Technical University of Munich
Department of Informatics

Bachelor's Thesis in Informatics: Games Engineering

Dynamic Storytelling Based on Complex Graphs

Dominik Huber

Technical University of Munich
Department of Informatics

Bachelor’s Thesis in Informatics: Games Engineering

Dynamic Storytelling Based on Complex Graphs

Dynamisches Storytelling auf Basis komplexer Graphen

Author:	Dominik Huber
Supervisor:	Prof. Gudrun Klinker, Ph.D.
Advisors:	Daniel Dyrda, M.Sc.
Submission Date:	15.07.2021

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

Munich, 15.07.2021

DOMINIK HUBER

Abstract

Many interactive storytelling tools and plugins for game engines like "Fungus" are based on graphs. An important type of complex graphs are the statecharts extended and modernized by David Harel. Therefore, the topic investigated in this thesis is the extent to which statecharts can be used as environment for representing dynamic stories in games. At the beginning, we summarize the most important statechart feature definitions and introduce some new features that are useful for story representation. Thereupon, it is determined, which storytelling elements can be represented by which statechart features. With the help of example statecharts and a paper-prototype representing example game scenarios, it can be demonstrated that this mapping makes it possible to successfully use statecharts as environment for representing dynamic storytelling in games. Thereby, the disadvantages, like the fast-arising complexity due to many dependencies between parallel components, the lack of a uniform syntax and the ambiguous semantic, are dominated by the advantages. Example benefits would be the structured overview and the visualization of the hierarchical or parallel structure that reveals logic gaps in the story as soon as they are created. Additional advantages also lie in the factor that interactive elements and the interaction of the story with game-mechanics and with the game itself can be implemented very well by statecharts using actions, activities, or trigger events. Furthermore, the existing statechart-feature-canon can be directly used in large parts to represent storytelling elements and can also be easily extended by further features, like the extended history entrance or the feeder states compound.

Keywords: Statecharts, dynamic storytelling, mapping, statechart features, prototype

Acknowledgements

The author would like to thank Prof. Dr. Gudrun Klinker for the interesting topic of this paper. In addition, deepest thanks go to the supervisor Daniel Dyrda for the always informative support and discussion on topics of this thesis.

Contents

Abstract **iv**

Acknowledgements **v**

1 Introduction **1**

2 Related Work **2**

3 Storytelling and Statecharts **4**
3.1 Storytelling 4
3.1.1 Dynamic Storytelling 4
3.1.2 Story Elements 4
3.2 Definition of Statecharts 5
3.3 Selected Statechart Features 5
3.3.1 State-Types 5
3.3.2 Event-Types 9
3.3.3 Transition-Types 10
3.3.4 Actions and Activities 12
3.3.5 Refinement and Abstraction 13
3.3.6 Abstract Concepts 14
3.4 Final States and Exits 16
3.4.1 Definition Final States 18
3.4.2 Completion Event Handling with Final States 18
3.4.3 Definition Exits, Exit Receivers and Implicit Exit States 21
3.4.4 Completion Event Handling with Exits 22
3.4.5 Differences between Final State and Exit 27
3.4.6 Reasons for Leaving-Option of Final States and Implicit Exit States 29
3.5 New Statechart Features and Pattern 29
3.6 Mapping Storytelling Elements onto Statechart Features 36

4 Prototype **48**
4.1 Advantages of a Paper-Prototype 48
4.2 Interaction of Statechart with Game Environment in the Prototype-Context 49
4.3 Setting of the Prototype 53
4.4 Quests and Interrupts 53
4.5 Story of the Prototype 55
4.6 Parallel Components in the Prototype 59
4.7 History Entries and Final States 61
4.8 Results of the Prototype 62

5 Outlook **63**

6 Conclusion **64**

Appendix **71**
1 Base States of the Knight 72
2 Paper-Prototype 77

1 Introduction

"Wenn die drei ??? sich zeigen und mit den Erwachsenen reden sollen, lies weiter auf Seite 84. Wenn die drei ??? sich doch lieber zurückziehen sollen, lies weiter auf Seite 87." (English: "If you want the three investigators to show up and talk to the adults, continue reading on page 84. If you would rather have the three investigators withdraw, continue reading on page 87.") [1]. In this snippet from a German book of the series "The Three Investigators", the reader has the choice to determine how the three protagonists should proceed in the detective story. Books in which the reader decides the progression of the story by continuing reading on a particular page are classified as "choose your own adventure" or "find-your-fate" books. This book category uses dynamic storytelling. This means that the reader creates the storyline of the book while reading by choosing between already predefined paths. Therefore, every decision influences the development of the storyline. Although the possible sub-paths and the various endings are predefined in such books, the final storyline is created by the reader's decisions. However, dynamic storytelling is not only used in books, but also in many story-based video games, where the player's actions and decisions affect the development and outcome of the storyline. Examples are the horror game "Until Dawn" or the action-adventure game "Detroit: Become Human" [2][3]. On wiki and game-guide webpages, the developer or the player-community often publish overviews, which decisions or actions lead to which results in these games. As visualization format, in most cases they choose something similar to a graph. This leads to the idea that dynamic storytelling could also be implemented in video games using graphs. Since in such graphs states of the storyline must be modeled, statecharts would be a suitable approach. These diagrams, modernized and expanded by David Harel, originate from the world of reactive technical embedded systems, like aviation [4]. In essence, dynamic stories in games are reactive systems. Hence, this thesis examines the extent, to which statecharts can be used as environment for representing dynamic stories in games.

2 Related Work

As mentioned in the introduction, storylines of interactive storygames are often illustrated by informal graphs. For the interactive game "Detroit: Become Human", for example, an overview of which action and decision results in which situation can be found on the website "Powerpyx" [5]. As the cutout of this overview in figure 2.1 shows, this is an informal diagram that indicates the various decision options on the nodes and refers to the next resulting decisions with unlabeled transitions. Therefore, this could also be seen as an informal statechart, which describes a storyline.

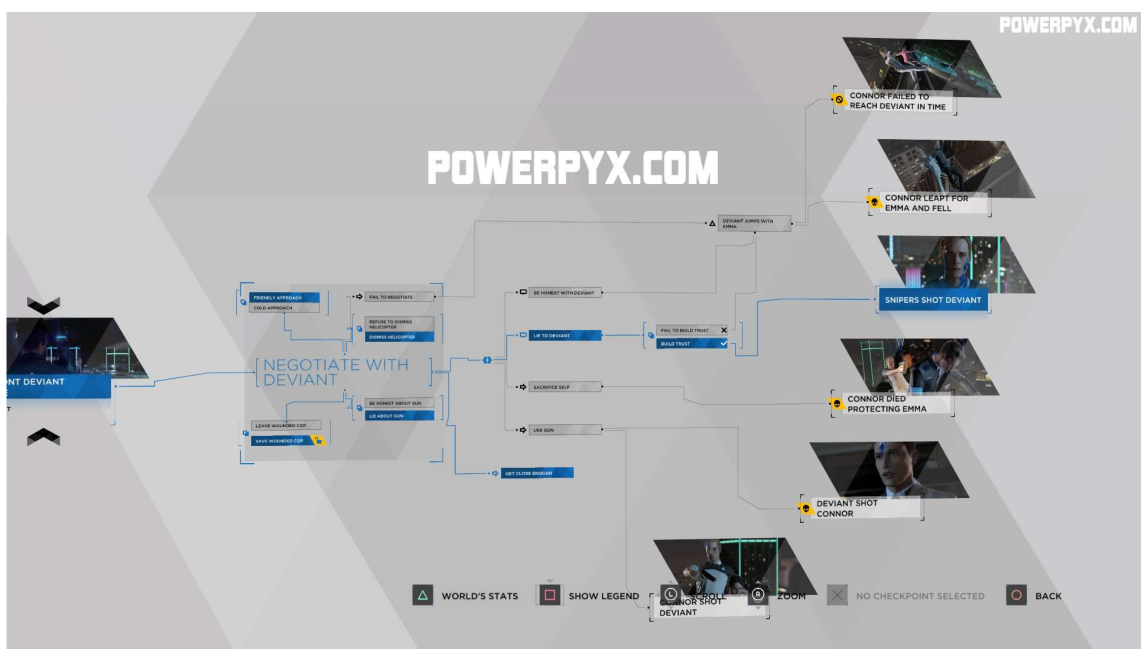


Figure 2.1: Cutout of overview of decision-path of the game "Detroit: Become Human" (Source: [5])

Statecharts are not the only way to achieve interactive storytelling in games. For game engines like "Unity 3D", there are countless plugins and tools that enable narrative games and interactive narratives. Example tools would be "Inky", "Twine", "YarnEditor", "Talkit" or the "Fungus" plugin. The latter uses flowcharts to implement interactive storylines in "Unity 3D" [6]. The developer edits the story by working directly in the nodes or transitions of these flowcharts [6]. Figure 2.2 shows an example flowchart from "Fungus" in "Unity 3D", representing a short game scenario. This tool follows a very similar approach to the one used in this thesis, except that instead of statecharts, it uses flowcharts.

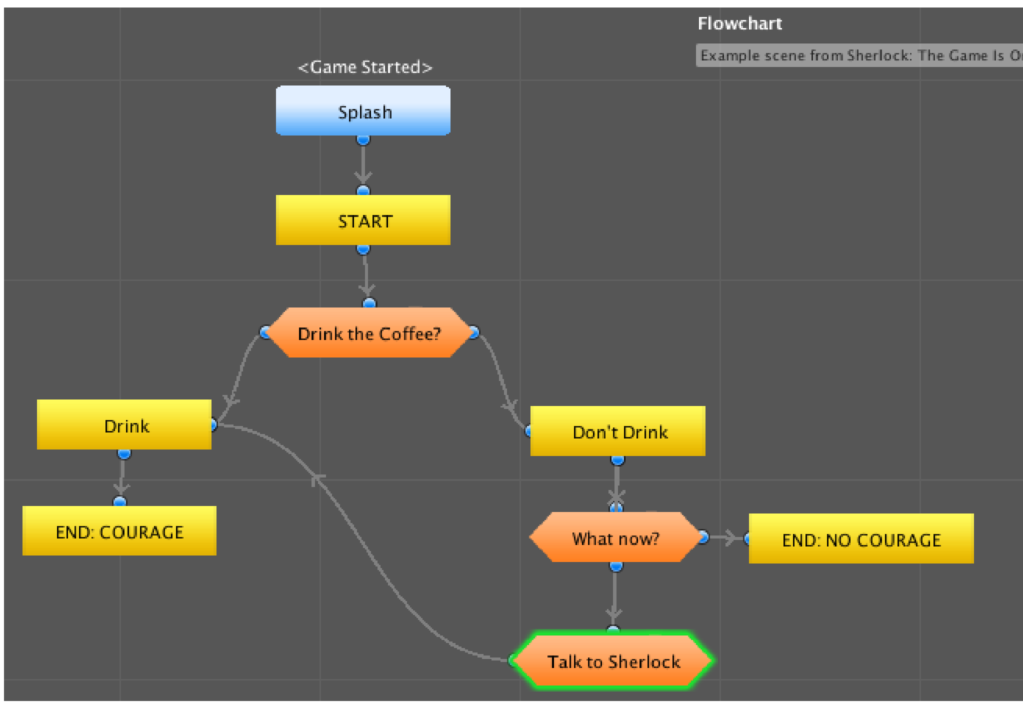


Figure 2.2: Example game scenario from "Fungus" in "Unity 3D" (Source: [6])

To the ideas presented in this theses related approaches can not only be observed in editors. Also, in scientific papers considerations to dynamic storytelling and their best implementation in games exits. For example, in the paper "Character-Based Interactive Storytelling" by Cavazza, Charles and Mead, a hierarchical task network is used to implement the decision options of a character to achieve the main goal [7]. Other approaches to achieve dynamic stories are presented in the paper "Interactive Storytelling: Approaches and Techniques to Achieve Dynamic Stories" by Merabti et al. [8]. There, planning algorithms are investigated that select the events, the timing, and the way of presenting the events the player experiences [8]. Also, behaviors of emotional non-playable characters and their relationships to the player are addressed to enable more realistic storytelling [8].

Another area of interest that is strongly related to statecharts is SCXML. This is an attempt to convert XML in such a way that it can be used to render statecharts [9]. The way in which XML is adapted to obtain SCXML is described in the paper "Developing Natural Language Enabled Games in (Extended) SCXML" by Brusk and Lager [9]. Since SCXML is merely a realization format of statecharts, it seems obvious that it can also be used for storytelling. In the paper "DEAL - Dialogue Management in SCXML for Believable Game Characters", for example, Brusk et al. describe a way to use SCXML to manage dialogs so that believable non-playable characters are created [10].

3 Storytelling and Statecharts

3.1 Storytelling

3.1.1 Dynamic Storytelling

There can be no better story than one that you write yourself. In this way, the story meets all own expectations about what the story should be and how it should best develop. Even character decisions can be determined by oneself. The big disadvantage of own stories is that the tension is lost, because one inevitably knows how one's own story ends. But that does not necessarily have to be the case if the distant future is not yet fixed. This is exactly the case with dynamic storytelling. Through decisions and actions, the reader, or in video games, the player, can shape his/her own path. The story changes dynamically. This means that in advance it is not clear, how exactly the story will unfold. Similarly, Alain Leclerc von Bonin described the core characteristic of dynamic storytelling in an interview in the context of a presentation back in 2015. He said that "the major characteristic of dynamic storytelling is not having a fixed narration in advance" [11].

3.1.2 Story Elements

Stories can generally be described as a sequence of events involving entities [12]. As this definition indicates, a story consists of many different story elements. The basic framework for the story is set by the setting. It describes the background where the action takes place and the place of the story within the storyworld. The term "storyworld" describes "the shared universe in which the settings, characters, objects, event, and actions of one or more narratives exist." [13]. Examples for storyworlds would be the "Star Wars" or the "Harry Potter" universe. Furthermore, in every story there is a main character and an antagonist. The former can be a hero or an antihero. In video games, the protagonist is usually the character through whom the player dives into the story and interacts with the gameworld.

The story patterns "3-act scheme" or the "Hero's Journey" often serve as a structure for stories. However, these will not be discussed in more detail in this paper. Instead, structure elements and storytelling methods and components are of more relevance for this thesis. In video games, the main storyline and multiple parallel side storylines – also called plot and sub-plot – can contain quests and interrupts. The difference between the two terms is explained in the chapter "Quests and Interrupts" using the prototype as an example. In addition to events and actions of the characters, within the quests and interrupts, various storytelling methods can be applied. For example, characters can have dialogs, foreshadowing of future events can take place, or flashbacks can refer to earlier events.

However, this paper is not about classical dynamic storytelling, but about realizing dy-

namic storytelling through statecharts. Therefore, we now move on to these complex diagrams.

3.2 Definition of Statecharts

When we are speaking about statecharts in this thesis, we are referring to statecharts according to Harel's definition in the paper "Statecharts: A Visual Formalism for Complex Systems" from 1987. He described: "Statecharts constitute a visual formalism for describing states and transitions in a modular fashion, enabling clustering, orthogonality (i.e., concurrency) and refinement, and encouraging 'zoom' capabilities for moving easily back and forth between levels of abstraction." [14].

3.3 Selected Statechart Features

When considering statechart features, the definitions of these are mainly based on the same paper from which the definition above was taken. We refer to this paper, since Harel modernized and extended the statecharts with his publication. Today, the features described in it are still implemented in editors, like the "VisualParadigm Online"-editor, in the same form, with only a few minor changes [15]. In addition, almost all papers about statecharts refer to the statechart features of Harel defined in the paper of 1987. These features are also very easily customizable and extensible, which will be used later in this thesis in the "New Statechart Features and Pattern"-section 3.5.

Before examining the mapping of storytelling elements onto statechart features, we will take a closer look at selected groups of statechart features of great importance. While discussing those central elements of statecharts in more detail and elaborating the differences between the various types, additional reference is made to recent literature and to the implementation in statechart editors or projects like the "statecharts.dev"-project, see [16].

3.3.1 State-Types

States are the nodes of the statechart [14]. These can be fundamentally divided into concrete and non-concrete states [17]. The state machine can only be resting in concrete states, whereas in non-concrete states the state machine immediately continues to transit [17]. Therefore, non-concrete states are also called "pseudostates" or "transient states" [17]. The first example of the concrete type is the atomic state, also known as simple state. In Figure 3.1 "A", "B", "C", "E", "F", "G" and "H" are atomic states. This state-type has no substates [17]. It should be mentioned that substates are states within other states, the super-states [14]. In Figure 3.1 "OrState" and "AndState" are super-states while "A", "B", "C", "E", "F", "G" and "H" are substates. If a substate is active, the super-state also counts as active. So, in Figure 3.1 when "A" is active, also "OrState" is active. A compound state, on the other hand, has substates. Examples of compound states are the XOR state – where only one of the substates can be active at a time – and the AND state – also called parallel state or orthogonal state [14][17]. In this state, the substates are divided into separate parallel regions which run in parallel [17]. Here, several orthogonal substates can be active at the same time. In the context of orthogonality, a state must be

active in each parallel region [14]. Harel visualizes parallel components in the way it is done in Figure 3.1 in the "AndState", but the label of the whole component is not placed in the header but is written in an outer box added to the AND state [14]. The labels of the parallel regions are noted at the top of the respective region in both visualization forms. In Figure 3.1 in the "OrState" either "B" or "C" can be active, while in the "AndState" in "D" both "E" (or "G") and "F" (or "H") are active at the same time. XOR states require a branching decision, which results in two or more exclusive, independent paths [14]. With AND decomposition, on the other hand, a single event leads to two or more simultaneous happenings [14]. This is a certain kind of synchronization [14]. Within the AND block prevails independence between the two paths, but this can be broken by an "in <stateName>"-condition [14]. This condition of a transition ensures that a substate in an AND-component depends on another substate being active in a different parallel region [14]. In Figure 3.1, "G" in "Region1" just gets active, when "H" is active in "Region2". So, this creates a dependency between AND-components.

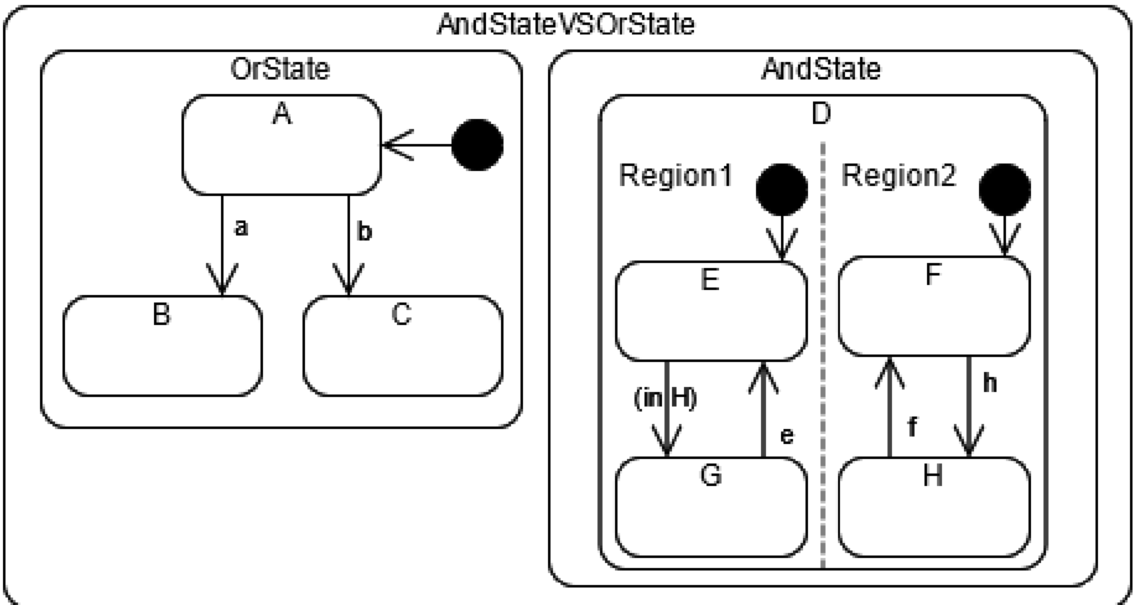


Figure 3.1: OR state and AND state

Besides atomic states and compound states there are final states and implicit exit states. However, both will be discussed in more detail in the "Final States and Exits" chapter 3.4 and are therefore not described further in this chapter. Two rather rarely occurring concrete state types are parametrized states and overlapping states. As shown in figure 3.2 parametrized states have a special form of visualization due to their identical inner structure, which differs only in one parameter [14].

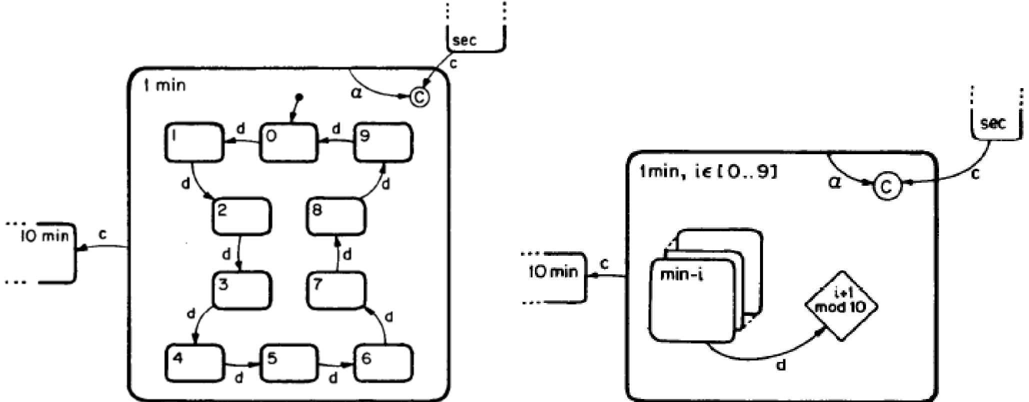


Figure 3.2: Situation from left is shown on the right side with the help of a parametrized state (Source: [14], Fig. 38, 39)

Overlapping states are substate of two super-states simultaneously. Figure 3.3 shows the visual representation of overlapping states. Reasons for using these states are conceptual similarities between the involved states or more convenient way to model joint exits [14]. By overlapping states, it is now also possible that OR states emerge and no longer only XOR states exist. This means that in figure 3.3, "A" or "D" can not only be exclusively active, but "A" and "D" can be at the same time active if state "C" is active.

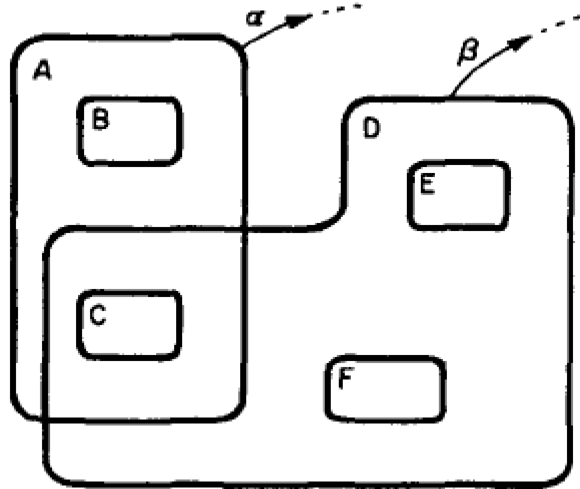


Figure 3.3: Overlapping states (Source: [14], Fig. 41)

After all concrete states have been listed, the non-concrete states follow. The default state, also called initial state, start state, or default-entrance, is used in almost every statechart. It marks the first active state of the whole statechart or just a compound state [14]. In a compound state the default state is the substate that is entered by default when the super-state gets active [17]. It is visualized with a black circle that is left by one transition. An example for an initial state can be found in figure 3.1 in the "OrState". Furthermore, exit and exit receiver are non-concrete states. But just as with the concrete final states and implicit exit states, these will be discussed in more detail in the chapter "Final States and Exits". Two other non-concrete states to reduce transitions are the condition states and the selection states. Condition entrances choose the substate to enter based on a condition

[14]. In the visualization, transitions with only a condition leave the condition entrance state, represented by a circled C, and target all substates. In figure 3.4, we can see that with a conditional entrance, the statechart appears more organized than when traversing into the substates unbundled from directly outside the super-state. Furthermore, the visual representation indicates that conditional entrances can be compared with an if-else block in programming languages like "Java" or "C".

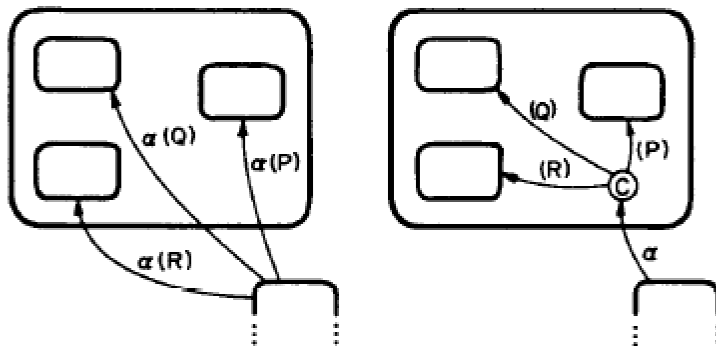


Figure 3.4: Comparison of situation with conditional entrance and without (Source: [14], Fig. 33)

The selection entrance, on the other hand, can be compared with a switch statement of these programming languages. In this entrance, the substate to be entered is determined based on a generic event [14]. The event is the selection of one of several clearly defined options, which are placed as triggering events on each transition between the selection state and the substates. This means that where in figure 3.4 conditions are written on the transitions, in the case of a selection entrance events would be written, one of which occurs in each case. Similar to the comparison entrance, the selection entrance state is represented by a circled S.

The last non-concrete state type to be mentioned at this point is the history state. This pseudo-state remembers the most recent sibling states that were active [14]. So, when the history state becomes active, it is not dwelled in it, since it is a non-concrete state-type, but it immediately moves on to the most recently visited state, which then gets active. History states are divided into shallow and deep history states [17]. Shallow history states appear in the statechart as an encircled H. They only remember which of their direct siblings was active last. This one then becomes active again. But if the sibling has more than one substate, the default state within the sibling of the history state becomes active and not the last active substate in it, because the history does not know which of the substates was active last. This is different for the deep history state. This state represented with circled H* remembers the deepest active state. So, the last active substate of the last active sibling of the deep history state becomes active. In figure 3.5 in "ShallowHistory" the shallow history only remembers if "A" or “B” was active last. If "B" was active, "C" will be active in "B" due to the default state, even if "D" was active last, because shallow history only knows if "A" or "B" was active most recently, but not which one of "C" or "D" was active. In "DeepHistory" the deep history remembers if "E", "G" or "H" was active last. So, the last active state gets active again no matter if it is a direct sibling of the history state or the deepest substate. When the history state is entered for the first time, the state to which the transition points becomes active. In figure 3.5, these are state "A" and "E". It should also be mentioned that transitions which originate directly at a super-state, i.e., in figure 3.5 the transitions with the events "a" and "e", can

be traversed out of all substates of the respective super-state when the respective event occurs. So, if event "e" occurs while the system is in state "G" or in "H", the transition to state "E" is activated.

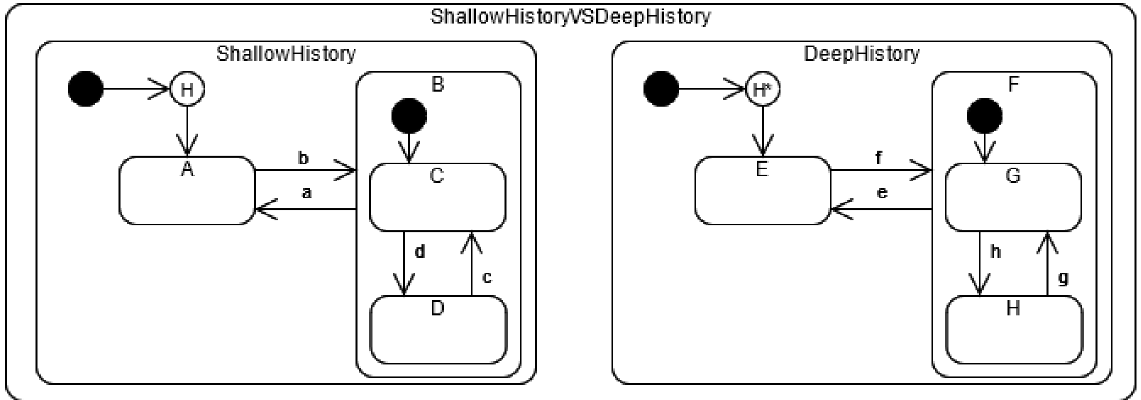


Figure 3.5: Shallow history and deep history

In addition to the division of states into concrete and non-concrete types, states can also be characterized by their relation to other states. Two states can be exclusive, orthogonal or ancestoral. Exclusive means that the states are not on the same path and their last common ancestor is an XOR state [14]. Orthogonal states are not on the same path as well and their last common ancestor is an AND state [14]. Ancestoral states are on the same path [14].

3.3.2 Event-Types

Events, also called triggering events, are responsible for triggering transitions in statecharts and therefore are written next to these transitions [14]. If the situation described by the triggering event occurs, the transition becomes active, and the target state of the transition is entered. In figure 3.5, all lowercase letters are triggering events. A basic distinction is made between external and internal events. The former signalizes that something has happened [17]. In contrast to internal events, external events are sent from the outside of the system to the statechart, which then responds to these [17]. On the other hand, internal events are nameless events that are generated automatically within the statechart [17]. This can be the case when a transition occurs, when a service started by the statechart completes, or when an exit or final state is entered [17]. Internal events can be sensed by transitions that specify events such as the "in <stateName>"-condition introduced in the section 3.3.1 "State-types" [17].

As with states, events also occur in different types. The first one is the signal event. This asynchronous event is queuing until the element is ready to process it [18]. The sender "sends and forgets" the event [18]. That means that it does not know if the event has been successfully processed. Also, a signal event can be broadcasted to all objects of a system [19][20]. Therefore, it is plausible that exceptions can be named as an application example for this type of event [19][20]. The notation for signal events used in this thesis is "<Signal> (<Condition>) / <Action>". The condition and the action, both of which will be discussed later, can be omitted for all event types presented here. Also, further conditions can be added via "&&" or "||" and additional actions can be attached via "&".

Unlike the asynchronous signal event, the next event type, the call event, is synchronous.

Here the sender is blocked from continuing until the state processing it is completed [18]. As notation in this paper "<Operation> (<Condition>) / <action>" is chosen for call events, where "<Operation>" represents the call event.

The next type of event is the time event or sometimes only called timeout. This event is triggered due to the expiry of a time interval [18]. If the state with the timeout interval is re-entered or receives another event, the interval starts again from the beginning [18]. If it has expired, the transition triggered by the time event is traversed [18]. These events can be used to synchronize time-dependent parts of the system [19]. Also, they will appear in delayed transitions, which are discussed in the section 3.3.3 "Transition Types". In this paper we will use the notation "after <amountOfTime> s (<Condition>) / <Action>".

Another event type is the change event. It represents a change in a state variable or in an attribute of the classifier [18]. To trigger change events, a condition must be met by the change [19]. A generic example for a change event would be "when(x > y)" [20]. Here the condition that the state variable "x" is greater than the variable "y" must apply, so that the corresponding transition is travelled. The notation can also be extended by conditions and actions as in the previous types: "when(<SomeSortOfTest>) (<Condition>) / <Action>".

The last event-type is the completion event or null event. As the name suggests, there is no specific trigger event [18]. The event occurs as soon as the state is entered or when the state's activities are completed [18]. Nevertheless, the entry actions are executed in the state before the completion event occurs [18]. The notation also lacks the actual event: "(<Condition>) / <Action">

As a small preview of the mapping table in section 3.6 "Mapping Storytelling Elements onto Statechart Features", we will conclude this section by listing possible elements from the game context that can be represented by events in statecharts. As can be seen later from example statecharts, game elements depicted by events are versatile. Events can represent changes in the gameworld, events or situations occurring in this world or interactions with it. Also, effects and results of the interaction with non-playable characters or the gameworld as well as effects and results of player actions can be depicted by events. But not only the outcomes of the player's actions can be mapped onto events, but also the actual actions and decisions of the player, as well as technical input and changes of the game environment in the background, like "OnTriggerEnter", "OnButtonDown", "OnKeyDown" or "Timeout" can be represented by events.

3.3.3 Transition-Types

Transitions are the arrows of statecharts [14]. Here several different types exist, too. The first transition, the automatic transition, also called completion transition or eventless transition, uses the completion event. Therefore, it is a transition without a triggering event that is traversed as soon as the state is entered and all entry actions and activities are completed [17]. Automatic transitions are usually guarded [17]. That means they have a condition, also called guard, which is checked if the state they are leaving is active. This condition is not an active trigger but is just a barrier that must be true for travelling the corresponding transition [14]. Multiple conditions can be combined with "&&" or "||". If conditions are connected with "&&", it means that all conditions must be fulfilled for the result to be true, and if conditions are linked with "||", only one must be fulfilled for the outcome to be true. For comparisons in conditions the logical operators are used: <, >, <=,

>=, !=, ==. In this paper conditions are written in brackets behind the triggering event and before the action, which is separated with a "/". In some literature and editors, conditions must be written into square brackets. But after the short excursion to conditions back to automatic transitions: In figure 3.6, the transition leaving state "A" and targeting state "B" with the generic condition "(x > 3)" and the transition leaving "B" and entering "C" without a condition are examples for automatic transitions. However, the latter is only traversed as soon as the entry action "x=x+1" and the activity "activityB" are completed.

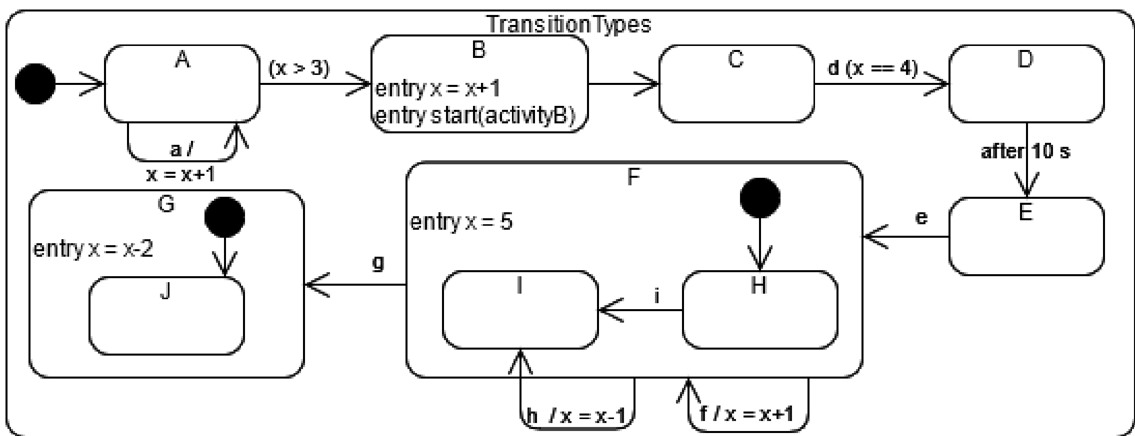


Figure 3.6: Different transition-types

In figure 3.6, the transition from state "C" to "D" is a conditional transition. To travel this transition, the attached condition must be met [17]. If the condition is not fulfilled, the transition is ignored [17].

The next transition-type is the delayed transition, which occurs in figure 3.6 between states "D" and "E". This transition uses a time event to become active after a certain period of time. As already mentioned in the explanation of the time event, the state from which the transition starts must have been active continuously until the end of the period. Otherwise, the time interval starts again from the beginning [17]. Also, self transitions, which is another transition-type, interrupt the continuity since they restart the state [17]. This means that when leaving and re-entering the same state, the exit and entry actions are executed and the timer that indicates how long the state has been active is restarted [17]. An example of a transition that enters the same state it leaves is the self transition in "A" with the triggering event "a" and the action "x=x+1" in figure 3.6. When a self transition leaves and re-enters a compound state all active substates are left and the default state gets active [17]. Thus, in figure 3.6, when the self transition with the event "f" and the action "x=x+1" is triggered in "F", state "H" becomes active after the entry action "x=5" has been performed, regardless of whether "I" or "H" were active before. Transitions leaving a super-state like this self transition or the transition with the event "g", can be triggered in each substate and when this happens all substates are left [14]. Therefore, when in "I" or "H" "g" occurs, the whole super-state "F" is left to state "G".

Self transitions are also examples of external transitions. Unlike local transitions, these transitions are from one super-state to another super-state, performing exit and entry actions [17]. Examples of external transitions are the transition with the trigger event "g" between "F" and "G" and the self transition in "F" in figure 3.6 For these, the actions "x=x-2" or "x=5" are executed when they enter their target state. Local transitions, or internal transitions, as they are also called, do not leave the super-state, and therefore do

not perform exit or entry actions [17]. They occur between substates within a super-state, like the transition between "H" and "I" in figure 3.6, or they appear between the super-state itself and a substate of it, like the transition with the triggering event "h" and the action "x=x-1" in figure 3.6. Since both examples are local transitions, the entry action "x=5" of state "F" is not executed when these transitions get traversed.

Furthermore, there are transition hubs, which try to combine transitions through a pseudo-state and thus require fewer transitions and provide more structure. Examples would be transitions with common targets (figure 3.7 (a)), triggering events (figure 3.7 (b)), or sources (figure 3.7 (c)) [14]. Transition hubs for common sources and common targets are often used in AND-components [14].

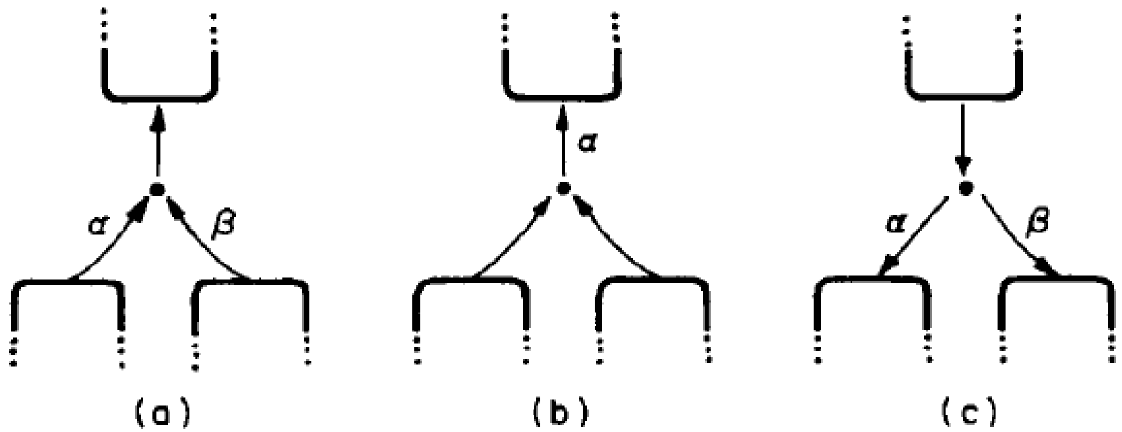


Figure 3.7: Transition hubs (Source: [14], Fig. 17)

3.3.4 Actions and Activities

In the preceding chapters, the keyword "action" has often been mentioned. Now it will be explained in more detail. Actions are split-second happenings that occur instantaneously and ideally take zero time [14]. Furthermore, they can generate events, which in turn can immediately trigger a transition in an orthogonal component without necessarily having any immediate external effects [14]. The created event will have the same name as the action and the transition labeled with this triggering event gets immediately traversed [14]. Moreover, actions can change the value of a condition or a variable [14]. They can be placed in states or can be attached to the label of transitions by the notation "<event> (condition>) / <action1> & <action2> & . . . ". In states, actions can be performed when entering ("entry <action>") or leaving the state ("exit <action>"). In some papers and editors, the notation "entry / <action>" is also common for actions in states. Another characteristic of actions is that they cannot be interrupted by events [19].

This does not apply for activities. These can be interrupted by events [19]. Activities are durable. That means they will be carried out continuously throughout the state being active [14]. The notation for activities in states is "throughout <activity>". Since they are only allowed in states, there is no notation for activities attached to labels of transitions. In addition, activity "X" can be started by the action "start(X)" and stopped by the action "stop(X)" [14]. "throughout <activity>" combines "entry start(<activity>)" and "exit stop(<activity>)" [14]. Furthermore, the new condition "active(X)" gives feedback whether the activity "X" is still active or not [14].

A special action that should be mentioned at this point explicitly is the clear-history action. This action causes forgetting of recently visited states on the first level or on all levels [14]. After this action has been performed, history states no longer remember the last active state and therefore apply their default transition as if the area containing the history entrance had never been entered before [14]. Clear-history actions are allowed in states and can be attached to labels of transitions. In states, they are noted as "clh(<stateName>)" when only the first level of history in the state "<stateName>" should be forgotten or when all levels of history in the state "<stateName>" should be deleted, they are written down as "clh(<stateName>*)". Attached on the label of states the notation "<event> (<condition>) / clh(<stateName>)", respectively "<event> (<condition>) / clh(<stateName>*)" is used.

In figure 3.8, the application of actions and activities are demonstrated in a practical example statechart. If the event "d" is triggered in state "C" at the beginning, the action "ActionC" is executed, and state "D" is entered afterwards. In this state, activity "ActivityD" is played continuously. However, if event "b" is triggered, "A", and thus also substate "D", is left. The "ActivityD" is also interrupted at this point. In the now active parallel component "Region1", action "h" is executed when state "F" is entered. This generates an event of the same name "h", which in the parallel component "Region2" causes state "G" to be exited to state "H". If event "g" occurs in this state, the clear-history action is executed during the transition back to "G", whereby the history on the first level is forgotten in state "A". This means that if event "a" occurs and state "A" is entered again, the history state no longer knows that "D" was last active when "A" was exited. Since the default transition of the history state points to "C", "C" would now become active instead of "D".

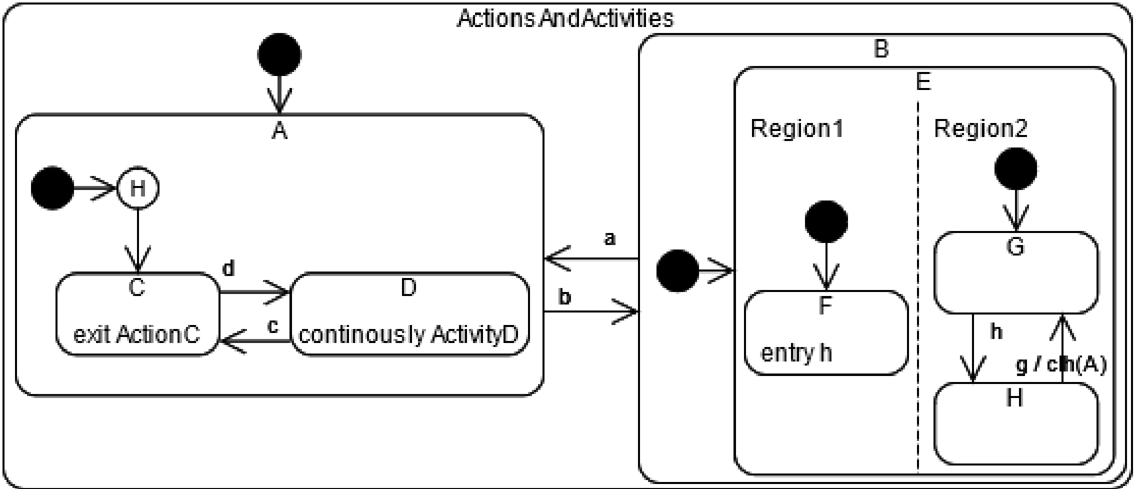


Figure 3.8: Application of actions and activities

3.3.5 Refinement and Abstraction

Unlike the features discussed so far, refinement – zooming-in –, and abstraction – zooming-out – are not features that can occur directly in statecharts but are processes that can be performed on statecharts to convert them into white-box-view or into black-box-view. The white-box-view describes a visualization of a super-state in which all substates and transitions of it are shown in detail. In the black-box-view, on the other hand, super-states

can be represented as black boxes into which one cannot look. In this way substates and transitions can be hidden. The black-box-view is mainly used when the inner structure of super-states is irrelevant or has not yet been determined. To express in the black-box-view that a transition leaving the black box originates from a specific substate and not from the entire super-state, the transition is visualized by an arrow starting from a bar inside the black box [14]. And the other way round: Transitions from outside the black box to a specific substate inside the black box are represented as arrows that end on a dash inside the black box [14]. To get into the black-box-view – so zooming-out –, the substates and transitions of the respective super-states are omitted, and incoming and outgoing transitions are represented by the two new visualization options for transitions [14]. This can be seen in figure 3.9, where the right statechart is the left one but in black-box-view. To get into the white-box-view – so zooming-in – , the inner structure of the super-state is displayed and the outer elements are omitted [14]. This is visualized in figure 3.10, where the right statechart is the "A"-state of the left one in white-box-view.

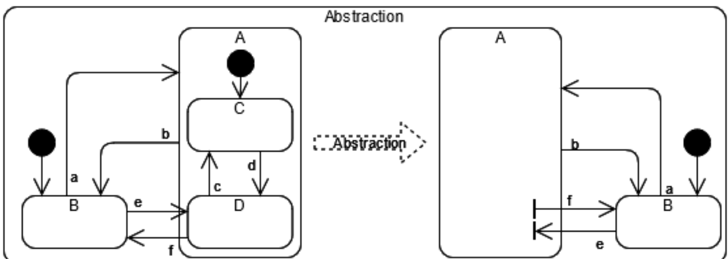


Figure 3.9: Abstraction to black-box-view

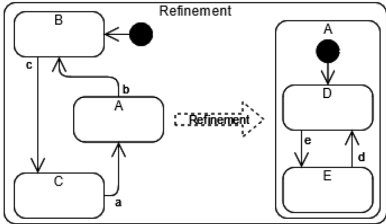


Figure 3.10: Refinement to white-box-view

3.3.6 Abstract Concepts

This section covers some features that are mentioned only briefly in Harel's "Statecharts: A Visual Formalism for Complex Systems". Nevertheless, they must be explained, as they are also included in the mapping in section 3.6 "Mapping Storytelling Elements onto Statechart Features".

Unclustering is a feature that provides improved overview in statecharts [14]. For this purpose, parts of the statechart are placed outside their natural location [14]. This would also be a feature that could enrich a possible statechart framework in game engines. For example, a developer could be given the option that when he/she clicks on states, they appear in large size outside the statechart. This could also be realized in the combination of unclustering with the concepts of refinement and abstraction. Figure 3.11 shows a possible approach for unclustering.

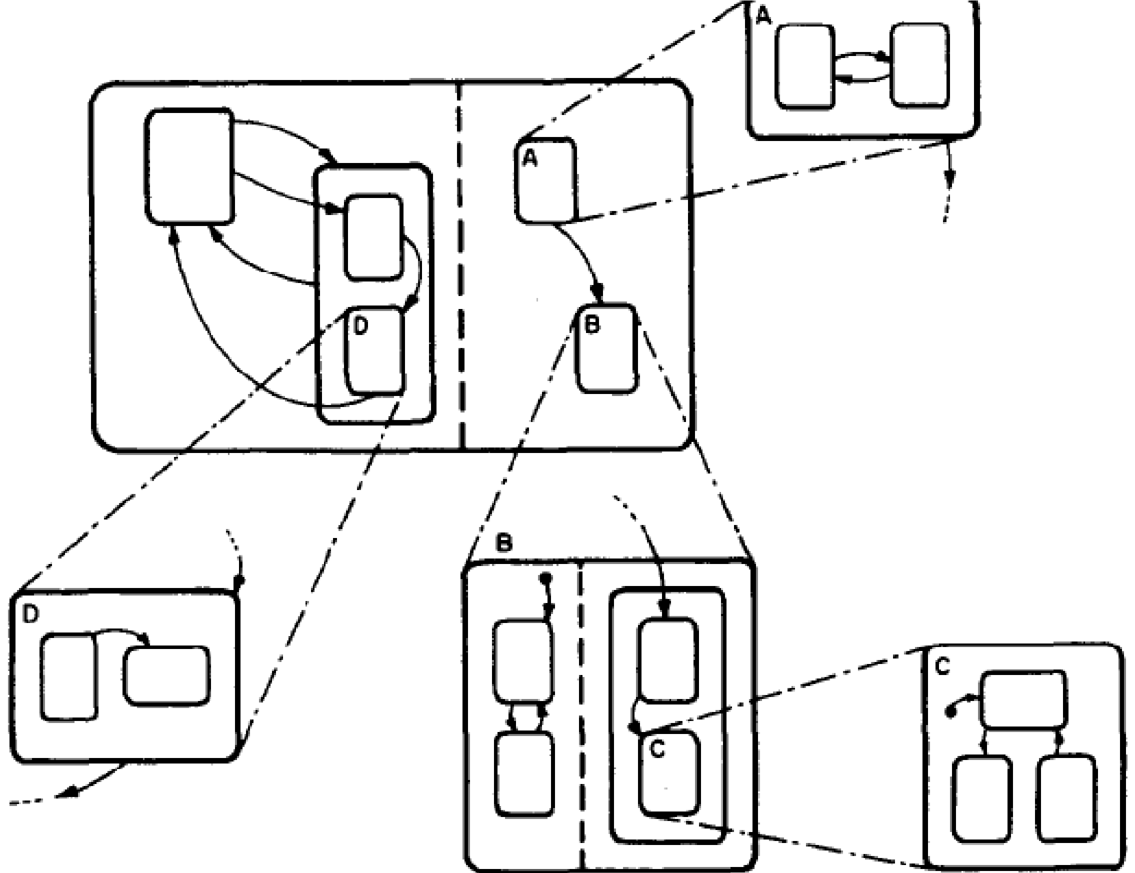


Figure 3.11: Unclustering (Source: [14], Fig.36)

Temporal logic, which is already used in many programs, can also be beneficially applied to statecharts [14]. Temporal logic can be used, for example, to specify many types of global conditions, like timing constraints, eventualities, or absence from deadlocks in advance [14]. The temporal logic clauses must then be met by every statechart [14]. However, since temporal logic is another far-reaching area that would be too large for this paper, it will be left at that.

