“klangpong”
an approach to sensocial gaming

Interaction Project summer term 2008

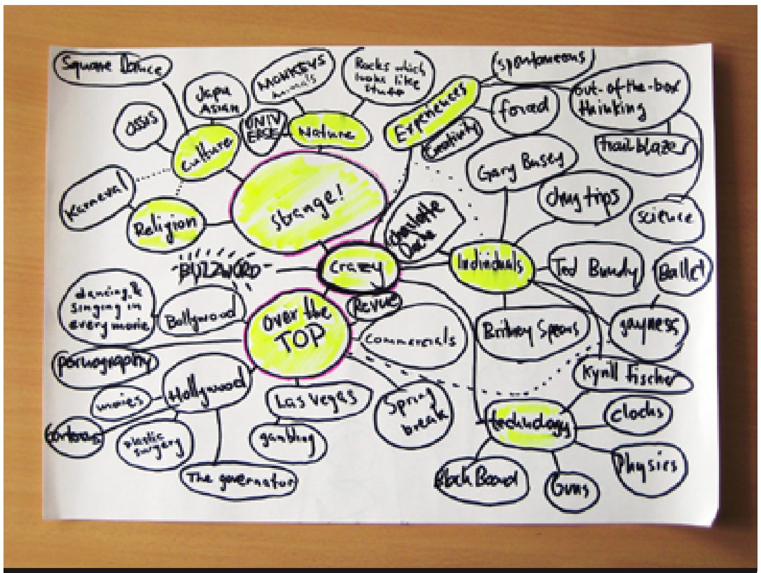
Alexander Ewald, Johannes Freund,
Sebastian Gerhard and Dominic Szablewski

“Construct a Crazy Machine, which the user can run and use!”

The task, which we were assigned to, challenged our team of four people to develop a crazy machine, which can be run and used interactively by users. The machine should make sense and should have a reasonable purpose – that was one important criterion. Obviously, the most important feature of the machine should be its possibilities for interaction: the user must be able to interact with the machine in different ways and with diverse actions; the steps of interaction and the usage of the crazy machine should be recognized by the user without a manual or a help function.

What means “crazy” for us?
While talking about our assignment, we faced the first challenge. What exactly does the term “crazy” mean? And what is the subjective impression of the expression “crazy” for us as a group?

We began to analyze the meaning of “crazy” during a practical session for the Media Design 2 course, where Mr Gilgen and Ms Söller-Eckert taught us various creative techniques.
While working on a mind-map with the topic “crazy” and using the technique of visual synectics, we discussed several aspects and perspectives of “craziness”. The first things, which came into our minds, were especially the negative sides of being crazy: insanity, loony people, mental illness, craziness in a dangerous and aggressive way or maniacs. But after a while, we realized that “craziness” can also have very positive aspects. We thought about great geniuses of science, physics or creativity, who were crazy, but in an ambitious, innovative, resourceful and world-shaking way. Consequently, we discovered, what “crazy” really means for us: being groundbreaking, unexpected and imaginative.



The mind map we created during the lesson

A look into the New Oxford American Dictionary afterwards revealed an adequate definition of the word "crazy":

crazy: *"appearing absurdly out of place or in an unlikely position".*

We decided on the the "crazy" meaning "appearing absurdly out of place". Due to the fact that the our course is titled "interface project" we 've been looking for an idea that describes and makes use of a crazy interface. An interface you wouldn't expect in the context it is used in. Or to come up with an interface that allows communication and interaction between users that are not able to do that with "mainstream" machines.

Alternative Ideas

Battle Painting

Battle Painting was intended as a timebased two player fun game. The goal of the game was to fill in the given time as much as you can of a blank canvas with your color, to beat your opponent. To extend the gameplay and make it more fun to play, we also thought about Power Ups, like colorbombs, paint buckets etc. To make the game more gripping, we planned to have a Online Highscore, where you could compete against others.

But finally we decided against this idea, because of the limited interaction possibilities.

Mechanical Parts
This idea was about an mechanical organism which a player could build in a Petri dish.
The goal of the game was to collect as much „food“ as you can, so your organism can be extended with further parts. We thought of different parts as engine, mouth, and so on, consistent of different materials, which the player could combine to toss out the best combination. To garantuee a long-term game experience, we had the idea to unlock new parts for your organism.

