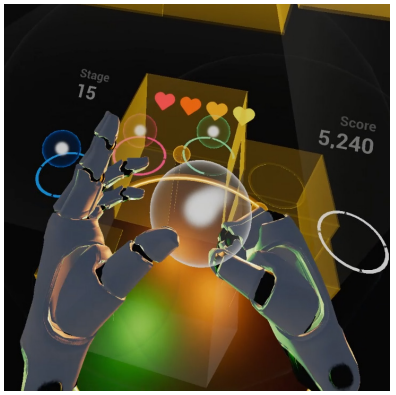
~ Execution ~

This section will go into more detail of each game entity. It will discuss, whether and why it does inherit from other classes, as well as explain its main functions. Please keep in mind, that some of the examples will be simplified, in favor of readability and comprehensibility. I believe it is more important, to be able to follow the general idea, than not being able to comprehend anything due to the petty listing of all details.

Pawn

The pawn I am using is the one from the Virtual Reality Template. It already comes with the function to grab and release objects, as well as the option to teleport itself to another location. While the teleport function can be disregarded, since I removed this function from the pawn, it might be of interest to know how the grab and release functionality works.

The pawn has virtual hands attached to it. These hands are mimicking the position of the real hands by tracking and following the motion controllers. These virtual hands have colliders around them. If they overlap with colliders of other game objects, they will check, whether the other object is a pickup object. If it is a pickup object, it will attach this other object to the hand on button press and hold. On attach it also deactivate the attached objects physics simulation. Releasing the button will result in detachment and physics simulation reactivation of the object.



Pickup Object

A pickup object is simply an object, that is marked as an object, which can be attached to the hand. I used this pick-up object to create the base class of my orbs.

Pickup Object: ***Default Orb***

This default orb class inherits all traits from the pickup object. I then added additional traits, which are valid for all upcoming orbs. For the audiovisual appearance, I added a sphere mesh, a particle system for the trail and a sound, which is depended on its velocity. For its flying behavior, I then overrode the physics settings and deactivated gravity. I added two main functions to the orb:

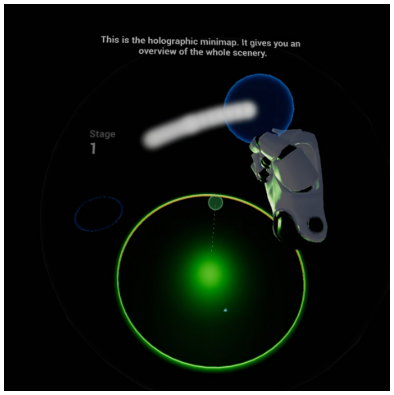
Activate - sets the orbs state to active and add a reference of itself to a global orb base class array.

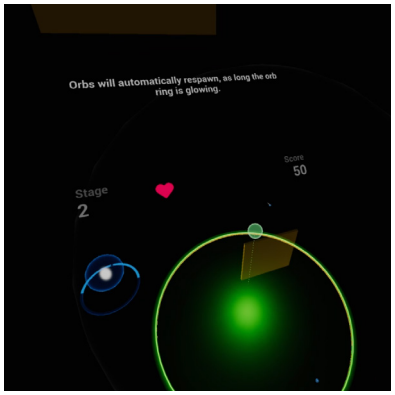
Destroy - removes itself from that array and destroy itself while spawning an explosion with sound.

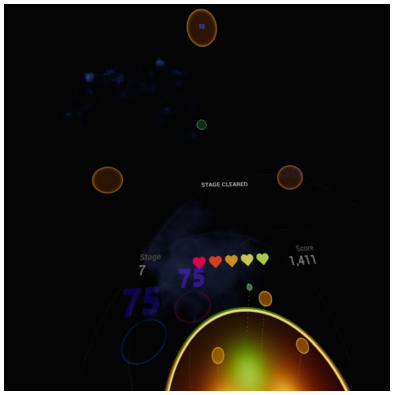
The Activate function is called after the player has thrown the orb, while the Destroy function can be called by other game objects, that want this orb to be removed from the game. This can be the case when obstacles detect a collision with this orb.