Even since, according to Harel, there are only a few use cases for it in practice, recursion can also prevail in statecharts [14]. In this case, the name of a separate statechart, of which the present one is a special case, is specified in the label of a transition of the present statechart [14].

Statecharts in pure form are deterministic [14]. This can be overcome if probabilism is added. Thus, one could allow nondeterminism and specify a bias on the coin to be tossed when it arises [14]. Nondeterminism could be used in a variety of ways in dynamic and interactive storytelling, like for an increasing probability of a dragon attack when the player gets stronger or for non-playable characters, which enable non-deterministic dialogs. However, these considerations are too far-reaching for this paper, as the focus lies more on the fundamentals.

3.4 Final States and Exits

Ending things plays a big role in storytelling. Be it in ending dialogs, tasks, quests, subplots, backstories, the main storyline or even the lives of characters. Nevertheless, as figure 3.12, figure 3.13 and figure 3.14 show, the author had to realize that neither in editors nor in literature, there is a uniform definition for final states or exits of statecharts. For this reason, the following chapters will attempt to develop a uniform definition of final states and exits as well as determine their precise mode of operation.

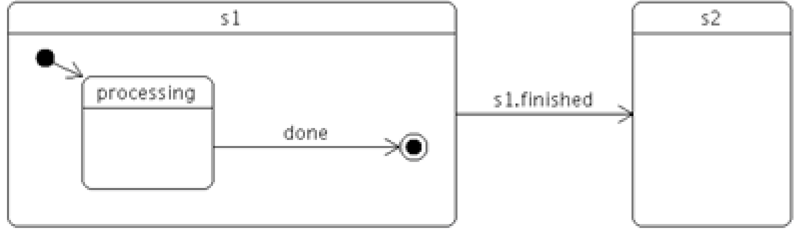


Figure 3.12: Final state and finished-signal of "QT Core - The State Machine Framework" (Source: [21])

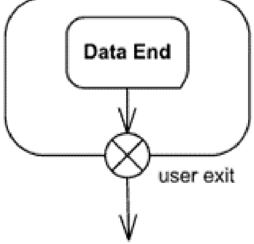


Figure 3.13: Exit point of „UML Diagrams – State Machine Diagrams" (Source: [22])

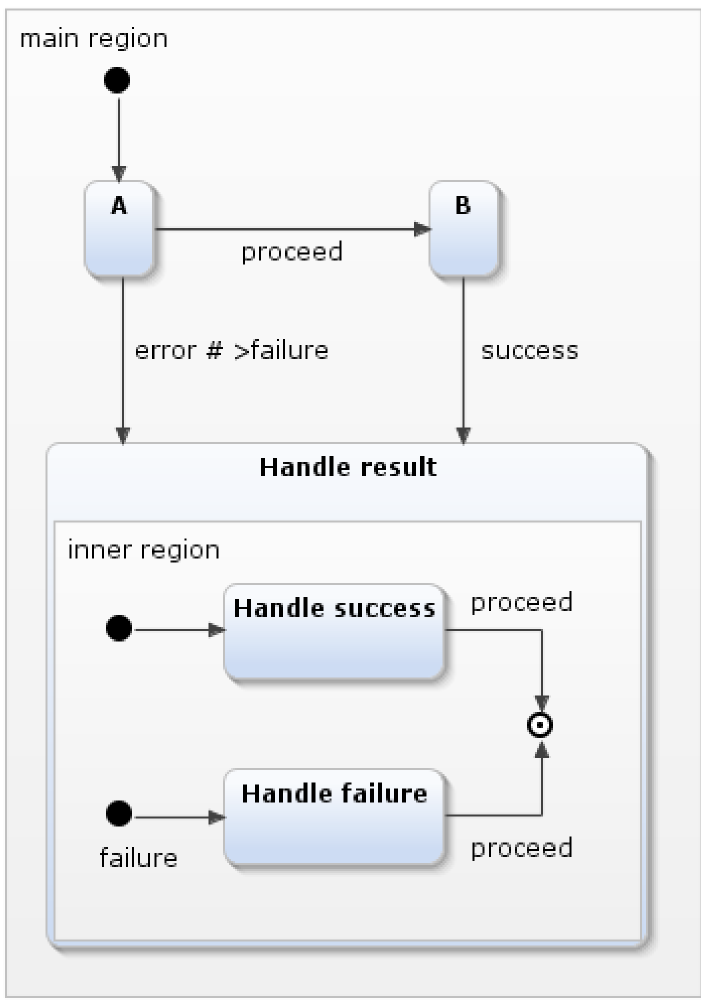


Figure 3.14: Named exit state - notation: #<exitStateName» of "YAKINDU Statechart Tools" (Source: [23])

3.4.1 Definition Final States

A final state is defined as a state in a compound state which designates that the compound state has completed, i.e., will not process any further events and denotes the end of the execution flow of a statechart or region [17][24][23]. Final states can be left again in other states at any time by direct leaving transitions. Event propagation or "bubbling" is not possible with final states, since the final state compound is left via transition with "in <componentName>.final/finalAll"-condition and not via a broadcasted event. Final states are visualized by an ordinary state with a second dashed surrounding. This way, it can be achieved that in implementations, final states can be implemented like normal states and no additional code is needed.

The label of the transition leaving the final state component must also be defined. The exiting transition leaves the super-state when the "in <componentName>.final/finalAll"-condition is met. So, if one respectively all final states within the super-state are active, the super-state will be exited via that transition, which can also be extended by further conditions or actions. The advantage of the condition chosen here over the "in <finalStateName>"-condition already present in the classic Harel-features is that with the latter, the name of the final states must always be specified. This is not necessary with the first condition. By using the in "<componentName>.final/finalAll"-condition, a decoupling takes place.

3.4.2 Completion Event Handling with Final States

In the following, the application of the definition will be shown by means of some representative example situations. Figure 3.15 shows a feeder states compound, which is explained in more detail in the "New Statechart Features and Pattern" section 3.5. This component is left to an external state via final state "StateN" and "in FeederStatesCompound.final"-condition. In this example, if the event "EventN" occurs in the state "StateN", the final state is exited again.

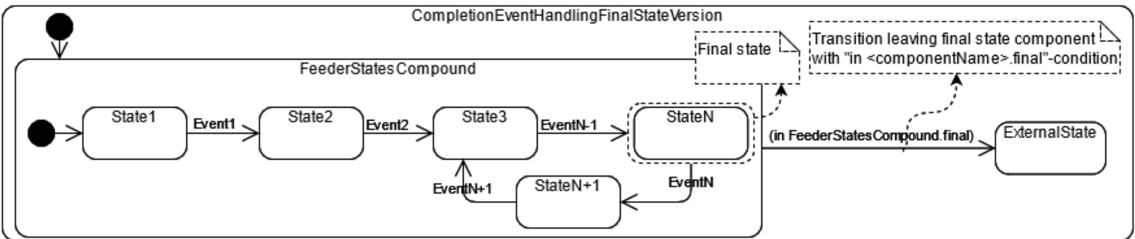


Figure 3.15: Final state with "in <componentName>.final"-condition

In figure 3.16 and figure 3.17, several final states occur in a parallel component. The difference is that in figure 3.16, due to the "in FeederStatesCompound.final"-condtion, the "FeederStatesCompound" is exited as soon as the first final state is reached. In figure 3.17, on the other hand, all parallel final states must be active because of the "in FeederStatesCompound.finalAll"-condition. Until this is the case, the system waits in already active final states until all of them are active. Only then, the transition to "ExternalState" is traversed.

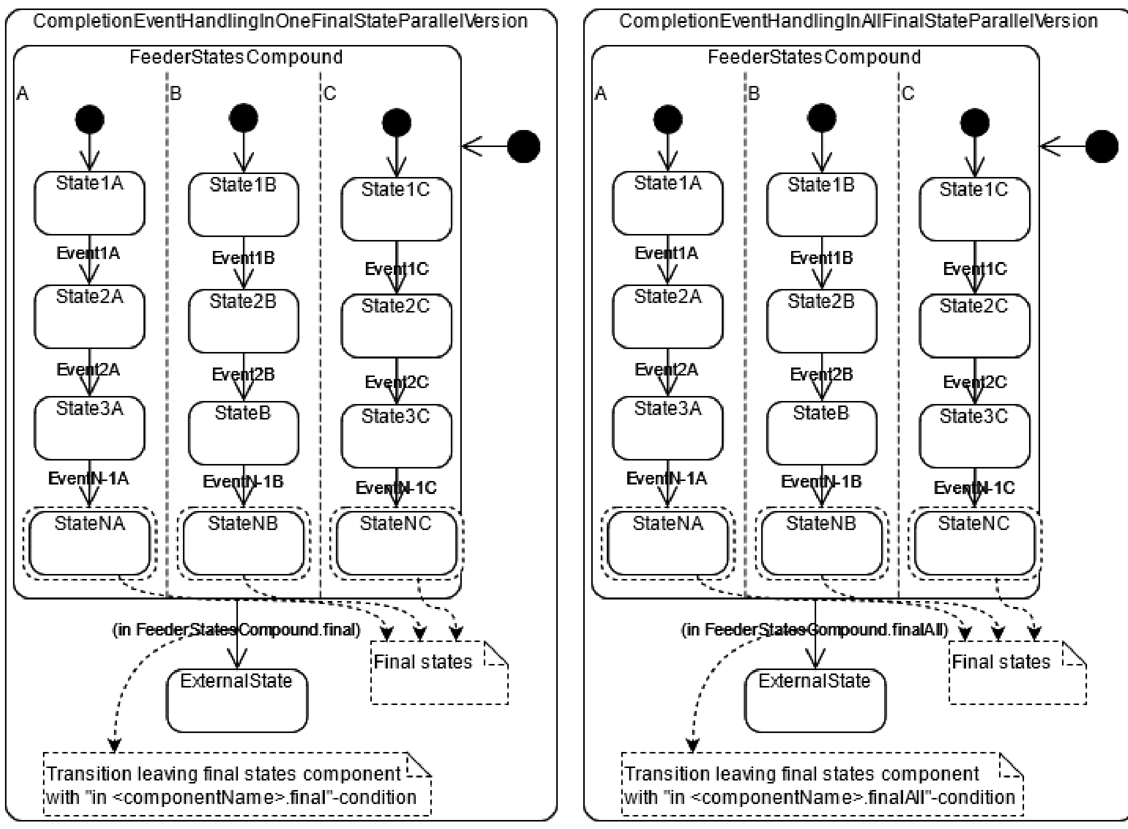


Figure 3.16: Multiple final states with "in <componentName>.final"-condition

Figure 3.17: Multiple final states with "in <componentName>.finalAll"-condition

This leaves the case where final states are used in a hierarchical structure. This is quite simple since transitions with "in <componentName>.final/finalAll"-condition must start in the active state to be traversed and can end in all other states of the system. In contrast to exits, the problem does not emerge that the broadcasted event is caught on several levels by several receivers due to event propagation, since we are not broadcasting an event but using the transitions with "in <componentName>.final/finalAll"-condition to leave the completed component. When more than one transition can be taken, the transition starting in the least-nested state – the state on the outermost level – is traversed, because leaving an outer level compared to leaving an inner level has a greater effect, since the inner levels are also left when leaving the outer level of a nested state. So, the more important event will be executed. Some statechart editors provide both the "parent-first execution"- and the "child-first execution"-option. In "YAKINDU Statechart Tools", for example, the code lines "@ParentFirstExecution" and "@ChildFirstExecution" can be used to specify which execution order is to be followed [25]. If all matching transitions start in the same state, then a hidden order is created. Depending on the implementation,

this can yield different results. In this paper, the first transition found is traversed as it is done in SCXML [26].

In figure 3.18 are several matching exit-transitions. The transition ending in "ExternalState1" is taken, since it is the transition starting in the least-nested state on the outermost level.

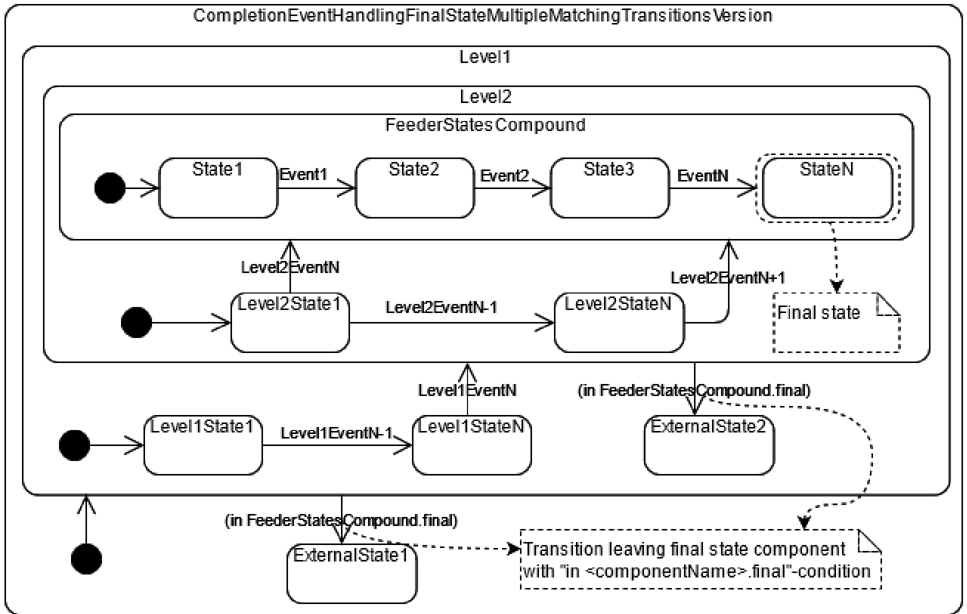


Figure 3.18: Final state with multiple "in <componentName>.final"-conditions

Figure 3.19 shows a scenario in which two final states and two leaving transitions exist. Nevertheless, in this example there is always only one matching exit-transition for each final state. "StateN" is left into "ExternalState2" due to the condition "in FeederStatesCompound.finish" and "Level2StateN" is left into the "ExternalState1" because of the "in Level2.finish"-condition.

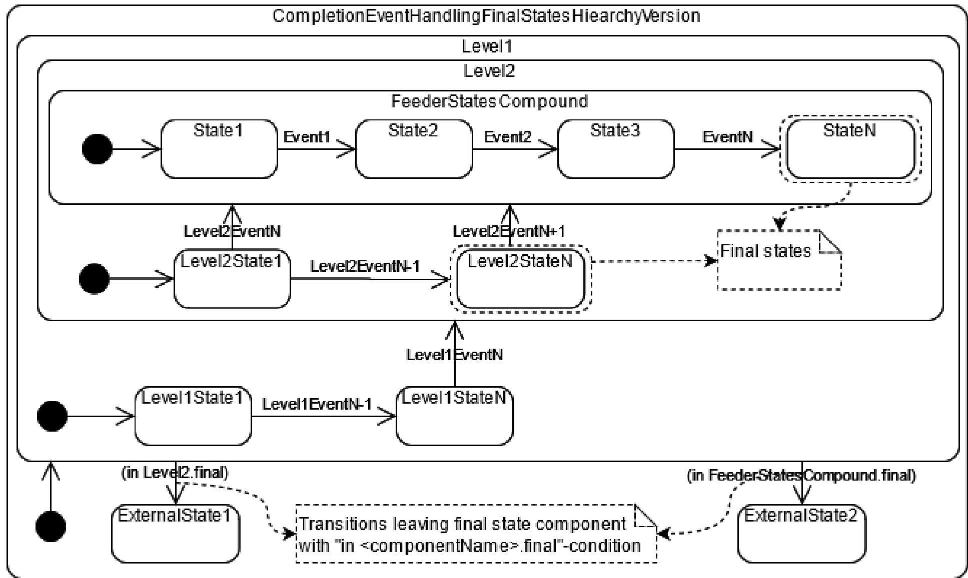


Figure 3.19: Multiple final states with multiple "in <componentName>.final"-conditions

3.4.3 Definition Exits, Exit Receivers and Implicit Exit States

Exits are pseudo-states that are used to leave a previously entered super-state, while also exiting possible existing substates. Since these are non-concrete states, it is not possible to dwell in them. Furthermore, they can only be entered when a counterpart exists, otherwise the system throws an exception. Additionally, exits are not allowed to have direct leaving transitions. When an exit is entered, it broadcasts an EXIT event. This procedure allows limited event propagation. Why this is only limited event propagation will be explained in the section 3.4.4 "Completion Event Handling with Exits". The exit is visualized as a black square inside another square and is located within the component that will be left.

Exit receivers – also called counterparts – are pseudo-states that receive EXIT events from exits and control the transition to the next component of the system. Therefore, they have an outgoing transition, which is explained in more detail at the end of this definition chapter. At this point it should only be mentioned that the transition can be labeled with an EXIT trigger event and an EXITALL trigger event. These trigger events are also called flags in this section. If an exit receiver catches an EXIT event broadcasted from an exit and the transition leaving the exit receiver has an EXIT flag, the super-state is left to the external state targeted by the transition leaving the exit receiver. But if an exit receiver catches a single EXIT event broadcasted from an exit, but the transition leaving the exit receiver is labeled with an EXITALL flag instead of an EXIT flag, the exit receiver sends back that not all parallel regions are ready to exit the whole component. Thereupon the exit in the finished orthogonal component is left to the respective implicit exit state, which is also defined in this chapter. The implicit exit state is then active until the exit receiver has gotten an EXIT event from all exits of all parallel components. If this has happened, the gates of the transitions from the implicit exit states to the exits are opened again and all components enter their exits. Via the transition of the exit receiver, the entire super-state is then exited to an external state. Furthermore, one exit receiver can receive EXIT events from multiple exits of the component. Therefore, the exit-exit receiver interaction presented in this paper can be understood as a transition hub. Also, all connections to the outside run via the exit receiver. By broadcasting an EXIT event, even the physical transitions within the component between exits and exit receiver can be omitted. In addition, the internal components do not know the external ones. Therefore, an interface is needed, which is provided by the exit receiver. The visualization of exit receivers is the same as for exits, except that now a transition leaves the state and does not enter it. In addition, exit receivers are positioned on the line of the component to be exited, as this better expresses the fact that the entire component with all substates is being exited. Exits and exit receivers can also melt. This happens when the receiver is positioned at the same level as the exit. In this case the EXIT event broadcasted by the exit is immediately caught again by the merged exit receiver. Visually, both states melt as well. This means that only the exit receiver is still on the line, but it has both, an outgoing and an incoming transition. This can be seen in figure 3.20, figure 3.21 and figure 3.22 in the next section.

For completeness, implicit exit states and labels of transitions leaving the exit receiver must also be defined. Implicit exit states are the predecessor states of exits. Since these states are concrete states, it is possible to dwell in them. What makes them special is that these states have a transition to the exit. However, this transition can only be travelled if there is a valid counterpart of the exit. This means that the system waits in the implicit exit state until the component can be exited through the exit ant the exit receiver. This is, where the name implicit exit state comes from. Implicit exit states can also have transi-

tions to other states through which they can always be left, like final states. As mentioned in the definition of exit receivers, when an EXITALL flag is set in the label of the transition leaving the exit receiver, the system waits in the implicit exit states until all parallel components have reached their exit states. Only when the implicit exit states are active in all parallel components, the transition into the exit becomes active, to then leave the entire component via the exit receiver and the transition out of them. Since the implicit exit states are normal states of the statechart, which have a transition to the exit as the only extraordinary feature, they are visualized as normal states.

The last thing to define are the labels of the outgoing transitions of the exit receivers. As triggering events there can be the flags EXIT and EXITALL. The EXIT flag means that if the exit receiver receives an EXIT event from an exit, the transition is immediately traversed. The EXITALL flag, on the other hand, activates the transition only if in all parallel components the implicit final states are active, so if the exit receiver has caught a broadcast EXIT event from all parallel components. In addition, the transition can also have actions, that are performed while traversing, and conditions, that must be met, when exiting the component through this exit receiver. If not all conditions are met, the broadcasted EXIT event is not handled by the receiver but is passed on to the next receiver. If an EXIT event is never caught, it will continue to propagate until it causes the entire program to terminate. This could be prevented by the implementation of a warning system.

3.4.4 Completion Event Handling with Exits

As it was done with the final states, in the following, the application of the definitions for exits, exit receivers and implicit exit states will be shown by means of some representative example situations.

In figure 3.20 the exit is also the exit receiver. This is an example situation where they melt because they are on the same level. As flag for the transition leaving the exit receiver, the EXIT flag is chosen. Furthermore, in this example, the implicit exit state "StateN" can be left when event "EventN" occurs.

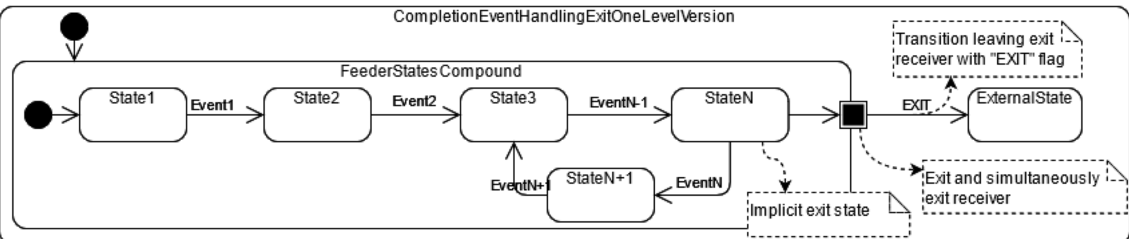


Figure 3.20: Melted exit and exit receiver on single level with an EXIT flag and a leaving transition in implicit exit state

Figure 3.21 and figure 3.22 show situations with one orthogonal component and three exits, all of which melted with one exit receiver. The difference between the two figures is that in figure 3.21, an EXIT flag is used, while in figure 3.22 an EXITALL flag is applied. This means that in figure 3.21 the component "FeederStatesCompound" is exited to the "ExternalState" as soon as the first orthogonal region reaches the exit. In figure 3.22, however, the orthogonal components must wait in the implicit exit states until they are active in all parallel regions. Only then is the entire component exited to the "ExteralState".

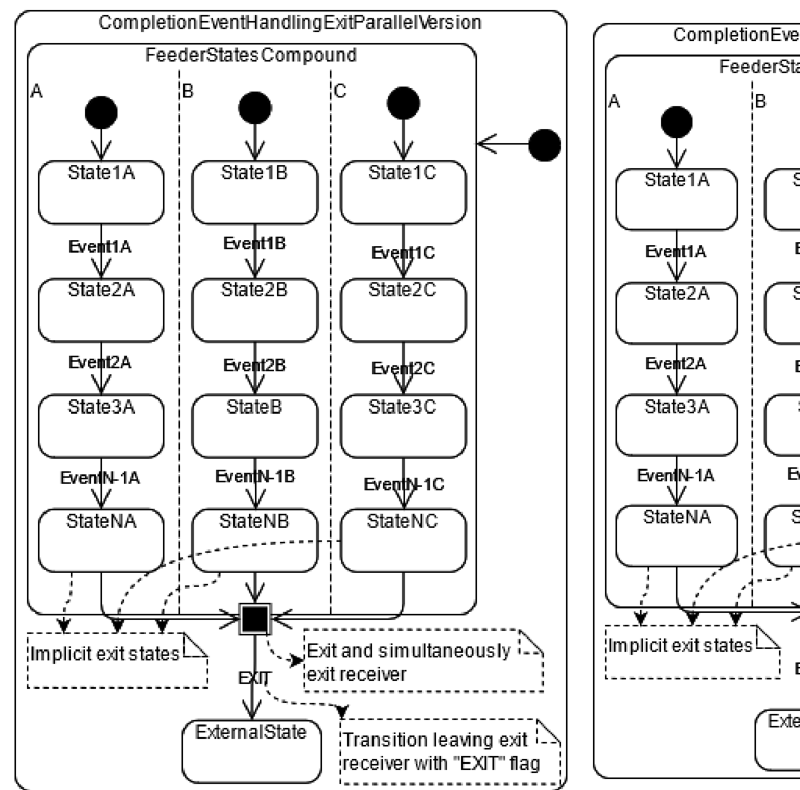


Figure 3.21: Orthogonal component with multiple exits melted with one receiver with EXIT flag

Figure 3.22: Orthogonal component with multiple exits melted with one receiver with EXITALL flag

Next, situations where exit and exit receivers appear in hierarchies must also be considered. The exit broadcasts an EXIT event, which is caught by the exit receiver. Due to hierarchy, there could be more than one exit receiver. Now the problem arises, which exit receiver manages the broadcasted EXIT event. To be able to solve this problem, we first must answer what broadcasting means in the statechart context. Yacoub and Ammar described in "A Pattern Language of Statecharts" broadcasting in the context of orthogonal components. In this paper, Broadcasting is used to enable events and actions to interact in different parallel components [27]. For this purpose, broadcasting is implemented in the sense that a kind of virtual super-state contains all other components and can thus access all of them, enabling the system to broadcast something [27]. This approach can also be used when broadcasting the EXIT event. The EXIT event propagates through the virtual super-state until it finds an exit receiver that handles the broadcasted event. At this point, it will stop propagating to outer levels like it is done with exception handling in "Java" or other programming languages. Since the event no longer bubbles after it has handled, we speak of limited event propagation.

In the following figures, the use of exits and exit receivers will be illustrated by selected, hierarchical example scenarios. Figure 3.23 shows a situation where the exit and the exit receiver are not melted, since they are located on different levels. In this example the exit in "FeederStatesCompound" broadcasts an EXIT event that is caught by the exit receiver on level "Level1". After receiving the broadcasted event, the whole component is left to "ExternalState".

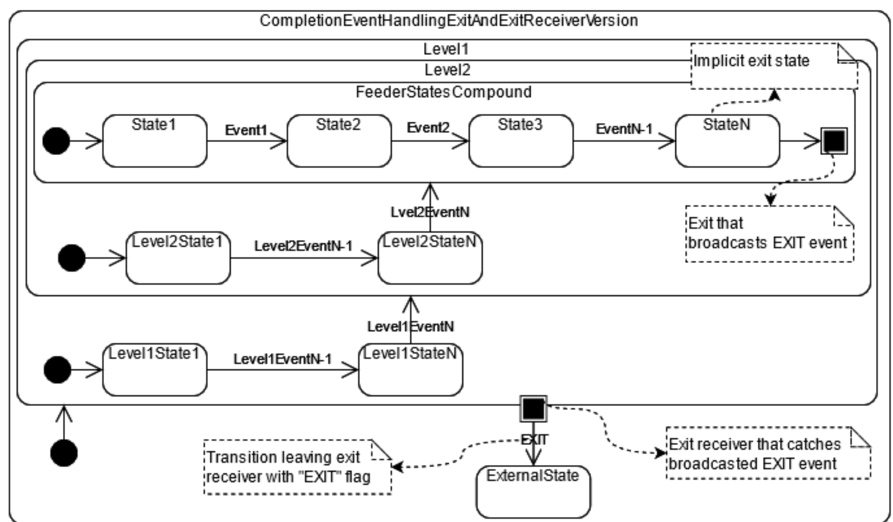


Figure 3.23: Exit and exit receiver

Now the scenario of figure 3.23 is extended by another exit in figure 3.24. The broadcasted EXIT events from both exits are caught by the same exit receiver. Therefore, no matter which exit is reached, the entire nested component is always exited to the state "External-State" via the same exit receiver and the same transition. Furthermore, the generic state "Level2StateN" is again an implicit exit state, which can be left to another state, or in this case to another super-state.

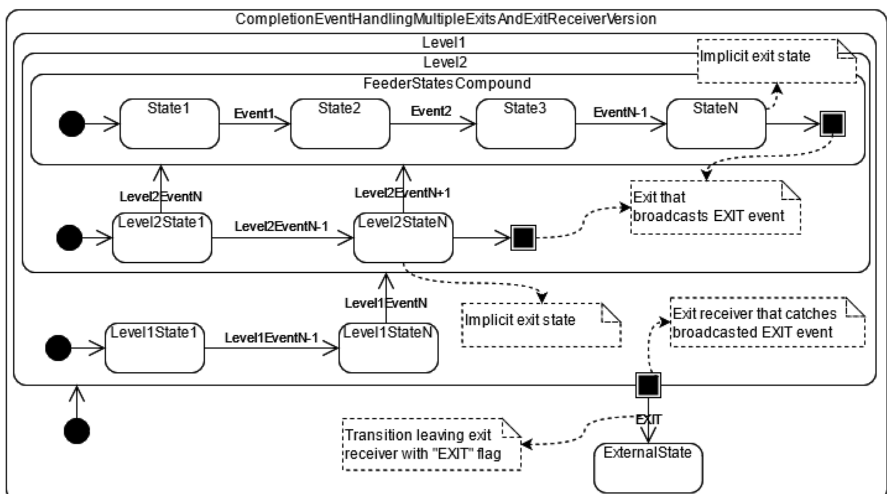


Figure 3.24: Multiple exits and one exit receiver

Figure 3.25 shows the exact opposite case. Now there is only one exit and two exit receivers on different levels. In addition, in this scenario the transitions leaving the exit receiver are guarded by a condition. After the EXIT event was broadcasted, it is first

received by the exit receiver in "Level2". When the value of the variable "VarX" is zero, the state "ExternalState2" is entered, and the broadcasting of the EXIT event is stopped. But if the Value of "VarX" is not zero, the EXIT event is passed on. Next, the exit receiver in "Level1" catches the EXIT event, tests if "VarX" is equal to one and if this is fulfilled, it leaves the component to "ExternalState1". However, if the value of "VarX" is not one or zero, the broadcasted EXIT event is never caught and is passed on until it is handled on a higher level or eventually causes the entire program to terminate, when it is never caught.

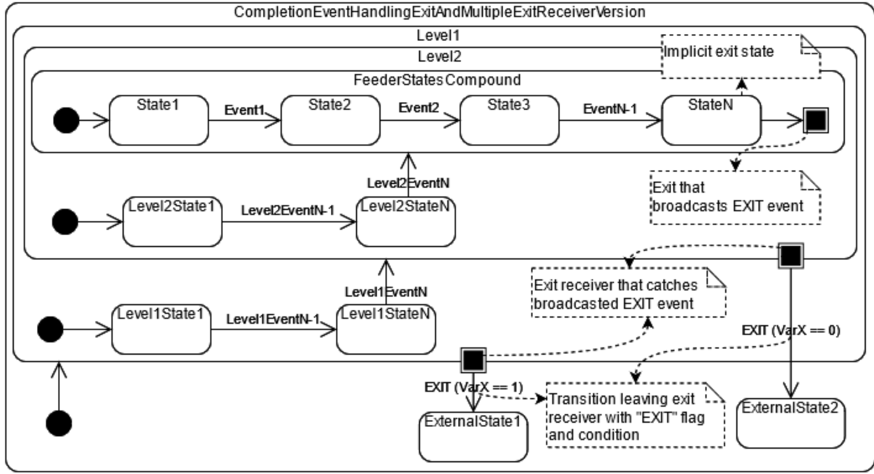


Figure 3.25: One exit and multiple exit receiver

The last scenario presented in this section to explain exit and exit receiver interaction combines a melted exit and exit receiver with a normal exit and exit receiver in figure 3.26. When the exit in "FeederStatesCompound" is entered, it broadcasts an EXIT event which is immediately caught again by the fused exit receiver. Afterwards, the exit transition is travelled and the "ExternalState2" gets active. When the exit in "Level2" is entered, an EXIT event is broadcasted. This event is caught by the exit receiver in "Level1" and the "ExteralState1" gets active. In this case the melted exit receiver does not receive the EXIT event since the broadcasted event only bubbles up and not down.

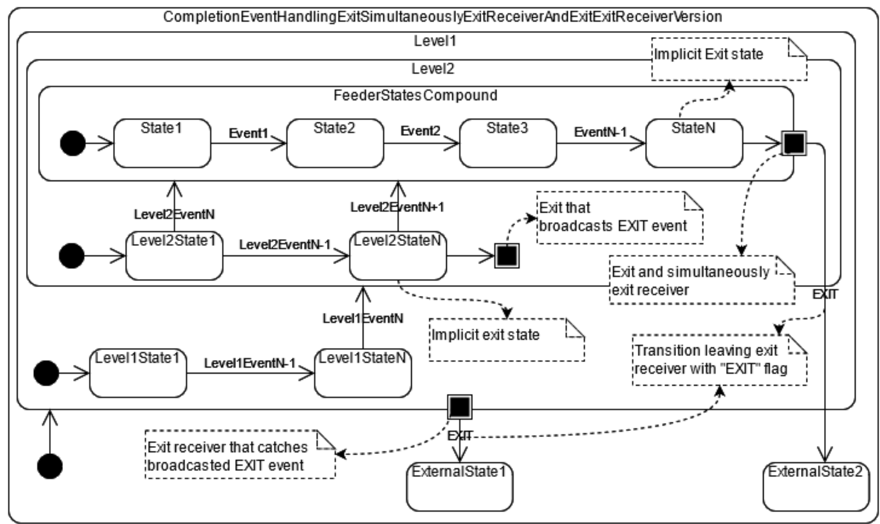


Figure 3.26: Melted exit and exit receiver and not melted exit and exit receiver

In the first approach, it was tried to implement unlimited event propagation. What this means and why it was finally decided not to do so will be explained in the following. Since event propagation is not a big issue with conventional Harel statecharts, let us see, how it is handled on the web with "JavaScript". Event propagation is defined in the web context as the following process: A single event, fired on a particular node, propagates through the tree hierarchy, and indirectly triggers a series of other event-handlers attached to other nodes [28]. In this context, the term tree hierarchy represents the entire structure of a website since they consist of nested hierarchical elements. Furthermore, the event handler can be triggered in two directions. If bubbling is enabled, the event first triggers the handler of the deepest child element, on which the event was fired and then it bubbles up and triggers the parents' handlers [28]. That means that here we proceed from the most-nested element to the least-nested one. If capturing – also called trickling – is enabled the opposite direction is done. The event is first captured by the parent element and then passed to the event handlers of the children [28]. So, the deepest child element is the last one capturing the event [28]. Therefore, Event handling has the effect that, for example, a single click on an element causes several boxes to expand due to bubbling. Thus, the user's click is processed by several hierarchically arranged handlers. Applied to statecharts, this could be used to allow broadcasted EXIT events to bubble up. In this case, all exit receivers would catch the EXIT event and travel their EXIT or EXITALL transition. So, all levels with exit receivers are exited until the least-nested transition gets triggered and the respective level is left. Consequently, not only the closest level having an exit receiver is left, but also the levels up to the level with the least-nested exit receiver are exited. In the end, this results in the fact, that all levels up to the level containing the least-nested exit receiver are left to the state targeted by the transition leaving this exit receiver, regardless of whether they were left already before due to an own exit receiver to another external state. This is the reason why unlimited event propagation is not useful in the statechart context and why only limited event propagation is used for the propagation of EXIT events. In the web context it makes sense that bubbling events are handled multiple times even at different levels. For example, it is useful that an "onClick"-event expands several hierarchically arranged areas. In the statechart case, however, it does not make sense to first exit the component on an inner level and move to another state, and then to exit the entire component, including the state just entered, when the outer receiver catches the broadcast event. In addition to the unnecessary extra work, this can also lead to side effects in the only briefly entered state. For example, in this state an entry action could be triggered. But possible exit actions are ignored since the state does not really become active and therefore is not regularly exited. Figure 3.27 shows this problematic. When the EXIT event was shot by the exit in "FeederStatesCompound", it is received by both exit receivers due to bubbling. At first the exit receiver on the level "Level2" catches the EXIT event and processes it, i.e., the exit receiver is left to the state "ExternalState2". While entering this state the entry action "ActionA" is performed. But when the second exit receiver on the outer level "Level1" catches the broadcasted EXIT event, the "Level1" and all inner levels are left immediately to the "ExternalState1". That includes that the "ExternalState2" is not active anymore. However, the exit action "ActionB" is not performed since the state is not left regularly. So in the end, only "ExternalState1" is active and in state "ExternalState2" exclusively the entry action was performed, but not the exit action. This could lead to an unwanted behavior or also to bugs. Therefore, only limited event propagation is used for EXIT evens. Nevertheless, some application examples for unlimited event propagation in statecharts would be conceivable. The "Outlook" section 5 provides some ideas for this.

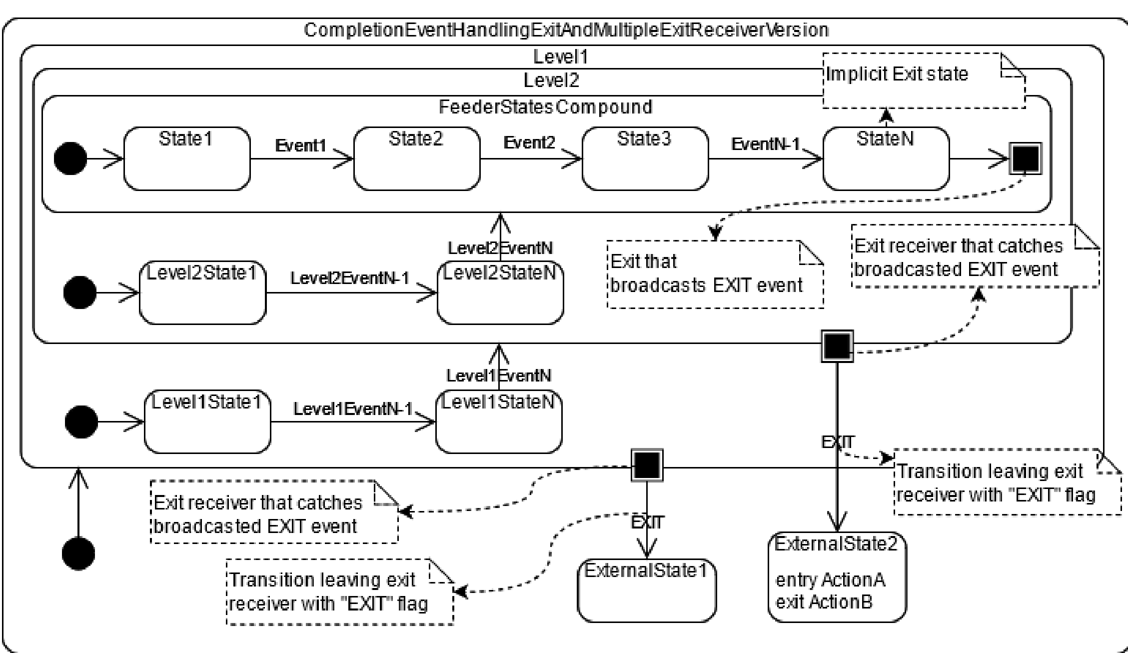


Figure 3.27: Multiple exit receivers - example for unlimited event propagation

3.4.5 Differences between Final State and Exit

After defining final states and exits and explaining their application, the question arises in which way they differ. Table 3.1 compares final states and exits by considering the most important aspects.

Final state	Aspect	Exit
Designates in a compound state the completion and denotes the end of the execution flow of a state machine or region.	**Intended use (leaving component intrinsically motivated)**	Pseudo-state to leave a previously entered super-state, while also exiting possible existing substates.
Concrete state	**State type**	Non-concrete state – pseudo-state: in exit cannot be dwelled
Can always be entered.	**Entering condition**	Can only be entered when valid counterpart exists.
Can be left in other states at any time.	**Leaving transitions**	No direct leaving transitions, implicit exit states (predecessor state of exit) can be left in other states at any time.
Transition with “in <componentName>.final/finalAll”-condition	**In general leaving Component in which exit or final state was entered**	EXIT event is broadcasted – limited event propagation
“in <componentName>.final”-condition	**Leaving parallel components when one is in its final state/exit**	EXIT flag as event of transition leaving the exit receiver
“in <componentName>.finalAll”-condition	**Leaving parallel components when all are in their final state/exit**	EXITALL flag as event of transition leaving the exit receiver
Additional condition linked to “in <componentName>.final/finalAll”-condition by "&&".	**Additional condition for leaving**	Condition is included in the system as a condition of the leaving transition of the exit receiver.
Stuck in final state	**Additional condition never fulfilled/broadcasted event is never caught because of additional condition**	Entire program terminates
Multiple possible transitions that can be travelled: - Traversing Transitions starting in the least-nested state (most important leaving event) - Multiple matching transition starting on the same level: A secret order is created (transition found first)	**Hierarchy**	Limited event propagation: Broadcasted EXIT event, that is propagated upwards to less-nested levels until it is managed by the next matching exit receiver, then event propagation stops.
Like an ordinary state with a flag: It can be seen as a state in which the component is considered as completed	**Role**	Counterpart of default-entrance: There exists a process/transition, which completes the component

Table 3.1: Comparison of final states and exits

3.4.6 Reasons for Leaving-Option of Final States and Implicit Exit States

Finally, only one aspect remains to be clarified before final states and exits are considered fully defined. Final states and implicit exit states are defined in such a way that it is permissible to leave them to another state. This allows the criticism that final states and implicit exit states are not final since they can be left. The fact that they can be left is merely the default behavior selected by the author. The default behavior can be changed simply by redefinition so that these states cannot be exited, i.e., outgoing transitions are not evaluated. This would lead to dead-code, since now transitions, which were valid transitions before, can no longer be traveled. However, this would not limit the general functionality of a statechart. Nevertheless, the author has decided that final states and implicit exit states can be left. This is because external influences can exist, which affect a component in such a way that it is necessary to leave the final state or the implicit exit state. This can be the case because the system is influenced by the external influences in such a way that the current situation of the system no longer corresponds to the conditions and circumstances of the final state or the implicit exit state. As an example, a state diagram representing the landing process of a helicopter onto a ship can be provided at this point. When the helicopter is just above the landing zone, the statechart is in the final state. However, if the position of the ship deviates or the helicopter is driven off by a gust of wind (the external influences), the helicopter is no longer in a state that allows landing. Consequently, the final state is left again in the statechart until the helicopter is hovering directly above the landing zone again.

3.5 New Statechart Features and Pattern

As shown in the chapter "Mapping Storytelling Elements onto Statechart Features", with Harel's statechart features almost any situation that might arise in storytelling can be realized. Nevertheless, some new features and patterns are conceivable to facilitate storytelling in statecharts.

The first new pattern, the feeder states compound, has already been used in the figures of the last section. The idea behind this new pattern is, like a feeder road of a highway, to make sure that a path is traveled that leads only in one direction and can be left only at one point. Formally the feeder states compound consists of a sequence of states that terminates in an exit and has no outgoing transitions. A generic example can be seen in figure 3.28.

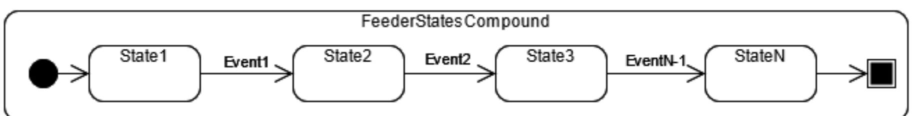


Figure 3.28: Generic feeder states compound example

Ahead of the mapping chapter, it can be said that feeder states compounds can represent a sequence of events of the plot that ends in a fixed state. This fixed state of the plot is always the same state. A concrete example from the storytelling context for which the feeder states compound can be used, is an introduction sequence before a quest. In this case, the only end point lies in the beginning of the quest. So, apart from this point, there is no other possibility where the introduction can be left. Figure 3.29 shows an example

scenario where the introduction of the "Blacksmith"-quest is realized by a feeder states compound. In it, the player meets the blacksmith, who introduces himself to the player through a cutscene, which is realized as an activity. Once this activity is completed, the introduction is exited to the end point. In this state, the blacksmith asks the player for help. He offers him a quest in a cutscene realized by an activity that can be aborted at any time by leaving the forge. After the payment for the forged sword has been delivered to the blacksmith or the player has incurred the displeasure of the blacksmith or left the forge, the quest ends in one of three final states that describe the relationship between the player and the blacksmith.

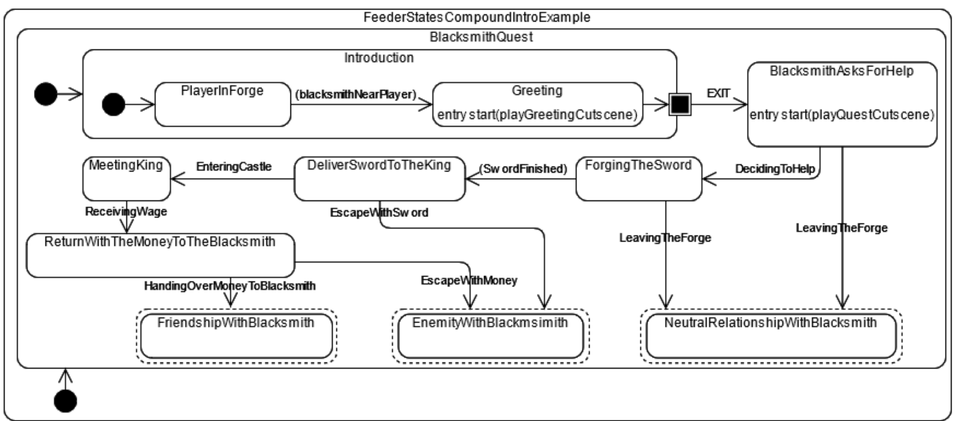


Figure 3.29: "Blacksmith"-quest with the introduction being realized via feeder states compound

Feeder states compounds are used in the next new feature, the extended history entrance. The basic problem to solve is that with the Harel features, it is very cumbersome to create a possibility to offer an individual introduction for each state after a history entrance. This can be seen in figure 3.30. However, this feature is often needed in storytelling to preserve the immersion. Why should a blacksmith give the player a complete introduction including introducing himself, if the player was already in the middle of the quest and only interrupted it? In this case, only a short question about where the player was and a brief repetition of where the player stopped when he interrupted the quest would be more appropriate. Furthermore, in the short individual introduction before re-entering the last entered state, reference could be made to current events and the ongoing situation. This would make the game experience even more immersive.

Technically, the extended history entrance is a history state with feeder states compounds as substates. Thus, the extended history entrance is not a pseudo-state in contrast to the conventional history state. Nevertheless, the basic function is the same: The last active sibling state is revisited after entering the extended history state. Before that, however, a feeder states compound is traversed within the extended history state. Which feeder states compound of several feeder states compounds is selected, depends on the H-entrance decision, so which state was last active when the component was left. This is implemented in such a way that in the extended history state, there is a condition entrance whose various conditions consist of the states in which the system may have been before the interrupt. Thus, when the extended history entrance is entered, the history state is evaluated, resulting in the fulfillment of a condition of the condition entrance. In this context, the condition "first time" means that the extended history state is visited for the first time, i.e., when the component, in which the history entrance is located, has not

been active before. In addition, within the feeder states compound, events and states of a parallel compound can also be referred to by the "in <stateName>"-condition. After traversing the feeder states compound, as with the conventional H-state, the last state visited in the component is entered.

The extended history entrance is represented by an "EH", which is circled and labeled with a name. The substates within the extended history state are represented due to refinement and unclustering outside the statechart.

As mentioned at the beginning of the explanation of the new feature, extended history entrances are ideal for offering varying introductions when starting and resuming quests based on the progress already made before the interruption or the current situation of the storyworld. To show this in practice, the following statecharts (figure 3.30, figure 3.31) take up the situation of figure 3.29, in which the player carries out a quest for a blacksmith. But now this scenario is extended by an interrupt that allows the player to pause the quest. When the player resumes the mission, i.e., leaves the interrupt, an individual introduction is performed based on different cutscenes. Which introduction is chosen depends on the state the player was in when he left the quest for the interrupt. For example, if the player left the quest in the "ReturnWithTheMoneyToTheBlacksmith"-state, when he resumes the quest, he will be asked by the blacksmith's wife when he will finally deliver the money. But if the player was only in the state "ForgingTheSword", the blacksmith will ask, where the player was and will briefly explain the current blacksmith step in which the player was before he left.

If we compare figure 3.30 and figure 3.31, we can observe that the scenario in figure 3.30 using only the Harel features, is significantly less structured than the same situation in figure 3.31 with the extended history feature.

