TEnTENintro
Tomb Editor and Tomb Engine
An introduction

rickyturaz

September 21, 2025

Preface

Legal disclaimer

Tomb Engine uses modified MIT license for non-commercial use only. For more information, see license. Tomb Engine is unaffiliated with the Crystal Dynamics group of companies or Embracer Group AB.

Tomb Raider is a trademark of the Crystal Dynamics group of companies. Tomb Engine team is not responsible for illegal use of Tomb Engine source code and built binaries alone or in combination with third-party assets or components. Tomb Engine source code is released as-is and continues to be maintained by non-paid contributors in their free time.

6

Contents

Part I

Introduction

Chapter 1

History of Tomb Raider Level Editors

1.1 Back to 2000: *Tomb Raider Level Editor*

Tomb Raider marked a sensational new approach to 3rd person gaming. Fans not only fell in love with Lara and her moves, but with the imaginative and intriguing worlds of her adventures. It all started with Lara's visit to some Egyptian ruins back in 1996, and now with the release of the Tomb Raider Level Editor has come full circle, offering a different sort of adventure in another Egyptian setting. *Tomb Raider Chronicles* marks the end of the line of Tomb Raider games made with these development tools; but rather than viewing this as an end, the release of the editor makes it seem more like a beginning...

The **Tomb Raider Level Editor (TRLE)** includes a tutorial that will walk you through the basics needed to create your own stand alone Tomb Raider levels (but please read the legal disclaimer about commercial use of this product). Even though you will not be able to edit objects or animations (that means Lara's outfits), you have a wonderful variety of object sets from which to choose. You can sculpt and design on many different 'levels' – trigger events, create awe-inspiring spaces, simple to complex. . . and as you experiment you will learn more about what can be done, and quite possibly discover new methods of applying the knowledge you have acquired.

We sincerely hope you will enjoy inventing, designing, and building with

and for Lara as much as we have over the past 4 years. We thank all those who have held the enthusiasm for the Tomb Raider series, thereby contributing to its success. We wish you happy adventuring with Lara and the tools used to create her worlds. [2]

1.2   Paolone's *Next Generation Level Editor*

The **Next Generation Level Editor**, often abbreviated **NGLE**, is a modified version of the Tomb Raider Level Editor, created by Paolone, and released in January 2007. [11]

Tomb Raider Next Generation (TRNG) tools, improve the TRLE tools used to build custom levels with the engine, supplied by Eidos, of Tomb Raider - The Last Revelation.

Many objects have beed added, some imported by other TR adventures, like boat or frog-man, other builded ex-novo like Detector or Elevator.

There is a new scripter program named NG_Center. This program other to build your script.dat supplies other little tools. [4]

1.3   MontyTRC89's *Tomb Editor*

Tomb Editor (TE) is a level editor designed for the full range of classic Tomb Raider game series (1-5), as well as for contemporary engine reimplementation projects and game engines designed for community modding and level building. [3]

Chapter 2

Editor and engines

It's time to introduce some fundamental definitions before actually getting started.

Remark. *A **Tomb Raider editor** is an application used to make Tomb Raider games.*

In this book we'll use the ***Tomb Editor***, also abbreviated in ***TE***. It's fundamental to know what is an engine:

Remark. *A **Tomb Raider engine** refers to the software framework used to run the Tomb Raider games.*

Remark. *Each Tomb Raider game can use one and only one engine, to be chosen during game development in the Tomb Raider editor.*

There are several engines available, all supported by the Tomb Editor:

- **Tomb Raider 1 TR1X** and **Tomb Raider 2 TR2X** by *Lost Artefacts Team*: these are enhanced engines for Tomb Raider 1 and 2, offering smooth, true 60 FPS gameplay and expanded creative tools. They preserve the original feel while adding flexibility through full gameflow scripting in a simple JSON format, and an injection system that unlocks new features and removes engine limitations.

- **Tomb Raider 2 TR2Main** by *Arsunt* and **Tomb Raider 3 tomb3** by *Troye*: these are the original Tomb Raider 2 and 3 engines with numerous bug fixes and restore features from the console versions. From

lightning to UI elements and gameplay details, they bring back lost aspects which are fully customizable by the players. Both engines utilize a game-flow scripting language for simple gameplay tweaks, while keeping the experience authentic.

- **Tomb Raider 4 Original TRLE** by *Core Design*: this is the original, unmodified Tomb Raider 4 engine, exactly as it was released with the original TRLE editor. Perfect for creators seeking to build custom levels within the classic, traditional framework. *This engine has never received any update since its original release. It may not function correctly on model systems.*

- **Tomb Raider Next-Generation** by *Paolone*: this enhanced Tomb Raider 4 engine, featuring the Next-Generation addon, expands creative possibilities with advanced scripting tools and a plugin system. It gives level builders greater control over gameplay mechanics, allowing for complex interactions, custom events and deeper game logic customization. *Some anti-virus software, including* Windows Defender*, may flag this engine as a false positive. This engine is known to not function correctly on modern systems. External software, such as* dgVoodoo*, might be required for proper operation.*

- **Tomb Engine** by *MontyTRC89 and The Tomb Engine Team*: Tomb Engine is a powerful and flexible engine with modern enhancements, including 60 FPS rendering, SSAO, dynamic shadows, and more. Built for both experienced developers and newcomers, it offers advanced features while remaining easy to use. With Lua programming support and an intuitive node system, it allows for deep customization and creativity. *This engine is actively being developed, and changes to the scripting API may occur.*

In this book, we'll use *Tomb Engine*, abbreviated into *TEN*.

Chapter 3

Installing Tomb Editor

First of all, you need to download and install the Tomb Editor pack on your computer. It is available eg. here.

The default route of Tomb Editor installed is `C:\Tomb Editor`. The contents of this main folder are:

- Tomb Editor program.

- Side programs dedicated to Tomb Editor: SoundTool, TombIDE, Wad-Tool.

- Most of the files which are necessary to start a basic project and level for Tomb Engine. (But texture files for room faces must be find somewhere else. But this is still not necessary now, when you start reading this tutorial.)

- Other important files for Tomb Editor pack.

So when you have the Tomb Editor pack installed on your computer, then you are just ready to start building levels for Tomb Engine. [1]

Chapter 4

Tools

Part II

Getting started

Chapter 5

Starting a new project

After the installation of Tomb Editor pack, you are ready to make your very
first Tomb Engine project. (Level map file extensions are well-known as
"PRJ" files, but do not misunderstand: what we call "project" now is not
a level, but a whole level set - i.e. your current Tomb Engine game itself,
which will be released when you fully made it.)

But where do you need to place your projects? Well, NOT in Tomb
Editor main folder - that is a place you usually never modify while editing.
I suggest placing all of your projects nicely collected in a so-called general
project folder. This could be called eg. `"My_Tomb_Raider_projects"`, cre-
ated manually. (I created it in Documents folder.)

Each project you make will be placed in its own main folder. Does it
mean now you should also create manually a project main folder in the
general folder? No, there is a TombIDE wizard which will do the whole
project-creating procedure for you.

Projects are handled in **TombIDE (Tomb Integrated Development
Environment or TIDE)** program, that is why the whole project-creating
procedure is also being done there. So start `TombIDE.exe`, and the panel of
TIDE Start page opens up. Click on "Create a new project" button now.
The first page of a new panel opens up (General Information, 5.1):

- Let's suppose the project you start now has the name of "Lara's Newest
  Adventures". So type it now here.

- Click on "Browse" button, find and select the general project folder.

- After that, the little window in the middle of this first page shows that
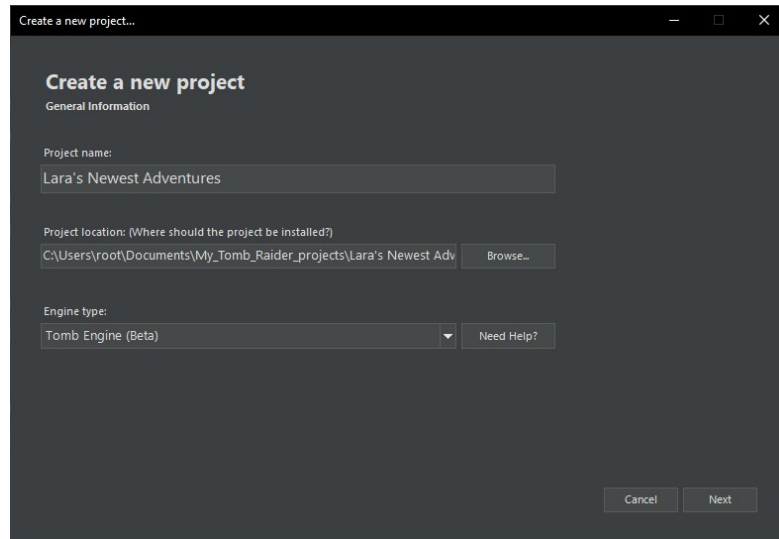
Figure 5.1: General Information

a subfolder in the general project folder will be created as the main folder of this project, having the project name.

- The engine type you choose now is naturally Tomb Engine.

Now click on "Next" button to continue the procedure on the next page of the panel (Extra Options, 5.2).

I suggest changing nothing here. Which means level map files will be handled in a folder called "Levels", which is a subfolder in the main folder of the project. (I mean, this is the default place for level map files, and you, the beginner probably should keep it like this.)

Now click on "Create" button here, then look at the increasing bar at the bottom of the panel. When the bar is at 100 %, then you get a message that the project has been successfully created (5.3).

And this project main folder has been also created on the selected route, with the basic contents a TEN project should have. (Including Levels folder - still being empty - in that default position., 5.4)

Double-click on that row (or click on "Open selected button" below), so the project opens in TIDE, you will be able to work on it. Each project opened in TIDE has more pages, now you can see its Level Manager page. (See the panel header which names the current project.)
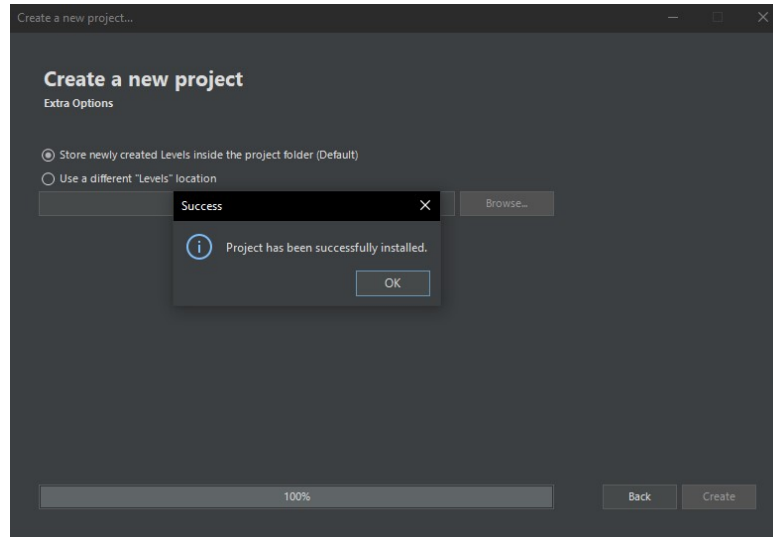
Figure 5.2: Extra Options

