Tomb Editor Manual

rickyturaz

August 23, 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.

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. [3]

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. [5]

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. [4]

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, WadTool.
- 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"`, created 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

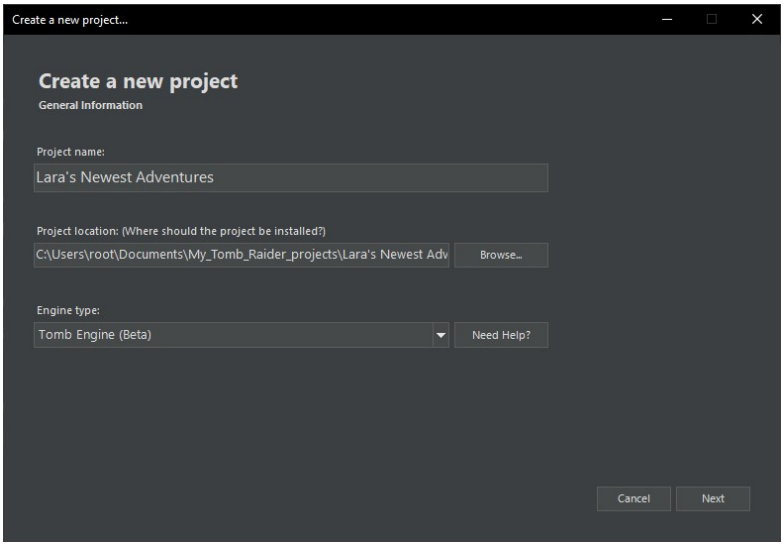

Figure 5.1: General Information

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.)

