# PlaneSmith - Quick Guide

# Overview

PlaneSmith is a level editor for 2D games. It does not make any other assumptions about the games you make. One thing it offers you is the ability to specify what output you get. Do you want an XML file? A piece of C++ code? Or maybe a simple list of integers and strings that your program will interpret? PlaneSmith lets you do all that!

How does PlaneSmith work? There are definitions which are basically types that objects can take. Set of all definitions is the dictionary. There are also objects which are instances of those definitions. They make up the level.

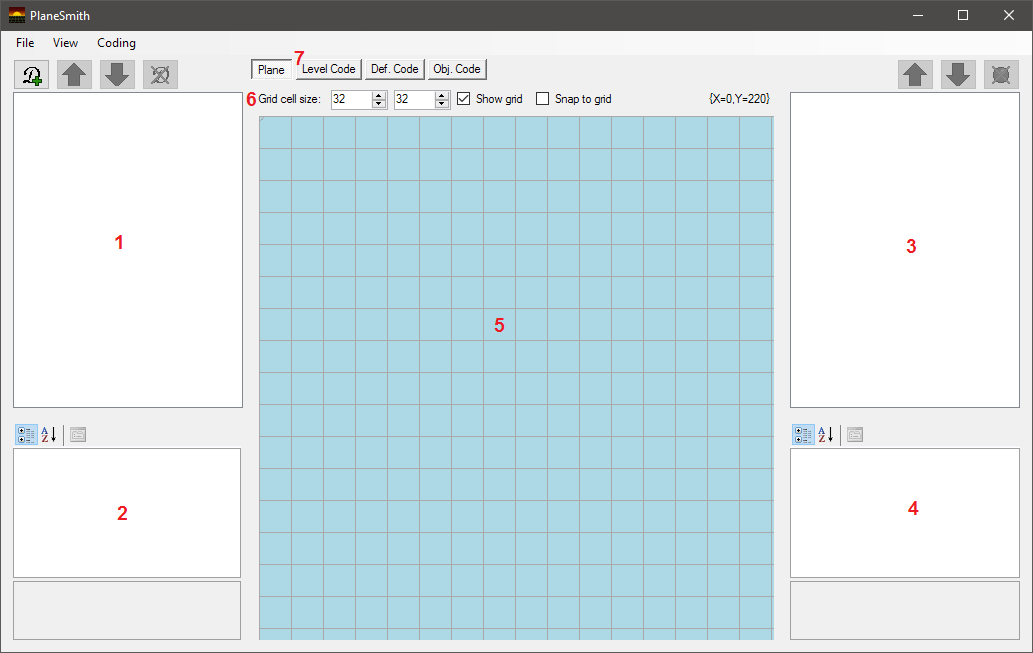
One PlaneSmith project consists of two files: dictionary and level. Multiple levels can use the same dictionary.

# Startup

This document will guide you through a creation of a simple project in PlaneSmith. You can open finished projects from the example folder.

Since PlaneSmith levels remember the path to their dictionary as relative paths, it is good to make one folder for your project and keep all level and dictionary files inside it.

Keeping that in mind, open PlaneSmith. You will probably see a screen like this.



Here's what various things are:

**1** List of all current definitions. Above it are buttons for manipulating this list.

**2** Properties of currently selected definition.

**3** List of all objects in the level. Above it are buttons for manipulating this list.

**4** Properties of currently selected object.

**5** This is the plane. It is a visual representation of the level.

**6** Options regarding plane grid (gray lines). Grid is visual only.

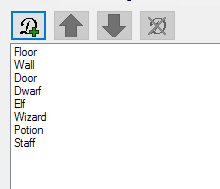
**7** Tabs other than Plane contain textboxes for coding (defining output).

# Example project - Level design

Click on Add Definition button. Input definition name and choose an image file. When you create a definition it is automatically selected. You will see an object following your cursor on the plane. Right-clicking on plane will deselect definitions.



In this example we will have following definitions (png images have corresponding names):



Now let's design our level! We will use a 32x32 grid. Check Snap to grid. This way objects will be aligned to grid lines.