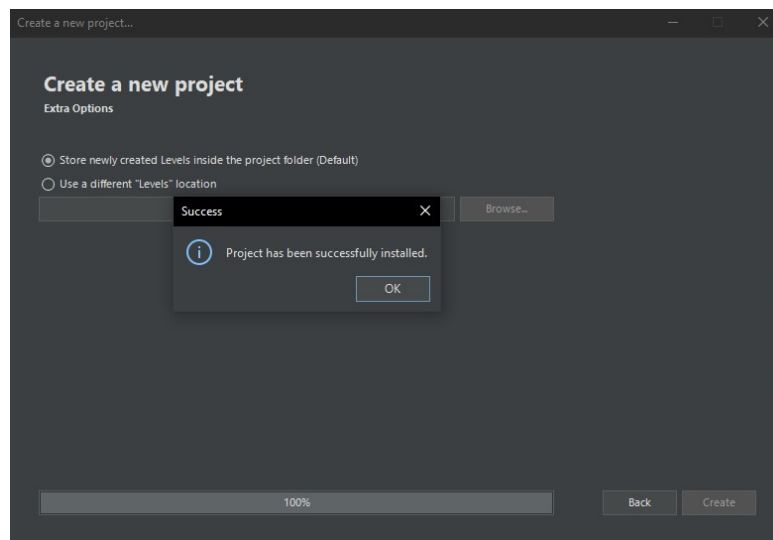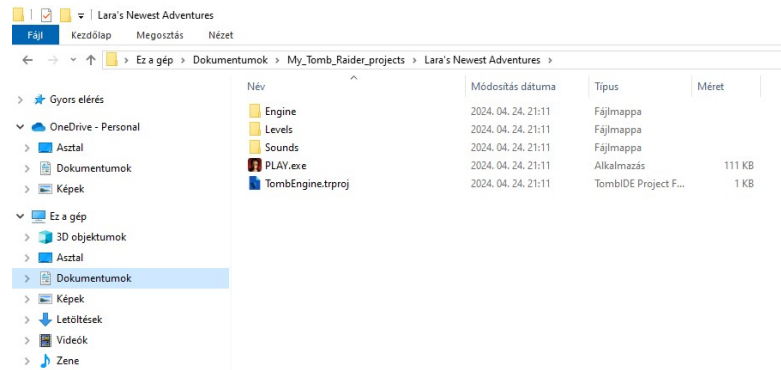Figure 5.3: Project has been successfully installed

Figure 5.4: Project directories have been created (Since TEN 1.7., the project main folder has an Assets subfolder as well.)

Now click on the blue TIDE icon in the upper left corner of the page, then a menu opens. One of the menu options is an arrow, with "Back to Start Window..." name. Click on it to close this project now, going back to TIDE Start page. [1]

Chapter 6

What is a level?

Before we discuss how to start a new level, important to understand, that actually what a "level" is. *(Don't do anything now, just read and listen.)*

6.1   Level map

Level maps are the editable versions of levels. So when you create, modify a level in the level editor program, then the changes are saved and stored in level map files. TE has its own level map formula, whose extension is called PRJ2.[1]

At the moment you don't have PRJ2 files, but soon you will create them and save them in Levels folder of the project main folder. (In fact, in sub-folders of Levels folder, and each subfolder is dedicated only to one level of the project, having a name which nicely refers to the level name.)

6.2   Playable level

The playable version of the level is a level file what the game will play as a level. This playable file is made by a conversion, converted from the PRJ2 level map file of this level. (The conversion will be a very simple task to you: only a simple click on a button.)

---

[1]Previously level map files was known as files with PRJ extension.

6.3   Level script

Script means game or level data, described simply by typing some texts. There is a tool in TE pack dedicated to edit script. This is TIDE you have already used to create your project, and you definitely will use it later much to edit your script. [2]

6.4   Item files

An item file (attached to a level) contains all the Moveable and Static objects (Lara, creatures, statues, furniture, effect emitters etc.) and sprites [3] which can be used in that level. In TE you can select more than one item file for a level. (And if an object is there in both, then you can set which one of them should be applied in the level.) [4]

- Builders using TRLE/NGLE previously definitely know that the extension of an item file is `WAD`, acronym standing for *Where's All the Data?* [5], made/edited probably with WadMerger program. But it is not obvious any more, if your level editor program is Tomb Editor. In TE WAD extension is usually supported, but not preferred.

- `WAD2` extension means an enhanced item file (eg. with the feature of UV-mapping [6] which is not possible in WAD files). WAD2 files are made and useable only for TE levels, it is the preferred extension here. There is a tool in TE pack dedicated to edit item files, called WadTool. Just open a WAD in it, and save it. It will be saved automatically as

---

[2]In TRLE or NGLE there is a "level block" in the script, where all the scripted data of that level are typed. In TE it works the same way - except if the engine for your TE project is TEN. In that case script works a bit differently, as you will see. (Though, a level block also exists there.)

[3]In computer graphics, a **sprite** is a two-dimensional bitmap that is integrated into a larger scene, most often in a 2D video game. [6]

[4]Unlike the older editors, where only one item file could be selected per level.

[5]WAD is generally a main architectural component of retro games - see e.g. https://en.wikipedia.org/wiki/Doom_modding for usage in *Doom* custom games.

[6]**UV mapping** is the 3D modeling process of projecting a 3D model's surface to a 2D image for texture mapping. The letters "U" and "V" denote the axes of the 2D texture because "X", "Y", and "Z" are already used to denote the axes of the 3D object in model space. [8]

a WAD2 item file. (Or you can naturally arrange even a brand new WAD2 in WadTool.)

- TEN engine is able to use only specific WAD2 files, which means WAD extension is not supported for TEN engine. (Except if WAD has only objects having non-TEN specific behaviors, like Statics.)
  You can find a menu option in WadTool, to convert your other item files into a TEN WAD2.
  "Specific" means eg. TEN Lara object is optimized for TEN, so a "casual" (non-TEN) Lara would not be animated properly there. (Later we discuss it in this tutorial - but not the details, because it is not a TEN WAD2 tutorial.)

WAD2 files for a TEN level should be placed in Levels folder, in the subfolder of that level. (It is recommended to use a sub-subfolder there for it.) - Though, technically it is not a must, they can be placed even anywhere in your computer.

6.5 Texture files

A texture [7] file (attached to a level) contains all the texture tiles which can be placed on room faces (floor sector, ceiling sector, wall section) in that level.

In TE:

- you can select more than one texture file for a level, placing tiles even from each in your level,

- not only TGA extension is supported for texture files, but even other ones: BMP, JPG, PNG etc.

Texture files for a TEN level should be placed in Levels folder, in the subfolder of that level. (It is recommended to use a sub-subfolder there for it.) - Though, technically it is not a must, they can be placed even anywhere in your computer.

---

[7]A *texture map* refers to a 2D image ("texture") that adds visual detail to a 3D model. The image can be stored as a raster graphic. A texture that stores a specific property—such as bumpiness, reflectivity, or transparency—is also referred to as a *color map* or *roughness map*. [7]

6.6   Sound files and their catalog files

Sounds saved in sound files can be emitted mostly by Moveable objects (mostly Lara or other creatures), but even by any effect (like a rumbling earthquake). Etc.

Sound files have WAV extensions - just like in the old times.

Sound files are not directly attached to a level map, for two reasons:

- Sound files for a TEN level should be placed in Sounds subfolder of project main folder. (Though, technically it is not a must, they can be placed even anywhere in your computer.) Initially you can find all the original sounds here of the legacy engines, in their own subfolders. - But naturally you are allowed to edit these contents, modifying, deleting, adding sound files here.
  These places are needed to be attached to that level.

- There must be one catalog file (or, unlike the older editors, even more catalog files), attached to the level, where the sounds you want to attach are named. The main reason is that sounds can have different properties for different levels, and the catalog will describe the properties that this sound will have on this level
  Sound catalogs can be even older types, familiar from TRLE/NGLE (Sounds.txt or SFX/SAM files of WAD files), or the ones having XML extensions, which are made for TE. Later you will be able to make custom XML files, but some of them are created when saving a WAD2 file. And there are even some preset XML catalogs in Catalogs/TEN Sound Catalogs subfolder of Tomb Editor main folder.
  There is a tool in TE pack dedicated to edit sound catalogs, called SoundTool. If you want, just open a non-XML catalog into it, and save it as an XML one.

6.7   Conclusion

When you start building a new level for your project, then these are your tasks, recommended in this order:

1. Create a PRJ2 level map file.

2. Add this level to the project.

3. Check the needful contents of the level script. If your script has less contents than that, then the level is useless in the game.

4. Check the main settings (in TE) for the level. If they are wrong, then the level is useless in the game.

5. Check the crucial attachments.

6. You can try to edit something in the level map - then save it.

7. Convert the PRJ2 level map into a TEN playable level.

8. Start the level in the game, try what you have just edited in the map.

[1]

Chapter 7

Starting a new level map

1. Open the "Lara Newest Adventures" project in TIDE, remaining on Level Manager page.

2. See the left big window here, with the title of "Level list". As the text says in it, click on it, or - because the text is not available when there is at least one level in this window - click on the "+" button on the upper left corner of the window.

3. The panel of "Create a New Level" opens (7.1):

   - **Level Name**: this level name will be shown in the game for this level. Let's say it is "Test Level 1" now.

   - **Use custom PRJ2 / DATA name**: if you tick it, then you can add an alternative name key instead of the default one.
     Name keys are the names of the most important files of this level. The default name key is mostly the Level Name, except: there cannot be spaces. So eg. for Test Level 1, level files will be auto-created with this name key later: Test_Level_1.prj2, Test_Level_1.ten etc.
     Let's suppose we don't want the default name key now, so tick the option and type "Test1" as the name key. (So the file names will be Test1.prj2, Test1.ten etc.)

   - **Generate Lua script**: as I said above, there is a scripted part even for TEN. If you untick it, then the wizard will skip this part, not creating the needful contents of the level script. So later you

33
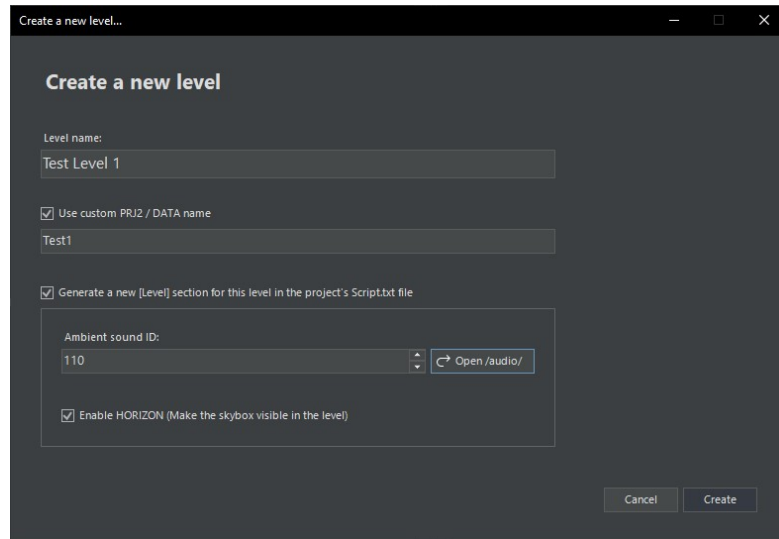
Figure 7.1: *Create a New Level* panel

have to create it manually (but before starting the level at the very first time in the game).