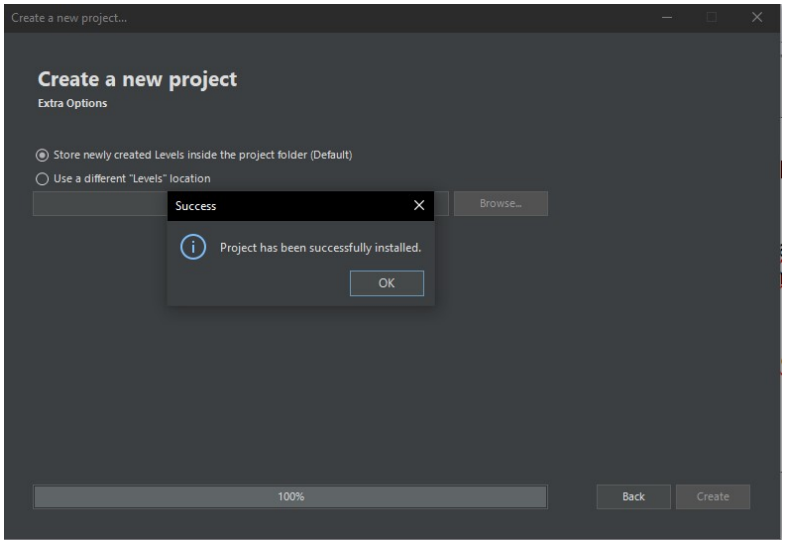

Figure 5.2: Extra Options

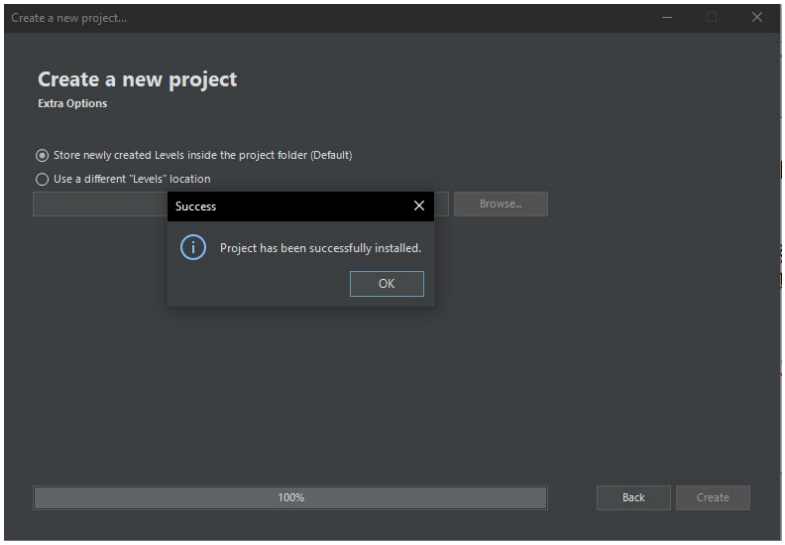

Figure 5.3: Project has been successfully installed

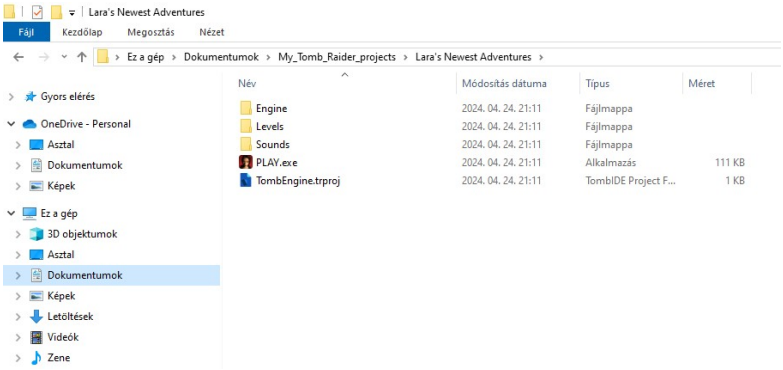

Figure 5.4: Project directories have been created (Since TEN 1.7., the project main folder has an Assets subfolder as well.)

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 subfolders 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. [8]

[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. [10]

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*. [9]

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

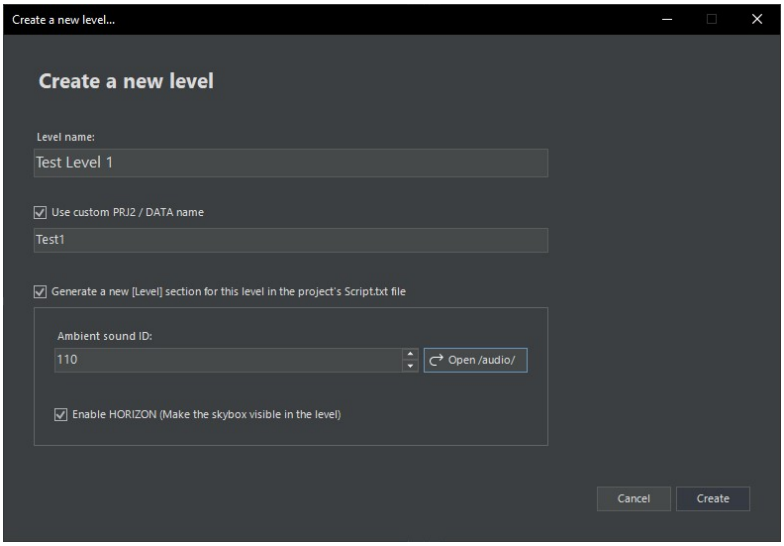

Figure 7.1: *Create a New Level* panel

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.

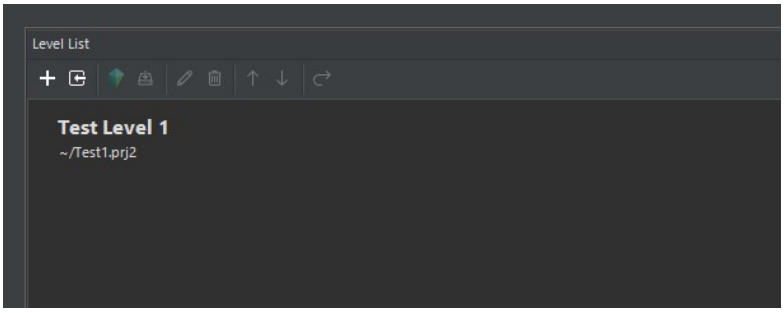

Figure 7.2: *Level list* panel showing level you've created

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 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. You can check this is the 3D view of a rectangular