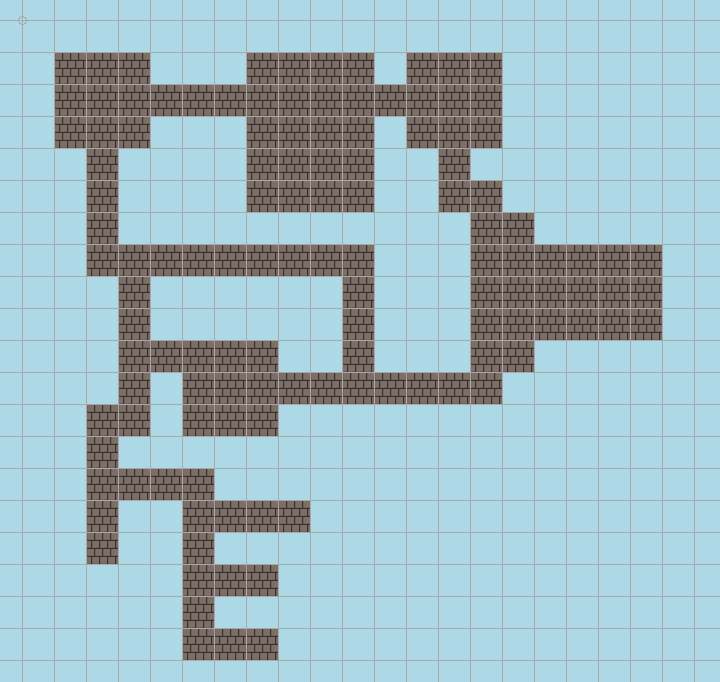
Untitled

Select Floor definition by clicking on it on the list. We place an object by left clicking inside the plane. You can hold shift and left mouse button to drag-place multiple objects, though only when Snap to grid is active.

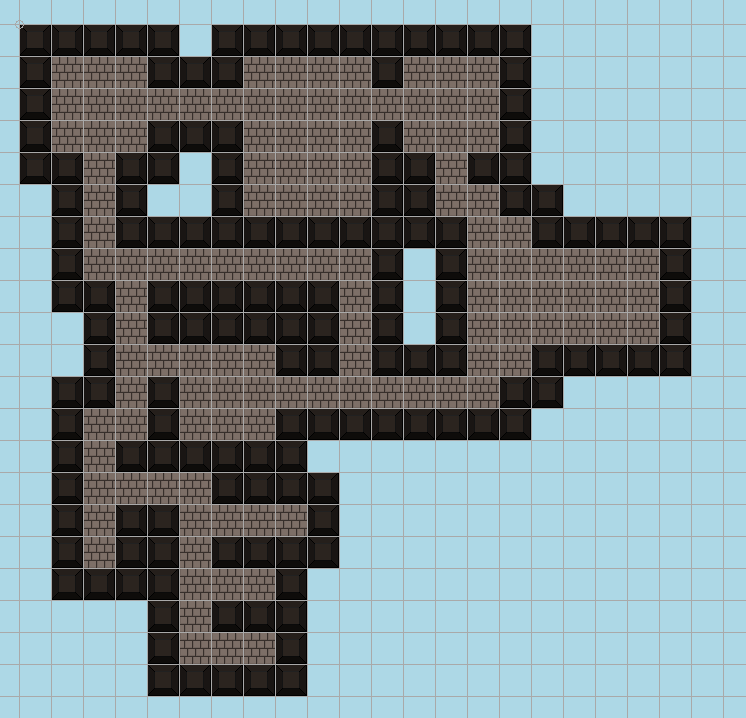
Right-clicking deselects a definition. Right-clicking again deletes an object at cursor (top most object). You can left-click an object to select it, further left-clicks will cycle through objects found at that location (when you have multiple objects on top of each other).

By holding middle mouse button (i.e. wheel) and dragging around the plane, you can move the view around. Use this to move the level slightly closer to center. You don't have to do this, but it might make things easier. Double clicking the plane with middle mouse button will reset any such movement.

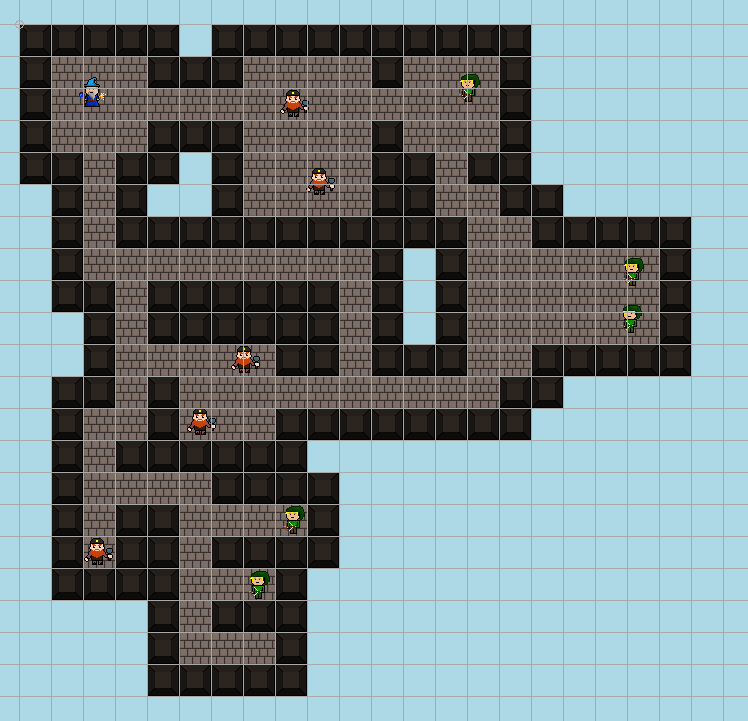
Fill out the level with some Floor objects.