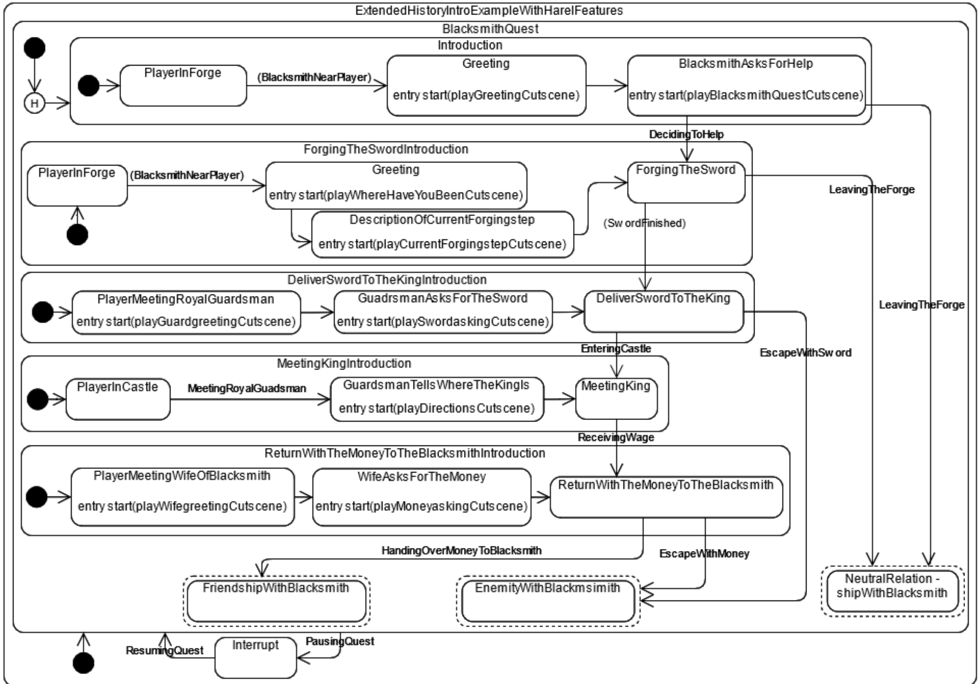


Figure 3.30: Extended history situation but with only Harel features

Figure 3.32 shows the same situation as in the previous figures. But this figure has only been extended by a parallel side quest, which can be referred to in the introductions in the

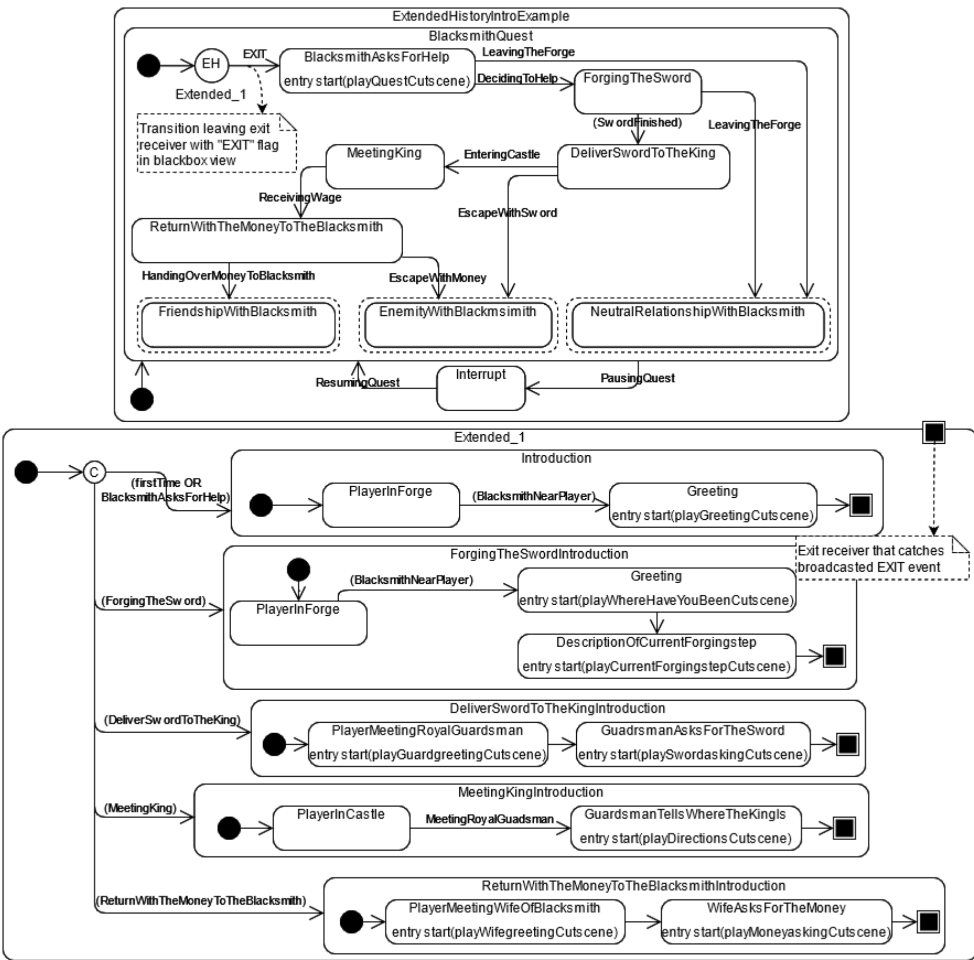


Figure 3.31: Example situation extended history entrance

extended history. For example, the player may be alerted to a bandit hideout by a guard when he re-enters the "DeliverSwordToTheKing"-state. Another parallel interaction takes place when the player re-enters the "ReturnWithTheMoneyToTheBlacksmith"-state. The blacksmith’s wife, in addition to asking for the money, also gives a warning about the mercenaries, if the player has accepted the mercenaries’ job in the parallel "Mercenary"-quest.

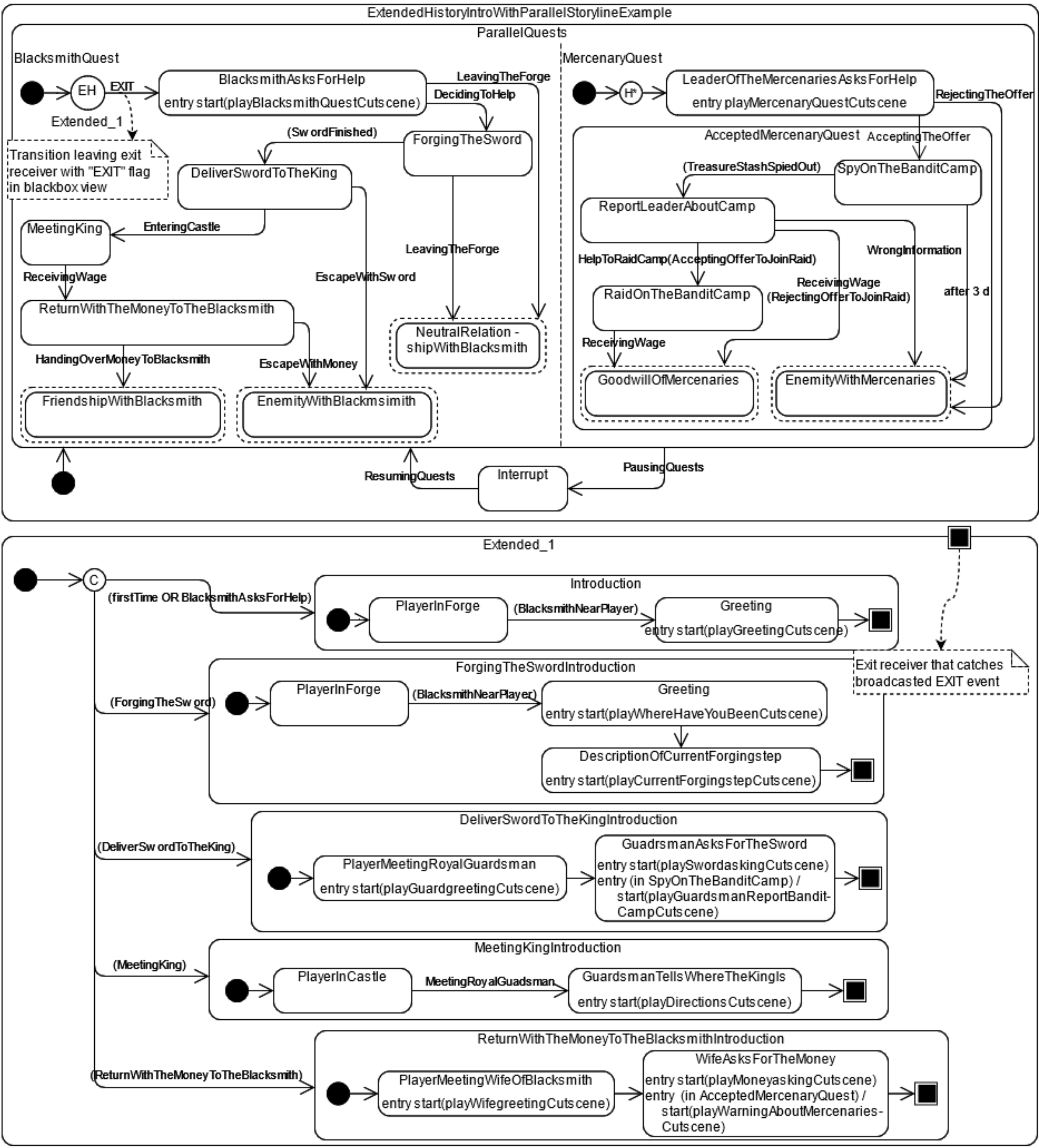


Figure 3.32: Example situation extended history entrance with two parallel quests

Despite the better structure of the extended history feature compared to the approach using only Harel features, it also has some disadvantages. In an implementation, many additional lines of code are needed to implement this feature. Furthermore, the name of states appears in several locations, which breaks the decoupling. To avoid these disadvantages, the new feature "extended colored history entrance" will be introduced. The

purpose of this feature should be the same as for extended history states: After a history entrance, individual paths should be traversed before the last active state becomes active again. Therefore, a conventional history state determines the last active state. In front of all states, however, there is a colored feeder states compound that is entered instead of the state determined by the conventional history state before. After this compound has been traversed, the last active state is entered as usual. As with the extended history entrance, only a colored and circled "ECH" is shown in the actual statechart before the states. The corresponding-colored feeder states compound can be found outside the actual statechart.

Figure 3.33 shows the same scenario as figure 3.30 and figure 3.31 but realized with the extended colored history entrance feature

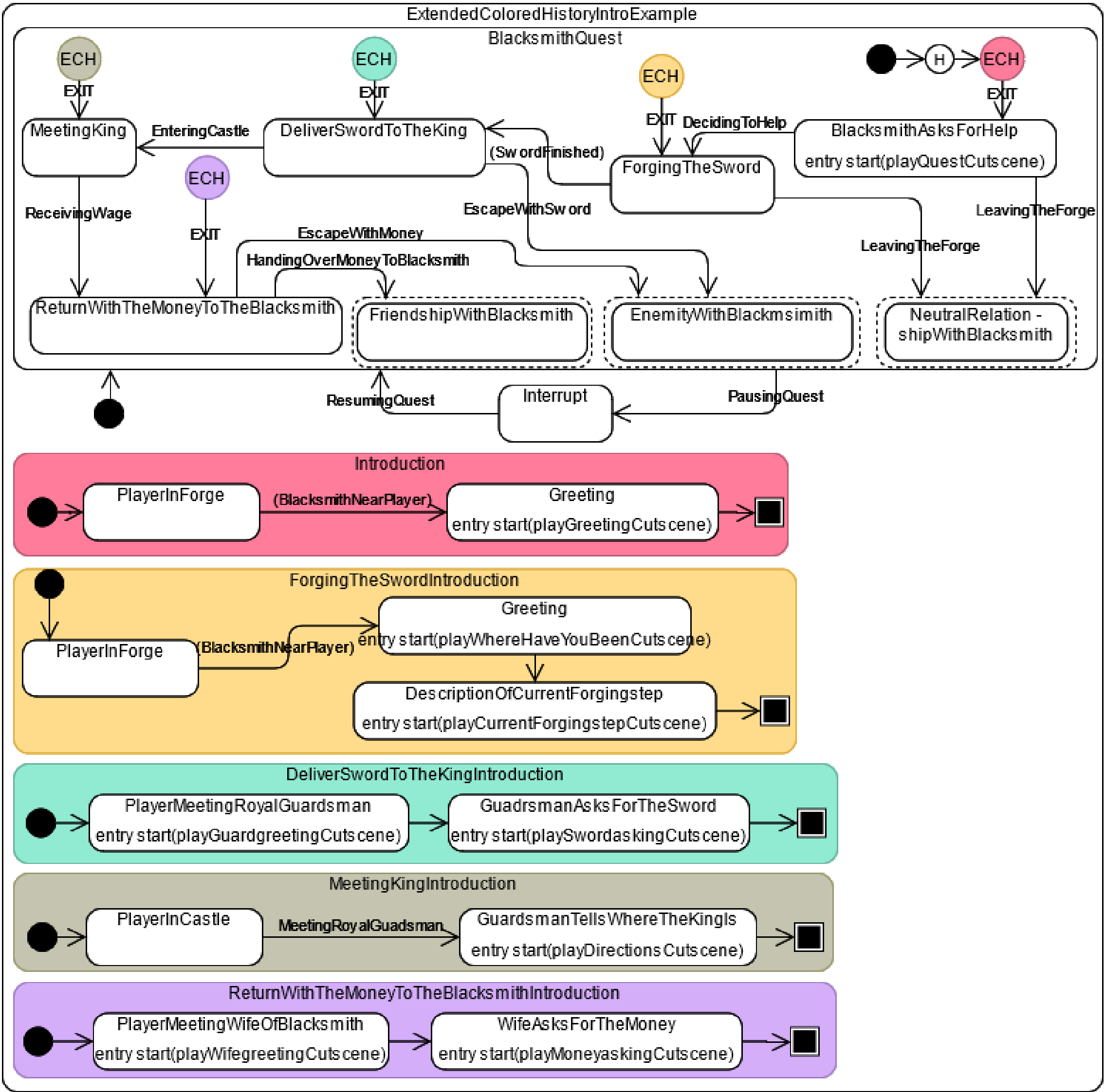


Figure 3.33: Example situation extended colored history entrance

To conclude the section on extended history features, table 3.2 compares extended history, extended colored history and the realization with only Harel features in the most important aspects. Red text shows a disadvantage and green text indicates that the respective aspect has a positive effect.

	Extended History	Extended Colored History	Harel Statechart Features
The same states before multiple states	One feeder states compound in the extended history can be used to be executed before several other states of the statechart by adding their state-names in the condition of the condition state in the extended history.	A feeder states compound can be used to be executed before several other states in the superstate by simply coloring the respective extended colored states in the same color.	To be able to guarantee that the same states are executed before multiple states, overlapping super states would be needed. → confusing
Clarity and structure	Clearly structured and well-arranged, as the inner of the extended history is separated from the visual symbol in the statechart by means of unclustering and refinement.	Clearly structured and well-arranged, as the inner of the extended history is separated from the visual symbol in the statechart by means of unclustering and refinement. In addition, the colors contribute to quick recognition of the relationships of the individual parts.	Confusing: You can hardly recognize the actual relationships and progressions of the states, since the states that only become active during a history entry take up a large part of the model.
Reaction to external changes	Within the extended history it is possible to react to changes that have taken place in the interrupt or in a parallel component (e.g., values of variables that are changed in the interrupt).	Within the extended colored history it is possible to react to changes that have taken place in the interrupt or in a parallel component (e.g., values of variables that are changed in the interrupt).	If there are too many branches and variants of the states, which are only active during a history entry, the statechart quickly takes on oversized dimensions.
Hierarchical Complexity	States within the extended history can be super-states themselves without the hierarchy getting out of hand.	States within the extended colored history can be super-states themselves without the hierarchy getting out of hand.	Hierarchy quickly gets out of hand, since two levels are already necessary to be able to visualize the situations through Harel features.
Expense	Additional feature that needs to be added to the statechart-canon.	Additional feature that needs to be added to the statechart-canon.	Modeling only with already established features.
Costs	The result of the H-entrance decision must be retrieved twice.	The result of the H-entrance decision must be retrieved only once.	The result of the H-entrance decision must be retrieved only once.
Maintain ability	State-names occur more than one time.	All state-names occur just one time.	All state-names occur just one time.

Table 3.2: Comparison of extended history, extended colored history and Harel statechart features

3.6 Mapping Storytelling Elements onto Statechart Features

As the last part of the theoretical section, storytelling elements will be mapped onto the statechart features. This means that we now determine which storytelling elements can be represented by which feature of the statechart. A table with three columns was selected as the most suitable format for this. The first column contains the statechart features, the second column stores the mapped story elements, and the last column provides examples from the games-context for the respective feature and references to figures in which the mapped feature is realized. Some of these figures are described below the table, while others have already appeared in previous sections.

Statechart Features	Storytelling Elements	Examples in the context of games
Hierarchical structure elements		
Node, atomic state	Concrete state of the plot	The Player tries to shoot the dragon – see figure 3.35
Super-state	Quest, Part of the plot	House-building quest or dragon attack – see figure 3.35
Substate	Part of a quest like a dialog or action	In the "BuildingAHouse"-quest the player is searching for a good building site – see figure 3.35
Parametrized state	Different states of the plot, which differ only in one point	Given answer options within a dialog, from which the player can choose one – see figure 3.36
Overlapping state	Part of a quest like a dialog or action, which is part of two quests	The player must visit the president's place of residence to make sure he is alive for the first quest and also to get advice from him for another quest.
Feeder states compound	A sequence of events of the plot that ends in a fixed state of the plot that is always the same	An introduction sequence before a quest – see figure 3.29
Transitions		
Arrow/Transition	Step to the next section of the plot, part of the storyline	Change in the gameworld, like when a dragon is sighted – see figure 3.35

Automatic transition/completion transition/eventless transition	Reaction to change in a state, but which is not connected with a concrete event (1) or a dynamically occurring event (losing life) (2)	(1): A cutscene represented by an activity started at entry is finished and thus the state is finished and can be exited – see figure 3.31 in "Extended_1" (2): The player is in a state where he fights a dragon. If the number of lives falls below 0, the player dies and enters a "PlayerDead" state. This transition can be modelled by an automatic transition – see figure 3.35
Conditional transition	Step to next section of the plot but only when something has happened or has been achieved	When building a house, the roof can just be started when the walls are finished. The completion of the walls can be managed by a condition – see figure 3.35
Delayed transition	Step to next section of the plot after an amount of time has passed	The player tries to kill the dragon outside the village. If this takes too long, the dragon flies into the village and devastates it – see figure 3.35
Self transition	Retry a part of the plot again or return to the same situation in the plot after an event has occurred	The player tries to kill the dragon inside the village. If this takes too long, the dragon devastates the house the player was building before the dragon attack. Afterwards the player tries to kill the dragon again – see figure 3.35
External transition	Step on a high level from a part of the plot to the next part of the plot	From one quest to another quest: The player is building a house, then a dragon is sighted, and he tries to fight the dragon – see figure 3.35

Local transition/ internal transition	Step from a state of the plot to the next state of the plot within a quest or part of the plot	After the player has found a good place to build his house and has decided on it, he start digging a hole – see figure 3.35
Transition leaving a super-state	Finishing a quest or interruption by an event that triggers another quest	A dragon is sighted while building a house – see figure 3.35
Transitions with common sources	Choice, Decision (not only through dialog but also through actions of the player or the environment)	Deciding whether handover the money to the blacksmith or to escape with the money, after the player has received the wage from the king
Transitions with common targets	Reunion of two storylines/story paths	If the player fights a dragon in the open field and dies, it leads to the same condition as if he dies fighting in the village.
Transitions with common triggering event	A global change in the storyworld effecting all storylines	The player is in the a "Blacksmith"-quest and in a "Mercenary"-quest and suddenly an enemy army attacks the kingdom. Then the player pauses in both quests, because he is drafted by the king to defend the borders.
Flow of events		
Triggering event	Change in the storyworld (also see "Event-Types" last paragraph)	A dragon is sighted – see figure 3.35
Condition	Event or Action that must have occurred (1) or variable that must have a certain value (2)	(1): When building a house, the roof can just be started when the walls are finished. The completion of the walls can be managed by a condition – see figure 3.35 (2): Checking after returning to the "BuildingAHouse"-quest if the village was attacked – see figure 3.35

XOR in super-state	Choice, Decision (not only through dialog but also through actions of the player or the environment)	When the mercenaries ask for help, the player can refuse or agree to help, which will either disgrace him or give him a quest – see figure 3.32
Clear-history (clh(<stateName>))	Event or decision in interruption or parallel storyline that makes it impossible to return to the abandoned point in the plot or in the main storyline.	The "BuildingAHouse"-quest is interrupted by a dragon attack. If the dragon devastates the entire village, the player cannot just return to the point where he left the building site but must start building from scratch. – see figure 3.35
Clear-history (clh(<stateName>*))	Event or decision in interruption or parallel storyline that makes it impossible to return to the abandoned point in the plot or in the main storyline and that has global significance.	Same example as above, but now individual construction steps are subdivided: Building a house wall as a super-state with the substates cutting down trees, hammering in pillars, sawing trees into boards, nailing boards to supporting pillars – however, the player does not return to the step he left, but starts all over again even the small substates in the super-states, i.e., cutting down trees.
Time event	Obvious or indirect timer for quests, dialogs, or actions; higher level: Timer that indicates how long the player can stay in the current state of the plot until an event happens; can also be used to lock later events until a certain time.	The player tries to kill the dragon outside the village. If this takes too long, the dragon flies into the village and devastates it – see figure 3.35
Final state	State of a storyline/quest where it is finished	Player dead or threat averted, e.g., the dragon was killed – see figure 3.35
Exit	Indicates that a quest or dialog or part of the plot is finished	End of a greeting and introduction sequence – see figure 3.31

Implicit exit state	State before a quest or dialog or part of the plot is finished	The blacksmith welcomes the player – see figure 3.31
Orthogonality		
And decomposition in superstate/orthogonality/concurrency	One single event causing two independent happenings leading to two parallel storylines, like plot and sub-plot, like actions of the main character and actions of the non-playable characters or antagonist, or two parallel quests (1), or different subsystems (2)	(1): Two parallel quests: In one the player helps the blacksmith forging a sword for the king and in the other he helps mercenaries to plan and execute a raid on a bandit camp in parallel – see figure 3.32 (2): Player is in tutorial (storyspace) and the tutorial takes place in the castle (worldspace) – see figure 4.1 (Prototype)
"in <stateName>"-condition	Dependencies and interferences between two parallel storylines (1) or between subsystems in parallel components (2)	(1): In the main storyline, the task is to get the king's sword for the blacksmith. At the same time, the miller offers to give more money for the sword in a side quest. If the player has already given the sword to the blacksmith in the main storyline, he can steal it back. If he still has the sword, he can give it to the miller instead of the blacksmith. – see figure (2): The personal room (worldspace) can just be left, when the tutorial (storyspace) is over – see figure 4.1 (Prototype)
Entrances		
Default state	Initial situation of the plot or of a quest/part of the plot	The player is in a medieval village and wants to build a house – figure 3.35

Shallow history (H), deep history (H*)	Coming back to a point in the plot; digress and return in a dialog; return after interrupt in the plot; pause the story/game	The player is building a house when a dragon attacks. After defeating the dragon, the player returns to the construction site. Now there are two options: Return to unchanged point, where storyline was interrupted – construction site unchanged (see figure 3.34) or return to changed point, where storyline was interrupted – construction site was devastated by the dragon (see figure 3.35)
First time entering History	Nothing special – just entering a situation of the plot for the first time	The player is in a medieval village and wants to build a house – figure 3.35
Extended history entrance, extended colored history entrance	Varying introductions when starting and resuming quests/dialogs based on progress already made before interrupting/leaving the mission	At the first meeting, the blacksmith introduces himself and gives detailed instructions on what to do, while when the blacksmith-quest is resumed after an interruption, he briefly asks where the player has gone and briefly tells him which task he was left with – see figure 3.31, figure 3.2
Conditional entrance (C)	Entrance to a quest under fulfilment of certain preconditions, which then specify the path	The player starts a quest with a heavy armour (resp. with silent clothes) à he is advised to attack from the front (resp. to sneak behind enemy lines) to steal the treasure
Selection entrance (S)	Entrance to a quest based on an event, which then specify the path	The president was killed or was only wounded in the previous event. When he is dead the player's task is to chase the attacker and when the president is still alive the task is to take care of his wounds.
Output generator/Broadcast communication		

Action	Instantaneous action or occurrence triggered or produced (indirectly) through the player's action which can have an influence on the entire storyworld (1); Generation of an internal event or change of an internal variable, which represents something of the game world or of the game mechanics (2)	(1): The player reaches the presidential suite, but he is dead. That action triggers a transition with the event "presidentWasKilled" in a parallel storyline. In this orthogonal storyline, the player cannot give the gun to the president anymore. Mix of (1) and (2): Change of the night-time to daytime through sleeping (example label of a transition: "sleeping/time = daytime"). Now somewhere else in the system the condition "time == daytime" becomes true and enables a transition. (2): The variable "VillageAttacked" is set during the attack of the dragon on the village to "true". After coming back to the building site, the condition "VillageAttacked == true" evaluates to true and results in an earlier state of the building process, since the building site was devastated by the dragon. – see figure 3.35
Activity	Activity in the storyworld, which is durable and can have an influence on the entire storyworld; also used for durable output via execution of a game mechanics method, like saying something (1) or playing cutscenes (2).	(1): Saying a text loudly in a dialog – see figure 3.36 (2): Playing of cutscenes: Returning after the dragon attack to the building site and seeing in a flashback-cutscene how the dragon destroyed the house – see figure 3.35
Abstract concepts		
Refinement/zooming-in	Starting a chapter of the storyline	Selecting a quest in the quest-overview menu and getting into the quest

Abstraction/zooming-out	Overview over abstract story parts, like introduction ->mission1 ->mission2 ->final mission	Getting into the quest-overview menu, which shows all possible quests and the path of these quests (Previous quests and subsequently unlocked quests)
Transition leaving a state with a bar (in Blackbox-view)	Finishing a quest and going back to the quest-overview menu	The player has killed the dragon and thus successfully completed the quest, unlocking the next level of the quest path.
Stubbed entrance arrow (in Blackbox-view)	An event leading to the fact that the player starts directly in the middle of a storyline and not at the beginning.	The player finds the key to the dungeon, which contains the sword he wants to steal. This event of finding the key leads to the "Sword-robbery"-quest, but not from the beginning where the player must get the key first, but directly to the middle of the quest where he has to break into the dungeon.
Unclustering	Showing details of quests, dialogs, actions	Feature in framework, where the developer can click onto a part of the statechart, which then is shown in big. à better overview
Probabilism	Events and actions that occur only to a certain probability	The dragon attacks the village with a probability of 20%. With every passing day the probability increases for a dragon attack by 5%.
Recursion	Quest in a quest	Steal the key to break into the dungeon and steal the sword
Temporal Logic		
Temporal logic	Specifying the settings	Every quest needs a victory state and at least one losing condition.
Relation of states		

Relation of states – exclusive	Two states in the plot that are mutually exclusive	The player helps the mercenaries to plan and execute their raid on the bandit camp and thus secure the goodwill of the mercenaries or, on the other hand, he refuses the mercenaries' offer and fall out of favor with the mercenaries. à two ways a storyline can play out – see figure 3.32
Relation of states – orthogonal	Two states in the plot that coexist.	The player spies in the "Mercenary"-quest on the bandit camp and delivers the sword to the king in the "Blacksmith"-quest à two parallel storylines/quests – see figure 3.32
Relation of states – ancestoral	One state in the plot that was before another state à the direct, influencing history	After sawing trees to boards, the player nails these boards together to form walls. – see figure 3.35

Table 3.3: Mapping of storytelling elements onto statechart features with gamestory examples

The following four statecharts show example scenarios of how storytelling can be realized through statecharts in a games-context. Furthermore, these examples are often referenced in the third column of the mapping table above. Figure 3.34 and figure 3.35 are very similar. Both present a quest where the player builds a house until suddenly a dragon is sighted. In both figures, a fight follows against the dragon outside the village. If the player does not manage to kill the dragon within five minutes, it flies into the village and causes destruction there. If the player manages to kill the dragon in the village, in figure 3.34, he returns exactly to the point of the "BuildingAHouse"-quest that he left. Nothing has been changed in the progress of this quest. This is different in figure 3.35. Here, a variable is set when the dragon flies into the village. When the player returns to his construction site after defeating the dragon in the village within ten minutes, he will find his house destroyed. Thus, he must start over from the point where he sawed boards. However, if the player takes longer than ten minutes to kill the dragon in the village, when he returns, he will find his entire construction site destroyed. The destruction is of such magnitude that it is no longer worthwhile to build another house on this site. Therefore, the player must start from scratch with the search for a suitable building site. Technically this is done via a clear-history action.

Figure 3.36 shows the first attempt of a dialog pattern between a non-playable character and the player, who can choose one out of four predefined answer and question options.

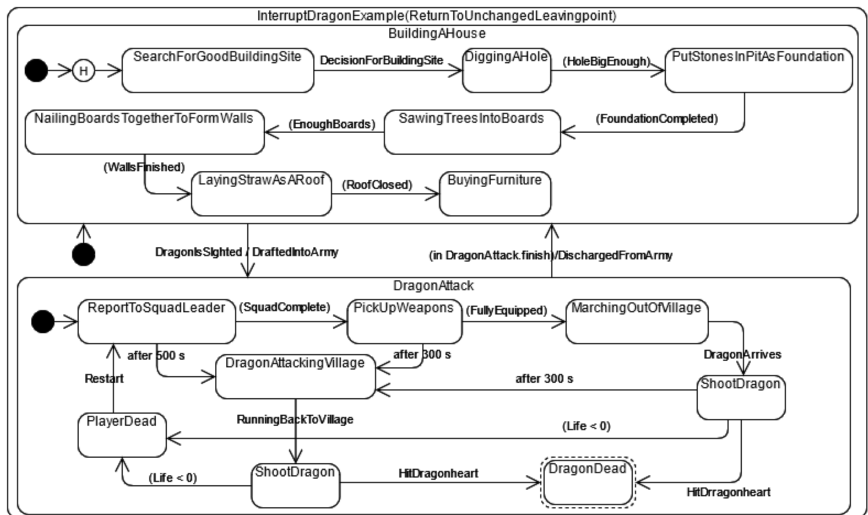


Figure 3.34: "BuildingAHouse"-quest with "DragonAttack"-interrupt - returning to unchanged building site

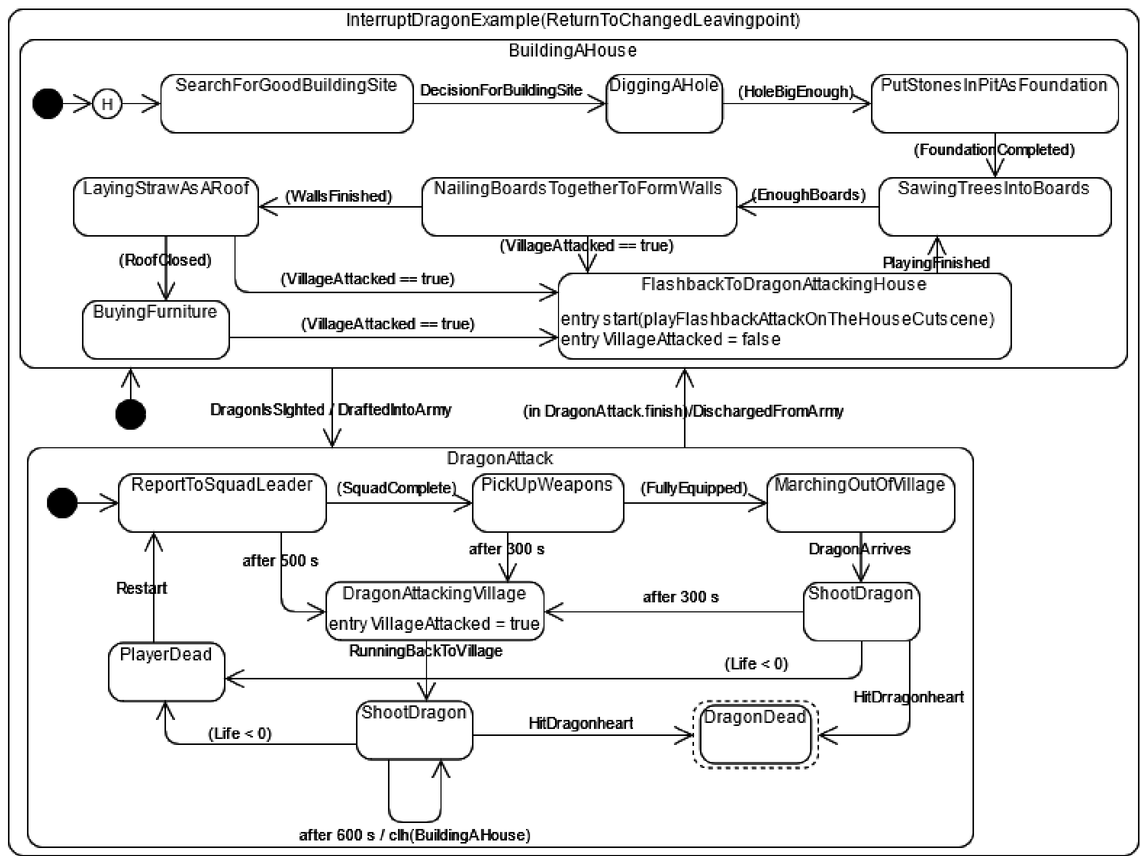


Figure 3.35: "BuildingAHouse"-quest with "DragonAttack"-interrupt - returning to changed building site

The main point in this example is that it presents a useful application of the parametrized state. The answer options the player can select only differ in the variable, which contains the text. Furthermore, in this diagram a possibility is presented to use activities for durable output by the execution of a game mechanics method, in this case speech output.

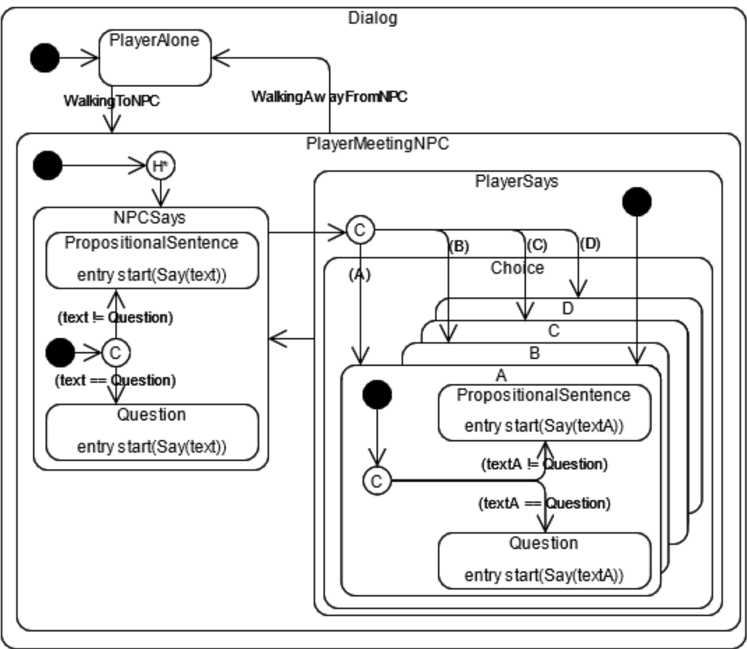


Figure 3.36: Dialog pattern - example for parametrized state and activity executing a game mechanics method

Figure 3.37 is a good example for two parallel storylines that interact with each other. In the main storyline, the player gets an order from the blacksmith to find the king's sword. Once he has found it and given it to the blacksmith, he receives a reward. After the player has found the sword in the main storyline, he will meet the miller in the parallel side storyline, who promises him a big reward for the sword. If the player still has the sword, he can hand it directly to the miller and collect the reward. If, on the other hand, he has already given it to the blacksmith, he can steal it back and then give it to the miller. In this example, the path of the subplot depends strongly on the progression of the main plot.

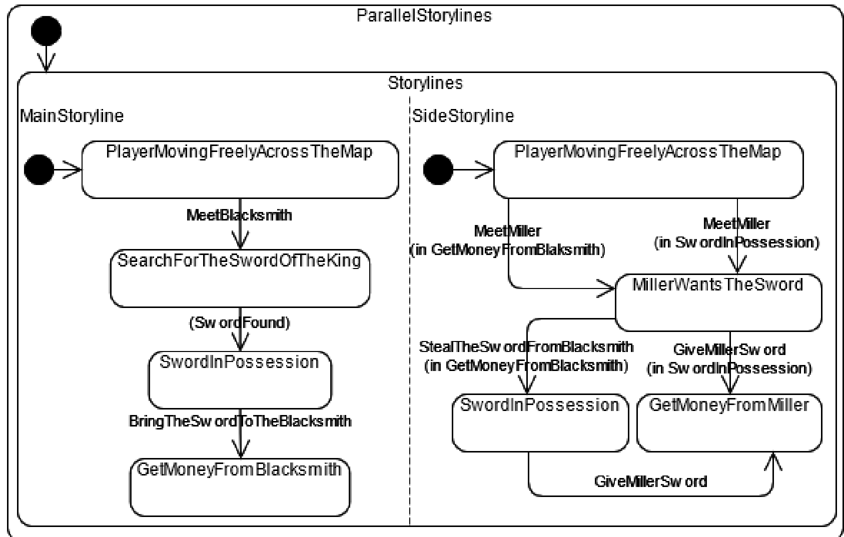


Figure 3.37: Parallel storylines - two parallel storylines interacting with each other

4 Prototype

Up to this point, it has only been shown for single features of statecharts that they can represent story elements. Therefore, in the following section, a prototype is developed to show how the story of a concrete game scenario can be managed by a statechart. Furthermore, it should be practically proven that statecharts can be a possible management form and implementation of storylines and storytelling-mechanics in games.

4.1 Advantages of a Paper-Prototype

The prototype is not an engine-based videogame, but a kind of paper-prototype. Paper-prototyping is a means to conduct user testing early, quickly, and frequently, and to elicit user ideas, as well as to test the concepts of the designers in the early design phase [29]. For this purpose, a graphical user interface (GUI) is tinkered with the simplest means such as paper, scissors, stickers, and glue [29]. In this way, the time and effort required to create a working, coded user interface can be easily and inexpensively bypassed [29]. As a result, design concepts can be quickly created, tested, modified on the fly, and retested [29]. Since production costs and development times are so small, many alternative designs can be tested simultaneously, and because the production tools are so easy to use, users can participate themselves and suggest changes and new ideas [29]. However, not only GUIs can be sketched, but also other graphical elements of a software. In this section, paper-prototyping is used to depict the game world and gameplay. Though, this is not done with paper as traditionally, but through photographically captured gameplay-situations represented by "Playmobil" figures. In this way it is possible to create graphically better proximity to a real implemented computer game. Buxton described in 2007 that sketching the user experience can involve more than simple paper-based prototypes [30]. As experiments have shown, cameras, televisions, tablet PCs, string, cardboard, and people are also effective means of sketching out elements of a software application in an early stage of development, with the same advantages that a paper-prototype offers [30].

Specifically, with respect to the goal of showing that a story of a concrete game scenario can be managed by a statechart, a paper-prototype has other advantages beyond simplicity and low cost. The basic idea that runs through the entire prototype and is thus also expressed in the form of the prototype-medium itself is reduction to show the essentials. An attempt is made to fade out everything unimportant and unnecessarily complicating to test the core concept. The chosen medium offers the optimal framework for this. The essential points can namely be shown without time-consuming and error-prone implementation in a computer game-engine like "Unity 3D" or "Unreal Engine". For the goal pursued here, an implemented prototype would be particularly costly, since one would first have to provide a framework for the “Unity 3D” engine to handle statecharts or hard-code the scenario, for example by Enums and switch-cases. However, this would shift the focus away from the core concepts we actually want to show and towards the framework needed to implement them. Furthermore, the additional effort that would have to

be put into level design, character design, non-playable character implementation and game mechanics would not be in proportion to the added value of knowledge that one only gets through a real implementation of the prototype. To show only the practicability of the concept of representing stories by statecharts, an abstract paper-prototype is sufficient in the first step. In a next step, the concept can then be transferred to an in a game-engine coded prototype to fully show the practicability.

4.2 Interaction of Statechart with Game Environment in the Prototype-Context

In Advance: The term game environment describes the part of the game that includes the rest of the game logic, game mechanics and game elements besides the statechart. As Mifrah Ahmad defined in his paper "Educational Games as Software Through the Lens of Designing Process", the game environment is a dimension which "collaborates game rules, objectives, subject, and theoretical aspects together as a whole to provide an interactive flow of activity." [31].

The overall idea of the prototype is to design a statechart that represents the story of a medieval open world game. However, the representation in this prototype should be abstract and reduced to be able to better highlight the essential aspects and thus avoid unnecessary complexity that offers no added value. For that reason, the statechart-events and -actions represent only non-concrete interfaces to elements of the game environment. They are not to be understood as concrete method-calls or functions interacting with specific game mechanics. For example, in figure 4.1 the action "Alarm" in the "Strategy"-component only expresses that "Alarm"-events are triggered within the statechart and that there is a possibility that an "Alarm"-method can be executed outside the statechart in the game environment. Whether this is the case and if so, what happens in this method is not relevant. It is equally irrelevant which method in the game environment triggers an event of the statechart. For example, in figure 4.1, it does not matter whether the "DragonSighted"-event in the "Storyspace"-component was triggered by an action of the statechart or by an external method. Also, it does not matter how the external method that triggered "DragonSighted" is implemented and what other effects it has on other game-parts. Nevertheless, it should also be mentioned that the statechart must interact with the remaining parts of the game environment via these interfaces in a real implementation, since it would not be possible to implement all the necessary game mechanics in a statechart. For example, many physical calculations or also the control of many graphic elements firmly belong to a computer game, which can be converted only elaborately or not at all into Statecharts. Specifically, for instance, a fanfare-sound could be played in the game environment method triggered by the "Alarm"-action of the statechart. Also, the non-playable characters could be moved to the current player position to see why the alarm was raised. Especially the latter would be difficult to implement in a statechart, because then the movements for every single non-playable character would have to be managed in another parallel component. However, as already mentioned, this is hidden in the prototype for simplicity, as the intention is to show only the functionality of the core concept.

Besides events and actions, the most important part of statecharts are the states. In the prototype, these represent the present state of the game, as was already determined in the mapping part of this work. States can act as interfaces to the game environment just

like actions and events. For example, in figure 4.1, a method of the game environment can be called in the "ReportLeaderAboutCamp"-state. This method then plays a cutscene based on the "VarTreasureFound"-variable. In the first version the player reports about the treasure stash, while in the other version the player reports about his escape. After the cutscenes have been played, the variable "ReportFinished" can be set to true in the game environment method, which then fulfills the condition in the statechart and activates the transition to the state "PickUpEquipment". So, there can be a bilateral exchange between the statechart and the game environment. As with actions and events, however, the prototype only gives vague reference to these interfaces and does not call concrete methods or interact in any other specific way with concrete elements of the game environment.

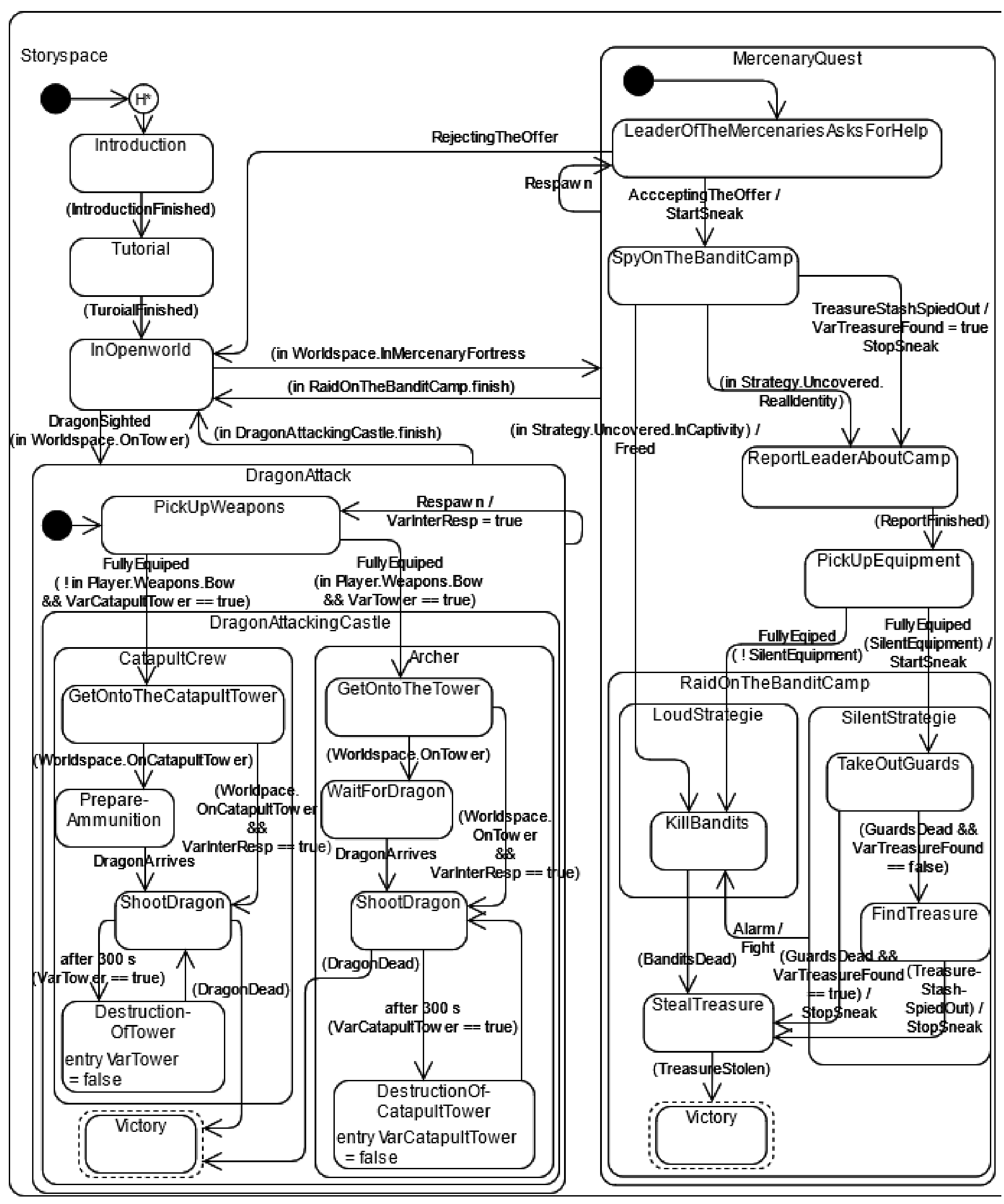
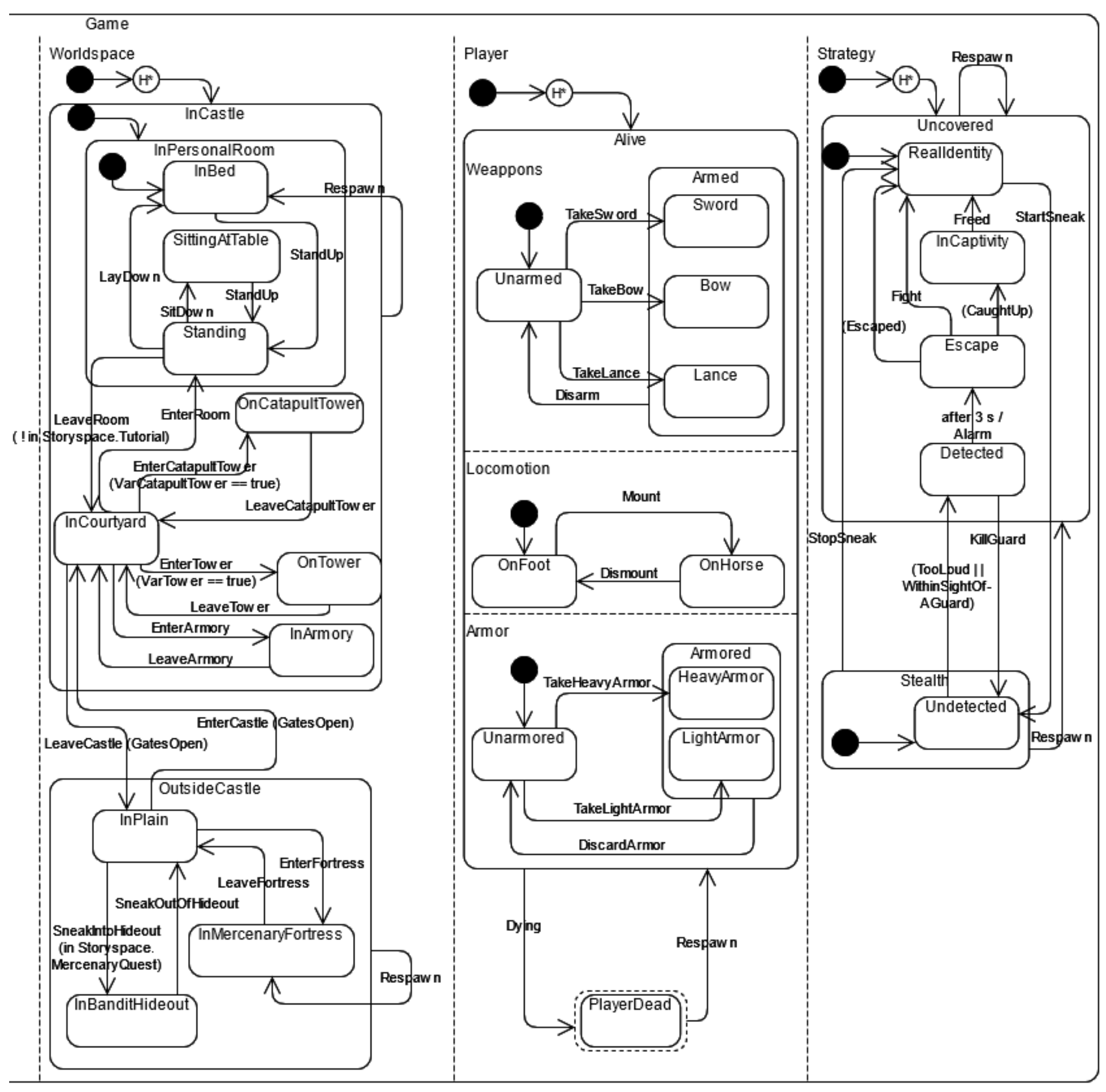


Figure 4.1: Statechart-Prototype


Game
Worldspace
InCastle
InPersonalRoom
InBed
Respawn
SittingAtTable
StandUp
LayDown
StandUp
SitDown
Standing
LeaveRoom
(! in Storyspace.Tutorial)
EnterRoom
OnCatapultTower
EnterCatapultTower
(VarCatapultTower == true)
LeaveCatapultTower
InCourtyard
EnterTower
(VarTower == true)
OnTower
LeaveTower
EnterArmory
InArmory
LeaveArmory
EnterCastle (GatesOpen)
LeaveCastle (GatesOpen)
OutsideCastle
InPlain
EnterFortress
LeaveFortress
SneakOutOfHideout
SneakIntoHideout
(in Storyspace.
MercenaryQuest)
InMercenaryFortress
Respawn
InBanditHideout
Player
Alive
Weappons
Armed
TakeSword
Sword
Unarmed
TakeBow
Bow
TakeLance
Lance
Disarm
Locomotion
Mount
OnFoot
Dismount
OnHorse
Armor
Armored
TakeHeavyArmor
HeavyArmor
Unarmored
LightArmor
TakeLightArmor
DiscardArmor
Dying
Respawn
PlayerDead
Strategy
Respawn
Uncovered
RealIdentity
Freed
StartSneak
InCaptivity
Fight
(CaughtUp)
(Escaped)
Escape
after 3 s /
Alarm
Detected
StopSneak
KillGuard
(TooLoud ||
WithinSightOf-
AGuard)
Stealth
Undetected
Respawn

4.3 Setting of the Prototype

Now that all the technical conditions have been clarified, the next sections will focus on the concrete content of the prototype. First, we will take a closer look at the setting.