We dropped the idea of Mechanical Parts, based on the fact, that we discovered, Electronic Arts was working on a game named „Spore“, which had quite a lot of similarities with our idea.

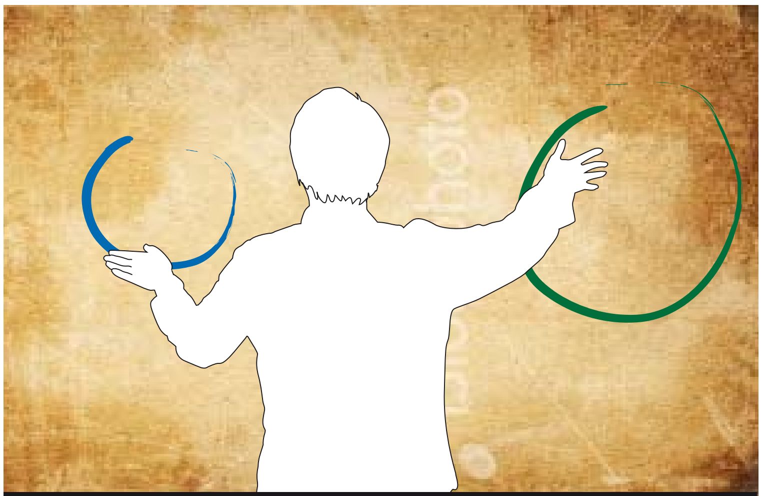
sketch of a mechanical organism

Audicolor

Audicolor was intended to be an interactive experience that allowed the user to generate sound and visual effects using nothing but his fingertips, using gestures like drawing on a virtual canvas projected on a wall in front of the user. The gestures and the lines which were drawn thereby, should be analysed and sounds were intended to be generated after them.
Because of this natural input-method the user had a rapid access to the game and could play it without many experience with games or any type of interface.

Fascinated by a video-demonstration given by a MIT-student, where he used a Wiimote to track the movement of fingers, we tried to reproduce this technique. The video-presentation as astarting point, we gathered a lot of information about the needed hardware, software libraries for Processing and so on. It took us quite a while, but it was quite interesting to toss out the right configuration and to puzzle with thiselectronic engineering part.

In the end the idea of „audicolor“ was dropped and we have some electronic parts left. But the fun we had with this „experiment“ outplays the misinvestment.


conceptual sketch of "Audicolor"

The Final idea

Our final conceptional process, which led us to our final product was driven by three basic ideas:

1. Design an interface you wouldn't expect in the context it is used in.

2. Shift the user/machine and user/user communication on another level and allow a way of communication that is not possible under "non-crazy" circumstances.

3. Due to the fact that the outcome of our project will be more likely a game but not an application which serves a certain purpose, we looked for a game principle that runs nonlinear and thereby encourages the user to play it again and again.

An audio project in the first semester, where we dealt with acoustic visualisation of a "dream vs. reality" situation, led us to the idea of a whole environment, which is tangible entirely by sounds.

This process led us to "Klangpong", an auditive interpretation of the 1960ies Arcade classic.

First Development & Research

The research we conducted revealed that there are games for blind people and also games that are both playable by blind and seeing people. Not only translations of existing games but games that are developed explicitly as audiogames.
The only working audio-mod of a mainstream game is "audio quake". Quake is a first-person-shooter Quake, which needs intense training both for seeing and visual impaired people to succeed in the game. "Pure" audiogames are mostly little challenging due to simple game concepts. Only very few offer a multiplayer mode, which would make the whole audio-gaming more interesting by letting blind and seeing people interact.

A conclusion of our research was definitely that blind people are mostly mistaken in terms of their ability to deal with the world of the seeing people and particularly with technical devices.

In the next step, we analysed modern gaming consoles and their concepts.

Gaming on gaming consoles has undergone a impressive development in the last years. Nintendo, one of the veterans in building gaming consoles and developing compelling games, released their Wii and DS lite nearly two years ago. Both machines are part of their “touch! generations” concept. “touch! generations” is intended, as the name implies, to address all ages and types of gamers. The main idea behind is that the gamers don’t need you to play it for hours to succeed or to have broad experience in gaming.
But by using an intuitive controller-concept and custom-tailored games, not by transferring titles from other platforms, they get back to the core of gaming.