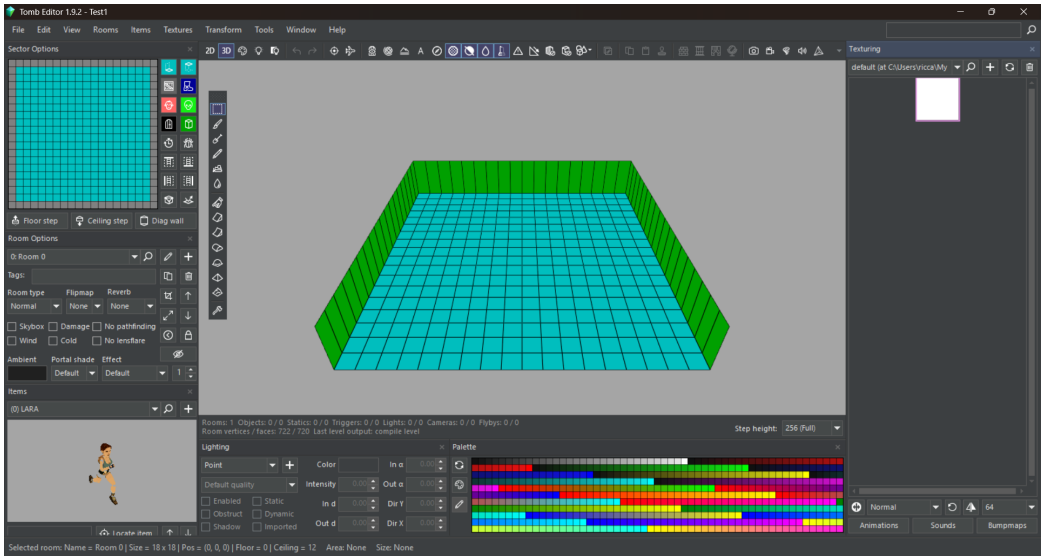

Figure 8.1: *Tomb Editor* main screen

cuboid, by left-clicking on the room and moving the mouse while keeping the left button of the mouse 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.

Hey! Do you remember we talked about the *crucial attachments*? 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

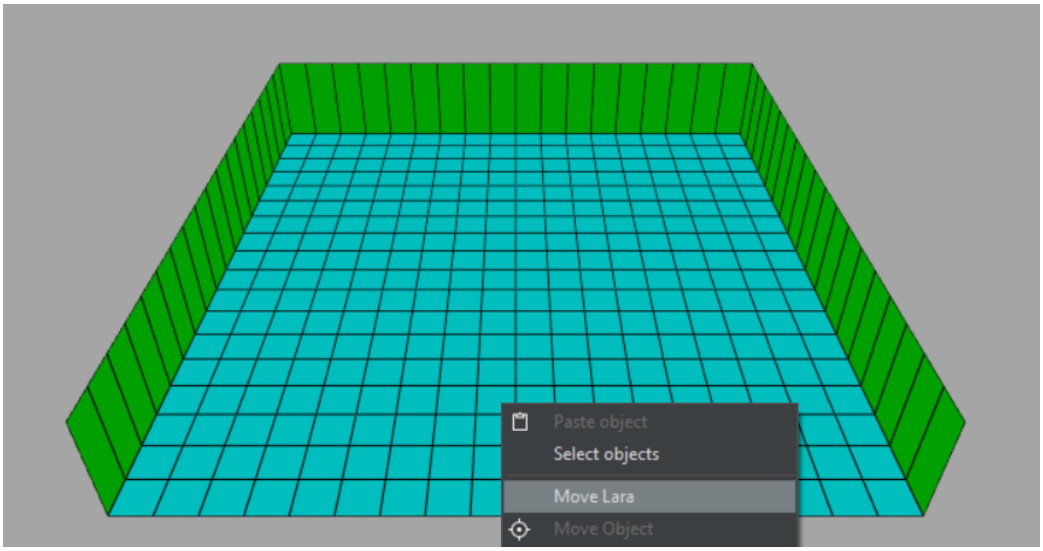

Figure 8.2: Placing Lara

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.

8.3 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 (8.2):

- left click on a floor box inside the room
- click on "Move Lara"

Now you can see Lara is placed in the room.

8.4 Ready to play your first room?

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.

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, 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.

* * *

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

Overview of *Tomb Editor* UI

The User Interface of Tomb Editor is composed by different panels or *windows* (9.1):

