DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics: Games Engineering

Interactive Visualization of Complex State Diagrams for Optimal Developer Experience

Moritz Will

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics: Games Engineering

Interactive Visualization of Complex State Diagrams for Optimal Developer Experience

Interaktive Visualisierung komplexer Zustandsdiagramme für ein optimales Entwicklererlebnis

Author:	Moritz Will
Supervisor:	Prof. Gudrun Klinker, Ph.D.
Advisor:	Daniel Dyrda, M.Sc.
Submission Date:	August 15, 2021

I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Munich, August 15, 2021 Moritz Will

Acknowledgments

I would like to greatly thank my bachelor supervisor Daniel Dyrda, M.Sc., for his guidance and many valuable discussions. Without his support, this thesis would not have been possible. I would also like to sincerely thank Prof. Gudrun Klinker, Ph.D., without whom this thesis would not have been written.

Abstract

This bachelor thesis is about the interactive visualization of complex state diagrams for optimal developer experience. Various results of existing work will be outlined and then applied in the practical part. Different interaction methods in connection with statecharts are examined and evaluated. This includes the use of the *motion parallax effect*, which tracks the user's head position to create a better 3D effect. Animations to better highlight informations of the statechart were also tested. These are among others an *outlining effect* and a *frames effect* which highlight certain relations in the state diagram and thus make them easier to understand. Also an experiment to visualize the execution of a statechart was performed and analyzed. The work concludes that there is probably a great potential in the interactive visualization of complex state diagrams, but it needs to be further investigated.

Contents

1 Introduction

State diagrams are of great importance for developers. They allow to visualize the functionality and different states of a system in an understandable way. This contributes significantly to a better comprehensibility. For this reason, their scope of application is very wide and covers many areas. For example, state diagrams can be used in the automotive industry, in telecommunications, in user interface design and in computer operating systems [Har87]. One example for the broad applicability of state diagrams is given in [RK91], where they are used in the context of food processing and food development.
Due to the heavy and versatile use of state diagrams, it is important to optimize their visualization and the corresponding interaction methods. This ensures that the developer can get the required information from the state diagram as quickly as possible and productivity is maximized.

1.1 Research question

In this thesis it it will be investigated whether and which interactive visualizations can contribute to an optimal developer experience. It will be examined whether animations as well as interaction methods like the motion parallax effect can contribute to reduce the cognitive load and improve the immersion. It will also study how best to highlight the elements that are important to the user to support productivity.

1.2 Outline of the thesis

In the following, related work will be presented and the relevant terminology will be introduced and explained. In addition, different results from previous studies will be summarized. These insights are then used in the subsequent implementation part of the thesis. There a statechart with different interaction methods and animations is implemented. Also the visualization of the execution of the statechart is developed. Then the individual features are investigated and evaluated. Finally, the results of the evaluation are analyzed and an outlook is given.

2 Related work

Interactive visualization of statecharts has not been studied yet much. However, there have been several studies on UML diagrams. UML class diagrams and statecharts have many similarities in their visualization and scope. Both diagrams consist of nodes and edges and have their application in the field of computer science. In addition, both are static and do not move. Because of these similarities, a transfer of the obtained results in the study of interactive visualization of UML diagrams to statecharts is perhaps possible and should be investigated.
In [RG00], the three-dimensionality in the three-dimensional representation of UML class diagrams is used to direct the user's focus to certain elements and areas. Furthermore, animations were used to create a more natural user interaction and visualization, allowing the user to better understand sudden changes in the image. [TS03] is a similar approach that uses the interactive three-dimensional representation of UML diagrams to support the didactic aims of the diagram to make comprehension easier and more intuitive. However, both studies did not have a well-founded investigation with test persons and present individual perceptions of the authors.
In [Kru12], different interaction methods for three-dimensional applications were presented and investigated. It is also emphasized that the interaction with three-dimensional visualizations is difficult for humans on the computer. This is mainly due to the fact that many of the cues that help three-dimensional perception are often missing in three-dimensional applications. For this reason, it is particularly important that the user interfaces and interaction methods for three-dimensional applications are optimized as much as possible. The book also explains that interaction methods such as head tracking can contribute to this.

3 Statecharts

In this work we use the term "statechart" as defined in [Har87]. Thereby a statechart is a state diagram, which additionally can have levels, an orthogonality and a broadcast communication. The orthogonality and broadcast communication are not needed for this work and are explained in the mentioned literature [Har87].
State diagrams are directed graphs consisting of states and transitions. States are denoted as nodes and are symbolized by rounded rectangles. Transitions are denoted as arrows with the triggering events and guarding conditions shown next to them.
However state diagrams have disadvantages. For example, if a state diagram should always reach a certain new state when a specific event is triggered, as many transitions as states would have to be added. Figure 3.1 shows three nodes "a", "b" and "c". "d" is the new node that should always be reached when the event "n" is triggered. Therefore, for every state a new transition has to be added to node "d" with the trigger-event "n". These many new transitions make a diagram very fast unreadable.
This problem is solved by making all affected states children of a superstate, which then has a transition to this new state. This creates different levels in the diagram. The superstate then lies on a lower level than its child-states. Figure 3.2 shows again the nodes "a", "b", "c" and the new node "d". Additional there is the node "s", the new parent-state of "a", "b" and "c". As can be seen it now only requires one new transition to "d". The resulting parent-child relationships are an essential part of the statechart and therefore of great importance for the developer.
This work uses for the investigations a statechart from [Har87], which is part of a statechart presenting an electric wristwatch. It can be seen in Figure 3.3.

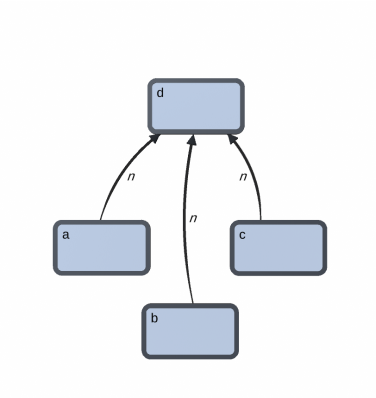

Figure 3.1: Diagram without depth.

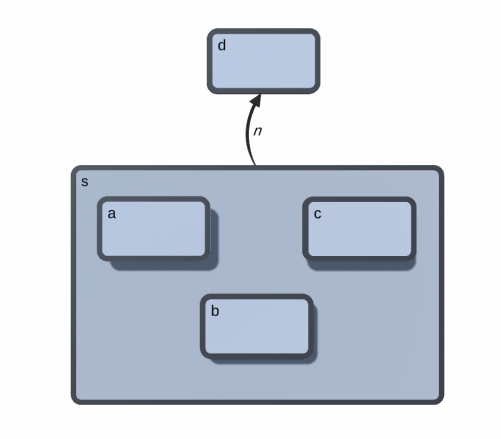

Figure 3.2: Statechart with depth.

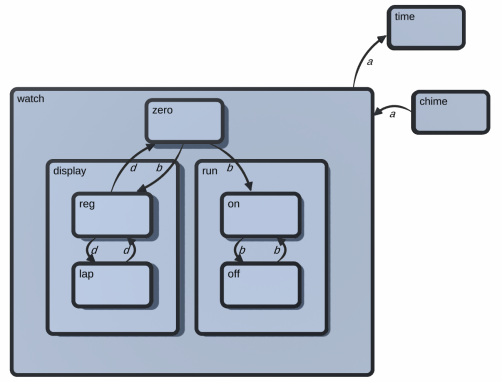

Figure 3.3: Used statechart for this work.