The game is set in a medieval fantasy environment, in which the player is immersed as a knight. Armed with sword, lance or bow, the knight experiences many adventures, such as the attack of a dragon or the raid of a bandit camp with mercenaries. The player can always choose between no armor, light armor and heavy armor. Also, in quests, different paths lead to the goal. Often, tasks can be accomplished with a loud, conflict-ridden crowbar-strategy or with the help of a quiet stealth-approach. Outside of quests, the open world can be explored on foot or on horseback. Thereby, the player can visit some interesting locations, such as the castle where the knight lives or the mercenary fortress. In the appendix, a table (appendix 2) can be found contrasting the base states of the knight and corresponding images of the paper-prototype.

4.4 Quests and Interrupts

The story is structured in such a way that first an introduction and a tutorial must be gone through before the player is released into the open world, as can be seen in figure 4.2. In the same figure it can also be observed that in the open world state, quests and interrupts are started based on the player's position. For example, when the player meets the mercenaries in their fortress, the "Mercenary"-quest can be accepted, in which a bandit hideout is to be robbed. If the player enters the lookout-tower, a dragon is sighted and the "Dragon Attack"-interrupt begins. However, in the case of the interrupt, there is also the "DragonSighted"-event, which must occur in addition to the player entering the observation tower. This is because without the event, the interrupt would be triggered every time the player is in the open world – so not in a quest or interrupt – and enters the tower. Unlike quests, interrupts cannot be rejected and played later, but start right away regardless of decisions and the current situation of the player. Therefore, the existence of this event prevents the interrupt from being triggered accidentally at an inappropriate time and offers the game environment the possibility to co-determine when the interrupt should take place. For example, the game environment can control that the player is only able to trigger the "Dragon Attack"-interrupt after a certain event or quest, for instance, when the knight has learned to shoot from an archer in an "archery"-quest. This gate-event also prevents the player from retriggering the interrupt after it has already been completed. Beside or instead of an event behind which the interrupt is locked, a condition like whether it is day or night may exist.

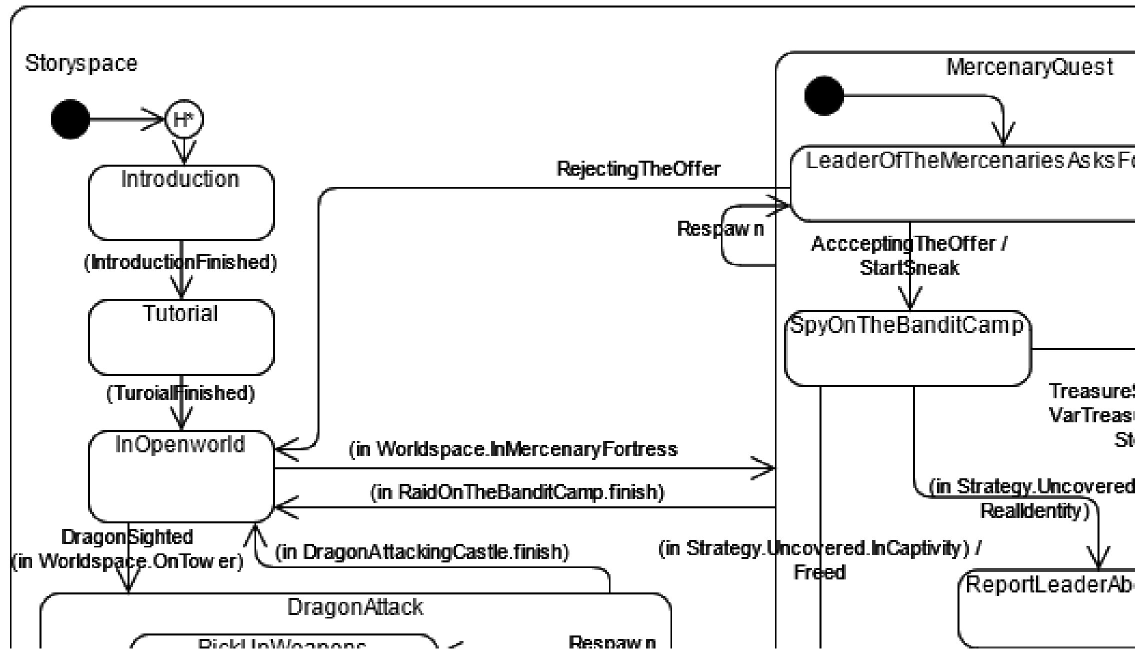


Figure 4.2: Cutout from Figure 1: TriggeringQuestInterrupt

The above section has already mentioned the biggest difference between interrupt and quest. As said Interrupts occur suddenly and cannot be rejected. This stems from the fact that an interrupt is an exciting, unpredictable event that interrupts the storyline, but also provides increased excitement and better immersion, since it cannot be postponed and the player usually must react quickly to the event. So, a certain suddenness is combined with time pressure. Quests, on the other hand, are tasks that the player gets in different places and from different non-playable characters, for example the mercenaries. Unlike interrupts, quests can be started at the desired time. If the player activates the quest unintentionally, for example by accidentally getting close to the mercenaries, he has the possibility to reject it first, and then return later to start the task.

Besides the time when quests or interrupts are started, they also differ in the respawn-process after death. Quests require the player to restart the entire task, while interruptions allow re-entry at a certain point. This difference results from the fact that, unlike quests, interrupts cannot simply be repeated, as the desired surprise effect and time pressure would then be lost. This would make the game experience far less immersive. However, if the player is thrown directly back into the action, as before death, the immersion can be maintained, since the same emotions, such as time pressure and stress, are immediately perceived again. The immersion is also preserved by the fact that the player is just knocked out and does not die in the interrupt. So, being knocked out can be thematized in the story in the form of cutscenes or tasks that have changed compared to the first time or simply by the plot that has continued, for example, that the dragon has raged even more in the castle during the protagonist's fainting in the "Dragon Attack"-Interrupt. If the player would die in the interrupt and respawn instead of just get knocked out, this could not be referred to in the story, because then a logical break in the storyworld would be created, since the protagonist does not have unnatural abilities. So, in the thematization of "dying" within the storyline lies another difference between interrupts and quests, since in a quest however, reference is never made to the previous failed attempt. Therefore, in a quest the protagonist can actually die and is not just knocked out, since, when

this occurs, the part of the story must be repeated, and the failure does not simply flow into the storyline as is the case with interrupts.

To conclude the quest and interrupt section, it should be mentioned that the interrupt and quest implemented in the prototype are only examples and can be extended by further quests and interrupts. For example, a quest in which the tower is rebuilt or an interrupt in which the castle must be defended from an attack by an enemy army is conceivable.

4.5 Story of the Prototype

Whereas in the previous section quests and interrupts were mainly examined formally, we will now focus on their storyline. Figure 4 in the appendix shows for every step of the story a comparison of the active states and transitions of the statechart with images of the paper prototype to illustrate the example story flow described in the following in text form. The complete statechart behind this story can be found in figure 4.1. Since the player experiences the world through the knight, the following will narrate the player's tasks and actions through the eyes of the knight. So, when it is said that the knight is trying to hit the dragon with the catapult, it means that the player, acting through the knight, is shooting at the dragon with the catapult.

As already mentioned, the player first goes through an introduction and a tutorial in the personal room in the castle. Once this is completed, the knight can leave the room and move freely in the open world. Thereby, the equipment and the mode of moving around can be freely chosen.

When the knight enters the observation tower, the interrupt "DragonAttack" is triggered as already explained above. In the for this necessary "DragonSighted"-event, a guard alerts the knight about an approaching dragon. This could be implemented in the game by a cutscene. Now, the knight must hurry to the armory to get equipped for the up-coming attack. The weapon selection determines, which squad the knight will join. If he chooses a bow, he will take position with the rest of the archers on the lookout-tower and if he chooses another weapon or even no weapon, he will help the catapult-crew on the catapult-tower. In figure 4, the knight decides to use a lance as a weapon, so he sets off for the catapult-tower. Once he has climbed the tower, he must prepare the ammunition. As soon as the dragon reaches the castle, the knight tries to shoot it with the catapult. If this takes more than five minutes, the dragon destroys the other tower. So, if the player is an archer, the catapult-tower is destroyed and if the player is in the catapult-crew, the lookout-tower is brought down. The tower is then not only destroyed in the interrupt, but also after completion of the interrupt in the open world and can no longer be entered by the player. The interrupt will only end when the dragon is killed. If this is achieved, the knight will be released back into the open world after a sprawling feast. If the knight is knocked out in the quest - that is, if he dies - he wakes up unarmed in his bed. Since the attack is still ongoing, the knight takes out his weapon and armor from the armory and gets to the tower corresponding to his equipment. Once he has reached it, however, this time there is no need to prepare ammunition or wait for the dragon to arrive, but he immediately returns to the battle with the dragon. This is technically achieved by a variable in a condition, which is set to true on respawn. If the other tower was already destroyed before the respawn, this is also the case after the respawn. Accordingly, an equipment selection option in the armory would then also be omitted. If the lookout tower is destroyed, the player can no longer choose the bow, since the tower from which

the archers shoot has been destroyed, and if, on the other hand, the catapult tower is destroyed, the player must choose the bow, since the catapult has been destroyed. But if no tower has been destroyed before death, this happens after five minutes of combat after the respawn. Here again, a variable in a condition ensures that the tower cannot be destroyed twice by mistake.

Now that the dragon is defeated, the knight can move freely through the open world again. If one of the two towers was destroyed in the attack, it can no longer be entered, which is achieved by a condition on the entry transition. The destroyed tower is also a good example of an element that creates a dynamic world. What happens in a quest or interrupt has far-reaching effects on game elements and story progression. For example, now a quest, which can only be played when a tower has been destroyed in the interrupt and in which the player must convince craftsmen to help rebuild this tower, would be conceivable.

One quest implemented in the prototype is the "Mercenary"-quest. This quest is started when the knight approaches the mercenary fortress. In the fortress, the knight meets the leader of the mercenaries, who tells about a bandit hideout and stolen gold coins. But to steal back the coins, they need the knight's help. At this point, it is up to the player to accept or decline the offer, and then come back later to play the quest. In this context, the new statechart feature "ExtendedHistory" described in section 3.5 "New Statechart Features and Pattern" could be used, so that the player is not welcomed twice with the same cutscene by the leader of the mercenaries but is asked if he has changed his mind now. For reasons of clarity and reduction, however, the author has decided not to use this feature in this case, since it has already been demonstrated in section 3.5 "New Statechart Features and Pattern" using an example game situation and would therefore not offer much added value in this context.

If the knight decides to help the mercenaries, he sets out to spy on the bandit camp and locate the treasure stash. Once sighted, the knight sneaks back to the mercenary camp to report to the leader. However, if the knight makes too much noise or enters the field of vision of a guard, the bandits will take notice of him. It should also be noted that heavy armor is more susceptible to cause noise than light armor or none. However, this is difficult to model in a statechart since we only check the "TooLoud"-condition. In the game environment, the caused noise level can be calculated every tick in the game loop, differentiated by armor and movement speed. If the noise level is too high or the character is in the field of view of a guard, either the condition "TooLoud" or "WithinSightOfAGuard" is set to "true", and the knight is detected. If he kills the guard within three seconds, his unmasking will not be noticed, and he can continue sneaking. But when this fails, the guard sounds the alarm, alerting the other bandits to the intruder. For the knight, the only option is to flee, because the fight option requires an event in the "Strategy"-component, which does not occur in this situation, but will only play a role later during the raid. There, if the player opts for the silent strategy and is discovered, he cannot flee but must fight. But back to the escape after the knight was discovered scouting the bandit hideout: The escape is more likely to succeed if the knight has left his horse nearby, as riding is faster than escaping on foot. However, this cannot be directly modeled in the statechart – just like the different noise levels of armor when sneaking –, since an escape without a horse should not be one hundred percent unsuccessful in advance. A horse should only increase the chances of escape. Therefore, the condition whether the knight is on foot or on horseback cannot be the only aspect that determines whether the escape is successful, but also factors such as where he rides or runs, how saddle-firm he is or whether he is a

fast runner, or also if he is lightly or heavily equipped play a role. Evidently, this requires a lot of calculations and the evaluation of many aspects in the game environment, which then transmits the result of the escape to the statechart in the form of the conditions "Escaped" or "CaughtUp". However, if one would only include a transition in the statechart, checking whether the knight is on horseback or on foot, all other factors except the locomotion factor would be disregarded. But back to the storyline. If the knight is caught, he is imprisoned by the bandits. Thereupon, the mercenaries attack the bandits' hideout and free the knight. As a result, the player no longer has the choice of opting for the quiet strategy but is drawn directly into the loud solution approach. In the loud strategy, the mercenaries and the knight try to capture the treasure directly without sneaking up quietly, looking for the open fight where they must take on all the bandits. Once all of them are dead, the treasure can be captured, and the quest is over. However, if the knight is not caught by the bandits after being discovered, there are two ways the story can develop. First, as is often done in video games, everything could be reset to the situation before the knight was discovered. This means that the guards would patrol around again, as if no incident had ever happened, and the player would try once more to steer his knight through the hideout unnoticed. However, on the story level this approach is irrational and would violate immersion. Why would guards, who have just pursued and lost an intruder, go back to day-to-day business? A logical step is to increase the number of guards and make it impossible for another unnoticed intrusion. In this prototype, the increased vigilance is used as a logical justification why scouting again after a successful escape is no longer an option for the knight and he returns to the mercenaries instead. In the mercenary fortress, he reports to the leader about his escape and the knowledge gained until then. After that, the knight equips himself and thus chooses the strategy he wants to use. This is implemented by a Boolean variable set by the game environment. If he chooses the heavy armor and sword or lance, the plan, as described above, is to attack the bandits in an open battle and fight his way to the treasure. If he chooses the light armor or no armor and the bow, he thus chooses the quiet strategy. The execution of this strategy is shown in figure 4. First, the knight sneaks up to the bandit guards to take them out quietly. The same applies as before when scouting. If the knight is noticed by a guard, he must kill him within three seconds, otherwise the guard will sound the alarm, which will result in a change of strategy to the loud one, where the mercenaries must kill all the bandits to get to the treasure. Technically, being spotted in this situation is nothing different than being spotted when scouting earlier in the quest, except that a different way out is chosen. As described above, now the knight does not try to flee but goes over to the loud strategy where the mercenaries openly attack the bandits. To prevent the player after he was discovered from escaping as in the spy-phase, a series of actions and events are executed. The "Alarm"-action in the "Strategy"-component triggers the "Alarm"-event in the "Storyspace"-component, which in turn activates the "Fight"-action in the same component. Then, this "Fight"-action triggers the "Fight"-event in the "Strategy"-component, which ensures that the player does not flee, but that the "RealIdentity" state is reached immediately, thus ending the sneaking. This is just a technical trick that is necessary to be able to use the general "Strategy"-component in all sneak-situations, since it includes both basic natural behaviors in the face of danger – escape and attack – to get out of the situation after being spotted. Also, the escape- and the attack-option are both required to make sure that the component is always tailored to the current situation, so that the logic of the storyline is maintained. From the story level, it is more logical if an open fight is not possible when spying and an escape is not possible when raiding, because both would not be effective in the respective situation and would lead to unwanted side

effects, such as being caught and imprisoned while raiding. These, in turn, could lead to bugs, such as being captured during the raid but then not being freed because the raid is already in progress. In figure 4.3 are both paths for the raid-situation and the spy-situation illustrated.

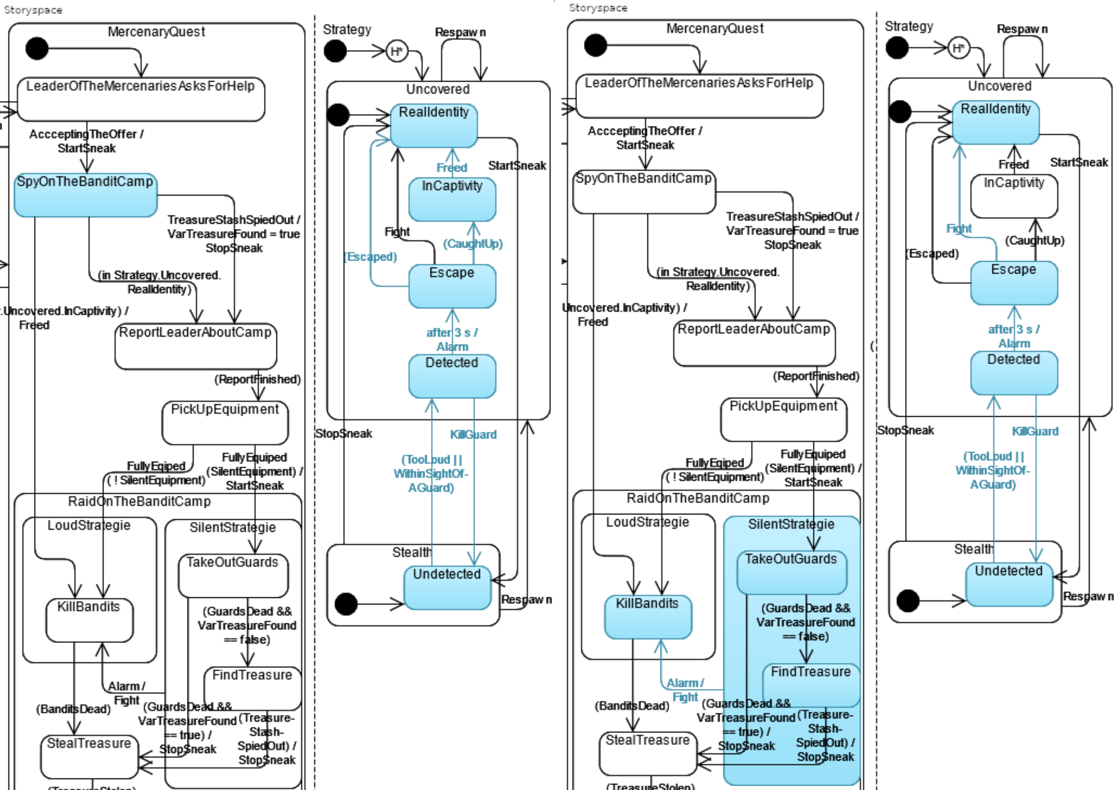


Figure 4.3: Cutout from Figure 4.1: Different paths in the "Strategy"-component for the spy-part and the raid-part of the "Storyspace" after being spotted.

But back to the story. Once the knight has successfully eliminated the guards, if he has not succeeded in doing so while spying, the search for the treasure begins. At any time, however, the mercenaries must be on the lookout for newly emerging bandits, as the troop can be discovered at any time. When the treasure is finally found, the knight's last task is to steal it with the mercenaries and bring it to the mercenary fortress. There, a gleeful feast is celebrated, and the quest is successfully completed. The prototype could now be extended in such a way that the knight has gained the goodwill of the mercenaries by successfully completing this quest, which means that they will now rush to his aid in dangerous situations, or he will receive further quests from them.

To conclude the story chapter, let us mention what happens when the knight dies in the quest. If this is the case, the quest must be restarted from the beginning. This provides an additional difficulty. However, since the game should not be frustrating, the knight wakes up unequipped in the closest point to the starting point of the quest, so that he does not have to travel a long way. In the case of the "Mercenary"-quest, the knight would respawn in a bed in the mercenary fortress. On the story level, respawning and starting from the beginning does not really make sense. However, this is a common practice in video games to solve the problem of dying. By treating the failed attempts on the story level as if they never happened, it allows to tell a stringent story where there is no need to create supernatural characters that come back to life. Another widely used approach is

to explain that the player does not die but is just knocked out as mentioned in the "Quest and Interrupt"-section 4.4. This also prevents the player from needing supernatural abilities. This approach, which is also employed in the "Grand Theft Auto" game series [32], is not only used in the prototype for "dying" in interrupts but is also applied when the knight "dies" in the open world. If he is knocked out in the castle, he will wake up in the bed in his personal room and if he is knocked out outside the castle he will spawn in a bed in the mercenary fortress. Both times he is unequipped, which means that he is on foot, without weapon and armor. On the story level, this can be justified by the fact that people find the unconscious knight and carry him to the nearest bed.

4.6 Parallel Components in the Prototype

As already shown by the stealth feature, orthogonality is essential for the implementation of game stories by statecharts. Basically, it can be said that for each core element of a game a parallel component exists as representation in the statechart. This is also the case in a "Unity 3D" project, where there is a separate "C#"-script for each core feature. On the one hand, the parallel components represent different levels of the game, for example, as in the prototype, the "Storyspace" level, the "Worldspace" level and, on the other hand, orthogonal components also manage different elements, such as states and equipment of the player in "Player" or core features, such as the stealth feature in "Strategy".

All components depend on each other and interact in numerous ways. This can be seen for example in figure 4.1 in the "Worldspace"-component, where the private room can only be left after the "Tutorial" state has been left in the "Storyspace"-component or in the "Storyspace" where the next step of the story after “SpyOnTheBanditCamp” depends on the active state in the component "Strategy". Another example of an interaction of different components by a condition is whether the towers in the "Worldspace" can be entered. Here it depends on a Boolean variable set in the "Storyspace" and not on an active state of another component, because in contrast to the examples with the tutorial only in the own room or the continuation of the story based on the active state of the "Strategy"-component, in the "Storyspace"-component, it is not lingered in a state when the tower is not enterable. The storyline continues while the tower is still destroyed. Therefore, the accessibility of the towers is stored in variables. Also, conditions between components can make areas and features accessible only in certain situations. For example, in figure 4.1, the player can only venture into the bandits’ hideout if he is in the "Mercenary"-quest. This can prevent the player from accidentally encountering the bandits beforehand and interacting with them in a way that makes the later quest impossible or at least implausible. As the numerous examples show, dependencies between parallel components are represented either by Boolean variables or by the "in <parallelComponentName>.<stateName>"-condition.

However, parallel components can interact not only due to conditions, but actions and events of different components can also provide interaction of orthogonal components. A good example for interaction of actions, which in turn trigger events in different components, is given in the previous section and in figure 4.3 by the situation in which one was spotted while sneaking in the silent strategy of the raid. However, the exact interaction has already been discussed in detail in the previous section, so we will not go into more detail here.

Outsourcing core mechanics and core features into parallel components leads to low cou-

pling and high cohesion, which is always desirable in software engineering for reasons of reusability, better maintainability, and extendibility [33]. Coupling describes the strength of the interconnection of one module to another module [33]. At the desired low coupling, little interaction takes place between different modules of a software project [33]. Cohesion, on the other hand, specifies the level of exchange within a module [33]. When talking about high cohesion, there is a high level of exchange within each module [33]. However, as we have seen in the examples above, the prototype has many dependencies and interactions between the different parallel components, so that, strictly, one cannot speak of low coupling in this example. Nevertheless, the advantages of high cohesion remain.

Due to the parallel subsystems of a game, when a global event occurs, reactions to it take place in many distributed locations. This can be seen very well during respawn. Here, every component reacts in the form of a transition, usually to the default state of the component. This is visualized in figure 4.4. To make the knight lie unequipped in the nearest bed, the respawn element in the "Player"-component must reset the settings of the knight via the default-entries in the "Alive"-state. Likewise, in the “Worldspace” the knight must be moved to the nearest bed depending on its death point via direct transition and his behavior must be transferred to "RealIdentity" in the "Strategy"-component. In the "Storyspace" the state to be reached depends on the state in which one was at the time of death. If one was in the open world, one is also in this world after respawning, therefore the self-transition was renounced here. It is also possible to provide for change at another point by, for example, setting a variable during the respawn, as it is the case in the prototype in "DragonAttack".

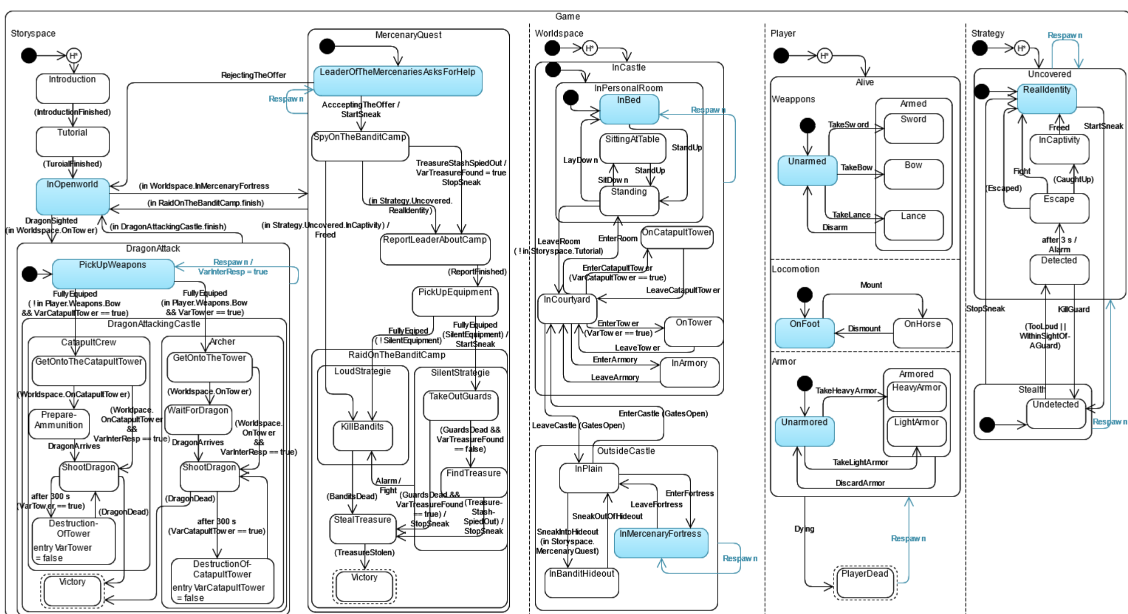


Figure 4.4: Illustration of all distributed respawn elements.

Even if the parallel components mean additional work for global events, as we have just seen, there is a great advantage in this approach. As shown in figure 4.5 using an example situation from the "Mercenary"-quest, the simple use of colors makes it possible to see immediately which states the game is in. So, for each part of the game, which is represented in the statechart, the concrete progress can be determined. Thus, in figure 4.5, it is very easy to determine that the knight is spying on the bandit hideout on foot with sword

and light armor as part of the "Mercenary"-quest and has not yet been discovered. This advantage could be used in a framework tool for statecharts in game engines. During debugging, the developer could immediately see which states are currently active and thus quickly and easily determine where the statechart is in the wrong state and thus know in which section of the game system the error must lie.

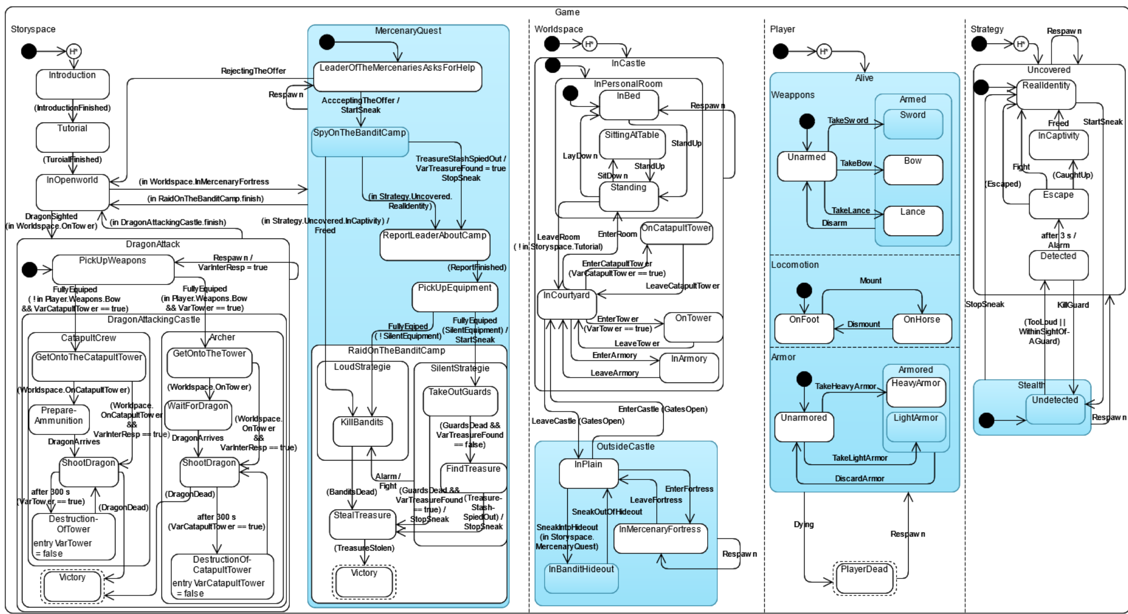


Figure 4.5: Example scenario to show the simplicity of seeing immediately which states the game is in due to colored statecharts.

Also, in a further step, a new statechart feature could be developed that simplifies the implementation of dependencies between parallel components. It can already be stated in the relatively small prototype-statechart that many dependencies between components quickly arise, which can be only implemented by "in <stateName>"-conditions using the conventional Harel features. But if we now imagine an extension of the prototype by another ten parallel components, like a night-day component or fight-mechanics, it is obvious that the further necessary dependencies get out of hand quickly. Developers would also benefit from a tool provided by the framework that visually displays dependencies. As you can see, more statechart features and framework extensions are conceivable.

4.7 History Entries and Final States

To conclude the chapter on the prototype, we will briefly discuss the use of history entries and final states in the prototype. In each parallel component, the default state leads to a history entry. The history entries have nothing to do with the respawn process, as the first thought might be, but ensure that if the large state "Game" should be left, the same situation before the exit can be recovered. This can be used for example for a menu or pause function or to provide a "safe and reload game"-feature when the program is shut down and restarted.

Three final states occur in the prototype's statechart. The two "Victory" final states represent the end of the quest or the interrupt. If these are reached, the quest or the interrupt

was successfully completed and the state open world becomes active again, which means that the player moves freely in the open world again. The third final state "PlayerDead", on the other hand, is never consciously used as a final state in the current prototype. However, on the logical level it makes sense to see the death of the player as a final state, which can be left again by respawning. "PlayerDead" can also be seen as the final state for the entire game, i.e., the "Game"-state. For example, you could easily remove the respawn feature and end the game with the death of the player. So, the death of the player would be the bad counterpart to the victory, by completing all quests or at least the main storyline.

4.8 Results of the Prototype

In this section, the paper prototype was successfully used to demonstrate that statecharts can manage concrete game scenarios and successfully implement storylines and storytelling-mechanics in games. At the beginning, the advantages of a paper prototype were identified, which lie particularly in its simplicity and cheapness. Subsequently, it was explained that the prototype-statechart is only meant to serve as an abstract illustration that interacts with the game environment, but the exact interaction will not be discussed. After the fantasy-medieval setting around a knight as the game character was set up, the major story parts interrupt and quest were compared, whereby it was determined that they differ primarily in the suddenness and inevitability of the Interrupt. Following, a run-through of an example quest and an example interrupt proved that storylines and game scenarios can also be implemented in statecharts in practice. This was supported by the comparison of the paper prototype, which represents the game scenarios with "Playmobil" figures in the form of pictures, and the flow of the storyline in the statechart. Finally, the peculiarity of splitting all core elements of a game into parallel components were discussed. This splitting then satisfies the important principles of software development: Low coupling and high cohesion. Whereby it had to be restricted that in the prototype the parallel components are strongly dependent, since often conditions of transitions are bound to the fact that a certain state is active in a parallel component or because actions trigger events in orthogonal components. This leads to the fact that low coupling is not completely given in this prototype. Nevertheless, with the splitting of all core elements of a game into parallel components, it is easy to see immediately for each part of the game in which state or situation it is. This was identified as a great advantage especially for debugging, which is also conceivable in future statechart frameworks in game engines.

5 Outlook

The next step would be to implement the paper-prototype in a proper game-prototype. For this, a games engine like "Unity 3D" could be used. However, one would first have to find or program a framework that allows statecharts-handling in "Unity 3D". Another option without a framework would be to hardcode the statechart scenarios in the game, which would be less time consuming

At this point, it should also be recalled that a new feature is needed to better display and add dependencies between parallel components, as identified in the prototype. Not only a more efficient feature for modeling dependencies between orthogonal components should be developed, but also in possible frameworks it is necessary to provide a tool that allows developers to immediately identify dependencies, since dependencies quickly get out of hand.

Furthermore, in the section 3.4.4 "Completion Event Handling with Exits", it was argued that unlimited event propagation is not useful when propagating the broadcasted EXIT-events. However, there are some use cases, where unlimited event propagation could be needed in context with statecharts. The caught event could be modified in the exit receiver when transmitted to the next receiver. This could be used, for example, to add a counter, which counts the number of receivers it traverses or the time until the next receiver catches the event. Moreover, a condition or the event itself can be changed, for instance, to prevent uncontrolled errors. If an error event is caught, it could be modified to only communicate to the outside that an error has occurred in this module, but due to this modification, the whole system can be kept from crashing. Furthermore, the side effects in the state entered only shortly until the next-outer receiver catches the broadcasted event could be used consciously. For example, the shortly visited state could perform actions that change variables or trigger events in other states.

Lastly, exits and final states can be extended to clean up the component that is to be exited. So, for example in these states, un-subscriptions or error-handling in the sense of concealment of the exception can be done. In this context, concealment of the exception means that one announces to the outside only that in this component an error has occurred and thus the exception is caught and not the entire system has to terminate, since, if possible, one always wants to avoid that the system comes to a standstill in an error state. This management of errors allows the system to continue running despite a component, in which an uncontrolled error has occurred. In addition to the message that an error has occurred in the component, a list can be passed on to the outside world in which it is possible to trace, where the unhandled exception is thrown in the faulty component.

6 Conclusion

As already summarized at the end of the prototype section, statecharts can be successfully used as environment for representing dynamic storytelling in games. Namely, it is possible to express storytelling elements through the features of statecharts, which can be seen in the mapping table Table 3.3, the several example-figures of statecharts, and the prototype. However, some disadvantages of using statecharts for representing game-stories were also identified. The representation of game scenarios quickly becomes complex and sprawling due to the many dependencies between the parallel components of the system. Furthermore, there is a risk that an event flow diagram is modeled instead of a statechart, because when developing the statechart, the first thing one has in mind is the sequence of events in the storyline. However, it can be helpful to first model the storyline as an event flow diagram consciously and only after that develop the statechart of the system with all its subsystems in orthogonal components. Another problem of statecharts is the lack of uniform syntax, as became visible when looking at the implementation of features in editors during the definition of exits and final states. An additional fundamental problem of statecharts is the ambiguous semantic, as already described by Harel, which also appeared in the section 3.4.2 "Completion Event Handling with Final States" when selecting the transition to be traversed in the case of several matching transitions [14]. Nevertheless, the use of statecharts to represent dynamic storytelling is worthwhile, since there are many positive aspects that dominate the negative ones. Statecharts provide a structured overview and a visualization of the hierarchical structure that reveals logic gaps in the story as soon as they are created. Furthermore, interactive elements and the interaction of the story with game-mechanics and with the game itself can be implemented very well by statecharts using actions, activities, or trigger events, since statecharts were originally developed for reactive technical embedded systems [4]. In addition, parallel storylines can hardly be modeled as well with anything else as with statecharts. Even in text form, concurrent situations are harder to describe. Also, the existing statechart-feature-canon can be extended very easily by further features, which can be important for story representation. Apart from that, many features out of the existing statechart-feature-canon can be directly used to represent storytelling elements, as the section 3.6"Mapping Storytelling Elements onto Statechart Features" has demonstrated.

List of Figures

2.1 Cutout of overview of decision-path of the game "Detroit: Become Human" (Source: [5]) 2
2.2 Example game scenario from "Fungus" in "Unity 3D" (Source: [6]) 3

3.1 OR state and AND state 6
3.2 Situation from left is shown on the right side with the help of a parametrized state (Source: [14], Fig. 38, 39) 7
3.3 Overlapping states (Source: [14], Fig. 41) 7
3.4 Comparison of situation with conditional entrance and without (Source: [14], Fig. 33) 8
3.5 Shallow history and deep history 9
3.6 Different transition-types 11
3.7 Transition hubs (Source: [14], Fig. 17) 12
3.8 Application of actions and activities 13
3.9 Abstraction to black-box-view 14
3.10 Refinement to white-box-view 14
3.11 Unclustering (Source: [14], Fig.36) 15
3.12 Final state and finished-signal of "QT Core - The State Machine Framework" (Source: [21]) 16
3.13 Exit point of „UML Diagrams – State Machine Diagrams" (Source: [22]) . 16
3.14 Named exit state - notation: #<exitStateName» of "YAKINDU Statechart Tools" (Source: [23]) 17
3.15 Final state with "in <componentName>.final"-condition 18
3.16 Multiple final states with "in <componentName>.final"-condition 19
3.17 Multiple final states with "in <componentName>.finalAll"-condition . . . 19
3.18 Final state with multiple "in <componentName>.final"-conditions 20
3.19 Multiple final states with multiple "in <componentName>.final"-conditions 20
3.20 Melted exit and exit receiver on single level with an EXIT flag and a leaving transition in implicit exit state 22
3.21 Orthogonal component with multiple exits melted with one receiver with EXIT flag 23
3.22 Orthogonal component with multiple exits melted with one receiver with EXITALL flag 23
3.23 Exit and exit receiver 24
3.24 Multiple exits and one exit receiver 24
3.25 One exit and multiple exit receiver 25
3.26 Melted exit and exit receiver and not melted exit and exit receiver 25
3.27 Multiple exit receivers - example for unlimited event propagation 27
3.28 Generic feeder states compound example 29
3.29 "Blacksmith"-quest with the introduction being realized via feeder states compound 30
3.30 Extended history situation but with only Harel features 31

3.31 Example situation extended history entrance 32
3.32 Example situation extended history entrance with two parallel quests . . . 33
3.33 Example situation extended colored history entrance 34
3.34 "BuildingAHouse"-quest with "DragonAttack"-interrupt - returning to unchanged building site 45
3.35 "BuildingAHouse"-quest with "DragonAttack"-interrupt - returning to changed building site 45
3.36 Dialog pattern - example for parametrized state and activity executing a game mechanics method 46
3.37 Parallel storylines - two parallel storylines interacting with each other . . 47

4.1 Statechart-Prototype 51
4.2 Cutout from Figure 1: TriggeringQuestInterrupt 54
4.3 Cutout from Figure 4.1: Different paths in the “Strategy”-component for the spy-part and the raid-part of the “Storyspace” after being spotted. . . 58
4.4 Illustration of all distributed respawn elements. 60
4.5 Example scenario to show the simplicity of seeing immediately which states the game is in due to colored statecharts. 61

List of Tables

3.1 Comparison of final states and exits 28
3.2 Comparison of extended history, extended colored history and Harel statechart features 35
3.3 Mapping of storytelling elements onto statechart features with gamestory examples 44

Bibliography

[1] B.Pfeifer, *Die drei ??? Kids und du: Rocky Beach steht kopf*. Franckh-Kosmos Verlags-GmbH & Co.KG, 2009.

[2] Sony Computer Entertainment Europe, "Detroit: Become human," 2018, Retrieved on 15.07.2021. [Online]. Available: https://www.quanticdream.com/en/detroit-become-human

[3] ——, "Until dawn," 2015, Retrieved on 15.07.2021. [Online]. Available: https://www.playstation.com/en-us/games/until-dawn/

[4] H. Züllighoven, "Object-oriented construction handbook. dpunkt. verlag," 2005.

[5] PowerPyx, "Detroit become human full walkthrough – all chapters," 2018, Retrieved on 15.07.2021. [Online]. Available: https://www.powerpyx.com/detroit-become-human-full-walkthrough-all-chapters/

[6] S. Halliwell, "What is a flowchart?" 2019, Retrieved on 15.07.2021. [Online]. Available: https://github.com/snozbot/fungus/wiki/flowcharts

[7] M. Cavazza, F. Charles, and S. Mead, "Character-based interactive storytelling," *IEEE Intelligent systems*, vol. 17, no. 4, pp. 17–24, 2002.

[8] M. Merabti, and A. El Rhalibi, Y. Shen, J. Daniel, A. Melendez, and M. Price, "Interactive storytelling: Approaches and techniques to achieve dynamic stories," in *Transactions on Edutainment I*. Springer, 2008, pp. 118–134.

[9] J. Brusk, and T. Lager, "Developing natural language enabled games in (extended) scxml," in *Proceedings from the international symposium on intelligence techniques in computer games and simulations (Pre-GAMEON-ASIA and Pre-ASTEC), Shiga, Japan, March*, 2007, pp. 1–3.

[10] J. Brusk, T. Lager, A. Hjalmarsson, and P. Wik, "Deal: dialogue management in scxml for believable game characters," in *Proceedings of the 2007 conference on Future Play*, 2007, pp. 137–144.

[11] A. Leclerc von Bonin, "Dynamic storytelling at internet briefing in zürich," 2015, Retrieved on 15.07.2021. [Online]. Available: https://www.dizmo.com/dynamic-storytelling/

[12] B. Ip, "Narrative structures in computer and video games: Part 1: Context, definitions, and initial findings," *Games and Culture*, vol. 6, no. 2, pp. 103–134, 2011. [Online]. Available: https://doi.org/10.1177/1555412010364982

[13] I. Donald, and H. Austin, "Playing with the dead: transmedia narratives and the walking dead games," in *Handbook of Research on Transmedia Storytelling and Narrative Strategies*. IGI Global, 2019, pp. 50–71.

[14] D. Harel, "Statecharts: A visual formalism for complex systems," *Science of computer programming*, vol. 8, no. 3, pp. 231–274, 1987.

[15] Visual Paradigm, "Online state machine diagram tool," 2021, Retrieved

on 15.07.2021. [Online]. Available: https://online.visual-paradigm.com/de/diagrams/features/state-machine-diagram-software/#

[16] E. Mogensen, "Welcome to the world of statecharts," Retrieved on 15.07.2021. [Online]. Available: https://statecharts.dev/

[17] ——, "Full list of glossary terms," Retrieved on 15.07.2021. [Online]. Available: https://statecharts.dev/glossary/

[18] B. Douglass, *Design patterns for embedded systems in C: an embedded software engineering toolkit*. Elsevier, 2010, pp. 26–27.

[19] Rational Software Corporation, "Concepts: Events and signals," 2002, Retrieved on 15.07.2021. [Online]. Available: https://sceweb.uhcl.edu/helm/RationalUnifiedProcess/process/workflow/ana_desi/co_event.htm

[20] K. Stenzel, and H. Seebach, "Softwaretechnik - kapitel 11: Zustandsdiagramme," Retrieved on 15.07.2021. [Online]. Available: https://www.informatik.uni-augsburg.de/lehrstuehle/swt/se/teaching/fruehere_semester/ws1112/softwaretechnik/unterlagen/Kapitel11-Update.pdf

[21] The Qt Company Ltd., "The state machine framework," 2021, Retrieved on 15.07.2021. [Online]. Available: https://doc.qt.io/qt-5/statemachine-api.html

[22] K. Fakhroutdinov, "State machine diagrams," 2020, Retrieved on 15.07.2021. [Online]. Available: https://www.uml-diagrams.org/state-machine-diagrams.html

[23] J. Dicks, "Advanced state machine modeling with entry, exit and final states," Retrieved on 15.07.2021. [Online]. Available: https://blogs.itemis.com/en/advanced-state-machine-modeling-with-entry-exit-final-states

[24] YAKINDU Statechart Tools, "Documentation: Statechart elements," 2021, Retrieved on 15.07.2021. [Online]. Available: https://www.itemis.com/en/yakindu/state-machine/documentation/user-guide/sclang_graphical_elements#sclang_exit_points

[25] R. Klute, R. Beckmann, S. Wendler, T. Kutz, R. Herrmann, A. Mülder, and Tangele, "Statechart language reference," 2019, Retrieved on 15.07.2021. [Online]. Available: https://github.com/Yakindu/statecharts/blob/master/plugins/org.yakindu.sct.doc.user/src/user-guide/statechart_language.textile#sclang_parentfirstexecution

[26] J. Barnett, R. Akolkar, R. Auburn, M. Bodell, D. Burnett, J. Carter, S. McGlashan, T. Lager, M. Helbing, R. Hosn, T. Raman, K. Reifenrath, N. Rosenthal, J. Roxendal, "State chart xml (scxml): State machine notation for control abstraction," 2015, Retrieved on 15.07.2021. [Online]. Available: https://www.w3.org/TR/scxml/#AlgorithmforSCXMLInterpretation

[27] S. Yacoub, and A. Ammar, "A pattern language of statecharts," in *Proc. Fifth Annual Conf. on the Pattern Languages of Program (PLoP'98)*, 1998, pp. 98–29.

[28] S. Alimadadi, S. Sequeira, A. Mesbah, and K. Pattabiraman, "Understanding javascript event-based interactions," in *Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 367–377.

[29] S. Brown, "Paper prototypes and beyond," *Visible Language*, vol. 43, no. 2, p. 198, 2009.

[30] B. Buxton, "Sketching user experience: Getting the right design and the design right," *São Francisco: Morgan-Kaufmann*, 2007.

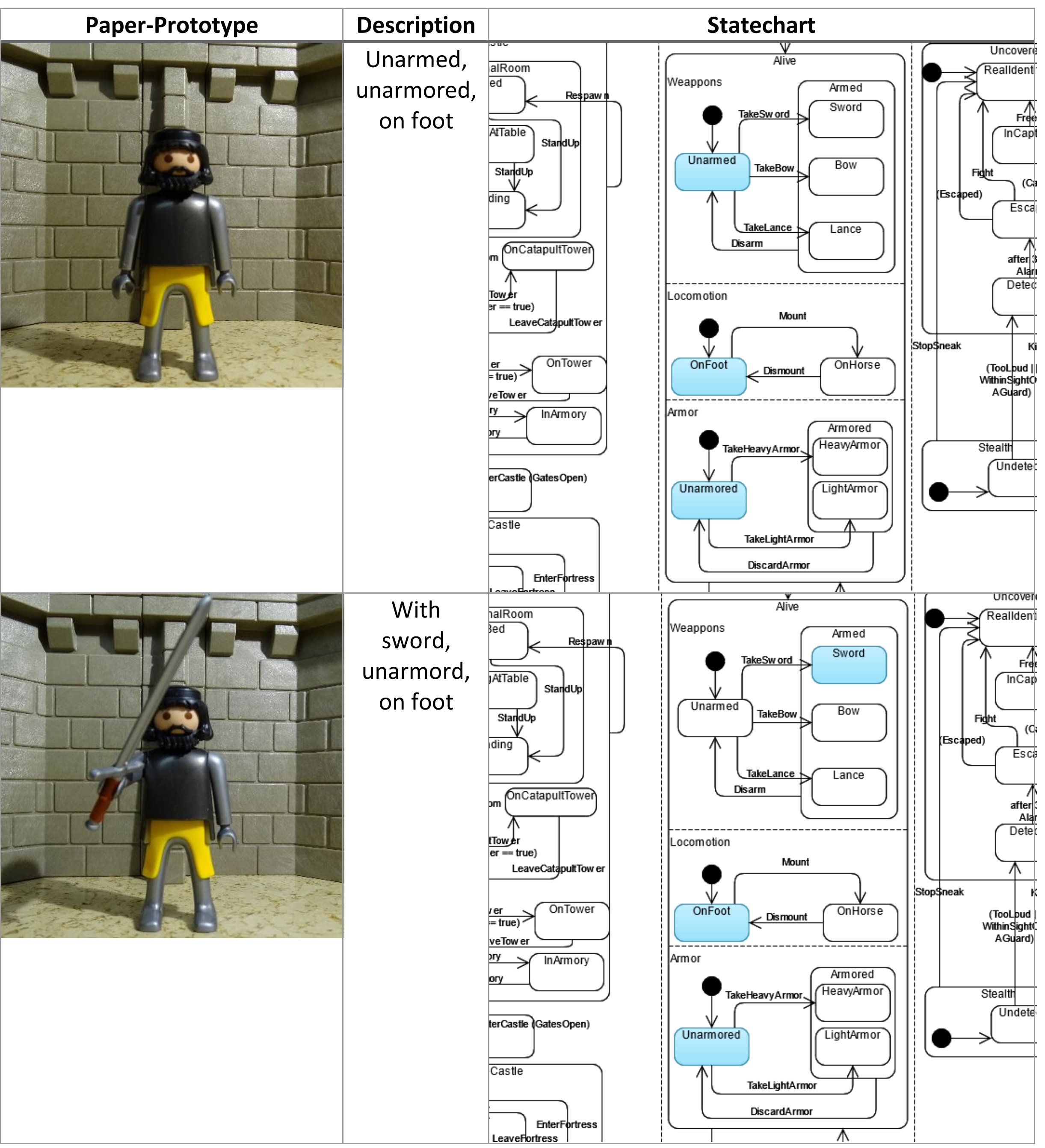
[31] M. Ahmad, "Educational games as software through the lens of designing process," in *Handbook of Research on Modern Educational Technologies, Applications, and Management*. IGI Global, 2021, pp. 179–197.