Default Orb: ***Stop Orb***

The stop orb has an additional function, which sets the velocity of the orb to zero. It can be called by the pawn. The pawn remembers every latest thrown actor. On ability button press it will call exactly this function and bring this orb to hold.





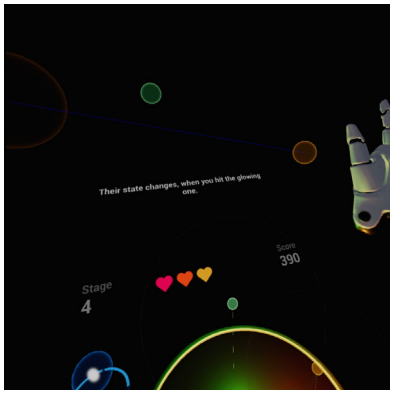


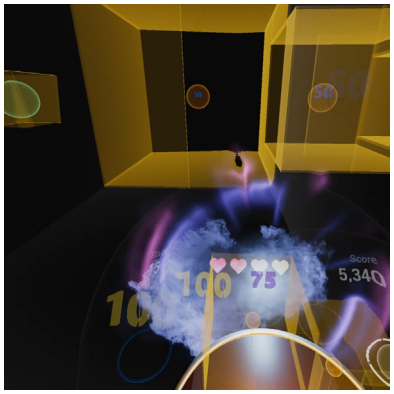
Default Orb: ***Thruster Orb***

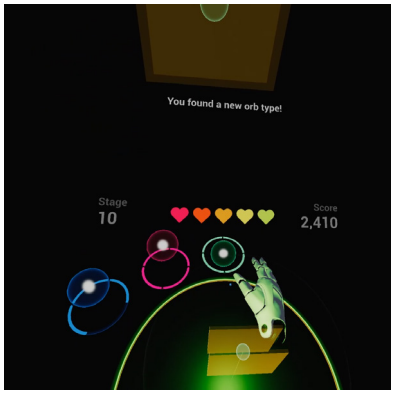
The thruster orb works quite similar. The difference is, that its ability function will increase the current velocity by multiplying it with a factor as long the ability button is being hold down.

Default Orb: ***Destroyer Orb***

The destroyer orb has an extra function, which calls the destroy function through a blueprint interface. Blueprint interfaces allow game objects to call other game objects functions without having to cast them, nor to know their class. They are especially useful in order to call widely generic functions. This destroy function is a perfect example: I was not sure yet, which objects I want this orb to be able to destroy. Sure, I ended up with the destroyer orb only being capable of destroying spheres. Therefore, I could have also simply cast the other game object to a sphere class and then call the destroy function of the sphere class. But if new game objects would have been added during development, which are supposed to be destructible, I do not want to add another cast attempt to the function each and every time. That is where blueprint interfaces are coming in handy. The same blueprint interface can be added to any blueprint class type. Therefore it is possible to call desired functions of different class types through the same blueprint interface. In my explicit example, I have created a generic blueprint interface. This interface has a function called 'Destroy'. I added this interface to the default sphere blueprint class. In this default sphere blueprint class it is set up in a way, that the interface function 'Destroy' will call the sphere to be destroyed. I can now call the interface 'Destroy' function from the destroyer orb, without it having to know what class it wants to destroy. Whenever I want another game object to be destroyable by the destroy-



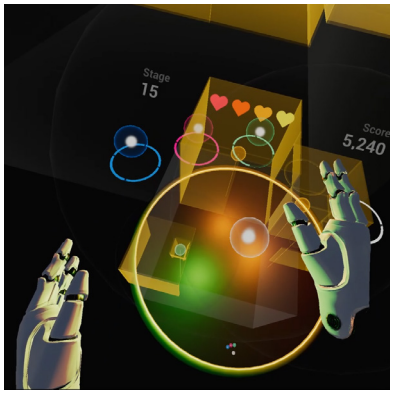




er orb, I just add exactly this generic blueprint interface to this game object and set it up in a similar way.

Default Orb: ***Gravity Orb***

For the gravity orb I was overriding the physics settings, so that global gravity is being applied. I also changed its behavior in a way, that the Activate function will not be called after it has been thrown and that it will not be destroyed when the destroy function is being called. It seems strange to have it inherit traits from the default orb, when exactly those traits just gets overridden. But in my opinion, it is still belonging to the category of orbs and I rather have those traits to be overridden, than splitting it up to a new class. It is also not unlikely, that during development, decisions are made to add other crucial functions to the base class. Taking care of the right dependencies can spare a lot of work in a later stage.

