ETH Zürich, Software Architecture
Projektarbeit von Adrian Rabenseifner, Fabrizio Steiner,
Ralph Wiedemeier und Philipp Wolfensperger

Developer Documentation

Beschreibung der Engine

Die XAE Engine ist verantwortlich für die Spiellogik und entspricht im MVC Pattern dem Modell. Die Interaktion mit dem GUI verläuft über die Features, welche in ENGINE_INTERFACE deklariert sind. Die aktive Rolle im Spiel wird jedoch vom GUI Übernommen, welches der Engine Events zur Bearbeitung gibt und jeweils bestimmte Flags abfragt und die Szenen entsprechend neu zeichnet.

Ein Teil der Engine ist vorwiegend eine komplexe Datenstruktur welche die ganze virtuelle Spielwelt repräsentiert, so gibt es naturgemäss Klassen für alle Gegenstände die im Spiel vorkommen: GAME, SCENE, AREA (Klickbereich), ITEM (Spielgegenstand), CLOSEUP (Auswahlfenster für Dialoge mit Personen etc.).

Diese Datenstruktur wird vom Parser generiert und verlinkt und widerspiegelt das Gerüst eines Spiels.

Ein anderer Teil der Engine kümmert sich um den Spielzustand: Im Wesentlichen sind das die Klassen REGISTRY und REPOSITORY.

In der Registry werden bestimmte Flags gesetzt, während im Repository aufgelesene Gegenstände aufbewahrt werden.

Ein dritter Teil der Engine befasst sich mit Event Handling. So gibt es ACTION Klassen für alle ausführbaren Aktionen, welche den Items und Klickbereichen zugeordnet werden können. Erfolgt ein Klick auf so einen Bereich, sendet das GUI der Engine ein EVENT Objekt, welches dann behandelt wird.

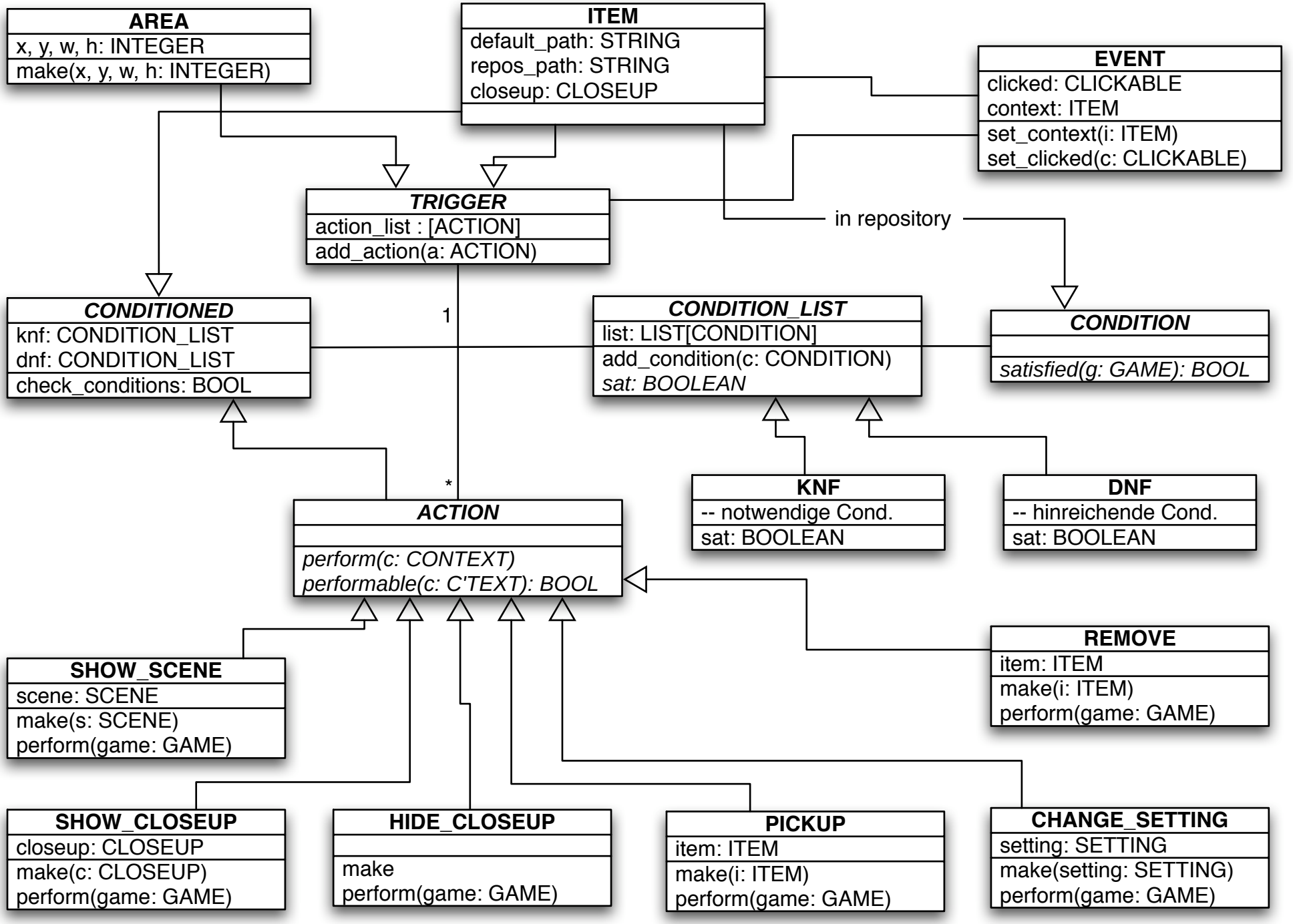
Dabei werden alle Aktionen ausgeführt, welche auf dem angeklickten Gegenstand in der Action-List eingetragen sind.