[32] GTA Wiki, "Wasted," Retrieved on 15.07.2021. [Online]. Available: https://gta.fandom.com/wiki/Wasted

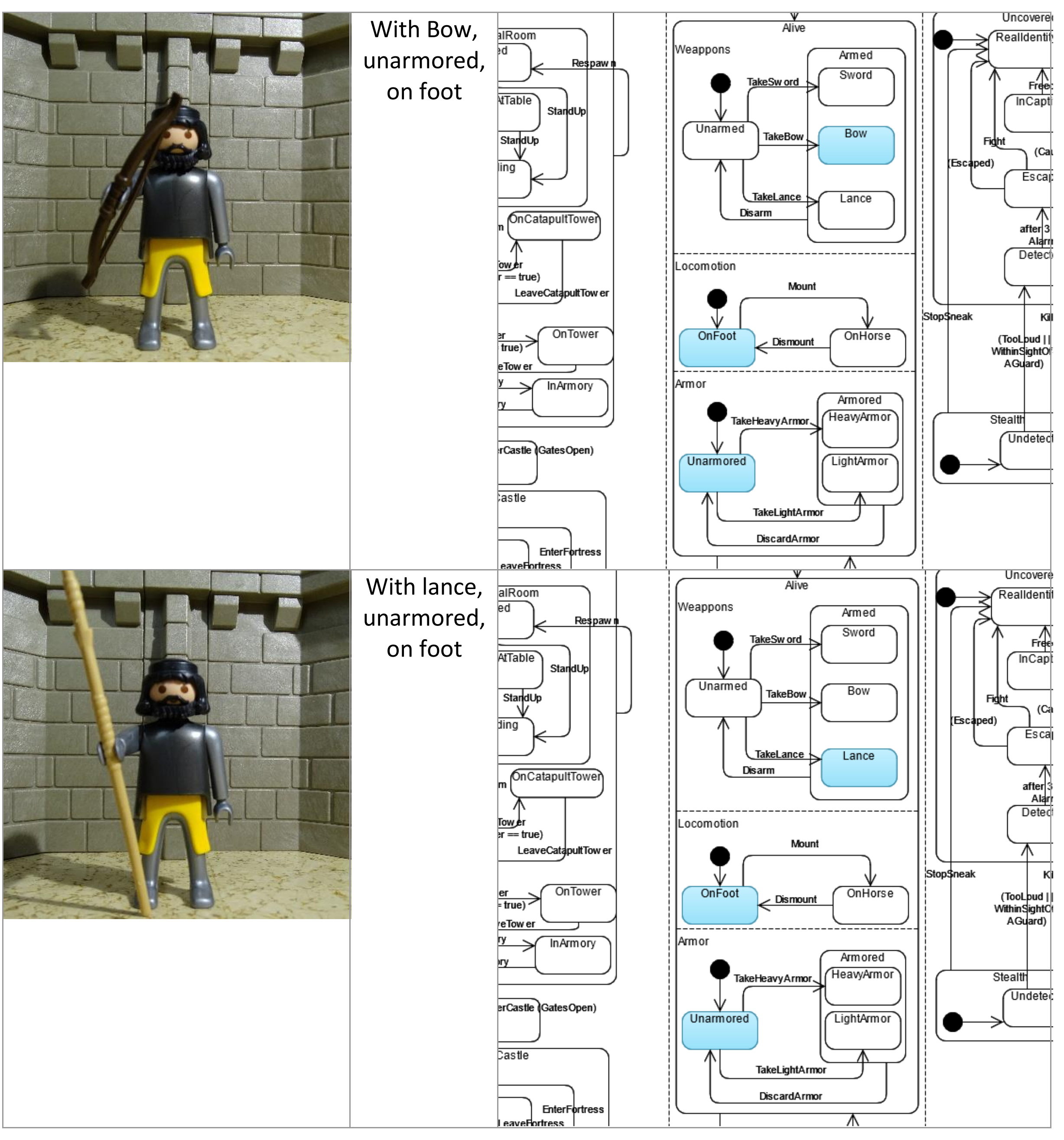
[33] S. Yadav, S. Sikka, and U. Shrivastava, "A review of object-oriented coupling and cohesion metrics," *International Journal of Computer Science Trends and Technology*, vol. 2, no. 5, pp. 45–55, 2014.

Appendix

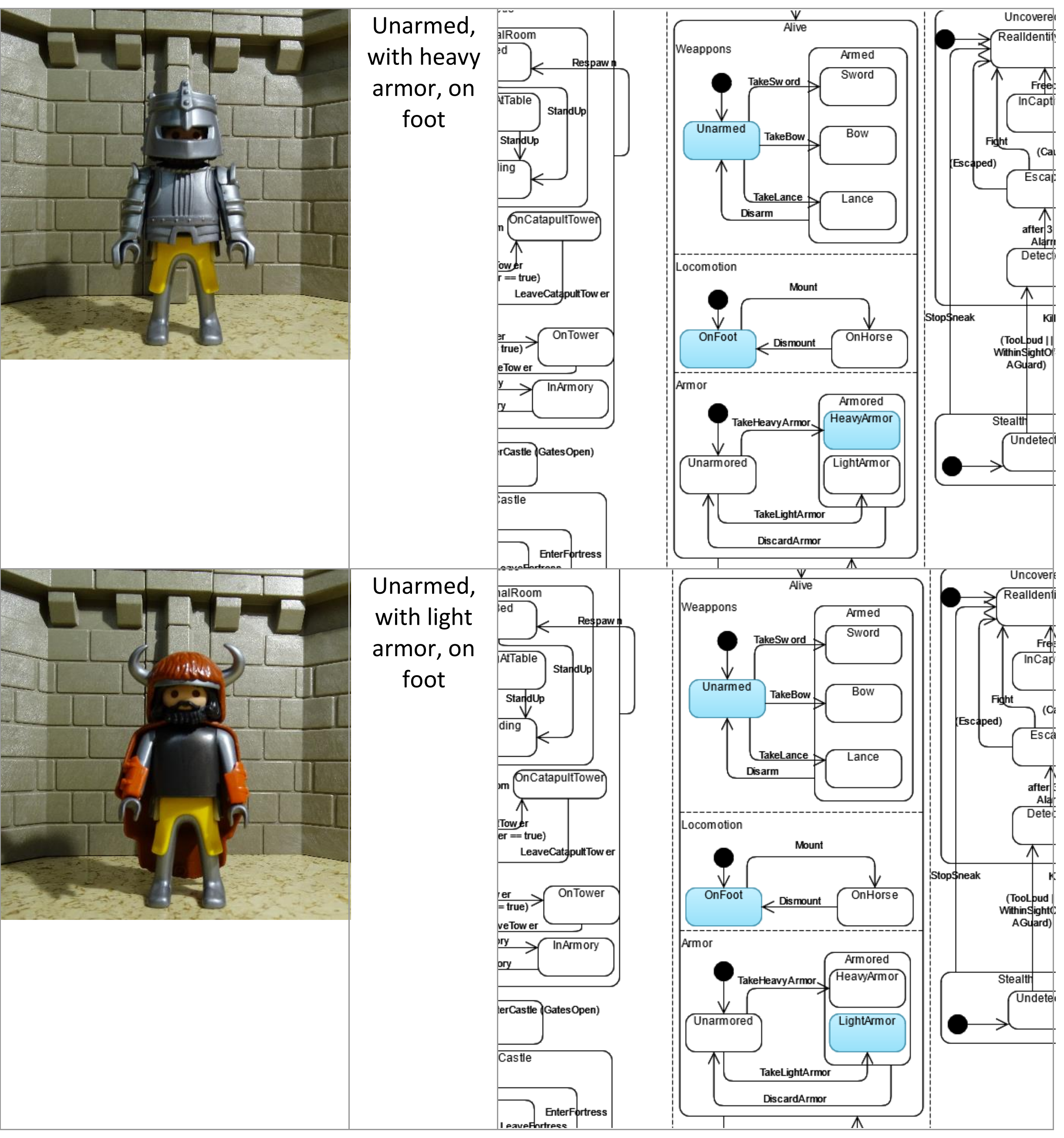
1 Base States of the Knight



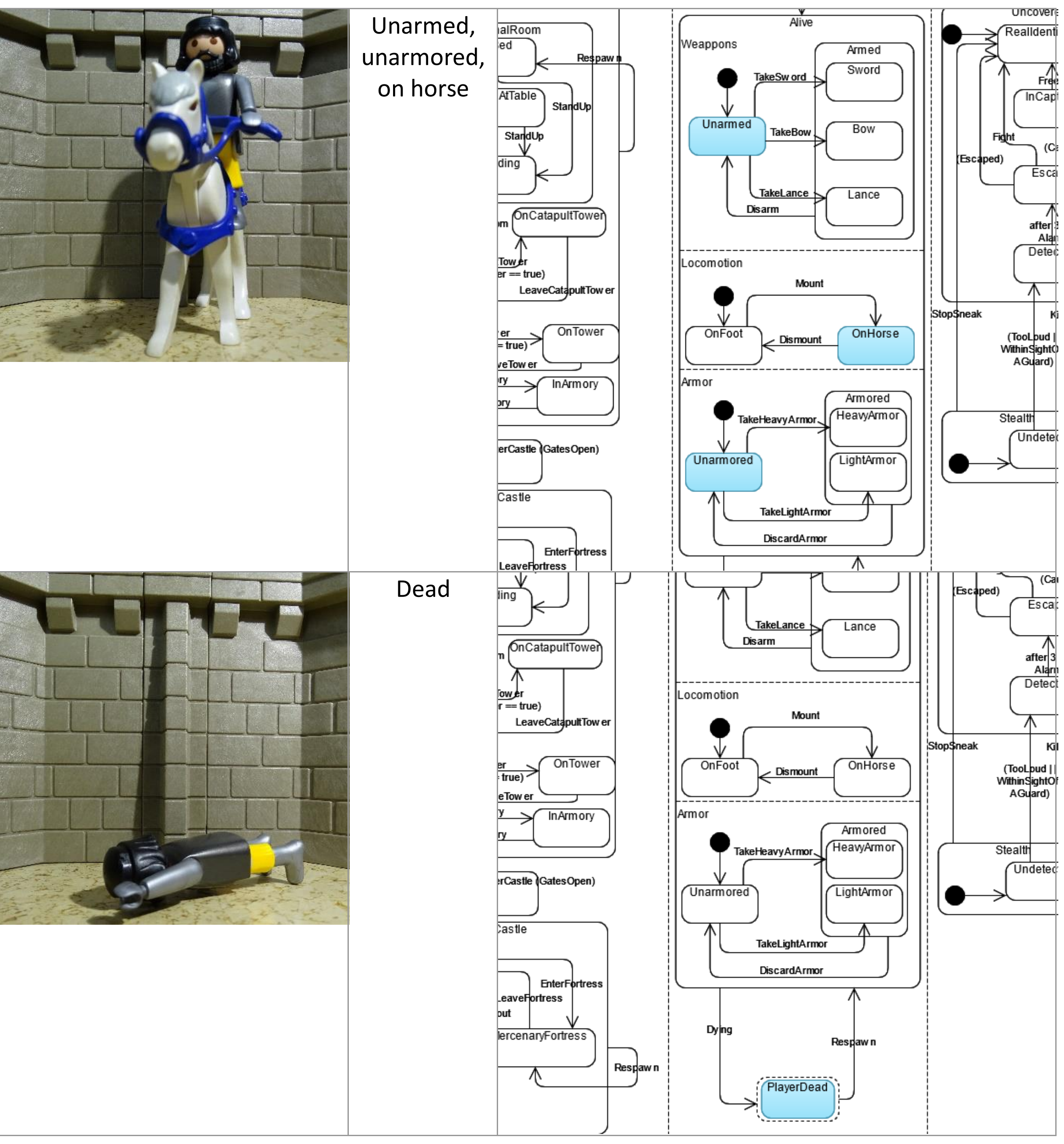
Paper-Prototype	Description	Statechart
	Unarmed, unarmored, on foot	
	With sword, unarmord, on foot	



With Bow, unarmored, on foot
Alive
Weappons
Armed
Sword
TakeSword
Unarmed
TakeBow
Bow
TakeLance
Lance
Disarm
Locomotion
Mount
OnFoot
Dismount
OnHorse
Armor
Armored
TakeHeavyArmor
HeavyArmor
LightArmor
Unarmored
TakeLightArmor
DiscardArmor
OnCatapultTower
LeaveCatapultTower
OnTower
InArmory
(GatesOpen)
EnterFortress
Respawn
StandUp
(Escaped)
Fight
StopSneak
(TooLoud ||
AGuard)
Stealth
With lance, unarmored, on foot



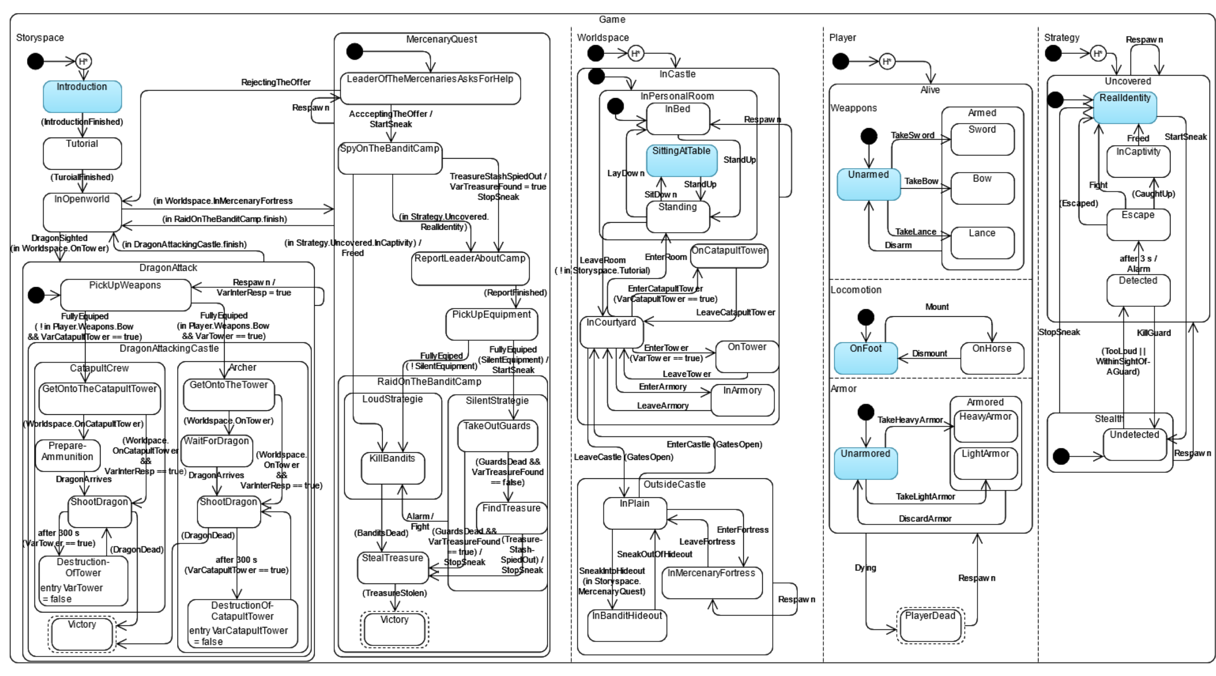
Unarmed, with heavy armor, on foot
Alive
Weappons
Armed
Sword
Bow
Lance
Unarmed
TakeSword
TakeBow
TakeLance
Disarm
Locomotion
Mount
OnFoot
Dismount
OnHorse
Armor
Armored
HeavyArmor
LightArmor
Unarmored
TakeHeavyArmor
TakeLightArmor
DiscardArmor
OnCatapultTower
LeaveCatapultTower
OnTower
InArmory
StandUp
Respawn
(GatesOpen)
EnterFortress
StopSneak
Stealth
Fight
(Escaped)
Unarmed, with light armor, on foot

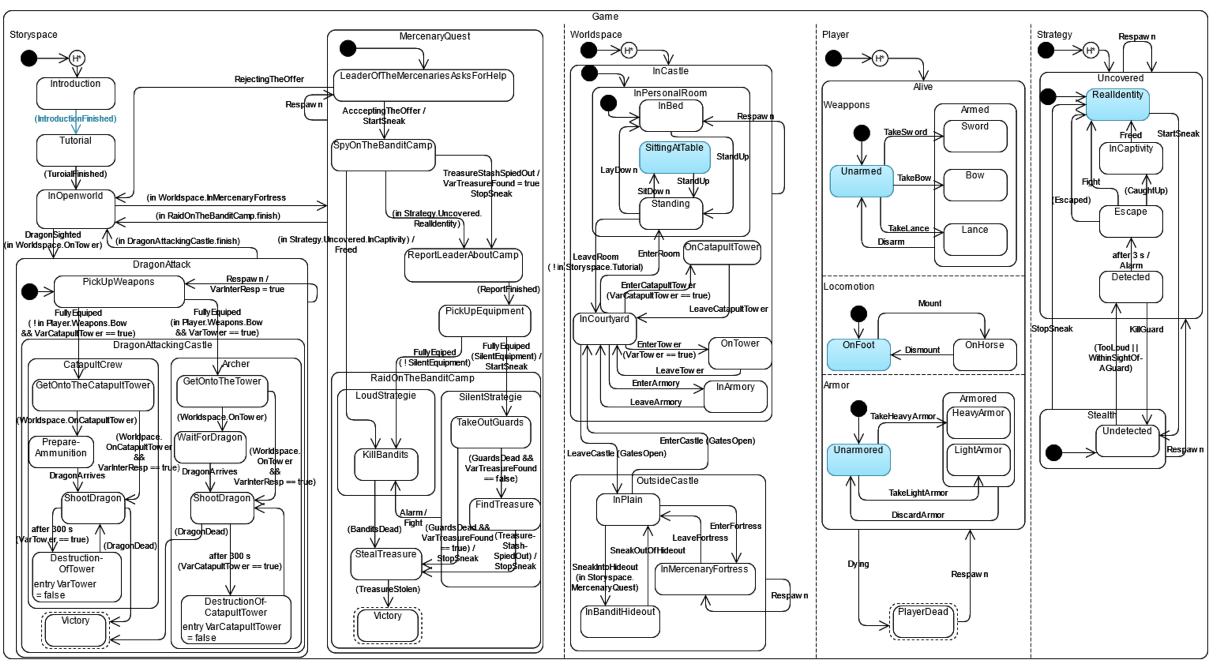


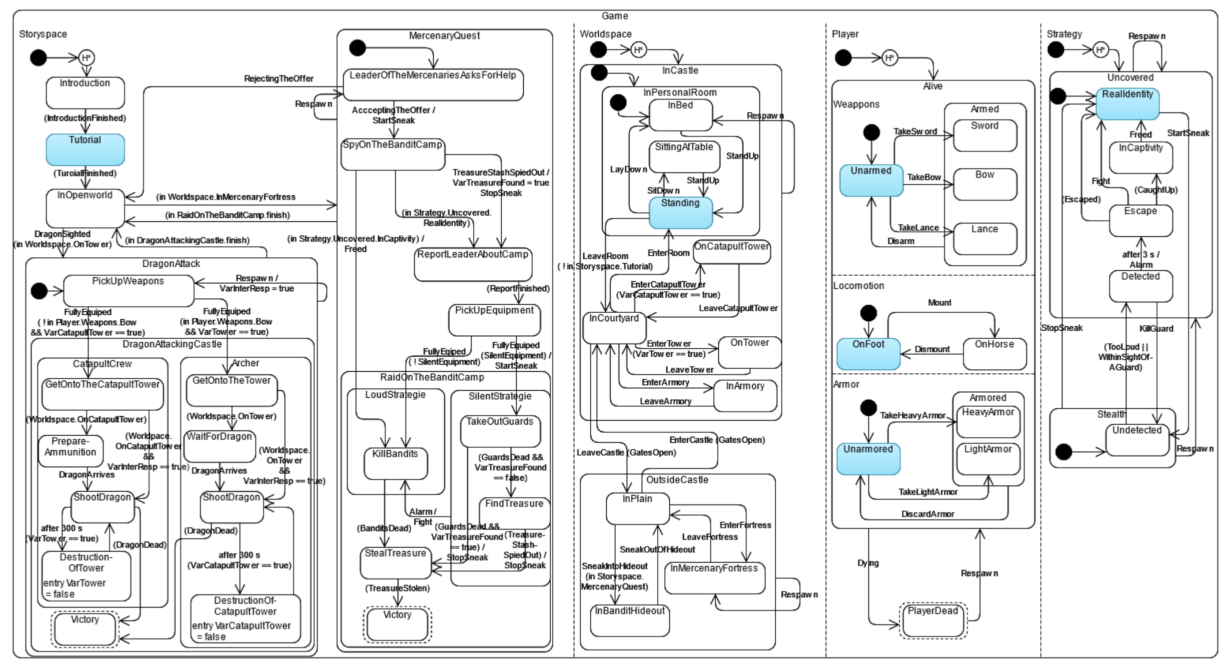
Unarmed, unarmored, on horse
Dead
Alive
Weappons
Armed
TakeSword
Sword
Unarmed
TakeBow
Bow
TakeLance
Lance
Disarm
Locomotion
Mount
OnFoot
Dismount
OnHorse
Armor
Armored
TakeHeavyArmor
HeavyArmor
Unarmored
LightArmor
TakeLightArmor
DiscardArmor
Dying
Respawn
PlayerDead
OnCatapultTower
LeaveCatapultTower
OnTower
InArmory
(GatesOpen)
EnterFortress
LeaveFortress
StandUp
Fight
(Escaped)
StopSneak
Stealth

2 Paper-Prototype

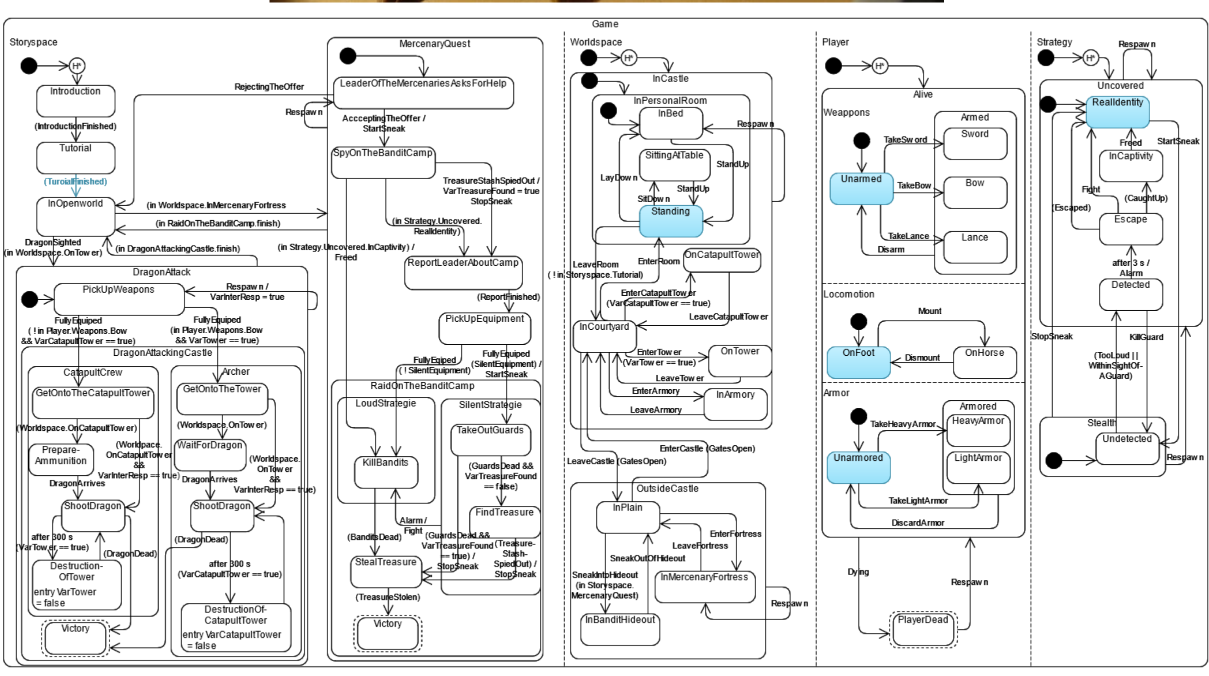
Storyline – Interrupt:



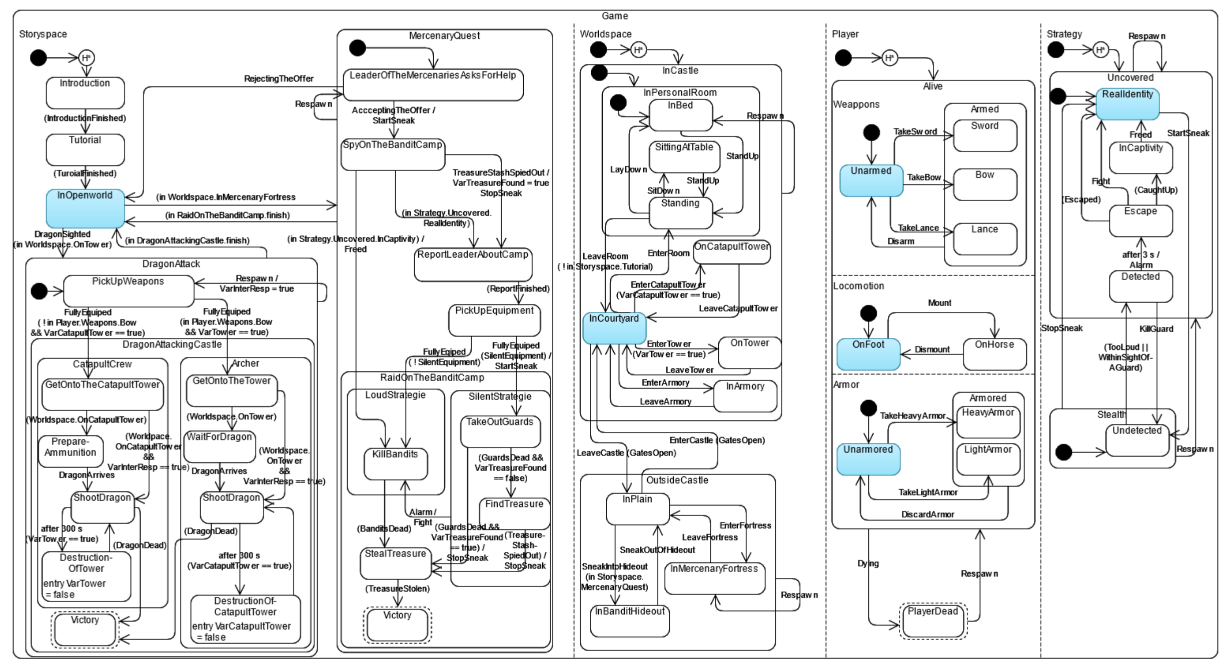


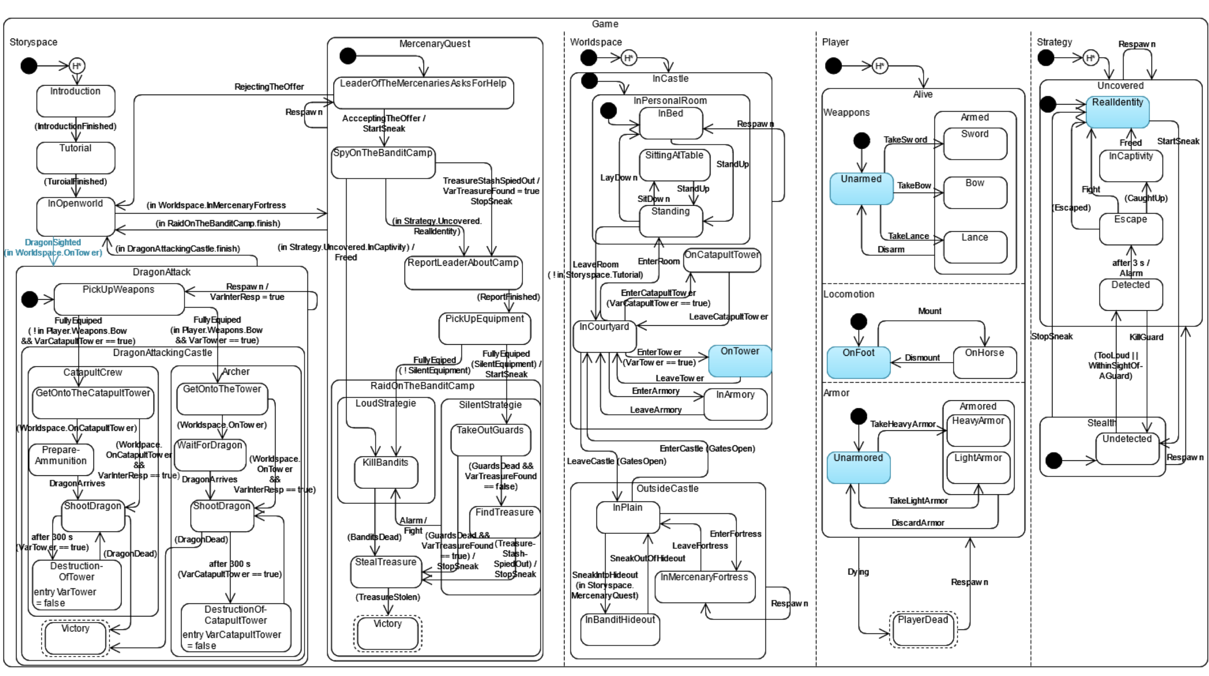


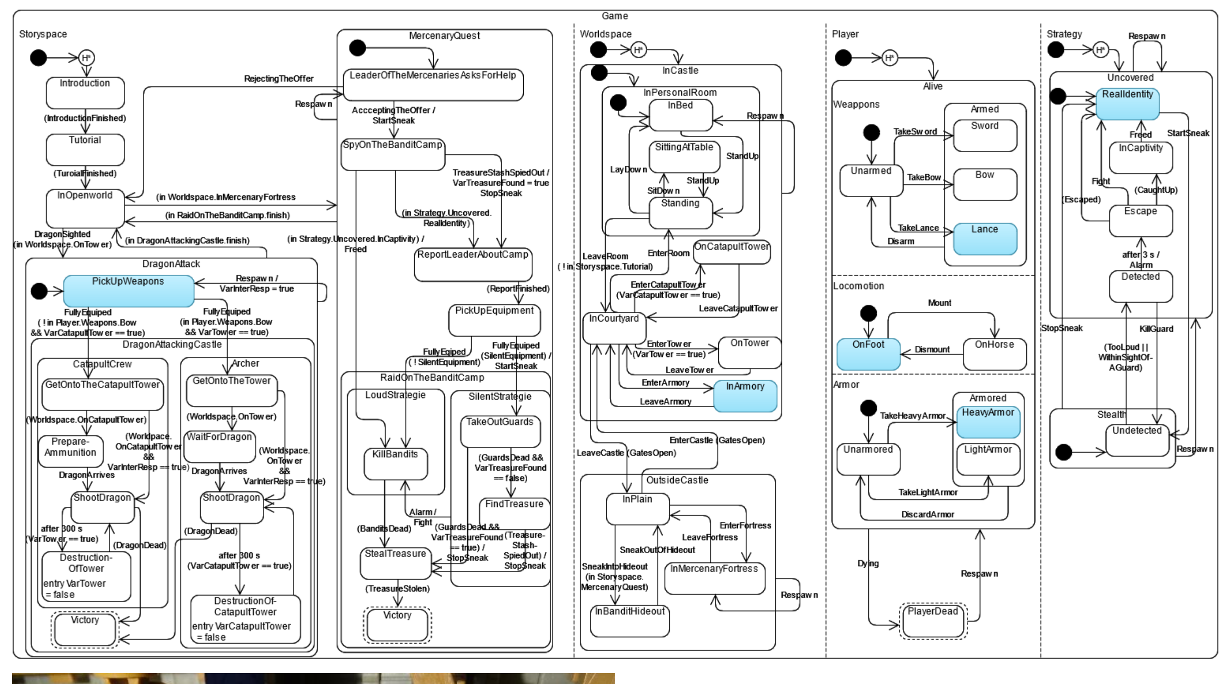
Game
Storyspace
Introduction
(IntroductionFinished)
Tutorial
(TuroialFinished)
InOpenworld
RejectingTheOffer
Respawn
(in Worldspace.InMercenaryFortress)
(in RaidOnTheBanditCamp.finish)
(in DragonAttackingCastle.finish)
DragonSighted
(in Worldspace.OnTower)
(in Strategy.Uncovered.InCaptivity) /
Freed
DragonAttack
PickUpWeapons
Respawn /
VarInterResp = true
FullyEquiped
(! in Player.Weapons.Bow
&& VarCatapultTower == true)
FullyEquiped
(in Player.Weapons.Bow
&& VarTower == true)
DragonAttackingCastle
CatapultCrew
GetOntoTheCatapultTower
(Worldspace.OnCatapultTower)
Prepare-
Ammunition
(Worldspace.
OnCatapultTower
&&
VarInterResp == true)
DragonArrives
ShootDragon
after 300 s
(VarTower == true)
(DragonDead)
Destruction-
OfTower
entry VarTower
= false
Victory
Archer
GetOntoTheTower
(Worldspace.OnTower)
WaitForDragon
(Worldspace.
OnTower
&&
VarInterResp == true)
DragonArrives
ShootDragon
(DragonDead)
after 300 s
(VarCatapultTower == true)
Destruction-Of-
CatapultTower
entry VarCatapultTower
= false
MercenaryQuest
LeaderOfTheMercenariesAsksForHelp
Respawn
Acccepting TheOffer /
StartSneak
SpyOnTheBanditCamp
TreasureStashSpiedOut /
VarTreasureFound = true
StopSneak
(in Strategy.Uncovered.
RealIdentity)
ReportLeaderAboutCamp
(ReportFinished)
PickUpEquipment
FullyEqiped
(! SilentEquipment)
FullyEquiped
(SilentEquipment) /
StartSneak
RaidOnTheBanditCamp
LoudStrategie
KillBandits
SilentStrategie
TakeOutGuards
(GuardsDead &&
VarTreasureFound
== false)
FindTreasure
Alarm /
Fight
(BanditsDead)
(GuardsDead &&
VarTreasureFound
== true) /
StopSneak
(Treasure-
Stash-
SpiedOut) /
StopSneak
StealTreasure
(TreasureStolen)
Victory
Worldspace
InCastle
InPersonalRoom
InBed
Respawn
SittingAtTable
StandUp
LayDown
SitDown
StandUp
Standing
LeaveRoom
(! in Storyspace.Tutorial)
EnterRoom
OnCatapultTower
EnterCatapultTower
(VarCatapultTower == true)
LeaveCatapultTower
InCourtyard
EnterTower
(VarTower == true)
OnTower
LeaveTower
EnterArmory
InArmory
LeaveArmory
EnterCastle (GatesOpen)
LeaveCastle (GatesOpen)
OutsideCastle
InPlain
EnterFortress
LeaveFortress
SneakOutOfHideout
SneakIntoHideout
(in Storyspace.
MercenaryQuest)
InMercenaryFortress
Respawn
InBanditHideout
Player
Alive
Weappons
Armed
TakeSword
Sword
Unarmed
TakeBow
Bow
TakeLance
Lance
Disarm
Locomotion
Mount
OnFoot
Dismount
OnHorse
Armor
Armored
TakeHeavyArmor
HeavyArmor
Unarmored
LightArmor
TakeLightArmor
DiscardArmor
Dying
Respawn
PlayerDead
Strategy
Respawn
Uncovered
RealIdentity
Freed
StartSneak
InCaptivity
Fight
(Escaped)
(CaughtUp)
Escape
after 3 s /
Alarm
Detected
StopSneak
KillGuard
(ToLoud | |
WithinSightOf-
AGuard)
Stealth
Undetected
Respawn



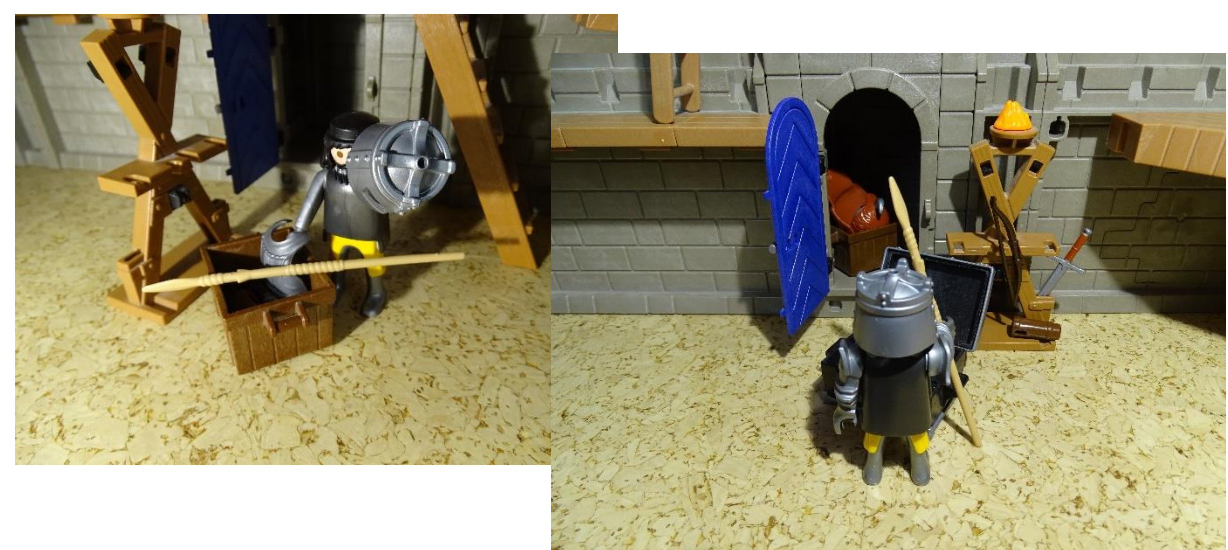
Game
Storyspace
Introduction
(IntroductionFinished)
Tutorial
(TuroialFinished)
InOpenworld
RejectingTheOffer
Respawn
(in Worldspace.InMercenaryFortress)
(in RaidOnTheBanditCamp.finish)
(in DragonAttackingCastle.finish)
DragonSighted
(in Worldspace.OnTower)
(in Strategy.Uncovered.InCaptivity) /
Freed
DragonAttack
PickUpWeapons
Respawn /
VarInterResp = true
FullyEquiped
(! in Player.Weapons.Bow
&& VarCatapultTower == true)
FullyEquiped
(in Player.Weapons.Bow
&& VarTower == true)
DragonAttackingCastle
CatapultCrew
GetOntoTheCatapultTower
(Worldspace.OnCatapultTower)
Prepare-
Ammunition
(Worldspace.
OnCatapultTower
&&
VarInterResp == true)
DragonArrives
ShootDragon
after 300 s
(VarTower == true)
(DragonDead)
Destruction-
OfTower
entry VarTower
= false
Victory
Archer
GetOntoTheTower
(Worldspace.OnTower)
WaitForDragon
(Worldspace.
OnTower
&&
VarInterResp == true)
DragonArrives
ShootDragon
(DragonDead)
after 300 s
(VarCatapultTower == true)
Destruction-Of-
CatapultTower
entry VarCatapultTower
= false
MercenaryQuest
LeaderOfTheMercenariesAsksForHelp
Respawn
Acccepting TheOffer /
StartSneak
SpyOnTheBanditCamp
TreasureStashSpiedOut /
VarTreasureFound = true
StopSneak
(in Strategy.Uncovered.
RealIdentity)
ReportLeaderAboutCamp
(ReportFinished)
PickUpEquipment
FullyEqiped
(! SilentEquipment)
FullyEquiped
(SilentEquipment) /
StartSneak
RaidOnTheBanditCamp
LoudStrategie
KillBandits
SilentStrategie
TakeOutGuards
(GuardsDead &&
VarTreasureFound
== false)
FindTreasure
Alarm /
Fight
(BanditsDead)
(GuardsDead &&
VarTreasureFound
== true) /
StopSneak
(Treasure-
Stash-
SpiedOut) /
StopSneak
StealTreasure
(TreasureStolen)
Victory
Worldspace
InCastle
InPersonalRoom
InBed
Respawn
SittingAtTable
StandUp
LayDown
SitDown
StandUp
Standing
LeaveRoom
(! in Storyspace.Tutorial)
EnterRoom
OnCatapultTower
EnterCatapultTower
(VarCatapultTower == true)
LeaveCatapultTower
InCourtyard
EnterTower
(VarTower == true)
OnTower
LeaveTower
EnterArmory
InArmory
LeaveArmory
EnterCastle (GatesOpen)
LeaveCastle (GatesOpen)
OutsideCastle
InPlain
EnterFortress
LeaveFortress
SneakOutOfHideout
SneakIntoHideout
(in Storyspace.
MercenaryQuest)
InMercenaryFortress
Respawn
InBanditHideout
Player
Alive
Weappons
Armed
TakeSword
Sword
Unarmed
TakeBow
Bow
TakeLance
Lance
Disarm
Locomotion
Mount
OnFoot
Dismount
OnHorse
Armor
Armored
TakeHeavyArmor
HeavyArmor
Unarmored
LightArmor
TakeLightArmor
DiscardArmor
Dying
Respawn
PlayerDead
Strategy
Respawn
Uncovered
RealIdentity
Freed
StartSneak
InCaptivity
Fight
(Escaped)
(CaughtUp)
Escape
after 3 s /
Alarm
Detected
StopSneak
KillGuard
(ToLoud | |
WithinSightOf-
AGuard)
Stealth
Undetected
Respawn

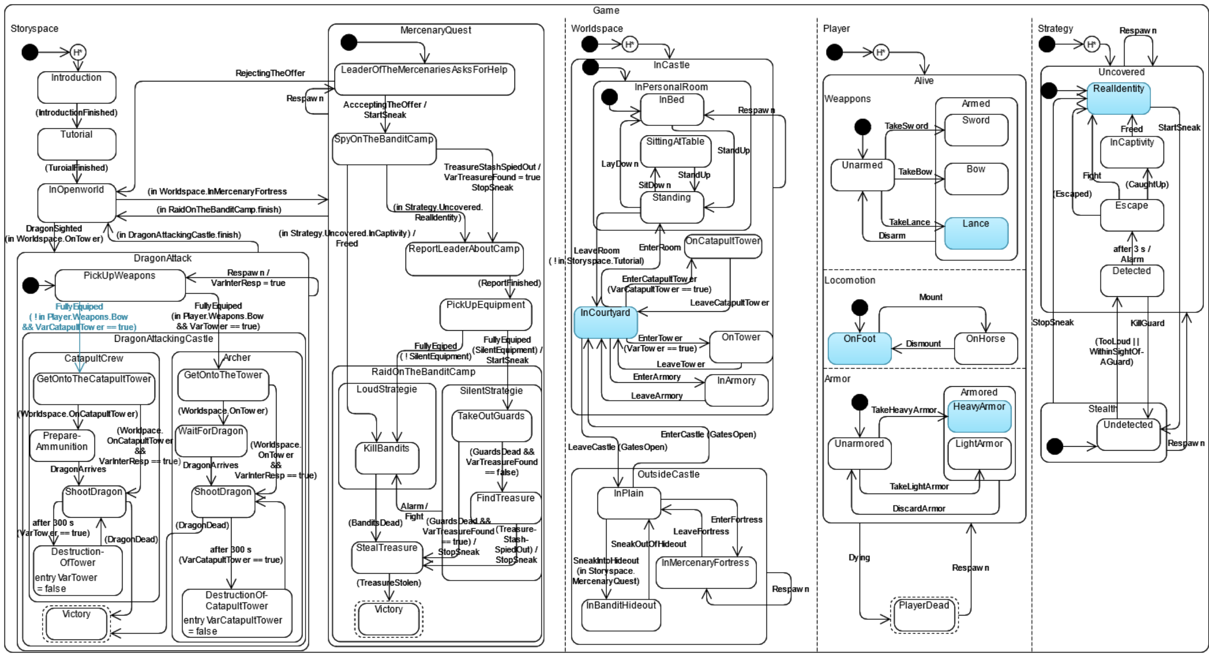

Game
Storyspace
Introduction
(IntroductionFinished)
Tutorial
(TuroialFinished)
InOpenworld
RejectingTheOffer
Respawn
(in Worldspace.InMercenaryFortress)
(in RaidOnTheBanditCamp.finish)
DragonSighted
(in Worldspace.OnTower)
(in DragonAttackingCastle.finish)
DragonAttack
PickUpWeapons
Respawn / VarInterResp = true
FullyEquiped (! in Player.Weapons.Bow && VarCatapultTower == true)
FullyEquiped (in Player.Weapons.Bow && VarTower == true)
DragonAttackingCastle
CatapultCrew
Archer
GetOntoTheCatapultTower
GetOntoTheTower
(Worldspace.OnCatapultTower)
(Worldspace.OnTower)
Prepare-Ammunition
WaitForDragon
(Worldspace.OnCatapultTower && VarInterResp == true)
(Worldspace.OnTower && VarInterResp == true)
DragonArrives
ShootDragon
after 300 s (VarTower == true)
(DragonDead)
Destruction-OfTower
entry VarTower = false
after 300 s (VarCatapultTower == true)
DestructionOf-CatapultTower
entry VarCatapultTower = false
Victory
MercenaryQuest
LeaderOfTheMercenariesAsksForHelp
AccceptingTheOffer / StartSneak
SpyOnTheBanditCamp
TreasureStashSpiedOut / VarTreasureFound = true StopSneak
(in Strategy.Uncovered.RealIdentity)
(in Strategy.Uncovered.InCaptivity) / Freed
ReportLeaderAboutCamp
(ReportFinished)
PickUpEquipment
FullyEqiped (! SilentEquipment)
FullyEquiped (SilentEquipment) / StartSneak
RaidOnTheBanditCamp
LoudStrategie
SilentStrategie
TakeOutGuards
KillBandits
(GuardsDead && VarTreasureFound == false)
FindTreasure
Alarm / Fight
(BanditsDead)
(GuardsDead && VarTreasureFound == true) / StopSneak
(Treasure-Stash-SpiedOut) / StopSneak
StealTreasure
(TreasureStolen)
Victory
Worldspace
InCastle
InPersonalRoom
InBed
SittingAtTable
StandUp
LayDown
SitDown
StandUp
Standing
Respawn
LeaveRoom (! in Storyspace.Tutorial)
EnterRoom
OnCatapultTower
EnterCatapultTower (VarCatapultTower == true)
LeaveCatapultTower
InCourtyard
EnterTower (VarTower == true)
OnTower
LeaveTower
EnterArmory
InArmory
LeaveArmory
EnterCastle (GatesOpen)
LeaveCastle (GatesOpen)
OutsideCastle
InPlain
EnterFortress
LeaveFortress
SneakOutOfHideout
SneakIntoHideout (in Storyspace.MercenaryQuest)
InMercenaryFortress
Respawn
InBanditHideout
Player
Alive
Weappons
Armed
TakeSword
Sword
Unarmed
TakeBow
Bow
TakeLance
Disarm
Lance
Locomotion
Mount
OnFoot
Dismount
OnHorse
Armor
Armored
TakeHeavyArmor
HeavyArmor
Unarmored
LightArmor
TakeLightArmor
DiscardArmor
Dying
Respawn
PlayerDead
Strategy
Respawn
Uncovered
RealIdentity
Freed
StartSneak
InCaptivity
Fight
(Escaped)
(CaughtUp)
Escape
after 3 s / Alarm
Detected
StopSneak
KillGuard
(TooLoud || WithinSightOf-AGuard)
Stealth
Undetected
Respawn


Game
Storyspace
Introduction
(IntroductionFinished)
Tutorial
(TuroialFinished)
InOpenworld
RejectingTheOffer
Respawn
(in Worldspace.InMercenaryFortress)
(in RaidOnTheBanditCamp.finish)
DragonSighted
(in Worldspace.OnTower)
(in DragonAttackingCastle.finish)
DragonAttack
PickUpWeapons
Respawn / VarInterResp = true
FullyEquiped (! in Player.Weapons.Bow && VarCatapultTower == true)
FullyEquiped (in Player.Weapons.Bow && VarTower == true)
DragonAttackingCastle
CatapultCrew
Archer
GetOntoTheCatapultTower
GetOntoTheTower
(Worldspace.OnCatapultTower)
(Worldspace.OnTower)
Prepare-Ammunition
WaitForDragon
(Worldspace.OnCatapultTower && VarInterResp == true)
(Worldspace.OnTower && VarInterResp == true)
DragonArrives
ShootDragon
after 300 s (VarTower == true)
(DragonDead)
Destruction-OfTower
entry VarTower = false
after 300 s (VarCatapultTower == true)
DestructionOf-CatapultTower
entry VarCatapultTower = false
Victory
MercenaryQuest
LeaderOfTheMercenariesAsksForHelp
AccceptingTheOffer / StartSneak
SpyOnTheBanditCamp
TreasureStashSpiedOut / VarTreasureFound = true StopSneak
(in Strategy.Uncovered.RealIdentity)
(in Strategy.Uncovered.InCaptivity) / Freed
ReportLeaderAboutCamp
(ReportFinished)
PickUpEquipment
FullyEqiped (! SilentEquipment)
FullyEquiped (SilentEquipment) / StartSneak
RaidOnTheBanditCamp
LoudStrategie
SilentStrategie
TakeOutGuards
KillBandits
(GuardsDead && VarTreasureFound == false)
FindTreasure
Alarm / Fight
(BanditsDead)
(GuardsDead && VarTreasureFound == true) / StopSneak
(Treasure-Stash-SpiedOut) / StopSneak
StealTreasure
(TreasureStolen)
Victory
Worldspace
InCastle
InPersonalRoom
InBed
SittingAtTable
StandUp
LayDown
SitDown
StandUp
Standing
Respawn
LeaveRoom (! in Storyspace.Tutorial)
EnterRoom
OnCatapultTower
EnterCatapultTower (VarCatapultTower == true)
LeaveCatapultTower
InCourtyard
EnterTower (VarTower == true)
OnTower
LeaveTower
EnterArmory
InArmory
LeaveArmory
EnterCastle (GatesOpen)
LeaveCastle (GatesOpen)
OutsideCastle
InPlain
EnterFortress
LeaveFortress
SneakOutOfHideout
SneakIntoHideout (in Storyspace.MercenaryQuest)
InMercenaryFortress
Respawn
InBanditHideout
Player
Alive
Weappons
Armed
TakeSword
Sword
Unarmed
TakeBow
Bow
TakeLance
Disarm
Lance
Locomotion
Mount
OnFoot
Dismount
OnHorse
Armor
Armored
TakeHeavyArmor
HeavyArmor
Unarmored
LightArmor
TakeLightArmor
DiscardArmor
Dying
Respawn
PlayerDead
Strategy
Respawn
Uncovered
RealIdentity
Freed
StartSneak
InCaptivity
Fight
(Escaped)
(CaughtUp)
Escape
after 3 s / Alarm
Detected
StopSneak
KillGuard
(TooLoud || WithinSightOf-AGuard)
Stealth
Undetected
Respawn