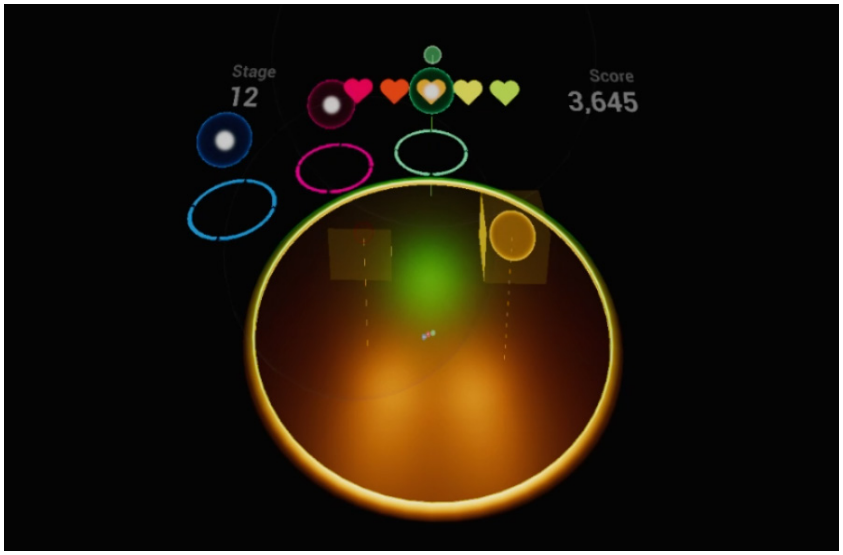


Orb Spawner

The orb spawner major responsibility is to supply the player with orbs to throw. It holds information about the orb type, the orbs count and latest offered orb. Whenever the latest offered orb was thrown and the player has orbs left, it will automatically call the function to spawn a new one.

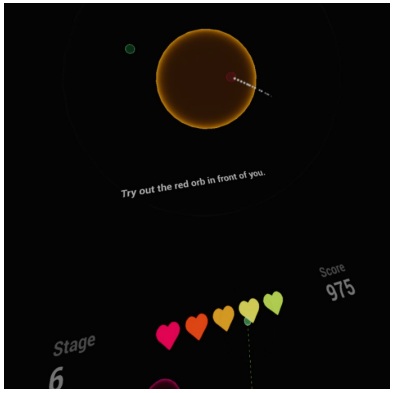
Base Station

The base station has no real functionality other than holding widget, holomap and orb spawners in one place. It is referenced in the game state blueprint and therefore allows for easy access of its child components from all other blueprint classes.



Obstacle

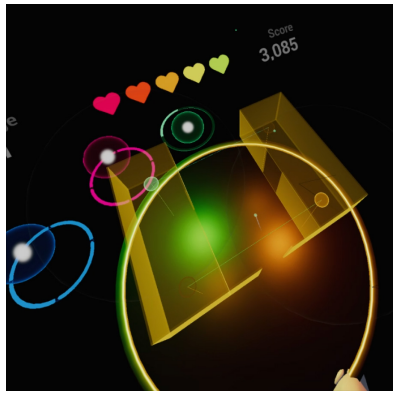
Obstacles are all the game entities, that can be placed in maps. They have a mesh and a collider. They call a destroy function on the other colliding object. Looking back, I would probably would have add another layer of two obstacle types. Meaning obstacles splitting up into “Hard Obstacles” and “Soft Obstacles”, where the hard ones would then carry the function to call the destroy function of the colliding object, while the soft one would not. If I would now want to add obstacles which do not destroy the colliding object, I would have to override this function. These are the examples, where I see room for improvement on my side in terms of planing dependencies.

