Table of Contents

1. Project Description and Scope

ArenaRails is a competitive multiplayer game, in which futuristic hovertanks are used in combat on a complex rail-system inside an arena for the sole purpose of entertainment. Players are able to control both the path of their own vehicle, and the vehicle of the other player, by directing fire towards the crossings and thereby switching the routes. This is necessary to direct the hovertanks to useful pickup-items, guarded zones, areas that facilitate or hinder enemy attacks, or checkpoints used to give a player an advantage. This forces the player to consider whether attacking his enemy, or redirecting him, would bring him closer to a victory at any given time.

This prototype presents the core elements of this new and quite unique type of a competitive multiplayer game. The idea for the basic gameplay is actually almost two years old, but remained inside my head until shortly before I started with my preparations. The project's target was to create a working multiplayer prototype to find out whether the game is fun to play, and eventually even to watch. Also I wanted to develop an aesthetic for the world the game takes place in. Through the final prototype, I would like to exhibit my widely spread skillset in programming, 3D-modelling, game design and visual development, focussing on programming the necessary gameplay and the level design. Originally I planned to have a completely finished and proven visual concept, however this will need to be implemented after the deadline.

Over the course of the project, I faced several major obstacles. Working alone was a conscious decision. On the one hand, I was in charge, I didn't have to rely on anyone else, and could (try to) bring in my ideas as it pleased me. On the other hand, I had the full workload, and I had no-one to motivate and manage me but myself.

Furthermore I'm not a highly skilled concept artist, and it was hard to realize my vision for the visual world-design on paper. Combining this with the fact that I attempted to develop a completely new style of gameplay led to a shift of priorities in the visual area to create a clear and functional design for the interface, including the gameplay-relevant objects inside the game, which left the development of a strong visual-style and all the "eye-candy" and assets behind.
And then, of course, there was time. Three months is an incredibly short amount of time, especially when christmas and new year's eve lie within.

2. Research, Inspiration and Preparation

2.1. Other Games for Reference

It's generally a good idea to look at what others did before, point out what you liked in a certain game and eventually try to improve or adapt it. For my prototype I picked some elements in other games which I enjoyed playing or impressed me with their look and feel.

Tribes: Ascend

When it comes to the competitive multiplayer part, Tribes: Ascend clearly had a big influence on me. Tribes claims to be the fastest game in its genre and certainly requires skill and training. I picked two elements from that game: the ability to build up speed by utilizing the characteristics of the various levels and the behavior of fired projectiles.

In Tribes, the player can decide to turn off his friction on ground. Standing on uneven areas results in a smooth downhill acceleration. This is how players build up speed and gain an advantage as they are harder to hit.

The rather slow projectiles in the Tribes series inherit a certain amount of velocity from the movement of the player. So if a player is moving quickly in a northerly direction and fires to the west, the projectile will travel mainly to the west with a slight but recognizable northern movement. This seemingly natural and physically accurate behavior is rather rare in the competitive shooter genre, which may be due to competitive shooters rarely employing slower projectiles. However, the high movement speeds of the players in contrast to the slow projectiles AND the velocity inheritance make it quite challenging to hit other players, as it requires a very good anticipation of movement. This results in a direct hit feeling satisfying and rewarding.

Gameplay Video: http://youtu.be/r4AtN4H6d4g

Official Website: https://account.hirezstudios.com/tribesascend/

Split/Second Velocity

An arcade racing game, Split/Second Velocity allows the players to use the environment surrounding the racing track against their opponents by triggering explosions, releasing heavy objects in front of them, or even completely and permanently changing the track. The camera movement and effects strongly contribute to the intensity and atmosphere of the game, and the clean and reduced interface is directly clamped to the car and not just some numbers in the corners of the screen. What is also uncommon in other racing games is the use of music. While other racing games usually just play modern titles of more or less well fitting bands, Split/Second had its own soundtrack that dynamically adapts to the state of the game.

Gameplay Video: http://youtu.be/hU-JdmBjTQ8
Official Website: http://disney.go.com/disneyinteractivestudios/splitsecond/

Dead Space

Dead Space is a Third-Person-Horror-Shooter. Although I already mentioned the interface being "inside" the game of Split/Second, Dead Space incorporates certain interface elements as part of the in-game scenery and character design. As the camera floats behind the player's character, the developers simply put the health bar on the character's back, built in as a physical part of his suit. I wanted to use the same technique for my tanks.

Gameplay Video: http://youtu.be/JVSV-yRDaCI
Official Website: http://www.ea.com/dead-space

Mario Kart

Mario Kart is not the only arcade racing game (but probably the first) that introduced items in race games to attack other players with. Items are stored in boxes marked with a question mark. When a player drives through one of these, the box will disappear and the player gets a randomly selected item from it. The item can be used at any time after being collected and the box will reappear at a certain amount of time. As long as a player has an item, driving through other boxes won't have any effect besides causing the box to disappear.

Gameplay Video: http://youtu.be/u4XQ-Eo2-T4
Official Website: http://www.mariokart.com

Doom Rails

Although I didn't play it, this game might be the one that is most comparable to this project, based upon the few videos and teasers I could find. Cartoony carts are battling each other on a rollercoaster-like rail-system. However Doom Rails does not allow control of an opponent's path, as each player can directly steer in a desired direction when the opportunity arises. Doom Rails seems to be a flop, as it was very hard to even find proper reviews and ratings.

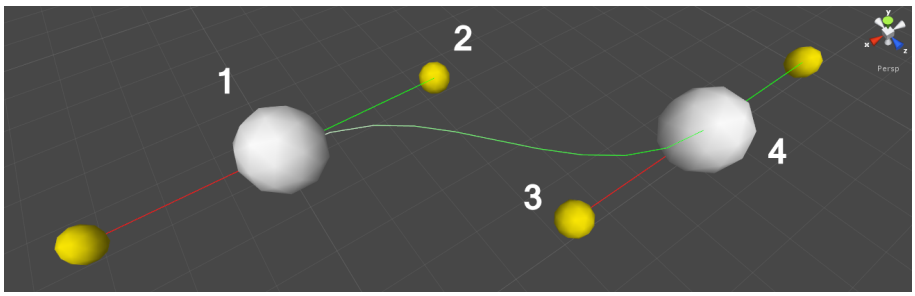
Gameplay Video: http://youtu.be/PfNNWffrf6g
Official Website: http://www.dreamforgegames.com/?page_id=20

2.2. The Node Tool

A major prerequisite to begin this project was the development of a tool to control the movement of the tanks and to make level design process faster and easier. For this I programmed something that allowed me to place nodes in the level and connect them as I needed.

First off I had to find a solution that calculated the curves between positions (vectors) in 3D-space to define the tracks. After a few trials the cubic Bezier curve (see: http://en.wikipedia.org/wiki/Bèzier_curve) seemed to solve my problem. It required four vectors to define a curve, whereby the curve starts and ends at the first and fourth vectors respectively and is influenced by the other two vectors in between. As such, my approach was to generate the vectors from invisible objects inside the level to allow the Bezier curves to be created at runtime. The end points of these curves will be referred to as nodes.

As the curves should go directly through these nodes, each node must be either the first or fourth vector required for the curve calculation. The other two vectors are defined by "Bezier points" that are automatically created when a node is placed. Each node has two of these Bezier points as they also define the in- and out-direction of the node.

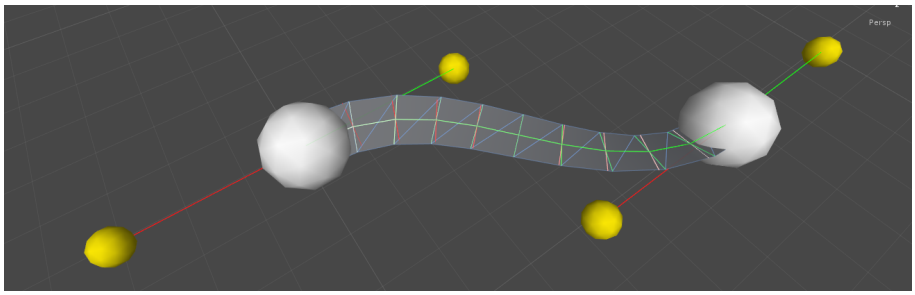


Two connected nodes (white spheres) and their Bezier points (yellow spheres). Numbers show which positions are used in the calculation of the resulting curve. Straight red lines represent the out-direction, straight green lines the in-direction.

To connect two nodes with each other, each node needs to know at which "end" it should connect, so the calculation process can pick the corresponding Bezier points it needs. Proper setup will result in a curve between the nodes.

Nodes can only connect to one node from the in-direction but to multiple nodes from the out-direction. This is necessary to allow for the possibility of route-switching. And after adding a banking angle to the nodes, that would lean the objects following the curves, the movement part was done.

However for level design purposes I needed more. So after learning the basis of low-level mesh generation, I used the curves to create a polygon-strip for each connection. This meant I didn't have to create a visual representation of the track manually and let me quickly iterate on the overall rail-system as a whole. This mesh generation can be set to run in real-time and facilitated a "what you see is what you get" editing process. After I found a method to export my generated mesh from Unity into an .obj file, I was able to quickly create a satisfactory track, export it, and create a beautified model for the track in another 3D-software at any point later in time.



Generated polygon-strip along one curve between two nodes.

3. Detailed Gameplay Description

3.1. Implemented Gameplay

a) General Rules and Mechanics

The current prototype has the common “death-match” mode as a basic ruleset. Killing opponents will give points. Players respawn as long as the game is running. The game ends after a certain amount of time, or when a player reaches a certain amount of points. If a player kills himself, he loses points.

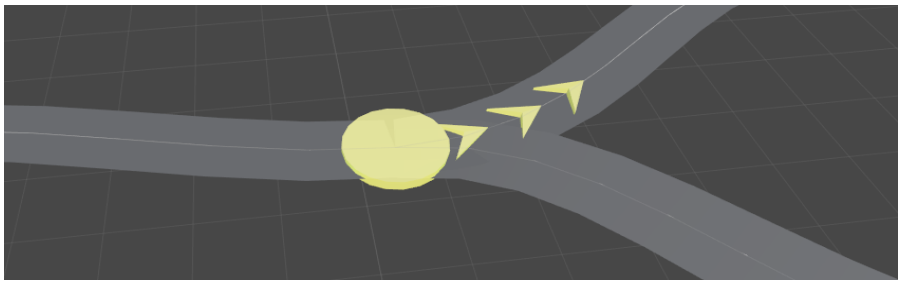
A player dies when his tank’s health-points (HP) are lowered to zero. Different types of tanks (see 3.3. for Tank Classes) have different amounts of HP. To lower an opponent’s HP, players can deal damage in several ways. Shooting an enemy with a main weapon deals an amount of damage dependent on the firing tanks damage value and whether the shot was a direct hit or if damage is dealt as the result of the projectile exploding nearby (splash damage).
Another way to deal damage is through collisions. Collision damage depends on the current speed, the maximum amount of HP, and the current amount of HP of each tank involved. The HP calculations represent the mass of the tanks. Collisions have an influence on both the current speed and the direction of the tanks.
Also, players can pick up items by collecting green boxes that are found at certain locations in the rail-system. Items will generally deal more damage than the main weapon of any tank .