That’s were we intended to link our approach, but with the extension to bring up a social gaming experience for seeing and visually impaired gamers.

Experimenting with sounds

We tried to define different types of sounds that help to elaborate the game and the various in-game situations.

a. abstract sounds
Sounds that can’t directly be identified as specific objects. Like warnings and indications for statuses and so on.

b. concrete Sounds
Sounds that can directly be identified as a certain object, background sounds and athmos.

c. spoken language and music
Spoken language can be used to illustrate in-game situations, to generate a menu or to inform the player about things that are too complex to be “shown” by abstract sounds. Music is mostly kind of obsolete because it generates “noise” that distracts the user from the more important signals.

Besides sound, another relevant technology, which is quite important and useful for audiogames, is the force feedback-technology. It does not offer the same range of possibilities as sound but can support and emphasize a particular situation.

Refining the idea

So, the first content was clear: “Klangpong” should be a game with a simple concept for two players. Pong, the revolutionary but very simple game, which nearly everybody knows, provided the groundwork for our concept. Comparable to a tennis game, two rival players try to score points by hitting a ball in a way that the opponent cannot reach the ball.

The creation and the definition of the match field and the whole gameplay is based on sounds and audio effects—it’s played without a graphical interface so it must be represented entirely by sounds and be understood by hearing. The two players are represented by paddles and face each other, one player on each narrow side of the rectangular playing field. The ball, boards on the left and right side and the individual goal area behind each player are the needed elements.

In this early stage of conception we already decided that our application needs a menu from which the user can choose a play mode, and a tutorial, which leads the user into the world of “Klangpong”.

Interaction

Finding a metaphor for the steering of the paddle was quite difficult. How can we explain or illustrate the main interaction to someone, who probably has never seen a pong game or has never played tennis? Thinking about a box, shaped like a paddle that the user can grab with both hands and move from side to side, a balance board which transforms the balance of the user’s body in the paddles movement, seemed to far fetched and raised another wave of problems. For example a real “paddle”: This would require the user to know the maximum angle or position to move the device, requiring the user to “learn” another layer of sound or another abstract sound, indicating the position.

interaction

So we ended up with a steering wheel, which seemed to fit our requirements best. The maximum position of a not affixed steering wheel is defined by the anatomy of your elbow- and wrist joints. Steering to the right requires you to move your right hand down which implies you to lean a little to the right, the direction you move the paddle to. Vice versa when moving to the left. Also for people who know how to ride a car—turning to the left makes your car to move to the left.
Due to the position of the players by "really standing" face to face, it works for both players. All our testers stated that the idea of the steering wheel worked great for them.

Furthermore the constantly changing in-game situation require the user to react very quickly which is much easier with smooth moves of the turning wheel.

To simplify the handling of the Wiimote for blind users or players, which never used or saw the device before, we decided that the usage of Nintendo's Wii Wheel (a little wheel, which can be combined with the Wiimote) would be very reasonable for the controlling and understanding of the gameplay.

So the interaction in "Klangpong" happens on another level than simple mouse-clicks or tapping hot keys on the keyboard. Therefore, the interaction does not happen stepwise, but dynamically and not limited.

Interface

"Klangpong" is intended to be played "blind" without seeing the screen. Therefore, the interface is invisible – our aim was to design a strong and understandable audio interface. Spoken language, sounds, tones, experimentations with volume, audio channels and audio effects should guide the user through the menu and the game itself. Of course, we thought a lot about the question, which sounds are enjoyable for the user and which ones hurt or annoy. Also the question if the sounds can be easily distinguished or not and if they guide the user succesfully, was discussed frequently.

Voice User Interfaces

Thinking about the design of our interface we thought about audible interfaces, called "Voice User Interfaces". Do we have encountered such interfaces yet? The first thing we came up with were car navigation systems. While driving, the user of the navigation system is unable to focus on the tiny screen or to wait for response after having made an input. So the whole system must be able to guide the user to the selected destination, mainly without looking at the screen. The direct interaction between the navigation system and the user is reduced to the setup before starting the ride and the changes the driver makes while following the navigation systems instructions. The navigation system has a defensive design, it is not focussed on how to avoid mistakes by the driver, but how to guide him back on the right route, whenever he didn't follow the instructions.