Some values in the scripted part you need to define now:

– **Ambient sound ID**: the initial ambient background noise of the level. I suggest accepting this 110.wav now, because later you can change it any time. Or you can click on Open /audio/ button to open Engine \Audio subfolder in the project main folder now, to try to find another one. [1]

– **Enable HORIZON (Make the skybox visible in the level)**: the sky above you can be visible in the outdoor areas of the level only if it is ticked. I suggest ticking it, because later you can change it any time. footnoteSkybox and horizon are working together, but they are not the same. But it doesn't matter now.

4. Click on Create button.

---

[1]Audio subfolder has the original audio file set of The Last Revelation game. Later you can change the contents any time. Now I suggest choosing a WAV above 100 value here, because in that set, background noises are placed on those IDs. However, for TEN you can use any ID for a background noise - but we won't discuss it in this tutorial.
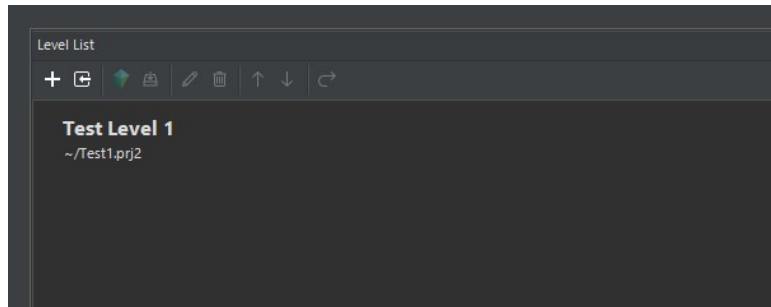
Figure 7.2: *Level list* panel showing level you've created

What happened now are:

- As I said, each level should have a folder dedicated to it, in Levels folder of this project. Now a dedicated folder to Test Level 1 automatically has been created there, having this level name.

- A Test1.prj2 file has been also automatically created, in Test Level 1 folder.

- The main settings of this level have also been done properly.

- Some crucial attachments of the level has been automatically attached. (Later we discuss it in the tutorial.)

- In Level list window of this project (on Level Manager TIDE page), a row for this level has been also automatically created (7.2).

- The needful contents of the level script has been also automatically created. (Later we discuss it in the tutorial.)

Now you can click on the arrow menu option in the menu of the blue TIDE icon to go back to the Start page of TIDE.

Chapter 8

Time to run the first (default) room

8.1   Open your level in *Tomb Editor*

Once you're in the TombIDE start page, double click on your project *Lara's Newest Adventure* to open it (or alternatively, select the project and click on the "Open selected project" button), then:

- double click on your "Test Level 1"

- or alternatively, select "Test Level 1", then click on the icon in the upper part of the Level, similar to a green prysm. This is the icon of the *Tomb Editor*.

This action will open the Tomb Editor (8.1).

8.2   Overview of *Tomb Editor* UI

The User Interface of Tomb Editor is composed by different panels or *windows* (8.2):

1. **Editor Window**: main working window for modeling, texturing and viewing in 2D, 3D and preview modes

2. **Tool Palette (floating)**
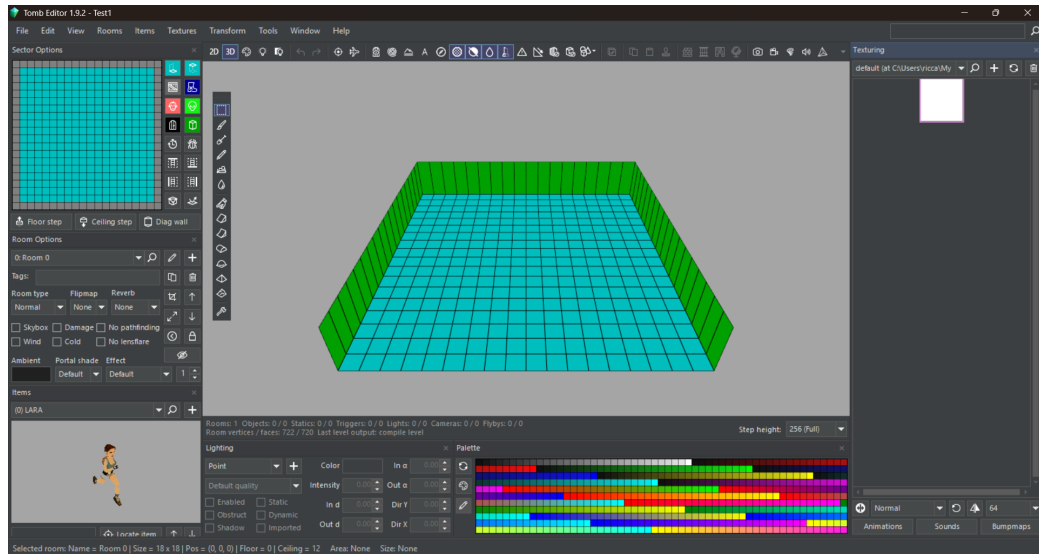
Figure 8.1: *Tomb Editor* main screen

3. **Editor Window Buttons**

4. **Drop down menu bar**

5. **Drop down menu search**: search commands in drop down menu bar by their name

6. **Sector Options**: top down view of the selected room; secondary working window

7. **Room Options**: create rooms; add/edit features in rooms

8. **Item Browser**: preview, select and place objects. Please notice there's Lara here!

9. **Lightning**: select, place and adjust lights

10. **Palette**: provides colors used for transparency; quick way to assign colors to lights and objects

11. **Texture Panel**: select tiles for texturing model; set animation ranges and texture sounds
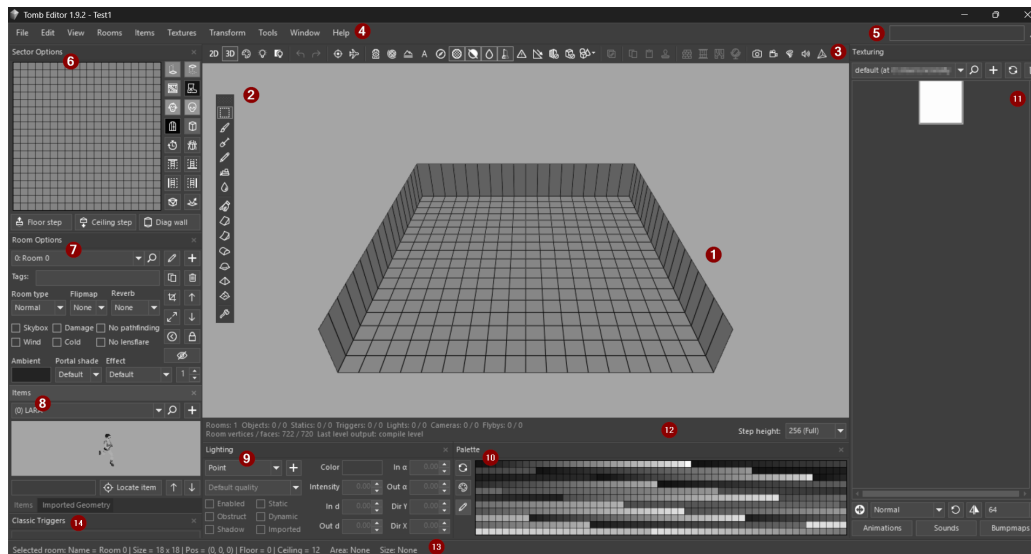
Figure 8.2: Tomb Editor *UI*

12. **Statistics**: room location and statistics; total number of triggers and objects in project

13. **Room info box**

14. Other windows - **please note:** by default, the following panels are active in the user interface left bar, but not visible since the bar itself is too short to include them all:

    - **Trigger List**
    - **Imported Geometry Browser**

Panels can be hidden or shown, using the `Window` menu bar. As well, here you can restore the default layout.

Panels can be moved within the interface, placing them where you prefer by dragging and dropping their title.

8.3   Default level properties

When you create a new level, by default the TombIDE tool creates for you a very basic level, containing one squared room, having size 18x18x3. We'll

discuss about size later on in details, talking about blocks, rooms and their properties.

For now, just look in front of you: you can see the square room represented in a tridimensional view in the editor window. You can check this is the 3D view of a rectangular cuboid, by RMB clicking the room and moving the mouse while keeping the mouse button pressed.

Please note, this room created by default:

- **is already textured in white** - in other words, basically is fully painted in white color (floor, walls and ceiling).

- is dark, since no lights are defined by default. Don't worry about that, we'll plenty discuss about lights later on.

Now you're not able to see here this room is white and dark: please, trust me for now.

Well, when you create a new level, TombIDE automatically adds:

- a default *item file*, or WAD, that is a revisited version of objects from TR4

- a default *texture file*, containing a single white texture the whole room is automatically "painted" with, as above mentioned

- default *sound files and their catalog*, from TR4 as well

For now, don't worry about where you can check and change the attachments in Tomb Editor: we'll discuss the through entire chapters dedicated to each of them later on.

8.4   Wait! Something is missing...

There's only one step before we can play the level. Guess what is missing? Our Lara.

This is a general rule:

Remark. *A level can be played only if Lara's object is placed into the level.*

To place Lara in the room, it's enough to (8.3):

- RMB click on a floor box inside the room

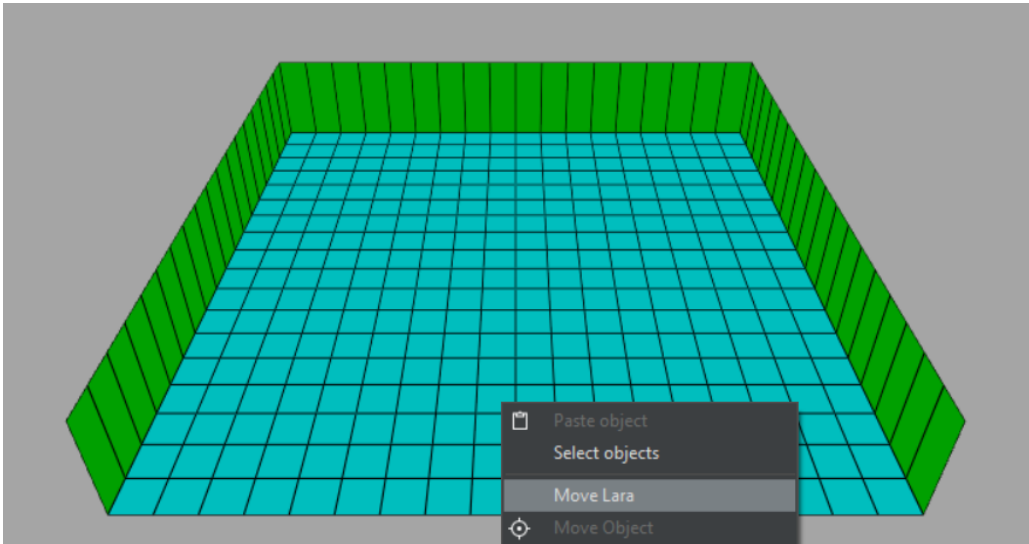- click on "Move Lara"

Now you can see Lara is placed in the room.

Figure 8.3: Placing Lara

8.5   Ready to play your first room?

Please note by placing Lara into the level, we actually introduced a change that is lost, unless we save it.

