Technical University of Munich
Institute for Computer Science

Fraunhofer Einrichtung Experimentelles Software Engineering

UNIVERSITÄT KAISERSLAUTERN
AG SE
FACHBEREICH INFORMATIK

Hauptseminar

Modelling of the Tamagotchi with AutoFocus

Adaptation Alexander Seitz,
Thomas Seliger,
Florian Kunkel
Maintenance Dr. Bernhard Schätz
Set of Issue Prof. Dr. Manfred Broy
Date of Speech 02.02.1999

Abstract

We have modelled the Tamagotchi with AutoFocus.

Contents

Equal goes it loose.
(Lübke)

1 Intorduction

1.1 Circumstances

We are one of eight groups modelling the Tamagotchi — each with another CASE[1] tool.

Except some basic knowledge about concepts of Software Engineering we did not know anything about our issue. We knew that there must be something and people call it CASE and mean Computer Supported Software Engineering but we had never seen a CASE tool and — of course — we did not know how many windows you have to deal with when you work with a CASE tool.

1.2 Purpose of the Notation/Method

> ”The purpose of AutoFocus is to aid software developters in specifying and desingning distributed systems, especially so-called ’embedded systems’.”
> (out of Autofocus Users Manual)

1.3 History of Development, Authors

> ”AutoFocus was developed and implemented during the one-semester practical course ‘Software Engineering’ by students and members of the chairs Prof. Broy and Prof. Endres at the faculty of computer science at the Technische Universität München in summer 1996. It will be used and enhanced in several cooperation projects with Bavarian industrial companies, including Siemens Public Networks.”
> (out of http://autofocus.informatik.tu-muenchen.de/Infos/afinfo-en.html)

1.4 A Survey of the Notation/Method

1.4.1 Conceptual Model of the Notation/Method

With AutoFocus you can model four different views on your system, accessible from a modelling center (figure 1): SSDs, STDs, EETs and DTDs (see table 1 and figures 3, 4, 5).

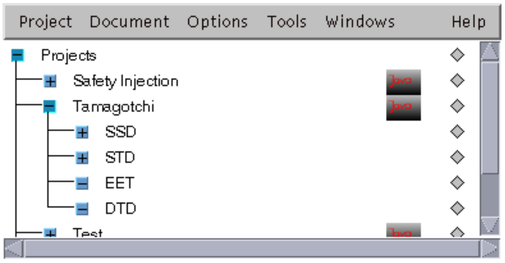
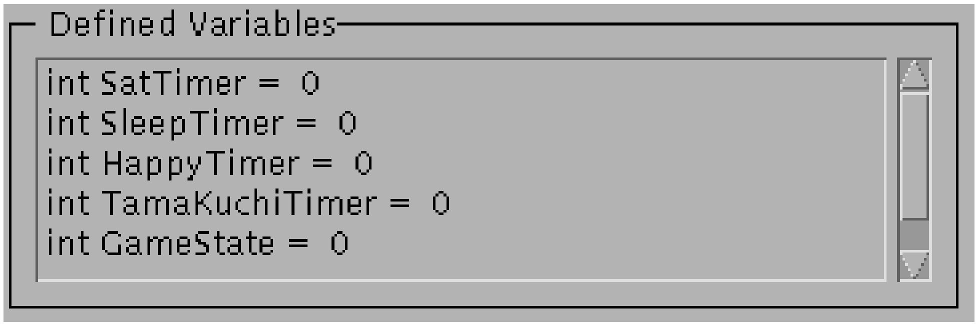


Figure 1: Modelling Center

[1] **CASE:** Computer Supported Software Engineering

SSD	(System Structure Diagrams) to specify the static communication structure of a distributed system as a network of interconnected components. An SSD can either have a sub-SSD or an STD (see below). With the sub-structure you can build up your system in a hierarchical way to not lose your grip on it. All the system's components described by the SSDs work in parallel. An component can have a set of properties (figure 2) that can only be accessed by its own STD, not even by the component STDs of its sub-SSDs.
STD	(State Transition Diagrams, or automata for short) to model the behaviour of a system's components (i.e., the state transitions dependent on the data a component reads). Again, you can use hierarchical technique on it: every state can be splitted up in a sub-STD. Technically this sub-STD is on the same level as its parent and has access to the same properties.
EET	(Extended Event Traces, see Message Sequence Charts) to describe the behaviour of a (part of a) system from a view based on the communications history between several components. EETs can be automatically created while simulation of a component.
DTD	(Data Type Definitions) to describe data types, and some functions working on them. DTDs are a textual description technique.

Table 1: Views in AutoFocus



```
int SatTimer = 0
int SleepTimer = 0
int HappyTimer = 0
int TamaKuchiTimer = 0
int GameState = 0
```

Figure 2: Properties of a component

1.4.2 Methodology, HOWTO use AutoFocus

You are not forced to follow any given sequence, but in the current version of AutoFocus it is impossible to connect a sub-structure to its parent without designing the parent first. But you can very well assign an STD to its SSD in any situation.

Anyway, a recommendable order of constructing views is to design the top level SSDs first, think about their responsibilities and connect them with channels according to the information to be exchanged.

If you allready have use cases, you can describe their information flow with EETs and avoid forgetting important functionality. As soon as your components get their behaviour, AutoFocus can check if the described system can perform your use cases.

Then you should try to devide the responsibilities of each component into sub-SSDs as long as you find possibilities to make them work in parallel. Be aware that in the current version of AutoFocus you can only simulate components with sub-SSDs but not with an STD assigned to the component itself. However, you can set a pseudo component between the component to be simulated and its STD. It means stupid work, but then you can simulate and debug your SSD.

The behaviour of every component which has no sub-SSD is to be designed now with

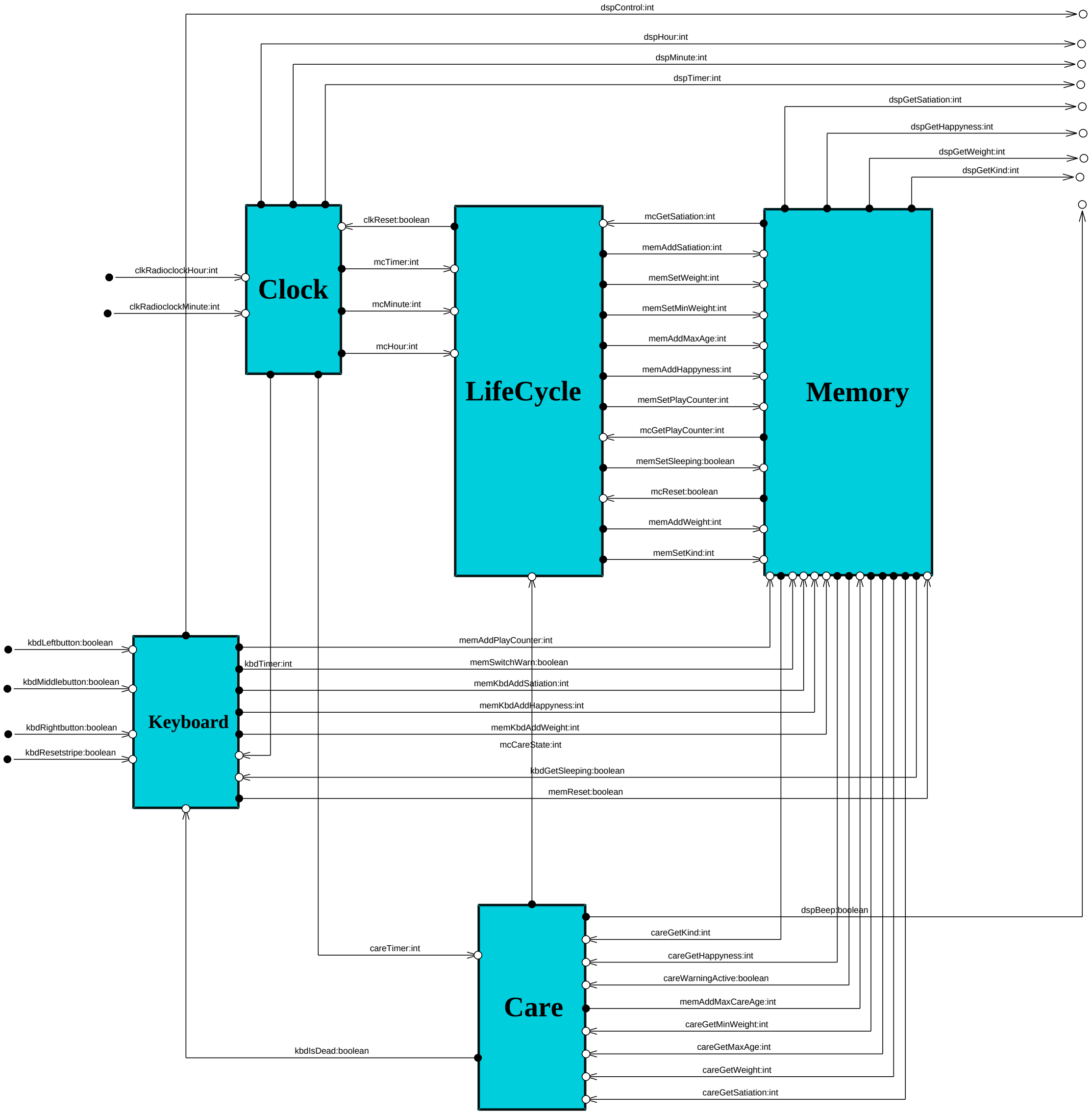


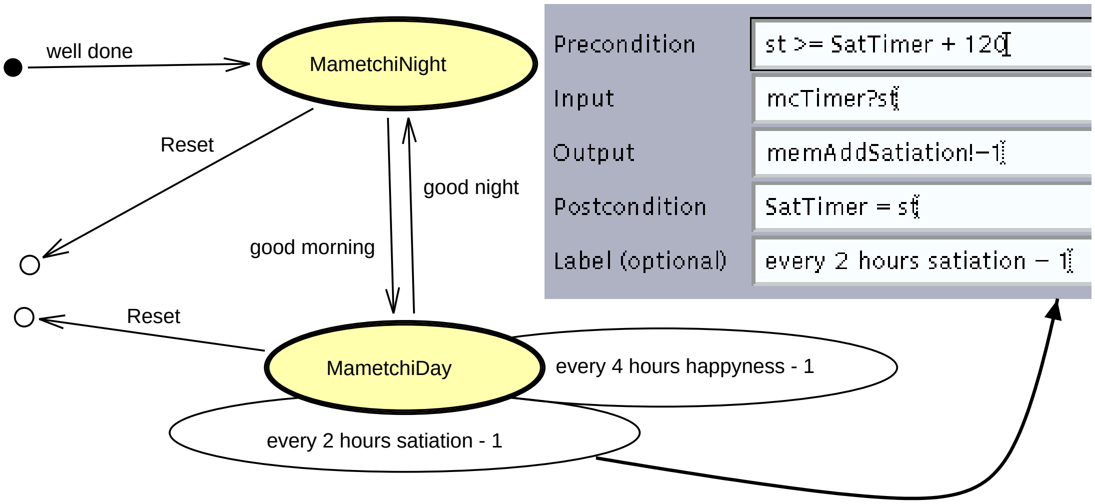
Figure 3: System Structure Diagram

an STD. Make sure that this STD is easy to understand. Collect similar states in a single state and describe them in its sub-STD. It helps to oversee your STDs.

When you are ready with modelling a component with all its sub-components and their functionality, you can simulate it. Try to simulate as soon as possible to make sure that you cover as many use cases as possible. This is another reason to make the components become easy.

1.4.3 Modelling

There are no guidelines in the manuals respectively texts of AutoFocus describing, how things of the real world should be represented in the notation's concepts (e.g. how to represent states). But in consequence of the simplicity that is given by only four different kinds of views it should be possible to understand the idea behind the notification without a tutorial. However, a complete list of the tool's special operating needs could help a lot.



Precondition: st >= SatTimer + 120
Input: mcTimer?st
Output: memAddSatiation!−1
Postcondition: SatTimer = st
Label (optional): every 2 hours satiation − 1

Figure 4: State Transition Diagram

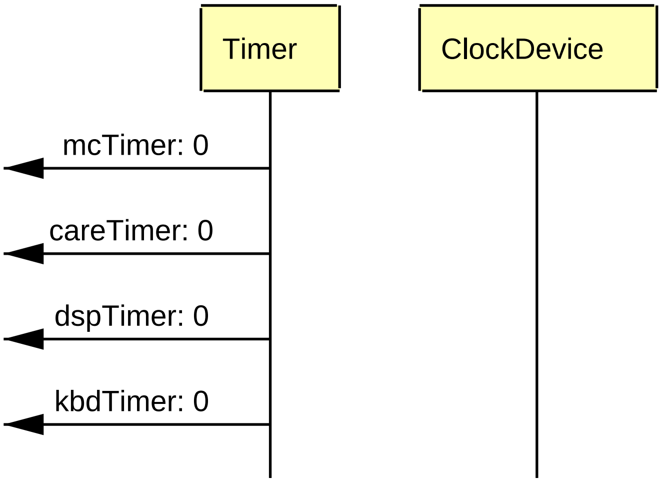


Figure 5: Extended Event Trace

1.5 A Survey of AutoFocus

1.5.1 Basic features of AutoFocus ...

Simulation. As long as you only use standard java data types and java based state transitions you can simulate your system in a mighty simulation environment. Interactivity is provided by the user beeing able to send data into the input channels of the simulated component. Both the state transitions and the activities and values of the channels can be watched in their according diagrams during simulation and in real time. An EET that represents the communication history of the simulation can be created automatically. It shows all the direct sub-components of the simulated component and, of course, the communication between them.

Consistency-Check. We have not used it and therefor we do not have any experience with it. (...)

Interface to the Tool's Environment. Just as Consistency-Check we did not use it, but there had been a project before in which AutoFocus had been commanding a real existing 'Fischertechnik' model of an elevator system.

Platform Independence. AutoFocus is completely written in Java, i.e., you can use it on any platform you can find Java for. However, the AutoFocus server needs a UNIX environment.

Multi User Capability. AutoFocus is designed to be a client server architecture. An arbitrary (is it really arbitrary?) number of developers can work on a project at the same time. Each view of the project that one works on is locked for the others. The server should allways be started within the same UNIX account due to access rights of the data storage files. Unfortunately it is not recommendable to develop parts of the project without connection to the server, because diagrams cannot be transfered between different projects well enough. Maybe this restrictions will disappear when the data storage of AutoFocus will be reengineered using a database management system, which is planned.

2 Apporach and Result

2.1 Which Concrete Strategy has been Followed with Regard to Modelling?

First of all we discussed the top level components and their basic communication channels drawing everything on a sheet of paper and made a hardcopy of it for all of us.

Then the LifeCycle and the Clock were designed in parallel. That was possible due to the little amount of communication between them.

After development of the LifeCycle had been proceded far enough to exactly know which functionality the Memory should provide we started to design this and also the Care component which is very closely connected to the Memory. Due to that Memory and Care were created by the same developer.

The Keyboard component was designed time-independently but with very active coordination to the development of Memory.

2.2 How have the Requirements been Devided and how have the Interfaces been Defined?

All of us were responsible for one or two components each (see table 2).

Alexander Seitz	LifeCycle and AutoFocus learning
Thomas Seliger	Clock and Keyboard
Florian Kunkel	Memory and Care

Table 2: Divide and Conquere

AutoFocus supported us by providing multi user capability. We had to work in the same room at the same time but were able to work on our components in parallel.

2.3 Scope/Precision of the Specification

Did we model all of the requirements exactly? No, we did not. There is no display. The Memory does not check its values (e.g., satiation can be greater than 4). Our Tamagotchi cannot starve. If the Tamagotchi's properties are not in their green ranges, there is a signal every minute. Not every state transition is deterministic. The timer (Clock) is not more detailed than one minute.

2.4 Did we Already Simulate while Modelling?

Yes, we did. Every component has been simulated separately. AutoFocus did a very good job according to this.

2.5 Verification/Validation of the Specification

2.5.1 Is the Specification Consistend and Complete?

Not completely (see above).

2.5.2 Has the Specification been Reviewed Completely According to the Requirements?

Due to the inconsistencies of the requirements itself, it was not really easy to make sure that every requirement is satisfied. But we tried to fulfil them all.

2.5.3 Has the Specification been Tested Completely, Supported by the Tool and According to the Requirements?

Testing all the requirements manually was nearly impossible, because we could not save the state of our Tamagotchi's life. If we wanted to do so, we would have had to play the game again and again. The only thing we did was playing several times to make sure that it works at all.

2.5.4 Statement about Problems with Consistency Check, Review or Simulation

We had not the time to work with the consistency check, but AutoFocus supports review and simulation quite well.

2.5.5 Has the Specification been debugged?

In consequence of simulating every single component with sub-components we debugged during development and of course we did fix it whenever we found a bug. Exceptions are found above.

2.6 Size of the Specification

We designed 6 SSDs and 33 STDs.

3 Specification of the Funktionality "Playing"

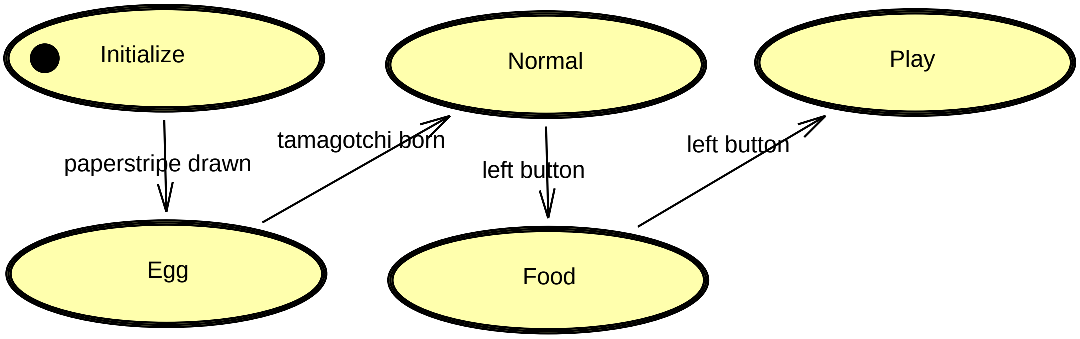


Figure 6: simplyfied menu-logic

The State-Transition-Diagram figure STD 6 shows the simplyfied menu logic. The full STD can be found in the appendix. The automaton starts at the initial-state "Initialize" and waits until the paperstripe will be drawn. If this is done, the automaton has to stay five minutes in the "Egg" - state before the state "Normal" is achieved.

To get to the state named "Play", the user must press the left button twice. This activates the two transitions labeled "left button". The state "Play" contains a sub-automation, which handles the complete game.

This sub-automaton shows figure STD 7. It continues at the state "PlayMain" which is the target of the second transition labeled "left button" transition coming from the superior "Play" state. The actual game starts, if the user presses the middle button.