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

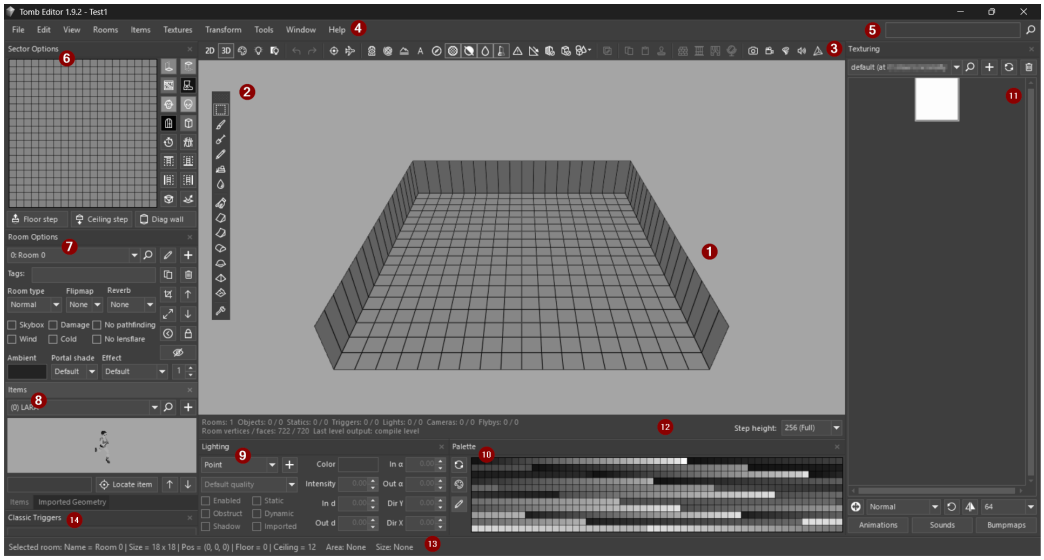

Figure 9.1: Placing Lara

11. **Texture Panel**: select tiles for texturing model; set animation ranges and texture sounds

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.

Chapter 10

Main contents of the 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?

10.1 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. [7]

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

10.2 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 (10.1):

- 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.

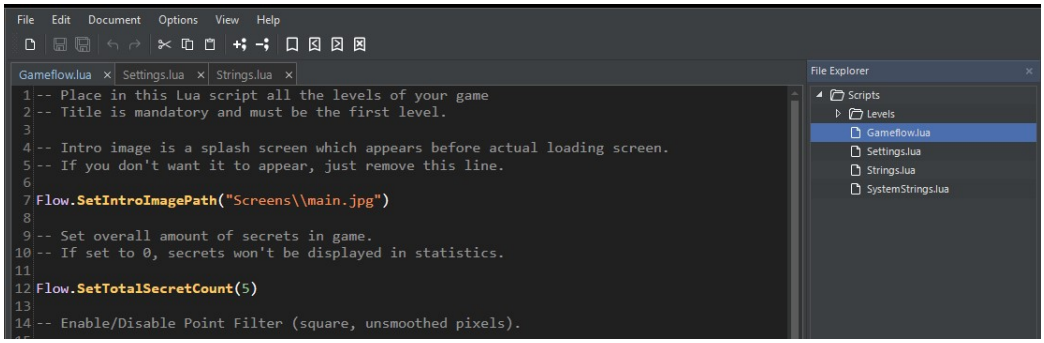

Figure 10.1: TombIDE Scripting Studio

10.3 Visual Studio Code

10.3.1 What is VS Code?

Visual Studio Code (VS Code) is an integrated development environment developed by Microsoft for Windows, Linux, macOS and web browsers. 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. [11]

10.3.2 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].

10.4 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).

[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.

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.

10.5 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.

10.6 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 (10.2). 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.

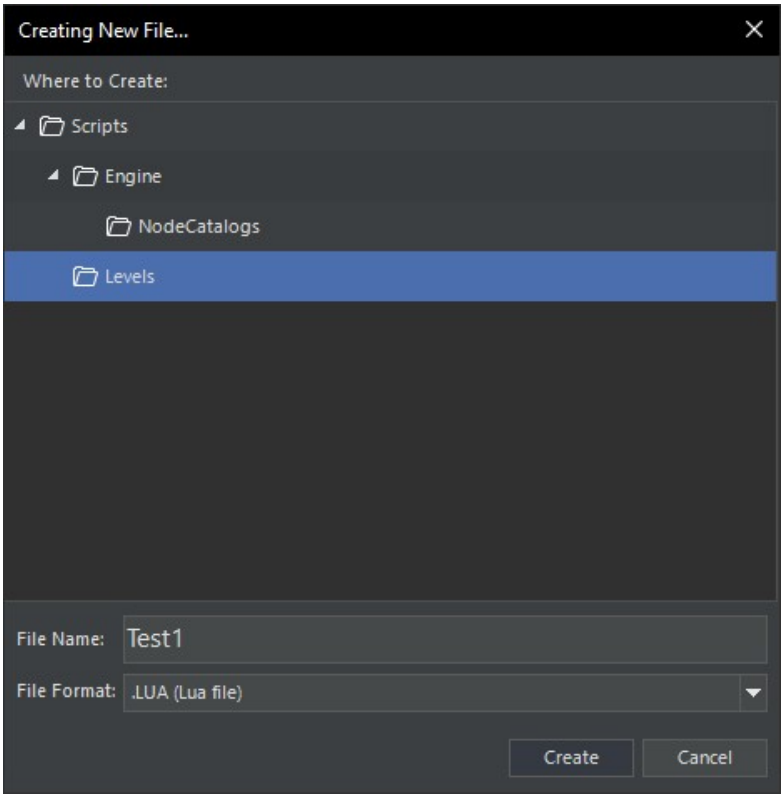

Figure 10.2: *Creating new file...* popup

Chapter 11

Main TE settings for the level

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 (11.1).