4 Ways to improve the developer experience through interactive animations

In this section, the benefits of animations will be explored to ascertain if they could be helpful in this project as well. Animations are defined as *a series of varying images presented dynamically according to user actions in ways that help the user to perceive a continuous change over time and develop a more appropriate mental model of the task* [Gon96].

4.0.1 Advantages of animations

Research shows that sudden visual changes in the user interface of an application can interrupt the user's task workflow and force a reorientation [CU93]. This takes time and causes uneasiness. Animations can be used to avoid this negative effect. By providing the necessary cues, the animation can help the user to understand what happened, what is happening and what will happen. As a result, the flow is no longer interrupted and the user's effectiveness and sense of well-being are increased.
Compared to static images, animations also have the advantage of interactivity [FH95]. This is also a result of [TMB02]. By enabling the user to start, stop and replay the animation, reinspecting and focusing on specific parts get easier. Also zooming, alternative perspectives, close-ups and control of speed enforce the understanding process by helping to overcome the difficulties of perception and comprehension.
Another advantage of animations is that they can be helpful not only in isolated cases, but in very many areas. In [BS90] many of these areas are analysed to understand what benefit animations have in different situations. When an animation is used as feedback, it can provide current information about the status of a system. Thereby it can communicate the user if the process is active or how close it is to completion. Animations can be used as demonstration and increase the information's content of symbols or other elements. They can be used as identification to help to identify the application when it starts. In doing so it can give the user a small overview about what the application is used for. As mentioned before, they can be used as transitions and orient the user when switching from one process to another. Other use-case areas are using an animation as choice, as explanation, as history or as guidance.

[CU93] also highlights that employing animations lead to a more pleasant and comfortable interface. To this result comes [Dra+11] too, which states that animations improve the aesthetics of a user-interface and help the users to understand the underlying data. Additionally [BB99] states that animations can also improve the user's task performance, because users maintain better object constancy.
In some cases, the learning process can also be accelerated with animations [TMB02]. This was the case in study [Lar+96], for example, in which animated and static diagrams of the heart and blood vessels were compared. Also in [PG92], where students were supposed to learn the troubleshooting of an electronic circuit, performance improvements were measured with animated graphics.
It becomes clear how versatile animations can be used and how many different benefits they can have for the user.

4.0.2 Effective animations

In the past, various criteria for effective animations have been developed. One source suggests that effective animations must satisfy two high-level principles [HR07]: The "Congruence Principle" and the "Apprehension Principle". The "Congruence Principle" states that the structure and content of the external and internal representation should match. The "Apprehension principle" states that the external representation should be comprehensible to the user.
Another research declares that an effective animation has to be smooth, simple, interactive and be adapted to the user's mental model of the task [Gon96]. Furthermore it states that the user should be able to manipulate different segments of the animation in a parallel order. The existing computer and task knowledge of the users should also be taken into consideration.
Principles of cartoon animation can help too [CU93]. The work emphasizes the principles solidity, exaggeration and reinforcement. Solidity means that the animated objects should be solid and should move as if they are real things. They should be animated as if they have mass and are susceptible to inertia. To convey the animation's message even more effective, it should not just mimic reality, but exaggerate dramatically. That is why the second principle is called exaggeration. Reinforcement means that the illusion of reality should be amplified to keep the user's attention. This can also be done by animations that are not very noticeable and are only perceived subconsciously.
[HR07] declares ten rules of design considerations for animations:

1. Maintain valid data graphics during transitions
2. Use consistent semantic-syntactic mappings
3. Respect semantic correspondence

4. Avoid ambiguity
5. Group similar transitions
6. Minimize occlusion
7. Maximize predictability
8. Use simple transitions
9. Use staging for complex transitions
10. Make transitions as long as needed, but no longer

[Gre18] also points out that attention must also be paid to the transitions between several animations in order to avoid a jarring and unpolished appearance.
All these rules and suggestions are taken into account in the practical part.

4.0.3 Afterglow effects

Most animations like the "Slow In Slow Out" animation force the user to wait until it is finished [Bau+06]. Even when the animation time is short, this can delay the user's task and lead to a disturbance of the flow. This issue can be solved with so called *phosphor transitions*. This type of animations shows immediately the outcome of the user's action, but also explains the change in retrospective by using a diagrammatic depiction. It focuses the user's attention on the objects that have changed but allows to continue the work instantly. One good *phosphor transition* is the afterglow effect called "speed lines style". It is used when an element moves promptly from one to another position on the screen. Instead of moving the element slowly to the target location so that the user can track it well, it changes its position abruptly to the target. But a tail lasting several seconds indicates that the element has been moved there from its origin. In this way, the user can easily follow the action without having to wait.

5 Depth cues for better depth perception

One very important part of statecharts is the parent-child relationship between nodes [Har87]. This relation is illustrated by the parent node being placed behind the child node. Thus it lies on a deeper level than the child node. For an optimal developer experience it is important to recognize the different levels and the depth of the nodes easily to understand the statechart fast and correctly. This section examines whether using depth cues can help to amplify the user's depth perception and consequently improve the developers flow.

5.0.1 Types of depth cues

In [Pfa02] depth cues are categorized in several groups. Two-dimensional sources of information that can be interpreted as three-dimensional are called "pictorial depth cues". All perspective-based cues are part of this category. These include shadows and occlusion, the effect that nearer objects can overlap objects that are farther away. Also the relative size between objects with different distances to the user is a corresponding depth cue. The distance to the horizon can also illustrate depth. Other pictorial depth cues are shading the object so that it looks three-dimensional, coloring closer objects other than objects that are far away and the relative brightness and letting far objects slowly disappear in the atmosphere. Also focusing can create a sense of depth. Partially perspective based cues like using objects that have a familiar size to the user, or using the texture gradient as effect also belong to this group. Combining many of these pictorial depth cues can already create a strong sense of three-dimensionality.
Another category is "depth from motion". These cues provide information about the location, the velocity, the acceleration and the direction of the movement of the viewer or the object. These cues contain kinetic occlusion, the change of the amount one object overlaps another, and motion parallax, which will be surveyed later. Also motion perspective, that points move in space according to the laws of linear perspective, and familiar speed, that an element has a velocity that is familiar to the user (e.g. a walking person), are two depth cues of this category.
The category "oculomotor depth cues" includes convergence and accomodation. Convergence involves the rotation of the user's eyes. Accomodation is the eye focus at a specific distance.

The last category for depth cues is the "binocular depth perception" and is not needed for this work. It can be looked up at [Pfa02].

5.0.2 Motion parallax

Motion parallax is an effect that has been studied experimentally since about 1867 [Von25]. A good description provided Helmholtz:

> *In walking along, the objects that are at rest by the wayside [...] appear to glide past us in our field of view [...]. More distant objects do the same way, only more slowly [...]. Evidently, under these circumstances, the apparent angular velocities of objects in the field of view will be inversely proportional to their real distances away; and consequently, safe conclusions can be drawn as to the real distance of the body from its apparent angular velocity. Moreover, in this case there is a relative displacement of objects at different distances with respect to each other.* [Von25]

He also emphasized that motion parallax is a very important and effective depth cue. But several studies came to the conclusion that motion parallax may not be as effective as presumed [RG79]. However these studies all had in common that they either investigated only externally produced parallax situations, where the user did not move, or used a stimulus array consisting of just a few objects.
[RG79] studied the advantages of motion parallax when the user itself moves. It states that motion parallax can be an effective cue to better perceive the shape and depth of three-dimensional objects. Even in the absence of all other depth cues, it can give the user enough information to understand the scene structure. It also indicates that parallax motion is more effective when it is a self-produced motion and not an externally generated parallax condition. It supports the idea that motion parallax can be used as an accurate, quantitative source for depth perception. It is also very interesting, that in his study larger amplitudes of relative movement did not produce larger amounts of perceived depth. Even small movements were sufficient to let the user recognize the three-dimensional shapes.