The responsible transition attributes looks like this:

PlayMain $\Longrightarrow$ NewRound **start game**	
Input	reset?, kbdMiddlebutton?true, kbdTimer?x
Output	dspControl!41
Postcondition	Round=0; RoundsWon=0; Savetimer=x

Start game will be activated, if no reset signal is present and the middle button is pressed (signal from "kbdMiddlebutton"). The actual time value x is stored in the "Savetimer" variable and will be used as timeout-timer ("kbdTimer" is always valid). x is called a transitions-local-variable and is only present in this transition. It's used as a temporary memory to read the signal from the port "kbdTimer" and store it in the SSD-local-variable "Savetimer". To show a playing tamagotchi on the the display, the constant 41 is sent over "dspControl". At last the variables "Round" and "RoundsWon" are set to zero.

To leave the game, the user has to press the R-key ("about game"). It will also be left if the timeout occours ("time passed"), the tamagotchi starts sleeping ("starts sleeping") or the user pushes the paperstripe ("reset").

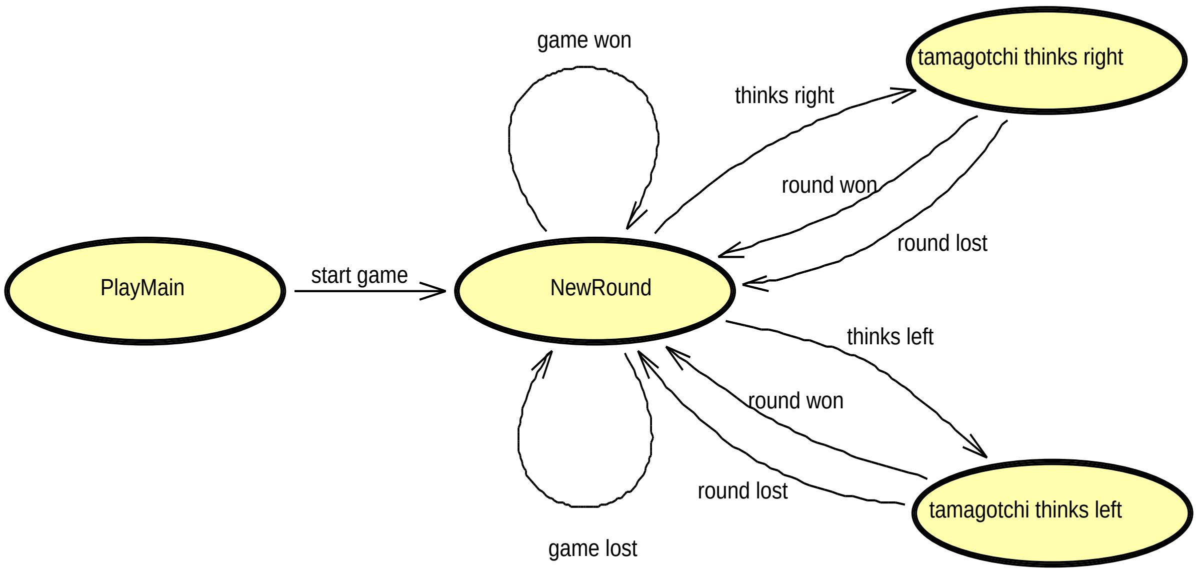


Figure 7: simplyfied game

NewRound ⟹ tamagotchi thinks left **thinks left**	
Precondition	Round<5
Input	reset?, kbdGetSleeping?false
Output	dspControl!41

NewRound ⟹ tamagotchi thinks right **thinks right**	
Precondition	Round<5
Input	reset?, kbdGetSleeping?false
Output	dspControl!41

Now we are at "NewRound" and the tamagotchi is looking left and right (the display component is also responsible for beeping). At first the tamagotchi selects a direction. This is done by the two transitions "thinks left" and "thinks right". Both transitions have the same pre- and input-condition. Because of this situation is not deterministical AutoFocus randomizely selects on of the two transitions. The the display receives again 41 because the use may not yet see, what the tamagotch is thinking and the automaton changes into the "tamagotchi thinks right" or "tamagotchi thinks left" state. Now its the users turn to guess the direction the tamagotchi is thinking of.

tamagotchi thinks left ⟹ NewRound **round won**	
Input	reset?, kbdMiddlebutton?true
Output	dspControl!44
Postcondition	Round++; RoundsWon++

Supposing the tamagotchi thinks left, then we are in the state “tamagotchi thinks left”. If the user selects the middle button (signal from “kbdMiddlebutton”), then the transition “round won” will be activated because the users choice is wrong and the tamagotchi has won this round. The display receives the constant 44, which stands for a playing tamagotchi looking left happily. The variables “Rounds” and “RoundsWon” in the post-condition are both increased by one.

tamagotchi thinks left ⟹ NewRound **round lost**	
Input	reset?, kbdLeftbutton?true
Output	dspControl!45
Postcondition	Round++

When the left button is pressed by the user, he has guest tamagotchis choice and the round is lost from tamagotchis point of view. Therefore the transition “round lost” is activated. The display receives now the value 44, which stands for a playing tamagotchi looking left sadly. “Rounds” is increased by one and “RoundWon” stays equal. Again we are at the “NewRound” state and round two starts.

NewRound ⟹ NewRound **game won**	
Precondition	Round==5 && RoundsWon>2
Input	reset?, kbdTimer?x, kbdGetSleeping?false
Output	dspControl!41, memKbdAddHappyness!1, memKbdAddWeight!-1, memAddPlayCounter!1
Postcondition	Savetimer=x; Round=0; RoundsWon=0

NewRound ⟹ NewRound **game lost**	
Precondition	Round==5 && RoundsWon<3
Input	reset?, kbdTimer?x, kbdGetSleeping?false
Output	dspControl!41, memKbdAddWeight!-1, memAddPlayCounter!1
Postcondition	Savetimer=x; Round=0; RoundsWon=0;

The procedure described above is repeated for rounds two to five. If five rounds are over (the variable “Round” has reached the value 5), the game is over and the transition “game won” or “game lost” is activated. The game is won if at least three rounds of five are won (the games the tamagotchi has won are stored in “RoundsWon”). In this case the happyness is increased by one and the weight is decreased by one. This is done by sending 1 to “memKbdAddHappyness” and -1 to “memKbdAddWeight” which adds and subtracts the givel value. Instead of a Get- and Set-function-pair for each memory-port, this method has the advantage that more than one module can alter the values at the same time whithout interfering with each other.

The signal “memAddPlayCounter” is used for the lifecycle component and has no effect on the game. “kbdGetSleeping” cares, that a new game can only be started, if the tamagotchi is awake. This signal is, like “kbdTimer” and the memory-signals always present. At least the variables “Rounds”, “RoundsWon” and “Savetimer” are again initialized for the next game and the display is directed to show a tamagotchi looking left and right (value 41 to port “dspControl”).

3.1 Realisation of the informal demands

Overview and SSDs To get a structure overview the tamagotchis components are modelled through System-Structure-Diagrams (SSD) and Sub-System-Structure-Diagrams (Sub-SSD). Each component has its own funktionality, which implementation is hidden in this view. Behind some Sub-SSDs like the "Memory" are separate Sub-SSDs which give again an overview of this spezial component (the "Memory" is devided in memory-cell Sub-SSDs).

The components are connected to each other, so it can be seen which component sends signals to another component. For example, the "LifeCycle" has to alter many variables in the memory. Because the signals are one-way, two ports are necessary, one for reading and one for altering a variable.

Functionality and STDs If no refinement is necessary, the proper funktionality of the (Sub)components is spezified in State-Transition-Diagrams (STD) and their Sub-STDs. The memory-cells in "Memory" are all single automata, which handle the Get- and Add-operations.

The individual states of the tamagotchi are represented in the states of the State-Transition-Diagrams. A good example is the "LifeCycle" diagram. For each life-period exists an own state. Some of this states consists of an own easy to read Sub-State-Transition-Diagram. If the tamagotchi is a "Mametchi" and is sleeping, the state "MametchiNight" is active.

Timer Another important functionality is the timer. In other CASE-tools, there is an explicite timer provided by the CASE-tool. In AutoFocus there is no such timer, which can easily be used. The timer in our tamagotchi is implemented in the "Clock"-component. In this component there is a clock, to count minutes and hours, and a timer, which counts only minutes. The timer is implemented with a SSD-local variable, which is increased in every step by one minute. The timer-signal is sent in every step, so a component can read a valid timer-value at any time needed.

The same technique is used by the "Memory". Its easier to send the actual memory-cell-content at every step instead of doing complex synchronisation, because the other components can use the memory at every time they want.

4 Experiences

4.1 Training Period

The first project we have done with AutoFocus was the Safety-Injection-System. This simple project took approximately 16 hours work in, design and completion. There are some difficulties in learning AutoFocus. With AutoFocus comes only a short and simple documentation which describes only the concepts and the menu-structure of the program. For example, is it descibed that there are some transition-attributes and how they are named. But what you have to fill in and which possibilities there are is missing. You have no change to guess yourself how it works if there is nobody who can help you. But god thanks we had someone (thanks to Barnie) who could explain us the hidden features of AutoFocus.

If we had overcome this hurdle the work in was fast and easy. But the user interface handling is to complicated and not intuitiv. For example, the attributes-dialog, which is used very often, is only reachable in the menu and has no key-shortcut. And the window positions are not saved. This is especially annoying in the simulator, where many windows must be opened every time.

When we worked with AutoFocus there were 3 new versions coming out. On the one side, it's very good, because more functions were implemented or worked more correct, but there is no history list, where the changes are noticed. Therfore we had to test every new update. The greatest advantage was, that we could simulate each component alone. One big problem we had, that if we worked at home with the linux port from AutoFocus and wanted to merge our work together to one project, was not possible, because the cut/copy/paste and the import/export functions did not work correctly.

At the beginning we didn't realize the cycle-concept correctly. If you want to catch all possibilities you have to take every port at every cycle in a STD into consideration. Even the possibility that no signal is present at one port. The more ports there are, the more complex are the transition-conditions. We minimized the problem thereby we defined some signals, that are present at every cycle (like the timer and memory signals).

4.2 Tamagotchi implementation duration

The complete develpment took about 140 hours. This time can be devided into four parts.

First we made a rough structure of the tamagotchis components and how they have to interact with each other. Then we defined the components interface, which channels have to be used and the name and type of the ports. Half of this work was not done with AutoFocus, but on paper. Approximately 15% of the time were required for this task.

The components modelling and implementation has been done in approximately 25% of the time. This means drawing the System-Structure-Diagrams, the State-Transition-Diagrams and the transition attributes.

Testing and debugging with the help from the simulator has been taken approximately 10%. Correction of our own errors included.

The remaining 50% has been spent to the problems with AutoFocus. This time is divided up to the points mentioned above. It's not easily possible to assign the problems to one phase of our work. Out problems with AutoFocus were conditioned by some bugs, the complicated user interface, incomplete documentation and only partial implementations of functions, which we noticed unfortunately later at work that they are not complete. Sometimes the clients and sometimes the server stopped working and we had to restart AutoFocus. Not only once it has happend, that the last work we have done was lost and we had to do it once again. Another time, AutoFocus disconnects the memory-Sub-SSD, which is very complex, from its hihger SSD and there is no way in AutoFocus to reconnect an existing Sub-SSD. Because this component took a long time to create, we searched for another way to repair it. We analyzed the files in the repository and altered manually the internal database structure.

4.3 Modelling the Tamagotchi

Notation The representation of the System-Structure-Diagrams ans State-Transition-Diagrams is very clear and easy to understand. The diagrams are not overloaded and there are no mystical signs. Everyone who is familiar with automata can understand the graphics and how it works. The clearness is among other things achieved with the help of sub-diagrams. Every component in a SSD can have an own SSD inside. So you can only see what is neccesary and the details are hidden. The same technique is used for states in STDs.

A disadvantage is, that the more sub-diagrams you have, the more windows you have to open, if you want to edit something or simulate your modell. If you don't assign transition labels, all attributes are displayed in one row. It would be better to have a well-formated, multiline string, so you can easily see the conditions of the transition, and which variables are affected.

Another disadvantage in SSDs is the amount of channels between the components. For every signal in every direction an own channel is necessary. If there were bidirectional channels or a bus-system, many things could be done easier and the clearness of the reprentation would raise.

Starting Problems We had some problems to understand the mechanism behind transition-local-variables, SSD-variables, ports and their combination. Here is an example:

KuchitamatchiKuchitamatchiNight $\Longrightarrow$ KuchipatchiKuchipatchiNight not good	
Precondition	T > NextStateTimer+7200 && C <= 2
Input	mcTimer?T, mcCareState?C
Output	dspMCState!14, memWeight!20, memMinWeight!20, memAddMaxAge!12
Postcondition	SatTimer = T; HappyTimer = T

In this transition there are two transition-local-variables ("T" and "C"). First the Input-line is executed. If the Input-line is true, a signal from the port "mcTimer" and a Signal from "mcCareState" must be present. The values are stored in the transition-local-variables "T" and "C" and the Precondition-line is evaluated. There the values of "T" must be greater than "NextStateTimer+7200" and "C" must be less or equal than 2. If all these conditions are met the transition is activated and the signals in the Output-line are sent. At last, the value of the transition-local-variable "T" is stored in the SSD-variable "SatTimer" and "HappyTimer".

Secondly, the visibleness of the SSD-variables is not obvious. These variables are only visible in one SSD and not in its Sub-SSD.

These problems we had would be no problems, if they were declared in the manual.

Tool A very positiv feature of the actual version from AutoFocus is the possibility of simulating individual components. This is helpfull for complex systems, where many components are put together to one big system. A restriction in AutoFocus is, that a component (SSD) can only be simulated separately, if this component has sub-components. Again one thing, that is not mentioned in the manual and that we found out by chance.

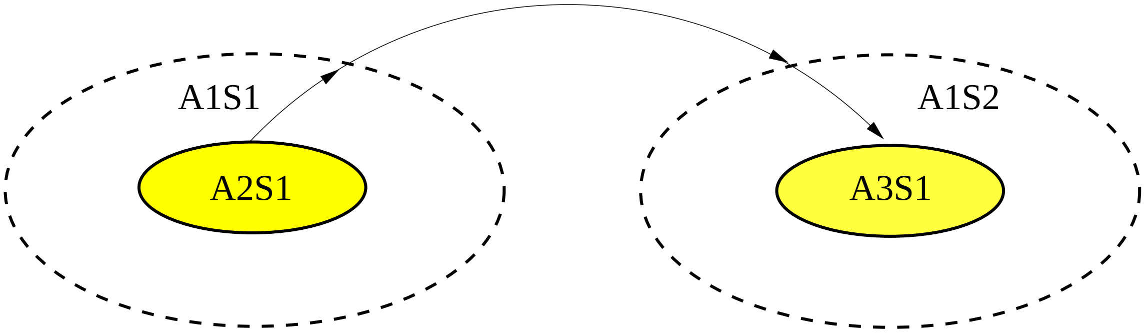
Another feature are Consistency-Checks. There are checks for garanteeing that all channels are connected, there are no empty ports, etc. Own checks can also be defined. We didn't use Consistency-Checks, because they didn't work correctly in all the version of AutoFocus we used.

Later changes in the strucure of the modell are complicated. One example: If you have two states (A1S1, A1S2) in an automaton A1 and these state have both a sub-automaton (A2, A3). A state (A2S1) in sub-automaton A2 is connected with a state (A3S2) in sub-automaton A3, then you have three transitions with the same transition-attributes, although it is logically the same transition:

1. from state A1S1 to A1S2

2. from state A2S1 to A1S1

3. from state A1S2 to A3S1



If you want to change e.g. the Input-condition, you have to alter all three Input-conditions. All three attributes must exactly be the same, because AutoFocus recognices these three transitions as one transition through the textual attributes entered in the corresponding dialog-box. But, it is planned, to give AutoFocus an object-oriented structure, so this problem won't be existant in the future any more.

Some more problems caused the non-existent error-check. In the dialog-boxes you can type in what you want and AutoFocus accepts it. To check, if your input is correct, you have to simulate the component. And if you don't work on a fast machine, the preparations for the simulation (create java-classes, compile code and execute it) is too long. Especially if you are a beginner in AutoFocus and e.g. are unsteady in the syntax of the transition-attributes, which is not described in the manual, it would be a great help, if the input would be check for correctness.

Errors are hard to find, because to find them, the component has to be simulated. The simulator first generates java-classes from the SSDs and STDs and then compiles the generated code. If something is wrong (e.g. channel-names, transition-attributs), you get many error-messages from the java-compiler. From this messages, you have to conclude where the real errors are. Sometimes, it's necessary to look into the generated java-code, to locate the error and find the cause.

5 Conclusions

5.1 Teamwork with autofocus

AutoFocus comes with a good teamwork support. After we got used to some peculiarities of AutoFocus the teamwork features were saving a good amount of time. AutoFocus Server architecture is client/server based, so we could start one server and different people could work on different parts of the specification at the same time using mutiple AutoFocus clients.

To get the most out of distributed teamwork we had divided the specification into a few subcomponents. These subcomponents were connected to each other by channels, so we could start to work distributed after specifying the gerneral interface (the names and types of the interconnecting channels).

If a clients accesses a part of the specification (e.g. an certain sytem structure diagram), this part is locked and no other client may alter this part. Locking of a document is shown in the project browser. Also a locked document can be viewed in read only mode by other client users, while it is edited. After the user of the client closes the edit window, the document gets editable by other clients again.

The latest version of AutoFocus we used also supported simulation of subcomponents correctly. This is a very usefull feature, as we could test small subcomponents for bugs. This reduced the possible errors and made it possible to simulate parts of the tamagotchi, although essential parts of the tamagotchi were not realized in AutoFocus at this time.

Although the teamwork support of AutoFocus is already very good, there are some peculiarities that costed time: The server should be always started by the same person, as the server runs with the user identity of the user that started it. If the server is started by another user later there is a chance that one ore more AutoFocus document file cant be read by this user. Also the client had to be restarted after deleting two documents out of a project.

5.2 Subsumption of Autofocus strengths and weaknesses

AutoFocus has its strengths and weaknesses. We appreciated the visualisation of the specification. The graphs of the system structure diagrams and the state transition diagrams look good, with only a few ”styling” mouse clicks. You get a image of what the system does in a short time. Also AutoFocus uses only a few graphical elements (states and state transitions, structures and channels) and thus the graphs are not ”visually overloaded”.

A further big advantage of AutoFocus is that you can do a quite raw design of your specification at first. You can assign general behaviours to components to get a superficial realisation. Then you can ”deepen” the specification by assigning subcomponents to components. This works also with state transition diagrams, where you can assign sub-states to existing states.