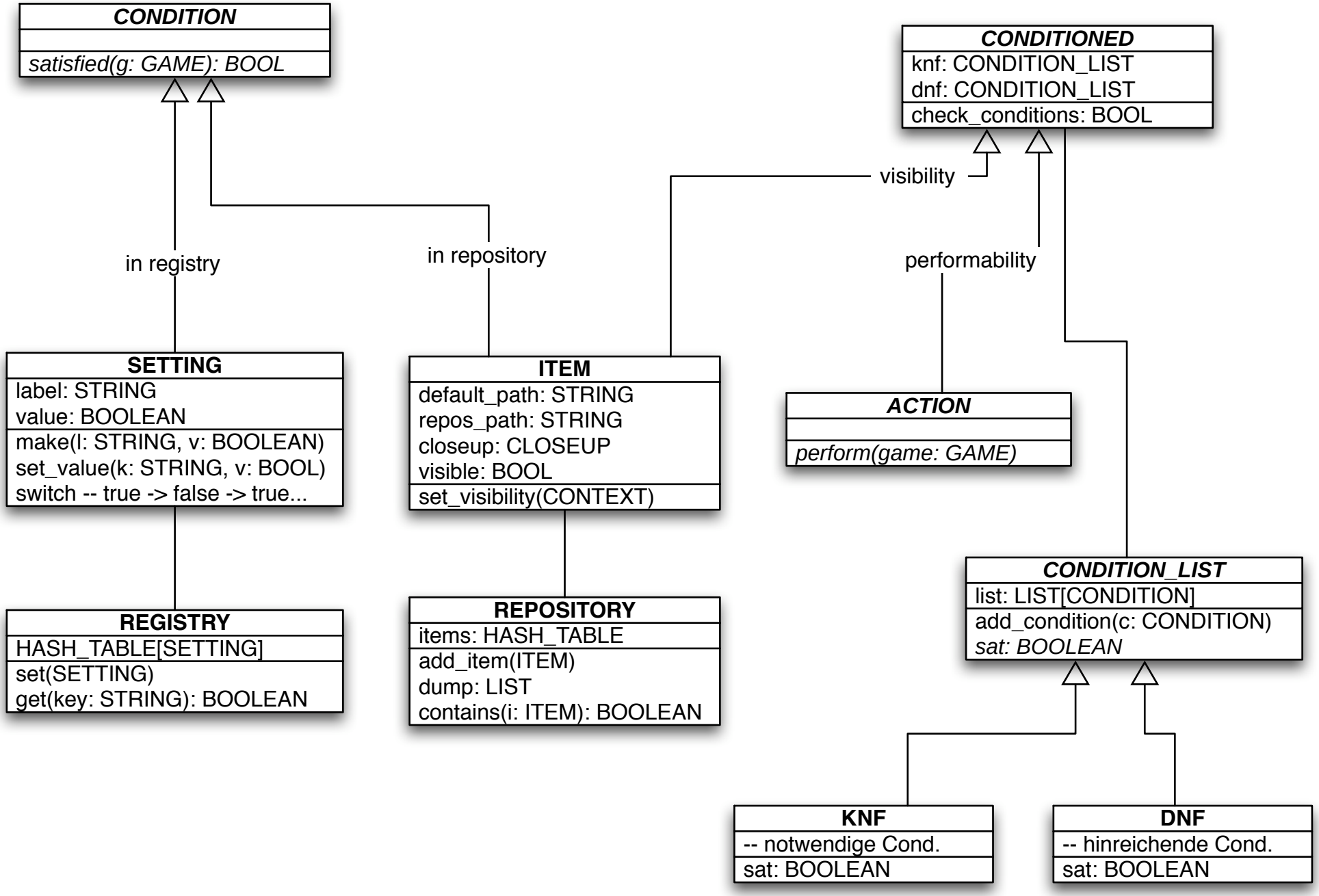
Damit jedoch eine gewisse Spiellogik möglich ist, verfügt die Engine über eine Conditional Architektur. So kännen bestimmte Bedingungen erfüllt werden und in 1KNF und 1DNF den Aktionen zugewiesen werden. Sichtbarkeit von Gegenständen wird analog gelöst.