6 Implementation

The project is implemented in C# using the game-engine Unity (version 2021.1.1f). Unity enables us to visualize and animate three-dimensional objects efficiently.
The different visualizations, effects, and interaction methods have been recorded. These videos are available under the following QR code (Figure 6.1).

Figure 6.1: QR code with the link to the videos.

6.1 Statechart visualization

The statechart is generated from a manually prepared scene consisting of boxes, which represent nodes, and spheres, which mark the positions of the arrows origins and targets. This can be seen at Figure 6.2. Figure 6.3 shows the good-looking statechart, created by the program. It interprets the elements in the preparation and adds the additional objects into the scene. The arrows always stick to their origin and their target nodes and thus can be stretched by repositioning the nodes.
When the user clicks on a state, the statechart moves, so that the clicked state is then centrally located on the screen.
The statechart structure that is used for this thesis is from [Har87] and is part of a statechart simulating a watch.

6.1.1 Statechart depth variants

At the beginning three different types of depth structures were implemented and tested. Thus, it was possible to investigate which depth layer variant is the most suitable for

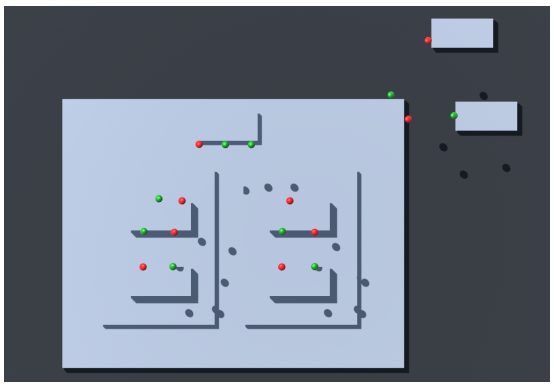

Figure 6.2: Preparation of statechart.

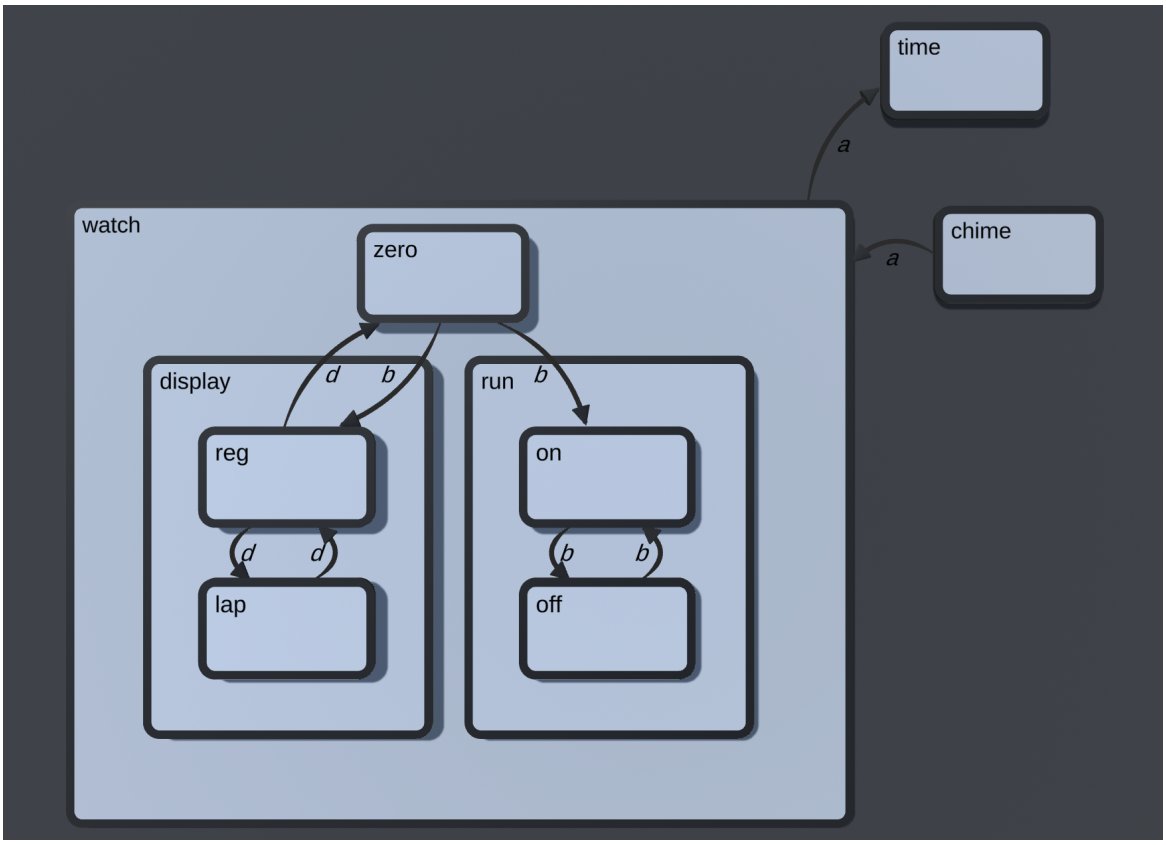

Figure 6.3: Created statechart by the program.

statecharts.
In the first variant, the parent states were always lower than the child states. The individual layers always had the same distance to the layer below. In addition, the nodes were as thick as the layers were spaced apart. This resulted in a pyramid-like structure, where the child node was always directly sitting on top of the parent node. This can be seen in Figure 6.4.
The second variant differed from the previous one in that the nodes were thinner. As can be seen in Figure 6.5, this made it look like the child nodes were floating above the respective parent nodes.
The third type of depth structure had an inverse depth of the layers. This way, the child states were always lower than the parent states. The parent states had holes where the child states were, so that they still could be seen. In addition, walls connected the sides of the child states with the holes of the parent states. The result is an inverse pyramid-like statechart, which can be seen in Figure 6.6.

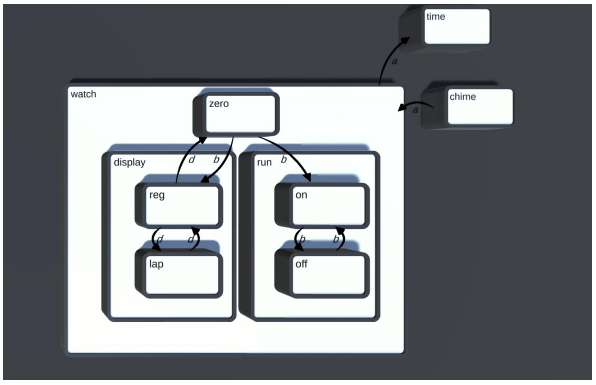

Figure 6.4: Pyramid-like statechart.

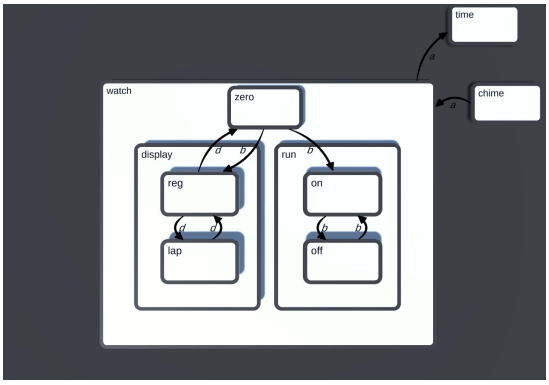

Figure 6.5: Statechart with floating states.

