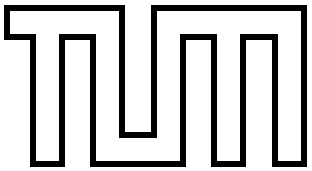


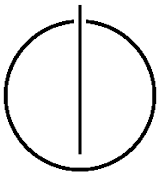
FAKULTÄT FÜR INFORMATIK

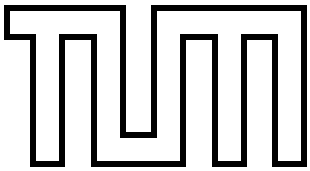
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelorarbeit in Informatik

TUMtris - A tangible multiuser Tetris game

Simon Schenk





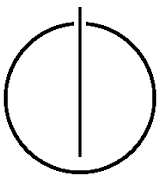
FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelorarbeit in Informatik

TUMtris - A tangible multiuser Tetris game

Author:	Simon Schenk
Supervisor:	Prof. Gudrun Klinker, Ph.D.
Advisor:	Eva Artinger, M.Sc.
Date:	November 15, 2010



I assure the single handed composition of this bachelor's thesis only supported by declared resources.

Munich, November 15, 2010 Simon Schenk

Acknowledgments

I would like to thank all the people who made this project possible. First of all I want to thank Professor Gudrun Klinker for offering me such an interesting topic to work on. Also my thanks goto my advisor, Eva Artinger, who offerd a lot of support during the last four months. My special thanks go to my fellow student Martin Schanzenbach, who was patient enough to help me find some bugs in my code more than once. Furthermore I want to thank my brother, Tobias, for taking the pictures used in this thesis

Besides following people (shown in alphabetical order) contributed to my work by being testers during the playtest:

Andreas Eichner
Aram Messdaghi
Christopher Corbeil
Florian Hartmann
Huifeng Li
Joachim Schenk
Manuela Kunkel
Martin Albert
Michael Hofstetter
Ran Qu

Last but not least, I want to thank my brother Joachim and my girlfriend Verena Saller for reviewing this thesis.

Abstract

During the last years the number of video games has increased steadily. While the complexity, the game speed, and the overall look of the games increase, the control types of these games stay the same. However, the increasing speed and the complex control of video games result in a lack of communication between players during the game.

To counter this trend Nintendo has introduced its Wii game console, which can be controlled not only by using buttons but also with gestures. Thus, games can be played in a collaborative and communicative manner.

In this work the development of a Tetris-style game is described. It uses a tangible user interface as game control. Hence, TUMtris merges video- and board games. Furthermore different game modes are developed, including a single mode and two multiplayer modes.

Finally, the novel control type is evaluated in respect to its usability and effectiveness in video games.

Zusammenfassung

In den letzten Jahren ist die Beliebtheit von Videospiele stark gewachsen. Während die Komplexität, die Spielgeschwindigkeit und der graphische Aufwand der Spiele stetig gestiegen ist, blieb die Spielsteuerung nahezu gleich. Jedoch fällt es wegen der Komplexität der Spiele und der relativ komplexen Steuerung den Spielpartnern teilweise schwer, sich zu unterhalten. Darüber hinaus sind viele der Videospiele nicht mit einem Mehrspieler-Modus ausgestattet.

Um diesen Trend entgegen zu wirken, hat Nintendo vor einigen Jahren die Wii-Spielekonsole eingeführt, bei der es möglich ist, Spiele nicht nur mithilfe von Knöpfen, sondern auch mithilfe von Gesten zu steuern. Dadurch kann das kollaborative und kommunikative Spielen gefördert werden.

Diese Arbeit beschreibt die Entwicklung eines Tetris-Spiels, welches auf einem Tangible-User-Interface basiert. Damit soll das Gefühl erzeugt werden, ein Brettspiel zu spielen, während man ein Videospiel spielt. Des Weiteren besitzt TUMtris drei verschiedene Spielmodi: einen Einzelspieler- und zwei Mehrspieler-Modi.

Am Ende der Arbeit wird evaluiert, ob die tangible Steuerung funktioniert und ob sie sich für den Einsatz in Videospielen eignet.

Contents

1. Introduction

1.1. Motivation

During the last years the popularity of video games has increased continuously. The newest sales figures of video games [6] confirm this. In fact, the video game industry is a very big part of today's economy. One reason for video games being attractive especially for the younger generation is the increased play speed, the overall 'good look' and the complexity of most of the video games compared to board games. Thus, game mechanics can be realized, which are impossible for board games. In fact, many video games have more in common with movies than with traditional board games.

However, the new types of games that have been published for video game platforms produce a new play behavior. Since the faster game speed and the complex game controls demand a high amount of concentration, the players have no chance to talk to each other like the would during a board game. The author can assure this claim due to his own experience. Also, this can easily be seen by observing a so called 'LAN Party' (the game devices communicate with each other over a Local Area Network, hence LAN) where many players play video games together. At these 'LAN-parties' each player sits in front of a PC and plays on his own against the other players. Although there are many people playing the same game, almost no communication exists. If people communicate it is almost never a face-to-face communication as it would be during a board game play, but communication takes place using a headset.

This lack of communication is not only caused by using one game device for each of the players (like in most PC games). On game consoles multiple players are able to play on the same screen via 'split screen': the screen is divided into pieces, each one showing one player's view. Even in multiplayer sessions using this type of mode, where all the players look at the same screen, very few chatting takes place.

Not only the game itself demands high attention but also the game's controls. Compared to board games it is quite hard to control modern video games. For instance, in the well known game 'Ludo' the game control consists of throwing the dice and moving the game tokens, which seems not very hard to accomplish. However, today's game controller (figure 1.1) have about 20 buttons and one can imagine that controlling a game using all of them is rather challenging.

It seems to be quite necessary to improve the communication in modern video games. One way to do so is to simplify the game controls. One step in this direction has been done by Nintendo with the launch of its Wii[1]. This game console does not only use a normal gamepad for control, but also accelerometers which are integrated in the gamepads. Thus the game can also be controlled by using gestures. One goal of this simplification is that the control of a game is not a task at all (like it is in board games). Thus the players can

[1] www.wii.com

Figure 1.1.: A Xbox 360 controller developed by Microsoft as an example for a today's game controller

draw their attention completely to the game itself and the play partners. Another way to do so is to use a Tangible User Interface (TUI). A TUI is an interface that enables controlling a computer by tangible objects instead of traditional input devices such as a keyboard, mouse or a touchpad [8]. By using different objects for different effects a huge variety of functionalities can be achieved. Figure 1.2 shows the 'reacTable' [10], a table supporting a TUI. In this example the user can create music by putting different cubes on top of the table device. In video games, a TUI can evoke a board game-like playing feeling. Hence, there are some projects which implement a TUI in a video game, for instance 'Weathergods' [1], 'KnightMage' [13] or a clone of 'Monopoly' [13]. Although all of them implement a game especially made for the TUI device, these games are quite similar to board games concerning the used game mechanics.

In this project, however, Tetris – a very popular video game – is used to be transferred on a TUI table. Given its popularity, this game is suited to be used in this project, since almost everyone knows the principal rules for playing. Furthermore it offers the typical dynamics of a video game, which means that the game tokens are moved automatically by the computer and therefore the player has to react in time to complete the game.

Apart from changing the game controls the communication between the people playing a video game can be increased by implementing a cooperative game mode. In this mode the players are no longer opponents but partners. To accomplish the game, the partners have to work hand in hand, which normally leads to an increase in communication between them.

1.2. Structure of this thesis

The thesis is structured as follows:

In the next chapter, chapter 2, several publications related to this work will be introduced. They are either about games using a tangible user interface or about modifications

Figure 1.2.: The reacTable using a tangible user interface [10]

of Tetris. In chapter 3 the general approach in game development will be described, followed by the modification made to fit this approach to this project. In chapter 4 the requirements of this project will be defined, characterizing the major points in which this project differs from the state of the art described in the related work. This chapter also includes a description of the development of the hardware used in TUMtris, followed by the description of the software development of TUMtris. At the end of the thesis the play test, which is also the evaluation of this project, is described in chapter 6 amongst a conclusion and an outlook to future work.

2. Related work

Before going into detail about the TUMtris project itself, a look at the major related work and projects is taken. These related projects can be assigned to the two topics of this thesis. The first one is Tetris itself. There are already some projects that aiming at modifying the game mechanics of Tetris, as it is done in this project. The second topic refers to games using a tangible user interface in general.

Besides, some research has been done on using multi touch devices, especially for cooperative playing [11]. To get the results two games have been developed on a multi touch table device. One game is a single and multiplayer game, whereas the other one is a cooperative game only. Although TUMtris is not only a multi touch, but much more a tangible game, some results of this research are interesting for the TUMtris project. It is concluded that the players cooperate automatically in these games, although people who had no experience with video games in general hesitated to use the multi touch device. After a short introduction, however, the controls worked quite well. One problem mentioned is that sometimes it was difficult for the players to determine which objects on the screen are touch-sensible and which are not. In summary it is said that the multi touch mode offers great opportunities for cooperative playing.

2.1. Tetris modifications

Before elaborating the modifications of the Tetris game mechanics, the original is introduced. Tetris was invented by the Soviet computer engineer Alexey Pajitnov in 1984. Being available solely in the Soviet Union first, the game was released in the western world in 1986[1]. The simple but challenging game mechanics of building up solid lines using different types of tokens made this game the third best selling game ever[2].

Because of its popularity it is not surprising that this game has been addressed in research during the last years. The first example for a modification of Tetris is a project called TetraTetris [2]. This game is a multi player Tetris game, likewise TUMtris. It was developed for the DiamondTouch table [4].

Compared to the original Tetris game, the goal of TetraTetris is not to build solid lines but to create symmetric shapes using the original Tetris tokens. Therefore the tokens appear in the middle of the game board, which is divided into two halves (one for each player). One of the players has to grab the appearing token and can then move and rotate it using his fingers. To create bigger shapes the tokens have to be pasted together. This is done by simply moving the tokens close to each other. Once a threshold is undercut the tokens stick together automatically. To remove the shape it has to be moved in one of the corners of

[1] http://en.wikipedia.org/wiki/Tetris
[2] http://en.wikipedia.org/wiki/List_of_best-selling_video_game_franchises

the game board. Thereby the player gets points according to the symmetry of the removed shape.

The second game to be introduced here is a game called T-Touch [17]. The mechanics of this game are more or less the same as the original ones. However, in T-Touch two players are using the same device. Therefore two game boards are available on one screen. They are placed in the way that the two opponents are face-to-face to each other. The only change in the game mechanics is that the Tetris tokens do not appear automatically, but the players have to grab them and put them onto their game board.

The third example is not a video game but a board game and it is called Tetris Tower 3D. In contrast to the other games explained beforehand, this game is not the topic of scientific research, but is an actual game, published by Radica[3] in 2003. The main idea behind this game is to bring the Tetris experience to a board game. Thus it is a mixture of the famous 'Connect Four'[4] game and Tetris. In this game the opponents have to drop a given Tetris token from the top into a vertical grid. Thereby they have to try – similar to the original Tetris – to build a complete line.

2.2. Games with a tangible user interface

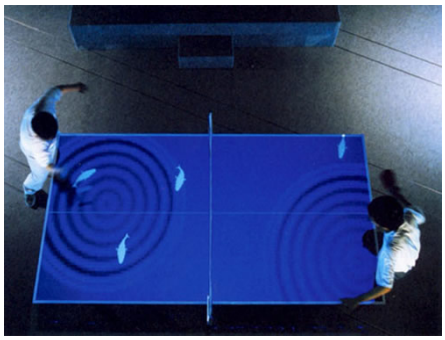
As mentioned in chapter 1, games with Tangible User Interface (TUI) have already been investigated for several years and, hence, several publications on this topic occurred. Before giving information about tangible games which work quite similar to TUMtris, a project is described providing the basis of all today's tangible games.

2.2.1. PingPongPlus

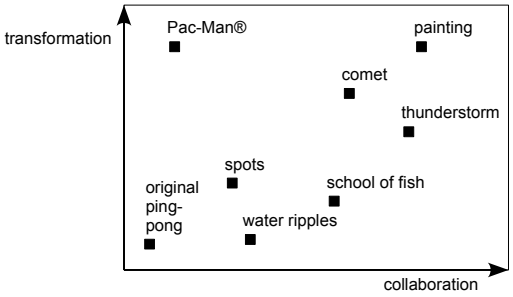
This game is called PingPongPlus (PPP) and was developed by the Tangible Media Group of the Massachusetts Institute of Technology in 1998 [9].