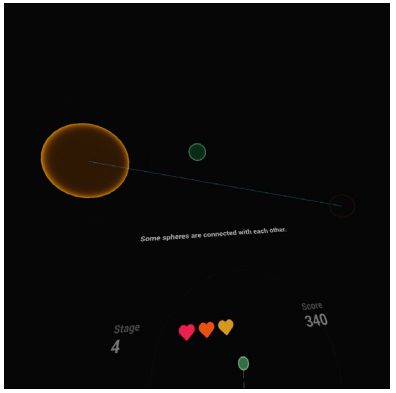


Obstacles: ***Default Sphere***

The default spheres main function is to apply gravitation forces depending on the size and distance to each and every orb. It looks at its own size and runs it through some equations, which results in gravitation force and maximum effective distance. It has access to all active orbs, through the global orbs array, which was mentioned in the section of the *default orb*. It evaluates for each orb, whether it is in its effective range and if so, it will apply gravitation force depending on the distance within the effective range. The applied force starts from almost zero and goes slightly exponential to the center of the sphere.

Default spheres have a boolean variable called active and an integer variable called ‘groupIndex’. If this is set to 1 or higher, they will search for other spheres with the same





‘groupIndex’ and remember them. Whenever an active, interconnected sphere is being hit, it will toggle its own and others active status.

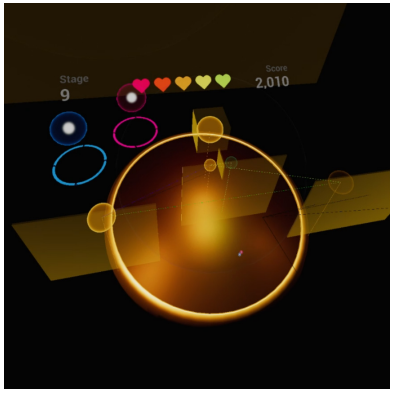
Default Sphere: ***Target Sphere***

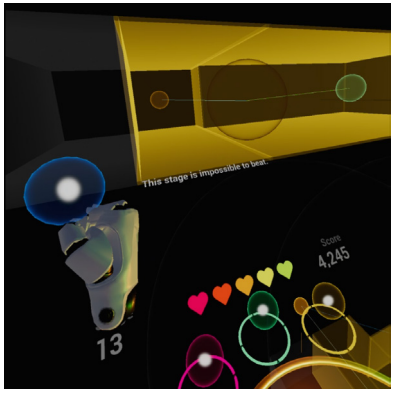
I have created a stage blueprint interface . This interface is constantly informing the current stage about all events. This way the stage knows, when an orb is picked up,

put back, thrown or colliding with something. The Target Sphere enables the stage to distinguish, whether a default or a target sphere was hit.

Default Sphere: ***Trigger Sphere***

The trigger sphere adds another sphere type to inform the stage on hit. In addition to that, it destroys itself after a hit.





Default Sphere: ***Alphabet Sphere***

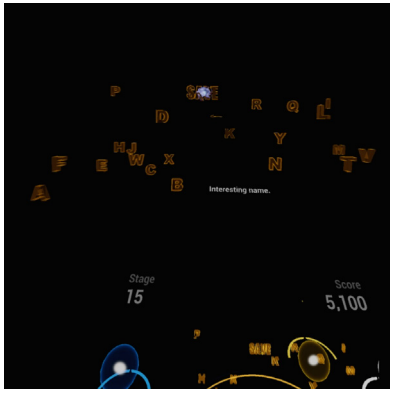
The alphabet sphere replaces the sphere mesh with an three dimensional letter mesh. For that I have converted fonts into meshes using the text option in Autodesk Maya. There is also an additional function added, which appends the corresponding letter to the gamer tag string on hit.

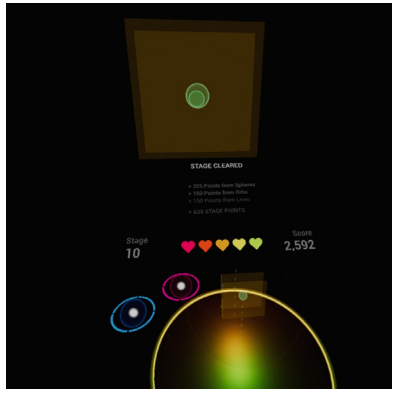
Default Sphere: ***Delete Sphere***

The delete sphere has a function which deletes the last character of the gamer tag string on hit

Default Sphere: ***Save Sphere***

The save sphere calls the Save High Score function of the alphabet stage blueprint on hit.