6.2 Implemented depth cues

Since the parent child relationship plays an important role, several features have been implemented and tested to make this relationship clearer. All the optional features can be activated and deactivated individually in a menu. The menu can be reached by clicking the settings button in the top left corner. It can be seen in Figure 6.7.

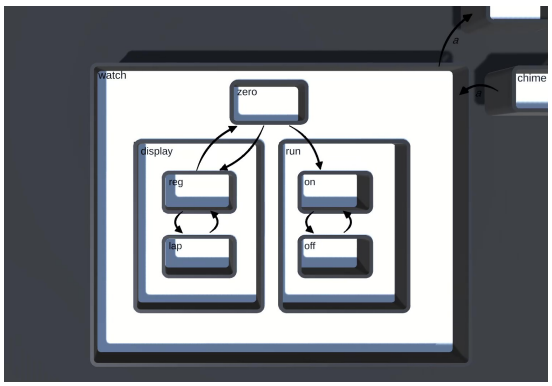

Figure 6.6: Inverse pyramid-like state-chart.

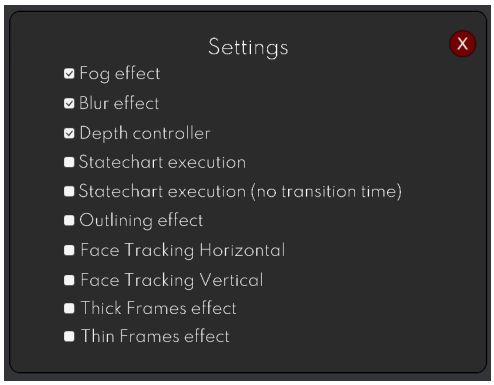

Figure 6.7: Settings menu.

6.2.1 Motion parallax

One important depth cue is the motion parallax. When the user moves their head, the view-position moves in the state-chart scene equivalently. In Unity the view-position is symbolized by a camera. To obtain the position of the user's head relative to the monitor, a connected video camera is required, which is placed on top of the monitor and points at the user. Using the software called *OpenCV*, the program then recognizes the user's face in the transmitted live image. This allows the head position to be calculated.
Normally, the viewer's head is always placed centrally in front of the screen. In Unity, by default, this means that also the camera is always centrally located above the projection plane. However, since the user's head moves, the viewer's position is now no longer centered in front of the screen. In Unity, this has the consequence that the

camera also must be moved relative to the projection plane. Therefore the camera's field of view has to be recalculated for every movement of the user. This is shown in Figure 6.8 and Figure 6.9. Figure 6.10 and Figure 6.11 show the resulting rendered images.
In order for face recognition to work well, it is important that the user is sitting in a well-lit room and that there is a strong contrast between the person and the background color. Furthermore, only one face should be in the camera's field of view at a time and not several. To optimize the user experience and avoid false head height detection, the user can decide which head movements should be considered in the processing. It is possible to choose between taking all head movements into account, taking only the horizontal head movement into account or taking only the vertical head movement into account.

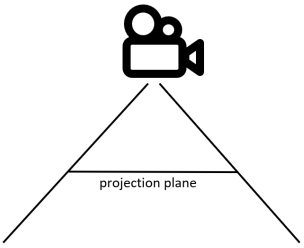

Figure 6.8: Camera's field of view at the beginning.

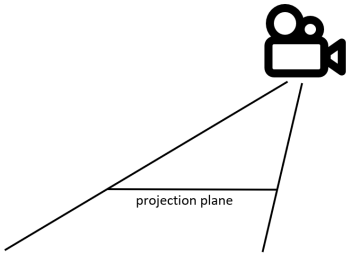

Figure 6.9: Camera's field of view after the user moved.

6.2.2 Depth controller

Another implemented feature is the *depth controller*. It enables the user to change the distance between the layers. This can be done by adjusting a slider shown at the bottom of the screen. When the slider is moved to the right the distance between the layers increases. This can be seen in Figure 6.12 and Figure 6.13.

6.2.3 Fog

Fog lets far objects slowly disappear. This feature was also implemented in the project. If it is activated, the lower layers slowly disappear in gray fog. Figure 6.14 shows the state-chart when the fog is disabled, Figure 6.15 when it is activated.

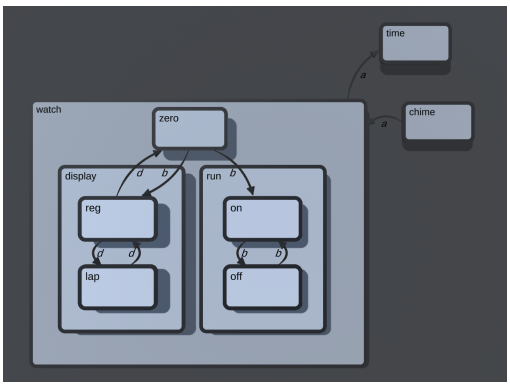

Figure 6.10: View at the beginning.

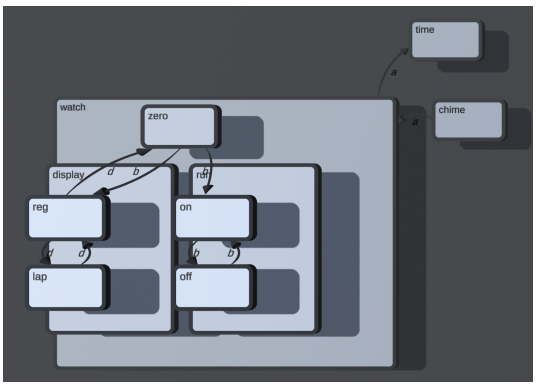

Figure 6.11: View after the user moved their head.

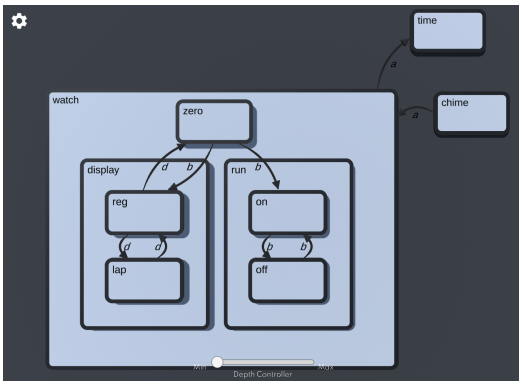

Figure 6.12: Minimal depth between layers.

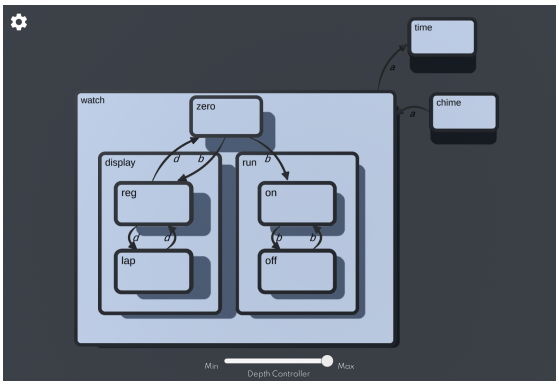

Figure 6.13: Maximal depth between layers.

6.2.4 Blur effect

Furthermore, a *blur effect* was implemented. When the user hovers with the mouse cursor over a node, all nodes that are on lower layers than this node get blurred. All nodes that are on the same layer or on a higher layer stay sharp. This effect can be seen in Figure 6.16 and Figure 6.17. It is realized using Unity post-processing. The transitions between two blur states are smooth and look like a real camera is refocusing on the respective layer.

