

---

## Table of Contents

Reference I: RGB Color Reference

Glossary: Game Terms

Chapter 1: About Solarus, Basic History, Download Instructions, Shortcuts, and Documentation

About

    Basic Solarus History

    Download the Solarus ARPG Engine

    Shortcuts

Documentation Reading

    Normal Functions

    Method Functions

    Event Functions

Chapter 2: Free Graphics, Audio, Scripts, and Basic Free License Information

Free Graphics & Audio

    Sample Quest

    Children of Solarus

    Faryolica World

    OpenGameArt

Scripts

    Solarus Fan Games

    Solarus Forum Scripts

Solarus Help Guide

Basic Free License Information

Chapter 3: Moving Around The Solarus Editor

First Appearance

File

    File > Quest Properties

Edit Menu

View Menu

Run Menu

Audio Menu

Tools Menu

    Tools > Options

    Tools > Options > General Tab

    Tools > Options > Text Editor Tab

    Tools > Options > Map Editor Tab

    Tools > Options > Sprite Editor Tab

    Tools > Options > Tileset Editor Tab

Help Menu

Icon Shortcut Bar

Resource Manager List

    Resource Manager List > Tileset Editor

    Resource Manager List > Sprite Editor

    Resource Manager List > Sound Player

    Resource Manager List > Text Editor

[Resource Manager List > Music Player](#)

[Resource Manager List > Map Editor](#)

[Resource Manager List > Map Editor > Entities](#)

[Resource Manager List > Map Editor > Drag](#)

[Resource Manager List > Language > en > Dialog Editor](#)

## Chapter 4: Using The Sprite Editor

[Resource Manager > Sprites > Sprite Editor](#)

[Sprite Properties](#)

[Animation Properties](#)

[Direction Order](#)

[Direction Properties](#)

[Size](#)

[Position](#)

[Origin](#)

[Frames](#)

[Columns](#)

### Setting up a Sprite

[Making a Sprite](#)

[Name Sprite](#)

[Create Animation](#)

[Name Animation](#)

[Load Sprite Image](#)

[Making Directions - Drag Click Release](#)

[Set Frame Delay & Loop](#)

[Set Columns, Rows, & Origin](#)

[Show Origin](#)

[Zoom On Sprite Preview](#)

[Zoom On Sprite Animation Direction Setup Area](#)

[Previous & Next Sprite Frame](#)

[Last & First Frame + Play & Stop](#)

[Dat File Editing](#)

[Bounding box or Hitbox](#)

### Basic Sprite Information

[Sword](#)

[Shield](#)

[NPC](#)

[Enemy](#)

[Hero](#)

[Destructible](#)

[bomb](#)

[block](#)

[chest](#)

[explosion](#)

[shadow](#)

[crystal\\_block](#)

[crystal](#)

▪ [switch](#)

▪ [arrow](#)

▪ [star](#)

▪ [ground1](#)

▪ [ground2](#)

## o Chapter 5: Using The Tileset Editor

generated by haroopad

- Resource Manager > Tileset > Tileset Editor
  - Selection Properties
  - Rename ID
  - Ground
  - Repeatable
  - Animation
  - Making Tile Patterns
  - 8x8 Formatting
- Chapter 6: Very Basic Lua Scripting, Tutorial Point Lua PDF, ways to load script, and Lua console
  - Very Basic Lua Scripting
    - ZeroBrane IDE
    - Using ZeroBrane
    - Whitespace
    - Variables
    - Identifiers
    - Keywords
    - Data Types
    - Relational Operators
    - Logical Operators
    - Arithmetic Operators
    - Other Operators
    - Escape Sequences
    - Repetitions Pattern Modifiers
    - Character Pattern Classes
  - Common Variable Naming Rules
    - Clear Variable Name
    - Variable Length
    - UPPERCASE vs lowercase Variables
  - Lua Programming
    - Comments
    - Print Text
    - Declaring variables
    - Tables
    - Arrays
    - If Statement & Operators
    - Goto Statement
    - Loops Types
    - Math
    - Strings
    - tonumber()
    - string.format()
    - string.len() or :len()
    - string.reverse() or :reverse()
    - string.sub() or :sub()
    - string.gmatch(string, pattern) or string:gmatch(pattern)
    - Clear Table
    - Math/Arithmetic in an if statement
    - Simple table.concat()
    - table.concat() and table.insert()
    - table.sort
    - Defining a function

- pairs() and ipairs()
- Associative table
- Error Handling
- Declaring Multiple Variables On One Line
- Coroutine
  - Make a Coroutine
  - Coroutine Yield
  - Coroutine Resume
  - Tasks Between Resume
  - Coroutine Status
  - Coroutine Callback
  - Coroutine Wrap Shortcut
- Module
  - Make a table
  - Add Tasks
  - Require Module
  - Global Module Table Issue
  - Module Shortcut
  - Assign New Variable
  - Dot vs Colon
- Function Objects
  - Almost Method
  - Method
  - Method With Colon
  - Colon Hidden Parameter
- Tutorial Point Lua PDF
- Ways To Load Script In Solarus Part 1
  - ▪ Require()
  - ▪ Sol Main Load
- Grab Numbers & Letters
  - ▪ Grab Numbers
  - ▪ Grab Letter
- Chapter 7: Setting up Dialog and Pause
  - ▪ ▪ Setting Up The Dialog Script
  - ▪ Using the Dialog box
  - ▪ Mouse Control Fix
  - ▪ Passing a value and string into the dialog
  - ▪ Setting up yes\_no
  - ▪ Dialog lines
  - ▪ Adding on\_paused
- Chapter 8: Displaying an image, opacity, color fill, blend modes, and font display
  - Script For The Lesson
  - Breaking Down The Script
    - ▪ Explaining Surfaces
    - ▪ Explaining Draw Function
    - ▪ Drawing an image, font, and a fill\_color
    - ▪ Filling a Color
    - ▪ Opacity or Semi-transparency
    - ▪ on\_unpaused function & clear pixels
    - ▪ Blend Modes
    - ▪ Drawing to a Surface
  - Drawing a Sprite

- ▪ Sample
- Script
- Breaking down the Script
- Chapter 9: Key press, Mouse press, Image fade, and Playing Audio
  - Lesson Script
  - Breaking Down the Script
    - ▪ Key Pressed
    - Set Pause
    - Changing Volume
    - Playing Music & Sound
    - Making Your Own Function
    - Fade In and Out
    - Mouse Pressed
- Chapter 10: Timers and Getting Coordinates
  - Basic Timer Map Script Example
  - Timer
    - ▪ Timer Function Delay
    - Timer Anonymous Function Delay
    - Repeat Timer with True
    - Stop Repeating Timer With Variable
    - Repeat Timer a Number of Times
    - Context Timer
  - Timer Display Drag Click Script Example
    - Stopping a Timer
    - Stop all Context Timers
    - Getting Remaining Time
    - Displaying Timer
    - Timer Sound
    - Change Remaining Timer
    - Suspend Timer
  - Get Coordinates
    - Getting Cursor Location
    - Applying Coordinates To Images
    - Using Timer To Drag Image
- Chapter 11: Map editor
  - ▪ Making a Map
  - Map Properties
  - Map Script
  - Map Property Features
  - Map Drag
  - Map Zoom
  - Selecting Map Tiles
  - Dragging Map Tile
  - Bring to Back Map Tile
  - Change Map Tile Layer
  - Selecting Multiple Tiles
  - Dynamic Tiles
  - Resizing Tiles
  - Making Obstacles Vanish
  - Making Traversable Vanish
  - Making Layer[num] Vanish
- Chapter 12 Menus and Window Options

- Window Options
  - Video Mode
  - Fullscreen
  - Cursor Visibility
  - Window Size
  - Default Window
- Menu
- Menu function
- Menu: Starting and Stopping
  - Starting Menu
  - Stopping Menu
  - Menu Script Example
- Chapter 13: Entities
  - get\_() & related
    - map:get\_game()
    - map:get\_hero()
    - entity:get\_sprite()
  - Entity Position & Name
  - Types of entities
  - Destination Entity
    - Setting up Maps and Starting Location
    - Inside Store
    - Add Destination Entity
    - Edit Destination Entity Options
  - Teletransporter Entity
    - Add Teletransporter
    - Edit Teletransporter Entity Options
  - Pickable Entity
    - Add Pickable Entity
    - Edit Pickable Entity Options
    - Name Connection
    - Item Script
  - Destructible Entity
    - Inside Store to Map
    - Add Destructible Entity
    - Edit Destructible Entity Options
    - Destructible Entity Sprite
  - Chest Entity
    - Add a Chest Entity
    - Edit Chest Entity Options
  - Jumper Entity
    - Add Jumper Entity
    - Resize Jumper
    - Edit Jumper Entity Options
    - Jump Walls
    - Jump Preview
  - Enemy Entity
    - Inside Store
    - Enemy Breed Script
    - Enemy Script Generated
    - Enemy Four Way Direction
    - Enemy Animation

- Add Enemy
- Edit Enemy
- NPC Entity
  - Add NPC
  - Usual NPC
  - Set Up Dialog Box
  - Sample
  - Usual NPC Call Map Script
  - Showing NPC Script
  - Setting up a Simple NPC dialog
  - NPC Dialog Yes\_No
  - NPC Answer No
  - NPC Answer Yes
  - NPC Treasure
  - Prevent Repeating NPC Dialog With Boolean
  - Prevent Repeating NPC Dialog With Get Value
  - NPC Shortening the Script
  - NPC Wonderful Day
  - NPC Dialogs
  - Generalized NPC
  - Generalized NPC Desk Method
- Movable Block Entity
  - Add Block
  - Block Properties
- Switch Entity
  - Add Switch
  - Switch Properties
  - Walkable Switch Coding
  - Block Switch Coding
  - Solid Switch Coding
- Dynamic Tile
  - Convert to Dynamic
  - Convert to Static
  - Dynamic Properties
  - Dynamic Scripting
- Wall Entity
  - Add Wall Entity
  - Wall Entity Properties
  - Wall Entity Scripting
- Sensor Entity
  - Add Sensor Entity
  - Sensor Properties
  - Sensor Scripting
  - Sensor Get Position
  - Sensor Multiple Dynamic Tiles
- Crystal Switch Entity
  - Add Crystal Switch
  - Crystal Switch Properties
  - Default Crystal Switch Graphic Names
- Crystal Block Entities
  - Add Crystal Block
  - Crystal Block Properties

- Default Crystal Block Graphic Names
- Stream Entity
  - Add Stream
  - Stream Properties
- HUD
  - Setup HUD Scripts
  - Hud Configurations hud\_config.lua
  - Health Heart Display hearts.lua
  - Money System Rupee Style
- Money (Gem) Setup
- Shop Entity
  - Add Shop Entity
  - Shop Entity Properties
  - Shop Setup
  - Shop Entity Dialogs
- Door Entity
  - Add Door
  - Door Properties
  - Door Graphic Setup
  - Door Script
  - Opening Door With Key
  - Door Key
  - Door Opening By Hero
- Stairs Entity
  - Add Stairs
  - Stairs Properties
  - Stairs Setup
- Separator Entity
  - Add Separator
  - Separator Properties
  - Separator viewpoint
- Custom Entities
  - Add Custom Entity
  - Custom Entity Properties
  - Create Custom Entity Script
  - Trick Chest Entity Scripting
  - Custom Entity Switch
  - Custom Animation
  - Get Distance to Entity
  - Make Entity Turn to Entity
  - Obstacle Detection
  - On Hero State
- Hero Entity
  - Hero Teleport
  - Hero Get Animation
  - Hero Set Animation
  - Set Sword Sound
  - Hero Walk
  - Hero Jump
- Chapter 14: Abilities, Save Game, Quest Launcher, and Game Over
  - Abilities: Swim, Lift, Sword, Run, etc
    - Set Ability

- Get Ability
  - Has Ability
  - Tunic Ability Setup
  - Sword Ability Setup
  - Sword Knowledge
  - Shield Ability Setup
  - Lift Ability Setup
  - Water Jump Ability Setup
  - Run Ability Setup
  - Swim Ability Setup
  - Quest Launcher
    - Activate Quest Launcher
    - Play Game
    - Game - Add & Remove
    - Video > Fullscreen, 2D Acceleration, Window Size, & Zoom
    - Quest Launcher > File > Add, Remove, & Exit
    - Game - Play & Stop
    - Help > About
  - About Logo & Icons:
    - Quest Logo
    - Quest Icons
  - Save Game
  - Game Over
- Chapter 15: Title Screen, Save menu, Movements, Map Types, Camera, I/O
    - Title Screen
    - Movements
      - Create Movement
      - Jump Movement
      - Random Path Movement
      - Target Movement
      - Path Finding Movement
      - Random Movement
      - Straight Movement
      - Path Movement
      - Pixel Movement
      - Circle Movement
      - Walk through Entity
    - Moving Image
    - Map Types
      - Making a Map Entity
    - Camera Entity
      - Camera Size
      - Camera Tracking
      - Camera Position
      - Camera & Movements
      - Camera Example Script
    - I/O - Input/Output
      - Opening, writing, and closing file
      - Reading, searching, and outputting
      - Making New Lines
      - File Seek: Grab a Variable
      - Read & Write Functions Script



- Habits
- Main Fantasy Character Classes
  - Fighter Class
  - Magician Class
  - Rogue Class
  - Cleric Class
  - Ranger Class
  - Rarer Class
- Chapter 18: Upgrading, Export Project, and Making Libraries
  - ▪ ▪ Upgrading:
  - ▪ Export Project:
- Making Libraries
  - ▪ Sample
  - ▪ Documentation
  - ▪ Basic Markdown Formatting
- Types of Libraries
  - ▪ Entity Library Sample
  - ▪ Library Setup
  - ▪ Self Function
  - ▪ Special Entity
  - ▪ Function Arguments
  - ▪ Function Tables
  - ▪ Player 2 Movement
- Debugging
  - ▪ Debug Levels
  - ▪ Lua Debug Library function
  - ▪ Debug Traceback
  - ▪ Pcall
- Chapter 19: Credits
  - ▪ ▪ Graphics
  - ▪ Audio
  - ▪ Scripting Credits
  - ▪ Writing Credits
  - ▪ Solarus Team
  - ▪ Special Thanks
- Fairyolica World Credits
  - ▪ About:
- Credit
  - ▪ [CC0 Public Domain]
  - ▪ [CC-BY 3.0 Credit by their terms]
  - ▪ [CC-BY-SA 3.0 Credit by their terms and share artwork under the same terms]

## Reference I: RGB Color Reference

---

Source: [Rapidtables](#)

### RGB REFERENCE LIST:

Color/Colour	RGB

generated by haroopad

Color/Colour	RGB
maroon	(128,0,0)
dark red	(139,0,0)
brown	(165,42,42)
firebrick	(178,34,34)
crimson	(220,20,60)
red	(255,0,0)
tomato	(255,99,71)
coral	(255,127,80)
indian red	(205,92,92)
light coral	(240,128,128)
dark salmon	(233,150,122)
salmon	(250,128,114)
light salmon	(255,160,122)
orange red	(255,69,0)
dark orange	(255,140,0)
orange	(255,165,0)
gold	(255,215,0)
dark golden rod	(184,134,11)
golden rod	(218,165,32)
pale golden rod	(238,232,170)
dark khaki	(189,183,107)
khaki	(240,230,140)
olive	(128,128,0)
yellow	(255,255,0)
yellow green	(154,205,50)
dark olive green	(85,107,47)
olive drab	(107,142,35)
lawn green	(124,252,0)
chartreuse	(127,255,0)
green yellow	(173,255,47)
dark green	(0,100,0)
green	(0,128,0)
forest green	(34,139,34)
lime	(0,255,0)

Color/Colour	RGB
lime green	(50,205,50)
light green	(144,238,144)
pale green	(152,251,152)
dark sea green	(143,188,143)
medium spring green	(0,250,154)
spring green	(0,255,127)
sea green	(46,139,87)
medium aqua marine	(102,205,170)
medium sea green	(60,179,113)
light sea green	(32,178,170)
dark slate gray	(47,79,79)
teal	0,128,128)
dark cyan	(0,139,139)
aqua	(0,255,255)
cyan	(0,255,255)
light cyan	(224,255,255)
dark turquoise	(0,206,209)
turquoise	(64,224,208)
medium turquoise	(72,209,204)
pale turquoise	(175,238,238)
aqua marine	(127,255,212)
powder blue	(176,224,230)
cadet blue	(95,158,160)
steel blue	(70,130,180)
corn flower blue	(100,149,237)
deep sky blue	(0,191,255)
dodger blue	(30,144,255)
light blue	(173,216,230)
sky blue	(135,206,235)
light sky blue	(135,206,250)
midnight blue	(25,25,112)
navy	(0,0,128)
dark blue	(0,0,139)
medium blue	(0,0,205)

Color/Colour	RGB
blue	(0,0,255)
royal blue	(65,105,225)
blue violet	(138,43,226)
indigo	(75,0,130)
dark slate blue	(72,61,139)
slate blue	(106,90,205)
medium slate blue	(123,104,238)
medium purple	(147,112,219)
dark magenta	(139,0,139)
dark violet	(148,0,211)
dark orchid	(153,50,204)
medium orchid	(186,85,211)
purple	(128,0,128)
thistle	(216,191,216)
plum	(221,160,221)
violet	(238,130,238)
magenta / fuchsia	(255,0,255)
orchid	(218,112,214)
medium violet red	(199,21,133)
pale violet red	(219,112,147)
deep pink	(255,20,147)
hot pink	(255,105,180)
light pink	(255,182,193)
pink	(255,192,203)
antique white	(250,235,215)
beige	(245,245,220)
bisque	(255,228,196)
blanched almond	(255,235,205)
wheat	(245,222,179)
corn silk	(255,248,220)
lemon chiffon	(255,250,205)
light golden rod yellow	(250,250,210)
light yellow	(255,255,224)
saddle brown	(139,69,19)

Color/Colour	RGB
sienna	(160,82,45)
chocolate	(210,105,30)
peru	(205,133,63)
sandy brown	(244,164,96)
burly wood	(222,184,135)
tan	(210,180,140)
rosy brown	(188,143,143)
moccasin	(255,228,181)
navajo white	(255,222,173)
peach puff	(255,218,185)
misty rose	(255,228,225)
lavender blush	(255,240,245)
linen	(250,240,230)
old lace	(253,245,230)
papaya whip	(255,239,213)
sea shell	(255,245,238)
mint cream	(245,255,250)
slate gray	(112,128,144)
light slate gray	(119,136,153)
light steel blue	(176,196,222)
lavender	(230,230,250)
floral white	(255,250,240)
alice blue	(240,248,255)
ghost white	(248,248,255)
honeydew	(240,255,240)
ivory	(255,255,240)
azure	(240,255,255)
snow	(255,250,250)
black	(0,0,0)
dim gray / dim grey	(105,105,105)
gray / grey	(128,128,128)
dark gray / dark grey	(169,169,169)
silver	(192,192,192)
light gray / light grey	(211,211,211)

Color/Colour	RGB
gainsboro	(220,220,220)
white smoke	(245,245,245)
white	(255,255,255)

## Glossary: Game Terms

Glossary	Description
1-up	In games where players have a number of "lives" to complete a game or level, an object or the act of gaining an extra life. The term "1-UP" also commonly referred to Player number 1. Two player games scores were displayed as "1-UP" and "2-UP".
1v1	Abbreviation of 1 versus 1, which means two players battling against each other.
Console	A video game hardware unit that typically offers connects to a video screen and controllers, along with other hardware. Unlike personal computers, a console typically has a fixed hardware configuration defined by its manufacturer and cannot be customized.
8-bit	A descriptor for hardware or software that arose during the third generation of video game consoles, targeting 8-bit computer architecture.
16-bit	A descriptor for hardware or software that arose during the fourth generation of video game consoles, targeting 16-bit computer architecture.
32-bit	A descriptor for hardware or software that arose during the fifth generation of video game consoles, targeting 32-bit computer architecture.
2D graphics	The game features 2-dimensional objects.
2.5D graphics	A game consisting of 3D graphics set in a 2D plane of movement, where objects outside of this 2D plane can have an effect on the gameplay.
Mode 7	is a graphics mode on the Super NES video game console that allows a background layer to be rotated and scaled on a scanline-by-scanline basis to create many different effects. Thus, an impression of three-dimensional graphics is achieved.
Action role-playing game (ARPG)	A genre of role-playing video games where battle actions are performed in real-time instead of a turn-based mechanic.
ARPG	Abbreviation of Action role-playing game.
Alpha release	An initial, incomplete version of a game (compare 'beta version'); alpha versions are usually released early in the development process to test a game's most critical functionality and prototype design concepts.
Beta release	An early release of a video game, following its alpha release, where typically the game developer seeks to remove bugs prior to the released product through feedback from players and testers.
Closed beta	A beta period where only specific people have access to the game
Open beta	The opposite of a 'closed beta'; the players are not bound by non-disclosure agreements and are free to show the game to others.

generated by haroopad

Glossary	Description
Bonus stage	A level that is usually unlocked and not normally on the level choose screen until unlocked.
Campaign mode	A campaign mode, story mode, or simply campaign refers to one of several possible operating modes of a game in which levels are specifically encountered in a linear or branching fashion, often with more story elements present compared to other modes (such as a skirmish mode or sound test).
Challenge mode	A mode of gameplay offered beyond the game's normal play mode that tasks the player(s) to replay parts of the game or special levels under specific conditions that are not normally present or required in the main game, such as finishing a level within a specific time, or using only one type of weapon. If a game doesn't feature a 'challenge mode', players will often create self-imposed challenges by forbidding or restricting the use of certain game mechanics.
Character class	A character type with distinct abilities and attributes both positive and negative, such as a warrior, thief, wizard, or priest.
Saved game (game save, savegame, savefile)	A file or similar data storage method that stores the state of the game in non-temporary memory, enabling the player to shut down the gaming system and then later restart the system and load the game state from the game file to continuing playing where they left off. Saved games may also be used to store the game's state before a difficult area that, should the player-character die, the player can restart from that save point. Known a lot as a checkpoint.
Save point	A place in the game world of a video game where a game save can be made. Some games do not have specific save points, instead allowing the player to save at any point.
Chiptune	Music composed for the microchip-based audio hardware of early home computers and gaming consoles. Due to the technical limitations of earlier video game hardware, chiptune came to define a style of its own, known for its "soaring flutelike melodies, buzzing square wave bass, rapid arpeggios, and noisy gated percussion."
Combo	Combinations of attacks in a fighting game, during which an opponent is helpless to defend themselves. To correctly execute a combo, a player needs to learn complex series of joystick and button combinations.
Continue	A common term in video games for the option to continue the game after all of the player's lives have been lost, rather than ending the game and restarting from the very beginning. There may or may not be a penalty for doing this, such as losing a certain number of points or being unable to access bonus stages.
Controller	A means of control over the console or PC you are playing the game on.
PC(s)	Player Character(s)
Cutscene	A (usually) short video which provides detail and exposition to the story. These videos, usually in much higher graphical resolution and detail than the basic game, are used extensively in MMOs and RPGs to move the story forward.
D-pad	A 4-directional rocker button that allows the player to direct game action in eight different directions: up, down, left, right, and the diagonals involving these.
Damage over time (DoT)	An effect, such as poison or catching on fire, that reduces a player's health over the course of time or turns.

generated by haroopad

Glossary	Description
Damage per second (DPS)	Used as a metric in some games to allow the player to determine their offensive power.
Difficulty	The level of difficulty that a player wishes to face while playing a game; at higher difficulty levels, the player usually faces stronger NPCs, limited resources, or tighter time-limits.
Double jump	An additional jump that follows the first in quick succession
Downloadable content (DLC)	Additional content for a video game that is acquired through digital download and often requiring additional purchase. Increasingly replacing the traditional retail 'expansion pack'.
DPM	Abbreviation of "damage per minute", used as a metric in some games to allow the player to determine their offensive power.
DPS	Abbreviation of Damage per second
DRM	Abbreviation of Digital rights management
Emulator	A software program that is designed to replicate the software and hardware of a video game console on more modern computers and other devices. Emulators typically include the ability to load in software images of cartridges and other similar hardware-based game distribution methods from the earlier hardware generations, in addition to more traditional software images.
Experience point	In games that feature the ability for the player character(s) to gain levels such as CRPGs and JRPGs, experience points are used to denote progress towards the next level.
Farming	Repeating a battle, quest or a similar part of the game in order to receive either experience points, money, or specific items that can be gained through that battle or quest.
First-person shooter (FPS)	A genre of video games where the player experiences the game from the first person perspective, and where the primary mechanic is the use of guns and other ranged weapons to defeat enemies.
FPS	An abbreviation for either First-person shooter or Frames per second.
Frame rate	A measure of the rendering speed of a video game, typically in frames per second (FPS).
Fog of war	A 'fog' that covers unexplored areas of the map, hiding enemy units in that area. Common in strategy games.
Game over	The end of the game (failure screen).
Game port	When a game is ported from one platform to another; cross-platform ports are often criticized for their quality, particularly if platform-specific design elements (such as input methods) are not updated for the target platform.
God mode (infinite health/life, invincibility, invulnerability)	A cheat that makes player-characters invulnerable. Occasionally adds invincibility, where the player can hurt enemies by touching them (e.g., the Super Mario Super Star). The effect can also be temporary.
Graphic content filter	A setting that controls whether the game displays graphic violence.

Glossary	Description
Griefer	A player in a multiplayer video game who deliberately irritates and harasses other players within the game. Many online multiplayer games enforce rules that forbid griefing.
Health	The remaining amount of metered damage that a character or player can take before dying or losing a life.
Indie game (independent video game)	Loosely defined as a game made by a person or studio without any financial, development, marketing, or distribution support from a large publisher, though there are indie games that are exceptions to this definition.
Inventory	A menu or area of the screen where items collected by the player during the game are stored. This interface allows the player to retrieve any item to use it as an instant effect, or to equip the player character with the item.
Joystick	Note: Do not confuse this with "an Analog stick." An input device consisting of a stick that pivots on a base and reports its angle or direction to the device it is controlling, such as a plane.
JRPG	Abbreviation of Japanese role-playing game, typically referring to a subgenre of RPGs that originated from Japan.
Jump	A basic move where the player jumps vertically upon pressing the action button.
Kill screen	A stage or level in a video game (often an arcade game) that stops the player's progress due to a software bug. <sup>[26]</sup> Kill screens can result in unpredictable gameplay and bizarre glitches.
Lag	The delay between an action and its corresponding result, most commonly in an online environment. Often the result of delayed network traffic.
Let's Play	A type of video game run-through done by players, through screenshots or video, where the player provides commentary about the game as they work through it.
Level	A stage in a game. Level may also represent experience levels or difficulty levels, depending on context.
Level editor	A program, either provided within the game software or as an additional program, that allows players to create new levels for video games.
Life	In video games, if a player character loses all of its health points, it loses a life. Losing all of one's lives is usually a loss condition and may force the player to start over. It is common in action games for the player character to have multiple lives and chances to earn more during the game. This way, a player can recover from making a disastrous mistake.
Loot system	Methods used in multiplayer games to distribute treasure among cooperating players for finishing a quest. While early MMOs distributed loot on a 'first come, first served' basis, it was quickly discovered that such a system was easily abused, and later games instead used a 'need-or-greed' system, in which the participating players roll virtual 'dice' and the loot is distributed according to the results.
Magic	Any of a variety of systems in games to render fantastical or otherwise unnatural effects utilizing a game mechanic, either accessories (scrolls, potions, artifacts) or a pool of resources inherent to the character ("mana", magic points, etc).
Main	A term meaning to focus on playing a certain character, sometimes exclusively.

Glossary	Description
Main Quest	In games with multiple quests, the 'main quest' is a chain of quests that comprise the game's storyline and must be completed to finish the game. The opposite is side quest which still offer rewards but don't advance the main quest.
Massively multiplayer online game (MMO)	A game that involves a large community of players co-existing in an online world, in cooperation or competition with one another.
Massively multiplayer online role-playing game (MMORPG)	An MMO that utilizes traditional role-playing game mechanics into its gameplay. Classic games such as EverQuest and Dark Age of Camelot were progenitors of the genre. The most popular and most well-known game of this type is Blizzard Entertainment's World of Warcraft.
MMO	Abbreviation of Massively multiplayer online game
MMORPG	Abbreviation of Massively multiplayer online role-playing game
MMR	Abbreviation of Matchmaking rating
Mob	Mob is a term for an in-game enemy who roams a specific area. It is an abbreviation of "mobile", and came into prevalence with the explosion of MMOs and the greater computing power available to these games. In older games, enemies you encountered were stationary, only occupying a specific location within the game and were therefore not "mobs".
MOBA	Abbreviation of Multiplayer online battle arena
Mod	A third-party addition or alteration to a game. Mods may take the form of new character skins, altered gameplay mechanics or the creation of a new story or an entirely new game-world. Some games (such as Fallout 4 and Skyrim) provide tools to create game mods, while other games that don't officially support game modifications can be altered or extended with the use of third-party tools.
Minigame	A 'game-within-a-game', often provided as a diversion from the game's plot. Minigames are usually one-screen affairs with limited replay value, though some games have provided an entire commercial release as a 'mini-game' within the primary game-world.
Multiplatform	A game which can be played on multiple platforms. Also called cross-platform
Multiplayer	A game that allows multiple players to play at once.
MP	Abbreviation of 'Magic Points'. Abbreviation of Multiplayer.
Nerf	A change intended to weaken a particular item, tactic, ability, or character, ostensibly for balancing purposes.
New Game Plus	A mode typically offered in games with campaign modes that allows the player to replay the campaign and carry over characters, attributes, or equipment from a prior run-through.
Newbie	Someone new to the game, generally used as a pejorative, although usually light-heartedly. See also, "Noob" (below).
Noob	A pejorative used to insult a player who is making mistakes that an experienced player would be expected to avoid; by far the most common insult in any gaming community.
Non-player character (NPC)	A computer controlled character, or any character that is not under the player's direct control.

Glossary	Description
NPC	Abbreviation of Non-player character
Patch	The process by which a developer of a video game creates an update to an already released game with the intention of possibly adding new content, fixing any bugs/glitches currently residing in the game, balancing character issues (especially prevalent in online multi-player games with competitive focuses), or updating the game to be compatible with DLC releases.
Physical release	A version of a video game released on an optical disc or other storage device, as opposed to a digital download.
Player-character	A Player Character, or PC, is the main protagonist controlled and played by the human player in a video game. Tidus from Final Fantasy X, Doomguy from the Doom series, and Commander Shepard from the Mass Effect series are all “player characters” developed by their game studios. Your characters you create in MMOs and MMORPGs are also “player characters”.
Player versus environment (PvE)	Refers to fighting computer-controlled enemies (non-playing characters), as opposed to PvP (player versus player).
Player versus player (PvP)	Refers to competing against other players, as opposed to PvE (player versus environment).
PvE	Abbreviation of Player versus Environment
PvP	Abbreviation of Player versus Player
Playthrough	The act of playing a game from start to finish, in one or several sessions.
Power-up	Objects that instantly benefit or add extra abilities to the game character, usually as a temporary effect. Persistent power-ups are called perks.
Quest	A “quest” is any objective-based activity created in-game for the purpose of either story or character level advancement. Quests follow many common types, such as “Kill X number of Y monster”, “Gather X number of Y item”, or “Escort this person from point A to point B and keep them safe”. Some quests involve more detailed information and mechanics and are either greatly enjoyed by players as a break from the above common monotony, or are reviled as uselessly more complicated than necessary to the game.
Quicksave/Quickload	A mechanism in a video game where progress to or from a saved game can be done by pressing a single controller button or keystroke, instead of opening a file dialog to locate the save file. Typically, quicksaving will overwrite any previous saved state.
Rage Game	A video game which is designed to be extremely difficult and frustrating, with elements that intentionally try to ‘cheat’ in some way or form, with the intent of causing a player to become extremely angry and rage quit.
Rage Quit	Quitting a game in an act of anger; rage-quitting in an online game is widely considered poor sportsmanship.
Real-time strategy (RTS)	A genre of video games where the player controls one or more units in real-time combat with human or computer opponents.
Retrogaming	The playing or collecting of older personal computer, console, and arcade video games in contemporary times.
Roguelike	A genre of video games featuring procedurally-generated level generation and permanent death.
RPG	Abbreviation of Role-playing video game

Glossary	Description
Role-playing video game (RPG)	<p>An RPG is a game where the human player takes on the role of a specific character “class” and advances the skills and abilities of that character within the game environment. RPG characters generally have a wide variety of skills and abilities available to them, and much theorycrafting (the art of developing a specific character type to its highest in-game potential) is involved in creating the best possible form of each of these character classes.</p> <p>This is different from games such as First Person Shooters (FPS), wherein the “player character” in those games are all standardized forms and the physical skills of the player involved are the determining factor in their success or failure within the game. In an RPG, a human player can be the best player in the world at the game, but if they are using a character build that is substandard, they can be significantly outplayed by a lesser player running a more optimal character build.</p>
Rush (or Zerg rush)	<p>A tactic in strategy games where the player sacrifices economic development in favour of using many low-cost fast/weak units to rush and overwhelm an enemy by attrition or sheer numbers.</p>
Sandbox game	<p>A game wherein the player has been freed from the traditional structure and direction typically found in video games, and is instead given the ability to choose what, when, and how they want to approach the available choices in content. The term is in reference to a child’s sandbox in which no rules are present and play is derived from open-ended choice. While some sandbox games may have building and creation aspects to their gameplay, those activities are not required. Sandbox games usually take place in an open-world setting as to facilitate the freedom of choice a player is given.</p>
Side quest	<p>A quest that is optional, completing these won’t advance the main story line (Main quest).</p>
Simulation game (Sim)	<p>A video game that simulates some aspect of reality, though the degree of realism may vary. They are usually open-ended and have no intrinsic goals to be met. Inclusive definitions allow for any video game that models reality, such as sports games, while exclusive definitions generally focus on city-building games, vehicle simulation games, or both.</p>
Single-player	<p>A game that can only have one player at a time. Contrasted with multiplayer.</p>
Skill tree	<p>A character development gaming mechanic typically seen in role-playing games. A skill tree consists of a series of skills (sometimes known as perks or by other names) which can be earned by the player as he or she levels up or otherwise progresses his or her player character. These skills grant gameplay benefits to the player; for example, giving the character the ability to perform a new action, or giving a boost to one of the character’s stats. A skill tree is called a “tree” because it uses a tiered system and typically branches out into multiple paths.</p>
Split-screen multiplayer	<p>A game that presents 2 or more views seen by different players in a multiplayer game on the same display unit.</p>
Stat point	<p>A discrete amount of points for the player to distribute among their character’s attributes, e.g., to choose their player’s trade-offs between strength, charisma, and stamina.</p>

Glossary	Description
Status effect	This is an overarching term that covers both “buffs” and “debuffs”. Essentially, any effect to a character that is outside of the normal baseline is a “status effect”. Common negative status effects are poisoning (damage over time), petrification/paralysis (inability to move), or armor/damage reduction (lowering of defensive/offensive abilities). Common positive status effects include a heal-over-time (a small, pulsing heal that triggers multiple times over a set period), armor/damage increases, or speed increases.
Title screen	The initial screen of a computer, video, or arcade game after the credits and logos of the game developer and the publisher are displayed. Earlier title screens often included all the game options available (single player, multiplayer, configuration of controls, etc.) while modern games have opted for the title screen to serve as a splash screen. This can be attributed to the use of the title screen as a loading screen, in which to cache all the graphical elements of the main menu. Older computer and video games had relatively simple menu screens, that often featured pre-rendered artwork.
Touchscreen	When the screen of the console can be touched and get a response.
Turn-based game	When a game consists of multiple turns. When one player’s turn is up, they must wait until everyone else has finished their turn.
Upgrade	A way to make the given item, character, etc. more powerful.
World	A series of levels that share a similar environment or theme. A boss fight will typically happen once all or most of these levels are completed rather than after each individual level.
XP	Abbreviation of Experience Point
Zero-day patch	A software patch that is set to be released on the day of the game’s official release (“the 0th day”), reflecting updates and fixes that were added after the final release candidate was prepared.
Sprite (computer graphics)	In computer graphics, a sprite is a two-dimensional bitmap that is integrated into a larger scene.
Pixel	In digital imaging, a pixel, pel, dots, or picture element is a physical point in a raster image, or the smallest addressable element in an all points addressable display device; so it is the smallest controllable element of a picture represented on the screen.
Source:	<a href="https://en.wikipedia.org/wiki/Glossary_of_video_game_terms#Retrogaming">https://en.wikipedia.org/wiki/Glossary_of_video_game_terms#Retrogaming</a>
Source:	<a href="https://en.wikipedia.org/wiki/Mode_7">https://en.wikipedia.org/wiki/Mode_7</a>
Source:	<a href="https://en.wikipedia.org/wiki/Sprite_%28computer_graphics%29">https://en.wikipedia.org/wiki/Sprite_%28computer_graphics%29</a>
Source:	<a href="https://en.wikipedia.org/wiki/Pixel">https://en.wikipedia.org/wiki/Pixel</a>
Sound Novel	A Sound Novel is a type of visual novel. The term is a trademark by Chunsoft and emphasizes the sound aspects as opposed to visual aspects. Sound novels are actually older than visual novels but because of the trademark the term “visual novel” is the one that’s stuck in the long term. For older games the distinction between the two is somewhat notable, but for most modern games there’s basically no difference between the meaning of the two terms.

Glossary	Description
Kinetic Novel	A Kinetic Novel is a type of visual novel without any gameplay at all. That includes things like making decisions for the protagonist. Because of the lack of any player input, the story is entirely preset and the player just reads through it without any input. These are quite close to ordinary novels with added graphics, sound, and typically more focus on dialogue. Planetarian would be an example of such a game. Kinetic novels are usually shorter than other visual novels, though there are exceptions to this like Higurashi.
Dating Sims	Dating Sims are a different category of game from Visual Novels. These games do feature gameplay, but the object of the gameplay is to get into a romantic relationship with a character from the game. The most familiar example is probably the Tokimeki Memorial series, though there are many other highly successful examples such as Love Plus. Unlike visual novels, the gameplay here can be fairly complex.
Visual novel	<p>Visual Novel (ビジュアルノベル), often shortened to VN, is a general type of game with a lot of dialogue and minimal gameplay (usually the gameplay is reduced to just making choices at a few plot points to determine what route one enters). It may or may not involve any romance or sexual encounters. For example, Danganronpa could qualify as a visual novel, but probably not for any of the other categories here. A more canonical example would be Higurashi no Naku Koro ni, though purists will sometimes insist on calling it a "Sound Novel" as this is the official description. More on this in the related terms section. Calling something a visual novel emphasizes the "novel" aspect and suggests that there is at least some semblance of a story. It would not usually be used to describe eroge which are solely sex scenes.</p> <p>Visual novel can also be used in a more technical way to describe games where the text is overlayed over the background as opposed to being presented in dialogue boxes. This distinction is more common among Japanese speakers than English speakers. In English usually people will abbreviate this as NVL, and games where the dialogue is in a box at the bottom of the screen are called ADV. Fate/Stay Night is an example of this style.</p>
Eroge	Eroge (エロゲ) is a Japanese shortening of "erotic game". These can also be called H-games. Native Japanese speakers sometimes use the term to describe games without any sexual content (e.g. Clannad), but in English this isn't really a correct use of the term. Eroge can be any game with sex scenes (also called H-scenes). This includes some games that are not traditionally included as visual novels or dating sims. For example, Kamidori Alchemy Meister is an example of a game without a great deal of visual novel content but which still qualifies as an eroge.
Logan M:	<a href="http://anime.stackexchange.com/questions/4926/what-are-the-differences-between-visual-novel-eroge-gal-game-and-a-dating-sim">http://anime.stackexchange.com/questions/4926/what-are-the-differences-between-visual-novel-eroge-gal-game-and-a-dating-sim</a> (CC-BY-SA 3.0)
Artificial intelligence (AI)	Artificial intelligence is used to generate intelligent behaviors primarily in non-player characters (NPCs), often simulating human-like intelligence. The techniques used typically draw upon existing methods from the field of artificial intelligence (AI).
Beat 'em up	Beat 'em up (also known as brawler) is a video game genre featuring hand-to-hand combat between the protagonist and an improbably large number of opponents. These games typically take place in urban settings and feature crime-fighting and revenge-based plots, though some games may employ historical, sci-fi or fantasy themes. Traditional beat 'em ups take place in scrolling, two-dimensional (2D) levels

Glossary	Description
Central processing unit (CPU)	A central processing unit (CPU) is the electronic circuitry within a computer that carries out the instructions of a computer program by performing the basic arithmetic, logical, control and input/output (I/O) operations specified by the instructions.
Camera:	The camera is what follows the character around. It causes the screen to scroll with the player character.
Cheats Codes:	Special codes that allow you bypass the normal limitations of a game.
Spawning	Spawning is the live creation of a character, item or mob. Respawning is the recreation of an entity after its death or destruction, perhaps after losing one of its lives.
Shoulder buttons: (Triggers)	Shoulder buttons are found on the back of modern controllers.
Source:	<a href="https://en.wikipedia.org/wiki/Artificial_intelligence_(video_games)">https://en.wikipedia.org/wiki/Artificial_intelligence_(video_games)</a>
Source:	<a href="https://en.wikipedia.org/wiki/Beat_%27em_up">https://en.wikipedia.org/wiki/Beat_%27em_up</a>
Source:	<a href="https://en.wikipedia.org/wiki/Central_processing_unit">https://en.wikipedia.org/wiki/Central_processing_unit</a>
Source:	<a href="https://en.wikipedia.org/wiki/Spawning_%28video_gaming%29">https://en.wikipedia.org/wiki/Spawning_%28video_gaming%29</a>
Chipset	Commonly known now as tileset. This word is kinda old school now. This is the file that holds most graphics. Normally, it does not contain character graphics.
Charset	Short for character sets.Charsets hold character graphics.
Layer	Think of a layer like a sandwich. Cheese layer goes on the meat layer. Layers are basically images layered on top of each other. You can normally change the layers in some way.
Entity	An entity is something that exists separately from other things and has a clear identity or purpose. For example, a door entity would be used for only door like functions.
Map	A map in game development is the place where you put together the tiles from the tileset. Your map is the world you create with the tile graphics.
Cursor	The cursor is the black arrow (normally a black arrow) that you move around on the screen.

## Chapter 1: About Solarus, Basic History, Download Instructions, Shortcuts, and Documentation

---

### About

Solarus is a free GPLv3 2D ARPG game engine. ARPG stands for action role playing game. That means it for making games like Secret of Mana or Zelda, but you can do anything in Solarus if you code it. Visual novels, sidescrollers, and all those good game types are possible as well. There is already code in the community for most of it. The coding language used for making games is Lua and it is a super easy programming language. The easiest part is that Solarus's creator, "Christopho", has put together many functions to make tasks even easier.

## Basic Solarus History

Solarus began as a Zelda-like RPG Maker 2000 game. Due to limitations in RPG Maker 2000, the creator Christopho created a Java Engine called Solarus. The Solarus engine was named after the game made with RPG maker. The engine was scripted in Java, but later rewritten in C++ for speed.

Solarus has advanced greatly since being rewritten in C++, but that is greatly due to Christopho and his team. Their hard efforts keep the Solarus Community growing. The future of Solarus is getting brighter by each passing day.

## Download the Solarus ARPG Engine

Go to this URL to download Solarus.

<http://www.solarus-games.org/engine/download/>

Scroll down to the bottom and choose the version that fits your OS. Ubuntu, Archlinux, Gentoo, OpenBSD / FreeBSD, OpenSUSE, Mac OS X, Microsoft Windows, and ReactOS Windows.

 <b>Windows</b> Engine, editor and sample quest for Windows XP/Vista/7/8/10 <a href="#">Download v1.5.2 ↗</a>	 <b>Mac OS X</b> Engine, editor and sample quest for Mac OS X 10.6 or greater. <a href="#">Download v1.5.0 ↗</a>	 <b>Ubuntu</b> <code>sudo apt-add-repository ppa:nate-dev/solarus</code> <code>sudo apt-get update</code> <code>sudo apt-get install solarus solarus-quest-editor</code>	
 <b>Archlinux</b> Archlinux package of the engine <a href="#">Download ↗</a>	 <b>Gentoo</b> Available in "Armageddon" overlay under the name "solarus". layman -a Armageddon	 <b>OpenBSD / FreeBSD</b> On OpenBSD and FreeBSD, the Solarus engine is available from the ports collection: games/solarus.	 <b>OpenSUSE</b> OpenSUSE package of the Solarus engine <a href="#">Download ↗</a>

## Shortcuts

Task	Shortcut
New project	CTRL + N
Load project	CTRL + L
Close tab	CTRL + F4
Close all tabs	CTRL + W
Save project	CTRL + S
Save all	CTRL + SHIFT + S
Quest properties	CTRL + P
Undo	CTRL + Z
Redo	CTRL + Y
Cut	CTRL + X
Copy	CTRL + C

<b>Task</b>	<b>Shortcut</b>
Paste	CTRL + V
Select all	CTRL + A
Find	CTRL + F
Show grid	CTRL + G
Show console	F12
Run quest	F3
Link to Documentation	F1
Edit tile details	Enter or return key
Resize tile	R
Tile up one layer	SHIFT +
Tile down one layer	minus key (-)
Bring tile to back	T
Bring tile to front	B
Delete	DEL
Rename	F2
Show/hide Tile	CTRL + E, CTRL + 1
Show/hide Destination	CTRL + E, CTRL + I
Show/hide Teletransporter	CTRL + E, CTRL + T
Show/hide Pickable Treasure	CTRL + E, CTRL + P
Show/hide Destructible Object	CTRL + E, CTRL + D
Show/hide Chest	CTRL + E, CTRL + C
Show/hide Jumper	CTRL + E, CTRL + J
Show/hide Enemy	CTRL + E, CTRL + E
Show/hide NPC	CTRL + E, CTRL + N
Show/hide Block	CTRL + E, CTRL + B
Show/hide Dynamic Tile	CTRL + E, CTRL + 2
Show/hide Switch	CTRL + E, CTRL + H
Show/hide Wall	CTRL + E, CTRL + W
Show/hide Sensor	CTRL + E, CTRL + S
Show/hide Crystal	CTRL + E, CTRL + L
Show/hide Crystal Block	CTRL + E, CTRL + K
Show/hide Treasure	CTRL + E, CTRL + U
Show/hide Stream	CTRL + E, CTRL + M
Show/hide Door	CTRL + E, CTRL + O

Task	Shortcut
Show/hide Stairs	CTRL + E, CTRL + R
Show/hide Separator	CTRL + E, CTRL + A
Show/hide Custom Entity	CTRL + E, CTRL + Y

## Documentation Reading

This section is an attempt to help basic understanding of the documentation.

You will be looking at the documentation a lot.

You can download at the [Solarus website](#). I will provide a PDF version in this Github too.

### Normal Functions

Normal functions are related to `sol`. Most of the time, a variable can be assigned to the `sol` functions.

#### Example:

```
local map_metatable = sol.main.get_metatable("map")
```

### Method Functions

Method functions are attachment functions. They attach to an entity name or other functions. For example, the variable name assigned, like the movement example below.

#### Movement:

```
local move_straight = sol.movement.create("straight")
move_straight:set_ignore_obstacles(true)
```

#### Hero:

Hero methods would logically use the name "hero". If no name can be given to the entity, then most likely you will just get and use the variable name you want.

```
local game = map:get_game()
local hero = map:get_hero()

hero:freeze()
```

#### Camera:

Camera methods would logically use the name "camera". If no name can be given to the entity, then most likely you will just get and use the variable name you want.

```
local camera = map:get_camera()
camera:start_tracking(entity)
```

#### Enemy:

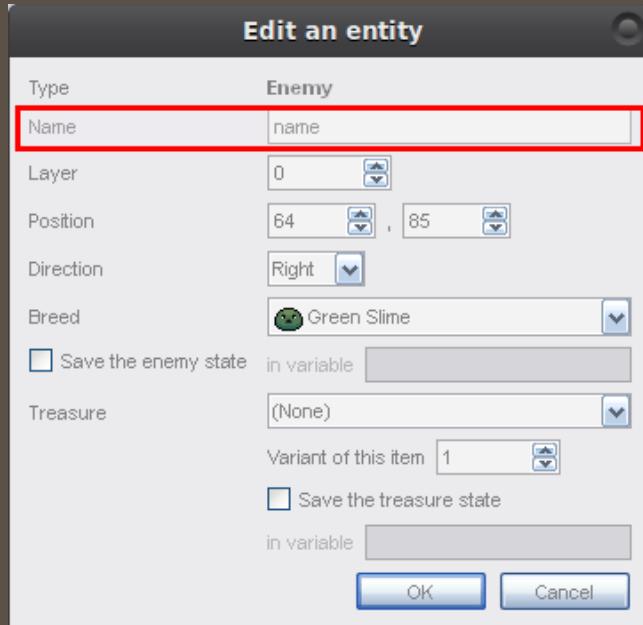
Enemy functions use the name "enemy", but one can get the name of the entity from the map and use it.

```
local map = enemy:get_map()
enemy:get_life()
```

or

```
local map = enemy:get_map()
local name = map:get_entity("name")

name:get_life()
```



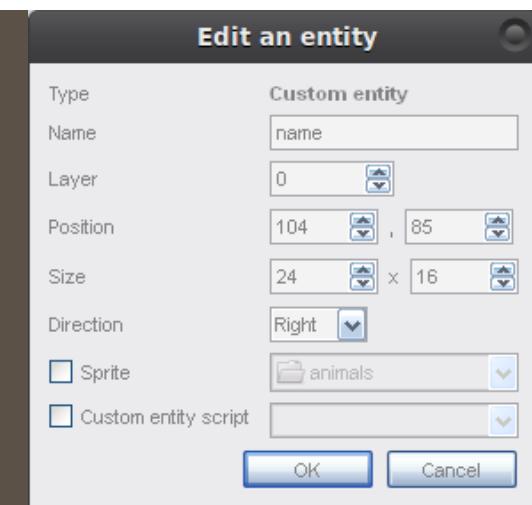
### Custom Entity:

Custom entity functions use the name "entity", but one can get the name of the entity from the map and use it.

```
local game = entity:get_game()
local map = entity:get_map()
entity:remove()
```

```
local name = map:get_entity("name")

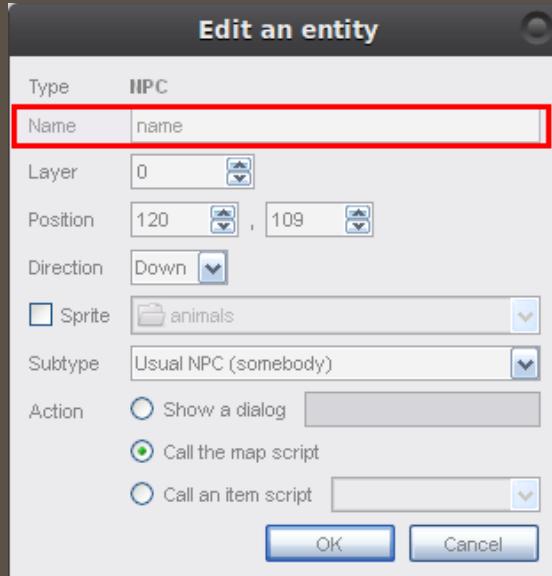
name:remove()
```



### Other Entties:

Entities like NPC will just use the name you give it.

```
name:remove()
```



### Event Functions

Event functions are setup up by putting the name `function` in front of them and having an `end` after it.

The function event `custom_entity:on_update()` would be setup like this:

```
function entity:on_update()
    --code here
end
```

## Chapter 2: Free Graphics, Audio, Scripts, and Basic Free License Information

# Free Graphics & Audio

## Sample Quest

Inside the Solarus Engine [Download](#) contains many free graphics, scripts, and audio in the sample quest. You can thank Diarandor for that because he is constantly adding resources to the sample quest.

<https://gitlab.com/solarus-games/solarus-sample-quest>

<https://gitlab.com/solarus-games/diarandor-art>

## Children of Solarus

This opensource project contains opensource graphics.

<https://gitlab.com/solarus-games/children-of-solarus>

## Faryolica World

This is an outside world tileset I put together with opensource art. I did make a few things, but that does not really matter.

<https://github.com/Zefk/Fairyolica-World>

<https://github.com/Zefk/Fairyolica-World/releases>

## OpenGameArt

OpenGameArt has a ton of free graphics and sounds.

<https://opengameart.org/>

# Scripts

## Solarus Fan Games

You can get a lot of free enemy scripts and more.

<http://www.solarus-games.org/games/solarus-team/>

## Solarus Forum Scripts

[Script Section](#)

[Script Post](#)

# Solarus Help Guide

The Solarus Help Guide is just a big chunk of useful information that I have gathered. It is on a post in the Solarus forum.

<http://forum.solarus-games.org/index.php/topic,611.0.html>

# Basic Free License Information

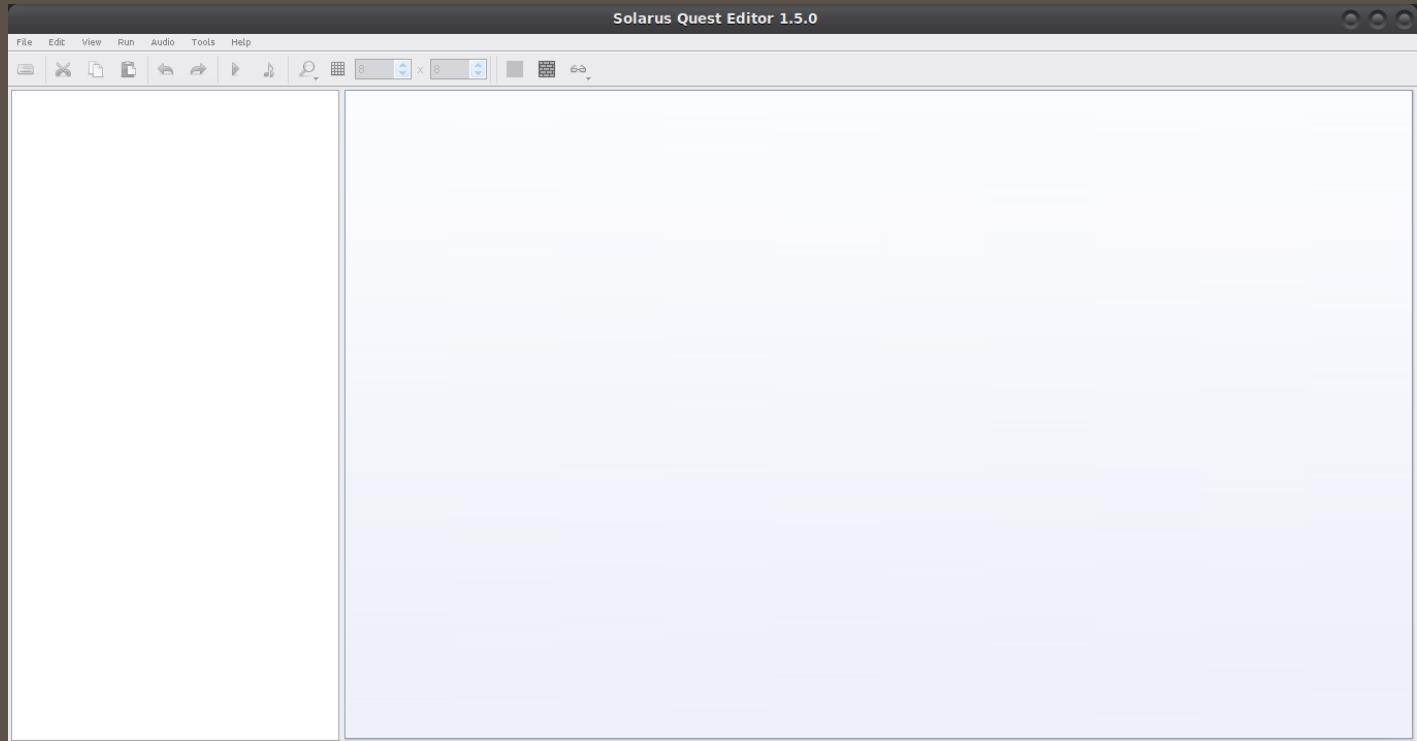
Refer to [this link](#) for the most up to date information. I will provide a pdf in this GitHub as well.

# Chapter 3: Moving Around The Solarus Editor

generated by haroopad

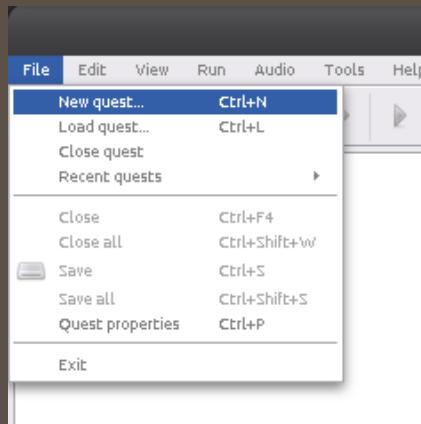
## First Appearance

The Solarus editor looks like this when one first opens it. This is before any quests are loaded or made.



## File

You will notice **file** at the upper left corner of the Solarus Editor. This is where one would make a new quest, load a quest, close a quest, check recently loaded quests, close tabs, save game, and check the quest properties.



### File > Quest Properties

The quest properties is about the quest information. The name of the save directory, quest title, summary, description, author, quest version, release date, website, and quest size.

**Quest properties**

**Quest information**

Solarus version

Write directory  Folder where to write savegames, relative to "\$HOME/.solarus/". Must identify your quest to avoid confusion with other quests.

Quest title  The name of your quest.

Summary  One line describing your quest.

Description  A more detailed description of your quest.

Author  People who develop this quest.

Quest version  Current release of your quest.

Release date  In progress  Released  Status and date of the current release.

Website  Official website of the quest.

## Example of filled in Quest Properties:

**Quest properties**

**Quest information**

Solarus version 1.5

Write directory  Folder where to write savegames, relative to "\$HOME/.solarus/". Must identify your quest to avoid confusion with other quests.

Quest title  The name of your quest.

Summary  One line describing your quest.

Description  A more detailed description of your quest.

Author  People who develop this quest.

Quest version  Current release of your quest.

Release date  In progress  Released  Status and date of the current release.

