General Technological Overview

The main components of the setup are as follows:

The Twitch-Chat-Scraper Script: If a Chatter types a command into the chat window this script receives it via the Twitch-API and parses it into Keyboard-Commands.

The main Unity Application: (Has two purposes) Receives those Keyboard-Commands and executes them. Accordingly, it adds votes to the ballot, switches cameras and relays commands to the Gravitrax Connect Client Script via other specific keyboard command.  
Aside from that it arranges and displays all camera feeds - with visualizations for all of the voting processes and interaction options.

The Gravitrax Connect Client Script:  
Receives the Keyboard-Commands output by the Unity Application and transfers those via Bluetooth to the “Gravitrax Power Connect Brick”.

Gravitrax Power Connect Brick:  
This in Turn relays this information to the respective Gravitrax Power Elements (e.g. Switches, Starters or Levers) as a Signal on their “red”, “green” or “blue” channel.

Cameras:  
A mix of mirrorless DSLMs and Webcams capture the Marble track in its entirety with emphasis on important hotspots, such as controllable Switches. Their video feed gets ingested into the PC running the Unity Scene Manager Application in real time through either USB or HDMI links.

OBS: Captures the entire Unity graphical Frame and streams it to Twitch for everyone to see.

OG:

1. Abstract
2. Introduction
   1. State of the Art
   2. Motivation
   3. Goal
3. Finding Ideas and creative processes [taking a step back here]
   1. Extraneous Circumstances and initial request
   2. Creative Process Paradigm: 4D-Phases
      1. Discover: (Get Ideas via looking at Creative Tools/Processes & Models<such as Player Types>)
   3. Game Idea A: Melody Playground
   4. Game Idea B: Logistics Puzzler
      1. Player Types: Found via Identifying Achievers as not targeted audience (sandbox-y, therefore free, but non-linear)
   5. Streaming Idea: Interactive Livestream
      1. (Basic Concept)
      2. Inspiration and History of interactive live Video Feeds
      3. Multicam
      4. Moderation?
         1. A complexity I excluded, for focusing on all options for integrating technology – so you can sample them all…
      5. Streamers
         1. Interacting with a built track
         2. Building a track with feedback
4. Implementation (/Transfer?)
5. OR Technical Implementation
   1. Physical Set
   2. Core Software Framework (Unity + 2 PyScripts)
   3. Unity
      1. Video Input
      2. Camera Array (Video-Mixer?)
      3. More interactivity: Sfx
   4. Light-Up Switches
   5. Ausblick: Virtual Particle Systems in 3D Space
   6. Ausblick2: Clickmaps
   7. Ausblick (more vague): Build of an Independent webapp
6. AND Evaluation
   1. Ausblick hier?

Klemens ver.

1. Abstract
   1. Tags and Buzzwords! (unity3d, 3dPrint, CAD-Software)
   2. Ergebnisse (coole latenz…
2. Introduction
   1. Motivation
   2. Goal (ausgearbeitet)
3. Finding Ideas and creative processes *[taking a step back here]*
   1. {Extraneous Circumstances and }initial request
   2. Creative Process Paradigm: 4D-Phases
      1. *Discover: (Get Ideas via looking at Creative Tools/Processes & Models<such as Player Types>)*
   3. Game Idea A: Melody Playground
   4. Game Idea B: Logistics Puzzler
      1. *Player Types: Found via Identifying Achievers as not targeted audience (sandbox-y, therefore free, but non-linear)*
   5. Streaming Idea: Interactive Livestream
      1. *(Basic Concept)*
      2. Inspiration and History of interactive live Video Feeds
      3. Multicam
      4. Moderation?
         1. *A complexity I excluded, for focusing on all options for integrating technology – so you can sample them all…*
      5. Streamers
         1. Interacting with a built track
         2. Building a track with feedback
4. State of the Art
   1. Grundlagen? (Technische basics, api, twitch how does it communicate)
5. Hardware Implementation
   1. Camera Setup
   2. Lighting and Set
   3. Marble Tracks
   4. *Expansion*: Configuring a custom Daughterboard for a tightly integrated Set
6. OR Technical Implementation
   1. Physical Set
   2. Core Software Framework (Unity + 2 PyScripts)
   3. Unity
      1. Video Input
      2. Camera Array (Video-Mixer?)
      3. More interactivity: Sfx
   4. Light-Up Switches
   5. *Ausblick*: Virtual Particle Systems in 3D Space
   6. *Ausblick2*: Clickmaps
   7. *Ausblick3*: Development of an independent Webapp

Manu ver.

1. *Abstract*
   1. *Tags and Buzzwords! (unity3d, 3dPrint, CAD-Software)*
   2. *Ergebnisse (coole latenz…*
2. Introduction
   1. Motivation *(kurz halten)*
      1. *WICHTIGSTER:Streams immer cooler, interaktive noch cooler! -> immer mehr Firmen (z.B. Product launches)*
      2. *(Digitale testversionen easy) Physische produkte testen und ausprobieren*
      3. *Digitalspielzeuge habens schwer, ich hole die leute ab wo sie eh schon sind (kein produktkauf nötig, soziales zusammenspielen)*
   2. Goals and /Abgrenzung/ (ausgearbeitet)
   3. Structure *(Inhaltsverzeichnis in prosa)*
3. Conception: Finding Ideas and creative processes *[taking a step back here]*
   1. {Extraneous Circumstances and }initial request
   2. Creative Process Paradigm: 4D-Phases
      1. *Discover: (Get Ideas via looking at Creative Tools/Processes & Models<such as Player Types>)*
   3. Game Idea A: Melody Playground
   4. Game Idea B: Logistics Puzzler
      1. *Player Types: Found via Identifying Achievers as not targeted audience (sandbox-y, therefore free, but non-linear)*
   5. Streaming Idea: Interactive Livestream
      1. *(Basic Concept)*
      2. UI and 3 Wireframes
      3. Inspiration and History of interactive live Video Feeds
      4. Multicam
      5. Moderation?
         1. *A complexity I excluded, for focusing on all options for integrating technology – so you can sample them all…*
      6. Streamers
         1. Interacting with a built track
         2. Building a track with feedback
      7. //Gibt keinen Programmablauf, nonlinear, come in and have fun…oder eben Phasen
      8. *ANforderungsanalyse? (Ziele optimierung visual quality vs latency/performance) (vielleicht später)*
4. State of the Art
   1. Grundlagen? (Technische basics, api, twitch how does it communicate)
   2. Conflicts: Telegames (I have: no fail condition, lower laterncy, social chaos is feature)
   3. Potential (Motivation an konkreten beispielen) (konkrete beispiele vlt hier und mein Wunsch/pipe dream oben in konzeption)
5. Hardware Implementation
   1. Camera Setup
   2. Lighting and Set
   3. Marble Tracks
   4. *Expansion*: Configuring a custom Daughterboard for a tightly integrated Set
6. OR Technical Implementation
   1. Physical Set
   2. Core Software Framework (Unity + 2 PyScripts)
   3. Unity
      1. Video Input
      2. Camera Array (Video-Mixer?)
      3. More interactivity: Sfx
   4. Light-Up Switches
7. Evaluation
8. Fazit & Ausblick
   1. *Ausblick*: Virtual Particle Systems in 3D Space
   2. *Ausblick2*: Clickmaps
   3. *Ausblick3*: Development of an independent Webapp

Manu – redacted ver.