Also AutoFocus had some nasty weaknesses: Signals that are required by a lot of components (e.g. the timer of the tamagotcho component clock), have to be passed to each component by a seperate channel. This can lead to a confusing visualisation. Another feature is the lacking of some timer objects. It is hard to realize realtime systems, because all you can do is a clock that is step based. The speed such a clock is based on the performance of the simulation computer you use. The menus of AutoFocus didnt support hotkeys for operations that were needed often and editing the diagrams was partly very unintuitive and the cut and paste functions didnt work properly. Also it was sometimes very annoying to alter a part in a diagram, which has a substructure assigned to it. Alterations arent passed to referencing parts automatically, it has to be done manually.

5.3 Proposal of how to improve Notation, Tool and AutoFocus

After specifying a rather complex system, like the tamagotchi, in autofocus, we have a few proposals how to improve and expand the method and the tool.

The Method should be expanded with two new features: a reset signal for the automatons and a bus object. A reset signal is very usefull for big state transition diagrams: if there is a reset, the reset signal is set only for one clock step. So you have to check for a reset signal in every state of the automaton and you have an reset transition out of every state of it. While is is correct from the automtaton therory point of view, it makes a large state transition diagram confusing. It would icrease the readability of such a diagram, if you could define one reset-signal for this automaton, which when it is triggered put the automaton into a certain state after executing a certain transition. The other valuable expansion would be a bus object. If a signal is needed by a lot of components, it has to be passed to every component by a seperate channel. This can render a system structure diagram rather confusing. With the help of a bus object you could pass the signal to all components which are connected to this object.

As the tool is still under developement, we have a lot of proposals what should be included or expanded: First of all the most important thing that AutoFocus lacks is a real documentation. It should contain general instructions for the tool and a known bug list. Also it should include multiple small and easy to understand examples, sorted by theme. It would save much time if you could look for a how do i put a variable value into a channel-example, instead of endless trial and error or hidden feature searching. Maybe this can be done in the style of a tutorial with several lections.

Another important issue is the way AutoFocus manages and saves the data. By now it saves data document based. Many problems could be solved if data storange is done object orientated e.g. changes of variable or channel names could be passed to referencing objects automatically or already existing substructures can be assigned to a new structure. Also it would eliminate an other problem we had. Transitions are referenced by its whole data, including the test description: so if a transition in a substructure has only one space character more in its text description, than its equivalent it the structure above, this would lead to a error in the sumulation.

Furthermore AutoFocus should do some error checking directly after entering data. This should be done for variable, channel and port names. So that one can eliminate typos immediately without trying to compile the whole specification every time.

5.4 Usefulness of the Modelling and AutoFocus

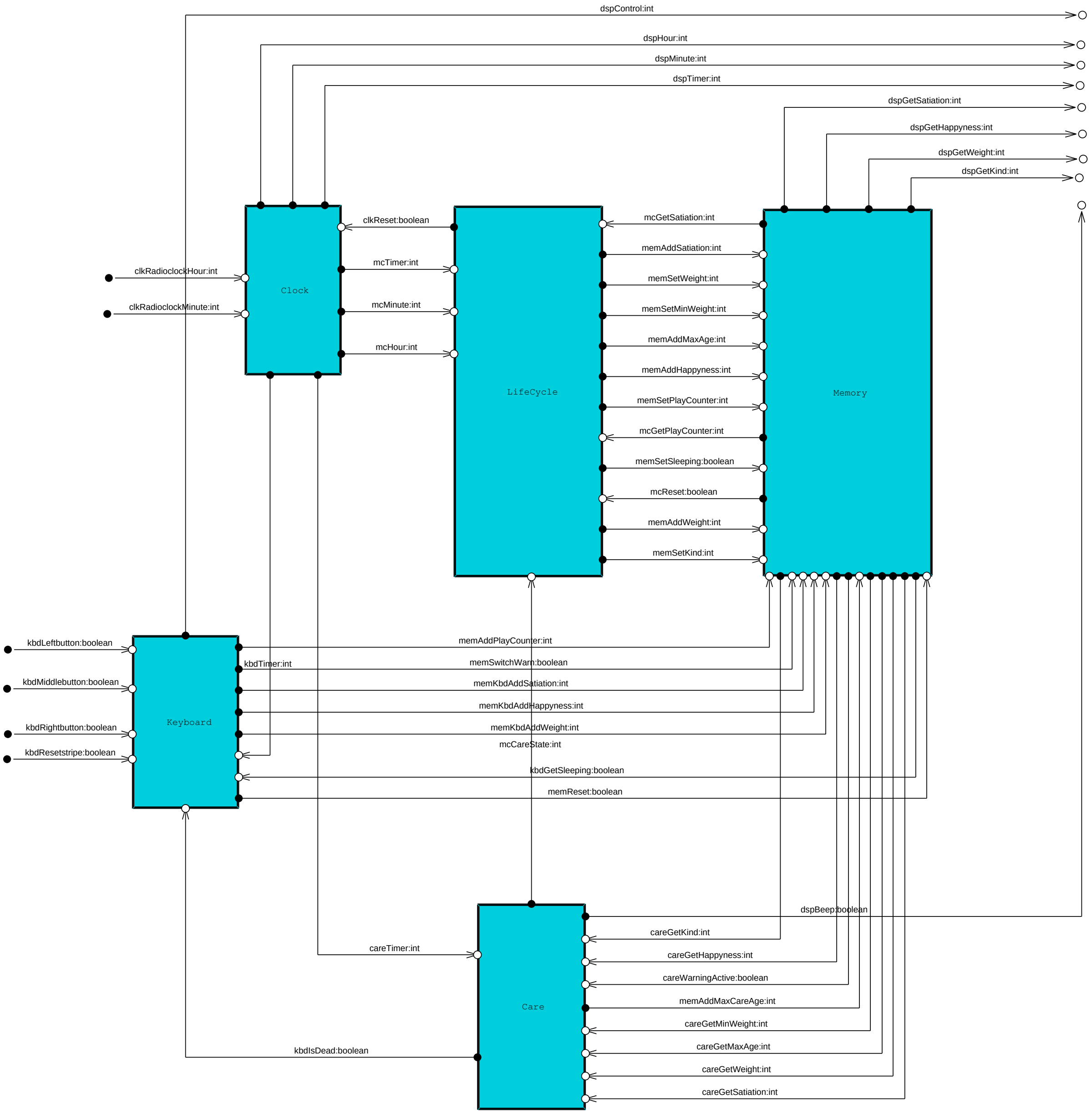
We think that using a case tool to develope the tamagotchi consumed more time than just to do a general specification and then code it in a programming language. One of the advantage of doing the tamagotchi with a case tool and as a team was, that you look into the specification more often, to ensure that you are doing right and that your parts still fits to the pieces that are done by your teammates. In general we would say, modelling the tamagotchi with autofocus led to a better understanding of the specification. Another advantage is, that a specification that is modelled with a case tool, is much more easy to understand that just code. Once you get used to a modelling type and a tool, the benefits will usually surpass the disadvantages. The last big advantage was the simulation environment, which enabled us to test sub parts of the specification for bugs, before testing the whole specification. As a resume we could say that using a flowcharter to develope the modell of the system and then programming it may have been faster. But for testing the system for correctness, and make it understandable for others AutoFocus did a good job.

5.5 Use AutoFocus again?

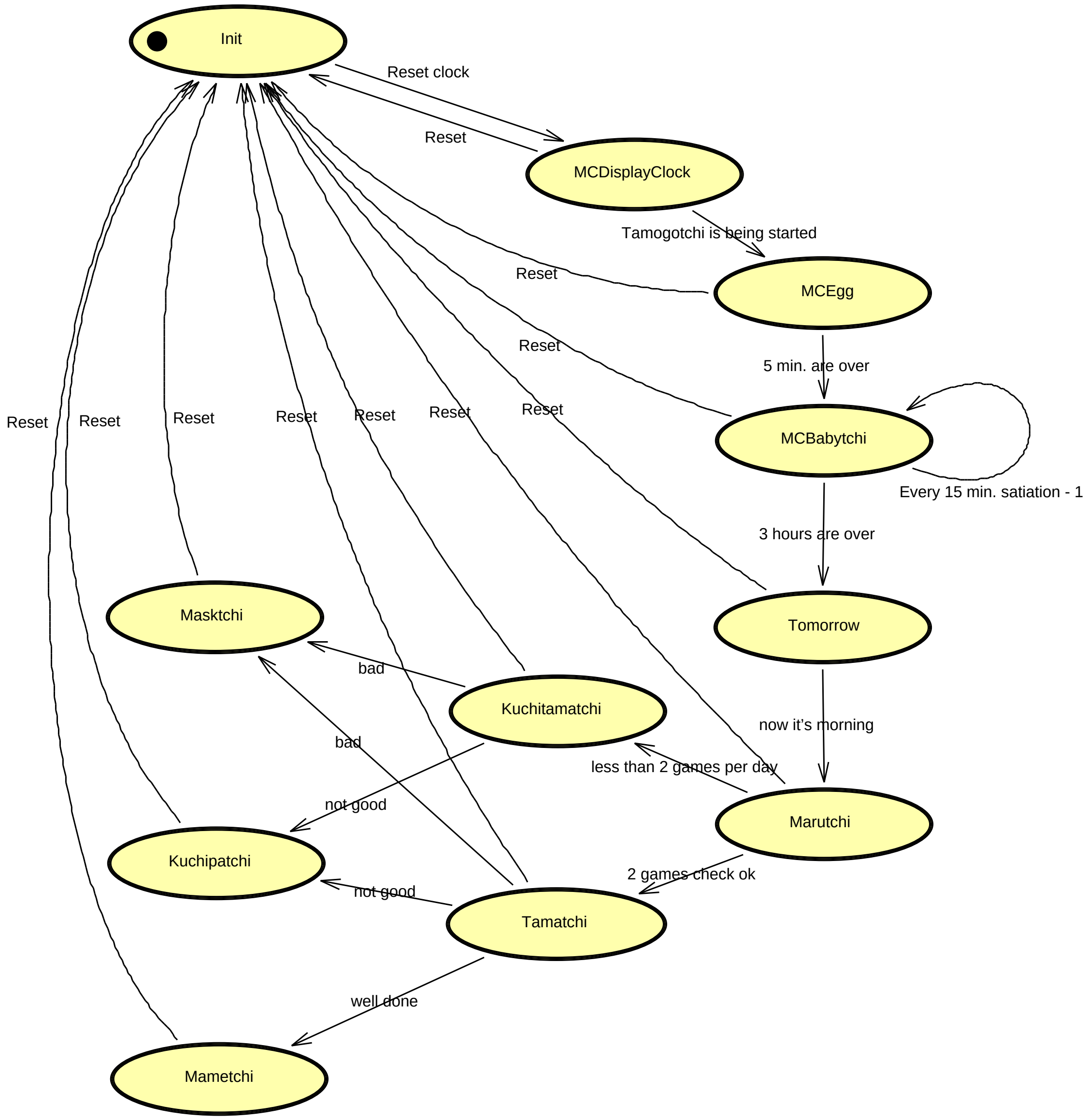
Once we got used to AutoFocus, wih its peculiarities, the work progress was good. We would use AutoFocus again, if some bugs would be eliminated. We think that this tool should be developed further, and with a good documentation and a tutorial the time costs for trial and error developement with AutoFocus would decrease noticeable.

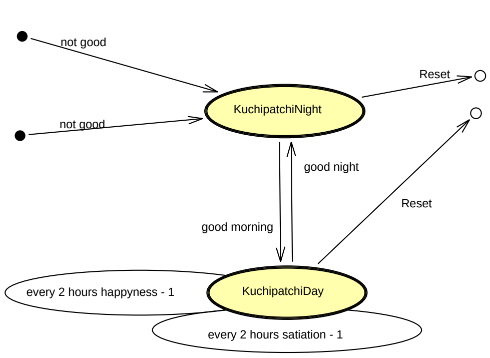
6 Appendix

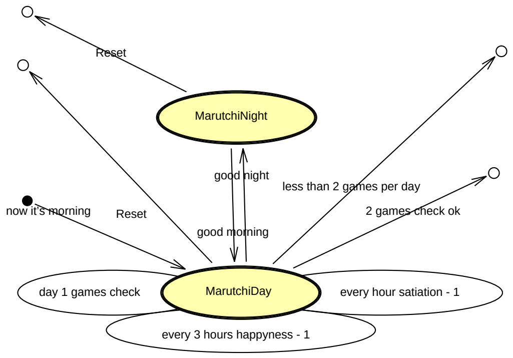
6.1 Tamagotchi

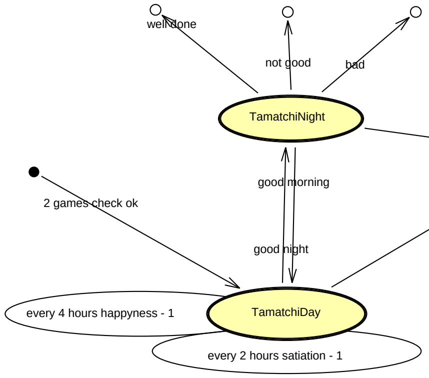


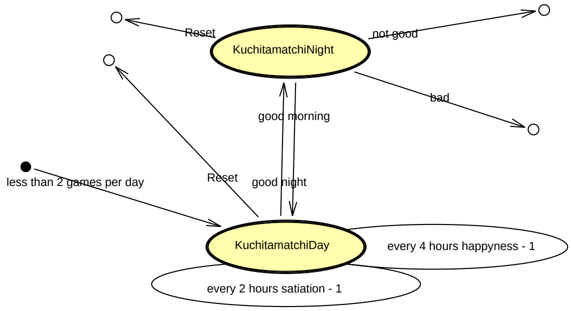
6.2 Lifecycle

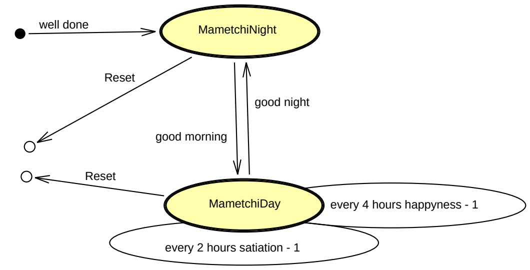


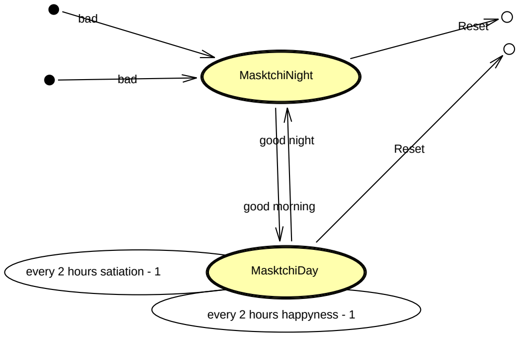







Reset
KuchitamatchiNight
not good
bad
good morning
less than 2 games per day
Reset
good night
KuchitamatchiDay
every 4 hours happyness - 1
every 2 hours satiation - 1





MCDisplayClock $\Longrightarrow$ MCEgg **Tamogotchi is being started**	
Precondition	
Input	mcTimer?T, mcReset?false
Output	dspMCState!2
Postcondition	SleepTimer = T

MCEgg $\Longrightarrow$ MCBabytchi **5 min. are over**	
Precondition	st >=SleepTimer + 5
Input	mcTimer?st
Output	memWeight!5, memMinWeight!5, dspMCState!3, clkReset!true
Postcondition	SatTimer = st; SleepTimer = st

MCBabytchi $\Longrightarrow$ MCBabytchi **Every 15 min. satiation - 1**	
Precondition	st >= SatTimer + 15
Input	mcTimer?st
Output	memAddSatiation!-1
Postcondition	SatTimer = st

Init $\Longrightarrow$ MCDisplayClock **Reset clock**	
Precondition	
Input	
Output	clkReset!true, dspMCState!1
Postcondition	

MCBabytchi ⟹ Tomorrow 3 hours are over	
Precondition	st >= SleepTimer + 180
Input	mcTimer?st
Output	memSleeping!true, dspMCState!4
Postcondition	

Tomorrow ⟹ MarutchiMarutchiDay now it's morning	
Precondition	H >= 9
Input	mcHour?H, mcTimer?T
Output	memSleeping!false, dspMCState!5, memWeight!10, memPlayCounter!0, memMinWeight!10
Postcondition	SatTimer = T; HappyTimer = T; TamaKuchiTimer = T; GameState = 0

MarutchiMarutchiDay ⟹ TamatchiTamatchiDay 2 games check ok	
Precondition	TT >= TamaKuchiTimer + 2880 && GameState == 1 && PC >= 4
Input	mcTimer?TT, mcPlayCounter?PC
Output	dspMCState!7, memWeight!20, memMinWeight!20
Postcondition	SatTimer = TT; HappyTimer = TT; NextStateTimer = TT

MarutchiMarutchiDay ⟹ KuchitamatchiKuchitamatchiDay less than 2 games per day	
Precondition	TT >= TamaKuchiTimer + 2880 && (GameState != 1 || PC < 4)
Input	mcTimer?TT, mcPlayCounter?PC
Output	dspMCState!9, memWeight!20, memMinWeight!20
Postcondition	SatTimer = TT; HappyTimer = TT; NextStateTimer = TT

TamatchiTamatchiNight ⟹ KuchipatchiKuchipatchiNight not good	
Precondition	T > NextStateTimer + 7200
Input	mcTimer?T, mcCareState?2
Output	dspMCState!14, memWeight!20, memMinWeight!20, memAddMaxAge!12
Postcondition	SatTimer = T; HappyTimer = T

TamatchiTamatchiNight ⟹ MasktchiMasktchiNight bad	
Precondition	T > NextStateTimer + 7200
Input	mcTimer?T, mcCareState?3
Output	dspMCState!16, memWeight!30, memMinWeight!30, memAddMaxAge!18
Postcondition	SatTimer = T; HappyTimer = T

KuchitamatchiKuchitamatchiNight $\Longrightarrow$ KuchipatchiKuchipatchiNight **not good**	
Precondition	T > NextStateTimer+7200 && C <= 2
Input	mcTimer?T, mcCareState?C
Output	dspMCState!14, memWeight!20, memMinWeight!20, memAddMaxAge!12
Postcondition	SatTimer = T; HappyTimer = T

KuchitamatchiKuchitamatchiNight $\Longrightarrow$ MasktchiMasktchiNight **bad**	
Precondition	T > NextStateTimer + 7200
Input	mcTimer?T, mcCareState?3
Output	dspMCState!16, memWeight!30, memMinWeight!30, memAddMaxAge!18
Postcondition	SatTimer = T; HappyTimer = T

TamatchiTamatchiNight $\Longrightarrow$ MametchiMametchiNight **well done**	
Precondition	T > NextStateTimer + 7200
Input	mcTimer?T, mcCareState?1
Output	dspMCState!12, memWeight!30, memMinWeight!30, memAddMaxAge!22
Postcondition	SatTimer = T; HappyTimer = T