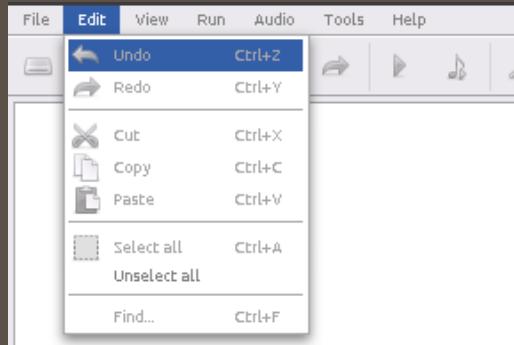
Website  Official website of the quest.

**Quest properties**

Summary	A short sample quest provided with Solarus One line describing your quest.
Description	The Solarus sample quest. There is not much to play yet, but any help is appreciated!
	A more detailed description of your quest.
Author	Christopho People who develop this quest.
Quest version	
Release date	<input checked="" type="radio"/> In progress <input type="radio"/> Released 12/15/2016
Website	<a href="http://www.solarus-games.org">http://www.solarus-games.org</a> Official website of the quest.
<b>Quest size</b>	
Normal quest size	320 <input type="button" value="▲"/> x 240 <input type="button" value="▲"/>
	Size of the logical game area (before any scaling). This will be the visible space of the current map.
Minimum quest size	320 <input type="button" value="▲"/> x 240 <input type="button" value="▲"/>
	Only useful to support a range of logical sizes.
Maximum quest size	320 <input type="button" value="▲"/> x 240 <input type="button" value="▲"/>
	Only useful to support a range of logical sizes.

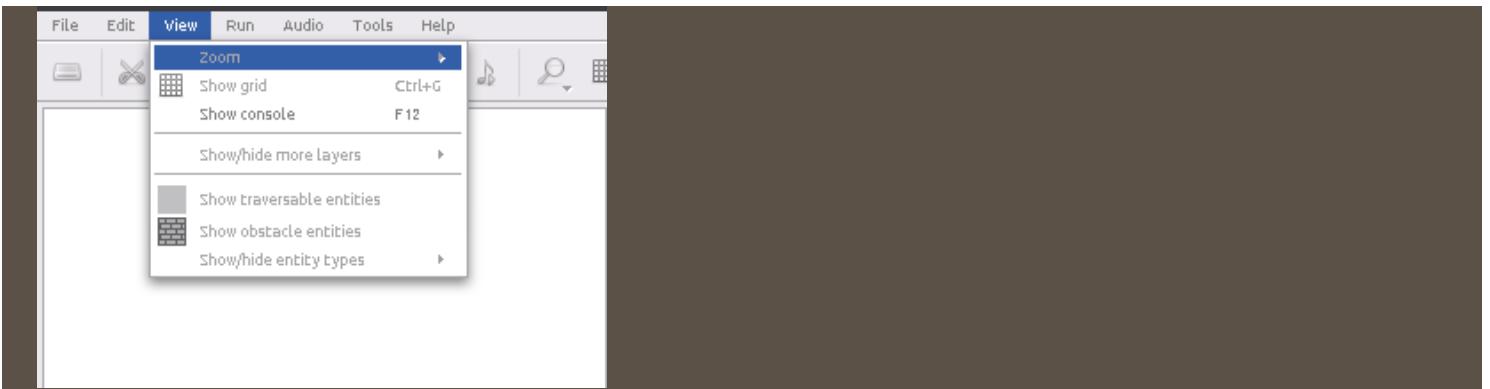
## Edit Menu

One can undo, redo, copy, paste, cut, select all, deselect all, and find text (In the Solarus text editor). The **edit** menu is located at the upper left corner next to **file**.



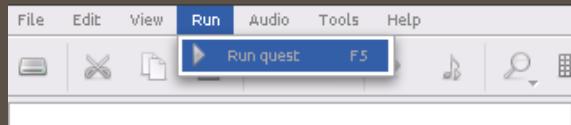
## View Menu

One can zoom, show grid, show console, show/hide layers, show/hide entities, and show/hide entity types in the **view** menu next to **edit**.



## Run Menu

One can run the game from the `run` menu or press F5. This menu is right next to the `view` menu.



## Audio Menu

One can play selected audio files from this menu. This menu is right next to the `run` menu.



## Tools Menu

This menu is a bit more complex than all the others. The `tools` menu has a lot of options that makes the Solarus experience even better.



### Tools > Options

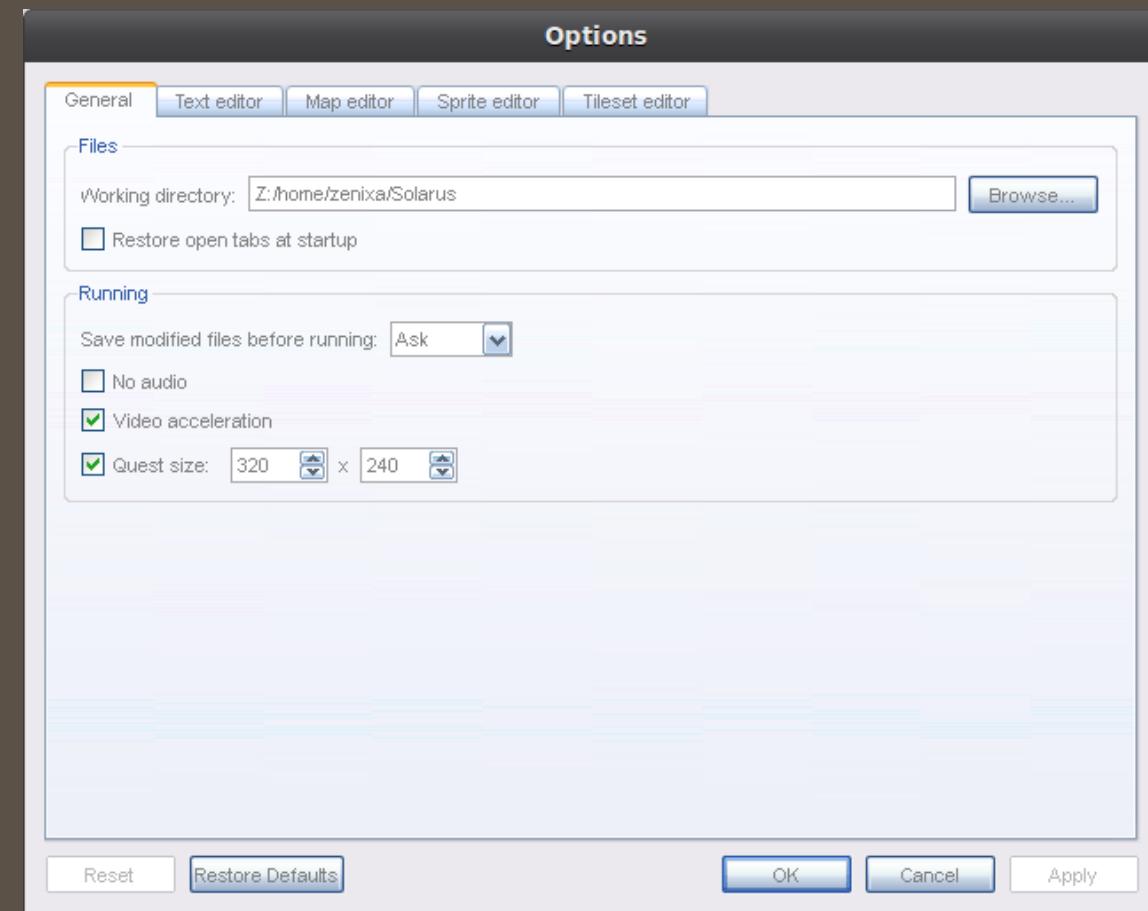
There are many tabs. At the `Options` menu. The options in the tabs are General, Text editor, Map editor, Sprite editor, and tileset editor.

#### Tools > Options > General Tab

- 1.The General tab contained the `working directory` this is where our projects will be located.
- 2.There is an option to `restore opened tabs at startup`.
- 3.Another option is to `Save modified file` before running the quest.
- 4.One can disable the audio
- 5.An ability to disable and enable video acceleration.
- 6.Change the default quest size.

generated by haroopad

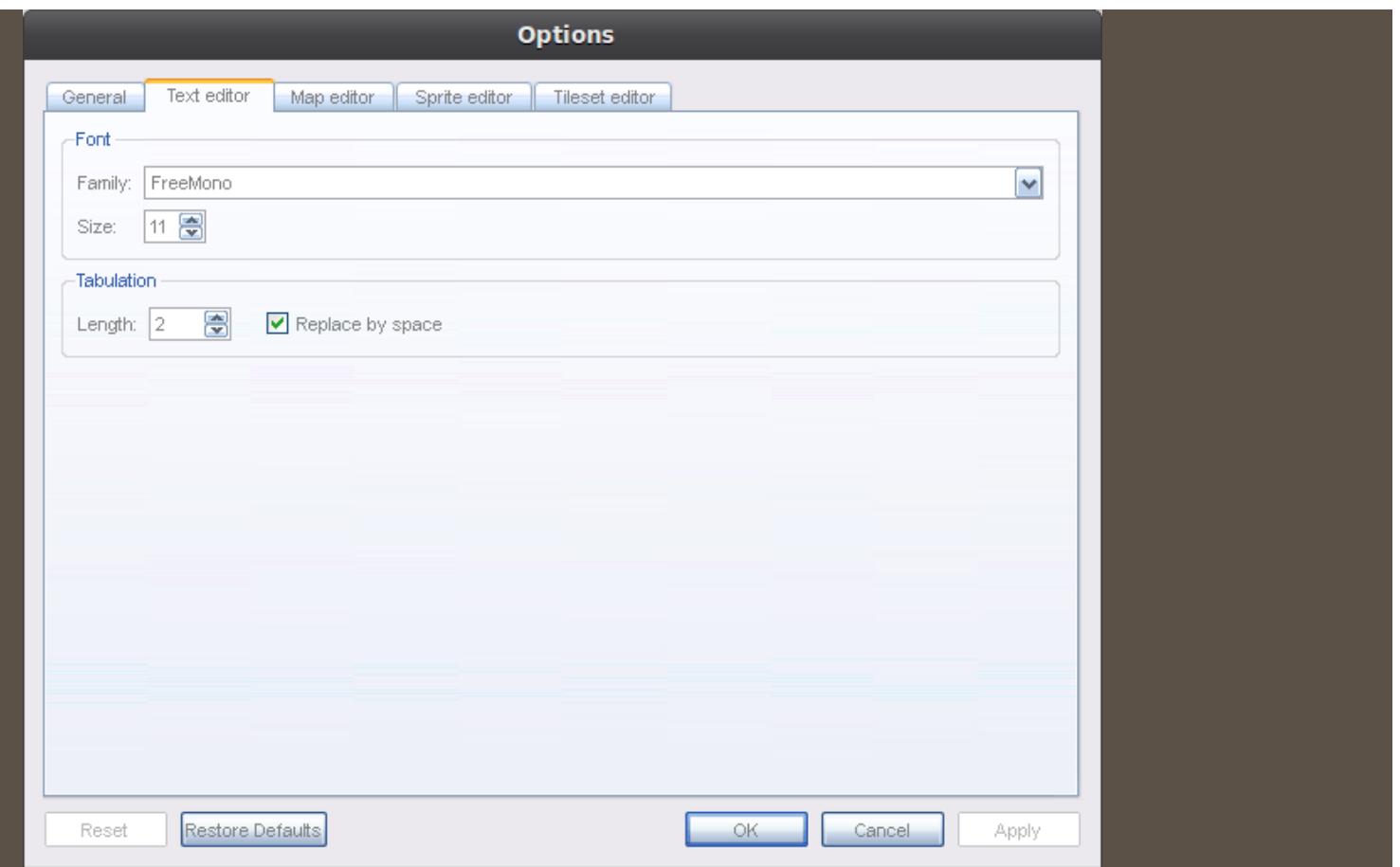
7. At the very bottom of the window you can **restore default** settings.



### Tools > Options > Text Editor Tab

The **Text editor** options are very simple.

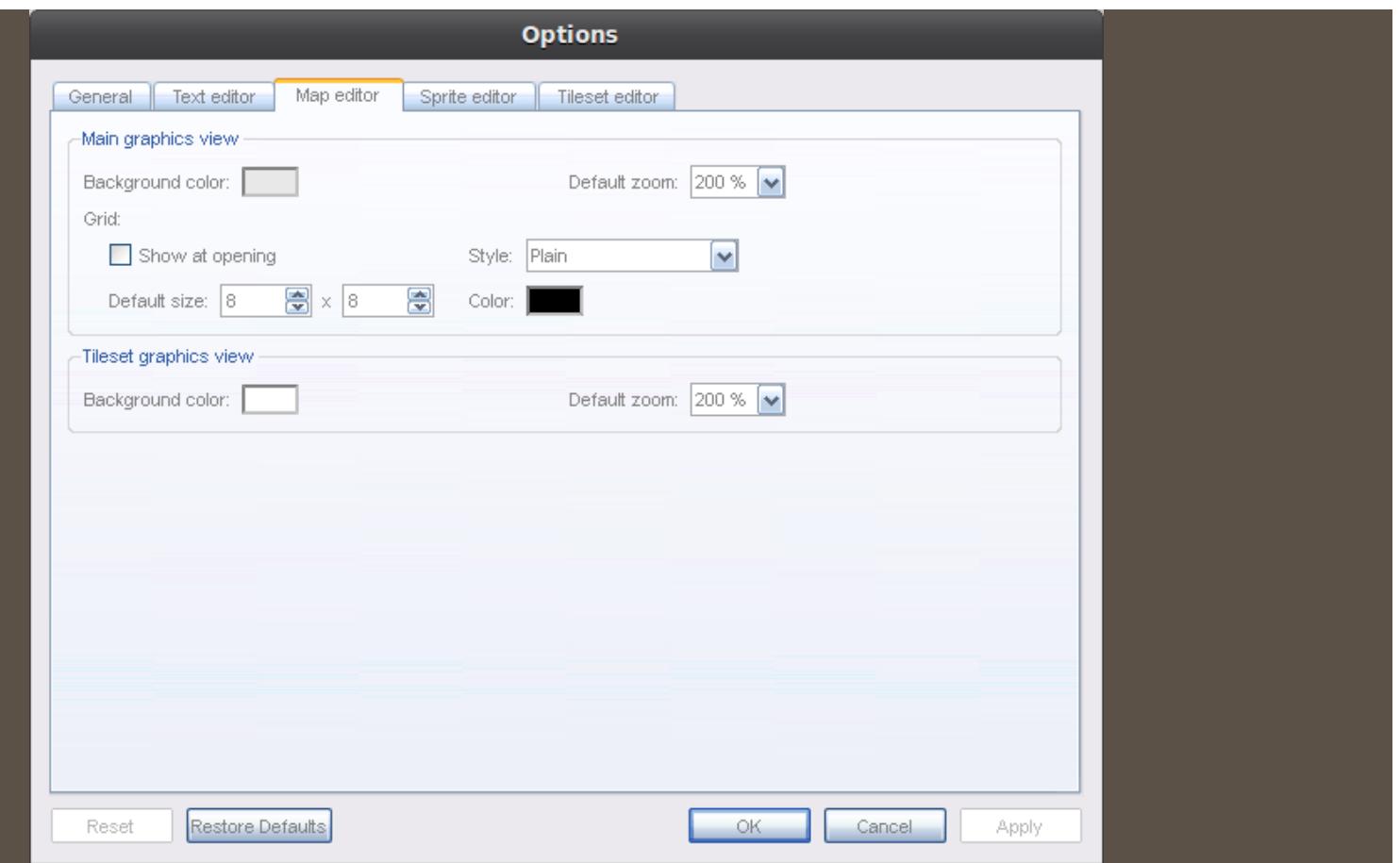
1. Able to change the font.
2. An option to change the font size.
3. Tabulation option — which is the space length when one hits the tab key.



## Tools > Options > Map Editor Tab

The [map editor](#) has a lot of useful options.

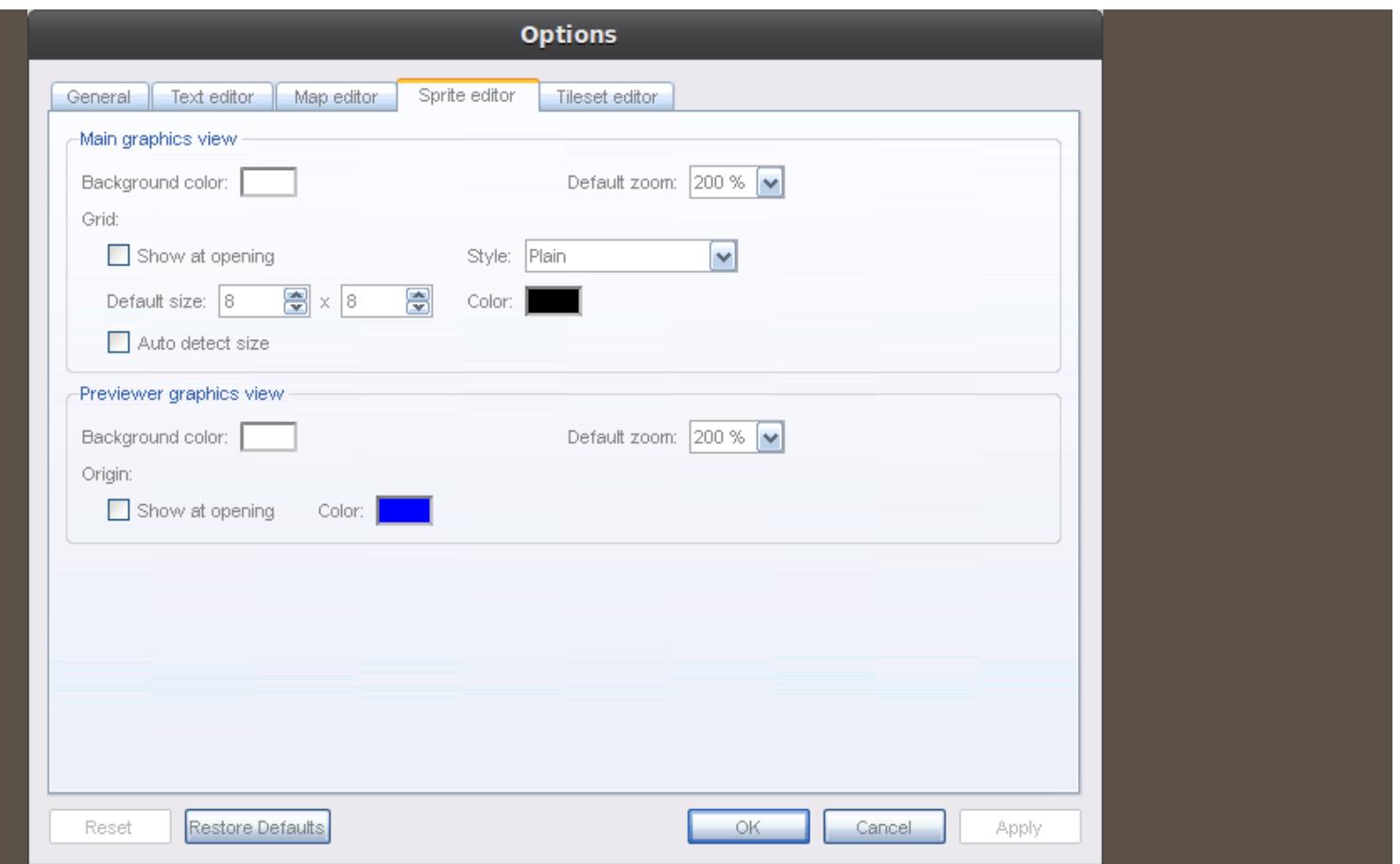
1. One can change the background color of the map.
2. One can change the grid options. Have the grid open at start, change the default grid size, change the grid color, and change the grid style.
3. Change the default zoom percentage for the map.
4. Change the zoom and background color for the tileset view.



## Tools > Options > Sprite Editor Tab

The [sprite editor](#) has almost the same options as the [map editor](#).

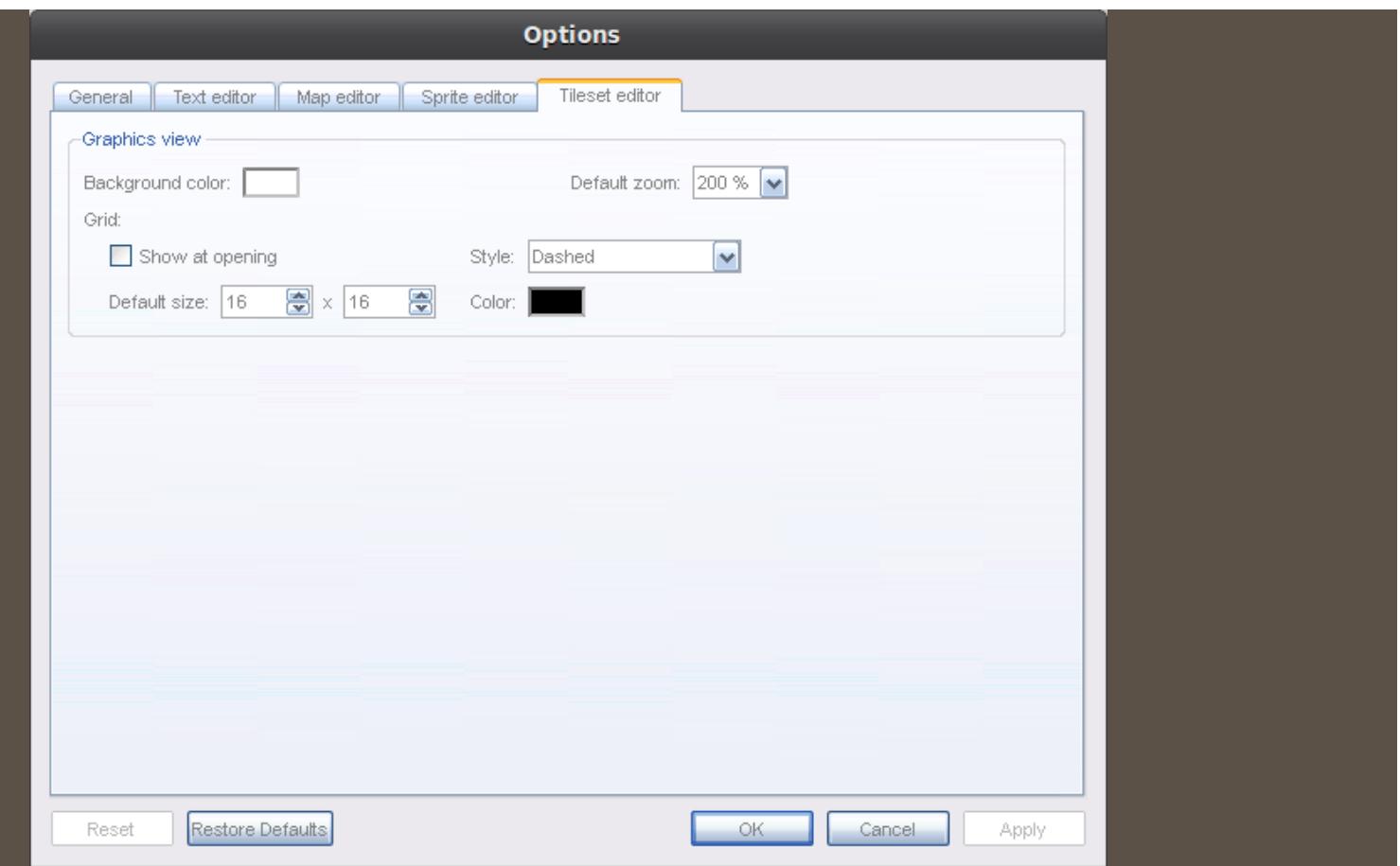
1. One can change the background color of the sprite editor.
2. One can change the grid options. Have the grid open at start, change the default grid size, change the grid color, and change the grid style.
3. Change the default zoom percentage for the map.
4. Change the zoom and background color for the graphic view.
5. Make the origin show at opening and change the origin color



## Tools > Options > Tileset Editor Tab

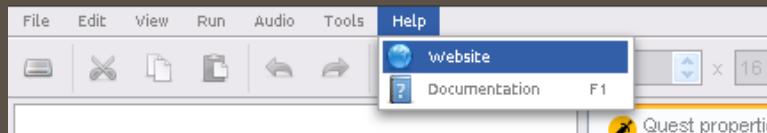
The [sprite editor](#) has almost the same options as the [map editor](#) and [sprite editor](#).

1. One can change the background color of the tileset editor.
2. One can change the grid options. Have the grid open at start, change the default grid size, change the grid color, and change the grid style.
3. Change the default zoom percentage for the tileset.



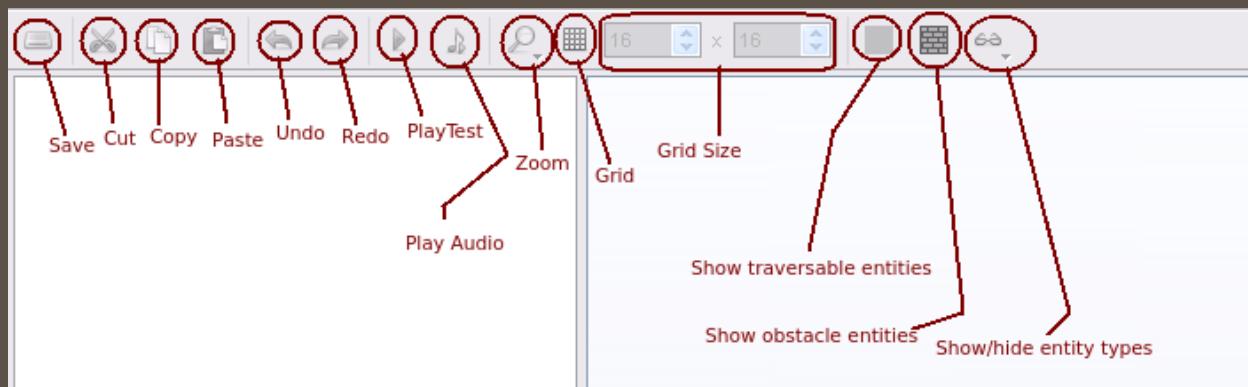
## Help Menu

The help menu links one to the Solarus Website or the Solarus Documentation Website.

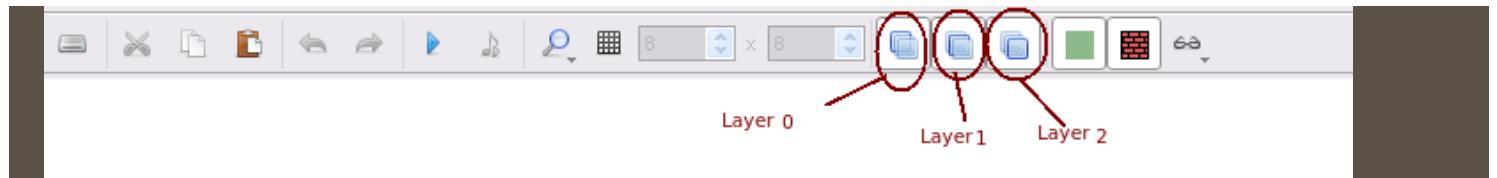


## Icon Shortcut Bar

The **icon bar** has a bunch of shortcuts. This is how it looks when a quest is not loaded.



The main difference for the **icon bar** from not having a quest open is color and layers. See the following image.



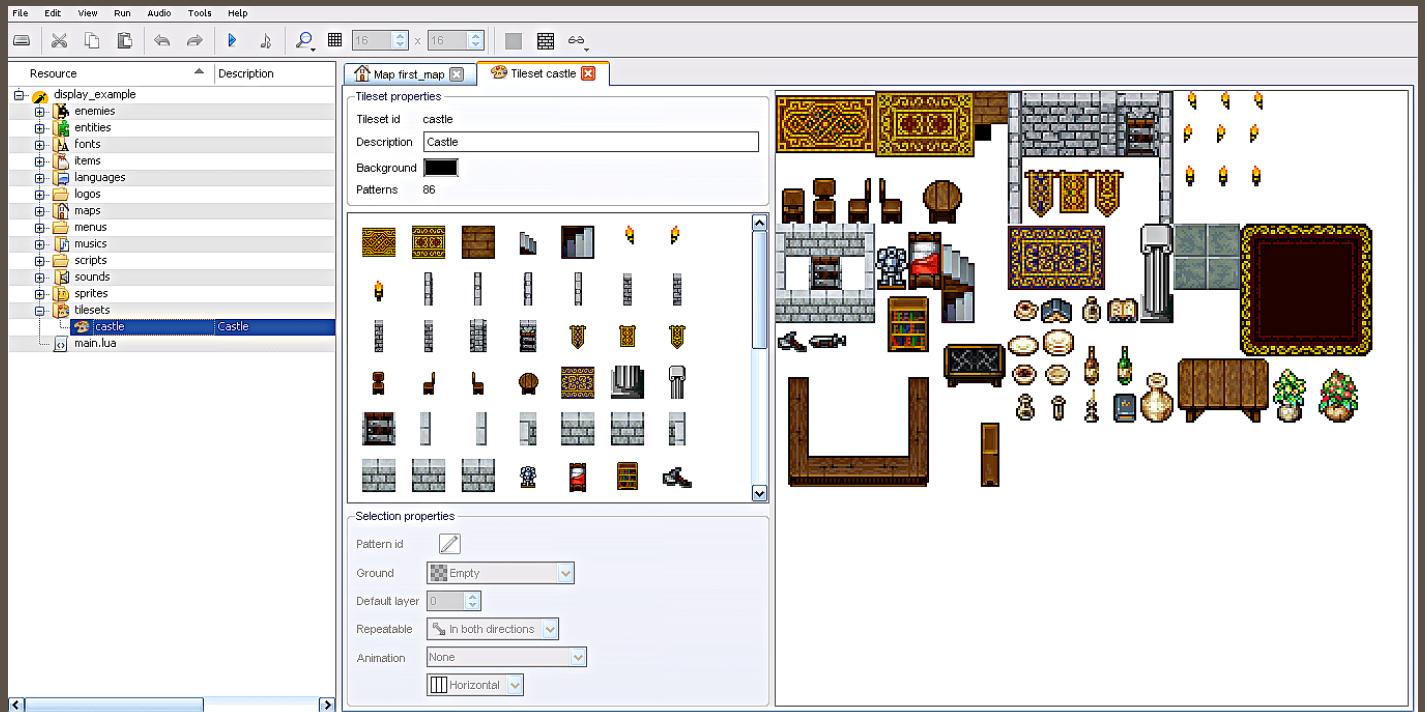
## Resource Manager List

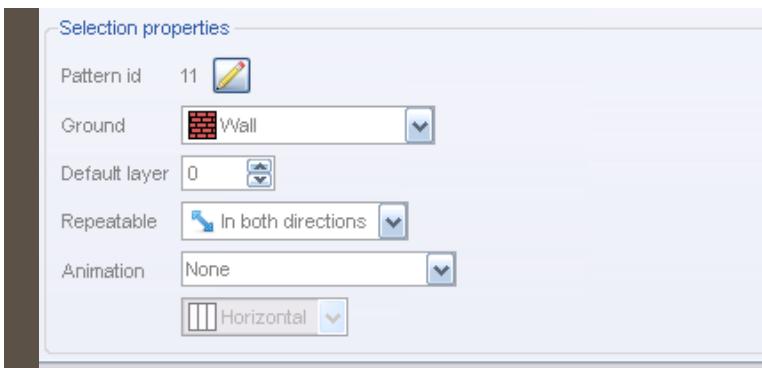
The resource list appears at the left side of the editor when a quest is loaded from [file > load quest](#).



## Resource Manager List > Tileset Editor

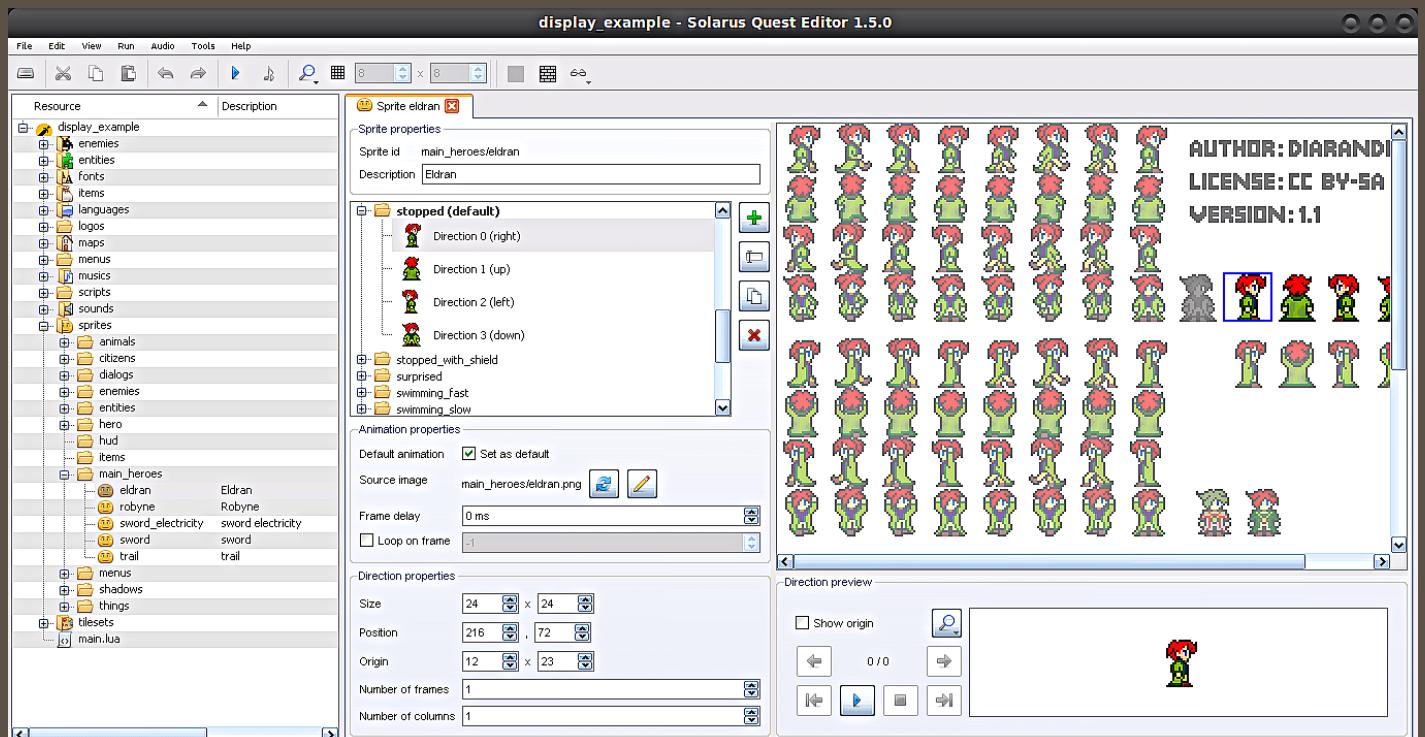
The tileset editor appears when a tileset is selected from the tileset folder. I will not get too much into the tileset editor this chapter.





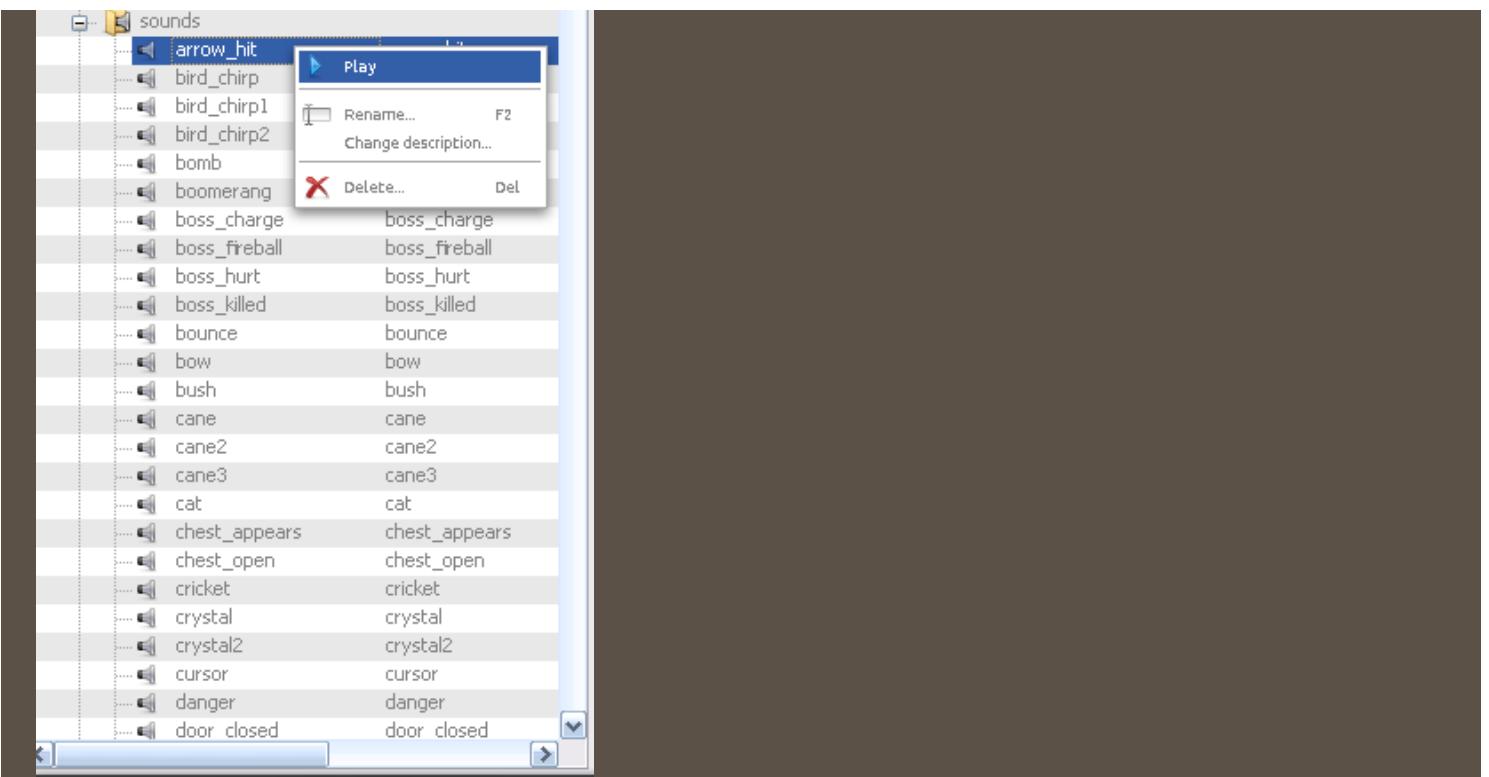
## Resource Manager List > Sprite Editor

The sprite editor appears when a sprite character is selected from the sprites folder. I will not get too much into the sprite editor this chapter.



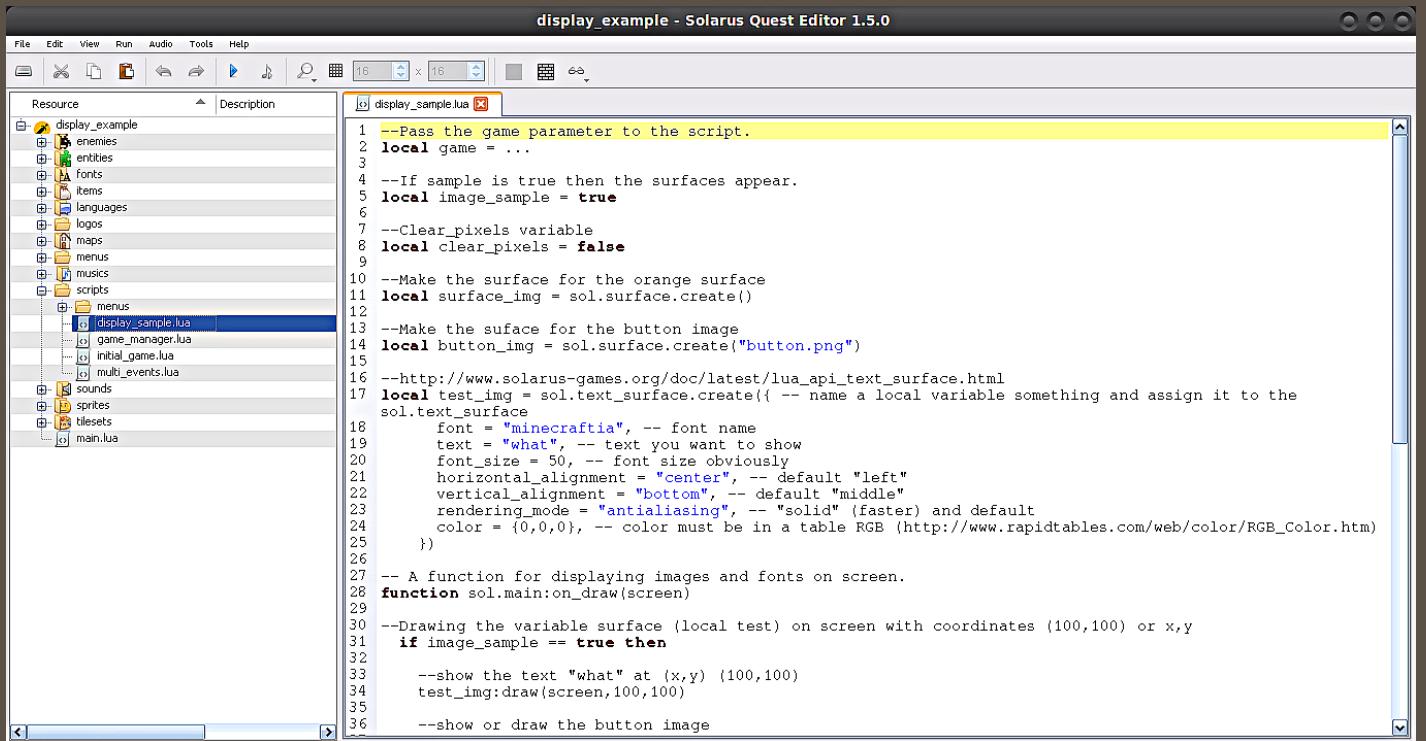
## Resource Manager List > Sound Player

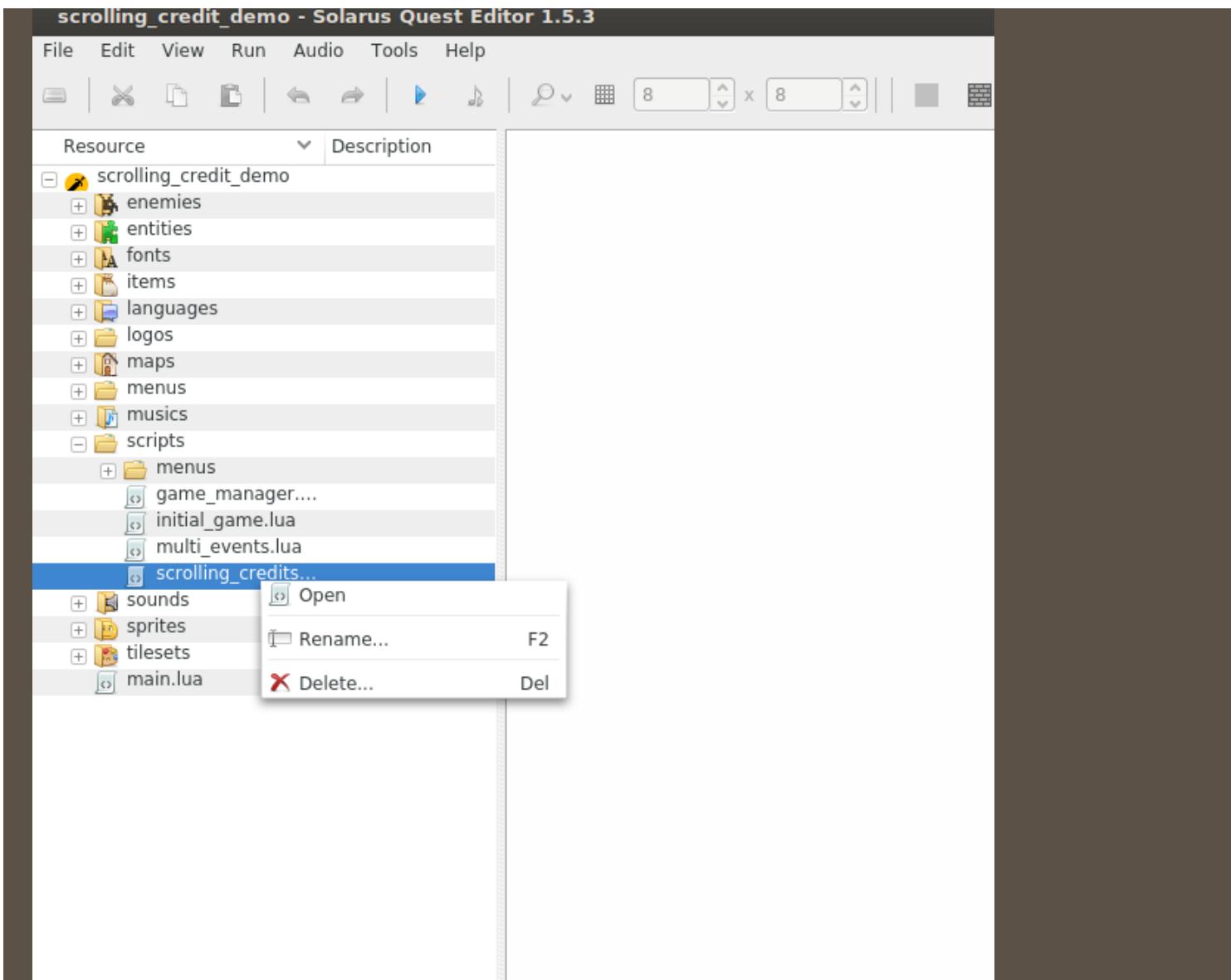
When a sound is selected from the sound folder and the right mouse button is clicked, the option to play audio shows up.



## Resource Manager List > Text Editor

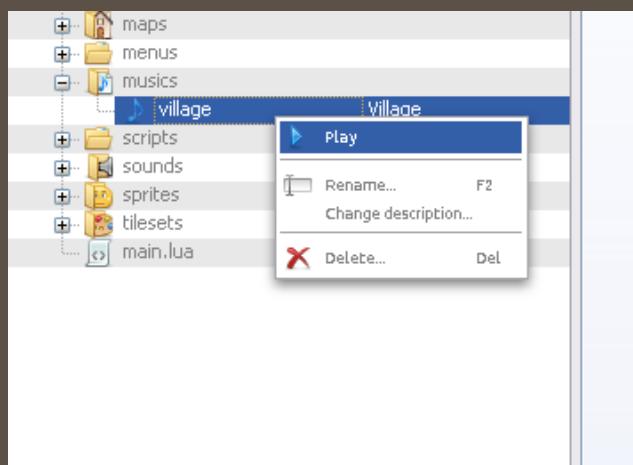
A script editing window will open if one were to right click on a script and open it. Also, one can rename the script from that window as well. Also, if the left mouse button is double clicked on the script, then script editor will appear.





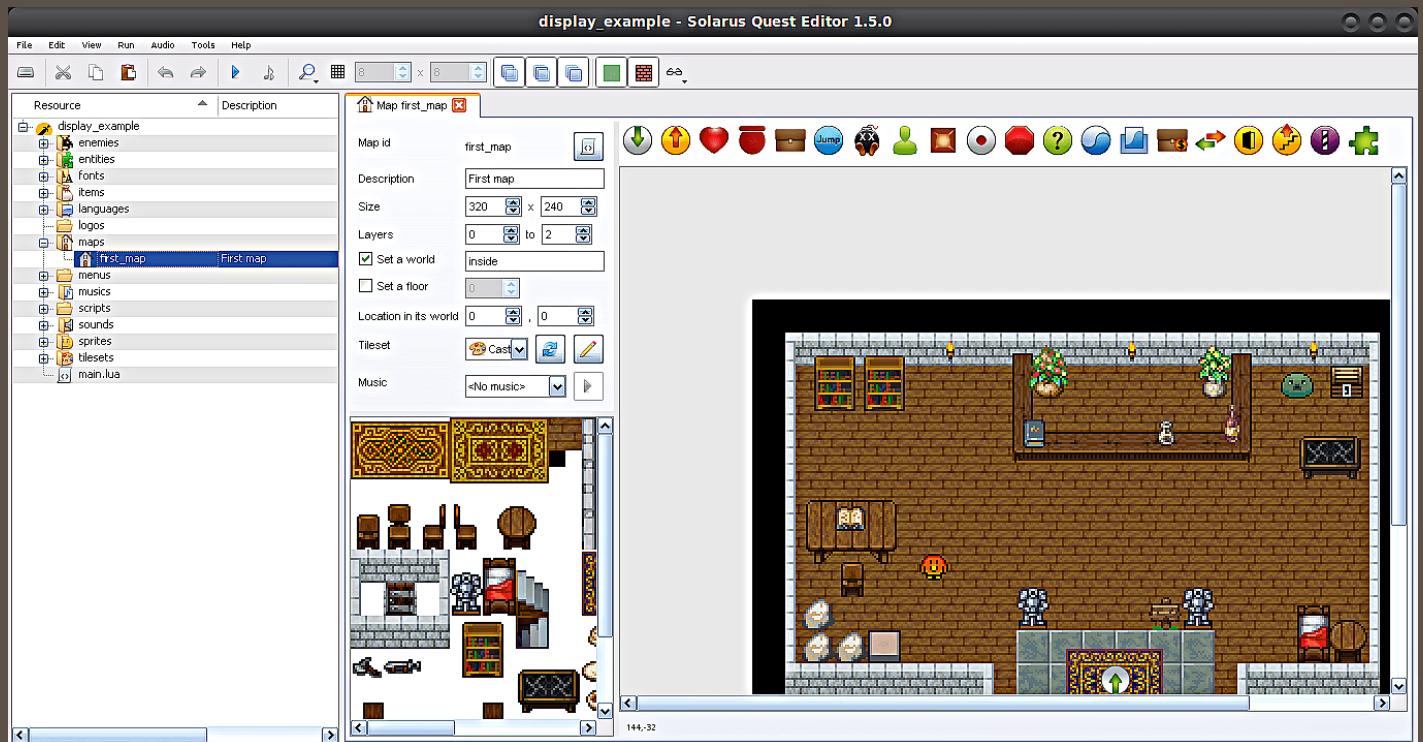
## Resource Manager List > Music Player

When a music is selected from the music folder and the right mouse button is clicked, the option to play audio appears.



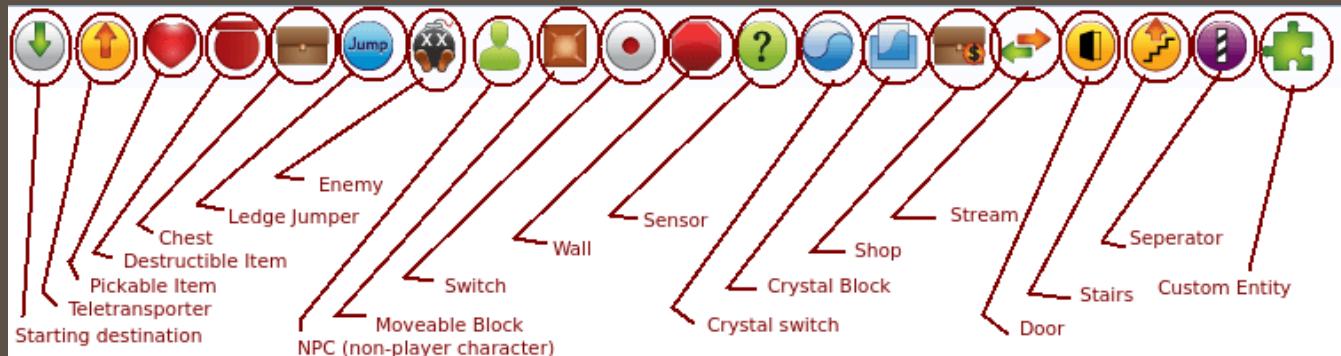
## Resource Manager List > Map Editor

When a map is selected from the map folder and the left mouse button is double clicked, the map editor activates.



## Resource Manager List > Map Editor > Entities

There are many entities for the map.

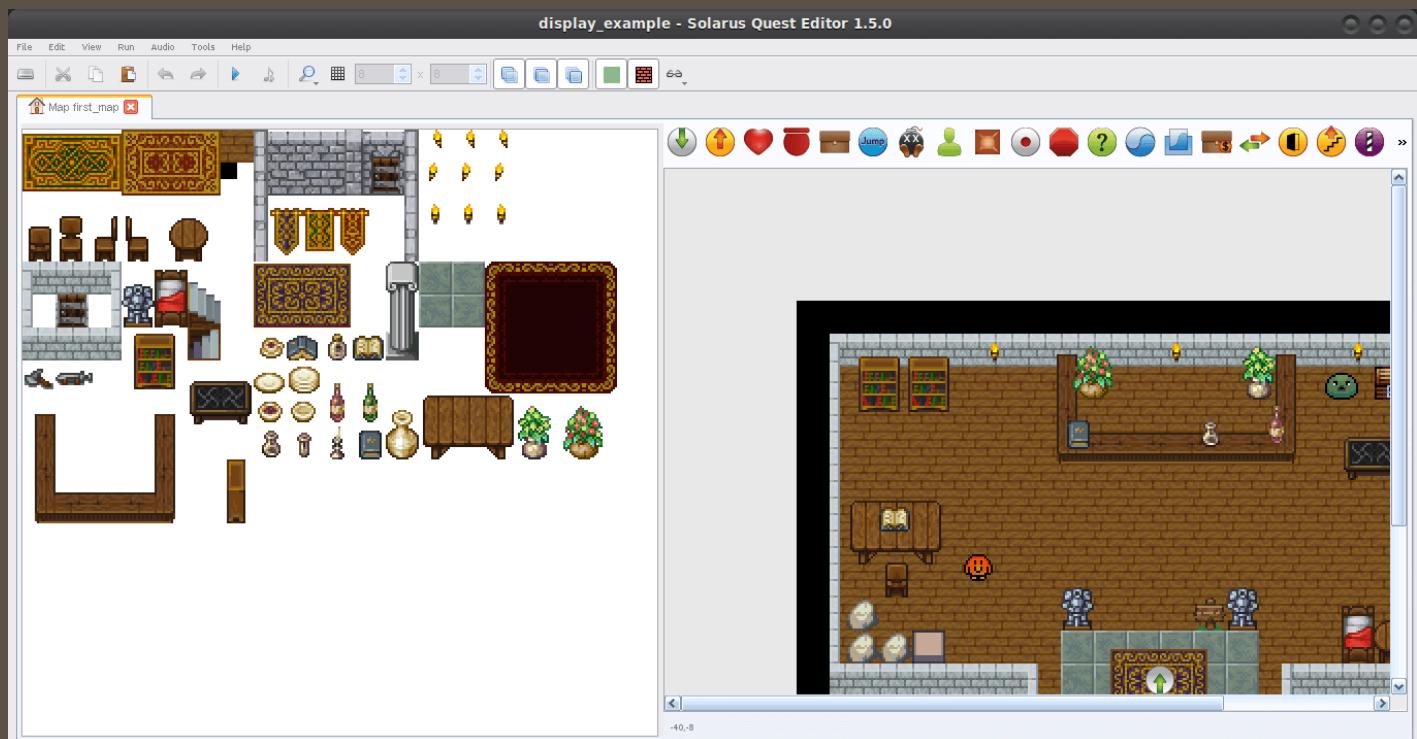


Entity	Description
Destination	The starting point.
Teletransporter	Where the character appears on another map
Pickable	An item that is picked up, like a coin or rupee.
Destructible	Something that can be destroyed
Chest	A container for items. You normally find chests in dungeons.
Jumper	This makes it so the player can jump over cliff edges and so on.
Enemy	A character that tries to harm or kill you.
NPC	A player that does not harm you and normally gives you advice or a quest.
Movable block	A block the player can move around or push.

Entity	Description
Switch	Once activated on something occurs. Ex: Flip a light switch and a light comes on.
Wall	A custom wall. Maybe it can vanish.
Sensor	Once stepped on something occurs. Ex: Step on a mine and boom!
Crystal	The crystal is related to the crystal block. When it is hit, the crystal blocks go under ground or block the player's path.
Crystal block	Crystal blocks go under ground or block the player's path.
Shop	A simple shop that can be set up.
Stream	A stream. Possibly can be used for current.
Door	A door that can be opened and closed.
Stairs	Used for a walking up stair animation.
Separator	Blocks the camera or player from seeing parts of the dungeon or room.
Custom Entity	Make your own entity!