PingPongPlus was the first project ever that combined a tangible interface with a game. The main goal of PPP was to evaluate in which way it is useful to change the game mechanics of a real game by using a computer. Since the computational power of that time was insufficient, the Ping Pong ball and the racked were used as the tangible interface. Therefore four microphones were mounted underneath each table side to detect the position where the ball hits the table. Using the microphones the computing of the position was possible using a 300 MHz processor. In order to change the gameplay a visualization of the game was applied. Therefore a projector was installed above the Ping Pong table and the table itself was used as a canvas. Due to the different kinds of visualizations the developer changed the game mechanics of the original Ping Pong into two different manners. The first variation is to make the game a more cooperative one but not to change the goal of the game. The second was to create a completely new game. In particular seven different modifications were implemented. This modifications can be set into a diagram where the x-axis shows the change into a cooperative play and the y-axis shows the change of game mechanics (figure 2.1(b)). To evaluate the project and find the most popular game

[3] http://en.wikipedia.org/wiki/Radica_Games
[4] http://en.wikipedia.org/wiki/Connect_Four



(a) The 'school of fish' mode of PingPong-Plus



(b) A graph showing the game modes of Ping-Pong-Plus and whether the actual mode transforms the game mechanics or makes the game more collaborative

Figure 2.1.: PingPongPlus [9]

modes the PPP table was set up on a conference and it was measured how often and how long each game mode was played.

The result of the evaluation was that the most popular modification was the one that did not change the game mechanics at all but made the game a cooperative one ('school of fish' shown in figure 2.1(a)). In fact the cooperative modifications were played much more often than the ones changing the game mechanics.

2.2.2. IncreTable

After the description of the first tangible game ever, some newer games evolved, which are more related to this work and are presented in the following. Those are video games using a TUI.

The first project is called IncreTable (a mixture of incredible and table) [12]. In this project a table has been developed that not only detects objects lying on top of the screen, but also has a height sensor to measure the height of the objects lying on top of the table. The game itself is a kind of domino game, where the user has to build a chain of domino tiles standing on top of the game board. Then the first tile is toppled and if the players did nothing wrong the complete chain would fall. The special thing is that the tiles are not only real tiles but also virtual ones. Therefore special interfaces were built called 'portals'. These interfaces are some kind of tunnels that are put above the beginning or the end of the chain made out of real tiles. Once the last tile of the chain falls, the interface token recognizes this using an infrared (IR) photo interrupter and the virtual tile standing in front of the real one gets toppled. If the other way around the last virtual tile of a chain falls the first real tile is toppled by the interface token using a rotating arm. These portals are connected with the table using Bluetooth.

As mentioned before the IncreTable can not only recognize objects lying on top of the table (in this case the domino tiles) but also measure the height of the object. To make use of this features some kind of pyramids are provided that can be put on the table. At the position of those pyramids, a mountain appears in the virtual environment. Since

there are also other virtual objects like a ball, these can be used in the game by creating mountains to topple the tiles virtually. Figure 2.2 shows one scene of the IncreTable. A video demonstrating the functionality of the IncreTable is available on youtube[5].

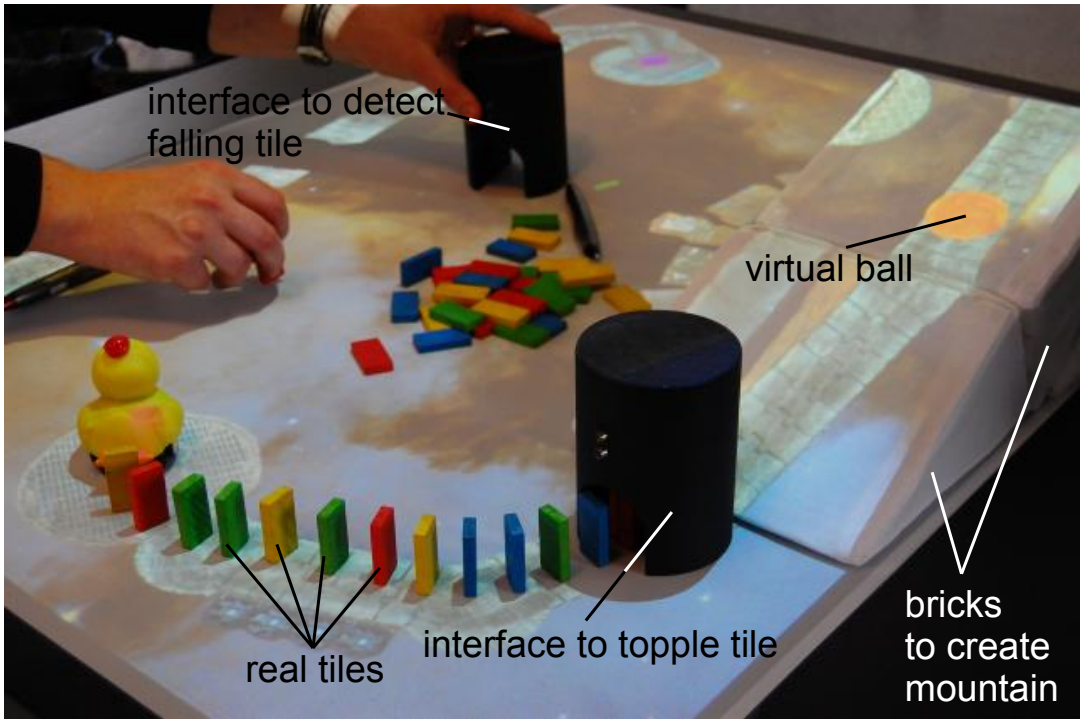


Figure 2.2.: A picture of the IncreTable [12] with an overview of its functionality

2.2.3. Towards the Next Generation of Tabletop Gaming Experiences

The next two projects to be described in this section are more similar to board games which are ported to a tabletop although the games themselves are not available as board games.

The first project of this kind is called 'Towards the Next Generation of Tabletop Gaming Experiences' [13] and was developed by the Frauhnhofer Institut für Integrierte Pu"-bli"-ka"-tions- und In"-for"-ma"-tions"-systeme (IPSI). This project investigates how games played on tabletop devices can record personal information – for instance the own cards in a card game. This is an interesting issue in table top games, since the possibilities of a game grow when each of the players can hold some secret information. The Frauenhofer IPSI solved this problem by including personal digital assistances (PDAs) into the game hardware. In that way it is possible for every player to hide some information. In order to demonstrate how this development can improve game experience, two different games were implemented.

[5]http://www.youtube.com/watch?v=eyON6SJpVus

Beside the mentioned PDAs the hardware of this system is composed of a tabletop display with TUI, a wall display and speakers. Both games implemented in this project are board/video game hybrids, i.e. the tokens are tangible whereas the game board is a virtual one displayed on the table.

The first game developed is an adapted version of Monopoly[6]. In this adaption, the game mechanics have been changed to a very little extend. However, the opportunity of using a computer screen in order to change the game board has been used. Thus the text on the game board rotates towards the player who is at the turn. Besides, the cards in Monopoly have been replaced by text boxes that display the content of the cards. Furthermore the money in Monopoly has been replaced by virtual money. While the tabletop screen displays the game board, the second screen is used to show the current money of each player.

The second game introduced in this paper is KnightMage. In contrast to the Monopoly game, KnightMages is not a ported board game, but a new development. However, it is still rather a board than a video game. In this game, the player acts as a knight in a dungeon. The dungeon is filled with monsters and in order to survive, the players have to cooperate with each other. Since there are also 'treasures' in the dungeon, the players have to compete in order to maximize their credits. This game takes much more advantage of the opportunities using a virtual game board. For instance some parts of the game board are invisible at the beginning and can only be seen once a door is opened.

2.2.4. Weathergods

The last project to be described in this theses is Weathergods [1]. With Weathergods, developed by the Eindhoven University of Technology and the Philips Research Laboratories Eindhoven, a board/video game hybrid has been created, similar to the games developed by the Fraunhofer IPSI. However, in Weathergods not only the game board takes usage of the fact that a computer is used to handle the game, which results in a dynamic game board, but also the game mechanics. Thus the game is quite complex.

In the story of Weathergods a huge dry happens in an Arabic savanna because the Weathergods are fighting. To calm the gods, the player has to collect offerings for the gods. Therefore a tangible camel token has to be moved over the virtual game board. Besides, gold can be found underneath some fields which can then be used to buy special offers. By moving the camels over a green field (area with grass) the camels produce milk that can be sold to gain more money.

The goal of the research group is to evaluate whether an iconic or a symbolic representation of the game tokens is preferred by the players. Iconic tokens are tokens whose shape gives some information about the usage of the token itself (i.e. a token representing a camel is formed as a camel). Symbolic tokens do not give any information. To reach that goal two different sets of game tokens were created as can be seen in figure 2.3. Unfortunately, there is no result available because the evaluation never took place.

The important thing about games with a TUI is that all described games are games with a rather slow game speed. The requirements on the detection do not include fast motion detection. In TUMtris, however, such a fast detection is indispensable.

[6] http://en.wikipedia.org/wiki/Monopoly

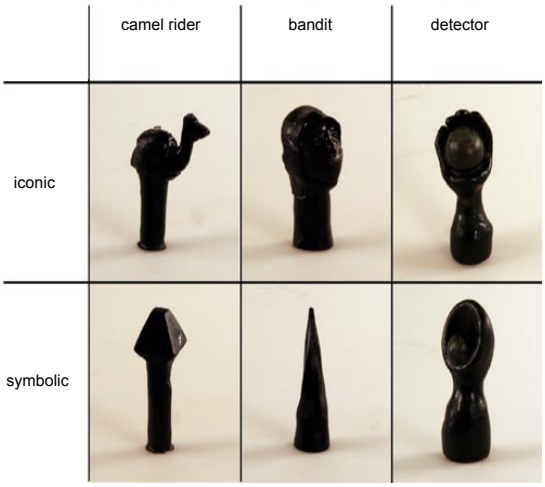


Figure 2.3.: A picture of the different types of game tokens used in Weathergods[1]

3. Concept

To begin with the development part of this project the general approach is described. Since the TUMtris project is mainly a game, some methods of game development are used In this thesis the steps for game development as described in [16] are followed.

3.1. General approach in game development

According to [16] the development of a game is divided into four phases, which are shortly described in this chapter.

First, methods of idea finding are applied. According to the authors it is useful to be inspired by the environment of the developer to find good ideas what the game could be about. Those ideas are written down and structured. Once the ideas are put together in groups of the same topics (for instance sports) a so-called idea tree can be created. This tree starts with a general topic and gets more and more precise and diversified.

After the idea finding process, it is important to bring these ideas into a game. Therefore in the book several question are addressed to the developer. Those questions can be about the number of players or the platform the game should be played on. During this process the main structure of the future game is created.

Once the main boundaries of the game are set, the developer has to start prototyping the game. This prototyping again is divided into two different parts: the software- and the hardware prototype. Though it might seem that a hardware prototype is not necessary for a video game, it can show some features before writing a single code line and sometimes show some impossibilities in the game mechanics. One important thing to be mentioned at this point is that prototyping is not about the look of the game. The aim is to find the components needed for the game and if the game mechanics can work at all.

Apart from the hardware prototype the software prototype is of major importance in the development of a video game. Like the hardware prototype it only includes the main functions of the game to prove if the game mechanics can work at all. It has to be kept in mind that only the core functions are implemented in this step. Other features like a menu are implemented at a later stage. It is quite useful to implement the software prototype in an object oriented programming language, as these offer the possibility to simply replace some code parts. One important thing in prototyping is to isolate the core game mechanics and implement and test them separately. Once all the elements are tested the software prototype can be put together.

After the prototype has been finished it has to be tested. Therefore a play test has to be performed. There are three different kinds of play tests, each of them with different goals. The first one is the so-called self test: the developer tests the game immediately after finishing the prototype or during its development. Since the developer naturally knows the game mechanics this test is only useful to test some fundamental concepts.

If the game is playable it is necessary to have it tested by others than the developer. The first testers should be some friends or colleagues. Those testers can propose different perspectives by looking at the game and may show up some issues the developer himself could not find. It is useful to choose people you know for the test, because it is more comfortable to show the prototype to someone you know and make sure it is playable before testing it with unknown testers. Naturally, the results of this test are not very significant because the test persons will not judge the game objectively.

Therefore it is necessary to test the game with unknown persons, which is the third test type. Its goal is to find out how people not knowing anything about the game can handle it. Hence, some questions which are being asked before, during and after the test have to be written down in the run up to the test. After the test the answers which were given have to be analyzed in order to improve the play experience.

After the play test the game has to be refined and completed. Since the tested program has been a prototype application, naturally there are some features missing. These features have to be included. Furthermore the results of the play test have to be integrated. After the first round of refinement another play test has to be performed to analyze the balance once more. Afterwards the results of this test have to be integrated as well. This testing-integrating cycle (figure 3.1) continues until the game launches.