A tank’s health will begin regenerating if the tank doesn't take any damage for a certain amount of time. Also players can decide to reduce the tank’s velocity and increase the regeneration rate. If the player increases speed, the tank slowly loses health. This will not destroy the tank though, as health can only be reduced to 1 by this process.

When a round starts, and when upon dying, players can choose one of several spawn positions from which their tank can enter the rail-system. The nodes that connect the spawn positions with the rail-system are the only nodes that have multiple outgoing connections but are not switches.

b) Playing with Switches

To collect to items, evade collisions or direct fire, players can control the rail-system by changing the switches. Switches are nodes that have at least 2 out-node connections, and thereby offer multiple curves for movement. When a player is approaching a node from the in-connection, his direction will be affected by the switch's currently active out-node which determines to which one of the multiple out-connections he will be led. Approaching a node from any of the multiple out-connections will always end in continuing to the in-connection.



A switch showing the currently active out-node through the three arrows.

To change the active out-node of a switch, players can either shoot at it or – if it is the switch they are currently approaching – they can control it directly with the A and D keys. Changing a switch's connection will "lock" the switch for a short amount of time, forbidding another change. This prevents the possibility of nullifying any changes that were just made and forces players to time their actions when they plan to change paths for themselves or other players.

c) Tank Classes

Before a match begins, each player has to select his tank. The tank will represent the player in the arena in the fight against other players. Different tanks have different values in the three categories of speed, damage and health points. Depending on the distribution of these values a tank has certain strengths and weaknesses which make it possible to classify them.

Rams: They have the highest values in speed and health points. As collision damage calculations rely on the speed and health of a tank, this is where a Ram's strength is. Their main cannon can only deal small amounts of damage and should mainly be used to control the switches and enforce collisions.

Kites: Their strength is speed and damage. They will be quicker out of dangerous situations, are harder to hit, and pack a large amount of firepower. However their lower health value doesn't allow them to take much damage and they should try to stay as far away from other tanks as possible.

Fortresses: Strong in damage and health points, these tanks lack speed. They can take more hits than the other classes, but as they are slow and easy to hit, they will be getting hit rather often. However it's a bad idea to stay in the line of sight of a Fortress, as their projectiles pack quite a punch.

d) Items

Classic shooters have always included several additional weapons positioned around the combat area. They create further possibilities for the players and become points of interest in the level, as they encourage players to try and collect them before the others do. In ArenaRails, this motivates players to make use of the switches for more than just collisions and cover. When being used, some items demand further attention of the players, as they may make some areas of the rail-system more dangerous. Item pickup locations are currently represented as green boxes on certain rails in the level. An item gets picked up, when a player drives through this box. The box will disappear for a certain amount of time after being picked up, preventing players from collecting another item at the same location. After colliding with a green item box, one of several possible items will be picked at random and gets visually attached to the player's tank. He can then choose to fire the item by pressing the right mouse button at any point in the game. As long as an item is attached to a player's tank, he won't be able to pick up another one, but driving through the green box will make the box disappear again and therefore hinder other players from getting an item at that location. Items will generally deal more damage than the main weapon of any of the tanks, but depending on the item, this can be dangerous for all players - even the one who launched it.

Sniper: A classic weapon in first-person shooters. In contrast to the tanks main weapons, the sniper fires a high-velocity projectile, which is not affected by gravity, and hits the target in the same moment as the weapon fires. So, provided the player aims properly, this weapon won't miss, and represents a secure way to deal damage to others without the option to get damaged by yourself.

Mine: When a player picks up a mine, he can place it anytime at his current location and drive away safely. As soon as another tank drives over the mine, it will detonate and deal a large amount of damage. With a well-placed mine, players can deter other players from entering certain areas of the rail-system. Furthermore, players can use mines to influence the path of an opponent, perhaps even directly towards the player himself. When a mine detonates, it disappears and the path is safe again. However mines don't differentiate between the player who placed them and the other players, so after placing a mine, the player might want to stay away from it himself.

Rail-Missile: The rail-missile is basically a moving mine. When a player launches it, it will follow the rail-system as any other tank would. It arms itself a short time after being fired, so that the launching tank can't trigger an immediate detonation and such that it can't be used to damage another player shortly before a collision. The missile moves very fast and may be somewhat predictable, but as the missile's movement is controlled by the switches, any player can control its path. This makes it the most dangerous weapon for any player and requires a significant amount of attention even after firing.

3.2. Unimplemented Gameplay-Possibilities

a) Spectator Mode

With the current growth of the E-Sport-scene and live-streaming of games and tournaments, I see the benefits of including a spectator mode in all new competitive and match-like games. A spectator mode allows others to join a game the same way as a player does, but restricts interaction merely watching the match unfold. A spectator can see the game from the point of view of each of the players, or to control an independently free-roaming camera. The spectators would also have access to additional information such as game statistics and player performance.

For ArenaRails I'd like to have more than this. Spectators should have the ability to let the players know what they think of, and how they feel about each player, just like an audience in real life. The arena's environment could facilitate the opportunity for the spectators to provide this. As in any modern stadium, the arena will have multiple big screens showing the action or providing information about the current match (e.g. times and scores). Some screens could display the camera view of a spectator. Other screens could display tables or statistics that are currently being viewed by the majority of spectators. Votes could even be held in regular intervals, asking the spectators which player he hopes will win. The results would then be display on screens inside the game. The spectators could also start cheers by controlling parts of the audience.

This way, the spectators could feel like their support is influencing their favorite players. This also puts pressure on the players as they would have to deal with being observed and judged; something that professionals have to deal with all the time. It could make a huge contribution to atmosphere, immersion and intensity. However this should remain an option for the players. When they create a game, they can decide whether or not others can spectate or even interact with the game-world as described above.

b) Additional Gameplay Modes

Besides the classic death-match and its team-based variant, ArenaRails provides a foundation that could support other game modes.

Capture the Flag: Probably one the favorite classic game modes in shooters. Two teams attempt to capture the flag of the opposing team. In ArenaRails, the hovertanks could pick up the flag like they pick up regular items. A team scores a point when they manage to return the opposing team's flag to their own flag-stand. In order to score, the team's own flag must be present when returning the opposing flag to their flag-stand.
Play-modes like this would require a skill in level design, as there have to be enough possibilities and paths to allow the players to maneuver. Also, player timings and communication get a higher priority than in other games, to avoid collisions with team members and intercept enemy flag-carriers.

Crazy Race: Providing a (more or less) classic racing play-mode in ArenaRails would be an interesting addition, as players can use the switches to disrupt any attempts by another player to get to the finishing line. With one (or multiple) checkpoints in the rail-system, players could kill their opponents, and use the advantage provided by the opponent's respawn delay to get to the finish and complete a lap without resistance. The player with the most laps in a given time-frame or completing the set number of laps first wins. This mode could also be played in teams and players can "collect" laps together.

Core Combat: The level would be divided into several ring-zones around a core. One team has to reach the core, while the other team has to defend it successfully in order to win. The ring-zones are connected with switches that begin locked. When certain objects in each zone are collected or destroyed, these switches would become unlocked; bringing the attacking team one zone closer to the core. The defending team needs to prevent the attacking team from achieving this. After the attacking team succeeds, the teams switch roles. After both teams have attacked, the team that needed less time to get to the core wins.

4. Visual Concept Development

Aircraft, spacecraft and vehicles that appear futuristic have always fascinated me and were my original motivation to begin 3D-modeling. I felt obliged to create a visual concept for my game idea that paid tribute to this inspiration.
During the project, I attempted matte painting for the first time, but I found out that I'm still far away from being a concept artist. The lack of these skills made it difficult to develop the game's look as it was in my imagination, and I focused instead on designing functional visualizations for the gameplay instead of good looking environments and tanks. The visual style and environment of ArenaRails should be a mix a several ages, representing their respective architecture or use.

4.1. Background Story

The game takes place in the distant future on far-away worlds - worlds that once were inhabited by large civilizations which have long since left. These civilizations became so large as they found a way of living in harmony with the planet by retrieving energy from certain minerals in the planet's surface, without consuming them. At areas with a high concentration of these minerals, the civilizations built large structures which served as both a place to live and as a way of honoring the planet itself. However, as no civilization can exist forever – be it because of catastrophes, plagues, famines or war – this civilization vanished, too. Only their structures remained. Although areas of erosion show the scars of time, the minerals and their energy had a conserving ability, holding the structure together and keeping it as a landmark.

An unclaimed energy-source like this could never be left alone for long. Long after the previous civilization had left, humanity had managed to leave earth behind and began to colonize worlds that were once beyond imagination. Many of these worlds contained these energy-giving minerals. Archaeologists were amazed to find the magnificent structures on each of them but were horrified when they discovered that many corporations had already sent their scouts to discover anything worth harvesting. The corporations built excavators, silos, and refineries directly inside the civilizations remains, drilling pipelines through the walls and floors to maximize the processing efficiency. What the corporations didn't know was that the minerals were not meant to be harvested. The more minerals the industry removed, the less energy they generated.

Each individual mineral's energy-output seemed to be connected to the overall amount of minerals lying in the planet's surface – a phenomenon that remains unexplained to this day. The corporations soon lost interest as all the minerals lowered their energy-output to below a cost-efficient value.

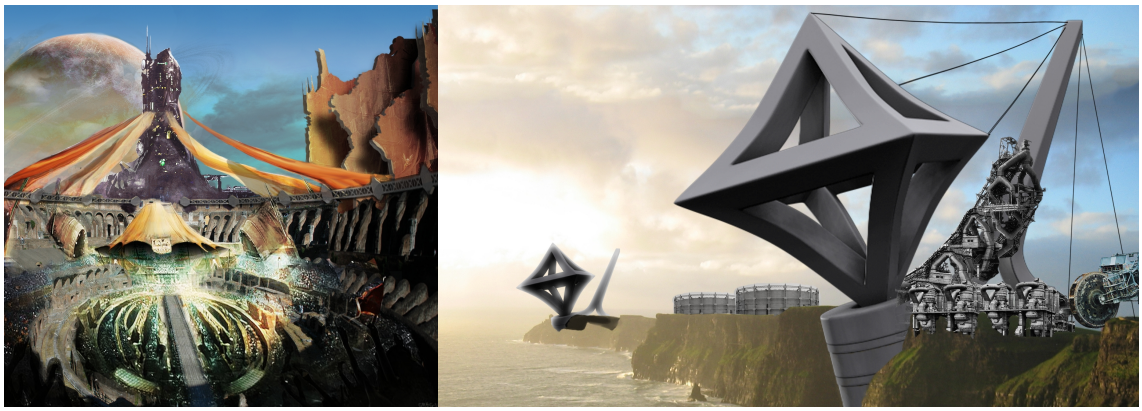
Yet what from one business leaves behind, others see profit. A rich racing-enthusiast bought several harvesting sites and created a new kind motor sports. The ancient ruins and the shutdown harvesting machinery served as an exciting arena for a new combination of race and war. Architects and engineers found a way to acquire enough power from the minerals to operate a complex magnetic rail-system that also passed on a supply of energy to any vehicle capable of using it. Universities of engineering, wealthy investors, and corporations wanting to prove their products all created teams to develop the drones that took part in this sport. The attention grew rapidly. With games being broadcast to all corners of the galaxy, it was received ambivalently, but pacifists couldn't complain as no pilots would sit inside the tanks and shield technology would protect the audiences.