Obstacles: ***Wall***

Walls are simple obstacles with a cube as mesh.

Mover

The mover can be placed into the map along with target points. It will first attach all obstacles with the same group index to itself and then move along the set up target points. This way, it is possible to make planets or walls move in a specific pattern. For that, I have been adding a Moving Group Index variable to the obstacles class.

Volume

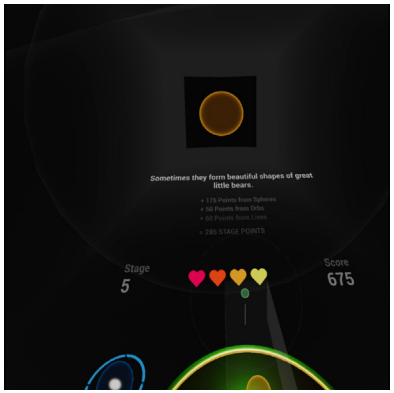
The volume is a collider, which affects the behavior of orbs, while they are overlapping. It removes the orb from the global array list on overlap enter, which effectively removes the reference of the orb for all spheres and results in no gravitation to be applied.

Widget

The widget constantly pulls data from the game state blueprint and displays information accordingly.

Stage

Stage blueprints can be seen as container with all information of the current stage. They handle all stage related operations, like for example adding lives and orbs or triggering text messages and events.

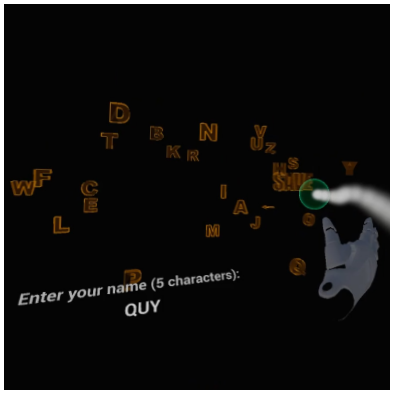


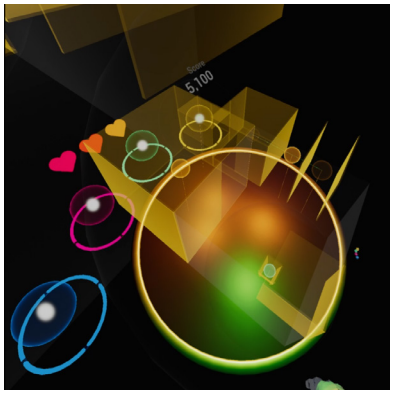
Stage: ***Default Stage***

There are sixteen different default stage blueprints in this game, each representing an unique stage in the game. (Starting from stage 0 as the title screen) Each stage is a child class of the stage base blueprint but are then individually set up to pose different problems for the player to solve. Each default stage stores all level elements. From the layout to what action triggers which behavior.

Stage: ***Alphabet Stage***

The alphabet stage has additional functions to be able to enter a gamer tag and handle the high scores. This includes the function to load and save game data containing high scores. Following operations are called in this given order, when saving a new high score: Add new high score to high score array, sort high score array according to the score from highest to lowest, cull high score array to the length of maximum ten items.





The high score array is an array of a custom struct. One struct contains gamer tag (string) and score (integer).

Stage: ***Game Over Stage***

The game over stage has additional functions to load and set the visibility of the high scores in the widget. It will also trigger the credits text and automatically load back into stage 0, which is the title screen, after a set time period.

Holomap

The holomap does not hold any functionality, other being a position in space. It represents the anchor from where the miniature holographic duplicates are replicated. This allows the holomap to be moved to any desired location.

The miniature replications are handled by all of the big counterparts themselves. Each obstacle item holds a reference to the holomap and updates a miniature version of itself onto this location.

Game State

The game state holds information about all globally relevant variables. I used the game state for this purpose, because it is easy accessible through the Get Game State node. It also checks, whether game objects, which are tagged as necessary to beat the stage, are still available. It will call the stage to fail if it thinks, that there is no way to beat the stage is anymore. (For example in case all necessary orbs are running out.)