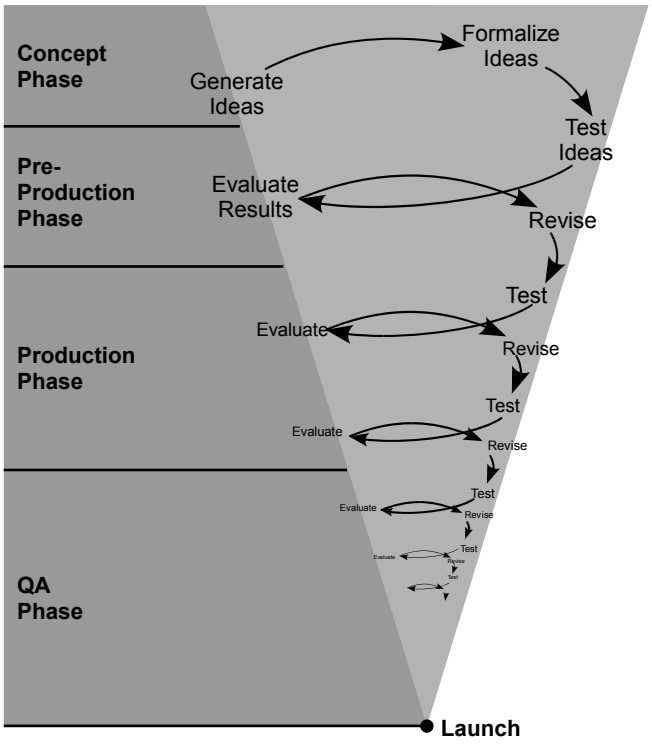


Figure 3.1.: The iterative game design [16]

3.2. Concept of TUMtris

In the following it is shown, how the general principles of game development were modified to suit the TUMtris project.

As in this project the game does not have to be developed from the scratch, some parts of the general approach are not necessary in its development. In particular this mostly affects the first part of game development. Because an already existing game is chosen to be modified, there is no need for any brainstorming to get an idea what the future game is about. However, the step of defining certain properties like the number of players and the control type of the game have been done by setting the requirements of the project. Since most of the first phase of game development has been done before the actual start of the project, the results of that phase are given as requirements (see chapter 4).

Furthermore the iterations of testing, evaluating and revising are only done once. All the problems that occurred and need to be changed are therefore described in chapter 7. Hence, the main focus lies on the prototyping part of TUMtris and on the first play test. Therefore the implementation of the game is divided into parts which can be tested separately.

Another change in the approach is that the created software and hardware is not a prototype by the definition given in section 3.1 but already shows the final look and the final functionality.

Due to these modifications the development process of TUMtris can be shortened tre"-mendously and, hence, the project can be completed in the given time period of four months by only one person. Figure 3.2 shows which part(s) of the game development is covered by the TUMtris project.

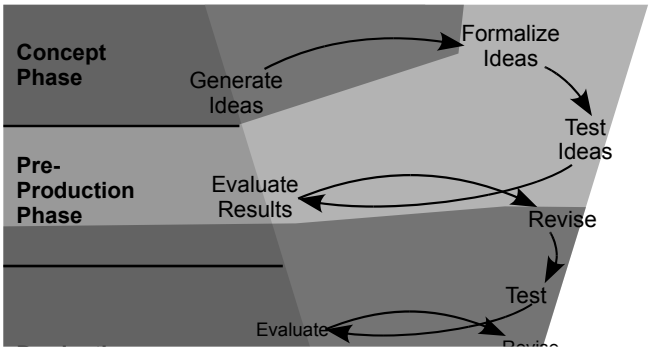


Figure 3.2.: The development of TUMtris. Only the highlighted part(s) is(are) covered

4. Requirements

As mentioned in chapter 3 the software requirements given by the problem definition do also cover most of the requirements given by the first phase in game development. To stick to the conceptual order of game development, in this chapter the software requirements of TUMtris shall therefore be shown first.

4.1. Software Requirements

As a starting point the project was announced as a tangible Tetris game that offers a multiplayer game mode. Hence, the first requirements are a tangible control as well as a multiplayer mode and the general game mechanics are given, since the game has to be a Tetris game. Accordingly the main goal of TUMtris is to build up lines with bricks, by moving and rotating Tetris tokens consisting of four bricks each. The seven different types of tokens are also defined by these requirements. These types differ only in shape and are L-Token, Z-Token, T-Token, Bar-Token, Square-Token, Inverse-Z-Token, and Inverse-L-Token (see figure 4.1).

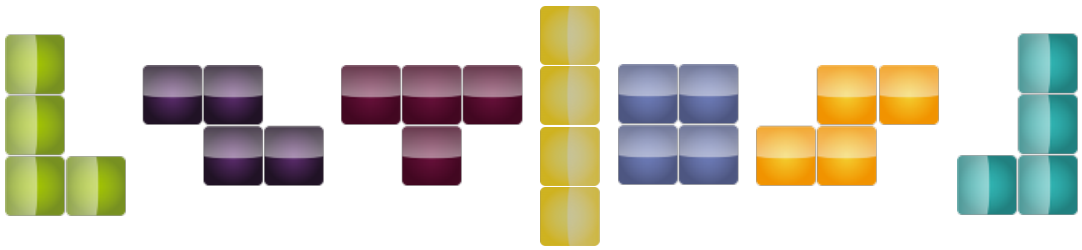


Figure 4.1.: The Tetris tokens from left to right: L-Token, Z-Token, T-Token, Bar-Token, Square-Token, Inverse-Z-Token, Inverse-L-Token

TUMtris is a tangible game, this means that the players have physical tokens to control the virtual tokens by taking one of the seven different kinds of token and place it on the corresponding virtual token displayed on the screen. If the type of the physical token is the same as the type of the virtual one, the virtual token can be moved. If it is the wrong type, no virtual token movement takes place.

For scientific reasons it is required to add a touch control to the tangible control, in order to evaluate whether the tangible control is useful and suitable in a video game. To make the touch control intuitive the player is enabled to move and rotate the token with well-known gestures. Therefore the token is moved by pressing on it and then moving the finger. To rotate the token two or more fingers are used.

With these first two requirements the game itself gets quite precise, since they already define major parts of the game. However, the number of players and the game mechanics still have to be defined.

According to the first announcement the game has to be a multiplayer game. Therefore it has to provide equipment for at least two players. Moreover these players should not play against each other but in cooperation. Thus the game mechanics itself has to be modified. In the original Tetris game the player has to build up a line with tokens appearing on the top of the board falling down to the bottom. But since the cooperative players should play face to face, the concept of building lines is insufficient. Therefore in TUMtris the players have to build up a square made of Tetris tokens appearing in the middle of a quadratic board. These tokens do not fall to the bottom only, but also to one of the three other sides (to the left, the right or the top). Thereby it can be assured that the players can work on the same board and have enough space to operate.

Again, there has to be a second game mode for comparison to evaluate the tangible control correctly. It might be the case that the new game mechanics do not work at all; consequently the game is not enjoyable. In that situation a proper determination, whether it is the game mechanics that does not work or the controls fails.

To that end, a traditional Tetris has to be realized as well; here the players have to build up lines according to the original Tetris concept. To enable a multiplayer mode in this game, a 'versus mode' is implemented. In this mode the players also play on the same screen, but on different boards.

The last requirement given for TUMtris is provided by the fact that the project has to be a game. Therefore it has to offer several properties that are expected from a game. These properties are an appealing visualization, including a smooth movement, an attractive looking texture, and an appropriate sound. Furthermore a suitable menu to select the game modes, quit the game and adjust the settings is expected.

4.2. Hardware requirements

Resulting from the software requirements, there are several hardware requirements in this project. These requirements can be divided into the requirements on the computer hardware and requirements on the tokens.

4.2.1. The input device

First, the computer hardware requirements have to be addressed. Since the screen of the computer is used as the board of TUMtris, the used device has to offer a tabletop. Since most of the devices providing a tangible user interface are in fact tabletops, this requirement can be met easily.

Thus it is more important for the hardware to offer both a touch and shadow detection. The Tangible Interactive Surface for Collaboration between Humans (TISCH) device developed by Florian Echtler [5] provides these features, which can be used independently from each other. A camera mounted underneath the tabletop keeps track on the table top by a mirror and recognizes any IR light appearing on the surface. For the implementa-

tion the routines of the libTISCH library [5] is used which is explained in further detail in chapter 5.

The IR light makes it possible to track any user input for interaction. To detect a touch interaction a method based on the frustrated total internal reflection (FTIR) developed by Jeff Han is used [7]. Thereby the light is created by infrared LEDs mounted at the side of the milk glass tabletop. Because of the reflections inside the tabletop the light is invisible for the camera until a finger touches the tabletop. The light gets reflected by the finger and an IR blob appears on the surface. Hence, the table provides the touch interaction.

To detect objects lying on top of the tabletop, the surface is illuminated by infrared illuminants mounted underneath the tabletop. This light does not get reflected by the milk glass, but by objects lying on the table. Thus objects on the table appear for the camera as infrared blobs, too.

To determine the type of the blob detected, the infrared illuminants for the shadow detection and the ones for the touch detection are alternating switched on and off every second frame of the camera. Thus the camera interprets all the blobs detected at even frames as touch interactions and all the blobs at the odd frames as tangible interactions.

A computer screen on the milk glass is projected via a projector. This projection can be seen from the top of the table.

Figure 4.2 sketches the build-up of the table to illustrating the functionality of the TISCH device.

Although the TISCH device offers tangible interaction in theory, its practical implementation proves otherwise: As mentioned above, LEDs are used as illuminants. While this light source is sufficient for the touch detection, it is less appropriate for the shadow detection: the illumination of the table top has to be quite homogeneous, in order to enable the camera to detect the objects lying on top of the table while ignoring objects located above the table (such as the hand of the user). Illuminating a rectangle surface homogeneously by using LED spots is challenging, especially since a dimming for the LEDs' light intensity is not possible.

For the illumination 10 LED spots with 30 LEDs each are used, which have to be adjusted individually. To set the spots in the correct position, the debug window of the libTISCH is used. Therefore a sheet of paper is put on top of the table to reflect the infrared light. Afterwards the spots get adjusted individually to illuminate the surface as uniformly as possible. Since there are still areas on the surface which are brighter than the rest of the table, the spots causing those areas are dimmed physically by pasting tape on some of the LEDs. The final configuration of the TISCH device can be seen in figure 4.3.

Once the tabletop is illuminated homogeneously the overall brightness of the LEDs has to be set. As mentioned there is no dimmer for LEDs available. That is why the adjustment has to be done by software. The library used by the TISCH device offers a calibration file, where the threshold for the shadow detection can be set. After adjusting this value the table can be used for tangible interaction.

4.2.2. The tokens

Besides the multi touch table the game tokens are the second hardware to be considered during the development of TUMtris. Designing the game tokens is rather challenging since there are three different aspects that have to be taken into account.

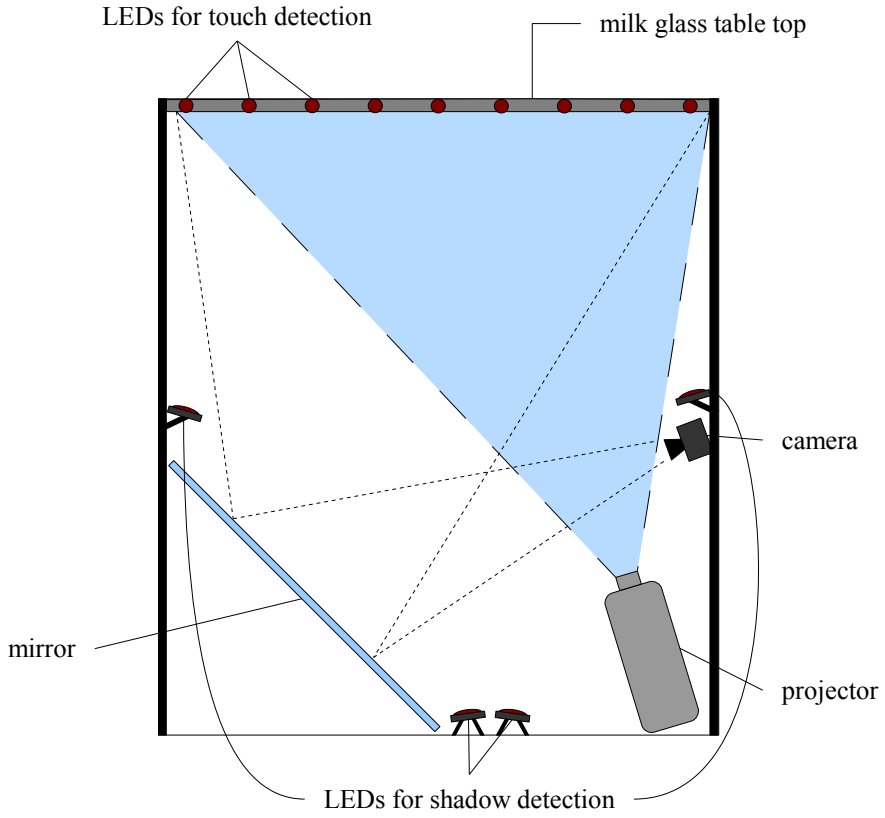


Figure 4.2.: A sketch of the TISCH device, showing its functionality

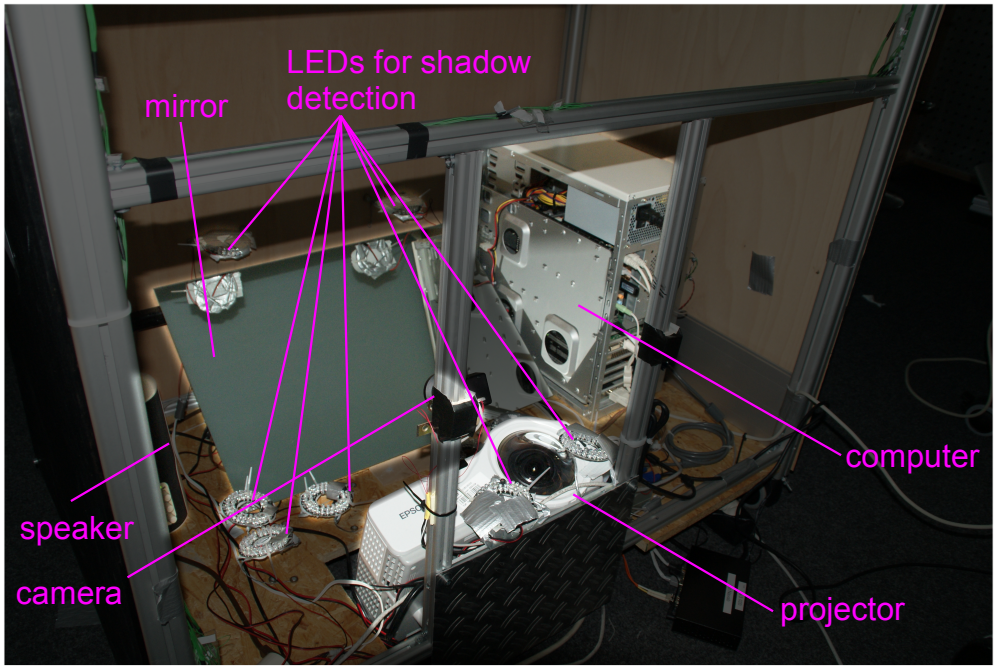


Figure 4.3.: A picture of the TISCH device with marks on all important features

The first aspect is, again, the fact that the project is a game. Therefore the tokens have to look appropriate and furthermore the design has to fit into the design of the program. Moreover the tokens have to be handy but not too small. After all, it has to be easy to play with them.