4.2. Visual Development of the World

a) Alien Ruins

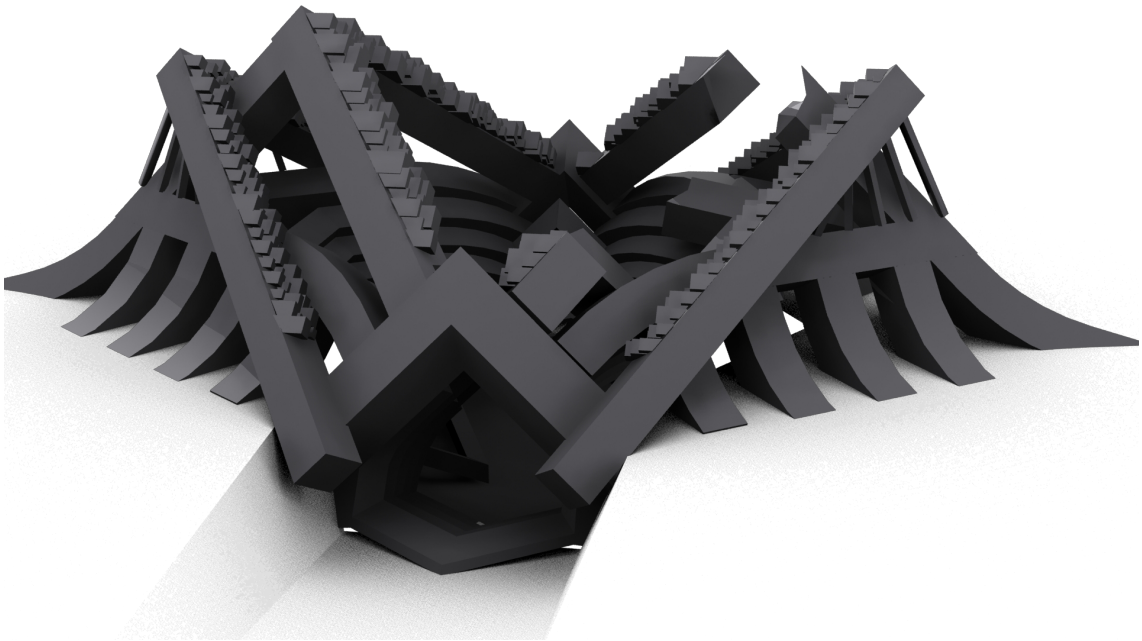
The remains of the old civilization form the environmental backdrop. They provide a feeling of being inside an arena – the players are gladiators, performing purely for the entertainment of the audience. The architecture is based on building straight and curved columns or overlaying walls from the basic shapes of triangles and squares.

The architectural process was easier for me to do in 3D than in 2D. As such, I mostly did rough concept models in 3DS Max, rendered an image and attempt to merge it with real-life footage of interesting landscapes. After that, I added the photographs of industrial equipment (mainly photographs by Bernd and Hilla Becher) to complete the environment.

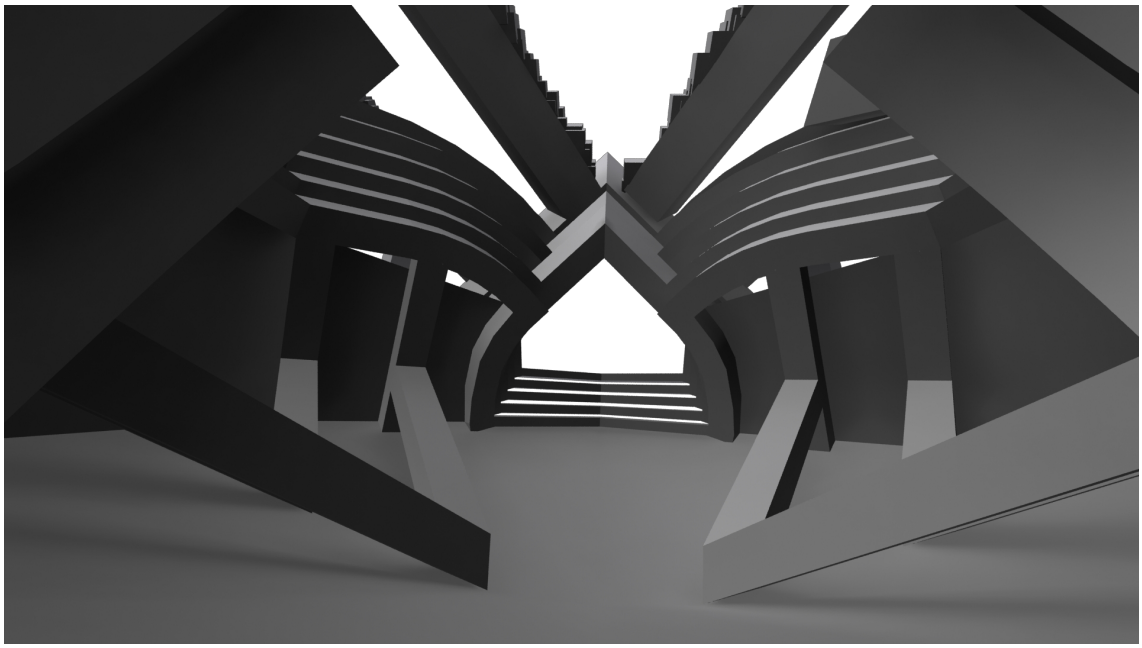


(left) Very first environmental concept made purely out of parts of other concept art. (right) 3D concept model in real-life footage of landscape and industrial structures.

However, I soon found out that this wasn't the best way of determining whether the feeling of being in an arena could be achieved. Here it was very helpful that I already had 3D models as I could simply place them inside the current game-prototype and look around. I quickly found that my very first 3D model was the only one to deliver this feeling.



Rough skeleton of the ancient structure. The houses of the civilization's population remain atop the framework.

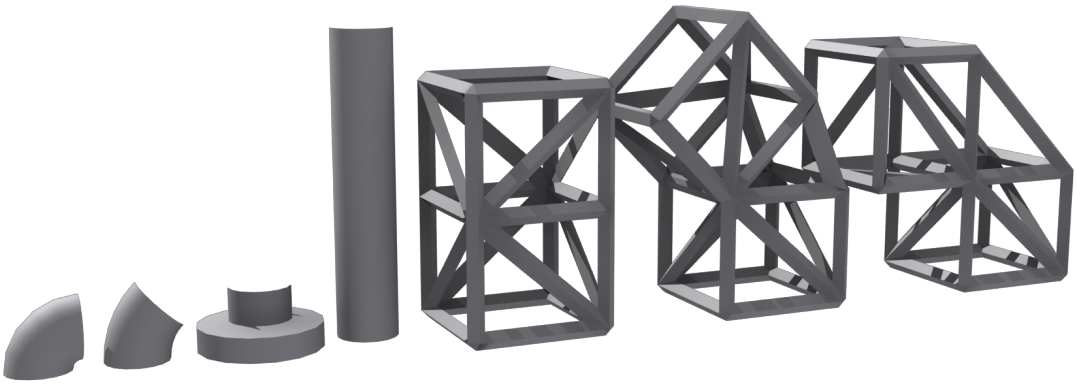


One perspective from inside the structure. The inward leaning of the architecture induces an oppressing feeling of being surrounded. In further development, stands for the audience can be added above, just like the romans did with the coliseum.

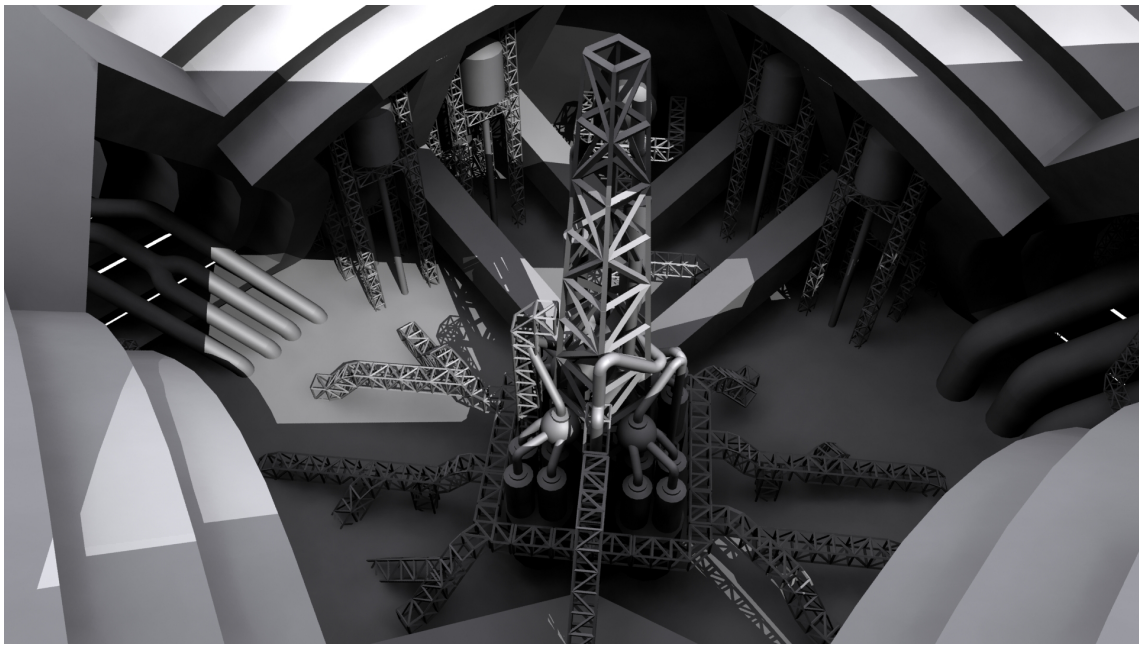
To enhance the feeling of belonging to an old ancient civilization I figured that the architecture should display aspects of daily life or an honoring of the planet. The minerals energy should also be represented, so I tried to combine it with the relief by adding glowing lines. I imagine that in further development, these lines might glow in the respective color of the player leading the scores to represent his status in the old environment. The pseudo-logical explanation for this would be some kind of energy back coupling with the minerals through the energy they supply the tanks with.

b) Industrial Energy Harvesting

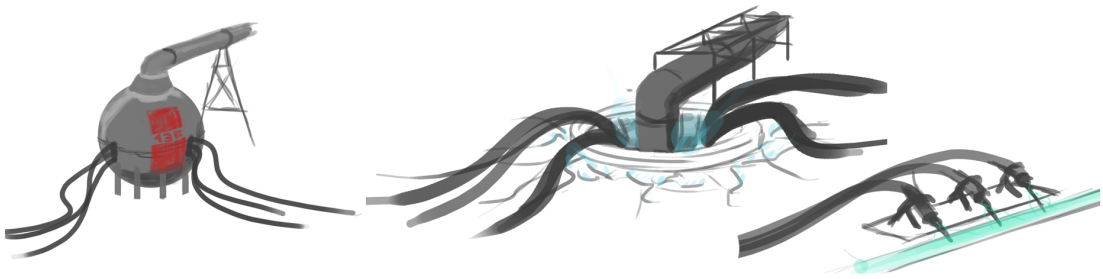
Being purely functional, the industrial elements didn't have to differ much from our current industrial sites. As I mentioned already, the pictures by Bernd and Hilla Becher provided good footage for matte paintings in the first concepts. However to prove whether the combination of ancient civilization and industrial sites could really work I wanted to have them in 3D again. Building a full-size industrial complex was no option as time was too short, so I reduced the industry to some basic things: pipes, storage tanks, and scaffold. I created a small construction kit containing the pieces I needed and just arranged them in a more or less logical way inside the ancient structure.