Looking back, I am not satisfied with having this conditional check run by the game state blueprint and would probably rather let the stage blueprint handle those checks. Because this check is in the game state, it will be applied on all stages. This might cause a problem, when I decide to expand the game with completely different types of stages, which require another form of check.

Right now there is an array in the game state with all the playable stages listed. The game state will cycle through this array accordingly and load up the desired stages. It is a neat feature, but I would like to move this feature to another blueprint class. (e.g. level manager blueprint class) The score in the widget counts up instead of being set directly to the new score. This counting up is also handled by the game state. Yet again, I am not satisfied with this decision and would move this little function to the widget itself.

I am aware of the fact, that I abused the game state blueprint to handle too many different aspects of the game. The game state blueprint is untidy because of that. I will keep the game state blueprint as a data holding blueprint only in the future.

Review

Pitfalls & Best Practices · Useful Tips · Conclusion

Review

~Pitfalls & Best Practices~

During the development of this project, I was facing a lot of problems to solve. This section will go into more detail about the major problems and pitfalls I have encountered and how I was going about dealing with them. A lot of the following guidelines are based on common sense, but due to their importance I felt like including them anyways.

Keeping the Project Clean

One of the major efforts went into keeping clean and logical dependencies of classes and functions. Nonetheless, I ended up with some untidy blueprint classes carrying functions, which do not belong to them. I also felt unsure about the granularity of the devisions. There are no universal applicable rules on how to divide classes into subclasses. This often varies from project to project.

Best Practices:

I believe, that one can practice to get a better feeling for this by carefully planing, constantly reviewing and reflecting over the current but also previous projects. Having this awareness will help to make the right decision next time.

I also suggest to put effort into keeping the event graph clean. Try to have everything in descriptive functions instead. This will improve the readability of the class and its functions. Reading a massive event graph is quite a challenge and often times confuses oneself.

Functions, which need to run every tick for a period of time can be run by timelines. Do not plug them into event tick, unless they need to be constantly running.

The most obvious one is to use comment boxes. I myself, often catch me, underutilizing this option.

Multiple 3D Widgets

I have made some bad experience with multiple widgets. The editor kept on crashing occasionally without any identifiable pattern. Crashes were not only occurring during 'Play in Editor' (PIE), but also when idling in the editor at any given time. It could happen within one hour after opening the project or only after half a day of work. I got very paranoid during this time frame and started to work extra slow and carefully. This was particular frustrating and it took me a few days to finally narrow down the problem to the widgets. I probably lost 4 days of work, because of this issue.

Best Practice:

I probably could have lost more days on this issue, if I had not set up a revision control system. It helped me to keep a history of all changes made over the course of the project and allowed me to roll back to a previous revision whenever an unidentifiable error occurred. In this case, it was odd, because I got no information on what caused the crash and had to roll back to an earlier version several times, until I reached a version before I have added the widget.

Packaging the Project

I know, that many students, including me, had some problems with packaging projects in the past. I do occasionally try to package my project during development, to see whether it fails or finishes. I would advise to do this frequently. At the beginning of this project the chances of a successful cooking process was quite low. It can be very bothering to not know, how much additional time will be needed at the end to solve all packaging problems. Especially when it comes to hitting certain deadlines, it can become a serious problem.

Best Practice:

I started to always keep the output log window open on a second screen. This allowed me to keep an eye on it at any given time during development. I realized, that I have overseen errors, which are not being displayed in the debug window. This way I was able to solve all minor errors and warnings, before they evolved into something big. After I have adapted this habit, I suddenly did not have any problems with packaging the project anymore. It just always works.

~ Lessons Learned ~

With every project I gain experience and learn valuable lessons. Here are some of which I would like to highlight, because I personally find them useful.

Call to Parent Function

Adding an event node from the parent class to the child class will override the parent event. In order to run both (child and parent event), right click on the event node and select 'add call to parent function'. This will add a node to the child class, which allows the child class to call the parent event of this particular event.

Blueprints Interfaces

"A Blueprint Interface is a collection of one or more functions - name only, no implementation - that can be added to other Blueprints. Any Blueprint that has the Interface added is guaranteed to have those functions." (UE4 documentation)

Blueprint Interfaces are a great way to let blueprints share and send data to each other, especially if casting to a specific blueprint class becomes inconvenient.