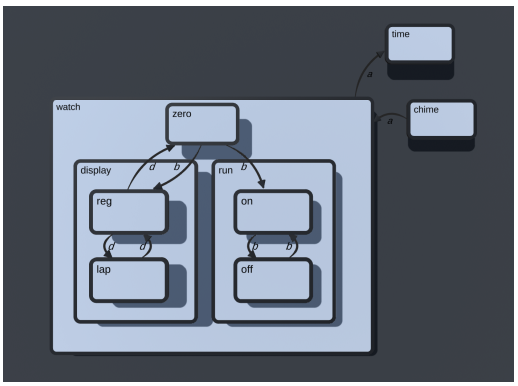

Figure 6.14: State-chart with fog disabled.

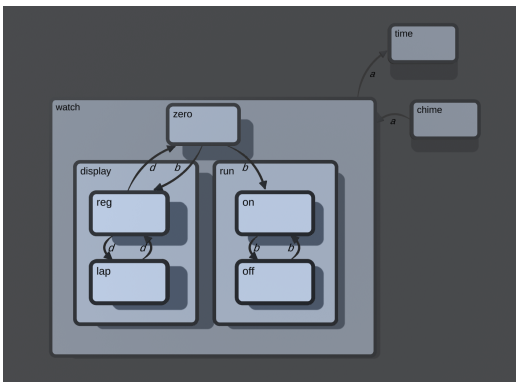

Figure 6.15: State-chart with fog enalbed.

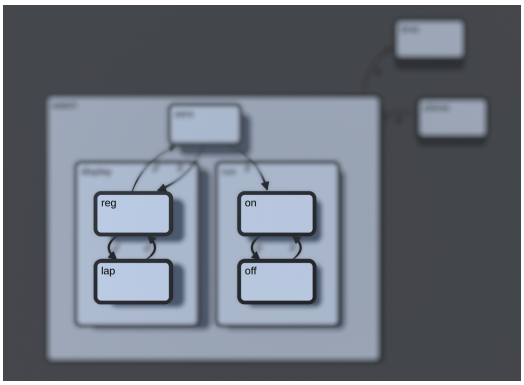

Figure 6.16: *Blur effect* when the user hovers over a node in the highest layer.

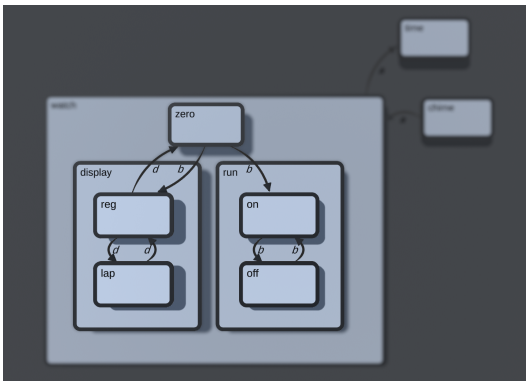

Figure 6.17: *Blur effect* when the user hovers over a node in the middle layer.

6.2.5 Other depth cues

Many depth cues like shadows, occlusion and the relative size of objects are automatically provided by Unity. Other by default enabled pictorial depth cues that are given by Unity are shading the objects three-dimensional and the relative brightness of scene-elements.

6.3 Animations

6.3.1 Outlining effect

The *outlining effect* highlights which statechart-node the user has clicked, and which nodes are children of the clicked node. When the user presses a node, it briefly gets a little bigger and then quickly gets smaller again until it returns to its initial size. During this animation, which lasts about a second, a red frame appears around the node and its child nodes. This frame continuously adapts to the user's perspective, so that it is always visible which node group is currently highlighted. The inner parts of the selected node and its child nodes also turn slightly red. The frame and the red coloring remain until the user clicks on something else. Figure 6.18 and Figure 6.19 show the *outlining effect* after the state "on" or the state "run" was pressed. In the second figure, the user's head is moved and thus the perspective is changed. This shows how the frame also includes the child states.

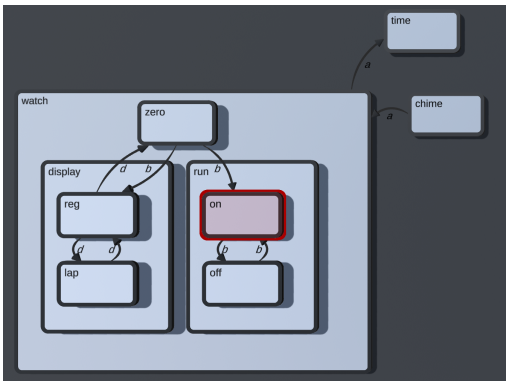

Figure 6.18: *Outlining effect* after state "on" was pressed.

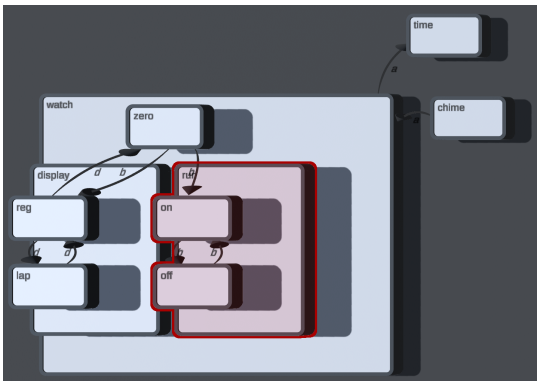

Figure 6.19: *Outlining effect* after state "run" was pressed and the user's head is moved.

6.3.2 Frames effect

Another implemented effect is the *frames effect*. When the user presses a node, for all parent nodes of the selected node a frame is generated. These frames have always the same size like the respective parent node. Two types of frames were examined in the process. For both variants when the frames are generated, they initially have the same position as their corresponding parent state. Then they quickly move upwards to the same height as the selected node, so that after the animation all frames are at

the same height. However, while in one variant the frames always remain thin and finally float above their parent states, in the second variant the frames become larger in depth so that they always still touch their parent state. The first variant can be seen in Figure 6.20 and Figure 6.21. The second variant is shown in Figure 6.22 and Figure 6.23. The frames remain until the user clicks on something else.

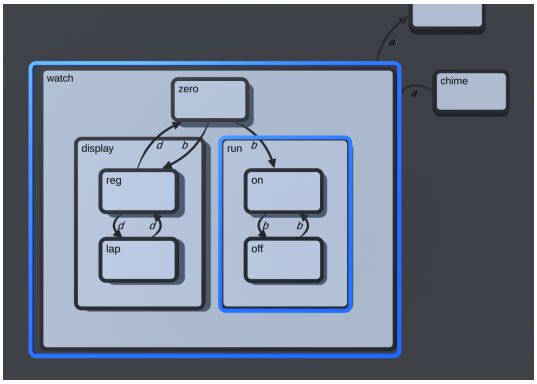

Figure 6.20: *Thin frames effect* after state "on" was pressed.

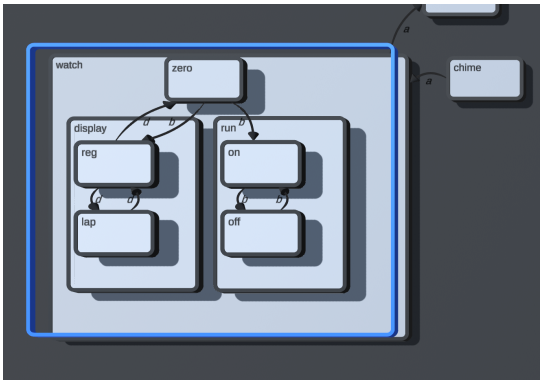

Figure 6.21: *Thin frames effect* after state "run" was pressed and the user's head is moved.