The construction kit for the most parts of the industrial 3D model.

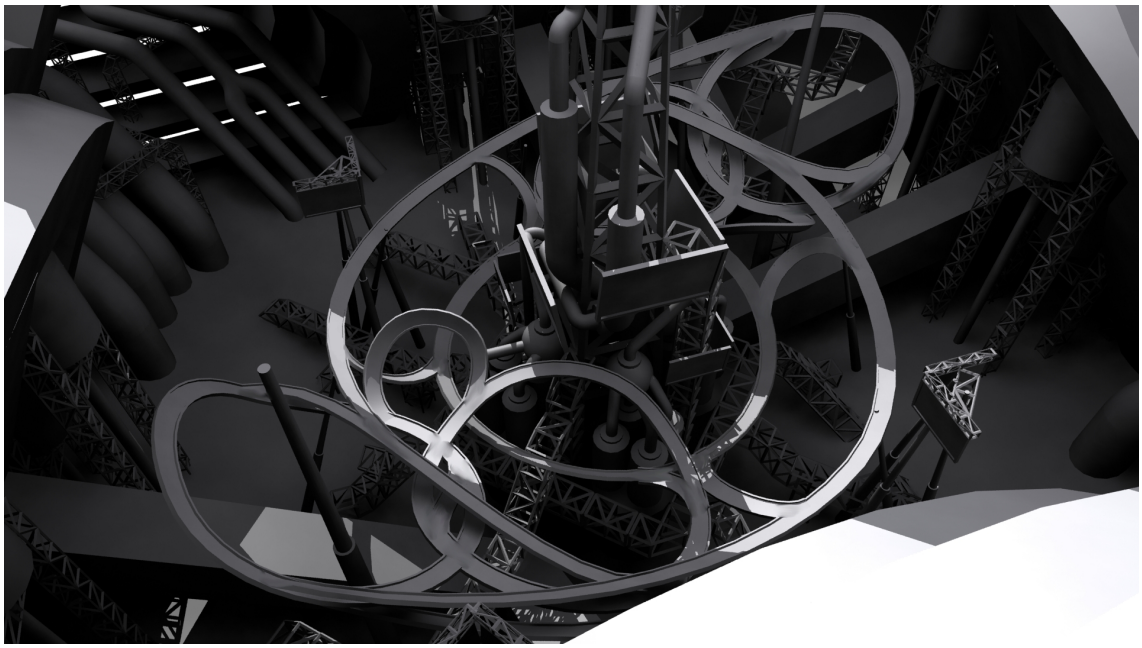


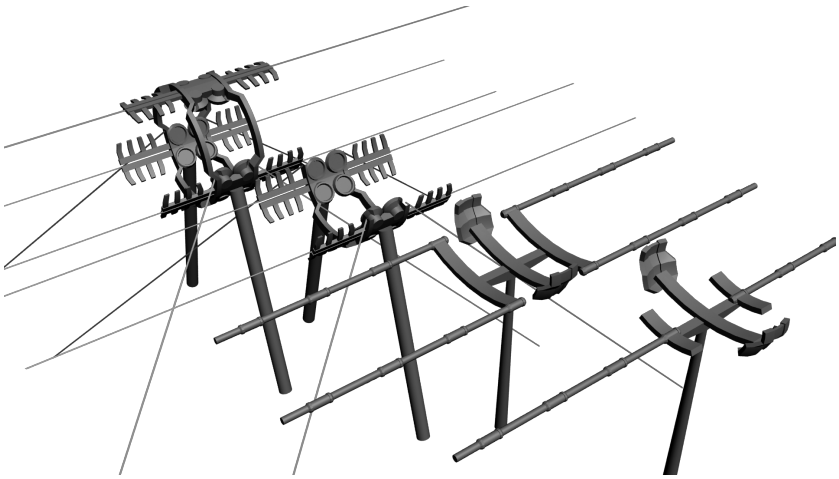
I also did some 2D concepts for the invasive methods the corporations used to extract the minerals' energy - pipes and hoses coming out of drilled and cracked holes in the floor, and needles "sucking" the energy from the energy-lines in the reliefs. The loose hoses lying around are a sign of how quickly the harvesting process started, and how fast it ended again.



c) Combat Track and Entertainment

Finally I had to create everything the rich racing-enthusiast brought in. The magnetic rails, giant screens, grand-stands, and so forth. These parts would need to look more modern and not purely functional like the industry pieces. I had the idea of making the rails like thin bars guiding a flow of visible energy, but this would make the track almost see-through and thereby induce confusion for the players. So I stuck to a simple, solid track.

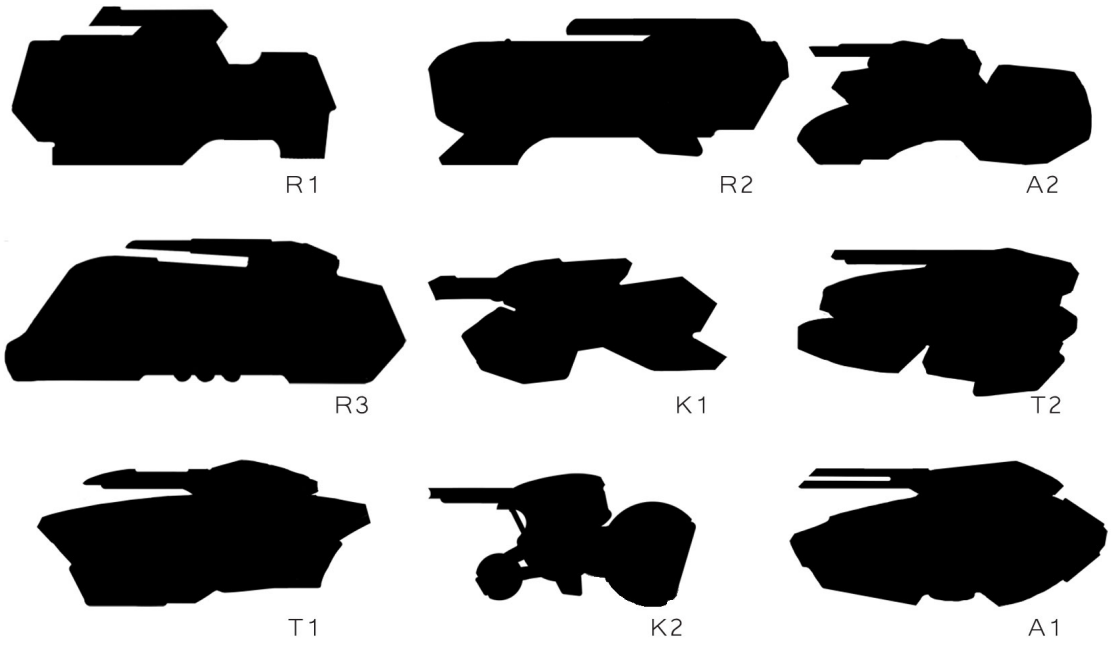




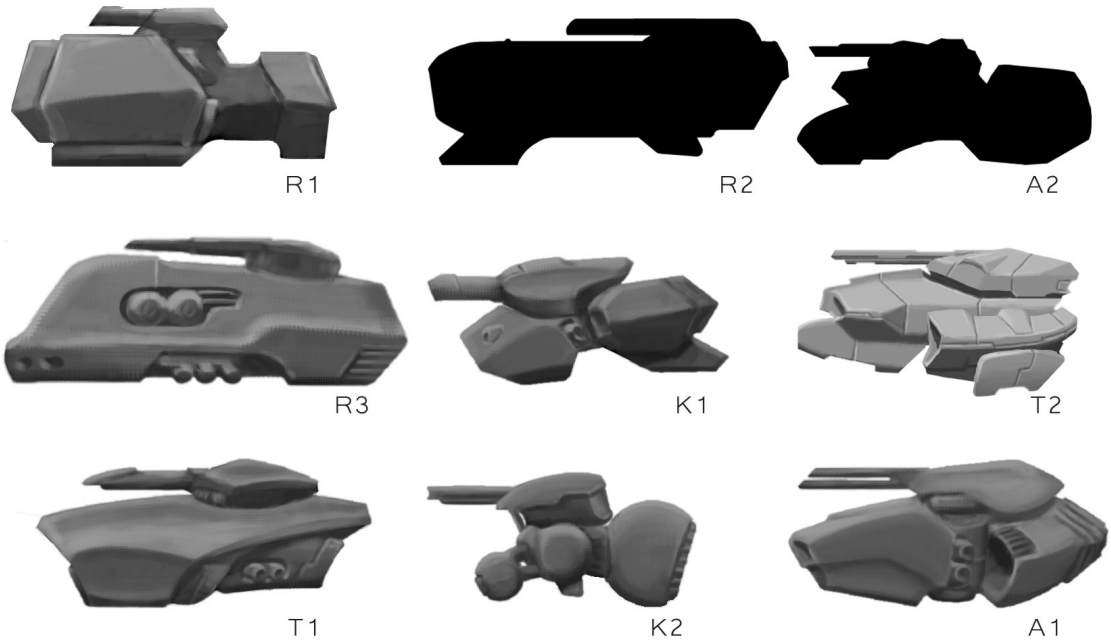
Various concepts of switches and the track.

3.3. Tank Development

Being the main "characters" in the game, the tanks required more design time as they receive a high level of attention. As players can pick their tanks before a fight, they needed to be visually appealing, while still representing their characteristics – specifically the health, speed, and damage attributes. I usually begin my designs for any kind of vehicle with a side view. This time I took it a step further and started with the silhouettes.



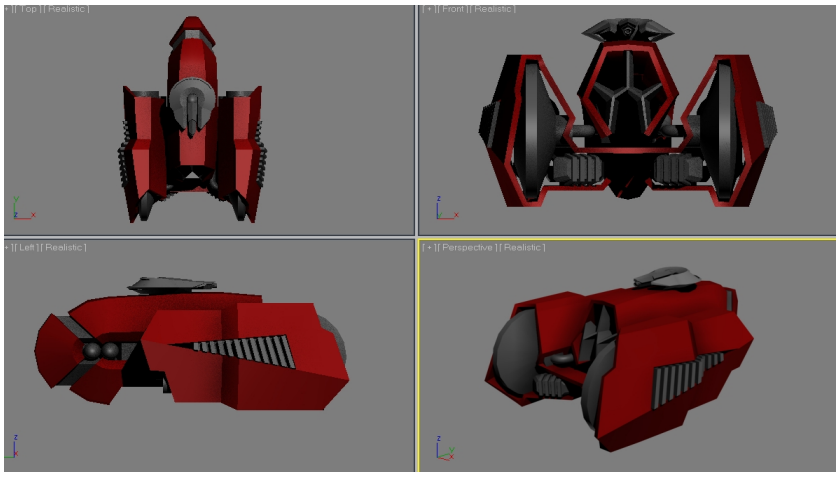
I wanted to develop seven concepts that covered seven specific distributions of the three values. After developing the silhouettes above, I surveyed my friends and asked which two of the nine designs they liked the least. Although their opinions differed strongly on the most tanks, some results were pretty clear (R1, R2 and R3 had the most dislikes, T1 and A1 the least).