MCDisplayClock $\Longrightarrow$ Init **Reset**	
Precondition	
Input	mcReset?true
Output	
Postcondition	

MCEgg $\Longrightarrow$ Init **Reset**	
Precondition	
Input	mcReset?true
Output	
Postcondition	

MCBabytchi $\Longrightarrow$ Init **Reset**	
Precondition	
Input	mcReset?true
Output	
Postcondition	

Tomorrow $\Longrightarrow$ Init **Reset**	
Precondition	
Input	mcReset?true
Output	
Postcondition	

MarutchiMarutchiNight $\Longrightarrow$ Init **Reset**	
Precondition	
Input	mcReset?true
Output	
Postcondition	

TamatchiTamatchiNight $\Longrightarrow$ Init **Reset**	
Precondition	
Input	mcReset?true
Output	
Postcondition	

MametchiMametchiDay $\Longrightarrow$ Init **Reset**	
Precondition	
Input	mcReset?true
Output	
Postcondition	

KuchitamatchiKuchitamatchiNight $\Longrightarrow$ Init **Reset**	
Precondition	
Input	mcReset?true
Output	
Postcondition	

KuchipatchiKuchipatchiNight $\Longrightarrow$ Init **Reset**	
Precondition	
Input	mcReset?true
Output	
Postcondition	

MasktchiMasktchiNight $\Longrightarrow$ Init **Reset**	
Precondition	
Input	mcReset?true
Output	
Postcondition	

MarutchiMarutchiDay $\Longrightarrow$ MarutchiMarutchiDay **Marutchievery 3 hours happyness - 1**	
Precondition	ht >= HappyTimer + 180
Input	mcTimer?ht
Output	memAddHappyness!-1
Postcondition	HappyTimer = ht

MarutchiMarutchiDay ⟹ MarutchiMarutchiDay **Marutchievery hour satiation - 1**	
Precondition	st >= SatTimer + 60
Input	mcTimer?st
Output	memAddSatiation!-1
Postcondition	SatTimer = st

MarutchiMarutchiDay ⟹ MarutchiMarutchiNight **Marutchigood night**	
Precondition	h < 9 || h >= 20
Input	mcHour?h
Output	memSleeping!true, dspMCState!6
Postcondition	

MarutchiMarutchiNight ⟹ MarutchiMarutchiDay **Marutchigood morning**	
Precondition	h >= 9 && h < 20
Input	mcHour?h
Output	memSleeping!false, dspMCState!5, memAddWeight!-2
Postcondition	

MarutchiMarutchiDay ⟹ MarutchiMarutchiDay **Marutchiday 1 games check**	
Precondition	TT >= TamaKuchiTimer + 1440 && GameState == 0 && PC >= 2
Input	mcTimer?TT, mcPlayCounter?PC
Output	
Postcondition	GameState = 1

TamatchiTamatchiDay ⟹ TamatchiTamatchiDay **Tamatchievery 2 hours satiation - 1**	
Precondition	st >= SatTimer + 120
Input	mcTimer?st
Output	memAddSatiation!-1
Postcondition	SatTimer = st

TamatchiTamatchiDay ⟹ TamatchiTamatchiDay **Tamatchievery 4 hours happyness - 1**	
Precondition	ht >= HappyTimer + 240
Input	mcTimer?ht
Output	memAddHappyness!-1
Postcondition	HappyTimer = ht

TamatchiTamatchiDay ⟹ TamatchiTamatchiNight **Tamatchigood night**	
Precondition	h < 9 || h >= 21
Input	mcHour?h
Output	memSleeping!true, dspMCState!8
Postcondition	

<table>
<tr><th colspan="2">TamatchiTamatchiNight $\Longrightarrow$ TamatchiTamatchiDay
Tamatchigood morning</th></tr>
<tr><td><b>Precondition</b></td><td>h >= 9 && h < 21</td></tr>
<tr><td><b>Input</b></td><td>mcHour?h</td></tr>
<tr><td><b>Output</b></td><td>memSleeping!false, dspMCState!7, memAddWeight!-4</td></tr>
<tr><td><b>Postcondition</b></td><td></td></tr>
</table>

<table>
<tr><th colspan="2">KuchitamatchiKuchitamatchiDay $\Longrightarrow$ KuchitamatchiKuchitamatchiDay
Kuchitamatchievery 2 hours satiation - 1</th></tr>
<tr><td><b>Precondition</b></td><td>st >= SatTimer + 120</td></tr>
<tr><td><b>Input</b></td><td>mcTimer?st</td></tr>
<tr><td><b>Output</b></td><td>memAddSatiation!-1</td></tr>
<tr><td><b>Postcondition</b></td><td>SatTimer = st</td></tr>
</table>

<table>
<tr><th colspan="2">KuchitamatchiKuchitamatchiDay $\Longrightarrow$ KuchitamatchiKuchitamatchiDay
Kuchitamatchievery 4 hours happyness - 1</th></tr>
<tr><td><b>Precondition</b></td><td>ht >= HappyTimer + 240</td></tr>
<tr><td><b>Input</b></td><td>mcTimer?ht</td></tr>
<tr><td><b>Output</b></td><td>memAddHappyness!-1</td></tr>
<tr><td><b>Postcondition</b></td><td>HappyTimer = ht</td></tr>
</table>

<table>
<tr><th colspan="2">KuchitamatchiKuchitamatchiDay $\Longrightarrow$ KuchitamatchiKuchitamatchiNight
Kuchitamatchigood night</th></tr>
<tr><td><b>Precondition</b></td><td>h < 9 || h >= 21</td></tr>
<tr><td><b>Input</b></td><td>mcHour?h</td></tr>
<tr><td><b>Output</b></td><td>memSleeping!true, dspMCState!10</td></tr>
<tr><td><b>Postcondition</b></td><td></td></tr>
</table>

<table>
<tr><th colspan="2">KuchitamatchiKuchitamatchiNight $\Longrightarrow$ KuchitamatchiKuchitamatchiDay
Kuchitamatchigood morning</th></tr>
<tr><td><b>Precondition</b></td><td>h >= 9 && h < 21</td></tr>
<tr><td><b>Input</b></td><td>mcHour?h</td></tr>
<tr><td><b>Output</b></td><td>memSleeping!false, dspMCState!9, memAddWeight!-4</td></tr>
<tr><td><b>Postcondition</b></td><td></td></tr>
</table>

<table>
<tr><th colspan="2">MametchiMametchiDay $\Longrightarrow$ MametchiMametchiDay
Mametchievery 2 hours satiation - 1</th></tr>
<tr><td><b>Precondition</b></td><td>st >= SatTimer + 120</td></tr>
<tr><td><b>Input</b></td><td>mcTimer?st</td></tr>
<tr><td><b>Output</b></td><td>memAddSatiation!-1</td></tr>
<tr><td><b>Postcondition</b></td><td>SatTimer = st</td></tr>
</table>

<table>
<tr><th colspan="2">MametchiMametchiDay $\Longrightarrow$ MametchiMametchiDay
Mametchievery 4 hours happyness - 1</th></tr>
<tr><td><b>Precondition</b></td><td>ht >= HappyTimer + 240</td></tr>
<tr><td><b>Input</b></td><td>mcTimer?ht</td></tr>
<tr><td><b>Output</b></td><td>memAddHappyness!-1</td></tr>
<tr><td><b>Postcondition</b></td><td>HappyTimer = ht</td></tr>
</table>

MametchiMametchiDay ⟹ MametchiMametchiNight Mametchigood night	
Precondition	h < 10 || h >= 21
Input	mcHour?h
Output	memSleeping!true, dspMCState!12
Postcondition	

MametchiMametchiNight ⟹ MametchiMametchiDay Mametchigood morning	
Precondition	h >= 10 && h < 21
Input	mcHour?h
Output	memSleeping!false, dspMCState!11, memAddWeight!-6
Postcondition	

KuchipatchiKuchipatchiDay ⟹ KuchipatchiKuchipatchiDay Kuchipatchievery 2 hours satiation - 1	
Precondition	st >= SatTimer + 120
Input	mcTimer?st
Output	memAddSatiation!-1
Postcondition	SatTimer = st

KuchipatchiKuchipatchiDay ⟹ KuchipatchiKuchipatchiDay Kuchipatchievery 2 hours happyness - 1	
Precondition	ht >= HappyTimer + 120
Input	mcTimer?ht
Output	memAddHappyness!-1
Postcondition	HappyTimer = ht

KuchipatchiKuchipatchiDay ⟹ KuchipatchiKuchipatchiNight Kuchipatchigood night	
Precondition	h < 10 || h >= 21
Input	mcHour?h
Output	memSleeping!true, dspMCState!14
Postcondition	

KuchipatchiKuchipatchiNight ⟹ KuchipatchiKuchipatchiDay Kuchipatchigood morning	
Precondition	h >= 10 && h < 21
Input	mcHour?h
Output	memSleeping!false, dspMCState!13, memAddWeight!-6
Postcondition	

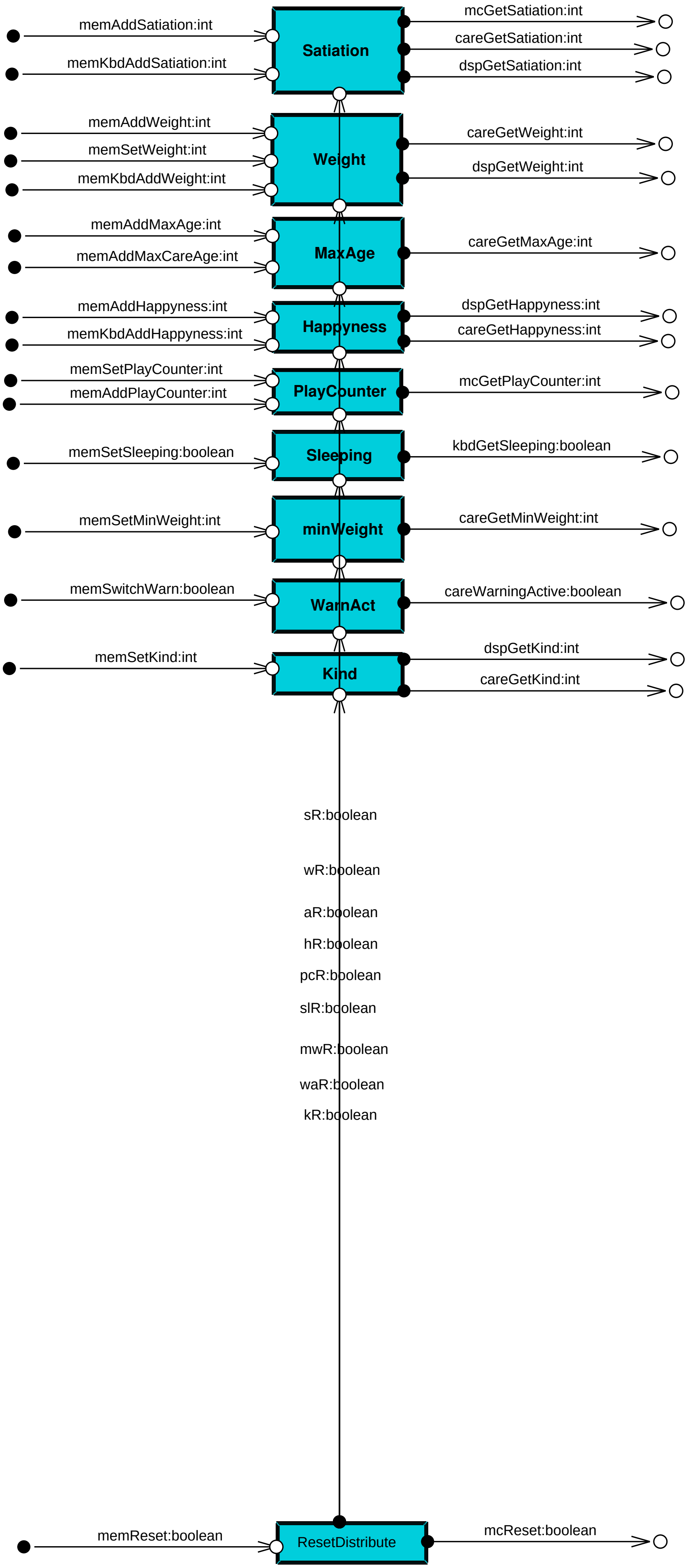
MasktchiMasktchiDay ⟹ MasktchiMasktchiDay Masktchievery 2 hours happyness - 1	
Precondition	ht >= HappyTimer + 120
Input	mcTimer?ht
Output	memAddHappyness!-1
Postcondition	HappyTimer = ht

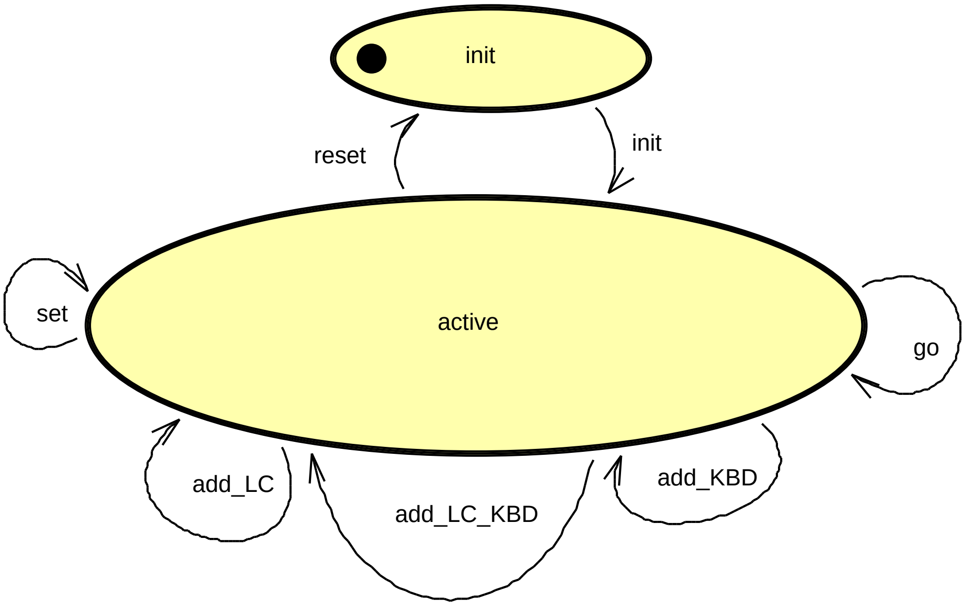
<table>
<tr><th colspan="2">MasktchiMasktchiDay $\Longrightarrow$ MasktchiMasktchiDay
Masktchievery 2 hours satiation - 1</th></tr>
<tr><td><b>Precondition</b></td><td>st >= SatTimer + 120</td></tr>
<tr><td><b>Input</b></td><td>mcTimer?st</td></tr>
<tr><td><b>Output</b></td><td>memAddSatiation!-1</td></tr>
<tr><td><b>Postcondition</b></td><td>SatTimer = st</td></tr>
</table>

<table>
<tr><th colspan="2">MasktchiMasktchiDay $\Longrightarrow$ MasktchiMasktchiNight
Masktchigood night</th></tr>
<tr><td><b>Precondition</b></td><td>h < 11 || h >= 23</td></tr>
<tr><td><b>Input</b></td><td>mcHour?h</td></tr>
<tr><td><b>Output</b></td><td>memSleeping!true, dspMCState!16</td></tr>
<tr><td><b>Postcondition</b></td><td></td></tr>
</table>

<table>
<tr><th colspan="2">MasktchiMasktchiNight $\Longrightarrow$ MasktchiMasktchiDay
Masktchigood morning</th></tr>
<tr><td><b>Precondition</b></td><td>h >= 11 && h < 23</td></tr>
<tr><td><b>Input</b></td><td>mcHour?h</td></tr>
<tr><td><b>Output</b></td><td>memSleeping!false, dspMCState!15, memAddWeight!-6</td></tr>
<tr><td><b>Postcondition</b></td><td></td></tr>
</table>

6.3 Memory


memAddSatiation:int
memKbdAddSatiation:int
Satiation
mcGetSatiation:int
careGetSatiation:int
dspGetSatiation:int
memAddWeight:int
memSetWeight:int
memKbdAddWeight:int
Weight
careGetWeight:int
dspGetWeight:int
memAddMaxAge:int
memAddMaxCareAge:int
MaxAge
careGetMaxAge:int
memAddHappyness:int
memKbdAddHappyness:int
Happyness
dspGetHappyness:int
careGetHappyness:int
memSetPlayCounter:int
memAddPlayCounter:int
PlayCounter
mcGetPlayCounter:int
memSetSleeping:boolean
Sleeping
kbdGetSleeping:boolean
memSetMinWeight:int
minWeight
careGetMinWeight:int
memSwitchWarn:boolean
WarnAct
careWarningActive:boolean
memSetKind:int
Kind
dspGetKind:int
careGetKind:int
sR:boolean
wR:boolean
aR:boolean
hR:boolean
pcR:boolean
slR:boolean
mwR:boolean
waR:boolean
kR:boolean
memReset:boolean
ResetDistribute
mcReset:boolean



active $\Longrightarrow$ active **go**	
Precondition	
Input	memAddWeight?, memKbdAddWeight?, wR?, memSetWeight?
Output	dspGetWeight!Value, careGetWeight!Value
Postcondition	

init $\Longrightarrow$ active **init**	
Precondition	
Input	
Output	dspGetWeight!0, careGetWeight!0
Postcondition	Value = 0

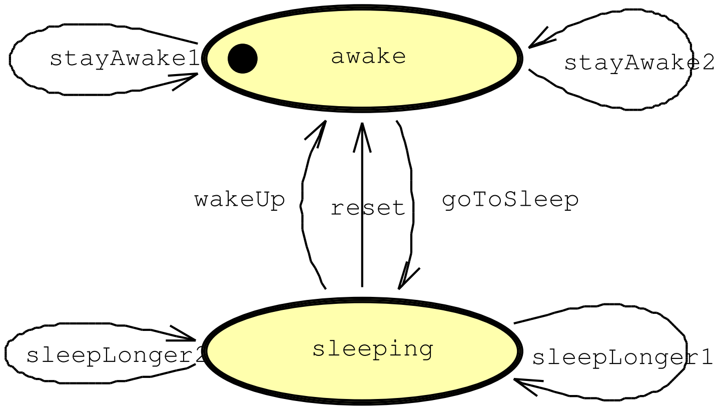
active $\Longrightarrow$ active **add_LC**	
Precondition	
Input	memAddWeight?temp, memKbdAddWeight?, wR?, memSetWeight?
Output	dspGetWeight!Value, careGetWeight!Value
Postcondition	Value = Value + temp

active $\Longrightarrow$ init **reset**	
Precondition	
Input	wR?true
Output	dspGetWeight!0, careGetWeight!0
Postcondition	Value = 0

active $\Longrightarrow$ active **set**	
Precondition	
Input	memSetWeight?temp, wR?
Output	careGetWeight!temp, dspGetWeight!temp
Postcondition	Value = temp

active $\Longrightarrow$ active **add_KBD**	
Precondition	
Input	memKbdAddWeight?temp, memAddWeight?, wR?, memSetWeight?
Output	dspGetWeight!Value, careGetWeight!Value
Postcondition	Value = Value + temp

active $\Longrightarrow$ active **add_LC_KBD**	
Precondition	
Input	memAddWeight?temp1, memKbdAddWeight?temp2, wR?, memSetWeight?
Output	dspGetWeight!Value, careGetWeight!Value
Postcondition	Value = Value + temp1 + temp2



awake ⟹ sleeping **goToSleep**	
Precondition	
Input	memSetSleeping?true, slR?
Output	kbdGetSleeping!true
Postcondition	

sleeping ⟹ awake **wakeUp**	
Precondition	
Input	memSetSleeping?false
Output	kbdGetSleeping!false
Postcondition	