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 11.1: *Level Settings*

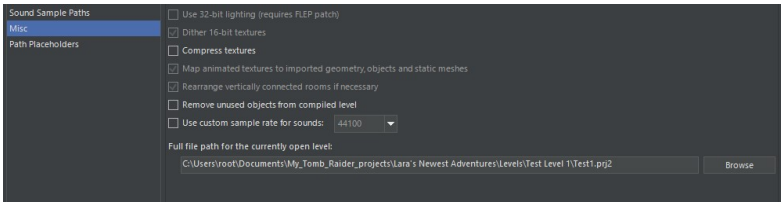

Figure 11.2: Level Settings *Misc* page

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 (11.2): **Full file path for the currently open level** value should be the route of this PRJ2.

Chapter 12

The WAD

Open Test1.prj2 in TE now, and check these.

12.1 WAD2 file

As I said above, in TE you can select more than one WAD2 file for a TEN level. - But, for a basic tutorial like this, only one WAD2 attached is enough to us to start editing the level. That is why we are examining now only one WAD2 attached to Test1.prj2.

Part III

Principles of game development

Chapter 13

Beta testing

13.1 In General

Beta testing is perhaps the most important phase of any video game's development. Whether you've spent a month or a few years on your adventure, if you don't have it beta tested properly, all that work could be ruined by one simple little gameplay killer. If you understand the importance of beta testing, then read this tutorial carefully and take it to heart. *If you don't understand the importance of beta testing, then read this tutorial carefully and take it to heart.* I cannot stress enough how important the beta testing phase is.

When should beta testing begin? The short answer is after alpha testing is finished. When is alpha testing finished? That's easy! That's when you think your game is ready for release. Your game is the best you can get it and you're sure there is no more you can do and you feel happy about releasing it. Now it's time for beta testing. This seems to be a difficult concept for new Level Builders (and a few seasoned Level Builders) to get their heads around. Trust me, your game does not go for beta testing until you are convinced it's finished. If it's not finished, you're still in the alpha testing stages and if you employ your beta testers now they will be wasting their time and your game will not be beta tested properly.

When a game moves from alpha testing to beta testing, the only changes you should make would be to fix bugs, gameplay errors and textural errors. You should never build new areas or new gameplay or new puzzles during or after beta testing. Again I stress, do not begin beta testing until your

game is FINISHED. Beta testing is purely to iron out problems in finished games. If you ignore this advice the chances are you will release a buggy game. Beta testers take their job seriously and if games are released with bugs it reflects on them and the job they did. If those bugs are down to the Level Builder changing things during or after beta testing and the game is released with bugs you will never get beta testers to help you again. Beta testers will invest days and even weeks of their personal time to help you with your game. Respect that and don't let them down.

13.2 Alpha Testing

Don't cut corners and skip this step. Yes, I know, you're excited about your game and you want to show it off and that's marvellous and wonderful and admirable and quite understandable, but your beta testers are not there to do your job for you. If you upload an appallingly buggy game that hasn't been properly alpha tested, don't be surprised if your beta testers don't do a good job and don't volunteer later when you actually need beta testers. Alpha testing is your job, not your beta testers, and it's extremely important. I consider my alpha testing to be complete when I have a package I think is ready for release. In other words, I can find nothing to change or fix and in my heart I know my game is the best I can make it and I feel it's ready for release. Now I begin beta testing.

During alpha testing, it is easy to move Lara around and forget to put her back, place temporary triggers to open a door and then forget to remove them and a few other little things, so after I've packaged up my game for beta testing guess what? Yes, I then test the beta package from start to finish using only the files I will be uploading for my beta testers. This is my final Alpha test. If this goes well and all the level jumps work properly and I successfully hit a finish trigger, only then is the game uploaded and the download link sent out to the beta testers.

[6]

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] *Definition of Next Generation Level Editor on WikiRaider*. https://www.wikiraider.com/index.php/Next_Generation_Level_Editor.

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

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

[6] George Paolone and AkyV. *The NGLE Manual*. https://www.treditor.hu/7/gmac/nglemanual.html.

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

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

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

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

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