They don't make use of concrete or abstract sounds—the interface is designed using spoken language only, like a co-driver.

problems regarding VUIs

The second "VUIs" we thought about, were telephone hotlines or telephone shopping systems. We summed up the experiences, issues and problems with such systems on a short list (leaving speech recognition out):

» users impatience

» awkward language

» open ended prompts, endlessly repeated confirmation-questions

» misleading mental model behind the whole system, meaning that the different steps were not in the order you would expect them.

» missing skip functions for returning users

So we tried to avoid those mistakes by creating an intuitive structure for both parts of the interface.

Interface part 1 — menu and tutorial
We had to create a mental picture in the user‘s head, which explains the setup and the whole gameplay, by a spoken tutorial, which stepwise brings the user closer to the game. To make it easier to understand, we connected the tutorial directly to the actions required in-game.
At the beginning of the game the user receives a blindfold,

headphones and the steering wheel. If the steering wheel has been picked up, the accelerometer of the Wiimote recognizes the movement and the tutorial starts with a short time delay.

(If the user has already undergone the tutorial he is able to skip it by a press of the button.)

Now the Voice User Interface comes in use: the voice tutor explains, by different scenarios, how to interact with the steering wheel and how the ball sounds, as he crosses the field. By combining the single sounds due to the game-situations, the user should be able to play the game after passing the tutorial once.
To give an example how one of these scenarios sound like, this is Step 8 of the tutorial.

Voice: „Your opponent plays the ball quite similiar to you. Now it‘s time to play some rallys!
Pay attention about the difference in sound, between you, and your opponent playing the ball.“

The last tutorial step, is about the button, used for skipping the tutorial and for menu selections, at the rear of the steering wheel. The voice tutor guides the user to find the button and press it.

In the main menu, the user navigates through the existing menu items guided by the voice-over. Each menu item is read to the user, as he hovers above it.

Interface Part 2 — the game
To find a convincing audio interface, we experimented at the beginning of the conception with many different sounds and sound-themes for the elements of the game: The paddles, the ball, the boards and a “miss”-sound that alerts the player when he has missed the ball.

We analysed these elements and pointed out the details that have to be transported and communicated by the sounds. After having set up a working prototype of the game, we tested the “sound themes” we had developed.

Urban/city/traffic theme
We tried to create an atmosphere which reminds the user of standing in a crowded city with sounds of cars and traffic.

The game situation could have been designed like the situation when a blind person has to cross a street and must pay attention to all dangers of a crowded street.

Outer space theme
we thought about designing an abstract atmosphere and an abstract space. So we imagined a sound theme with sounds that reminds the user of flying through the outer space or standing in an aerospace port. The used sounds were influenced by typical effects of science-fiction like laser-sounds or energy fields.

Forces of nature/natural elements theme
the imagination of giving the user the impression of playing with thunder, wind or lightnings was very exciting for us. As a next step, the user could have created earthquakes or typhoons during the game.

Conversation/emotion theme
this theme moved away from the idea to use real object in the game itself. Instead of playing with a ball for example, we figured out that it could be interesting to play with "emotions". The users hit "laughter", "screams" or spoken words – instead of a rally, a conversation between the players develops. Naturally, it was more a conceptional and experimental idea.

Water theme
a discussion, which sounds are enjoyable and comfortable for our users, led us to a water theme. The idea was to create a calm and relaxing atmosphere for the players by using slight bubbling water sounds or a light waterfall sound.

Ice hockey/shuffle board theme
our "realistic" theme. When we thought about playability and understanding of the game situation, we imagined that a realistic theme, with realistic sounds, which the user knows from sports or arcade-halls, could enhance the flow of the game. The whole theme had an analogue sound impression, which helped us finding an adequate sound theme that sounded not too artificial.

decision on audio themes

After our first tests in the design class we decided on the realistic theme, which was the most understandable, playable and distinguishable.