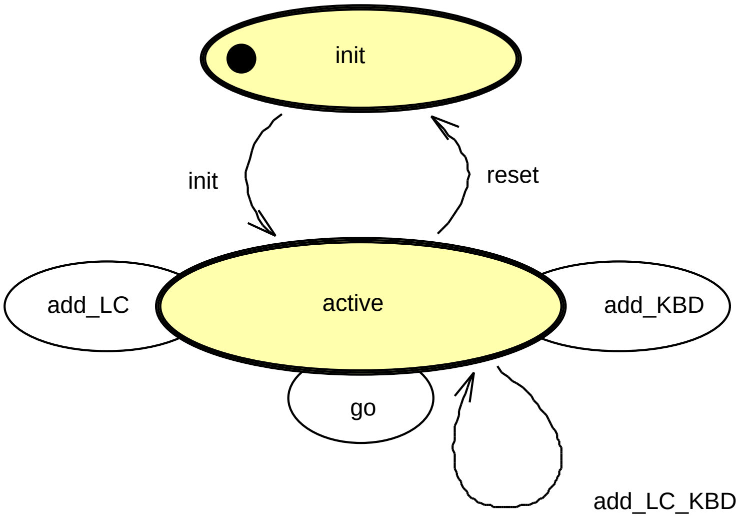
awake ⟹ awake **stayAwake1**	
Precondition	
Input	memSetSleeping?
Output	kbdGetSleeping!false
Postcondition	

awake ⟹ awake **stayAwake2**	
Precondition	
Input	memSetSleeping?false
Output	kbdGetSleeping!false
Postcondition	

sleeping ⟹ sleeping **sleepLonger1**	
Precondition	
Input	memSetSleeping?, slR?
Output	kbdGetSleeping!true
Postcondition	

sleeping ⟹ sleeping **sleepLonger2**	
Precondition	
Input	slR?, memSetSleeping?true
Output	kbdGetSleeping!true
Postcondition	

sleeping ⟹ awake **reset**	
Precondition	
Input	slR?true
Output	kbdGetSleeping!false
Postcondition	



active ⟹ **active** **go**	
Precondition	
Input	memAddSatiation?, memKbdAddSatiation?, sR?
Output	mcGetSatiation!Value, dspGetSatiation!Value, careGetSatiation!Value
Postcondition	

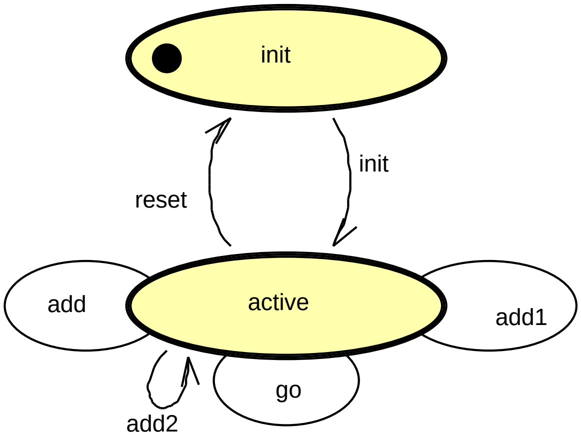
init ⟹ **active** **init**	
Precondition	
Input	
Output	mcGetSatiation!4, dspGetSatiation!4, careGetSatiation!4
Postcondition	Value = 4

active ⟹ **active** **add_LC**	
Precondition	
Input	memAddSatiation?temp, memKbdAddSatiation?, sR?
Output	mcGetSatiation!Value, dspGetSatiation!Value, careGetSatiation!Value
Postcondition	Value = Value + temp

active ⟹ **init** **reset**	
Precondition	
Input	sR?true
Output	dspGetSatiation!4, mcGetSatiation!4, careGetSatiation!4
Postcondition	Value = 4

active $\Longrightarrow$ active **add_KBD**	
Precondition	
Input	memKbdAddSatiation?temp, memAddSatiation?, sR?
Output	mcGetSatiation!Value, careGetSatiation!Value, dspGetSatiation!Value
Postcondition	Value = Value + temp

active $\Longrightarrow$ active **add_LC_KBD**	
Precondition	
Input	memAddSatiation?temp1, memKbdAddSatiation?temp2, sR?
Output	mcGetSatiation!Value, careGetSatiation!Value, dspGetSatiation!Value
Postcondition	Value = Value + temp1 + temp2



active $\Longrightarrow$ active **go**	
Precondition	
Input	memAddMaxAge?, memAddMaxCareAge?, aR?
Output	careGetMaxAge!Value
Postcondition	

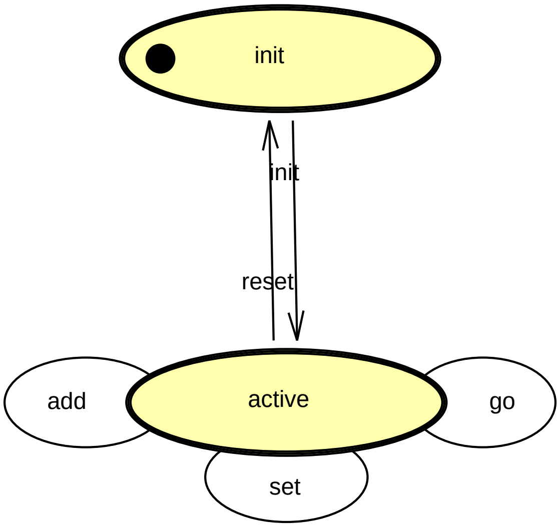
init $\Longrightarrow$ active **init**	
Precondition	
Input	
Output	careGetMaxAge!0
Postcondition	Value = 0

active $\Longrightarrow$ active **add**	
Precondition	
Input	memAddMaxAge?temp, memAddMaxCareAge?, aR?
Output	careGetMaxAge!Value
Postcondition	Value = Value + temp

active $\Longrightarrow$ init	
reset	
Precondition	
Input	aR?true
Output	careGetMaxAge!0
Postcondition	Value = 0

active $\Longrightarrow$ active	
add1	
Precondition	
Input	memAddMaxCareAge?temp, aR?, memAddMaxAge?
Output	careGetMaxAge!Value
Postcondition	Value = Value + temp

active $\Longrightarrow$ active	
add2	
Precondition	
Input	memAddMaxAge?temp1, memAddMaxCareAge?temp2, aR?
Output	careGetMaxAge!Value
Postcondition	Value = Value + temp1 + temp2



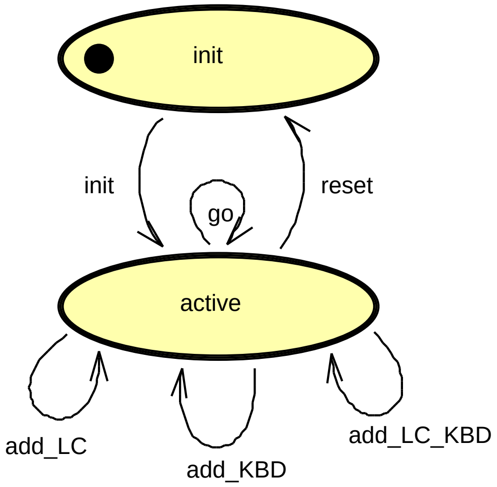
init $\Longrightarrow$ active	
init	
Precondition	
Input	
Output	mcGetPlayCounter!0
Postcondition	Value = 0

active ⟹ init **reset**	
Precondition	
Input	pcR?true
Output	mcGetPlayCounter!0
Postcondition	Value = 0

active ⟹ active **set**	
Precondition	
Input	memSetPlayCounter?temp, pcR?
Output	mcGetPlayCounter!temp
Postcondition	Value = temp

active ⟹ active **go**	
Precondition	
Input	memSetPlayCounter?, pcR?, memAddPlayCounter?
Output	mcGetPlayCounter!Value
Postcondition	

active ⟹ active **add**	
Precondition	
Input	memAddPlayCounter?temp, pcR?, memSetPlayCounter?
Output	mcGetPlayCounter!Value
Postcondition	Value = Value + temp



active ⟹ active **go**	
Precondition	
Input	hR?, memAddHappyness?, memKbdAddHappyness?
Output	dspGetHappyness!Value, careGetHappyness!Value
Postcondition	

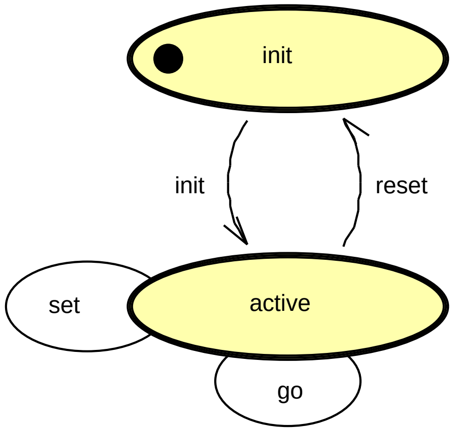
init ⟹ active **init**	
Precondition	
Input	
Output	dspGetHappyness!4, careGetHappyness!4
Postcondition	Value = 4

active ⟹ active **add_LC**	
Precondition	
Input	memAddHappyness?temp, memKbdAddHappyness?, hR?
Output	careGetHappyness!Value, dspGetHappyness!Value
Postcondition	Value = Value + temp

active ⟹ init **reset**	
Precondition	
Input	hR?true
Output	dspGetHappyness!4, careGetHappyness!4
Postcondition	Value = 4

active ⟹ active **add_KBD**	
Precondition	
Input	memKbdAddHappyness?temp, memAddHappyness?, hR?
Output	careGetHappyness!Value, dspGetHappyness!Value
Postcondition	Value = Value + temp

active ⟹ active **add_LC_KBD**	
Precondition	
Input	memKbdAddHappyness?temp1, memAddHappyness?temp2, hR?
Output	careGetHappyness!Value, dspGetHappyness!Value
Postcondition	Value = Value + temp1 + temp2

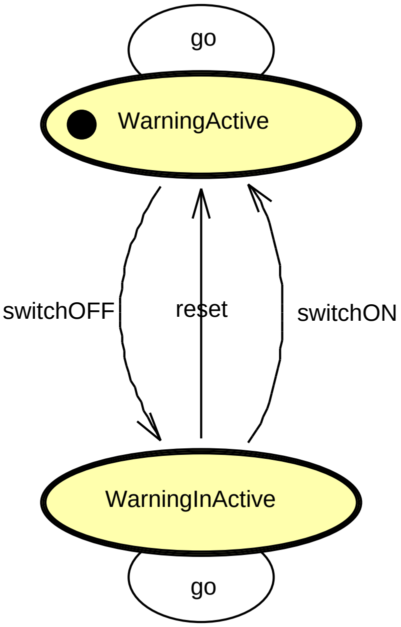


active $\Longrightarrow$ active **go**	
Precondition	
Input	mwR?, memSetMinWeight?
Output	careGetMinWeight!Value
Postcondition	

init $\Longrightarrow$ active **init**	
Precondition	
Input	
Output	careGetMinWeight!0
Postcondition	Value = 0

active $\Longrightarrow$ active **set**	
Precondition	
Input	memSetMinWeight?temp, mwR?
Output	careGetMinWeight!temp
Postcondition	Value = temp

active $\Longrightarrow$ init **reset**	
Precondition	
Input	mwR?true
Output	careGetMinWeight!0
Postcondition	Value = 0



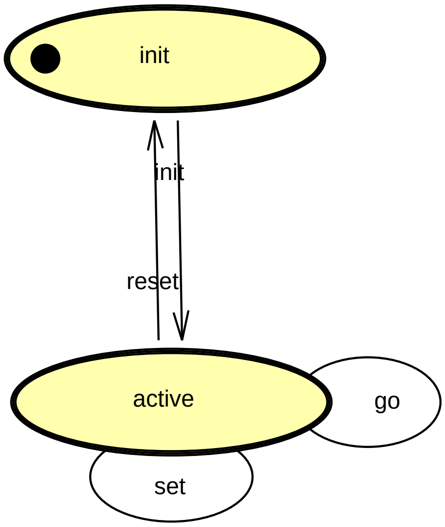
WarningInActive $\Longrightarrow$ WarningActive **reset**	
Precondition	
Input	waR?true
Output	careWarningActive!true
Postcondition	

WarningActive $\Longrightarrow$ WarningInActive **switchOFF**	
Precondition	
Input	memSwitchWarn?true, waR?
Output	careWarningActive!false
Postcondition	

WarningInActive $\Longrightarrow$ WarningActive **switchON**	
Precondition	
Input	memSwitchWarn?true, waR?
Output	careWarningActive!true
Postcondition	

WarningActive $\Longrightarrow$ WarningActive **go**	
Precondition	
Input	memSwitchWarn?
Output	careWarningActive!true
Postcondition	

WarningInActive $\Longrightarrow$ WarningInActive **go**	
Precondition	
Input	waR?, memSwitchWarn?
Output	careWarningActive!false
Postcondition	

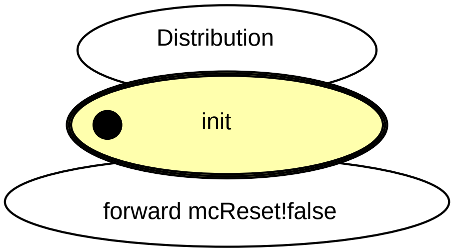


init ⟹ active **init**	
Precondition	
Input	
Output	dspGetKind!1, careGetKind!1
Postcondition	Value = 1

active ⟹ init **reset**	
Precondition	
Input	kR?true
Output	dspGetKind!1, careGetKind!1
Postcondition	Value = 1

active ⟹ active **set**	
Precondition	
Input	memSetKind?temp, kR?
Output	dspGetKind!temp, careGetKind!temp
Postcondition	Value = temp

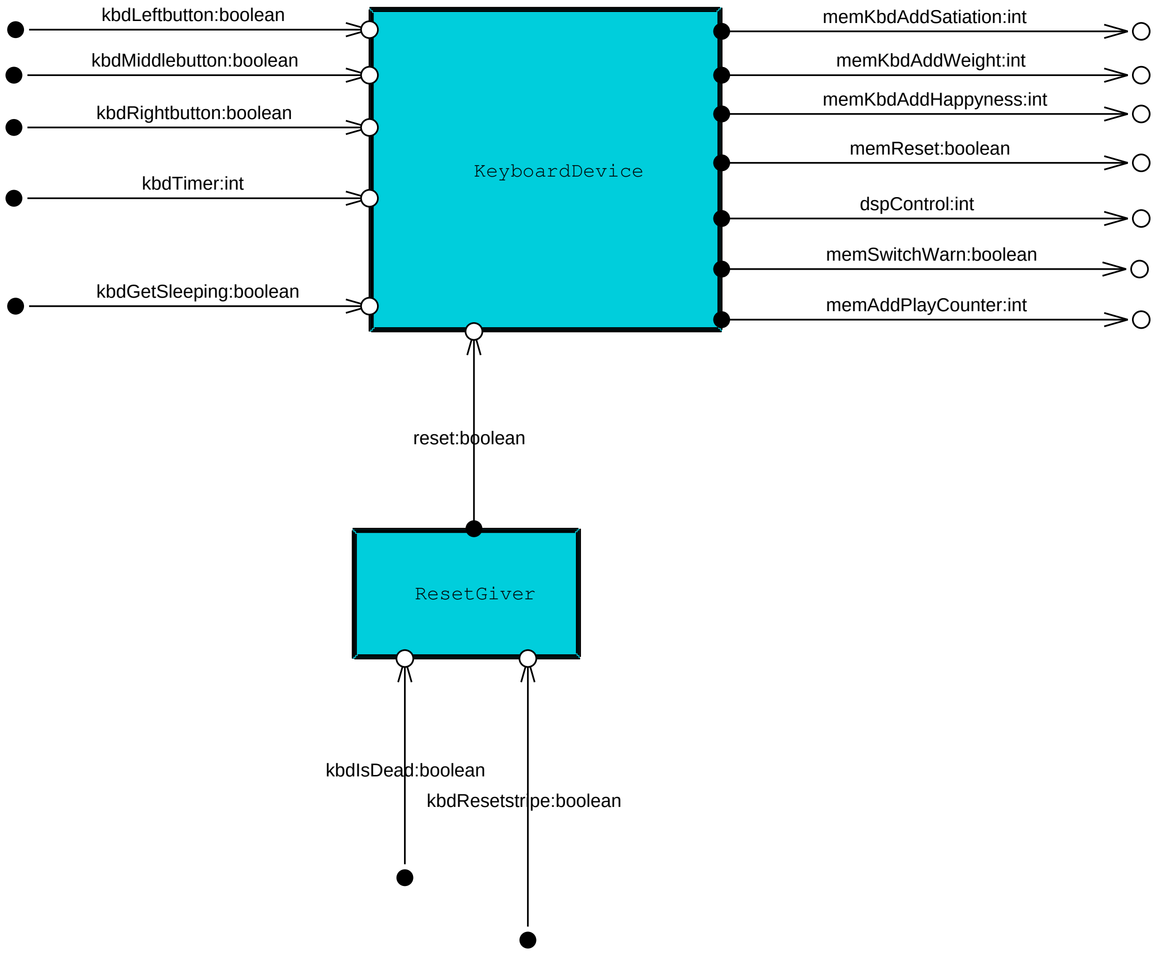
active ⟹ active **go**	
Precondition	
Input	memSetKind?, kR?
Output	dspGetKind!Value, careGetKind!Value
Postcondition	