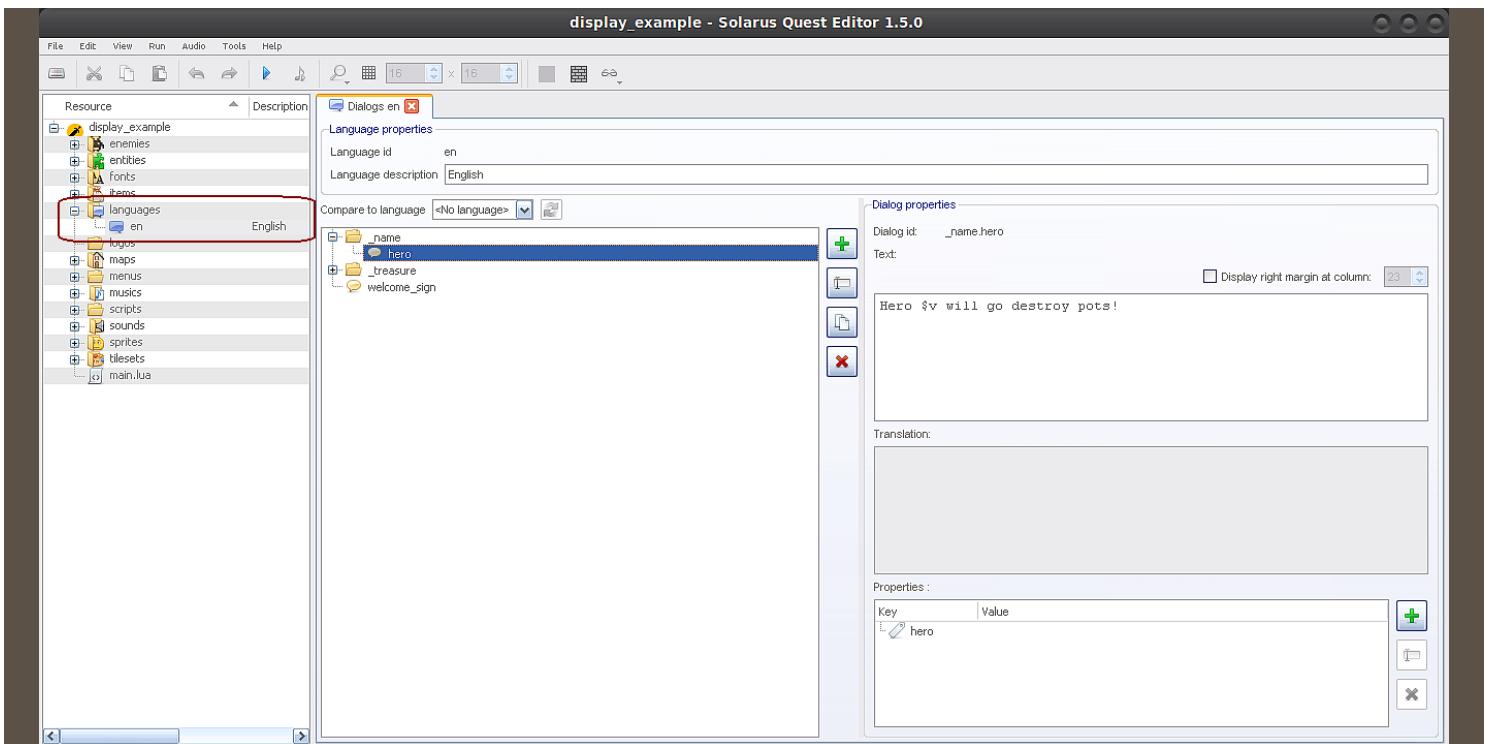
### Resource Manager List > Map Editor > Drag

One can drag the edges of the Solarus Editor to have a better view of the tileset for mapping the game.



### Resource Manager List > Language > en > Dialog Editor

When the language type `en` is double click on with the left mouse button inside the language folder, the dialog editor appears.

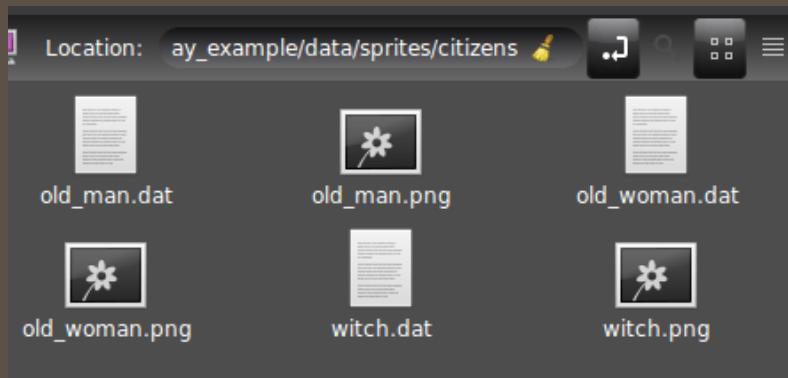


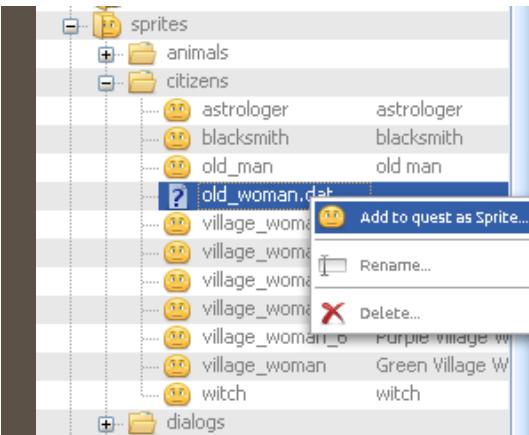
## Chapter 4: Using The Sprite Editor

### Resource Manager > Sprites > Sprite Editor

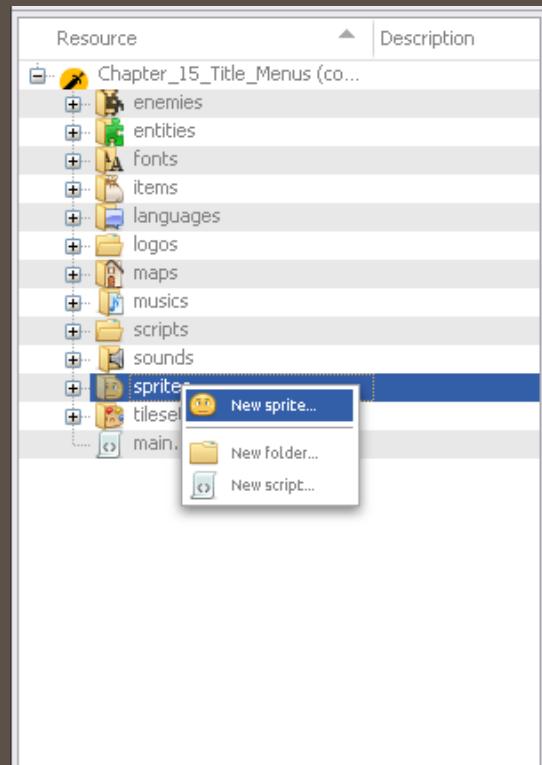
You can add a sprite character by putting a sprite image in the sprites folder directory and add it by right clicking in the Solarus Editor resource manager.

For example, `old_woman.dat` for `old_woman.png`.

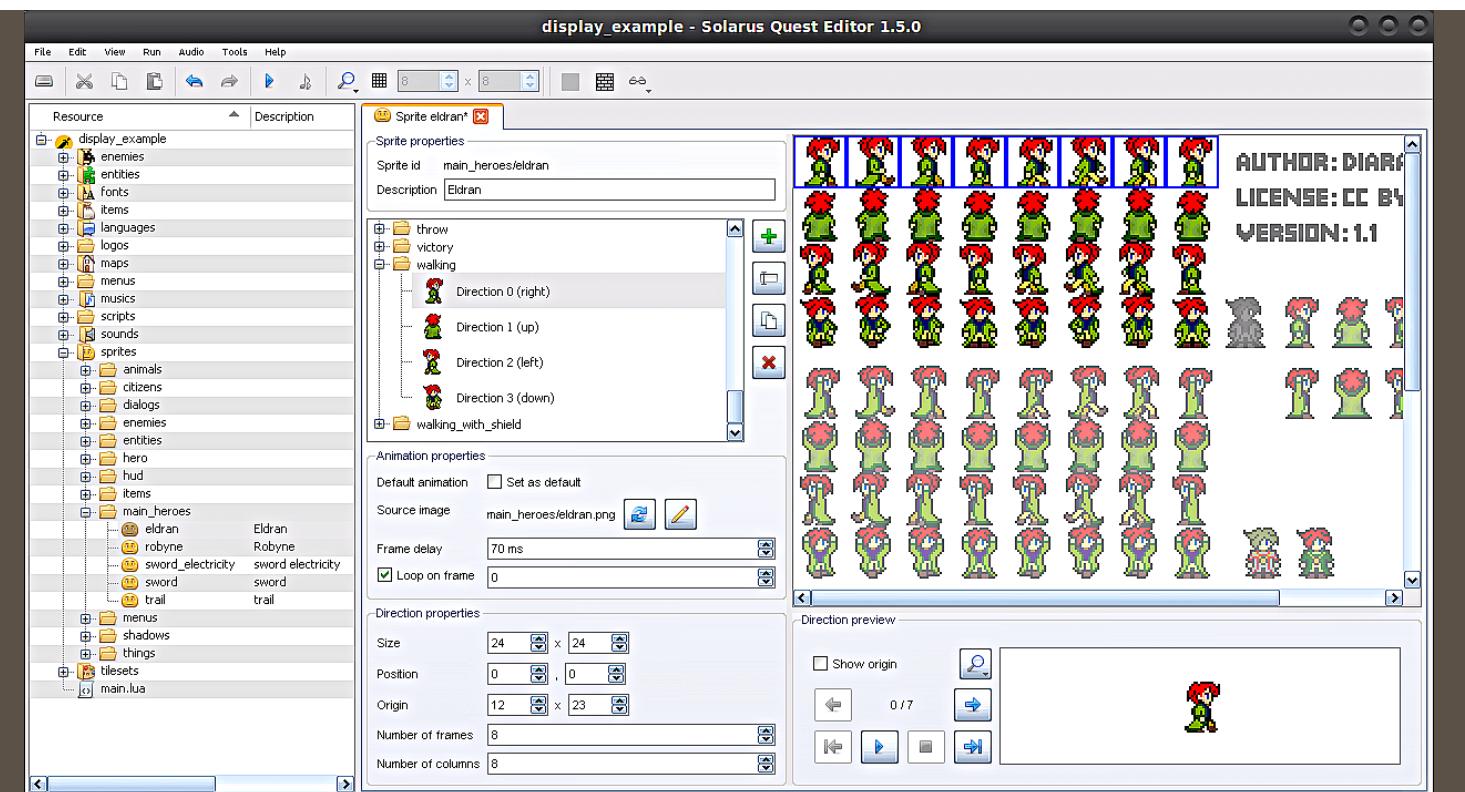




You can right click on the **sprites** folder in the resource manager to add as well.

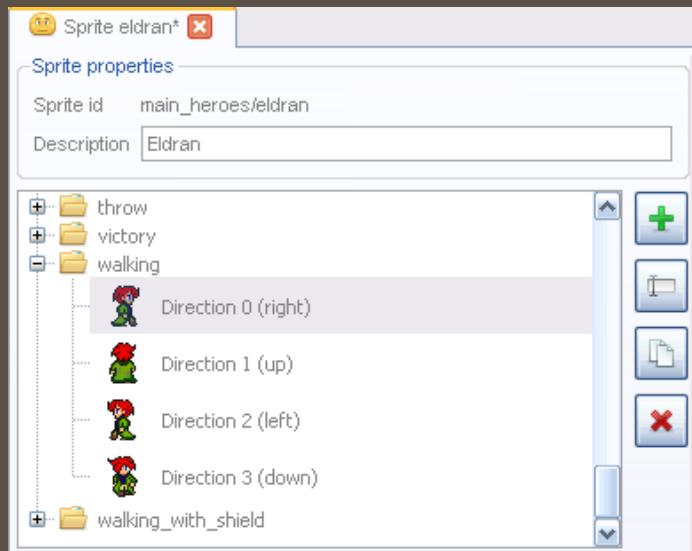


The sprite editor appears when one double clicks on a sprite name.

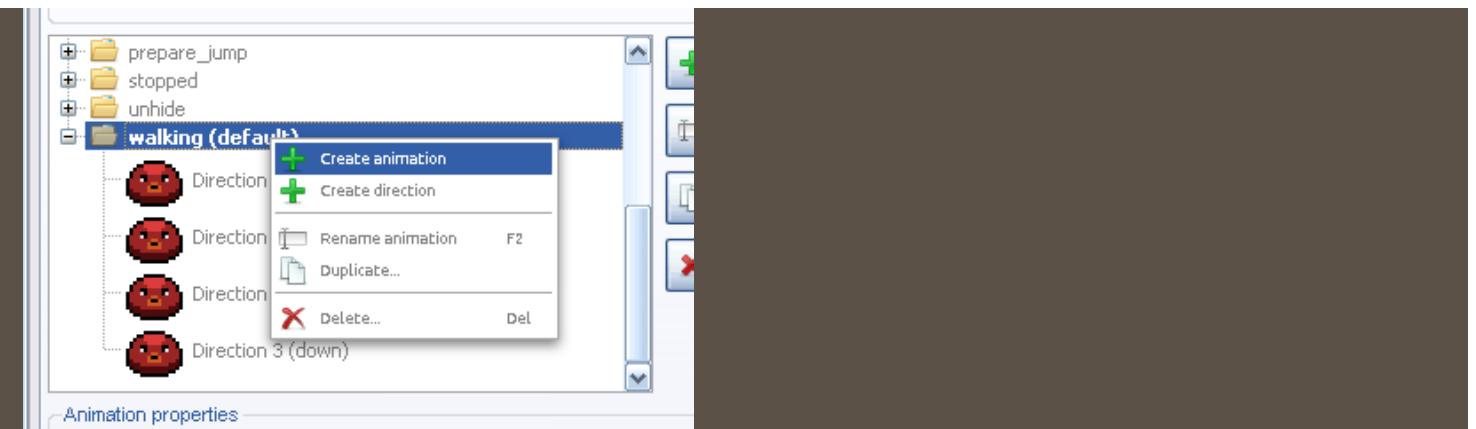


## Sprite Properties

You can add various different types of movements in the sprite editor. You can use the green plus(+) sign to add new ones. The other options allow duplicating an animation, deleting(x) an animation, and renaming them.

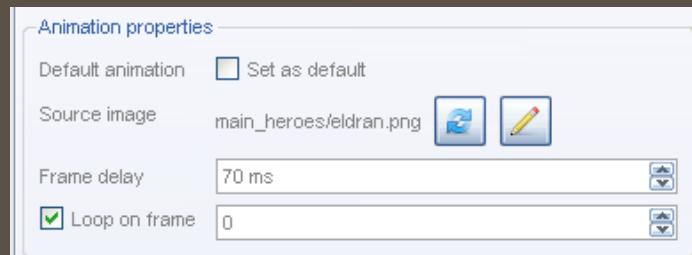


You can right click the mouse to do the same task.

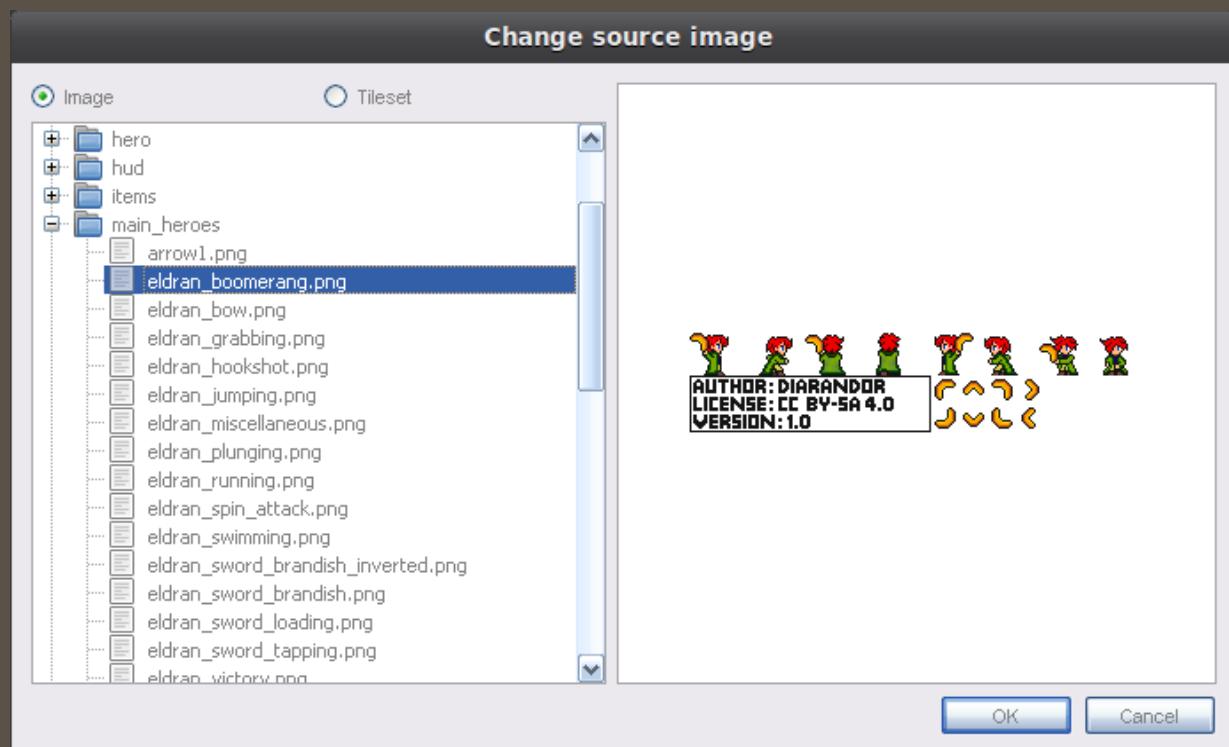


## Animation Properties

There are many options when it comes to the animation properties.



1. Setting an animation as default
2. Selecting a source image



1. Frame delay
2. Loop on a certain frame number. Ex: Loop on frame 0

It can be used for the preview if checked. Also, if this is checked on the walking animation for a sprite, then the sprite will walk in place. It will not move.

## Direction Order

The direction numbers represent the animation direction.

### EX:

**Direction 1** would be walking to the right.

Number	Direction
Direction 1	Right
Direction 2	Up
Direction 3	Left
Direction 4	Down

## Direction Properties

The **direction properties** contains dimension options that allows one to easily set up a sprite.



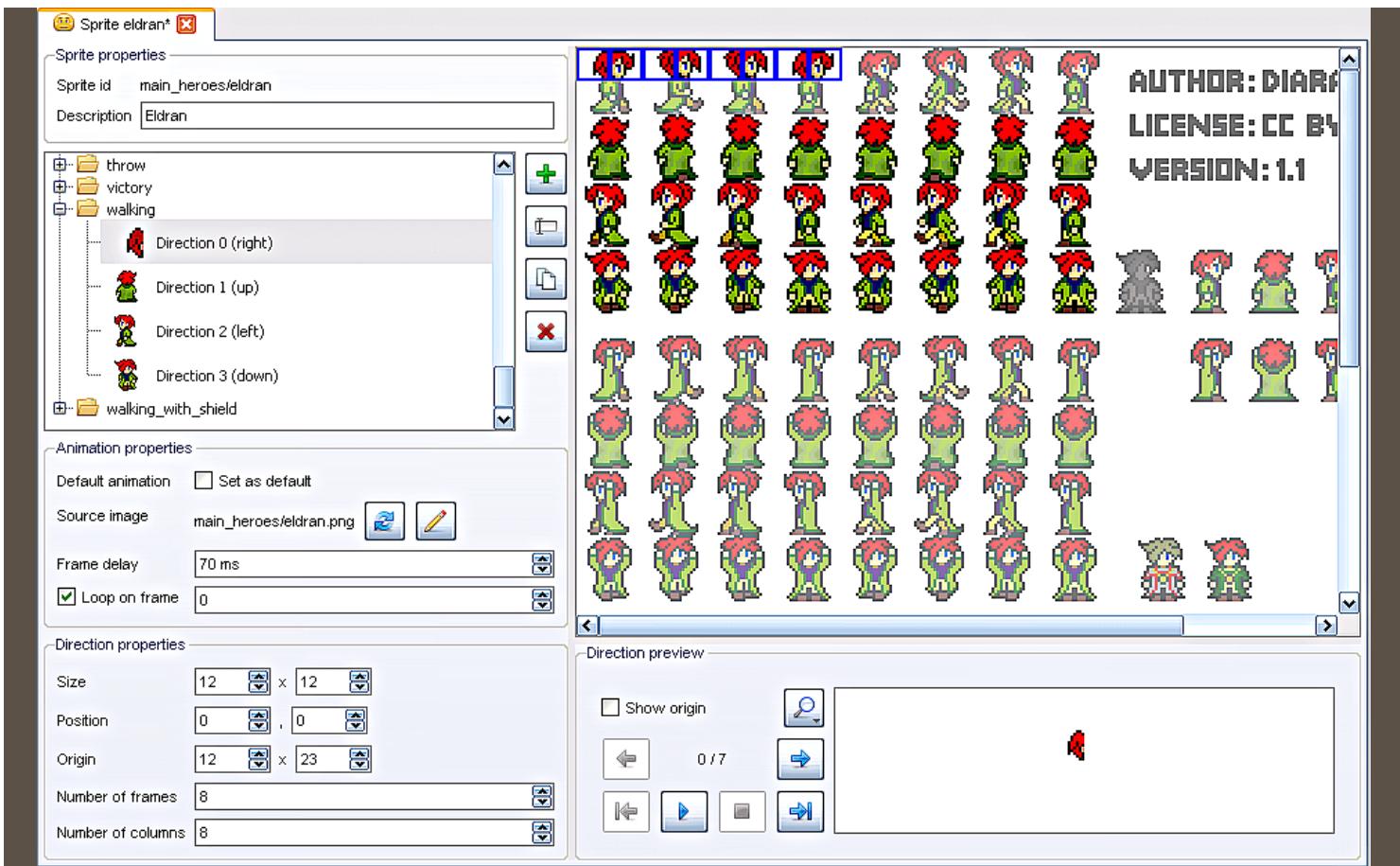
1. Size
2. Position
3. Origin
4. Number of frames
5. Number of columns

### Size

This is what it looks like when the size is changed. I changed the size from 24 x 24 to 12 x 12.



*The result:*

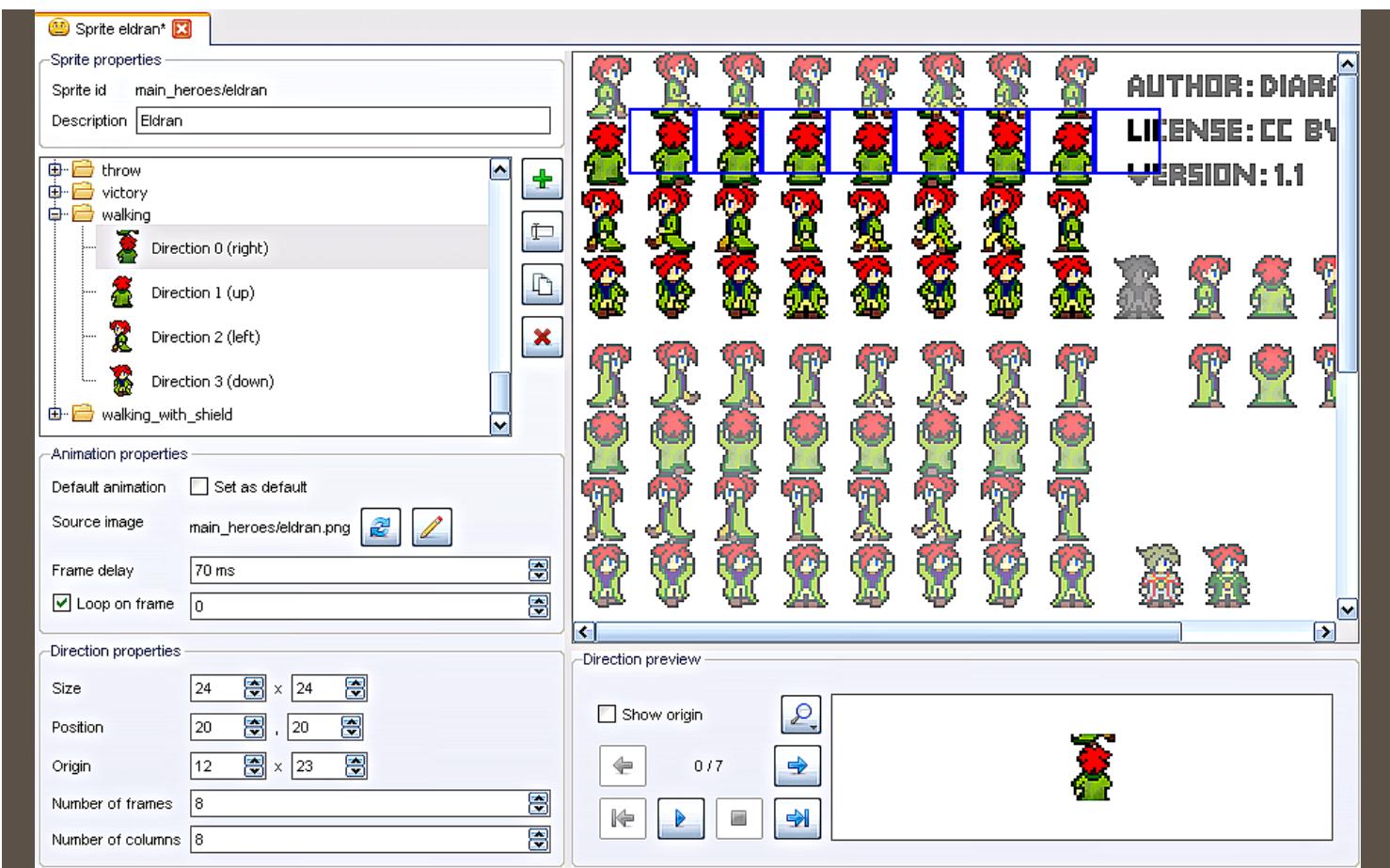


## Position

This is what it looks like when the position is changed from  $0 \times 0$  to  $20 \times 20$ .

Position	20	<input type="button" value="Up"/>	,	20	<input type="button" value="Up"/>
----------	----	-----------------------------------	---	----	-----------------------------------

*The result:*



## Origin

The origin is the center of the player character. One would normally have it  $y$  centered and  $x$  lowered down to the sprite's feet. This helps with proper collision with tileset objects. For this sprite the  $x$  is 12 and  $y$  is 23. ( $12 \times 23$ )



I will change the origin from  $12 \times 23$  to  $6 \times 12$ .



## The result:

The player will not collide with objects properly this way.

EX: A tree. No one wants to bump into a tree 4 steps before it is reached.



## Switch Origin

The origin is different for switches and some other sprites. The origin needs to be centered to the upper left corner for a switch. You check the sample in chapter 15. The samples are in the directory

[Lessons/Chapter\\_15/Chapter\\_15\\_Sample.zip.](#)

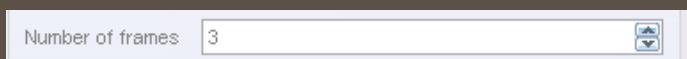


## Frames

This is what it looks like when the frames are set to 8.

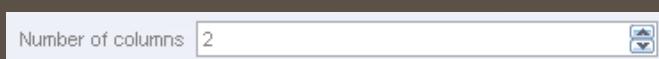


Now this is what it looks like when the frames are changed from 8 to 3.



## Columns

This is what it looks like when the column is changed from 8 to 2.

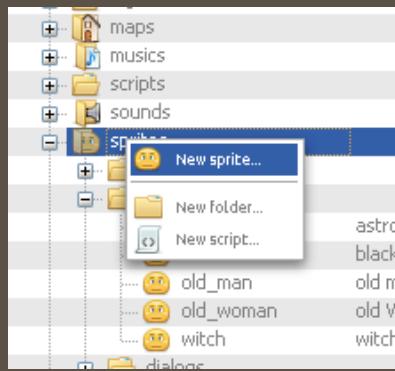


## Setting up a Sprite

Everyone needs to know how to setup a sprite. The Solarus Sprite editor is super amazing and makes this task easy as pie!

## Making a Sprite

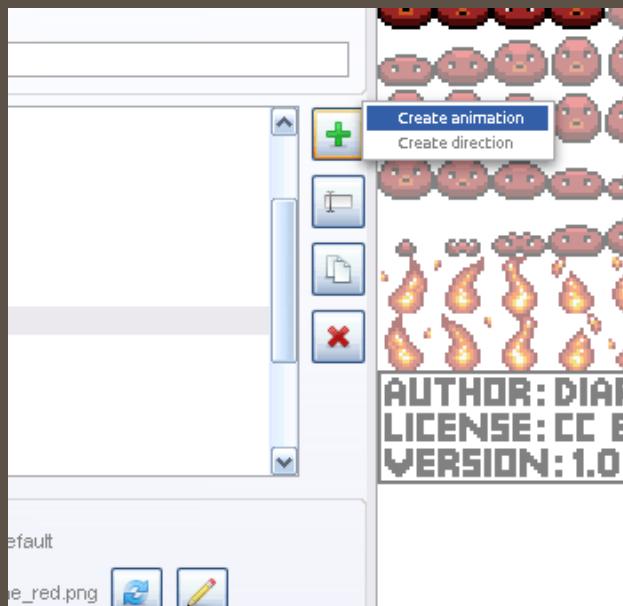
Making a new sprite is first thing that needs to be done.



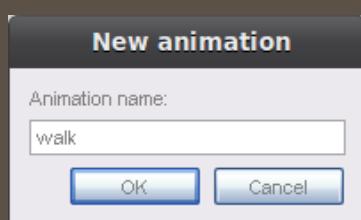
### Name Sprite



### Create Animation

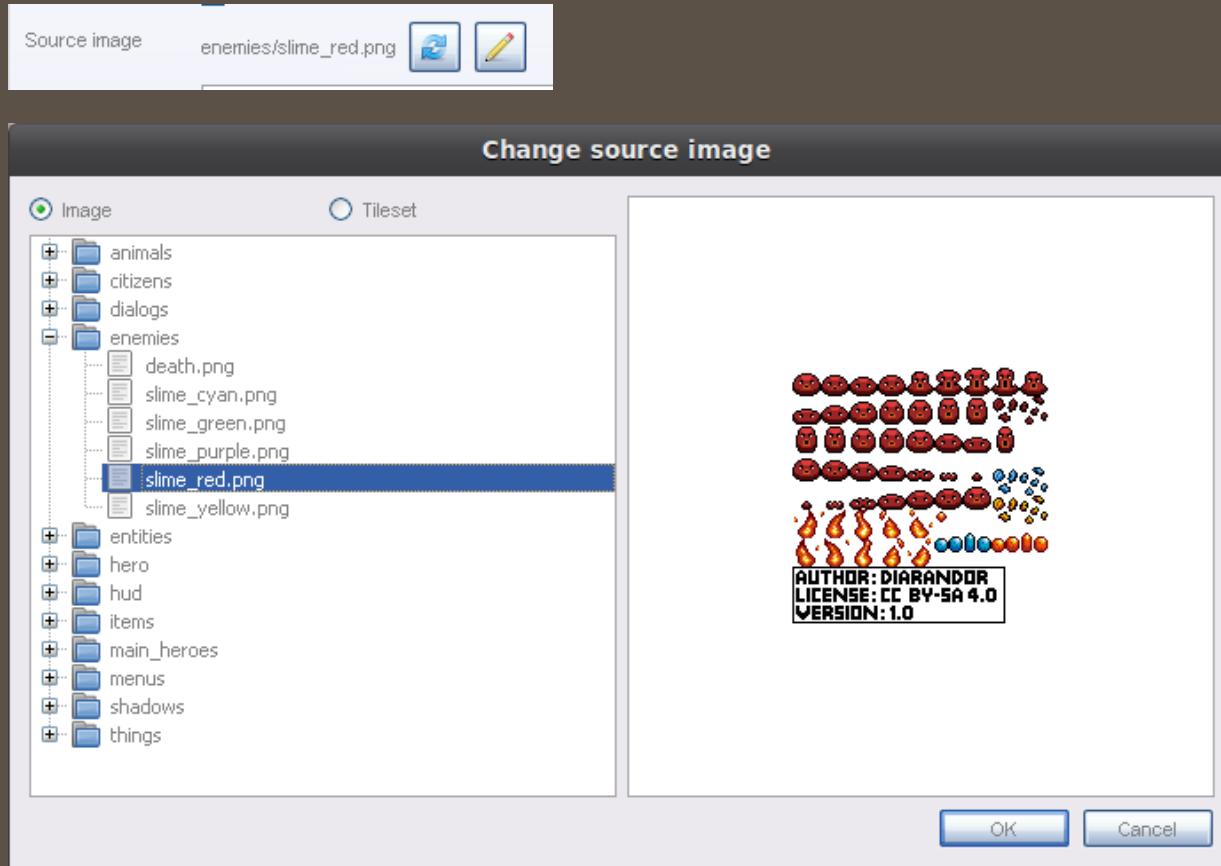


### Name Animation



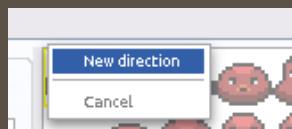
## Load Sprite Image

Click the pencil to load an image from the sprite directory.

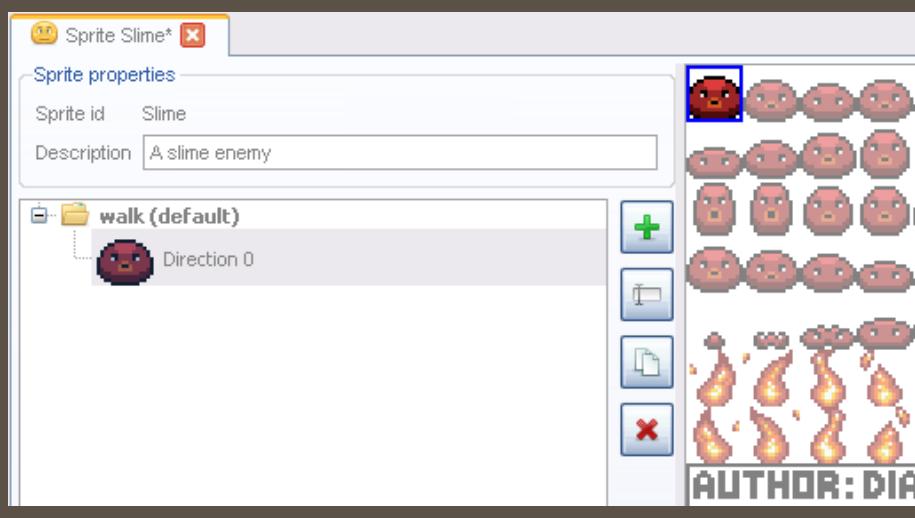


## Making Directions - Drag Click Release

Click one of the sprite image frames and add it as a new direction. This can be done by holding down the left mouse button and dragging to highlight one of the red slime sprites. The first sprite at the upper left corner would be best.

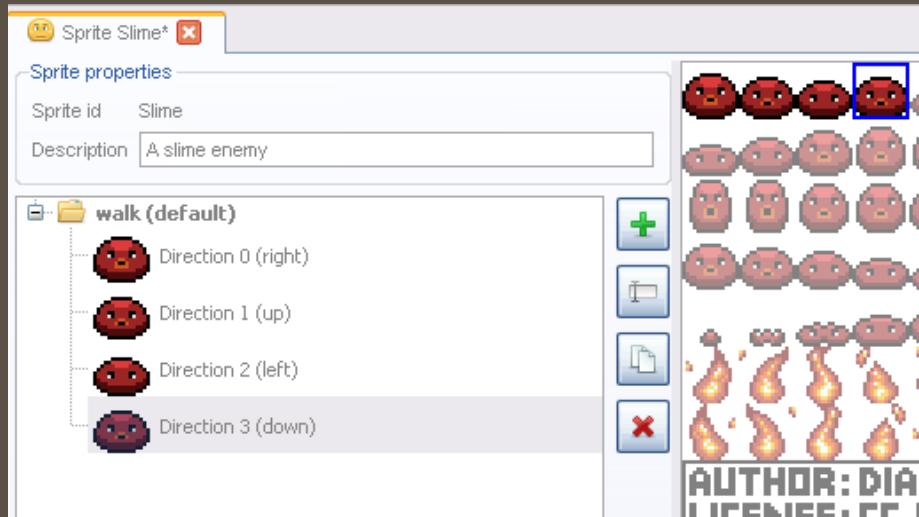


That creates a direction called, “[Direction 0](#)”.

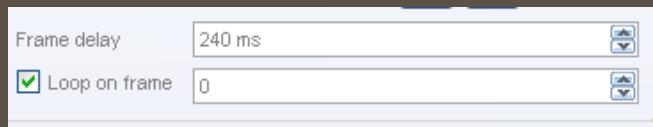


generated by haroopad

Repeat this process until you get 4 frames.



### Set Frame Delay & Loop



### Set Columns, Rows, & Origin

We are using 4 frames, so column and rows are 4.

Origin should be fine at 8x8.



### Show Origin

Click the check box to show the origin.



### Zoom On Sprite Preview



### Zoom On Sprite Animation Direction Setup Area

You can use **CTRL + Scroll Wheel** to zoom in and out of the animation direction setup area.



## Previous & Next Sprite Frame



## Last & First Frame + Play & Stop

Previous and next are the arrows.

Stop is the button with the square and play is the triangle that is on its side.



## Dat File Editing

**.dat** file editing is useful. If the dat file is going to be almost the same for every sprite, then the filenames just need to be changed. This would be faster than creating everything over and over again in the editor.

A few good notepad editors.

- Notepad++ (Windows/Wine)
- Jedit (Linux)
- Gedit (Linux): [advanced-find-add-on](#)

### Example:

Blue\_female.dat

```
src_image = "Erbarlow/Bard/Female/Blue_female.png",
```

Green\_female.dat

```
src_image = "Erbarlow/Bard/Female/green_female.png",
```

### Another Example:

Dat format is useful for trading animation to another sprite. Some animations can take a lot of time to setup.

```
animation{
    name = "stopped",
    src_image = "Erbarlow/Bard/Female/green_female.png",
    directions = {
        { x = 0, y = 64, frame_width = 32, frame_height = 32, origin_x = 16, origin_y = 32 },
        { x = 0, y = 96, frame_width = 32, frame_height = 32, origin_x = 16, origin_y = 32 }
    }
}
```

```

    { x = 0, y = 32, frame_width = 32, frame_height = 32, origin_x = 16, origin_y = 32 },
    { x = 0, y = 32, frame_width = 32, frame_height = 32, origin_x = 16, origin_y = 32 },
    { x = 0, y = 0, frame_width = 32, frame_height = 32, origin_x = 16, origin_y = 32 },
},
}

```

## Bounding box or Hitbox

The default size value is 16x16 pixels. This is the effective size used to detect obstacles when moving, but the sprite(s) of the custom entity and enemy may be bigger than 16x16.

The functions `custom_entity:set_size(width, height)` for the custom entity and `enemy:set_size(width, height)` can be used for enemies.

## Basic Sprite Information

Solarus has a sprite editor to set up animations and you can make as many as you want. You can activate your custom animations for the hero with `hero:set_animation("name_of_animation")` and entities with `entity:get_sprite():set_animation("name_of_animation")`.

Solarus goes by 8x8 formatting and it would be best to make a grid for proper positioning. The sprite can be any size, but only the bounding box size can be changed for the entities "enemy" and "custom entity." The other entities bounding box is 16x16. A hero, NPC, etc.

You can make more animations and code them using the sprite drawable functions, but I will cover most of the default animations built into the engine. You can make any sword attack for the hero. As far as I know.

### Sword

You will need to create a file called "`sword1`" and it can contain the following animations.

Animations
spin_attack
super_spin_attack
sword
sword_loading_stopped
sword_loading_walking
sword_running
sword_tapping
victory

### Shield

It is the similar for the file "`shield1`".

Animations
stopped
walking
sword
sword_loading_stopped

## Animations

sword_loading_walking
sword_running
sword_tapping

## NPC

A NPC needs two animations:

## Animations

stopped
walking

## Enemy

You honestly only need animations “hurt” and “walking”.

## Animations

hurt
walking
immobilized
shaking

## Hero

The hero file is called, “[[tunic1](#)]“.

This is a list of most default hero animations, but there are more. You do not need them all. For example, the hero can still work without swimming, the spin attack can be disabled with code, and so on.

## Animations

boomerang1
boomerang2
bow
carrying_stopped
carrying_walking
falling
grabbing
hookshot
hurt
jumping
lifting
plunging_water

**Animations**

pulling  
pushing  
running  
spin\_attack  
super\_spin\_attack  
stopped  
swimming\_fast  
swimming\_slow  
swimming\_stopped  
sword  
sword\_loading\_stopped  
sword\_loading\_walking  
sword\_tapping  
throw  
walking  
walking\_with\_shield

## Destructible

**Animations**

destroy  
on\_ground  
stopped  
walking

## bomb

**Animations**

stopped  
stopped\_explosion\_soon  
walking  
walking\_explosion\_soon

## block

**Animation**

block

## chest

<b>Animations</b>
-------------------

closed
--------

open
------

explosion

<b>Animation</b>
------------------

explosion
-----------

shadow

<b>Animations</b>
-------------------

big
-----

small
-------

crystal\_block

<b>Animations</b>
-------------------

blue_lowered
--------------

blue_raised
-------------

orange_lowered
----------------

orange_raised
---------------

crystal

<b>Animations</b>
-------------------

blue_lowered
--------------

orange_lowered
----------------

switch

<b>Animations</b>
-------------------

activated
-----------

inactivated
-------------

arrow

<b>Animations</b>
-------------------

flying
--------

reached_obstacle
------------------

star

<b>Animation</b>	<b>Description</b>
normal	stars or other animation for spin attack

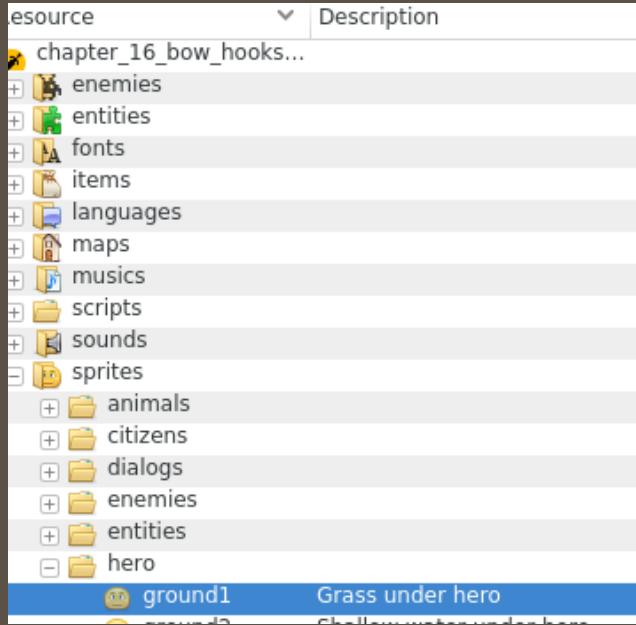
## ground1

### Animation

- Stopped
- walking

### Sound

walk\_on\_grass.ogg



Sprite ground1 ✖

**Sprite properties**

Sprite id: hero/ground1  
Description: Grass under hero

**Animation properties**

Default animation:  Set as default  
Source image: Castle [Edit] [Delete]

stopped (default) [Add] [Edit] [Delete]

- Direction 0
- walking

## ground2

### Animation

- Stopped

### Sound

walk\_on\_water.ogg

The screenshot shows the Solarus ARPG Game Development Book 2 editor interface. On the left is the Resource Manager tree view, which includes categories like chapter\_16\_bow\_hooks..., enemies, entities, fonts, items, languages, maps, musics, scripts, sounds, and sprites. Under sprites, there are folders for animals, citizens, dialogs, enemies, entities, and hero. The hero folder contains two entries: ground1 (Grass under hero) and ground2 (Shallow water under hero), both of which are currently selected.

The main right-hand panel is titled "Sprite properties". It displays the following details for the selected sprite:

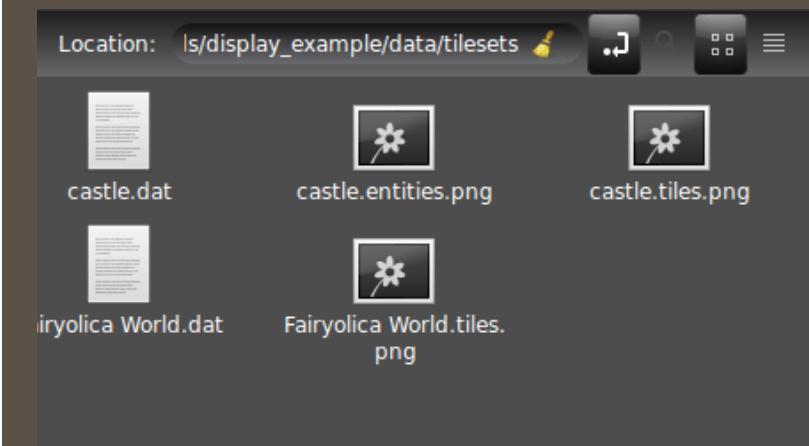
- Sprite id: hero/ground2
- Description: Shallow water under hero
- Animation properties section:
  - Default animation: walking (default)
  - Direction 0
  - Buttons: +, -, <, >, X
- Animation properties section:
  - Default animation: checked Set as default
  - Source image: Castle
  - Buttons: <-, >, Pencil

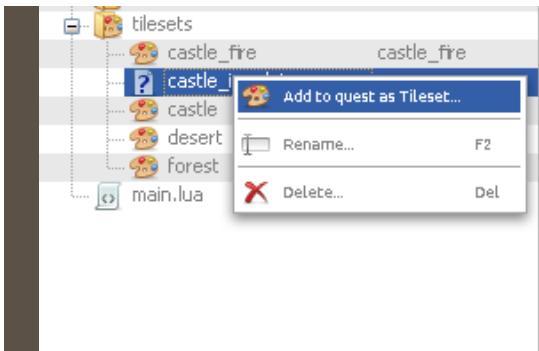
## Chapter 5: Using The Tileset Editor

### Resource Manager > Tileset > Tileset Editor

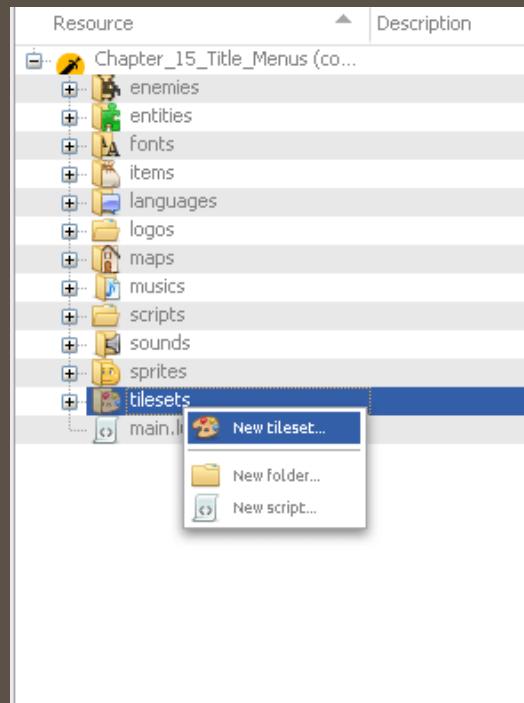
You can add a tileset by putting an image with the following extension `[file_name.tiles.png]` and making a `[file_name.dat]` in the tileset directory and add it by right clicking.

For example, `Fairyolica World.png` to `Fairyolica World.tiles.png`. You will need a `Fairyolica World.dat` as well.

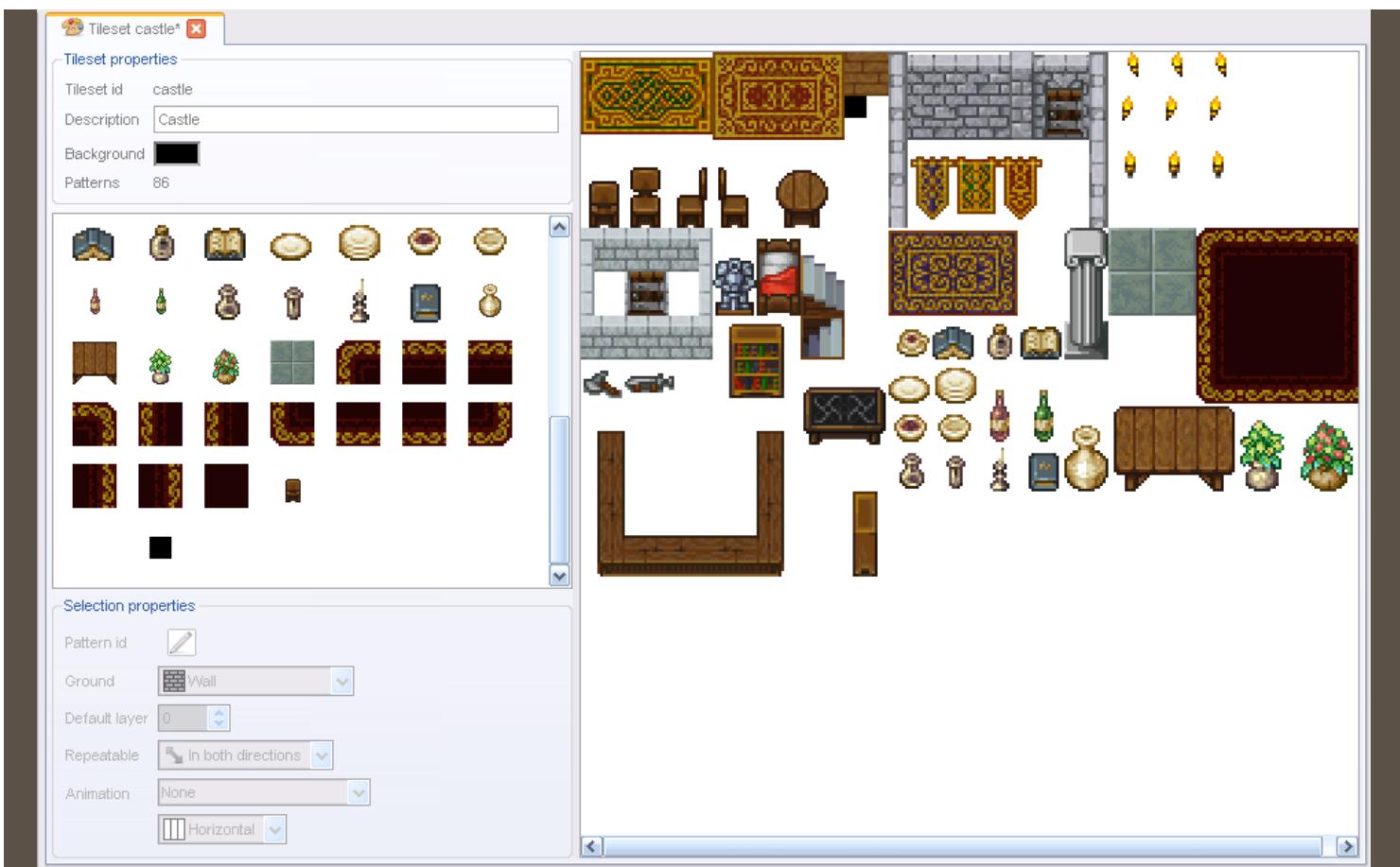




Also, one can make a `.dat` file or new tileset by right clicking the `tilesets` folder in the editor.



When one double clicks on a tileset, the tileset editor appears.



## Selection Properties

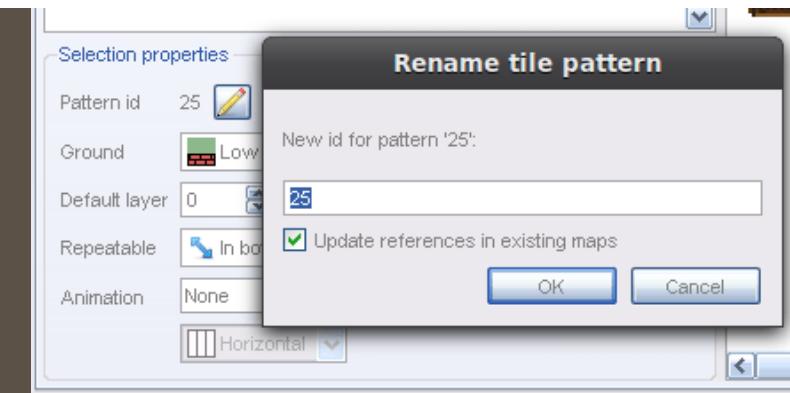
Properties appear in the **selection properties section** when one selects a pattern image id in a tileset.

**Selection properties**

- Pattern id: 25
- Ground: Low wall
- Default layer: 0
- Repeatable: In both directions
- Animation: None

## Rename ID

A pattern id can be changed by clicking the pencil icon:



A pattern ID is important because if you have two tilesets that are exactly the same except one is a different color, then you can simply give the same patterns to the other tileset and the map will be changed to that different color when you switch tilesets. Also, it is good with tasks that need certain tiles. For example, a shovel to only work on specific dirt tile.

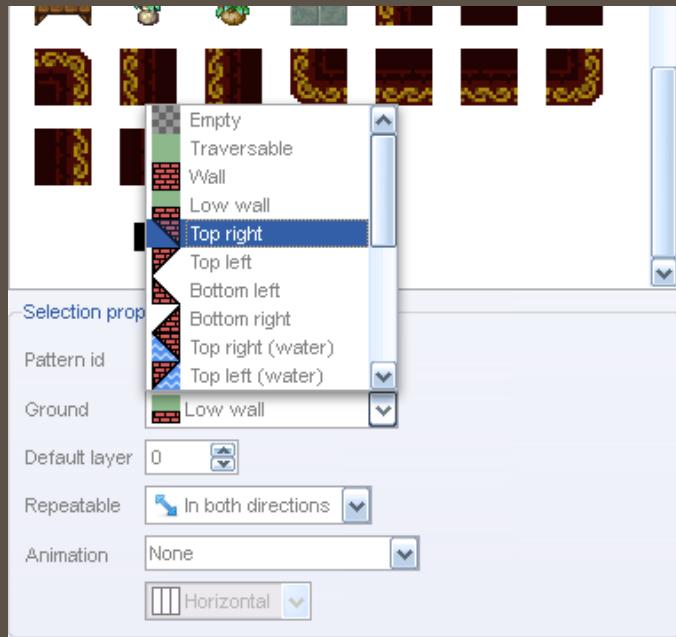
### Example:

A forest can become an instant desert once tilesets are changed. The patterns must be the same or similar enough for a decent change without much editing. This could be useful for different dimensions or terrain changes due to some cause.

You can use `map:set_tileset("tileset_name")` to change tilesets.

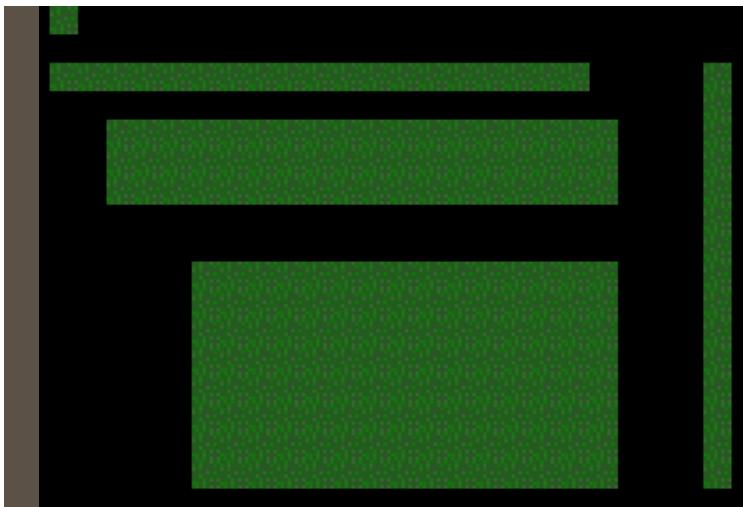
### Ground

You can select different grounds to fit your tile. For example, the tile would be `traversable` if the player character can walk on it.

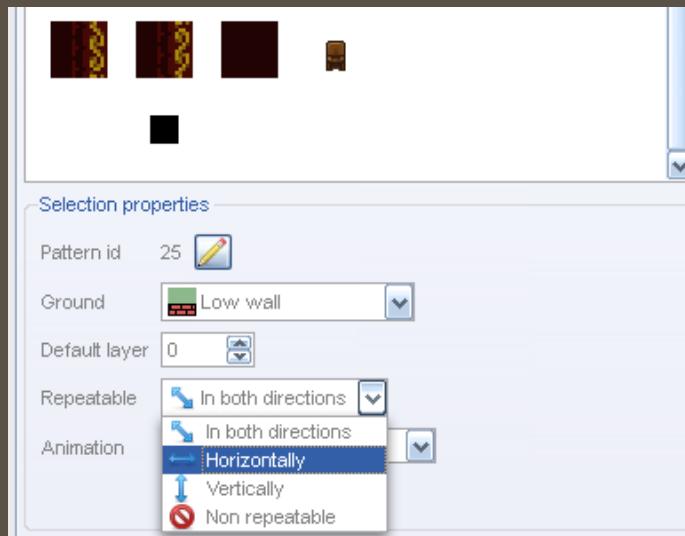


### Repeatable

A repeatable tile is normally a tile that looks the same while it is being duplicated or by being set right next to the same tile. Grass ground tiles normally look the same in games. They are repeated or duplicated over and over again.

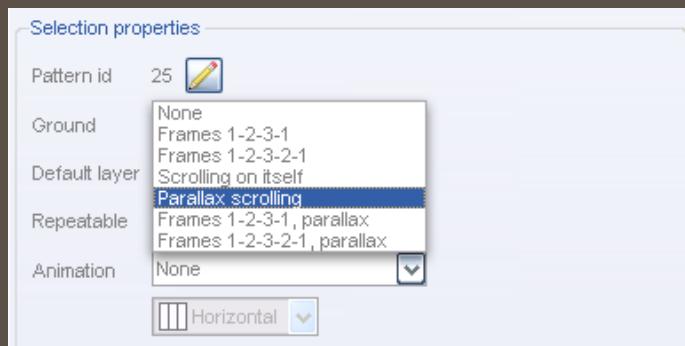


You can set a direction. For example, in the case that the tile only repeats in the horizontal direction.



## Animation

Animation can make moving flowers or waterfalls. You can set different frame patterns to fit your needs.