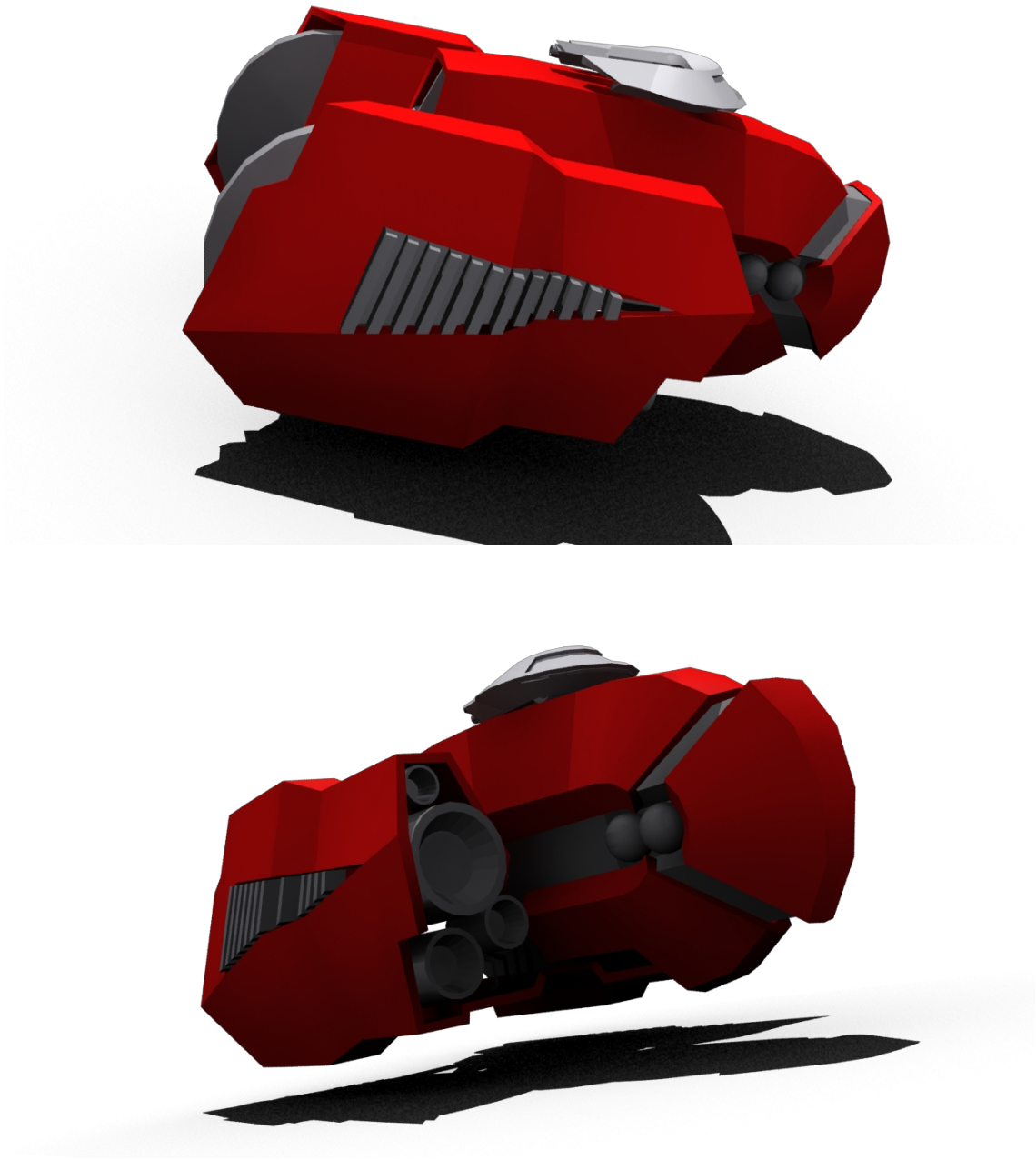


I decided to take their personal opinions as just that - personal opinions - and decided to leave R2 and A2 out.

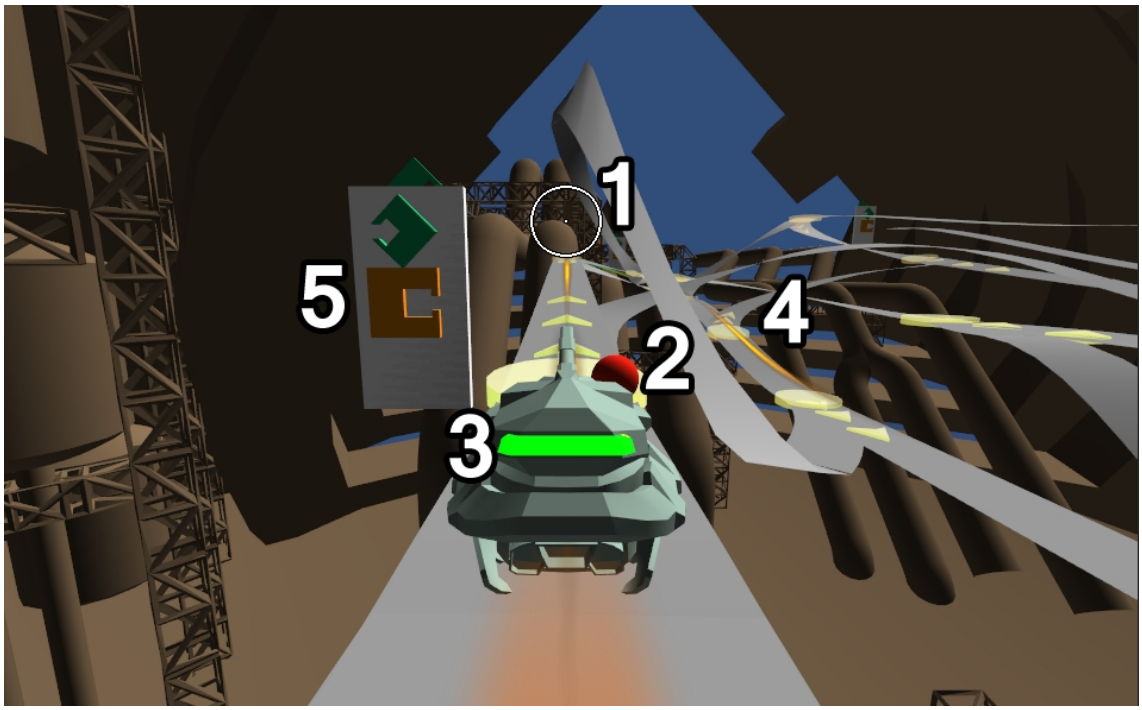
I then tried to give the silhouettes volume. After some more feedback I put the most effort in T2 and tried to verify the plans by making a rough 3D model which I could also use for in-game testing later.

Although I knew I wasn't satisfied, I left the vehicle design alone for a time to develop other aspects of the game. Towards the end of the project, I decided to develop the R1 model further and drew on inspiration from the design of model sports-cars, such as those from Lamborghini. Although it still is a rough model, I'm satisfied with the visual direction the model has taken. There is still more modeling to be done, but all that's missing are smaller details like air inlets and lights, a proper shading to show a shiny polished paintjob, and some sponsors and team logos on the hull.





4.4. Interface and Feedback



1.Crosshair

2.Currently collected Item

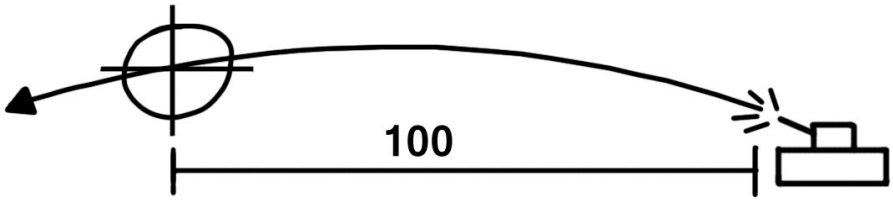
3.Healthbar

4.Prediction Line

5.Leaderboard

Crosshair

I wanted to keep as far away from any screen-attached interface as possible. All relevant information should be represented by objects inside the game. A task almost impossible for a game like ArenaRails. As aiming is one of the most required actions by the players it must stay convenient and familiar, so the typical crosshair remained. However because the crosshair knows the velocity of the projectile, the crosshair performs a calculation to angle the gun which results in the projectile reaching the vertical positioning of the crosshair after traveling a distance of 100 units (meters). This should make the aiming easier while still keeping the challenge of requiring the player to compensate for the velocity inheritance of the projectile.



To give the player feedback when his shot hits another tank, the crosshair shortly flashes red, and a number emerges from the hit tank which displays the amount of the dealt damage (aka. combat text). This will also happen if the damage was dealt by an item.

Items

Items are attached directly on the tank's hull. As the player's tank is nearly always visible on screen, he can immediately see which item he just picked up.

Health bar

The health bar (displaying the tanks remaining health points) is built into the tank's turret. As the rotation of the turret follows the player's camera rotation, the health bar will always face the player's camera and is visible as long as the player is driving a tank (and therefore has a health value). When the tank takes damage, the health bar will shortly flash red, and when the tank regenerates health, it flashes green.

Prediction Line

I wanted to avoid a mini-map on the HUD as racing or shooter games often have. Being aware of the state of the track, using the disadvantages or advantages of higher positions of the track, and avoiding being a sitting duck in the possibly confusing center of the track are all essential elements of the game. So I noticed that I needed to compensate the missing map. To help with predicting tank movement, animated lines show part of the upcoming path by simply moving faster than the respective tank and following the switches current directions. The lines use the player-colors, which are picked before the match starts. The currently implemented version of this concept isn't optimal, as the line doesn't update in the same instant that the switches change their out-nodes, and the full path of each tank isn't displayed.