1. *Abstract*
   1. *Tags and Buzzwords! (unity3d, 3dPrint, CAD-Software)*
   2. *Ergebnisse (coole latenz…*
2. Introduction
   1. Motivation *(kurz halten)*
      1. *WICHTIGSTER:Streams immer cooler, interaktive noch cooler! -> immer mehr Firmen (z.B. Product launches)*
      2. *(Digitale testversionen easy) Physische produkte testen und ausprobieren*
      3. *Digitalspielzeuge habens schwer, ich hole die leute ab wo sie eh schon sind (kein produktkauf nötig, soziales zusammenspielen)*
   2. Goals and /Abgrenzung/ (ausgearbeitet)
   3. [Structure *(Inhaltsverzeichnis in prosa)]*
3. Conception: Finding Ideas and creative processes *[taking a step back here]*
   1. {Extraneous Circumstances and} initial request
   2. Creative Process Paradigm: 4D-Phases
      1. *Discover: (Get Ideas via looking at Creative Tools/Processes & Models<such as Player Types>)*
   3. Game Idea A: Melody Playground
   4. Game Idea B: Logistics Puzzler
      1. *Player Types: Found via Identifying Achievers as not targeted audience (sandbox-y, therefore free, but non-linear)*
   5. Streaming Idea: Interactive Livestream
      1. *(Basic Concept)*
      2. UI and 3 Wireframes
      3. Inspiration and History of interactive live Video Feeds
      4. Multicam
      5. Moderation?
         1. *A complexity I excluded, for focusing on all options for integrating technology – so you can sample them all…*
      6. Streamers
         1. ModerationInteracting with a built track
         2. Building a track with feedback
      7. //Gibt keinen Programmablauf, nonlinear, come in and have fun…oder eben Phasen
      8. *ANforderungsanalyse? (Ziele optimierung visual quality vs latency/performance) (vielleicht später)*
4. State of the Art
   1. Grundlagen? (Technische basics, api, twitch how does it communicate)
   2. //Conflicts: Telegames (I have: no fail condition, lower laterncy, social chaos is feature)
   3. Potential (Motivation an konkreten beispielen) (konkrete beispiele vlt hier und mein Wunsch/pipe dream oben in konzeption)
5. Hardware Implementation
   1. Camera Setup
   2. Lighting and Set
   3. Marble Tracks
   4. *Expansion*: Configuring a custom Daughterboard for a tightly integrated Set
6. Software Implementation and Custom Solutions
   1. //Physical Set
   2. Core Software Framework (Unity + 2 PyScripts)
   3. Unity
      1. Video Input
      2. Camera Array (Video-Mixer?)
      3. Supporting features: Sfx, Animations & more
   4. Light-Up Switches
7. Evaluation
8. Conclusion & Outlook
   1. *Ausblick*: Virtual Particle Systems in 3D Space
   2. *Ausblick2*: Clickmaps
   3. *Ausblick3*: Development of an independent Webapp

Motivation:

Livestreams have been getting more and more popular over the last ten years.

The internet has made way for probably the most abrupt push towards democratization of media creation. The popularization of the internet has made it easy for people to spread ideas and opinions, truly anyone could participate. While this freedom does not come without risks, it has enabled even such complex media as videos to become user-generated easily nowadays. While clips shared online are already quite powerful, traditional offline media has kept monopoly over one particular aspect – the real time nature of a live-TV broadcast has not been reached by online video.

So simply by being created in real-time, live streams can be much more engaging and interactive as a form user generated content. Streamers can directly respond to comments the second posted to build a deeper connection to their fan base, and react to complaints the moment they are raised. Therefore streamers can not only change their style of commentary on the spot but even go as far as to adapt the content itself to the general mood of viewers.

This novel kind of interactivity is interesting to me. I would like to build upon this interactivity and experiment how to further expand it in hopes of giving streamers a bigger toolkit to rely on in the future.  
On the other hand watching lives treams can feel a little arduous/less curated or engaging at times, since it captures downtime that would get cut out from both long and short form video content (such as YouTube videos, gameplay highlight reels, Shorts or TikToks etc.). By giving viewers/chatters something to keep busy and play around with I hope to bridge said downtime more effectively.  
Furthermore do I wish to even improve the highs during a stream by means of giving viewers a way to influence the content directly.

So when Ravensburger, the toy company located in Swabia, Germany, contacted me about working together a creative project around their smart, Bluetooth connected marble track system, I soon proposed a project, involving live streams. The ability to integrate the Gravitrax marble tracks into any software is a truly unique opportunity, so passing the control on to a live streaming audience came pretty came to me pretty fast.  
# Since Livestreams can feel a

On one hand they are the logical next step for user generated content without needing to be produced by a team of professionals

On the other hand they replace conventions – or at least their tedious convention part. (Example E3)

Livestreams are filling an interesting hole.

Goal:

This project is not about producing the prettiest/best *(no shipping-ready)* possible product but exploring various different Options, strategies and tools – and evaluating those. We are researching possibilities for improving interactivity and engagement in the respective fields we’re going to touch upon or dive into. With the goal of giving future projects a wider variety of vantage points and what to expect, giving inspiration and a different look on lives treams.

In terms of the 4D-creative process (further discussed in chapter XY) I plan to leave off after the completion of stage 3, Develop. While I will give insights into results and learnings from user testing and development the prototype will not be reduced down to a deliverable complete product but rather released as a tool inviting interested creatives to experiment with its features.

3.0/3.1

Before arriving at a streaming concept there were other prototypes defined and pitched to Ravensburger. The following chapter outlines the creative process leading up to the final pitch.  
To that end I took a look at different creative tools and processes, with the goal of finding ideas that could build upon the concepts and mechanics, that are at work in Gravitrax and give more value to an already working system.

We will look at the brainstorming phase through the lens of one of the creative processes used, the 4D P. This common process dictates, that the development of a product consists of 4 phases, all beginning with a “D”.  
In the 1st phase, “Discover”, Designers are encouraged to think as creatively as they can. Ideas get collected no matter their ease of implementation or relevance on the market, to encourage producing more out-of-the-box concepts.  
This is what I’ll be looking at in this chapter.

Ravensburger Quest here.

This request by Ravensburger gives this project another dimension, requiring me to conceptualize the project such that it adds considerable value to the Gravitrax-ecosytem, in one way or another.  
While this requirement did step more and more into the background as the project progressed, the concept was planned in a way to both improve upon the common live streaming experience and simultaneously give Gravitrax a new type of interactive experience.