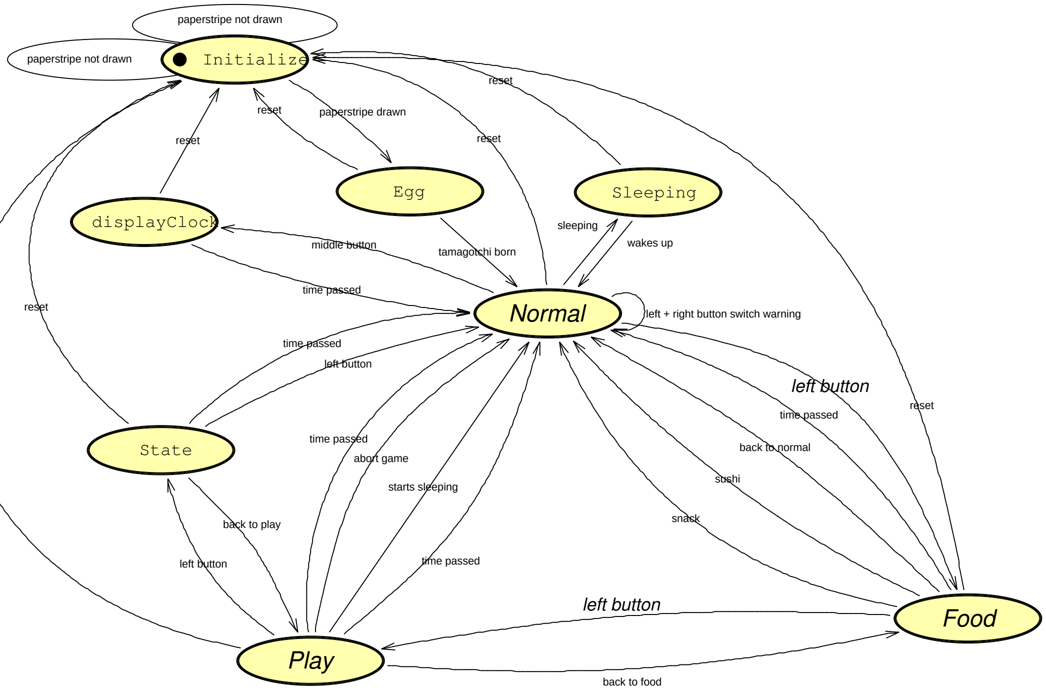


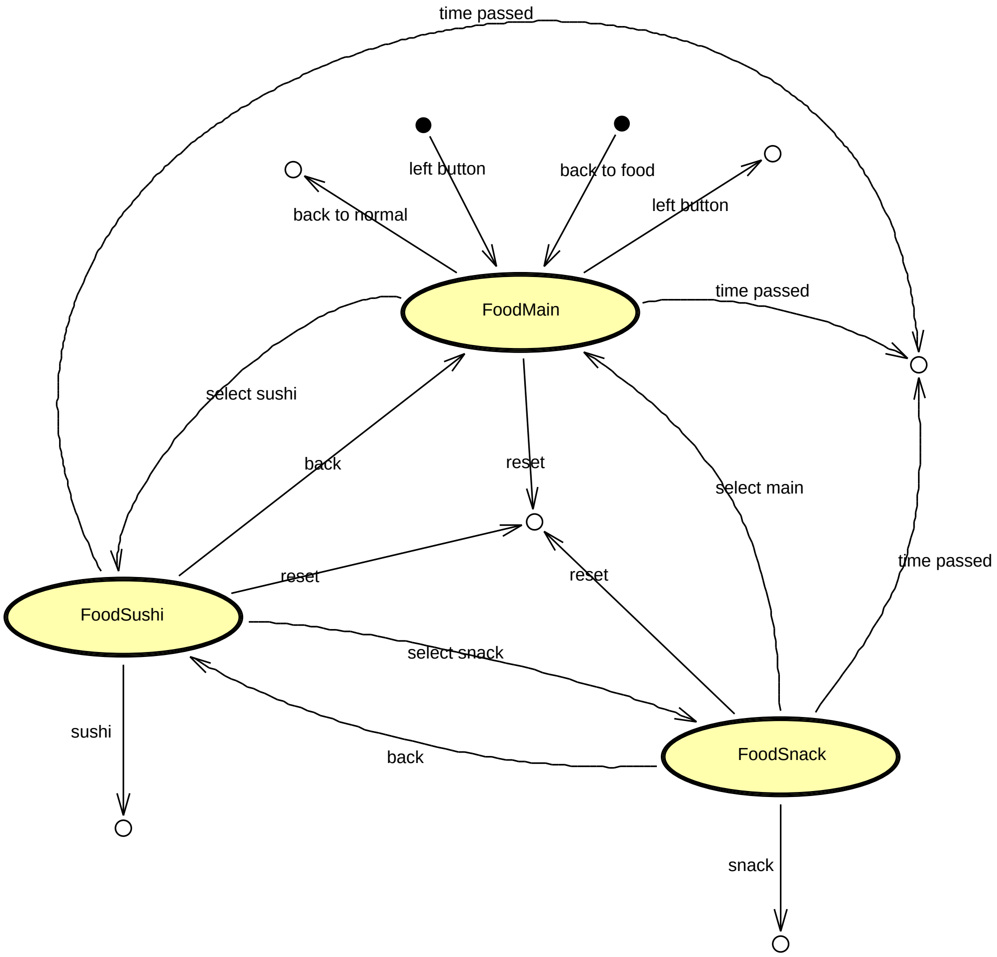
init ⟹ init **Distribution**	
Precondition	
Input	memReset?true
Output	sR!true, wR!true, aR!true, hR!true, pcR!true, slR!true, mwR!true, mcReset!true, waR!true, kR!true
Postcondition	

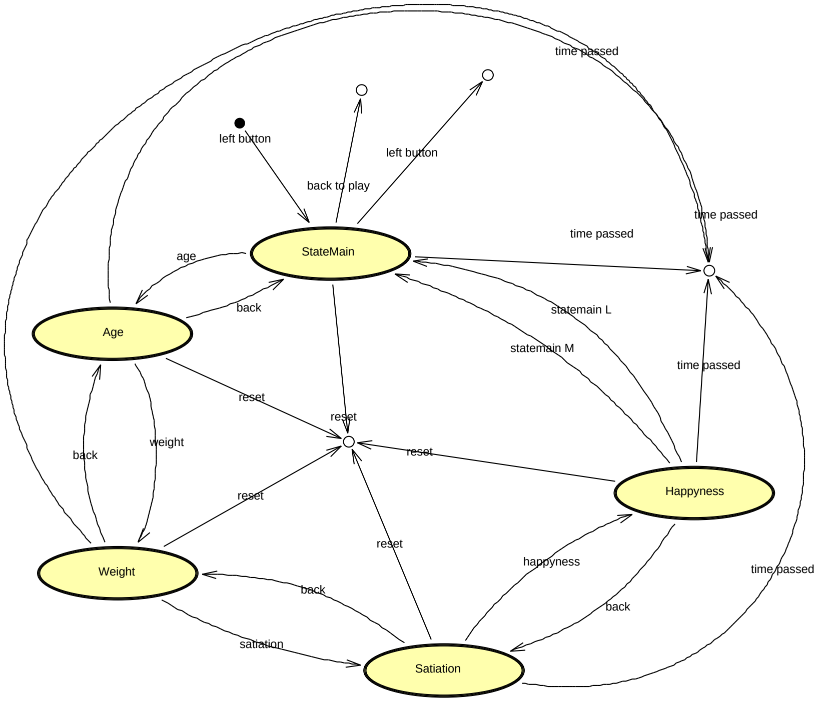
init ⟹ init **forward mcReset!false**	
Precondition	
Input	memReset?false
Output	mcReset!false
Postcondition	

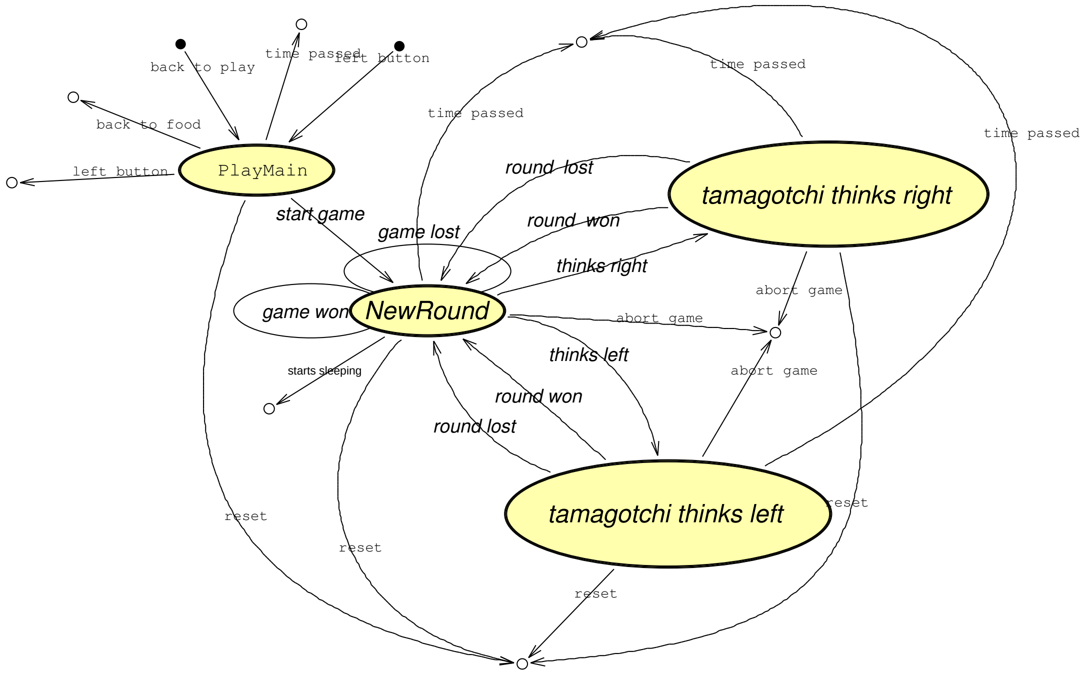
6.4 Keyboard


kbdLeftbutton:boolean
kbdMiddlebutton:boolean
kbdRightbutton:boolean
kbdTimer:int
kbdGetSleeping:boolean
KeyboardDevice
memKbdAddSatiation:int
memKbdAddWeight:int
memKbdAddHappyness:int
memReset:boolean
dspControl:int
memSwitchWarn:boolean
memAddPlayCounter:int
reset:boolean
ResetGiver
kbdIsDead:boolean
kbdResetstripe:boolean


paperstripe not drawn
paperstripe not drawn
Initialize
reset
reset
paperstripe drawn
reset
reset
Egg
Sleeping
displayClock
middle button
tamagotchi born
sleeping
wakes up
time passed
Normal
left + right button switch warning
reset
time passed
left button
reset
left button
time passed
reset
back to normal
State
time passed
abort game
sushi
starts sleeping
snack
back to play
left button
time passed
left button
Food
Play
back to food


time passed
left button
back to food
back to normal
left button
FoodMain
time passed
select sushi
back
reset
select main
reset
reset
time passed
FoodSushi
select snack
sushi
back
FoodSnack
snack


time passed
left button
left button
back to play
StateMain
time passed
time passed
age
back
Age
statemain L
statemain M
time passed
reset
reset
back
weight
reset
reset
Happyness
reset
Weight
happyness
time passed
back
back
satiation
Satiation


back to play
time passed
left button
back to food
left button
PlayMain
time passed
time passed
time passed
round lost
tamagotchi thinks right
start game
round won
game lost
thinks right
abort game
game won
NewRound
abort game
starts sleeping
thinks left
abort game
round won
round lost
reset
tamagotchi thinks left
reset
reset
reset

Initialize $\Longrightarrow$ Initialize **paperstripe not drawn**	
Precondition	
Input	reset?true
Output	dspControl!10
Postcondition	

Initialize $\Longrightarrow$ Initialize **paperstripe not drawn**	
Precondition	
Input	reset?
Output	dspControl!10
Postcondition	

Initialize $\Longrightarrow$ Egg **paperstripe drawn**	
Precondition	
Input	reset?false
Output	memReset!false, dspControl!20
Postcondition	

Egg $\Longrightarrow$ Initialize **reset**	
Precondition	
Input	reset?true
Output	memReset!true, dspControl!10
Postcondition	

Egg $\Longrightarrow$ Normal **tamagotchi born**	
Precondition	
Input	kbdTimer?0
Output	
Postcondition	

Normal $\Longrightarrow$ Initialize **reset**	
Precondition	
Input	reset?true
Output	memReset!true, dspControl!10
Postcondition	

Normal $\Longrightarrow$ displayClock **middle button**	
Precondition	
Input	reset?, kbdTimer?x, kbdMiddlebutton?true
Output	dspControl!10
Postcondition	Savetimer=x

displayClock $\Longrightarrow$ Normal **time passed**	
Precondition	x>Savetimer+5
Input	kbdTimer?x
Output	dspControl!20
Postcondition	Savetimer=0

displayClock $\Longrightarrow$ Initialize **reset**	
Precondition	
Input	reset?true
Output	memReset!true, dspControl!10
Postcondition	

Normal $\Longrightarrow$ Normal **left + right button switch warning**	
Precondition	
Input	reset?, kbdLeftbutton?true, kbdRightbutton?true, kbdGetSleeping?false
Output	memSwitchWarn!true
Postcondition	

Normal $\Longrightarrow$ FoodFoodMain **left button**	
Precondition	
Input	reset?, kbdLeftbutton?true, kbdTimer?x, kbdGetSleeping?false
Output	dspControl!30
Postcondition	Savetimer=x

FoodFoodSushi $\Longrightarrow$ Normal **time passed**	
Precondition	x>Savetimer+20
Input	kbdTimer?x
Output	dspControl!20
Postcondition	

FoodFoodSnack $\Longrightarrow$ Initialize **reset**	
Precondition	
Input	reset?true
Output	memReset!true, dspControl!10
Postcondition	

FoodFoodMain $\Longrightarrow$ Normal **back to normal**	
Precondition	
Input	reset?, kbdRightbutton?true
Output	dspControl!20
Postcondition	

FoodFoodMain $\Longrightarrow$ PlayPlayMain **left button**	
Precondition	
Input	reset?, kbdLeftbutton?true
Output	dspControl!40
Postcondition	

PlayPlayMain $\Longrightarrow$ FoodFoodMain **back to food**	
Precondition	
Input	reset?, kbdRightbutton?true
Output	dspControl!30
Postcondition	

PlayNewRound $\Longrightarrow$ Normal **time passed**	
Precondition	x>Savetimer+20
Input	reset?, kbdTimer?x
Output	dspControl!20
Postcondition	

Playtamagotchi thinks left $\Longrightarrow$ Initialize **reset**	
Precondition	
Input	reset?true
Output	memReset!true, dspControl!10
Postcondition	

PlayPlayMain $\Longrightarrow$ StateStateMain **left button**	
Precondition	
Input	reset?, kbdLeftbutton?true
Output	dspControl!50
Postcondition	

StateStateMain $\Longrightarrow$ PlayPlayMain **back to play**	
Precondition	
Input	reset?, kbdRightbutton?true
Output	dspControl!40
Postcondition	

StateWeight $\Longrightarrow$ Normal **time passed**	
Precondition	x>Savetimer+20
Input	reset?, kbdTimer?x
Output	dspControl!20
Postcondition	

StateStateMain ⟹ Normal **left button**	
Precondition	
Input	reset?, kbdLeftbutton?true
Output	dspControl!20
Postcondition	

StateHappyness ⟹ Initialize **reset**	
Precondition	
Input	reset?true
Output	memReset!true, dspControl!10
Postcondition	

Normal ⟹ Sleeping **sleeping**	
Precondition	
Input	reset?, kbdGetSleeping?true
Output	dspControl!21
Postcondition	

Sleeping ⟹ Normal **wakes up**	
Precondition	
Input	kbdGetSleeping?false
Output	dspControl!20
Postcondition	

Sleeping ⟹ Initialize **reset**	
Precondition	
Input	reset?true
Output	memReset!true, dspControl!10
Postcondition	

FoodFoodSushi ⟹ Normal **sushi**	
Precondition	
Input	reset?, kbdMiddlebutton?true
Output	dspControl!20, memKbdAddSatiation!1, memKbdAddWeight!1
Postcondition	

FoodFoodSnack ⟹ Normal **snack**	
Precondition	
Input	reset?, kbdMiddlebutton?true
Output	dspControl!22, memKbdAddHappyness!1, memKbdAddWeight!2
Postcondition	

Playtamagotchi thinks right $\Longrightarrow$ Normal **abort game**	
Precondition	
Input	reset?, kbdRightbutton?true
Output	dspControl!20
Postcondition	

PlayNewRound $\Longrightarrow$ Normal **time passed**	
Precondition	x>Savetimer+50
Input	reset?, kbdTimer?x
Output	dspControl!20
Postcondition	

PlayNewRound $\Longrightarrow$ Normal **starts sleeping**	
Precondition	
Input	reset?, kbdGetSleeping?true
Output	dspControl!20
Postcondition	

FoodFoodMain $\Longrightarrow$ FoodFoodSushi **Foodselect sushi**	
Precondition	
Input	reset?, kbdMiddlebutton?true
Output	dspControl!31
Postcondition	

FoodFoodSushi $\Longrightarrow$ FoodFoodSnack **Foodselect snack**	
Precondition	
Input	reset?, kbdLeftbutton?true
Output	dspControl!32
Postcondition	

FoodFoodSushi $\Longrightarrow$ FoodFoodMain **Foodback**	
Precondition	
Input	reset?, kbdRightbutton?true
Output	dspControl!30
Postcondition	

FoodFoodSnack $\Longrightarrow$ FoodFoodSushi **Foodback**	
Precondition	
Input	reset?, kbdRightbutton?true
Output	dspControl!31
Postcondition	

FoodFoodSnack $\Longrightarrow$ FoodFoodMain **Foodselect main**	
Precondition	
Input	reset?, kbdLeftbutton?true
Output	dspControl!30
Postcondition	

PlayPlayMain $\Longrightarrow$ PlayNewRound **Playstart game**	
Precondition	
Input	reset?, kbdMiddlebutton?true, kbdTimer?x
Output	dspControl!41
Postcondition	Round=0; RoundsWon=0; Savetimer=x

PlayNewRound $\Longrightarrow$ Playtamagotchi thinks left **Playthinks left**	
Precondition	Round<5
Input	reset?, kbdGetSleeping?false
Output	dspControl!41
Postcondition	

Playtamagotchi thinks left $\Longrightarrow$ PlayNewRound **Playround won**	
Precondition	
Input	reset?, kbdMiddlebutton?true
Output	dspControl!44
Postcondition	Round++; RoundsWon++

Playtamagotchi thinks left $\Longrightarrow$ PlayNewRound **Playround lost**	
Precondition	
Input	reset?, kbdLeftbutton?true
Output	dspControl!45
Postcondition	Round++

Playtamagotchi thinks right $\Longrightarrow$ PlayNewRound **Playround lost**	
Precondition	
Input	reset?, kbdMiddlebutton?true
Output	dspControl!43
Postcondition	Round++

Playtamagotchi thinks right $\Longrightarrow$ PlayNewRound **Playround won**	
Precondition	
Input	reset?, kbdLeftbutton?true
Output	dspControl!42
Postcondition	Round++; RoundsWon++