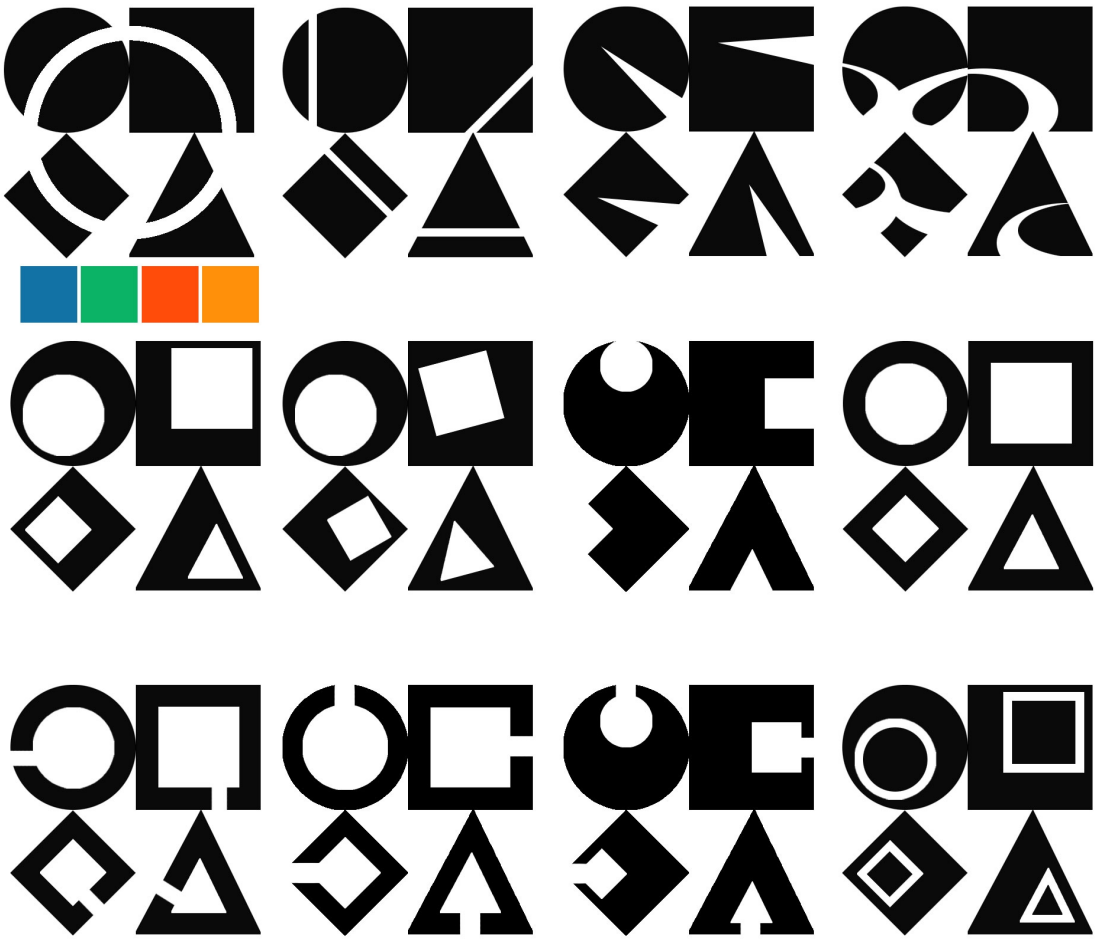
Also each fired projectile will draw a well visible trail in the firing players color so other players can anticipate which player shot from which position.

Leaderboard

A prototype of the screens displaying the match statistics and leaderboard is also implemented. To choose their colors before the match starts, the players select a symbol in this color. These symbols represent the players on all displays as this is quicker and

easier to distinguish than reading nicknames. The display prototype will simply show the current placement on the leaderboard in a top-to-bottom order using their respective symbols. For team-based play modes (e.g. 4 vs 4) each player of a team would have his own symbol and his own shade of the team color.

The symbols are basic shapes with a minor change to the original form. By placing openings in different positions for each shape, the symbols can be quickly distinguished from one-another.



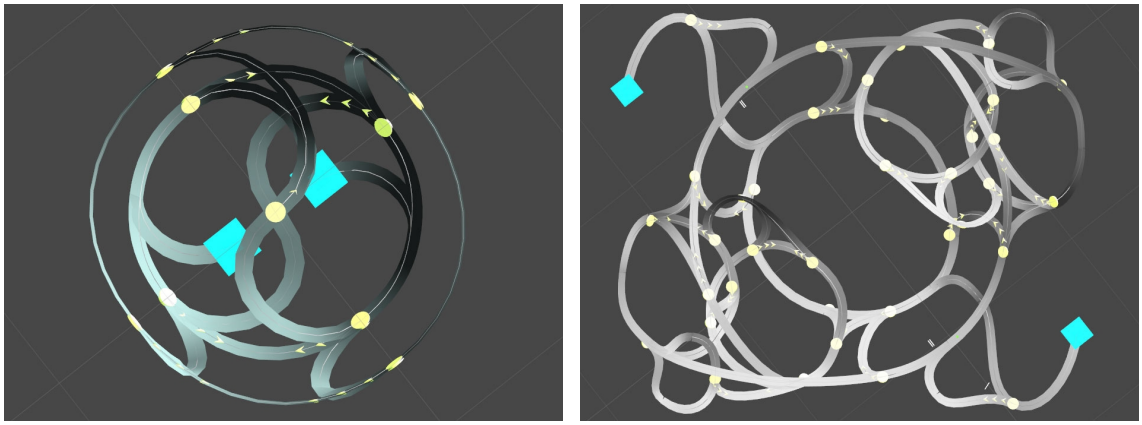
Basic shape study

5. Prototype Development

5.1. Level Design

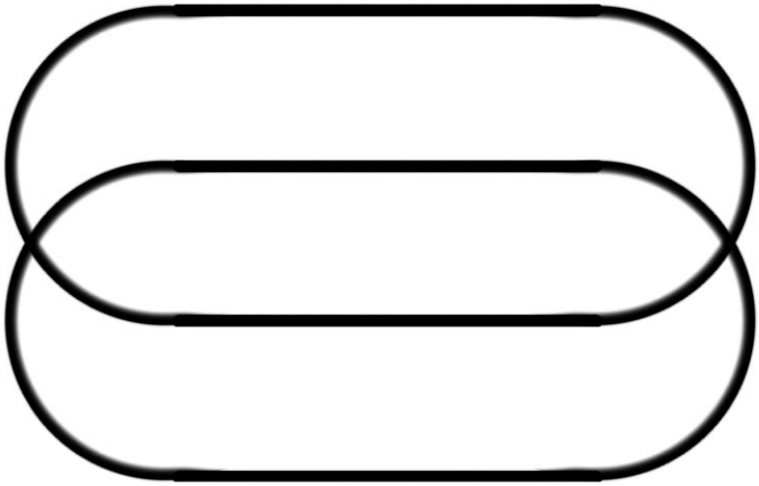
The level design is a major factor of ArenaRails gameplay as it based on the rail-system itself and describes all possible paths for the players.

a) Overview vs. Possibility

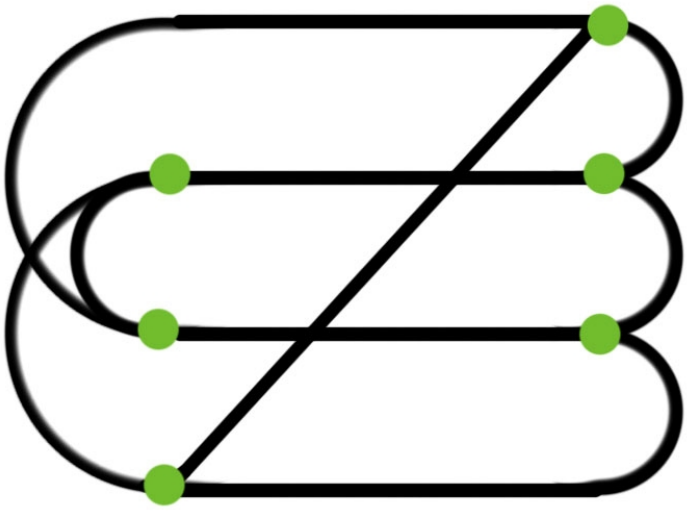


(left) very first simple approach for testing purposes
(right) first serious level design, later declared as "too complex"

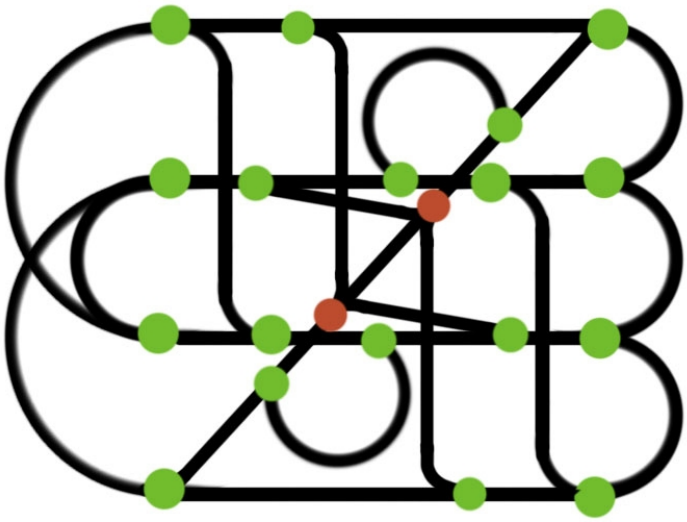
The path-bound movement of the tanks is both a blessing and a curse. I don't have to worry about glitches in the level-borders, or that players might reach locations they aren't meant to, as they can only follow the rail-system. This has the disadvantage that some players – especially absolute beginners – might find themselves in situations they perceive as unfair. Situations like heading toward a collision or mine and not having any opportunity to evade because they missed their last possible switch to leave the path. However this is what ArenaRails is all about, so I faced the challenge of creating a rail-system that delivers enough opportunities for evasion, without turning it into a pile of spaghetti or making the rail mechanic useless. Also there needed to be some order in the overall form to help with general orientation: symmetry and asymmetry, concentric circles, straight and curved paths, parallels and crossings or an overall "topic" to assist the player's understanding. The following describes the development of the currently implemented multiplayer level.



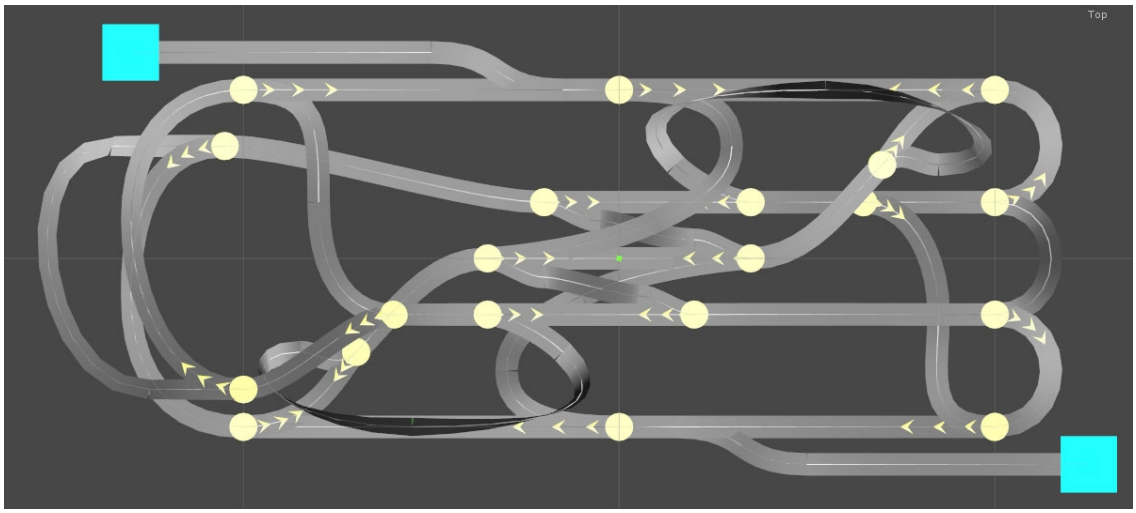
I started with two overlapping Nascar-typical oval routes. At this point, the two loops physically intersect, but there is no connection between the nodes, so it is impossible to change between the loops. These four straight paths running parallel from one end to the other is the main "topic" of the level. At this stage the track possesses symmetry on all axes and point symmetry.



I added another straight route running diagonally from one end to the other, creating the first connection between the two ovals. This broke the axis symmetry, but as the point symmetry remains, the ends of the straight paths could still be mixed up. I proceeded to change the curves on one side to create different connections between the four straight paths. This may have destroyed the two first routes, but the overall topic remains while adding some variety and uniqueness to each corner of the rail-system.

