# Level Designers

Dark and Under was released in December with three levels that were designed to provide a couple of hours of game play but not be so large and over-powering that people simply gave up. As part of the development process, we put together a level designer and have released this alongside the game to allow players to develop their own levels.

I have noticed a few other games have released level designers - including @Dreamer2345’s [RogueBoy](https://community.arduboy.com/t/earlyrelease-rogueboy/4349) – and think other developers should embrace the idea as it will extend the life of their own game.

In my opinion, allowing others to design and incorporate new levels should not require them to understand how your game is designed or programmed. It should be a simple process that generates the necessary artefacts and allows the developer to simply compile up the resultant game and play.

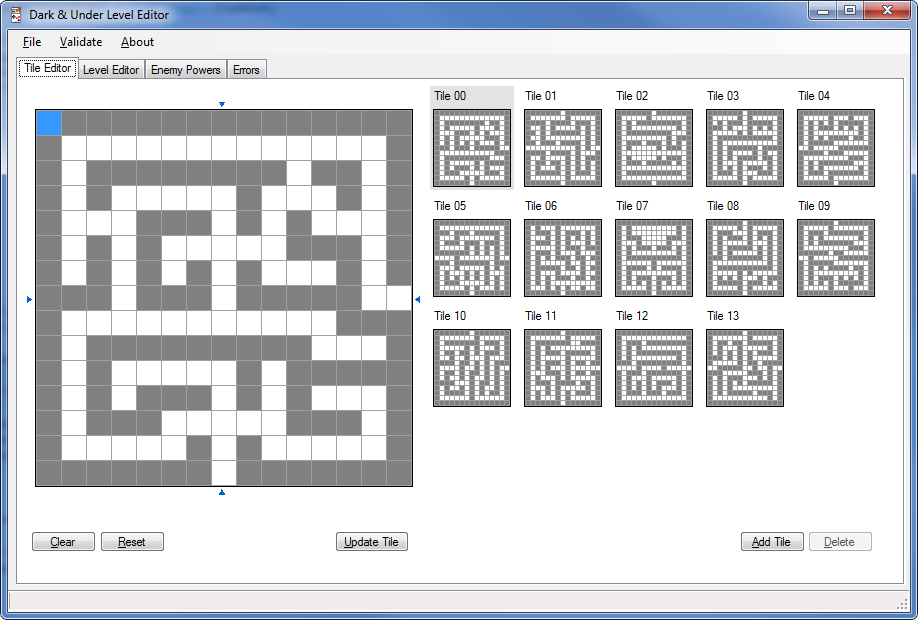
Making the process as simple as possible requires thought when designing and building your game. Sure, it can be added later but this may require some significant re-work of code.

This article describes how we developed Dark and Under and the associated level designer to use a single configuration file that both could read and manipulate.

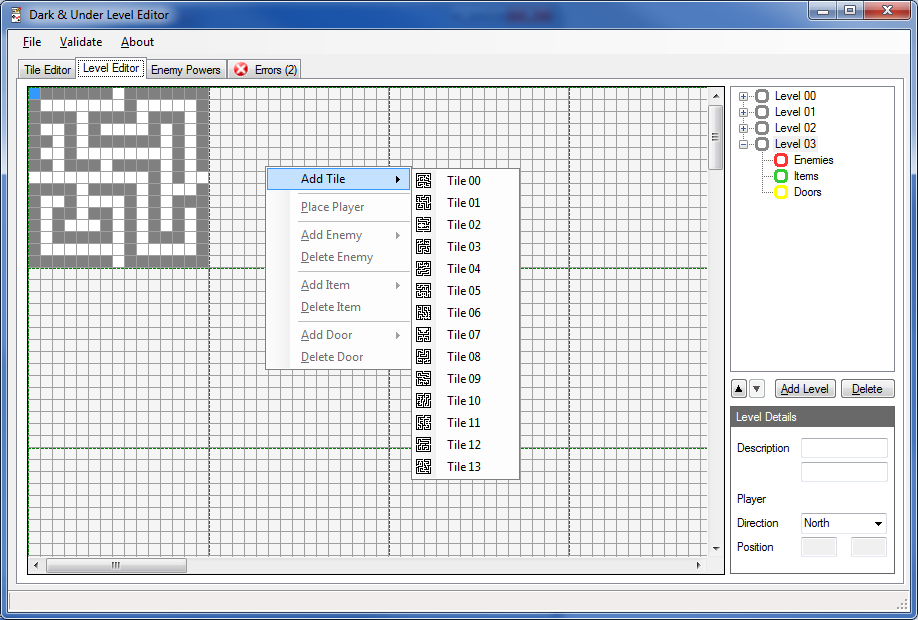
**The Dark and Under Level Designer in Action**

In Dark and Under, the various levels (or dungeons) are constructed from a palette of tiles. Each tile is 15x15 elements and each level can be up to 15 x 15 tiles in size. Although this doesn’t sound like much, the third level of the released game is only 5x4 tiles in size and it feels huge!