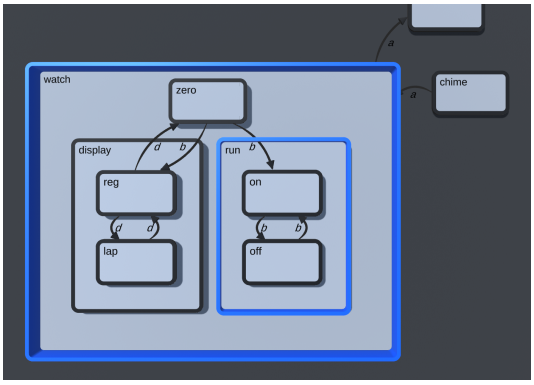

Figure 6.22: *Thick frames effect* after state "on" was pressed.

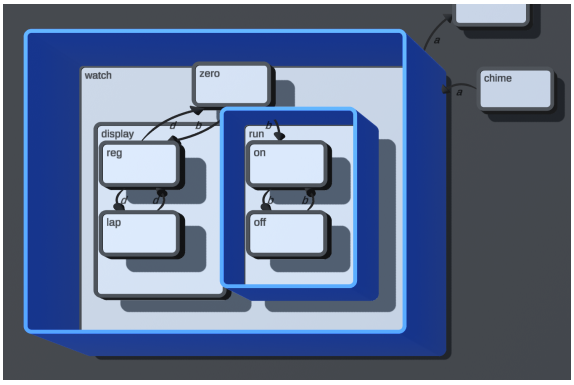

Figure 6.23: *Thick frames effect* after state "run" was pressed and the user's head is moved.

6.3.3 Statechart execution

Also the execution of the statechart was implemented. In this project, executing a statechart means that a start state and an input sequence are given. Then, starting from the start state, the input is read step by step and thus new states are reached. The execution is over when the input sequence has been read completely. In this investigation, the execution sequence is always the same and cannot be changed by the user.
The user can choose between two types of statechart executions. In one type, the states and the transitions are animated one by one. In the other, the transition happens immediately, and therefore the animation of the next state and the respective transition are played simultaneously. The second reflects the system execution more realistically.
When the statechart execution is activated in the menu, an "Execute" button appears in the middle of the bottom of the screen, which can be seen in Figure 6.24. By pressing this button, other buttons become visible and the execution of the statechart starts. This is shown in Figure 6.25. With the new buttons the execution can be paused, fast-forwarded or rewound.
At the beginning the camera moves over the start state, so that the start state is central on the screen. Figure 6.26 shows how then the start state briefly becomes larger and smaller again, and at the same time turns red. After that the start state changes back to its original color. This animation lasts about one second. As soon as it has reached the original color again, the first input is read and the matching arrow slowly starts with the same animation. This can be seen in Figure 6.27. Also the camera moves above the arrow. When the arrow has finished the animation starts at the state where the arrow leads to. After that the next input is read and so on. When the execution is finished, it can be restarted by clicking the "Execute" button again.
Figure 6.28 shows how the statechart looks when the transitions do not consume any time and their animations are played simultaneously with those of the states.

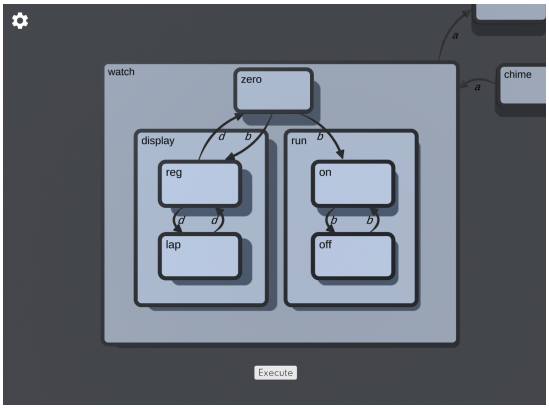

Figure 6.24: View before execute was pressed.

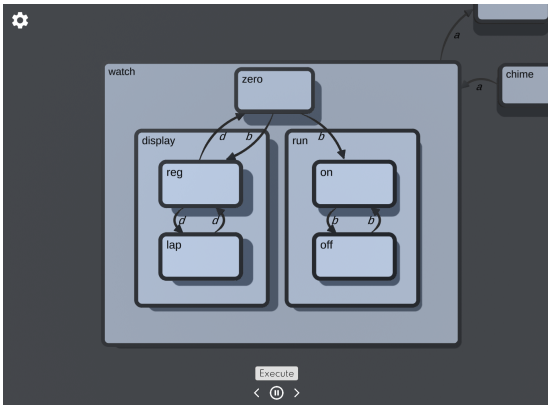

Figure 6.25: User interface with additional buttons, after execute was pressed.

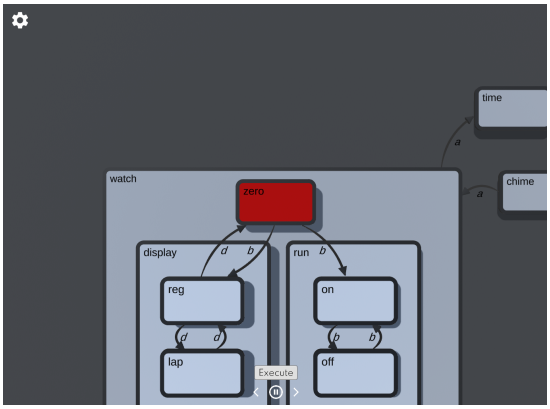

Figure 6.26: Highlighted state during the execution.

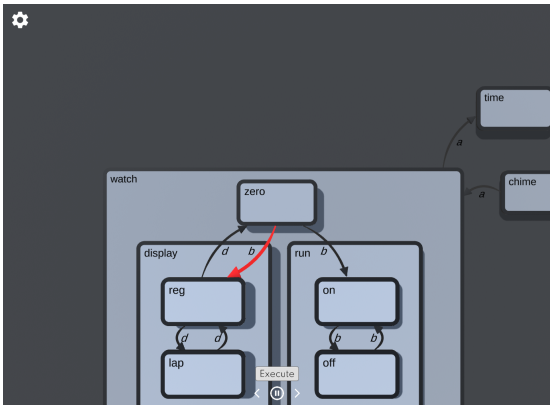

Figure 6.27: Highlighted arrow during the execution.

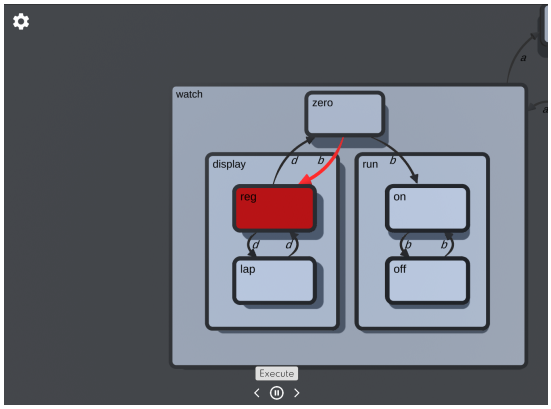

Figure 6.28: Execution with zero time between transitions.

7 Results

The following results are personal assessments of the author and his supervisor.