The second requirement is of technical nature and provided by the fact that the tokens have to be able to be detected by the used detection device, in this case by the TISCH device. Since the libTISCH abstracts any shape detected on the TISCH into an elliptical shape, the TISCH device cannot distinguish between any shapes. Thus there has to be another differentiating factor between the different tokens. To that end each token does not only have to produce one single shadow, but multiple ones. Moreover, the shadows produced by each token have to have a minimum size. This is because the TISCH device cannot detect objects which produce a too small shadow.

The third requirement for the game tokens is that they have to be cheap in price and easy to be manufactured. This is a quite practical requirement, since the tokens are handmade.

The first two requirements for the tokens already define how the tokens may look like. For the first approach the tokens are designed by using acrylic glass cubes. Four of those cubes are pasted together to create one Tetris token. Since the acrylic glass cubes do not produce any shadow at all, a small piece of paper is stuck on the bottom of each cube.

Although this draft of the game tokens complies with the requirements given by the design, it poses a disadvantage: the maximum edge length of the cubes which being monetarily feasible is three centimeters. Any larger edge sizes would exceed the economic

limitation of the project for the 28 cubes needed for one set of game tokens. ??? Da sitzt man schon eine Zeit an so einem Satz ;-)???

Thus the only possibility is to use smaller cubes which are less expensive. The problem caused by the usage of such small cubes for the game tokens is that the produced shadow is too small to be detected by the TISCH device.

Given this limitation, another design approach for the token was developed: Instead of taking transparent objects (acrylic glass cubes) and pasting shadow producing objects (pieces of paper) on the bottom, shadow producing objects are used and fit together with a transparent object in order to have four separate shadows appearing per token.

As shadow producing objects in this project, cubes made out of woods are used: wooden cubes are cheap in costs and can be painted in any color suiting any design and hence look classy.

To connect the cubes with each other, a profile made out of 6mm thick acrylic glass is used. The profiles for one set of game tokens are cut out of one layer of acrylic glass using a laser cutter.

The advantages of this design are that it is quite reasonable (costs apply only for the layer of acrylic glass and the color used) and the size can be chosen as needed.

The disadvantage of this design is that it takes some time to create all the tokens needed for one set.

To build the game tokens we use wooden cubes with an edge length of 3 centimeters. This size is necessary for the cubes to be detected by the TISCH device. Each brick of the tokens is required to have an edge length of 5 centimeters at the end. Knowing these key data a mold is drafted in Adobe Illustator[1] for each type of token. The resulting file is used to cut the shapes out of a piece of acrylic glass. Afterwards the bottom side of each profile has to be cleaned because the laser burns-in some charred acrylic glass. Therefore the bottom sides first have to be grounded down and then are polished to become transparent again.

In order to create the cubes a log of wood is used. This log is cut into cubes. Afterwards the cubes are painted multiple times to color them appropriately.

At the end the cubes have to be pasted into the shapes. The resulting game tokens are shown in figure 4.4.

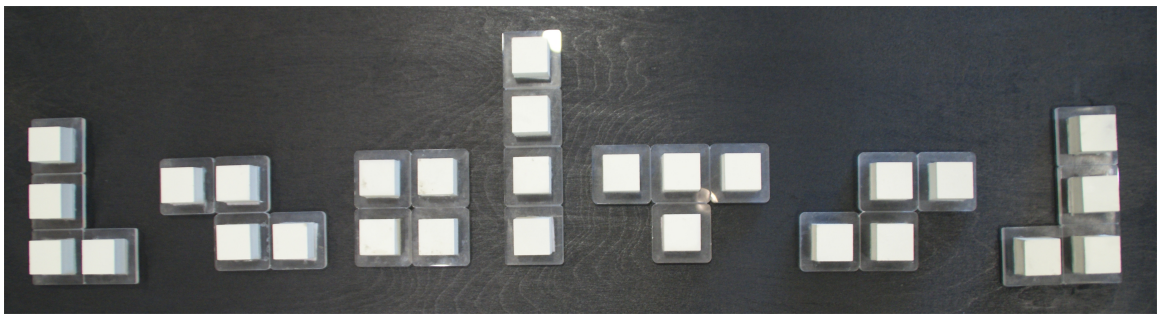


Figure 4.4.: Physical tokens used in TUMtris

[1] http://www.adobe.com/de/products/illustrator/

5. Implementation

As mentioned in chapter 4, the hardware used in this project is the TISCH device developed by Florian Echtler. The library libTISCH is used as the main library since it was developed on the TISCH device. The libTISCH is a library for the development of software using multi touch and tangible user inputs. As the libTISCH is divided into several layers, whereas the lowest is the abstraction of the hardware, it can not only be used on the TISCH device but also on several other platforms. Thus the TUMtris game shall be portable to other platforms as well.

Before explaining the software part of TUMtris in detail, some functionalities of the libTISCH – especially of the Widget Layer – are described first, since the widgets and gestures provided by the libTISCH are an important tool for the development of the game.

5.1. Usage of the libTISCH

As mentioned before the libTISCH is divided into different layers. The only layer the software developer works with is the Widget Layer. The interface used for interactions between the libTISCH – and consequently the hardware – and the application consists of Widgets. A Widget is an object to which several gestures can be added. Once a gesture is matched in the region (the area the widget covers on the screen), the action method of the widget is called.

A Gesture is a set of predefined Features such as count, move, rotate, etc. Each Feature can be set to operate either with touch or shadow. Once all Features of a Gesture match, the action method of the associated Widget is called. In this manner the Widgets handle all of the user inputs. A Gesture does not have to be a complex sequence of movements but can also be a simple tap or release. To toggle the Features and in the last consequence the Widgets, all detected objects (either a finger-press or a shadow) are abstracted into elliptical shapes which are then stored.

Using the Gestures it is possible to create specific Gestures and Widgets quite easily. In addition to the possibility of creating self constructed Gestures, the libTISCH includes several predefined gestures and widgets. Those Widgets are, for instance, Buttons, Textfields and Tiles (Widgets that can be moved, rotated, and scaled). The Tiles inherit from the Buttons, so that they also can be pressed and released. The most important specialized Widget for this implementation, however, is the Container Widget. A Container is a Widget inheriting from Tile, so it can be moved, rotated and scaled as well. Besides, it also can hold other Widgets, see figure 5.1. Once a Container is moved, rotated or scaled all associated Widgets move as well. In order to work with the libTISCH, it is advisable to use as many predefined Gestures and Features as possible, however in some cases it is necessary to create own Gestures.

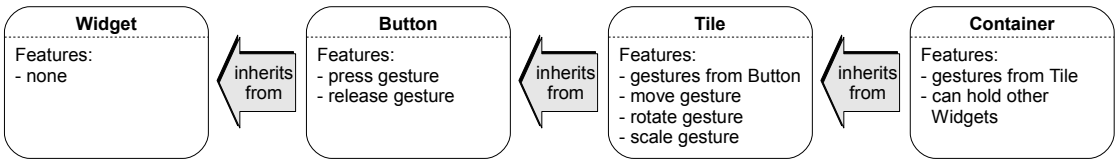


Figure 5.1.: The inheriting order of the libTISCH Widgets

The second tool of the libTISCH that is used in this project is the Window class, provided by the tool library of the libTISCH. The Window class creates a window in a given size with an openGL environment. Since the libTISCH library needs an openGL environment to operate, this is very useful. Furthermore the Window class is also a special kind of Container which cannot be rotated or pressed, but is resized by resizing the window on the desktop. Thus widgets simply can be added to a Window object.

5.2. TUMtris game implementation

The implementation process of TUMtris is divided into four parts that are developed independently. To redeem the requirement given by the approach of game development described in chapter 3, each of the parts is implemented and tested afterwards; first by the developer and then by their friends.

The first part is the control mechanics of TUMtris. It realizes the communication between TUMtris and the libTISCH. Therefore an abstract Token class is created which inherits from the Container class of libTISCH.

The second part of the game is the collision handling. The handling consists of two phases: the detection of collisions and the collision avoidance.

The third part is the implementation of the game mechanics. This part handles the behavior of the virtual tokens when they are not controlled by the user. To manage the mechanics a virtual game board is created, which holds all the virtual tokens in the game.

The fourth and last part includes all the rest needed for a game – such as a graphical user interface to set up the game or the sounds played in TUMtris.

5.2.1. The control part

Generally TUMtris offers two different kinds of control types: the tangible control and the touch control. Since both types work fundamentally different, they were implemented separately.

Tangible control

In the tangible mode the control part has to offer three different functions. The first function is to detect a physical token lying on the table top above one of the virtual tokens. The second one is to check if the given physical token matches the virtual one and the third function is to move and rotate the virtual token into the position of the physical token.

As mentioned above, in the libTISCH each detected object is reduced to an ellipse represented by a midpoint and a semi-major and semi-minor axis. Thus it is impossible to recognize the shape of an object lying on the table. To distinguish between the various tokens used in TUMtris, the tokens are not detected as one single object, but each single brick of a token is detected itself. Hence, the tokens can be recognized and, furthermore, it can be checked, whether the applied token corresponds to the virtual token shown on the screen.

Four widgets have to be used to detect one token (one widget for each of the physical bricks). Therefore a class called BrickWidget is created, which inherits from the Widget class of the libTISCH. All four Widgets are held by a Token class. Since the Token class itself is transparent, the Widgets holding the texture of the virtual token are placed according to the type of token. To make the virtual tokens good looking the BrickWdgets do not touch each other, but there is a small gap between them (figure 5.2).

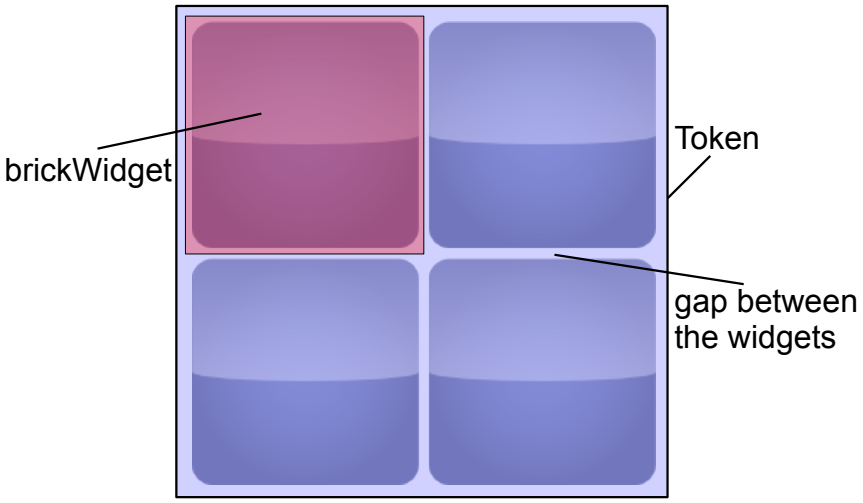


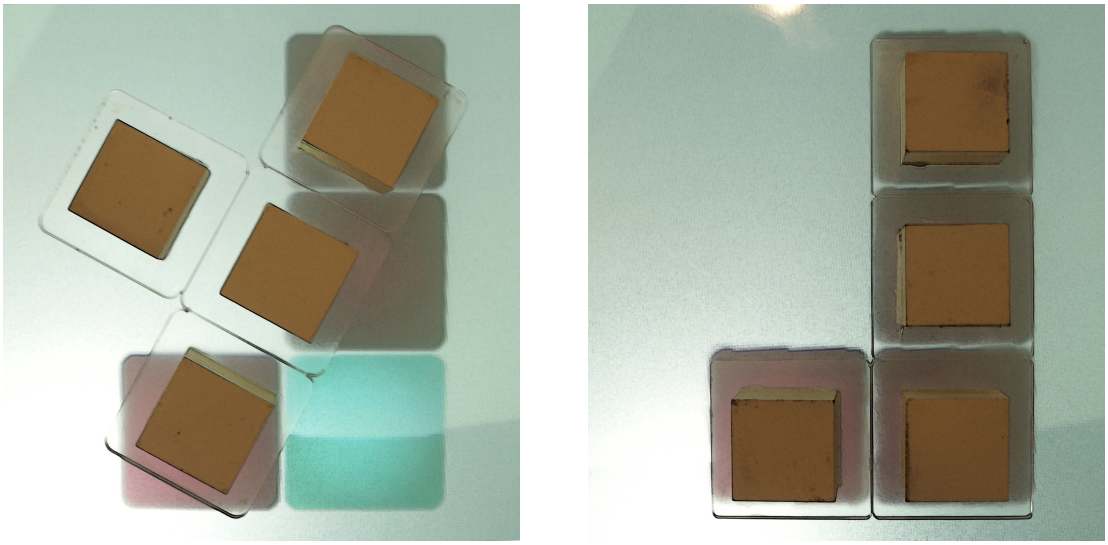
Figure 5.2.: A Square-Token and its components as an example for a virtual token

The BrickWidget class holds three different Gestures. One Gesture, the lockBrick-Ges"-ture, is used to detect whether an object lies above the widget. It works similar to the tap Gesture of the libTISCH but with shadow detection instead of touch detection. The second Gesture, the releaseBrick-Gesture, is used to detect if an object lying above the widget is taken away. This is quite similar to the release Gesture of the libTISCH, but again works with shadow detection. The third Gesture is the moveBrick-Gesture, which recognizes any movement of the object lying above the widget. It holds the new midpoint of the ellipse which indicated that the object was moved.