Everytime we update the level, we can acknowledge about unsaved changes by looking at the Tomb Editor window title: if there are unsaved changes, a star characted will appear near to the level title (8.4). Anyway, if you close the window forgetting the changes, the editor will ask you if you want to save them.

To save changes for this level, you can alternatively:

- Click on `File > Save level` main menu entry.

- Use `Ctrl`+`S` on the keyboard.

All done! Now we can build the level and play it by alternatively:

- press `F5` on the keyboard

- or click on the "Build level and play (F5)" button in the top menu.

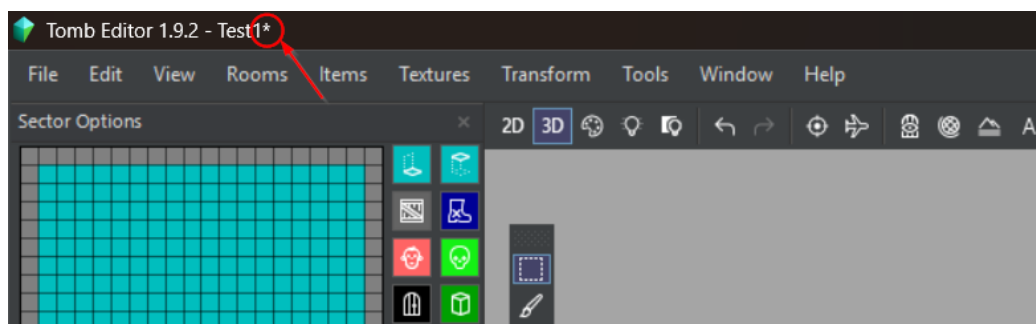Now the Tomb Engine will build the level for you and run it.

Figure 8.4: Marker for unsaved changes

Well, I know... it's a unique, squared room all painted in white where Lara is trapped forever into: you'll need to close the game manually (please note a quick way is using key combination $\boxed{\text{Alt}}+\boxed{\text{F4}}$), cause we've not set anything but Lara's position, neither a way out of the room nor anything to say to the game the level is actually finished. But, **congratulations**! You've build and run your first level successfully.

<div align="center">* * *</div>

As you've seen, this first level has been actually, fully, automatically generated by the Tomb Integrated Development Environment. In the next chapters, we will see what the application done now for us, having a short overview of the level main settings and components.

Chapter 9

Level scripts

All the script files for a TEN project are available in `Engine \ Scripts` subfolder of the project main folder. Some of them are not important for us now, and some of them won't be important even later. - So let's see which script files are important, when you start building a new level:

- `Gameflow.lua`

- `Strings.lua`

- `.lua` file with the key level name (in `Levels` subfolder of the `Scripts` folder), i.e. `Test1.lua` now.

You can open and edit these script using whatever text editor you'd like. However, there are two integrated ways to open scripts in TombIDE:

- Scripting Studio in TombIDE

- Visual Studio Code

But, first of all: why these files have `.lua` extension?

9.1 What is Lua?

Lua is a lightweight and high-level programming language designed mainly for embedded use in applications. Lua is cross-platform software.

In video game development, Lua is widely used as a scripting language, mainly due to its perceived ease of embedding, fast execution, and short

learning curve. Notable games which use Lua include *Roblox, Garry's Mod, World of Warcraft, Payday 2, Phantasy Star Online 2, Dota 2, Crysis* and many others. Some games that do not natively support Lua programming or scripting have this function added by mod.

In 2003, a poll conducted by GameDev.net showed that Lua was the most popular scripting language for game programming. [5]

Scripts written in Lua language have `.lua` file extension.

9.2   Scripting Studio in TIDE

Once opened the project in TombIDE, you can open the integrated Scripting Studio clicking on the icon *Scripting Studio* on the left. Naturally this is the page where you can edit your script.

The main parts of this page are (9.1):

- Dropdown menu.

- Toolbar with buttons.

- The big window to see/edit the script: any text you can see here is some textual data for your game/level. Or, if a row starts with a `--` sign, that means the row is just a comment, a note, so you can type anything here.

- File Explorer panel on the right side: Here you can find all the available script files of the project. Click on any file name of File Explorer, to open that file in the big window. There are tabs above the window, for the script files are just open there. Gameflow.lua are always open initially. (Naturally you can close any tab if you just doesn't care about it any more.) Click on any of those tabs to see its contents in the window.

9.3   Visual Studio Code

9.3.1   What is VS Code?

Visual Studio Code (VS Code) is an integrated development environment developed by Microsoft for Windows, Linux, macOS and web browsers.
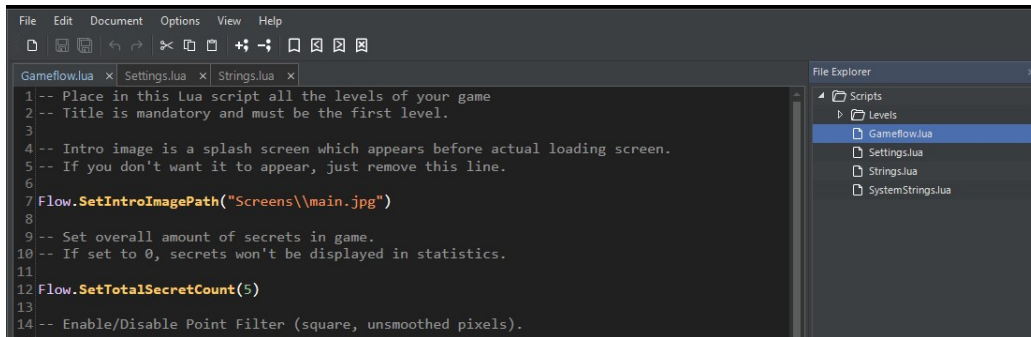
Figure 9.1: TombIDE Scripting Studio

Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded version control with Git. Users can change the theme, keyboard shortcuts and preferences, as well as install extensions that add functionality.

Visual Studio Code is proprietary software released under the "Microsoft Software License", but based on the MIT licensed program named "Visual Studio Code - Open Source" (also known as "Code - OSS"), also created by Microsoft and available through GitHub.

In the 2024 Stack Overflow Developer Survey, out of 58,121 responses, 73.6% of respondents reported using Visual Studio Code, more than twice the percentage of respondents who reported using its nearest alternative, Visual Studio. [9]

9.3.2 Integration with TIDE

Once installed VS Code on your PC, you can click on the *Open Project in Visual Studio Code...* button on the left side of TombIDE. This action opens VS Code automatically placed on the `Engine \ Scripts` folder and opens the `Gameflow.lua` script.

Furthermore, if not installed already, TIDE will propose you to install the *Lua language server* extension in VS Code [1].

---

[1]The *Lua language server* extension provides various language features for Lua to make development easier and faster. With nearly a million installs in Visual Studio Code, it is the most popular extension for Lua language support.

9.4   Gameflow.lua

This script file exists initially in TE pack, and always very simple. However, it is crucial, your level won't run in the game (properly) if you do not set here the main data of the level (properly). Gameflow.lua is is separated into three parts with the

```
---------------------------------------------------
```

rows (which are comments, because they are start with

```
--
```

signs):

- In the first (uppermost) part there are some **general settings of the game**, like eg. the maximum number of secrets. - This part is unimportant to us now (because their default values are nicely useable - or, at least, their effect is currently irrelevant), so we skip it.

- In the second (middle) part you can find the **level block for the title**, existing initially. Yes, as I said, technically the title is also a level - but we won't discuss it now, it is not a title tutorial. What matters is it works nicely for your project even now, so now you don't need to care about its default script values or any other part. (Though, as I said, it is not the well-known The Last Revelation title, but something much simpler.)
  However, you need to know that not the "title" name turns a level into title, but the fact that its level block is the first (uppermost) level block in Gameflow.lua.

- In the third (lowest) part you can find the **level blocks for all the "real" (i.e. non-title) levels of your game**. This part is initially empty, but, when a wizard creates a level for the project, then the level block for this level will be automatically created (placed in the order of creating those levels).

So what we are interested in now is how this level block looks for `Test1.prj2`, which the rows and their default values are in this block. Initial level block rows are simple commands for that level. ("Officially" I shouldn't call them commands, but something else. However, I bet you will remember them more easily, if I call them commands. In latter tutorials we will be

more precise, but in a basic one like this it is enough.) - These commands are, from above to bottom (technically the command order doesn't really seem important here, though):

```
Test1 = TEN.Flow.Level()
```

This command indicates that here starts a level block. The Test1 tag of the command indicates that all the commands of this block will start with this tag.

```
Test1.nameKey = "Test1"
```

It tells that the name key of this level will be Test1. (Please notice that title level has no name key defined this way.) As you can see, the command tag is the name key you set. But it is not necessary, you can change the tag manually even now, so they will be different. (Technically you can also change the name key manually - but don't forget to change it in all the places where it is already used.)

```
Test1.scriptFile = "Scripts\\Levels\\Test1.lua"
```

As I told above, there should be a LUA file with the name key of the level in Scripts\Levels folder, which file is defined here. (Please notice that \\ sign are used instead of \ sign.)

```
Test1.ambientTrack = "110"
```

As you set it previously, the initial ambience background noise for the level should be 110.wav.

```
Test1.horizon1.enabled = true
```

As you set it previously, the sky above you can be visible in the outdoor areas of the level. (That is why it is "true".) Please note from horizon1 that even more than one horizon are available at the same time.

```
Test1.levelFile = "Data\\Test1.ten"
```

As I told above, the playable level that belongs to Test1.prj2 level map should be called Test1.ten. And, as I also told above, each TEN file will be automatically available in Engine\Data subfolder of the project main folder, including Test1.ten. These infos are defined here.

```
Test1.loadScreenFile = "Screens\\loading.png"
```

This command sets the image shown when the game loads that level - this is `loading.jpg` now, should be placed in `Engine\Screens` subfolder of the project main folder.

```
TEN.Flow.AddLevel(Test1)
```

This command tells that the level block for the level with Test1 tag name (not Test1 name key!) ends here, and the playable level with these block parameters is set as a part of this game.

These default values are proper for our purposes, so we won't change them now. (But you can naturally do that later, though.)

Be awared it's possible to create Test1.prj2 level map file manually: in that case, you'll need to add this level block manually to Gameflow.lua as well. We'll see how to create a level manually later on.

9.5   Strings.lua

The TEN script file **used to edit the texts which are seeable in the game** is called `Strings.lua`, existing initially in TE pack.

With comments, this file is also separated into parts. We are interested now only in the lower part now, which the comment names as "Level name strings". There are some initial (and perhaps ununderstandable - but it doesn't matter now) level names are set here. But when the wizard created Test1.prj2, then a new entry was also created here, telling that the name you will see in the game for the level which has Test1 name key is Test Level 1, as you set it previously:

```
Test1 = { "Test␣Level␣1" },
```

Naturally you need to type this entry manually, if you create `Test1.prj2` file manually.