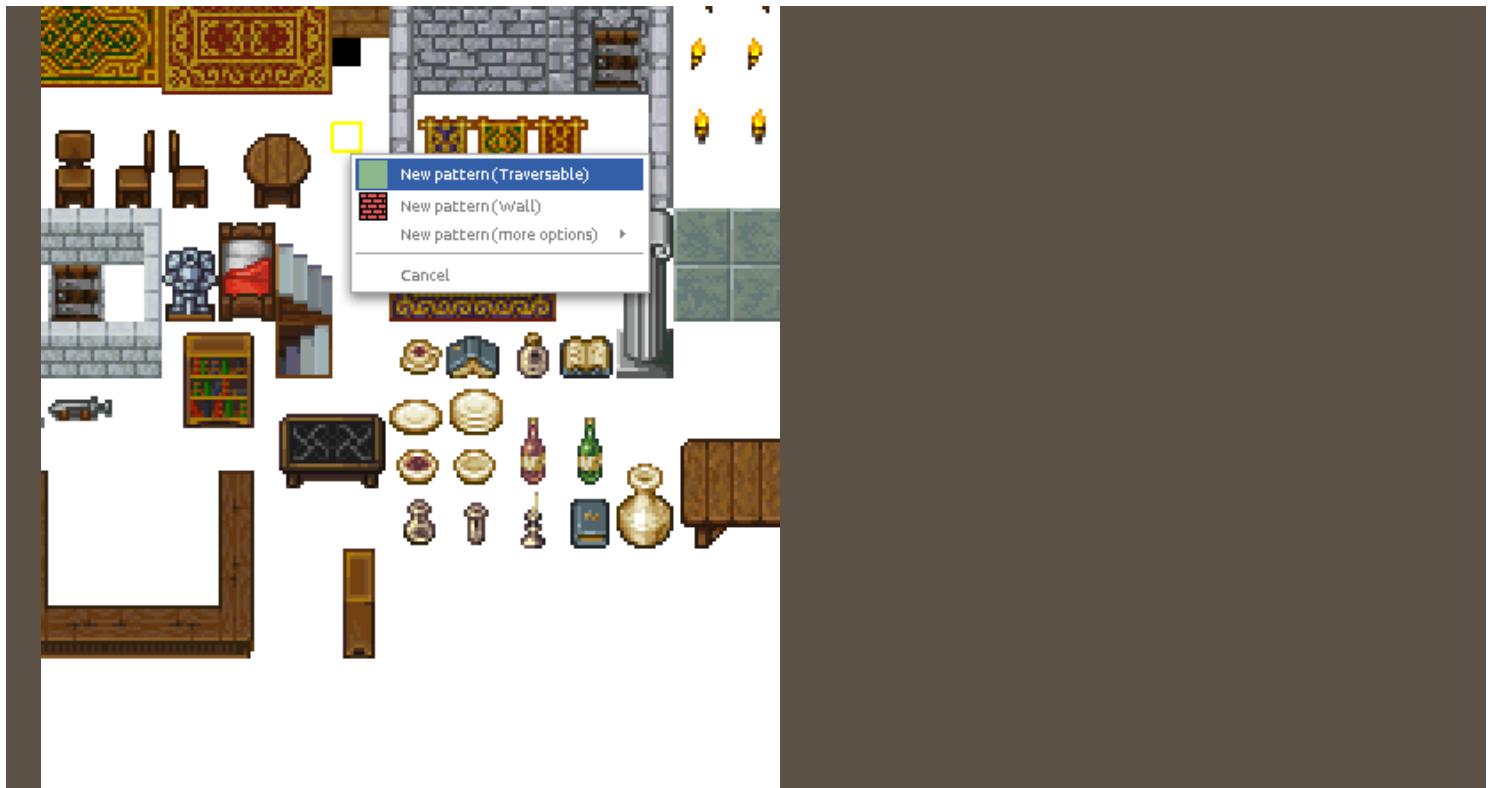
## Making Tile Patterns

The tilesets in Solarus follow an 8 x 8 pattern per block. That means you will have to set up your tileset in a grid to make it work properly.

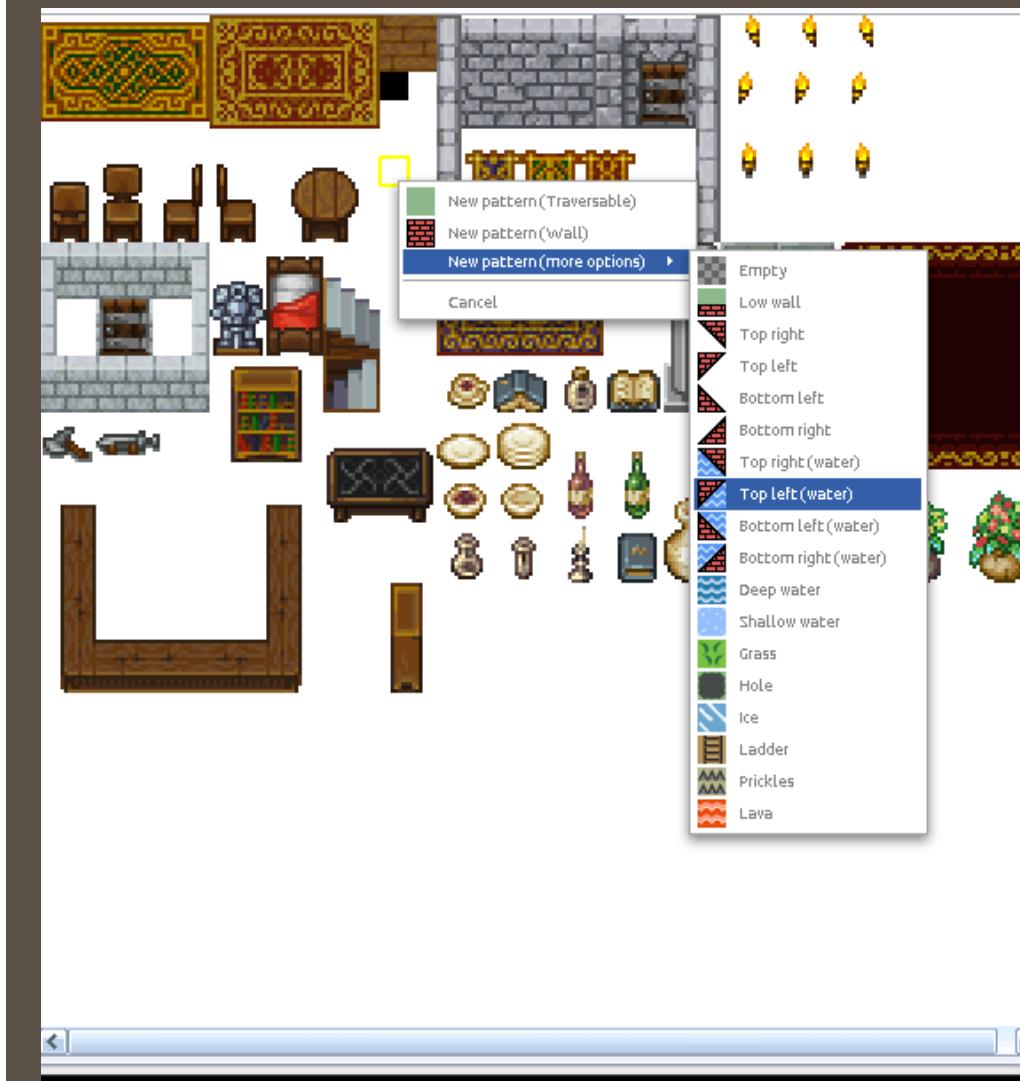
For example, if you overlay Fairyolica World tileset with an 8x8 grid, then you will see that mostly everything fits a 8x8 pattern.



To make a pattern you just click and drag to make a box.

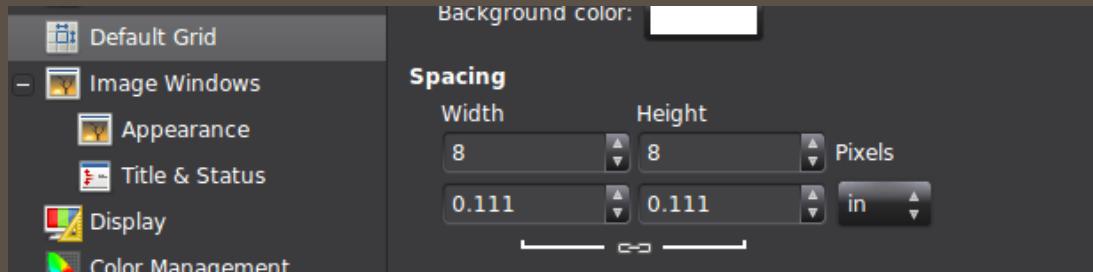


You can also select the pattern option for the tile.



## 8x8 Formatting

I will use Gimp as an example for 8x8 formatting. In [edit > preferences](#) menu go down to grid.



Now you may have to restart Gimp after saving your new grid preferences.

Afterward, go to [view > show grid](#), you will notice black lined boxes. Each box is an 8x8 block as you can see in the image below. In this example image each tile is in an 8x8 block.



Your graphics do not have to be in a single 8x8 block, but they will have to be in the same range. For example, the whirlpool and larger sand tile below. They are 16 x 16 blocks.



## Chapter 6: Very Basic Lua Scripting, Tutorial Point Lua PDF, ways to load script, and Lua console

### Very Basic Lua Scripting

The files for the very basic programming knowledge are added the Github.

[Lessons > Chapter\\_6 > Chapter\\_6\\_Lua\\_Quick\\_Basics](#)

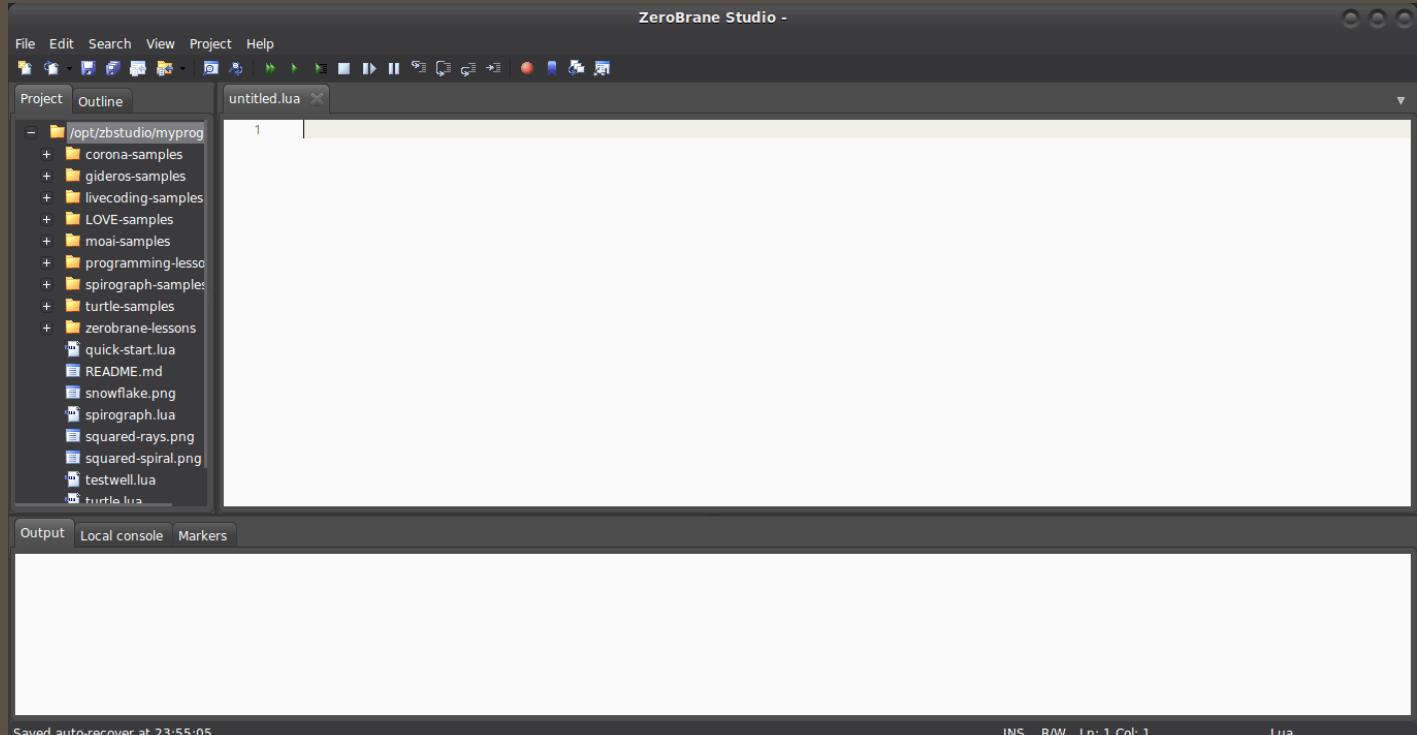
generated by haroopad

## ZeroBrane IDE

For the basic lessons I recommend [ZeroBrane IDE](#) or you can skip to "ways to load scripts in Solarus."

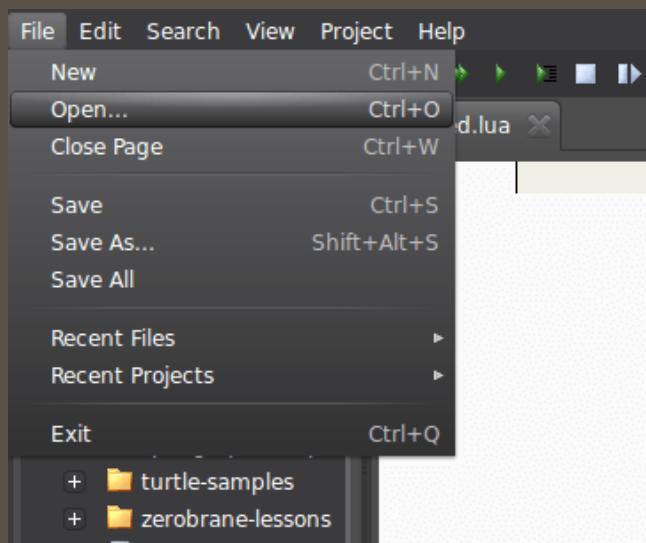
### Using ZeroBrane

1. Install and open ZeroBrane.

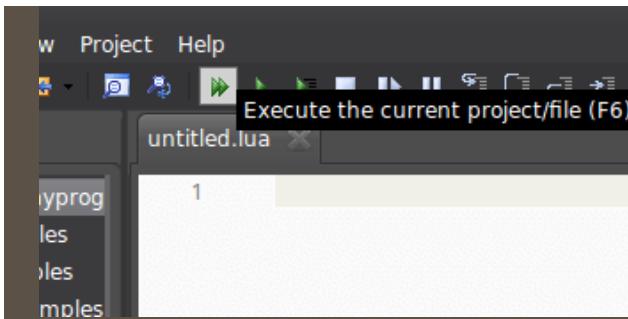


2. Open a Lua file or add the following code and save it somewhere.

```
--Hello_Solarus.lua
print("Hello Solarus!")
```



3. Click the first green arrow to compile and run the script. You can press the key **F6** as well.

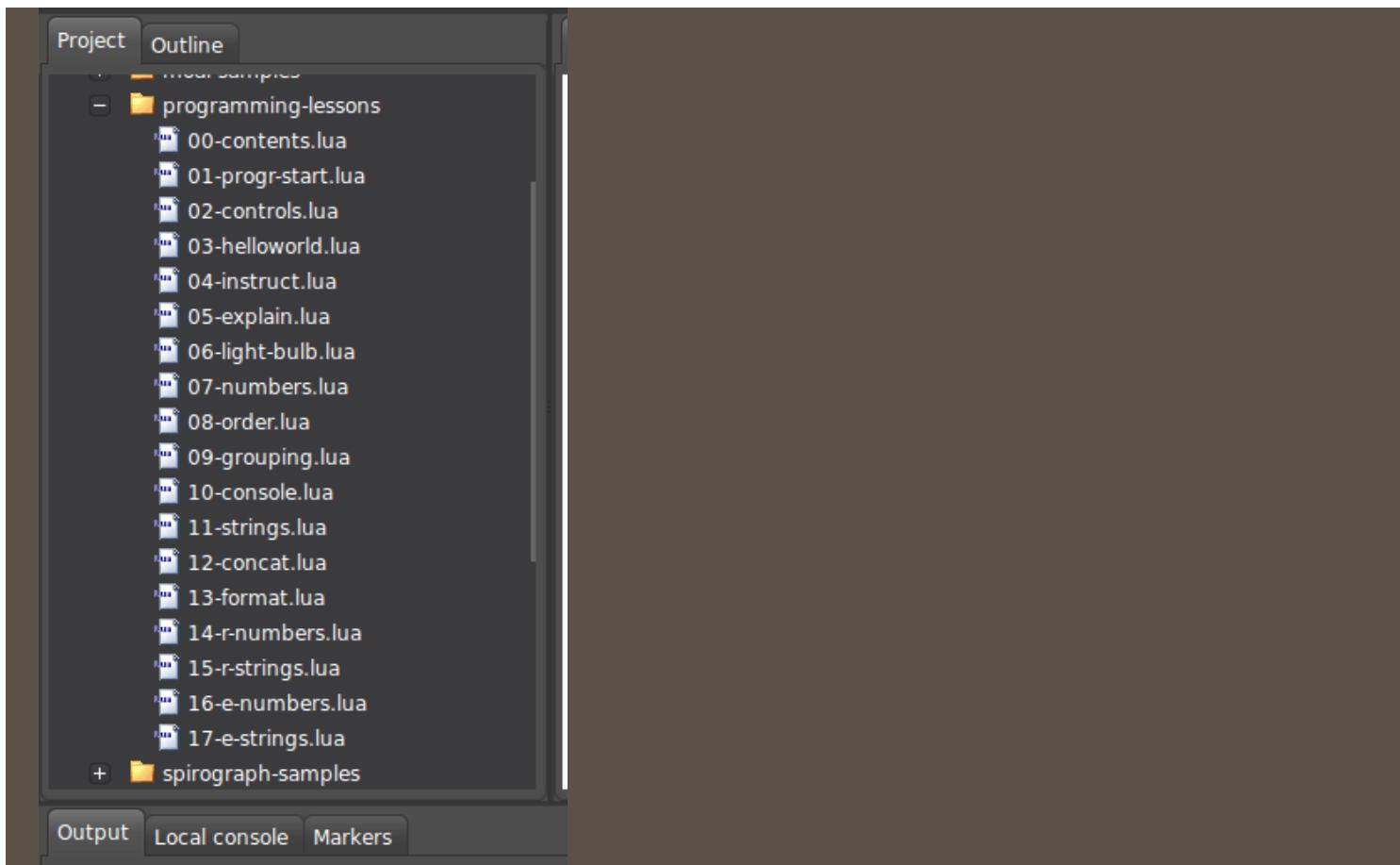


4. When you run the script with F6, the console at the bottom will show the text.

A screenshot of the Solarus IDE interface. On the left, the project tree shows 'myprog' with several sample folders and files like 'quick-start.lua', 'README.md', and 'snowflake.png'. In the center, a code editor window titled 'preview.lua' contains the line 'print("Knights of Sidonia!")'. At the bottom, the 'Output' tab is selected, showing the console output:

```
Program starting as '/opt/zbstudio/bin/linux/x86/lua' -e "io.stdout"
Program 'lua' started in '/opt/zbstudio/myprograms' (pid: 4550).
Knights of Sidonia!
Program completed in 0.05 seconds (pid: 4550).
```

5. ZeroBrane has a bunch of free basic lessons.



That is all!

## Whitespace

Blank lines and spaces that are ignored.

## Variables

Variables are just names that can be manipulated. Global variables are not marked with local or global.

Three types:

1. Local
2. Global
3. Tables can hold anything, but a nil.

```
name50 -- No local before it makes it global.

local name = 20

local name2 = 10

name + name2
```

## Identifiers

When making a variable name, there are some rules.

Unacceptable:

1. Punctuation characters: @#\$%

2. Cannot start with a number

3. No spaces between variables. EX: `local the variable` should be: `local the\_variable`.

Acceptable:

1. A - Z EX: SHE
2. a - z EX: she
3. Underscore followed by numbers and letters \_8Wn
4. Case-sensitive EX: She and she are different

Examples of the acceptable identifiers:

yodz	zata	abcd	number_five	z_456
sefra34	_ten	x	f2359y2	SpiritBlast

## Keywords

Reserved words in Lua. You cannot name them as variables.

for	break	false	true	and	or	not	
in	else	elseif	local	if	nil	repeat	
return	function	while	end	then	until	do	

## Data Types

Type:	About
Number:	Represents real numbers.
String:	Represents array of characters.
Boolean:	True and false as values. Used for checking a condition.
Nil:	Has no data.
Function:	A method and most of the time it does a task for you. EX: Drawing images.

```
print(type(true))           ----> boolean
print(type(2.1*z))          ----> number
print(type(nil))            ----> nil
print(type(type(XYZ)))      ----> string
print(type(print))          ----> function
print(type(false))          ----> boolean
print(type(type))           ----> function
```

## Relational Operators

Operator:	About
==	Checks if they are equal.

<b>Operator:</b>	<b>About</b>
<code>~=</code>	Checks if they are not equal
<code>&gt;</code>	Checks if one is greater than the other
<code>&lt;</code>	Checks if one is less than the other
<code>&gt;=</code>	Checks if one is greater or equal to each other
<code>&lt;=</code>	Checks if one is less or equal to each other

## Logical Operators

<b>Operator:</b>	<b>About</b>
<code>and</code>	If both are true, then it activates.
<code>or</code>	If one of them are true, the it activates.
<code>not</code>	If it is true, then it will be false.

## Arithmetic Operators

<b>Operator</b>	<b>Math</b>	<b>Examples</b>
<code>+</code>	Adds	Three + three = 6
<code>-</code>	Subtracts	Three - three = 0
<code>*</code>	Multiplies	Three * three = 9
<code>/</code>	Divides	Three / three = 1
<code>%</code>	Remainder	Three % five = 2
<code>^</code>	Exponent	three^2 = 9
<code>-</code>	Can act as a negative	-Three * three = -9

## Other Operators

<b>Misc:</b>	<b>About</b>
<code>..</code>	Concatenates or combines text/string. aa..5 = aa5
<code>#</code>	Length of text/string. #text = 4 ("text" is equal to 4 letters.)

## Escape Sequences

<b>Escape</b>	<b>Task</b>
<code>\n</code>	New line
<code>\t</code>	Tab
<code>\'</code>	Single quotes
<code>\"</code>	Double quotes
<code>\</code>	Backslash
<code>\b</code>	Backspace
<code>\f</code>	Formfeed

Escape	Task
\a	Bell
\r	Carriage return
\v	Vertical tab
[	Left square bracket
]	Right square bracket

## Repetitions Pattern Modifiers

Modifier	About
+	1 or more repetitions
*	0 or more repetitions
-	also 0 or more repetitions
?	optional (0 or 1 occurrence)

## Character Pattern Classes

Pattern	About
.	all characters
%oa	letters
%oc	control characters
%od	digits
%ol	lower case letters
%op	punctuation characters
%os	space characters
%ou	upper case letters
%ow	alphanumeric characters
%ox	hexadecimal digits
%oz	the character with representation 0

## Common Variable Naming Rules

This is not a Lua naming rule and one can completely ignore this because everyone can program the way they want, but this will help people read your code. This is standard in most programming languages.

### Clear Variable Name

Variable `yellow` is easier to understand than `var2`.

### Variable Length

The variable name can be of any length, but do not get make too long. One will have to type this name.

```
local the_bunny_crossed_the_road_at_street_number = 50
--The street number that the bunny crosses the road on.
local bunny_crossed = 50
```

## UPPERCASE vs lowercase Variables

Constants are normally uppercased. This variable will always stay the same.

```
local SPEED = 50
```

Deciding on StudlyCaps vs. camelCase vs SCREAMING\_CAPS is worth maybe 90 seconds discussion with a programming team one is working with, but I always see ALL\_CAPS.

Lowercase variables are mostly normal variables.

```
local color = "yellow"
color = "green"
```

# Lua Programming

## Comments

```
--[[
Lua is a very simple programming language, but very powerful.
This is a quick start for the "very" basics of Lua and more will be added later based on examples
```

Read this book for more information and detail.  
[https://www.tutorialspoint.com/lua/lua\\_tutorial.pdf](https://www.tutorialspoint.com/lua/lua_tutorial.pdf)

Comments:

You can make comments in a few ways in Lua. These are ignored by the code and one uses them to help you.

Single comment:

You must put "--" before the text.

Example:

```
--Hello
```

Multiple line comments:

You must use two dashes and opening plus closing brackets. -- [ [ ] ]

--Example:

```
--[[
Write all the text you want between here.
--]]
```

## Print Text

```
--[[
Lua is a very simple programming language, but very powerful.
This is a quick start for the "very" basics of Lua and more will be added later based on examples
```

Read this book for more information and detail.

generated by haroopad

```
https://www.tutorialspoint.com/lua/lua\_tutorial.pdf
--]

--Ways to print text in the terminal. Printing text is very useful for testing your code.

print "Hello Solarus World! - Method 1 without ()"

print ("Hello Solarus World! - Method 2 with ()")
```

## Declaring variables

```
--[]

Lua is a very simple programming language, but very powerful.
This is a quick start for the "very" basics of Lua and more will be added later based on examples

Read this book for more information and detail.
https://www.tutorialspoint.com/lua/lua\_tutorial.pdf

--]

--Declaring a variable. A variable is just storing a name. They can be global or local. One will r

--Pie is local because it is not everywhere on earth
local pie = 0

--Air is global because it is everywhere on earth.
air = 0

--String is basically text in double quotes.
local pie_type = "pumpkin pie."

--For testing, it is good to print a value or string to know it is working. A simple comma after t
print ("My favorite pie is", pie_type)

--Concatenation is used to combine. A simple "..
print ("My favorite pie is ..pie_type")

--Value is basically numbers.
local pie_number = 5

--Separate two different pieces of text and a single variable.
print ("I have", pie_number, "pies.")

--Concatenation is used to combine. A simple "..
print ("I have "..pie_number.." pies.")

--adding variables or names
local add = 5
local pie = 3

print (pie + add)
```

## Tables

```
--[]

Lua is a very simple programming language, but very powerful.
This is a quick start for the "very" basics of Lua and more will be added later based on examples

Read this book for more information and detail.
https://www.tutorialspoint.com/lua/lua\_tutorial.pdf
```

```
--]]

--Tables. You need tables if you plan to have a many variables or you will get an upvalue error. T

local muffin = {
    chocolate,
    blueberry,
}

muffin.chocolate = 5

muffin.blueberry = 2

print ("I have "..muffin.chocolate.." chocolate muffins and "..muffin.blueberry.." blueberry muffi
```

## Arrays

```
--[]

Lua is a very simple programming language, but very powerful.
This is a quick start for the "very" basics of Lua and more will be added later based on examples

Read this book for more information and detail.
https://www.tutorialspoint.com/lua/lua\_tutorial.pdf

--]]

--Arrays and for loops.
--Arrays are not scary. They help with repetition. They can be used in tables too.

--Declare an array.
local array = {} -- that is it.
local pie = {} -- Simple to declare

-- All you have to do with arrays is to add a number in square brackets after each new variable.
pie[1] = "Chocolate"
pie[2] = "Apple"

--Easier than doing this
chocolate_pie = "Chocolate"
apple_pie = "Apple"

--Using arrays to print many variables with a "for loop"
--rep stands for repetition. You do not have to use rep. 1,2 is basically 1 through 2, so only pri
for rep = 1,2 do
    print(pie[rep])
end

--You can use variable names for this as well. 1,2
pie[4] = 1
pie[5] = 2

for rep = pie[4],pie[5] do
    print(pie[rep])
end

--Making variables equal to a variable. You need this with calculations or you will get a "nil erri
for rep = 1, 10 do
    array[rep] = 0
    print("Array "..rep..":"..array[rep])
end

--Math in lua is pretty simple.
for rep = 11, 21 do
```

```
array[rep] = 5
print("Array "..rep..".."..array[rep] + 5 * rep - 10 / 2)
end

--A string in the square brackets of an array. This can be good for an inventory.
local cake = {}

cake["chocolate"] = 5

print (cake["chocolate"])

--Increment
cake["chocolate"] = cake["chocolate"] + 1

print (cake["chocolate"])

--Decrement (Declare again or it will be 5)
cake["chocolate"] = 5

cake["chocolate"] = cake["chocolate"] - 1

print (cake["chocolate"])

--Multidimensional array
--This is array type is important in cases where one wants to keep the code basically the same and
--Lets pretend you want 3 rows for each array number block.

-- Initializing the array
row = {}

-- Initializing the multidimensional array
for rep_d1=1,3 do
    row[rep_d1] = {}
    for rep_d2=1,3 do
        row[rep_d1][rep_d2] = 0
    end
end

--row 1
row[1][1] = 5
row[1][2] = 3
row[1][3] = 2

for rep = 1,3 do
    print("Row 1: "..row[1][rep])
end

--row 2
row[2][1] = 7 + 2
row[2][2] = 3 * 4
row[2][3] = 1 - 3

for rep = 1,3 do
    print("Row 2: "..row[2][rep])
end

--row 3
row[3][1] = "pie"
row[3][2] = "cake"
row[3][3] = "muffin"

for rep = 1,3 do
    print("Row 3: "..row[3][rep])
end
```

## If Statement & Operators

```
--[]  
Lua is a very simple programming language, but very powerful.  
This is a quick start for the "very" basics of Lua and more will be added later based on examples  
  
Read this book for more information and detail.  
https://www.tutorialspoint.com/lua/lua\_tutorial.pdf  
--]  
  
--if statement  
local frog = 4  
  
-- use "==" instead of "=". You use "==" for reach points.  
if frog == 4 then  
    print("You do not see ..frog.." frogs everyday!")  
end  
  
--if else statement  
local frog = 5  
  
if frog == 4 then  
    print("You do not see ..frog.." frogs everyday!")  
else  
    print("I see no frog.")  
end  
  
--Greater than  
local frog = 10  
local bat = 9  
  
if frog > bat then  
    print("I see more frogs.")  
end  
  
--Less than  
local frog = 10  
local bat = 11  
  
if frog < bat then  
    print("I see more bats.")  
end  
  
--Greater than or equal to  
--Logical Operators: and  
local frog = 10  
local bat = 9  
  
if frog > bat and frog >= 11 then  
    print("I see 10 frogs.")  
else  
    print("I do not see 11 frogs.")  
end  
  
--Less than or equal to  
--Logical Operators: and  
local frog = 10  
local bat = 9  
  
if frog > bat and frog <= 11 then  
    print("I see 10 frogs.")  
else
```

```
    print("I do not see 11 frogs.")
end

--if does not equal ~=
local frog = 3

if frog ~= 2 then
    print("There are more than 10 frogs!")
end

--Logical Operators: or
--As long as one is true, then it will be true.
--As long as one of the variables are equal to something other than "nil"
local a = 5
local z = nil

if a or z then
    print("It is true." )
end

local a = 5
local z = 1

if a or z then
    print("It is true." )
end

--Logical Operators: not
--Remember a and z are equal to a value. That names them true, but "not" will make that true into
--Changing the values.
local a = 1
local z = 1

if not (a and z) then
    print("It is true." )
else
    print("It is false." )
end

local a = 1
local z = nil

if not (a and z) then
    print("It is true." )
else
    print("It is false." )
end

--elseif statement
local frog = 3

if frog ~= 3 then
    print("There are more than 10 frogs!")
elseif frog < 2 then
    print("There are 2 frogs!")
elseif frog == 3 then
    print("There are 3 frogs!")
end

--Nested if statement. That means an if statement inside an if statement
--Example 1
local a = 10
local b = 5

--If 'a' is greater than 'b'
```

```

if a > b then
--if 'a' is greater than 9
  if a > 9 then
    print ("a is greater than 9!")
  end
end

--Example 2
local a = 8
local b = 5

--If 'a' is greater than 'b'
if a > b then
--if 'a' is greater than 9
  if a > 9 then
    print ("a is greater than 9!")
  else
    print ("a is less than 9!")
  end
end

```

## Goto Statement

```

--Use goto as a nested break statement
local a = 1
local b = 6

for z=1,10 do
  if a < b then
    a = a + 1
    print("a is less than b")
  else
    goto done
  end
end
::done::

```

## Loops Types

```

--While loop: Repeats a statement a given condition is true.

local z = 7

while( z < 18 )
do
  print("value of z:", z)
  z = z*3
end

print("End of while loop")

--For loop: Executes statements multiple times simplifies the code

--[[
for init or start value,max or min value, increment or decrement
do
  statement(s)
end
--]]
for start = 10,1,-1 do

```

```

print(start)
end

print("End of for loop part 1")

for start = 1,10 do
    print(start)
end

print("End of for loop part 2")

--Repeat...until loop: Repeats till the until condition is met.

local z = 3

repeat
    print("value of z:", z)
    z = z + 5
until( z > 15 )

print("End of Repeat...until loop")

```

--Nested loops: You can use one or more loop inside any another while, for or do..while loop.

```

for i=1,10 do
    for s = 1,2 do
        print("Nested loop.")
    end
end

print("End of nested loop")

```

--break statement: This stops a loop.

```

for i=1,10 do
    for s = 1,2 do
        print("break loop.")
        break
        print("Muffin loop.")

    end
end

```

```
print("End of break loop")
```

--The Infinite Loop: Keeps going forever. Normally causes a crash.

```

--[[
while( true )
do
    print("Print this F-O-R-E-V-E-R.")
end
--]]

```

## Math

```
--[[
Math Library List:
```

Check the following link for more information:

generated by haroopad

<http://lua-users.org/wiki/MathLibraryTutorial>

```
math.acos
math.cos
math.asin
math.sin
math.atan
math.tan
math.ceil
math.deg
math.exp
math.fmod
math.huge
math.log
math.min
math.mininteger
math.max
math.maxinteger
math.modf
math.random
math.randomseed
math.abs
math.sqrt
math.pi
math.rad
math.floor
math.tointeger
math.type
math.ult

--]]

print("")

--math.abs

--Return the absolute value. Basically, the non-negative value.
print("The absolute vale is: "..math.abs(-50))

print("The absolute vale is: "..math.abs(19.37))

print("The absolute vale is: "..math.abs(0))

print("")

--math.deg

--Angle degrees
print("The angle degree is: "..math.deg(math.pi * 2))
print("The angle degree is: "..math.deg(math.pi))
print("The angle degree is: "..math.deg(math.pi / 2))
print("The angle degree is: "..math.deg(math.pi / 4))

print("")

--math.min , math.max

--Minimum or maximum value
print("The min is: "..math.min(3,6))
print("The min is: "..math.min(2.2, 3.6, 8))
print("The min is: "..math.min(1.4, -6, 4))
print("The max is: "..math.max(1.9, -12, 4))
print("The max is: "..math.max(1.22, 5, 2))
```

```
print("")  
  
--math.sqrt  
  
--Square root of a number. Only non-negative values are allowed.  
print("The square root is: "..math.sqrt(25))  
print("The square root is: "..math.sqrt(9))  
print("The square root is: "..math.sqrt(200))  
  
print("")  
  
--math.random  
  
--math.random() Generates a number between 0 and 1.  
  
print("The random number is: "..math.random())  
print("The random number is: "..math.random())  
print("The random number is: "..math.random())  
  
print("")  
  
--math.random(upper) generates a number between 1 and upper.  
  
print("The random upper number is: "..math.random(120))  
print("The random upper number is: "..math.random(110))  
  
print("")  
  
--math.random(lower, upper) generates numbers between lower and upper.  
  
print("The random lower to upper number is: "..math.random(75,80))  
print("The random lower to upper number is: "..math.random(81,85))  
  
print("")  
  
--math.randomseed  
--The "seed" is a starting point. Basically, you will always get the same random numbers no matter what seed you choose.  
  
math.randomseed(12)  
  
print("The randomseed number is: "..math.random(15))  
print("The randomseed number is: "..math.random(17))  
print("The randomseed number is: "..math.random(19))  
print("The randomseed number is: "..math.random(22))  
  
print("")  
  
--math.pi  
  
print("pi is: "..math.pi)  
  
print("")  
  
--math.floor is for rounding down and math.ceil is for rounding up  
print ("1.5 rounded down: "..math.floor(1.5))  
print ("1.5 rounded up: "..math.ceil(1.5))  
  
print ("-1.5 rounded down: "..math.floor(-1.5))  
print ("-1.5 rounded up: "..math.ceil(-1.5))
```

## Strings

generated by haroopad

```
--String Examples

--Escape Sequence example:
print ("What do you want\n with\r me\n\t? \nYou \"mushroom\" \'woman\' mutated monster!")

--Replacing a string

local name = "Zeta Ataria"

print("Her old name was "..name)

-- replacing strings
local replace_name = string.gsub(name,"Ataria","Setrita")
print("Her new name is",replace_name)

--Case Manipulation

print(string.lower(replace_name))
print(string.upper(replace_name))

-- Length of string
print("Length of replace_name is ",string.len(replace_name))

-- String Concatenations with ..
print("Concatenated:",replace_name..name)

-- Repeating strings
local repeating = string.rep(replace_name,4)
print(repeating)

--Formatting Strings
local name = "zeta"
local rules = "rules"

-- Basic string formatting
print(string.format("Basic %s format %s",name,rules))

-- Decimal formatting
print(string.format("%.0f",1/3))
print(string.format("%.1f",1/3))
print(string.format("%.2f",1/3))
print(string.format("%.3f",1/3))
print(string.format("%.4f",1/3))
print(string.format("%.5f",1/3))

-- Date format + wacky
local month = 2; local day = 1; local year = 2014; local wacky = 2
print(string.format("Date %02d/%02d/%03d/%09d", month, day, year,wacky))

--Find and Reverse

local name = "zerta galxeria"

print("Her name was:"..name)

print(string.find(name,"zerta"))
reverse_text = string.reverse(name)
print("The new name is now:"..reverse_text)
```

```
--Calculate the length with #
print ("The length is: ..#"Length")
```

## tonumber()

```
--A string of numbers. These would need to be converted in order to do math.
local string = "5678"

local number = tonumber(string)

print("End of tonumber(): ..number - 178)
```

## string.format()

```
local number = 3300

--converts variable 'number' into a string with 4 digits.
--Digit about can be changed. %0(number)d EX: %09d
local number_string = string.format("%07d", number)

print("End of string.format(): ..number_string)
```

## string.len() or :len()

```
local string = "What"

print("The string length is: ..string.len(string))
print("The string length is: ..string:len())
```

## string.reverse() or :reverse()

```
local string = "Programming"

print("Reversing programming: ..string.reverse(string))
print("Reversing programming: ..string:reverse())
```

## string.sub() or :sub()

```
--Print 7 until the end. "Print " = 6 characters (with space), so they are not included
print(string.sub("Print seven characters", 7))

--Print 7 until 9. "Print " = 6 characters (with space), so they are not included
print(string.sub("Print seven characters", 7, 9))

--Print -7 until the end. "Print " = 6 characters (with space), so they are not included
print(string.sub("Print seven characters", -7))

--Print -7 until -9. "Print " = 6 characters (with space), so they are not included
print(string.sub("Print seven characters", 7, -9))
```

## string.gmatch(string, pattern) or string:gmatch(pattern)

generated by haroopad

```

local word

--"%a" is a "letters" character class. Check at the start of the programming lesson for all of the
for word in string.gmatch("Hello Lua user", "%a+") do
    print("%a+: "..word)
end

for word in string.gmatch("Hello Lua user", "%a") do
    print("%a: "..word)
end

for word in string.gmatch("Hello Lua user", "%a*") do
    print("%a*: "..word)
end

for word in string.gmatch("Hello Lua user", "%a?") do
    print("%a?: "..word)
end

--[[

Repetitions pattern Modifiers:

+ 1 or more repetitions
* 0 or more repetitions
- also 0 or more repetitions
? Optional (0 or 1 occurrence)

--]]]

--[[

Character pattern classes:

. all characters
%a letters
%c control characters
%d digits
%l lower case letters
%p punctuation characters
%s space characters
%u upper case letters
%w alphanumeric characters
%x hexadecimal digits
%z the character with representation 0
--]]]

```

## Clear Table

```

--Makes a table with 2 and 5 in it.
local vars = {"2", "5"}

--prints the table above
print("Table before clearing: "..vars[1]..vars[2])

--clears the table and this makes vars[2] = nil
vars = {}

vars[1] = 6

print("Table after clearing: "..vars[1], vars[2])

```

## Math/Arithmetic in an if statement

```

local test = 2
local test2 = 1
local limit = 5

-- If variable 'test' added to variable 'test 2' are less than limit, then print, "Cannot go beyond limit"
if test + test2 < limit then
    print("If/math:Cannot go beyond limit!")
end

local test = 4
local test2 = 2
local limit = 5

-- If variable 'test' added to variable 'test 2' are greater than limit, then print, "Going beyond limit"
if test + test2 > limit then
    print("If/math:Going beyond the limit!")
end

```

## Simple table.concat()

```

local char = {}

char[3] = "g"
char[2] = "p"
char[1] = "h"

--adds the char in the table together.
local foo = table.concat(char)

print("Table concat: "..foo)

```

## table.concat() and table.insert()

```

local multiple = {}
local character_num = {}
local character = {}
local char = {}

local j = 1

multiple[1] = true

for i = 1,15 do
    char[i] = "q"
    char[15] = "g"

    --Variable j = 1 above and multiple[1] is true above, so it activates.
    if multiple[j] == true then

        --inserts 'q' into 'character_num' until char[15] because it equals 'g'
        table.insert(character_num, char[i])

        --concat combines each 'q' from the array table
        character[i] = table.concat(character_num)
    end
end

```

```
--print each character concat line
print("Concat insert: "..character[i])

end
end
```

## table.sort

```
local test_table = {"b", "a", "c", "e", "d"}

-- Sort the table from a to z.
table.sort(test_table)

for i = 1,5 do
    print(test_table[i])
end

print("")

--sort the table from z to a.
local test_table2 = {"b", "a", "c", "e", "d"}

table.sort(test_table2, function(a, b) return a > b end) -- Sort our table, but this time lets make it random

for i = 1,5 do
    print(test_table2[i])
end

print("")
```

## Defining a function

The `local` scope, arguments, and return are optional for functions.

```
--[[
scope_optional function name( arg1, arg2, arg3....., arg[num])
    function_body
return something
end
--]]]

--If number1 is less number 2 then 1 will be added to number 2
--local scope is optional. It will be global without it.
local function increase(number1, number2)

    if number1 < number2 then
        result = number2 + 1
    end

    --Result is what is printed.
    return result;
```

```

end

print ("The result: "..increase(2,3))

print("")

--You do not have to return it in this case.
function thetruth()
    print("You were a mutant!")
end

print(thetruth())

```

Functions can be declared in a straight line, in a table, and assigned to variables.

```

x = {pos_x = function(x,y) return x end, pos_y = function(x,y) return y end, }

print("Dot: ",x.pos_x(3,4))
print("Dot: ",x.pos_y(3,4))

```

Result:

```

Dot: 3
Dot: 4

```

## pairs() and ipairs()

```

--for key, variable in pairs() (no particular order)
--for key, variable in ipairs() (in order)
--pairs() and ipairs() loop through a table

local itemList = {
    {bName = "Candy ", bCountry = "Gestra ", bType = "50HP"}, 
    {bName = "Chocolate ", bCountry = "Fragrath ", bType = "100HP"}, 
    {bName = "Sword ", bCountry = "Nerzarta ", bType = "90ATK"}, 
    {bName = "Shield ", bCountry = "kelboax ", bType = "100DEF"}, 
}
local sort_func = function( a,b ) return a.bName < b.bName end
table.sort( itemList, sort_func )

for i, record in ipairs( itemList ) do
    print(record.bName..record.bCountry..record.bType)
end

```

## Associative table

```

--[[
Sorting an Associative table - not possible.
You can only sort a table of keys which has a number index
]]

Assosciative = {muffin = "strawberry", tree = "oak",
               cake = "oreo", seed = "apple",
               pie = "chocolate", fruit = "orange"}

print ("\nMethod 1 - sort an array of keys")

list = {}

```

generated by haroopad

```

for name,value in pairs(Assosciative) do
    list[#list+1] = name
end

print ("*****by key")

table.sort(list)

for key=1,#list do
    print (list[key] .. " is " .. Assosciative[list[key]])
end

print ("*****by value")

function byvalue(a,b)
    return Assosciative[a] < Assosciative[b]
end

table.sort(list,byvalue)

for k=1,#list do
    print (list[k] .. " is " .. Assosciative[list[k]])
end

print ("\nMethod 2 - create an array of pairs")

arraypairs = {}

for name,value in pairs(Assosciative) do
    table.insert(arraypairs,{name=name, value=value})
end

table.sort(arraypairs,function(a,b) return a.name < b.name end)

--The variable consisting of only an underscore "_" is commonly used as a placeholder when you want
for _,line in ipairs(arraypairs) do
    print (line.name .. " is " .. line.value)
end

```

## Error Handling

```

--Common errors people make in lua

--Forgetting do
for rep = 1,3
    print(rep)
end

--13_error_handling.lua:6: 'do' expected near 'print'

--Forgetting to assign a number value to be. Be should equal at least zero. EX local b = 0
local b

local c = 5

print(b + c)

--13_error_handling.lua:20: attempt to perform arithmetic on local 'b' (a nil value)

```

```
--This should be a single equal. "="
local d == 5

print(d)

--13_error_handling.lua:25: unexpected symbol near '=='


--For getting to add "then"
local b = 5

if b == 5
    print(5)
end

--13_error_handling.lua:38: 'then' expected near 'print'
```

--[[  
Some other common errors. That happen when people do not use tables and arrays.

1. Function at line has more than 60 upvalues for local variables, arrays, & 200 Local Variables I
2. Control structure too long near  
--]]

## Declaring Multiple Variables On One Line

```
--One can use a comma to declare many variables at one time.
local a,b,c,d = 5,6,7,8

print(a)
print(b)
print(c)
print(d)

print(a + b + c + d)
```

## COROUTINE

A coroutine can stop (yield) and restart (resume) a task.

### Make a Coroutine

It is quite simple to create a coroutine.

```
variable = coroutine.create(function()
--code here
end) )
```

### COROUTINE YIELD

The `coroutine.yield()` function stops the coroutine.

```
co = coroutine.create(function()
    for i = 1, 5 do
        print(i)
        coroutine.yield()
```

```
    end
end)
```

## C coroutine Resume

The function `[coroutine.resume()]` needs to be called to start a coroutine.

```
co = coroutine.create(function()
    for i = 1, 5 do
        print(i)
        coroutine.yield()
    end
end)

coroutine.resume(co)
coroutine.resume(co)
```

Result:

```
1
2
```

## C Tasks Between Resume

One can do other things between `[coroutine.resume()]`.

```
--[[
co = coroutine.create(function()
    for i = 1, 5 do
        print(i)
        coroutine.yield()
    end
end)

coroutine.resume(co)
print("do something")
coroutine.resume(co)
print("do something")
coroutine.resume(co)
print("do something")
```

Result:

```
1
do something
2
do something
3
do something
```

## C Coroutine Status

The function `[coroutine.status()]` tells you if the coroutine is dead, running, and suspended. A coroutine dies if it exceeds the limit. In this case, the limit is 5 times according to the for loop.

```
co = coroutine.create(function()
    for i = 1, 5 do
        print(i)
        coroutine.yield()
    print(coroutine.resume(co)) -- Resume in a function
```

generated by haroopad

```

        print(coroutine.status(co))
    end
end)

coroutine.resume(co)
print("do something")
coroutine.resume(co)
print("do something")
coroutine.resume(co)
print("do something")
print(coroutine.resume(co))
print(coroutine.status(co))
coroutine.resume(co)
coroutine.resume(co)
print(coroutine.resume(co))
print(coroutine.status(co))

```

**Result:**

```

1
do something
false cannot resume running coroutine
running
2
do something
false cannot resume running coroutine
running
3
do something
false cannot resume running coroutine
running
4
true
suspended
false cannot resume running coroutine
running
5
false cannot resume running coroutine
running
false cannot resume dead coroutine
dead

```

**Another example:**

Above I showed the output if `print(coroutine.resume(co))` was in the function. I will demonstrate the result of it not being there.

```

co = coroutine.create(function()
    for i = 1, 5 do
        print(i)
        coroutine.yield()
        print(coro

```

```
 coroutine.resume(co)
 coroutine.resume(co)
 print(coroutine.resume(co) )
 print(coroutine.status(co) )
```

Result:

```
1
do something
running
2
do something
running
3
do something
running
4
true
suspended
running
5
running
false cannot resume dead coroutine
dead
```

## Coroutine Callback

Calls can be added to `coroutine.resume()`.

```
coroutine.resume(co, call, call)
```

```
co = coroutine.create(function()
    for i = 1, 5 do
        print(coroutine.yield(i)) -- true or false
    end
end)

local three = 3
print(coroutine.resume(co))
print(coroutine.resume(co, three, 4))
print(coroutine.resume(co, 5, 6))
print(coroutine.resume(co, 7, 8))
print(coroutine.resume(co, 9, 10))
print(coroutine.resume(co, 11, 12))
print(coroutine.resume(co, 13, 14))
print(coroutine.resume(co, 15, 16))
```

Result:

```
true
1
3 4
true 2
5 6
true 3
7 8
true 4
9 10
true 5
11 12
true
```

```
false cannot resume dead coroutine
false cannot resume dead coroutine
```

## COROUTINE WRAP SHORTCUT

One can use the function `coroutine.wrap` to shorten the coroutine process.

```
c = coroutine.wrap(function()
    for i = 1, 5 do
        print(i)
        coroutine.yield()
    end
end)

c()
c()
```

Result:

```
1
2
```

## MODULES

First off, make a Lua file called `Module_1.lua` or any name you want to be honest, but it will be that name in this example. Put it in the scripts directory of the project data folder.

### MAKE A TABLE

The first task is to make a table. It is better to keep your table local, but it will be global to show you why it should be local later.

```
--[[
m = {} -- make local

return m
--]]
```

### ADD TASKS

You will need to add `m.` before any table item like normal.

```
m.printHello = function()
    print("hello")
end

m.pi = 3.1415

m.t = {1, 2, 3}

m.number = function(number)
    print("Number is: "..number)
end
```

### REQUIRE MODULE

If you plan to use this module all of your scripts, then it would be best to require it at the top of `main.lua`.

In this case I will use a map script because it will be discarded when leaving the map. generated by haroopad

In `first_map.lua` require the module and activate the tasks. One will have to assign a variable to the required script to use it.

```
c = require("scripts/module_1")

--print hello
c.printHello()

--print pi
print(c.pi)

--print all values in table
for _, v in pairs(c.t) do
    print(v)
end

--number is:
c.number(400)
```

**Result:**

```
hello
3.1415
1
2
3
Number is: 400
```

## Global Module Table Issue

Making the table in the module `global` is not a good idea because at some point there might be mix ups with code in the module.

Let me show you what I mean.

```
c = require("scripts/module_1")

c.printHello()
m.printHello()
```

**Result:**

```
hello
hello
```

As you can see "hello" printed twice because there is no local restriction in the module for `m.printHello()`. This might become a problem at some point. An easy fix is to make the table in the module local.

```
local m = {} -- make local

m.printHello = function()
    print("hello")
end

m.pi = 3.1415

m.t = {1, 2, 3}

m.number = function(number)
```

generated by haroopad

```

print("Number is: "..number)
end

return m

```

Now if you try it an error will occur with `m.printHello()`.

```
Error: In maps/first_map: [string "maps/first_map.lua"]:21: attempt to index global 'm' (a nil value)
```

## Module Shortcut

Instead of writing `m.` over and over again, one can set a metatable and point the table to `_ENV` (Environment).

```

local m = {}
setmetatable(m, {__index = _G}) -- index items in table to be global (_G)
_ENV = m -- use this to remove "m." in code.

printHello = function()
    print("hello")
end

pi = 3.1415

t = {1, 2, 3}

number = function(number)
    print("Number is: "..number)
end

return m

```

## Assign New Variable

One can assign a new variable to require and use it.

```

c = require("scripts/module_1")

c.printHello()
print(c.pi)

for _, v in pairs(c.t) do
    print(v)
end

c.number(400)

value = require("scripts/module_1")

value.number(700)
c.number(500)

maths = c

maths.number(2000)

```

## Result:

```

hello
3.1415

```

generated by haroopad

```

1
2
3
Number is: 400
Number is: 700
Number is: 500
Number is: 2000

```

## Dot vs Colon

The colon `:` is for implementing methods that pass `self` as the first parameter and the dot `.` does not unless you pass it into itself.

```

x = {pos_x = function(x,y) return x end, pos_y = function(x,y) return y end, }

print("Dot: ",x.pos_x(3,4))
print("Dot: ",x.pos_y(3,4))

print("Colon: ",x:pos_x(3,4))
print("Colon: ",x:pos_y(3,4))

--Same as Colon.
print("Dot self: ",x.pos_x(x,3,4))
print("Dot self: ",x.pos_y(x,3,4))

```

Result:

```

Dot: 3
Dot: 4
Colon: table: 0x06a57fb8
Colon: 3
Dot self: table: 0x06a57fb8
Dot self: 3

```

## Function Objects

An object has:	
state	Being in a form for growth or development.
independent value identity	A self.
independent life cycle	Does not matter what created them or where they were created.
assignability	Receive operation.

## Almost Method

A table in Lua is basically a object. However, this function will work only for this object.

```

--almost method
MoneyBag = {amount = 0}
function MoneyBag.add(value)
    MoneyBag.amount = MoneyBag.amount + value
end

MoneyBag.add(200)

print("You have the total amount of "..MoneyBag.amount.." in your bag.")

```

Result:

generated by haroopad

You have the total amount of 200 in your bag.

You can change the name of MoneyBag.

```
bag = MoneyBag
bag.add(50)
print("You have the total amount of "..MoneyBag.amount.." in your bag.")

or

print("You have the total amount of "..bag.amount.." in your bag.")
```

**Result:**

You have the total amount of 250 in your bag.

The reason this is almost a method is because if we do the following we get an error.

```
MoneyBag = nil
bag.add(50)
print("You have the total amount of "..bag.amount.." in your bag.")
```

**Result:**

Error: attempt to index global 'MoneyBag' (a nil value)

## Method

This function will work only for more than one object. **Self** refers to itself.

```
MoneyBag = {amount = 0}
function MoneyBag.add(self, value)
    self.amount = self.amount + value
end

MoneyBag.add(MoneyBag, 200)

bag = MoneyBag

MoneyBag = nil

bag.add(bag, 50)

print("You have the total amount of "..bag.amount.." in your bag.")
```

**Result:**

You have the total amount of 250 in your bag.

## Method With Colon

You can shorten the method with a Colon. This will pass **self** for you.

```

MoneyBag = {amount = 0}
function MoneyBag:add(value)
    self.amount = self.amount + value
end

MoneyBag:add(200)

bag = MoneyBag

MoneyBag = nil

bag:add(50)

print("You have the total amount of "..bag.amount.." in your bag.")

```

**Result:**

You have the total amount of 250 in your bag.

## Colon Hidden Parameter

The colon passes a hidden parameter by assigning the function to a variable called `[add]` in this case.

That means the colon is doing the following:

```

MoneyBag = { amount = 0,
            --Assign variable to dot function
            add = function (self, value)
                    self.amount = self.amount + value
                    end
            }

MoneyBag:add(400)

print("You have the total amount of "..MoneyBag.amount.." in your bag.")

```

**Result:**

You have the total amount of 400 in your bag.

I prefer using a colon though. It is cleaner and easier to organize.

```

MoneyBag = {amount = 0}
function MoneyBag:add(value)
    self.amount = self.amount + value
end

MoneyBag:add(400)

print("You have the total amount of "..MoneyBag.amount.." in your bag.")

```

**Result:**

You have the total amount of 400 in your bag.

## Tutorial Point Lua PDF

generated by haroopad

Read this book for more information and detail.

[https://www.tutorialspoint.com/lua/lua\\_tutorial.pdf](https://www.tutorialspoint.com/lua/lua_tutorial.pdf)

## Ways To Load Script In Solarus Part 1

Require()

### Require Part 1:

Using require is the best way to load your file, but I use sol.main.load\_file a lot when simply testing because it is faster to set up.

In the file `game_manager.lua`

put `local load_that = require("scripts/0_Lua_Quick_Basics.lua")` at the top of `game_manager.lua`

put `load_that:load_test()` above `return game_manager` at the bottom of `game_manager.lua`

### Require Part 2:

In the file `0_Lua_Quick_Basics.lua`

```
`--Make a table`  
`local load_that = {}`  
  
`--Make a function`  
`function load_that:load_test(game)`  
`--Code of file 0_Lua_Quick_Basics.lua`  
`end -- end of function`  
  
`return load_that --return the table`
```

### Breaking down the script:

In the file `game_manager.lua`

`local load_that = {}` is making a table named `load_that`

`require` is for loading and remembering code files

`"scripts/0_Lua_Quick_Basics.lua"` is a directory. Scripts folder > file 0\_Lua\_Quick\_Basics.lua

In the file `0_Lua_Quick_Basics.lua`

Now you should know everything basic except for the function from the quick Lua knowledge files. A function does a certain task and in this case it loads the script.

```
function table:give_name_to_function(parameter)  
--Code in here  
end  
  
`return table`
```

`(parameter)` is like the scope or range of something. Normally, it is either the (game) or (map)

Playtest the game with F5 and the script will run. Sometimes the console will open. If not, then press F12. You can drag it up to show more of the output.

### Lesson Sample File:

generated by haroopad

I added a test file in the Github repo. Lessons folder > Chapter\_6 > chapter\_6\_require\_load.zip

## Sol Main Load

```
sol.main.load_file(script_name)
```

This one way to load a Lua script with Solarus. Not the best way. Slower than require because require actually remembers the script after the first run. I use this mainly for testing purposes.

In the `file game_manager.lua`

put `sol.main.load_file("scripts/0_Lua_Quick_Basics.lua") (game)` above `(game:start()`

### Breaking down the script:

`sol.main.load_file()` is for loading files

`"scripts/0_Lua_Quick_Basics.lua"` is a directory. Scripts folder > file 0\_Lua\_Quick\_Basics.lua

Direct it to the game parameter: `(game)`

The end result is this:

```
sol.main.load_file("scripts/0_Lua_Quick_Basics.lua") (game)
```

Playtest the game with F5 and the script will run. Sometimes the console will open. If not, then press F12. You can drag it up to show more of the output.

I added a test file in the Github repository. [Lessons folder > Chapter\\_6\\_sol\\_main\\_load.zip](#)

## Grab Numbers & Letters

Grabbing letters and numbers can make some interesting puzzles or scripts.

I made a very bulky **Menu Dialog Display Script** using letter grabs and I used digit grabs in my **ADVANCED DIGIT DISPLAY SCRIPT**. I made those scripts as I was first learning Lua.