So the second dimension of “Motivation” consists…  
2. It’s always difficult to give potential buyers a taste of a physical toy. While it is easy to give out Demo versions of

Most of the digital entertainment industry likes to give out samples/testers of their products. Movies publish trailers, podcasts release entire episodes or shows from their catalogue on demand, and even video-games offer demo versions of their final product before launch for download – usually free of charge. All of the examples give consumers a way to experience in much the same way as the full product.  
The closest things to testers Board games and toys offer are promotional material like photographs, illustrations, renders and videos. All of those play in a different medium than the full product, crucially missing any way of interaction.  
In some rare cases demo products are displayed in some of the bigger branches of the bigger national retailers, but even so shelf space is limited, and space for demo installations are at a premium. As a result, far from every interested buyer has the chance to test out a toy for themselves, regardless of who they are shopping for.

3. Motivation – Digital Boardgames have had a hard time arriving on the market. Ravensburger themselves have launched two of these in the past, achieving mixed success.  
After asking boardgame-enthusiasts, they came to a consensus: Enthusiasts themselves say they don’t feel appealed by digital boardgames, since they identify with their hobby through its analogue and haptic nature, while fearing the games becoming unplayable in the future via losing compatibility, like companion apps “explainiation” with future OSes, or the apps vanishing from the internet and app stores entirely. This situation clashes with most board game enthusiasts identifying themselves as collectors.  
At the same time digital games seem to appear more complex and are thus less likely to be recommended to, or picked up by, beginners.  
Therefore a secondary goal/motivation was to look out for some ways to incorporate technology into the marble tracks, without interfering with the underlying game and gameplay loop.

To gain an understanding for the toy and brainstorm for expansion possibilities I used a model popular in Ludology to identify the main motivators for interacting with Gravitrax. This model does not only explain what type of players engage with Gravitrax but could show, which extensions can Gravitrax benefit from. Even though it is originally derived from video games, it can give some insight into boardgames and toys as well.

I have conceptualized three options of realizing this concept with varying levels of complexity. They differentiate each other by the type of challenge they pose to the players, resulting in different games and make the marble tracks and details in UI to look vastly different.

It is the most reduced version of this challenge. One to a handful of players try out different constellations of track switches to make the marble take different paths. The input is handled directly and only few viewers are on stream, so there is as little input delay as possible, meaning the marble tracks can be built in a way to let the marble zoom past objects at higher speeds.  
The marble does not even have to have a specific goal and in its simplest form with only one track switch it is the easiest and best to develop tools for and verify their correct function and influence on players’ experience.

The second option is much more complex. It requires multiple players to trigger commands simultaneously. Players need to work together to find out which channels to activate at the same time, to synchronize marbles. Once again a direct input system allows players to synchronize and coordinate their actions. This track can even feature multiple marble runs, running independently from one another. This track is ideally quite compact, but dense with an intricate maze of tracks, to make this a complex problem to solve, but not more complicated.   
This concept has the unique advantage, that it promotes a lot of social interaction within the community, which is a very unique selling point of live streams.

If though, there is a big amount of players that need to be entertained, option three is a better choice. Its underlying change is that any action a user requests does not get executed immediately via a direct input system. Instead, it gets added to a vote, and after a certain amount of time passes or a critical majority is reached, the most requested command gets executed. This requires some downtime between decisions for these votes to take place. A simple way of achieving this is to arrange the marble tracks onto multiple terraces, connected by a few marble lifts. This gives some time for votes to take place and each level can be shown with its own camera. To reach the top and activate a reward like an animation or “light show”, users need to cooperate and decide on which channels to enable and disable for each level.

But logically for this proof of concept and tool development followed the path of the first concept. This is what will be targeted in this bachelors thesis, but wit the prospect of switching to a different path in the future, once tools are developed and kinks are ironed out.

Hardware: Light up brick (Rv calls them “tiles”)

...so I built a frame resembling a Gravitrax’ brick, able to house my LEDs.  
Instead of cutting out perfectly spaced holes for each LED and lining them up one by one, I instead got ahold of some transparent Filament. It’s some PLA “EXPLAIN” without any added pigments, resulting in it being being almost perfectly see through.   
(Addendum: It has a very high light transmission coefficient of up to 90%, comparable to glass. Furthermore I’ve found out this figure is really consistent for PLA across the entire spectrum of visible light, meaning it barely distorts colors shining through it. A surprisingly good candidate for my application. - https://www.researchgate.net/figure/UV-and-visible-light-transmission-of-PLA-and-PLA-BHT-films\_fig2\_232382940)  
Ideally I’d want to use some transparent PET-G, which is regarded as one of the best transparent 3D-Printing filaments. With lower virtuosity at its higher printing temperatures air gaps are minimized and print flow rate is increased, resulting in a more consistent lighting, and prettier look.

This lets me use the frame not only as a mount for the LEDs and a face plate to hide the unattractive wiring – but more importantly as a diffusor, by mouting the LEDs directly behind the solid wall of the part.

This feature is very welcome since I used a sub-optimal kind of led for this prototype on purpose. Ideally I wanted to use so-called “Filament Style LEDs”, a relatively newer kind of LED, at least in the hobbyist space. These LEDs produce a thin, homogenous line of light, much like the more high-end drop-in replacements for light bulbs, trying to emulate the glow of an oldschool tungsten wire. While the produced lighting effect would have been very pleasant and sleek, I could only find them in a monochrome-colored design, without individually adressible segments, without a driver, at a higher price. This was for me personally too little flexibility for this iteration, so I opted for the highest-density strips of WS2812 RGB-LED-Modules. These have been tried and tested in prototyping, due to their easy 5V operation voltage, individual per-led addressability and relatively low cost. I could find them in thin, 166 Modules per meter strips, a resolution that should prove to be plenty for this application.

After some test prints I’ve also experimented with larger wall thickness and even a hollow wall, to abuse the properties of refration, the principle visually enlarging text placed behind a glass of water. PLA has an even higher refractive Index (greater 1.45) than water (around 1.33), which makes this effect not negligible. So by hollowing out every wall we have in total four changes in medium, altering the resulting visual noticably.

Top to bottom: 1.6 mm wall strength, solid; 2.4mm wall thickness, solid; 2.4 mm thick wall, hollow, coming out to effectively 2 times 0.4mm thick lines per wall (with 1.6mm of air in between).

The light appears much brighter and clearer head-on on the part with the hollowed out walls. But from an isometric perspective above the light looks suddenly much fainter than its counterparts. When tilting the part along the x-Axis you can even notice a break-point when the refraction changes the appearance of the light abruptly, roughly around 30° from horizontal. Which is sadly no good for a multi cam recording setup, so I went with one of the solid walls.  
As an aside: The refractive effect is virtually destroyed on the solid prints, since imperfections, air gaps between every line and layer, and impurities in every single line disrupt any discernible pattern, resulting in a milky impression resembling frosted glass. Light now simply gets diffused and softened as it passes through.