9.6   Test1.lua

This file has been automatically created when the wizard created Test1.prj2. The contents of the file isn't important now, we won't use these entries now, to test this new level. (This will be the place later of the thing I've just called "additional scripting".)

```
-- FILE: Levels\Test1.lua
LevelFuncs.OnLoad = function() end
LevelFuncs.OnSave = function() end
LevelFuncs.OnStart = function() end
LevelFuncs.OnLoop = function() end
LevelFuncs.OnEnd = function() end
LevelFuncs.OnUseItem = function() end
LevelFuncs.OnFreeze = function() end
```

However, the fact that this file exists by the wizard means you should create this non-existing file manually, when you create your level manually:

1. There is a dropdown menu option, a toolbar button and even a shortcut key in Scripting Studio, to open a new script file. When you do this, a panel pops up (9.2). Set here the file route (in Levels subfolder of Scripts folder), the file name (typing Test1) and the format (LUA), then create Test1.lua for Test Level 1. (That `--` FILE comment will be automatically created now.)

2. Type the basic contents in Test1.lua.

3. Don't forget to refresh the script by saving in TIDE the script file you have just modified.
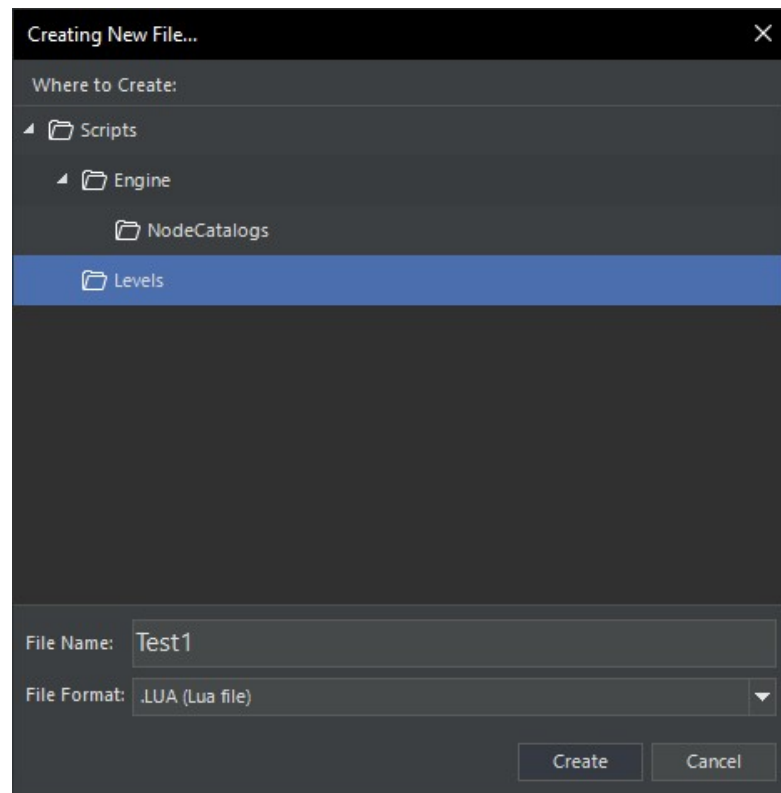
Figure 9.2: *Creating new file...* popup

Chapter 10

Level settings

The main TE settings for a level map can be checked in dropdown `Tools\`
`Level Settings` menu, in Tomb Editor once opened the level.

So open Test1.prj2 in TE, and check these settings (10.1).

If the wizard created the PRJ2 file for you, then the values here always
should be the proper ones. But if you create it manually, then you need to
set the proper values:

- **Game version to target**: naturally you choose here TombEngine now.

- **Folder in which all runtime game components reside**: select Engine folder of the project main folder.

- **Target folder and filename for level file**: select Test1.ten in `Engine\` `Data` folder of project main folder. (I mean, Test1.ten naturally doesn't

Figure 10.1: *Level Settings*

51
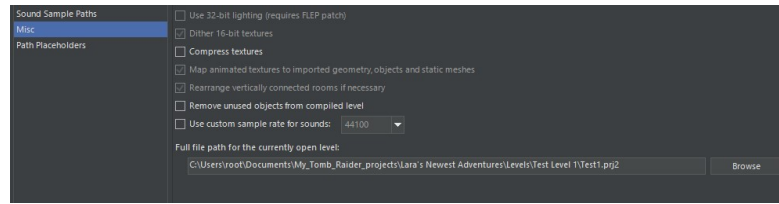
Figure 10.2: Level Settings *Misc* page

exist now if you've never built your PRJ2 level. If so, you should manually type the file name now.)

- **Path of Lua level script file**: select `Engine\Scripts\Level` folder of the project main folder.

- **Target executable that is started with the 'Build and Play' button**: select TombEngine.exe in `Engine\Bin\x64` folder of the project main folder. (Or choose the EXE in x86 folder, if your system configuration is different.)

Don't forget to also check *Misc* page here (10.2): **Full file path for the currently open level** value should be the route of this PRJ2.

Chapter 11

The WAD

11.1 *Item browser* panel

When TombIDE created the level for us, it automatically attached to it what we defined as item file, or *WAD*. At a glance, we can check it in Tomb Editor, by looking at the *Item browser* panel (11.1): we can find objects there, Lara's first.

This is the WAD2 which has been automatically attached to Test1.prj2, when the wizard created this PRJ2. Yes, this WAD2 is not an item file with the custom objects you want for your level - however, its contents is a proper object set to start learning how to handle Moveable objects and sprites in TEN. (This WAD2 has no Static objects - but I am sure soon you will be able to add easily Statics to this, with WadTool program.)

This WAD2 is called `TombEngine.wad2`, and you can find it in `Assets\ Wads` subfolder of the project main folder. This WAD2 is also available in Tomb Editor main folder, same subfolder.

`TombEngine.wad2` is a basic TEN WAD2 file made and mantained by TEN developers, which contains a TEN-specific Lara, some other TEN specific-objects - and also some other, general (but surely TEN-compatible) objects, which are needful to edit a level.

We'll deep dive contents of the WAD for Tomb Engine, including compatibility, in later chapters.
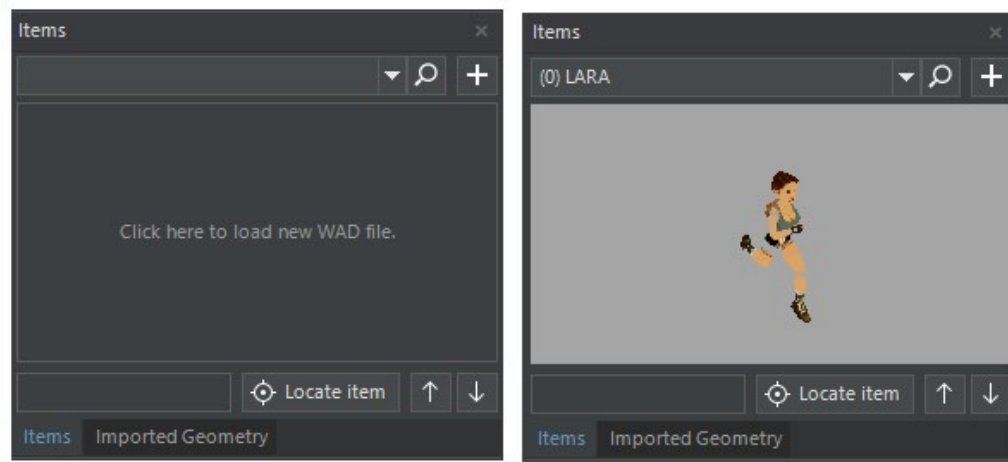
Figure 11.1: *Item browser* panel, with a WAD loaded (on the right) vs. without (on the left)

11.2   Level settings for WADs

General way to check which WADs have been loaded into the level is through the *Level settings* - as already seen, we can check the level settings through main drop down menu, *Tools* and then *Level settings....*

Under the *Object Files* section of level settings (11.2) we can check the `TombEngine.wad2` is used and has been successfully loaded.

Actually this page is a list which can contain more than one WAD file: in fact, in TE you can select more than one item file for a level and if an object is there in both, then you can set which one of them should be applied in the level. Now it's enough to know this feature exists: we'll find out more about later on.
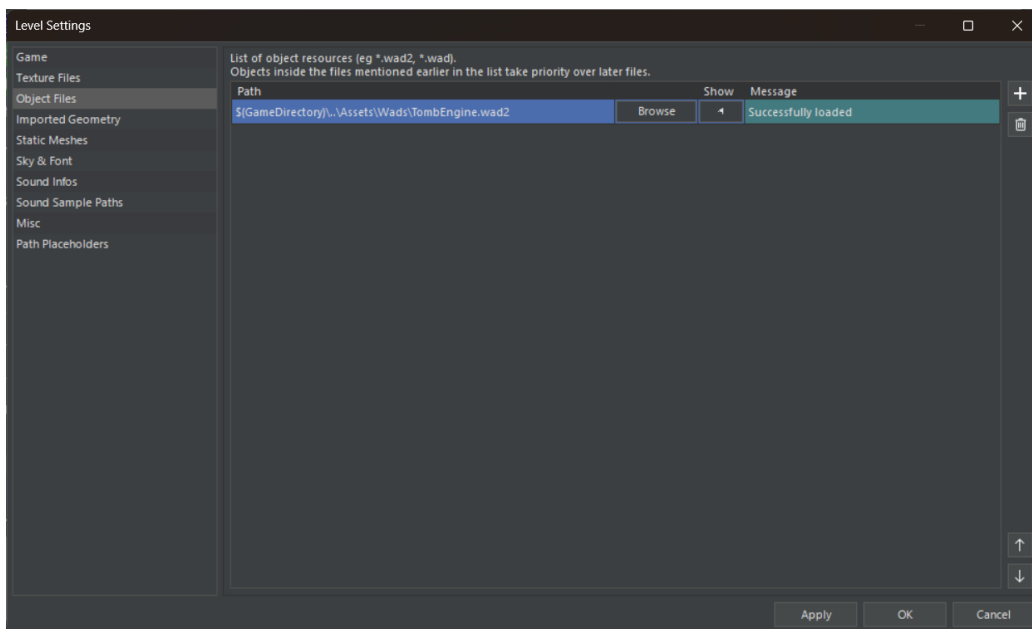
Figure 11.2: List of loaded WADs in *Level settings*

Chapter 12

Texture file

12.1 *Texture* panel

You remember the room you played was fully painted in white? Well, this was because when we created the level, TombIDE automatically attached into the level a simple 64x64 pixel white image with grey borders, placed into the `Assets\Wads` and called `default.png`.

You can check it by looking at the *Texture* panel (12.1).

The whole room is painted, or more precisely *textured* using that image.