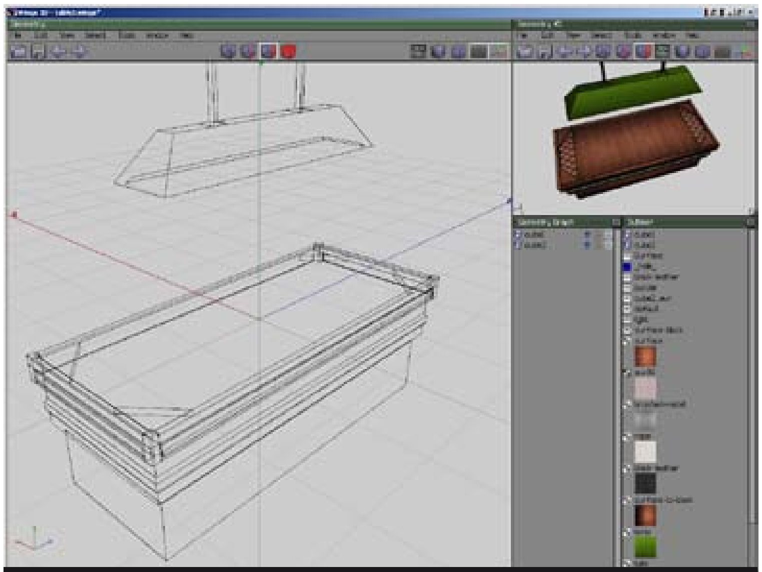
Aside from the design of the sound interface, we thought right from the beginning about a visualization of the game. Of course, the game itself can not be seen by the users, but spectators would be annoyed very quick, if there would be no visualization at all. But we were not sure about the way we could design the visual aspect. One suggestion was to visualize the game analogue to those shuffle boards we know from arcade-halls. Another idea was to design the visualization very abstract, like a wormhole, with changing colors and forms, which represent the process of the game.

Since we settled for a realistic sound theme, we decided to design a rather realistic visualization, which fits to the used sounds. While designing the visualization, we were mainly inspired by the looks of existing air-hockey tables, billard tables and bowling alleys.

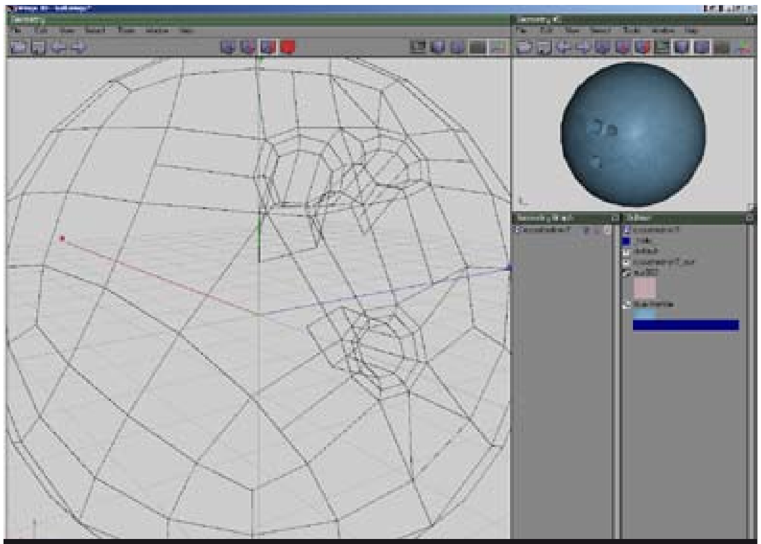
Design concept

3D-Models
The table and bowling ball (Puck) models, where build with the open source subdivision modeler Wings3D. Wings3D can only do polygonal modeling and lacks more advanced features such as NURBS, Patches and other functionality often found in other 3D packages such as Blender, Max, Maya or Lightwave. However, this lack of features provides a fairly shallow learning curve.

The UV-unwrapping functions of Wings3D are often a bit too basic, but still good enough for our purposes.


Wings3D environment with the pong-table

With Wings3D we were basically able to learn the program and build and texture both models over the course of just two days. The finished models could be easily exported in OBJ format and were as good as ready for use in Processing. Only the texture paths needed to be adjusted to work properly, which wasn't a problem, since the 3D vertex data (OBJ) and corresponding material data (MTL) are stored in plain text format.
Most textures were made out of photo sources an post processed in Photoshop. The lamp texture was borrowed from a game.