### Grab Numbers

#### Script 1:

This script converts the number value into a string, splits it up, and converts it back into a number.

```
local value = 25436 -- value
local value_string = string.format("%04d", value) --value_string = "25436" (make value a string)
-- sting format: "%d"
-- Add zeros: "%04d" (if below 4 digits)

local lenth = value_string:len() --Grabs lenth of string for soda array

local value_split = {}
for i = 0,lenth do
    local index = value_string:len()-i --decrease by i
    local num = tonumber(value_string:sub(index,index)) -- convert string to number and decrease
    table.insert(value_split, num) --insert letter into table value_split
end

print("value "..value_split[1])
print("value "..value_split[2])
print("value "..value_split[3])
```

generated by haroopad

```
print("value "..value_split[4])
print("value "..value_split[5])
```

**Result:**

```
value 6
value 3
value 4
value 5
value 2
```

**Script 2:**

This script inserts each digit into a table.

```
local value = 25436
local value_split = {}

for digit in string.gmatch( tostring(value), "%d" ) do
    table.insert(value_split, digit)
end

print("value "..value_split[1])
print("value "..value_split[2])
print("value "..value_split[3])
print("value "..value_split[4])
print("value "..value_split[5])
```

**Result:**

```
value 2
value 5
value 4
value 3
value 6
```

**Script 3:**

This script uses a somewhat classic calculation in an advanced way.

```
local value = 25436
local value_split = {}

local num = value
if num == 0 then value_split = {0} end
while num > 0 do
    local digit = num % 10
    num = math.floor(num / 10)
    table.insert(value_split, digit)
end

print("value "..value_split[1])
print("value "..value_split[2])
print("value "..value_split[3])
print("value "..value_split[4])
print("value "..value_split[5])
```

**Result:**

```
value 6
value 3
value 4
value 5
value 2
```

The script above takes the remainder of the value and decreases the place value.

```
local value = 25436

local num = value

local digit = num % 10
num = math.floor(num / 10)

print(num) -- 2543
print(digit) -- 6

--Takes the remainder
digit = num % 1 -- remainder of the 1st place is 0 ( No remainder)
digit = num % 10 -- remainder of the 10th place is 6
digit = num % 100 -- Remainder of the 100th place is 36
digit = num % 1000 -- Remainder of the 1000th place is 436
digit = num % 10000 -- Remainder of the 10000th place is 5436
digit = num % 100000 -- Remainder of the 100000th place is 25436

--Decreases the value place
num = math.floor(num / 1) -- 25436
num = math.floor(num / 10) -- 2543
num = math.floor(num / 100) -- 254
num = math.floor(num / 1000) -- 25
num = math.floor(num / 10000) -- 2
```

## Grab Letter

### Script 1:

This script uses the alphabet pattern `"%w"` to get letters and inserts them into a table.

```
local string = "test"
local string_split = {}

for char in string.gmatch(string, "%w") do
    table.insert(string_split, char)
end

print("letter "..string_split[1])
print("letter "..string_split[2])
print("letter "..string_split[3])
print("letter "..string_split[4])
```

### Result:

```
letter t
letter e
letter s
letter t
```

### Script 2:

This one shows how one can reverse the string with `string.reverse()`.

generated by haroopad

```

string_split = {}
local text = "test"
local length = text:len()
local reverse = false

if reverse == true then
    for msg = 0, length do
        index = text:len()-msg
        letter = text:sub(index, index)
        table.insert(string_split, letter)
    end
else
    for msg = 0, length do
        index = string.reverse(text):len()-msg
        letter = string.reverse(text):sub(index, index)
        table.insert(string_split, letter)
    end
end

print("letter "..string_split[1])
print("letter "..string_split[2])
print("letter "..string_split[3])
print("letter "..string_split[4])

```

Result:

```
reverse = false
```

```

letter t
letter e
letter s
letter t

```

Result:

```
reverse = true
```

```

letter t
letter s
letter e
letter t

```

## Chapter 7: Setting up Dialog and Pause

### Setting Up The Dialog Script

In `game_manager.lua` put the following require directory at the top of the script. (I explained how to use require already. This is the dialog box script designed by Christopho.)

```
require("scripts/menus/altp_dialog_box")
```

in folder `scripts/menus/`

put `altp_dialog_box.lua`

in folder `scripts/`

put `multi_events.lua`

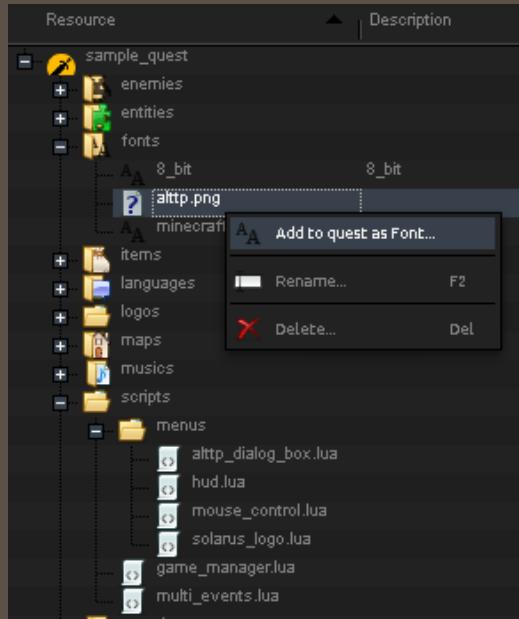
in folder `sprites/hud`

put `dialog_box.png` (I provide a free image to use in the lesson > chapter\_7\_Dialog.zip)

in folder `fonts/`

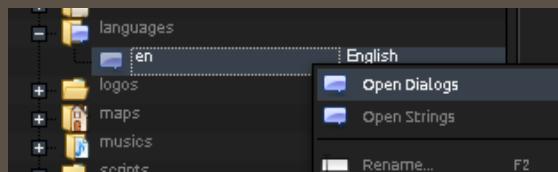
put bitmap font `alttp.png`

In the editor in the font section. Make sure to add the alttp.png.



## Using the Dialog box

Go to the dialog section of the editor. Languages > en > double click



Press the big green plus(+) sign.

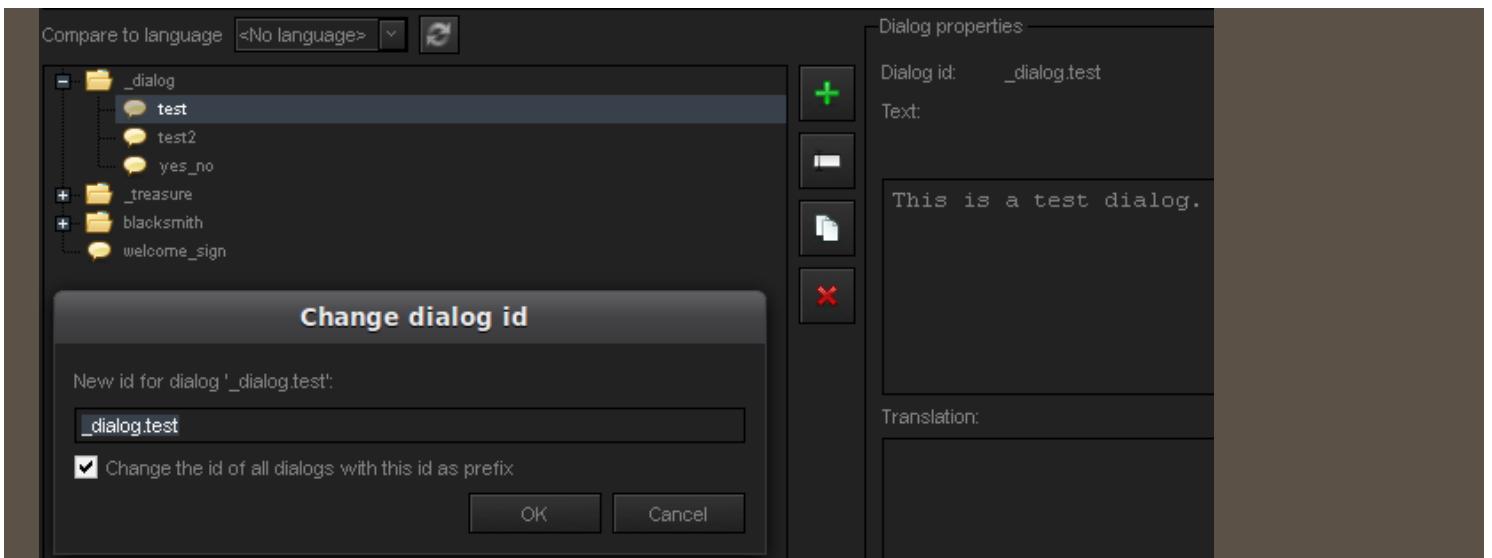
type `_dialog.test`

`_dialog` is the main folder

`.test` is the dialog

Double click on `_dialog`, go to dialog properties and down to the first rectangle box where it says text (At the right)

Type any text you want in that box. For example, "This is a test dialog."



## Mouse Control Fix

In `game_manager.lua`

Basically for the mouse control to work `function game:on_started() end` needs to be replaced with `game:register_event("on_started", function() end)`.

Change this:

```
function game:on_started()
    -- HUD menu.
    local hud = require("scripts/menus/hud")

    sol.menu.start(game, hud)
    hud:create(game)

    -- Mouse control.
    local mouse_control = require("scripts/menus/mouse_control")

    sol.menu.start(game, mouse_control)
    mouse_control:create(game, hud)

    local hero = game:get_hero()
    hero:set_tunic_sprite_id("main_heroes/eldran")
end
```

To this:

```
game:register_event("on_started", function()
    -- HUD menu.
    local hud = require("scripts/menus/hud")
    sol.menu.start(game, hud)
    hud:create(game)

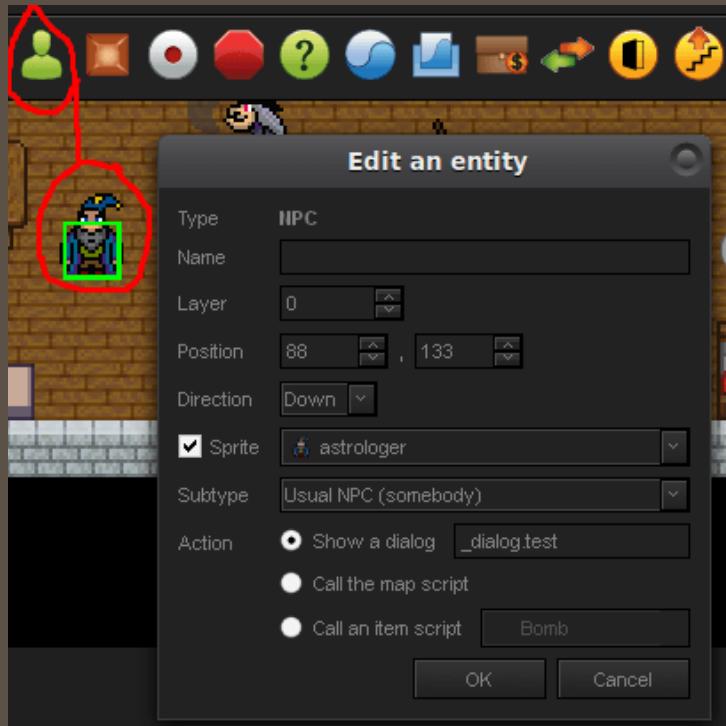
    -- Mouse control.
    local mouse_control = require("scripts/menus/mouse_control")
    sol.menu.start(game, mouse_control)
    mouse_control:create(game, hud)

    local hero = game:get_hero()
    hero:set_tunic_sprite_id("main_heroes/eldran")
end)
```

Add `multi_events` at the beginning of the script:

```
require("scripts/multi_events")
```

You can add a NPC and add `_dialog.test` to it.



Save the project and playtest with F5. You should see a dialog box appear after "pressing the space bar" to talking to the NPC.



### Passing a value and string into the dialog

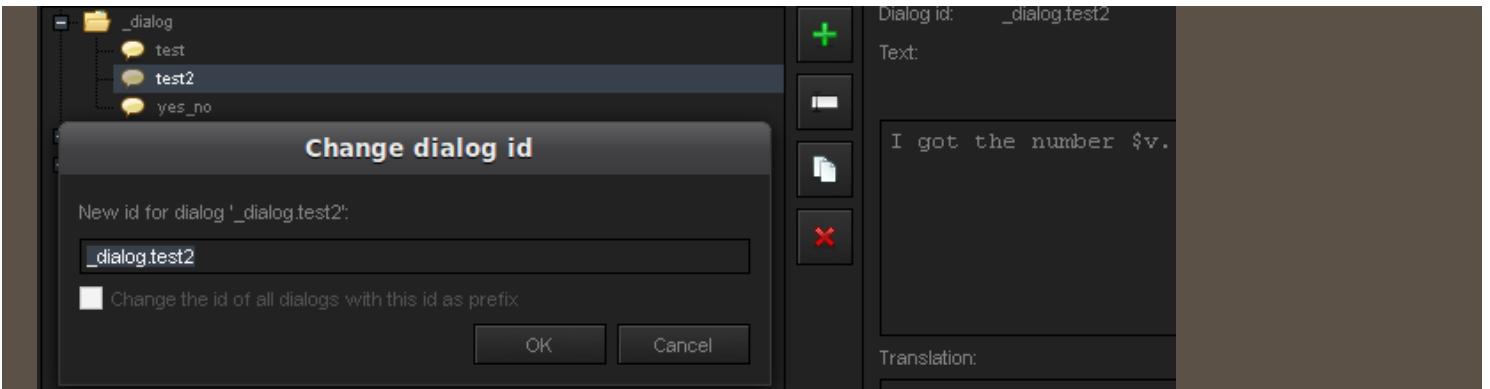
Now what if you wanted to pass a name or number value into the dialog?

Make a new dialog. Click test > add new dialog (+) > change it to "`_dialog.test2`

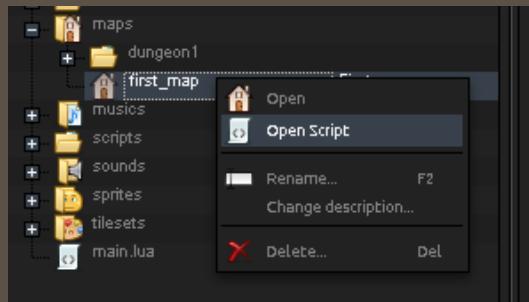
In the text section, add the `$v` in the text. This passed a variable into the dialog.

### Example:

"I got the number \$v."



Open your map script. map folder > right click > open script



You want to pass the game parameter into the map.

```
local map = ...
local game = map:get_game()
```

You would make a variable to pass to `$v`

```
local value = 5
```

To start a dialog in the map script.

```
game:start_dialog("_dialog.test2", value)
```

`game:start_dialog("_dialog.test2", )` starts the dialog

`, value)` passes the number 5 from value variable into the dialog.

The following would stop the dialog, but you cannot start a dialog and stop one at the same time. You most likely would use a key press to stop the dialog, but we will get to that in a later chapter.

```
game:stop_dialog("_dialog.test2")
```

Save the project and playtest with F5. It should just appear. Press the "space bar" to get rid of the text.



## Setting up yes\_no

I personally script my own yes\_no menu with images, but the altp dialog box can make yes and no.

First off, we will start with making the dialog for the pause: `_dialog.yes_no`

Secondly, make the text: `"Do you want to print yes?"`

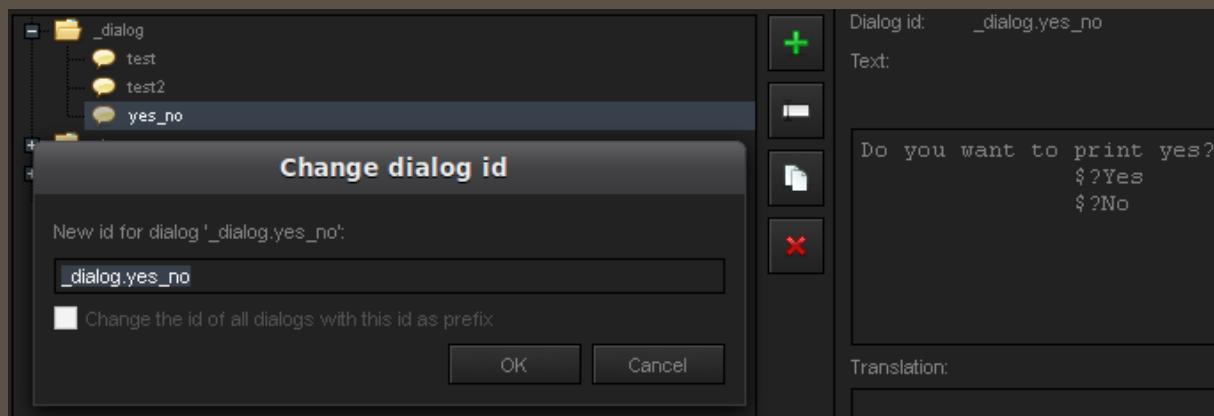
The characters for activating the arrow and picking of yes/no is: `$?`

In the text section:

`"Do you want to print yes?"`

`$?Yes`

`$?No`



## Dialog lines

You must remember the line order for yes/no in order to correctly code it.

`"Do you want to print yes?"` (line 1)

`$?Yes` (line 2)

`$?no` (line 3)

## Adding on\_paused

In the `game_manager.lua` put the following code above `game:start()`

```

function game:on_paused()
    game:start_dialog("_dialog.yes_no", function(answer)
        --Answer 2 is line 2
        if answer == 2 then
            print("Yes")
        end
    end)
end

```

`function game:on_paused()` is the pause function. This activates when the game is paused. The key `d` is default for pausing in Solarus. There is function `game:on_unpaused()` as well. In unpause, code you write will activate when you press `d` again to unpause the game.

`function(answer)` is part of the dialog box. Basically, `answer == 2` is line 2.

Playtest with `F5` and press the key `d` and press the `space bar` to choose yes.



### Dialog Lesson Sample:

The lesson file is located in the folder:

[Lesson > chapter\\_7\\_Dialog.zip](#)

## Chapter 8: Displaying an image, opacity, color fill, blend modes, and font display

### Preview:

I would like to show a preview before starting the lesson. This is everything you will be learning in this lesson.



## Script For The Lesson

I wanted to show the script before breaking it down. This is file [display\\_sample.lua](#).

```
--Pass the game parameter to the script.
local game = ...

--If sample is true then the surfaces appear.
local image_sample = true

--Clear_pixels variable
local clear_pixels = false

--Make the surface for the orange surface
local surface_img = sol.surface.create()

--Make the surface for the button image
local button_img = sol.surface.create("button.png")

--Make the surface for blend multiply
local blend_multiply_img = sol.surface.create("blend.png")

--Make the surface for blend add
local blend_add_img = sol.surface.create("blend.png")

--Make the surface for no blending
local blend_none_img = sol.surface.create("blend.png")

--Make the surface for blend
local blend_green_img = sol.surface.create("green.png")
```

```
--Make the surface for blend
local blend_red_img = sol.surface.create("red.png")

--Make the surface for blend
local blend_blue_img = sol.surface.create("blue.png")

local blend_test_img = sol.surface.create()

--http://www.solarus-games.org/doc/latest/lua_api_text_surface.html
local text_img = sol.text_surface.create({ -- name a local variable something and assign it to the
    font = "minecraftia", -- font name
    text = "what", -- text you want to show
    font_size = 50, -- font size obviously
    horizontal_alignment = "center", -- default "left"
    vertical_alignment = "bottom", -- default "middle"
    rendering_mode = "antialiasing", -- "solid" (faster) and default
    color = {0,0,0}, -- color must be in a table RGB (http://www.rapidtables.com/web/color/RGB\_color.html)
})

-- A function for displaying images and fonts on screen.
function sol.main:on_draw(screen)

--Drawing the variable surface (local test) on screen with coordinates (100,100) or x,y
if image_sample == true then

    --"multiply" blend mode.
    blend_multiply_img:draw(screen,100,100)
    blend_multiply_img:set_blend_mode("multiply")

    --"add" blend mode.
    blend_add_img:draw(screen,100,150)
    blend_add_img:set_blend_mode("add")

    --"none" blend mode.
    blend_none_img:draw(screen,160,200)
    blend_none_img:set_blend_mode("none")

    --show or draw the surface
    surface_img:draw(screen)

    --fill surface with a orange color
    surface_img:fill_color({245,68,0})

    --"blend" blend mode.
    blend_test_img:draw(screen)
    blend_red_img:draw(surface_img,100,200)
    blend_red_img:set_blend_mode("blend")

    blend_green_img:draw(surface_img,100,200)
    blend_green_img:set_blend_mode("blend")

    blend_blue_img:draw(surface_img,100,200)
    blend_blue_img:set_blend_mode("blend")

    --show the text "what" at (x,y) (100,100)
    text_img:draw(screen,100,100)

    --show or draw the button image
    button_img:draw(screen)

    --opacity to 50% semi-transparent
    surface_img:set_opacity(50)

    --if clear pixels is true
    if clear_pixels == true then

```

```

--clear surface
surface_img:clear()
blend_test_img:clear()
end
end -- end of if image_sample is true
end --end of draw function

--on_paused
--Press key 'd' to make the orange surface to appear
function game:on_paused()
  clear_pixels = true
end

--on_unpaused
--Press key 'd' again to clear the orange surface
function game:on_unpaused()
  clear_pixels = false
end

```

## Download Sample:

You can download the sample for this lesson in the lessons folder.

[Lessons > Chapter\\_8 > Chapter\\_8\\_display\\_example\\_added\\_blend.zip](#)

## Breaking Down The Script

### Explaining Surfaces

First off, I would like to say that I will not repeatedly break down the "same" parts of the script. I will only give examples of new material I introduce. Let us begin!

In order to pass the game parameter into the script, one needs to add `local game = ...` at the beginning of the script. You need this to use functions like `function game:on_paused()`.

The next line that you may not understand is `local surface_img = sol.surface.create()`. In this line of code I create and assign a variable called `surface_img` to a newly created surface `sol.surface.create()`. This surface will make pixels fill the whole screen with a single RGB color value for `surface_img`. We will get to that later in the script.

The next line of code `local button_img = sol.surface.create("button.png")` is exactly the same as the previous line I explained, but we create a surface for an image file. In this case a portable graphic image (.png) `button.png`. This image is located in the sprites' directory. [sprites > button.png](#)

The next line of code is making a surface for a `.ttf` font package or bitmap font like `alttp.png`. You cannot change colors for the bitmap font and there are a few other limitation with that font type. You can do everything with a `.ttf` font(TrueType Font). Solarus supports .ttf, .tcc and fon. The surface that is created for the font, uses a table in order to assign properties to it. You must never forget the comma at the end of each line.

```

local test_img = sol.text_surface.create({ -- name a local variable something and assign it to the
  font = "minecraftia", -- font name
  text = "what", -- text you want to show
  font_size = 50, -- font size obviously
  horizontal_alignment = "center", -- default "left"
  vertical_alignment = "bottom", -- default "middle"
  rendering_mode = "antialiasing", -- "solid" (faster) and default
  color = {0,0,0}, -- color must be in a table RGB (http://www.rapidtables.com/web/color/RGB\_color.htm)
})

```

generated by haroopad

1. `font = "minecraftia",` is the name of the font package. You can leave off the extension `.ttf`.
2. `text = "what",` is the text you want to draw or show on the screen. The best way to do this would be with an array.

### Example:

```
for rep = 1, line_amount do
    text = line[rep]
end

line[1] = "what"
line[2] = "the"
line[3] = "sprite"
```

3. `font_size = 50,` This is..well, the size of your font.
4. `horizontal_alignment = "center",` The horizontal alignment. The default is default "left." I personally ignore the alignments. I just remove them from the table.
5. `vertical_alignment = "bottom",` The vertical alignment. The default is "middle."
6. `rendering_mode = "antialiasing",` The rendering mode is how your font looks. `"Solid"` is faster and default.
7. `color = {0,0,0}` This is the RGB value for the font color. `{0.0.0}` is black. You can look at the RGB reference section in the book for more color values.

### Explaining Draw Function

Now we are going to go over the `function sol.main:on_draw(screen)`. We need this function for showing images and text on the screen.

```
function sol.main:on_draw(screen)
end
```

### Drawing an image, font, and a fill\_color

We need to add `:draw(screen,x,y)` on to a surface we created `test_img` and `button_img` or `surface_img` in order to draw a surface and set coordinates.

```
function sol.main:on_draw(screen)
    test_img:draw(screen,100,100)
end
```

### Filling a Color

On the surface `surface_img` that was created we need to add `fill_color({245,68,0})` on to `surface_img`. That will create an orange color.

```
function sol.main:on_draw(screen)
    surface_img:draw(screen)
    surface_img:fill_color({245,68,0})
end
```

### Opacity or Semi-transparency

We currently have a screen filled with orange color and now we cannot see anything. One will need to make it see through or semi-transparent. That is the opacity. You need to assign the surface `surface_img` to `set_opacity(number)`.

For example:

```
surface_img:set_opacity(50)
```

Opacity at 50% will make it perfect most of the time.

```
function sol.main:on_draw(screen)
    surface_img:draw(screen)
    surface_img:fill_color({245,68,0})
    surface_img:set_opacity(50)
end
```

### on\_unpaused function & clear pixels

I never explained the `on_unpaused` function fully before. I did mention it, but not in actual use. When the game is paused with the key `d` the orange semi-transparent pixels are cleared. In order to clear pixels the `clear()` needs to be added onto the surface `surface_img`.

```
if clear_pixels == true then
    --clear surface
    surface_img:clear()
end
```

That means when the 'd' key is pressed to pause the game, all the orange pixels will vanish off the screen. Also, when 'd' is press again, the orange pixels will once again appear.

```
--on_paused
--Press key 'd' to make the orange surface to appear
function game:on_paused()
    clear_pixels = true
end

--on_unpaused
--Press key 'd' again to clear the orange surface
function game:on_unpaused()
    clear_pixels = false
end
```

### Blend Modes

```
surface:set_blend_mode(blend_mode)
```

<b>mode(blend_modes)</b>	<b>About</b>
"none"	No blending.
"blend"(default)	The surface is alpha-blended
"add"	The surface is colored and lightened.
"multiply"	Darken the surface without degrading the image

### Examples:

```
--"multiply" blend mode.
blend_multiply_img:draw(screen,100,100)
blend_multiply_img:set_blend_mode("multiply")

--"add" blend mode.
blend_add_img:draw(screen,100,150)
blend_add_img:set_blend_mode("add")

--"none" blend mode.
blend_none_img:draw(screen,160,200)
blend_none_img:set_blend_mode("none")

--"blend" blend mode.
blend_red_img:draw(screen,100,200)
blend_red_img:set_blend_mode("blend")

blend_green_img:draw(screen,100,200)
blend_green_img:set_blend_mode("blend")

blend_blue_img:draw(screen,100,200)
blend_blue_img:set_blend_mode("blend")
```

## Drawing to a Surface

I am not going mention this in the sample, the same way, but one can draw images to a single surface and remove them all at the same time.

```
local blend_test_img = sol.surface.create()

-- A function for displaying images and fonts on screen.
function sol.main:on_draw(screen)
    --"blend" blend mode.
    blend_test_img:draw(screen)
    blend_red_img:draw(blend_test_img,100,200)
    blend_red_img:set_blend_mode("blend")

    blend_green_img:draw(blend_test_img,100,200)
    blend_green_img:set_blend_mode("blend")

    blend_blue_img:draw(blend_test_img,100,200)
    blend_blue_img:set_blend_mode("blend")

    --if clear pixels is true
    if clear_pixels == true then
        --clear surface
        blend_test_img:clear()
    end
end
```

The example shows that everything is being drawn on `blend_test_img` instead of the `screen`. All three of the images will vanish at the same when `clear_pixels is true`.

## Drawing a Sprite

You can draw more than just font, images, and color surfaces. Solarus has the ability to draw sprites and animate them.

A common usage for sprite drawing is for scenes. For example, when an boss enemy or hero dies because the drawn sprite can be `unpaused when the game is paused`.

## Sample

You can grab the sample in the directory [[Lessons > Chapter\\_8 > Chapter\\_8\\_sprite\\_draw.zip](#)].

## Script

This script is in the `first_map.lua` in the sample.

```

local map = ...
local game = map:get_game()

local x_pos, y_pos = 100, 100 -- x,y coordinates
local x,y -- drawing coordinates

sol.timer.start(100, function() --Timer for checking. Normally not needed if you pause the game.
    local camera_x, camera_y = map:get_camera():get_bounding_box()-- Get camera coordinates and bound
    y = y_pos - camera_y -- y_pos = y_pos - camera_y will constantly minus the camera coordinates, u
    x = x_pos - camera_x
    return true -- repeat timer
end)

local sprite = sol.sprite.create("main_heroes/eldran") -- sprite you want to draw.
sprite:set_animation("dying") -- do not loop it in the sprite editor or it will keep doing the dyin

function sprite:on_animation_finished() --function for when the dying animation ends.
    sprite:set_animation("dead")
end

function map:on_draw(screen) -- draw sprite
    sprite:draw(screen, x, y)
end

```

## Breaking down the Script

1. First, make the coordinates. We will not use the `x_pos`, `y_pos` variables for drawing. Instead, we will use `x`, `y` variables for checking reasons.

```

local x_pos, y_pos = 100, 100 -- x,y coordinates
local x,y -- drawing coordinates

```

2. Second, make a timer for checking. Normally not needed if you pause the game. If your map size is beyond 320 x 240, then you must minus the camera to get proper drawing coordinates. I use the `x`, `y` variables because I do not want the `x_pos`, `y_pos` to be constantly subtracted from the coordinates, when checking with the timer.

```

sol.timer.start(100, function() --Timer for checking. Normally not needed if you pause the ga
    local camera_x, camera_y = map:get_camera():get_bounding_box() -- Get camera coordinates and
    y = y_pos - camera_y -- y_pos = y_pos - camera_y will constantly minus the camera coordinates
    x = x_pos - camera_x
    return true -- repeat timer
end)

```

3. Thirdly, create the sprite from the sprite directory and set a desired animation. Do not loop the animation in the sprite editor or it will repeat. Most likely one will want an end animation because one would want a finishing animation.

```
local sprite = sol.sprite.create("main_heroes/eldran") -- sprite you want to draw.  
sprite:set_animation("dying") -- do not loop it in the sprite editor or it will keep doing th
```

4. Use the function `on_animation_finished()` to set an end animation or you could use a timer, but that would be more effort.

```
function sprite:on_animation_finished() --function for when the dying animation ends.  
sprite:set_animation("dead")  
end
```

5. Lastly, draw the sprite at the x, y coordinates.

```
function map:on_draw(screen) -- draw sprite  
sprite:draw(screen, x, y)  
end
```

## Chapter 9: Key press, Mouse press, Image fade, and Playing Audio

### Lesson Preview:

One can watch a video preview of the lesson.

[Lessons > Chapter\\_9 > Chapter\\_9\\_Video\\_Preview.ogv.zip](#)

### Lesson Sample:

One can download the completed lesson.

[Lessons > Chapter\\_9 > Chapter\\_9\\_key\\_mouse\\_fade.zip](#)

## Lesson Script

I would like to show the script before beginning the lesson and how to use it.

1. The script can be activated with the 'a' key.

**Sample quest - Solarus 1.5.0**

1. The green image will fade out when 'o' is pressed.
2. The green image will fade in when 'i' is pressed.

**Sample quest - Solarus 1.5.0**

1. The volume will go up by 5 if the 'u' key is pressed.
2. The volume will go down by 5 if the 'y' key is pressed.
3. The blue block will move up, down, left, and right by three (3). Only if the up, down, left, right keys are pressed. One will notice it moved a little.

**Sample quest - Solarus 1.5.0**

1. Right clicking the mouse "on" the red block will make it vanish and left clicking will make it appear again.

**Sample quest - Solarus 1.5.0**

generated by haroopad

```
--Pass the game parameter to the script.
local game = ...

--Creating surfaces for the images and loading in the images.
local key_img = sol.surface.create ("key.png")
local mouse_img = sol.surface.create ("mouse.png")
local fade_img = sol.surface.create ("fade.png")

local key_image = false
local mouse = false
local fade = false

--Variables for the sound
local volume = 0

--Up and down direction variables
local up_direction = 0
local down_direction = 0

--Left and right direction variables
local right_direction = 0
local left_direction = 0

--X and y for key_img surface
local key_x = 0
local key_y = 0

--Solarus key pressing function
function sol.main:on_key_pressed(key)

    --Pause the game
    game:set_paused(true)

    --set volume to zero if it goes into the negatives.
    if volume < 0 then
        volume = 0
    end

    --If key is 's' then the volume goes down by 5 as long as it is greater than zero.
    if key == "y" then

        --Decreases the volume
        volume = volume - 5
        print("Volume down: "..volume)

        --Change the volume
        sol.audio.set_music_volume(volume)
        sol.audio.set_sound_volume(volume)

        --Play music and sound
        sol.audio.play_music("village")
        sol.audio.play_sound("chest_appears")
    end

    --If key is 'a' then the volume goes up by 5 as long as it is greater than or equal to zero.
    if key == "u" then
        --Volume increases by 5 only if it is less than or equal to 100
        if volume <= 100 then

            --Increases the volume
            volume = volume + 5
            print("Volume up: "..volume)

            --Change the volume
        end
    end
end
```

```

sol.audio.set_music_volume(volume)
sol.audio.set_sound_volume(volume)

--Play music and sound
sol.audio.play_music("village")
sol.audio.play_sound("chest_appears")
end
end

--Activates images on key 'a'
if key == "a" then
    key_image = true
    fade = true
    mouse = true
end

----Activates fade in on key 'i'
if key == "i" then
    fade_img:fade_in(100,stop_draw())
end

----Activates fade out on key 'o'
if key == "o" then
    fade_img:fade_out(100,stop_draw())
end

--Activates when the up key is pressed
if key == "up" then

    --Decreases down by 1 if it is greater than 0
    if down_direction > 0 then
        down_direction = down_direction - 1
    end

    --If up is less than 3, then up increases by 1
    if up_direction < 3 then
        up_direction = up_direction + 1
        print("Up: "..up_direction)

        --For loop for when up direction equals a certain number. This way I do not have to write t
        for rep = 1,3 do
            if up_direction == rep then
                --surface Key_img y coordinate decreases by 3. This moves the blue block. (key_img:dra
                key_y = key_y - 3
                print("Up_key: "..key_y)
            end
        end --end of for loop
    end -- end of up direction
end -- end of up key

--Activates when the down key is pressed
if key == "down" then

    --Decreases up by 1 if it is greater than 0
    if up_direction > 0 then
        up_direction = up_direction - 1
    end

    --If down is less than 3, then up increases by 1
    if down_direction < 3 then
        down_direction = down_direction + 1
        print("Down: "..down_direction)

        --For loop for when up direction equals a certain number. This way I do not have to write t
end

```

```

for rep = 1,3 do
    if down_direction == rep then
        --surface Key_img y coordinate increases by 3. This moves the blue block. (key_img:dra
        key_y = key_y + 3
        print("Down_key: "..key_y)
    end
end --end of for loop
end -- end of down direction
end -- end of down key

--Activates when the left key is pressed
if key == "left" then

    --Decreases right by 1 it it is greater than 0
    if right_direction > 0 then
        right_direction = right_direction - 1
    end

    --If left is less than 3, then up increases by 1
    if left_direction < 3 then
        left_direction = left_direction + 1
        print("Left: "..left_direction)

    --For loop for when up direction equals a certain number. This way I do not have to write t
    for rep = 1,3 do
        if left_direction == rep then
            --surface Key_img x coordinate decreases by 3. This moves the blue block. (key_img:dra
            key_x = key_x - 3
            print("Left_key: "..key_x)
        end
    end --end of for loop
    end -- end of left direction
end -- end of left key

--Activates when the right key is pressed
if key == "right" then

    --Decreases right by 1 it it is greater than 0
    if left_direction > 0 then
        left_direction = left_direction - 1
    end

    --If right is less than 3, then up increases by 1
    if right_direction < 3 then
        right_direction = right_direction + 1
        print("Right: "..right_direction)

    --For loop for when up direction equals a certain number. This way I do not have to write t
    for rep = 1,3 do
        if right_direction == rep then
            --surface Key_img x coordinate increases by 3. This moves the blue block. (key_img:dra
            key_x = key_x + 3
            print("right_key: "..key_x)
        end
    end --end of for loop
    end -- end of right direction
end -- end of right key

end -- end of key press function

--Function for drawing or showing images
function sol.main:on_draw(screen)

```

```
--Show the mouse image if true
if mouse == true then
    mouse_img:draw(screen)
end

--Show the key image if true
if key_image == true then
    key_img:draw(screen, key_x, key_y)
end

--Show the fade image if true
if fade == true then
    fade_img:draw(screen)
end

end -- end of draw function

--Mouse press coordinates. Use a simple paint program. I used Kolourpaint on Linux.
function sol.main:on_mouse_pressed(button,x,y)

--The red button shows if the left mouse button is pressed
if button == "left" then
    if (x > 210 and x < 268) and (y > 52 and y < 74) then
        mouse = true
    end
end -- end of left button

--The red button vanishes if the right mouse button is pressed
if button == "right" then
    if (x > 210 and x < 268) and (y > 52 and y < 74) then
        mouse = false
    end
end -- end of right button
end -- end of mouse press function

--An example on how to make a function. This makes the key and mouse image vanish when fade is active
function stop_draw()
    key_image = false
    mouse = false
end
```

## Breaking Down the Script

I will not mention anything related to how to use the draw function because that was explained last chapter.

### Key Pressed

The key press function is very easy to use.

```
function sol.main:on_key_pressed(key)
    if key == "The key you want" then
        -- something happens
    end
end
```

### EX:

This will activate if the key 'a' is pressed.

```

function sol.main:on_key_pressed(key)
--Activates images on key 'a'
  if key == "a" then
    key_image = true
    fade = true
    mouse = true
  end
end

```

## Set Pause

By default, the game pauses with the key 'd', but that is not the only way to pause the game. One can use `game:set_pause` to pause and unpause the game.

**EX:**

```

--pause the game
game:set_paused(true)

--unpause the game
game:set_paused(false)

```

## Changing Volume

Changing the volume is very easy. All one has to do is type `sol.audio.set_music/sound_volume(number or variable from 1 - 100)`.

**EX:**

```

--Change the volume
sol.audio.set_music_volume(20)
sol.audio.set_sound_volume(5)

--A variable called volume
local volume = 50
sol.audio.set_music_volume(volume)
sol.audio.set_sound_volume(volume)

```

## Playing Music & Sound

Play music/sound is as easy as changing the volume. All one has to do is type `sol.audio.play_music/sound("the name of the audio")`.

**EX:**

```

--Play music and sound
sol.audio.play_music("village")
sol.audio.play_sound("chest_appears")

```

## Making Your Own Function

I mentioned functions in chapter 6, but here is a review.

```

--Making function
function name_you_want(parameters)
  --What you want to happen
end

--Calling function
name_you_want()

```

**Example:**

```
function stop_draw()
    key_image = false
    mouse = false
end

fade_img:fade_in(100, stop_draw())
```

**Fade In and Out**

I would have covered fading in/out during the last chapter, but the key pressed was needed for a decent example. One takes the surface [fade\_img] and adds [fade\_in/out(delay time in milliseconds, A function)].

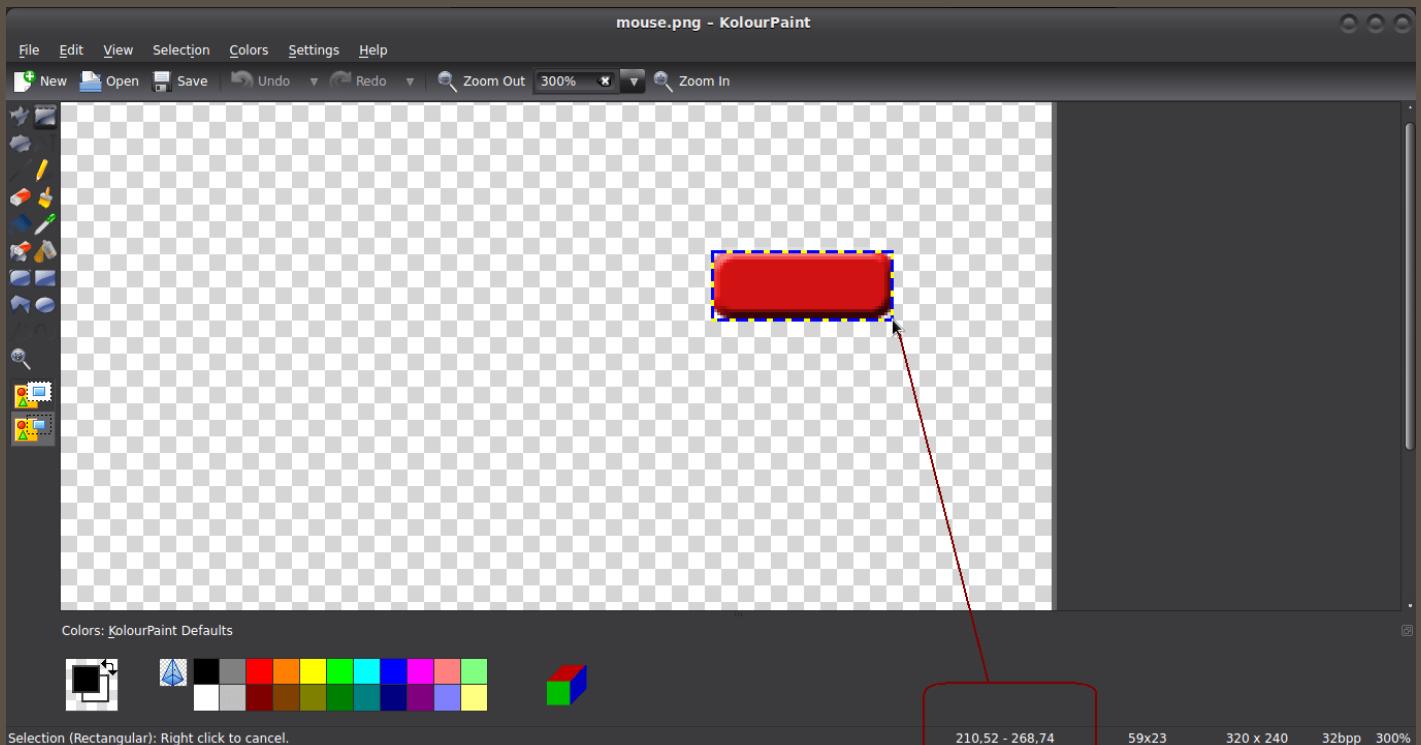
**EX:**

```
----Activates fade in on key 'i'
if key == "i" then
    fade_img:fade_in(100, stop_draw())
end

----Activates fade out on key 'o'
if key == "o" then
    fade_img:fade_out(100, stop_draw())
end
```

**Mouse Pressed**

The hardest part about the mouse press function is getting the coordinates. I use a program on the Linux operating system called KolourPaint, but you can do this in most paint programs.



The mouse press function works like this:

```
function sol.main:on_mouse_pressed(button,x,y)
    if button = "left/right/middle/" then
```

generated by haroopad

```

if (x > value and x < value) and (y > value and y < value) then
    -- something happens
end
end -- end of if button
end -- end of mouse press function

```

**EX:**

```

--Mouse press coordinates. Use a simple paint program. I used Kolorpaint on Linux.
function sol.main:on_mouse_pressed(button,x,y)

--The red button shows if the left mouse button is pressed
if button == "left" then
    if (x > 210 and x < 268) and (y > 52 and y < 74) then
        mouse = true
    end
end -- end of left button

--The red button vanishes if the right mouse button is pressed
if button == "right" then
    if (x > 210 and x < 268) and (y > 52 and y < 74) then
        mouse = false
    end
end -- end of right button
end -- end of mouse press function

```

## Chapter 10: Timers and Getting Coordinates

**Preview:**

[Lessons > Chapter\\_10 > Chapter\\_10\\_Click\\_drag\\_mouse\\_preview.ogv.zip](#)

### Basic Timer Map Script Example

```

--[]

-----
Instructions:
-----

Key "o" to stop some timers.

--]]
local map = ...
local game = map:get_game()

--[]
--Timer will go off (as long as you are still on the same map) in 5 seconds. 5000 milliseconds is
--Timer will not go off if you go to another map before 5 seconds are reached.
--If the game is paused with "d", then the timer is paused as well.
sol.timer.start(5000, function()
    sol.audio.play_sound("secret")
end)

--Using the "game" context or parameter will make the timer go off even though you leave the map.
--context (map, game, item, map entity, menu or sol.main; optional)
--
sol.timer.start(game, 5000, function()
    sol.audio.play_sound("secret")

```

```
end)
```

```
local sup_timer = sol.timer.start(5000, function()
    sol.audio.play_sound("secret")
end)
```

```
--Unsuspend the timer. You can do this if you do not want to use the game parameter or context.
--You can use suspend with map too: "timer:is_suspended_with_map(false)"
sup_timer:set_suspended(false)
```

```
-- Call a function half second.
sol.timer.start(500, function()
    sol.audio.play_sound("cane")
    return true -- To call the timer again (with the same delay).
end)
```

```
--]]
-- Call a function half second with a number value variable
--Press "o" to stop the timer
local variable_1 = 0
sol.timer.start(500, function()
    sol.audio.play_sound("cane")
    return variable_1 ~= 1
end)
--[[
```

```
-- Call a function half second with a true/false
--Press "o" to stop the timer
local on = true
sol.timer.start(500, function()
    sol.audio.play_sound("cane")
    return on == true
end)
```

```
-- Call a function ten times, with half second between each call.
local num_calls = 0
sol.timer.start(500, function()
    sol.audio.play_sound("bomb")
    num_calls = num_calls + 1
    return num_calls < 3
end)
--]]
```

```
--Solarus key pressing function
function sol.main:on_key_pressed(key)

if key == "o" then
    on = false
    variable_1 = 1
    print(variable_1)
end

end -- end of key press function
```

## Timer

```
sol.timer.start([context], delay (in milliseconds) , callback)
```

## Timer Function Delay

1. Timer will go off (as long as you are still on the same map) for 1 second in the example below.
2. There are 1000 milliseconds in 1 seconds.
3. Timer will not go off if you go to another map before 1 second is reached.
4. If the game is paused with "d", then the timer is paused as well.

```
-- Play sound "secret" in one second.
local function play_secret_sound()
    sol.audio.play_sound("secret")
end

sol.timer.start(1000, play_secret_sound)
```

## Timer Anonymous Function Delay

This is a shorter form of the example before.

1. Timer will go off (as long as you are still on the same map) in 5 seconds.
2. There are 5000 milliseconds in 5 seconds.
3. Timer will not go off if you go to another map before 5 seconds are reached.
4. If the game is paused with "d", then the timer is paused as well.

```
sol.timer.start(5000, function()
    sol.audio.play_sound("secret")
end)
```

## Repeat Timer with True

This timer repeats after every half second.

```
sol.timer.start(500, function()
    sol.audio.play_sound("cane")
    return true -- To call the timer again (with the same delay).
end)
```

## Stop Repeating Timer With Variable

```
-- Call a function with a number value variable
--Press "o" to stop the timer
local variable_1 = 0
sol.timer.start(500, function()
    sol.audio.play_sound("cane")
    return variable_1 ~= 1
end)

-- Call a function with a true/false
--Press "o" to stop the timer
local on = true
```

```

sol.timer.start(500, function()
    sol.audio.play_sound("cane")
    return on == true
end)

--Solarus key pressing function
function sol.main:on_key_pressed(key)

if key == "o" then
    on = false
    variable_1 = 1
    print(variable_1)
end

end -- end of key press function

```

## Repeat Timer a Number of Times

```

-- Call a function ten times, with half second between each call.
local num_calls = 0
sol.timer.start(500, function()
    sol.audio.play_sound("bomb")
    num_calls = num_calls + 1
    return num_calls < 3
end)

```

## Context Timer

1. Using the "game" context or parameter will make the timer go off even though you leave the map.
2. Context (map, game, item, map entity, menu or sol.main; optional)

```

sol.timer.start(game, 5000, function()
    sol.audio.play_sound("secret")
end)

```

## Timer Display Drag Click Script Example

```

--[ [
-----
Instructions:
-----
Key "m" is for moving the cursor image around on the screen. You can turn off the cursor and use i
Key "c" makes it so one can click the cursor around (Press "m" first because "c" will stop the tim
Key "t" displays the timer
Key "a" sets the timer to 20 seconds.
=====
--]

--Pass the game parameter to the script.
local game = ...

--Creating surfaces for the images and loading in the images.
local arrow_img = sol.surface.create ("arrow.png")

--Makes the timer display, clicking the mouse around, and moving the mouse not being at start of s
local click_mouse_xy = false
local move_mouse_xy = false
local time_display = false

```

generated by haroopad

```
--Variables for the timer display
local milliseconds = 0
local seconds = 0
local minutes = 0
local hours = 0

--MOVES IMAGE AROUND SCREEN BY CLICKING VARIABLES
--A table because I like tables and it prevents upvalue errors
local click = {

    place_x,
    place_y,
}

--DRAG IMAGE AROUND THE SCREEN VARIABLES
--A table because I like tables and it prevents upvalue errors
local move = {

    place_x,
    place_y,
}

--Function for mouse releases
function sol.main:on_mouse_released(button, x, y)

--MOVES IMAGE AROUND SCREEN BY CLICKING.
if click.mouse_xy == true then
    if button == "left" then
        --Setting x,y drag to get mouse position.
        local drag_x,drag_y = sol.input.get_mouse_position()
        click.place_x = drag_x
        click.place_y = drag_y
        print("x,",click.place_x, "y",click.place_y)
    end
end
end

--DRAG IMAGE AROUND THE SCREEN
local num_calls = 0
--Assign variable "test" to the timer
local test = sol.timer.start(80, function()
    --Setting x,y drag to get mouse position.
    local drag_x,drag_y = sol.input.get_mouse_position()
    move.place_x = drag_x
    move.place_y = drag_y
    print("x,",move.place_x, "y",move.place_y)
    return num_calls ~= 1
end)

--Solarus key pressing function
function sol.main:on_key_pressed(key)
    --Pause the game
    game:set_paused(true)

    if key == "c" then
        click.mouse_xy = true
        move.mouse_xy = false
        --Stop the timer "test"
        test:stop()
        --Timer "test" sound is off
        test:set_with_sound(false)
    end

```

```

if key == "m" then
    click_mouse_xy = false
    move_mouse_xy = true
    --Timer "test" sound is on
    --The timer will be very fast and annoying at 80 milliseconds. It will go slower with like 500
    test:set_with_sound(true)
end

if key == "t" then
    --Turn on the timer display
    time_display = true
end

if key == "a" then
    --Change the time to 20 seconds
    test:set_remaining_time(20000)
end
end -- end of key press function

--The draw function for showing images
function sol.main:on_draw(screen)

--Basic time calculations. There is no int (integer) declaration for variables in Lua, so I used numbers
--Math.floor rounds down and math.ceil rounds up math.floor(0.5) = 0 and math.ceil(0.5) = 1
seconds = math.floor((milliseconds / 1000) % 60)
minutes = math.floor((milliseconds / (1000*60)) % 60)
hours   = math.floor((milliseconds / (1000*60*60)) % 24))

--TEXT ON SCREEN
--http://www.solarus-games.org/doc/latest/lua_api_text_surface.html
local text_display = sol.text_surface.create({ -- name a local variable something and assign it
    font = "minecraftia", -- font name
    text = hours..":"..minutes..":"..seconds..":"..milliseconds, -- text you want to show
    rendering_mode = "solid", -- "solid" (faster) and default
    color = {0,0,0}, -- color must be in a table RGB (http://www.rapidtables.com/web/color/RGB_color.html)
})
}

if move_mouse_xy == true then
    --Get the remaining time of the timer "test." This is displayed at the end, "text = hours..":"..minutes..":"..seconds..":"..milliseconds = test:get_remaining_time()
    print(test:get_remaining_time())
end

if click_mouse_xy == true then
    arrow_img:draw(screen, click.place_x,click.place_y)
end

if move_mouse_xy == true then
    arrow_img:draw(screen, move.place_x,move.place_y)
end

if time_display == true then
    text_display:draw(screen, 20,20)
end

end --end of draw function

```

## Stopping a Timer

Stopping a timer is simple.

1. Assign a variable to the timer. "local timer" in this case.

```
local timer = sol.timer.start(game, 5000, function()
    sol.audio.play_sound("secret")
end)
```

1. variable:stop(). "timer" is the variable.

```
timer:stop()
```

## Stop all Context Timers

1. Context (map, game, item, map entity, menu or sol.main; optional)

```
sol.timer.stop_all(context)
```

## Getting Remaining Time

This is very useful for when wanting to display a time count down.

```
variable:get_remaining_time()
```

```
local milliseconds

local test = sol.timer.start(game, 5000, function()
    sol.audio.play_sound("secret")
end)

test:get_remaining_time()
print(test:get_remaining_time())

milliseconds = test:get_remaining_time()
```

## Displaying Timer

```
--The draw function for showing images
function sol.main:on_draw(screen)

--Basic time calculations. There is no int (integer) declaration for variables in Lua, so I used math.floor and math.ceil
--Math.floor rounds down and math.ceil rounds up math.floor(0.5) = 0 and math.ceil(0.5) = 1
seconds = math.floor((milliseconds / 1000) % 60)
minutes = math.floor((milliseconds / (1000*60)) % 60)
hours   = math.floor((milliseconds / (1000*60*60)) % 24)

--TEXT ON SCREEN
--http://www.solarus-games.org/doc/latest/lua_api_text_surface.html
local text_display = sol.text_surface.create({ -- name a local variable something and assign it to the function
    font = "minecraftia", -- font name
    text = hours..":"..minutes..":"..seconds..":"..milliseconds, -- text you want to show
    rendering_mode = "solid", -- "solid" (faster) and default
    color = {0,0,0}, -- color must be in a table RGB (http://www.rapidtables.com/web/color/RGB_color.htm)
})

--Get the remaining time of the timer "test." This is displayed at the end, "text = hours..":"..minutes..":"..seconds..":"..milliseconds = test:get_remaining_time()
print(test:get_remaining_time())

--Draw text display
text_display:draw(screen, 20,20)

end --end of draw function
```

## Timer Sound

```
Variable:set_with_sound(true/false)
```

These uses built in sound names:

1. timer\_hurry
2. timer

The sound will go slower the more seconds. It will go faster with only a few milliseconds.

```
local test = sol.timer.start(game, 5000, function()
    sol.audio.play_sound("secret")
end)

test:set_with_sound(true)
```

## Change Remaining Timer

You can change the remaining time.

```
variable:set_remaining_time([time in milliseconds[]])
```

```
local test = sol.timer.start(game, 5000, function()
    sol.audio.play_sound("secret")
end)

test:set_remaining_time(20000)
```

## Suspend Timer

When you pause the game with **D**, the timer pauses too. You can unsuspend the timer, so the timer still goes off for when the game is paused.

```
variable:set_suspended(false/true)
```

or

```
variable:is_suspended_with_map(false/true)
```

```
local sup_timer = sol.timer.start(5000, function()
    sol.audio.play_sound("secret")
end)
```

```
--Unsuspend the timer. You can do this if you do not want to use the game parameter or context.
--You can use suspend with map too: "sup_timer:is_suspended_with_map(false)"
sup_timer:set_suspended(false)
```

## Get Coordinates

### Getting Cursor Location

You must assign variables to the following to get coordinates of the cursor.

```
sol.input.get_mouse_position()
```

```
local x,y = sol.input.get_mouse_position()
```

## Applying Coordinates To Images

```
sol.input.get_mouse_position()
```

```
local x,y = sol.input.get_mouse_position()
```

--The draw function for showing images

```
function sol.main:on_draw(screen)
    arrow_img:draw(screen, x,y)
end
```

## Using Timer To Drag Image

```
local drag_x
local drag_y

--DRAG IMAGE AROUND THE SCREEN
local num_calls = 1
--Assign variable "test" to the timer
local test = sol.timer.start(80, function()
    --Setting x,y drag to get mouse position.
    local x,y = sol.input.get_mouse_position()
    x = drag_x
    y = drag_y
    print("x",x, "y",y)
    num_calls = num_calls + 1
    return num_calls ~= 1
end)
```

--The draw function for showing images

```
function sol.main:on_draw(screen)
    arrow_img:draw(screen, drag_x,drag_y)
end
```

### Documentation:

Check the [documentation](#) for more information on timers.

### Timer Project Sample(s)

[Lessons > Chapter\\_10 > Chapter\\_10\\_Timer.zip](#)

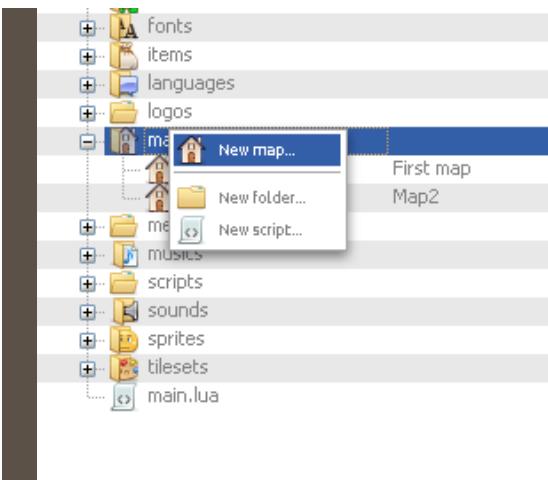
[Lessons > Chapter\\_10 > Chapter\\_10\\_Click\\_drag\\_mouse.zip](#)

## Chapter 11: Map editor

### Making a Map

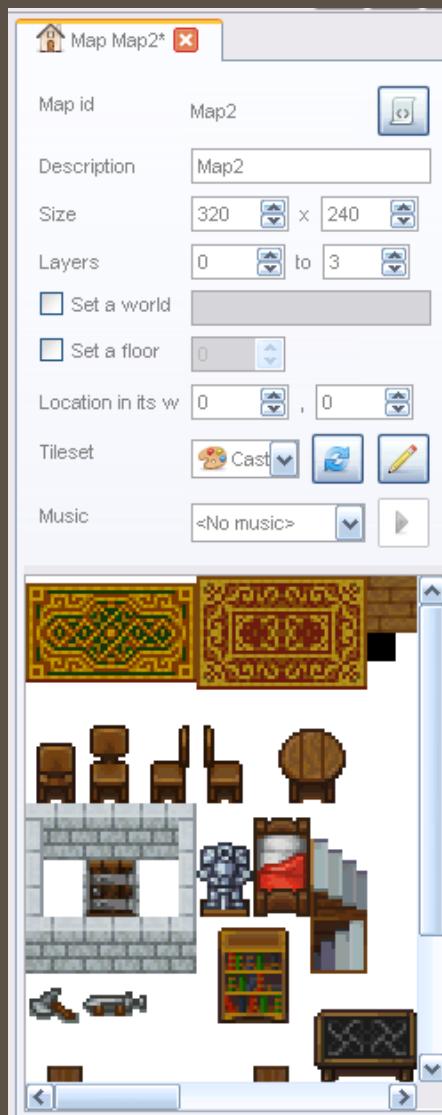
Right click the Map directory folder and give the map/description a name.

generated by haroopad



## Map Properties

By default, the map size is 320 x 240 and change the layer amount to 3 or more if you desire. I will go over these options in more detail soon.