7.1 Statechart visualization

The pyramid-shaped statechart (Figure 6.4) made the depth of each level clearly visible. The three-dimensional shape of the statechart was well recognizable and understandable. The fact that the child states were directly on top of the parent states made the parent-child relationship lucid. However, the deep frames of the states made the overall picture confusing. Important elements such as the letters next to connections no longer stood out strongly, as the constellation appeared a bit cluttered.
The statechart variant with floating states (Figure 6.5), on the other hand, was very intelligible. Individual letters were directly and good visible. The entire image looked very tidy. The three-dimensionality was also directly noticeable here. This was mainly due to the shadow, which made it clear that the child states were floating above the parent states. This also clarified the parent-child relationships, since the shadow of the child state was always visible on the parent state.
The inverse pyramid-like statechart variant (Figure 6.6) was very different from the previous variants. The three-dimensional structure was recognizable, but was not as obvious as the previously discussed variants. In addition, the shadows made the image confusing due to their large area. The texts of the connections and the names of the states were often partially in the shadows, which made reading difficult. Without the shadows, however, the three-dimensionality was hard to perceive. Nevertheless, the parent-child relationship was very clearly recognizable because the children were lying in their parents' holes.
Comparing the three variants, the floating state variant is the best way to display statecharts.

7.2 Motion parallax

The *motion parallax effect* significantly enhances depth perception. Even small unconscious movements of the head make the three-dimensional structure clear. In this way, parent child connections in particular become easily recognizable. The effect creates a pleasant, natural feeling of responsiveness. The *motion parallax effect* is particularly good when the head is centered in front of the screen and makes small movements from there. If the head is too far to the side, the distortion of the image is so strong that the comprehensibility of the statechart suffers. Therefore, this function is especially useful when the user does not move much in the room but is sitting or standing firmly in place.
Since the face detection does not always recognize the position of the head correctly, sometimes jerky faults can occur in which the camera does not move exactly like the head in the scene. To minimize these inconveniences, it is better in some cases to use only the horizontal head position in the program, as this reduces the frequency of errors. Since vertical head movement is unnatural when sitting and looking at the desktop screen, this limitation of camera movement is hardly noticeable. However, if the user is in a different work environment where head movements take place in all directions, the height of the head position should also be taken into account. For this reason, it is good to give the user the option of which head movements should be taken into account and which should not.
It is also important to note that the processing of the monitor camera image requires a lot of computing power. Therefore, the responsiveness can be disturbed on weaker devices. In addition, the necessity that it must be bright for the camera detection is a limitation.
In summary, the *motion parallax effect* can be a good tool to improve depth perception and by this help to understand the structure of the statechart. However, it should be considered how, where and when the user normally uses the application.

7.3 Depth controller

The *depth controller* provides a good opportunity for the user to customize the statechart to best suit the individual preferences. The interface is intuitive and easy to use.
When the slider is set to the lowest level, the child states are only very slightly above their parent states (Figure 6.12). As a result, the depth and thus the three-dimensionality of the statechart is lost. The shadows are also barely visible, which leads to a loss of depth perception. Also the *motion parallax effect* is hardly perceptible. All these factors contribute to the fact that the statechart appears almost two-dimensional, and

consequently the parent child relationships are no longer so clearly recognizable.
When the slider is moved to the maximum depth, the distance between the individual layers is very large (Figure 6.13). As a result, the three-dimensionality of the statechart quickly becomes apparent. In combination with the motion parallax, however, it can happen that if the user moves their head too much to the side, the parent states are no longer completely visible on the screen due to the slanted perspective. This can lead to a loss of information.
If the slider is in the middle range, the three-dimensional perception is always given and the statechart fits well into the image. This creates a pleasant user experience and all information can be displayed well. For this reason, the middle setting of the controller seems best for use.
It is good, but not absolutely necessary, that the user can control the depth between the layers. It can help to adapt the operation and visualization of the statechart optimally to the individual user.

7.4 Fog

Another effective feature is the fog. When this is activated, lower levels are slightly darkened (Figure 6.15). This greatly helps depth perception. Thus, the distances between the child and parent states become even more noticeable. The fog is usually only noticed subconsciously. It is also helpful that arrows whose start and end states are on different layers get a grayish color gradient in the depth. This makes their position and orientation in space clearer. Furthermore, the *depth controller* can also be used to change the fog intensity at the same time. If the controller is set to a large distance, the fog effect is also particularly strong. If the distance between the layers is set to very low, the fog effect has little effect on the image.

7.5 Blur effect

The *blur effect* offers the possibility to focus on certain layers in the scene. This can be helpful if only some nodes are of interest to the user at that moment. It also makes it clear which other nodes are on the same level as the selected node. Another advantage of this effect is that the relevant information is lucidly visible, but the states that are not important at the moment can still be seen blurred in the background. This way, the user always has an overview of the context and knows their position in the scene. Like in Figure 6.16, arrows with one end on the focused level and the other end on a lower level are still very sharp in the higher area and then slowly become blurred in

the lower area. This again contributes to a better understanding of the orientation of the arrows in space. Furthermore, this allows that it is still clear which arrows end or begin at the focused nodes.
However, if the mouse is moved over the statechart a lot, the constant refocusing of the camera can be perceived as disturbing. For this reason, it is important to be able to turn this feature off, or to provide the option to focus only on the particular layer when the user actively clicks on a node.

7.6 Outlining effect

With the *outlining effect*, one or more states can be highlighted. This has the advantage that the user is immediately given feedback for having just clicked on a state. In addition, it is also clear which state was clicked. Another good aspect is that it indicates which states are the child states of the clicked state by coloring them. This is especially important if the user is looking at the screen at an angle and the child states are not or only partially displayed above the parent states due to the parallax effect. In this case, the red frame provides a good indication that the child states belong to the examined state group (see Figure 6.19).
It is important to consider, however, that the user's focus is drawn to this frame by the animation of the *outlining effect* when it is activated. This could interrupt the user's flow while concentrating on another task. It can also be disturbing when the state was only clicked to move the camera position there. In addition, the increase in the amount of information could make the overall image confusing and thus detract from the user experience.

7.7 Frames effect

The *frames effect* emphasizes in which context the selected state is located. The frames highlight which states are the parent states of the selected state and where they are located. The short animation when generating the frames makes it easy to see which parent state they belong to. This feature is also useful in connection with the *motion parallax effect*. Since the generated frames are at the same height as the selected state, they do not shift in relation to the state when the user moves their head. This makes it easier to locate the parent states at deeper levels and to understand the shape of the statechart.
The two different frame variants (thin/thick) have different advantages. In the variant with the thin frames, more of the lower states are less occluded(Figure 6.21). This is particularly advantageous when the user looks at the statechart at an angle with the

motion parallax. In this case, the variant with the thick frames would cover large parts of the lower states, which could entail considerable loss of information. If, however, the perspective is not moved much and is mainly central, it is easier to recognize which parent state the individual frames belong to, since they are still touching their respective states.
With this effect, it should again be considered that the frames could lead to an overload of information for some users and thus even worsen the user experience.

7.8 Statechart execution

The statechart execution provides an understandable visualization to execute a statechart. The fact that the three buttons "pause", "rewind" and "forward" only become visible after the user presses "execute" is an advantage. This way, at the beginning, the user is not overwhelmed by an unnecessarily large number of buttons. It also makes it clear that these three buttons relate to the execution of the statechart.
During the execution of the statechart the active elements are highlighted and change their size. This pleasantly directs the user's focus to them. It also helps that all animation components are changed smoothly and not abruptly. Keeping the camera moving to the highlighted state or transition ensures that the area of interest is always well in view, while still allowing the user to maintain a good orientation in the space. By not starting the animation of the next step until the current animation is over, the execution order of the statechart remains clear. Even small details, such as the fact that when a state grows larger, the adjacent arrows move with it and thus always remain with their ends exactly at the edge of the state, are not directly perceived, but improve the user experience.
The choice between the two execution options is important to satisfy the different user needs. The execution variant, in which the transition to another state does not take any time, reflects the realistic execution. The other variant, in which the state transition is animated individually, takes longer to execute, but at the same time becomes more comprehensible. It depends on the user's preferences whether the realism or the exact comprehension is more important.