Light up-Tile:PCB, Connection and Control (Electronics)

To drive and control the Light-up-Tile modules, I have designed a circuit board. It connects the modules to a micro controller board, which in turn receives signals from the streaming pc, and powers them from a separate power source.

For the micro controller I use an “Arduino Nano”. It features plenty IO-ports, has a small footprint and is quite affordable. Additionaly this version features a USB-C port, that is easy to connect to the PC and can be used to send & receive data from this PC via a serial connection.

Each LED strip’s data-in port is simply wired to a digital pin on the microcontroller respectively, each with a 330 Ohms resistor for current limiting.   
The WS2812 Led-Chipset is daisy-chainable, meaning only the first LED-Module in a strip must be connected up to a micro controller, all consecutive modules on the LED strip get the signal passed on through the previous LED’s data-out pin. There is another reason for using it here. Compared to its preedecessor, the WS2811 modules, the WS2812 powered strips only control one LED with every chip, as opposed to three per chip on WS2811 powered ones. This quirk results in three consecutive LEDs lighting up in the same colour on the WS2811. So opting for its successor gives me more horizontal resolution inside a tile to work with, which in turn makes conveying information with animations clearer. Even though we have a more severe voltage drop across the chips on the strip compared to the WS2811s’ 12 volt power.

Since they are powered from an external, variable voltage power supply, the LEDs can be replaced with an LED-chipset supporting higher power or switching frequency, as there is a risk of pwm-switching showing up as flickering or banding on video. Luckily this did not turn out to be a problem here, so the WS2812 strips stayed in use.

Power for the LEDs (and microcontroller) come from a seperate power supply [PSU], as to not overload the USB connection from the pc. Its 5 volt power rail usually maxes out at 900 milliamps for a USB 3.0 port, corresponding to 4.5 watts of power provided. [<https://www.elektronik-kompendium.de/sites/com/1310061.htm>]  
With up to 125mA per LED (160 LEDs/meter at 20 watts/meter according to the sellers data sheet <https://de.aliexpress.com/item/1005005922059410.html?spm=a2g0o.order_list.order_list_main.30.12475c5fYDTsO9&gatewayAdapt=glo2deu>) and 38 LEDs in every tile and up to 5 tiles connected to the board at once, a theoretical peak usage of 23.75 watts is too much for a pure USB 3.0 connection to handle.  
Instead, I use a USB Type-C Power supply, supporting the USB-Power-Delivery standard (commonly known as USB-PD) version 3.0 or above. These can support up to 60 or 100 watts of continuous power (depending on the combination of both power supply and cable), which is plenty for this application. This way I do not have to resort to limiting current in software and can make use of the LED’s full brightness at all times.  
This separate power supply gets connected to a special “USB-PD Trigger Board”, a small daughter board module soldered up to my pcb (printed circuit board, in this case not specifically printed for this purpose, but with a simple dot matrix of through-hole solder points, ideal for prototyping).  
This specific configuration has a few specific advantages. For starters, the trigger board can ask the power supply for any voltage between 3.3 and 20 Volts and expose it to the circuit, selectable easily via a set three dip-switches on the trigger board. This ensures most addressable LED strip can be swapped in down the line, for reasons detailed previously, without even needing to replace the PSU, making this setup more flexible. Secondly, this solution still uses an off-the-shelf power supply, available in any consumer electronics store, as these USB-PD PSUs are commonly used to power modern laptops and high end smart phones. Using widely spread components makes it easier for people willing to implement, if it were ever sent out to content creators working with Gravitrax.[ or replicate this setup.]

Anyhow, this separate PSU primarily powers the LED strips as discussed, but can optionally power the micro controller as well. A jumper connects the V-In pin of the micro controller up, which can accept all of the voltages interesting for powering different led chipsets, with up to 12 Volts. This jumper however can be disconnected if the second usb port, the one directly on the micro controller, is connected to the PC for receiving commands during a live stream (or reprogramming of the micro controller when setting up the build of course). This USB connection gets seamlessly wired to the same USB hub already transferring video data captured by the two webcams to the computer.

Software

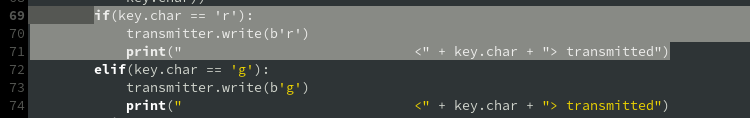
The animations displayed on the LED strips are coded into the microcontroller, while the computer only decides on which animation precisely to play.

Before taking a look at the animations themselves, I will go over the connection to the streaming PC. A serial uplink gets establishes via the USB interface to a dedicated python script, serving as an interface between the Unity application and the micro controller. A dedicated python script opens up a Serial Port

A serial port is opened up on the pc, to which the micro controller can connect via the USB tether. This serial connection is used by a dedicated python script, serving as an interface between the Unity application and the micro controller. The script tells the micro controller what animation to play on which tile, by sending short commands.  
Currently this gets done by simply notifying the µC of changes in the states of the red, green and blue channels in the gravitrax as they happen. The µC calculates the rest (timings, maybe sequences or positions), depending on the configuration programmed into it beforehand. Every connected tile gets assigned one of the three colour channels.  
To make this solution more flexible in the future it is conceivable to make this communication protocol more complex, since this connection functioned very stably and without loss of commands.

Example:

As the light-up-tile script registers a key press – the same key press that is also seen by the script controlling the Gravitrax Bluetooth bridge – it passes this command on to the micro controller.



//As it registers a Key-Press, the Python code outputs the according char out via the Serial Output (called “transmitter” here). Simultaneously a message gets displayed in the console.//

As soon as the micro controller detects commands on the serial port, it references which tiles are assigned to this color.  
If this tile is registered as a track switch, then the tiles’ LEDs are toggled from an animation highlighting the track switch’s left exit, to the right exit – or vice versa. Thereby the LEDs’ animation always highlights the path, incoming marbles are destined to follow.  
Although there is a detail complicating this sequence. There is a second type of animation, that gets only played temporarily. To indicate which tile has experienced changes and alert the user of said changes, every change of states gets highlighted with a separate transitional animation. So before illuminating the new exit, the entire LED-Module brightly flashes for 2 seconds.

[Explain animation math? Or what the anims look like, what their intention is.]

On the other hand, the assigned tile can be of a different type. Marble launcher,s or [see-saw?], are all Gravitrax elements that do not have two distinct states, but return to their original state after performing an action.  
To reflect this property the animation on their corresponding Light-up-Tiles start the flashing transition-animation as usual. But in that case it returns back to the animation it was on before.

-----

Camera Setup

Since there are multiple frames in my stream layout, I can show multiple angles at the same time. But not all views have the same requirements.   
Some cameras may need to provide an overview of the marble run, giving viewers an outline of the path, while others provide a close-up view of higher density and higher importance segments, often those that require user interaction.