The Token class is abstract and represents the Tetris tokens. To specify the different kinds of Tetris tokens such as the Bar-, the Square- or the L-token, seven classes are created, which inherit from the Token class and implement the abstract methods (figure 4.1). In each of the subclasses the BrickWidgets are arranged in a different manner to generate unique shapes for each of the tokens.

Once the lockBrick-Gesture of one of the four BrickWidgets is toggled, a Boolean variable in an array, indicating which bricks of the virtual token is locked, is set to true.

Once all the variables in this field are set to true, the virtual token is locked and can then be moved. Thus it is possible to recognize the physical token and if it corresponds to the virtual token. Figure 5.2.1 illustrates how the correct token can be detected. The red square indicates that the lockBrick Gesture of the according BrickWidget has been toggled, the blue square indicates the contrary.



(a) A wrong physical token is used and thus the virtual token is not locked

(b) The correct physical token is used and thus the virtual token is locked

Figure 5.3.: Token detection of TUMtris

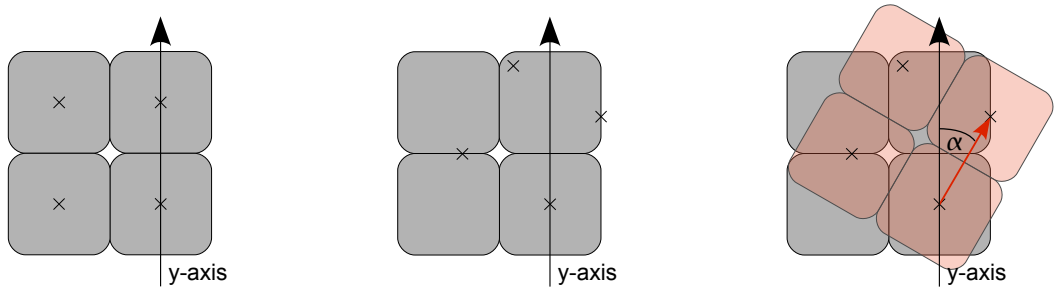
Once the releaseBrick Gesture is toggled, the associated variable in the lock array is set to false. If at least two of the variables of this array are false, the Tetris token is unlocked and cannot be moved anymore.

The third requirement of the tangible control is to detect any movement of the token and compute its new position. This is divided into two operations. The first one computes the new angle of the token related to its ground position in order to rotate the token in the right angle. The second one computes the new midpoint of the token which is necessary for moving it to the correct position. It has to be kept in mind that both the angle and the midpoint are not computed relatively to the previous one, but absolutely. This is necessary because the virtual tokens always have to lie underneath the physical ones. By computing the changes in the relative angle and position, the computational errors sum up quite fast causing an imprecise positioning. For the computation of the new angel and position not all the new positions of the physical bricks are needed. Thus it is possible to move and rotate the virtual tokens even though one token's bricks are accidentally not detected.

To compute the new angle of a Tetris token, the changed midpoints which were obtained by the moveBrick-Gesture of two BrickWidgets originally lying on top of each other get connected and normalized. By computing the dot-product of this vector and the y-axis

$((0,1)^{\mathrm{T}})$, and taking the arccosine, the new angle is calculated, see figure 5.2.1. It can be shown that this is the needed angle just by using the definition of the dot product:

$$\vec{a} \cdot \vec{b} = |\vec{a}| \cdot |\vec{b}| \cdot \cos\left(\angle\left(\vec{a}, \vec{b}\right)\right)$$
$$\rightarrow \angle\left(\vec{a}, \vec{b}\right) = |\vec{a}| \cdot |\vec{b}| \cdot \cos^{-1}\left(\vec{a} \cdot \vec{b}\right)$$



(a) origainal position

(b) midpoints of the detected shadows

(c) computing the new angle by using the dot product

Figure 5.4.: Computation of the new angle of the virtual token

Since both vectors are normalized, their length is 'one' and the angle is simply the arccosine of the dot product.

The computation of the new midpoint works in a similar way. At first two changed midpoints, obtained by the moveBrick-Gesture, are connected. Afterwards this vector is multiplied by 0.5 and added to the first of the two coordinates.

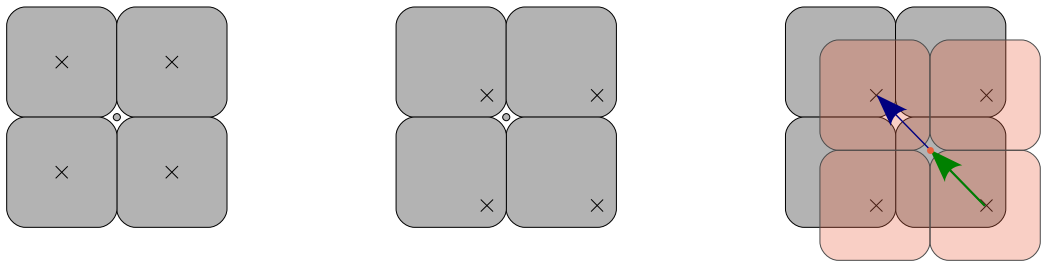
Since the widgets that have to be used to compute the new midpoint and angle are different for each type of Tetris token, these methods are implemented individually in every subclass. In figure 5.2.1, for instance, the first and fourth coordinate are connected to compute the midpoint, and the first and second coordinate are used to compute the new angle.

The method of using a single widget for each of the blobs produced by the physical token comprises two disadvantages:

The first one is that each BrickWidget only can detect movements appearing in its region (the area on the screen covered by the Widget). There is a flag that can be set to provide detection outside of its borders. However due to a flaw in the implementation this flag does not work properly. Thus the bricks can only be moved very slowly.

The second disadvantage is that the methods for computing the new angle and midpoint should only be called after all the locked BrickWidgets have saved its new midpoint. Hence, another Boolean variable is held by the virtual token for each BrickWidget that signals if the according BrickWidget has moved or not. Only if the moved Boolean is the same as the locked Boolean and if the virtual token is locked, it will move.

However, since it is impossible to detect all four blobs with one widget and then assign the detected blobs to the bricks of the Tetris token, these disadvantages have to be accepted.



(a) Original position

(b) Midpoints of the detected shadows

(c) Computing the new midpoint by using two shadows

Figure 5.5.: Computation of the new midpoint of the virtual token

Touch control

Besides the tangible control TUMtris also provides a touch control. The implementation of the touch control, however, is significantly easier than it is for the tangible control. This is because the positioning of the Tetris token by using touch control does not have to be as accurate as it has to be by using the tangible control. Thus the new angle and midpoint can simply be computed by relying on the relative moving vector and the relative angle. Therefore the Gestures that are already provided by the Container widgets can be used. These Gestures are tap, release, move, rotate and scale.

In this mode the Gestures of the BrickWidget are not used at all.

5.2.2. The game mechanics

Although the game mechanics of Tetris itself is quite simple, there is enough complexity to make it necessary using a dedicated class to perform them. This is done by the virtual game board, a class that inherits from the libTISCH Container class and consequently is able to hold other Widgets and Containers. The dominating data structure of this class is a two dimensional array representing the game board of TUMtris. Beside the virtual tokens the game board class also holds the bricks that have been finished and cannot be moved anymore. Since the finished bricks all lie in a grid, the usage of an array as data structure seems quite reasonable. Each field of the game board array represents one possible position of a brick in the game. Therefore the class FinishedBrick is used, which inherits from the libTISCH Widget class. The FinishedBrick class holds the texture of the brick and the polygon marking the outline. The polygon is used for the collision detection which is explained later on.

It is important to avoid multi threading because the libTisch is not thread safe. Therefore all the automatic move operations are called from the draw method which is called approximately 170 times per second.

The first function is to automatically move all the tokens that are not locked currently. But in order to move a Tetris token, it first has to be put into a position that fits into the raster of the game board. This means that it has to be turned to the nearest multiple of

90 degree and the midpoint has to be moved to the nearest grid coordinate. Thus it is assured that tokens in their final position are either next to each other or the gap's measure between them is a multiple of the brick size.

The movement into the game board raster does not happen immediately after the Tetris token is unlocked but with a delay of about one second. Hence some detection dropouts of the hardware can be compensated for.

After a virtual token is put into the raster, it is moved by the game board with a velocity of three pixels ten times per second. Hence, the movement seems to be smooth to the user. The moving direction of the token depends on both the game mode and the position of the token on the game board. In the versus mode all the tokens are moved to the bottom of the board. In the other game modes, however, the Tokens are moved into all four directions: to the left, to the right, to the bottom and to the top, see figure 5.6. Once a token hits the border of the game board or a finished brick, the bricks in the token are finished as well. Thereby the Tetris token itself is automatically removed from the game board and four finished bricks are created and set to the coordinates where the bricks of the token were before. Furthermore these finished bricks are added to the associated fields of the array representing the game board.

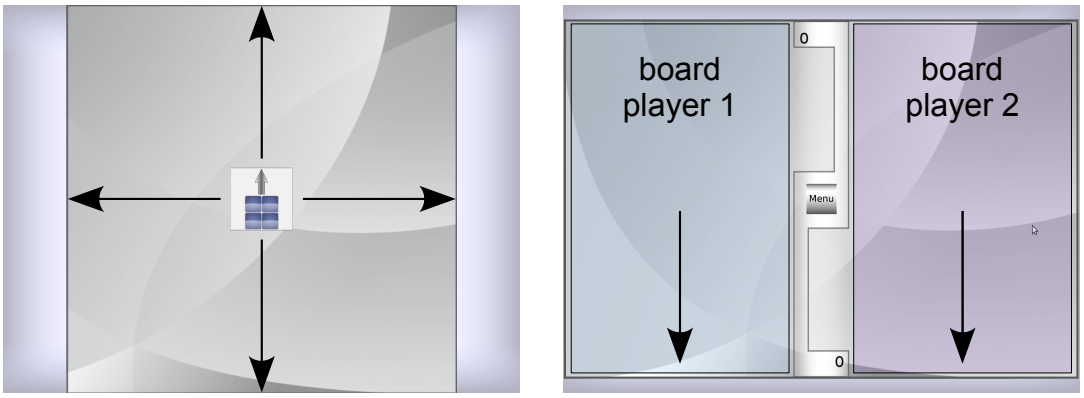


Figure 5.6.: The two different game boards; the arrows indicate the moving directions of the virtual tokens. Left: verus-game-board; right: single-/coop-game-board

After adding the bricks it has to be checked whether either a line (versus mode) or a square (coop- and single mode) has been completed. Therefore the array of FinishedBricks held by the game board is used.

In case of the versus mode the array is checked line by line if it includes any complete line. If a complete line is detected, its bricks are removed from the game board and the according fields are set to null. Subsequently all the bricks above are moved downwards for one brick size. This is done repeatedly until the complete array is checked.

In case of the single or the coop mode it has to be checked, if a square has been completed. To avoid gaps in the edges that cannot be filled by bricks, in contrary to the rest of the bricks of the square the edge bricks of a completed square are not removed. After that,

the bricks are again moved in the direction according to the place it has on the game board (figure 5.6).

Apart from the automatic movement of the unlocked tokens the game board offers several other functions. One of them is that a new token has to appear at some points in the game. In the original Tetris a new token always appears at the time instance where the old token was finished. However, since in TUMtris it is possible to prevent a token to be finished forever, a new token appears after a defined period of time. This time period depends on the difficulty level of the game and gets smaller the higher the difficulty level is set. This forces the user to make a quick decision where the tokens are to be placed.