## Map Script

You can open the map script by right clicking on the map name in the directory or by going to this icon.

The screenshot shows the Solarus ARPG Game Development Book 2 interface. At the top, there is a navigation bar with a back arrow, a search icon, and a user profile icon. Below the navigation bar, the title "Map Property Features" is displayed. The main content area contains a numbered list of 8 features, each with a corresponding screenshot of the Solarus map editor's properties panel.

1. One can change the description of the map. I changed it to `book map`.
2. The `size` is how large the map will be. By default it is 320 x 240.
3. The `layers` in Solarus are super amazing. One is not limited to 2 or 3 layers, many layers can be added. I set it to 0 to 3 layers.
4. One can `set a world`. Your maps are organized under the world name.
5. The `floors` are as the name suggests. They are the floors on your map
6. The location is the coordinates where your map is located in the world.
7. One can pick the tileset to use from the `tileset` option.
8. The `music` section is where one can set default music for the map.

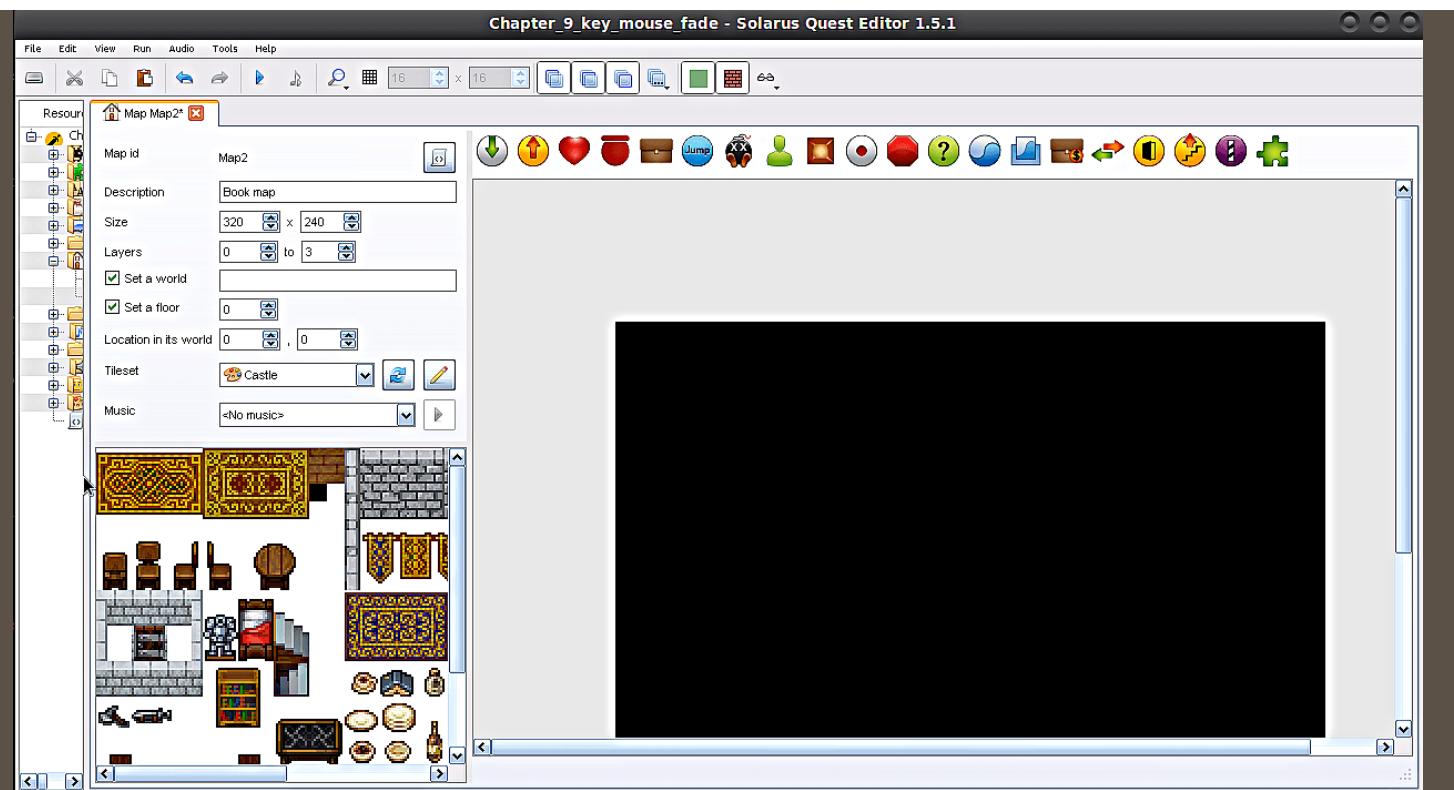
The properties panel shown in the screenshot includes the following settings:

- Description: Book map
- Size: 320 x 240
- Layers: 0 to 3
- Set a world: Checked
- Set a floor: Checked
- Location in its world: 0, 0
- Tileset: Castle
- Music: <No music>

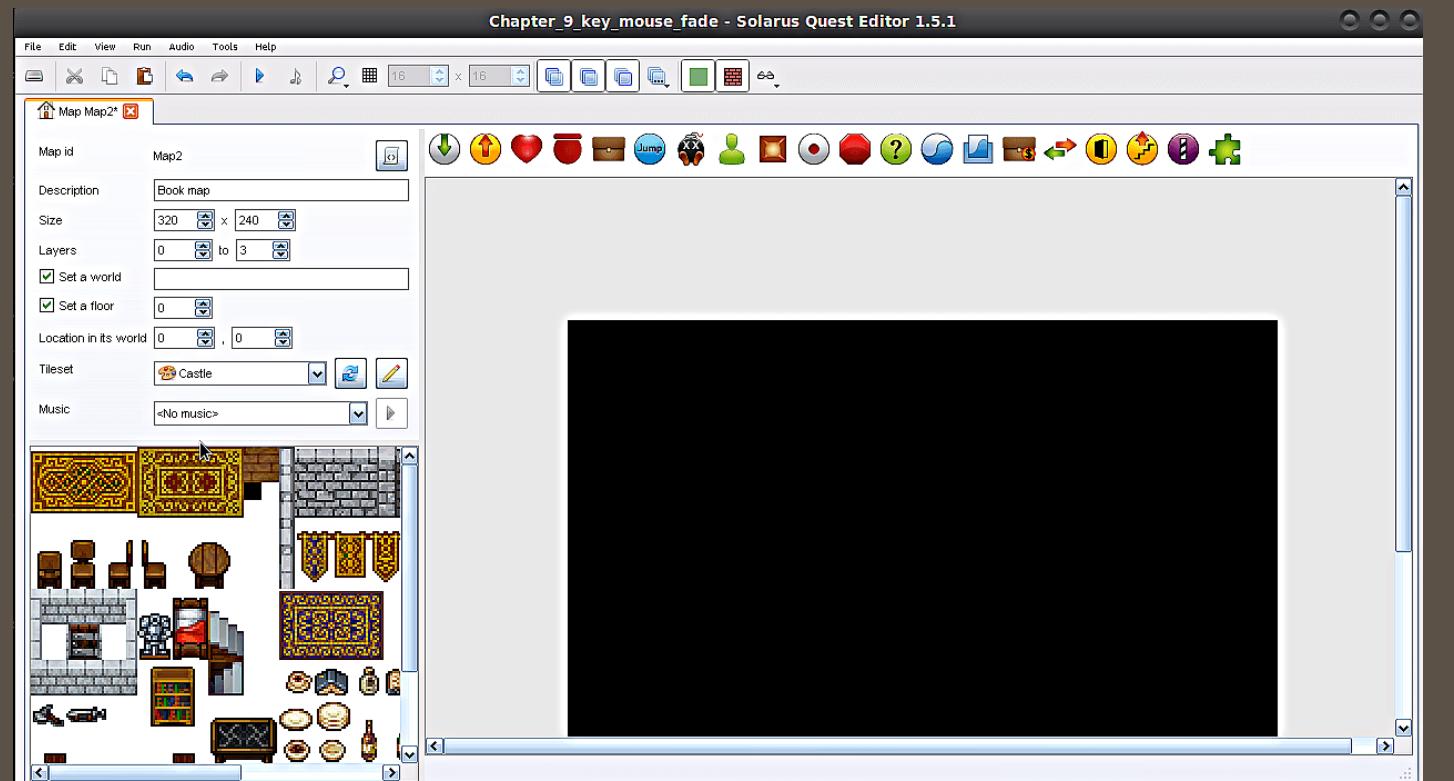
## Map Drag

One can drag the edges of the map editor for a better view of the tileset.

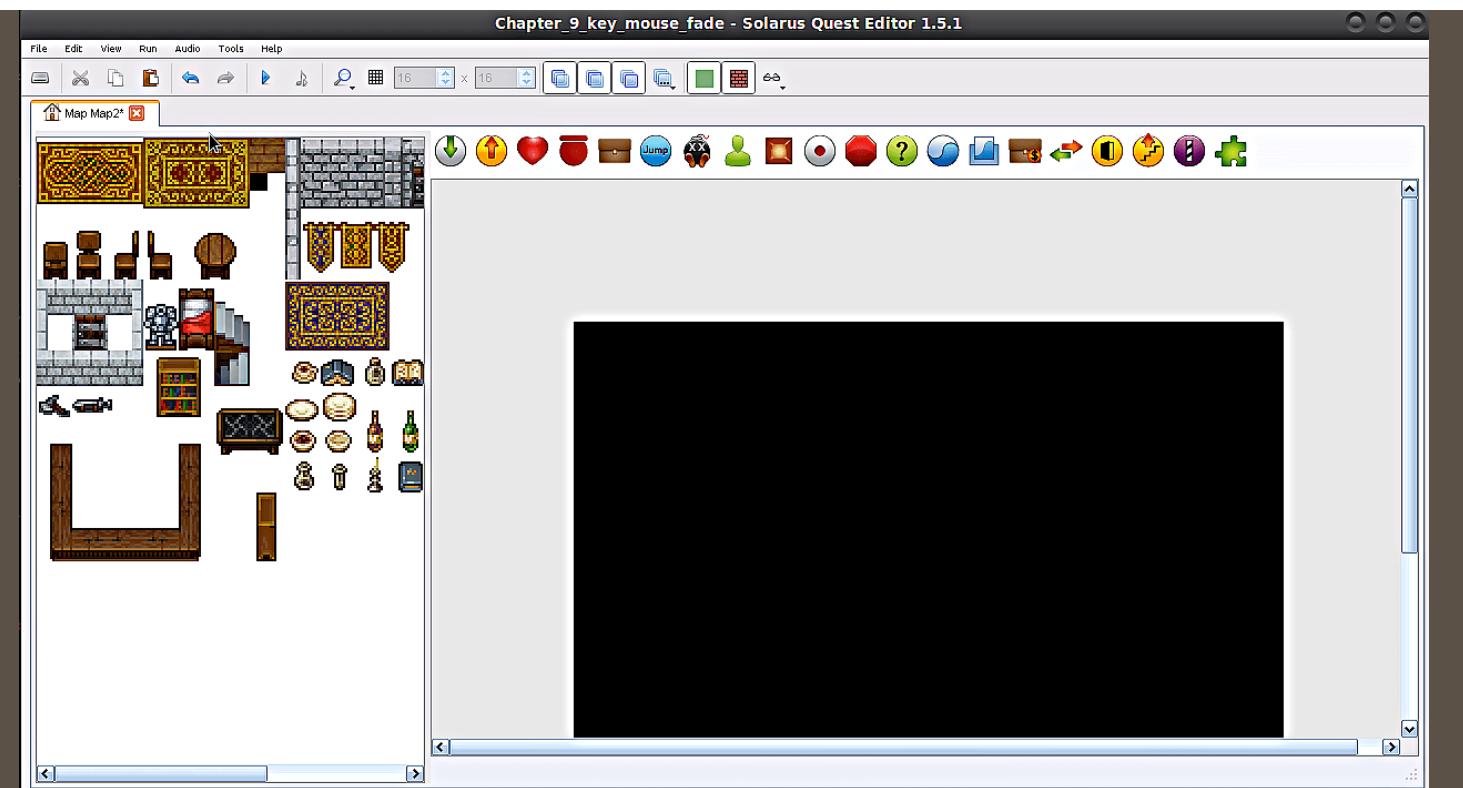
Clicking and dragging the left edge.



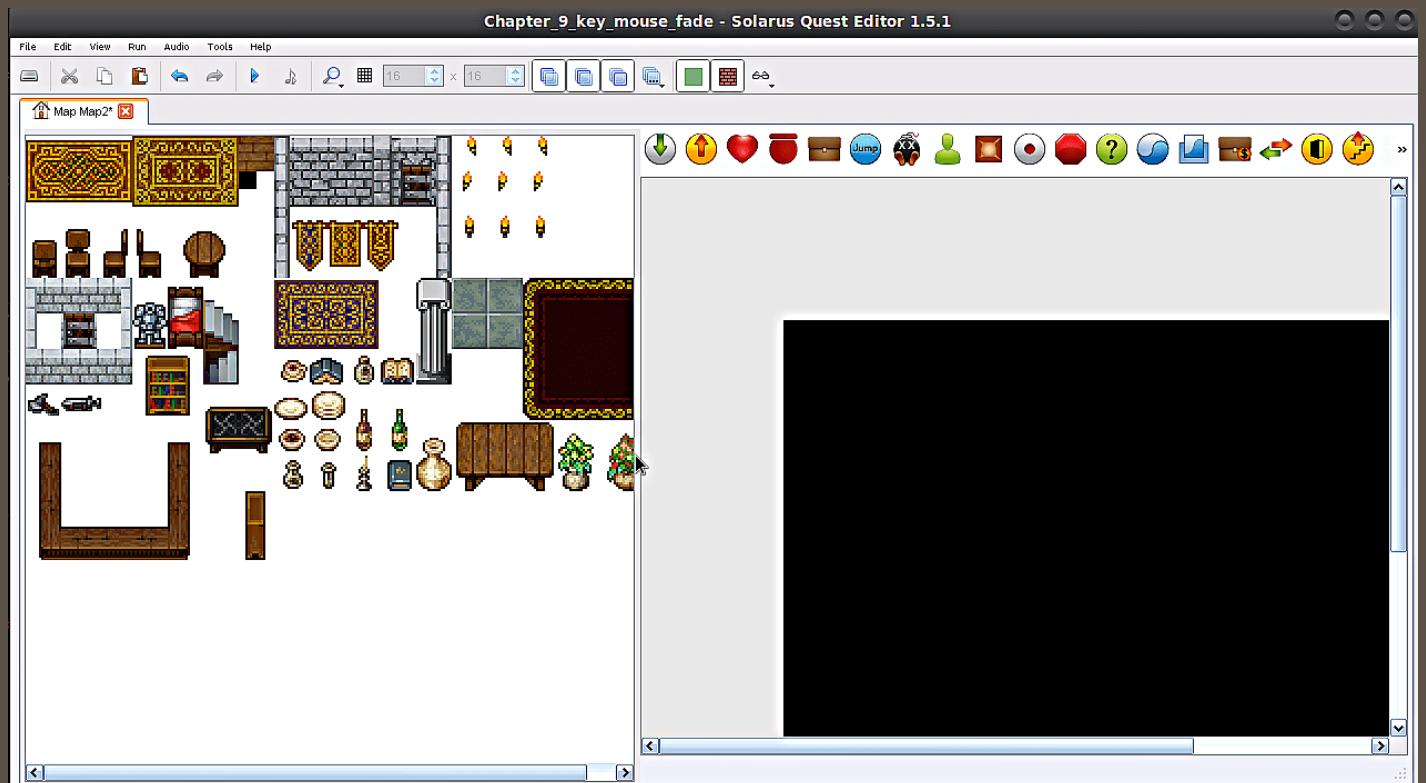
Clicking and dragging up.



Dragging up results in this.

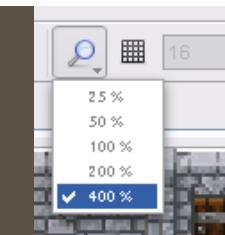


Click and drag the right edge.



## Map Zoom

You can zoom into the map with CTRL + Z or with the zoom icon. You can do the same where you select the tiles for the map.



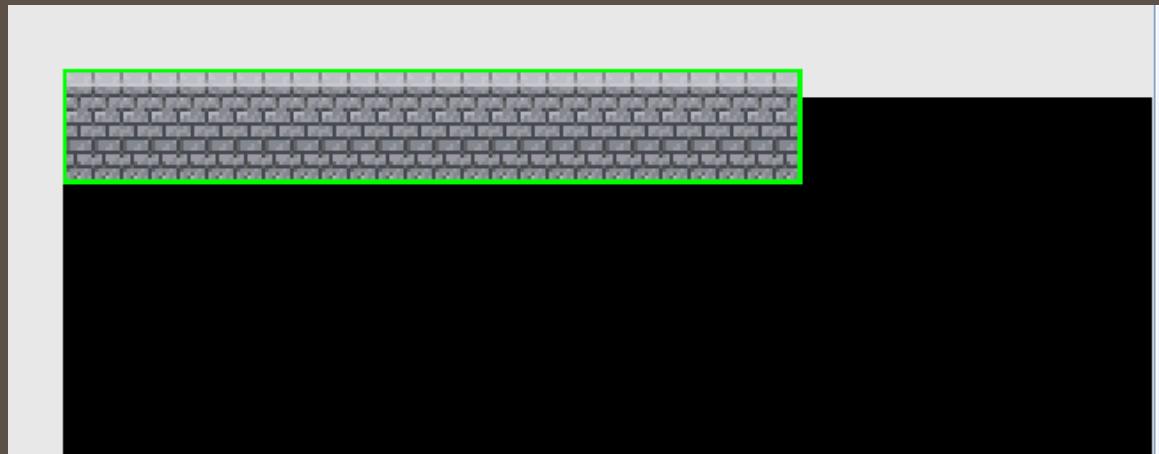
## Selecting Map Tiles

Click on the tile that has already been made in the tileset editor.



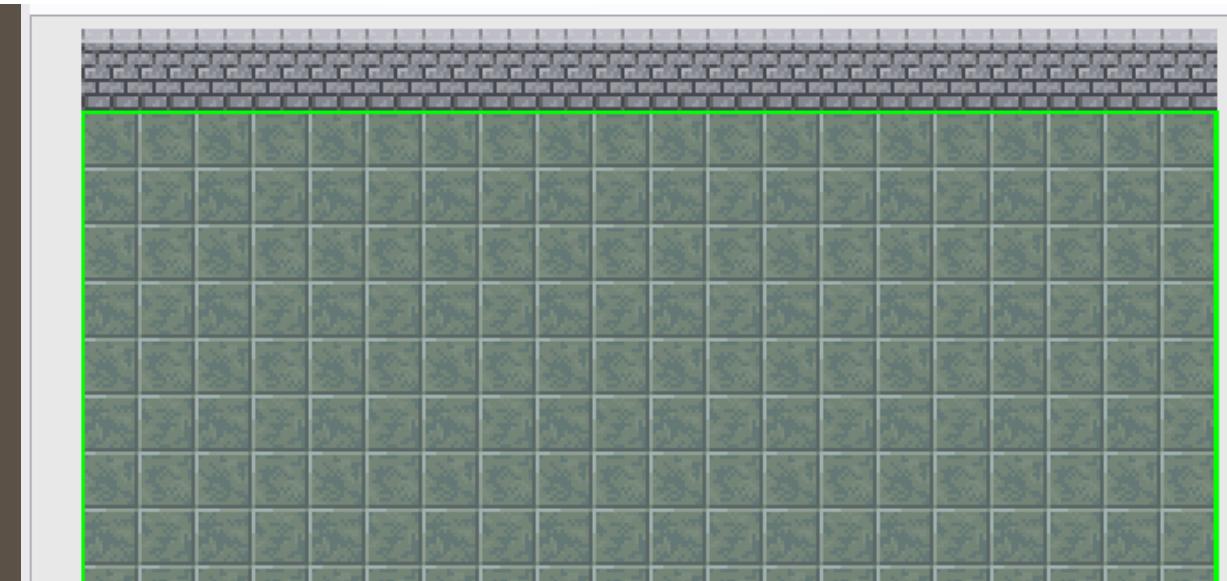
## Dragging Map Tile

Move the tile over to the map, click (Hold the click), and drag. One can resize it with the **R** shortcut key.

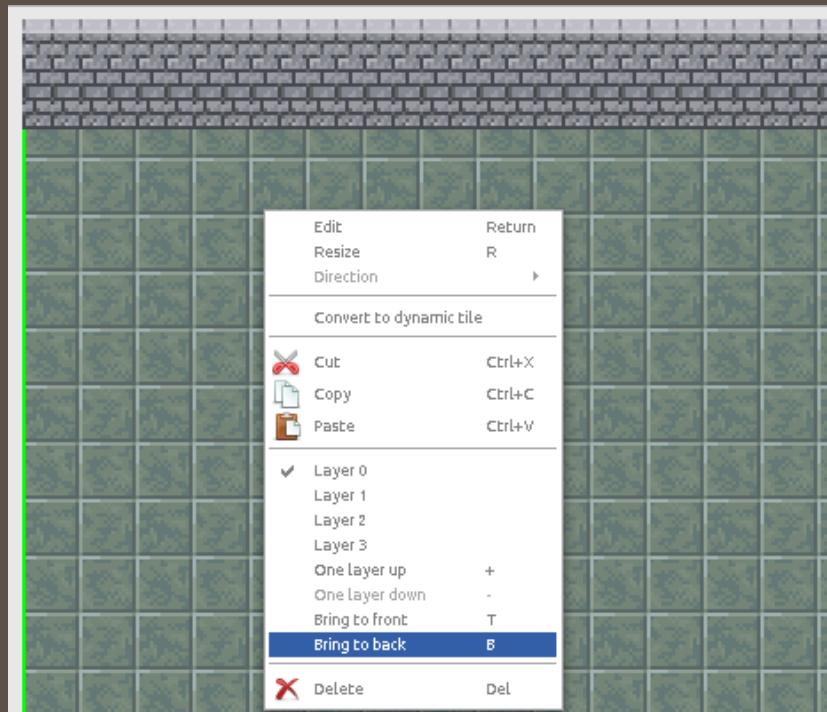


## Bring to Back Map Tile

As one can see in the image, the stone tile is overlapping the brick wall.



One needs to right click on the tile and select **bring to back** or click the shortcut key **B**.

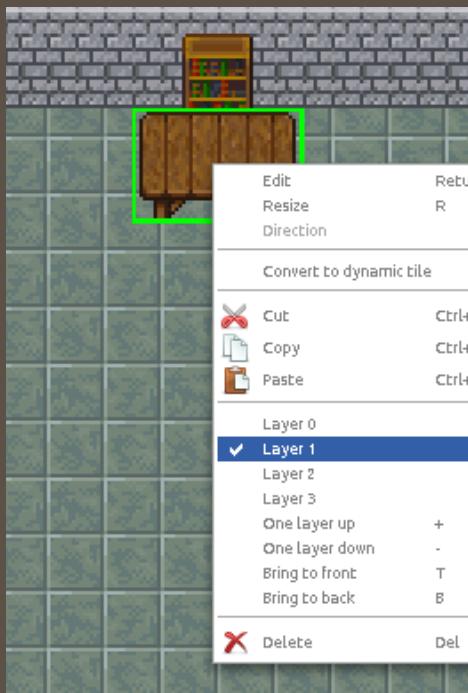


## Change Map Tile Layer

As one can see, the bookcase is on top of the table because they are both on layer 0.

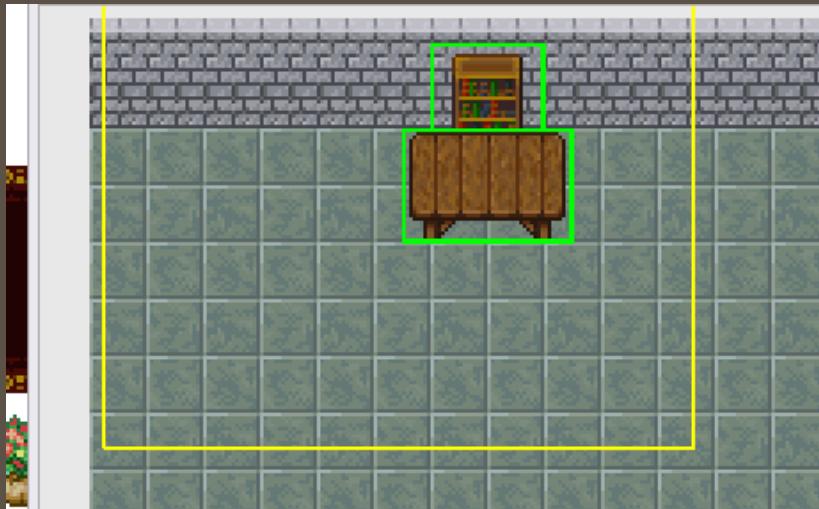


The table's layer needs to changed from layer 0 to layer 1. This can be done by right clicking on the table.



## Selecting Multiple Tiles

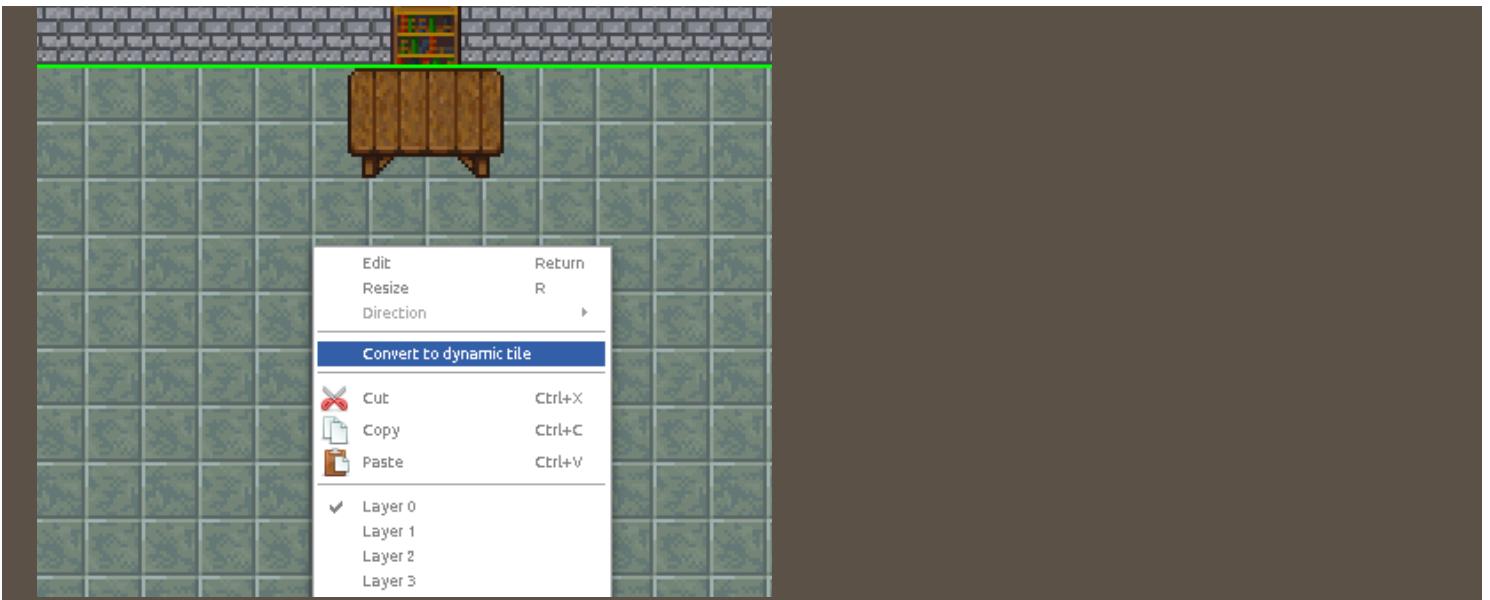
Selecting multiple tiles is very useful for when one needs to move many objects around. This can be done by pressing CTRL + dragging the mouse.



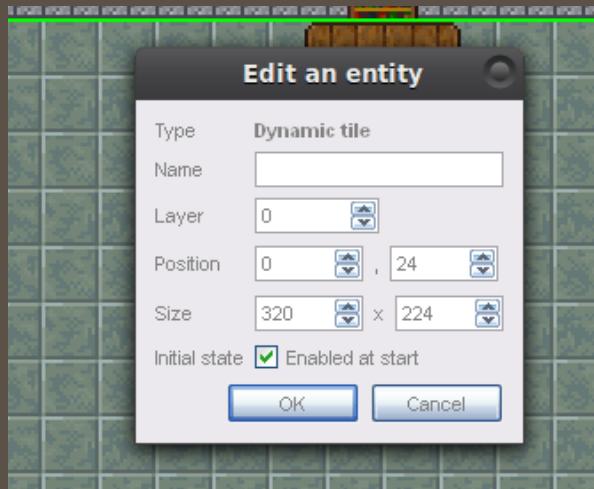
## Dynamic Tiles

Dynamic tiles can be manipulated by script. For example, making water vanish. The main difference from a normal tile is that one can add a name.

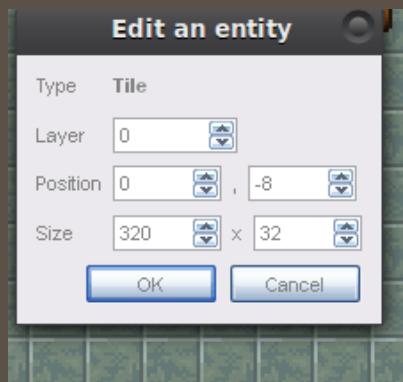
One can convert a normal tile to a dynamic tile like this.



Dynamic tile properties. One can get to this by selecting a tile and pressing **enter** or by right clicking and selecting **edit**.

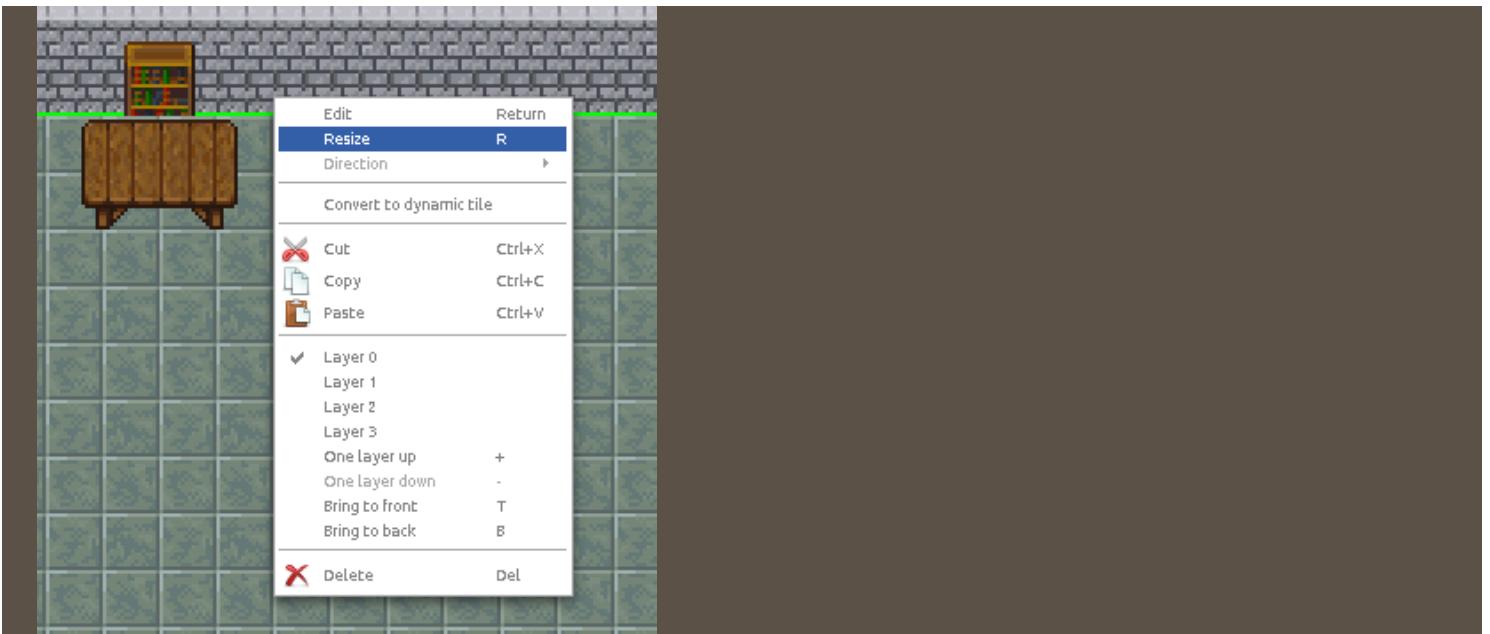


Normal tile properties. One can get to this by selecting a tile and pressing **enter** or by right clicking and selecting **edit**.



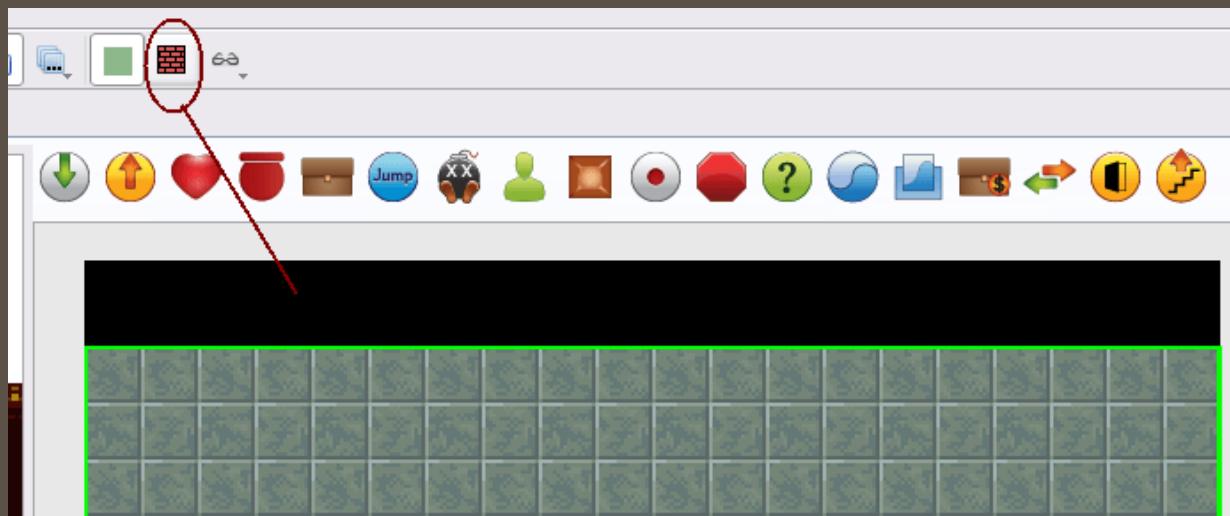
## Resizing Tiles

I mentioned resizing tiles before, but who does not like pictures? One can right click and select resize or by using the key shortcut **R**.



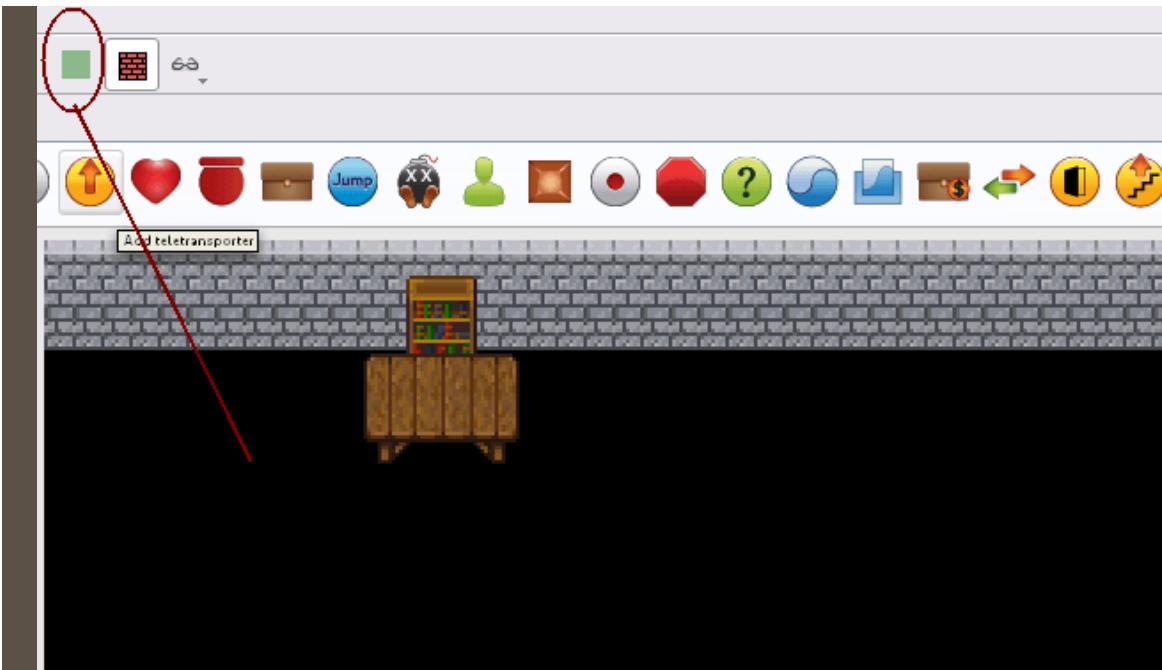
### Making Obstacles Vanish

One can select the remove obstacle brick wall icon on the icon bar to make obstacles vanish. Anything that does block the player's path is an obstacle. The brick wall, table, and bookcase were obstacles.



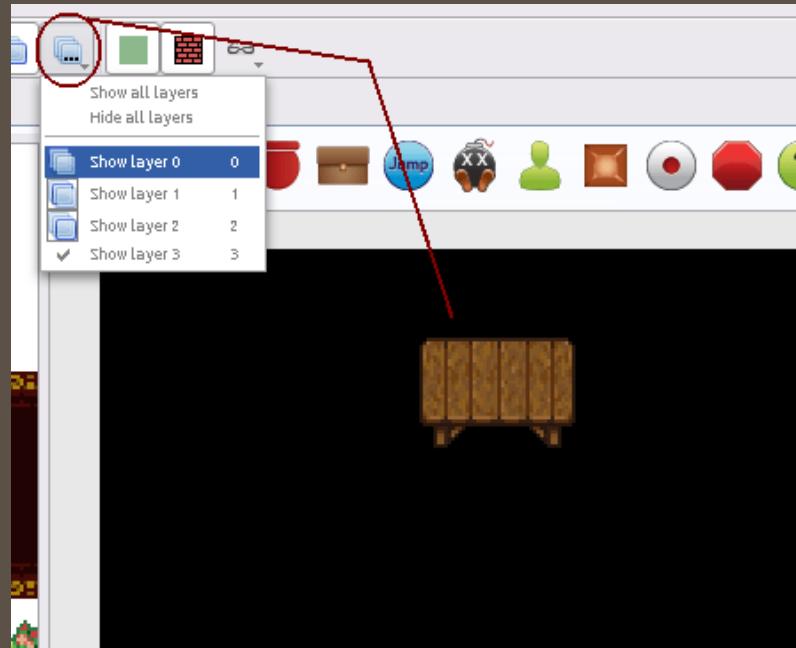
### Making Traversable Vanish

One can select the remove traversable grass icon on the icon bar to make traversable tiles vanish. Anything that does not block the player's path is a traversable tile. The stone floor tile was traversable.



### Making Layer[num] Vanish

One can select the remove layer number icon on the icon bar to make layer numbers vanish. Everything, but the table in the images vanishes because I selected layer 0. The table is on layer 1.



## Chapter 12 Menus and Window Options

### Preview:

[Lessons > Chapter\\_12 > Chapter\\_12\\_menu\\_preview.gif](#)

### Window Options

The window is related to video. Not playing a video clip, but the window.

generated by haroopad

## Video Mode

```
sol.video.set_mode("video_mode")
```

There are a few options for the `video_mode`.

<b>"video_mode"</b>	<b>Description</b>
"normal" (default):	The quest screen is unchanged. In windowed mode, the image is stretched by a factor of 2.
"scale2x":	The quest screen is scaled by a factor of 2 with the Scale2X algorithm.
"hq2x":	The quest screen is scaled by a factor of 2 with the hq2x algorithm.
"hq3x":	The quest screen is scaled by a factor of 3 with the hq3x algorithm.
"hq4x":	The quest screen is scaled by a factor of 4 with the hq4x algorithm.

### Example:

The "hq3x mode" is bigger than the "normal" mode.

```
sol.video.set_mode("hq3x")
```



```
sol.video.set_mode("normal")
```

generated by haroopad

**Sample quest - Solarus 1.5.0****Fullscreen**

```
sol.video.set_fullscreen(true/false)
```

The screen is full at "true" and default at "false."

```
sol.video.set_fullscreen(true)
```



## Cursor Visibility

```
sol.video.set_cursor_visible(true/false)
```

The cursor is the arrow on the screen that is moved with the mouse. Your cursor will not show if it is set to false.

```
sol.video.set_cursor_visible(false)
```

## Window Size

```
sol.video.set_window_size(width, height)
```

Window size sets the dimensions of the window.

### Example:

```
sol.video.set_window_size(500, 100)
```



## Default Window

generated by haroopad

This defaults the window to normal size.

```
sol.video.reset_window_size()
```

## Menu

Making a menu is quite simple. Menus are needed because one cannot call same parameter or context at the same time from different scripts. The first parameter or context would simply overwrite the newly called parameter or context. That is why menus are needed.

Context/parameter (game, sol.main, etc)

**In chapter\_12.lua:**

All the functions use the menu name instead of sol.main or game. In this case the menu is called, "menu."

```
function menu:on_draw(screen)
end -- end of draw function
```

Instead of:

```
function sol.main:on_draw(screen)
end -- end of draw function
```

or

```
function game:on_draw(screen)
end -- end of draw function
```

Example:

```
--A table or the require loading function
local menu = {}

function menu:button_menu(game)

    --Function menu for drawing or showing images
    function menu:on_draw(screen)

        end -- end of draw function
    end

    return menu
end
```

## Menu function

As you saw in the example above there is a function called menu:button\_menu(game). That function will be placed in the game\_manger.lua.

You can name the function any way you want after "menu:" menu:name\_you\_want(pass parameter).

# Menu: Starting and Stopping

## Starting Menu

```
sol.menu.start(context/parameter, menu_name, on_top(true/false))
```

Context/parameter (map, game or table)

Example:

```
sol.menu.start(game, menu, true)
```

## Stopping Menu

```
sol.menu.stop(menu_name)
```

```
sol.menu.stop(menu)
```

## Game\_manager.lua - Example:

```
local menu = require("scripts/chapter_12.lua")

function game_manager:start_game()

--Starting required script
menu:button_menu(game)

function game:on_key_pressed(key)
    if key == "m" then
        sol.menu.start(game, menu)
        print("m")
    end

    if key == "s" then
        sol.menu.stop(menu)
    end
end
game:start()
end
```

## Menu Script Example

[Game\\_manager.lua](#)

```
local game_manager = {}
local on = true

local menu = require("scripts/chapter_12.lua")
local initial_game = require("scripts/initial_game")
require("scripts/menus/alttp_dialog_box")

-- Starts the game from the given savegame file,
-- initializing it if necessary.

function game_manager:start_game()
    local exists = sol.game.exists("save1.dat")
    local game = sol.game.load("save1.dat")
    if not exists then
```

```
-- Initialize a new savegame.
game:set_max_life(12)
game:set_life(game:get_max_life())
game:set_ability("lift", 2)
game:set_ability("sword", 1)
end

menu:button_menu(game)

function game:on_key_pressed(key)
if key == "m" then

    if on == true then
        sol.menu.start(game, menu)
        print("m")
        on = false
    end
end

if key == "s" then
    sol.menu.stop(menu)
    on = true
end
end

game:start()

local hero = game:get_hero()
hero:set_tunic_sprite_id("main_heroes/eldran")
end

return game_manager
```

### Chapter\_12.lua

```
--[]

-----
Instructions:
-----

=====
Press keys "up" and "down"
=====

--]]

local menu = {}

function menu:button_menu(game)

--Creating surfaces for the images and loading in the images.
local overlay = sol.surface.create ("overlay.png")
local button_menu = sol.surface.create ("menu.png")
local down_up = 0
local direction = 0

--Function for drawing or showing images
function menu:on_draw(screen)
    overlay:draw(screen,0,down_up)
    button_menu:draw(screen)
end -- end of draw function

--Solarus key pressing function
function menu:on_key_pressed(key)
--Pause the game
```

```

game:set_paused(true)

if key == "up" then
    if direction > 0 then
        direction = direction - 1
        down_up = down_up - 27 - 0.7
        print("UP: ..direction.." down_up: "..down_up)
    end
end

if key == "down" then
    if direction < 6 then
        direction = direction + 1
        down_up = down_up + 27 + 0.7
        print("Down: ..direction.." down_up: "..down_up)
    end
end

if key == "e" then
    if direction == 0 then
        print("Item Menu ON!")
    end

    if direction == 1 then
        print("Skills Menu ON!")
    end

    if direction == 2 then
        print("Equipment Menu ON!")
    end

    if direction == 3 then
        print("Status Menu ON!")
    end

    if direction == 4 then
        print("Formation Menu ON!")
    end

    if direction == 5 then
        print("Save Menu ON!")
    end

    if direction == 6 then
        print("Logout Menu ON!")
    end
end -- end of key press function

end --end of button menu function

return menu

```

## Documentation:

Check the [documentation](#) for more information on menus.

## Menu Project Sample:

[Lessons > Chapter\\_12 > Chapter\\_12\\_menu.zip](#)

# Chapter 13: Entities

You may use the `inside_store` map in the samples ([Lessons > Chapter\\_13\\_Entities > inside\\_store.zip](#)) to help you with this learning process.

The inside store contains pre-made maps. These can be selected, copied, and pasted to your maps to help you do the entity chapter. Also, there are graphics included in the samples.



## get\_() & related

If you have not already noticed, then there are ways to get certain functions in scripts.

For a map script to use map functions it needs `local map = ...` and the map can get game functions with `map:get_game()`.

When simply loading external scripts one needs `local game = ...` to use game functions in it, but it is better to use `require()` for external scripts and pass the game through a function like `function game_over_menu:start(game)`. One can use `local map = game:get_map()` to get the map functions in external scripts.

I will use the map script as an example .

### **Some ways to get certain functions are:**

```
get_map()
get_sprite()
get_game()
get_hero()
```

`map:get_game()`

generated by haroopad

If you wanted to use game functions on a map, then you use the following line of code.

```
local game = map:get_game()
```

You can then use a function like:

```
game:set_starting_location("Map_4", "starting_destination") -- Starting location.
```

`map:get_hero()`

If you wanted to use hero functions on a map, then you use the following line of code.

```
local hero = map:get_hero()
```

You can then use a function like:

```
hero:set_animation("dead") -- calls an animation you made for the hero.
```

`entity:get_sprite()`

`entity:get_sprite()` is a little different. You mostly use it when using **sprites methods**.

```
entity_name:get_sprite():set_animation("animation_name")
```

## Entity Position & Name

You can hover over entities to see their names and position.



## Types of entities

Here are the existing types of entities.

<b>Entity</b>	<b>Description</b>
Hero	The character controlled by the player.
Tile	A small brick that composes a piece of the map, with a pattern picked from the tileset.
Dynamic tile	A special tile that can be enabled or disabled dynamically (usual tiles are optimized away at runtime).
Teletransporter	When walking on it, the hero is transported somewhere, possibly on the same map or another map.
Destination	A possible destination place for teletransporters.
Pickable treasure	A treasure placed on the ground and that the hero can pick up.
Destructible object	An entity that can be cut or lifted by hero, and that may hide a pickable treasure.
Carried object	A destructible object lifted and carried by the hero.
Chest	A chest that contains a treasure.
Shop treasure	A treasure that the hero can buy for a price.
Enemy	A bad guy (possibly a boss) who may also drop a pickable treasure when killed.
Non-playing character (NPC)	Somebody or something the hero can interact with.
Block	An entity that the hero can push or pull.
Jumper	When walking on it, the hero jumps into a direction.
Switch	A button or another mechanism that the hero can activate.
Sensor	An invisible detector that detects the presence of the hero.
Separator	An horizontal or vertical separation between two parts of the map.
Wall	An invisible object that stops some kinds of entities.
Crystal	A switch that lowers or raises crystal blocks.
Crystal block	A low wall that can be lowered (traversable) or raised (obstacle) using a crystal.
Stream	When walking on it, the hero automatically moves into a direction.
Door	A door to open with an equipment item or another condition.
Stairs	Stairs between two maps or to a platform of a single map.
Bomb	A bomb that will explode after a few seconds and that may be lifted by the hero.
Explosion	An explosion that can hurt the hero and the enemies.
Fire	A flame that can hurt enemies and interact with other entities.
Arrow	An arrow shot by the bow.
Hookshot	A hookshot shot by the hero.

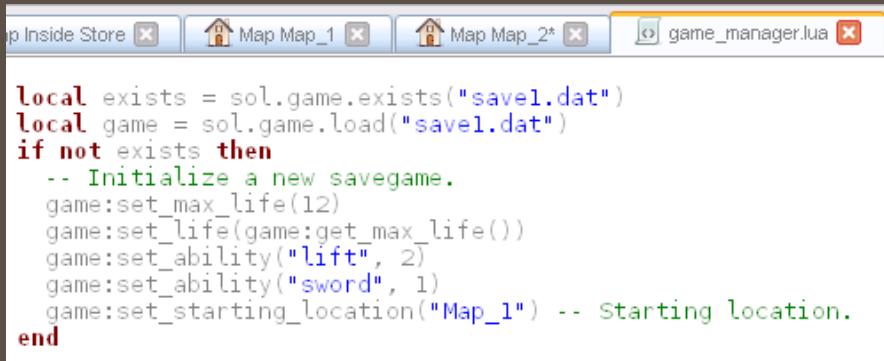
Entity	Description
Boomerang	A boomerang shot by the hero.
Camera	A rectangle that determines the visible area of the map.
Custom entity	An entity fully controlled by your Lua scripts.

## Destination Entity

### Setting up Maps and Starting Location

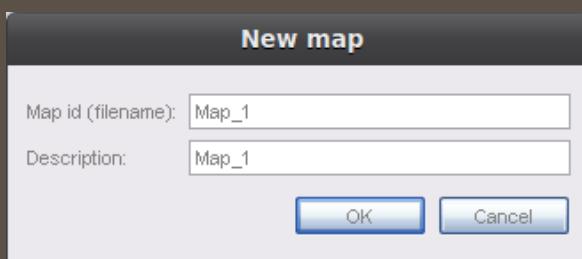
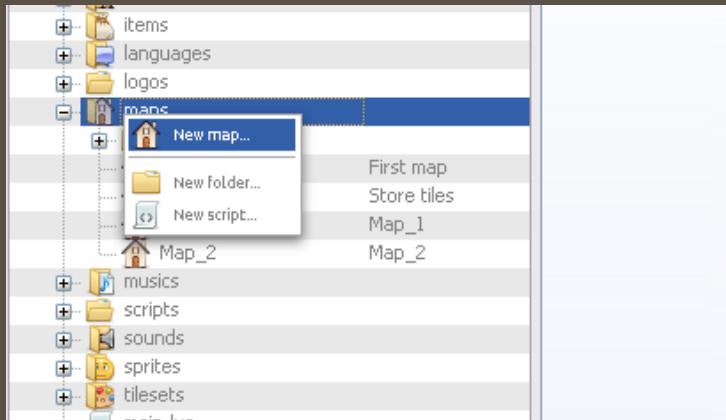
In `game_manager.lua` there is a game script for the starting location. It is set to `Map_1`.

```
game:set_starting_location("Map_1") -- Starting location.
```



```
local exists = sol.game.exists("savel.dat")
local game = sol.game.load("savel.dat")
if not exists then
    -- Initialize a new savegame.
    game:set_max_life(12)
    game:set_life(game:get_max_life())
    game:set_ability("lift", 2)
    game:set_ability("sword", 1)
    game:set_starting_location("Map_1") -- Starting location.
end
```

`Map_1` is a map that I created. You can name it that or something else.



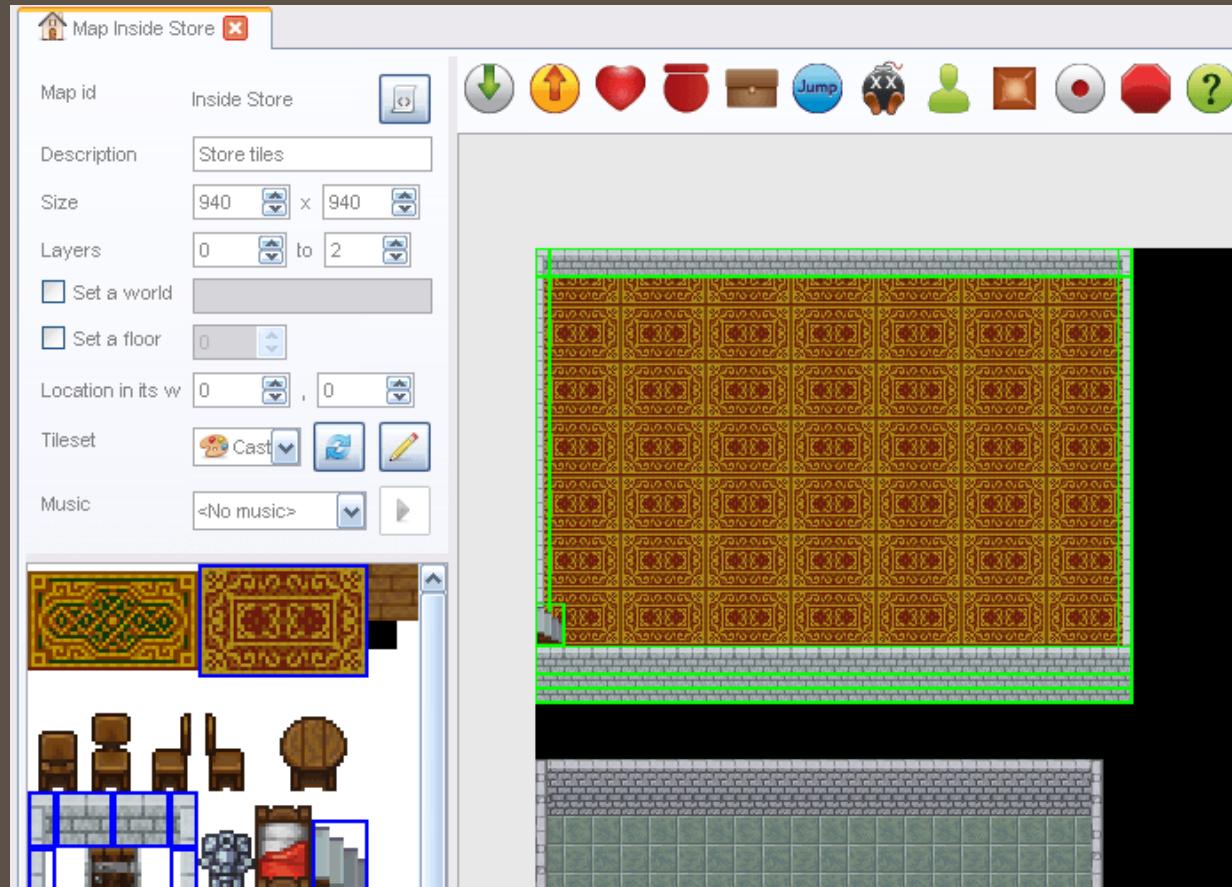
Make a map called `Map_2` as well.

### Inside Store

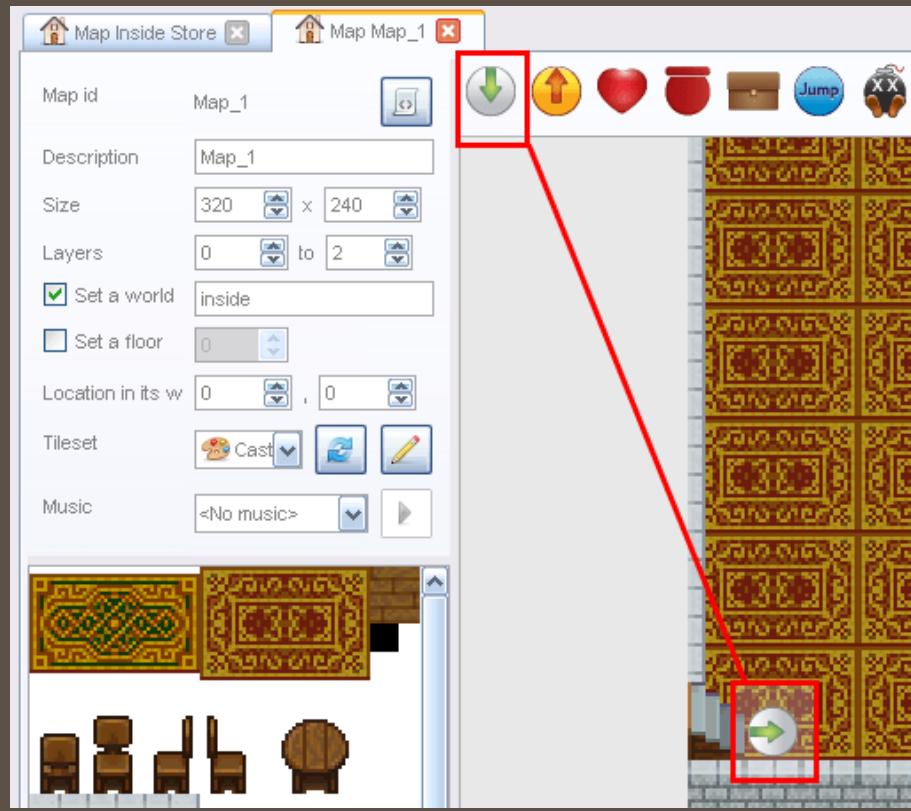
1. Copy the red mat floor room to `Map_1`.