The closeups are harder to realise since a higher focal length is the only solution to avoid a disorienting distortion in its picture, create focus through depth of field and keep the camera body outside of the marble course and the other cameras views. Thus I chose DSLMs, mirrorless cameras for their strengths in video recording (as opposed to DSLRs, designed to excell at taking stills) and naturally their mount for interchangeable lenses.

Using mirrorless Cameras for every angle would undoubtedly be the best solution. But since funds for this project are not unlimited, enacting this wish would overstep the budget – in multiple ways. On a surface level, I would have to buy/borrow multiple Cameras with additional lenses, that is self-explanatory. But additionally these mirrorless cameras are - for all intents and purposes – designed to primarily record video and save it onto a memory card. Not for directly passing that video feed on – in real time. That is why their only direct video output is an HDMI port meant to connect to a monitor, whether mounted onto the camera or somewhere on set, not to a computer for streaming into the internet.

Since computers cannot natively handle an incoming video streams, that are meant for direct display output, we need a capture device. There are multiple form factors with different interconnects on the market. Experience shows, the best and most stable experience comes from an internal capture card, mounted directly into a PCI-Express slot on a workstation. That is mainly because the video data does not have to take a detour through a USB controller, that can be bandwidth-saturated. Thereby manufacturers can allocate and use as many PCIe-Lanes as they require for their implementation. A USB connection can be negatively impacted or even interrupted by other peripherals connected to the same USB-controller, such as via a bandwidth over-saturation. Whereas a device connected over PCIe can operate largely uninfluenced by other devices connected to the computer.

Therefore the simple decision to opt for more DSLM cameras requires not only extra Video capture cards to be purchased, but also a PC with a case that is big enough, a power supply with enough headroom, a platform (meaning motherboard-chipset + CPU combination) with enough PCIe lanes to spare (typically found in a workstation or an HEDT-class system) and of additionally enough unoccupied physical PCIE-slots. This simple change in requirements bumps up the price for such a setup tremendously in one go.

As a result of this chain of consequences I limited myself to two DSLMs, one of which I already owned. The other two cameras are simply webcams, their capabilities are plenty for basic wide angle shots and they can be crucially connected via a stable USB 3.2 Gen1 interface (colloquially known under its former name as USB 3.0). Therefore a simple USB hub with USB 3.2 Gen2 (having double the theoretical speeds, bidirectionally) capabilities should be capable of running both of the webcams simultaneously. That leaves us with two capture cards for the two DSLMs. I was able to bring this number down even further by opting for a special kind of capture card. The 2 in 1 capture card “Live Gamer DUO” from AverMedia features 2 HDMI inputs, being able to record 1080p video at 60 frames per second, plenty for our purposes. This is a relatively modern card, with earliest mentions in 2020, enabling me to reduce cost further and open up the possibility for a setup with all DSLMs for later, [previously very difficult without the need for custom equipment] if anyone wishes to translate these findings into a full production set – without the need for any custom made equipment.

On the note on custom equipment -   
From researching how to best hook up all of the recording equipment to the streaming pc, here is the way I would deem the best: Graphics card docks were quite popular in the mid- to late 2010s. Hooked up via a “Thunderbolt” cable, they were used to hook a laptop up with a discrete, full size GPU. This enabled consumers to harness almost the full Graphics rendering power of a desktop GPU for rendering or gaming workloads when using their laptop stationarily. While they have become less readily available with GPU prices on the rise, nothing prevents us from using docks in a different manner. Instead of occupying the PCIE slot in the GPU-Dock with a graphics card, we can instead install the video capture card in there.

This would allow me to place the dock underneath or in proximity to the set and hook up all of the cameras directly to it. It functions as a kind of “daughterboard” or wiring panel, consolidating all of the cabling to and from the PC into potentially a single long optical thunderbolt cable. This works because there are docks that feature USB ports in addition to their PCIe/GPU slot. Alternatively there exist rare, semi-custom docks (like this one by starTech) with two PCIe slots, tan can be outfitted with one capture card each (equaling up to 4 HDMI inputs) or a combination of one capture card and one USB expansion card (up to 2 HDMI inputs and up to 8 USB 3.2 ports).

The latter option is arguably the best solution I was able to find, giving both flexibility and a more organized and stable set. This is highly advisable for a full production as running 4 plus high speed connections all the way from a set to a fully outfitted and manned streaming workstation is risky for safety and stability reasons, especially if the main marble run is supposed to be changed, maintained and updated frequently for smooth operation and to keep the contents of the live stream fresh and interesting for viewers.  
Sadly I did not realize this option, since purchasing a dock, USB card and retrofitting my streaming station with a thunderbolt host card would have again exceeded budget. So this will stand as a suggestion for expansion.

6.2

Now, instead of using an approach based on solely using a software like “OBS” or “X-Split” to decode the video signal and directly putting it out to a live broadcast, as it is common (as established in “state of the art”), I have built my own suite of software solutions. This set of scripts and programs replicates selected functions of OBS and also introduces completely new functionality.

[Nachfolgend steht: Differentiation from the status quo]

This set of tools mainly consists of a central application built with Unity, a set of python scripts controlling peripherals and gathering data, and in addition still using an instance of OBS. The main distinction is that OBS is only used for encoding the live stream and setting up the streams configuration on the streamer’s Twitch channel. OBS is not used for managing scenes and layouting viewports. This task is taken over by the Unity application, that can resize and swap viewports as requested. It reads the camera feeds directly without intermediates and arranges them as textures on quads in Unity’s virtual 3D space(???). Sadly there is currently no way of setting exposure and focus natively with Unity’s [package name for cam capture] for the webcams as OBS can. This makes the DSLMs more advantageous from this point of view, since their output images are configured directly on the camera bodies.

This stitched image is directly recorded by OBS as a screen capture… That has not only the advantage that we do not have to automate switching our camera views within OBS, but this setup has gained the ability to display any overlays or animations – really anything I desire or deem useful. With the full computational power and flexibility of Unity at my disposal I plan to turn this Scene Manager into a smart Interface between hardware layer (marble tracks) and the live stream, that can react and adjust to the viewers and potentially the state of the connected marble tracks. //  
With its combined computational power and flexibility Unity promised to turn this Scene Manager into a smart Interface between hardware layer (marble tracks) and the live stream, that can react and adjust to the viewers – and even the state of the connected marble tracks.

################

The Unity Application consist of different modules(component), which in turn are made up of multiple scripts. I will define a component as a set of scripts and features that make up one distinct function of my program. These can operate largely independent of other components, with clear in- and outputs to and from other components (denoted in the data flow/software component chart).  
Not to be confused with Unity-Engine’s definition of a component (the class “UnityEngine.Component”). Within this ecosystem every script, such as a custom user script or a predefined collider, that is attached to a Game Object is called a “Component”.