Wings3D environment with the model of the ball

Technical Concept

Development Environment

Since KlangPong doesn't demand much from the hardware and it needed to be finished in a timely manner we choose the Processing environment to implement our game. In hindsight, this decision was a two-edged sword. The Processing API provides rich features to do about anything you normally would want to do in fast and easy way. But as we went further and further we needed more uncommon features and encountered a few Bugs in the API, which we had to work around. This sadly was a very time consuming task. In fact, we found that Java in general still has many problems when working on applications that demand realtime responses and handle with relatively low level resources. Still, we managed to put everything together as we wanted to. It's doubtful if any other API or programming language would've brought better results.

Sound

The most important aspect of our game is of course the sound. We needed a way to produce a convincing stereophony and make distances and speeds audible. The OpenAL API is the de-facto standard for producing high quality 3D Sound for games. It provides fast and easy access to the sound hardware, and allows you to define a virtual room in which the listener and multiple sound sources can be placed.

The Java OpenAL binding, which we used for our project, allowed us to directly talk to the API.

All sound effects, like the pucks moving noise, collision with paddles or the walls, are cached and kept in memory, as the game starts. All commentary, such as the current score, spoken menu descriptions and the tutorial voice, are loaded in a sound queue on demand. This queue allowed us to easily chain speech samples to construct sentences.

Despite Javas automatic memory management, sound buffers are not released automatically by the Garbage Collector, but had to be deleted by hand in the objects destructor. Furthermore, we had to call Javas Garbage Collector in several places manually, to ensure that sound sources are freed, before new ones are created. Otherwise Java would use up all hardware sound channels with long dead buffers.

Wiimote / Blue Tooth
To Access all functions of the Wiimote, we used the MoteJ library. It provides a fairly low level access to the Wiimote's accelerometer and buttons.

MoteJ can work with several different Blue Tooth libraries to talk to the Wiimote. The BlueCove library was chosen for this task, because it provides robust access to the Blue Tooth stack on all 3 platforms (Windows, Mac and Linux).

Graphics
For our visuals we choose the OpenGL rendering API and OBJLoader library to load and draw 3D-models into our scene. This combination proved to be problematic one, because of Processing Bug. We actually needed to adjust the OBJ Loader library to work around it.

OBJ is a common format, supported by many 3D graphics applications, to represent raw vertex positions as well as UV texture coordinates. Since OBJ files store data as plain text, they are easy to debug, but harder to parse than binary data. However the syntax is quite simple compared to the more powerful but overly verbose ASE format.

OpenGL allowed us to draw huge numbers of textured and anti-aliased polygons in each frame. Because, compared to modern games, our 3D scene is quite simple, we didn't need implement display lists or care about polygon batches. Normally you would want to bind a texture, draw all polygons that share this texture and repeat for all remaining textures. Instead, we just bound the texture for each polygon separately just prior to drawing it. This is inefficient, but the performance loss was irrelevant for our scene.

We didn't use any OpenGL lighting functionality. All lights and shadows are pre "baked" into the textures. The dynamic shadows of the Puck and Paddles are faked with semi-transparent textures.

Networking
To provide a multiplayer mode where two players could compete against each other, we needed a networked mode. Java provides extensive networking functionality, which is capsuled in Processings Client and Server classes. Sadly, these classes are not very well thought out, so we partly had to descent into Javas socket implementation to get it working properly.

A snapshot of the current game state is constructed by the client and the server every 20 milliseconds and shared over a TCP/IP connection. A fast UDP connection was considered, but ultimately dropped in favor of TCP/IP, as it guarantees to never lose a packet and maintain packet order. Since there is absolutely no random factor, other than the pucks starting angle, we decided to synchronize the complete game state only when something important happens. The opponents Paddle position is updated with each received snapshot, whereas the pucks position, speed and angle is only adapted when the opponent hits or misses the puck, or the client starts the game.

So the pucks physics and collision with the local players Paddle is computed completely client side. This ensured a very fluent and precise gaming experience, even when playing in high latency networks.