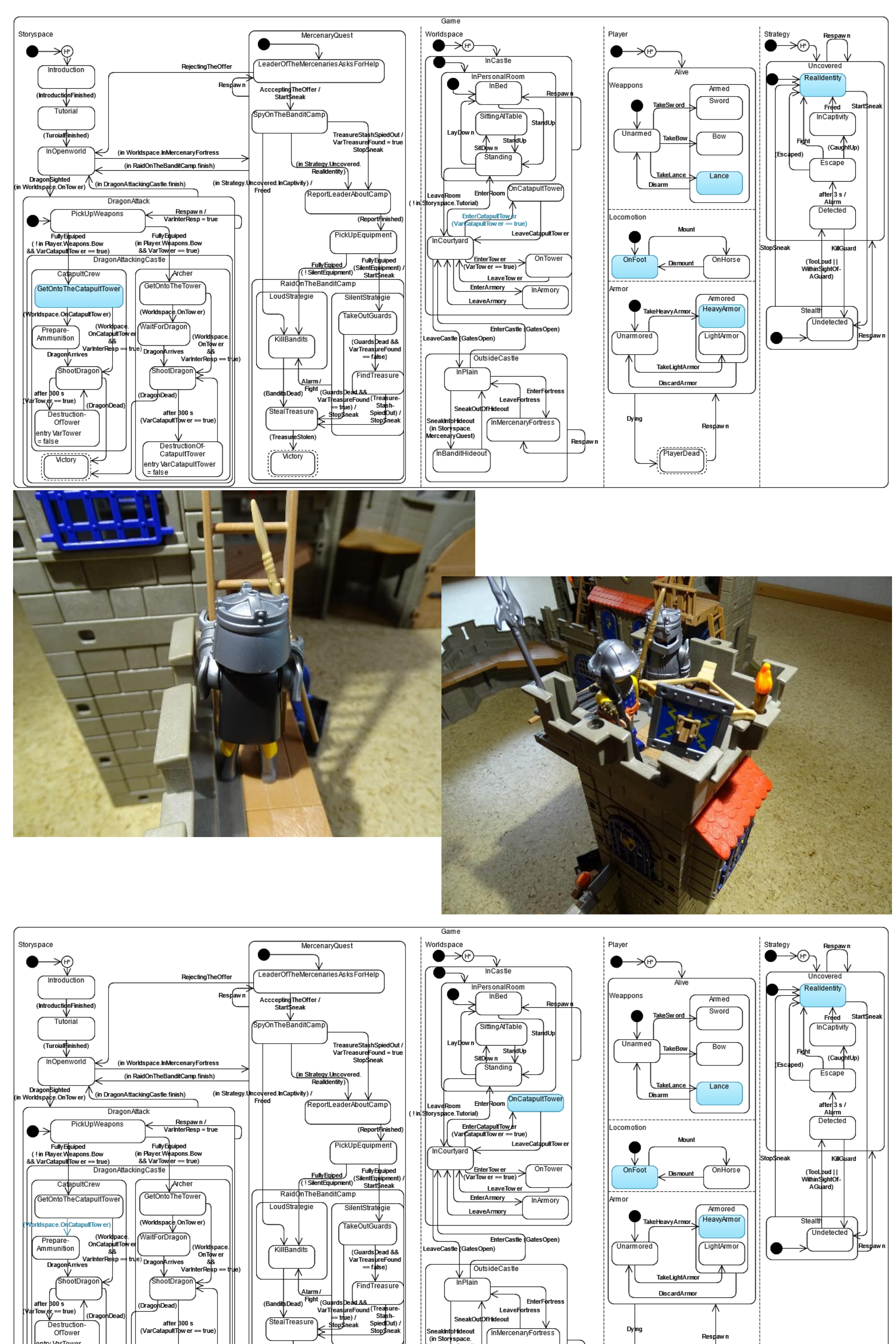


Game
Storyspace
Introduction
(IntroductionFinished)
Tutorial
(TuroialFinished)
InOpenworld
RejectingTheOffer
Respawn
(in Worldspace.InMercenaryFortress
(in RaidOnTheBanditCamp.finish)
DragonSighted
(in Worldspace.OnTower)
(in DragonAttackingCastle.finish)
(in Strategy.Uncovered.InCaptivity) /
Freed
DragonAttack
PickUpWeapons
Respawn /
VarInterResp = true
FullyEquiped
(! in Player.Weapons.Bow
&& VarCatapultTower == true)
FullyEquiped
(in Player.Weapons.Bow
&& VarTower == true)
DragonAttackingCastle
CatapultCrew
GetOntoTheCatapultTower
(Worldspace.OnCatapultTower)
Prepare-
Ammunition
(Worldspace.
OnCatapultTower
&&
VarInterResp == true)
DragonArrives
ShootDragon
after 300 s
(VarTower == true)
(DragonDead)
Destruction-
OfTower
entry VarTower
= false
Victory
Archer
GetOntoTheTower
(Worldspace.OnTower)
WaitForDragon
(Worldspace.
OnTower
&&
VarInterResp == true)
DragonArrives
ShootDragon
(DragonDead)
after 300 s
(VarCatapultTower == true)
DestructionOf-
CatapultTower
entry VarCatapultTower
= false
MercenaryQuest
LeaderOfTheMercenariesAsksForHelp
Respawn
AcceptingTheOffer /
StartSneak
SpyOnTheBanditCamp
TreasureStashSpiedOut /
VarTreasureFound = true
StopSneak
(in Strategy.Uncovered.
RealIdentity)
ReportLeaderAboutCamp
(ReportFinished)
PickUpEquipment
FullyEquiped
(! SilentEquipment)
FullyEquiped
(SilentEquipment) /
StartSneak
RaidOnTheBanditCamp
LoudStrategie
SilentStrategie
TakeOutGuards
KillBandits
(GuardsDead &&
VarTreasureFound
== false)
FindTreasure
Alarm /
Fight
(GuardsDead &&
VarTreasureFound
== true) /
StopSneak
(Treasure-
Stash-
SpiedOut) /
StopSneak
(BanditsDead)
StealTreasure
(TreasureStolen)
Victory
Worldspace
InCastle
InPersonalRoom
InBed
Respawn
SittingAtTable
StandUp
LayDown
SitDown
StandUp
Standing
LeaveRoom
(! in Storyspace.Tutorial)
EnterRoom
OnCatapultTower
EnterCatapultTower
(VarCatapultTower == true)
LeaveCatapultTower
InCourtyard
EnterTower
(VarTower == true)
OnTower
LeaveTower
EnterArmory
InArmory
LeaveArmory
EnterCastle (GatesOpen)
LeaveCastle (GatesOpen)
OutsideCastle
InPlain
EnterFortress
LeaveFortress
SneakOutOfHideout
SneakIntoHideout
(in Storyspace.
MercenaryQuest)
InMercenaryFortress
Respawn
InBanditHideout
Player
Alive
Weappons
Armed
TakeSword
Sword
Unarmed
TakeBow
Bow
TakeLance
Lance
Disarm
Locomotion
Mount
OnFoot
Dismount
OnHorse
Armor
Armored
TakeHeavyArmor
HeavyArmor
Unarmored
LightArmor
TakeLightArmor
DiscardArmor
Dying
Respawn
PlayerDead
Strategy
Respawn
Uncovered
RealIdentity
Freed
StartSneak
InCaptivity
Fight
(CaughtUp)
(Escaped)
Escape
after 3 s /
Alarm
Detected
StopSneak
KillGuard
(TooLoud | |
WithinSightOf-
AGuard)
Stealth
Undetected
Respawn

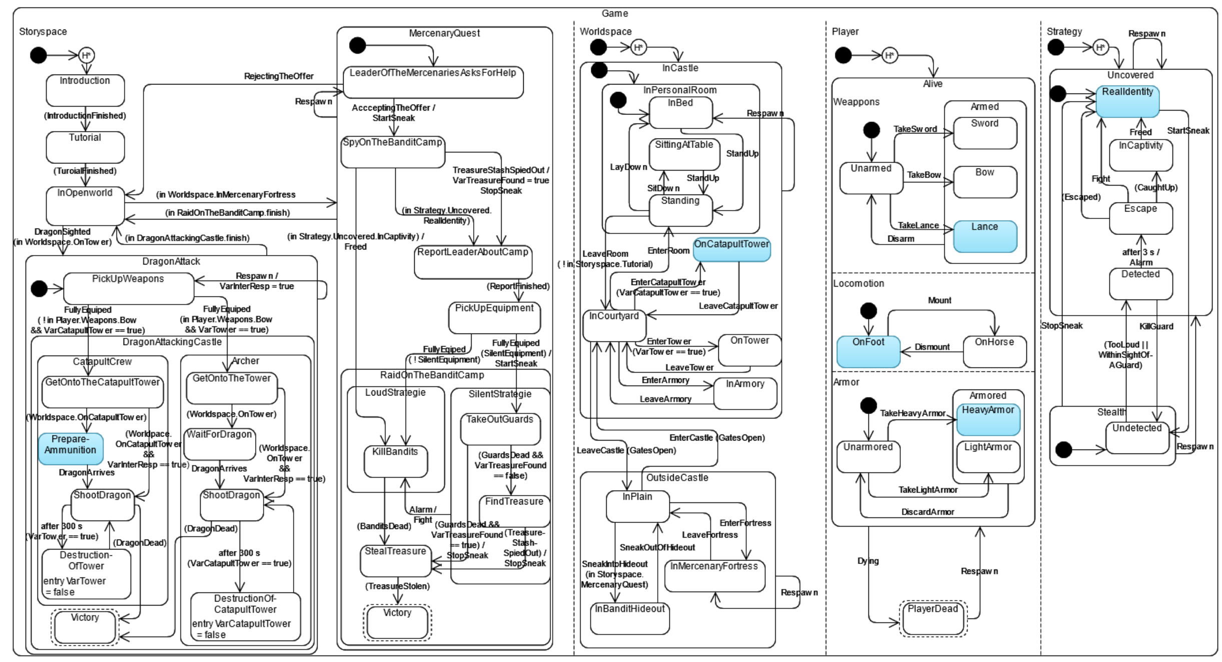




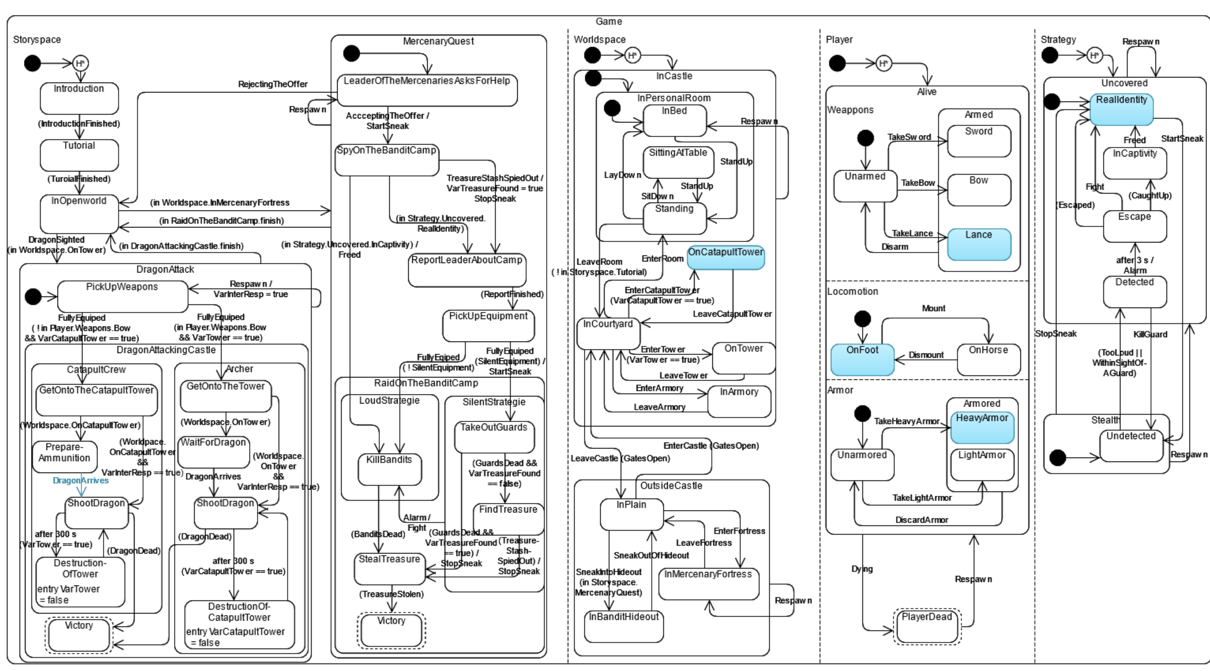
Game
Storyspace
Introduction
(IntroductionFinished)
Tutorial
(TuroialFinished)
InOpenworld
RejectingTheOffer
Respawn
(in Worldspace.InMercenaryFortress
(in RaidOnTheBanditCamp.finish)
DragonSighted
(in Worldspace.OnTower)
(in DragonAttackingCastle.finish)
(in Strategy.Uncovered.InCaptivity) /
Freed
DragonAttack
PickUpWeapons
Respawn /
VarInterResp = true
FullyEquiped
(! in Player.Weapons.Bow
&& VarCatapultTower == true)
FullyEquiped
(in Player.Weapons.Bow
&& VarTower == true)
DragonAttackingCastle
CatapultCrew
GetOntoTheCatapultTower
(Worldspace.OnCatapultTower)
Prepare-
Ammunition
(Worldspace.
OnCatapultTower
&&
VarInterResp == true)
DragonArrives
ShootDragon
after 300 s
(VarTower == true)
(DragonDead)
Destruction-
OfTower
entry VarTower
= false
Victory
Archer
GetOntoTheTower
(Worldspace.OnTower)
WaitForDragon
(Worldspace.
OnTower
&&
VarInterResp == true)
DragonArrives
ShootDragon
(DragonDead)
after 300 s
(VarCatapultTower == true)
DestructionOf-
CatapultTower
entry VarCatapultTower
= false
MercenaryQuest
LeaderOfTheMercenariesAsksForHelp
Respawn
AcceptingTheOffer /
StartSneak
SpyOnTheBanditCamp
TreasureStashSpiedOut /
VarTreasureFound = true
StopSneak
(in Strategy.Uncovered.
RealIdentity)
ReportLeaderAboutCamp
(ReportFinished)
PickUpEquipment
FullyEquiped
(! SilentEquipment)
FullyEquiped
(SilentEquipment) /
StartSneak
RaidOnTheBanditCamp
LoudStrategie
SilentStrategie
TakeOutGuards
KillBandits
(GuardsDead &&
VarTreasureFound
== false)
FindTreasure
Alarm /
Fight
(GuardsDead &&
VarTreasureFound
== true) /
StopSneak
(Treasure-
Stash-
SpiedOut) /
StopSneak
(BanditsDead)
StealTreasure
(TreasureStolen)
Victory
Worldspace
InCastle
InPersonalRoom
InBed
Respawn
SittingAtTable
StandUp
LayDown
SitDown
StandUp
Standing
LeaveRoom
(! in Storyspace.Tutorial)
EnterRoom
OnCatapultTower
EnterCatapultTower
(VarCatapultTower == true)
LeaveCatapultTower
InCourtyard
EnterTower
(VarTower == true)
OnTower
LeaveTower
EnterArmory
InArmory
LeaveArmory
EnterCastle (GatesOpen)
LeaveCastle (GatesOpen)
OutsideCastle
InPlain
EnterFortress
LeaveFortress
SneakOutOfHideout
SneakIntoHideout
(in Storyspace.
MercenaryQuest)
InMercenaryFortress
Respawn
InBanditHideout
Player
Alive
Weappons
Armed
TakeSword
Sword
Unarmed
TakeBow
Bow
TakeLance
Lance
Disarm
Locomotion
Mount
OnFoot
Dismount
OnHorse
Armor
Armored
TakeHeavyArmor
HeavyArmor
Unarmored
LightArmor
TakeLightArmor
DiscardArmor
Dying
Respawn
PlayerDead
Strategy
Respawn
Uncovered
RealIdentity
Freed
StartSneak
InCaptivity
Fight
(CaughtUp)
(Escaped)
Escape
after 3 s /
Alarm
Detected
StopSneak
KillGuard
(TooLoud | |
WithinSightOf-
AGuard)
Stealth
Undetected
Respawn



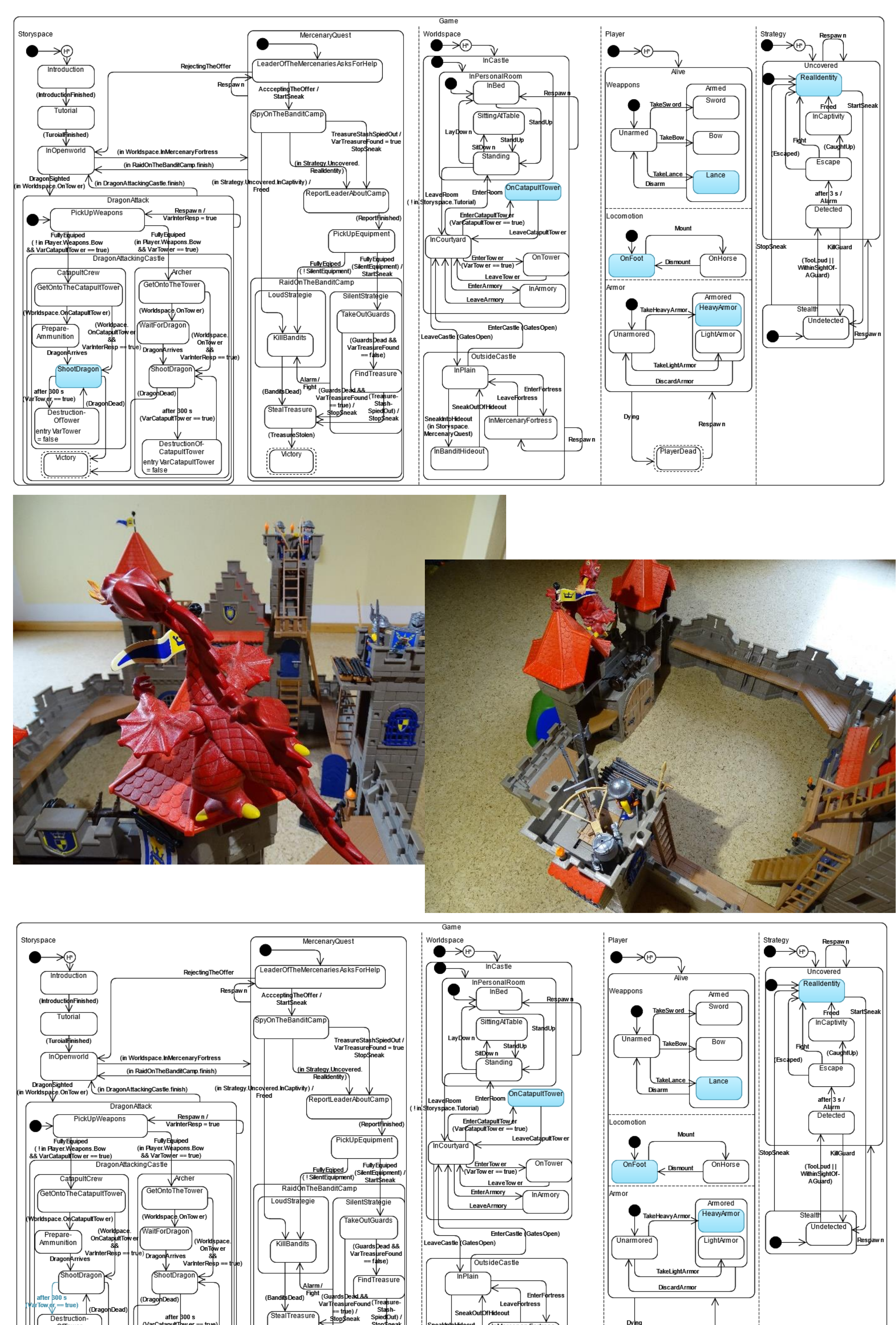
Game
Storyspace
Introduction
(IntroductionFinished)
Tutorial
(TuroialFinished)
InOpenworld
RejectingTheOffer
Respawn
(in Worldspace.InMercenaryFortress
(in RaidOnTheBanditCamp.finish)
DragonSighted
[in Worldspace.OnTower)
(in DragonAttackingCastle.finish)
DragonAttack
PickUpWeapons
Respawn /
VarInterResp = true
FullyEquiped
(! in Player.Weapons.Bow
&& VarCatapultTower == true)
FullyEquiped
(in Player.Weapons.Bow
&& VarTower == true)
DragonAttackingCastle
CatapultCrew
GetOntoTheCatapultTower
(Worldspace.OnCatapultTower)
Prepare-
Ammunition
DragonArrives
ShootDragon
after 300 s
[VarTower == true)
(DragonDead)
Destruction-
OfTower
entry VarTower
= false
Victory
Archer
GetOntoTheTower
(Worldspace.OnTower)
WaitForDragon
DragonArrives
ShootDragon
(DragonDead)
after 300 s
(VarCatapultTower == true)
DestructionOf-
CatapultTower
entry VarCatapultTower
= false
MercenaryQuest
LeaderOfTheMercenariesAsksForHelp
AcceptingTheOffer /
StartSneak
SpyOnTheBanditCamp
TreasureStashSpiedOut /
VarTreasureFound = true
StopSneak
(in Strategy.Uncovered.
RealIdentity)
(in Strategy.Uncovered.InCaptivity) /
Freed
ReportLeaderAboutCamp
(ReportFinished)
PickUpEquipment
FullyEqiped
(! SilentEquipment)
FullyEquiped
(SilentEquipment) /
StartSneak
RaidOnTheBanditCamp
LoudStrategie
SilentStrategie
KillBandits
TakeOutGuards
(GuardsDead &&
VarTreasureFound
== false)
FindTreasure
Alarm /
Fight
(BanditsDead)
(GuardsDead &&
VarTreasureFound
== true) /
StopSneak
(Treasure-
Stash-
SpiedOut) /
StopSneak
StealTreasure
(TreasureStolen)
Victory
Worldspace
InCastle
InPersonalRoom
InBed
SittingAtTable
StandUp
LayDown
SitDown
Standing
Respawn
LeaveRoom
(! in Storyspace.Tutorial)
EnterRoom
OnCatapultTower
EnterCatapultTower
(VarCatapultTower == true)
LeaveCatapultTower
InCourtyard
EnterTower
(VarTower == true)
OnTower
LeaveTower
EnterArmory
InArmory
LeaveArmory
EnterCastle (GatesOpen)
LeaveCastle (GatesOpen)
OutsideCastle
InPlain
EnterFortress
LeaveFortress
SneakOutOfHideout
SneakIntoHideout
(in Storyspace.
MercenaryQuest)
InMercenaryFortress
InBanditHideout
Player
Alive
Weappons
Armed
Sword
TakeSword
Unarmed
TakeBow
Bow
TakeLance
Disarm
Lance
Locomotion
Mount
OnFoot
Dismount
OnHorse
Armor
Armored
TakeHeavyArmor
HeavyArmor
Unarmored
LightArmor
TakeLightArmor
DiscardArmor
Dying
Respawn
PlayerDead
Strategy
Respawn
Uncovered
RealIdentity
Freed
StartSneak
InCaptivity
Fight
(CaughtUp)
(Escaped)
Escape
after 3 s /
Alarm
Detected
StopSneak
KillGuard
(ToLoud ||
WithinSightOf-
AGuard)
Stealth
Undetected
Respawn



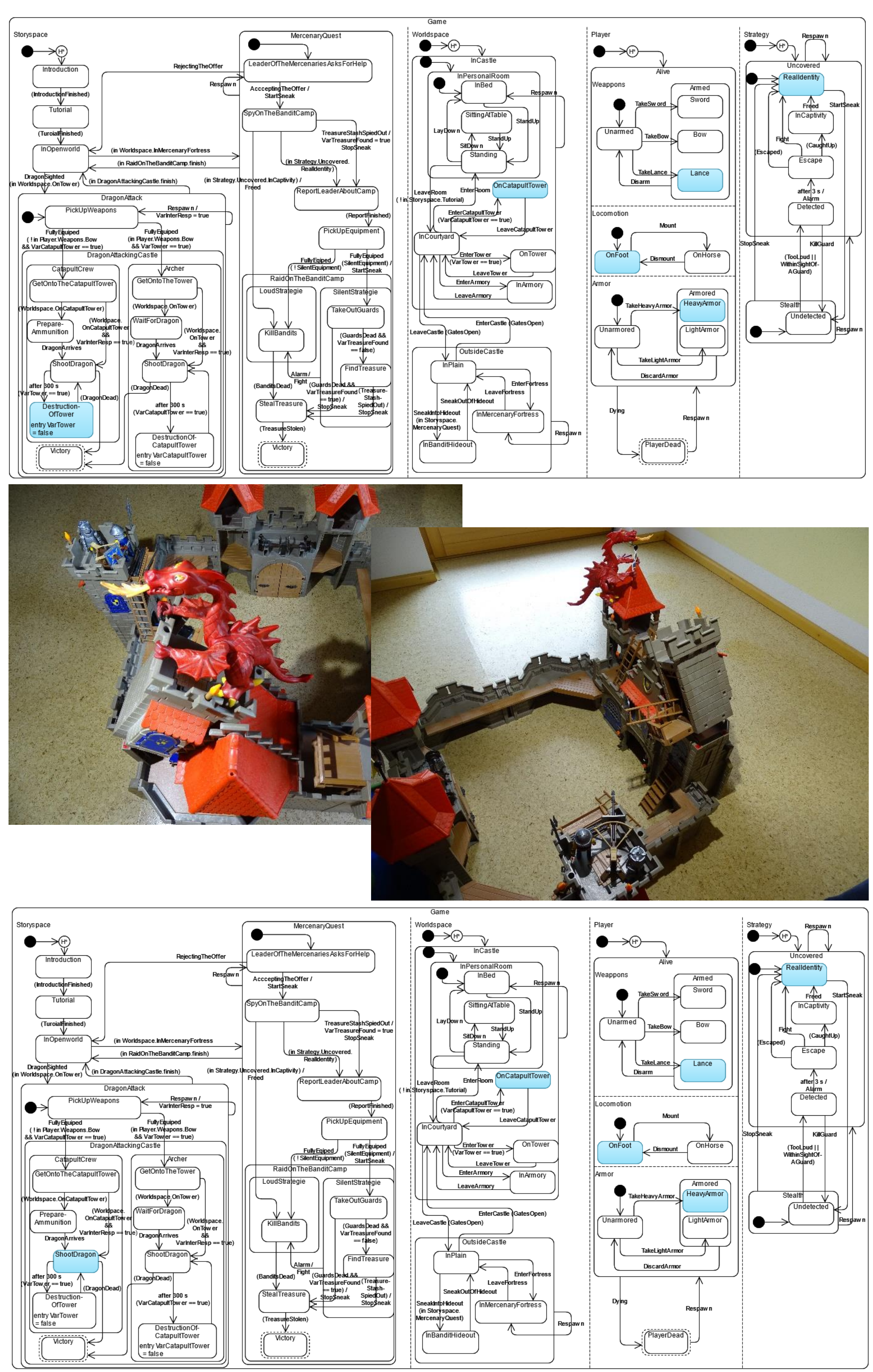
Game
Storyspace
Introduction
(IntroductionFinished)
Tutorial
(TurotialFinished)
InOpenworld
RejectingTheOffer
Respawn
(in Worldspace.InMercenaryFortress
(in RaidOnTheBanditCamp.finish)
DragonSighted
(in Worldspace.OnTower)
(in DragonAttackingCastle.finish)
DragonAttack
PickUpWeapons
Respawn / VarInterResp = true
FullyEquiped
DragonAttackingCastle
CatapultCrew
GetOntoTheCatapultTower
Prepare-Ammunition
DragonArrives
ShootDragon
Destruction-OfTower
entry VarTower = false
Victory
Archer
GetOntoTheTower
WaitForDragon
DestructionOf-CatapultTower
entry VarCatapultTower = false
MercenaryQuest
LeaderOfTheMercenariesAsksForHelp
AccceptingTheOffer / StartSneak
SpyOnTheBanditCamp
ReportLeaderAboutCamp
(ReportFinished)
PickUpEquipment
RaidOnTheBanditCamp
LoudStrategie
SilentStrategie
KillBandits
TakeOutGuards
FindTreasure
StealTreasure
(TreasureStolen)
Worldspace
InCastle
InPersonalRoom
InBed
SittingAtTable
Standing
LayDown
StandUp
SitDown
OnCatapultTower
InCourtyard
OnTower
InArmory
EnterRoom
LeaveRoom
EnterCatapultTower
LeaveCatapultTower
EnterTower
LeaveTower
EnterArmory
LeaveArmory
EnterCastle (GatesOpen)
LeaveCastle (GatesOpen)
OutsideCastle
InPlain
EnterFortress
LeaveFortress
SneakOutOfHideout
SneakIntoHideout
InMercenaryFortress
InBanditHideout
Player
Alive
Weappons
Armed
Unarmed
Sword
Bow
Lance
TakeSword
TakeBow
TakeLance
Disarm
Locomotion
OnFoot
OnHorse
Mount
Dismount
Armor
Unarmored
Armored
HeavyArmor
LightArmor
TakeHeavyArmor
TakeLightArmor
DiscardArmor
Dying
PlayerDead
Strategy
Uncovered
RealIdentity
InCaptivity
Escape
Detected
Freed
StartSneak
Fight
(Escaped)
(CaughtUp)
after 3 s / Alarm
StopSneak
KillGuard
(TooLoud || WithinSightOf-AGuard)
Stealth
Undetected



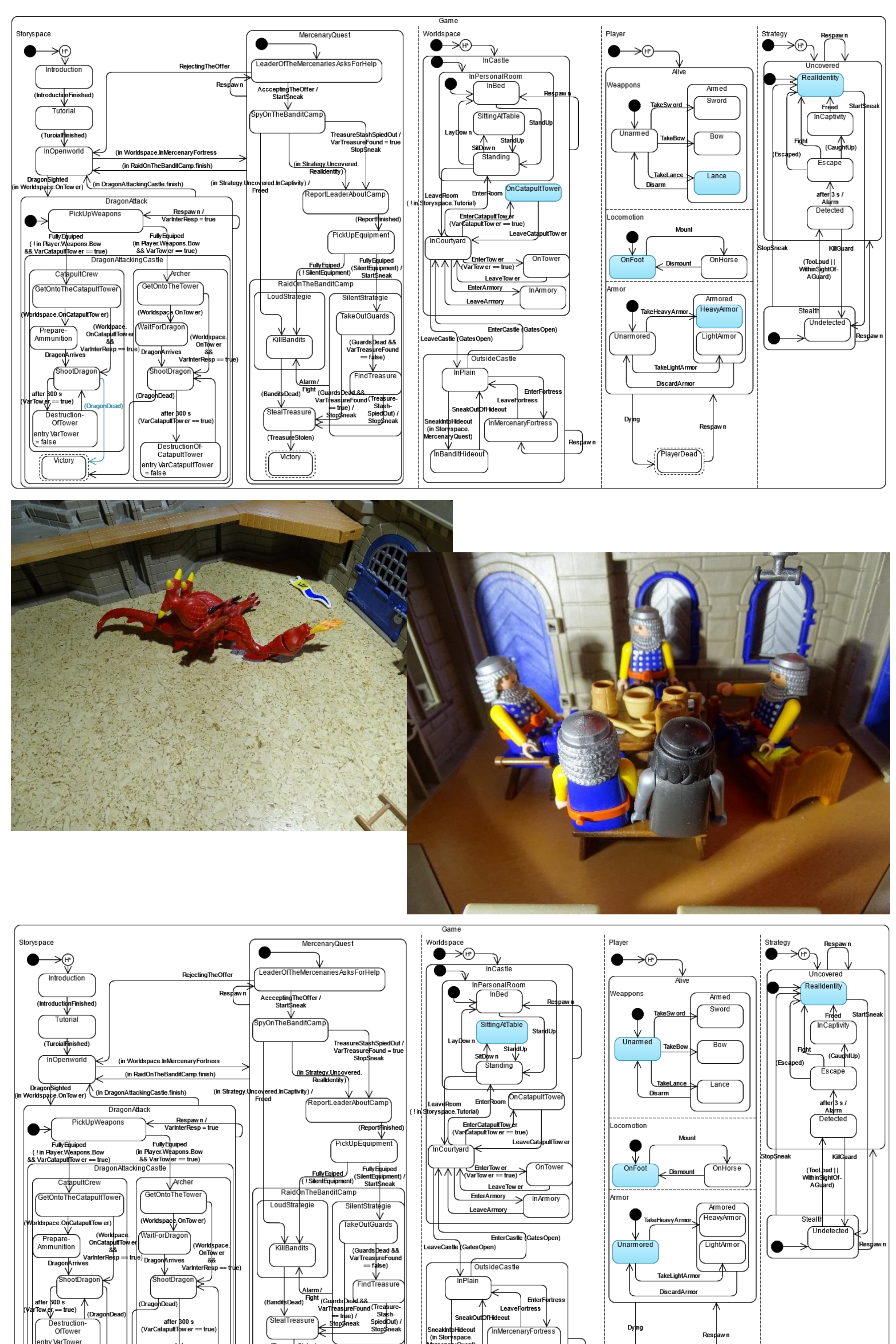
Game
Storyspace
Introduction
(IntroductionFinished)
Tutorial
(TurotialFinished)
InOpenworld
RejectingTheOffer
Respawn
(in Worldspace.InMercenaryFortress
(in RaidOnTheBanditCamp.finish)
DragonSighted
(in Worldspace.OnTower)
(in DragonAttackingCastle.finish)
DragonAttack
PickUpWeapons
Respawn / VarInterResp = true
FullyEquiped
DragonAttackingCastle
CatapultCrew
GetOntoTheCatapultTower
Prepare-Ammunition
DragonArrives
ShootDragon
Destruction-OfTower
entry VarTower = false
Victory
Archer
GetOntoTheTower
WaitForDragon
DestructionOf-CatapultTower
entry VarCatapultTower = false
MercenaryQuest
LeaderOfTheMercenariesAsksForHelp
AccceptingTheOffer / StartSneak
SpyOnTheBanditCamp
ReportLeaderAboutCamp
(ReportFinished)
PickUpEquipment
RaidOnTheBanditCamp
LoudStrategie
SilentStrategie
KillBandits
TakeOutGuards
FindTreasure
StealTreasure
(TreasureStolen)
Worldspace
InCastle
InPersonalRoom
InBed
SittingAtTable
Standing
OnCatapultTower
InCourtyard
OnTower
InArmory
OutsideCastle
InPlain
InMercenaryFortress
InBanditHideout
Player
Alive
Weappons
Armed
Unarmed
Sword
Bow
Lance
Locomotion
OnFoot
OnHorse
Armor
Unarmored
Armored
HeavyArmor
LightArmor
Dying
PlayerDead
Strategy
Uncovered
RealIdentity
InCaptivity
Escape
Detected
Stealth
Undetected



Game
Storyspace
Introduction
(IntroductionFinished)
Tutorial
(TurorialFinished)
InOpenworld
DragonAttack
PickUpWeapons
DragonAttackingCastle
CatapultCrew
GetOntoTheCatapultTower
Prepare-Ammunition
ShootDragon
Destruction-OfTower
Archer
GetOntoTheTower
WaitForDragon
DestructionOf-CatapultTower
Victory
MercenaryQuest
LeaderOfTheMercenariesAsksForHelp
SpyOnTheBanditCamp
ReportLeaderAboutCamp
PickUpEquipment
RaidOnTheBanditCamp
LoudStrategie
SilentStrategie
KillBandits
TakeOutGuards
FindTreasure
StealTreasure
Worldspace
InCastle
InPersonalRoom
InBed
SittingAtTable
Standing
OnCatapultTower
InCourtyard
OnTower
InArmory
OutsideCastle
InPlain
InMercenaryFortress
InBanditHideout
Player
Alive
Weappons
Unarmed
Armed
Sword
Bow
Lance
Locomotion
OnFoot
OnHorse
Armor
Unarmored
Armored
HeavyArmor
LightArmor
PlayerDead
Strategy
Uncovered
RealIdentity
InCaptivity
Escape
Detected
Stealth
Undetected

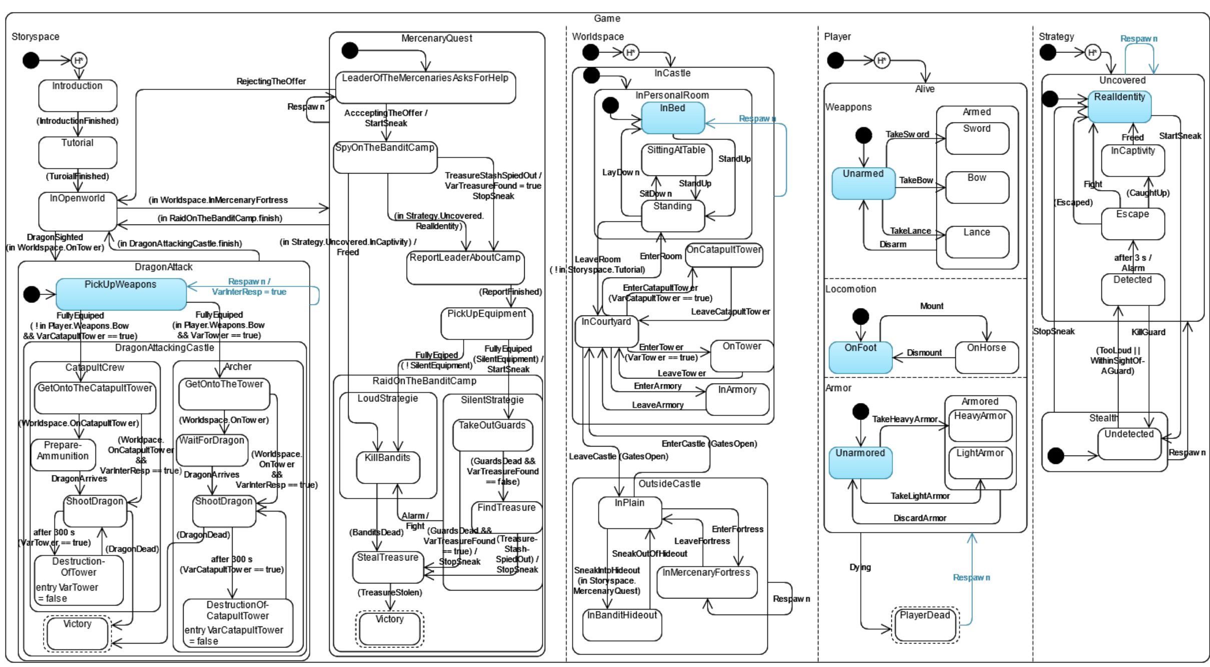


Game
Storyspace
Introduction
(IntroductionFinished)
Tutorial
(TuroialFinished)
InOpenworld
DragonSighted (in Worldspace.OnTower)
RejectingTheOffer
Respawn
(in Worldspace.InMercenaryFortress
(in RaidOnTheBanditCamp.finish)
(in DragonAttackingCastle.finish)
(in Strategy.Uncovered.InCaptivity) / Freed
DragonAttack
PickUpWeapons
Respawn / VarInterResp = true
FullyEquiped (! in Player.Weapons.Bow && VarCatapultTower == true)
FullyEquiped (in Player.Weapons.Bow && VarTower == true)
DragonAttackingCastle
CatapultCrew
GetOntoTheCatapultTower
(Worldspace.OnCatapultTower)
Prepare-Ammunition
(Worldspace.OnCatapultTower && VarInterResp == true)
DragonArrives
ShootDragon
after 300 s (VarTower == true)
(DragonDead)
Destruction-OfTower
entry VarTower = false
Victory
Archer
GetOntoTheTower
(Worldspace.OnTower)
WaitForDragon
(Worldspace.OnTower && VarInterResp == true)
DragonArrives
ShootDragon
(DragonDead)
after 300 s (VarCatapultTower == true)
DestructionOf-CatapultTower
entry VarCatapultTower = false
MercenaryQuest
LeaderOfTheMercenariesAsksForHelp
Acccepting TheOffer / StartSneak
SpyOnTheBanditCamp
TreasureStashSpiedOut / VarTreasureFound = true StopSneak
(in Strategy.Uncovered. RealIdentity)
ReportLeaderAboutCamp
(ReportFinished)
PickUpEquipment
FullyEquiped (! SilentEquipment)
FullyEquiped (SilentEquipment) / StartSneak
RaidOnTheBanditCamp
LoudStrategie
KillBandits
SilentStrategie
TakeOutGuards
(GuardsDead && VarTreasureFound == false)
FindTreasure
Alarm / Fight
(BanditsDead)
(GuardsDead && VarTreasureFound == true) / StopSneak
(Treasure-Stash-SpiedOut) / StopSneak
StealTreasure
(TreasureStolen)
Victory
Worldspace
InCastle
InPersonalRoom
InBed
Respawn
SittingAtTable
StandUp
LayDown
SitDown
Standing
LeaveRoom (! in Storyspace.Tutorial)
EnterRoom
OnCatapultTower
EnterCatapultTower (VarCatapultTower == true)
LeaveCatapultTower
InCourtyard
EnterTower (VarTower == true)
OnTower
LeaveTower
EnterArmory
InArmory
LeaveArmory
EnterCastle (GatesOpen)
LeaveCastle (GatesOpen)
OutsideCastle
InPlain
EnterFortress
LeaveFortress
SneakOutOfHideout
SneakIntoHideout (in Storyspace. MercenaryQuest)
InMercenaryFortress
InBanditHideout
Respawn
Player
Alive
Weappons
Armed
Sword
TakeSword
Unarmed
TakeBow
Bow
TakeLance
Lance
Disarm
Locomotion
Mount
OnFoot
Dismount
OnHorse
Armor
Armored
TakeHeavyArmor
HeavyArmor
Unarmored
LightArmor
TakeLightArmor
DiscardArmor
Dying
Respawn
PlayerDead
Strategy
Respawn
Uncovered
RealIdentity
Freed
StartSneak
InCaptivity
Fight
(CaughtUp)
(Escaped)
Escape
after 3 s / Alarm
Detected
StopSneak
KillGuard
(TooLoud || WithinSightOf-AGuard)
Stealth
Undetected
Respawn

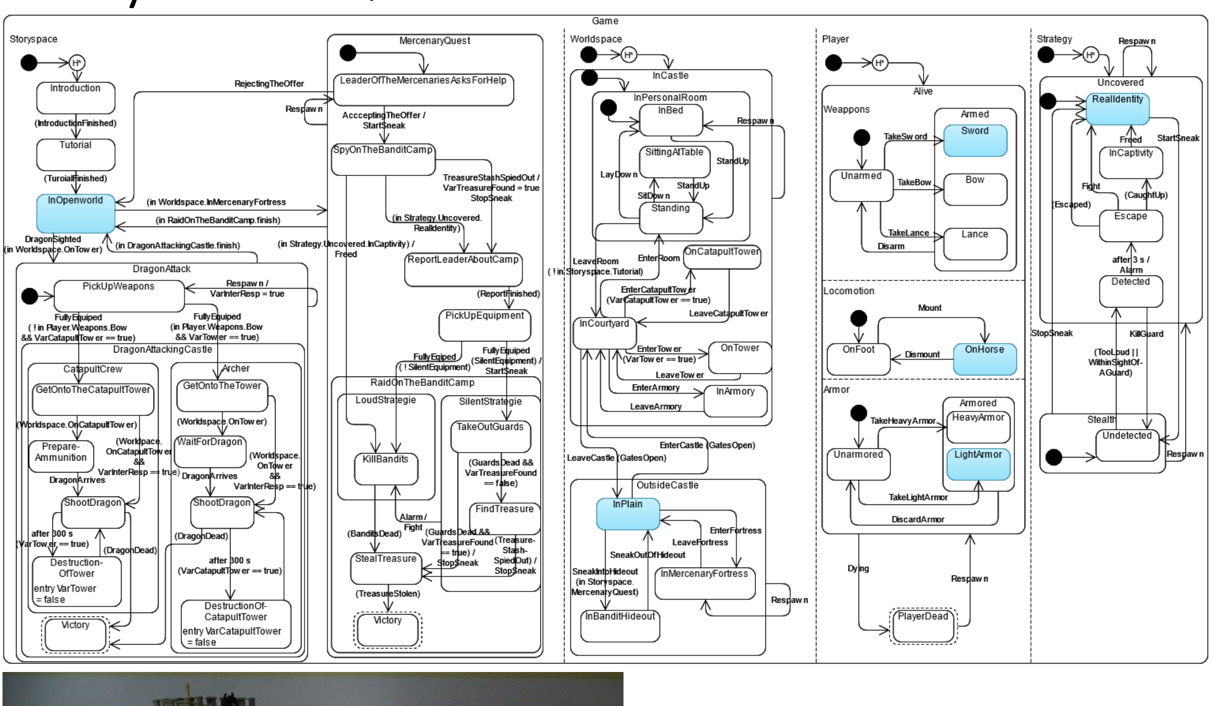


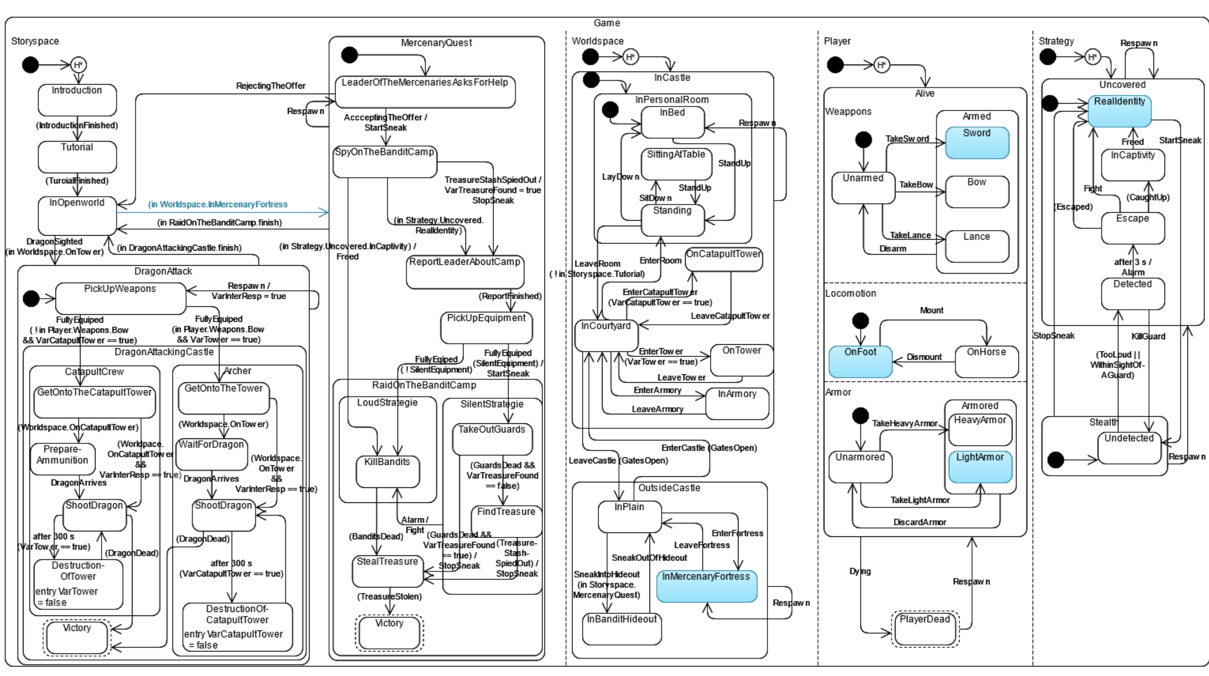
Game
Storyspace
Introduction
(IntroductionFinished)
Tutorial
(TurorialFinished)
InOpenworld
DragonSighted
(in Worldspace.OnTower)
RejectingTheOffer
Respawn
(in Worldspace.InMercenaryFortress
(in RaidOnTheBanditCamp.finish)
(in DragonAttackingCastle.finish)
DragonAttack
PickUpWeapons
Respawn /
VarInterResp = true
FullyEquiped
(! in Player.Weapons.Bow
&& VarCatapultTower == true)
FullyEquiped
(in Player.Weapons.Bow
&& VarTower == true)
DragonAttackingCastle
CatapultCrew
GetOntoTheCatapultTower
(Worldspace.OnCatapultTower)
Prepare-
Ammunition
(Worldspace.
OnCatapultTower
&&
VarInterResp == true)
DragonArrives
ShootDragon
after 300 s
(VarTower == true)
(DragonDead)
Destruction-
OfTower
entry VarTower
= false
Victory
Archer
GetOntoTheTower
(Worldspace.OnTower)
WaitForDragon
(Worldspace.
OnTower
&&
VarInterResp == true)
DragonArrives
ShootDragon
(DragonDead)
after 300 s
(VarCatapultTower == true)
DestructionOf-
CatapultTower
entry VarCatapultTower
= false
MercenaryQuest
LeaderOfTheMercenariesAsksForHelp
Respawn
AccceptingTheOffer /
StartSneak
SpyOnTheBanditCamp
TreasureStashSpiedOut /
VarTreasureFound = true
StopSneak
(in Strategy.Uncovered.
RealIdentity)
(in Strategy.Uncovered.InCaptivity) /
Freed
ReportLeaderAboutCamp
(ReportFinished)
PickUpEquipment
FullyEquiped
(! SilentEquipment)
FullyEquiped
(SilentEquipment) /
StartSneak
RaidOnTheBanditCamp
LoudStrategie
SilentStrategie
KillBandits
TakeOutGuards
(GuardsDead &&
VarTreasureFound
== false)
FindTreasure
Alarm /
Fight
(BanditsDead)
(GuardsDead &&
VarTreasureFound
== true) /
StopSneak
(Treasure-
Stash-
SpiedOut) /
StopSneak
StealTreasure
(TreasureStolen)
Victory
Worldspace
InCastle
InPersonalRoom
InBed
SittingAtTable
StandUp
LayDown
SitDown
StandUp
Standing
Respawn
LeaveRoom
(! in Storyspace.Tutorial)
EnterRoom
OnCatapultTower
EnterCatapultTower
(VarCatapultTower == true)
LeaveCatapultTower
InCourtyard
EnterTower
(VarTower == true)
OnTower
LeaveTower
EnterArmory
InArmory
LeaveArmory
EnterCastle (GatesOpen)
LeaveCastle (GatesOpen)
OutsideCastle
InPlain
EnterFortress
LeaveFortress
SneakOutOfHideout
SneakIntoHideout
(in Storyspace.
MercenaryQuest)
InMercenaryFortress
Respawn
InBanditHideout
Player
Alive
Weappons
Armed
TakeSword
Sword
Unarmed
TakeBow
Bow
TakeLance
Lance
Disarm
Locomotion
Mount
OnFoot
Dismount
OnHorse
Armor
Armored
TakeHeavyArmor
HeavyArmor
Unarmored
LightArmor
TakeLightArmor
DiscardArmor
Dying
Respawn
PlayerDead
Strategy
Respawn
Uncovered
RealIdentity
Freed
StartSneak
InCaptivity
Fight
(CaughtUp)
(Escaped)
Escape
after 3 s /
Alarm
Detected
StopSneak
KillGuard
(TooLoud | |
WithinSightOf-
AGuard)
Stealth
Undetected
Respawn