The Unity Application is doing double duty. Therefore, these components can be categorized into two categories, the two broad purposes the Unity Application fulfils. Each has its separate data stream, as seen in the flow chart [where to find].

The first one imitates the features that I need from a recording software like OBS and refines them towards my used case. I will refer to this as the Scene /Video composer. The second one is a custom set of components created only for this stream, with the purpose of letting viewers take control. It refers the users commands to the marble track or the Scene Composer. I will refer to this as the Viewer Control System?

The Scene Composer

The Scene Composer takes over a features that OBS handles in conventional streaming setups. (At its core) It accepts video feeds from the sources (Cameras) and arranges them into a final image, a multi camera view. The video feeds are live and not pre-recorded. Furthermore (features like) a reactive UI and animated swapping video feeds expand the conventional feature set.[more are Sfx etc]

[For the following paragraph, please refer to the classes “CamAssignHandler” and “CamAssignAgent”, to find the discussed code.]

wc – tx -mat – rect

A certain “WebCamTexture” is requested from the video camera – found in Unity as a “WebCamDevice”. This is a special type of texture, integrated into base Unity Engine, designed to update every frame, for use with live video feeds of any kind. Along gets passed our desired resolution and frame rate, here 1920 by 1080 pixels, with a 30 hertz refresh rate. This request is not always fulfilled correctly, based on the combination of recording hardware and drivers in windows, so you can find a check for the output resolution in class “CamAssignAgent”, lines 32 to 39. This data gets exposed to the user to help with troubleshooting on the video’s game object, when running the application in the Unity editor.

To actually display the received WebCamTexture it must be mapped onto a 3D model existing within Unity’s 3D game world, since it is seen as a texture by the engine, able to be displayed on e.g. a billboard standing within a game’s world.

This model has a material assigned, that has properties defining how it is displayed in this game world. One of which is a texture. So by getting the objects renderer, finding its material, and locating its “Texture” property the dynamically updating WebCamTexture mapped onto a model.

That model can be regarded as a “quad”, a two-dimensional face, consisting of only two triangles. It has no depth or volume, and no back-side. This quad is now replaced with a rounded rectangle, still with no depth, but made of more triangles. This rounded rectangle was made in Blender and imported into Unity as an .obj file. Since the texture can only be rendered on pixels where there is a mesh rendered, the rounded rectangles functions like an alpha or clipping mask. That is how the camera frames appear with rounded corners in the newest build.

[Screencap of resizing calc]

But since the rectangles have an arbitrary size, and their aspect ratio does not match the different types of cameras used, the corresponding texture of each camera needs to be scaled and cropped. To calculate the correct scaling, the code figures out, which side is smaller in relation and crops the other sides, to circumvent distortion.

The calculation takes into account the resolution and aspect ratio of the input frame and the size of the rectangle (in the viewport) the frame gets projected onto. To scale and crop the image without distortion. The later addition of rounded corners around each frame required me to compensate for the dimensions of the rectangle mesh in addition.

The result is four surfaces, with camera views correctly scaled, projeceted onto them.// with camera views textured onto them. To finally render them out as one image, I utilise traditional game-engine tools. A virtual camera with orthographic view is pointed at the rectangles, framing them with the desired margins and positions. While the rectangles already lie within a common plane (i.e. share a Z-position), an orthographic view ensures that no element including overlays gets distorted or scaled by the distance from the camera. Therefore it can replicate the same look on every computer and the resulting picture matches the arrangement seen in the “Scene” view.

On top of this rendered out view of all camera feeds, all “Screen-Space Effects” get rendered. [Screen-Space elements ignore the virtual world with its 3D objects behind the render, they only consider the final rendered out image, pixel by pixel] While there are currently no traditional Screen-Space Effects utilised, this includes a canvas with the UI, that gets added to the final image in this step.

The UI consists of a combination of sprites and text elements. These make up the UI components, showing users the available colour channels, names of cameras, a window for general information and incoming commands. [It will be detailed below, in the Viewer Control System.]

The assignment of cameras is done via a small on screen menu, only meant for the streamers eyes. It is a simple menu coded with the Unity IM-GUI feature and can be accessed by pressing the F12 key. Here the streamer can see all incoming video streams the pc recognizes and select, which are the real cameras showing the right angles of the set. This feature is necessary since even with all unrelated cameras disconnected from the computer, some software like OBS or the Logitech software for controlling their webcams create something called a “virtual camera”. These appear like actual web cams to the PC, but show a black frame until they are configured correctly.

Viewer Control System

The Viewer Control System is the unique selling point, that lead to the development of this software solution. Its purpose is to sift through messages, viewers send in the live chat, identify commands for controling the stream, and enacting those by distributing them to the appropriate components.

The first step in this chain is a set of Python scripts called Twitch Plays. This is a customized version, based on the implementation of Twitch creator DougDoug [link] used for his streams. The script logs into the Twitch Api with the streamers’ account credentials. Afterwards it checks for new messages, posted in the live chat. By comparing the messages to strings stored in code it detemines if a message should be treated as a command. Invalid commands get discarded. If a valid command is recognized, it gets passed on to the Unity application, in junction with the messaging viewers’ user name.

I achieve this by emulating a keyboard and encoding the message into individual key presses, that are received by unity. The library “Pynput” is responsible for sending the individual key presses. My code packs the message as follows:

A colon (:) starts the transmission, Unity is now listening. Now the command is transmitted, converted to upper case. A comma (,) marks the end of the command. Afterwards the user name is sent, again in upper case. Lastly, the entire transmission gets completed with a period (.).

Notably, all upper case keyboard inputs are reserved for “TwitchPlays” script to Unity communication, while Unity to “GravitraxCli” script communication ignores all inputs apart from lower case characters.  
It is ensured that only valid characters are processed, by comparing to a dictionary of all valid characters and digits. These dictionaries output the according key code (i.e. the unique identifier of the keyboard key to be pressed) and additionally convert German “Umlaute” to their regular counterpart.  
The script “Chat Interpreter” receives these transmissions and refers them to the next scripts in the chain (e.g. the “Gravitrax Connex” script).

The Gravitrax connection – pt1

After a command meant to manipulate the marble tracks has successfully reached Unity, it is distributed to the responsible systems. One half of the task is to reflect the command within the different visualizations of the UI.

The most eye-cathcing change is a pop-up, displaying the command with the exact user name. The user gets attributed for quick feedback and to give them a feeling for the amount of engagement and active users. The command gets displayed as a circle in the colour it represents. This blob flies in a parabolic (self-coded) animation to its button, to more directly a confirm a users input – in a timely manner.  
  
For the current tests a mode is activated where each command directly executes. But for other, more involved streaming concepts (like the last one detailed in Concepts) and higher viewer numbers, a voting-mode can be activated. This needs a threshold of votes to be collected before commands gets executed. The progress of voting is reflected in a circle bar filling up around those buttons. They progress to their next state in an animation when a coloured circle particle reaches it at the end of its animation.