7.9 Combining effects

Some correlations and relationships have already been described in the evaluations. In the following, the advantages and disadvantages of the combination of various effects will be examined.
When all effects are activated and the user presses a state, several animations are

executed at the same time. First, the selected state and its child states turn red and get a red frame. Secondly, the frames of the parent states are generated and move to the height of the selected state. In addition, the camera moves to the state and focuses on this level. The result is shown in Figure 7.2. When the *blur effect* is deactivated it looks like in Figure 7.1. This amount of simultaneous information is difficult to understand for users who are not familiar with these effects. However, if the user already knows the effects well, a quick understanding is possible. It should also be noted that often the user's goal is not to understand all the information, but to pay attention to the effects that provide the information that is just needed. This is well possible with some knowledge about the visualization types.

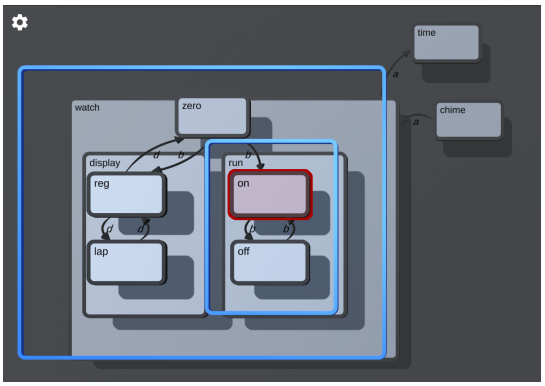

Figure 7.1: *Motion parallax effect*, *outlining effect* and *frames effect* activated at the same time.

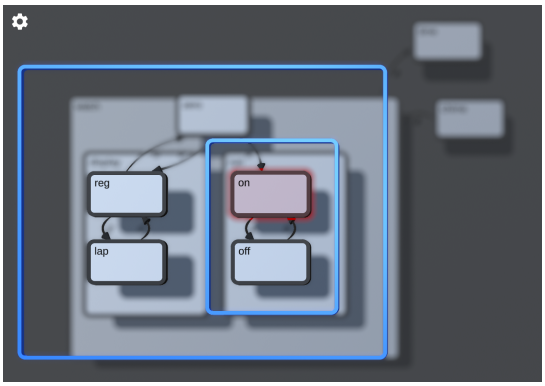

Figure 7.2: *Blur effect*, *motion parallax effect*, *outlining effect* and *frames effect* activated at the same time.

8 Conclusion

Considering the results, it can be seen that the interactive visualization of statecharts probably has great potential. Interaction methods such as the *motion parallax effect* can be used to make the three-dimensionality of the statechart more perceptible. In addition, effects such as the *outlining effect* and the *frames effect* can direct the user's focus to certain areas and thus, for example, emphasize the parent-child relationship of states. Also the *blur effect* can contribute to this. It is interesting to note that all the interaction methods and animations studied are useful in many cases, but not in all. This depends mainly on the needs of the user. For example, whether the parent-child relationships are important to recognize at the moment, or whether the user only wants to get a quick overview of the entire statechart. In order to meet these individual requirements, the option of switching the effects on and off separately in a menu is a very good solution. This functionality is also important for beginners, so that they can first get used to the effects.
To confirm these theories scientifically, further research is needed. For example, the *system usability scale* described in [Bro+96] could be used. This could serve as a robust and reliable evaluation tool to verify the advantages of the individual effects. It would also have the benefit of testing effectiveness, efficiency and satisfaction at the same time.

List of Figures

Bibliography

[Bau+06] P. Baudisch, D. Tan, M. Collomb, D. Robbins, K. Hinckley, M. Agrawala, S. Zhao, and G. Ramos. "Phosphor: explaining transitions in the user interface using afterglow effects." In: *Proceedings of the 19th annual ACM symposium on User interface software and technology*. 2006, pp. 169–178.

[BB99] B. B. Bederson and A. Boltman. "Does animation help users build mental maps of spatial information?" In: *Proceedings 1999 IEEE Symposium on Information Visualization (InfoVis' 99)*. IEEE. 1999, pp. 28–35.

[Bro+96] J. Brooke et al. "SUS-A quick and dirty usability scale." In: *Usability evaluation in industry* 189.194 (1996), pp. 4–7.

[BS90] R. Baecker and I. Small. "Animation at the interface." In: *The art of human-computer interface design* (1990), pp. 251–267.

[CU93] B.-W. Chang and D. Ungar. "Animation: from cartoons to the user interface." In: *Proceedings of the 6th annual ACM symposium on User interface software and technology*. 1993, pp. 45–55.

[Dra+11] P. Dragicevic, A. Bezerianos, W. Javed, N. Elmqvist, and J.-D. Fekete. "Temporal distortion for animated transitions." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2011, pp. 2009–2018.

[FH95] E. L. Ferguson and M. Hegarty. "Learning with real machines or diagrams: Application of knowledge to real-world problems." In: *Cognition and instruction* 13.1 (1995), pp. 129–160.

[Gon96] C. Gonzalez. "Does animation in user interfaces improve decision making?" In: *Proceedings of the SIGCHI conference on human factors in computing systems*. 1996, pp. 27–34.

[Gre18] J. Gregory. *Game engine architecture*. crc Press, 2018.

[Har87] D. Harel. "Statecharts: A visual formalism for complex systems." In: *Science of computer programming* 8.3 (1987), pp. 231–274.

[HR07] J. Heer and G. Robertson. "Animated transitions in statistical data graphics." In: *IEEE transactions on visualization and computer graphics* 13.6 (2007), pp. 1240–1247.

[Kru12] E. Kruijff. "3D user interfaces: theory and practice." In: *IEEE transactions on visualization and computer graphics* 18.4 (2012), pp. 565–572.

[Lar+96] A. Large, J. Beheshti, A. Breuleux, and A. Renaud. "Effect of animation in enhancing descriptive and procedural texts in a multimedia learning environment." In: *Journal of the American Society for Information Science* 47.6 (1996), pp. 437–448.

[Pfa02] J. D. Pfautz. *Depth perception in computer graphics*. Tech. rep. University of Cambridge, Computer Laboratory, 2002.

[PG92] O.-c. Park and S. S. Gittelman. "Selective use of animation and feedback in computer-based instruction." In: *Educational Technology Research and Development* 40.4 (1992), pp. 27–38.

[RG00] O. Radfelder and M. Gogolla. "On better understanding UML diagrams through interactive three-dimensional visualization and animation." In: *Proceedings of the working conference on Advanced visual interfaces*. 2000, pp. 292–295.

[RG79] B. Rogers and M. Graham. "Motion parallax as an independent cue for depth perception." In: *Perception* 8.2 (1979), pp. 125–134.

[RK91] Y. Roos and M. Karel. "Applying state diagrams to food processing and development." In: *Food Technol* 45.12 (1991), pp. 66–68.

[TMB02] B. Tversky, J. B. Morrison, and M. Betrancourt. "Animation: can it facilitate?" In: *International journal of human-computer studies* 57.4 (2002), pp. 247–262.

[TS03] U. Thaden and F. Steimann. "Animated UML as a 3D-illustration for teaching OOP." In: *7th Workshop on Pedagogies and Tools for Learning Object-Oriented Concepts in 17th European Conference on Object-Oriented Programming, Darmstadt, Germany, Springer*. Citeseer. 2003.

[Von25] H. Von Helmholtz. *Helmholtz's treatise on physiological optics*. Vol. 3. Optical Society of America, 1925.