## 2. Copy the gray tiled floor room to Map\_2.

You copy it by holding down the left mouse button and dragging or moving the mouse to highlight.

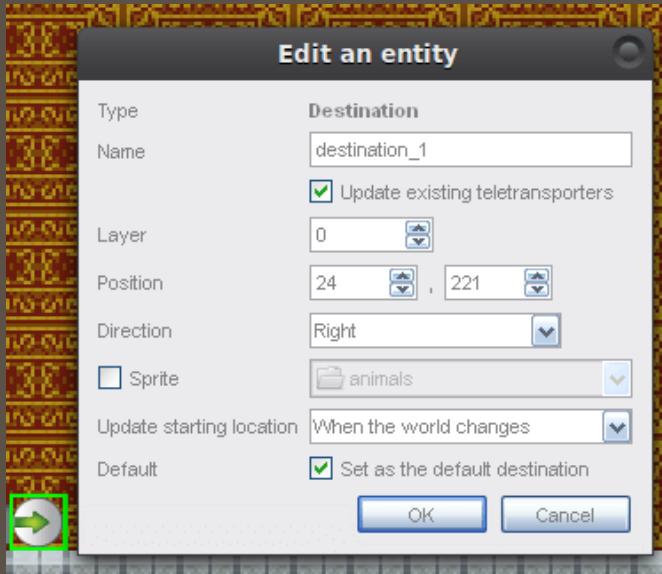


### Add Destination Entity



## Edit Destination Entity Options

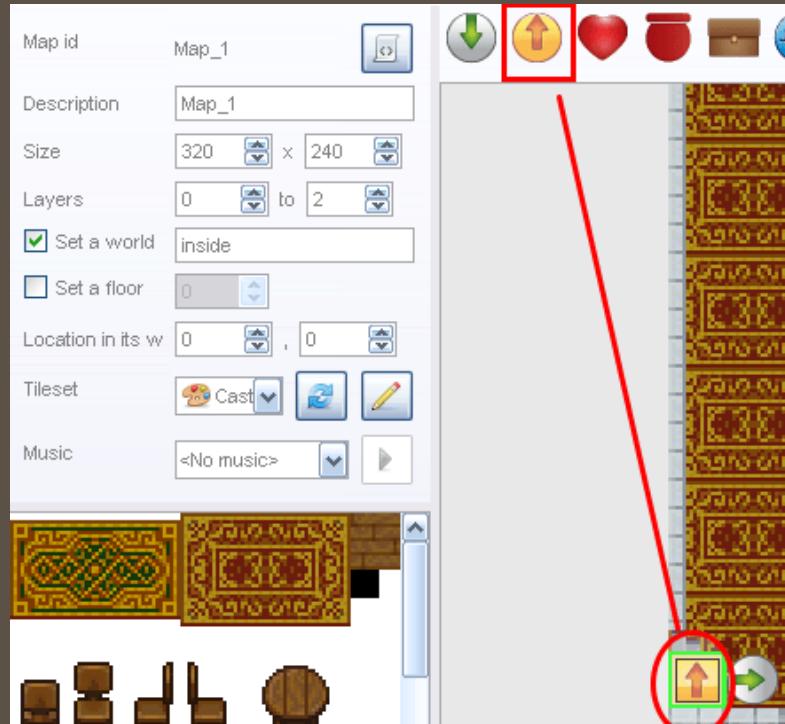
Double click on the destination entity in order to bring up the edit menu.



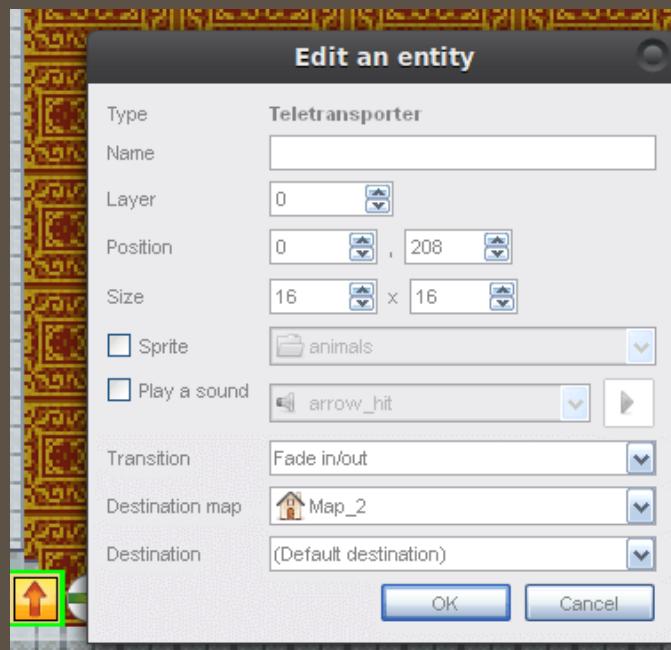
Option	Details
Name	A name is needed for scripting reasons and for Teletransporters.
Layer	The layer you want the entity on.
Position	The coordinates the entity is at. You can manually change them or move the entity with the cursor.
Direction	<p>The direction is important because it tells Solarus which way you want the player character to face when teletransporting to a new map.</p> <p>The options are:</p> <ul style="list-style-type: none"> <li>- Up</li> <li>- Down</li> <li>- Left</li> <li>- Right</li> </ul>
Sprite	You can pick a sprite for the Destination Entity.
Update Starting Location	<p>You can set when the destination entity updates and it is only possible when the destination entity has a name. You can leave it default, "When the world changes," it is fine most of the time. "Never" works too, but for scripting reasons you might want to update the destination entity.</p> <p>There are three options:</p> <ul style="list-style-type: none"> <li>- When the world changes</li> <li>- Always</li> <li>- Never</li> </ul>
Default	If there are more than one destination entity on the map, then the default one is where the player will begin at.

# Teletransporter Entity

## Add Teletransporter



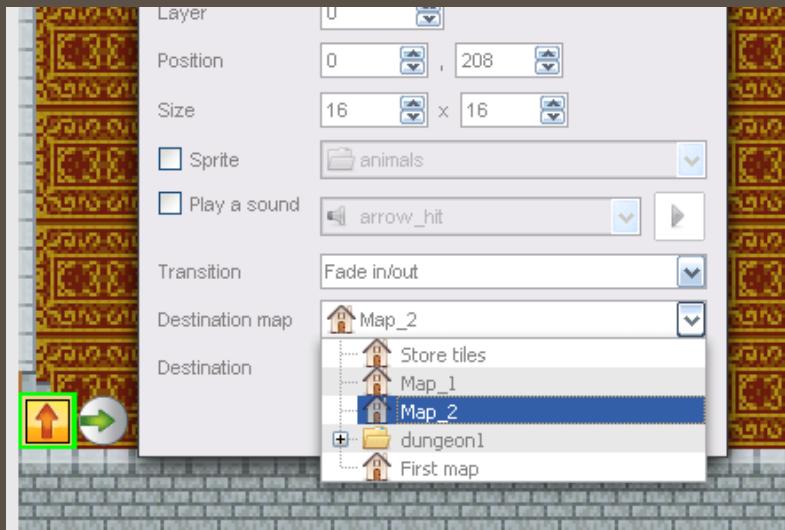
## Edit Teletransporter Entity Options



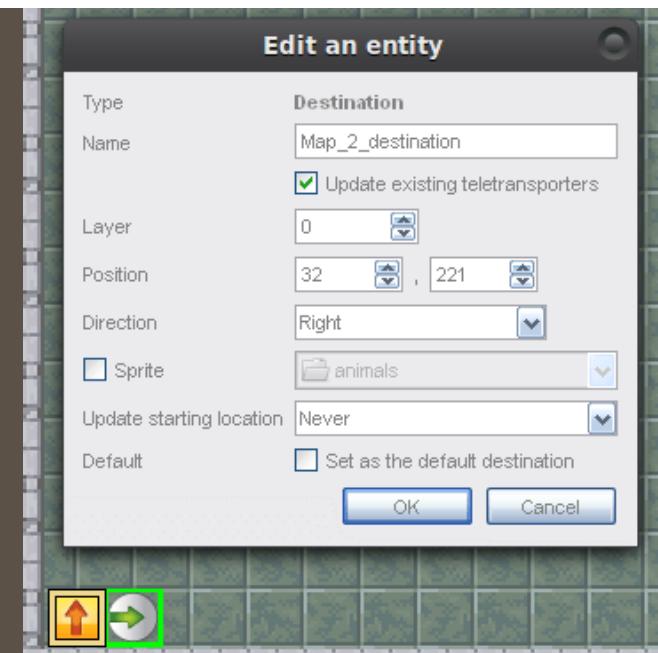
Option	Details
Name	A name is needed for scripting reasons. You can leave it blank.
Layer	The layer you want the entity on.
Position	The coordinates the entity is at. You can manually change them or move the entity with the cursor.

Option	Details
Size	You can resize the Teleporter with the <code>r</code> key. This is use for when there is a long gate entrance. You can resize it instead of adding many teletransporters. 
Sprite	You can pick a sprite for the Teleporter Entity.
Play a Sound	When the Teleporter is touched a sound is played.
Transition	A movement, development, or evolution from one form, stage, or style to another. There are 3 transition options: <ul style="list-style-type: none"><li>- Immediate: No fade or anything. Straight to next map.</li><li>- Fade in/out: Turns black and slowing vanishes on next map.</li><li>- Scrolling: Rolling to the next map.</li></ul>
Destination Map	Destination map is the map that you are going to next. In the sample it is <code>[Map_2]</code> .
Destination	This is the Destination Entity name. In the sample it is <code>[Map_2_destination]</code> for <code>[Map_2]</code> . Also, it is <code>[destination_1]</code> for <code>[Map_1]</code> .

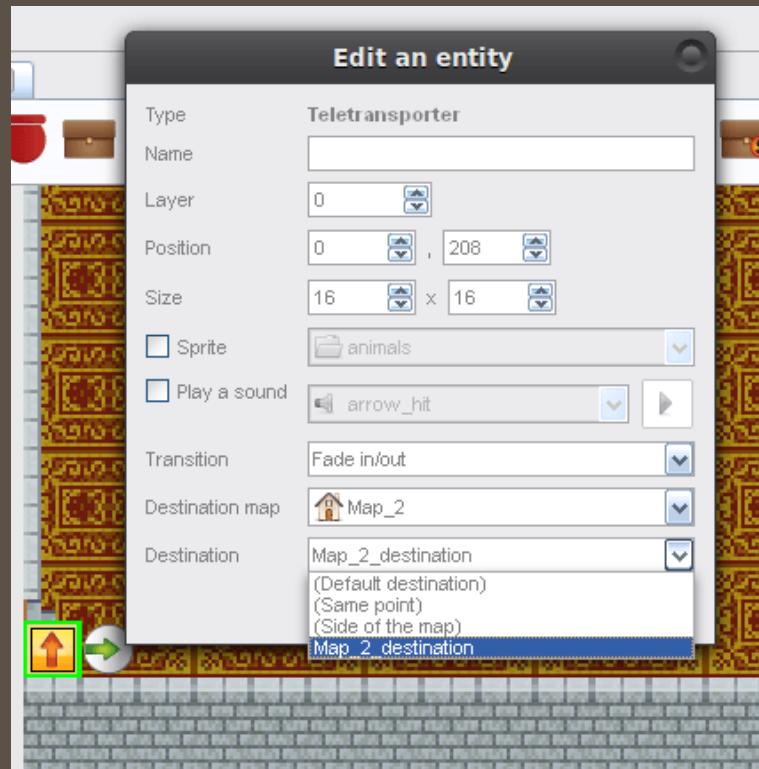
`Map_1` to `Map_2`.



`Map_2` destination entity name

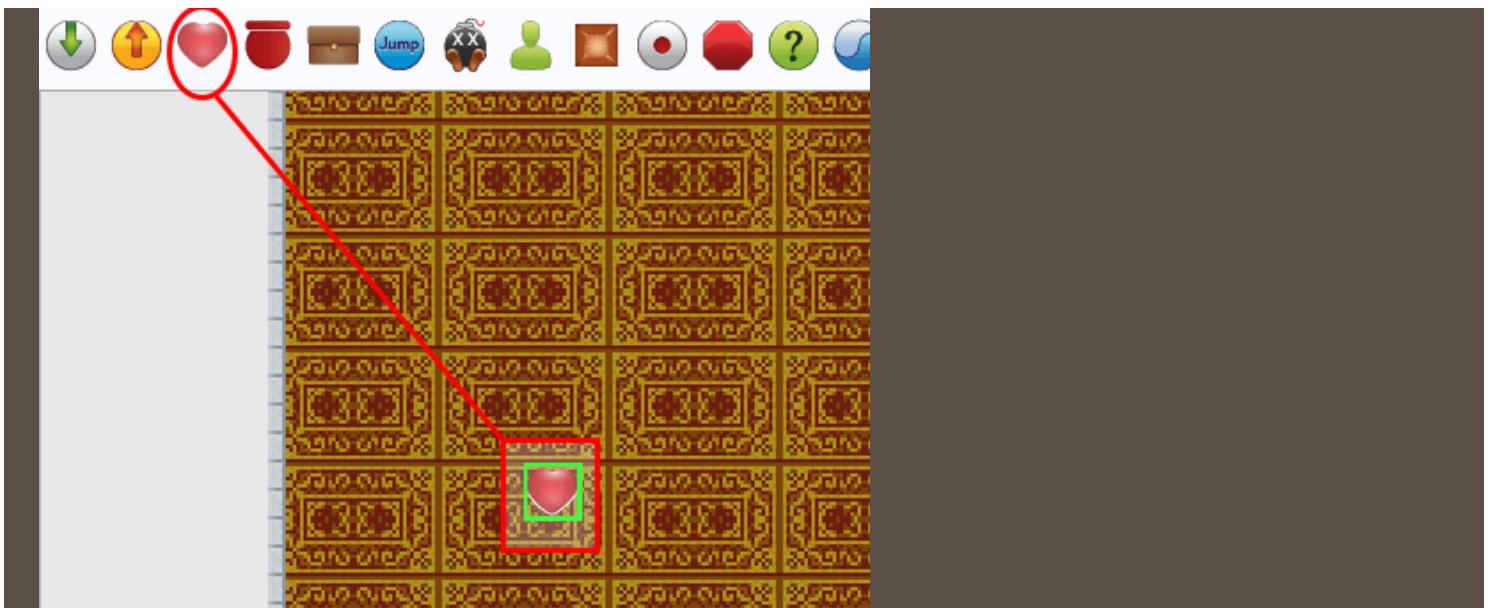


`Map_1` to `Map_2_destination` entity name



## Pickable Entity

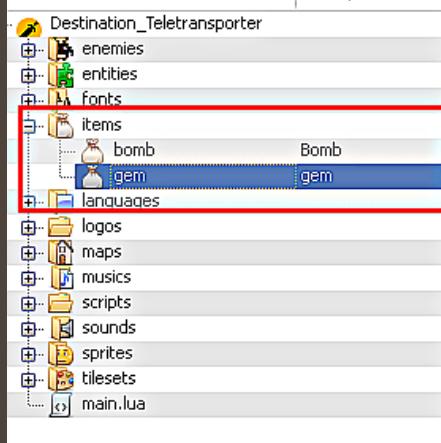
Add Pickable Entity



### Edit Pickable Entity Options



Option	Details
Name	A name is needed for scripting reasons. You can leave it blank.
Layer	The layer you want the entity on.
Position	The coordinates the entity is at. You can manually change them or move the entity with the cursor.

Option	Details
Treasure	<p>This is an item made in the item directory.</p>  <pre> 1 local item = ... 2 3 function item:on_created() 4 5     -- Define the properties of rupees. 6     self:set_shadow("small") 7     self:set_can_disappear(true) 8     self:set_brandish_when_picked(false) 9     self:set_sound_when_picked("picked_rupee") 10 end 11 12 function item:on_obtaining(variant, savegame_variable) 13 14     local amounts = { 1, 5, 20, 50, 100, 300 } 15     local amount = amounts[variant] 16 17     item:get_game():add_money(amount) 18 end </pre>
Variant	<p>A variant is the type. Most of the time the type is a different color and worth a higher value. For example, a blue gem for variant 2.</p> 
Save State	<p>You can save whether the item has been picked up or not. Otherwise, it would appear again.</p>

## Name Connection

The name of the animation is connected to the item connected to the name of the animation. In this case the name is `gem`.



generated by haroopad

## Item Script

Double click on the item gem and the script will appear in a tab.

```
local item = ...

function item:on_created()

    -- Define the properties of rupees.
    self:set_shadow("small")
    self:set_can_disappear(true)
    self:set_brandish_when_picked(false)
    self:set_sound_when_picked("picked_rupee")
end

function item:on_obtaining(variant, savegame_variable)

    local amounts = { 1, 5, 20, 50, 100, 300 }
    local amount = amounts[variant]

    item:get_game():add_money(amount)
end
```

### Breaking down the Script:

Most of this I do not have to explain because it is obvious. I will explain it all later though in a table.

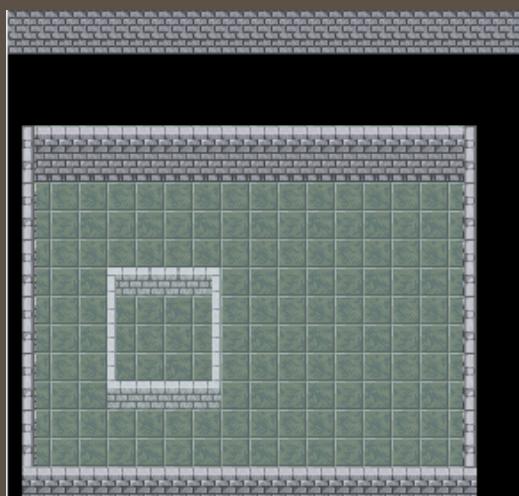
**self** refers to the item. For example, `Self(Gem):set_shadow("small")`.

**Brandish** is the sound that is normally made when opening a chest. For example, Da da ta daa! You know what I am talking about if you have played Zelda.

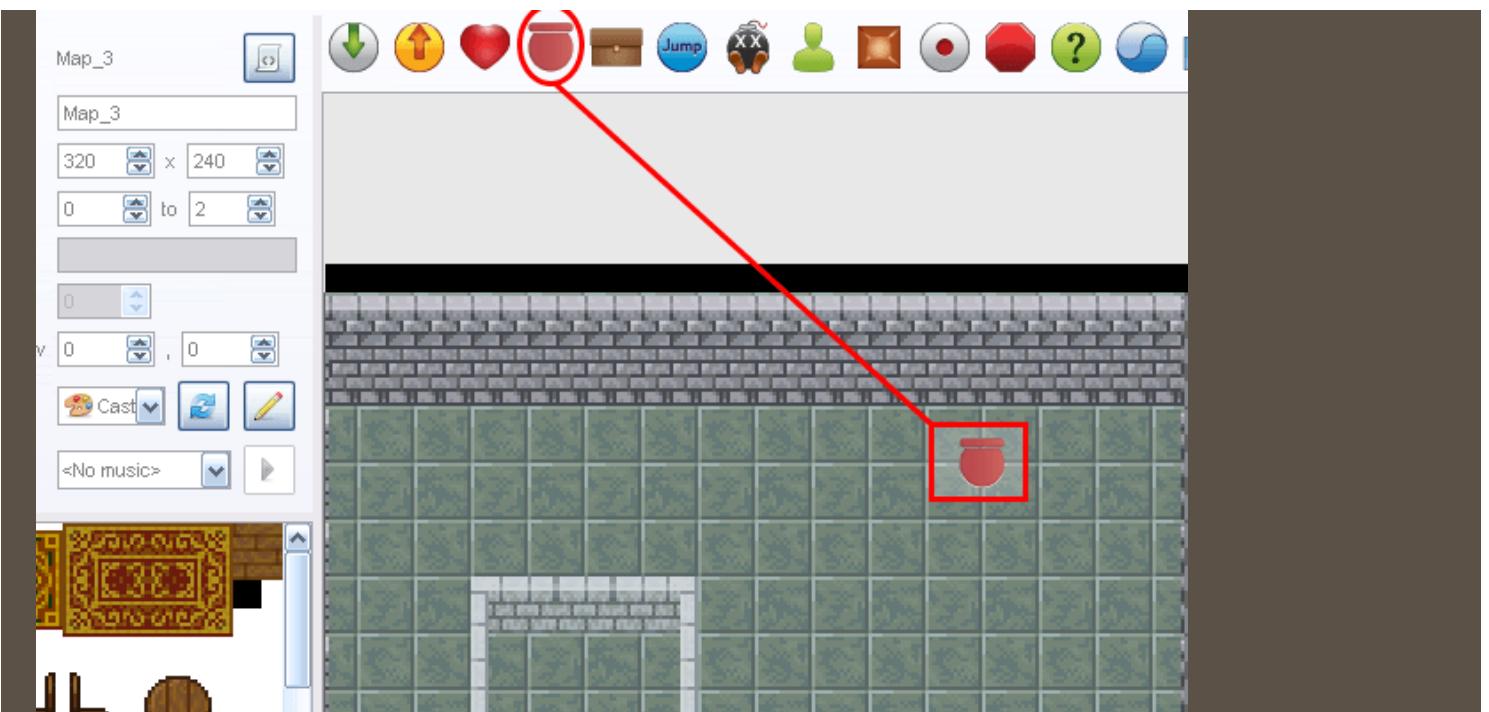
## Destructible Entity

### Inside Store to Map

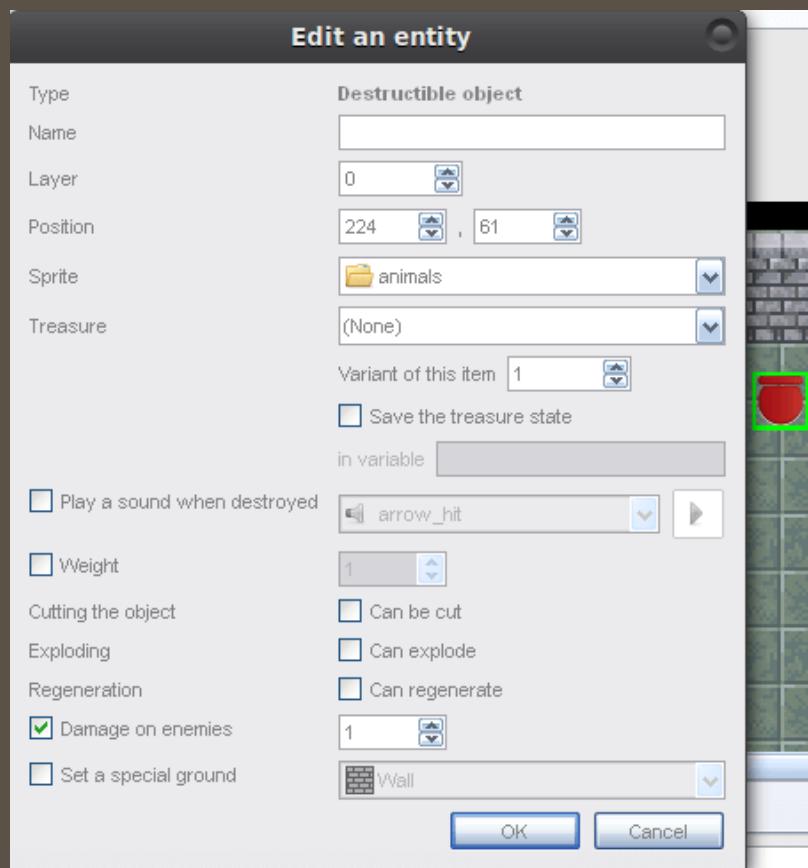
Create a map called **Map\_3** and copy this room from the inside store to the map.



### Add Destructible Entity



## Edit Destructible Entity Options



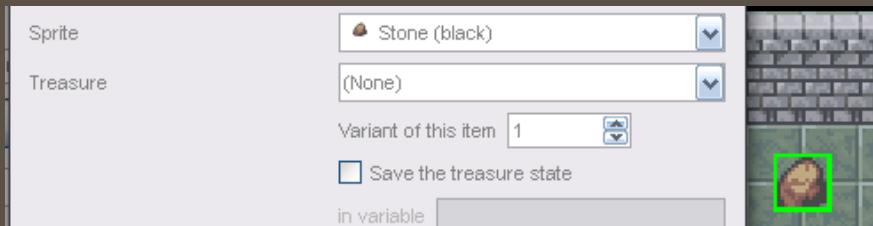
Option	Details
Name	A name is needed for scripting reasons. You can leave it blank.
Layer	The layer you want the entity on.
Position	The coordinates the entity is at. You can manually change them or move the entity with the cursor.

generated by haroopad

Option	Details
Sprite	Add the sprite that you want the entity to look like.
Treasure	A treasure appears When the destructible entity is lifted.
Variant	A variant is the type. Most of the time the type is a different color and worth a higher value.
Save State	You can save whether the item has been picked up or not. Otherwise, it would appear again.
Play a sound when destroyed	A sound you want to be played when the entity breaks.
Weight	<p>The weight is related to a game type.</p> <pre>game:set_ability(ability_name, level)</pre> <p>The level and ability can be set from there. In this case it would be the "lift" ability. More information about this will be added in the game type chapter. It will include an example of this.</p>
Cutting	It can be cut with the sword if the check box is checked.
Exploding	The destructible will explode like a bomb if thrown if the check box is checked.
Regeneration	The entity will appear again after like 5 seconds by default if the check box is checked.
Damage on enemies	Cause damage to enemies. Enter te damage amount.
Set a special ground	The destructible entity can become a special ground. For example, picking "prickles" will hurt the hero when bumping into the entity.

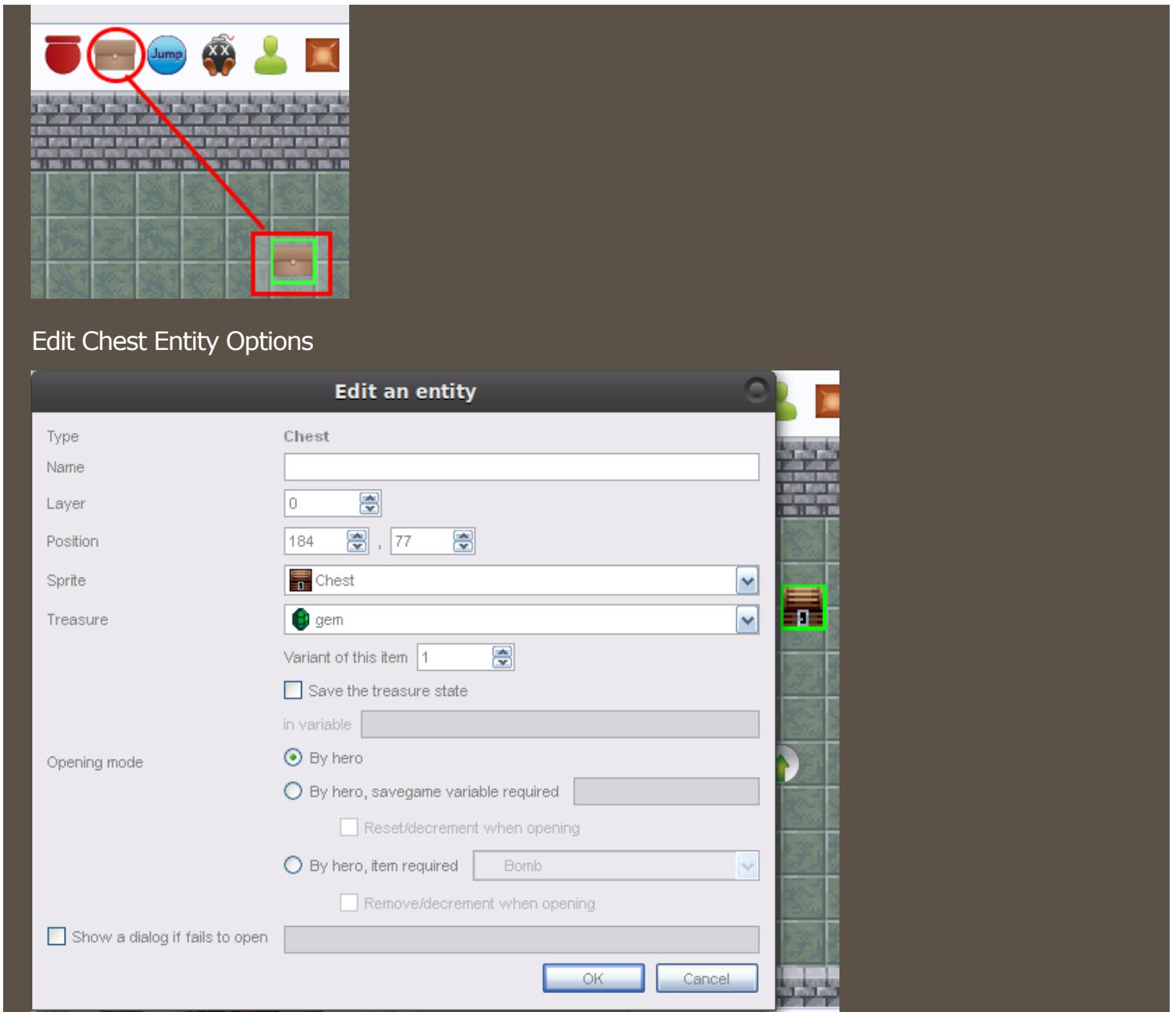
## Destructible Entity Sprite

You can pick up and throw the entity by just adding the sprite and not changing any options.



## Chest Entity

Add a Chest Entity

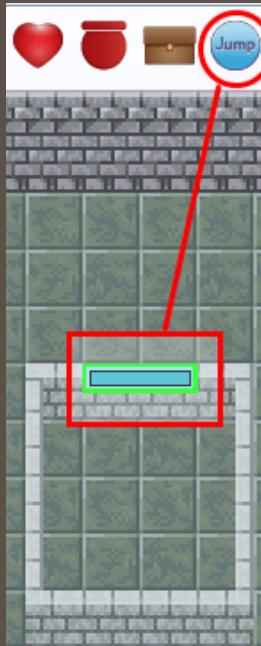


Option	Details
Name	A name is needed for scripting reasons. You can leave it blank.
Layer	The layer you want the entity on.
Position	The coordinates the entity is at. You can manually change them or move the entity with the cursor.
Sprite	Add the sprite that you want the entity to look like.
Treasure	A treasure appears When the destructible entity is lifted.
Variant	A variant is the type. Most of the time the type is a different color and worth a higher value.
Save State	You can save whether the item has been picked up or not. Otherwise, it would appear again.
Opening Mode	You can save an item in a variable and you can require an item to open the chest. For example, a key.

Option	Details
Reset/decrement	Decreases an item amount instead of increasing.
Show a dialog	Add a dialog for when the chest fails to open. For example, when you do not have the key.

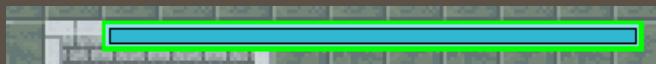
## Jumper Entity

### Add Jumper Entity

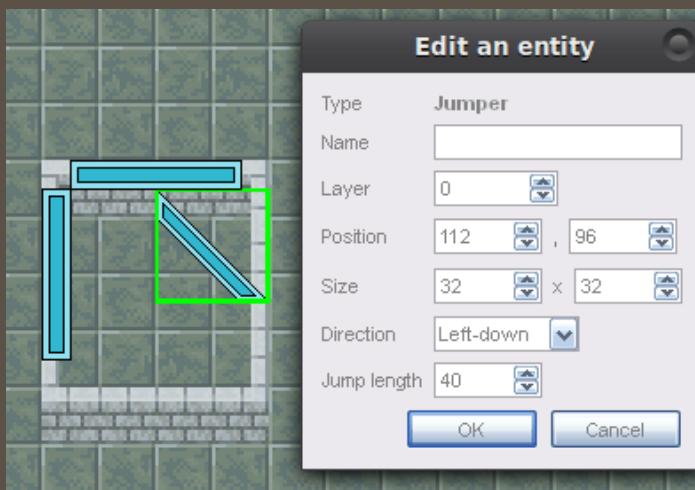


### Resize Jumper

Use the **r** key to resize the jumper.



### Edit Jumper Entity Options

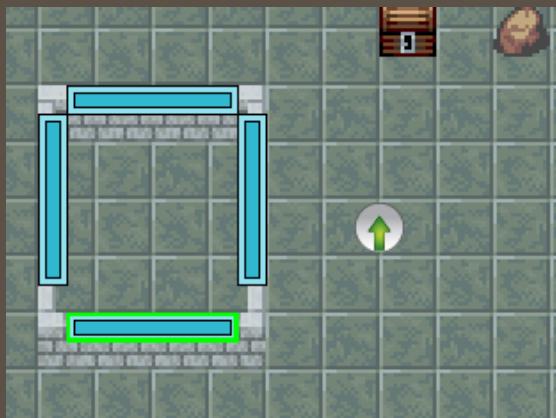


Option	Details

Option	Details
Name	A name is needed for scripting reasons. You can leave it blank.
Layer	The layer you want the entity on.
Position	The coordinates the entity is at. You can manually change them or move the entity with the cursor.
Size	The jumper Entity can be resized with the <code>r</code> key or manually from the edit menu.
Direction	<p>As you can see in the image, the jumper can be set in many directions.</p> <p>They are:</p> <ul style="list-style-type: none"> <li>- Up</li> <li>- Down</li> <li>- Left</li> <li>- Right</li> <li>- Left-down</li> <li>- Right-down</li> <li>- Left-up</li> <li>- Right-up</li> </ul>
Jump length	This is how far the sprite can jump. A common mistake is that the player gets stuck in a wall when jumping. Most of the time the jump distance is not long enough in those cases.

## Jump Walls

The player character now can jump over the wall at the four jumper entity locations.



## Jump Preview

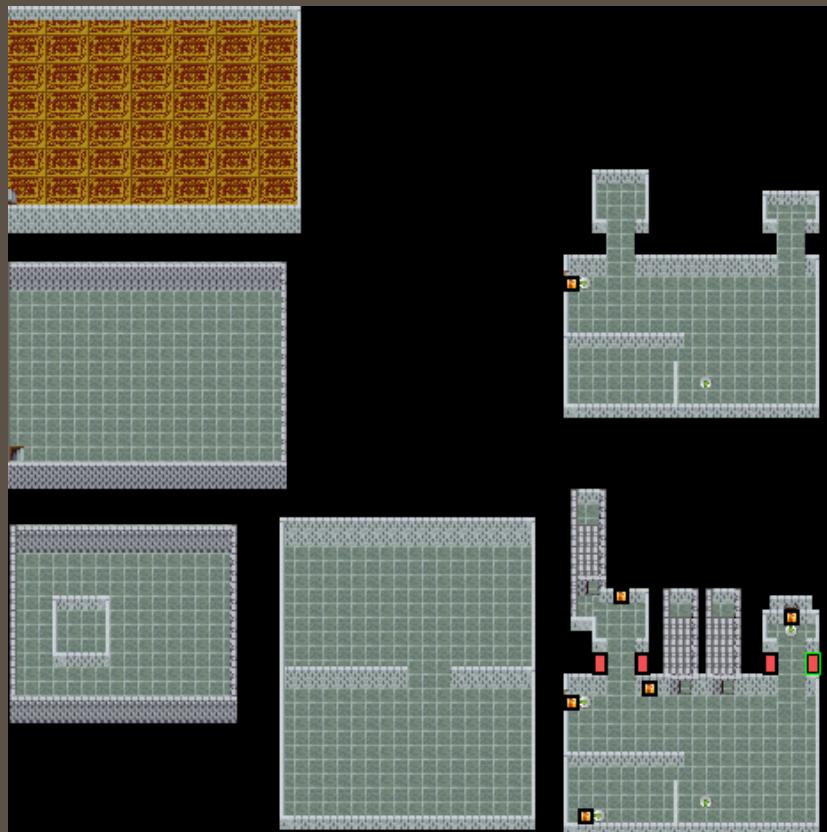


## Enemy Entity

The [documentation](#) for more entity functions.

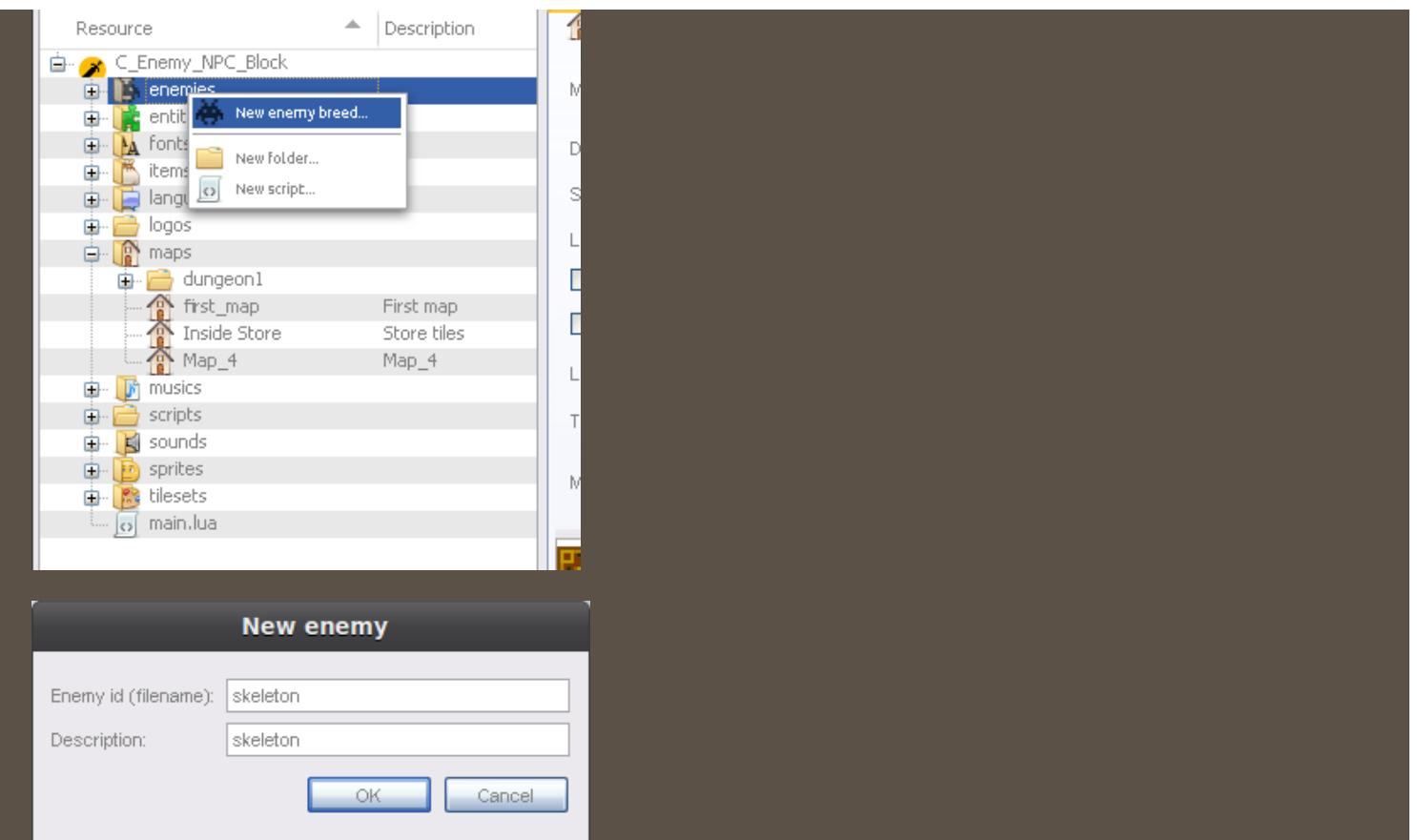
### Inside Store

First of all, copy the inside store room to a map name of your choosing. I named the map [map\\_4](#). You will have to download or update your current inside store. You can find it in the directory [Lessons > Chapter\\_13\\_Entities > inside\\_store.zip](#)



### Enemy Breed Script

Create a breed called [skeleton.](#) The name should be the same for the enemy sprite. One can make the description anything. I picked skeletron.



## Enemy Script Generated

Making this breed or any new breed generates a basic enemy script. The enemy does not change directions. For instance, it will face one way and not up, down, left, and right. This is useful for basic enemies, but it is simple to add a 4 way direction movement.

Double click on the breed **skeleton** to show the script.

```
-- Lua script of enemy skeleton.
-- This script is executed every time an enemy with this model is created.

-- Feel free to modify the code below.
-- You can add more events and remove the ones you don't need.

-- See the Solarus Lua API documentation for the full specification
-- of types, events and methods:
-- http://www.solarus-games.org/doc/latest

local enemy = ...
local game = enemy:get_game()
local map = enemy:get_map()
local hero = map:get_hero()
local sprite
local movement

-- Event called when the enemy is initialized.
function enemy:on_created()

    -- Initialize the properties of your enemy here,
    -- like the sprite, the life and the damage.
    sprite = enemy:create_sprite("enemies/" .. enemy:get_breed())
    enemy:set_life(1)
    enemy:set_damage(1)
end
```

generated by haroopad

```
-- Event called when the enemy should start or restart its movements.
-- This is called for example after the enemy is created or after
-- it was hurt or immobilized.
function enemy:on_restarted()

movement = sol.movement.create("target")
movement:set_target(hero)
movement:set_speed(48)
movement:start(enemy)
end
```

Everything is pretty simple and the documentation can be checked for more options. I would like to point out two things.

`function enemy:on_restarted()` is basically a reset for the sprite. If the enemy is hit, then it will reset or restart to this.

`sprite = enemy:create_sprite("enemies/" .. enemy:get_breed())` is creating the sprite for the enemy. The `enemy:get_breed()` gets the name of the enemy and since the image filename is the same as the enemy breed. That sprite is created.

## Enemy Four Way Direction

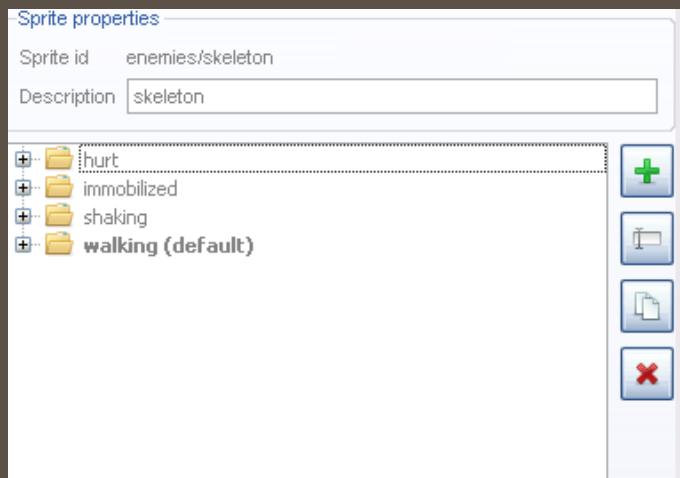
This can simply be added to the bottom of the script for the sprite to look in all four directions.

```
function enemy:on_movement_changed()
    sprite:set_direction(movement:get_direction4())
end
```

## Enemy Animation

There are four basic animation for an enemy.

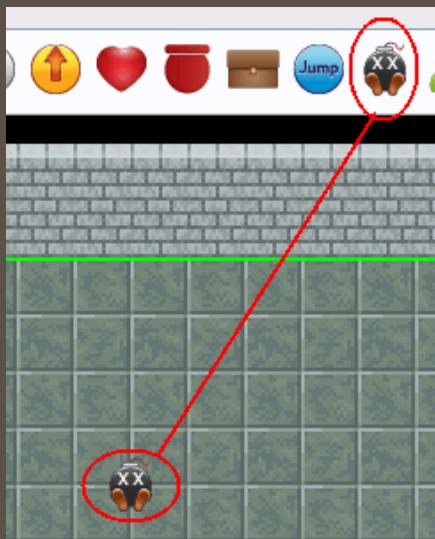
Animation	Description
Hurt	The player will blink when hurt or any animation you want for damage.
Immobilized	The enemy will not be able to move for a few seconds.
Shaking	The enemy can be set up to do a shake effect, but another animation can be used.
Walking	The enemy will travel as it walks.



## Add Enemy

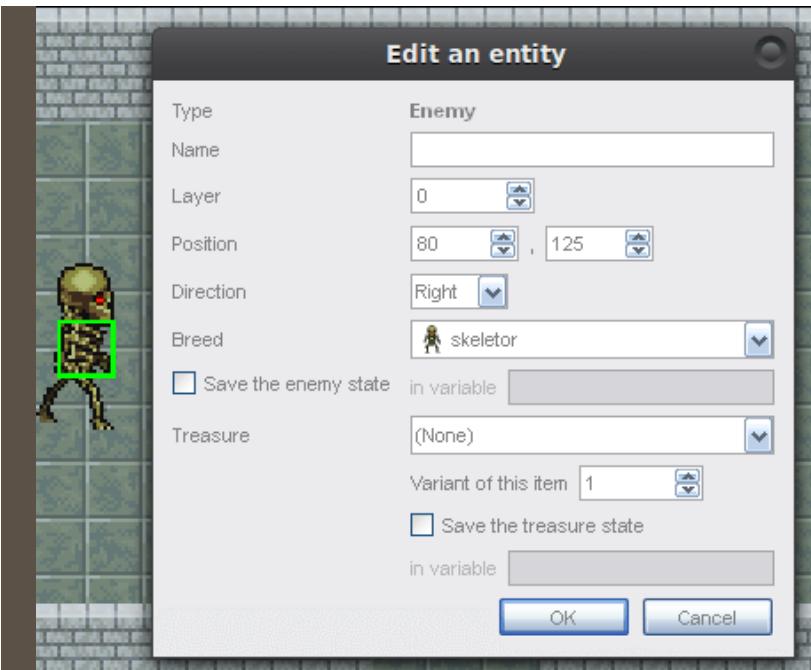
generated by haroopad

Select the bomb entity icon and add it to the map.



## Edit Enemy

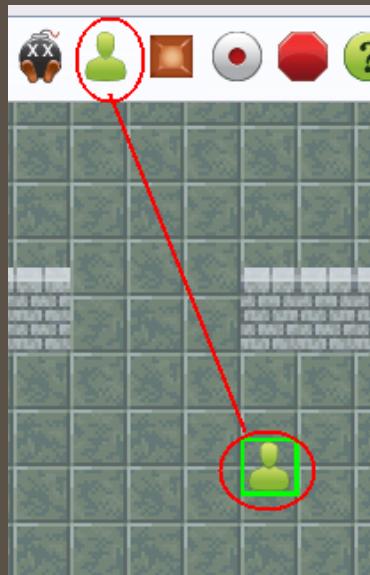
Type	Option
Name	A name is needed for scripting reasons. One can leave it blank.
Layer	The layer you want the entity on.
Position	The coordinates the entity is at. You can manually change them or move the entity with the cursor.
Direction	The direction the enemy faces at start.
Breed	The type of enemy.
Save Enemy State	You do not really need this for basic enemies, but if you do not want bosses or mini bosses appearing again, then it is best to save their state in a variable.
Treasure	A treasure will appear when the enemy is defeated. A heart or money is normal.
Variant	A variant is the type. Most of the time the type is a different color and worth a higher value.
Save Treasure State	If you do not want treasures like keys to appear again, then it is best to save their state in a variable.



## NPC Entity

### Add NPC

Select the green upper body person icon entity to add a NPC. It is right next to the bomb enemy entity.



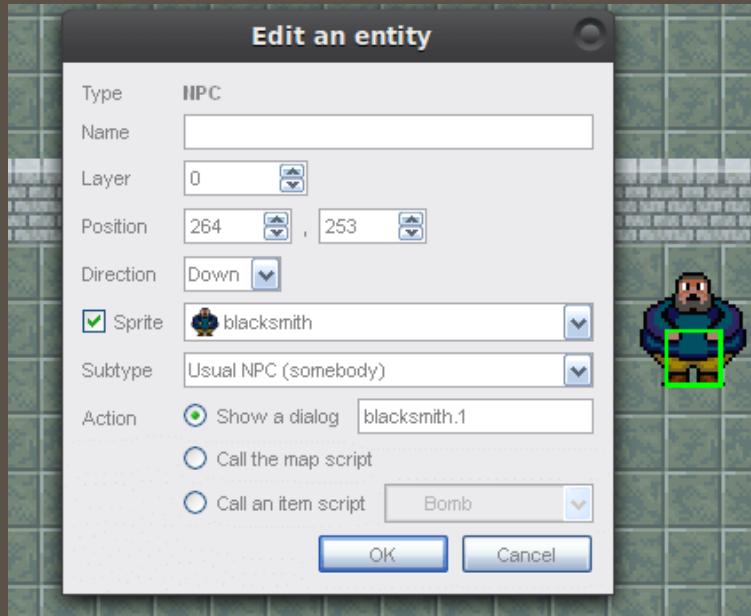
### Usual NPC

Type	Option
Name	A name is needed for scripting reasons. You can leave it blank.
Layer	The layer you want the entity on.
Position	The coordinates the entity is at. You can manually change them or move the entity with the cursor.
Direction	The direction the NPC faces at start.
Sprite	Pick the sprite image for the NPC.

generated by haroopad

Type	Option
Subtype	There are two subtypes. Usual NPC and Generalized.
Action	From here one can call a dialog, map script, and/or item script.

A Usual NPC will automatically change direction depending on what angle one tries to interact at. For instance, if one talks to the NPC from behind, then it will turn towards the player character. It will be the same for all other directions. A Usual NPC is used normally for somebody and not something.



## Set Up Dialog Box

First off, you need to set up Christopho's Dialog box. Go to [chapter 7](#) if you forgot how to set it up. You can grab the sample to copy and paste that.

[Lessons > chapter\\_7\\_Dialog\\_Mouse\\_control\\_fix.zip](#)

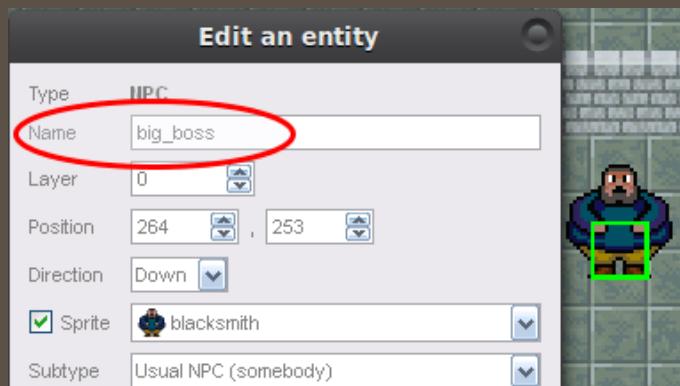
## Sample

You can find the sample for this lesson in the following directory.

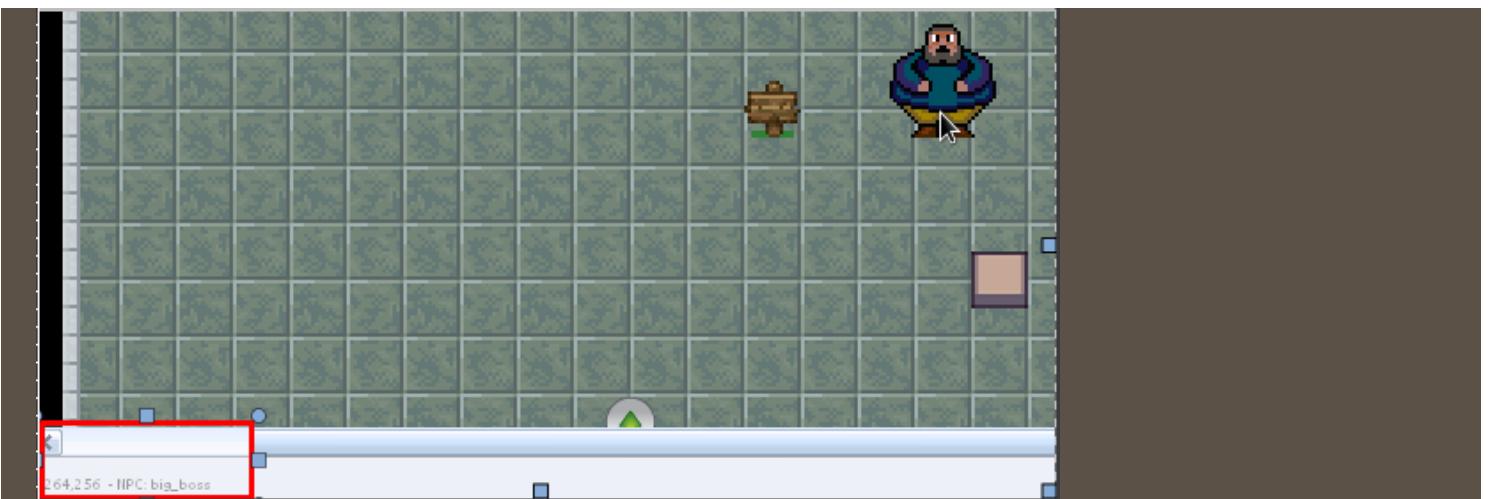
[Lessons > Chapter\\_13\\_Entities > C\\_Enemy\\_NPC\\_Block\\_updated.zip](#)

## Usual NPC Call Map Script

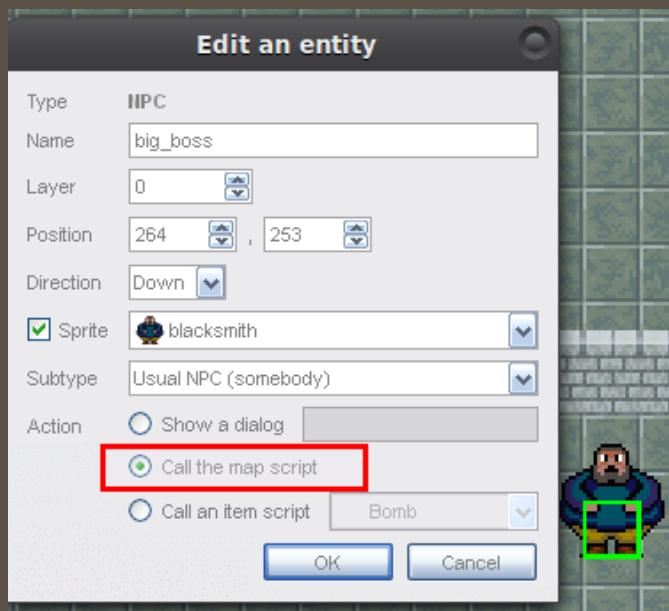
The first thing that needs to be done is to name the NPC. A name is required for scripting. I named the NPC [big\\_boss](#).



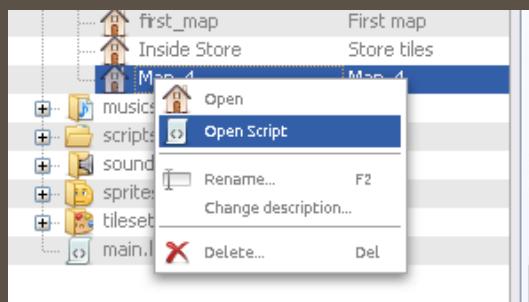
You can see the NPC name in the status bar.



Next is to select, “`call the map script`”



Open up the map script.



### Showing NPC Script

I like showing the full script before breaking it down. This script is an example of a basic quest conversation. The player character needs to find a hammer, but in this case there is no need to actually find the hammer. The `big_boss` gives the player a gem as a reward.

```
local map = ...
local game = map:get_game()

function big_boss:on_interaction()
```

generated by haroopad

```

if game:get_value("hammer_quest_started")then
    game:start_dialog("hammer.done")
else
    game:start_dialog("hammer.hello", function(answer)
        if answer == 3 then --No
            game:start_dialog("hammer.no")
        else
            game:start_dialog("hammer.yes", function()
                hero:start_treasure("gem", 1, "hammer_quest_started", function()
                    game:start_dialog("hammer.wonderful_day")
                end)
            end)
        end
    end)
end
end

```

## Setting up a Simple NPC dialog

The `hammer.hello` dialog states that `big_boss` needs help, and he is offering a reward.

I lost my hammer!

Help me find it? I will give you a reward.

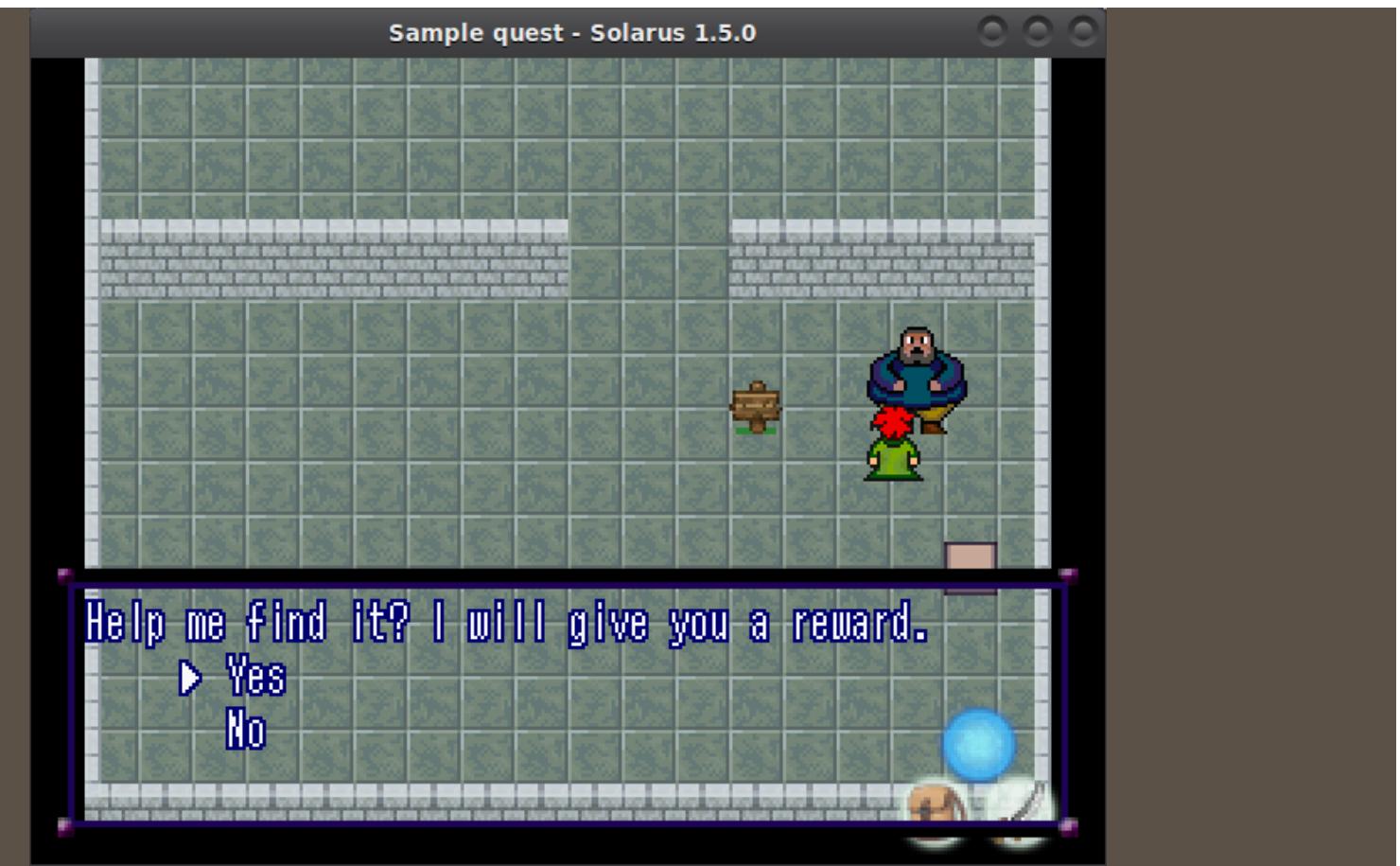
\$? Yes

\$? No

As you can see, there is a big space after, "I lost my hammer!" That is because I wanted the second part of text to begin on its own line. See the two following images to understand.



generated by haroopad



Above is a very simple script. We use the name of the NPC in the map script with a solarus function called `NPC:on_interaction()`.