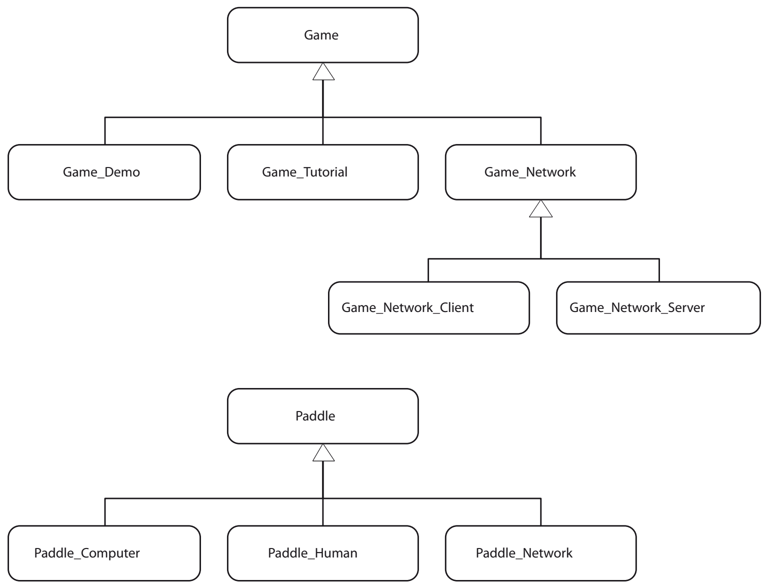
Code Structure
For the most basic functionality of the game we build three classes: Game, Puck and Paddle.

The Game class defines the playing area and its size. It creates and hosts an instance of the Puck class, as well as two instances of the Paddle class. The Game class' most important methods are update and draw, which are called periodically by Processings draw function. These two methods in turn call the update and draw methods of the Puck and Paddle objects.

Also the score for each player is kept in the respective Paddle object, all changes in the score run through the Game object. The same applies, when the Puck hits or misses a Paddle: The physical response of the Puck is managed in the Puck object itself, whereas the Game object handles stuff like sending a Snapshot over the network etc.

Class overview
To reproduce each of our game types we constructed several different Game classes, all derived from a basic Game type. Furthermore, these different classes host different types of Paddle objects, i.e. a Paddle controlled by the computer, by a human or by the network.

For a networked game we also had to distinguish between the server and the client, as the client directly connects to a host and the server listens on a specific port for clients. Still, most of the network functionality, such as constructing, sending, receiving and processing Snapshots works the same on client and server. This left us with the following class structure for our game types and Paddles.



Game
Hosts a Paddle_Human and a Paddle_Computer object and provides the most basic game. The game is finished when one player reaches the score of 11.

Game_Demo
Hosts two Paddle_Computer objects. The camera is rotated around the table. This non-interactive game type advertises the game itself when in idle mode. The demo game automatically ends as soon as someone picks up the Wiimote.

Game_Tutorial
Hosts a Paddle_Human and a Paddle_Computer object, but does not provide regular game behaiviour. Instead, a spoken tutorial can be heard, while the player can try out the game in severall steps.

Game_Network
Host a Paddle_Human and a Paddle_Network object. Snapshots are sent over the network every 20 milliseconds. Incoming snapshots update the opponents Paddle position, or the whole game state, if something special happened (game start, score or miss).

The Game_Network_Client and Game_Network_Server classes are derived from the Game_Network class and only differ in how they connect to each other over the network.

Snapshot
The Snapshot class is also closely related to networked game classes. It stores a complete game state wich can be converted into a byte stream, ready to be sent. Additionally, a Snapshot can also be constructed from a byte stream received from the network.

Paddle
The Paddle classes either accept input from the controller (Paddle_Human), or move autonomously (Paddle_Computer). The Paddle_Network class is updated from within the Game_Network object and therefore doesn't do much despite of drawing itself.

Puck
The Puck class handles all the physics of the itself. It bounces back from walls and Paddles and plays the appropriate sound effects. The puck behaves exactly the same in all game modes.

Controller
We also have Controller_Wiimote and a Controller_Mouse class; both derived from a basic Controller class. This allowed us to easily switch between mouse and Wiimote mode for testing purposes. The Controller_Mouse class is now unused.

MainMenu
The MainMenu allows the player to switch between a game against the computer, or a game against another player over the network. The MainMenus update method returns a Game object or NULL, if no game type has been chosen yet. Additionally, this method returns a Game_Demo object, if the controller has been idle for 20 seconds.

The currently selected game type is explained with speech samples.

Utility classes
Additionally to all of these classes, we also build some utility classes for sound sources and sound queues and adapted a small library for working with JOAL from the Processing.org forums.
A Vector2D and IntBytes class was created solely for convenience. Because of a Processing bug, we also had to build a ImageManager class that keeps track of all loaded textures.