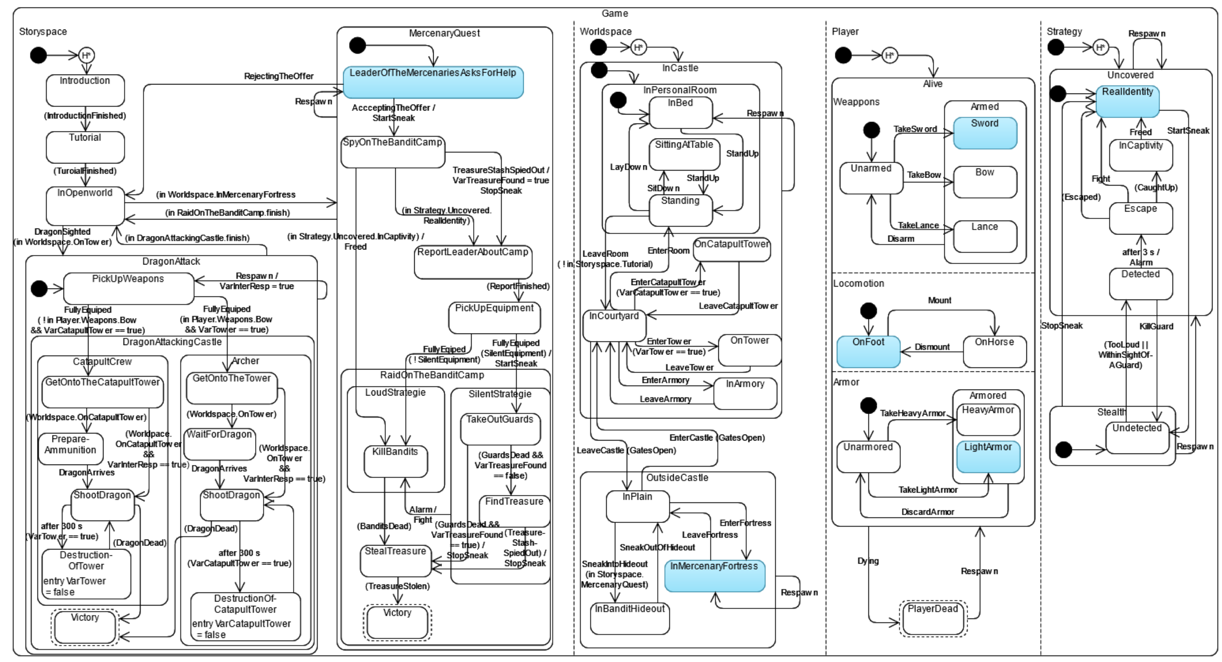
Respawn- Interrupt:

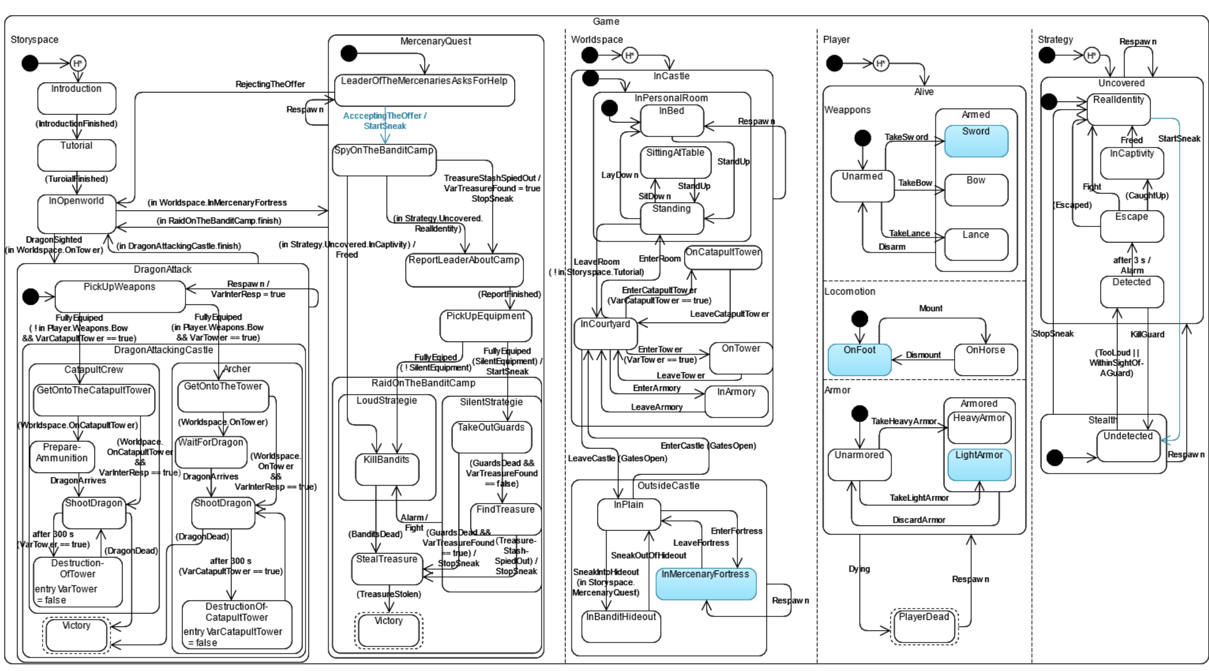


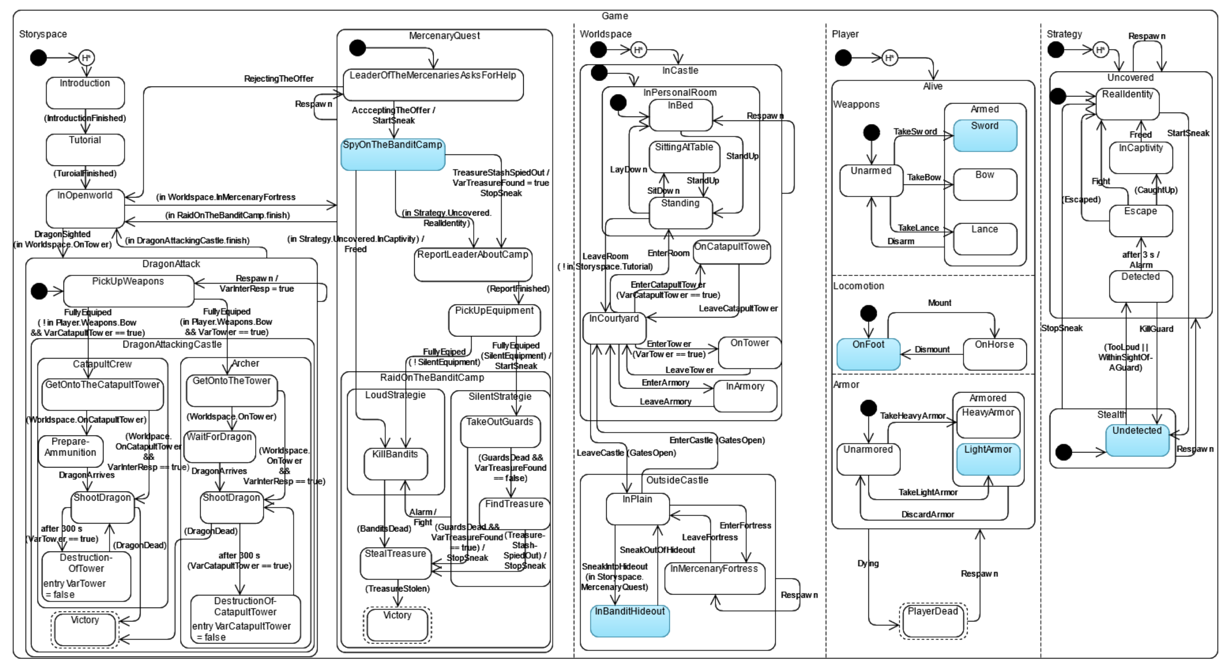
Storyline – Quest:



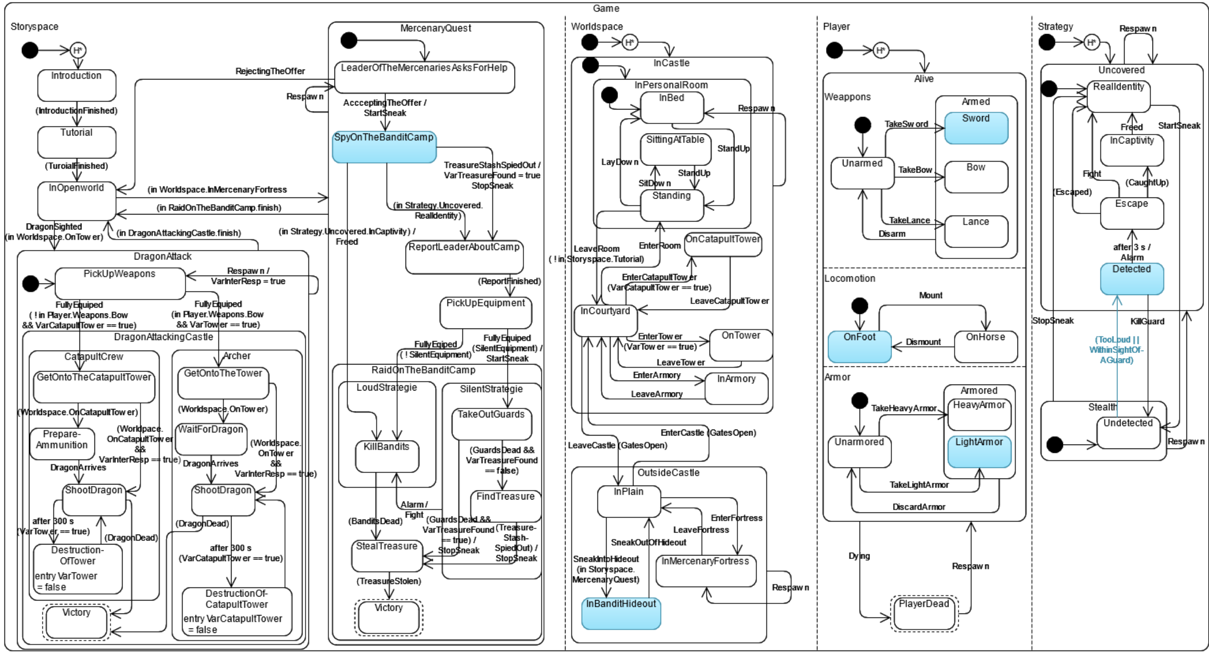



Game
Storyspace
Introduction
Tutorial
InOpenworld
DragonAttack
PickUpWeapons
DragonAttackingCastle
CatapultCrew
GetOntoTheCatapultTower
Prepare-Ammunition
ShootDragon
Destruction-OfTower
Archer
GetOntoTheTower
WaitForDragon
Destruction-Of-CatapultTower
Victory
MercenaryQuest
LeaderOfTheMercenariesAsksForHelp
SpyOnTheBanditCamp
ReportLeaderAboutCamp
PickUpEquipment
RaidOnTheBanditCamp
LoudStrategie
SilentStrategie
KillBandits
TakeOutGuards
FindTreasure
StealTreasure
Worldspace
InCastle
InPersonalRoom
InBed
SittingAtTable
Standing
OnCatapultTower
InCourtyard
OnTower
InArmory
OutsideCastle
InPlain
InMercenaryFortress
InBanditHideout
Player
Alive
Weappons
Unarmed
Armed
Sword
Bow
Lance
Locomotion
OnFoot
OnHorse
Armor
Unarmored
Armored
HeavyArmor
LightArmor
PlayerDead
Strategy
Uncovered
RealIdentity
InCaptivity
Escape
Detected
Stealth
Undetected

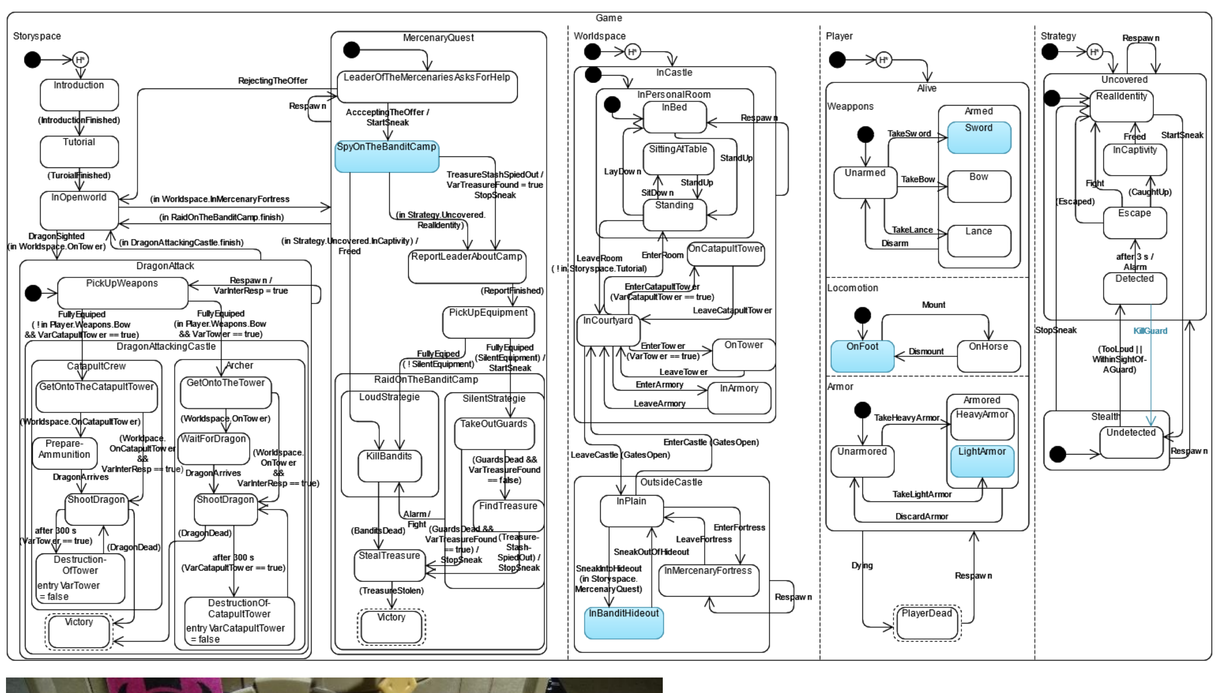

Game
Storyspace
Introduction
Tutorial
InOpenworld
DragonAttack
PickUpWeapons
DragonAttackingCastle
CatapultCrew
GetOntoTheCatapultTower
Prepare-Ammunition
ShootDragon
Destruction-OfTower
Archer
GetOntoTheTower
WaitForDragon
Destruction-Of-CatapultTower
Victory
MercenaryQuest
LeaderOfTheMercenariesAsksForHelp
SpyOnTheBanditCamp
ReportLeaderAboutCamp
PickUpEquipment
RaidOnTheBanditCamp
LoudStrategie
SilentStrategie
KillBandits
TakeOutGuards
FindTreasure
StealTreasure
Worldspace
InCastle
InPersonalRoom
InBed
SittingAtTable
Standing
OnCatapultTower
InCourtyard
OnTower
InArmory
OutsideCastle
InPlain
InMercenaryFortress
InBanditHideout
Player
Alive
Weappons
Unarmed
Armed
Sword
Bow
Lance
Locomotion
OnFoot
OnHorse
Armor
Unarmored
Armored
HeavyArmor
LightArmor
PlayerDead
Strategy
Uncovered
RealIdentity
InCaptivity
Escape
Detected
Stealth
Undetected

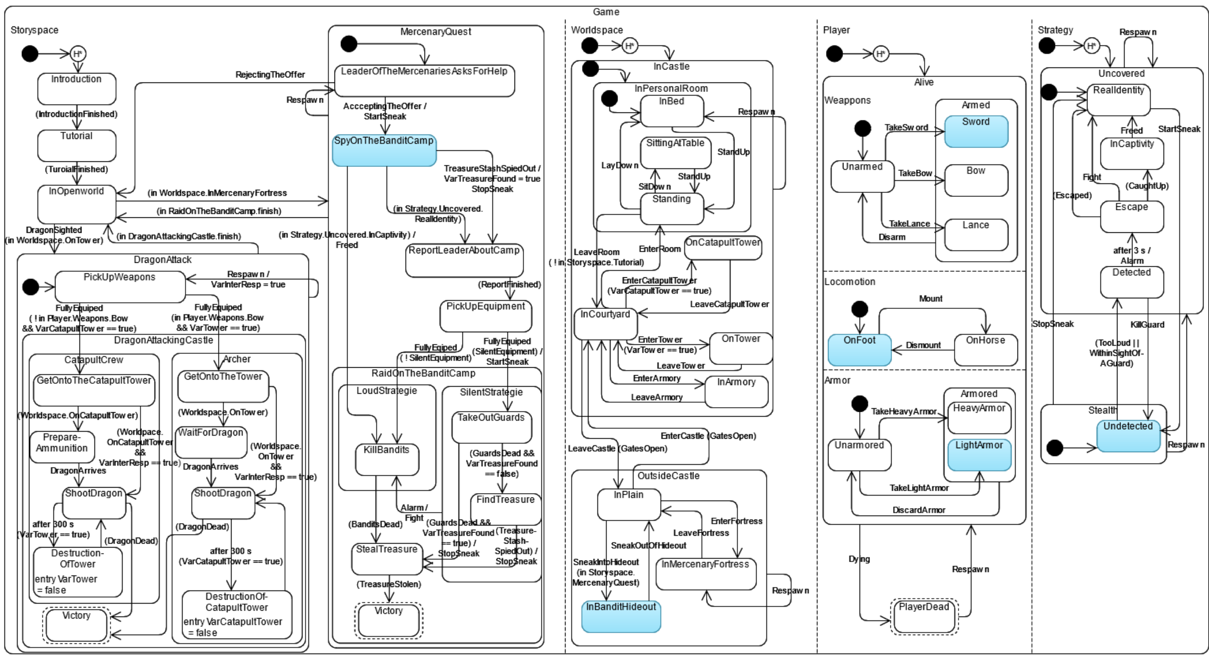


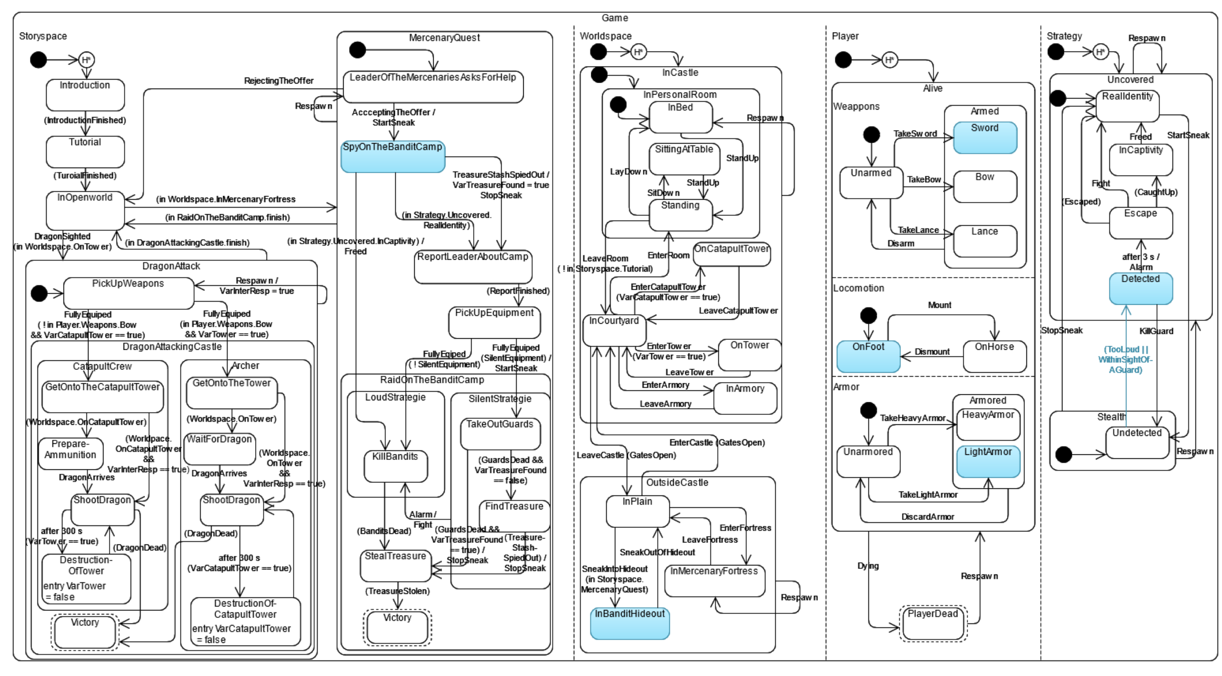
Game
Storyspace
Introduction
(IntroductionFinished)
Tutorial
(TuroialFinished)
InOpenworld
RejectingTheOffer
Respawn
(in Worldspace.InMercenaryFortress
(in RaidOnTheBanditCamp.finish)
DragonSighted
(in Worldspace.OnTower)
(in DragonAttackingCastle.finish)
(in Strategy.Uncovered.InCaptivity) /
Freed
DragonAttack
PickUpWeapons
Respawn /
VarInterResp = true
FullyEquiped
(! in Player.Weapons.Bow
&& VarCatapultTower == true)
FullyEquiped
(in Player.Weapons.Bow
&& VarTower == true)
DragonAttackingCastle
CatapultCrew
GetOntoTheCatapultTower
(Worldspace.OnCatapultTower)
Prepare-
Ammunition
DragonArrives
ShootDragon
after 300 s
(VarTower == true)
(DragonDead)
Destruction-
OfTower
entry VarTower
= false
Victory
Archer
GetOntoTheTower
(Worldspace.OnTower)
WaitForDragon
DragonArrives
ShootDragon
(DragonDead)
after 300 s
(VarCatapultTower == true)
Destruction-
OfCatapultTower
entry VarCatapultTower
= false
MercenaryQuest
LeaderOfTheMercenariesAsksForHelp
AccceptingTheOffer /
StartSneak
SpyOnTheBanditCamp
TreasureStashSpiedOut /
VarTreasureFound = true
StopSneak
(in Strategy.Uncovered.
RealIdentity)
ReportLeaderAboutCamp
(ReportFinished)
PickUpEquipment
FullyEquiped
(! SilentEquipment)
FullyEquiped
(SilentEquipment) /
StartSneak
RaidOnTheBanditCamp
LoudStrategie
KillBandits
SilentStrategie
TakeOutGuards
(GuardsDead &&
VarTreasureFound
== false)
FindTreasure
Alarm /
Fight
(BanditsDead)
(GuardsDead &&
VarTreasureFound
== true) /
StopSneak
(Treasure-
Stash-
SpiedOut) /
StopSneak
StealTreasure
(TreasureStolen)
Victory
Worldspace
InCastle
InPersonalRoom
InBed
SittingAtTable
StandUp
LayDown
SitDown
Standing
Respawn
LeaveRoom
(! in Storyspace.Tutorial)
EnterRoom
OnCatapultTower
EnterCatapultTower
(VarCatapultTower == true)
LeaveCatapultTower
InCourtyard
EnterTower
(VarTower == true)
OnTower
LeaveTower
EnterArmory
InArmory
LeaveArmory
EnterCastle (GatesOpen)
LeaveCastle (GatesOpen)
OutsideCastle
InPlain
EnterFortress
LeaveFortress
SneakOutOfHideout
SneakIntoHideout
(in Storyspace.
MercenaryQuest)
InMercenaryFortress
InBanditHideout
Respawn
Player
Alive
Weappons
Armed
Sword
TakeSword
Unarmed
TakeBow
Bow
TakeLance
Lance
Disarm
Locomotion
Mount
OnFoot
Dismount
OnHorse
Armor
Armored
TakeHeavyArmor
HeavyArmor
Unarmored
LightArmor
TakeLightArmor
DiscardArmor
Dying
Respawn
PlayerDead
Strategy
Respawn
Uncovered
RealIdentity
Freed
StartSneak
InCaptivity
Fight
(CaughtUp)
(Escaped)
Escape
after 3 s /
Alarm
Detected
StopSneak
KillGuard
(ToLoud ||
WithinSightOf-
AGuard)
Stealth
Undetected
Respawn



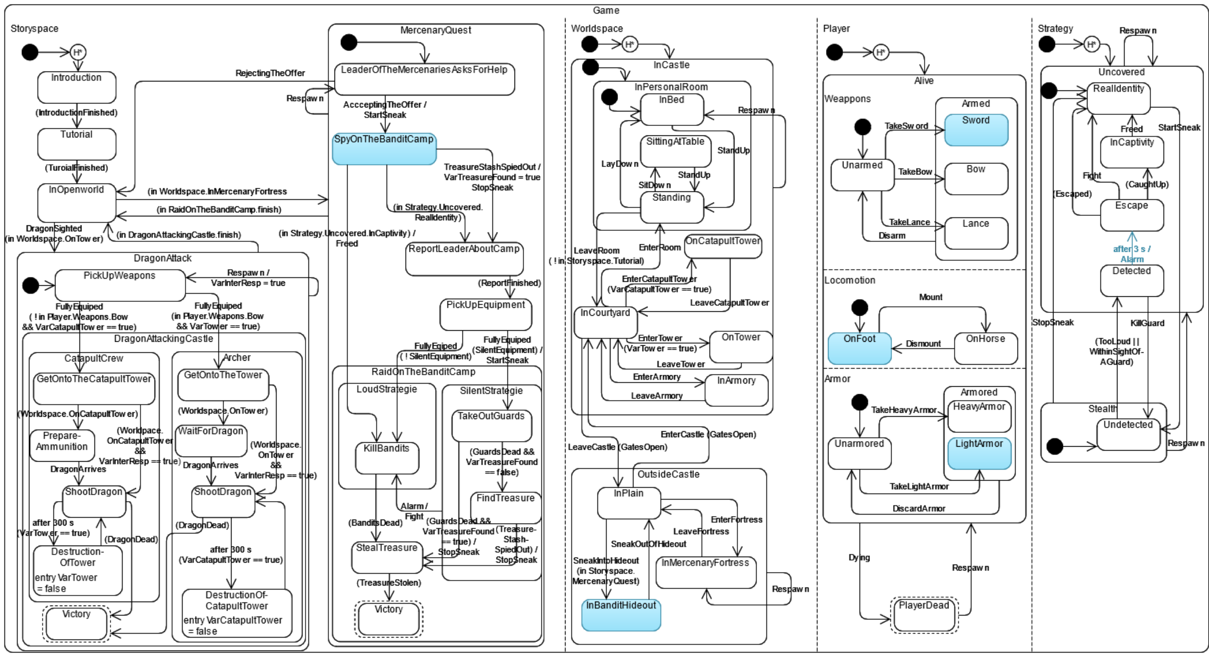
Game
Storyspace
Introduction
(IntroductionFinished)
Tutorial
(TuroialFinished)
InOpenworld
RejectingTheOffer
Respawn
(in Worldspace.InMercenaryFortress
(in RaidOnTheBanditCamp.finish)
DragonSighted
(in Worldspace.OnTower)
(in DragonAttackingCastle.finish)
(in Strategy.Uncovered.InCaptivity) /
Freed
DragonAttack
PickUpWeapons
Respawn /
VarInterResp = true
DragonAttackingCastle
CatapultCrew
GetOntoTheCatapultTower
Prepare-
Ammunition
ShootDragon
Destruction-
OfTower
Victory
Archer
GetOntoTheTower
WaitForDragon
ShootDragon
Destruction-
OfCatapultTower
MercenaryQuest
LeaderOfTheMercenariesAsksForHelp
SpyOnTheBanditCamp
ReportLeaderAboutCamp
PickUpEquipment
RaidOnTheBanditCamp
LoudStrategie
KillBandits
SilentStrategie
TakeOutGuards
FindTreasure
StealTreasure
Victory
Worldspace
InCastle
InPersonalRoom
InBed
SittingAtTable
Standing
OnCatapultTower
InCourtyard
OnTower
InArmory
OutsideCastle
InPlain
InMercenaryFortress
InBanditHideout
Player
Alive
Weappons
Armed
Sword
Unarmed
Bow
Lance
Locomotion
OnFoot
OnHorse
Armor
Armored
HeavyArmor
Unarmored
LightArmor
PlayerDead
Strategy
Uncovered
RealIdentity
InCaptivity
Escape
Detected
StopSneak
KillGuard
(ToLoud ||
WithinSightOf-
AGuard)
Stealth
Undetected


Game
Storyspace
Introduction
(IntroductionFinished)
Tutorial
(TurorialFinished)
InOpenworld
DragonSighted (in Worldspace.OnTower)
RejectingTheOffer
Respawn
(in Worldspace.InMercenaryFortress
(in RaidOnTheBanditCamp.finish)
(in DragonAttackingCastle.finish)
(in Strategy.Uncovered.InCaptivity) / Freed
DragonAttack
PickUpWeapons
Respawn / VarInterResp = true
FullyEquiped (! in Player.Weapons.Bow && VarCatapultTower == true)
FullyEquiped (in Player.Weapons.Bow && VarTower == true)
DragonAttackingCastle
CatapultCrew
GetOntoTheCatapultTower
(Worldspace.OnCatapultTower)
Prepare-Ammunition
(Worldspace.OnCatapultTower && VarInterResp == true)
DragonArrives
ShootDragon
after 300 s (VarTower == true)
(DragonDead)
Destruction-OfTower
entry VarTower = false
Victory
Archer
GetOntoTheTower
(Worldspace.OnTower)
WaitForDragon
(Worldspace.OnTower && VarInterResp == true)
DragonArrives
ShootDragon
(DragonDead)
after 300 s (VarCatapultTower == true)
DestructionOf-CatapultTower
entry VarCatapultTower = false
MercenaryQuest
LeaderOfTheMercenariesAsksForHelp
AccceptingTheOffer / StartSneak
SpyOnTheBanditCamp
TreasureStashSpiedOut / VarTreasureFound = true StopSneak
(in Strategy.Uncovered.RealIdentity)
ReportLeaderAboutCamp
(ReportFinished)
PickUpEquipment
FullyEqiped (! SilentEquipment)
FullyEquiped (SilentEquipment) / StartSneak
RaidOnTheBanditCamp
LoudStrategie
KillBandits
SilentStrategie
TakeOutGuards
(GuardsDead && VarTreasureFound == false)
FindTreasure
Alarm / Fight
(BanditsDead)
(GuardsDead && VarTreasureFound == true) / StopSneak
(Treasure-Stash-SpiedOut) / StopSneak
StealTreasure
(TreasureStolen)
Victory
Worldspace
InCastle
InPersonalRoom
InBed
Respawn
SittingAtTable
StandUp
LayDown
SitDown
StandUp
Standing
LeaveRoom (! in Storyspace.Tutorial)
EnterRoom
OnCatapultTower
EnterCatapultTower (VarCatapultTower == true)
LeaveCatapultTower
InCourtyard
EnterTower (VarTower == true)
OnTower
LeaveTower
EnterArmory
InArmory
LeaveArmory
EnterCastle (GatesOpen)
LeaveCastle (GatesOpen)
OutsideCastle
InPlain
EnterFortress
LeaveFortress
SneakOutOfHideout
SneakIntoHideout (in Storyspace.MercenaryQuest)
InMercenaryFortress
Respawn
InBanditHideout
Player
Alive
Weappons
Armed
TakeSword
Sword
Unarmed
TakeBow
Bow
TakeLance
Lance
Disarm
Locomotion
Mount
OnFoot
Dismount
OnHorse
Armor
Armored
TakeHeavyArmor
HeavyArmor
Unarmored
LightArmor
TakeLightArmor
DiscardArmor
Dying
Respawn
PlayerDead
Strategy
Respawn
Uncovered
RealIdentity
Freed
StartSneak
InCaptivity
Fight
(CaughtUp)
(Escaped)
Escape
after 3 s / Alarm
Detected
StopSneak
KillGuard
(TooLoud || WithinSightOf-AGuard)
Stealth
Undetected
Respawn

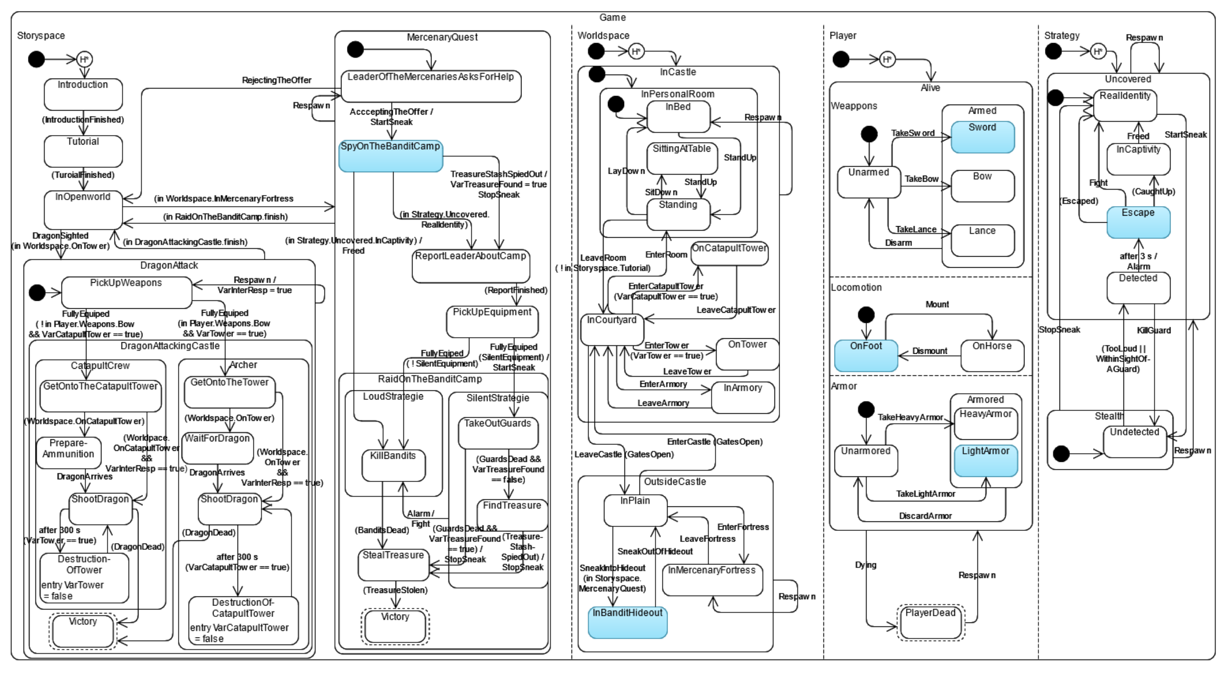

Game
Storyspace
Introduction
(IntroductionFinished)
Tutorial
(TurorialFinished)
InOpenworld
DragonSighted (in Worldspace.OnTower)
RejectingTheOffer
Respawn
(in Worldspace.InMercenaryFortress
(in RaidOnTheBanditCamp.finish)
(in DragonAttackingCastle.finish)
(in Strategy.Uncovered.InCaptivity) / Freed
DragonAttack
PickUpWeapons
Respawn / VarInterResp = true
FullyEquiped (! in Player.Weapons.Bow && VarCatapultTower == true)
FullyEquiped (in Player.Weapons.Bow && VarTower == true)
DragonAttackingCastle
CatapultCrew
GetOntoTheCatapultTower
(Worldspace.OnCatapultTower)
Prepare-Ammunition
(Worldspace.OnCatapultTower && VarInterResp == true)
DragonArrives
ShootDragon
after 300 s (VarTower == true)
(DragonDead)
Destruction-OfTower
entry VarTower = false
Victory
Archer
GetOntoTheTower
(Worldspace.OnTower)
WaitForDragon
(Worldspace.OnTower && VarInterResp == true)
DragonArrives
ShootDragon
(DragonDead)
after 300 s (VarCatapultTower == true)
DestructionOf-CatapultTower
entry VarCatapultTower = false
MercenaryQuest
LeaderOfTheMercenariesAsksForHelp
AccceptingTheOffer / StartSneak
SpyOnTheBanditCamp
TreasureStashSpiedOut / VarTreasureFound = true StopSneak
(in Strategy.Uncovered.RealIdentity)
ReportLeaderAboutCamp
(ReportFinished)
PickUpEquipment
FullyEqiped (! SilentEquipment)
FullyEquiped (SilentEquipment) / StartSneak
RaidOnTheBanditCamp
LoudStrategie
KillBandits
SilentStrategie
TakeOutGuards
(GuardsDead && VarTreasureFound == false)
FindTreasure
Alarm / Fight
(BanditsDead)
(GuardsDead && VarTreasureFound == true) / StopSneak
(Treasure-Stash-SpiedOut) / StopSneak
StealTreasure
(TreasureStolen)
Victory
Worldspace
InCastle
InPersonalRoom
InBed
Respawn
SittingAtTable
StandUp
LayDown
SitDown
StandUp
Standing
LeaveRoom (! in Storyspace.Tutorial)
EnterRoom
OnCatapultTower
EnterCatapultTower (VarCatapultTower == true)
LeaveCatapultTower
InCourtyard
EnterTower (VarTower == true)
OnTower
LeaveTower
EnterArmory
InArmory
LeaveArmory
EnterCastle (GatesOpen)
LeaveCastle (GatesOpen)
OutsideCastle
InPlain
EnterFortress
LeaveFortress
SneakOutOfHideout
SneakIntoHideout (in Storyspace.MercenaryQuest)
InMercenaryFortress
Respawn
InBanditHideout
Player
Alive
Weappons
Armed
TakeSword
Sword
Unarmed
TakeBow
Bow
TakeLance
Lance
Disarm
Locomotion
Mount
OnFoot
Dismount
OnHorse
Armor
Armored
TakeHeavyArmor
HeavyArmor
Unarmored
LightArmor
TakeLightArmor
DiscardArmor
Dying
Respawn
PlayerDead
Strategy
Respawn
Uncovered
RealIdentity
Freed
StartSneak
InCaptivity
Fight
(CaughtUp)
(Escaped)
Escape
after 3 s / Alarm
Detected
StopSneak
KillGuard
(TooLoud || WithinSightOf-AGuard)
Stealth
Undetected
Respawn



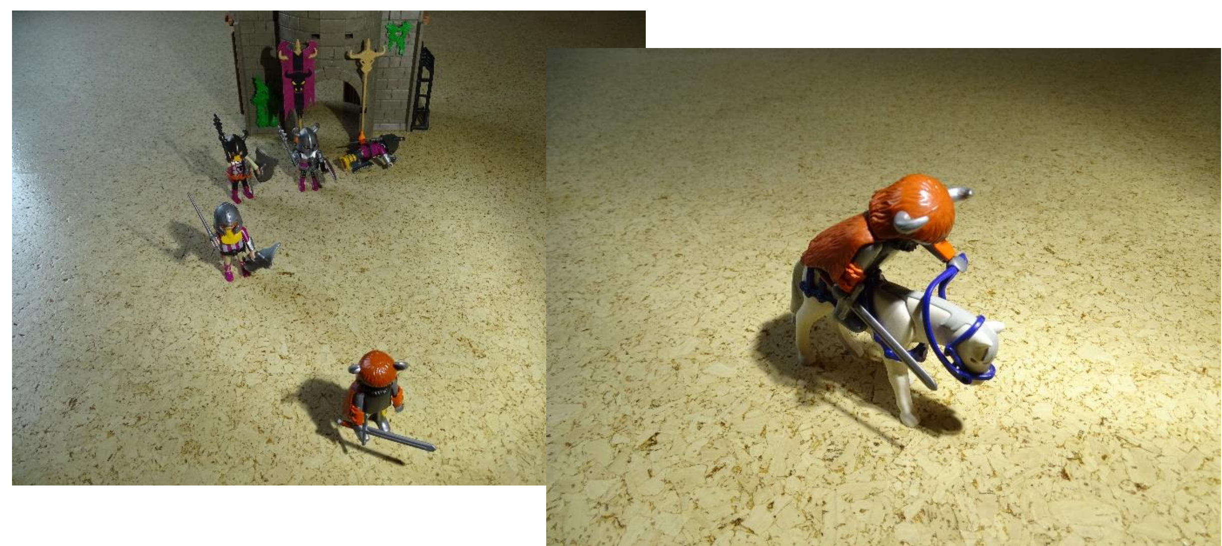
Game
Storyspace
Introduction
(IntroductionFinished)
Tutorial
(TuroialFinished)
InOpenworld
RejectingTheOffer
Respawn
(in Worldspace.InMercenaryFortress
(in RaidOnTheBanditCamp.finish)
(in DragonAttackingCastle.finish)
DragonSighted
(in Worldspace.OnTower)
DragonAttack
PickUpWeapons
Respawn /
VarInterResp = true
DragonAttackingCastle
CatapultCrew
GetOntoTheCatapultTower
Prepare-
Ammunition
DragonArrives
ShootDragon
Destruction-
OfTower
entry VarTower
= false
Archer
GetOntoTheTower
WaitForDragon
(DragonDead)
DestructionOf-
CatapultTower
entry VarCatapultTower
= false
Victory
MercenaryQuest
LeaderOfTheMercenariesAsksForHelp
Acccepting TheOffer /
StartSneak
SpyOnTheBanditCamp
TreasureStashSpiedOut /
VarTreasureFound = true
StopSneak
ReportLeaderAboutCamp
(ReportFinished)
PickUpEquipment
RaidOnTheBanditCamp
LoudStrategie
KillBandits
SilentStrategie
TakeOutGuards
FindTreasure
StealTreasure
(TreasureStolen)
Worldspace
InCastle
InPersonalRoom
InBed
SittingAtTable
Standing
StandUp
LayDown
SitDown
OnCatapultTower
EnterRoom
LeaveRoom
InCourtyard
OnTower
EnterTower
LeaveTower
InArmory
EnterArmory
LeaveArmory
EnterCastle (GatesOpen)
LeaveCastle (GatesOpen)
OutsideCastle
InPlain
EnterFortress
LeaveFortress
InMercenaryFortress
SneakOutOfHideout
InBanditHideout
Player
Alive
Weappons
Unarmed
Armed
Sword
Bow
Lance
TakeSword
TakeBow
TakeLance
Disarm
Locomotion
Mount
Dismount
OnFoot
OnHorse
Armor
Unarmored
Armored
HeavyArmor
LightArmor
TakeHeavyArmor
TakeLightArmor
DiscardArmor
Dying
Respawn
PlayerDead
Strategy
Uncovered
RealIdentity
Freed
StartSneak
InCaptivity
Fight
(CaughtUp)
(Escaped)
Escape
after 3 s /
Alarm
Detected
StopSneak
KillGuard
(ToLoud ||
WithinSightOf-
AGuard)
Stealth
Undetected

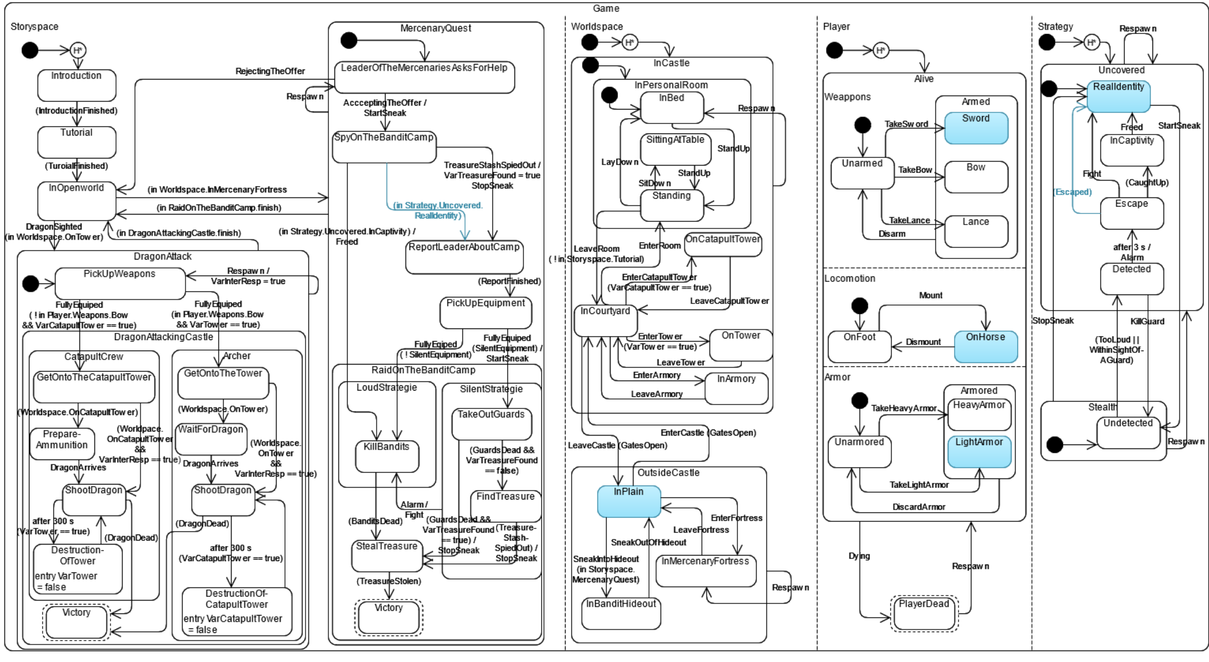


Game
Storyspace
Introduction
Tutorial
InOpenworld
DragonAttack
PickUpWeapons
DragonAttackingCastle
CatapultCrew
Archer
Victory
MercenaryQuest
LeaderOfTheMercenariesAsksForHelp
SpyOnTheBanditCamp
ReportLeaderAboutCamp
PickUpEquipment
RaidOnTheBanditCamp
Worldspace
InCastle
OutsideCastle
InBanditHideout
Player
Alive
PlayerDead
Strategy
Uncovered
Detected
Stealth
Undetected

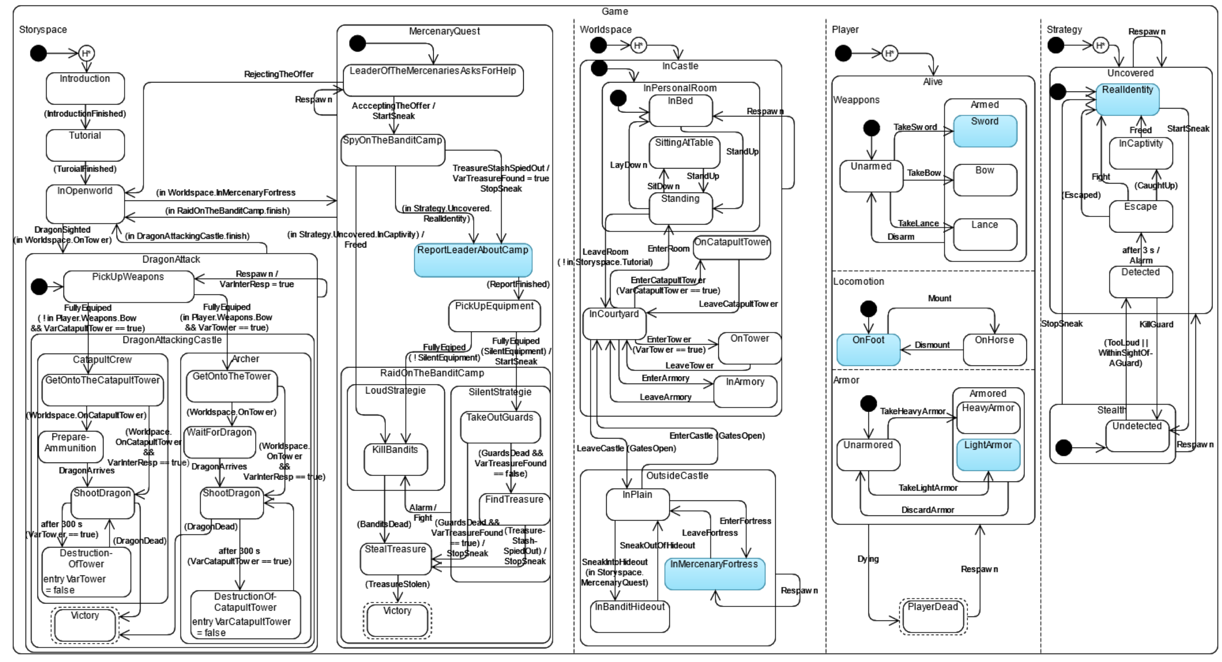


Game
Storyspace
Introduction
(IntroductionFinished)
Tutorial
(TurorialFinished)
InOpenworld
RejectingTheOffer
Respawn
(in Worldspace.InMercenaryFortress
(in RaidOnTheBanditCamp.finish)
DragonSighted
(in Worldspace.OnTower)
(in DragonAttackingCastle.finish)
DragonAttack
PickUpWeapons
Respawn / VarInterResp = true
FullyEquiped
DragonAttackingCastle
CatapultCrew
GetOntoTheCatapultTower
Prepare-Ammunition
DragonArrives
ShootDragon
(DragonDead)
Destruction-OfTower
entry VarTower = false
Victory
Archer
GetOntoTheTower
WaitForDragon
ShootDragon
(DragonDead)
DestructionOf-CatapultTower
entry VarCatapultTower = false
MercenaryQuest
LeaderOfTheMercenariesAsksForHelp
AcceptingTheOffer / StartSneak
SpyOnTheBanditCamp
TreasureStashSpiedOut / VarTreasureFound = true StopSneak
(in Strategy.Uncovered. RealIdentity)
(in Strategy.Uncovered.InCaptivity) / Freed
ReportLeaderAboutCamp
(ReportFinished)
PickUpEquipment
FullyEquiped
RaidOnTheBanditCamp
LoudStrategie
KillBandits
(BanditsDead)
Alarm / Fight
StealTreasure
(TreasureStolen)
Victory
SilentStrategie
TakeOutGuards
FindTreasure
Worldspace
InCastle
InPersonalRoom
InBed
SittingAtTable
Standing
LayDown
SitDown
StandUp
Respawn
LeaveRoom
EnterRoom
OnCatapultTower
EnterCatapultTower
LeaveCatapultTower
InCourtyard
EnterTower
OnTower
LeaveTower
EnterArmory
InArmory
LeaveArmory
EnterCastle (GatesOpen)
LeaveCastle (GatesOpen)
OutsideCastle
InPlain
EnterFortress
LeaveFortress
SneakOutOfHideout
SneakIntoHideout
InMercenaryFortress
InBanditHideout
Respawn
Player
Alive
Weappons
Armed
Sword
Bow
Lance
Unarmed
TakeSword
TakeBow
TakeLance
Disarm
Locomotion
Mount
OnFoot
Dismount
OnHorse
Armor
Armored
HeavyArmor
LightArmor
Unarmored
TakeHeavyArmor
TakeLightArmor
DiscardArmor
Dying
Respawn
PlayerDead
Strategy
Respawn
Uncovered
RealIdentity
Freed
StartSneak
InCaptivity
Fight
(Escaped)
(CaughtUp)
Escape
after 3 s / Alarm
Detected
StopSneak
KillGuard
(TooLoud || WithinSightOf-AGuard)
Stealth
Undetected
Respawn