Another function of the game board is that it holds a texture. Depending to the game mode there are two different textures available. Figure 5.6 shows both possible textures. More important for the game mechanics than the textures is the 'midpolygon' that is also held by the game board. The midpolygon is a transparent rectangle put in the middle of the game board and is used by the collision handling. In the versus mode the midpolygon is used to divide the screen into two game boards. In the other game modes (coop- and single mode) the screen does not have to be divided into two halves because the game board is quadratic and covers the complete screen. However, a midpolygon is needed to mark the outline of the menu button so that no virtual token can accidentally cover it.

5.2.3. The collision handling

At this stage, the program enables the player to move the virtual tokens, to build lines, dissolve them and that Tetris tokens, which are not locked, are moved automatically. However, since all of the Tetris tokens can be moved without any constraint on the board, a collision handling is needed to avoid that Tetris tokens can overlap or accidentally leave the screen.

To provide a fast and reliable collision handling, there are different approaches. The first one is to use an existing free library that offers the needed features. When working with the .NET framework however, it is quite hard to find such a library as almost no collision handling exists for this platform. This may be caused by the fact that .NET already includes a collision handling [14], which, however, cannot be used in this case, because it is much too complex and not available for Linux. There are some collision handling libraries implemented in C++ that theoretically can be transferred into C#, but these libraries have two disadvantages. The first one is that most of these libraries are quite big and offer much more than needed in TUMtris, such as 3D collision handling. The second and much worse disadvantage is that most of those libraries are developed for Windows only and not for Linux and therefore cannot be used on the TISCH device.

Thus an alternative has to be found. As already mentioned, the .NET framework offers a collision handling itself. Although it is quite complicated to use and, again, offers much more than needed in TUMtris, it seems to be an easy way to integrate a collision handling into TUMtris.

However, this method does not work. One reason might be that in order to use this collision handling a plug-in for the .NET framework would have to be used. Apparently that does not work on the mono framework used to emulate the .NET environment on a Linux machine.

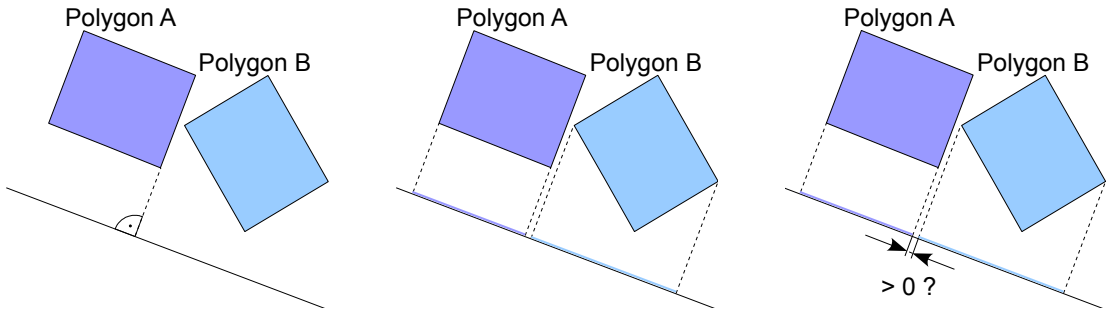
Due to the lack of existing collision handling libraries, a self implementation is necessary. Fortunately, there is a code example [3] written in C# that can be used to create one's own collision detection. This algorithm works in the two dimensional space only. But since the libTISCH does work in the two dimensional space only as well, this matches quite well.

The applied algorithm uses the separating axis theorem [18]. This theorem states that if two convex shapes have a separating axis – this is an axis where the projections of the polygons do not intersect –, the polygons themselves do not intersect either. Therefore it has to be proven that no intersection axis exists for two given polygons in order to show that they do not intersect.

For polygons the separation axis can only be a line which is orthogonal to one of the polygons' edges. Therefore the polygons are projected successively onto the possible separating axis and it is checked if the projections intersect. If no intersection occurs, the polygons do not intersect and the algorithm stops. Otherwise, it continues until all possible separating axes are checked. Hence, in order to implement the algorithm three different steps have to be performed.

- Take one edge of one polygon and rotate it by 90 degree
- project both polygons onto the resulting line
- check, if an intersection occurs; if it does not the polygons do not intersect (break); if it does continue with the next edge

They are illustrated in figure 5.2.3.



(a) choose one edge and rotate it

(b) project both polygons

(c) check for intersection

Figure 5.7.: Collision detection using separating axis

One important thing about the algorithm is that it only works with convex polygons. However, since some of the Tetris tokens are concave (for instance the L-token), the Tetris tokens have to be split into two convex polygons. Thereby each polygon consists of two bricks. Even Tetris tokens that are already convex, such as the Bar-token, are split to ensure that all of the Tetris tokens can be handled in the same manner.

To perform the collision handling a class called separate class is implemented. This class offers four methods, one for each collision that can occur. Those four types of collision are collisions with other tokens, collisions with finished bricks, collisions with the midpolygon and collisions with the borders of the game board.

Token-token collision

The first method is the token-token collision detection. In this method a given token is checked for collision against all the other tokens held by the game board.

In the following, the Tetris token that is given is called token-a and the selected token of the game board is called token-b.

The following procedure has to be carried out for every Tetris token held by the game board.

At first it is checked, whether a collision can principally occur. This is done by checking the bounding spheres of the Tetris tokens. A bounding sphere is the smallest sphere around the midpoint of an object that includes the whole object.

For checking the bounding spheres of the Tetris tokens, the midpoints of both tokens are connected. Then it is checked, whether the length of this vector is smaller than the sum of both bounding spheres' radii. Only when the bounding spheres intersect, the algorithm continues. Otherwise the next token held by the game board is checked.

If the bounding spheres overlap, the rectangles of both tokens are checked pair wise for collision. Therefore the algorithm described above is used. If a collision is detected, it has to be avoided. This is the second part of the collision handling. Therefore token-a is moved 10 pixels away from token-b. Thereby, the moving direction is the connecting vector of the midpoints of both tokens. After moving, both rectangles are checked for collisions again and – if necessary – token-a is moved again. This is repeated until no collision is occurring anymore.

Token-brick collision

The second collision that is checked for by the collision-class is the collision between a given token and the finished bricks held by the game board.

Again, first the bricks that are furthest away from the token of interest are checked for collision. As there are possibly very many bricks on the game board, however, the bounding sphere method is inefficient in this case. Thus the fact that the bricks can only be at coordinates of a gird is utilized. Hence a brick of a token can only collide with nine other bricks of the game board, which is illustrated in figure 5.8. To identify these nine bricks, it is checked in which finishedBrick the midpoint of the token's brick lies. The possible bricks are this very brick and the eight ones next to it. This relates to the fact that distance between the edge points of a brick and the midpoints equals at last $\frac{1}{\sqrt{2}}$ times the edge length, which is less than 1 times the edge length, see figure 5.8. To get the neighbors of the computed bricks, the array of the virtual game board is used once again. In that way, the bricks that have to be checked can be reduced to 18 bricks for each rectangle of the token (nine bricks for each brick of the rectangle) and consequently 36 bricks per token (one token consists of two rectangles).

The collision detection and avoidance work similar to the ones in the token-token collision handling. Again, it is checked, wheather a collision occurs and, if necessary, the token is moved away by 10 pixels.

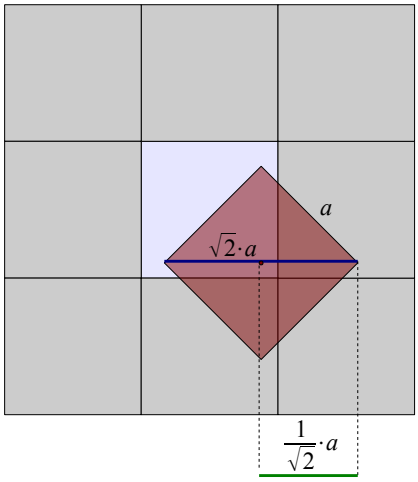


Figure 5.8.: The nine bricks (gray) that can possibly collide with the token's brick (red)

Token-midpolygon collision

The third type of collision handling that is performed by the collision class is the collision between a given token and the midpolygon of the game board. Since the midpolygon only consists of one rectangle, this collision handling is rather straight forward. It has to be checked whether any of the two rectangles of the Tetris token is colliding with the midpolygon. To avoid the collision the token is moved 10 pixels away from the midpoint until no further collision occurs.

Token-border collision

The last collision is the collision with the borders of the game board. This type of collision is by far the fastest and most simple one. Once again, at the two rectangles of the given Tetris token are considered. This time it is checked if the x- and y-coordinate of the rectangles' edge points are in range of the game board. If not, the token is moved until it fits in the game board.

To perform the complete collision handling the four described methods are called one after another. Therefore each method changes the midpoint of the token in a way such that it does not collide with the other object. The next collision handling method is then called with the new midpoint. Calling all methods it is ensured that all collisions are resolved.

5.2.4. Menu, sound and utilities

At this stage, the TUMtris program is already playable. Tokens appear, can be moved without colliding, and are moved automatically into the direction of the nearest border or to the bottom. Once the token hits a finished brick or the border the bricks of the token are finished and the computer checks if there is any completed square or line. At this stage, the game fulfills all the requirements demanded for a prototype application by the approach of game development.

However, since this application aims at being more than a prototype, there are a few more details that have to be implemented to ensure a remarkable gaming experience.

To that end, an appropriate menu to set up different settings or to start a new game is required. For this reason the GameMenu class is implemented, see figure 5.9. This class is a Container and holds different Buttons and Labels. The menu is called by pressing the menu button in the game. This button is in the middle of the screen. After pressing the button, the game will pause and the menu will be shown in the game board and raised to the top.

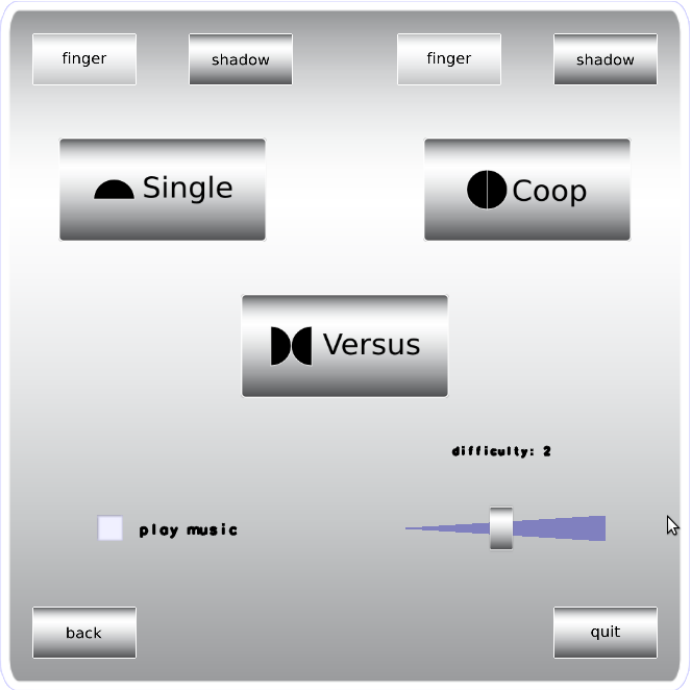


Figure 5.9.: The menu screen of TUMtris

The GameMenu offers following different options, which are provided via a Button: Start a new coop game, start a new versus game, start a new single game, turn music on/off, change control type, quit game and exit menu. Each of these buttons is a separate class inheriting from the libTISCH Button class. This is required because the original Button class does not provide any callback function. Furthermore each button has its own texture that is why there have to be different classes for the different buttons.

Besides the buttons the game menu provides a slider. A slider is a widget which can be used to adjust a numerical value by linearly moving the sliding part of the widget. This slider is used to adjust the difficulty of the game. Therefore the slider has to be moved to the left to make the game easier and to the right to make it more difficult. The higher the difficulty of the game, the faster the virtual tokens will appear and be moved.

The second vital element of video game are a background music and sounds in general. The first approach to integrate sound into TUMtris was by using .wav files and playing them with the built in SoundPlayer class of the .NET framework. The first test run in-

dicated however, that this approach is futile: the sound was played but with decreased speed.

Besides .wav files, the .mp3 format is popular for storing music and sounds. According to MSDN playing mp3 files through .NET requires the Windows Media Player [15]. Since the Windows Media Player is not available on a Linux machine, an external media player for Linux offers an alternative. The player of choice is mplayer[1] because of its short loading time and it is controllable via a command line interface. In TUMtris every sound that is played has its own mplayer process. Every time a sound has to be played this process starts. Therefore it is important that the chosen media player is quite fast. To provide a good gaming experience not only background music is included (a ska version of the original Tetris soundtrack), but also a special sound associated with the pushing of a button. Moreover an explosion sound appears every time a line/square is dissolved.

The third item that was implemented beside the two big parts is an Util and an Image class. The Util class has two functions: The first on is that it owns several useful static functions which are needed throughout the code. Those functions are mostly functions to manipulate vectors, for example: add vectors, inverse vectors, scale vectors and rotate vectors.