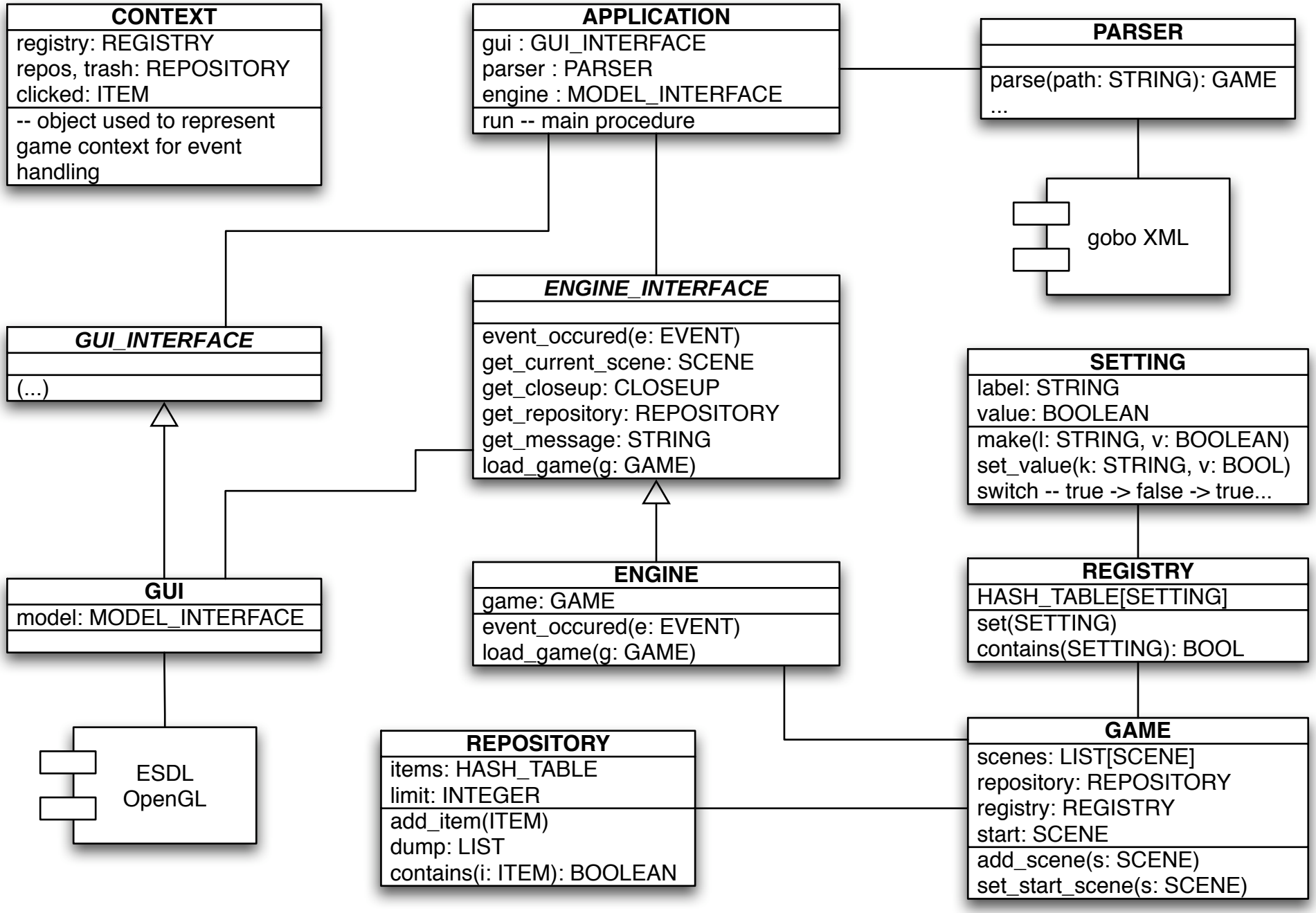
Conditions sind entweder "Gegenstand befindet sich im Repository" oder "Setting ist in der Registry eingetragen". Somit hat der Spieldesigner diverse Möglichkeiten, Actions und Sichtbarkeit von bestimmten hinreichenden oder notwendigen Bedingungen abhängig zu machen.

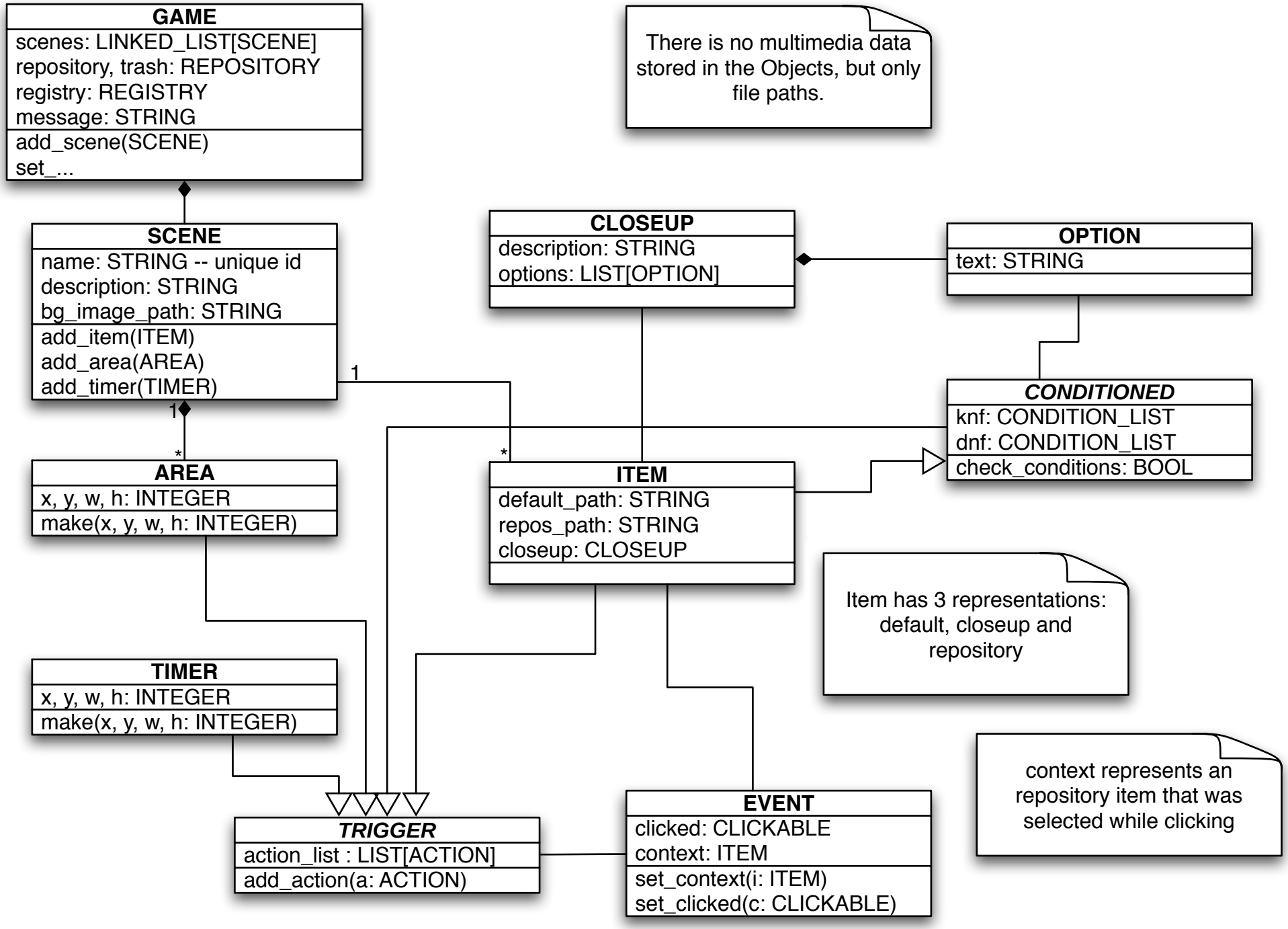
Um Gegenstände auch auf andere Gegenstände anwenden zu kännen (z.B. Schlüssel auf Tür), wir bei jedem Event jeweils ein Kontext im Sinn eines ausgewählten Repository Items angegeben. Dieser Kontext wird dann beim Event Handling entsprechend berücksichtigt.

Weitere Informationen zu den Klassen: siehe UML Diagramme und Quellcode (Kommentare)


AREA
x, y, w, h: INTEGER
make(x, y, w, h: INTEGER)
ITEM
default_path: STRING
repos_path: STRING
closeup: CLOSEUP
EVENT
clicked: CLICKABLE
context: ITEM
set_context(i: ITEM)
set_clicked(c: CLICKABLE)
TRIGGER
action_list : [ACTION]
add_action(a: ACTION)
in repository
CONDITIONED
knf: CONDITION_LIST
dnf: CONDITION_LIST
check_conditions: BOOL
1
CONDITION_LIST
list: LIST[CONDITION]
add_condition(c: CONDITION)
sat: BOOLEAN
CONDITION
satisfied(g: GAME): BOOL
*
KNF
-- notwendige Cond.
sat: BOOLEAN
DNF
-- hinreichende Cond.
sat: BOOLEAN
ACTION
perform(c: CONTEXT)
performable(c: C'TEXT): BOOL
REMOVE
item: ITEM
make(i: ITEM)
perform(game: GAME)
SHOW_SCENE
scene: SCENE
make(s: SCENE)
perform(game: GAME)
SHOW_CLOSEUP
closeup: CLOSEUP
make(c: CLOSEUP)
perform(game: GAME)
HIDE_CLOSEUP
make
perform(game: GAME)
PICKUP
item: ITEM
make(i: ITEM)
perform(game: GAME)
CHANGE_SETTING
setting: SETTING
make(setting: SETTING)
perform(game: GAME)


CONDITION
satisfied(g: GAME): BOOL
CONDITIONED
knf: CONDITION_LIST
dnf: CONDITION_LIST
check_conditions: BOOL
visibility
in registry
in repository
performability
SETTING
label: STRING
value: BOOLEAN
make(l: STRING, v: BOOLEAN)
set_value(k: STRING, v: BOOL)
switch -- true -> false -> true...
ITEM
default_path: STRING
repos_path: STRING
closeup: CLOSEUP
visible: BOOL
set_visibility(CONTEXT)
ACTION
perform(game: GAME)
CONDITION_LIST
list: LIST[CONDITION]
add_condition(c: CONDITION)
sat: BOOLEAN
REGISTRY
HASH_TABLE[SETTING]
set(SETTING)
get(key: STRING): BOOLEAN
REPOSITORY
items: HASH_TABLE
add_item(ITEM)
dump: LIST
contains(i: ITEM): BOOLEAN
KNF
-- notwendige Cond.
sat: BOOLEAN
DNF
-- hinreichende Cond.
sat: BOOLEAN


CONTEXT
registry: REGISTRY
repos, trash: REPOSITORY
clicked: ITEM
-- object used to represent game context for event handling
APPLICATION
gui : GUI_INTERFACE
parser : PARSER
engine : MODEL_INTERFACE
run -- main procedure
PARSER
parse(path: STRING): GAME
...
gobo XML
GUI_INTERFACE
(...)
ENGINE_INTERFACE
event_occured(e: EVENT)
get_current_scene: SCENE
get_closeup: CLOSEUP
get_repository: REPOSITORY
get_message: STRING
load_game(g: GAME)
SETTING
label: STRING
value: BOOLEAN
make(l: STRING, v: BOOLEAN)
set_value(k: STRING, v: BOOL)
switch -- true -> false -> true...
GUI
model: MODEL_INTERFACE
ENGINE
game: GAME
event_occured(e: EVENT)
load_game(g: GAME)
REGISTRY
HASH_TABLE[SETTING]
set(SETTING)
contains(SETTING): BOOL
ESDL
OpenGL
REPOSITORY
items: HASH_TABLE
limit: INTEGER
add_item(ITEM)
dump: LIST
contains(i: ITEM): BOOLEAN
GAME
scenes: LIST[SCENE]
repository: REPOSITORY
registry: REGISTRY
start: SCENE
add_scene(s: SCENE)
set_start_scene(s: SCENE)


GAME
scenes: LINKED_LIST[SCENE]
repository, trash: REPOSITORY
registry: REGISTRY
message: STRING
add_scene(SCENE)
set_...
There is no multimedia data stored in the Objects, but only file paths.
SCENE
name: STRING -- unique id
description: STRING
bg_image_path: STRING
add_item(ITEM)
add_area(AREA)
add_timer(TIMER)
CLOSEUP
description: STRING
options: LIST[OPTION]
OPTION
text: STRING
CONDITIONED
knf: CONDITION_LIST
dnf: CONDITION_LIST
check_conditions: BOOL
1
1
*
*
AREA
x, y, w, h: INTEGER
make(x, y, w, h: INTEGER)
ITEM
default_path: STRING
repos_path: STRING
closeup: CLOSEUP
Item has 3 representations: default, closeup and repository
TIMER
x, y, w, h: INTEGER
make(x, y, w, h: INTEGER)
TRIGGER
action_list : LIST[ACTION]
add_action(a: ACTION)
EVENT
clicked: CLICKABLE
context: ITEM
set_context(i: ITEM)
set_clicked(c: CLICKABLE)
context represents an repository item that was selected while clicking

Interface Widgets

GUI_CLICKABLE

Inherit from this class to create clickable objects which can process mouse clicks, mouse hoover events and key press events. The objects have a position and size given by x, y, width and height values.
You can attach a descriptive string to a clickable. You get this string back by querying either the 'description' or the 'out' attribute.
An arbitrary data object can be attached to the clickable for tracking purposes. Clickables have an enabled/disabled state.

GUI_DRAWABLE

Inherit from this class to create drawable objects. Drawables must implment 'draw' to draw themselves using OpenGL commands. They have a visibility state.

GUI_CLICKABLE_3D

Objects representing clickable areas or items on the world panorama cylinder. If they represent clickable areas then they are static and invisible. If they represent clickable items then they have a representing image and they are animatable.