12.2 Level settings for textures

Similarly as for WADs, you can check which texture files your level is using through the Level settings - section *Texture Files* (12.2).
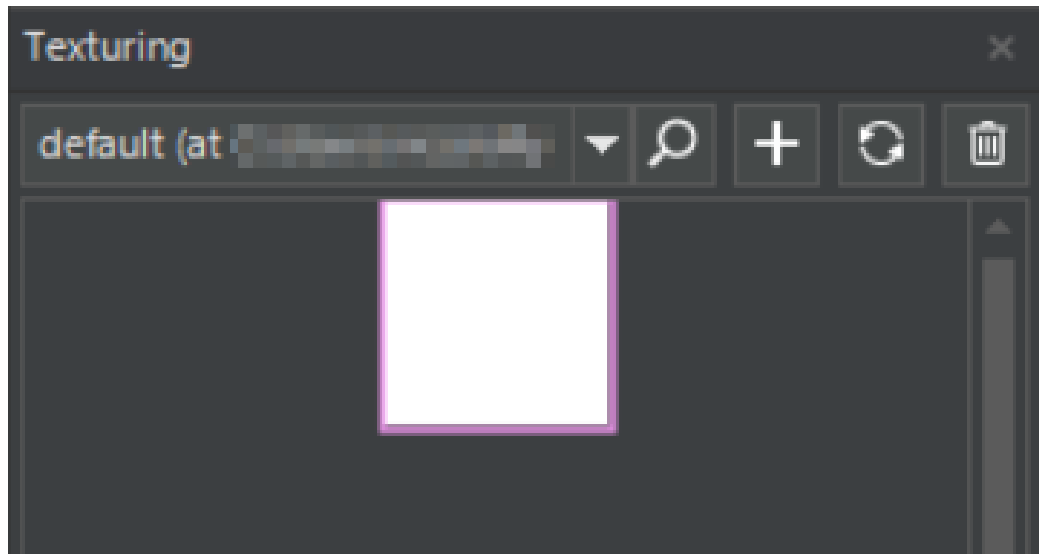
Figure 12.1: *Texture* panel with *default.png* texture loaded
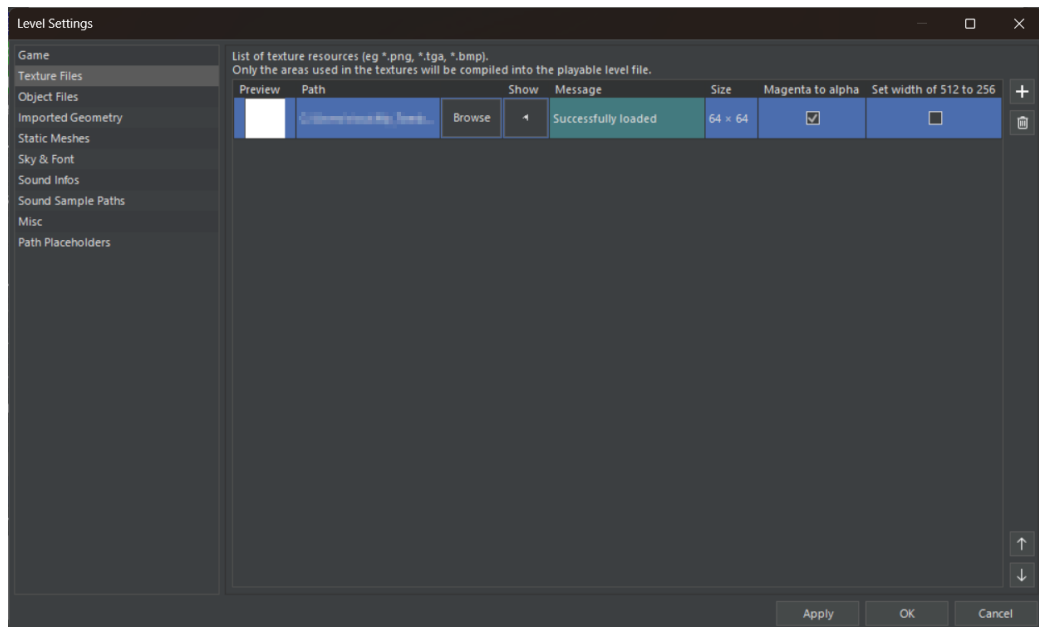
Figure 12.2: List of loaded textures in *Level settings*
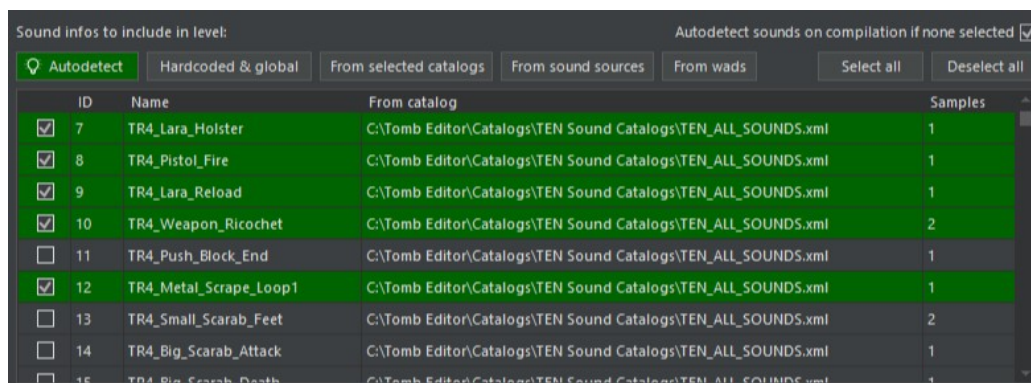
Chapter 13

Sound files and their catalog files

Let's analyze now what TombIDE done to make us hear the proper sounds in the game. Open the level settings, then:

1. Click on *Sound Sample Paths* tag on the left side, to open that page (13.1). You will see a row on this page. If you click on Browse button now, next to this row, then the abovementioned Sounds subfolder opens. Which means sound files for this level will be chosen from this subfolder (and its subfolders). - This is good to us now, so you can close the panel now.

2. Click on *Sound Infos* tag on the left side, to open that page (13.2). You will see a row in the upper window of this page called *Sound Catalogs*. If you click on "Folder" button now, in the middle of this row, then the abovementioned `Catalogs/TEN Sound Catalogs` subfolder opens. Which means the catalog we will use for this level is placed in this subfolder. (You can see in that row that this catalog is `TEN_ALL_SOUNDS.xml` now.) - This is good to us now, so you can close the panel now.

3. Still here on *Sound Infos* tag (13.2), "Autodetect sounds on compilation if none selected" option is probably ticked (in the middle of TE, on the right side). But if it is not, then tick it now. Now the proper sounds will be automatically selected when you convert this PRJ2 into a playable level later. When you come back to this page after the first conversion,
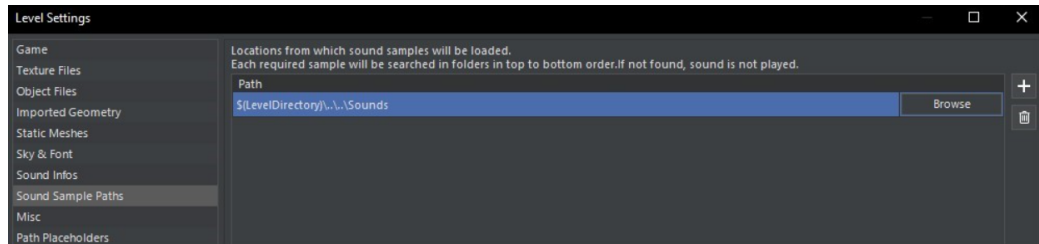
Figure 13.1: *Sound Sample Paths* in level settings
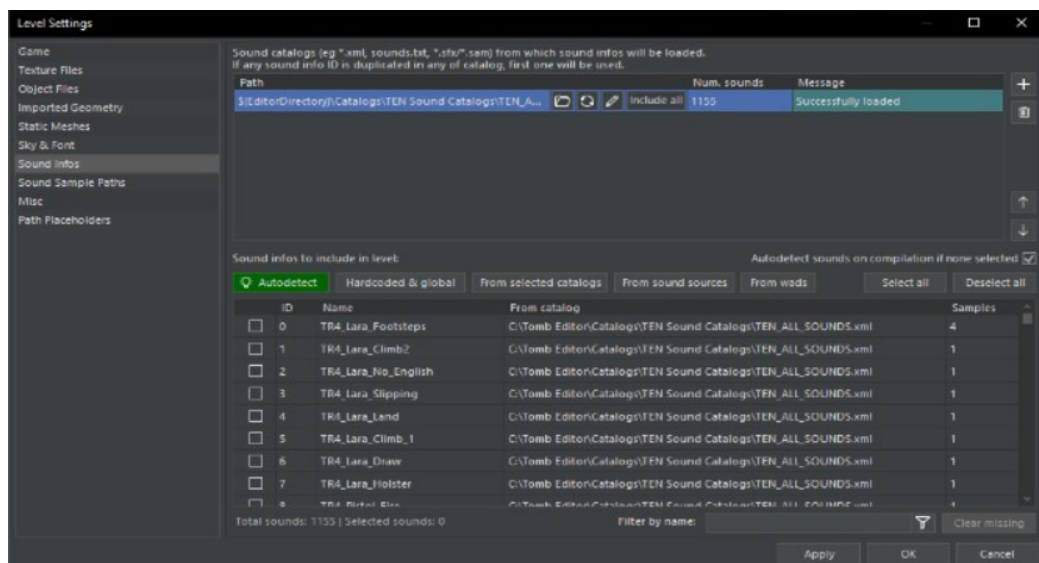
Figure 13.2: *Sound Infos* in level settings

then you will realize that some rows in the lower window went green. Which is good, they are the autodetected sounds of this catalog (13.3).

Figure 13.3: Autodetection of *Sound Infos*

Chapter 14

Tomb Editor *modes*

Get back to the main Tomb Editor window. We'll learn now there are 4 different ways you can see you level, according to what you actually need to do. First of all, you can find the four modes as the very first menu buttons (14.1). Here they are:

- **2D mode**: identified by the *2D* icon, shows the level map as *seen from the sky*. Keyboard shortcut to activate this mode is $\boxed{\text{F1}}$.

- **3D mode**: identified by the *3D* icon, shows the level represented in a 3D view. This is the default mode when you open a level. Keyboard shortcut to activate this mode is $\boxed{\text{F2}}$.

- **Face Edit (texture) mode**: identified by the *palette* icon, shows the level as the 3D view, plus showing textures applied to the surfaces (floor, walls and ceiling). Keyboard shortcut to activate this mode is $\boxed{\text{F3}}$.

- **Lightning mode**: identified by the *light bulb* icon, shows the level as the Face Edit mode, plus previewing how lights are rendered on surfaces (floor, walls and ceiling). Keyboard shortcut to activate this mode is $\boxed{\text{F4}}$.

In the screenshot (14.1) you can see I've actually highlighted for you 5 buttons. The fifth button on the right, with the *light bulb on a paper sheet*, is actually an option of the *lightning mode* - that's why it's there close to it. This option, called **Drawn untextured in Lightning Mode**, hides the textures in the lightning mode if activated, allowing to see you attributes of
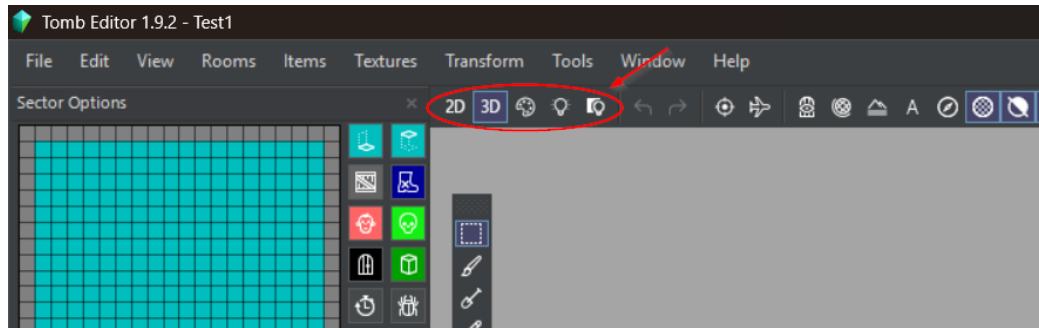
63

Figure 14.1: *Modes* buttons in main toolbar

lights you've added in your level, as if surfaces hadn't been textured. You
can toggle it on and off by just clicking the button.