The second feature provided by the Util class is the objectStillBeUsed list. All the features that are in the BrickWidget class are added to this list. This has to be done because the Features are added to a list of a C++ object only. Like this no C# object would hold any reference to this Feature and the garbage collector of C# would simply delete it. If the libTISCH accesses this Feature object a null pointer exception will be thrown. By adding the Feature to the objectStillBeUsed list, the list still holds a reference to the Feature and it is not deleted.

Besides the Util class, there is the Image class. The Image class is used to load the textures of TUMtris. Therefore it holds a collection of RGBAtextures. This collection is needed for two reasons: Since not every class holds its texture as a local variable, there has to be a C# reference to this texture to avoid the problem mentioned above. The second function of the Image class is that each texture has to be loaded only once, because afterwards the texture is in the collision and can simply be returned and has not to be loaded again.

In sum, there are 36 classes in 35 files used for the implementation of TUMtris, realized in about 3000 lines of code. A class diagram of the complete project is shown in the appendix.

[1] http://www.mplayerhq.hu

6. Play testing

According to the conceptual approach in game development described in chapter 3 the prototype application has to be tested. The play test is not only about the pleasure that the players experience during the game, but also about evaluating how well the tangible control works in a computer game.

6.1. Play test environment

The procedure of the test is the following: The test players first play the game with the different control types and game modes. Afterwards they fill in a questionnaire. To get a valid result, the test environment has to be the same during the whole test.

The main reason for the play test is to find out how the game modes and the control are working. Therefore the touch and the tangible control are tested separately. In order to test the tangible control type, the versus mode is chosen because this game mode is well-known from the original Tetris and therefore the game mechanics itself has already been tested extensively. Instead of providing a set of game tokens for each player, the two players playing against each other are using the same set. This is to improve the conversation between the two opponents.

The explanation of the functionality of the tangible control was very brief: the players were only told that the tokens have to be used as control and that there is only one set of tokens for both players. The instructions were rather short in order to verify whether the tangible control is as intuitive as expected. After this introduction the players had seven minutes to play the game. A seven minute period was chosen for providing the players sufficient time to get to know the control while preserving the memory of the difficulties at the start.

After the first test both the control type and the game mode were changed. For testing the altered game mechanics the touch control was used. This is because the touch control uses the known gestures for moving and rotating. Thus the results of the second test are expected to be not at all influenced by the control type. Before starting the test, the players got a short explanation about the new game mechanics (that they have to build up a square and that the tokens will move in all four directions). Then, again, the players had seven minutes to test the game mechanics.

For the test 12 test persons between 20 and 29 years of age were invited and grouped into six pairs. The genders were not distributed equally among the test persons: 10 were male and 2 female. Since one of the goals of TUMtris is to make video games more attractive for persons who do not play video games that much, only about half of the testers play video games regularly. These testers are invited to check the mechanics of TUMtris from a gamer's point of view.

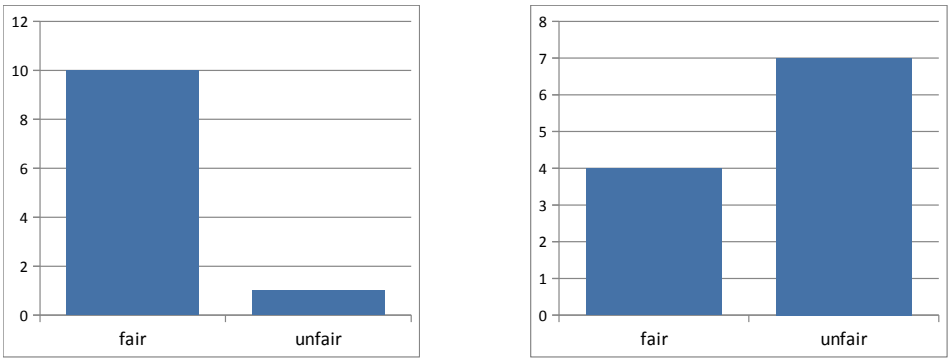
As mentioned above the games were played pair wise by two test players. Some of those test pairs knew each other while others were chosen randomly in order to get more differentiated information on the communication while playing TUMtris.

6.2. Results for the game mechanics

The results of the play test are structured into the outcomes that are important for the game development, such as the game mechanics and the occurred bugs and those which are of importance for the scientific part of this thesis, such as the tangible control and the communication between the play partners.

First, attention shall be drawn to the results for the game development. As mentioned before, the players had to play both game types of TUMtris, the classical Tetris game mechanics and the mechanics where they have to build a square instead of a row.

Afterwards they were asked, if one of the game modes was 'unfair' or 'too hard' in any way. The versus mode was judged as being quite fair, whereas about one third of the testers had the opinion that the coop mode is unfair, see figure 6.2. In fact, none of the testers was able to build a square during the test. The reason why this mode might be considered unfair is that the game board is too small for the token size. Thus the game cannot be completed after one token is misplaced.



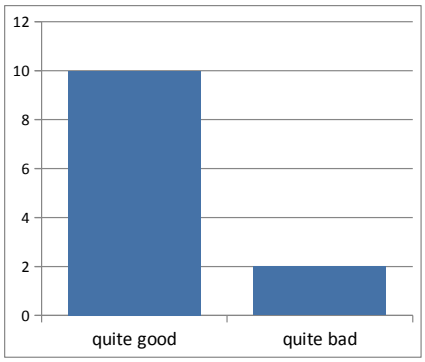
(a) Diagram showing the evaluation of the versus mode

(b) Diagram showing the evaluation of the coop mode

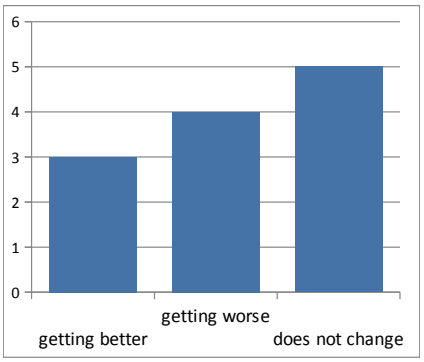
Figure 6.1.: Results of the game mode evaluation

Another question being asked was about the first impression of the game and how this impression changed while playing the game. Since this is an open question, it is very difficult to summarize it and put it into a diagram. Nevertheless, the overall first impression of most of the players was that the idea of TUMtris is quite interesting and that the look of the game is appealing. Only one test player disliked TUMtris on the first view. Unfortunately, the questionnaire was not followed by a personal interview; therefore no reason for this opinion can be given.

The opinions on how the impression was changing during the test are more diversified. About half of the testers said that the game was worse, then they thought at the beginning (Figure 6.2). The other half said that their impression did not change at all. Here it should be mentioned that not the game mechanics were the reason why the impression was influenced in a negative way, but the game controls.



(a) The first impression of TUMtris



(b) The change of the first impression during the game

Figure 6.2.: Results of the overall impression of TUMtris

Regarding the question on how often and long the testers would play this game, all of them answered that they would probably spend about 20 minutes per week on playing TUMtris. This answer is due to the game modes since there is no long time challenge. There is no high score which would allow for comparing the players' performance with other players' score in the single player mode and in the versus mode; the players do not really play against each other, since the strategy of one opponent does not affect the other opponent's actions.

In summary, the main idea behind TUMtris was considered as being quite good, but this is not enough to make the users play with TUMtris for a longer time. The game's design is looking good. However, the game mechanics have to be improved to make the game more challenging in the versus mode on the one hand, on the other hand the coop mode has to be more balanced.

6.3. Results for the control part and the communication

Besides the game mechanics and overall feeling while playing TUMtris, in this thesis it is more important to evaluate how well the tangible control works in a video game such as TUMtris. Therefore not only questions about the game mechanics and the fun factor while playing the game were asked, but also some questions about the control parts were included in the questionnaire. In particular, mostly the same questions were asked about the touch and the tangible control to get a useful and comparable result. The first question being asked was on how intuitive the handling of the specific control type was. This is

important to know, since the testers had not been told how to use the tokens at all. The result is quite clear, because only one test player had problems with understanding how to use the tokens. The other ones did not have any trouble with the control at all. For the touch control the result is not surprising, since well-known gestures for moving and rotating objects are used. Thus the touch control was intuitive for all of the testers.

Although the control mechanics are easy to adopt, the control itself is not working impeccably. Almost every tester had a problem in controlling the Tetris tokens in the correct way. This is caused by recognition errors on the hardware side. Sometimes the touch detection does not work and sometimes the tokens lying on the table are not recognized properly. Furthermore, the reaction time for the touch control is quite long. Thus it is almost impossible to play the game in an appropriate speed. This, in fact, is one of the biggest problems of TUMtris.

However, although the control of TUMtris in not very precise, the testers had fun playing the game. When answering the question on which control type they would prefer, the opinions of the play testers differed. One third would prefer the touch control, another third thinks the tangible control is the best and the rest does not have any opinion concerning this question. Surprisingly none of the testers preferred the classical Tetris control which involves using buttons, see Figure 6.3.

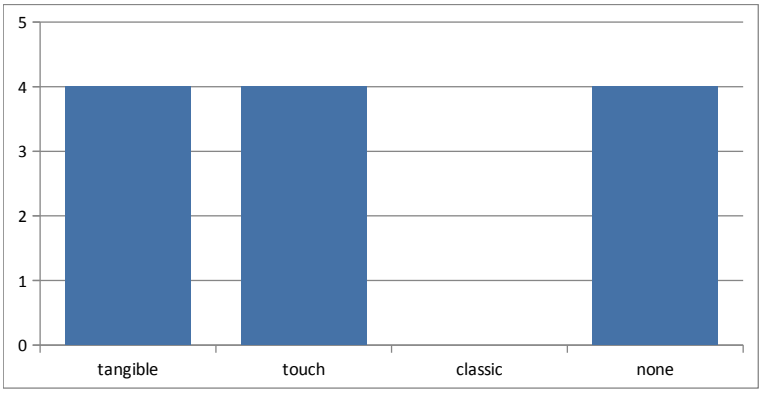


Figure 6.3.: Results for the game control

As one of the tasks of this thesis is to narrow the gap between board games and video games, the testers were also asked if they felt like playing a video game or a board game. Again the opinions differed. Three testers would describe TUMtris rather as a board game, five testers would choose the attribute video game and the remaining four testers said that the game was a hybrid of both types. According to these answers it can be assumed that TUMtris does in fact integrate the board game plying feeling in a video game. Besides, almost all of the test players think that even more complex games could easily be controlled by tangible tokens, although the accuracy of the detection would have to be improved.

Furthermore, it is important to know how much the play partners talked during the test. The fact whether the play partners knew each other before has to be considered when evaluating the results. In this test there were six pairs of testers out of which five pairs

knew each other and one pair was randomly put together. During the game five of the six pairs were chatting, but all of the conversations were about the game itself. Surprisingly the one pair that did not chat consisted of two persons knowing each other.

In a nutshell the idea behind the tangible control seemed to be a good one to all of the testers, but the accuracy of the tangible control definitely has to be improved. Besides that, nearly all of the test player would like to see more complex video games being provided with a tangible control.

6.4. Bugs occurred

Besides the evaluation part, the play test is also a tool for finding bugs in the software and in the hardware that have to be solved. The bugs that occurred during the test are described in this chapter.

The bug report is divided into three parts. The first part shows the hardware problems that occurred, the second part deals with the bugs of the libTISCH and the third part is about the bugs of TUMtris itself.

Although the hardware works nearly perfectly in most of the cases, there is one problem that occurs while working in the mixed mode (shadow and touch detection). In this mode, the right half of the table is not touch-sensitive. Thus it is impossible to use the touch control in the mixed mode. The lack of recognition is caused by the fact that the LEDs mounted on the right side of the table top do not work in the mixed mode. However in the touch-only mode of the TISCH device they work quite well. Thus it is necessary to change the operation mode of the TISCH device in order to switch from tangible control to touch control.

Apart from this hardware issue, there are several bugs in the libTISCH. The first one is that storing information in the configuration file does not work properly. Thus the threshold of the shadow detections changes after each start of the touch daemon. The daemon has to be started several times before the threshold is set correctly. The second bug that occurred during the test is that the sticky flag of the libTISCH does not seem to work in shadow detection. The sticky flag is responsible for a blob being detected even though it has left the region. Because of this bug, the tangible control does not perform as expected, since the game tokens can be moved only very slowly.

Two bugs occurred concerning the game itself. The first one is that the collision detection does not always work properly. This sometime forces a Tetris token to move into an unwanted position. This bug occurred twice during the test. The second bug is that from time to time not only one but two Tetris tokens appear at the same time.

7. Future work

Besides the improvements of TUMtris itself the results of the evaluation are taken into account in order to estimate whether tangible user interfaces will have an impact on game play in consumer games in the future.

7.1. Improvements for TUMtris

The play test clearly indicated that the game TUMtris itself is not finished yet. There are several features that have to be included, respectively changed. However, since this thesis ends with the play test, these improvements have to be carried out in future projects.

The test players of the play test were asked for the most important thing they would change to make TUMtris a better game.