Either way, if a command is decided to be executed, the info will be sent to the Gravitrax Power Connect Bridge. But once again via a python script and emulated keyboard presses. The InputSimulatorScript class handles this. It creates a new InputSimulator and has different keyboard input patterns saved, needed referring commands, but also for terminating any python script. Therefore it also handles launching all python scripts on start up, and terminates them as the app gets closed. The InputSimulator functionality for emulating a virtual keyboard is part of a .NET package called “WindowsInput”.

This approach was chosen for its two advantages. Firstly standard working scripts, often bundled with the API make it faster to get a working prototype up and running. Twitch Plays was implemented in my own way and severely altered, but no new libraries needed to be imported. And the GravitraxClient (found under gravitrax\_cli.py in the code) script is pretty much exactly as it gets distributed by Ravensburger. This makes it far less risky to develop, by incorporating fewer points of failure since most external scripts are tested and trusted. Furthermore this can make it easier to adapt new versions of APIs and new implemented features when companies behind them release updates. Besides, this modularity makes components individually testable up to today.  
Secondly did this enable me to build smaller demos, that can showcase select features without Unity running entirely. E.g. the TwitchPlays\_DEMO scripts directly sends the correct lowercase commands that the GravitraxClient and LightSpacer script accepts. This was useful for live demos.  
But a downside with this approach is, the MulticamStreamer app must remain highlighted at all times, to be able to process commands. This also means the keyboard is not reliably usable while the application is running.

The GravitraxClient script receives all of these commands as an individual lowercase character (such as ‘r’ for toggling the red channel). On startup it has connected to the Gravitrax Power Connect Brick via Bluetooth that receives this data, and refers them on via Ravensburger’s own radio protocol to all Gravitrax Power enabled interactive bricks.

Simultaneously those same keyboard inputs get also registered by the LightSpacer script. It reflects the change in state of the Gravitrax brick in the LED animation of the Light-Up module right underneath. This process is further detailed in the “Hardware” chapter.

Multicam configuration

If a command is meant for the configuration of the displayed camera views on the other hand, said command gets passed on to the other half of the Unity app discussed prior, the Scene Composer.  
Here the CamGridHandler, responsible for the viewport configuration, receives the command. It details, which camera the user wants to see in its full screen view on the left side. This camera view is pulled out of its current spot and placed in the full screen spot in a parametric parabolic animation using the same base calculations as the coloured blob animation used in the user name pop up.

Miscellaneous Features

There are also a handful of minor features, I developed trying to strengthen perceived responsiveness and improve feedback to the player...

A simple example is the addition of sound effects (sfx). The sfx accentuate changes in the UI and give players direct feedback that their commands have been received as soon as they come in without delay. This trick is commonly used in video games and sometimes in other UIs, to confirm every successfully input, mostly in menus. [<https://clarity.global/news-insights/team-insights/using-sound-in-ux-and-web-design>]  
These sounds are currently implemented to play at the beginning of a camera view swap.

Furthermore the animations mentioned prior help guide the user’s eyes to make changes in the system more apparent and to make the user’s impact more tangible. And they can visualize processes to make sequences more easily understandable (like adding a vote to a ballot by showing the vote as a coloured circle flying into the ballot visualization).

Then there is a small information panel at the bottom centre of the screen. By default it shows a short call to action on how to take control of the stream. But additionally it can display adaptive tool tips and respond to actions from viewers. Every sent command gets taken note of and some preliminary data is stored into a JSON list permanently (such as if a viewer is new, and when they have last sent a command, and their last command). While this system currently only greets new viewers, when they have never been here before as a proof of concept, this feature could be used to aid viewers, that have trouble typing in commands correctly. It can call a user by their name and suggest features for them to try out. It can give hints at specified times. In short, there is a lot of potential for testing and some truly promising options of reacting to users actions dynamically, a feature only possible by using this Unity approach.

Flags were once designed and 3D printed to indicate the colour channel an interactive tile is set to. These were a big improvement for readability. But ultimately I canned the flags since they are sometimes mistaken for showing the active path a track switch is set to. They communicate false information to test subjects, that was never intended. As a result I developed the Light-Up tiles, that offer full reconfigurability in what information they convey, through their LED animations.

Twitches SOTA

- Integrate stuff into stream as dono rewards/sub reactions

- Maybe toon-tubers / chuck things at me mechanics

- Sheer price of AWS video streaming (pirate software vid)

Let’s take a look at streaming platforms and the state of the art for streaming live video content. As a first layer of distinction there are two distinct extremes of a spectrum, a stream can be categorized into. The one end is colloquially referred to as IRL-Streams – “In real life”. Content creators can show themselves cooking a dish or building a computer, even exploring an interesting city with their viewers. {maybe mention relevance during 2020s covid pandemic and gather frequency from a couple of streamers}  
For instance: As a recent developments a particular streamer hosted himself conquering a marathon live for charity [CdawgVA cyclathon] and hosted an auction [CD auction] both raising money for a good cause (the Immune Deficiency Foundation) – and in turn winning them critical acclaim [CDAwgVA wins streamer awards].  
This designation exists in distinction to the second category. While it is difficult to find a concrete name for category number two, it is the one commonly associated with streaming as we know it today:  
“Justin.tv”, a web page created in 2007, was launched by Justin Kan and their partners with the intent of streaming their life to the public 24 hours a day. [<https://en.wikipedia.org/wiki/Justin.tv>] While the website was known in its niche it is not the site famous for livestreaming today. This honor falls to its sister-site Twitch.tv. Derived from Justin.tv in 2011, it was founded specifically with the intent of broadcasting gaming content. Twitch was able to outlive Justin.tv, all the while Justin.tv ended its service in August of 2014.

https://trends.google.de/trends/explore?date=all&q=%2Fm%2F0hgpf7h,%2Fm%2F02q7ws6

---continue with “virtual” content, the second category, consisting of gaming and coding etc.. Building a bridge to twitch plays maybe, definitely discerning facecams, vtubers and toon-Tubers

Out of the newly founded Twitch arose the second category of streams, I will refer to as a “virtual” or “captured” livestream. Notably, Twitch’s original terms of service banned any content apart from video games.  
Games → to coding, react, et cetera

This gaming content only requires a powerful enough computer and a broadcasting software in its simplest form. To capture a PC-games’ screen directly off of the computer screen, and stream it to the channel, no additional hardware is required. Digital video capture cards, that became more affordable in the mid-2010s, can be installed[retrofitted] into a PC and enable streamers to capture video off of a game console and also stream console games to the web. This can be optionally augmented by overlaying a small web cam feed of the streamers face onto the frame.