PlayNewRound $\Longrightarrow$ Playtamagotchi thinks right **Playthinks right**	
Precondition	Round<5
Input	reset?, kbdGetSleeping?false
Output	dspControl!41
Postcondition	

PlayNewRound $\Longrightarrow$ PlayNewRound **Playgame won**	
Precondition	Round==5 && RoundsWon>2
Input	reset?, kbdTimer?x, kbdGetSleeping?false
Output	dspControl!41, memKbdAddHappyness!1, memKbdAddWeight!-1, memAddPlayCounter!1
Postcondition	Savetimer=x; Round=0; RoundsWon=0

PlayNewRound $\Longrightarrow$ PlayNewRound **Playgame lost**	
Precondition	Round==5 && RoundsWon<3
Input	reset?, kbdTimer?x, kbdGetSleeping?false
Output	dspControl!41, memKbdAddWeight!-1, memAddPlayCounter!1
Postcondition	Savetimer=x; Round=0; RoundsWon=0;

StateStateMain $\Longrightarrow$ StateAge **Stateage**	
Precondition	
Input	reset?, kbdMiddlebutton?true
Output	dspControl!51
Postcondition	

StateAge $\Longrightarrow$ StateStateMain **Stateback**	
Precondition	
Input	reset?, kbdRightbutton?true
Output	dspControl!50
Postcondition	

StateAge $\Longrightarrow$ StateWeight **Stateweight**	
Precondition	
Input	reset?, kbdMiddlebutton?true
Output	dspControl!52
Postcondition	

StateWeight $\Longrightarrow$ StateAge **Stateback**	
Precondition	
Input	reset?, kbdRightbutton?true
Output	dspControl!51
Postcondition	

StateWeight $\Longrightarrow$ StateSatiation **Statesatiation**	
Precondition	
Input	reset?, kbdMiddlebutton?true
Output	dspControl!53
Postcondition	

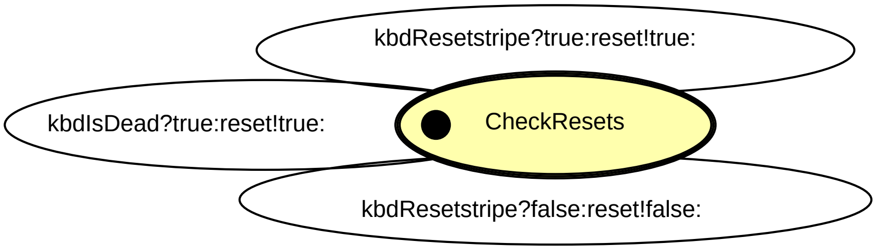
StateSatiation $\Longrightarrow$ StateWeight **Stateback**	
Precondition	
Input	reset?, kbdRightbutton?true
Output	dspControl!52
Postcondition	

StateSatiation $\Longrightarrow$ StateHappyness **Statehappyness**	
Precondition	
Input	reset?, kbdMiddlebutton?true
Output	dspControl!54
Postcondition	

StateHappyness $\Longrightarrow$ StateSatiation **Stateback**	
Precondition	
Input	reset?, kbdRightbutton?true
Output	dspControl!53
Postcondition	

StateHappyness $\Longrightarrow$ StateStateMain **Statestatemain L**	
Precondition	
Input	reset?, kbdLeftbutton?true
Output	dspControl!50
Postcondition	

StateHappyness $\Longrightarrow$ StateStateMain **Statestatemain M**	
Precondition	
Input	reset?, kbdMiddlebutton?true
Output	dspControl!50
Postcondition	

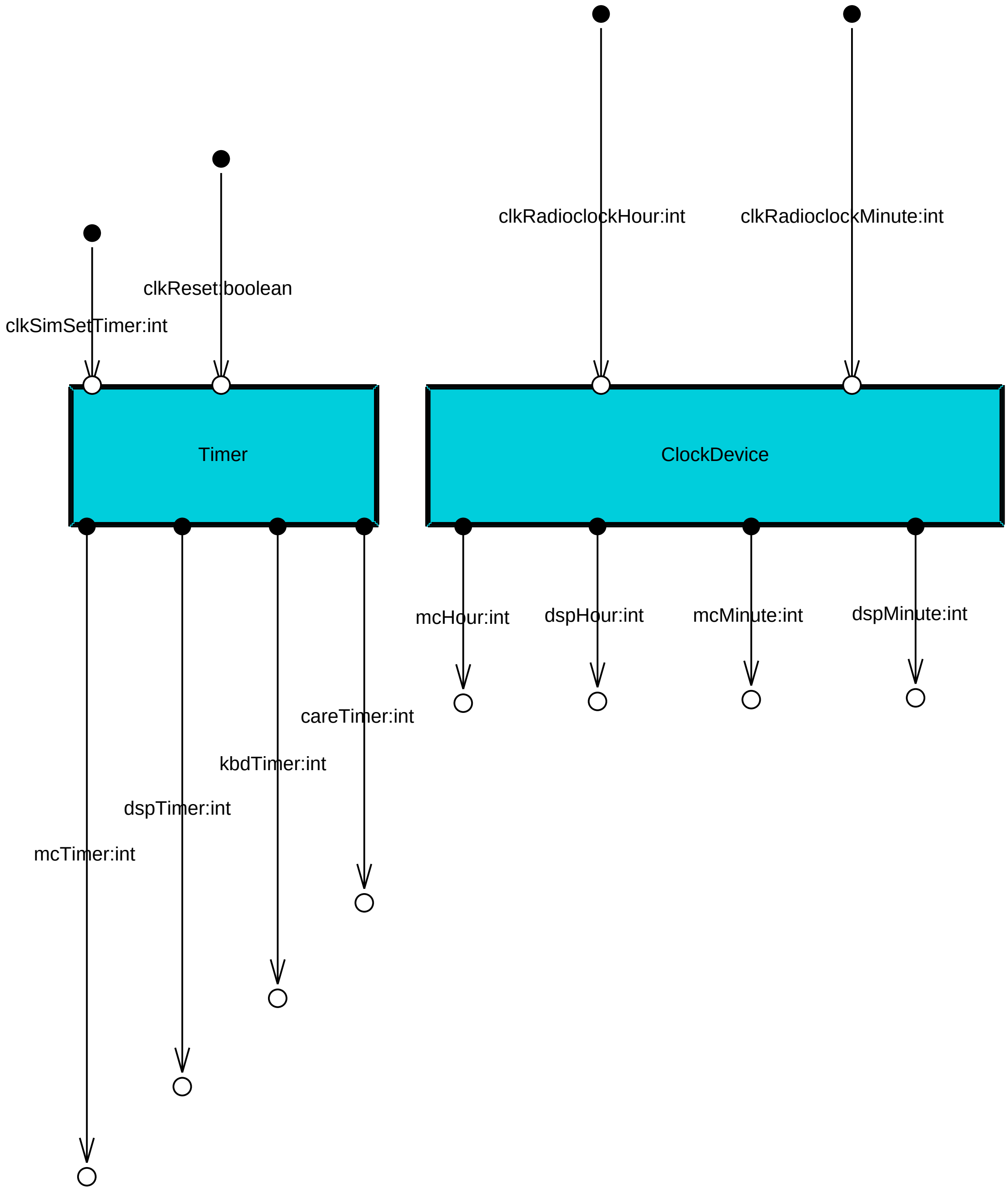

kbdResetstripe?true:reset!true:
kbdIsDead?true:reset!true:
CheckResets
kbdResetstripe?false:reset!false:

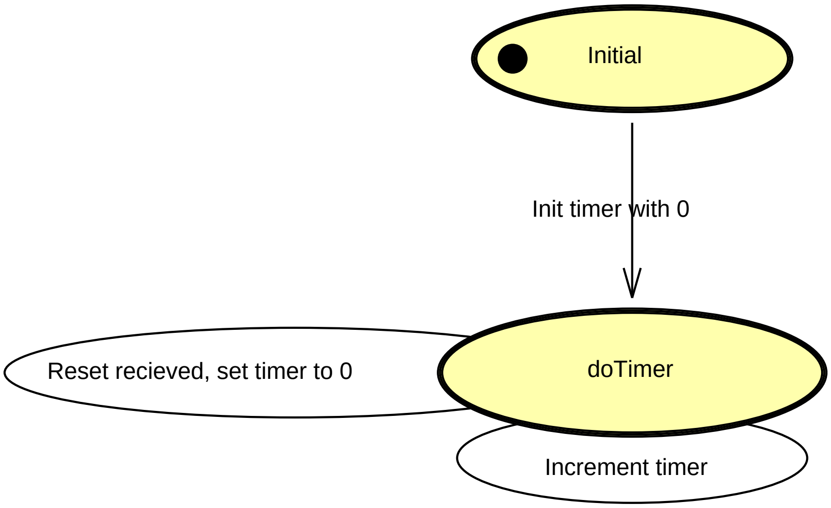
CheckResets ⟹ CheckResets **kbdResetstripe?true:reset!true:**	
Precondition	
Input	kbdResetstripe?true
Output	reset!true
Postcondition	

CheckResets ⟹ CheckResets **kbdResetstripe?false:reset!false:**	
Precondition	
Input	kbdResetstripe?false
Output	reset!false
Postcondition	

CheckResets ⟹ CheckResets **kbdIsDead?true:reset!true:**	
Precondition	
Input	kbdIsDead?true
Output	reset!true
Postcondition	

6.5 Clock

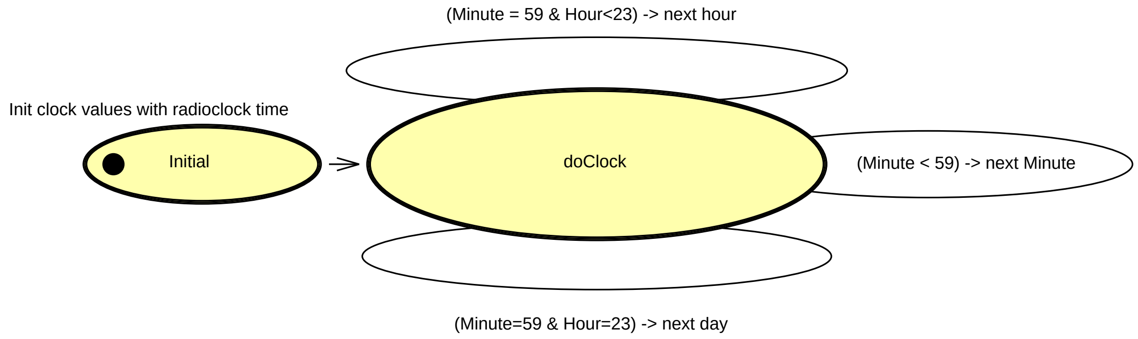

clkRadioclockHour:int
clkRadioclockMinute:int
clkReset:boolean
clkSimSetTimer:int
Timer
ClockDevice
mcHour:int
dspHour:int
mcMinute:int
dspMinute:int
careTimer:int
kbdTimer:int
dspTimer:int
mcTimer:int



Initial ⟹ doTimer **Init timer with 0**	
Precondition	
Input	
Output	mcTimer!Timervalue, careTimer!Timervalue, dspTimer!Timervalue, kbdTimer!Timervalue
Postcondition	Timervalue=0

doTimer ⟹ doTimer **Increment timer**	
Precondition	
Input	clkReset?
Output	mcTimer!Timervalue, careTimer!Timervalue, dspTimer!Timervalue, kbdTimer!Timervalue
Postcondition	Timervalue++

doTimer ⟹ doTimer **Reset recieved, set timer to 0**	
Precondition	
Input	clkReset?true
Output	mcTimer!Timervalue, careTimer!Timervalue, dspTimer!Timervalue, kbdTimer!Timervalue
Postcondition	Timervalue=0


(Minute = 59 & Hour<23) -> next hour
Init clock values with radioclock time
Initial
doClock
(Minute < 59) -> next Minute
(Minute=59 & Hour=23) -> next day

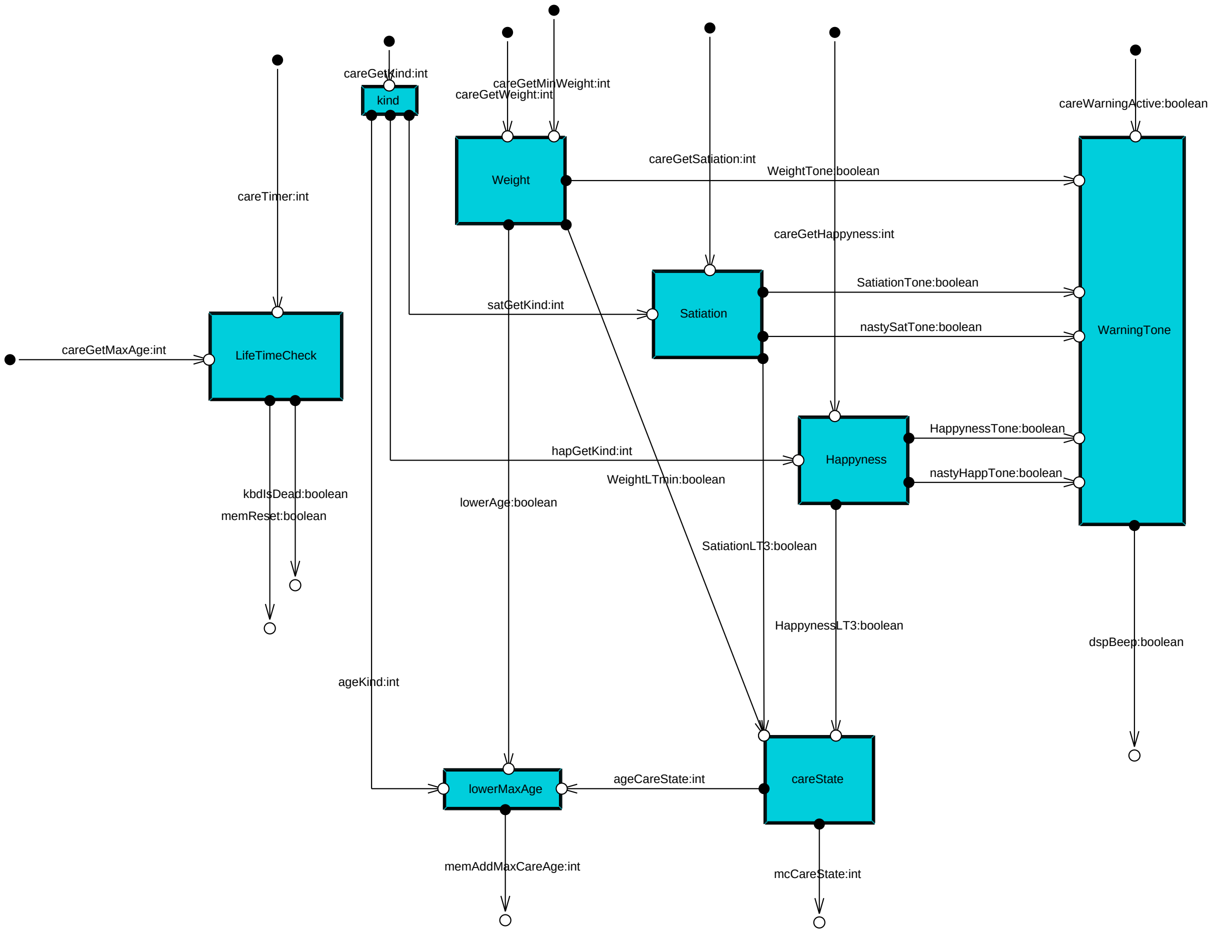
Initial $\Longrightarrow$ doClock **Init clock values with radioclock time**	
Precondition	
Input	clkRadioclockHour?x, clkRadioclockMinute?y
Output	mcHour!Hourvalue, mcMinute!Minutevalue, dspHour!Hourvalue, dspMinute!Minutevalue
Postcondition	Hourvalue=x; Minutevalue=y

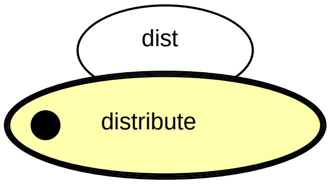
doClock $\Longrightarrow$ doClock **(Minute < 59) -> next Minute**	
Precondition	Minutevalue < 59
Input	
Output	mcHour!Hourvalue, mcMinute!Minutevalue, dspHour!Hourvalue, dspMinute!Minutevalue
Postcondition	Minutevalue++

doClock $\Longrightarrow$ doClock **(Minute = 59 & Hour<23) -> next hour**	
Precondition	Minutevalue==59 && Hourvalue<23
Input	
Output	mcHour!Hourvalue, mcMinute!Minutevalue, dspHour!Hourvalue, dspMinute!Minutevalue
Postcondition	Minutevalue=0; Hourvalue++

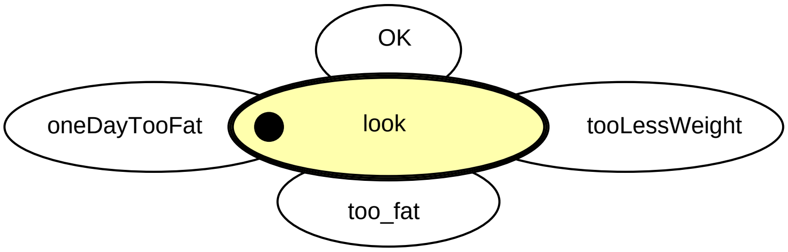
doClock $\Longrightarrow$ doClock **(Minute=59 & Hour=23) -> next day**	
Precondition	Minutevalue==59 && Hourvalue == 23
Input	
Output	mcHour!Hourvalue, mcMinute!Minutevalue, dspHour!Hourvalue, dspMinute!Minutevalue
Postcondition	Minutevalue=0; Hourvalue=0

6.6 Care





distribute ⟹ distribute **dist**	
Precondition	
Input	careGetKind?x
Output	ageKind!x, hapGetKind!x, satGetKind!x
Postcondition	

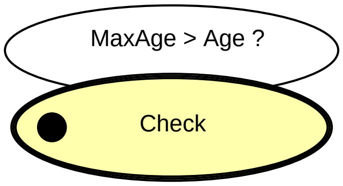


look ⟹ look **OK**	
Precondition	(2*w < 3*m) && (w>m)
Input	careGetWeight?w, careGetMinWeight?m
Output	WeightTone!false, lowerAge!false, WeightLTmin!false
Postcondition	Timer = 1440