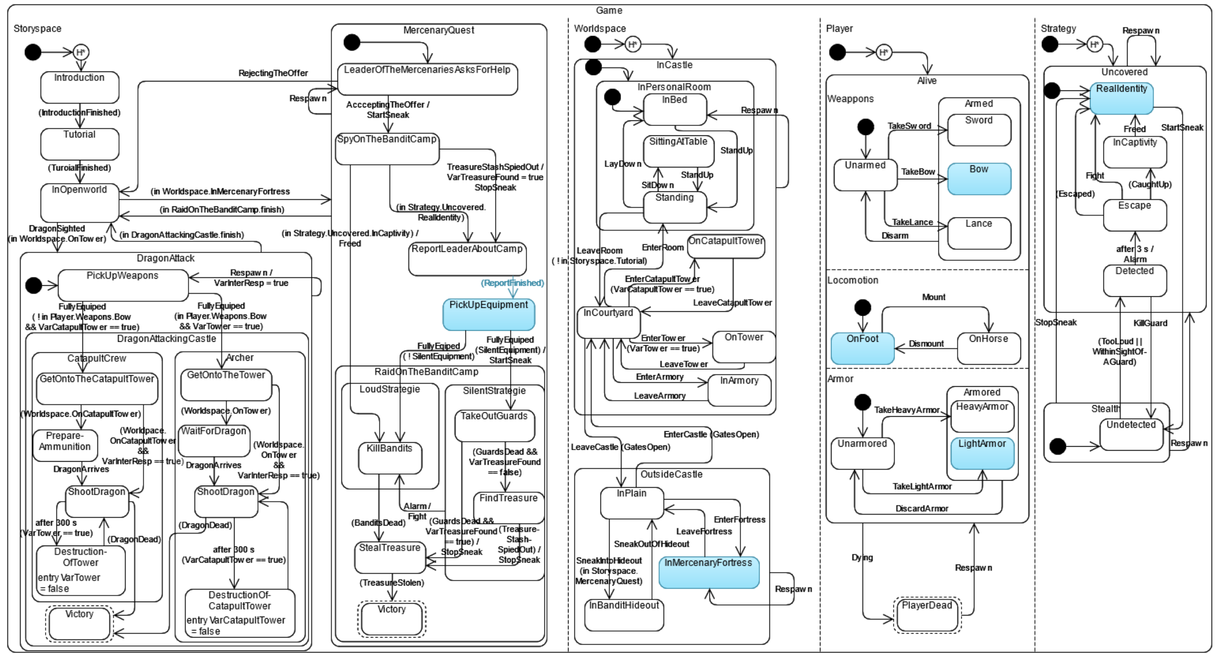




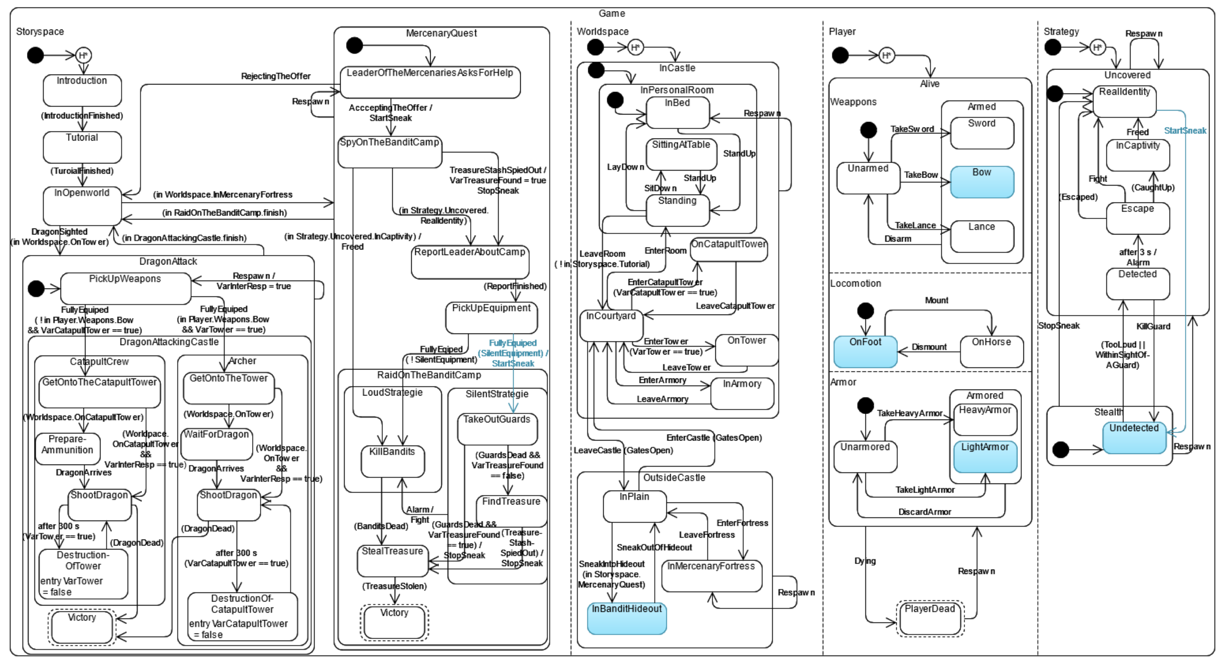
Game
Storyspace
Introduction
Tutorial
InOpenworld
DragonAttack
PickUpWeapons
DragonAttackingCastle
CatapultCrew
Archer
Victory
MercenaryQuest
LeaderOfTheMercenariesAsksForHelp
SpyOnTheBanditCamp
ReportLeaderAboutCamp
PickUpEquipment
RaidOnTheBanditCamp
Worldspace
InCastle
InPersonalRoom
InCourtyard
OutsideCastle
InPlain
InMercenaryFortress
InBanditHideout
Player
Alive
Weappons
Locomotion
OnFoot
OnHorse
Armor
PlayerDead
Strategy
Uncovered
RealIdentity
InCaptivity
Escape
Detected
Stealth
Undetected



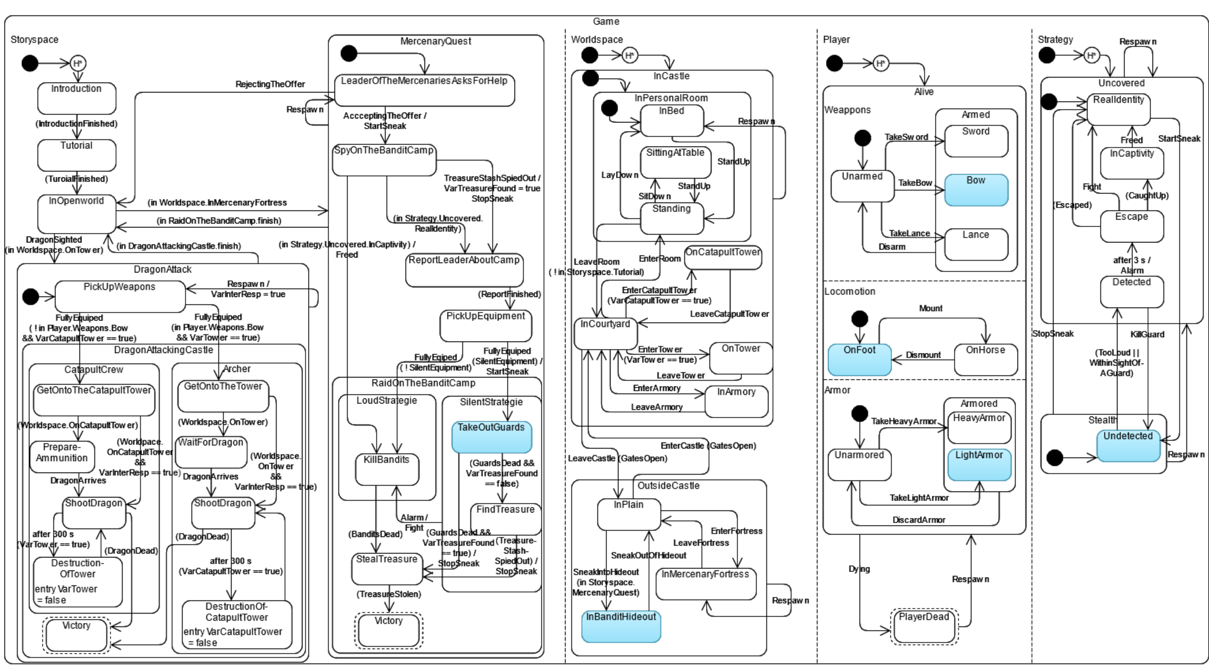
Game
Storyspace
Introduction
(IntroductionFinished)
Tutorial
(TurorialFinished)
InOpenworld
RejectingTheOffer
Respawn
MercenaryQuest
LeaderOfTheMercenariesAsksForHelp
Acccepting TheOffer / StartSneak
SpyOnTheBanditCamp
TreasureStashSpiedOut / VarTreasureFound = true StopSneak
(in Worldspace.InMercenaryFortress)
(in RaidOnTheBanditCamp.finish)
(in Strategy.Uncovered. RealIdentity)
(in Strategy.Uncovered.InCaptivity) / Freed
DragonSighted (in Worldspace.OnTower)
(in DragonAttackingCastle.finish)
ReportLeaderAboutCamp
(ReportFinished)
PickUpEquipment
DragonAttack
PickUpWeapons
Respawn / VarInterResp = true
FullyEquiped (! in Player.Weapons.Bow && VarCatapultTower == true)
FullyEquiped (in Player.Weapons.Bow && VarTower == true)
DragonAttackingCastle
CatapultCrew
Archer
GetOntoTheCatapultTower
GetOntoTheTower
(Worldspace.OnCatapultTower)
(Worldspace.OnTower)
Prepare-Ammunition
WaitForDragon
(Worldspace. OnCatapultTower && VarInterResp == true)
(Worldspace. OnTower && VarInterResp == true)
DragonArrives
ShootDragon
after 300 s (VarTower == true)
(DragonDead)
Destruction-OfTower
entry VarTower = false
after 300 s (VarCatapultTower == true)
DestructionOf-CatapultTower
entry VarCatapultTower = false
Victory
FullyEqiped (! SilentEquipment)
FullyEquiped (SilentEquipment) / StartSneak
RaidOnTheBanditCamp
LoudStrategie
SilentStrategie
KillBandits
TakeOutGuards
(GuardsDead && VarTreasureFound == false)
FindTreasure
Alarm / Fight
(BanditsDead)
(GuardsDead && VarTreasureFound == true) / StopSneak
(Treasure-Stash-SpiedOut) / StopSneak
StealTreasure
(TreasureStolen)
Victory
Worldspace
InCastle
InPersonalRoom
InBed
Respawn
SittingAtTable
StandUp
LayDown
SitDown
StandUp
Standing
LeaveRoom (! in Storyspace.Tutorial)
EnterRoom
OnCatapultTower
EnterCatapultTower (VarCatapultTower == true)
LeaveCatapultTower
InCourtyard
EnterTower (VarTower == true)
OnTower
LeaveTower
EnterArmory
InArmory
LeaveArmory
EnterCastle (GatesOpen)
LeaveCastle (GatesOpen)
OutsideCastle
InPlain
EnterFortress
LeaveFortress
SneakOutOfHideout
SneakIntoHideout (in Storyspace. MercenaryQuest)
InMercenaryFortress
Respawn
InBanditHideout
Player
Alive
Weappons
Armed
TakeSword
Sword
Unarmed
TakeBow
Bow
TakeLance
Lance
Disarm
Locomotion
Mount
OnFoot
Dismount
OnHorse
Armor
Armored
TakeHeavyArmor
HeavyArmor
Unarmored
LightArmor
TakeLightArmor
DiscardArmor
Dying
Respawn
PlayerDead
Strategy
Respawn
Uncovered
RealIdentity
Freed
StartSneak
InCaptivity
Fight
(CaughtUp)
(Escaped)
Escape
after 3 s / Alarm
Detected
StopSneak
KillGuard
(ToLoud || WithinSightOf-AGuard)
Stealth
Undetected
Respawn



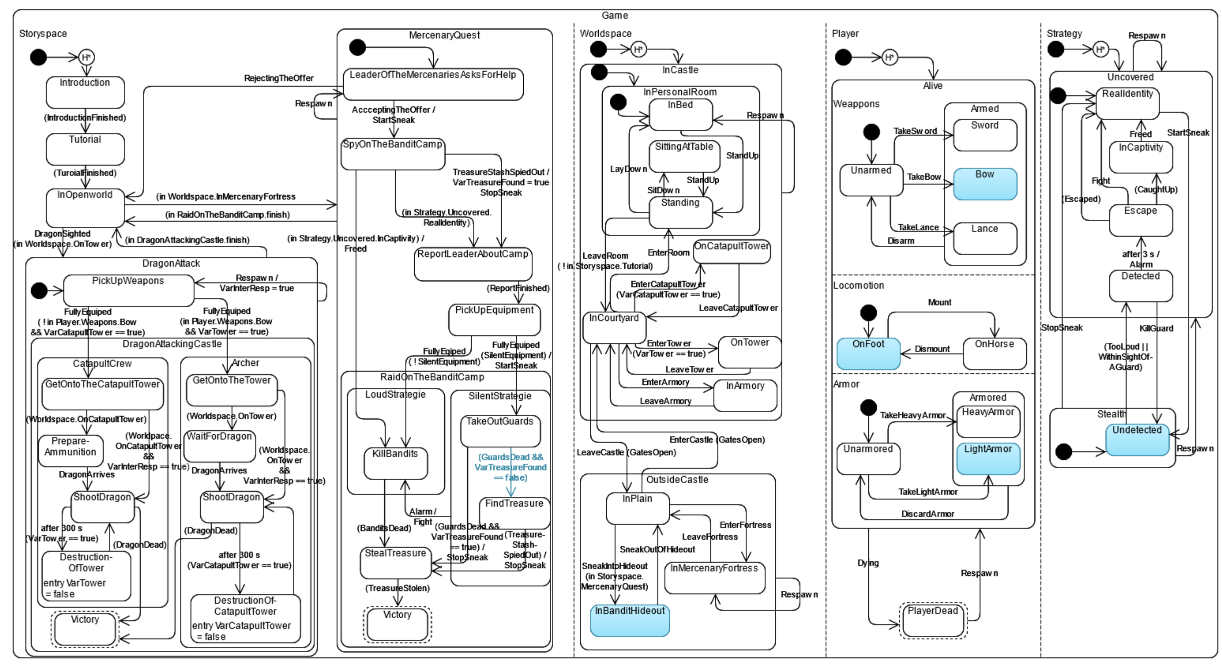
Game
Storyspace
Introduction
(IntroductionFinished)
Tutorial
(TurorialFinished)
InOpenworld
RejectingTheOffer
Respawn
MercenaryQuest
LeaderOfTheMercenariesAsksForHelp
Acccepting TheOffer / StartSneak
SpyOnTheBanditCamp
TreasureStashSpiedOut / VarTreasureFound = true StopSneak
(in Worldspace.InMercenaryFortress)
(in RaidOnTheBanditCamp.finish)
(in Strategy.Uncovered. RealIdentity)
(in Strategy.Uncovered.InCaptivity) / Freed
DragonSighted (in Worldspace.OnTower)
(in DragonAttackingCastle.finish)
ReportLeaderAboutCamp
(ReportFinished)
PickUpEquipment
DragonAttack
PickUpWeapons
Respawn / VarInterResp = true
FullyEquiped (! in Player.Weapons.Bow && VarCatapultTower == true)
FullyEquiped (in Player.Weapons.Bow && VarTower == true)
DragonAttackingCastle
CatapultCrew
Archer
GetOntoTheCatapultTower
GetOntoTheTower
(Worldspace.OnCatapultTower)
(Worldspace.OnTower)
Prepare-Ammunition
WaitForDragon
(Worldspace. OnCatapultTower && VarInterResp == true)
(Worldspace. OnTower && VarInterResp == true)
DragonArrives
ShootDragon
after 300 s (VarTower == true)
(DragonDead)
Destruction-OfTower
entry VarTower = false
after 300 s (VarCatapultTower == true)
DestructionOf-CatapultTower
entry VarCatapultTower = false
Victory
FullyEqiped (! SilentEquipment)
FullyEquiped (SilentEquipment) / StartSneak
RaidOnTheBanditCamp
LoudStrategie
SilentStrategie
KillBandits
TakeOutGuards
(GuardsDead && VarTreasureFound == false)
FindTreasure
Alarm / Fight
(BanditsDead)
(GuardsDead && VarTreasureFound == true) / StopSneak
(Treasure-Stash-SpiedOut) / StopSneak
StealTreasure
(TreasureStolen)
Victory
Worldspace
InCastle
InPersonalRoom
InBed
Respawn
SittingAtTable
StandUp
LayDown
SitDown
StandUp
Standing
LeaveRoom (! in Storyspace.Tutorial)
EnterRoom
OnCatapultTower
EnterCatapultTower (VarCatapultTower == true)
LeaveCatapultTower
InCourtyard
EnterTower (VarTower == true)
OnTower
LeaveTower
EnterArmory
InArmory
LeaveArmory
EnterCastle (GatesOpen)
LeaveCastle (GatesOpen)
OutsideCastle
InPlain
EnterFortress
LeaveFortress
SneakOutOfHideout
SneakIntoHideout (in Storyspace. MercenaryQuest)
InMercenaryFortress
Respawn
InBanditHideout
Player
Alive
Weappons
Armed
TakeSword
Sword
Unarmed
TakeBow
Bow
TakeLance
Lance
Disarm
Locomotion
Mount
OnFoot
Dismount
OnHorse
Armor
Armored
TakeHeavyArmor
HeavyArmor
Unarmored
LightArmor
TakeLightArmor
DiscardArmor
Dying
Respawn
PlayerDead
Strategy
Respawn
Uncovered
RealIdentity
Freed
StartSneak
InCaptivity
Fight
(CaughtUp)
(Escaped)
Escape
after 3 s / Alarm
Detected
StopSneak
KillGuard
(ToLoud || WithinSightOf-AGuard)
Stealth
Undetected
Respawn

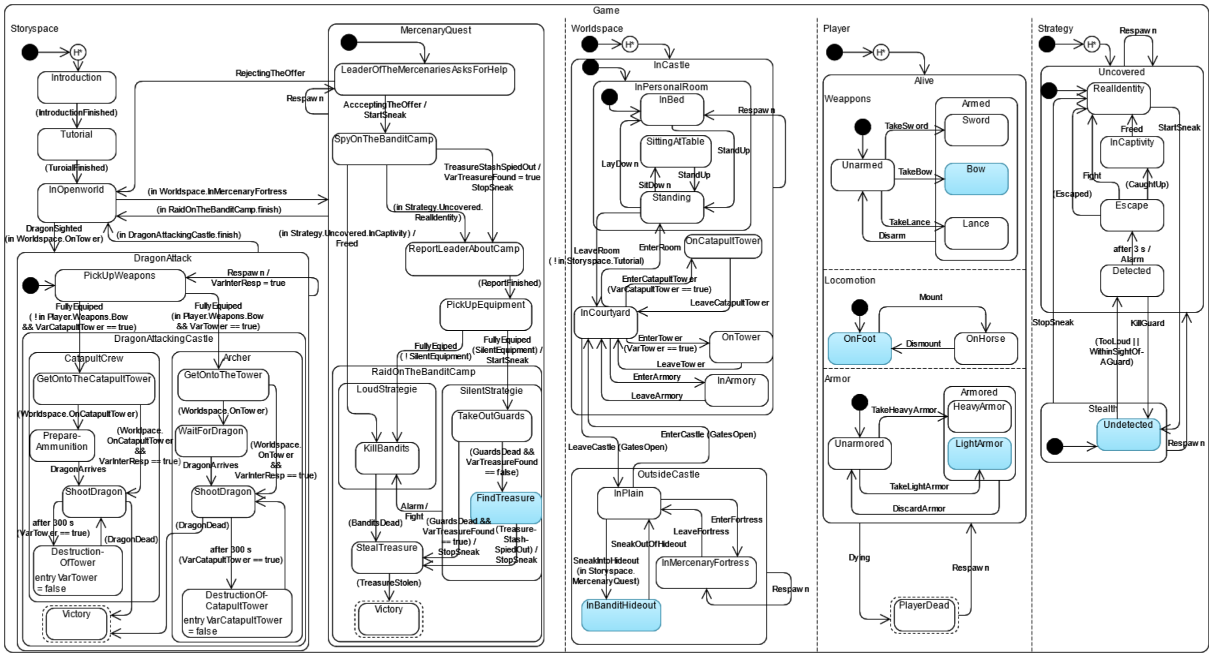


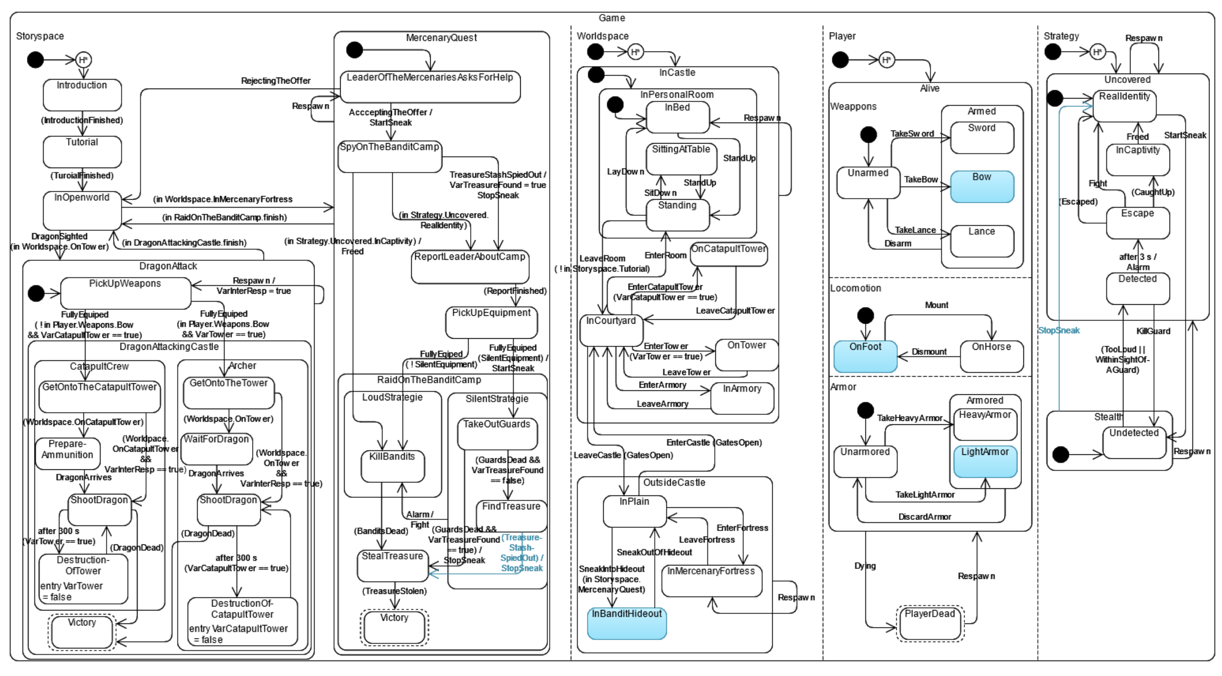
Game
Storyspace
H*
Introduction
(IntroductionFinished)
Tutorial
(TurotialFinished)
InOpenworld
(in Worldspace.InMercenaryFortress
(in RaidOnTheBanditCamp.finish)
(in DragonAttackingCastle.finish)
DragonSighted
(in Worldspace.OnTower)
RejectingTheOffer
Respawn
DragonAttack
PickUpWeapons
Respawn / VarInterResp = true
FullyEquiped
(! in Player.Weapons.Bow
&& VarCatapultTower == true)
FullyEquiped
(in Player.Weapons.Bow
&& VarTower == true)
DragonAttackingCastle
CatapultCrew
GetOntoTheCatapultTower
(Worldspace.OnCatapultTower)
Prepare-Ammunition
(Worldspace.OnCatapultTower && VarInterResp == true)
DragonArrives
ShootDragon
after 300 s (VarTower == true)
(DragonDead)
Destruction-OfTower
entry VarTower = false
Victory
Archer
GetOntoTheTower
(Worldspace.OnTower)
WaitForDragon
(Worldspace.OnTower && VarInterResp == true)
DragonArrives
ShootDragon
(DragonDead)
after 300 s (VarCatapultTower == true)
DestructionOf-CatapultTower
entry VarCatapultTower = false
MercenaryQuest
LeaderOfTheMercenariesAsksForHelp
Respawn
AcceptingTheOffer / StartSneak
SpyOnTheBanditCamp
TreasureStashSpiedOut / VarTreasureFound = true StopSneak
(in Strategy.Uncovered.RealIdentity)
(in Strategy.Uncovered.InCaptivity) / Freed
ReportLeaderAboutCamp
(ReportFinished)
PickUpEquipment
FullyEquiped (! SilentEquipment)
FullyEquiped (SilentEquipment) / StartSneak
RaidOnTheBanditCamp
LoudStrategie
KillBandits
SilentStrategie
TakeOutGuards
(GuardsDead && VarTreasureFound == false)
FindTreasure
Alarm / Fight
(BanditsDead)
(GuardsDead && VarTreasureFound == true) / StopSneak
(Treasure-Stash-SpiedOut) / StopSneak
StealTreasure
(TreasureStolen)
Victory
Worldspace
InCastle
InPersonalRoom
InBed
Respawn
SittingAtTable
StandUp
LayDown
SitDown
StandUp
Standing
LeaveRoom
(! in Storyspace.Tutorial)
EnterRoom
OnCatapultTower
EnterCatapultTower (VarCatapultTower == true)
LeaveCatapultTower
InCourtyard
EnterTower (VarTower == true)
OnTower
LeaveTower
EnterArmory
InArmory
LeaveArmory
LeaveCastle (GatesOpen)
EnterCastle (GatesOpen)
OutsideCastle
InPlain
EnterFortress
LeaveFortress
SneakOutOfHideout
SneakIntoHideout (in Storyspace.MercenaryQuest)
InMercenaryFortress
InBanditHideout
Respawn
Player
Alive
Weappons
Armed
TakeSword
Sword
Unarmed
TakeBow
Bow
TakeLance
Lance
Disarm
Locomotion
Mount
OnFoot
Dismount
OnHorse
Armor
Armored
TakeHeavyArmor
HeavyArmor
Unarmored
LightArmor
TakeLightArmor
DiscardArmor
Dying
Respawn
PlayerDead
Strategy
Respawn
Uncovered
RealIdentity
Freed
StartSneak
InCaptivity
Fight
(CaughtUp)
(Escaped)
Escape
after 3 s / Alarm
Detected
StopSneak
KillGuard
(TooLoud || WithinSightOf-AGuard)
Stealth
Undetected
Respawn



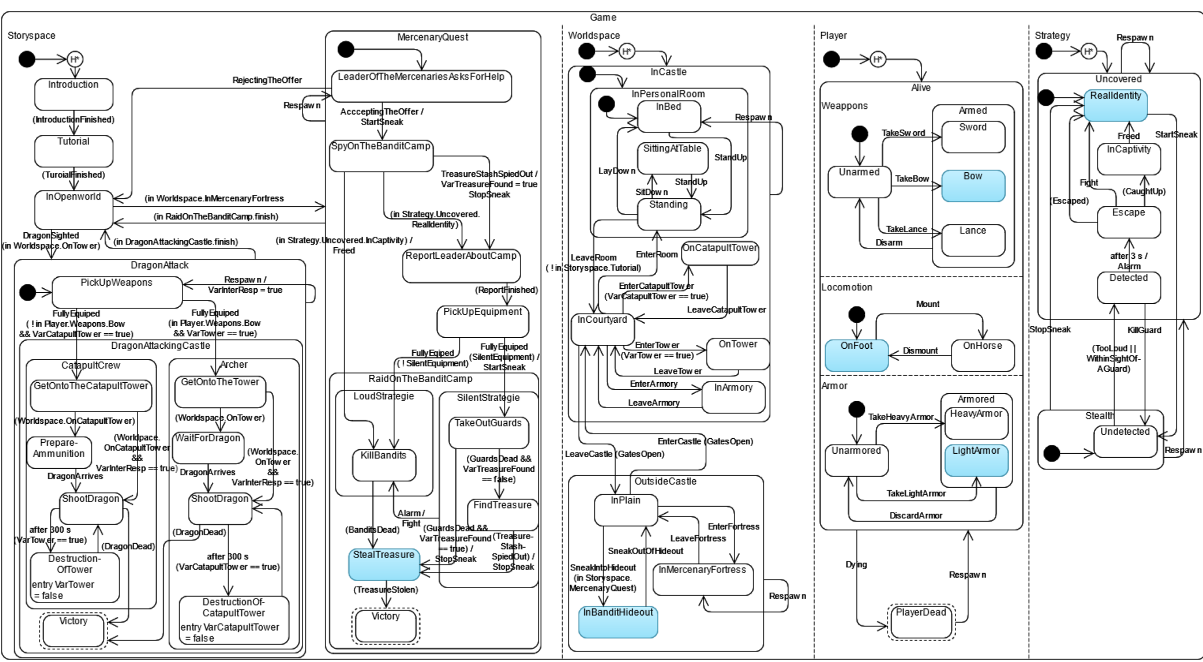
Game
Storyspace
H*
Introduction
(IntroductionFinished)
Tutorial
(TurotialFinished)
InOpenworld
(in Worldspace.InMercenaryFortress
(in RaidOnTheBanditCamp.finish)
(in DragonAttackingCastle.finish)
DragonSighted
(in Worldspace.OnTower)
RejectingTheOffer
Respawn
DragonAttack
PickUpWeapons
Respawn / VarInterResp = true
FullyEquiped
(! in Player.Weapons.Bow
&& VarCatapultTower == true)
FullyEquiped
(in Player.Weapons.Bow
&& VarTower == true)
DragonAttackingCastle
CatapultCrew
GetOntoTheCatapultTower
(Worldspace.OnCatapultTower)
Prepare-Ammunition
(Worldspace.OnCatapultTower && VarInterResp == true)
DragonArrives
ShootDragon
after 300 s (VarTower == true)
(DragonDead)
Destruction-OfTower
entry VarTower = false
Victory
Archer
GetOntoTheTower
(Worldspace.OnTower)
WaitForDragon
(Worldspace.OnTower && VarInterResp == true)
DragonArrives
ShootDragon
(DragonDead)
after 300 s (VarCatapultTower == true)
DestructionOf-CatapultTower
entry VarCatapultTower = false
MercenaryQuest
LeaderOfTheMercenariesAsksForHelp
Respawn
AcceptingTheOffer / StartSneak
SpyOnTheBanditCamp
TreasureStashSpiedOut / VarTreasureFound = true StopSneak
(in Strategy.Uncovered.RealIdentity)
(in Strategy.Uncovered.InCaptivity) / Freed
ReportLeaderAboutCamp
(ReportFinished)
PickUpEquipment
FullyEquiped (! SilentEquipment)
FullyEquiped (SilentEquipment) / StartSneak
RaidOnTheBanditCamp
LoudStrategie
KillBandits
SilentStrategie
TakeOutGuards
(GuardsDead && VarTreasureFound == false)
FindTreasure
Alarm / Fight
(BanditsDead)
(GuardsDead && VarTreasureFound == true) / StopSneak
(Treasure-Stash-SpiedOut) / StopSneak
StealTreasure
(TreasureStolen)
Victory
Worldspace
InCastle
InPersonalRoom
InBed
Respawn
SittingAtTable
StandUp
LayDown
SitDown
StandUp
Standing
LeaveRoom
(! in Storyspace.Tutorial)
EnterRoom
OnCatapultTower
EnterCatapultTower (VarCatapultTower == true)
LeaveCatapultTower
InCourtyard
EnterTower (VarTower == true)
OnTower
LeaveTower
EnterArmory
InArmory
LeaveArmory
LeaveCastle (GatesOpen)
EnterCastle (GatesOpen)
OutsideCastle
InPlain
EnterFortress
LeaveFortress
SneakOutOfHideout
SneakIntoHideout (in Storyspace.MercenaryQuest)
InMercenaryFortress
InBanditHideout
Respawn
Player
Alive
Weappons
Armed
TakeSword
Sword
Unarmed
TakeBow
Bow
TakeLance
Lance
Disarm
Locomotion
Mount
OnFoot
Dismount
OnHorse
Armor
Armored
TakeHeavyArmor
HeavyArmor
Unarmored
LightArmor
TakeLightArmor
DiscardArmor
Dying
Respawn
PlayerDead
Strategy
Respawn
Uncovered
RealIdentity
Freed
StartSneak
InCaptivity
Fight
(CaughtUp)
(Escaped)
Escape
after 3 s / Alarm
Detected
StopSneak
KillGuard
(TooLoud || WithinSightOf-AGuard)
Stealth
Undetected
Respawn


Game
Storyspace
Introduction
(IntroductionFinished)
Tutorial
(TurorialFinished)
InOpenworld
RejectingTheOffer
Respawn
(in Worldspace.InMercenaryFortress
(in RaidOnTheBanditCamp.finish)
DragonSighted
(in Worldspace.OnTower)
(in DragonAttackingCastle.finish)
DragonAttack
PickUpWeapons
Respawn / VarInterResp = true
DragonAttackingCastle
CatapultCrew
GetOntoTheCatapultTower
Prepare-Ammunition
ShootDragon
Destruction-OfTower
entry VarTower = false
Victory
Archer
GetOntoTheTower
WaitForDragon
DragonArrives
(DragonDead)
DestructionOf-CatapultTower
entry VarCatapultTower = false
MercenaryQuest
LeaderOfTheMercenariesAsksForHelp
AccceptingTheOffer / StartSneak
SpyOnTheBanditCamp
TreasureStashSpiedOut / VarTreasureFound = true StopSneak
(in Strategy.Uncovered.RealIdentity)
(in Strategy.Uncovered.InCaptivity) / Freed
ReportLeaderAboutCamp
(ReportFinished)
PickUpEquipment
RaidOnTheBanditCamp
LoudStrategie
KillBandits
SilentStrategie
TakeOutGuards
FindTreasure
StealTreasure
(TreasureStolen)
Victory
Worldspace
InCastle
InPersonalRoom
InBed
SittingAtTable
Standing
StandUp
SitDown
LayDown
OnCatapultTower
InCourtyard
OnTower
InArmory
EnterRoom
LeaveRoom
EnterCatapultTower
LeaveCatapultTower
EnterTower
LeaveTower
EnterArmory
LeaveArmory
LeaveCastle (GatesOpen)
EnterCastle (GatesOpen)
OutsideCastle
InPlain
EnterFortress
LeaveFortress
InMercenaryFortress
SneakOutOfHideout
SneakIntoHideout
InBanditHideout
Player
Alive
Weappons
Unarmed
Armed
Sword
Bow
Lance
TakeSword
TakeBow
TakeLance
Disarm
Locomotion
OnFoot
OnHorse
Mount
Dismount
Armor
Unarmored
Armored
HeavyArmor
LightArmor
TakeHeavyArmor
TakeLightArmor
DiscardArmor
Dying
Respawn
PlayerDead
Strategy
Uncovered
RealIdentity
InCaptivity
Escape
Detected
Freed
StartSneak
Fight
(CaughtUp)
(Escaped)
after 3 s / Alarm
StopSneak
KillGuard
(TooLoud || WithinSightOf-AGuard)
Stealth
Undetected

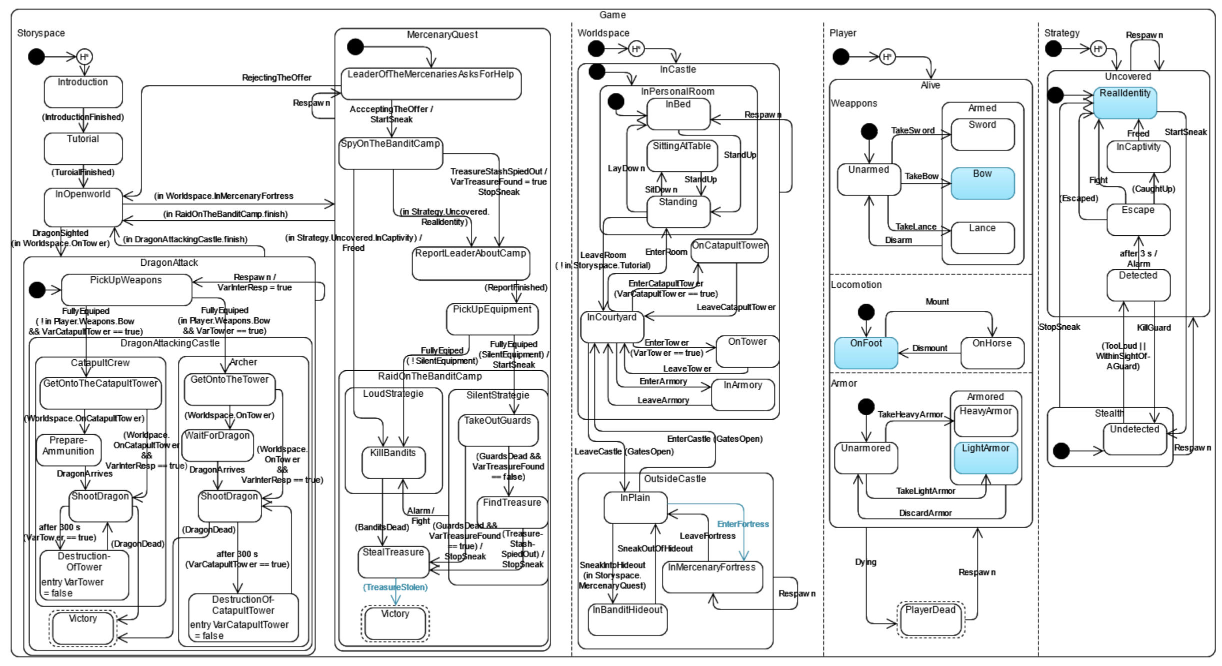




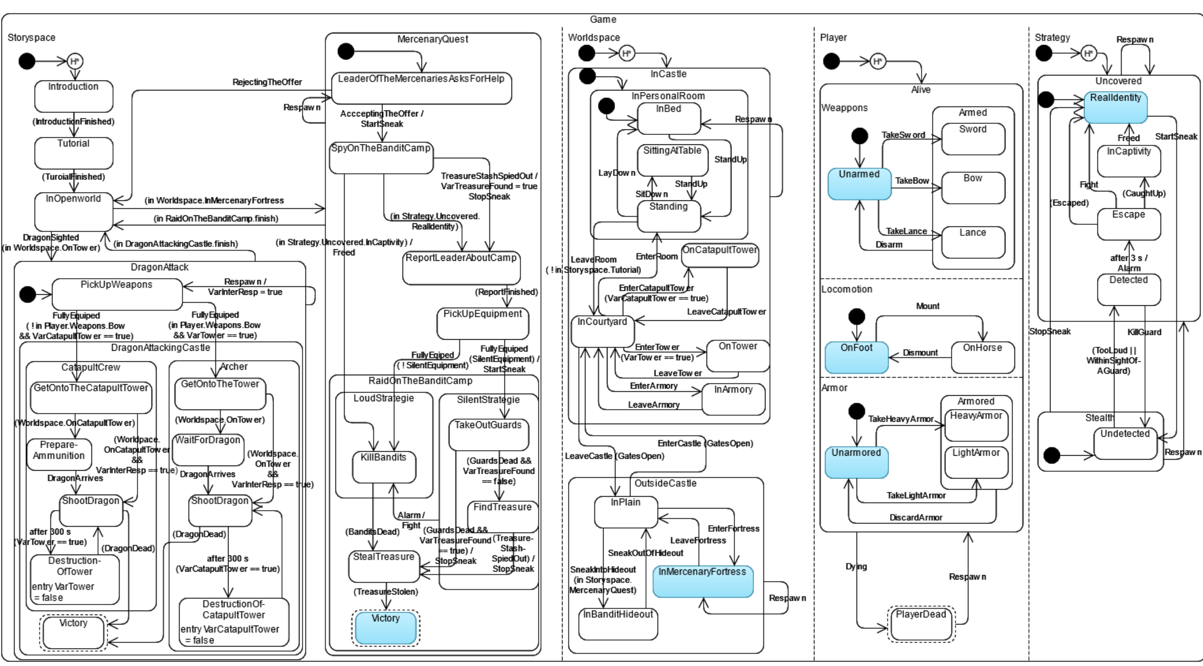
Game
Storyspace
Introduction
(IntroductionFinished)
Tutorial
(TuroialFinished)
InOpenworld
RejectingTheOffer
Respawn
(in Worldspace.InMercenaryFortress
(in RaidOnTheBanditCamp.finish)
DragonSighted
(in Worldspace.OnTower)
(in DragonAttackingCastle.finish)
DragonAttack
PickUpWeapons
Respawn / VarInterResp = true
FullyEquiped
(! in Player.Weapons.Bow && VarCatapultTower == true)
FullyEquiped
(in Player.Weapons.Bow && VarTower == true)
DragonAttackingCastle
CatapultCrew
GetOntoTheCatapultTower
(Worldspace.OnCatapultTower)
Prepare-Ammunition
DragonArrives
ShootDragon
after 300 s (VarTower == true)
(DragonDead)
Destruction-OfTower
entry VarTower = false
Victory
Archer
GetOntoTheTower
(Worldspace.OnTower)
WaitForDragon
DragonArrives
ShootDragon
(DragonDead)
after 300 s (VarCatapultTower == true)
DestructionOf-CatapultTower
entry VarCatapultTower = false
MercenaryQuest
LeaderOfTheMercenariesAsksForHelp
AccceptingTheOffer / StartSneak
SpyOnTheBanditCamp
TreasureStashSpiedOut / VarTreasureFound = true StopSneak
(in Strategy.Uncovered.RealIdentity)
(in Strategy.Uncovered.InCaptivity) / Freed
ReportLeaderAboutCamp
(ReportFinished)
PickUpEquipment
FullyEquiped (SilentEquipment) / StartSneak
RaidOnTheBanditCamp
LoudStrategie
KillBandits
SilentStrategie
TakeOutGuards
(GuardsDead && VarTreasureFound == false)
FindTreasure
Alarm / Fight
(BanditsDead)
StealTreasure
(TreasureStolen)
Victory
Worldspace
InCastle
InPersonalRoom
InBed
SittingAtTable
StandUp
LayDown
SitDown
Standing
EnterRoom
LeaveRoom
(! in Storyspace.Tutorial)
OnCatapultTower
EnterCatapultTower (VarCatapultTower == true)
LeaveCatapultTower
InCourtyard
EnterTower (VarTower == true)
OnTower
LeaveTower
EnterArmory
InArmory
LeaveArmory
EnterCastle (GatesOpen)
LeaveCastle (GatesOpen)
OutsideCastle
InPlain
EnterFortress
LeaveFortress
SneakOutOfHideout
SneakIntoHideout (in Storyspace.MercenaryQuest)
InMercenaryFortress
InBanditHideout
Player
Alive
Weappons
Armed
Sword
TakeSword
Unarmed
TakeBow
Bow
TakeLance
Lance
Disarm
Locomotion
Mount
OnFoot
Dismount
OnHorse
Armor
Armored
TakeHeavyArmor
HeavyArmor
Unarmored
LightArmor
TakeLightArmor
DiscardArmor
Dying
Respawn
PlayerDead
Strategy
Uncovered
RealIdentity
Freed
StartSneak
InCaptivity
Fight
(CaughtUp)
(Escaped)
Escape
after 3 s / Alarm
Detected
StopSneak
KillGuard
(TooLoud || WithinSightOf-AGuard)
Stealth
Undetected



Game
Storyspace
Introduction
(IntroductionFinished)
Tutorial
(TuroialFinished)
InOpenworld
RejectingTheOffer
Respawn
(in Worldspace.InMercenaryFortress
(in RaidOnTheBanditCamp.finish)
DragonSighted
(in Worldspace.OnTower)
(in DragonAttackingCastle.finish)
DragonAttack
PickUpWeapons
Respawn / VarInterResp = true
DragonAttackingCastle
CatapultCrew
GetOntoTheCatapultTower
Prepare-Ammunition
ShootDragon
Destruction-OfTower
Victory
Archer
GetOntoTheTower
WaitForDragon
ShootDragon
DestructionOf-CatapultTower
MercenaryQuest
LeaderOfTheMercenariesAsksForHelp
SpyOnTheBanditCamp
ReportLeaderAboutCamp
PickUpEquipment
RaidOnTheBanditCamp
LoudStrategie
KillBandits
SilentStrategie
TakeOutGuards
FindTreasure
StealTreasure
Victory
Worldspace
InCastle
InPersonalRoom
InBed
SittingAtTable
Standing
OnCatapultTower
InCourtyard
OnTower
InArmory
OutsideCastle
InPlain
InMercenaryFortress
InBanditHideout
Player
Alive
Weappons
Armed
Sword
Unarmed
Bow
Lance
Locomotion
OnFoot
OnHorse
Armor
Armored
HeavyArmor
Unarmored
LightArmor
PlayerDead
Strategy
Uncovered
RealIdentity
InCaptivity
Escape
Detected
Stealth
Undetected



Game
Storyspace
Introduction
(IntroductionFinished)
Tutorial
(TuroialFinished)
InOpenworld
RejectingTheOffer
Respawn
(in Worldspace.InMercenaryFortress)
(in RaidOnTheBanditCamp.finish)
DragonSighted
(in Worldspace.OnTower)
(in DragonAttackingCastle.finish)
(in Strategy.Uncovered.InCaptivity) /
Freed
DragonAttack
PickUpWeapons
Respawn /
VarInterResp = true
DragonAttackingCastle
CatapultCrew
Archer
GetOntoTheCatapultTower
GetOntoTheTower
Prepare-
Ammunition
DragonArrives
WaitForDragon
ShootDragon
(DragonDead)
Destruction-
OfTower
entry VarTower
= false
DestructionOf-
CatapultTower
entry VarCatapultTower
= false
Victory
MercenaryQuest
LeaderOfTheMercenariesAsksForHelp
AcceptingTheOffer /
StartSneak
SpyOnTheBanditCamp
TreasureStashSpiedOut /
VarTreasureFound = true
StopSneak
(in Strategy.Uncovered.
RealIdentity)
ReportLeaderAboutCamp
(ReportFinished)
PickUpEquipment
RaidOnTheBanditCamp
LoudStrategie
SilentStrategie
KillBandits
TakeOutGuards
FindTreasure
(BanditsDead)
Alarm /
Fight
StealTreasure
(TreasureStolen)
Worldspace
InCastle
InPersonalRoom
InBed
SittingAtTable
Standing
StandUp
LayDown
SitDown
LeaveRoom
EnterRoom
OnCatapultTower
EnterCatapultTower
LeaveCatapultTower
InCourtyard
EnterTower
OnTower
LeaveTower
EnterArmory
InArmory
LeaveArmory
EnterCastle (GatesOpen)
LeaveCastle (GatesOpen)
OutsideCastle
InPlain
EnterFortress
LeaveFortress
SneakOutOfHideout
SneakIntoHideout
InMercenaryFortress
InBanditHideout
Player
Alive
Weappons
Armed
Sword
Bow
Lance
Unarmed
TakeSword
TakeBow
TakeLance
Disarm
Locomotion
Mount
OnFoot
Dismount
OnHorse
Armor
Armored
HeavyArmor
LightArmor
Unarmored
TakeHeavyArmor
TakeLightArmor
DiscardArmor
Dying
PlayerDead
Strategy
Uncovered
RealIdentity
Freed
StartSneak
InCaptivity
Fight
(Escaped)
(CaughtUp)
Escape
after 3 s /
Alarm
Detected
StopSneak
KillGuard
Stealth
Undetected



Game
Storyspace
Introduction
Tutorial
InOpenworld
DragonAttack
PickUpWeapons
DragonAttackingCastle
MercenaryQuest
LeaderOfTheMercenariesAsksForHelp
SpyOnTheBanditCamp
ReportLeaderAboutCamp
PickUpEquipment
RaidOnTheBanditCamp
StealTreasure
(TreasureStolen)
Victory
Worldspace
InCastle
InCourtyard
OutsideCastle
InPlain
InMercenaryFortress
InBanditHideout
Player
Alive
Unarmed
OnFoot
Unarmored
PlayerDead
Strategy
Uncovered
RealIdentity
Stealth
Undetected

Respawn – Quest: