Vrije Universiteit Amsterdam
Faculty of sciences

**MASTER THESIS**

*Milan Slančík*

# Advanced floor plan designer in Flex

*Department of computer science*

Supervisor: *Prof dr Anton Æliëns*
Second reader: *Dr Evert Wattel*
Study program: *Informatics, Multimedia Computer Science*

# Acknowledgements

First of all, I wish to express my sincere gratitude and appreciation to my supervisor, *Prof Dr Anton Æliëns*, for his thoughtful guidance, his valuable suggestions, comments during discussions, prompt response to my emails and speedy feedback. My gratitude also goes to my second reader, *Dr Evert Wattel* for his ideas, willingness to read drafts and test the application in advance. Last, but not least, I would like to give my sincere thanks also to my *parents*, who have supported me throughout the writing process.

# Contents

# List of Tables

# List of Figures

Title: Advanced floor plan designer in Flex
Author: *Milan Slančík*
Supervisor: *Prof dr Anton Æliëns*
Second reader: *Dr Evert Wattel*

Abstract:
The aim of the project is to design and implement a simple easy-to-use online designer. The user will be able to draw a 2D floor plan of his apartment with the possibility of adding doors, windows and furniture from the menu with ease. Each added object has its own properties such as width, height, length, rotation etc. Walls can be edited as well - setting up height or changing color is straightforward. For advanced 3D presentation, the possibility to add viewpoints (e.g. a "kitchen view") and to indicate simple routes for animation is included. With extensions, the application can also be used as a presentation tool for estate agencies or even for furniture-making companies.

The designer will be written in Adobe Flex technology. The output of this application will be a descriptive XML file on which XSL transformation will be applied in order to create the target X3D format. A proper X3D player will then be used to render the 3D content.

Keywords:
Flex, RIA, Virtual reality, X3D

# 1 Introduction

## 1.1 Background

First ideas related to this project came up when I was working on a software project called "Realita" in the context of my studies at Charles University in Prague. It was a real-estate search engine and my share of the work was to develop a simple floor plan designer. The software project was completed successfully and I've decided to continue with the work. Within the frame of this thesis came the opportunity to improve and extend functionalities of the designer as well as automatic generation of 3D scenes. In order to accommodate new functionality and modern features of the Flex framework, the designer application was completely reworked in the Flex environment.

The basic goals of this thesis are following:

- To develop an easy-to-use software tool for floor plan drawing.
- To allow **online** plan design
- To provide a tool to generate 3D scenes in X3D format
- To provide means for real estate agencies to improve their presentation layer

## 1.2 Structure of this Document

**Chapter 2** explains the aim of this work, thoughts of the author and the technological choices (focusing on the Flex technology – related questions)

**Chapter 3** is an analysis of existing solutions. It contains brief descriptions and an overall comparison. The application Floorplanner.com is discussed thoroughly. The new Dragonfly project is also mentioned.

**Chapter 4** contains formal definitions used in this document. Terms like XML, XSL, XLST, Adobe Flex, VRML and X3D are precisely defined and explained with examples if needed.

**Chapter 5** describes the software design. It also explains the reasons of chosen architecture.

**Chapter 6** contains implementation details, deals with base classes, design patterns and X3D generation.

**Chapter 7** is an example demonstration. It includes two case studies a step by step tutorial for drawing a floor plan and generating a 3D presentation.

**Chapter 8** is a summary and a discussion of the proposed solution. Future work suggestions are also included.

# 2 Aim of the work and research issues

The purpose of this work is to design and create a new floor plan designer with the possibility to generate a 3D presentation from sketched 2D plans. This thesis was developed in three distinct stages:

1. Research of the existing software

The author had hands on experience with several of the existing software solutions, starting with the simpler ones like Paintbrush to the professional tools such as ArchiCAD. This subject is developed further in chapter 3, "Related work". Advantages and disadvantages of the existing solutions in interior design have been taken into account during the design and development of the designer application.

According to the author's experience, the properties that the desired designer should meet are:

- a familiar and friendly user interface
- easy plan drawing
- fast sketching process
- no installation required – the design is made online
- fast learning curve

2. Feasibility study and implementation proposal

With the onset of new technologies, the possibilities are expanding. Developing a simple client-side .EXE designer would be similar to trying to sell CD-ROM drives on the market already full of DVD-ROMs. Modern end-users tend to become more ambitious and distinguished day by day, they are demanding, "spoiled" and do not want to work with hard-to-control software. The current trends indicate that the attention of the user shifts towards rich internet applications.

To deliver and store content on web sites, the author considered the following technologies:

- *Java and Java applets* - not very suitable, it may be difficult to get Java applets to work correctly even for an experienced user
- *Ajax* - a better option; has XML support and pre-built frameworks which can be used, is familiar to those who know HTML, JavaScript, etc., there is no need for additional software except for an up-to-date browser and it is an open standard. However, the drawing possibilities and libraries are not so fully-featured and enhanced as in Flash.
- Another RIA considered technology is *Microsoft Silverlight*, a significant competitor to *Adobe Flex*, with which is shares many aspects. The development in Silverlight has some advantages for .NET users – the code is written in C#. Microsoft Visual Studio provides a comfortable working environment. On the other hand, the Silverlight plug-in is not as widespread as Adobe Flash.

Despite the author's fondness of the .NET framework, the Flex framework was chosen as the presentation layer of the application.

Flex can be used with any back-end technology, see chapter 4.2.3. Graphical and programmatic skinning is possible by making use of Illustrator, Flash, Photoshop or CSS. Moreover, the Flex framework is fully open source.

Considering the 3D generation, Flash has its own frameworks, e.g. Papervision3D, Sandy3D etc. They don't have hardware support thus even the simple scenes can be rendered with delays. On contrary the X3D technology allows to play even more demanding 3D scene easily using advantages of hardware support. That's why the use of X3D technology has been approved even at the cost that user has to install a separate player. To conclude the selection, Flex was chosen for the presentation layer and .NET was chosen for the back-end. See chapter 5 for more information.

3. Demonstration of results

The application was named **FloorPAD**. It is an abbreviation of *Floor plan advanced designer*. The application is hosted at www.floorpad.eu. The site contains a few examples and even the possibility to try the designer application. This paper also contains case studies, a regular use case and the real estate professional case study. See chapter 7 for more detailed information.

# 3 Related work

In this chapter, we analyze the existing solutions in the field of interior design. We briefly introduce a few existing similar software solutions among many others and then elaborate on a particular one - Floorplanner.com. At the end of the chapter, a new interesting project - the Dragon Fly - will be presented.

## 3.1 Similar software

### 3.1.1 Sweet home 3D

Sweet Home 3D [17] is a desktop application. Its purpose is interior design. It allows users to draw floor plans, insert furniture in 2D and preview the design in 3D immediately. It is distributed under the GNU Public License. It is available for Windows, Mac OS X 10.4 / 10.5, Linux and Solaris.

### 3.1.2 FloorPlan 3D

FloorPlan® 3D [18] is a desktop application that allows projection of the exterior and interior of dwelling spaces or offices in a 3D environment. Free templates of dwelling projects can be found on the Internet and reused. This software is shareware and available for Microsoft Windows.

### 3.1.3 IKEA Planner

This designer is intended for room furnishing with real IKEA furniture [19]. The use of this application is intuitive. It is available for Microsoft Windows only and does not support Mac.

### 3.1.4 Google SketchUp

Google SketchUp [20] is (in its Pro version) a proprietary application that can be used to create, modify and share 3D models on Google 3D Warehouse[1]. It is easier to learn than other 3D modeling programs, which contributed to its popularity and widespread use. It is available for Microsoft Windows and Mac OS X. Non-Pro versions are free.

### 3.1.5 ArchiCAD

ArchiCAD [21] is a highly professional tool intended for architectural design. From the very beginning, ArchiCAD has been developed by architects for architects, thus making it not suitable for common users without proper education. The trial version is free.

[1] Warehouse of 3D models created, hosted and maintained by Google Company.

### 3.1.6 SceneCaster

SceneCaster [22] is a 3D social media application and online community. It is a successful attempt to move a designer application into the web environment. Scene Caster 3D plug-in has to be installed in order to create any content.

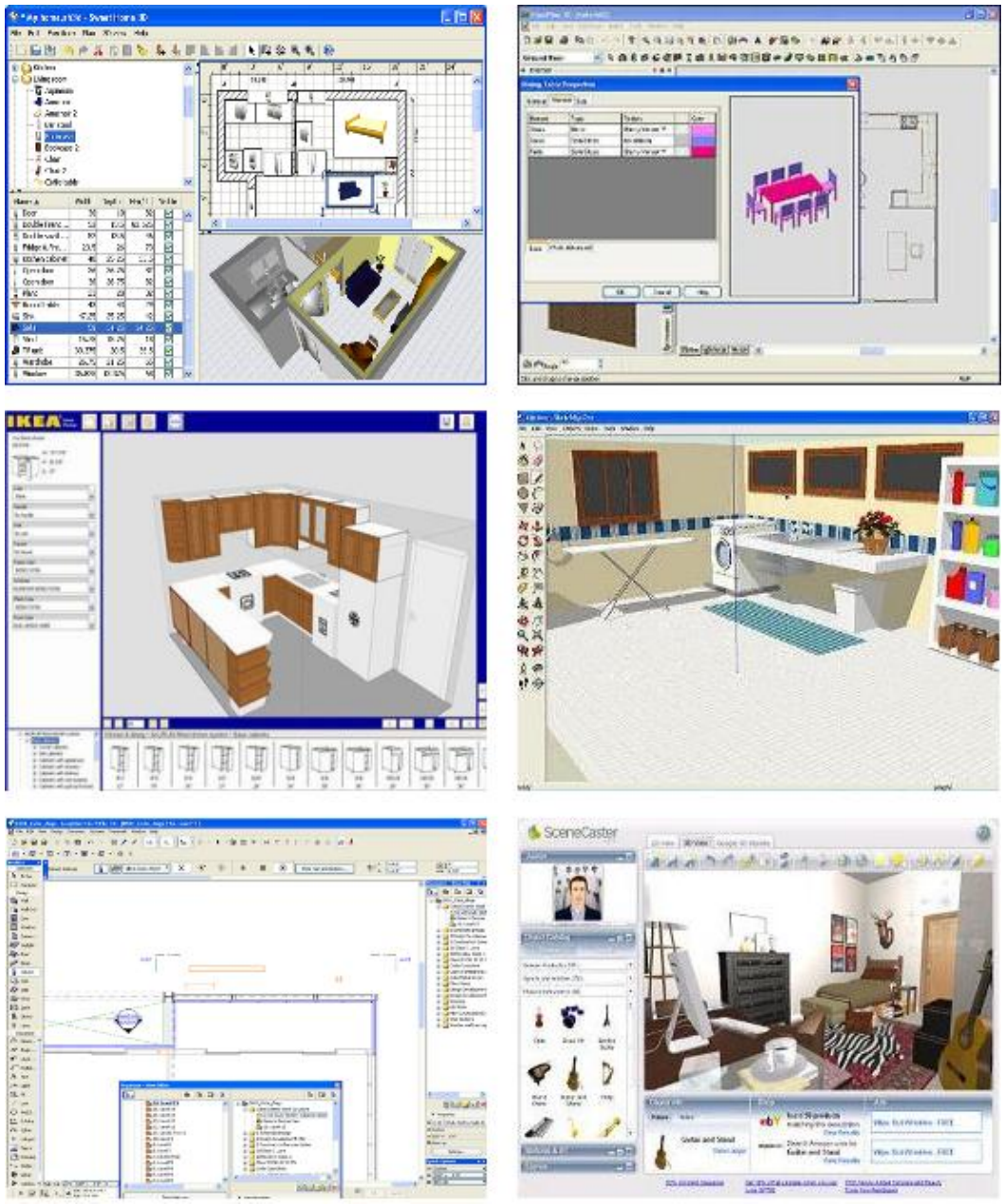


**Figure 1 User environments of select applications similar to FloorPAD**

Figure 1 shows the view of user environments of abovementioned applications. The first line contains the view of Sweet Home 3D (left side) and FloorPlan 3D (right side). The line below belongs to IKEA Planner (left side) and Google SketchUp (right side). The last line shows the user interface of ArchiCAD on the left and SceneCaster on the right.

## 3.2 Floorplanner

Floorplanner [23] is a highly professional, feature-rich internet-based application designed to create and share interactive floor plans. In its architecture and key features, the Floorplanner is very similar to FloorPAD. Let us take a look at their advantages and disadvantages.

Basic drawing algorithms are very similar. Floorplanner provides many more pieces of furniture (see figure 2) but on the other hand FloorPAD has a few intelligent object enhancements. For example, hanging objects (a wash basin, posters etc.) are attached to the wall and when the wall moves, so do the objects.

Both of the designers allow the user to draw easy-to-use floor plans and their 3D presentation is strongly dependent on the technology used. Floorplanner uses a Flash-based 3D framework (Papervision3D is very likely). Since Flash is a 2D platform and has no 3D hardware support, even a simple scene is played with noticeable delays.

The X3D technology in FloorPAD is fully hardware-supported meaning that 3D is rendered in high quality. It allows simple viewpoints editing and professionally-looking animations can be created fairly quickly. X3D also gave us the possibility to liven up objects on the scene – play a video on the TV, open and close doors or flush a toilet aloud. All of the above-mentioned features are introduced at the cost of one disadvantage – the necessity of an X3D player installation.

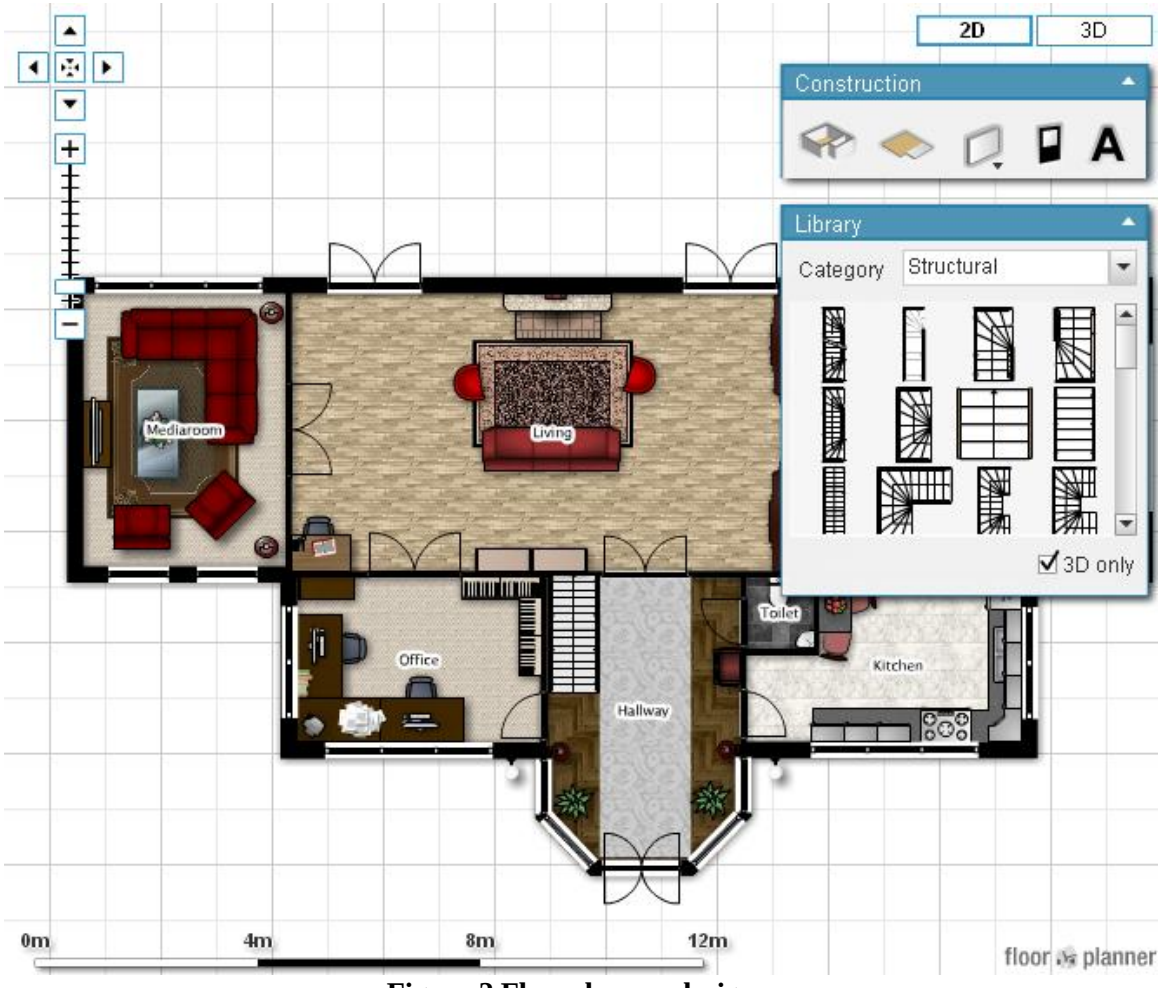


**Figure 2 Floorplanner designer**

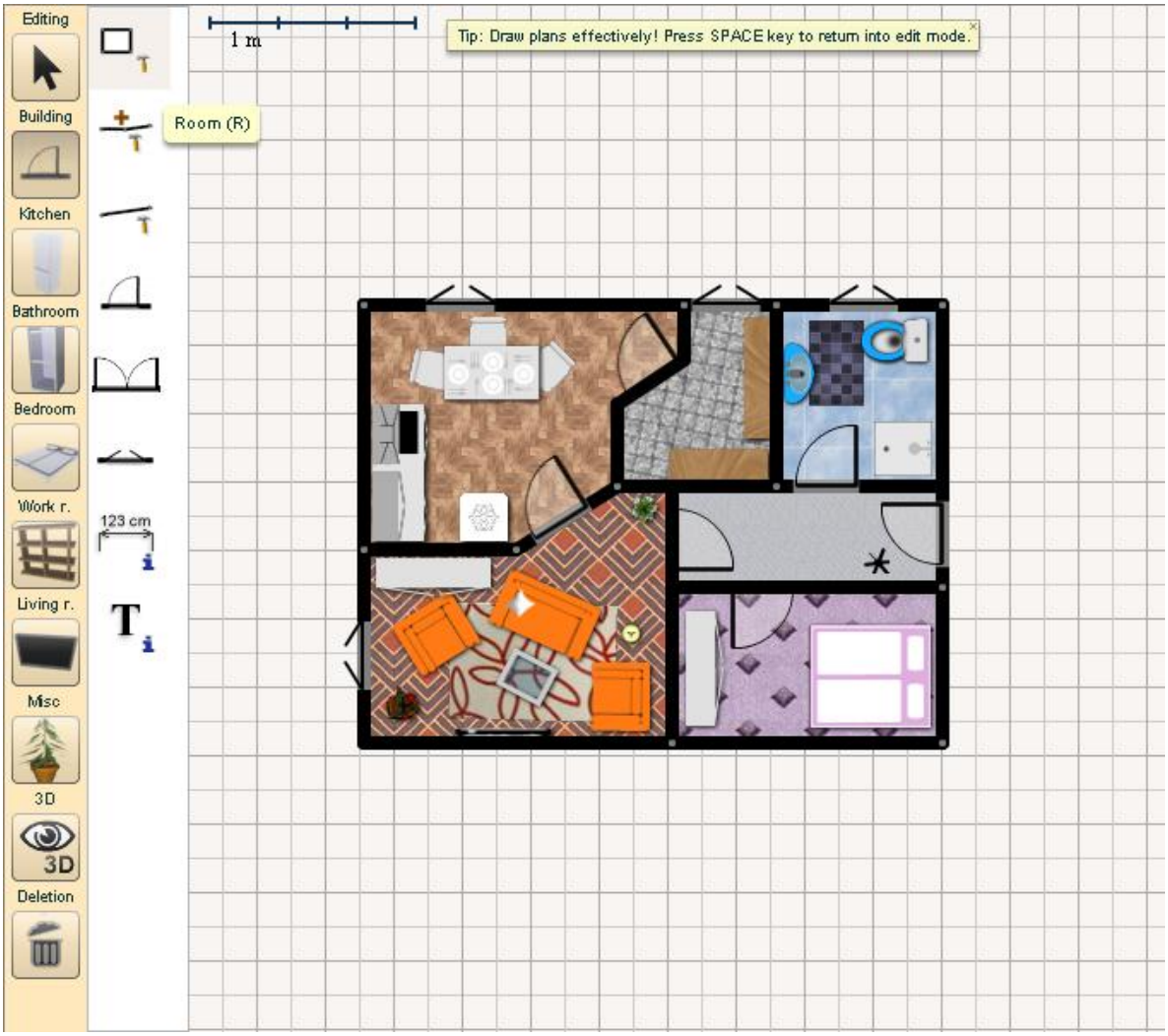


**Figure 3 FloorPAD demonstration**

To conclude the comparisons, it can be said that Floorplanner is oriented more on the common user, creating a kind of a social network. FloorPAD (figure 3), on the other hand, was also designed for the regular user but is primarily oriented on the real estate professionals, e.g. it is prepared to combine floor plans with photo indexes. It provides several interfaces to easily integrate with existing systems.

## 3.3 DragonFly

The DragonFly project [24] is in many aspects similar to FloorPAD or Floorplanner applications. It allows floor plan visualization in both 2D and 3D spaces, see figure 4. The project is interesting from the implementation point of view. It uses Flex framework with 2D Degrafa[2] library and Away3d[3] for the 3D implementation.

The interesting feature of this application is its ability to detect collisions and add wall attachments. For 3D generation, the Away3d framework was modified to keep processor-intensive tasks on the server thus reducing client processor activity.

[2] Declarative graphic framework library.
[3] Flash based 3D framework.

The project is covered by and developed under the Autodesk Labs group.


**Figure 4 Dragonfly designer**

# 4 Technology overview

In this chapter, all basic terms needed to better understand the rest of the document. Let us explain them one at a time from the beginning. We start with fundamental definitions of XML, XSL and XSLT; continue by Adobe Flex and eventually with 3D technologies. Some of the definitions contain examples when appropriate.

## 4.1 XML Technologies

### 4.1.1 XML

*Extensible Markup Language* (*XML*) is a structured, text-based way of data formatting and description. It has rapidly become an industry standard primarily because of its portability, especially for data exchange and interoperability between applications. In short, XML is a simple human-readable method of data representation [6]. Note that we can regard the XML document as a finite ordered tree.

### 4.1.2 XSL

*Extensible Stylesheet Language (XSL)* is a family of recommendations to define XML document transformation and presentation [15]. It consists of three parts:

1) *XSL Transformations* (XSLT) – a language for XML transformation
2) *XML Path Language* (XPath) – an expression language used by XSLT to access or refer to parts of an XML document
3) *XSL Formatting Objects* (XSL-FO) – an XML vocabulary for formatting semantics specification.

### 4.1.3 XSLT

*Extensible Stylesheet Language Transformation (XSLT)* is a language designed for transformation of XML documents into other XML documents. XSLT can be used either as a part of XSL or independently [16].

We can think of the program in XSLT as a set of rules (written in XML) transforming the input XML file into an XML output. The XSLT Processor will use rules specified in the XSLT and apply them to the XML input file in order to create the result document as shown in figure 5.

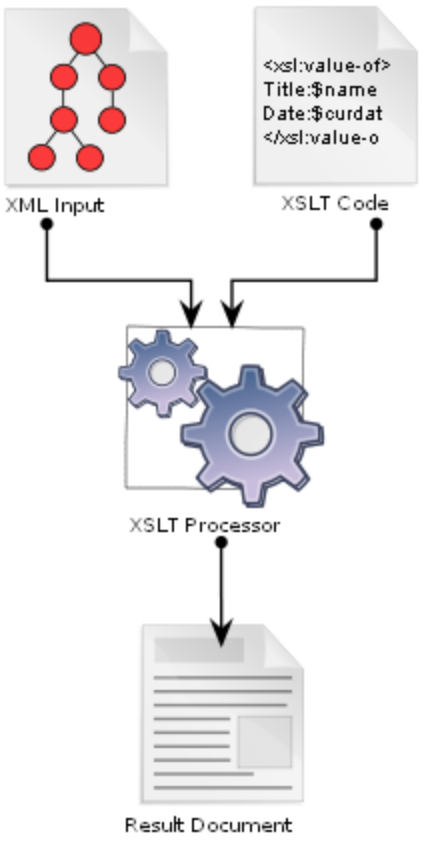


**Figure 5 Transformation process**

## 4.2 Adobe Technologies

*Adobe Flash*, or simply *Flash*, refers to both a multimedia authoring program and the Adobe Flash Player - written and distributed by Adobe Systems, Inc. Strictly speaking, Adobe Flash is the authoring environment and Flash Player is the virtual machine used to run the Flash files [9]. In colloquial language these have become mixed: "Flash" can mean the authoring environment, the player, or the application files.

### 4.2.1 Flash, the authoring environment

Flash is notoriously known as a tool to create animation, advertisements, simple games and various web page components. Flash can manipulate vector and raster graphics. It supports bi-directional streaming of audio and video. Integration of video with Flash is as simple as writing ordinary web pages. This authoring environment also provides the possibility to create content for mobile phones and other embedded devices. It uses ActionScript as the scripting language. The current version is 3.0.

### 4.2.2 ActionScript

*ActionScript* (AS) is the programming language of the Adobe Flash Player runtime. It enables efficient programming of Adobe Flash applications ranging from simple animations to complex, data-rich, interactive application interfaces [7].

ActionScript was initially focused on the animation. Early versions of Flash content offered few interactivity features and thus had very limited scripting capability. Flash MX 2004 introduced ActionScript 2.0 with improved development capabilities. After the arrival of Flash Player 9 alpha a newer version (3.0) of ActionScript has been released.

ActionScript 3.0 is an implementation of the ECMAScript standard thus featuring the same syntax as JavaScript. Comparing to ActionScript 2.0 these are the key values which were changed: the *Object oriented approach* [7]. Fundamental *data types* were changed [8]. The *ECMAScript for XML*, alternatively known as *E4X*, was introduced as a simpler, easier to read approach for working with XML objects than the traditional Document Object Model (DOM) [2]. Another feature would be the Unified *event handling* system. Last, but not least, support for *packages*, *regular expressions* and *namespaces* has been altered.

All of the above-mentioned differences provided significant gains in runtime performance and developer productivity. ActionScript 3.0 is faster than its predecessors. It has limited hardware acceleration support, allowing just a limited 3D content presentation. Recently, Flash libraries are being used together with XML to render rich browser content. This technology, known as Asynchronous Flash and XML, has pushed for a more formal approach called the Adobe Flex.

### 4.2.3 Adobe Flex

*Adobe Flex* is a highly productive, open source framework to build and maintain expressive web applications that can be deployed consistently on all major browsers, desktops, and operating systems [13]. While it is sufficient for the Flex applications to be built only using the free Flex software development kit (Flex SDK) with an open source editor such as FlashDevelop, developers can use the Adobe Flex Builder 3 software to accelerate development.

Apart from the ActionScript 3.0, Flex uses a new MXML (Macromedia XML) language to define the user interface components. It allows for a simpler composition of more components into one or extension of the basic components. The ability to separate the data and logic (in ActionScript 3.0 code) with the design in MXML is reminiscent of the MVC[4] architecture. Another strong feature of Flex is the advanced design layout possibility. There's a lot of shortcuts and "syntactic sugar" in MXML to help with writing code. Styles can be put into CSS, and styles and events can be set as attributes.

Flex framework is suitable to create data-driven applications. It has great support for working with XML formats as well as calling remote web services. The Flex framework can cooperate with any back-end technology (.NET, JAVA, PHP, Ruby, Python, etc.).

Applications like *MySpace*, *YouTube* or *Floorplanner* use Flex framework as well. All of these Flash-based applications have to be displayed in a proper player called the Adobe Flash Player.

### 4.2.4 Adobe Flash Player

The *Adobe Flash Player* is a widely distributed proprietary multimedia and application player

[4] MVC means Model View Controller architectural pattern

created by Macromedia and now developed and distributed by Adobe (after its acquisition). Flash Player runs `SWF` files that can be created by the Adobe Flash authoring tool, by Adobe Flex or by a number of other Macromedia and third party tools. The latest version, version 10, already has a limited support for 3D graphics (It has hardware accelerated 3D texture transforms and 3D features to the 2D drawing API).

Adobe Flash Player is available for most common web browsers, some mobile phones and other electronic devices. Flash Player is the world's most pervasive software platform, used by more than 2 million professionals and reaching 99.0% of Internet-enabled desktops[5] [10].

## 4.3 Virtual reality

*Virtual reality (VR)* is a system that provides the user or a group of cooperating users an illusion of being in an artificial environment called the *virtual world*, *virtual scene* or *virtual environment*.[1] An application from this scope has these typical characteristics:

- *Real time* – the display and interaction with the user are provided with a certain speed so that the movement appears fluent. The minimal sufficient speed is considered to be 25 frames per second.
- *3D space* – scene and objects have a 3D character or at least are creating a 3D illusion.
- *Navigation* – user can explore the world from the outside or enter the scene and walk through it in various modes (Walk, Fly, Jump, Examine ...).
- *Interactivity* – a scene may contain interactive objects. User can manipulate them or they are animated according to given scenarios.
- *Multimedia* – sounds and videos are used to magnify the VR experience.

### 4.3.1 VRML

*Virtual reality modeling language (VRML)* is a standard definition language for virtual world description as well as a file format in which these worlds are stored and distributed.

### 4.3.2 3D anywhere

*EXtensible 3D (X3D)* is a scene-graph architecture and a file-format encoding that improves on the VRML international standard. X3D uses XML to express the geometry and behavior capabilities of VRML [14]. Figure 6 shows their similarities.

[5] Millward Brown survey, conducted June 2008

| XML Syntax (.x3d) | Classic VRML Syntax (.x3dv) |
|---|---|
| <Shape DEF="MyShapeNode"<br>bboxCenter="0 0 0"<br>bboxSize="-1 -1 -1"><br><Box DEF="SingleGeometryNode"/><br><Appearance<br>DEF="SingleAppearanceNode"/><br></Shape> | DEF MyShapeNode Shape {<br>geometry DEF SingleGeometryNode<br>Box {}<br>appearance DEF SingleAppearanceNode<br>Appearance {}<br>bboxCenter 0 0 0<br>bboxSize -1 -1 -1<br>} |

**Figure 6 Example of X3D and VRML syntax**

It is particularly important to note that XML benefits are numerous: XML features customized meta-languages for data structuring, is easily readable by both humans and computer systems, has verifiable data constraints, etc. XML is license-free, platform independent and well supported.

Another advantage of X3D is the existence of profiles (figure 7). They help to enable browser builders to achieve intermediate levels of support for X3D. Profiles help extend the reach of X3D to smaller, lightweight devices such as cell phones [4].

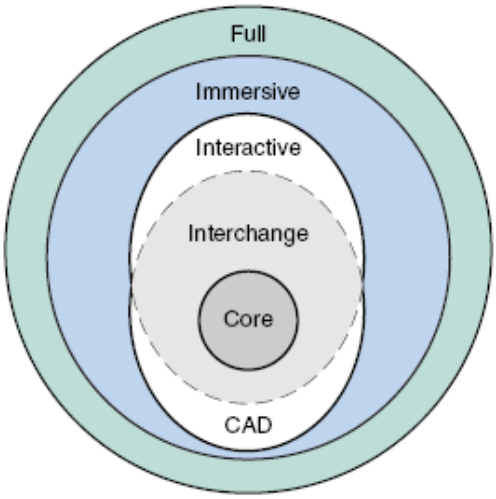


**Figure 7 X3D profiles**

The *Core* profile only includes basic X3D definitions (such as ROUTE) and metadata nodes. The *Interchange* profile is the base profile for geometry models exchange between various 3D applications. This encompasses all of the basic geometry (primitives, triangles and polygons), appearance (material and texture) and animation. *CADInterchange* is a specialized profile that supports import of CAD models. In includes most (but not all) nodes from the Interchange profile. The *Immersive* profile matches VRML97 the closest. It includes everything defined in the Interactive profile with the addition of several advanced capabilities and a number of new nodes: 2D geometry, environmental effects, event utilities and so on. The Full profile event further extends the Immersive profile and incorporates all of the X3D capabilities including Distributed Interactive Simulation (DIS), Humanoid Animation (H-Anim), GeoSpatial, Non-Uniform Rational B-spline Surfaces (NURBS) and other advanced

components.

Files X3D can be encoded as text, binary files or XML. The text version is a superset of VRML; therefore backward compatibility is guaranteed [1]. X3D contains a rich set of componentized features that are tailored for use in many areas such as multimedia, entertainment and education but also in engineering and scientific visualization.

### 4.3.3 X3D browser

An *X3D browser* is a software application that can parse (read) X3D scenes and render (draw) them, not only showing 3D objects from various viewpoints but also enabling object animation and user interaction. An X3D browser is often implemented as a plug-in that works as an integral part of a regular hypertext markup language (HTML) web browser (such as Internet Explorer or Mozilla Firefox). An X3D browser can also be delivered as a standalone or embedded application.
Table 1 shows the overview of most popular browsers and their comparison according to target platform and technology [25].

| Browser / System | Windows | Mac OS | Linux | Pocket PC | VRML | X3D |
|---|---|---|---|---|---|---|
| BS Contact [26] | Yes | No | No | Yes | Yes | Yes |
| Cortona [27] | Yes | Yes | No | Yes | Yes | From 5.0 |
| Flux 2 [28] | Yes | Planned | Planned | No | Yes | Yes |
| FreeVRML [29] | No | Yes | Yes | No | Yes | Yes |
| Octaga [30] | Yes | No | Yes | No | Yes | Yes |

**Table 1 Overview of VRML/X3D browsers**

## 4.4 ASP.NET

*ASP.NET* is a web application framework developed and marketed by Microsoft Corporation to build dynamic web sites, web applications and web services [31]. ASP.NET is built on the Common Language Runtime (CLR), allowing programmers to write ASP.NET code using any supported .NET language (C#, Visual Basic...). It was first released in January 2002 with version 1.0 of the .NET Framework. Currently, the latest stable version is 3.5 and the version 4.0 Beta 1 has been released recently.

## 4.5 Design patterns

Design patterns can be characterized as generic solutions for some types of problems. We tend use patterns because they have proven to be useful and/or efficient in certain situations. In general, they provide a better solution, maybe not the best one, but suitable enough in most cases.

Their use depends on experience and the underlying problem. The understanding of the problem is the first and the most important step when applying a suitable pattern. With the knowledge of patterns, the communication is also easier. When the Singleton design pattern is mentioned, everyone knows that a certain object can be instantiated only once (new operator cannot be used). FloorPAD uses several patterns, which will be discussed later in chapter 6.5, "Code culture".

# 5 Architecture and design

The architecture of FloorPAD is not complex. For this purpose, only two UML diagrams [5] (deployment and activity) should be enough to render an account of whole concept.

## 5.1 Overview

The application is hosted on a web server. It consists of a presentation layer (www pages and plan designer) and a transformation engine. The user loads a web page (via HTTP protocol) into a web browser (e.g. Mozilla, IE ...) as shown on the deployment diagram in figure 8.

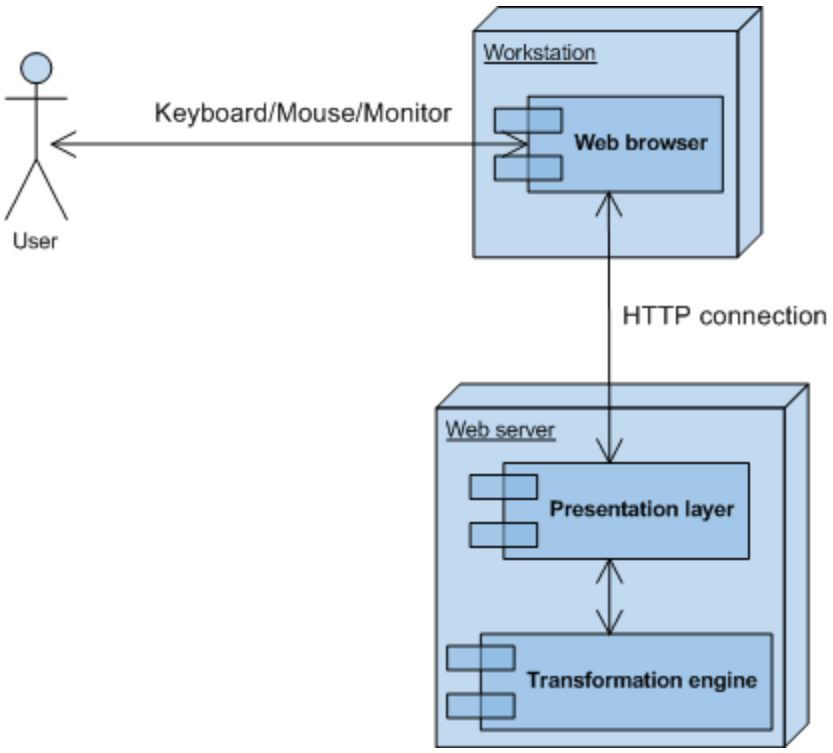


**Figure 8 Architecture overview**

The plan designer is a Flex application (SWF file). The Flash platform is quite widespread and the size is considerably small (the whole application takes up less than 500 kB). It is written in ActionScript 3.0 mainly because of its above mentioned improvements (in section 4.2.2) compared to its previous versions.

The server side of the solution uses ASP.NET technology in C# programming language. The .NET framework 2.0 provides simple and professional approach to web pages creation including sufficient XML support. MS SQL server 2005 was used to store user data.

The process of 3D output creation is described by the activity diagram in figure 9. It is relatively straightforward. The user draws a plan in the embedded plan designer on the web page. Delineated objects are described in an XML file. This output is sent back to server to apply XSL transformation and to show new X3D result.

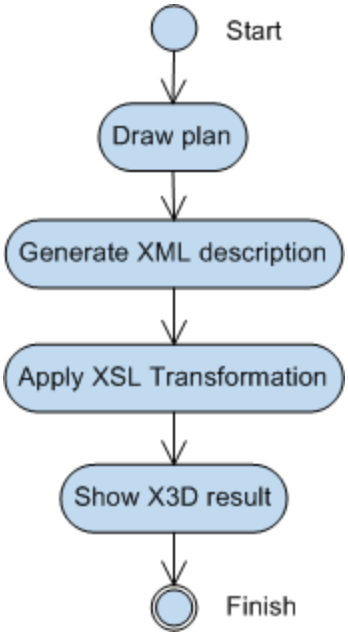


**Figure 9 Activity diagram of 3D scene generation**

## 5.2 System properties

Table 2 provides a quick overview of the expected system properties.

| Category | Expected performance |
|---|---|
| Performance | Initial launch (download) of the Flex designer depends on the internet connection speed. The 3D presentation depends on the type of the graphic card and the X3D browser installed. |
| Reusability | The application can be used in other areas as well. It can be easily moved to desktop platforms with the help of Adobe Air[6] technology. |
| Extensibility | The designer is extensible by using furniture extensibility framework. |
| Testability | It is fairly simple to check whether the generated X3D file is correct [14]. |
| Scalability | As a web application, FloorPAD is executable on most computers. Many instances of the designer loaded in the browser may noticeably slow down performance of the whole computer. |
| User interface | The application has GUI designed in a way to make drawing as easy as possible. |
| Localization | The application supports language configuration files. |

**Table 2 Non-functional application requirements**

[6] Adobe Air is a cross-platform runtime environment intended for building rich internet applications using Adobe Flash, Flex, Html or Ajax that can be deployed as a desktop application.

## 5.3 User interface

The emphasis should be taken on the simplicity and user amiability. Sketching with FloorPAD should be as easy as with Paintbrush. Figure 10 show the basic decomposition of designer's parts. The menu of the application, with the action buttons, is located on the left, options are on the right. The drawing place is in the center. These parts are mapped into class model in chapter 6.1, "Fundamental classes".

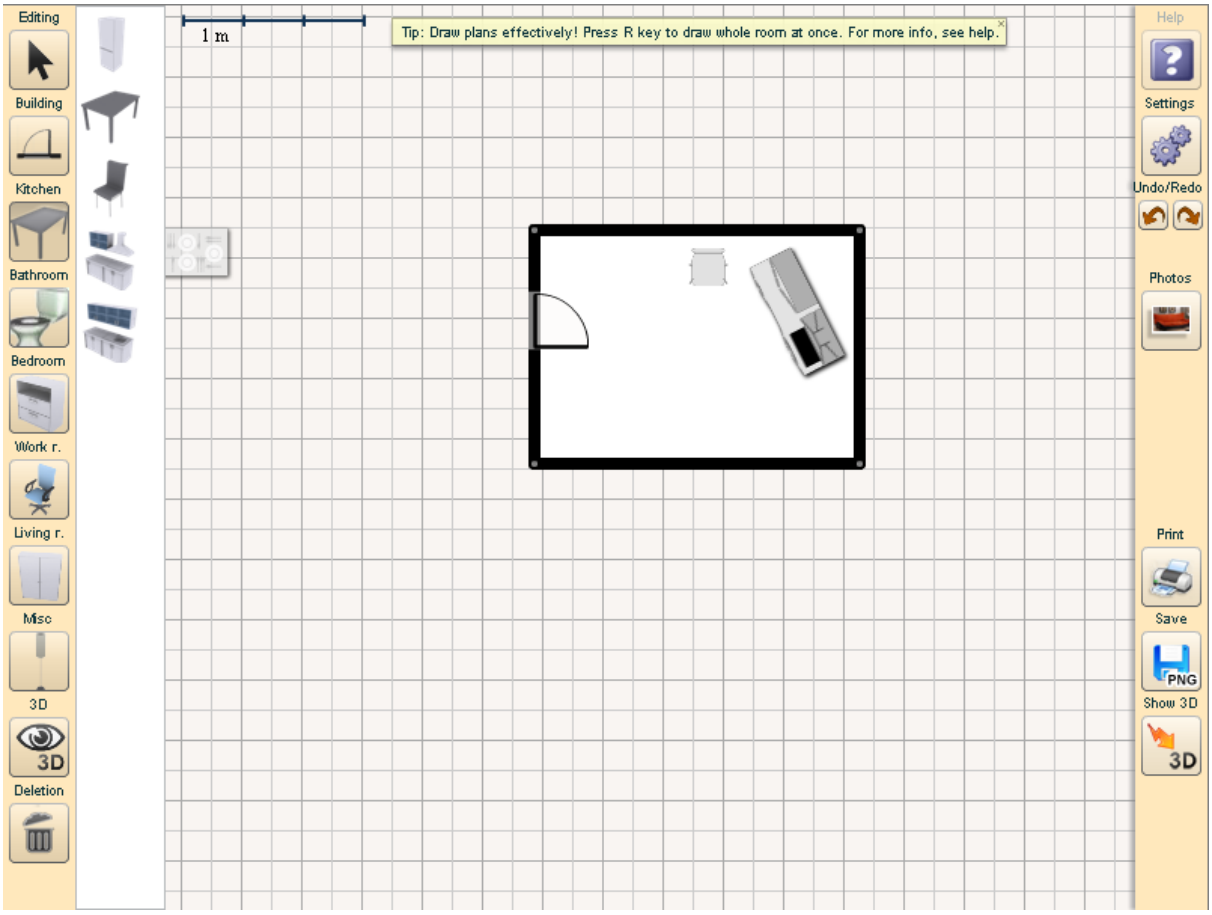


**Figure 10 User interface**

The sketching process has to be as easy as possible, like it was mentioned in part 5.2. All drawing algorithms are described in detail in appendix C, "User documentation".

# 6 Realization and implementation

This chapter treats the implementation details of FloorPAD. At first, the Flex application is discussed. Initial passages explain the whole concept and fundamental class hierarchy. The following parts are dedicated to objects' depth management, some notable ideas in the designer and the use of design patterns. The following section deals with server-side X3D generation. The description of the process of adding a new piece of furniture in detail is included in the appendix.

## 6.1 Fundamental classes

In this section, essential classes needed to better understand the rest of the chapter are described. For some of the classes, UML diagrams are included [5]. For sake of clarity, diagrams contain neither attributes, nor methods.

### 6.1.1 Ancestry classes

Class *Sprite* is the ancestor of all of the essential classes in the designer. Figure 11 illustrates the hierarchy.

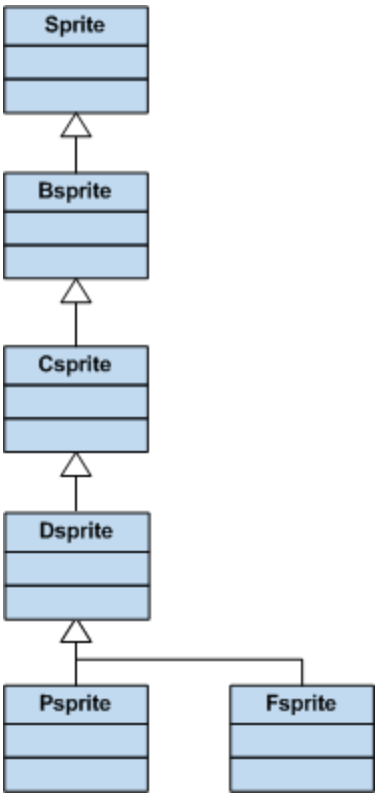


**Figure 11 Base class hierarchy**

A short description of these classes is as follows:

- *BSprite (Base Sprite)* - contains properties and methods to facilitate work with descendant class object types.

- *CSprite (Customized Sprite)* – includes properties related to furniture information: dimensions, scaling possibilities and restrictions and depth level (see part 6.2). It also contains some generally useful methods.

- *DSprite (Dynamically extendable Sprite)* – is the descendant of all dynamically extendable classes, representing pieces of furniture.

- *PSprite (Placed Sprite)* – is the descendant of all classes (representing furniture) that cannot be built in a wall.

- *FSprite (Fill sprite)* – bears attributes and methods to fill shapes with color or texture.

### 6.1.2 The application class skeleton

The FloorPAD designer works in two modes:

1. *Normal (default) mode* - allows drawing rooms, adds furniture, viewpoints and animation.

2. *Face-wall mode* – allows editing of a face of a particular side of a wall, adding tiles and small wall-hung objects that cannot be seen in *Normal mode*.

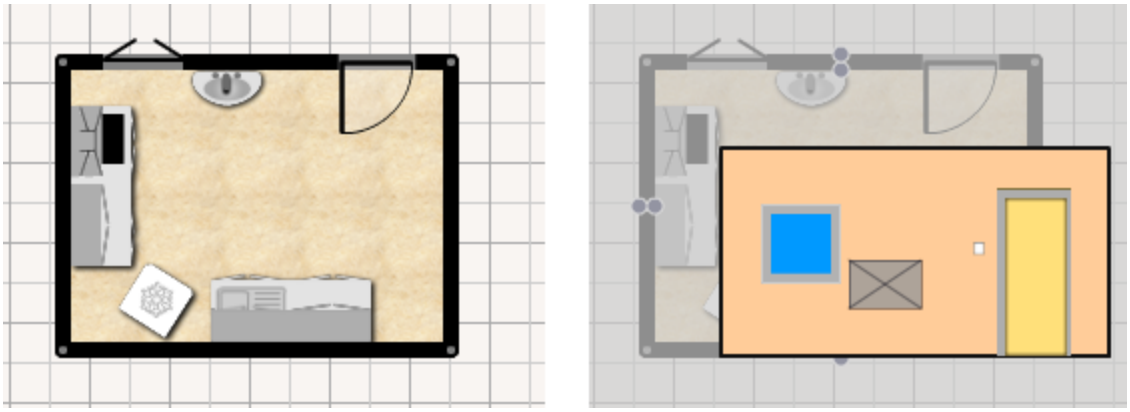

**Figure 12 Designer modes**

Figure 12 shows the designer modes: the *Normal mode* on the left and the *Face-wall mode* on the right. The red rectangle shows the edited side of the wall on the right.

The list of important classes representing the skeleton of the application is as follows:

The *Application* class consists of menus (class *MenuBox*), options (class *OptionsBox*), main working places (class *MainBox*) and tips (class *TipLabel*) as shown in figure 13. The class *MainBox* is considered to be the container of canvases for various modes. It also contains gauges (class Cgauge).

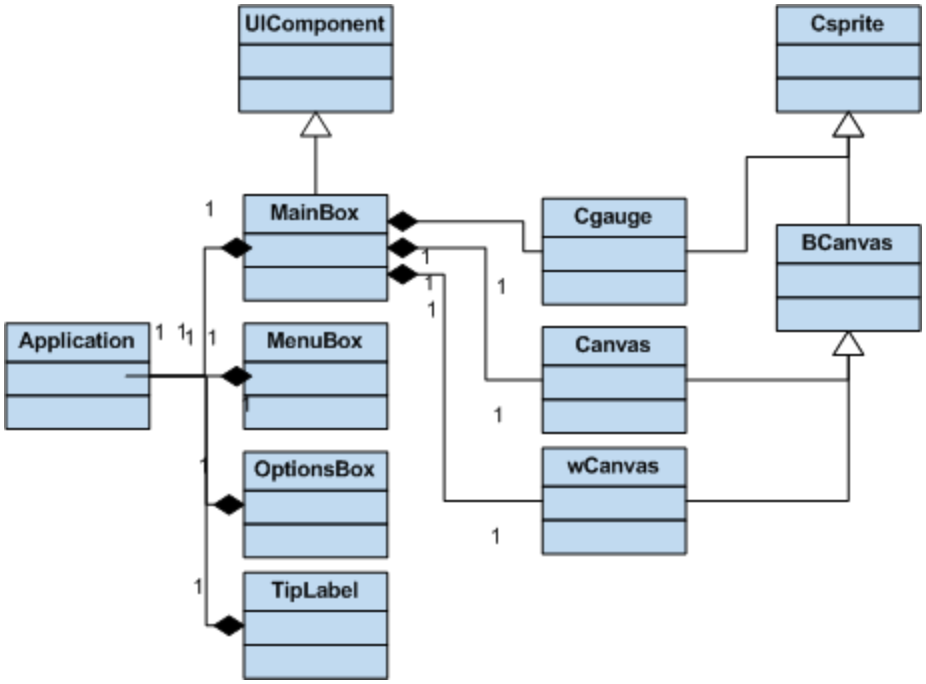


**Figure 13 The relationship between Main class and its parts**

- Class *MenuBox* is the container of all menu buttons. Each button represents a different action, e.g. placing a piece of furniture on the canvas, editing, deleting etc.

- Class *OptionsBox* is the container of all options buttons. Each button represents a different action, e.g. application settings, help, save, preview in 3D etc.

- An instance of *Canvas* class represents a place where objects are drawn while the designer is in the first (Normal) mode. The canvas can be considered a container of all pieces of furniture, walls, joints etc. The *Canvas* class also contains important mouse event-handling routines (on mouse button down, button up etc.)

- *wCanvas (Wall Canvas)* class represents the canvas appearing in the second designer mode. It is in many aspects similar to the Canvas class.

Note that *Canvas* and *wCanvas* have the same ancestor class *BCanvas (Base canvas)*. This class contains fundamental characteristics typical for a canvas, for example a depth management routine for child objects, routines dealing with cursors, scaling methods and so on.

### 6.1.3 Furniture base classes

All of the furniture base classes are descendants of *Dsprite* class. Figure 14 shows the hierarchy of furniture base classes.

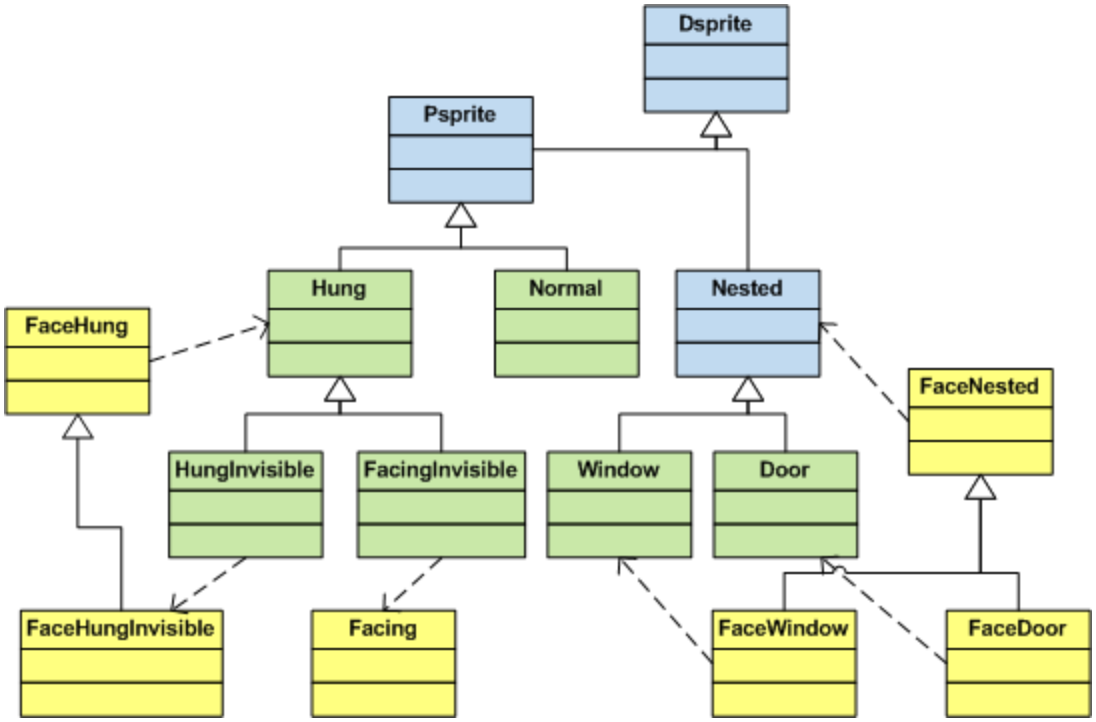


**Figure 14 The furniture base classes hierarchy**

The yellow-filled boxes represent the descendants of object classes that appear when the designer mode is switched to *Face-wall*. Similarly, the green boxes are descendants of objects appearing when the designer is in *Normal* mode. Note that dashed arrows depict their mutual mapping in different designer modes.

- *Normal (Normal type furniture)* – descendants of this class possess typical characteristics. They can be placed anywhere, rotated, scaled and colored.

- *Nested* – this class represents the ancestor of all objects that are "inside" of a wall or "nested" in a wall and can be seen from both sides. They are related to a particular wall. When the wall is moved, the nested type object related to this wall is moved too. The descendants of the Nested class are the *Window* and *Door* classes. In the second designer mode, instances of these classes are mapped to class instances with the same name prefixed by *Face*. Nested-type objects can be only scaled in the first mode; the second mode offers extended editing (coloring door cases and window frames, etc.) possibilities.

- *Hung (Wall-hung furniture)* – is an ancestor of all classes representing furniture that is wall-hung. An object of this type possesses additional properties in comparison to *Normal* type, e.g. a related wall. When the wall is moved, this object is moved too. Orientation of this object is the same as the wall orientation. A *FaceHung* type object is mapped to each *Hung* type object in *Face-wall* mode. *FaceHung* class introduces a face of wall-hung furniture. A wash basin is a fitting example.

- *HungInvisible* - has the same characteristics as *Hung*. This class is a descendant representing all small pieces of furniture that are not visible well in Normal mode (e.g. a painting). A *HungInvisible* object is mapped to *FaceHungInvisible* in *Face-wall mode*.

- *FacingInvisible* – similar to HungInvisible. It is a descendant of *Hung* class and represents all facings on the wall. It is mapped to *Facing* class in the second mode.

Note that all instances of classes representing faces of objects in second mode depend on a particular object from the first designer mode. This is caused by the fact that this object was created in the first mode and can only be edited in the second mode. Objects of type HungInvisible and FacingInvisible are the only exceptions as they depend on their Face-type objects – FaceHungInvisible and Facing. In other words, tile facing can only be added while editing the wall in the second designer mode and it is not visible in Normal mode.

### 6.1.4 Walls

*ProtoWall* class corresponds to a delineated wall. This descendant of *Csprite* class is basically represented by only 2 points – the start and end point of a line. In addition, it contains information about its above-mentioned nested objects (doors, windows), hung objects (a wash basin …), and hung-invisible objects (paintings …).

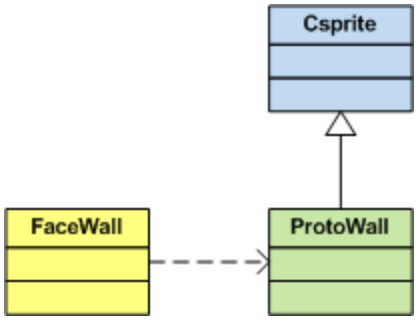


**Figure 15 The relationship between FaceWall and ProtoWall**

In the second designer mode, a wall is represented by an instance of FaceWall class. Figure 15 shows that class FaceWall is intuitively dependent on the ProtoWall class.

### 6.1.5 Glo class

The *Glo* class is a descriptive class of the FloorPAD application. It holds important data such as *planid*, *userid*, the list of prototypes and many more. Through this class, any language code string can be accessed. In order to maintain a single instance of this class at any given time, the Singleton design pattern [3] is used.

## 6.2 Depth management

Display programming in Flex can be performed in two ways: in the first, the displayed object is defined as a component and appears in the MXML definition and its display position is given by its position within the MXML file tree in or the object is added to the display list dynamically by calling the notorious ActionScript 3.0 method ***addChild(****objectinstance****)***. For the purpose of the designer, we use the second approach.
Before elaborating on designer's depth management in detail, let us introduce some theory. The basic, low level *DisplayObject* class is the core display class [11]. The *DisplayObjectContainer* class is its descendant. Any instance of this class can contain other DisplayObjects as children, which is an important feature. This class is in charge of imposing a kind of hierarchy among objects and a "parent-child" relationship between two objects.

The *DisplayObjectContainer* class is the foundation application's *depth management*. Each descendant of the basic class *DisplayObjectContainer* can contain other display objects, including other display object containers as well. The hierarchy of the displayed objects is known as the *display list*, see figure 16.

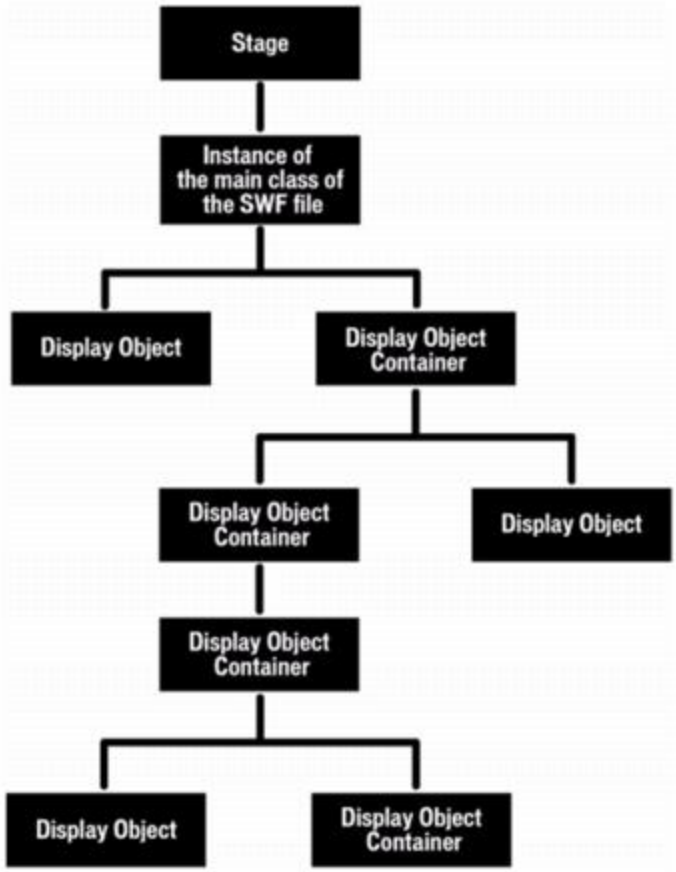


**Figure 16 Demonstration of display list structure**

The display list is a tree structure. The *DisplayObjectContainer* class includes properties and methods to traverse the display list by means of the child lists of display object containers [12].

The left part of figure 17 shows the situation where three display objects were added to a parent container's display list. The "a" object was added first, then the object "b" and eventually object "c". Object "a" is in position 0 of the display list, "b" is in the position 1 and "c" is on the top of the display list in the position 2. If the object "a" moved to the position 1, the object "b" would automatically move to position 0. The visual difference from the previous situation is shown on the right.


**Figure 17 Demonstration of a display list approach**

When a display object is moved to a new position in the child list of a DisplayObjectContainer instance, the other children in the display object container are repositioned automatically and assigned appropriate child index positions in the display object container. The display list can be easily traversed sequentially; there are no gaps in the index numbers of a child list of a display object container. In ActionScript 3.0, traversing the display list and managing the depth of objects is much easier than in the previous versions.

The display list is an interesting and effective approach to deal with depth management of displayed objects. Nevertheless, it is not sufficient for designer purposes. Different types of furniture have different heights and are seen differently from above. Each piece of furniture or even every object placed on the canvas is a display object (inherited from a display object). The default display list approach could cause some incorrect behavior like a carpet placed on a canvas after a table being positioned above the table which is incorrect.

These situations led to advanced categorization in the frame of display list itself. When a new display object (piece of furniture) is placed on the canvas (added to canvas child display list), advanced depth management methods find its exact position in the display list with respect to the *depth level* given by the object's type (e.g. type of furniture). The implementation of these methods is showed in appendix B, "Depth management core functions".

### 6.2.1 Depth levels

There are few depth levels in the normal mode of the designer. Table 3 includes all of them.

| Code | Constant name | Description |
|---|---|---|
| 0 | DEPTHFLOOR | Floor depth level. |
| 1 | DEPTHCARPET | Carpet depth level. |
| 2 | DEPTHFURUNDER | Depth level for small pieces of furniture, e.g. a chair (a part of it is placed beneath a table). |
| 3 | DEPTHFURNORMAL | Depth level for furniture. This value is default. |
| 4 | DEPTHFUROVER | Depth level of object types which are located "above" others. (e.g. small lamp placed on a table) |
| 5 | DEPTHWALL | Wall depth level. |
| 6 | DEPTHNESTED | Nested-type objects (doors etc ...) depth level. |
| 7 | DEPTHJOINT | Joint (an object connecting walls) depth level. |
| 8 | DEPTHPATH | Animation path depth level |
| 9 | DEPTHTEXT | Text field depth level |

**Table 3 Depth levels in Normal mode**

The higher the code, the higher depth index is given to a display object. For the purposes of new furniture type addition, there are some important constants: DEPTHCARPET, DEPTHFURUNDER, DEPTHFURNORMAL and DEPTHFUROVER.

Figure 18 demonstrates the situation in 2D. The floor object (DEPTHFLOOR) is at the lowest position in the display list at all times, followed by a level of carpets (DEPTHCARPET). The blue office chair represents furniture with DEPTHFURUNDER depth level, the Vika table is at DEPTHFURNORMAL level. The black laptop and a small lamp are at DEPTHFUROVER level. Note that in the 2D plan the windows and doors are seen over the wall, which explains why the value of DEPTHNESTED constant is higher than DEPTHWALL.

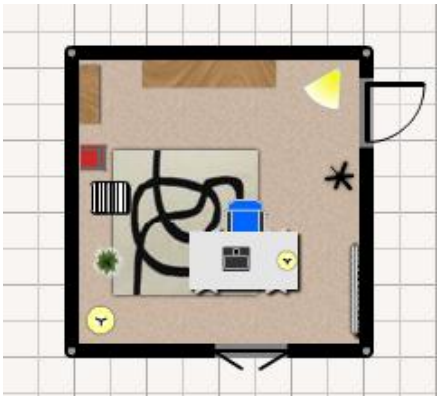

**Figure 18 Demonstration of depth management in Normal mode**

When an object at DEPTHFUROVER level is placed over an object with DEPTHFURNORMAL level, the object's height from the ground is added to the height of the object on which it is placed. This can be seen in figure 19 where the laptop at DEPTHFUROVER level is placed on the table at DEPTHFURNORMAL level.

**Figure 19 Designed scene in 3D**

### 6.2.2 Depth levels in Wall-edit mode

The wall-edit mode features a similar approach to object depth management. Table 4 shows the overview.

| Code | Constant name | Description |
|---|---|---|
| 10 | DEPTHFACEWALL | Wall face depth level |
| 11 | DEPTHFACEFACING | Facings (tiles, ...) depth level |
| 12 | DEPTHFACENESTED | Nested objects' (faces of doors, …) depth level |
| 13 | DEPTHFACEHUNGINVISIBLE | Small wall-hung objects' (portraits …) depth level |
| 14 | DEPTHFACEHUNG | Depth level of wall-hung objects' face from the first mode |

**Table 4 Depth levels in Wall-edit mode**

Figure 20 shows an example of an edited side of a wall. The face of the wall is at DEPTHFACEWALL depth level, being beneath all other objects. It is followed by tiles (DEPTHFACEFACING) and doors and windows (DEPTHFACENESTED). Small, wall-hung objects like lamps and switches are at DEPTHFACEHUNGINVISIBLE level. Hung objects are above all other objects at level DEPTHFACEHUNG.

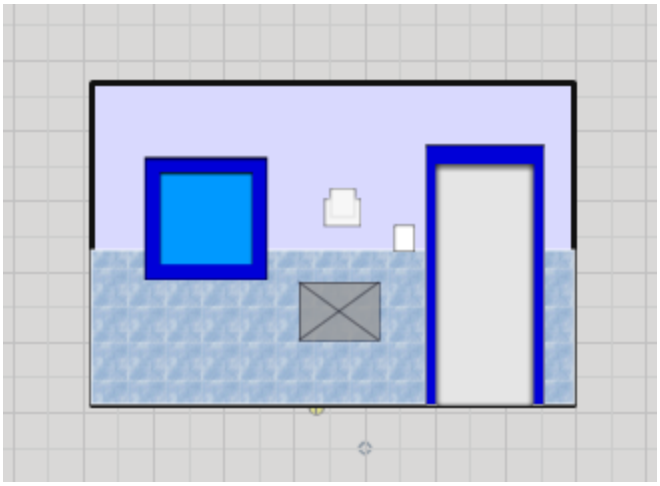

**Figure 20 Bathroom wall in 2D**

Figure 21 shows the 3D view of the edited wall.

**Figure 21 Bathroom wall in 3D**

## 6.3 Object descriptions

From the top of the inheritance hierarchy (class *Bsprite*) to the lowermost classes representing pieces of furniture, classes feature methods to generate XML files according to their properties and references to other objects. Each XML fragment contains basic information about the parent class name, type name, instance name, exact canvas position and orientation. In addition, more information according to type of the object can be stored, for example scaling ratios for width, height, length and ground elevation, part color and references to other objects.

Figure 22 shows an example of an XML fragment describing a wash basin object. This object is a descendant of wall - hung objects (tag c: Proto::Hung), thus preserving all of the properties of the class Hung. The prototype name is ProtoWashBasinSimple; for more information, see appendix A, "Furniture prototypes". It can be seen that the graphics of the wash basin is composed of two parts. The first part (Part1) can be coloured (tag colourable = 1) and the second one (Part2) cannot. It is also apparent that the basin is related to the wall (ProtoWall__5), on which it is placed.

```
<obj>
  <c>Proto::Hung</c>
  <t>ProtoWashBasinSimple</t>
  <n>Hung__10</n>
  <x>-121.95</x>
  <y>-213</y>
  <o>0</o>
  <sx>1.1112</sx>
  <sy>0.8888</sy>
  <sz>1</sz>
  <se>1.20833</se>
  <parts>
    <part>
      <name>Part1</name>
      <colorable>1</colorable>
      <r>204</r>
      <g>153</g>
      <b>255</b>
      <gl>1</gl>
      <sh>1</sh>
    </part>
    <part>
      <name>Part2</name>
      <colorable>0</colorable>
      <r>0</r>
      <g>0</g>
      <b>0</b>
      <gl>0</gl>
      <sh>0</sh>
    </part>
  </parts>
  <wall>ProtoWall__5</wall>
  <wallRP>0.5142156862745099</wallRP>
</obj>
```

**Figure 22 XML description of an instance of wash basin**

This approach allows for complete recreation of objects according to XML data. Drawing and saving of created plans is based on this concept. Each plan is saved to an XML file that contains descriptions of all drawn objects. When the plan is being loaded, the XML file is read and objects are restored according to XML values.

The restoration of objects works in three stages:

1. Object creation according to exact class name
2. Properties setting ( position, rotation, coloration, …)
3. Setting of all references according to valid names of existing objects

To implement an effective way of *undo* (previous) and *redo* (next) actions execution, the *memento* design pattern was used [3]. The new state of the whole plan is saved to the application stack after each action (placing a new object on the canvas, moving an existing object to a new position, colouring a part of furniture etc.). Applying undo or redo actions means picking up a state on the top of stack and changing the state of the whole plan. In this case, an otherwise complicated comparison of whether an object has been changed in the last action is reduced to simple string matching. This leads to an effective way to perform undo / redo actions. The designer allows saving of up to 30 states on the stack.

## 6.4 3D generation

### 6.4.1 Server side transformation

The Flex designer sends the XML file containing information about all drawn objects to the server. The XML is manipulated "on the fly" in memory. Before running the transformation process itself, a few adjustments have to be made:

- The range of RGB values have to be normalized
- The pixel values have to be recalculated to meters

As was explained in section 4.1.3, XSLT transformation is applied to the XML to produce the X3D output. Transformation rules are described in `PlanTransform.xsl` file. For more information, see appendix B, "Scene transformation".

### 6.4.2 The generated 3D scene

The newly created x3d file contains the actual scene. 3D objects (furniture) are loaded from external x3d files using the *ExternProtoDeclare* node. Representation of walls, floors and tile scopes are the only exceptions. In addition two viewpoints are generated automatically:

- *2D alike* - a view from above the plan
- *Animation* - rotations around the *z* axis

### 6.4.3 Walls, floors and tiles

Each wall has two sides coloured differently. Each side of a wall is broken into blocks defined by nested objects – doors and windows. The red lines in figure 23 demonstrate the borders of particular wall blocks. In X3D, each block is represented by a *Box* node.

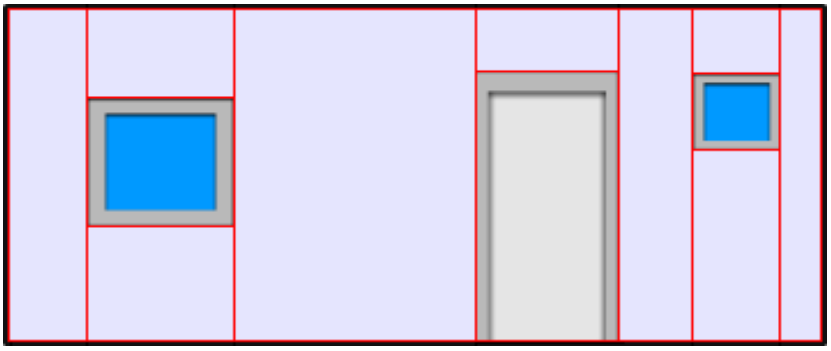

**Figure 23 Division of a wall into members**

Since the wall is represented by only two points (*from* and *to*), it is necessary to calculate "skewer" in joint places where the corresponding walls meet. Figure 24 shows an example where the calculated skewer is marked in red.

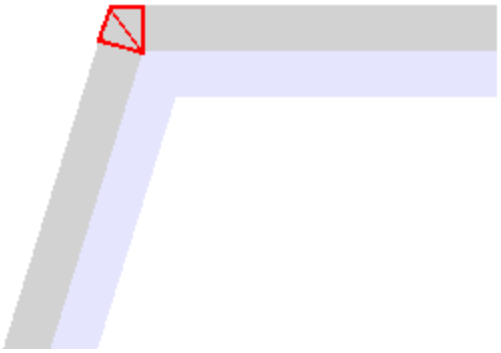

**Figure 24 Example with skewer**

The floor is described by points determined by wall joints (instances of *ProtoJoint* class) and is textured according to the fill. Proper transformation is applied on the texture according to the floor area.

The representation of the tiled area in 3D is similar to the wall representation. In the designer, a single continuous plate (area of tiles) is broken into several tile plates defined by the nested objects (doors and windows). The tile texture is then scaled and transformed to fit in same pattern in different plates.

### 6.4.4 Web traffic reduction

In order to reduce traffic over the Web, the generated X3D output is automatically compressed (using a *gzip* routine). The .NET framework supports the .zip format. Prototypes of existing furniture are compressed as well.

## 6.5 Code culture

Functions, variables and files are named in a descriptive way (we eschew ambiguous abbreviations). Nouns are used for types and variables, while "command" verbs (imperative) are reserved for functions. Function names are as descriptive as possible to make the code immediately understandable to a newcomer.

An ActionScript 3.0 class is defined and stored in a file with the same name – this rule is enforced by the Flex framework by default. This could imply that each class is located in a separate file which is usually true, but there are exceptions to this rule. User interface elements, for instance, are created using MXML components. For each MXML component name, one additional file with the same name and the suffix "code" exists. This file contains additional logic written in ActionScript 3.0. This way, application design can be separated from the application logic. The use of these components enhances code reusability and maintainability of the whole system.

Related classes are logically grouped into packages, e.g. classes responsible for Communication can be found in the package "communication", some classes dealing with geometric computations are in package "MatH" etc.

Design patterns [3] were applied to some problems. To implement *Undo/Redo* functionality, the *Memento* design pattern was used. The classes belong to separate package "Memento". The global class *Glo* is yet another example of the *Singleton* design pattern, as it was mentioned before in part 6.1.5 . To create exact copies of selected objects, the *Prototype* design pattern was used. The *clone()* method had to be implemented properly This methods have every descendants of *Dsprite* class, see chapter 6.1., "Fundamental classes".

## 6.6 Application customizability

Throughout the development process, great emphasis was placed on the possibility to integrate the application with other existing applications (e.g. to integrate the designer into third-party real estate web pages, or to send/receive data from/in the designer). The overall appearance can be modified just by editing a single CSS file.

# 7 Scenario-based user guidance

## 7.1 “Normal” user

A regular user may use this application to design a new apartment, sketch moving furniture or simply to fool around. Let us consider the design of a floor plan of a common flat. We provide a guide describing the process step by step.

1. Prepare the canvas, estimate the size of the whole plan, use mouse wheel to zoom in/out the canvas.

2. Sketch approximate rooms by drawing rectangles (figure 25).

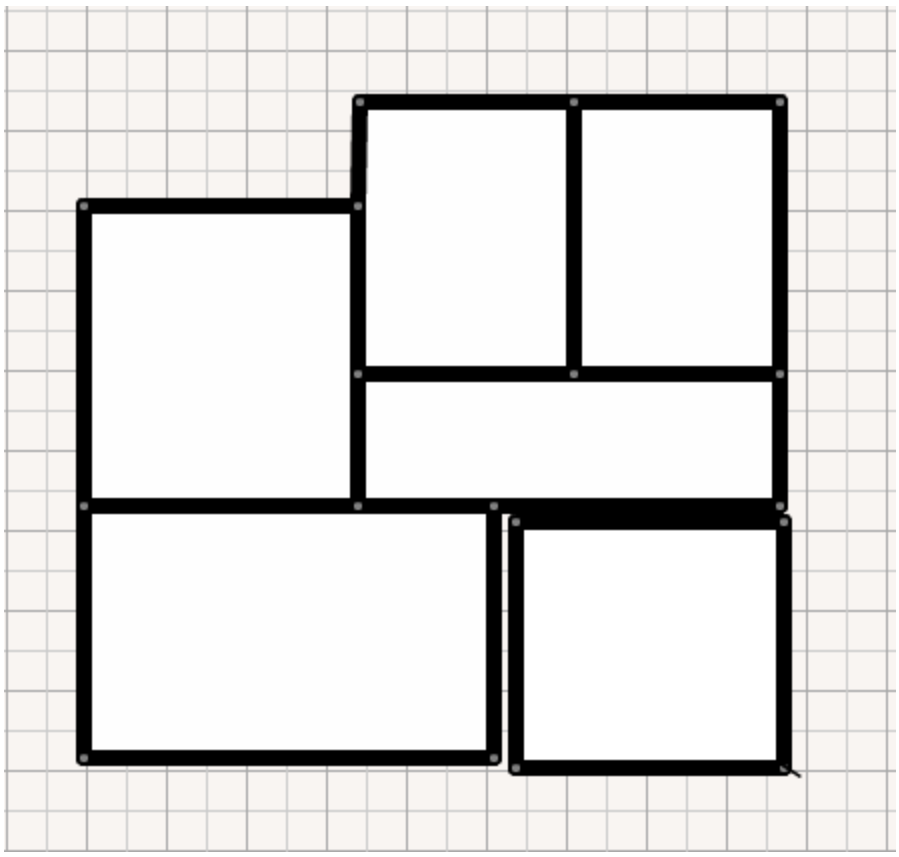

**Figure 25 Room design**

*Note that rectangles (rooms), which are close enough (in time of drawing) with their lines (walls), can be partly merged into conjunctive shape.*

3. Adjust and elaborate on the sizing of the room. Add wall joints to create atypical shapes, see figure 26.

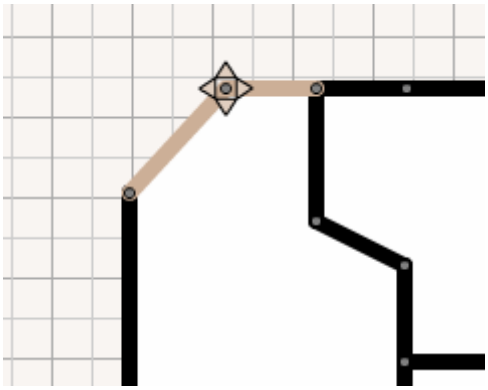

**Figure 26 Adjusting atypical shapes**

*Note.: The joint can be removed by merging it with his neighbouring joint.*

4. Add doors and windows as shown in figure 27.

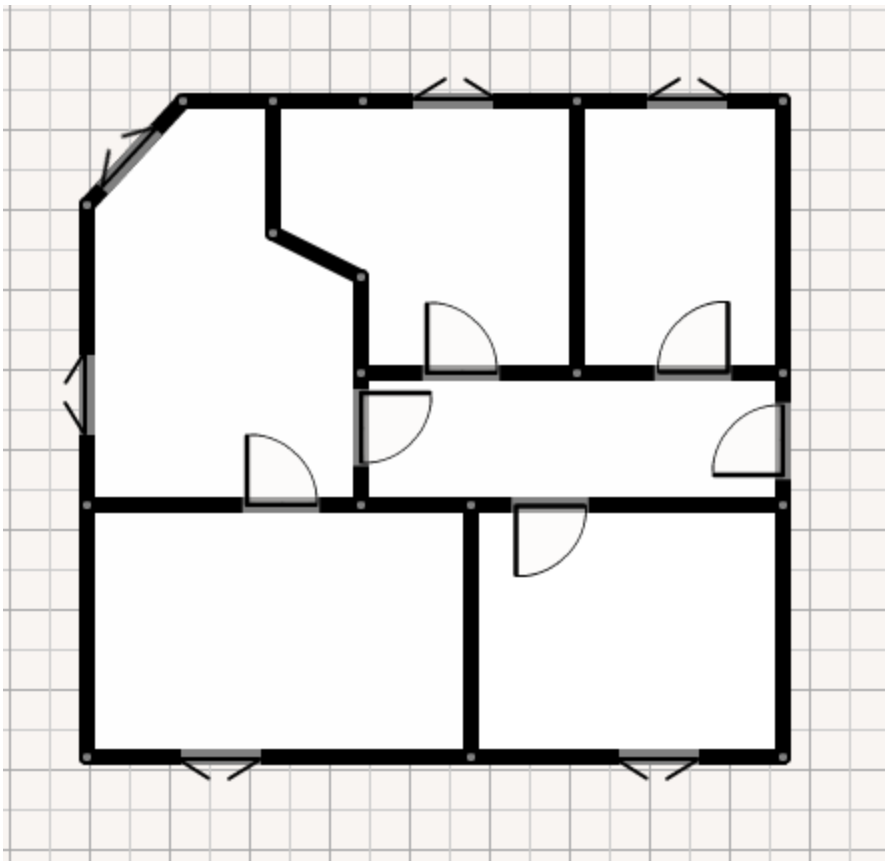

**Figure 27 Floors with doors and windows**

*Note that doors and windows will automatically "jump" onto the nearest wall.*

5. Fill the floor with suitable patterns; paint the inner walls by double-clicking on the floor in a particular room as shown in figure 28.


**Figure 28 Pattern selection**

6. Add pieces of furniture. *Note that wall-hung objects (e.g. a wash basin) automatically position themselves on the wall.*

7. Color, rotate and change the size of furniture (figure 29)

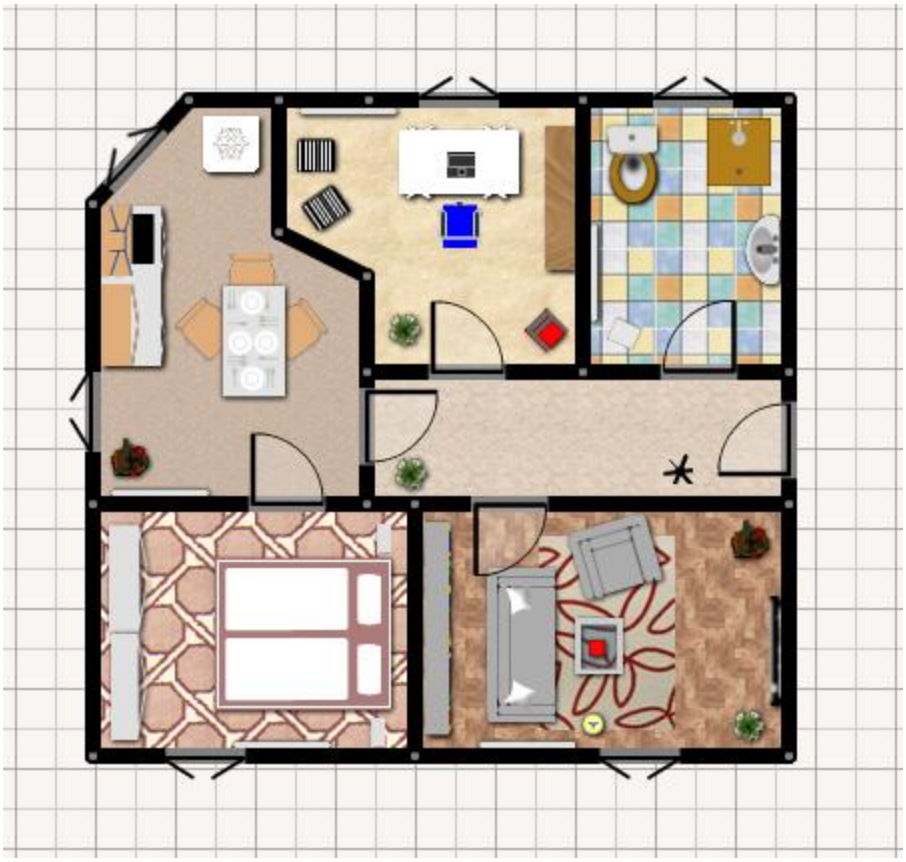

**Figure 29 A furnished flat**

8. If necessary, adjust properties manually by double clicking on selected objects, see figure 30.

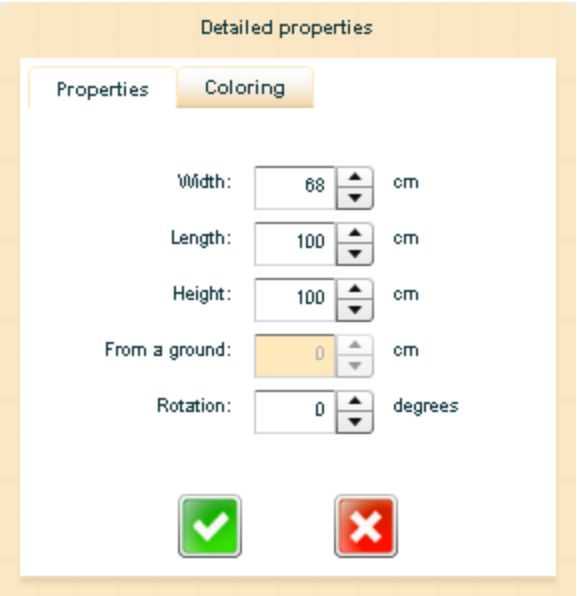


**Figure 30 Advanced object's properties**

9. Edit walls, add tiles and small wall-hung objects.

10. Add 3D viewpoints and name them properly as shown in figure 31.


**Figure 31 Viewpoint properties**

11. Draw an animation path (figure 32).

**Figure 32 Animation path**

*Note that each animation waypoint has its own properties (figure 33).*

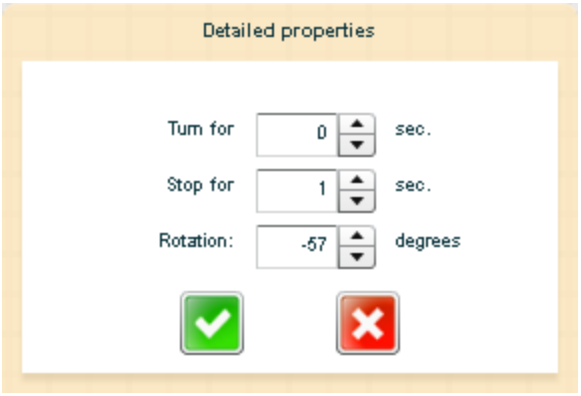


**Figure 33 Animation waypoint properties**

12. Finally, generate the 3D scene.

**Figure 34 "2D alike" viewpoint**

Figure 34 shows 3D scene from the first of the two generated viewpoints - “2D alike” viewpoint. The “Animation” viewpoint is showed in figure 35.

*Note that 3D presentation is also dependent on the used X3D player. These pictures were taken while using Flux X3D player.*

**Figure 35 "Animation" viewpoint**

For more detailed information please refer to the Appendix C, “User documentation”.

## 7.2 Case study for real estate professionals

From the author’s own experience, real estate professionals are not particularly interested in 3D presentations. On the contrary, they are more interested in the simplicity of drawing algorithms and pure text descriptions.

Usually, they do not want to spend more than a few minutes creating floor plans. The only goal is to sell or to rent the offered estate and the possibility to draw a floor plan is a just a way to improve the presentation layer thus attracting more potential customers. The sale starts with the personal estate presentation.

Let us consider the drawing of a plan of a typical flat by a real estate professional. Points 1 - 4 are the same as in the previous section. We then assume we have the floor plan in the state shown in the figure 27.

The remaining activities are as follows:

5. Add text descriptions (figure 36).

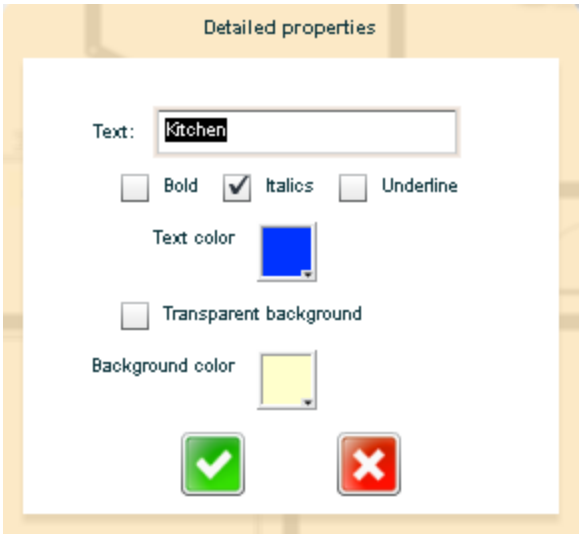


**Figure 36 Text field property**

6. Add quota fields (figure 37).


**Figure 37 Quotas example**

*Note that text fields can be also resized using "selection tool" (like the other objects).*

7. Save the designed plan as a picture (in PNG format), see figure 38.

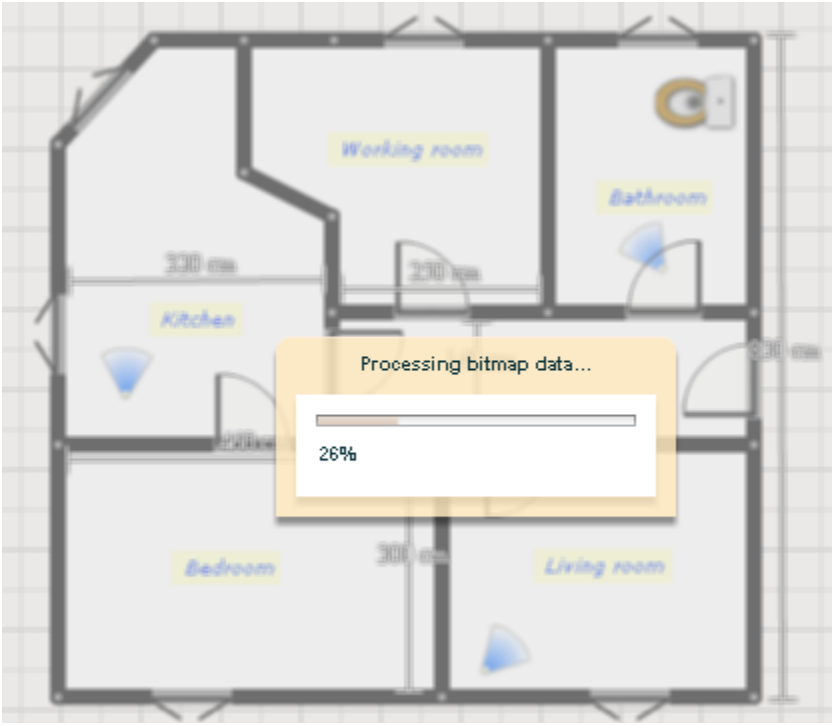


**Figure 38 Saving plan as a picture**

8. Print out the designed plan by clicking on Print button.

9. Add photo indexes to the plan, figure 39 (assuming there is a protocol to load photos into FloorPAD from an external real estate information system)


**Figure 39 Plan with photo indexes**

10. Publish the final presentation of the floor plan with photos (figure 40).

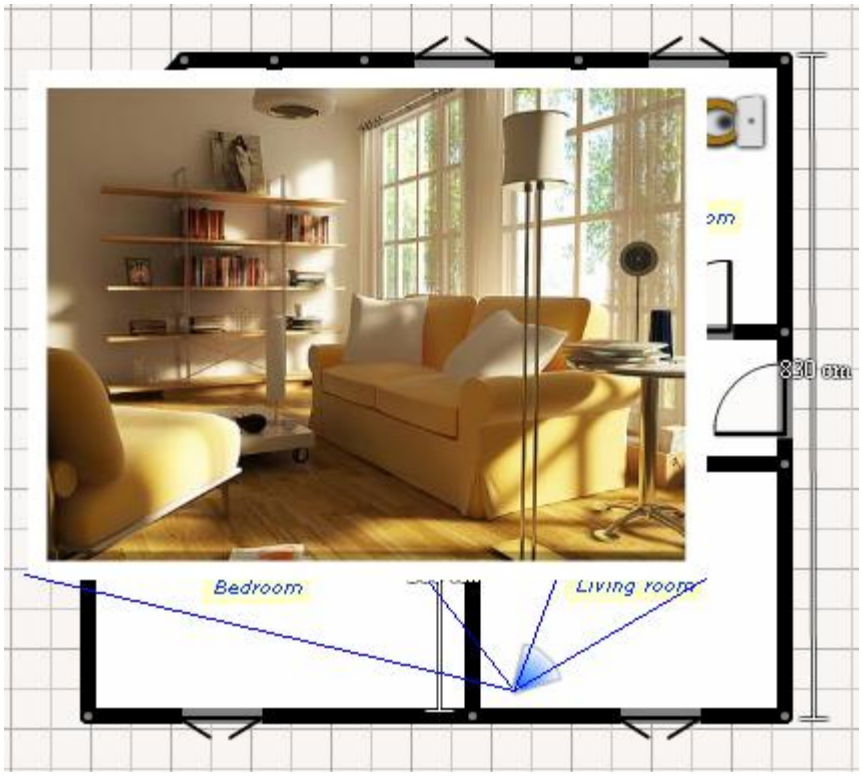


**Figure 40 Plan with photo**

*Note that the company logo can be placed on the plan easily.*

Emphasis is placed on the speed of plan design. Any experienced user of the designer, familiar with most of the key shortcuts, should be able to draw a whole plan within 2 minutes. See the Appendix C, "User documentation", for the complete list of keyboard shortcuts.

Adding photo indexes to floor plans brings in a new point of view. The floor plan shows how the rooms are organized, how they are shaped, whether the doors open inward or outward etc. The photo indexes add useful information on how a particular room looks like from various points of view. This is the main advantage to the common view based only on a set of photos.

# 8 Conclusions and future work

## 8.1 Main achievements

The FloorPAD is an intuitive and easy to use application to draw floor plans and subsequently generate 3D presentations. Advanced drawing algorithms allow the user to design plans in a matter of minutes, to place and edit various furniture types, insert viewpoints or designate paths for the 3D animation. Most of the important actions have keyboard shortcuts assigned to speed up the design process.

The FloorPAD is an online web application. This sets it apart from other applications, which have to be installed locally. The Flex designer is lightweight and is downloadable in a matter of seconds, depending on connection type. The application is built using modern technologies like Flex, the .NET framework, X3D and XSLT and demonstrates the way how these technologies can communicate and cooperate together.

The other advantage is the furniture extensibility framework. It allows adding of new pieces of furniture without having to re-compile the designer's SWF file. Another customizing feature lies in the simplicity of changing the overall look only by editing the CSS styles. In addition, the whole concept of the application allows to be easily connected with the existing systems.

The FloorPAD application can be considered as a tool to generate X3D files. This format can be imported and further processed by other graphics applications. The application can be also used as a professional tool to improve presentations of estate agencies. Apart from creating floor plans, the designer allows setting quotas and inserting text notes. With slight modifications, the designer allows to combine floor plans with photo indexes (this functionality is particularly interesting for the estate professionals). The final plan can be also saved as a picture in PNG format or printed out.

Within the frame of this thesis, the author has opportunity to improve his skills in many technologies like .NET, X3D and especially Flex. Customizing basic Flex components, creating new events, overriding the basic component functionalities also belongs to knowledge gained while writing the thesis.

The designer was tested only by a handful of people, who were pleased by the simplicity of a typical well-known Paintbrush program with intuitive adjustments to create impressive 3D scenes. Remarks of these users will be included in the future works. Some testimonials are included in the appendix D.

## 8.2 Future work

FloorPAD has potential for improvement in some aspects. The future work mainly lies in the functionality extensions. From the drawing point of view, functionality like a proportions display in meters while designing rooms or walls should be implemented. The selection tool may even work for multiple objects, the same principle could apply to object deletion. The designer could also load other texture types (in the current version, only square textures 40 by 40 pixels in size are supported).

The depth management of existing pieces of furniture could be more precise and stratified. Wall editing could be more flexible, allowing for new features such as carving out archway curves or building-in shelves. Photos could be placed on walls. Some unusual wall parts could be shown in this way.

The application will contain many more pieces of furniture, door and window types. Some pieces of furniture will be allowed to be placed on the canvas together at the same time allowing the creation of "combos", e.g. a table and kitchen chairs. The kitchen unit could be dynamically completed with a fridge, a cooker, a washing-machine and other equipment.

Each piece of furniture could be viewed not only from above (current version) but also from the front and sides. Multiple types of objects could be allowed to be chosen for a single model type, e.g. a toilet icon in 2D could represent a multitude of types of toilet models in 3D. Each object could be previewed in X3D. With the development of 3D support in Flash, there is a great chance that the 3D generation will be also rewritten in a Flash-related framework such as Papervision3D, Sandy3D etc.

At this moment the FloorPAD designer allows to create only one floor per plan. An upgrade to enable drawing multi-storey buildings would not be expensive. The application can be even extended to design the outside environment (e.g. a garage, a garden with a set of trees, bushes and flowers).

More work can be done to support the presentation web page, www.floorpad.eu. It mainly lies in marketing support, SEO optimization, blog writing, partner link building (furniture companies, estate agencies etc.) Another way to increase the popularity of FloorPAD application could lie in penetration of the social networks such as Facebook. The published floor plan could be embedded into other web pages via a small chunk of code (such as YouTube).

With some modifications, the FloorPAD can be extended to a desktop application by using Adobe AIR framework. The PDA would be a possible target platform.

# Bibliography


[1] Žára J., Beneš B., Sochor J., Felkel: Moderní počítačová grafika. Computer Press, 2005.

[2] Lott J., Schall D., Peters K.: ActionScript 3.0 Cookbook, O'Reilly, 2006.

[3] Sanders W., Cumaranatunge Ch.: ActionScript 3.0 Design Patterns, O'Reilly, 2007.

[4] Brutzman D., Daly L.: Extensible 3D graphics for web authors, Morgan Kaufmann, 2007.

[5] UML Tutorial, Types of UML diagrams
http://atlas.kennesaw.edu/~dbraun/csis4650/A&D/UML_tutorial/diagrams.htm

[6] W3C Extensible Markup Language (XML) 1.0 (Fourth Edition)
http://www.w3.org/TR/2006/REC-xml-20060816/

[7] Adobe Action script definition
http://www.adobe.com/devnet/actionscript/

[8] ActionScript 2.0 Migration
http://livedocs.adobe.com/flash/9.0/ActionScriptLangRefV3/?migration.html&all-classes.html

[9] ActionScript Virtual Machine Overview
http://www.adobe.com/devnet/actionscript/articles/avm2overview.pdf

[10] Flash Player Penetration statistics
http://www.adobe.com/products/player_census/flashplayer/

[11] ActionScript 3.0 core display object reference
http://livedocs.adobe.com/flash/9.0/main/wwhelp/wwhimpl/common/html/wwhelp.htm?context=LiveDocs_Parts&file=00000143.html

[12] Advantages of display list programming in ActionScript 3.0
http://livedocs.adobe.com/flash/9.0/main/wwhelp/wwhimpl/common/html/wwhelp.htm?context=LiveDocs_Parts&file=00000144.html

[13] Importing Flash CS3 Assets into Flex
http://livedocs.adobe.com/flex/3/flash_component_kit_flex3.pdf

[14] Web3D Consortium - X3D Overview and Features at a Glance
http://www.web3d.org/about/overview

[15] W3C Recommendation XSL
http://www.w3.org/Style/XSL/

[16] W3C Recommendation XSL Transformations
http://www.w3.org/TR/xslt.html

[17] Sweet home 3D designer
http://sweethome3d.sourceforge.net/cs/

[18] FloorPlan 3D presentation
http://www.floorplan.cz/index.php

[19] IKEA planner
http://www.ikea.com/ms/en_US/rooms_ideas/splashplanners.html

[20] Google SketchUp web page
http://sketchup.google.com/

[21] ArchiCAD product page
http://www.graphisoft.com/products/archicad/

[22] Scene Caster web page
http://www.scenecaster.com/web/home.php

[23] Floor planner web page
http://www.floorplanner.com/

[24] Dragonfly project web page
http://dragonfly.autodesk.com/

[25] X3D viewers and browsers
http://www.web3d.org/tools/viewers_and_browsers/

[26] The BS Contact player web page
http://www.bitmanagement.com/products/bs_contact_vrml.en.html

[27] The Cortona player web site
http://www.parallelgraphics.com/products/cortona/

[28] The Flux 2 player web site
http://www.mediamachines.com/downloadplayerty.php

[29] The FreeVRML plug-in
http://freewrl.sourceforge.net/download.html

[30] The Octaga player web site
http://www.octaga.com/download_octaga.html

[31] The official Microsoft ASP.NET site
http://www.asp.net/

# Appendix A – Furniture extension framework

In order to understand the whole process of adding new pieces of furniture, we need to explain the designer's configuration files, also the way how the piece of furniture is defined in 2D and in 3D. After that we provide a simple guide (featured with example) how to add another piece of furniture into application.

## Application configuration

The FloorPAD configuration is given by a set of XML files. They provide additional descriptions about menu items, prototypes, loaded textures and language descriptions. The configuration files can be found in the `media/Config` directory.

### Descriptions

The file `en_descs.xml` contains a list of numeric codes and corresponding descriptions in the English language. Each description, tooltip or a message is contained in this file. For example: In figure 41, the element "ms" (message) contains child elements "cd" (code) with value "50" and element "d" (description) with value "Building". In other words, the description string "Building" is represented by code 50.

```
<!-- Building category -->
<ms>
  <cd>50</cd>
  <d>Building</d>
</ms>
<!--  Kitchen category -->
<ms>
  <cd>100</cd>
  <d>Kitchen</d>
</ms>
<ms>
  <cd>201</cd>
  <d>Frige</d>
</ms>
```

**Figure 41 Code list fragment**

Using this approach, the matter of multi-language support is reduced to a simple task of loading the proper language description file.

## Categories

The file `Categories.xml` contains a list of basic menu and option categories as shown in figure 42.

```
<c>
  <n>None</n>
  <cd>1</cd>
</c>
<c>
  <n>Building</n>
  <cd>50</cd>
</c>
<c>
  <n>Kitchen</n>
  <cd>100</cd>
</c>
```

**Figure 42 List of categories**

Note that each element "c" (category) has these child elements:

- "n" (name) – represents the code name of a particular category, e.g. "Building", "Kitchen" etc.
- "cd" (code) – the value of this element has to exist in the description file

Note that the "Building" category in figure 42 corresponds to the menu category "Building" in figure 43.

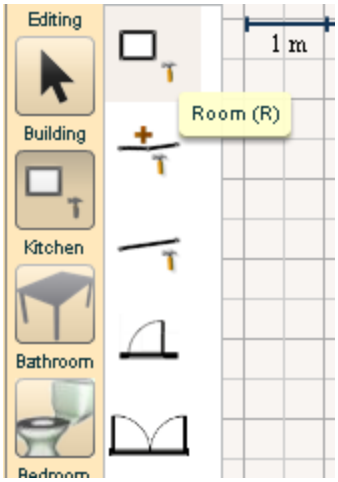


**Figure 43 A segment of menu categories**

Categorization provides grouping of menu items bearing similar characteristics. For example, the code "Kitchen" groups all pieces of furniture that are kitchen-related.

### Menu configuration

The file `MenuItems.xml` contains the list of menu items. Figure 44 shows a fragment of this file.

```
<!-- Buinding  -->
<i>
  <cg>Building</cg>
  <pt>CarrierRoom</pt>
  <k>82</k>
  <im>../media/Pics/MenuItems/ProtoRoom.png</im>
  <cd>12</cd>
</i>
<i>
  <cg>Building</cg>
  <pt>ProtoJoint</pt>
  <k>74</k>
  <im>../media/Pics/MenuItems/ProtoJoint.png</im>
  <cd>11</cd>
</i>
```

**Figure 44 A fragment of menu items XML file**

Each menu item is represented by the "i" element and its child elements are:

- "cg" (category) – the value of this element has to exist in `Categories.xml`
- "pt" (prototype) – defines action (e.g. drawing a wall) or a particular furniture prototype
- "k" (key) – ASCII code for a keyboard shortcut or value -1
- "im" (image path) – represents the exact file path of a menu image
- "cd" (code) – description code, has to exist in the description file

### Furniture prototypes

The file `Prototypes.xml` contains the list of prototypes definitions. Each prototype is represented by the "proto" element and its child elements are:

- "base" (base class) – the furniture prototype inherits all properties from this base class
- "type" (prototype) – defines action (e.g. drawing a wall) or a furniture prototype
- "sizing" (key) – describes the boundaries of "bounding box" (with its sides aligned to coordinate system axes) of the smallest dimensions for the object to fit in.
    - Note that designer automatically recounts all sizing values according the attribute "measure". The values are either in "cm" or "px".
    - Its child elements are:
        - "w" – object's "bounding box" width
        - "h" - object's "bounding box" height
        - "l" - object's "bounding box" length
        - "e" – elevation from a ground
- "scaling" – scaling values from different dimensions
    - Its child elements defines the minimal and maximal scaling possibilities (in percentage). E.g. tag "minw" (minimal width) with value 0.7 means that object can be scaled down to 70 % of its normal size.

- “depth” - depth level code – for more info, see the chapter 6.2, “Depth management”
- “tooltip” – tooltip code
- “parts” – this element contains another (one or more) “part” element. It defines the part of furniture piece, it has to be properly mapped to parts in 3D model.
    - Note that the position of parts within their local coordinate systems varies according to object types. For normal types (descendants of class Normal – value of tag “base”), e.g. a single bed, the coordinate origin equals to the crossing point of diagonals of their bounding rectangles, whereas wall-hung types objects (descendants of classes Hung or HungInvisible), are moved within their respective local coordinate system so that the coordinate center is located on the border adjacent to the wall.
    - In addition each element “part” contains different attributes:
        - “pos” – position in z-ordering is required!
        - “col” – color value
        - “fill” – flag whether that part is colorable (value 0) or not (other value, or does not exist this attribute)
        - “glow” – a size of glow filter applied on that part
        - “shadow” – a size shadow filter applied on that part

Figure 45 shows a fragment of this file, a definition of one prototype.

```
<proto>
  <type>ProtoKitchenSinkComplex</type>
  <base>Normal</base>
  <sizing measure="cm">
    <w>200</w>
    <h>80</h>
    <l>200</l>
    <e>0</e>
  </sizing>
  <scaling measure="%">
    <minw>0.7</minw>
    <maxw>1.3</maxw>
    <minh>0.7</minh>
    <maxh>1.3</maxh>
    <minl>1</minl>
    <maxl>1</maxl>
  </scaling>
  <depth>4</depth>
  <tooltip>211</tooltip>
  <parts>
    <part pos="1" col="0xCCCCCC" glow="1" shadow="2" />
    <part pos="2" col="0x999999" glow="1" shadow="0" />
    <part pos="3" fill="0" />
  </parts>
</proto>
```

**Figure 45 Prototype definition**

This configuration files `MenuItems.xml` and `Prototypes.xml` are essential for adding new pieces of furniture definition.

### Textures

The configuration file `Textures.xml` contains information about textures that are loaded into the designer.
Each element "i" (item) describes a single texture. Its child elements are:

- "cg" (category) – the value of this element has to exist in `Categories.xml`
- "im" (image path) – represents the exact image file path with respect to designer path
- "cd" (code) – represents the description code existing in the description file.

```
<i>
  <cg>OptionFloor</cg>
  <im>../media/Pics/Textures/FLOOR_2OWOODDARK.png</im>
  <cd>930</cd>
</i>
```

**Figure 46 A fragment of texture configuration file**

Figure 46 represents the texture of parquets that appears in the designer in figure 47.


**Figure 47 Texture list in the designer**

Adding a new texture to the application is reduced to 3 steps:

1. Creating a new texture file
2. Adding the new texture description to `Textures.xml` file:
    a. Category code – provides information whether the texture is of "floor fill" or "tile" type
    b. Exact path to texture file relative to designer application path
    c. Description code

Adding new description code and value to the descriptions file (e.g. `en_descs.xml`) mentioned in step 2.c.

## Furniture in 2D

There are strict rules on how to define a furniture piece in 2D. Each furniture consists of some parts defined in `Prototypes.xml`. In other words, each element "part" in configuration file is related with one part of furniture, represented by one picture. The pictures are placed one on the other. The z-order of this picture is given by value of attribute "pos" in element "part" in corresponding prototype fragment code in configuration file.

The pictures (in PNG format) are stored in directory `media/Pics/Prototypes/{value of element "type"}` and named like `"Part" + {value of attribute "pos" in element "part"} + ".png"` – e.g. Part0.png, Part1.png, Part2.png etc. Figure 48 shows the parts of Kitchen sink complex. The red border is demonstrating the picture sizing and white color inside this border is transparent color. *Note that all part pictures defining one piece of furniture have to have same picture **size**!*

**Figure 48 ProtoKitchenSinkComplex parts**

## Furniture in 3D

Each 3D model of a piece of furniture is defined as a prototype with parameters in a separate file in X3D format (.x3d). **The prototype name** and the **file name** have to be the same. The application design assumes that each piece of furniture consists of at least two parts differing in color; the prototype interface has to contain at least these two parameters of type SFColor – **Part1** and **Part2**. All names are **case sensitive**.

Figure 49 shows the tree structure in a file defining a piece of furniture. It contains two nodes – the declaration of prototype (node ProtoDeclare) and prototype instance (node ProtoInstance). Node ProtoDeclare contains a prototype interface (node ProtoInterface) with declaration of parameters and a prototype body (node ProtoBody) containing a model description. The parameters from the prototype interface are mapped to the matching types of model parts. Node ProtoInstance is intended only to check, whether the prototype declaration is working properly.

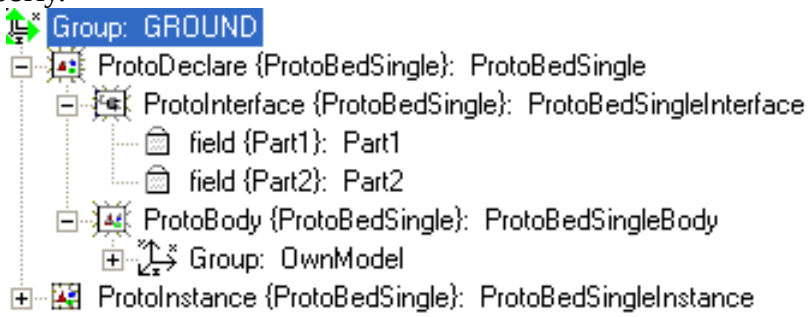


**Figure 49 ProtoBedSingle.x3d file content**

The prototype interface can also contain other parameters, but the XSL transformation definition file has to be **adjusted** for concrete types of parameters and prototypes.

## Local model coordinates

For the purpose of this application, the right-handed Cartesian coordinate system is used in the model design. The coordinate axes are marked in red for the x axis, blue for the y and green for the z axis.

The displacement of the model in its respective coordinate system depends on its type ("normal" or "wall-hung"). Note that each model can be imagined as being placed in a "bounding box" (with its sides aligned to coordinate system axes) of the smallest dimensions for the object to fit in; this visualization is needed to understand the following explanation.

The local coordinate system origin of "normal" type objects coincides with the center of the "bounding box" in the x and y axes and the object is positively shifted in the z axis by exactly a half of the height of the bounding box (figure 50).

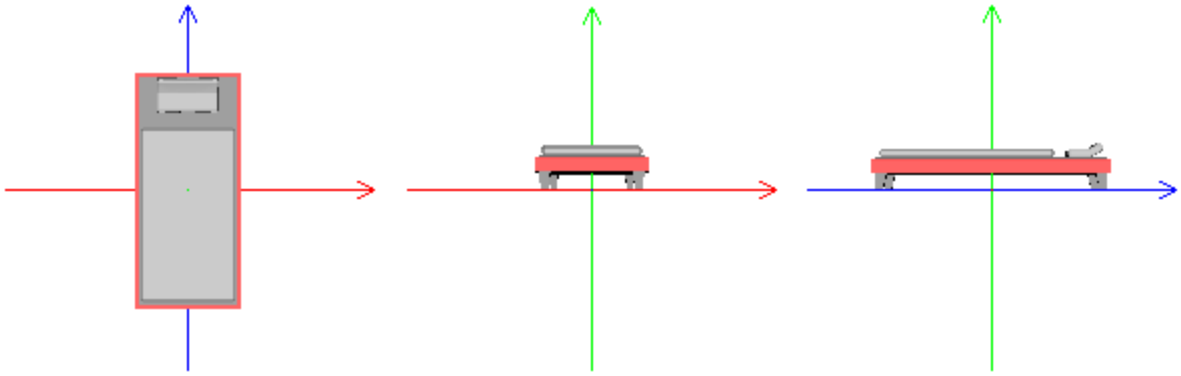

**Figure 50 Position of a "Normal" type object in 3D**

The local coordinate system origin for "wall-hung" coincides with the center of the "bounding box" in x axis and the object is shifted negatively in the y axis by exactly a half of the length of the box and positively shifted in the z axis by exactly a half of the height of the bounding box (figure 51).

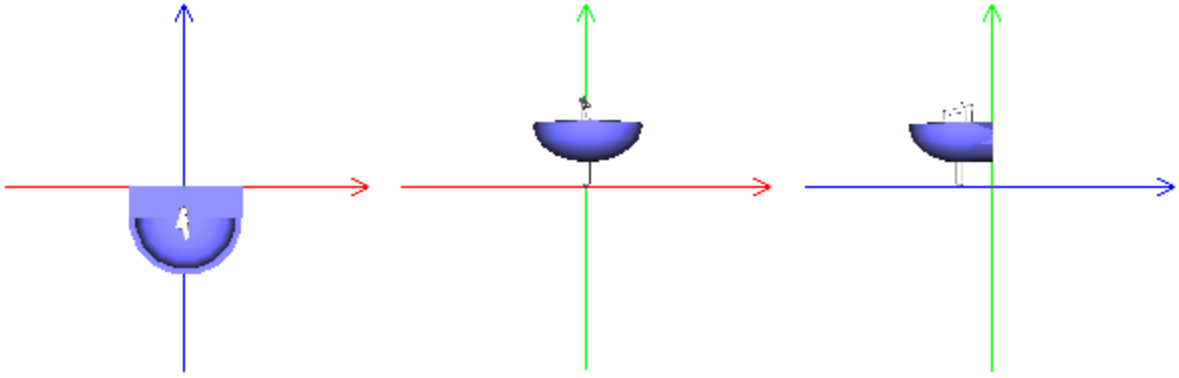

**Figure 51 Position of a wall-hung object in 3D**

The elevation of the model equals the height of its lowermost point from the ground. Note that in figure 51, the lowermost point is the bottom of the siphon, not the basin.
The 3D model can be enriched by any type of features that are allowed in X3D format. The doors can be smoothly opened by dragging them with mouse and the toilet can be flushed by clicking on the water tank or the TV plays a video.

## Furniture extensibility

FloorPAD provides the possibility to add new pieces of furniture. The framework allows the addition of new furniture only by editing/adding information to configuration files and adding new images. The main `FloorPAD.swf` file does not have to be recompiled.

Described in short, 2D parts (pictures) have to be correctly mapped to external 3D parts defined in an X3D prototype. Configuration files have to be updated in order to successfully add new pieces of furniture. Only wall-hung or normal types of furniture can be easily added to the furniture library. Adding new types of doors or furniture makes source code adjustments in many places necessary. The furniture extensibility will be explained on one example – a toilet.

### Adding a toilet

We consider adding a toilet to the designer's furniture collection. It should have the possibility to change the lavatory as well as board colour.

1. The toilet object will be named "ProtoWcSimple" – it will be also the name of prototype in designer. Make sure that there is no other object of the same name.

2. The X3D prototype "ProtoWcSimple" is created and saved to `ProtoWcSimple.x3d` file. The prototype definition has to at least contain parameters *Part1*, *Part2* of type SFColor. The prototype has an additional third parameter *SoundPath* – the path to sound file. This file should be placed in the same directory where the `ProtoWcSimple.x3d` file is located. It may have additional features. The toilet can be flushed by clicking on the flushing tank and the board can change position by dragging it with mouse. The node tree in ProtoWcSimple.x3d should look similar to figure 52.

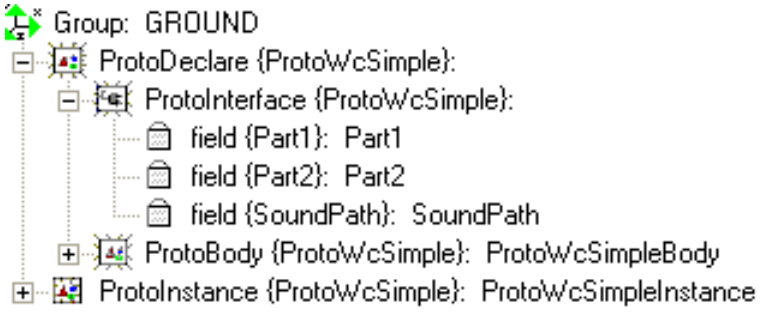


**Figure 52 Tree structure in ProtoWcSimple.x3d file**

The right positioning of model is shown in figure 53.

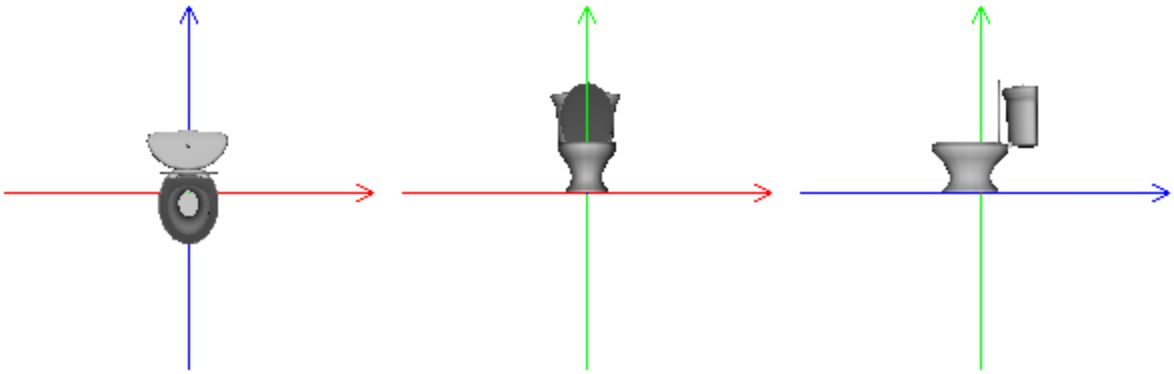

**Figure 53 Model positioning**

The `ProtoWcSimple.x3d` file must be zipped.

3. When the prototype definition contains more than two default parameters (*Part1* and *Part2*), the XSL transformation file has to be adjusted. Two parts of code have to be changed in the "Objects" section – the declaration of a new parameter and the definition of value for the new value of this parameter. The red area in figure 54 shows the declaration of a new field (SoundPath) in extern prototype declaration:

```
<ExternProtoDeclare>
  <xsl:attribute name="DEF">...
  <xsl:attribute name="name">...
  <xsl:attribute name="url">...
  <field name="Part1" accessType="inputOutput" type="SFColor" />
  <field name="Part2" accessType="inputOutput" type="SFColor" />

<xsl:if test="proto = 'ProtoWcSimple'">
  <field name="SoundPath" accessType="inputOutput" type="MFString" />
</xsl:if>
```

**Figure 54 Definition of a new field in ExternProtoDeclare tag**

Figure 55 shows the setting of a value to this field in the prototype instance:

```
<ProtoInstance >
  <xsl:attribute name="DEF">...
  <xsl:attribute name="name">...
  <xsl:for-each select="parts/part">...

<xsl:if test="proto = 'ProtoWcSimple'">
  <fieldValue name="SoundPath">
    <xsl:attribute name="value">
      <xsl:value-of select="$pathProto3D"/>
      <xsl:text>WcFlush.mp3</xsl:text>
    </xsl:attribute>
  </fieldValue>
</xsl:if>
```

**Figure 55 Definition of new fieldValue tag for ProtoInstance**

4. Add new xml fragment into `Prototypes.xml` configuration file describing the new furniture attributes, figure 56.

```
<proto>
  <type>ProtoWcSimple</type>
  <base>Normal</base>
  <sizing measure="cm">
    <w>68</w>
    <h>100</h>
    <l>100</l>
    <e>0</e>
  </sizing>
  <scaling measure="%">
    <minw>0.8</minw>
    <maxw>1.2</maxw>
    <minh>0.8</minh>
    <maxh>1.2</maxh>
    <minl>1</minl>
    <maxl>1</maxl>
  </scaling>
  <depth>3</depth>
  <tooltip>101</tooltip>
  <parts>
    <part pos="1" col="0xCCCCCC" shadow="1" glow="1" />
    <part pos="2" col="0x996600" glow="1" shadow="1" />
    <part pos="3" fill="0" />
  </parts>
</proto>
```

**Figure 56 Prototype definition**

*Note that the ProtoWcSimple class extends Normal class, thus inheriting all of the properties of the Normal class.*

5. Design new pictures (in PNG format) representing different parts of the toilet in 2D. They have to be named and designed with respect to X3D parts definition. Note that the toilette is created by 2 colourable parts (Part1 and Part2) and 1 part like additional no colourable graphic (Part3).

**Figure 57 ProtoWcSimple parts**

Figure 57 shows the parts of toilet. From left to right it is `Part1.png`, `Part2.png` and `Part3.png`. This is also its z-order (the first is Part1). These pictures have to be stored in directory: `media/Pics/Prototypes/ProtoWcSimple`.
*Note that Part3 is not colourable (attribute "fill" with value "0") according to xml fragment above in figure 56.*

6. Create a suitable picture representing the toilet in the menu (40 x 40 pixels) and save it (default) to `media/Pics/MenuItems` directory and update the configuration file `MenuItems.xml` with the following fragment (figure 58).

```
<i>
  <cg>BathRoom</cg>
  <pt>ProtoWcSimple</pt>
  <k>79</k>
  <im>../media/Pics/MenuItems/ProtoWcSimple.png</im>
  <cd>106</cd>
</i>
```

**Figure 58 Menu item description for the toilet symbol**

This XML fragment defines the category in which the menu item will be placed (Bathroom), the prototype name (ProtoWcSimple), keyboard shortcut ASCII code (79), the path menu picture and a description code.

7. Update all language configuration files (`xx_descs.xml`) with new code definitions. In this case the new codes are 101 and 106.

## Troubleshooting

Adding new pieces of furniture to the designer is fairly straightforward. Care must be taken to assure that every part is named and mapped properly. In case an error occurs during the 2D furniture part design, the designer prints out an error message on the console. If the 3D model prototype is improperly defined, it may lead to incorrect display of the model or even to issues in the whole scene.

# Appendix B – Code snippets

This appendix provides some examples of source codes dealing with designer's depth management as well as transformation into 3D scene from 2D plan.

## Depth management core functions

```
/**
 * Adds new child to the display list (with respect to the hierarchy)
 * @param   d - new child
 */
public function addHChild(d:DisplayObject):void {
    // test some exceptions
    if (d is CanvasNet){
        this.addChildAt(d, 0);
    } else if ( (d is Cursor.Cursor) ||
                (d is Proto.Face.EditWall) ||
                (d is PathHelp) ||
                (d is Ccarrier) ||
                (d is ProtoJoint) )
    {
        this.addChildAt(d, numChildren);
    } else { // other objects - pieces of furniture
        this.addChildAt(d, getLayerDepth((Csprite)(d)));
    }
}

/**
 * Gets Depth index for partincular depth of Csprite objects give by parameter
 * @param   d - Csprite instance
 * @return depth index
 */
public function getLayerDepth(d:Csprite):int {
    // satifactory depth index
    var dOK:int = 1;

    // walk through all children
    for(var i:int = 0; i < this.numChildren; i++){

        var child:DisplayObject = this.getChildAt(i);
        if (child is Csprite){
            if ((Csprite)(child).depthLevel <= d.depthLevel)
                dOK++;
            else
                break; // found sufficient depth index
        }
    }

    return dOK;
}
```

**Code snippet 1 Depth management functions**

## Scene transformation

The code snippet 2 demonstrates the content of `PlanTransform.xsl` file.

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Author: Milan Slančík -->
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
  <xsl:output method="xml" version="1.0" encoding="utf-8" doctype-public="ISO//Web3D/
    doctype-system="http://www.web3d.org/specifications/x3d-3.0.dtd"/>
  <!-- declaration of global variables brought from .NET environ.-->
  <!-- Variable sent from .NET: pathProto3D ("../../../Proto3D/BecauseOf/Octaga/")-->
  <xsl:param name="pathProto3D"/> <!-- path to prototype declaration files-->
  <xsl:param name="ExportTime"/> <!-- Export time -->
  <xsl:param name="ExportDate"/> <!-- Export date -->
  <xsl:param name="ExportId"/> <!-- Export identifier -->
  <!-- root element -->
  <xsl:template match="/">
    <X3D profile='Immersive' >
      <head>
        <meta name='ExportTime'>... <meta name='ExportDate'>... <meta name='ExportId'
      </head>
      <!-- scene transformation -->
      <Scene>
        <Transform rotation='-1 0 0 1.571'>
        <Background DEF='DefaultBackground' containerField='children' skyAngle='' sky
        <NavigationInfo DEF='DefaultNavigationInfo' containerField='children' avatarS
            speed='1' headlight='true' type='"WALK" "ANY" "FLY" "EXAMINE" "LOOKAT"'/>
        <!-- Autogenerated viewpoint "Animation" -->
        <Viewpoint DEF='ViewpointAnimation' containerField='children' description='An
            jump='true' fieldOfView='0.785' position='-2.84453 -9.34061 1.52862' orie
        <WorldInfo title='3D presentation' info='"This Web3D Content was created by F
        <!-- Whole plan transformation -->
        <Transform DEF='WholePlan'>
          <!-- Autogenerated viewpoint "2D Alike" -->
          <Viewpoint DEF="Viewpoint2DAlike" containerField="children" description="2D
          <Transform rotation='1 0 0 3.141592653589793'>...
        </Transform>
        <TimeSensor DEF="TimerAnim" cycleInterval="100.000" loop="true" startTime="-1
        <OrientationInterpolator DEF="InterpolRotation" key="0 .25 .5 .75 1" keyValue
        <TimeSensor DEF="vizx_init" cycleInterval="0.100" loop="true" />
        <ROUTE fromNode="vizx_init" fromField="cycleTime" toNode="TimerAnim" toField=
        <ROUTE fromNode="vizx_init" fromField="cycleTime" toNode="vizx_init" toField=
        <ROUTE fromNode="TimerAnim" fromField="fraction_changed" toNode="InterpolRota
        <ROUTE fromNode="InterpolRotation" fromField="value_changed" toNode="WholePla
        </Transform>
      </Scene>
    </X3D>
  </xsl:template>
</xsl:stylesheet>
```

**Code snippet 2 Scene transformation rules**

*Note that element "Transform" with attribute "DEF" with value "WholePlan" contains other transformation rules. Due to its extense, the content was removed. Code snippet 3 shows the content of this element more detailed.*

```
<!-- Whole plan transformation -->
<Transform DEF='WholePlan'>
  <!-- Autogenerated viewpoint "2D Alike" -->
  <Viewpoint DEF="Viewpoint2DAlike" containerField="children" description="2D Alike"
    jump="true" fieldOfView="0.785" position="0 0 9" orientation="0 0 1 0" />
  <Transform rotation='1 0 0 3.1415926535897793'>
    <Transform>
      <xsl:for-each select='plan/adjust'>...

      <!-- Skewers -->
      <Transform rotation='0 1 0 3.1415926535897793'>...

      <!-- Facing Color & Tile Invisible-->
      <xsl:for-each select="plan/facings/facing">...

      <!-- Doors -->
      <xsl:for-each select="plan/doors/door">...

      <!-- Furniture objects-->
      <xsl:for-each select="plan/objects/object">...

      <!-- Floor extursions -->
      <xsl:for-each select="plan/floors/floor">...

      <!-- Wall blocks -->
      <xsl:for-each select="plan/blocks/block">...

      <!-- Viewpoints -->
      <xsl:for-each select="plan/vps/vp">...

      <!-- Paths -->
      <xsl:for-each select="plan/paths/path">...

    </Transform>
  </Transform>
</Transform>
```

**Code snippet 3 Plan transformation rules**

## Viewpoint transformation example

Figure 59 shows the detailed properties of viewpoint in designer. After pressing the button with “Show in 3D”, the designer sends the raw 3D descriptions in xml format into server.

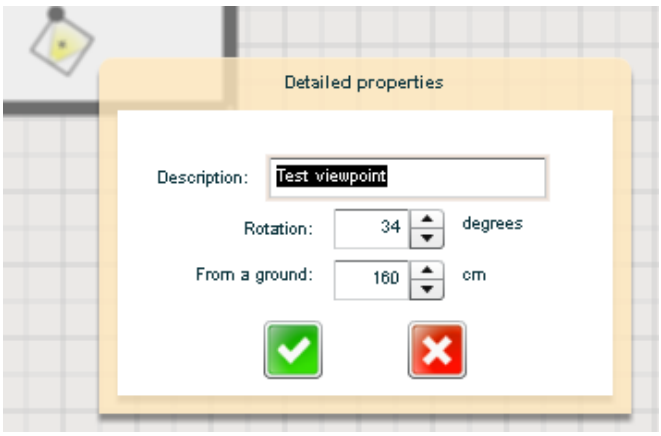


**Figure 59 Viewpoint properties**

```
<plan>
  <adjust>...
  <skewers>...
  <floors>...
  <blocks>...
  <objects>...
  <doors>...
  <facings>...
  <vps>
    <vp>
      <proto>ProtoViewPoint</proto>
      <x>-1</x>
      <y>-2.5</y>
      <z>-1.6</z>
      <o>0.5934119456780721</o>
      <t>Test viewpoint</t>
    </vp>
  </vps>
  <paths>...
</plan>
```

**Code snippet 4 Raw designer's xml output**

Code snippet 4 shows the viewpoint definition (corresponding with attributes in figure 59) before the X3D transformation. Code snippet 5 shows the rules applicable on viewpoints' definitions from the designer's output xml.

```
<!-- Viewpoints -->
<xsl:for-each select="plan/vps/vp">
  <Transform>
    <xsl:attribute name='translation' >
      <xsl:value-of select='x'/>
      <xsl:text> </xsl:text>
      <xsl:value-of select='y'/>
      <xsl:text> </xsl:text>
      <xsl:value-of select='z'/>
    </xsl:attribute>
    <xsl:attribute name='rotation'>
      <xsl:text>0 0 1 </xsl:text>
      <xsl:value-of select='o'/>
    </xsl:attribute>
    <Transform rotation='-1 0 0 1.641'>
      <Viewpoint DEF="Viewpoint" containerField="children" jump="true"
          fieldOfView="0.785" position="0 0 0" orientation="0 0 1 0" >
        <xsl:attribute name='description'>
          <xsl:value-of select='t'/>
        </xsl:attribute>
      </Viewpoint>
    </Transform>
  </Transform>
</xsl:for-each>
```

**Code snippet 5 Viewpoint transformation rules**

Code snippet 6 shows the viewpoint after XSLT transformation. The final result xml is valid against the X3D schema specification.

```
<Transform translation="-1 -2.5 -1.6" rotation="0 0 1 0.5934119456780721">
    <Transform rotation="-1 0 0 1.641">
      <Viewpoint DEF="Viewpoint" containerField="children" jump="true" fieldOfView="0.785"
          position="0 0 0" orientation="0 0 1 0" description="Test viewpoint" />
  </Transform>
</Transform>
```

**Code snippet 6 Viewpoint in X3D**

# Appendix C – User documentation

This appendix deals with the description of FloorPAD's particular functionalities. The detailed parts are completed with pictures.

*FloorPAD* is the online application that allows sketching floor plans and subsequently creates astonishing 3D previews. During the development of this unique designer, the emphasis was taken on the user friendly interface and its intuitive and simple manipulation. FloorPAD is suitable tool for estate professionals, it allows saving plans as pictures.

## Orientation in the application

Sketching in FloorPAD designer works in 2 modes:

1. **Drawing a 2D plan** is the basic application mode which allows delineation of a floor plan, addition and editing of furniture, addition of viewpoints and animation in 3D etc. This mode is showed in figure 60.


**Figure 60 Drawing a 2D plan mode**

2. **Mode editing wall face** is suitable for more detailed customization of wall face. It includes coloring the face wall, addition of tiles, customizing the position of windows, doors and other wall-hung objects (e.g. posters). This mode is showed in figure 61.


**Figure 61 Editing wall face mode**

*Note: It can be seen from a picture, that the menu was essentially changed.*

## Fundamental functionalities

List of functionalities follows:

- **Delineating floor plans –** advanced algorithms of merging walls allows creating a floor plan in fragments of one minute.
- **Adjustments of fill floor –** a default while fill color can be changed onto parquets, carpets or other materials.
- **Wall editing –** a wall width and height can be changed, as well as the color or tiles
- **Inserting various types of furniture –** application has wild range of furniture
- **Advanced objects' editing –** according to the type of furniture, it allows to adjust his sizing, rotation or even color of its parts.
- **Viewpoints and animation in 3D –** designed plan can be simply transformed into 3D and user can walk through his amazing flat virtually.
- **Saving designed plan, saving as picture, inserting indexes of photos and printing**– only for registered users.

## Drawing a plan

The particular ***action*** is chosen by pressing a button in menu application. The action may represent an editing of an object, drawing a wall, inserting new furniture etc.

The individual actions are organized into ***categories***, e.g.: category *Building* contains actions like *drawing room, drawing wall, adding door, window etc.* Other categories represent different types of rooms.

In frame of each category is chosen only one action that is represented by icon on the button. Only one action at time is ***active action*** in designer. For example let us consider the situation on the figure 62, in category *Bathroom* is chosen action *adding Toilette*. This action is also *active action* – the icon of toilette is on the menu button. The active action can be picked up also by ***keyboard shortcut***.


**Figure 62 The menu with chosen category Bathroom with chosen action "Toilette"**

## Drawing rooms

The icon  represents the *action* of *drawing whole room*. It can be simple delineated a rectangle that represents a room. The rectangles (rooms), which are close enough (in time of drawing) with their lines (walls, can be partly merged into conjunctive shape. The basic floor plan can be sketched this way very effectively. **Due to possible errors in 3D generations, it is not recommended drawing rectangle one into the other.**

Figure 63 and 64 demonstrate merging the lines of the rectangles (walls of the rooms).

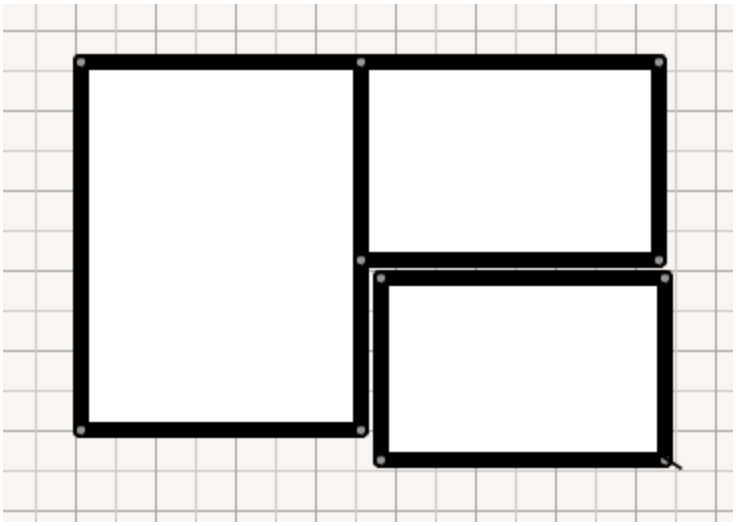

**Figure 63 The sketching the third room**

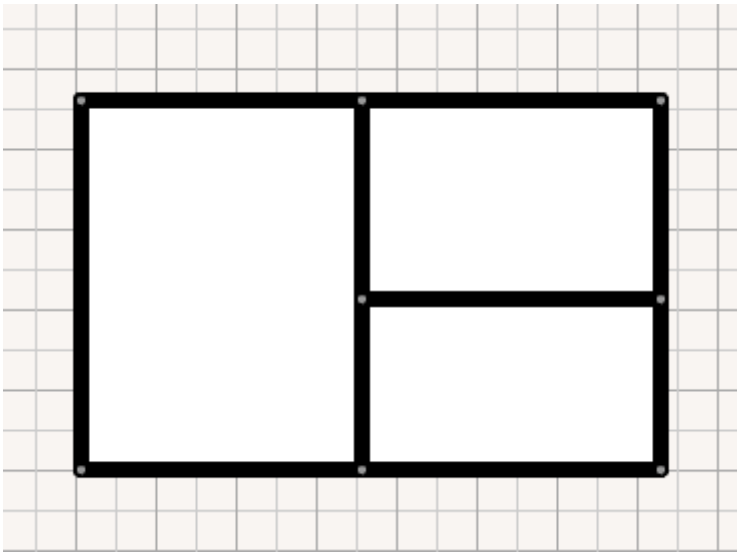

**Figure 64 Third rectangle was merged with the others**

After clicking on action *Editing* , delineated walls can be easily adjusted (only in **horizontal** or **vertical** position).

Figure 65 shows dragging wall in vertical position.

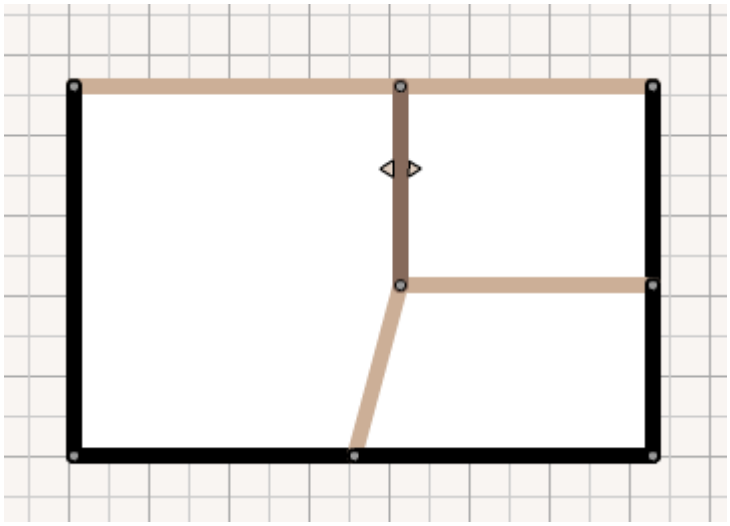

**Figure 65 Shifting an internal wall**

Figure 66 shows inability to move with wall that is not in horizontal or vertical position.

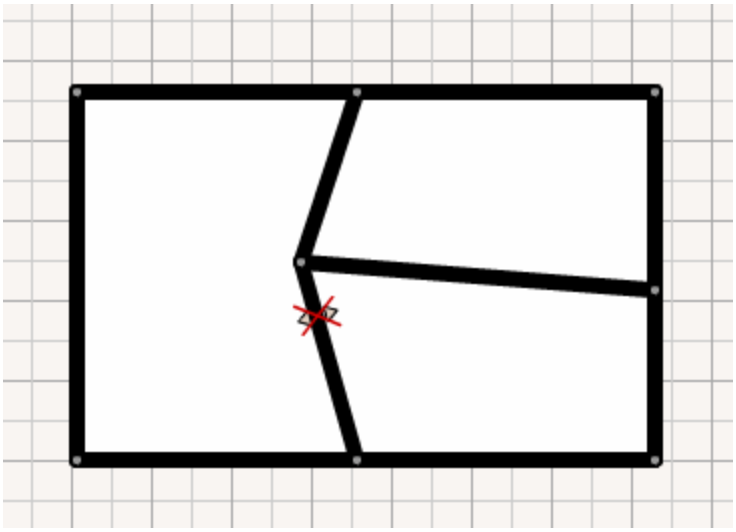

**Figure 66 Inability to move particular wall**

By holding the SHIFT key during shifting the wall it leads to shifting of all the walls lying on the same line as the dragged one, figure 67.

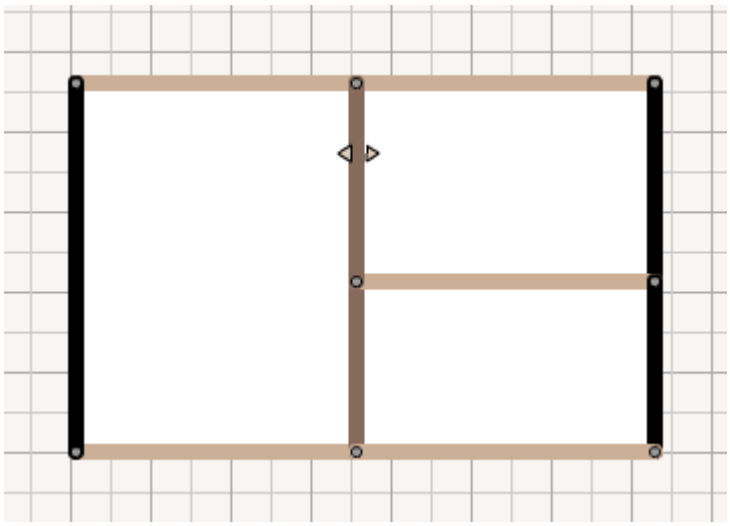

**Figure 67 Shifting more walls at once**

The room can have an irregular shape. It can be created by using action *Joint* (category *Building),*  . The joint „breaks apart“ picked up wall into two walls and that allows to add a curve. The example is on the figure 68.

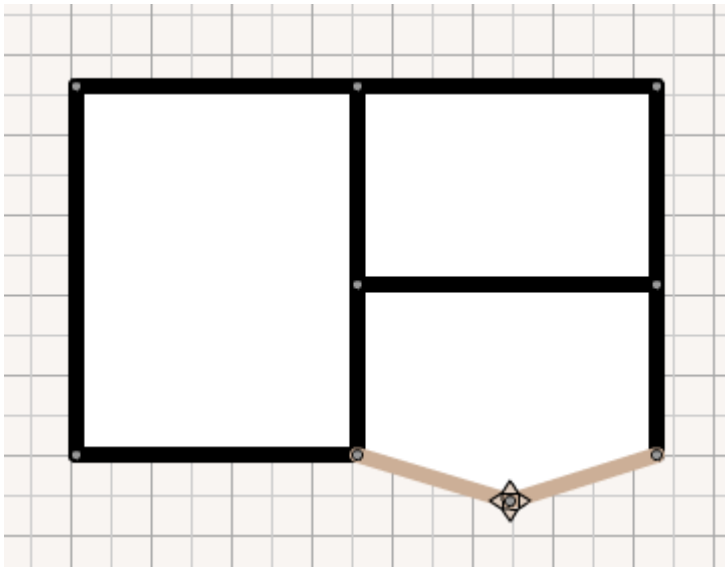

**Figure 68 New wall joint**

*Note.: The joint can be removed by merging it with his neighbouring joint.*

### Adding doors and windows

The actions adding doors and windows are also in category *Building*. They are represented by this icons:  and . These objects are always moving according to closest wall ("jumping" to the closest wall). When the wall is shifted, its' attached objects are shifted too, see figure 69.

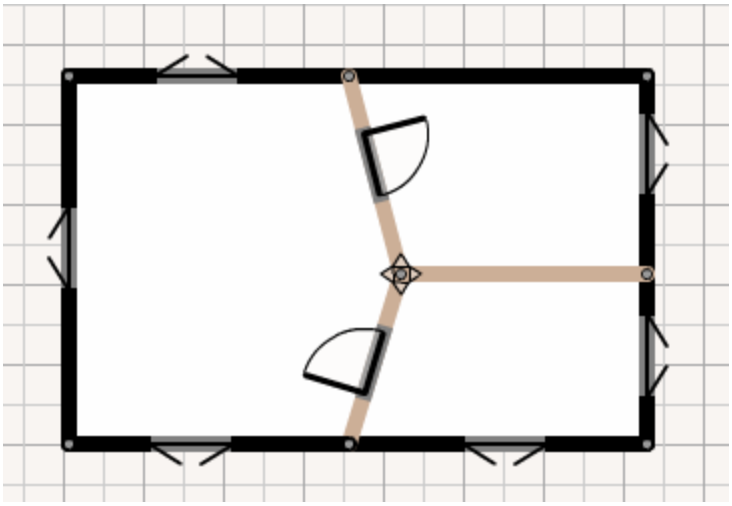

**Figure 69 Objects' position changing during dragging wall joint**

### Adding furniture

Inserting pieces of furniture is simple action. Just pick the furniture icon in the menu and click onto desired position in the plan. Some pieces of furniture are attached to the wall (e.g. a wash basin, a plasma TV etc). Their location and the rotation depend on the wall to which they are attached (figure 70).

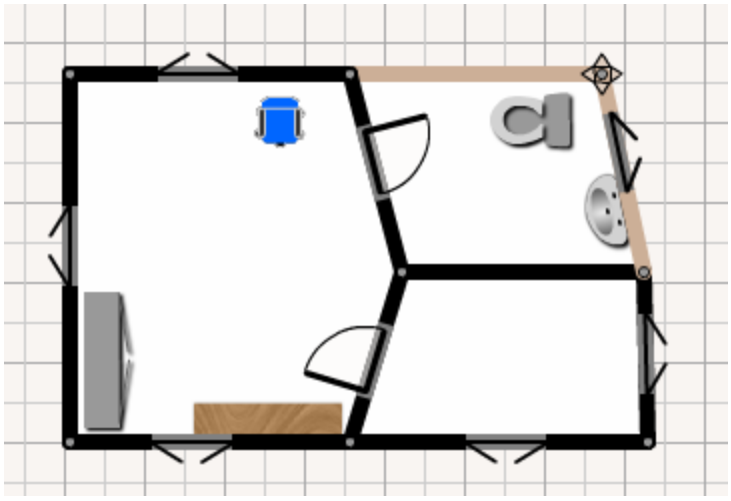

**Figure 70 Wash basin is changing the position together with dragged wall.**

### Filling the floor with the pattern

The advanced properties appears by choosing the action *Editing* and ***double-clicking*** on that floor. The pattern can be rotated like it is shown in figure 71.


**Figure 71 Floor pattern choice**

### Coloring inner walls

The inner walls can by colored by clicking on the tab "Coloring inner walls", see figure 72.

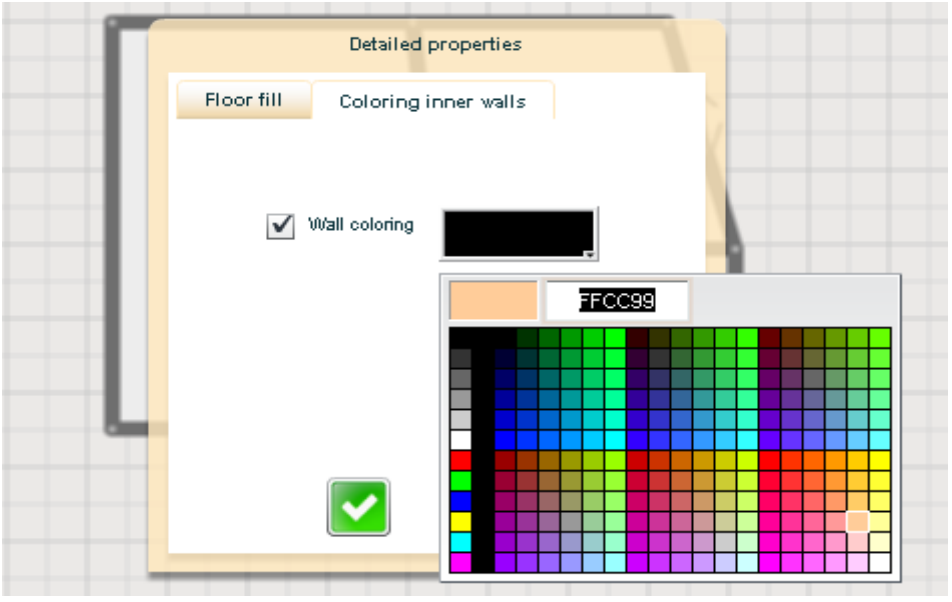


**Figure 72 Color picker**

Edit actions have to be confirmed by clicking on the button .

## Objects editing

***A selection tool*** appears when the particular object is clicked (during editing action ). The selection tool has shape of a rectangle described to selected object. In the left upper corner is black circlet (for rotating) and in the right lower corner is black filled square (for scaling). The selection tool is showed in figure 73.

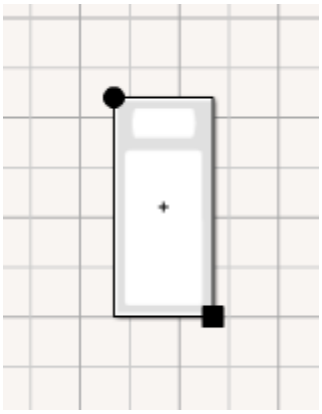

**Figure 73 Selected object – single bed**

### Size adjusting

Each object has ***minimal*** and ***maximal*** sizing boundaries. To change the size of object, press down mouse button on black square and change its position by dragging. The selection tool redraws itself according to new changed position of black square (figure 74). By releasing the black square (mouse button up) it applies to edited object according to new selection shape. When dragging the black square with pressed SHIFT key, size of the edited object changes proportionally.

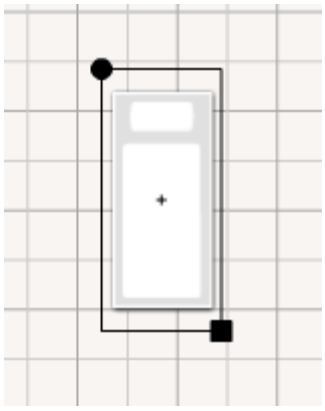

**Figure 74 Size adjusting**

### Rotation adjusting

The rotation may be changed analogously to the action mentioned above. Just press mouse button down with the cursor over the black circlet and drag. The selection is changing his rotation according to dragged position of black circlet, see figure 75. After releasing mouse button the edited object changes rotation according to selection's rotation.

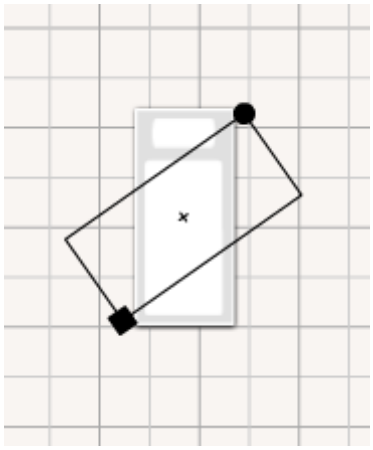

**Figure 75 Object's angular rotation**

*Note: Rotation can be adjusted only for objects that are not attached (dependent) to any wall.*

### Coloring objects

The majority of furniture pieces can be colored. Again, just double-click on the selected object to display advanced properties and select tab "Coloring" (figure 76).

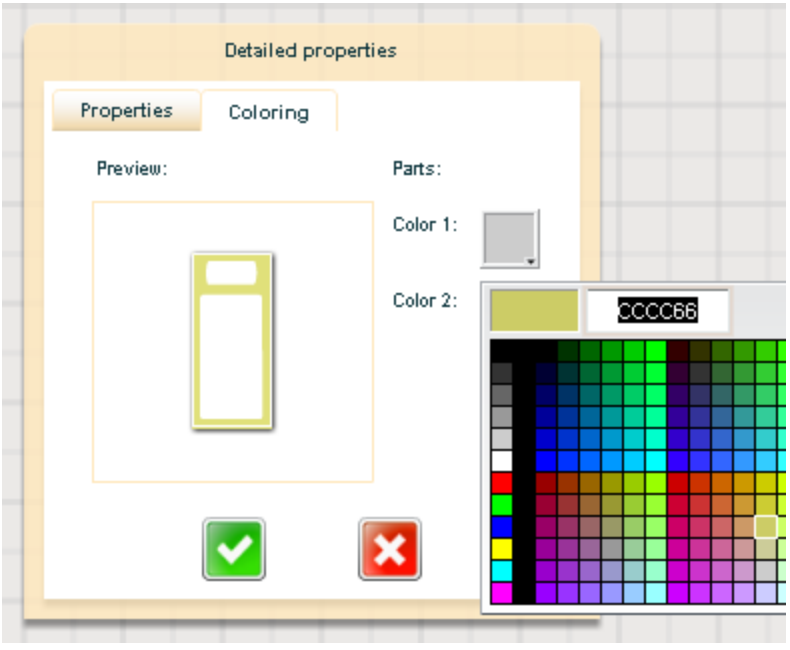


**Figure 76 Object coloring**

### Detailed properties

In order to show ***advanced properties*** the selected object has to be ***double-clicked*** (the actual action has to be *editing* ), figure 77.

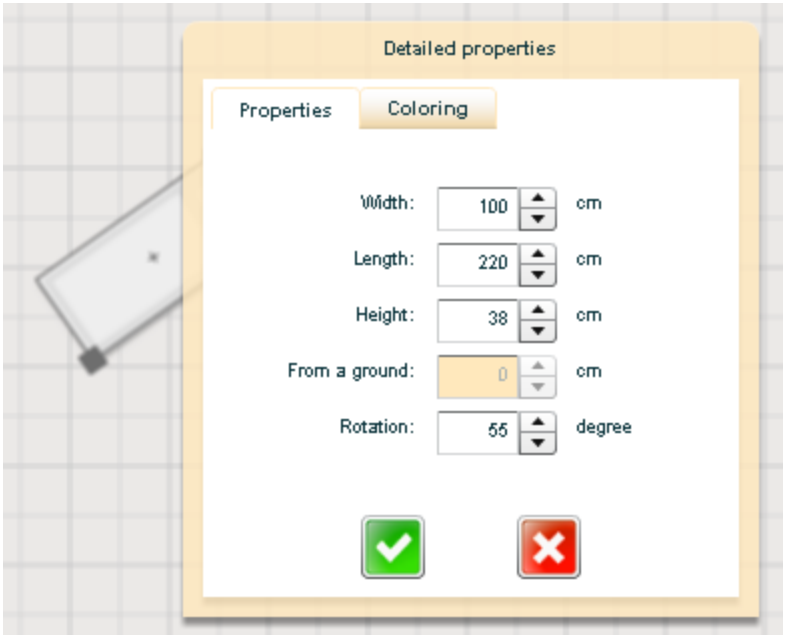


**Figure 77 Detailed object properties**

## Wall editing

### Setting the height of walls

By double clicking on the wall the detailed properties window appears, see figure 78.

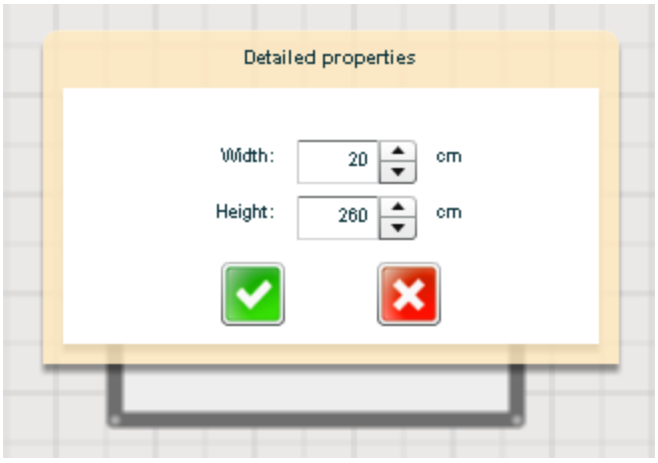


**Figure 78 Wall properties**

*Note: Wall height is same for every other wall height.*

### Face wall editing

Wall editing can be turn on by clicking on the action *Editing wall*  in the category *3D*, or by keyboard shortcut *E*. A grey circlet appears at each side of the wall, figure 79. By clicking on this grey circle, the **designer moves to 2. mode**.

**Figure 79 Selection of face wall**

### Face wall coloring

Just double-click on selected face wall and its properties show up (figure 80).

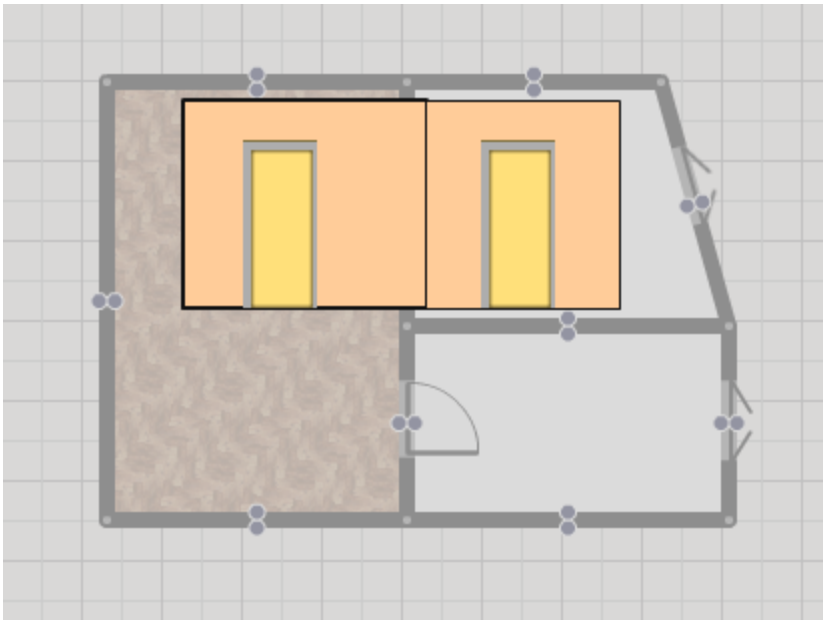

**Figure 80 Colored walls**

### Delineating of ranges and adding tiles

In order to add tiles on the face wall it has to be delineated the rage from tiles. (pick up action  in category *Ranges*), figure 81.

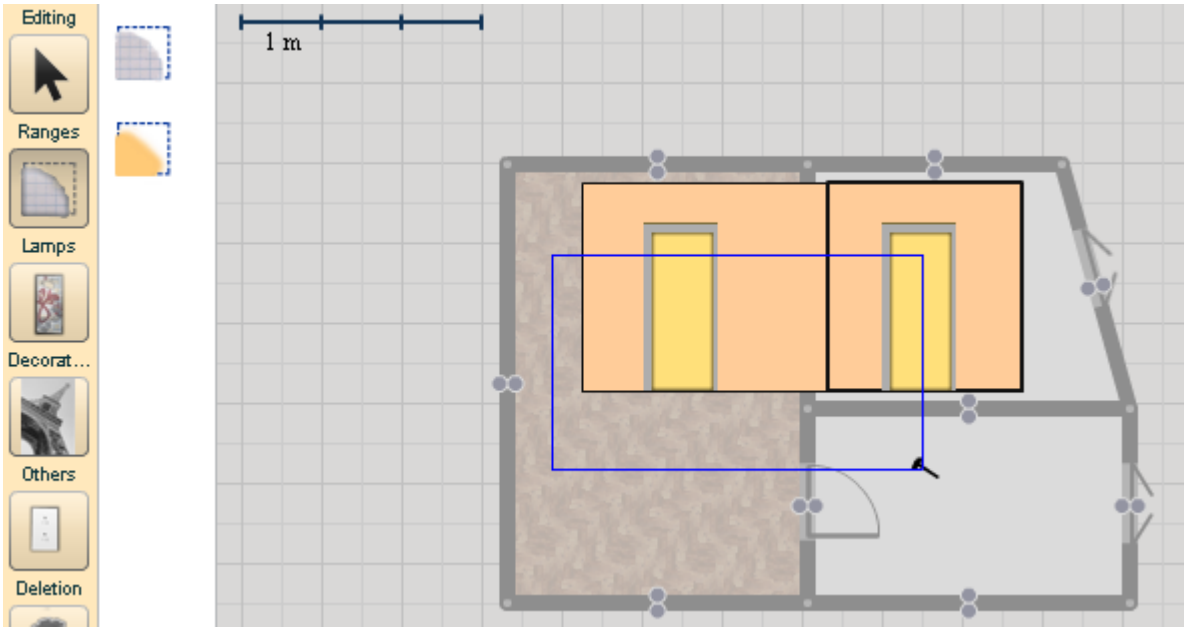


**Figure 81 Delineating of tiles range**

After this action, just double-click on the newly delineated range (figure 82).

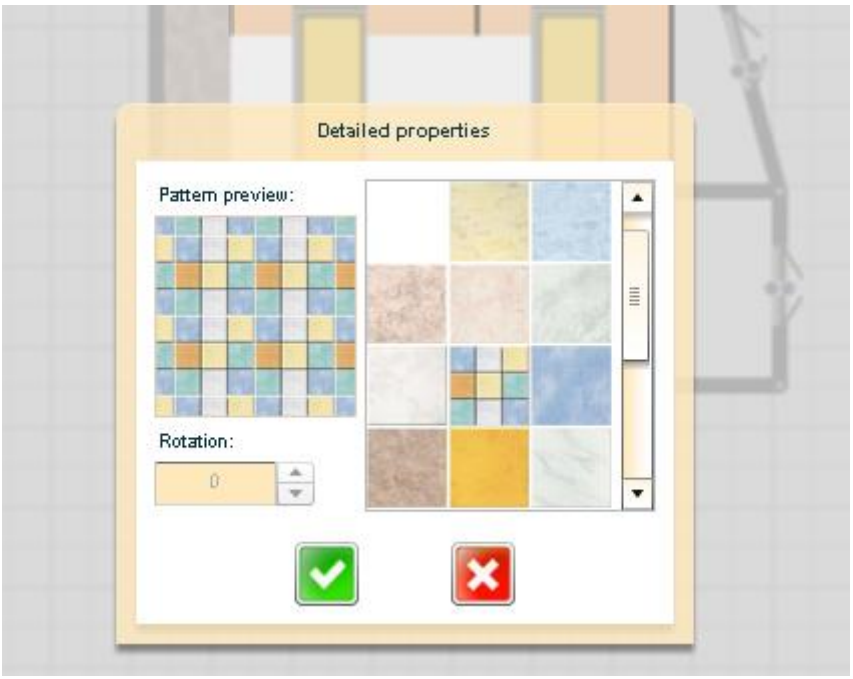


**Figure 82 Tiles selection**

The ***color range*** can be delineated and filled similarly.

### Adding small wall-hung objects

Small wall-hung objects (e.g. paintings, posters, switch, small wall lamps etc.) can be placed on the face wall in similar way like normal objects into plan – just click to menu item and click to the desired place in face wall, see figure 83.

**Figure 83 Face wall with hung objects**

*Note: These objects are **not visible** in **edit 2D plan mode**, only in 3D model.*

### Returning into "Draw 2D plan" mode

In order to get back into the *"normal" mode*, you have to *confirm* or *cancel* all actions made during the working in this "face wall editing" mode.

## Object deletion

To delete object, just select action with the icon in the category *Deletion* (or keyboard shortcut *Delete*). The object that is under the mouse cursor turn into red, see figure 84, and will be removed just by simply clicking on it.

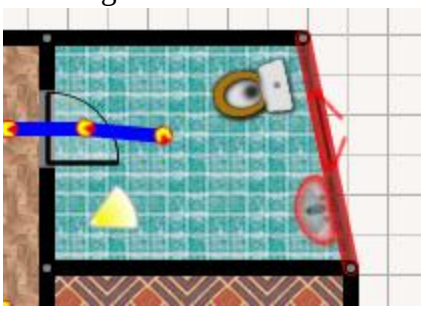

**Figure 84 Turning the wall and all attached objects into red color**

*Note: If the wall will be deleted, it also affects all objects attached to this wall.*

## Viewpoints and animations in 3D

### Viewpoints

*The viewpoint* is suitable to define new place in 3D scene where we would like to observe all objects. The viewpoint action is represented by icon 

in category *3D* (or keyboard shortcut *V*). Viewpoint in 2D is similar to light yellow flapper (figure 85).

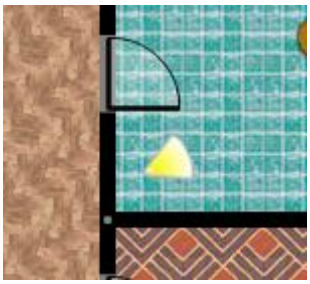

**Figure 85 Viewpoint for 3D scene**

It is important to name it properly and suitable rotate. The figure 86 shows bathroom view in 3D environment.

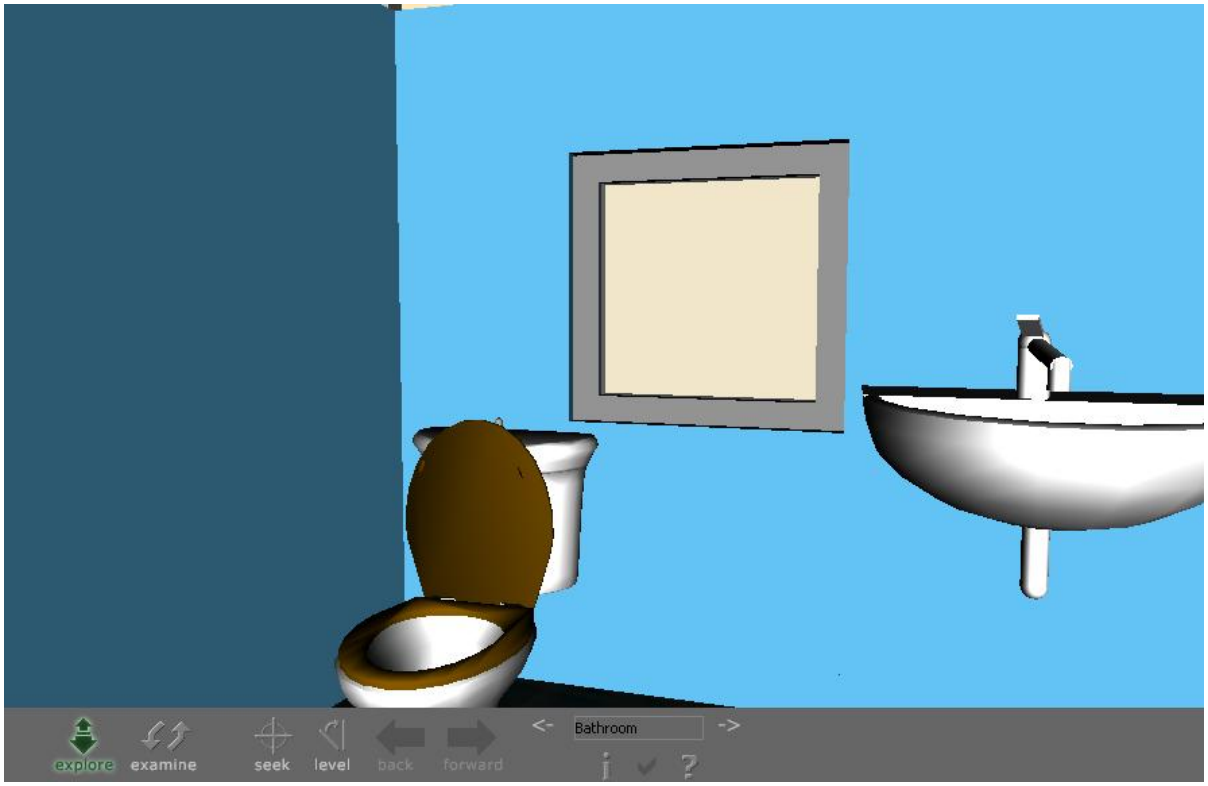


**Figure 86 Viewpoint in bathroom**

In order to set up viewpoint's name and other properties, just double-click on it, (figure 87).


**Figure 87 Viewpoint properties**

*Note: Its initial height from a ground is 150 cm.*

## Animation

Drawing the animation path represents the *action* with  in category *3D* (or key *P*). After choosing this action a double pink dots appears everywhere where the doors are placed. They are ***assisting dots*** to facilitate the drawing process thought doors, see figure 88. By drawing the path close to one of these dots, it will automatically adjust the values of last path joint of animation and create the crucial key moving through doors by itself to the other room.

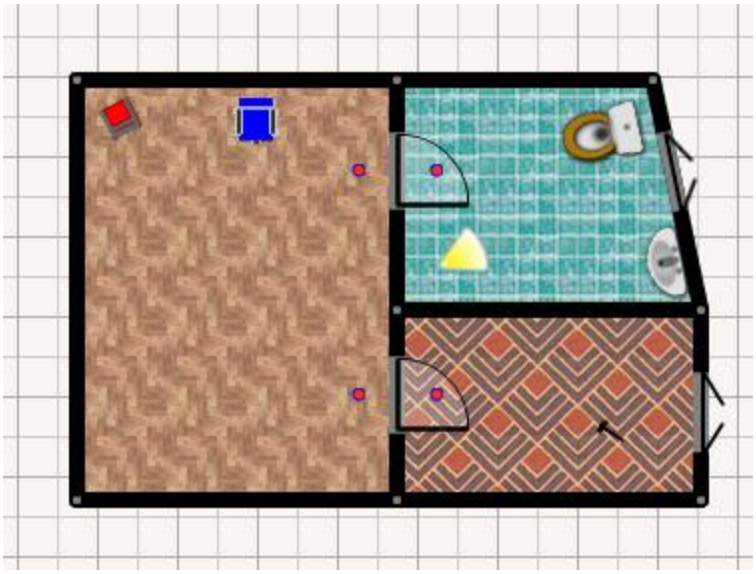

**Figure 88 Assisting door dots**

Figure 89 shows drawing the animation path in progress. The green color indicates that the drawing process is not over.

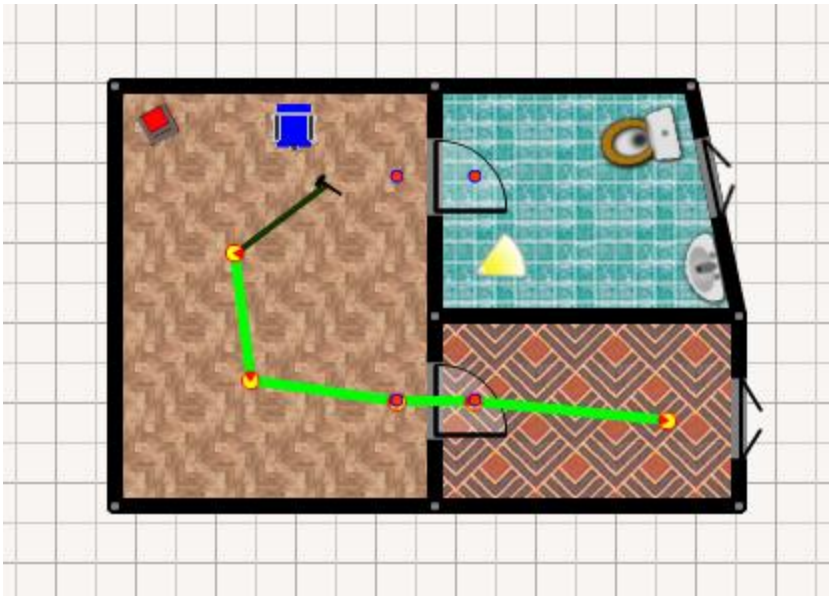

**Figure 89 Drawing path animation**

There are two possible ways how to **finish** drawing animation path:
1. by double clicking on the last time created path joint, or
2. by connecting with the beginning of the path (and creates closed **circle**).

Figure 90 shows closed path leading from room with carpet to the bathroom.

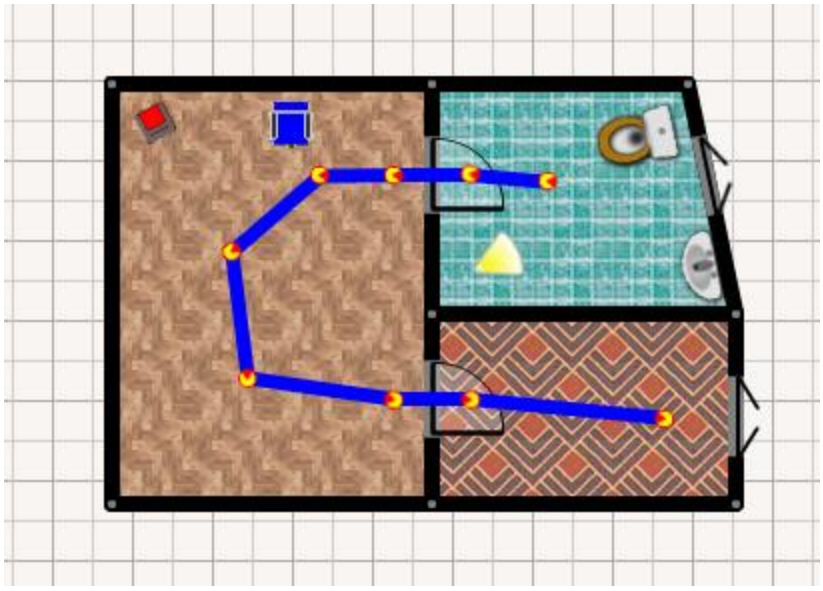

**Figure 90 Finished animation path**

Finished path animation is created by blue **abscissas** connected with **yellow-red joints**. The joints can be further adjusted – changed their position, rotation and other properties.

The red wedge of the path joint designates angular rotation of the viewpoint of the animation. Advanced properties of path joint (figure 91), except rotation, also allows to stop animation in that point for x seconds, or finish the rotation from the last joint for x seconds (it brings interesting effect). *Zero value for "Turn for" part means that rotation from last joint will be interpolated with current joint's rotation.*


**Figure 91 Path joint properties**

Figure 92 shows the advanced property for whole animation. It allows:

- to give a suitable name (if will appear in X3D player)
- to adjust the animation speed (value 1 is the fastest, the 10 is the slowest)
- to adjust height of animated viewpoint from a ground.

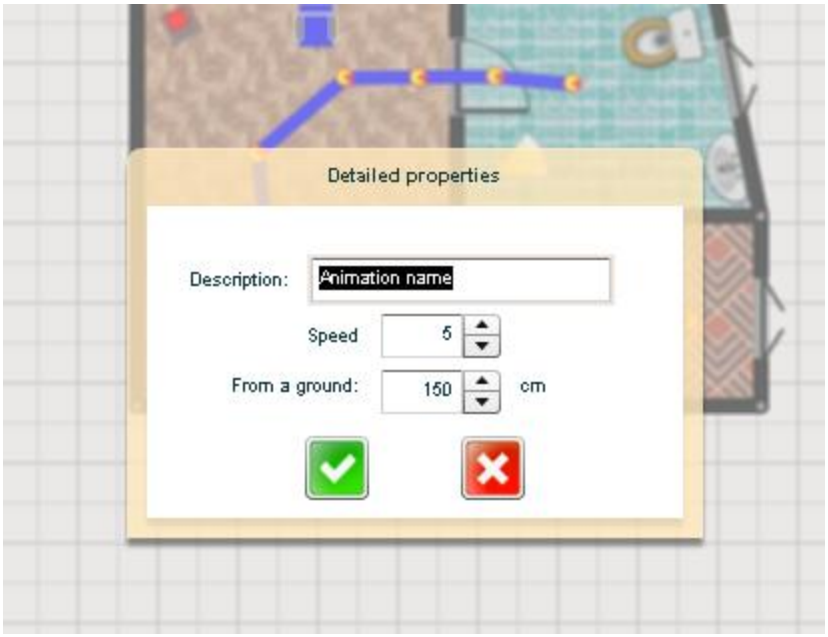


**Figure 92 Animation properties**

*Note: By default the animation sets 2 seconds to property "Stop for" for starting and finishing path joints. Playing animation is repeating endlessly.*

## Settings and other actions

Here is a list of other important menu parts and their brief description:

 - *Settings* – user may adjust some attributes concerning canvas, completely erase sketched objects etc.

 - *Undo and redo* – allows user to go forward / backward with the editing steps during floor plan delineating.

- *3D inspection* – by clicking on this button a 3D examination is launched in another tab.

 - *Print* – delineated plan is printed out, only for registered users.

- *Save as picture* – the content is saved as picture in (PNG format), only for reg. users.

 - *Save and exit* – saves and returns into management plans web site.

 - *Help*.

## Keyboard shortcuts

To speed up sketching process the most common actions have redefined keyboard shortcuts. The list follows:

**Building, editing and 3D presentation:**

- Space – none, editing 
- Delete – deleting 
- R (**R**oom) – to draw a room 
- J (**J**oint) – wall joint 
- D (**D**oor) – simple door 
- W (**W**indow) – simple window 
- T (**T**ext) – text field 
- Q (**Q**uota) – to draw quota 

- E (**E**dit wall) – editing face wall 
- V (**V**iewpoint) – view in 3D 

- P (**P**ath) – to delineate the animation path 

- Scrolling with mouse wheel – zooming canvas in/out
- Alt + D – duplication of selected object

**Furniture:**

- A (w**A**sh-basin) – 
- B (**B**ed) – 
- C (small **C**abinet) – 
- F (**F**ridge) – 
- I (cha**I**r) – 
- M (s**M**all table) – 
- N (si**N**gle seat) – 
- O (t**O**ilet) – 
- S (**S**hover) – 

## 3D presentation

In order to see 3D presentation, the proper X3D player has to be installed.

### Generated viewpoints

There are two generated basic viewpoints:

1. ***Animation*** – the view rotates around the Z axis (figure 93)

**Figure 93 Viewpoint „Animation“**

2. **2D alike** – the view from above upright to the floor plan plain, figure 94.

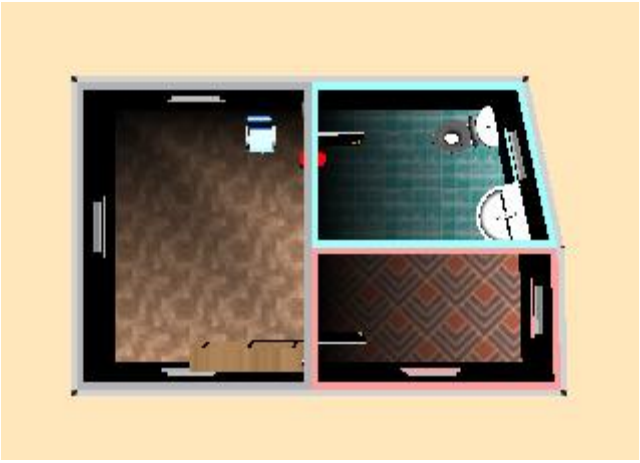

**Figure 94 Viewpoint „2D alike"**

### Navigation in X3D

The navigation in 3D scenes varies according to installed X3D player.
**Flux Player** – change viewpoint, keys: *PgUp, PgDown*
**BS Contact** – change viewpoint, keys: *PgUp*, *PgDown*, or right button of the mouse and select in menu.

## How to sketch the plan effectively

Here is simple step-by-step cookbook how to draw in FloorPAD designer effectively:

1. Find out about designers ratio, allocate the plan into the center of canvas
2. Delineate the basic floor plan with the respect to rooms.
3. Adjust particular rooms to irregular shapes (if needed).
4. Fill the floor with suitable pattern/ color inner walls.
5. Add furniture pieces.
6. Adjust inserted furniture (change color, etc.)
7. Adjust the face of the walls.
8. Save as picture / print out / …
9. Add viewpoints in 3D and name them properly.
10. Create animation path.
11. Show the 3D presentation.

## How to sketch the plan effectively

The installation works in few steps:

1. download the installation program from www.floorpad.eu web site. It is possible to pick up different types of players: *Flux Play*er, *BS Contact* a *Cortona Player*.
2. Install downloaded file on local computer
3. Restart web browser and test the X3D player

# Appendix D - Testimonials

“I saw FloorPAD at end of June 2009 for the first time. On the first sight it attracted me with nice graphics and possibility to generate 3D models just in web browser window. When I find out it is planned to offer FloorPAD to real estate agencies as a part of their current web pages, I wanted to took part in that plan, because I believe FloorPAD is interesting product, which should attract by quality of implementation, speed of model creation thanks to usage of modern technologies and special starting price offer for whole solution.
As the marketing manager of FloorPAD for Czech Republic, I enjoy intuitive user interface, fast flat and house model creation, inspectional result in both 2D and 3D models. These features are appreciated by our customers during everyday work. As for integration into current web pages of our clients, there is great approach of our programmers which gladly assists with occasionally problems.
I think FloorPAD is ideal supplement of current web pages for great variety of reality agencies which want to offer the better depiction of offered realities to their customers.”

Mgr. Radek Lano, marketing manager of FloorPAD, Czech Republic

“Recently I went through a couple of big real estate servers in Czech Republic looking for apartment possibilities in Prague. I was quite shocked. It looks like all servers remained in last century. At a present time where all this new technologies are available none of these servers provides even a simple plan of the flat. The only thing which is usually available are few photos of terrible quality. When I'm choosing an apartment I want to see the floor plan and have some possibility of 3D view of the apartment. At a nowadays with all flash technologies it shouldn't be any problem at all. When I heart about FloorPAD and then I tried it, I was wondering why this kind of thing is not already a part of every real estate server! After the author told me that he is negotiating with few real estate servers, I am really looking forward to see FloorPAD in practice!“

Tomas Marada, PhD student at VU Amsterdam, Netherlands

“I am furnishing my flat at the moment so I am looking for suitable pieces. Since my flat is small I want to buy really useful pieces that will suit my taste and will fit together as a whole. A friend of mine mentioned this application to me as user friendly and with very good outcome. I can see the final version of my flat even before buying anything. I think this is a good idea and I hope that new pieces of furniture will be added soon in menu.”

Lucia Chuda, accounting manager, Slovakia