Although, the platforms initial laser focus on gaming turned out to be quite restrictive for streamers active at the time. Long-Time Streamer “LilyPichu” recounts in an interview with the current Twitch-CEO Daniel Clancy:

I have been on Twitch for so may years – I’ve seen it start from just purely a gaming [platform]. […] I used to get in trouble if I wanted to draw on stream. […] They wouldn’t let you. […] I remember just sitting and talking and they came in my chat and were like: <<You have to play a game!>> […]  
So I just booted up an ARAM-game in League of Legends, [be]cause […] that’s easy to talk to. – Now there’s a whole “art” category.

[src in Telegram]

[While there is no information on when exactly this ban was lifted, this change seems reasonable, considering...]

This ban however was gradually lifted, especially considering that its partner-site Justin.tv went under in 2014, a mere three years after Twitches launch. In this process content formerly hosted on Justin.tv found its home on Twitch, organized into different categories for easier navigation.

In this wake not only IRL streams became more popular, but also “indoor content” brached off. People broadcast themselves not only gaming, but also creating[or working]. One derivative mentioned by LilyPichu are music streams. Here viewers can enjoy streamers jamming or learn about music production. Also such streams as drawing streams or animation streams are not uncommon. Nowadays even concepts like coding streams can find success, as seen in the work from Pirate Software, Michael Reeves and DougDoug.

While all of these are far removed from gaming, they happen within the confines of a creators streaming room, making it difficult to lump them in with IRL-streams.

The common denominator is their core content is captured off of a computer screen. Of course their layout can contain a face cam. Some streamers might even include a separate camera showing their pet and other interesting or interactive elements on their set. But this is not the main subject, therein does not lie the unique selling point.



Depending on the Genre, they have made different advancements in the last years, even though they do mutually inspire one another.

Nowadays between these two extremes, a lot of creative stream concepts have developed. These blend the two once distinct categories more and more, and have inspired my streaming idea.

Namely, DougDoug’s content was my biggest inspiration. His streaming ideas revolve around the idea of giving viewers a prominent place on his stage, their input is often the main content his stream revolves around. He writes code to let viewers write their own creative stories [DnD] … or interfere with his gameplay [example stream]. I have identified two unique advantages in his stream compared to traditional streams, firstly deepening the interactions between the viewers and him and secondly letting the viewers interactions with each other shine.

Streamer Scott Falco developed on-screen gimmicks, like an interactive pet, that reacts to viewers and give each user an individual appearance as an avatar on his stream [https://youtube.com/shorts/mmucDoUPslQ?si=Oe8SJSccgmpFTJPZ].

Michael Reeves has shown that viewers can enjoy by hardware tinkering and respond positively to electronic gadgets. Even simple ones like a drill that spins, when a new viewer subscribes to his channel. In his short streaming career he displayed quick growth and peaked at 34k viewers after only 54 stream days [<https://streamscharts.com/channels/michaelreeves>] (a lot of this rapid growth can be probably attributed to his fame on YouTube, building robots and tech gadgets).

But technology-wise, all of these, still simply use the industry standard. OBS, Open Broadcast Studio has been the state of the art for a while. It is a popular recording and streaming software, that collects all recognized video streams on the PC, like web cams and capture cams, but also open application windows and entire screens. All of these video feeds can be arranged into place and the stitched image is either recorded or streamed to any popular platform. Furthermore a simple audio mixer is integrated. All of these configurations can be saved into “scenes” that can be switched on the fly.

(The more experimental streams mentioned prior additionally use at time some Python scripts to read from the Twitch API. But layering Unity inbetween is quite unique, this is a unique feature to my approach.)

Features

- Greet users and remember them - adapt tooltips accordingly

- Sound effects

- Animations

- Voting capabilities

-

Evaluation

Getting here was a rocky road, with many ups and downs and detours.  
But in conclusion I have built a software package that works, with a set of recommendations for hardware plus best practices if a reader wants to build a similar system.

Qualitative user tests have shown this concept to be promising. Users reported generally to be interested by the concept, while some users were genuinely intrigued and entertained by the novelty factor for a good couple of minutes. These tests have shown some shortcomings, that have been fixed by now, most notably the disadvantages of the 3D printed coloured flags. Resulting from this I developed the Light-Up modules as a replacement.

So now as a next step, a new wave of user tests are in order, to gather more feedback and prepare an expansion of the scope and complexity of the project. But sadly my time with this project ends here, so I will place a cut-off point here. A continuation of this project is well possible, but this bachelor’s thesis ends at this point in time.

Delay times have also been managable, depending on the internet connection. With a stable wired connections, I could measure response times (from sending a command off on the client device to changing the state of a Gravitrax brick and receiving this final image back on the client device) as low as 1.8 seconds reliably. Sadly since getting a full access internet connection in the university was not possible I had to resort to a mobile data hotspot. Here the experience was very inconsistent and could get as bad as 6 seconds after 10 minutes of use.

Outlook and possibility for expansions

To keep motivation up for a longer time I would like to try one of the more involved concepts from the Concept chapter in the future. This concept has done its job of making it possible to develop the soft and hardware solution for remote controlling a table filled with marble tracks with reduced complexity.

But apart from structural points there are many features, that promise a very unique experience.  
Spatial orientation in a highly vertical marble track can be made much easier by implementing a type of stylized 3D map. Camera positions and points of interest can be indicated there, the perspective of this 3D map can also change (e.g. based on the current full-screen camera) in real-time, since Unity is a 3D game engine.  
A particle system could highlight points of interest on the video feeds directly, and accentuate changes, by transferring some points of the marble tracks into 3D space. These highlights could show what is about to happen directly on top of the cameras pictures and hereby bridge delays further.

Apart from those features, the benefit of transforming this from a prototype you can very well try at home with the expertise necessary, into a workinng product that works right out of the box is apparent. Currently there is too much tinkering necessary to get the software suite up and running on a new PC with its setup. So by refactoring and maintaining the code, and automating the installation of necessary python packages, this could be as easy to use as following a wizard with five minutes of setup time.

Making the software package more stable and resilient is a given. Switching the way of communication between Python scripts and Unity app from key presses on an emulated keyboard over to a system of web-sockets would accomplish this and let the streamer operate other software while letting the Multicam Streamer run in the background.

My personal priority might be creating a fork of this software. I would like to release a version of this software without the focus on a set of marble tracks, geared towards any streamer. Since a good bit of the features do not strictly apply to the marble tracks and during their design improving the viewing experience of any live stream was a major guiding principle of mine, the entire software package was always designed to be forked off to create a tool to improve any stream. It can still be used as a middleware between receiving video inputs and outputting the image to OBS, by extracting the Scene Composer system. The other half would be reworked, but users could still be greeted smartly and individually. Animations are still a nice immersive upgrade, to swapping scenes in OBS. Viewers could mess around with the cameras and modify the image of the streamers web cam. And the possibilities for customization or gimmicks like small mini games like animated avatars for viewers to bridge downtimes in streams are vast.  
All of these features just work just as well for a gaming live stream with a face cam, and can even give streamers an incentive to give their stream their own twist (e.g. by simply adding another camera, showing their pet).