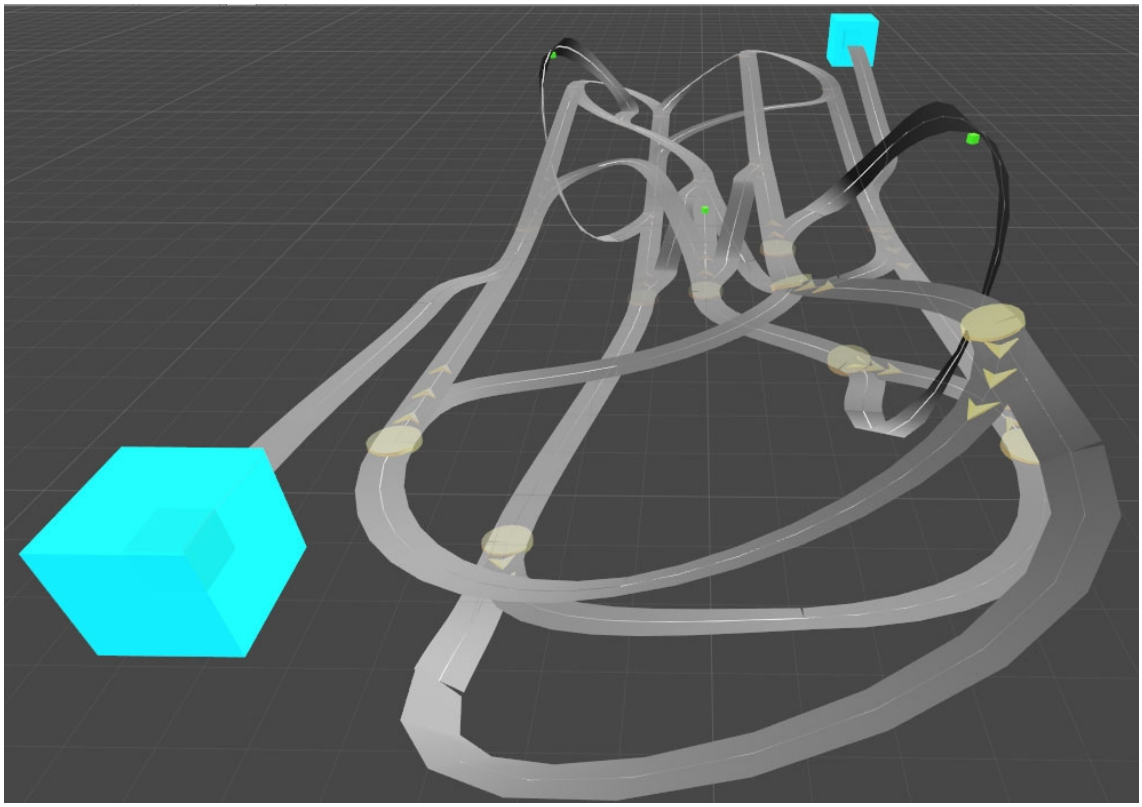


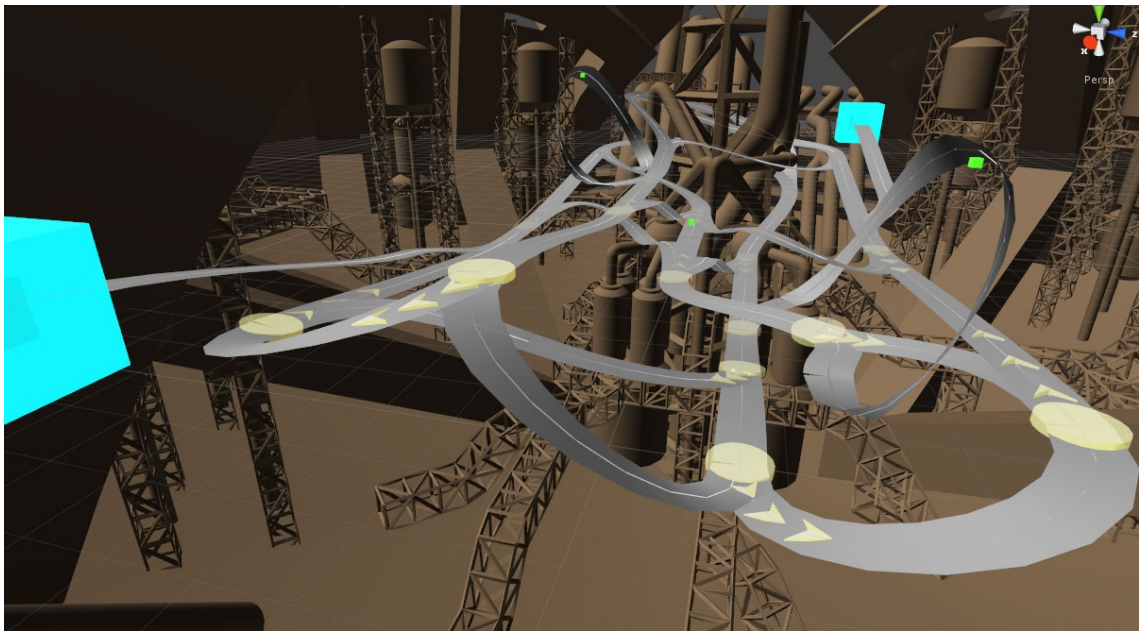
I then added more opportunities to switch routes in different directions and stuck to a point symmetric approach. The concept above has only two switches that have three out-connections. A low amount of three-way switches should contribute to a player's overview and planning ability, as it's not instantly clear in which order a three-way switch will swap between its connections. At this point I already experimented with different height levels in my head for each of the four straights and I was eager to give this schematic a try.



Although it might look different at a first glance, this rail-system still follows the connection schematic above. I only added two spawn positions (blue squares) that are connected to the outer straight paths. They are connected via two-way nodes which are not switches. This ensures that players can enter the rail-system, but never leave it. The deformations of the left end occur from the integration into the environment.

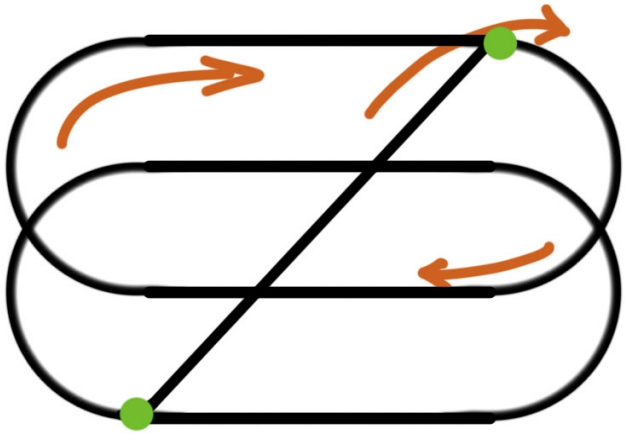
All other changes occurred as a result of avoiding collisions of the track itself due to height differences. When a switch offers to leave a straight route, this path has to be similar to the off-ramp of a highway, other designs result in tanks clipping through the track mesh.



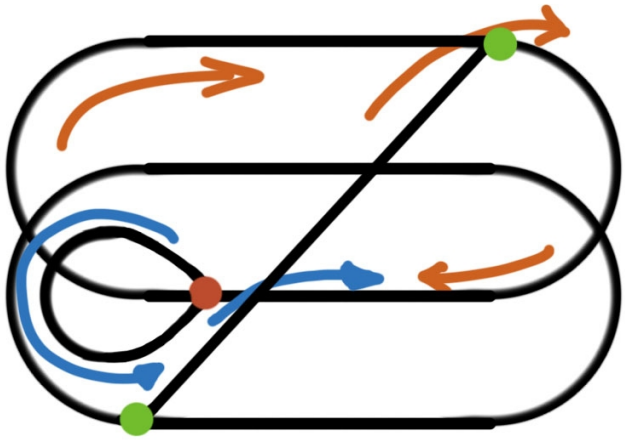


b) The Routing Problem

Besides a few entry points, the rail-system is a closed system, within which every possible path results in a re-occurring loop as long as the switches within it remain unchanged. While making my first levels I noticed that when there are too few switches in a loop, the likelihood of the loop becoming a “trap” gets higher: a loop that you can enter, but due to the direction of the switches, doesn’t let you leave.



This meant I had to keep an eye on covering both driving directions of a loop with always at least one switch providing an exit opportunity. I tried to ensure that there are at least two exits for each direction per loop to reduce some of the earlier mentioned possible unfairness, however overdoing the exits could lead back to the spaghetti problem.Another solution I found was to give the player the opportunity to reverse the driving direction of the loop. I figured this would also provide a good placement opportunity for items like flag-stands, or the objects for "Core Combat", to create the difficulty of getting in and out again.



5.2. Network

Creating a multiplayer game nowadays requires the implementation of the ability to play over a network, or even the internet. Network programming requires an insane amount of patience and/or experience and is much more difficult to solve with brute force (trial and error). I was glad I found the Photon Unity Network (PUN) system that is available for free, which handled the basics of communication, game creation and per-player-variables. The general problems of network programming remained, of course.

What, How and When to Synchronize

Like in any project, the most problems will begin and end with communication.
Well implemented network code communicates as much as necessary to ensure proper synchronous behavior of all participating players, but also as little as possible to save bandwidth and reduce lag. The objects of ArenaRails which required synchronization were:

- **Tanks**
- **Projectiles**
- **Switches**
- **Items**
- **Scores**

In general there are two methods of synchronization: authoritative and non-authoritative. Authoritative synchronization requires one master server and several slave clients. Slaves can send their inputs, the master processes it, and then sends the same results to all slave clients. This ensures synchronous behavior, but the slaves experience a lag between their input and the actual reaction of the game.

Non-authoritative synchronization allows each player to control his own objects directly and then send their processes results to the other players. This ensures that each players feeling of his own objects are right, but completely synchronization is not ensured.
I tried to use both concepts, depending on the situation, but generally I preferred the non-authoritative concept. I recognized that a multiplayer game doesn't have to be perfectly synchronous, just needs to seem synchronous.

At first I had to ensure a proper synchronization for the tanks movement.
Having a mismatch in player position reduces the immersion strongly as players quickly notice that something is just not quite right. PUN offers the possibility to stream an object's position directly through the network. As the universities WLAN and internet connection induced a lag of at least 150 milliseconds (always, at least for the Photon servers) I had to compensate. So instead of streaming the result of the original tanks path calculation, I found out that streaming the variables for this calculation might be a better idea: The "original tank" would be the tank that the player controls, and copies of the variables are used to reproduce the tank in the games of the other players.
So I streamed two float-type variables; a position parameter on the current curve (a value between 0 and 1 from start to end) and the current speed, instead of the position (a Vector3-type variable consisting of 3 float-types). Each player's game could now calculate the positions of the other tanks itself, and to further compensate the lag,
I implemented the current time difference (the 150 ms) into the equation, giving pretty satisfying results.
To ensure that the synchronization doesn't drift apart over time, the original tank sends a message through the network to its copies as soon as it reaches a node.
This message would reset the position parameter to 0 and transmit the new curves start- and end-nodes. This method also applies for the Track-Missile.