   Please note: now we've learnt about the modes and how to switch between
them. We'll go deeper into details about usage of each one, while we learn
things step by step together. For now, what I ask you to do is to:

1. switch back and forth between the *3D mode* and the *Face Edit mode*
   (remember: F3 and F4 keys).

2. Note the difference between them: in Face Edit mode, you can see
   the surfaces of the room have been textured by TombIDE, using that
   fully-white texture we've checked before.

3. *Just remember this will be the place where we'll apply textures.*

Chapter 15

Convert the level map into a playable level

When we tested our first level before using the "Build level and play (F5)" function, what we actually made the application do is:

1. put all the stuffs we defined in previous chapters together;

2. *mix them up* to create the playable file for our level and save it into the `Engine \ Data` subfolder of our project. If you open this folder, you'll find the `Test1.ten` file, updated when you built your level the last time.

3. open the game, starting from that playable level.

15.1 Wrap-up

That's all for the introduction! Together we've created our first level, played it, looked at the main components of TombIDE and Tomb Editor user interfaces and had an overview of the main *building blocks* of a level and a simple game containing just it.

Part III

Rooms: floors and ceilings

Chapter 16

Introduction

16.1 Blocks, sectors and clicks

Blocks, squares, sectors and clicks: get used to these terms because you'll hear them frequently. The Tomb Editor is designed to work with a basic *building block*, proportioned to Lara and her movements.

Levels are built by connecting a series of rooms comprised of walls and building blocks. The floor and ceiling of these rooms are sectioned into squares or sectors. The building blocks are created when you raise a square up from the floor or lower one down from the ceiling. Four mouse clicks up or down equals the width of these squares sections and creates a perfect cube *Remember all those "blocks" Lara pushed and pulled around?!*

The building blocks are not limited to cubes and columns with flat tops. Corners of the surfaces can be pulled up or down to create angled slopes and *organic* surfaces - great for creating rocky caves or sand dunes. [2]

16.2 Basic definitions

Let's start with the default room: in the previous part, we've learnt that when we create a new level, TombIDE creates for us a level containing one room having size 18x18x3. Ok but.. 18x18x3 what?

It's time to introduce some formal definitions, forming a common dictionary.

Remark. *Property of a room is its **area**, measured in* sectors *or* squares.

This corresponds to the area of room's floor and room's ceiling, excluding walls.

For our default room, its area is 18 squares. We can check it by alternatively counting squares in the Editor Windows, looking at the Sector Options or switching to the 2D mode in Tomb Editor.

Please note in the 3D view:

- in the main editor window, floor and ceiling are represented in light blue color, while walls are represented in green;

- in the Sector Options panel, floor and ceiling are represented in light blue color, while walls are represented in grey.

Remark. *Given a room, the area of its floor is equal to the area of its ceiling.*

Remark. *Number of floor sectors is equal to number of ceiling sectors.*

Now, recall in mind a way to define position of objects in the three-dimensional space we live our life: we can rely on the cartesian coordinate system, where we use the three axes x, y, z. Up to now, we cared about x and y. Let's introduce the way we can set z coordinate (height) and so, clicks:

Remark. *A **click** or full step is a linear transformation can be applied to a floor or a ceiling sector to increase or decrease its height.*

By using clicks, we can make a block.

Remark. *A **block** is the solid shape we can obtain by applying clicks to floor (raising) or ceiling (lowering).*

Finally, we reach the definition of the *Tomb Raider cube*:

Remark. *A **Tomb Raider cube** is a cube-shaped block, where every edge is 4 clicks long.*

The definition of the Tomb Raider cube is useful because it's a simple way to remember Lara's sizes compared to her world and the real world. Let's compare Lara's sizes and her movements with the cube, to better understand (16.1).

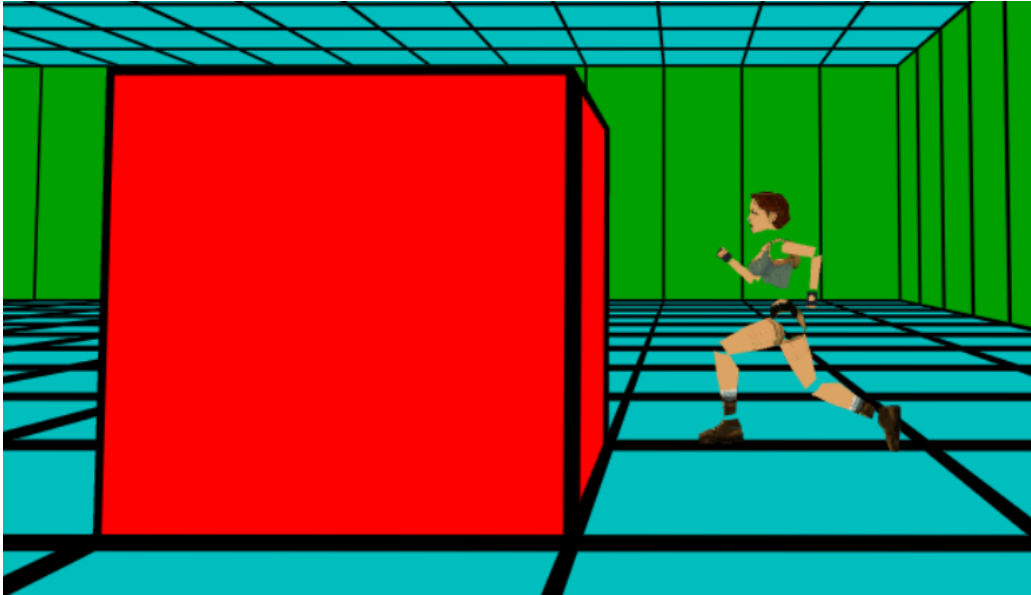**Remark.** *Lara's height is 3 clicks.*

Figure 16.1: Lara compared to the Tomb Raider cube.

Remark. *Assuming Lara's real height is 1.80m according to the original story [10], a* click *is 60cm and a* Tomb Raider cube *is 2.40m high.*

In the next chapter we'll check together all the ways Tomb Editor provides us to model our room geometry. But before, we need to understand how we can move ourselves in the space.

16.3   Tomb Editor 3D controls

Open *Test Level 1* we've created in the previous chapters. By default, the view is pointing to the north, to the center of the room, seen from above (45 degrees).

There are plenty of ways to move our view, basically looking at room and its contents from a different point in the space. Let's examine the available options.

- `MW`: by moving mouse wheel up and down, we can respectively increase or decrease the zoom. Geometrically, we are moving the camera along the vector perpendicular to its plan.

- **RMB** or ⬆️↦→: by keeping the right mouse button down and moving the mouse, we can rotate the camera. Geometrically, we're moving the camera along the sphere centered to the point in front of our view. We can use arrow keys as well.

- **MMB**: by keeping the scroll wheel button down and moving the mouse, we'll move the center of our view horizontally or vertically. Geometrically, we're moving the camera along its plan.

- `View - Relocate camera` or **ALT** + **z** : mouse pointer will change to a cross. By **LMB** we'll be able to set the new center of our view.

To **reset the camera position to its default** (useful in case you lost yourself), you can use `View - Reset camera position` or **F6** .

16.3.1 Fly mode

There's another funny and useful way to move our point of view, through the *fly mode*. Fly mode is activated by `View - Toggle fly mode` or **SHIFT** + **z** .

Once activated, it's like being a fly! You can move your camera by combining:

- mouse movement;

- **WASDQE** keys;

- **SHIFT** to go faster.

You can deactivate the fly mode anytime through **ESC**.

16.3.2 3D controls options

Settings for 3D controls can be changed by the `Tools - Editor Options`, then `3D controls panel`.

In particular, please note you can customize the **Fly Mode move speed** here.

Chapter 17

Modeling blocks and holes

17.1 Selecting sectors

Now that we know how to move our point of view in the room, let's see how
we can select sectors.

Sectors can be selected either:

- In the editor window by LMB : a **single** click selects a single sector,
  by moving the mouse keeping the button pressed you can select bigger
  areas.

- In the `Sector Options` panel by LMB : a **single** click selects a single
  sector, by moving the mouse keeping the button pressed you can select
  bigger areas.

Selected area:

- will be highlighted in red in the main editor window and its border will
  appear red as well in the `Sector Options` panel. **Please note**: take
  care of carefully single click here and do not click more than one time.
  We'll see afterwards what happens when you click more than one time;

- is described in the *room box info* panel of the Tomb Editor: here you'll
  find start and end position of the selection, as well as the its size.

To **deselect** a sector or a bunch of sectors, just press ESC .

17.2   Creating a block

Once a floor sector (or bunch of sectors) is selected, please check carefully: if you rotate the view in order to see both floor and ceiling together, you'll see not only the floor is selected, but also the ceiling. This is very important.

Remark. *Selecting a floor area selects the corresponding area of the ceiling and viceversa.*

To create a block from the floor:

- select the **Drag** command in the Tool Palette;

- LMB click on the selected area on the floor, keeping the button down;

- move the mouse up.

You'll see while you move the mouse up, selected floor raises step by step - or, using definitions we introduced before, *click by click.*

Similarly, you can create a block from the ceiling, by clicking on the selected area on the ceiling, but moving the mouse down this time.

Building blocks will actually change the height of the selected area: you can check it in the *room box info* panel of Tomb Editor (look at $y = [f, c]$).

17.3   Creating an hole

In the same way, we can set holes in the floor and ceiling, by reversing the mouse movement, that is:

- for the floor, move the mouse down;

- for the ceiling, move the mouse up.

Digging holes will actually change the height of the selected area: you can check it in the *room box info* panel of Tomb Editor (look at $y = [f, c]$).

17.4 Keyboard shortcuts

Once one or more sectors are selected, there are some useful shortcuts can be used for quickly creating blocks and holes, which are:

- For the floor:

  - Q : raises the floor 1 step up.
  - A : lowers the floor 1 step down.

- For the ceiling:

  - W : raises the ceiling 1 step up.
  - S : lowers the ceiling 1 step down.

These shortcuts can be used in the same way also when squares are raised or lowered by one of their sides or corners.

Chapter 18

Selecting and trasforming sides and corners

18.1 Sides

Once one or more squares are selected (by pressing LMB just once or via drag and drop), we said you can see them highlighted in red.

If you furtherly click on the selected area, you'll see a white arrow appears in each selected square. The arrow points to the specific side of the square the further transformation will be applied to.

By using now the ⌖ command - or the shortcuts QAWS we mentioned before - you'll see the raising or lowering transformation will be applied *to the selected side only* (or sides, if more than one square is selected).

If you further click, you'll see the arrow rotates. Generally, once the selection is done, sides are switched clockwisely:

- **1** LMB: selects north side;

- **2** LMB: selects east side;

- **3** LMB: selects south side;

- **4** LMB: selects west side;

- **5** LMB: resets side selection (all sides are transformed together).

Remark. *In a floor/ceiling selection, if no specific side is selected (means, no arrow is shown), all the sides will be raised or lowered together according to the applied transformation.*

18.2   Corners

Once selected a floor/ceiling area highlighted in red, if we press `CTRL` key while clicking, we'll be able to select corners of the square, instead of sides. You can see it because arrows now point to the corners of the squares.

