# PlaneSmith - Quick Guide

# Overview

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

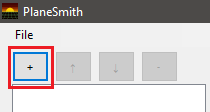
How does PlaneSmith work? There are definitions which are basically types that objects can take. Set of all definitions is the dictionary. There are also objects which are instances of those definitions. They make up the level.

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

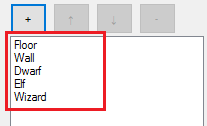
Finally, PlaneSmith's forte is that it allows you to configure the output you get through some clever ~~hacki~~ coding.

# Defining definitions

Open PlaneSmith. Click on Add Definition button. Input the definition name and choose an image file.



In this example we have the following definitions:



Before we dig into designing levels, let's do some coding. Go to Base Code tab. Base Code is the starting point from which output is generated. It is tied to the level, not to the dictionary!



Here we define the outline of our sorry code. Let's say we are making a video-game in C++ and we need to implement the setupLevel method of our class Level1. Here's code for that:

#include "Level1.h"

%<FECH(DEF)>%

void Level1::setupLevel() {

%<FECH(OBJ)>%

}

What's going on here? Aside from the obvious C++, PlaneSmith statements are written inside %< and >%. FECH will assemble all the codes from definitions in the first statement, and objects in the second. In this example we will use definition codes for including their headers.

If you go to Def. Code tab you will see two sections. One is definition codes. Here you will type in the includes for the headers of those definitions. **Make sure to add a newline at the end of this code.** All this is done in the example project.

The other part of definition code is the default code for all objects that get created using that definition. Here is our code for dwarves:

{

Dwarf \*dwarf = new Dwarf();

dwarf->setPosition(%<X>%, %<Y>%);

addEnemy(dwarf);

}

Curly brackets will avoid conflicts with other dwarf variables. %<X>% and %<Y>% are the coordinates of a dwarf object in the level. You can see all the codes in the example project. There is a separate document describing all the PlaneSmith code statements.

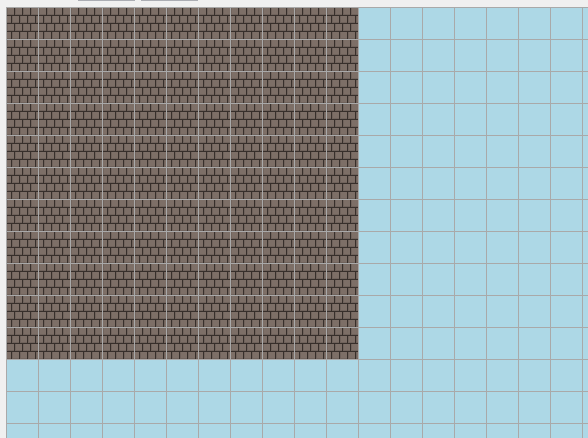
# Objecting objects

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

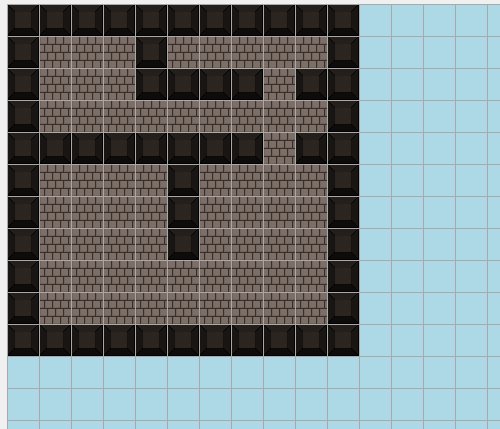


Select Floor definition by clicking on it on the left side list. We place an object by left clicking inside the plane. You can hold shift and left mouse button to drag-place multiple objects, though only when Snap to grid is active. Right clicking deselects a definition. Right clicking again deletes an object at cursor (top most object).

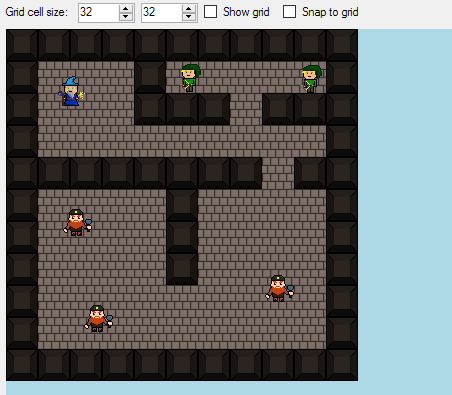
Fill a corner of the level with Floor objects.



Add some walls.



Now let's uncheck Snap to grid and add the wizard, some elves and few dwarfs.



At this point the codes for all objects have been auto-given by their definitions. I want to have that last dwarf drop 10 gold when he gets his due.

Select him from the list on the right and go to Obj. Code. Here you can alter codes of specific objects in your level. We can change it into this:

{

Dwarf \*dwarf = new Dwarf();

dwarf->setPosition(%<X>%, %<Y>%);

dwarf->setLoot(10);

addEnemy(dwarf);

}

Save your work with menu File->Save. You will first save the dictionary and then the level. You can then Load your work later.

Generate code by going to Base code tab, clicking on Generate! button and specifying a file to save to.

# Conclusion

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

There are a lot of features planned (path objects, custom properties, UI elements, etc.). There will probably be bugs and the interface could use some tidying up. Hopefully, I will get to this in the near future.

Thanks for reading and enjoy!