The first improvement of TUMtris which was advised to be done is that more features should be included. Especially the versus mode gets boring after playing it for a while. An improvement could be for instance that an incomplete line is added to the opponent's game board after disolving one. This would be similar to the versus mode of the original Tetris. A second possibility to improve the versus mode would be to integrate special bricks. Those bricks should have a special texture and cause a special effect when relieving them. Those effects could be that one of your incomplete lines is relieved or the bricks of the opponent's board are randomized. These special bricks are not a new idea. They can be found in a game called Tetrinet, mentioned in chapter 2, which is a versus only multiplayer Tetris game developed for Linux. A third way to make the game more challenging could be by including a ranking system. With this system the players could beat each other by setting new high scores. This would result in quite a good long-term motivation to play TUMtris.

Besides the versus game mode, the coop game mode has to be improved as well. As mentioned in chapter 6, the coop game mode is way too hard and almost not feasible. One possibility to change this is to change the size of the game board and, accordingly, the size of the tokens in a way such that more tokens fit into the game board. Thus there is more room between the occurring tokens and the borders which would make the game much easier. However, since there have to be more tokens used to build a complete square the game would also become more difficult in that way. Furthermore it is impossible to increase the size of the game board, as it already fills the complete screen and decreasing the size of the virtual tokens would result in new physical tokens having to be created.

Another improvement mentioned in the evaluation form is that the presentation has to be improved. Though TUMtris already has quite good looking textures and some audio, it could include some more visual effects and more themes used for background music. Besides it should be possible to change the volume of the music as well.

Besides the change of the coop game mode all the issues mentioned above are not very important for the game play itself. The one thing most of the testers (about one half) want to be changed is the control of TUMtris. In the testers opinion the tangible control using the game tokens has to be much faster. Furthermore the computer must not lose track of the game tokens lying on top of the screen. As mentioned before this is not a problem of TUMtris itself but a bug of the libTISCH. Once this bug is fixed the tangible control will automatically be improved as well.

According to the approach in game development described in chapter 3, another play test has to be performed after the improvement of the game in order to decide whether there are further changes to be done to improve the game.

7.2. Future opportunities for TUI games

The question whether it is useful to control video games with a TUI could not be completely answered during the play test. This is because the tangible control of TUMtris does not work well enough to draw a conclusion for tangible controls in general. However, some interesting results could be obtained.

One result is that there are much more requirements on the hardware in a game than for other usages. The shadow recognition should respond immediately, i.e. with no delay at all. Furthermore the hardware has to be able to detect fast movements and – most importantly – there must not be any lose of track. In other TUI applications than games losing track is normally not a serious issue. However, in a game with many actions taking place at the same time losing track can cause much more trouble. Therefore it has to be ensured that the shadow recognition works perfectly and is absolutely reliable before it can be used in a game.

In theory the TISCH device is able to offer all these requirements, but some bugs in the libTISCH counter these features.

Another result is that the tangible interface is as intuitive as expected. Thus developer of games using a TUI can set the focus more on the game mechanics itself, because they do not have to worry about the control being too difficult to understand. According to the evaluation form, most of the test persons could imagine playing a more complex game with a TUI. In fact, 25% of the testers are keen on playing more complex games with a tangible control. Compared to the fact that the tangible control in the test did not work perfectly, it can be stated without a doubt that providing tangible controls is a good way to improve game experience in the future.

Appendix

A. Questionnaire

Allgemeine Fragen:

Geschlecht:

Alter:

Beruf:

Wie oft spielen Sie Videospiele?

Wie oft spielen Sie Brettspiele?

Kannten Sie Ihren Spielpartner?

Wie viel Zeit würden Sie damit verbringen, TUMtris zu spielen?

Wie war Ihr erster Eindruck von TUMtris?

Wie hat sich dieser Eindruck im Laufe des Spiels geändert?

Was war für Sie ein Alleinstellungsmerkmal von TUMtris?

Würden Sie TUMtris auch alleine spielen (Singleplayer)?

Wie hat sich das Spiel angefühlt?

Wenn Sie eine Sache am Spiel ändern könnten, was wäre das?

Fragen zur Steuerung:

War die Steuerung mit den Spielsteinen eingängig?

Hatten Sie das Gefühl, ein Brett- oder Computerspiel zu spielen? Warum?

Könnten Sie sich vorstellen, komplexere Spiele mithilfe einer tangible Steurung zu bedienen?

Was hat Ihnen an der Steuerung mit den Spielsteinen gefallen?

Was hat Ihnen an der Steuerung mit den Spielsteinen nicht gefallen?

Fänden Sie es besser bei der Steuerung mit den Spielsteinen mit den Spielsteinen zwei Sets an Spielsteinen zu haben (für jeden Spieler ein Set)?

War die Touch Steuerung eingängig?

Was hat Ihnen an der Touch Steuerung gefallen?

Was hat Ihnen an der Touch Steuerung nicht gefallen?

Bevorzugen Sie die original Steuerung von Tertis, die Steuerung mit den Spielsteinen von TUMtris, oder die Touch Steuerung von TUMtris? Warum?

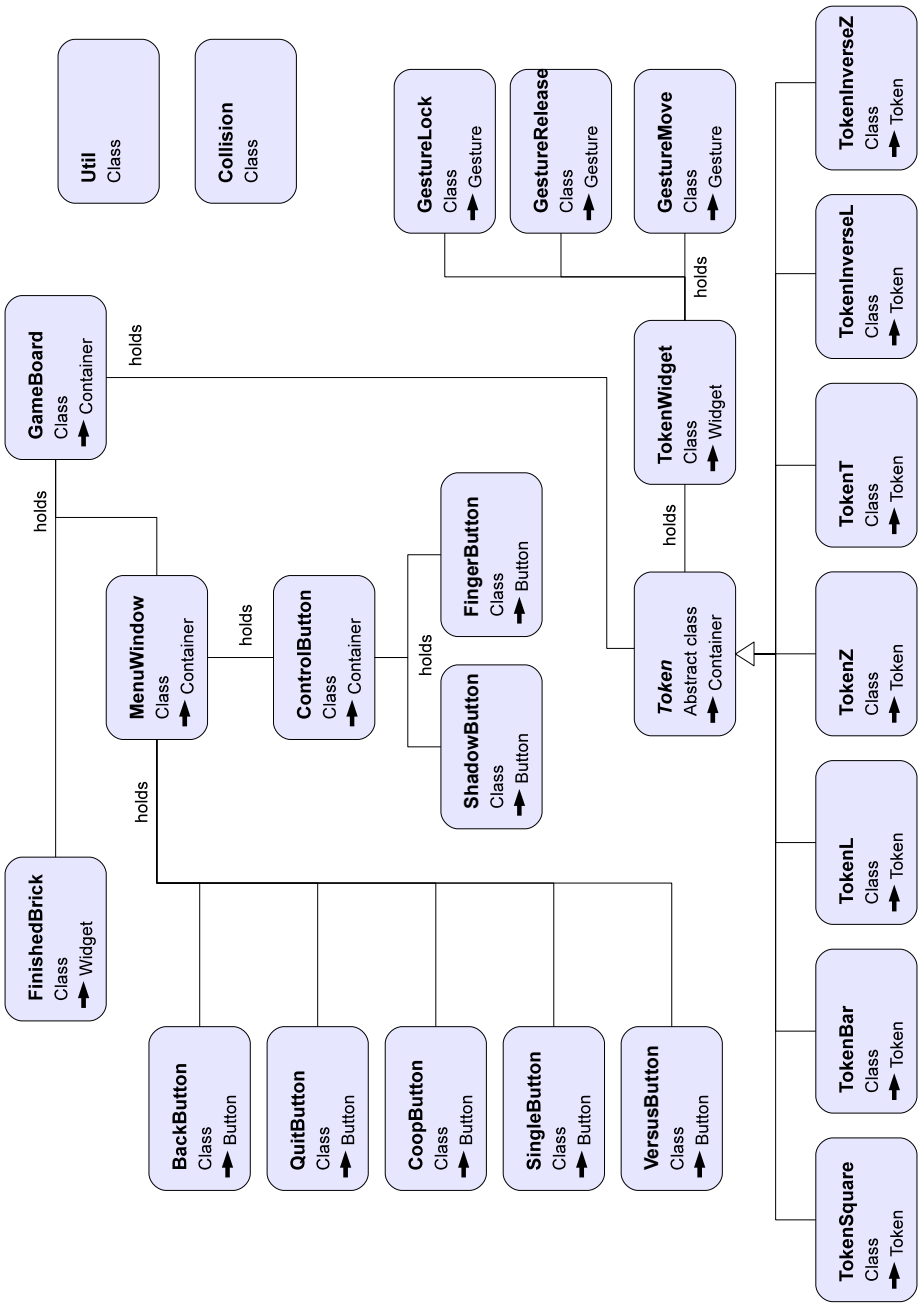
Fragen zu den Spielmodi:

Sind Sie mit Ihrem Spielpartner ins Gespräch gekommen? Wenn ja, drehte sich das Gespräch um das Spiel selbst?

Fanden Sie den Versus Spielmodus zu irgendeinem Zeitpunkt unfair? Wenn ja, warum?

Fanden Sie den Coop Spielmodus zu irgendeinem Zeitpunkt unfair? Wenn ja, warum?

B. Class diagram



Bibliography

[1] Saskia Bakker, Debby Vorstenbosch, Elise van den Hoven, Gerard Hollemans, and Tom Bergman. Weathergods: tangible interaction in a digital tabletop game. In *Proceedings of the 1st international conference on Tangible and embedded interaction*, TEI '07, pages 151–152, New York, NY, USA, 2007. ACM.

[2] Ben Smith Stephen Trush Gary Yee Christian Collberg, Stephen Kobourov Steven Kobes. Tetratetris: A study of multi-user touch-based interaction using diamondtouch. Technical report, Department of Computer Science, University of Arizona, 2003.

[3] Laurent Cozic. 2D Polygon Collision Detection. http://www.codeproject.com/KB/GDI-plus/PolygonCollision.aspx, September 2006.

[4] Paul Dietz and Darren Leigh. Diamondtouch: a multi-user touch technology. In *Proceedings of the 14th annual ACM symposium on User interface software and technology*, UIST '01, pages 219–226, New York, NY, USA, 2001. ACM.

[5] Florian Echtler. *Tangible Information Displays*. PhD thesis, 2009.

[6] BIU e.V. Gamesbranche im jahr 2008 mit neuem umsatzrekord. http://www.biu-online.de/home/news/16-februar-2009-gamesbranche-im-jahr-2008-mit-neuem-umsatzrekord, February 2009.

[7] Jefferson Y. Han. Low-cost multi-touch sensing through frustrated total internal reflection. In *Proceedings of the 18th annual ACM symposium on User interface software and technology*, UIST '05, pages 115–118, New York, NY, USA, 2005. ACM.

[8] Hiroshi Ishii. The tangible user interface and its evolution. volume 51, pages 32–36, New York, NY, USA, June 2008. ACM.

[9] Hiroshi Ishii, Craig Wisneski, Julian Orbanes, Ben Chun, and Joe Paradiso. Pingpongplus: design of an athletic-tangible interface for computer-supported cooperative play. In *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, CHI '99, pages 394–401, New York, NY, USA, 1999. ACM.

[10] Sergi Jordà, Martin Kaltenbrunner, Günter Geiger, and Marcos Alonso. The reactable: a tangible tabletop musical instrument and collaborative workbench. In *ACM SIGGRAPH 2006 Sketches*, SIGGRAPH '06, New York, NY, USA, 2006. ACM.

[11] Rilla Khaled, Pippin Barr, Hannah Johnston, and Robert Biddle. Let's clean up this mess: exploring multi-touch collaborative play. In *Proceedings of the 27th international*

conference extended abstracts on Human factors in computing systems, CHI '09, pages 4441–4446, New York, NY, USA, 2009. ACM.

[12] Jakob Leitner, Peter Brandl, Thomas Seifried, Michael Haller, Kyungdahm Yun, Woontack Woo, Maki Sugimoto, and Masahiko Inami. Incretable, bridging the gap between real and virtual worlds. In *ACM SIGGRAPH 2008 new tech demos*, SIGGRAPH '08, pages 19:1–19:1, New York, NY, USA, 2008. ACM.

[13] Carsten Magerkurth, Maral Memisoglu, Timo Engelke, and Norbert Streitz. Towards the next generation of tabletop gaming experiences. In *Proceedings of Graphics Interface 2004*, GI '04, pages 73–80, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2004. Canadian Human-Computer Communications Society.

[14] MSDN. Collision detection overview. http://msdn.microsoft.com/en-us/library/bb313876(XNAGameStudio.20).aspx.

[15] MSDN. Soundplayer class. http://msdn.microsoft.com/en-us/library/system.media.soundplayer.aspx.

[16] Christopher Swaine Tracy Fullerton and Steven Hoffman. *Game Design Workshop: Designing, Prototyping and Playtesting Games*. CMPBooks, 2004.

[17] Mischa Trecco and Ricardo Alvarez. 't-touch': A table computing multi-player game. Technical report, HSR Hochschule für Technik Rapperswil, 2009.

[18] Dirk Werner. *Funktionalanalysis*. Springer, 2. auflage edition, 1997.