Here, once the selection is done, corners are switched clockwisely:

- **1** `LMB`: selects north-west corner;

- **2** `LMB`: selects north-east corner;

- **3** `LMB`: selects south-east corner;

- **4** `LMB`: selects south-west corner;

- **5** `LMB`: resets corner/side selection (all corners and sides are transformed together).

Chapter 19

Related tools in palette

Now that we know how to select sectors, sides and corners of floors and ceilings, it's time to have a look one by one to all the tools available in the *tool palette*, along with their *modifiers*: e.g. while using tools, by pressing the $\boxed{\text{Alt}}$ key we can change the tool behavior, in ways different *tool by tool*.

19.1   *Quads* and *flat quads*

Before proceeding, we need to introduce a definition related to sectors: the *quad*.

Remark. *Given a sector delimited by four points or corners, a **quad** is a sector where all four corners are belonging to the same plan in space.*

The editor 3D view helps you checking if a sector is a quad at a glance: it's a quad if there's no diagonal line draw on it, while it's not a quad otherwise.
Then, we introduce now the definition of *flat quad*:

Remark. *A **flat quad** is a quad where all corner points have the same height.*

19.2   List of tools

- **Selection** ⬚: this is the tool for perform selections. It's active by default and was active when we learnt how to select sectors.

- **Brush** ✎ : when a selection is active, this tool creates *smooth* blocks in LMB clicked floor/ceiling sectors within selection, up to 4 clicks. If no selection has been done, transformation is done directly where you click.

- **Shovel** ✐ : when a selection is active, this tool creates *smooth* holes in LMB clicked floor/ceiling sectors within selection, up to 4 clicks. If no selection has been done, transformation is done directly where you click.

- **Pencil** ✏ : when a selection is active, this tool creates 1 click-high square block in LMB clicked floor/ceiling sectors within selection. If no selection has been done, transformation is done directly where you click.

- **Bulldozer** ⌂ : given a sector where corners are not at the same height, it sets height of all corners to the minimum height of the corners, actually *flatting* the sector. Using drag and drop to other contiguous sectors it propagates the same transformation to them. Used within a selection, it applies only to selected squares - otherwise, where you click. **Please note**: this transformation doesn't apply simmetrically if used on ceiling or floor.

- **Smooth** ◌ : if a surface presents 90° edges between two connected points, this tool removes them making the surface smoother. Used within a selection, it applies only to selected squares - otherwise, where you click.

- **Drag** ⬦ : this is the default tool for creating blocks, as described in previous chapters.

  Modifiers:

  - Alt : while using this command, if you keep pressed the Alt key while pressing LMB and moving the mouse, you'll see the editor keeps the outside sectors next to the selection linked to the sectors we're raising or lowering - that is, *edges are smoothly updated coherently.* If you just release the Alt key while applying

the transformation, you'll see the tool switches to the default behavior.

This is useful to easily create square pools with a smooth border, for instance.

- **Ramp** ⬙: given a selection and moving the mouse while keeping LMB pressed, this tool creates ramps, upwards or downwards on floors or ceilings.

  Modifiers:

  - **Edges and corners**: if you apply this transformation while an edge or a corner is set, the editor will apply the transformation towards the selected direction.
  - Alt: while using this command, if you keep pressed the Alt key while pressing LMB and moving the mouse, you'll see the editor will apply the required transformation using *flat quads as much as possible*.
    This is useful to easily create stairs, for instance.

- **Quarter pipe** ⬙: given a selection and moving the mouse while keeping LMB pressed, this tool creates quarter pipes [1], upwards or downwards on floors or ceilings.

  Modifiers:

  - **Edges and corners**: if you apply this transformation while an edge or a corner is set, the editor will apply the transformation towards the selected direction.
  - Alt: while using this command, if you keep pressed the Alt key while pressing LMB and moving the mouse, you'll see the editor will apply the required transformation using *flat quads as much as possible*.
    This is useful to easily create stairs, for instance.

- **Half pipe** ⬙: given a selection and moving the mouse while keeping LMB pressed, this tool creates half pipes [2], upwards or downwards on floors or ceilings.

---

[1] A *quarter pipe* is, literally, quarter of a sphere or pipe.
[2] An *half pipe* is, literally, half of a sphere or pipe.

Modifiers:

  – **Edges and corners**: if you apply this transformation while an edge or a corner is set, the editor will apply the transformation towards the selected direction.

  – `Alt`: while using this command, if you keep pressed the `Alt` key while pressing `LMB` and moving the mouse, you'll see the editor will apply the required transformation using *flat quads as much as possible.*

• **Bowl** ⌐: given a selection and moving the mouse while keeping `LMB` pressed, this tool creates bowls, upwards or downwards on floors or ceilings.

Modifiers:

  – **Edges and corners**: if you apply this transformation while an edge or a corner is set, the editor will apply the transformation towards the selected direction. **Please note**: used on corners, this tool is equivalent of the half pipe tool.

  – `Alt`: while using this command, if you keep pressed the `Alt` key while pressing `LMB` and moving the mouse, you'll see the editor will apply the required transformation using *flat quads as much as possible.*
  This is useful to easily create thrones, for instance.

• **Pyramid** ◇: given a selection and moving the mouse while keeping `LMB` pressed, this tool creates pyramids, upwards or downwards on floors or ceilings.

Modifiers:

  – **Edges and corners**: if you apply this transformation while an edge or a corner is set, the editor will apply the transformation towards the selected direction.

  – `Alt`: while using this command, if you keep pressed the `Alt` key while pressing `LMB` and moving the mouse, you'll see the editor will apply the required transformation using *flat quads as much as possible.*

- **Terrain** ⬦: given a selection and moving the mouse while keeping LMB pressed, this tool creates rough (so, irregular) terrains, keeping them *connected* to the sectors external to the selection (so, with no 90° angles), upwards or downwards on floors or ceilings.

  Modifiers:

  – **Edges and corners**: if you apply this transformation while an edge or a corner is set, the editor will apply the transformation towards the selected direction.

  – Alt: while using this command, if you keep pressed the Alt key while pressing LMB and moving the mouse, you'll see the sectors external to the selection will be *detached* (so, introducing 90° angles).

- **Portal Digger** ⚒: let's skip it for now, we'll talk about this in later chapters, talking about connected rooms.

Chapter 20

Related transformations

Through the `Transform` menu, we can apply some massive transformations to geometry of room floor and ceiling.

20.1 Transformations applicable to selections

First of all, let's examine the ones applicable **only if a selection is active**.

- **Smooth random**: transformations randomly increasing or decreasing floor or ceiling *smoothly* - that is, producing no 90° angles between any points of the surface. These are:

    - **Smooth random floor up**: raises floor (shortcut: Ctrl + Alt + A);
    - **Smooth random floor down**: lowers floor (shortcut: Ctrl + Alt + B);
    - **Smooth random ceiling up**: raises ceiling (shortcut: Ctrl + Alt + C);
    - **Smooth random ceiling down**: lowers ceiling (shortcut: Ctrl + Alt + D).

- **Sharp random**: transformations randomly increasing or decreasing floor or ceiling *sharply* - that is, producing 90° angles between points of the surface. These are:

    - **Sharp random floor up**: raises floor (shortcut: Ctrl + Alt + E);

- **Sharp random floor down**: lowers floor (shortcut: `Ctrl`+`Alt`+ `F`);

- **Sharp random ceiling up**: raises ceiling (shortcut: `Ctrl`+`Alt`+ `G`);

- **Sharp random ceiling down**: lowers ceiling (shortcut: `Ctrl`+ `Alt`+`H`).

- **Convert to quads**: in case there are non-quads sectors into the selected floor or ceiling, these transformations converts these sectors into quads. These are:

  - **Convert floor to quads**: applies to the selected floor;

  - **Convert ceiling to quads**: applies to the selected ceiling.

- **Smooth**: transformations *smoothing* selected floor or ceiling, by removing all 90° angles between two points. These are:

  - **Smooth floor**: applies to the selected floor;

  - **Smooth ceiling**: applies to the selected ceiling.

- **Average**: given selected sectors, this transformation converts each one into a flat quad, high as the average of the height of the former corners the sector was made. These are:

  - **Average floor**: applies to the selected floor (shortcut: `Ctrl`+`Alt` +`I`);

  - **Average ceiling**: applies to the selected ceiling (shortcut: `Ctrl`+ `Alt`+`J`).

20.2   Transformations applicable to selections or whole rooms

Let's examine now the ones applicable **either if a selection is active or not**. Please note: if a selection is active, transformation applied to it; otherwise, it applies to the whole room.

- **Flatten**: given selected sectors, this transformation *flats* them to flat quads, high as the minimum height (floor) or maximum height (ceiling) of the points in the whole selection. These are:

    - **Flatten floor**: applies to the selected floor;
    - **Flatten ceiling**: applies to the selected ceiling.

- **Reset all geometry**: applies both **Flatten floor** and **Flatten ceiling** with a single click.

Chapter 21

Steps height

When we defined a *click*, we said we can call it a *full step*. This is because we can decrease the step used by transformations we apply.

Step height can be:

- Full: its height is 256 *what?*

- Half: its height is 128

- Quarter: its height is 64

- Eighth: its height is 32

This setting can be changed either by:

- using the *Step height* dropdown menu in main editor window

- through keyboard shortcuts ⌞ ì ⌟ (increase) and ⌞ ' ⌟ (decrease).

Remark. *By default, transformations applied by the editor (raising or lowering floor and ceiling sectors) use a full step, unless we change this settings.*

21.1   Related transformations

Modeling rooms using a step different than the full one (which is the maximum applicable), it could happen you'd like to increase the step height you're using. In this situation, you may want to *reset* floors and ceilings to that step. To do so, once changed the step and selected the area you'd like to reset, you can use **Re-align to step height** transformations, available in `Transform` menu - respectively:

- **Re-align floor to step height**: applies to the selected floor;

- **Re-align ceiling to step height**: applies to the selected ceiling.

Index

Bibliography

[1] AkyV. *TEN - How to start building a new project, a new level.* https://www.tombraiderforums.com/showthread.php?t=229293.

[2] Eidos Interactive Core Design Ltd. *Tomb Raider Level Editor manual.* https://core-design.com/goodies_tr5_downloads.html. 2000.

[3] MontyTRC89. *Tomb Editor page on GitHub.* https://github.com/MontyTRC89/Tomb-Editor.

[4] Paolone. *Official website of the Next Generation Tools.* http://www.trlevelmanager.eu/ng.htm.

[5] Wikipedia. *LUA.* https://en.wikipedia.org/wiki/Lua.

[6] Wikipedia. *Sprite (computer graphics).* https://en.wikipedia.org/wiki/Sprite_(computer_graphics).

[7] Wikipedia. *Texture Mapping.* https://en.wikipedia.org/wiki/Texture_mapping.

[8] Wikipedia. *UV mapping.* https://en.wikipedia.org/wiki/UV_mapping.

[9] Wikipedia. *Visual Studio Code.* https://en.wikipedia.org/wiki/Visual_Studio_Code.

[10] WikiRaider. *Lara Croft.* https://www.wikiraider.com/index.php/Lara_Croft.

[11] WikiRaider. *Next Generation Level Editor.* https://www.wikiraider.com/index.php/Next_Generation_Level_Editor.