Game State

Game State is a great class to monitor the current state of the game and have this information accessible for all other blueprint classes. It is easy to access through the 'Get Game State' function.

Enums and Structs

Although this is a very obvious one, I do find myself underutilizing custom enums and custom structs. It often helps to use custom enums

and structs to improve the clarity of a project. For example: Instead of using and integer variable called, "orbTypeIndex", use an custom enum called "enum_OrbTypes" and add descriptive items.

Property Matrix

When properties of multiple objects, which are sharing these properties need to be edited, they can all be edited at the same time using the Property Matrix. Select all desired objects, right click on them, hover over Asset Actions, click on 'Bulk Edit via Property Matrix...' This feature can handle thousands of objects at the same time. (UE4 documentation)

~ Conclusion ~

There are still a lot of areas in which the game can be improved. Further development would include an audiovisual overhaul, as well as further balancing of the orb movement. Although a lot of care has been taken for proper planing, the code still got untidy in some places. I found it difficult to keep the event graph tidy. Especially the game state blueprint, was suffering as a multi task class, due to its easy access from other blueprints. Furthermore, I had a hard time to decide on the granularity of creating subclasses. Too little would end up in huge chunks, which are potentially less tidy, while too many can cause unnecessary overhead.

I know now where I can improve and believe, that I have reached most of my goals in setting up a foundation, which can be easily extended for further development.

I am certain, that this project helped me to improve my workflow for future development and that I will be even more aware of keeping a clean structure. I was able to overcome all obstacles in the way and execute the development without any major hiccups by following my best practice rules.

I believe, that this project posed a good practice especially because I continuously reflected over my decisions. This experience will carry over to future projects and help me to make batter decisions.

Because of the focus on establishing a solid framework, adding more content in form of extra stages became very easy. Due to the modular system it is purely about arranging the layout, setting few triggers and adding it to the level array. This circumstance allows for more time being spent in actually designing levels rather than solving implementation problems. I believe, that this fact is noticeable, while playing my game. I can now already build a lot of more content by just using the current available modules, but also adding new modules and features should be fairly easy.

Considering the amount of quality content I have been able to generate in this short amount of time, I would mark this project as a success.

Appendix

Table of Figures - References - Attachments - Declaration on Oath

~ Table of Figures ~

~ *Sources* ~

Figure 1 on page 05:

Rift [Official Oculus Rift website]. Retrieved from https://www.oculus.com/rift/

Figure 2 on page 05:

Rift [Official Oculus Rift website]. Retrieved from https://www.oculus.com/rift/

Figure 3 on page 05:

Rift [Official Oculus Rift website]. Retrieved from https://www.oculus.com/rift/

Figure 5 on page 10:

Prosser, G. (2017, January 1st). Sagittarius [Game entry at itch.io]. Retrieved from https://gprosser.itch.io/sagittarius

Figure 6 on page 10:

Prosser, G. (2017, January 1st). Sagittarius [Game entry at itch.io]. Retrieved from https://gprosser.itch.io/sagittarius

Quote 1 on page 24:

Blueprint Interface [Online documentation]. Retrieved from https://docs.unrealengine.com/latest/INT/Engine/Blueprints/UserGuide/Types/Interface/

Quote 2 on page 25:

Property Matrix [Online documentation]. Retrieved from https://docs.unrealengine.com/latest/INT/Engine/UI/PropertyMatrix/

Eidesstattliche Versicherung

Hiermit erkläre ich an Eides Statt, dass ich die vorliegende Arbeit selbständig und nur unter Zuhilfenahme der ausgewiesenen Hilfsmittel angefertigt habe. Alle aus fremden Quellen im Wortlaut oder dem Sinn nach entnommenen Aussagen sind durch Angaben der Herkunft kenntlich gemacht.

Die Arbeit wurde bisher keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

18.09.2017

Hien Quy Tran

Acknowledgment

My Sister · Special Thanks

~ I would like to express my very great appreciation to my sister for her support during the course of my studies. ~

Special Thanks to:

My Parents
Grita Balkute
Susanne Brandhorst
Thomas Bremer
Fabian Golz
Julien Heimann
Mona Leinung
Nico Paech
Svetlana Sobcenko