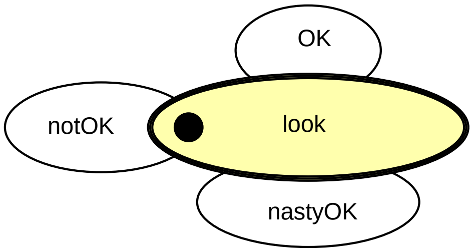
look ⟹ look **too_fat**	
Precondition	(2*w > 3*m) && (Timer > 0)
Input	careGetMinWeight?m, careGetWeight?w
Output	WeightTone!true, lowerAge!false, WeightLTmin!false
Postcondition	Timer = Timer - 1

look ⟹ look **oneDayTooFat**	
Precondition	(2*w > 3*m) && (Timer <= 0)
Input	careGetMinWeight?m, careGetWeight?w
Output	WeightTone!false, lowerAge!true, WeightLTmin!false
Postcondition	Timer = 1440

look ⟹ look **tooLessWeight**	
Precondition	w < m
Input	careGetMinWeight?m, careGetWeight?w
Output	WeightTone!true, lowerAge!false, WeightLTmin!true
Postcondition	Timer = 1440



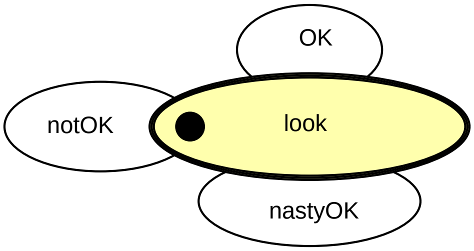
Check $\Longrightarrow$ Check **MaxAge > Age ?**	
Precondition	MaxAge * 1440 > Age
Input	careGetMaxAge?MaxAge, careTimer?Age
Output	careReset!true, kbdIsDead!true
Postcondition	



look $\Longrightarrow$ look **OK**	
Precondition	(s >= 3) && ((k != 15) || ((k == 15) && (s >= oldValue)))
Input	careGetSatiation?s, satGetKind?k
Output	SatiationTone!false, nastySatTone!false, SatiationLT3!false
Postcondition	oldValue = s

look $\Longrightarrow$ look **nastyOK**	
Precondition	(s >= 3) && (k == 15) && (s < oldValue)
Input	careGetSatiation?s, satGetKind?k
Output	SatiationTone!false, nastySatTone!true, SatiationLT3!false
Postcondition	oldValue = s

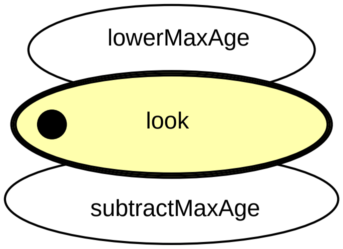
look $\Longrightarrow$ look **notOK**	
Precondition	(s < 3)
Input	careGetSatiation?s
Output	SatiationTone!true, nastySatTone!false, SatiationLT3!true
Postcondition	oldValue = s



look $\Longrightarrow$ look **OK**	
Precondition	(h >= 3) && ((k != 15) || (h >= oldValue))
Input	careGetHappyness?h, hapGetKind?k
Output	HappynessTone!false, nastyHappTone!false, HappynessLT3!false
Postcondition	oldValue = h

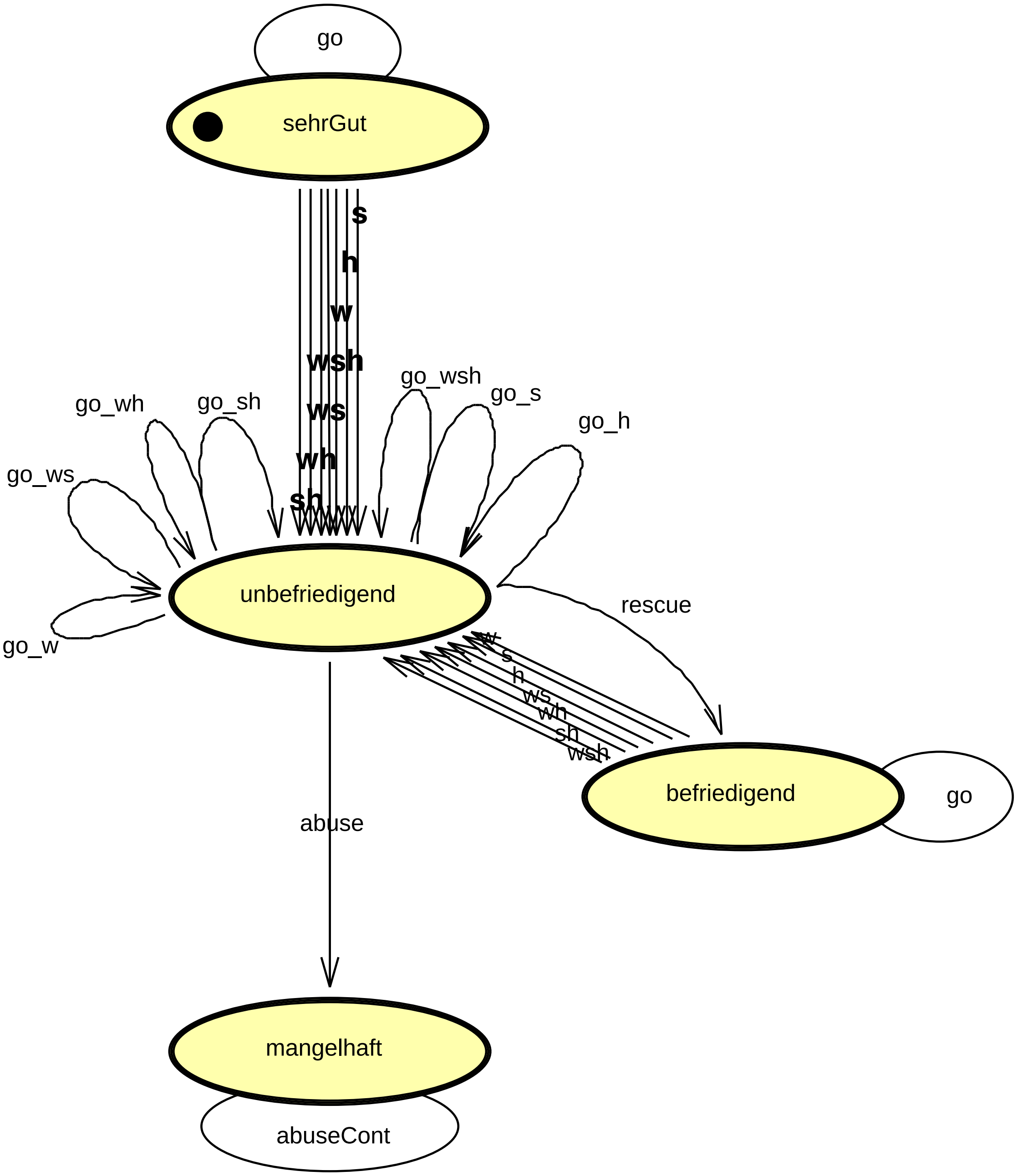
look $\Longrightarrow$ look **nastyOK**	
Precondition	(h >= 3) && (k == 15) && (h < oldValue)
Input	careGetHappyness?h, hapGetKind?k
Output	HappynessTone!false, nastyHappTone!true, HappynessLT3!false
Postcondition	oldValue = h

look $\Longrightarrow$ look **notOK**	
Precondition	(h < 3)
Input	careGetHappyness?h
Output	HappynessTone!true, nastyHappTone!false, HappynessLT3!true
Postcondition	oldValue = h



look $\Longrightarrow$ look **lowerMaxAge**	
Precondition	((k == 11) && (cs > 1)) || ((k == 13) && (cs > 2))
Input	ageKind?k, ageCareState?cs, lowerAge?false
Output	memAddMaxCareAge!-1
Postcondition	

look $\Longrightarrow$ look **subtractMaxAge**	
Precondition	
Input	lowerAge?true
Output	memAddMaxCareAge!-1
Postcondition	



sehrGut $\Longrightarrow$ unbefriedigend **s**	
Precondition	
Input	WeightLTmin?, SatiationLT3?true, HappynessLT3?
Output	mcCareState!2, ageCareState!2
Postcondition	Timer = 120

sehrGut ⟹ unbefriedigend **h**	
Precondition	
Input	WeightLTmin?, SatiationLT3?, HappynessLT3?true
Output	mcCareState!2, ageCareState!2
Postcondition	Timer = 120

sehrGut ⟹ unbefriedigend **w**	
Precondition	
Input	WeightLTmin?true, SatiationLT3?, HappynessLT3?
Output	mcCareState!2, ageCareState!2
Postcondition	Timer = 120

sehrGut ⟹ sehrGut **go**	
Precondition	
Input	WeightLTmin?, SatiationLT3?, HappynessLT3?
Output	mcCareState!1, ageCareState!1
Postcondition	

unbefriedigend ⟹ unbefriedigend **go_w**	
Precondition	Timer > 0
Input	WeightLTmin?true, SatiationLT3?, HappynessLT3?
Output	mcCareState!2, ageCareState!2
Postcondition	Timer = Timer - 1

sehrGut ⟹ unbefriedigend **wsh**	
Precondition	
Input	WeightLTmin?true, SatiationLT3?true, HappynessLT3?true
Output	mcCareState!2, ageCareState!2
Postcondition	Timer = 120

sehrGut ⟹ unbefriedigend **ws**	
Precondition	
Input	WeightLTmin?true, SatiationLT3?true, HappynessLT3?
Output	mcCareState!2, ageCareState!2
Postcondition	Timer = 120

sehrGut ⟹ unbefriedigend **wh**	
Precondition	
Input	WeightLTmin?true, SatiationLT3?, HappynessLT3?true
Output	mcCareState!2, ageCareState!2
Postcondition	Timer = 120

sehrGut $\Longrightarrow$ unbefriedigend **sh**	
Precondition	
Input	WeightLTmin?, SatiationLT3?true, HappynessLT3?true
Output	mcCareState!2, ageCareState!2
Postcondition	Timer = 120

unbefriedigend $\Longrightarrow$ unbefriedigend **go_s**	
Precondition	Timer > 0
Input	WeightLTmin?, SatiationLT3?true, HappynessLT3?
Output	mcCareState!2, ageCareState!2
Postcondition	Timer = Timer - 1

unbefriedigend $\Longrightarrow$ unbefriedigend **go_h**	
Precondition	Timer > 0
Input	WeightLTmin?, SatiationLT3?, HappynessLT3?true
Output	mcCareState!2, ageCareState!2
Postcondition	Timer = Timer - 1

unbefriedigend $\Longrightarrow$ unbefriedigend **go_ws**	
Precondition	Timer > 0
Input	WeightLTmin?true, SatiationLT3?true, HappynessLT3?
Output	mcCareState!2, ageCareState!2
Postcondition	Timer = Timer - 1

unbefriedigend $\Longrightarrow$ unbefriedigend **go_wh**	
Precondition	Timer > 0
Input	WeightLTmin?true, SatiationLT3?, HappynessLT3?true
Output	mcCareState!2, ageCareState!2
Postcondition	Timer = Timer - 1

unbefriedigend $\Longrightarrow$ unbefriedigend **go_sh**	
Precondition	Timer > 0
Input	WeightLTmin?, SatiationLT3?true, HappynessLT3?true
Output	mcCareState!2, ageCareState!2
Postcondition	Timer = Timer - 1

unbefriedigend $\Longrightarrow$ unbefriedigend **go_wsh**	
Precondition	Timer > 0
Input	WeightLTmin?true, SatiationLT3?true, HappynessLT3?true
Output	mcCareState!2, ageCareState!2
Postcondition	Timer = Timer - 1

<table>
<tr><th colspan="2">unbefriedigend $\Longrightarrow$ mangelhaft
abuse</th></tr>
<tr><td><b>Precondition</b></td><td>Timer <= 0</td></tr>
<tr><td><b>Input</b></td><td></td></tr>
<tr><td><b>Output</b></td><td>mcCareState!3, ageCareState!3</td></tr>
<tr><td><b>Postcondition</b></td><td></td></tr>
</table>

<table>
<tr><th colspan="2">mangelhaft $\Longrightarrow$ mangelhaft
abuseCont</th></tr>
<tr><td><b>Precondition</b></td><td></td></tr>
<tr><td><b>Input</b></td><td></td></tr>
<tr><td><b>Output</b></td><td>mcCareState!3, ageCareState!3</td></tr>
<tr><td><b>Postcondition</b></td><td></td></tr>
</table>

<table>
<tr><th colspan="2">unbefriedigend $\Longrightarrow$ befriedigend
rescue</th></tr>
<tr><td><b>Precondition</b></td><td></td></tr>
<tr><td><b>Input</b></td><td>WeightLTmin?, SatiationLT3?, HappynessLT3?</td></tr>
<tr><td><b>Output</b></td><td>mcCareState!2, ageCareState!2</td></tr>
<tr><td><b>Postcondition</b></td><td></td></tr>
</table>

<table>
<tr><th colspan="2">befriedigend $\Longrightarrow$ unbefriedigend
w</th></tr>
<tr><td><b>Precondition</b></td><td></td></tr>
<tr><td><b>Input</b></td><td>WeightLTmin?true, SatiationLT3?, HappynessLT3?</td></tr>
<tr><td><b>Output</b></td><td>mcCareState!2, ageCareState!2</td></tr>
<tr><td><b>Postcondition</b></td><td>Timer = 120</td></tr>
</table>

<table>
<tr><th colspan="2">befriedigend $\Longrightarrow$ unbefriedigend
s</th></tr>
<tr><td><b>Precondition</b></td><td></td></tr>
<tr><td><b>Input</b></td><td>WeightLTmin?, SatiationLT3?true, HappynessLT3?</td></tr>
<tr><td><b>Output</b></td><td>mcCareState!2, ageCareState!2</td></tr>
<tr><td><b>Postcondition</b></td><td>Timer = 120</td></tr>
</table>

<table>
<tr><th colspan="2">befriedigend $\Longrightarrow$ unbefriedigend
h</th></tr>
<tr><td><b>Precondition</b></td><td></td></tr>
<tr><td><b>Input</b></td><td>WeightLTmin?, SatiationLT3?, HappynessLT3?true</td></tr>
<tr><td><b>Output</b></td><td>mcCareState!2, ageCareState!2</td></tr>
<tr><td><b>Postcondition</b></td><td>Timer = 120</td></tr>
</table>

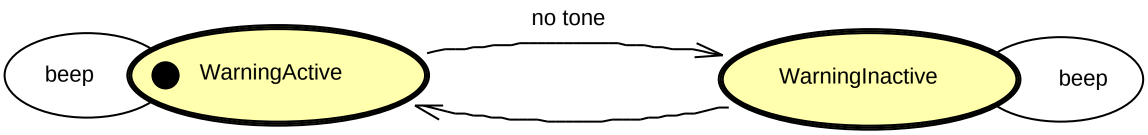
<table>
<tr><th colspan="2">befriedigend $\Longrightarrow$ unbefriedigend
ws</th></tr>
<tr><td><b>Precondition</b></td><td></td></tr>
<tr><td><b>Input</b></td><td>WeightLTmin?true, SatiationLT3?true, HappynessLT3?</td></tr>
<tr><td><b>Output</b></td><td>mcCareState!2, ageCareState!2</td></tr>
<tr><td><b>Postcondition</b></td><td>Timer = 120</td></tr>
</table>

<table>
<tr><th colspan="2">befriedigend $\Longrightarrow$ unbefriedigend
wh</th></tr>
<tr><td><b>Precondition</b></td><td></td></tr>
<tr><td><b>Input</b></td><td>WeightLTmin?true, SatiationLT3?, HappynessLT3?true</td></tr>
<tr><td><b>Output</b></td><td>mcCareState!2, ageCareState!2</td></tr>
<tr><td><b>Postcondition</b></td><td>Timer = 120</td></tr>
</table>

<table>
<tr><th colspan="2">befriedigend $\Longrightarrow$ unbefriedigend
sh</th></tr>
<tr><td><b>Precondition</b></td><td></td></tr>
<tr><td><b>Input</b></td><td>WeightLTmin?, SatiationLT3?true, HappynessLT3?true</td></tr>
<tr><td><b>Output</b></td><td>mcCareState!2, ageCareState!2</td></tr>
<tr><td><b>Postcondition</b></td><td>Timer = 120</td></tr>
</table>

<table>
<tr><th colspan="2">befriedigend $\Longrightarrow$ unbefriedigend
wsh</th></tr>
<tr><td><b>Precondition</b></td><td></td></tr>
<tr><td><b>Input</b></td><td>WeightLTmin?true, SatiationLT3?true, HappynessLT3?true</td></tr>
<tr><td><b>Output</b></td><td>mcCareState!2, ageCareState!2</td></tr>
<tr><td><b>Postcondition</b></td><td>Timer = 120</td></tr>
</table>

<table>
<tr><th colspan="2">befriedigend $\Longrightarrow$ befriedigend
go</th></tr>
<tr><td><b>Precondition</b></td><td></td></tr>
<tr><td><b>Input</b></td><td>WeightLTmin?, SatiationLT3?, HappynessLT3?</td></tr>
<tr><td><b>Output</b></td><td>mcCareState!2, ageCareState!2</td></tr>
<tr><td><b>Postcondition</b></td><td></td></tr>
</table>



<table>
<tr><th colspan="2">WarningActive $\Longrightarrow$ WarningInactive
no tone</th></tr>
<tr><td><b>Precondition</b></td><td></td></tr>
<tr><td><b>Input</b></td><td>careWarningActive?false, SatiationTone?false, nastySatTone?false, WeightTone?false, HappynessTone?false, nastyHappTone?false</td></tr>
<tr><td><b>Output</b></td><td></td></tr>
<tr><td><b>Postcondition</b></td><td></td></tr>
</table>

<table>
<tr><th colspan="2">WarningInactive ⟹ WarningActive
tone</th></tr>
<tr><td><b>Precondition</b></td><td></td></tr>
<tr><td><b>Input</b></td><td>careWarningActive?true, WeightTone?false, SatiationTone?false, nastySatTone?false, HappynessTone?false, nastyHappTone?false</td></tr>
<tr><td><b>Output</b></td><td></td></tr>
<tr><td><b>Postcondition</b></td><td></td></tr>
</table>

<table>
<tr><th colspan="2">WarningActive ⟹ WarningActive
beep</th></tr>
<tr><td><b>Precondition</b></td><td>x1 || x2 || x3 || x4 || x5</td></tr>
<tr><td><b>Input</b></td><td>WeightTone?x1, SatiationTone?x2, nastySatTone?x3, HappynessTone?x4, nastyHappTone?x5</td></tr>
<tr><td><b>Output</b></td><td>dspBeep!true</td></tr>
<tr><td><b>Postcondition</b></td><td></td></tr>
</table>

<table>
<tr><th colspan="2">WarningInactive ⟹ WarningInactive
beep</th></tr>
<tr><td><b>Precondition</b></td><td>x1 || x2</td></tr>
<tr><td><b>Input</b></td><td>nastySatTone?x1, nastyHappTone?x2</td></tr>
<tr><td><b>Output</b></td><td>dspBeep!true</td></tr>
<tr><td><b>Postcondition</b></td><td></td></tr>
</table>