The rotation of the tanks turret and cannon don't have a lag-countering calculation and use the simple streaming of Photon, as it is an unnecessary for this detail to be exact on every player's game. The fired projectiles also use the streaming for position and rotation, but are only able to react on the controlling player's side.
So if a projectile of player A hits the tank of player B, player A's client tells Player B's tank to receive damage, and also that it was fired by player A. This information gets stored by the Player B's tank as "the last player who hit me". The tank will then update its other copies and original with the new health value.
Although the projectile's owner decides when a tank gets damage, the tank's owner decides if the tank must explode due to a health value below zero. This ensures that the firing player (A) will perceive the correct response for his actions and the damage receiving player (B) perceives the correct behavior of his health bar.
As the tank's movement is rather accurate, and the projectiles move faster than the tanks and induce splash damage, it gets hard to feel a difference between the possibly varying impact positions of the projectiles.

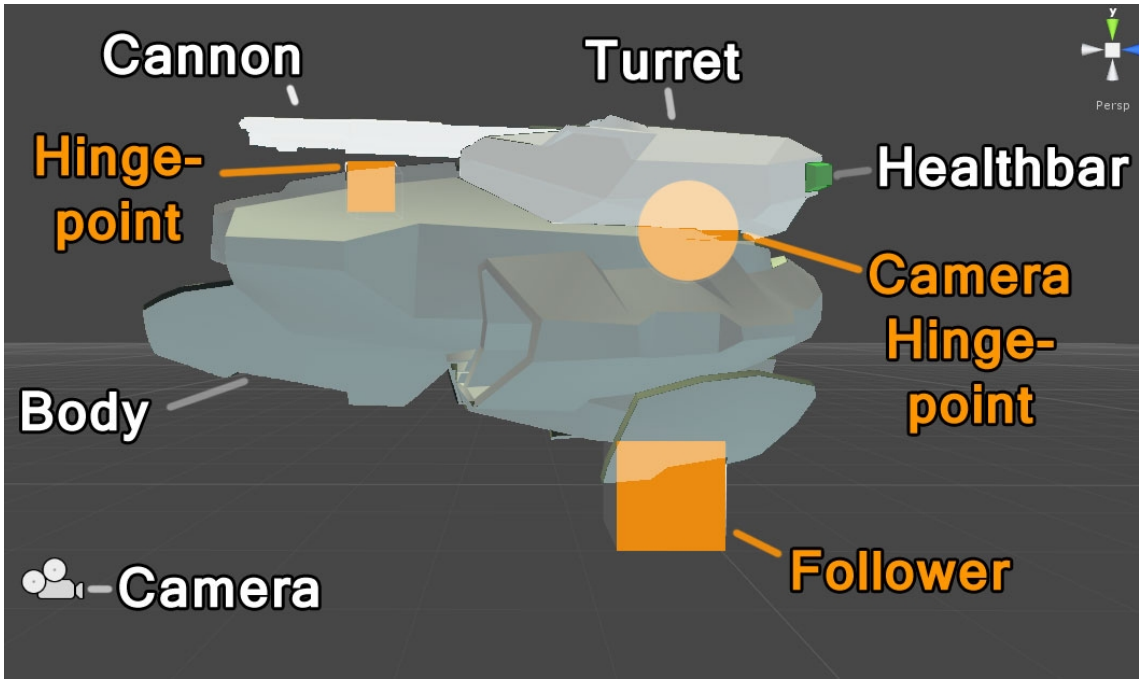
As switches have no controlling player they may be a weakness in my network programming. They have the same "rights" on each client. When a switch gets hit by a projectile on one client, it will send a message to its copies (in the other clients,) that will start the swap and lock the switch. Based on the worst case scenario that two players should hit a switch at the exact same point in time, the locking won't take place in time, and switching process may occur twice (as the switch in each client requests a change in the client of the other).
This has not happened yet, or it hasn't been noticed, as the result of this might just be a change of out-nodes in the opposite direction (on three-way switches) or no change at all (on two-way switches). Having only become aware of this possible bug while writing, I don't want to attempt to fix it, as attempting the fix based on an assumption may break other parts of the game.

The only part where I used the authoritative method is the collisions. As both the original tank and the tank copies must have the ability to experience collisions, there was no other option. Without this, no collisions could occur at all. If I had implemented the non-authoritative method for collisions, it is very likely that the collision-event would occur twice – similar to the worst case scenario for the switches - but as a collision always involves two objects being at the same position both in space and time (which is rare for the projectiles), I had to use the other method in this situation.
As a result, collisions are only calculated by the player who created the game, sending the necessary information to all copies and the other original tanks.

Finally, there's each player's score. The scores are communicated via a service of PUN that adds custom properties to each player. These properties are used to store each player's selected tank, his color/symbol, and the number of kills and deaths in each game. The kills and deaths are updated as soon as a tank gets destroyed.
As already mentioned, only a tank's owner processes the death, and as the tank has stored the information of its last attacker, the last attacker will receive a kill point from the dying tank.

5.3. Class Overview on a Tank

This chapter shall deliver a small overview of all the components each tank is made of in the current prototype.



Hierarchy:

- **Follower****
- **Camera Hinge-point***
- **Camera****
- **Body**
- **Hinge-point***
- **Turret**
- **Cannon**
- **Health bar**

* (not visible inside the game, generated by script)

** (not visible inside the game)

Scripts:

Follower Components

– Node Follower

Manages the movement via nodes and switches.

– Movement Stream

Streams the position parameter and speed of the Node Follower to its copies.

– Tank Speed Controller

Reacts to players keyboard input and communicates it to Node Follower

– Follower Auto Setup

Finds nearest Node and communicates them to Node Follower.

– Prediction Tracer Spawner

Creates the animated prediction line.

– Tank Stats

Allows fast access to balancing relevant variables (damage, speed, health) and overwrites them at game-start.

Camera Hinge-point

– Network Stream

General script to stream position and/or rotation to its copies.

Camera

– Camera Look Behavior

Reacts to player's mouse input and controls Camera rotation and position.

– Target Point Behavior

Calculates projectile's trajectory depending on its speed and creates a position in space, representing 100m distance height of the projectile.

– Crosshair Behavior

Displays crosshair in interface and manages reaction to hit feedback.

– Network Clean Up

General class that will destroys selected scripts if the tank is a copy

– Cam Disable

Script that disables the Camera component if the tank is a copy.

– FOV Control

Reacts to the change of speed and adapts field of view.

Body

– Health Behavior

Manages the tank's health, destruction, and the health bar.

– Collision Behavior

Manages all collision related calculations (assuming this tank is the master)

– Health Resource Manager

Reacts to input from Tank Speed Controller and incoming damage to manage health drain and regeneration.

– Pick Up Handler

Manages the ability to pick up and control items. Creates Hinge-point visualization. Places items at Hinge-point upon pick up.

Turret

– Turn To Target Behavior

Gets position from Target Point Behavior to rotate to this position on one or two axes.

– Network Stream

– Network Clean Up

Cannon

– Turn To Target Behavior

– Cannon Behavior

Stores the projectile and damage value and reacts to players input to fire projectile.

– Network Stream

– Network Clean Up

6. Further unimplemented ideas

6.1. Possible Business Model

A currently popular business model and one that ensures a large player base is Free-to-Play. F2P works only if players have an overarching measure of progress inside the game – progression that is slow, and strenuous for those who don't pay, and quicker and easier for those who do pay. This progress is often represented by gathering experience points which unlock further gameplay elements like items or special skills and classes. Also the possibility of visually customizing the own game characters/vehicles is an often used practice.

So for the current state of ArenaRails, players would have the opportunity to select between more tanks, get beautiful but non-gameplay influencing attachments for their tanks or symbols, or add more possible items to their pool. This might not be enough, to keep the players p(l)aying, so I may have to implement the previously mentioned experience points and a skill/talent/additional-gameplay-relevant system, that would benefit from it. This would of course create more classes of tanks and requires a high level of balancing and testing, but it should be worth it.

6.2. Characters

Many racing games don't leave the vehicles be the only heroes of the game. Characters that are actually driving those vehicles can help the players to develop a stronger connection to their avatar. They could also provide a platform for subtle in-game stories, as the characters could talk or yell at each other, when they meet in-game. Different characters would have different reactions to each other character. This is a mechanic that's already implemented in the new DOTA 2.
They would also provide another layer for the previously mentioned business model, as it provides an additional option for customization.

Another benefit would be the possibility to not bind characters to certain tanks, but to let the players combine their favorite tank with their favorite character. Each character could have certain attributes that might work well or badly with the selected tank.

6.3. Sound Concept

Being left until the deadline was almost upon me, the prototype doesn't have any sounds. However I had imagined the game's music being connected to the current player's situation, similar to Split/Second. The music should not change entirely when the situation changes – like when the player gets killed, scores a kill, drives faster or slower, or is about to collide with an opponent - but it should adapt in speed, and alter the number, volume, and selection of instruments to fit the situation.

When dead, it would be barely hearable - nothing more than some light drums and bells, keeping up the general rhythm. Driving slowly into the arena, the soundtrack would welcome the player with a short burst of fanfares, adding deeper and stronger drums as the player's speed increases. Switching a node could cue a fitting jingle - just as picking up items or scoring a kill. Highly dangerous situations, such as an imminent collision or a heavy hit from an item, would be introduced with a loud drumbeat, followed by a dampened but more intense rhythm, building into deeper drums with a simple melody to support the futuristic scenario. Objects that are close to the track will create a "whoosh" when the players drive by fast enough, while the heavy engines of the tanks rumble and hiss with a higher or lower pitch depending on their speed.

This would all be supported by typical sounds for explosions, weapon fire – be it projectile, missile or laser – and the occasional cheering audience. An announcer could announce a player entering the arena or killing another player, as it could simply refer to the player's symbol or color.

7. Involved Software

Unity3D v3.5.

Autodesk 3D Studio Max 2012

Adobe Photoshop CS5

Unity Plugins:

Photon Unity Network (http://www.exitgames.com/Photon/Unity)

.OBJ-Exporter (http://wiki.unity3d.com/index.php?title=ObjExporter)

Strumpy Shader Editor (http://u3d.as/content/strumpy-games/strumpy-shader-editor/1C4)

8. Conclusion

I enjoyed receiving good support from my supervisors, who showed interest in the project, even when I wasn't asking for help.
I learned a lot during this project, especially what it means to really work intensely and alone on one piece of work with a large scope in a short amount of time. Post-Its can provide a good overview of the remaining tasks, and it can become a rewarding feeling to rip a task to pieces, once it is done. I made this a ritual. Christmas is a bad ritual, especially when the deadline doesn't care.

However, I'm also aware that I didn't reach my original and official project target, mainly with respect to the visual development, but proving a visual representation of the gameplay relevant objects often is more important than a good looking asset, and if it requires time, take it. What are also necessary are good friends who don't care about your feelings when they are criticizing your hard work. If even the people, who should know and understand you the most, do not understand what's going on in your game, you have to suck up the fact that their criticism is appropriate. You'll lose an objective point of view, when you look at your "child" alone.

I'm glad and sad at the same time that I'm (almost) finished now. This idea, which dwelled so long inside my head, has finally been proved as having the potential to become a proper game. This is what I originally intended for myself, so this target is accomplished.
I want to recreate it with a cleaner code, finished visual concept and polished game mechanics. Maybe not today, but someday.