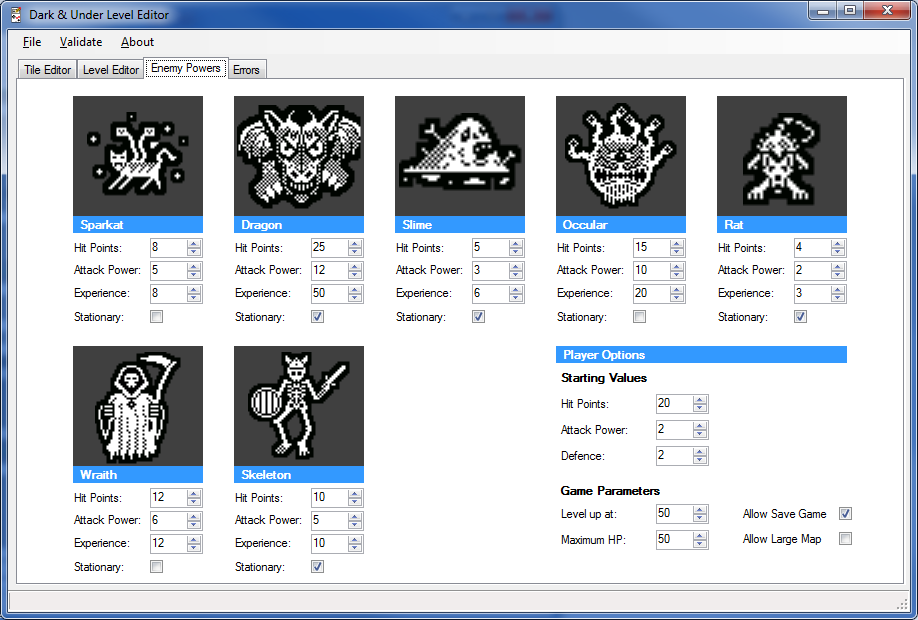
The screenshot below shows the tile editor – users can open an existing configuration file for editing from the *File* menu or start a new game configuration. New tiles can be created by clicking the *Add Tile* button or an existing one chosen for editing from the palette.



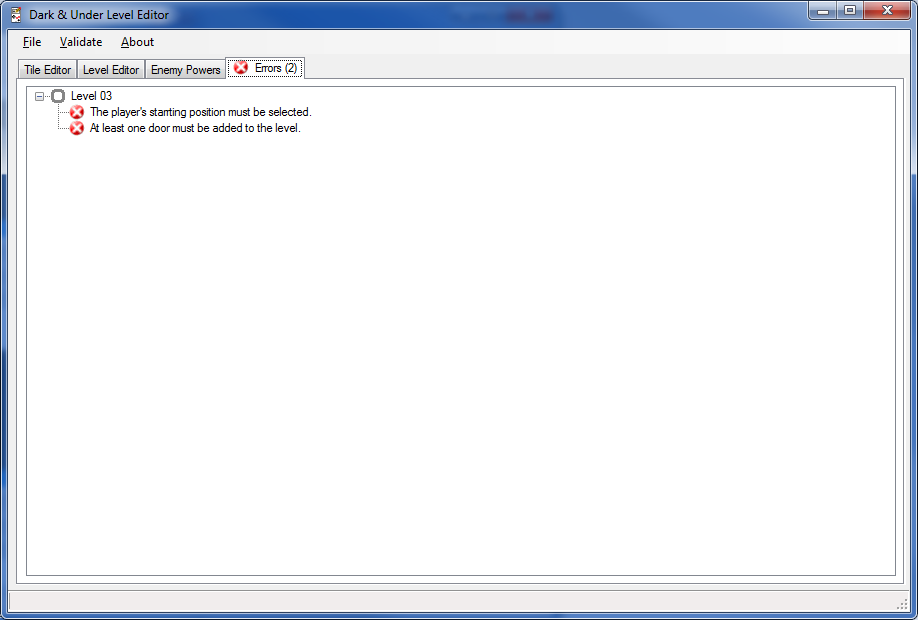
Once the tile design is complete, the levels or dungeons can be constructed by arranging the tiles and adding enemies, items and doors as required. Likewise, the initial position and direction of the player can be specified.



Enemy and player statistics can be altered and this will drastically alter the ‘beatability’ of the game (is that a real word or did I just make it up?)



Finally, any logical errors encounter while designing the levels are displayed on an errors tab. Double-clicking on an error will take you to the level in error and highlight the appropriate object. This screen is important as it ensures that the resultant configuration will result in a logical and playable game.



**The Configuration File**

In Dark and Under, all configuration items are stored in a single file called MapData.h – a valid C++ header file. The level editor knows how to read and manipulate this file in its native format and the resultant file will compile properly within the IDE.

The file has a number of sections that describe the overall game parameters (including starting health and attach poser for the player), enemy details and the tile and level configurations.

As you can see, the **#define** directive is used extensively as it allows the parameters to be incorporated into the game at compilation time without the need to have program logic embedded within the configuration file itself.

Our tiles and levels are encoded as arrays that are stored in PROGMEM. These can then be easily referenced by other sections of the code by simply **#including** the file where required as, after all, it is a valid class header file (.h).

#pragma once

#define START\_HP 20

#define START\_AP 2

#define START\_DF 2

#define MAX\_HP 50

#define LEVEL\_UP 50

#define ENEMY\_OCCULAR\_HP 15

#define ENEMY\_OCCULAR\_AP 10

#define ENEMY\_OCCULAR\_XP 20

#define ENEMY\_OCCULAR\_MV true

…

const uint8\_t PROGMEM tiles[] = {

0xFF, 0x81, 0xED, 0x05, 0xF5, 0x95, 0xD5, 0x05, 0xDD, 0x85, 0xB1, 0xA5, 0xED, 0x01, 0x7F, 0x7F, 0x40, 0x5E, 0x52, 0x5A, 0x4A, 0x6A, 0x02, 0x6E, 0x42, 0x5E, 0x54, 0x55,

...

};

const uint8\_t PROGMEM level\_00[] = {

72, 65, 76, 76, 87, 65, 89, 83, 32, 79, 70, // Level Caption Line 1

32, 84, 72, 69, 32, 68, 69, 65, 68, 32, 32, // Level Caption Line 2

7, 58, // Player Starting Pos

…

};

const uint8\_t PROGMEM level\_01[] = {

…

};

Building a level designer may not be as fun as building the actual game itself. However, it may entice people to build new level sets for your game and ultimately extend the life of the game.