Now, let's surround them with walls. Select the wall definition and place them on the plane.



Now uncheck Snap to grid and add a wizard (the player), some elves and dwarfs.



For fun, let's add some loot. Check Snap to grid again and add potions and a staff. While you're at it, add some doors too. If you change your mind, you can poke a hole in a wall by right-clicking on it to delete.



Ok, this is starting to look like an actual level.

At this point, it would be a good idea to save your project (with Ctrl+S). First you will save the dictionary and then the level. Preferably, group all levels and their dictionary inside the same directory (because of level's relative pathing to its dictionary).

# Example project - Coding

Let's say we are making an RPG in C++ and need to implement the setupLevel method of our class Level1.

Go to Level Code tab. Level Code is the starting point from which output is generated. Here we define the outline of our sorry code. Copy this into the textbox:

#include "Level1.h"

«FECH.DEF.

»#include "«NAME».h"«END»

void Level1::setupLevel() {

«FECH.DEF.

»«CODE»«END»

}

What's going on here? Aside from the obvious C++, PlaneSmith statements are written inside « and ». Click Ctrl+B to quickly add them to your code. Everything outside of these is considered a terminal string and will be used as-is.

The first statement is **FECH**. Name **FECH** is a mix between fetch and for each. Since we chose to fetch through definitions (with **DEF**) it will cycle through all known definitions and output something related to them. In this case it is printing include directives for our C++ code.

Notice the newline after the second **.** in **FECH**. This is a split terminal, which is inserted between every two definitions. In this case we want to separate includes in different lines of code.

A **FECH** statement is closed with an **END** statement. Everything inside is its body. **«NAME»** is an attribute statement. Here, the program will put the name of a definition in place of this statement for every definition it cycles through.

Further down, we have another **FECH**. This one prints the code assigned to our definitions and spreads them with two newlines.

To specify definition codes go to Def. Code tab. You will see two textboxes. The first is for definition codes. All our definitions will have the same code:

«FECH.OBJ.

»«CODE»«END»

Here we cycle through all objects belonging to this definition and printing the code they specify. They are separated by newline.

The textbox below is for codes that automatically get assigned to new objects belonging to selected definition. This way you don't have to copy the same code into every object.

This will be the code for Floor:

makeFloorAtTile(«X» / 32, «Y» / 32, "assets/sprites/«DEF.NAME».png");

It's a made up C++ function for putting a floor on a tile. The first two PlaneSmith statements return coordinates of object on the plane. The third one will return the name of this object's definition. Codes for Wall, Door, Potion and Staff are similar.

Because we've already placed our floor objects, this code isn't added to them. It is only added to new objects that get placed. What you will do now is copy this code into every floor instance in our level.

Nah, just kidding. Click on the *Reinsert obj. codes* button below to let the program do it for you.

Code for dwarves is:

{

Enemy \*dwarf = new Dwarf();

dwarf->setPosition(«X», «Y»);

addEnemy(dwarf);

}

Curly brackets will avoid conflicts with other dwarf variables. Codes for Elf and Wizard are similar. Don't forget to reinsert them into existing objects.

You can see all the codes in the example project.

If you go to *Obj. Code* and select an object you will see its code. For exercise, let's change codes of two specific objects.

Go to *Plane* tab. Select one of the doors (e.g. the one next to the staff). You can select it by left-clicking on it. If you selected a definition, first deselect it by right-clicking. Now go to *Obj. Code* tab.

Add this line:

getTile(«X» / 32, «Y» / 32)->getDoor()->setLocked(true);

Now go back to *Plane* and select one of the elves, and change its code to:

{

Enemy \*elf = new Elf();

elf->setPosition(«X», «Y»);

elf->getLoot()->addKey();

addEnemy(elf);

}

As you see, PlaneSmith allows you to alter the behaviour of individual objects in your level.

Finally, generate your code by with Ctrl+Shift+B. Your project will first be saved. If you made no errors you should get a file with a C++ method implementation.

You can compare it to the output from the example project. There is another example with the same level, but which generates an XML file.

# Conclusion

And there you have it. You can view the documentation on how to code your output in the other file.

There will probably be bugs, the UI could use some tidying up and some new features would be welcome. If you have any suggestions or feedback let me know. This project is open-source on GitHub: <https://github.com/OnionBurger/PlaneSmith>.

Thanks for reading and enjoy!