GUI_SPECIAL_FRAME

Objects representing a background frame which has a diagonal edge in one of the four corners. You can specify the frame and fill colors for both normal and mouse hoovering state, frame thickness and size of the diagonal edge.

GUI_SPECIAL_BUTTON

Button objects are made of a special frame containing a text and/or an image. Buttons are clickable and can change their visual style according to mouse events (clicking, hoovering).

GUI_CONTAINER

Objects of type GUI_CONTAINER onsist of a background of type GUI_SPECIAL_FRAME and a collection of special buttons which can be drawn on that background. The buttons are aligned horizontally. You can specify the spacing between the items.

GUI_CLOSEUP

A closeup dialog is composed of a GUI_SPECIAL_FRAME. Inside the frame an image, a headline and various option texts can be displayed. The user can select one of the options. You can specify an event action which is executed when the user selects an option.

GUI_FILE_DIALOG

Special version of a closeup dialog displaying a selection of files for storing/retreiving purposes.

GUI_ERROR_DIALOG

Special version of a closeup dialog displaying a non recoverable error. after the user accepts the message, the application will be terminated.

Scene Elements

GUI_SCENE

Deferred class serving as base class for all kind of scenes. Every scene is made of a world background where the game takes place and a overlay including skin mask, repository and main menu. The scene is able to draw itself onto the screen using the redraw feature. It handles mouse events and feeds them to the appropriate objects.

GUI_GAME_SCENE

Each game scene object contains all information for one adventure location including cylindrical background pane, static and moving foreground objects and clickable areas.

GUI_CREDITS

Scene displaying the game credits as a nice three dimensional typo animation.

GUI_DEFAULT_SCENE

GUI_DEFAULT_SCENE creates an initial game scene which is displayed when the xae adventure engine is started.

Global Skin including Repository and Main Menu

GUI_REPOSITORY

An object of type GUI_CONTAINER keeping and managing repository items; the items you collect during game play.

GUI_MAIN_MENU

An object of type GUI_CONTAINER holding the main menu buttons: Open script, load game, save game, credits and exit.

GUI_GAME_SKIN

Singleton object representing the mask being drawn on top of the game world. The mask consists of a background bitmap with alpha channel and two dialog elements: The repository and the main menu.

Globally Accessible Helper Classes

GUI

The singleton object of type GUI represents the graphical user interface. It has references to the screen, the game parser, engine and serializer. The GUI is responsible for creating and displaying the initial game scene.

GUI_SHARED

Inheriting from this class provides access to the singleton objects of GUI, GUI_CONSTANTS, GUI_TIMING and GAME_SKIN.

It also holds 2 global textures used for special effects.

GUI_CONSTANTS

The singleton object of GUI_CONSTANTS encapsulates various constants such as colors, fonts, default image paths and size parameters.

It also provides a 'ticks' function (which is not exactly a constant) giving the number of milliseconds since the app was started.

GUI_TIMING

The singleton object of GUI_TIMING encapsulates various interpolation smoothing functions.

The provided functions mainly base on bernstein polynomials.

OPEN_GL_LIBRARY

Inheriting from OPEN_GL_LIBRARY provides a comfortable way to implement all the openGL stuff you need in a certain class.

Simply inherit from this class and you automatically inherit from all classes which contain the openGL functions and constants.

In addition, you find a bunch of useful openGL macros defined here.

Proportional Font Support

GUI_PROPORTIONAL_FONT

Font objects drawing text onto an ESDL surface. text can be written on one line or on multiple lines within a bounding box including automated word wrapping. left, centered and right alignment of the text is possible.

bitmap fonts with charcter width information are used for drawing. alpha channels are fully supported. Text can be displayed in any color.

The class also supports effects like variable character and line spacing and maximum number of characters drawed (used for typing effects).

GUI_FONT_FACE

After attaching a proportional font to a font face object it can be used to keep and manage a snapshot of font attributes.

This class is not supported in the current version of the XAE adventure engine.

Transition Effects