[Documentation > map entities > non-playing character > NPC:on\\_interaction\(\)](#)

```
NPC_Name:on_interaction = big_boss:on_interaction
```

#### EX:

If you wanted a sound to play when interacting or talking to the NPC, then you would do the following.

```
function big_boss:on_interaction()
    sol.audio.play_sound("audio_name")
end
```

In the map script this would be it for the dialog above.

```
function big_boss:on_interaction()
    game:start_dialog("hammer.hello")
end
```

#### NPC Dialog Yes\_No

Now we have to set up a simple dialog.

```
function big_boss:on_interaction()
    game:start_dialog("hammer.hello", function(answer)
        if answer == 3 then --No
            game:start_dialog("hammer.no")
        else
            game:start_dialog("hammer.yes")
```

generated by haroopad

```
end)  
end
```

The answer 3 is "no" because it is the 3rd row on the next line.

### Example:

```
I lost my hammer! (1) - line 1
```

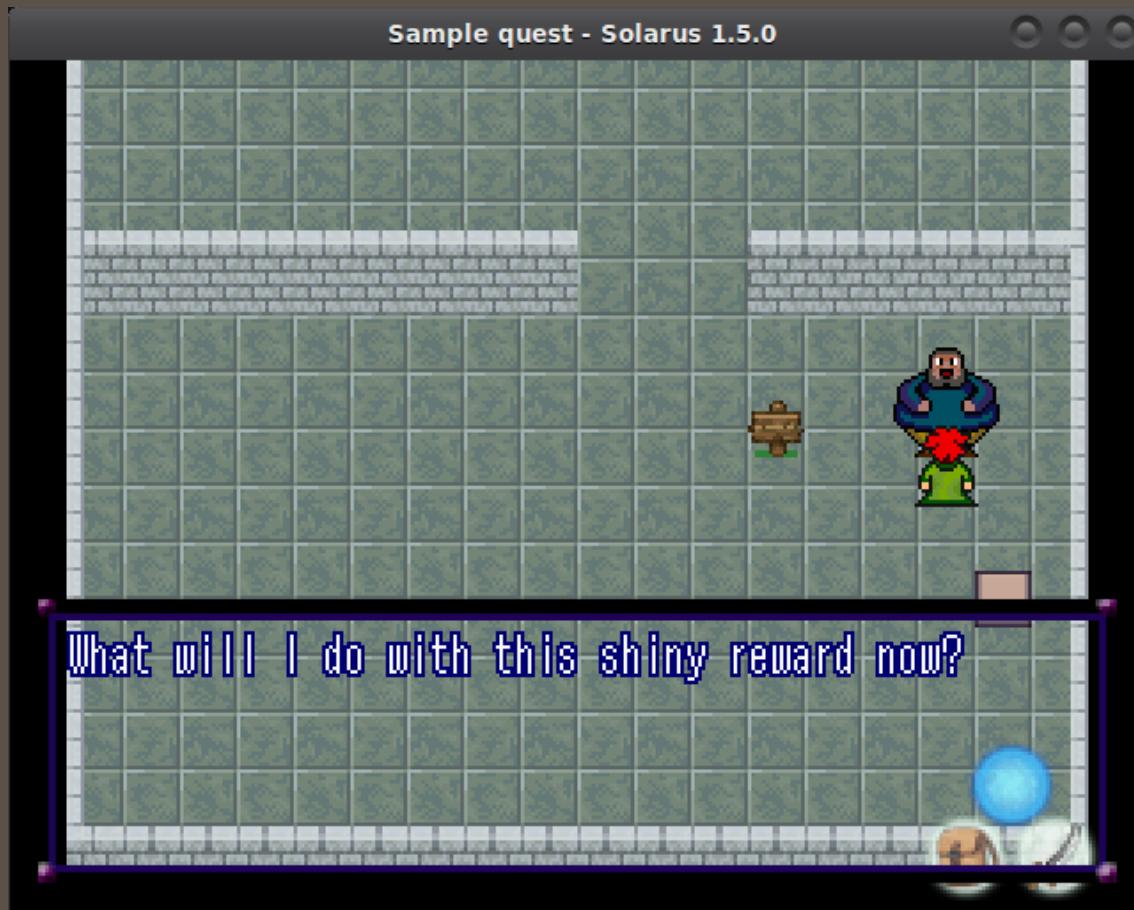
```
Help me find it? I will give you a reward. (1)
```

```
$? Yes(2)
```

```
$? No(3)
```

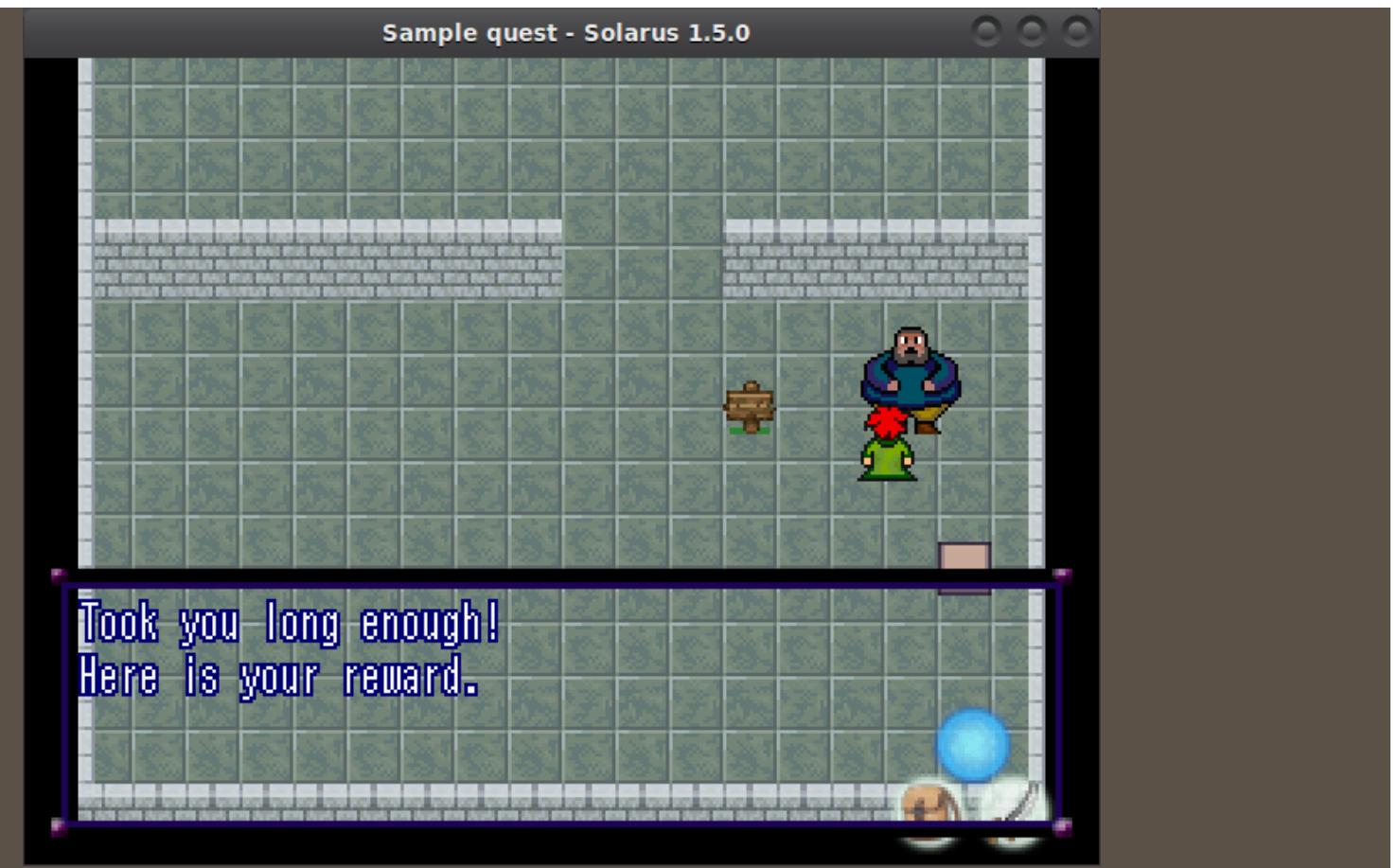
### NPC Answer No

The dialog, "What will I do with this shiny reward now?" is shown when the player chooses **No**.



### NPC Answer Yes

The dialog, "Took you long enough! Here is your reward." is shown when the player chooses **Yes**.



## NPC Treasure

The reward or treasure is a gem. In order to have the hero (player) receive a reward is by using the `hero:start_treasure("Treasure_name")` function.

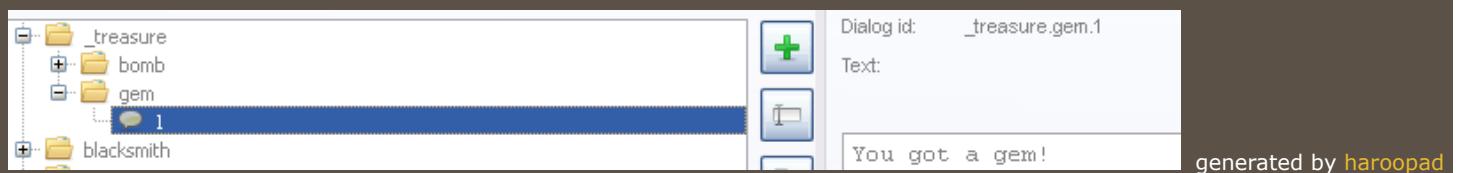
### Example:

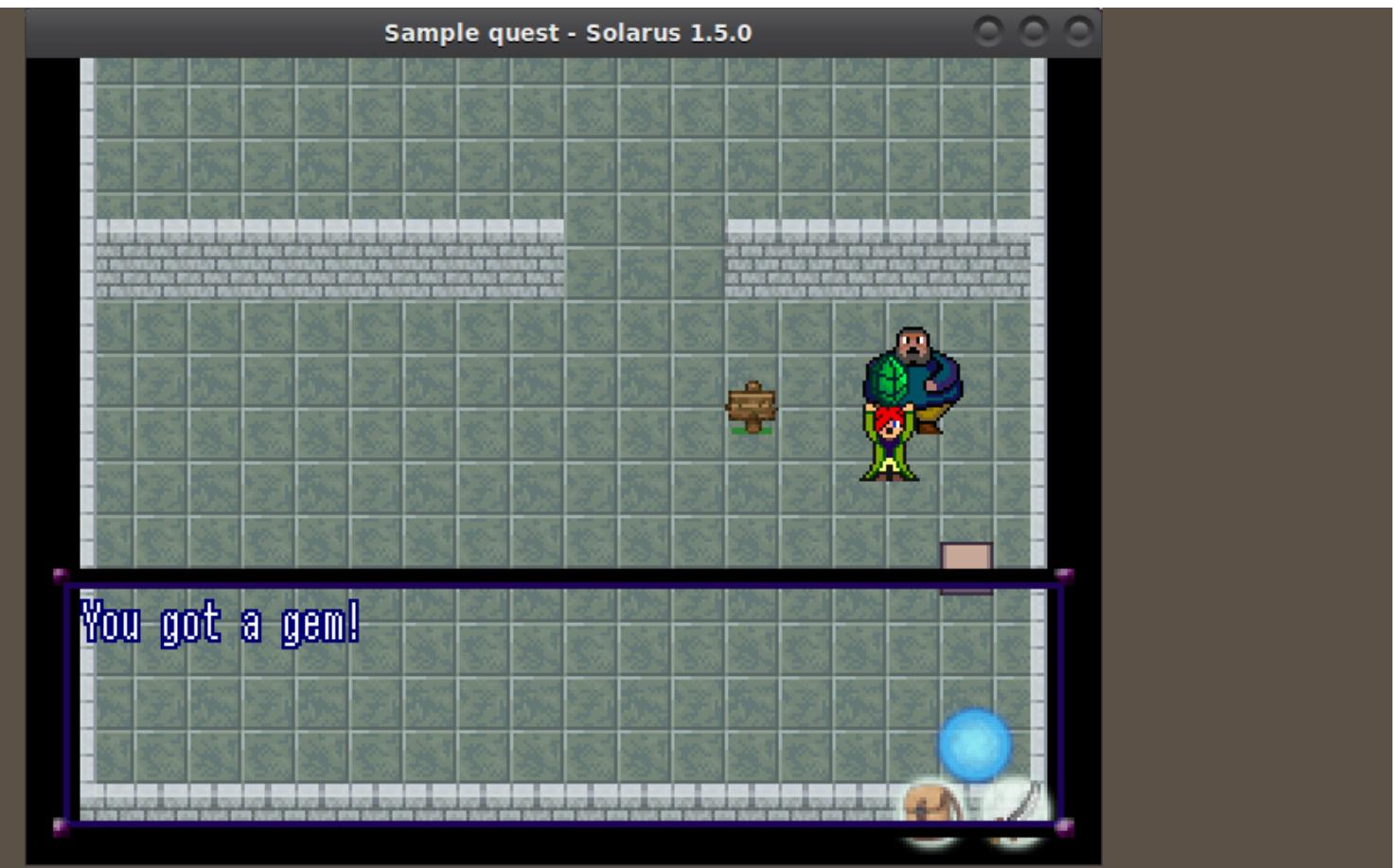
```
hero:start_treasure("gem")
```

The script would look like this.

```
function big_boss:on_interaction()
    game:start_dialog("hammer.hello", function(answer)
        if answer == 3 then --No
            game:start_dialog("hammer.no")
        else
            game:start_dialog("hammer.yes")
            hero:start_treasure("gem")
        end
    end
end)
```

The dialog, "you got a gem!," will automatically display if you put the dialog in `_treasure`. The number "1" is the variant.





## Prevent Repeating NPC Dialog With Boolean

Now we do not want the NPC to say the same thing over and over again. This could result in the player getting lots of gems too fast.

We can use a boolean to stop the repeating. This will only work if the player never returns to this map again after getting a reward. If `hammer_started` is `true`, then the dialog `hammer.done` will appear.

```
hammer_started = true

local hammer_started = false

function big_boss:on_interaction()

    if hammer_started then
        game:start_dialog("hammer.done")
    else
        game:start_dialog("hammer.hello", function(answer)
            if answer == 3 then --No
                game:start_dialog("hammer.no")
            else
                game:start_dialog("hammer.yes", function()
                    hero:start_treasure("gem")
                    hammer_started = true
                end)
            end
        end)
    end
end
```

## Prevent Repeating NPC Dialog With Get Value

generated by haroopad

Now we do not want the NPC to say the same thing over and over again. This could result in the player getting lots of gems too fast.

The game function `game:get_value("name_of_quest")` will work the same way as the boolean, but the dialog will not appear again when the player leaves and returns to the map. If the `get_value` hammer quest is `true`, then the dialog `hammer.done` will appear.

```
game:get_value("hammer_quest_started", true)
```

```
function big_boss:on_interaction()

if game:get_value("hammer_quest_started") then
    game:start_dialog("hammer.done")
else
    game:start_dialog("hammer.hello", function(answer)
        if answer == 3 then --No
            game:start_dialog("hammer.no")
        else
            game:start_dialog("hammer.yes", function()
                hero:start_treasure("gem")
                game:get_value("hammer_quest_started", true)
            end)
        end
    end)
end
end
end
```



## NPC Shortening the Script

There is a way to shorten the script. A few lines of script can be combined.

**Example:**

This line of code,

```
hero:start_treasure("gem")
game:get_value("hammer_quest_started", true)
```

can be the following line. The "1" is the variant. The quest will automatically return true.

```
hero:start_treasure("gem", 1, "hammer_quest_started", function()
```

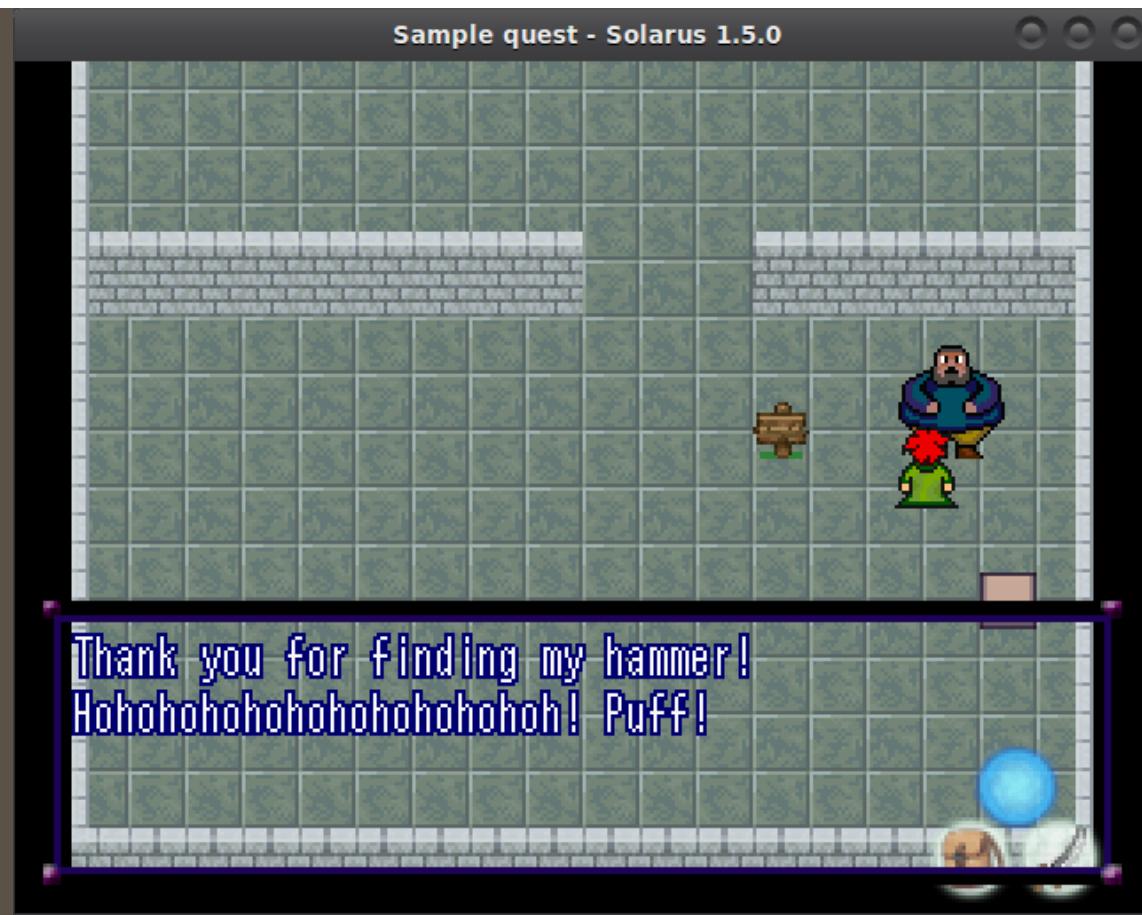
Below would result in our completed script.

```
function big_boss:on_interaction()

if game:get_value("hammer_quest_started") then
    game:start_dialog("hammer.done")
else
    game:start_dialog("hammer.hello", function(answer)
        if answer == 3 then --No
            game:start_dialog("hammer.no")
        else
            game:start_dialog("hammer.yes", function()
                hero:start_treasure("gem", 1, "hammer_quest_started", function()
                    game:start_dialog("hammer.wonderful_day")
                end)
            end)
        end
    end)
end
end
```

## NPC Wonderful Day

As one can see below, the NPC says something right after you get your reward. This is the dialog `hammer.wonderful_day`. This is a temporary reply by the NPC. Any other time the `hammer.done` will show.

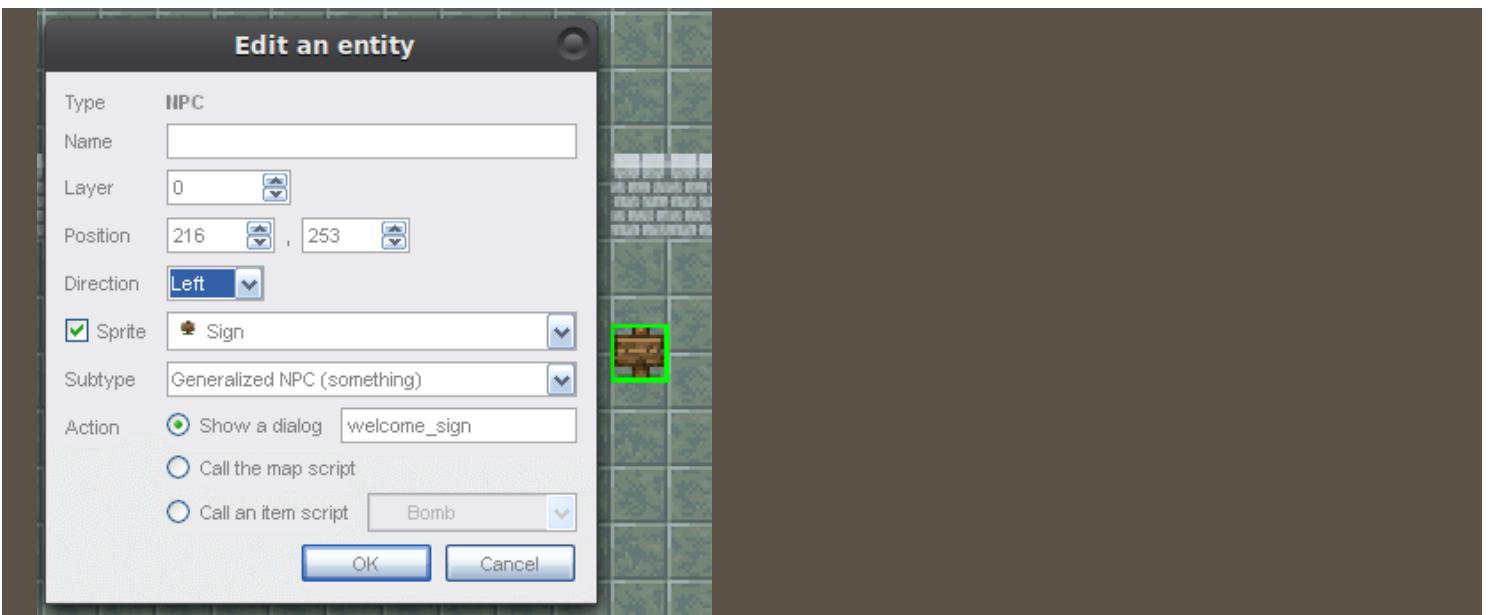


## NPC Dialogs

One can check [Christopho's tutorial](#) for more ways of setting up dialog near the end of the video. I explained the less error prone method.

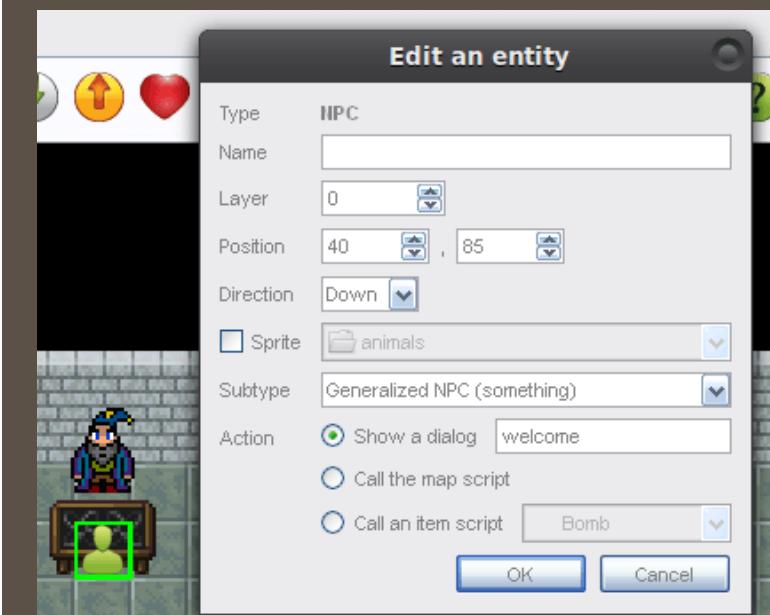
## Generalized NPC

A Generalized NPC can only be interacted based on its starting direction. Down is common for a Generalized NPC. An example of a Generalized NPC is a sign post. You would interact with it from the front, but one is able to pick up the sign by default and throw it from other directions. There is a special direction called, "any." This will allow the player character to interact from all four directions of the sign. Generalized NPC(s) are used normally for something and not somebody, but there are a few exceptions.



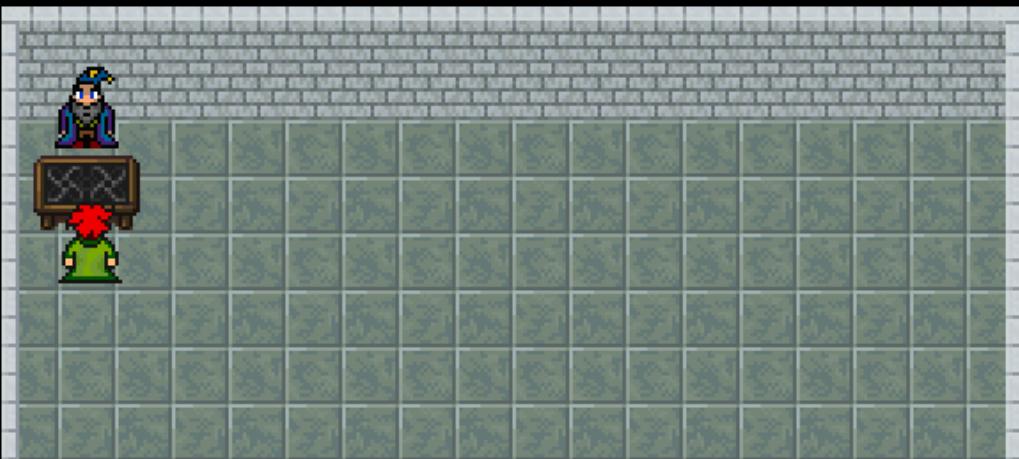
## Generalized NPC Desk Method

Generalized NPC(s) are useful for not only signs or other objects. They can be used to talk to a person across a table.



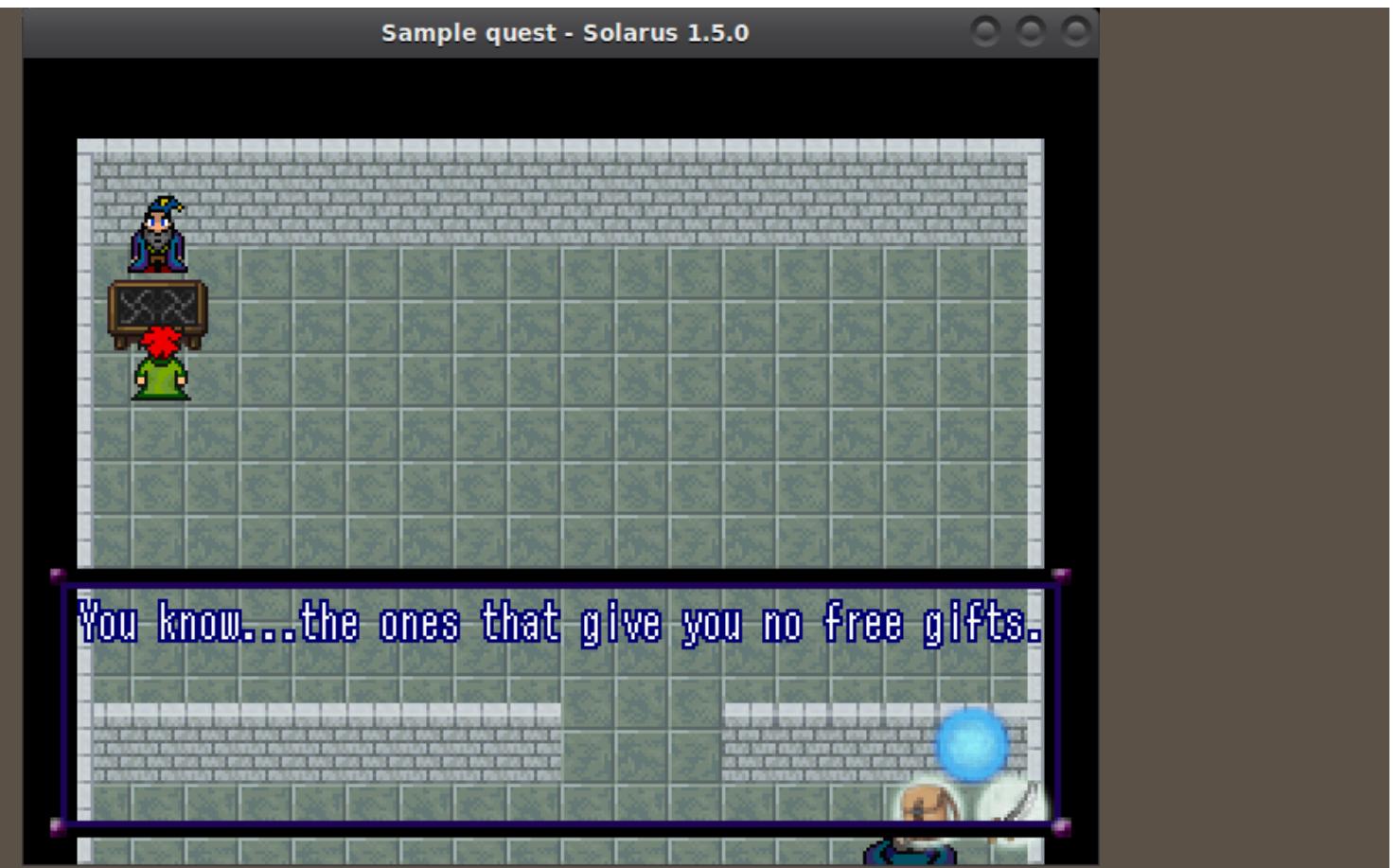
**Sample quest - Solarus 1.5.0**

Welcome to this random place!

**Sample quest - Solarus 1.5.0**

I am one of those NPC,





Now this isn't in the sample, but what if both NPC(s) are not a generalized NPC? If I talk to the magician from the right, then that makes him turn to the right. If the hero interact with the desk, then the magician will still be facing the wrong direction. In this case we use a function called, "`NPC:get_sprite():set_direction(Number)`".

Number	Direction
Direction(0)	Right
Direction(1)	Up
Direction(2)	Left
Direction(3)	Down

#### Example:

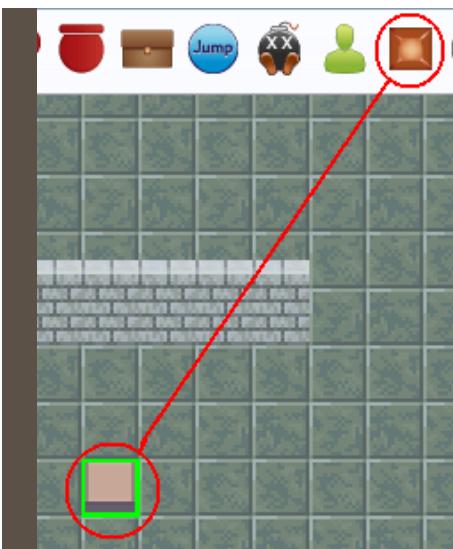
```
magician:get_sprite():set_direction(3)
```

The Magician will look down(3) in this case.

## Movable Block Entity

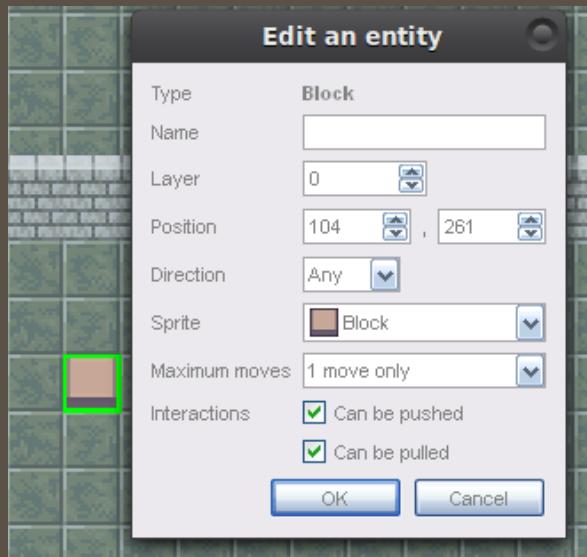
### Add Block

The block entity is right by the green NPC person entity. A block can be moved around.



## Block Properties

Type	Option
Name	A name is needed for scripting reasons. You can leave it blank.
Layer	The layer you want the entity on.
Position	The coordinates the entity is at. You can manually change them or move the entity with the cursor.
Direction	The direction you are allowed to push the block.
Sprite	Pick the sprite image for the block.
Max Moves	The options are <code>cannot move</code> , <code>1 move only</code> , and <code>unlimited</code> .
Interactions	There are 2 ways to interact with a block. It can be <code>pushed</code> or <code>pulled</code> .



The block entity is that simple. There will be more customization for the block entity in Solarus 1.6 version.

## Switch Entity

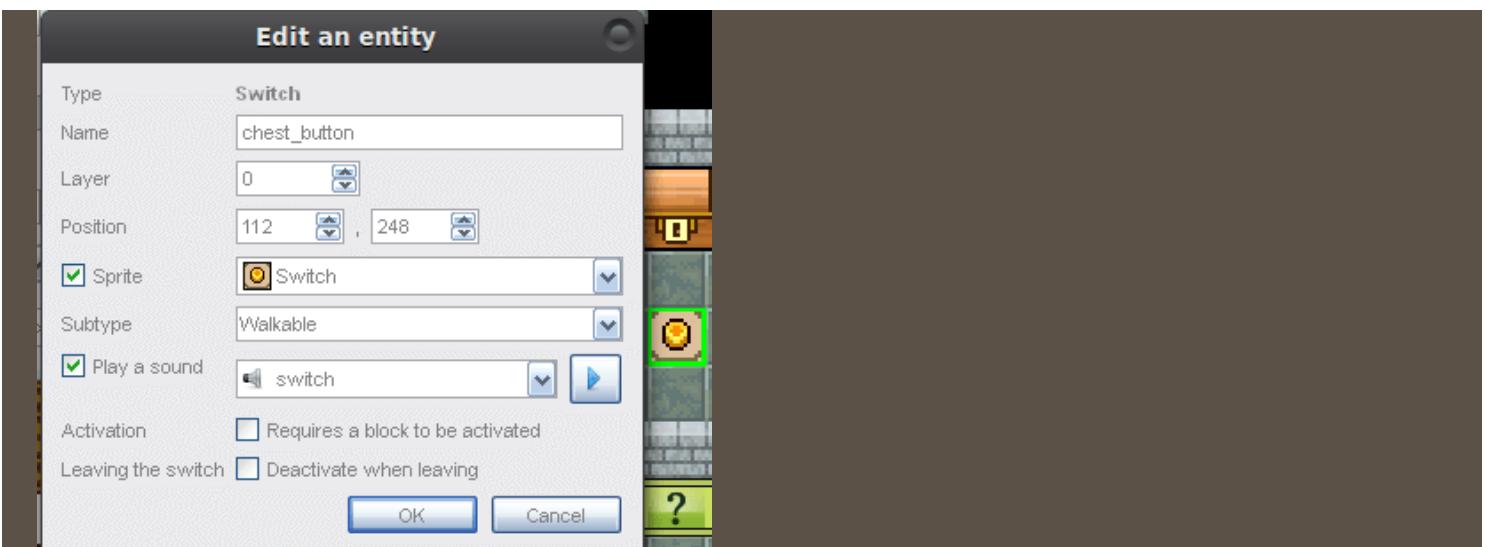
## Add Switch

The switch is the button with the red dot next to the block entity. You can step on the switch to activate certain events and there are a few other ways to activate it.



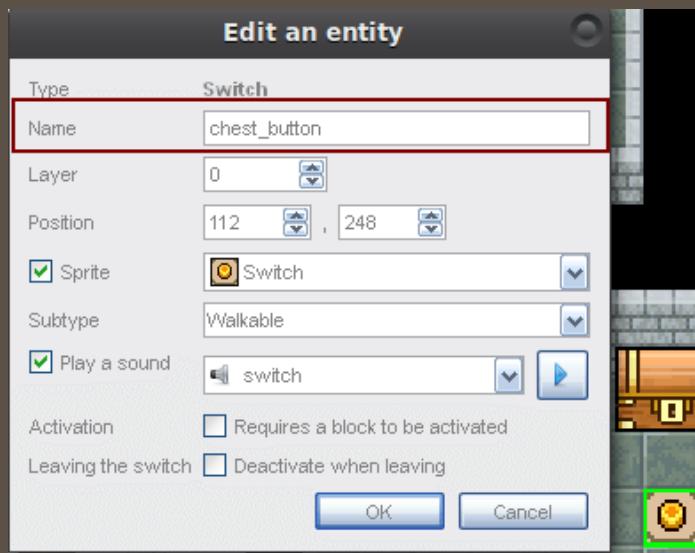
## Switch Properties

Type	Option
Name	A name is needed for scripting reasons. You can leave it blank, but unless you plan to have it there just for show, then a name is needed.
Layer	The layer you want the entity on.
Position	The coordinates the entity is at. You can manually change them or move the entity with the cursor.
Sprite	Pick the sprite image for the switch.
Subtype	<p><b>Walkable:</b> Ability to walk on the switch.</p> <p><b>Solid:</b> You can hit the switch with the 'c' key to activate it. Usually, that is a weapon like a sword.</p> <p><b>Arrow Target:</b> You can shoot an arrow at it. There is a built in bow and arrow.</p>
Play Sound	Pick a sound that you want played when the switch activates.
Activation	You can require a block to be on the switch for it to be activated.
Leaving switch	The switch can be deactivated if this option is checked.



## Walkable Switch Coding

In order to code a script one must give it a name. I gave it the name `chest_button`. In the following example I use the `on_activated()` function.

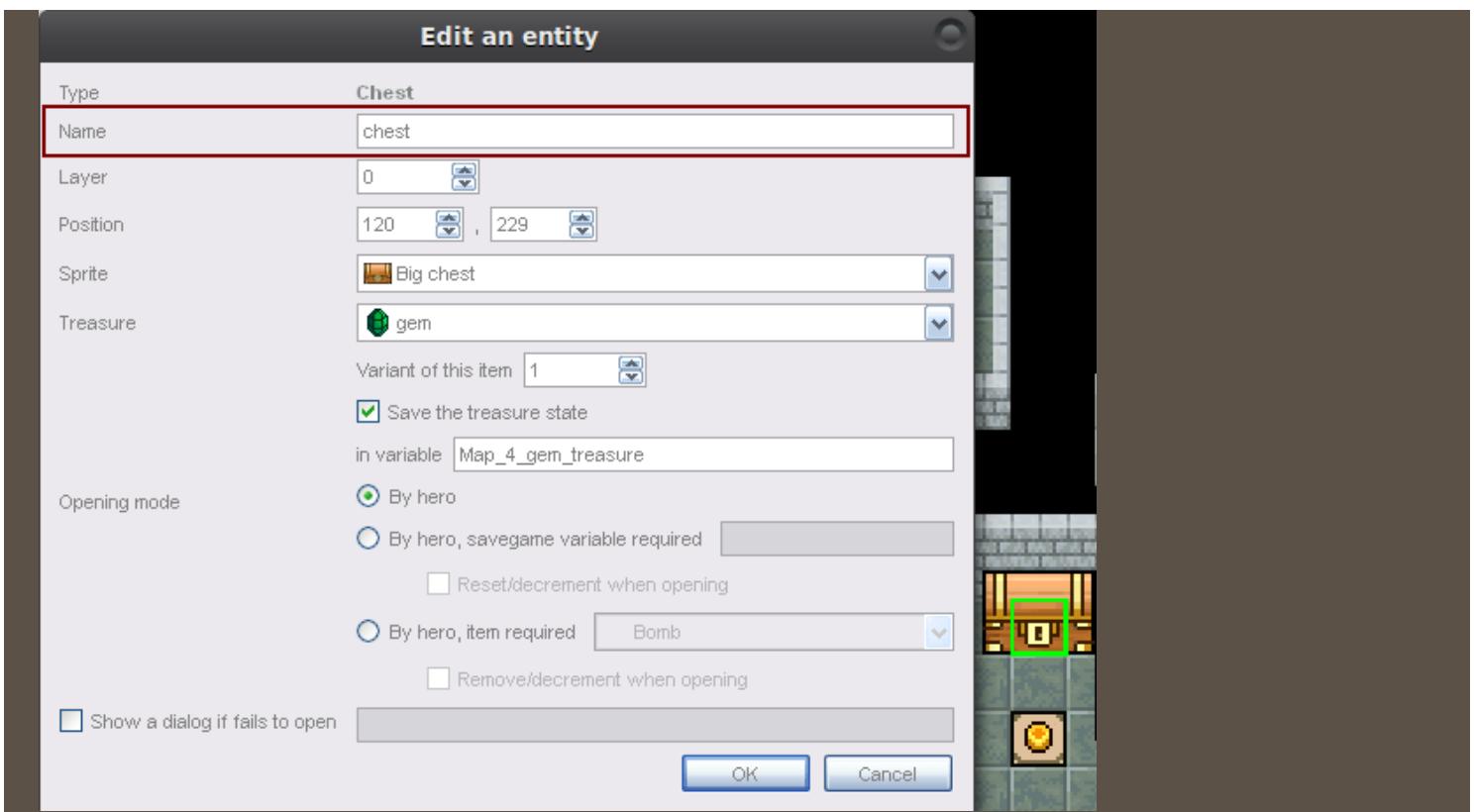


```
function chest_button:on_activated()
```

`on_activated` means that something will happen when the switch is stepped on. In this case a sound will be played and a chest will appear.

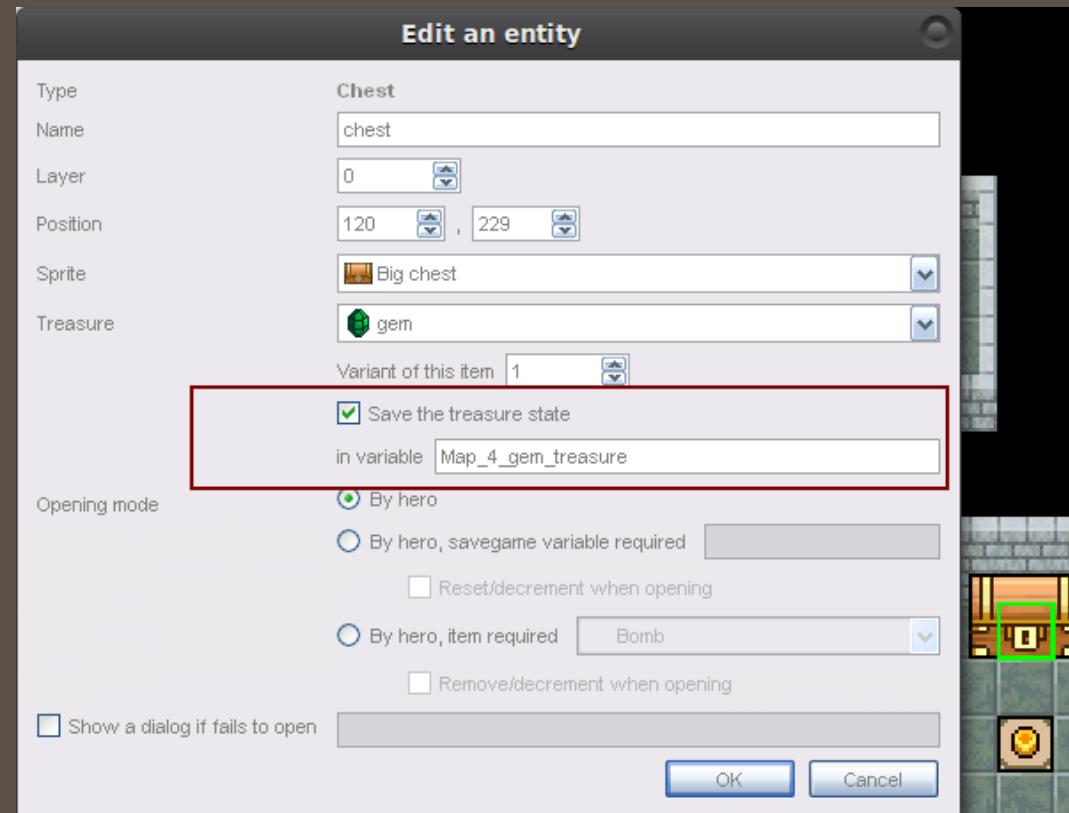
```
--chest_1
function chest_button:on_activated()
    sol.audio.play_sound("chest_appears")
    chest:set_enabled(true)
end
```

I named the chest the name `chest`. The function `Set_enabled(true/false)` is used to make it appear or not appear.



The first thing that needs to be carried out is that the chest needs a save value because we will be using this to see if the chest has been opened. I used the save value name `Map_4_gem_treasure`.

```
if game:get_value("Map_4_gem_treasure") then
```



Next we want to show the button as activated and not deactivated. The `set_activated(true/false)` function is used in order to achieve this.

generated by haroopad

```
chest_button:setActivated(true)
```

Now we want the chest to be invisible if the switch is not activated yet.

```
else
    chest:setEnabled(false)
end
end
```

The result is the following script.

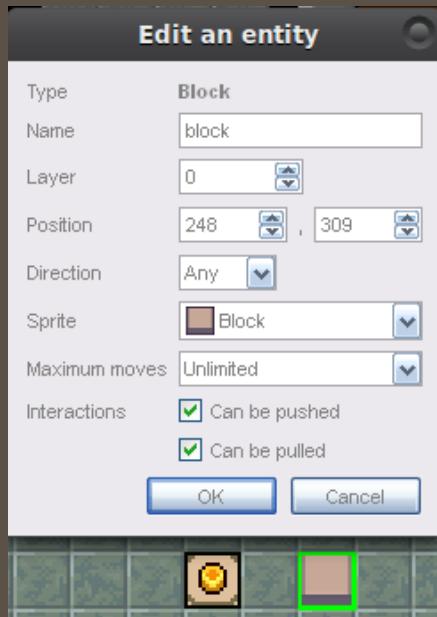
```
--chest_1
function map:on_started()
    if game:get_value("Map_4_gem_treasure") then
        chest_button:setActivated(true)
    else
        chest:setEnabled(false)
    end
end

--chest_1
function chest_button:on_activated()
    sol.audio.play_sound("chest_appears")
    chest:setEnabled(true)
end
```

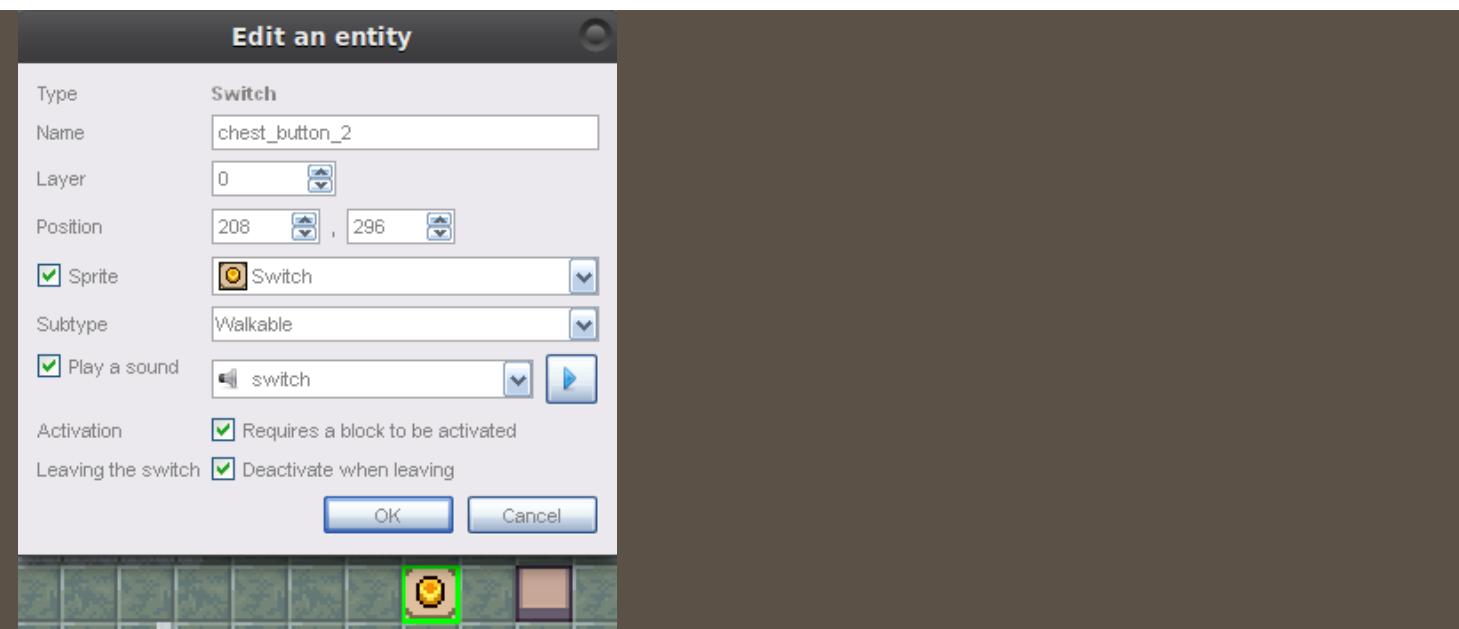
## Block Switch Coding

The block coding is almost exactly the same, so I will not be typing everything over again. I will cover the parts that are different.

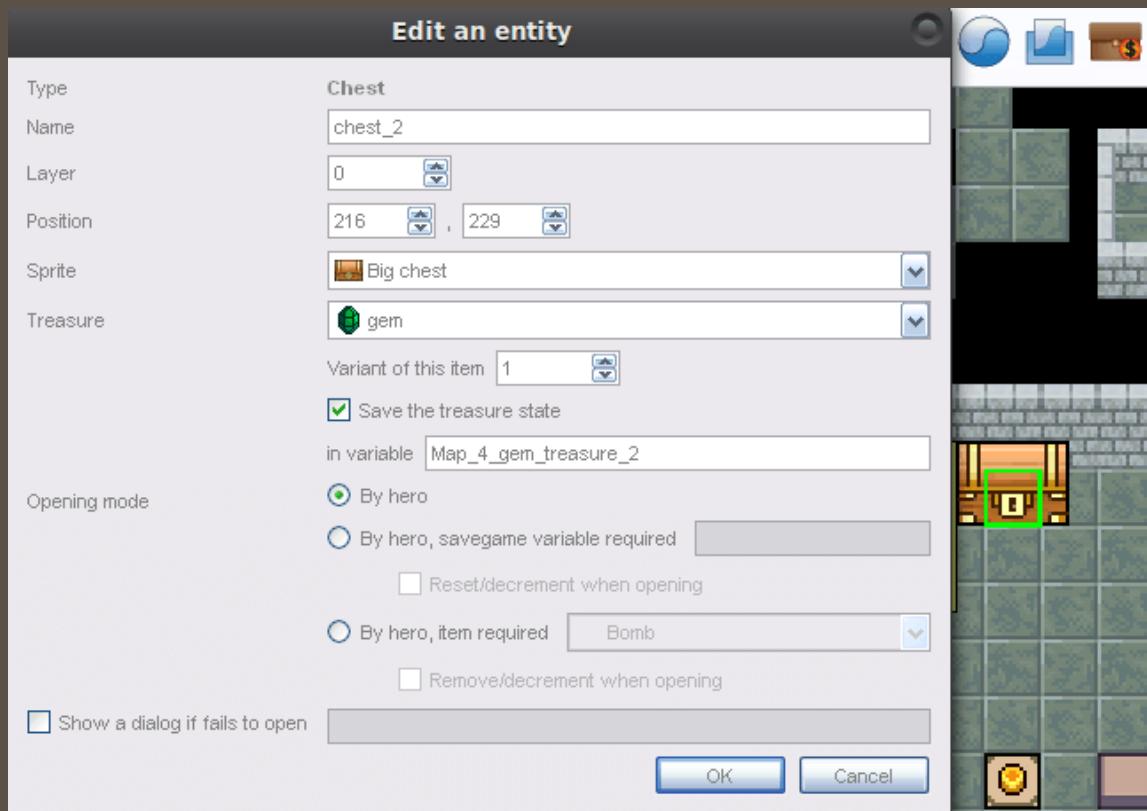
Name the block. I named it **block**.



Name the switch. I named it **chest\_button\_2**. Check **require a block to be activated** and **deactivate when leaving**.



Name the chest. I named it `chest_2`. I gave it the save state name `Map_4_gem_treasure_2`.



The script. The switch and chest will activate/deactivated if the block is moved on/off it.

```
function map:on_started()
    --chest_2
    if game:get_value("Map_4_gem_treasure_2") then
        chest_button_2:setActivated(true)
    else
        chest_2:setEnabled(false)
    end
end

--chest_2
```

```
function chest_button_2:on_activated()
    sol.audio.play_sound("chest_appears")
    chest_2:set_enabled(true)
end
```

Now we want the chest to vanish when the block leaves the button. We want this to happen only when the chest is not opened.

We need to check when the button is inactivated and will use the function `[on_inactivated()]` for this.

```
function chest_button_2:on_inactivated()
```

After that we need to check if the chest is open or not. We use the function `[is_open()]`. The chest needs to be `[set_enabled(false)]` because we do not want the chest to appear if the button is inactive.

```
if not chest_2:is_open() then
    chest_2:set_enabled(false)
end
end
```

The script will look like this.

```
function chest_button_2:on_inactivated()
    if not chest_2:is_open() then
        chest_2:set_enabled(false)
    end
end
```

Now we need to check the block position. We want the block to still be on the switch when we leave and come back to the map.

Check if the chest is open.

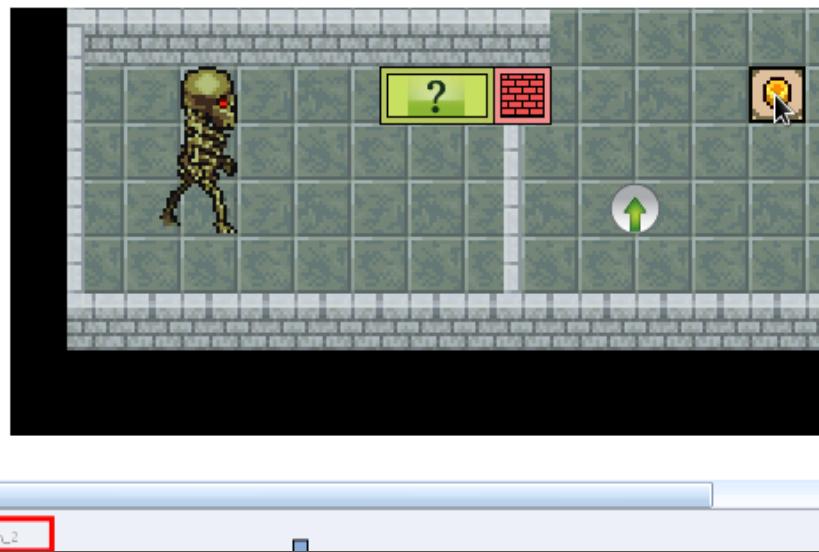
```
if chest_2:is_open() then
```

If the chest is open, then set button activated to true, otherwise set the chest to false.

```
    chest_button_2:set_activated(true)
else
    chest_2:set_enabled(false)
```

Set the position of the block. Use the function `[set_position(x,y)]`.

Hover over the location of switch and set the block to that position. Adjust it if you have to.



```
block:set_position(216,308)
```

The result.

```
function map:on_started()
    if chest_2:is_open() then
        chest_button_2:setActivated(true)
        block:setPosition(216,308)
    else
        chest_2:setEnabled(false)
    end
end
```

The full script.

Tip: When I say the chest is false, that means the chest does not appear.

```
function map:on_started()
    --chest_2
--Save chest state if chest is opened and set button activation to true. Otherwise, the chest is saved as closed and button activation is false.
    if game:getValue("Map_4_gem_treasure_2") then
        chest_button_2:setActivated(true)
    else
        chest_2:setEnabled(false)
    end

--if chest is open, then have the button activated and set block position. Otherwise, the chest is closed and button activation is false.
    if chest_2:is_open() then
        chest_button_2:setActivated(true)
        block:setPosition(216,308)
    else
        chest_2:setEnabled(false)
    end
end

--Set chest_2 to activate if the block is on the button
function chest_button_2:on_activated()
    sol.audio.play_sound("chest_appears")
    chest_2:setEnabled(true)
end

--Set chest_2 to deactivated if the block is not on or move off the button and if the chest is not open
function chest_button_2:on_inactivated()
```

generated by haroopad

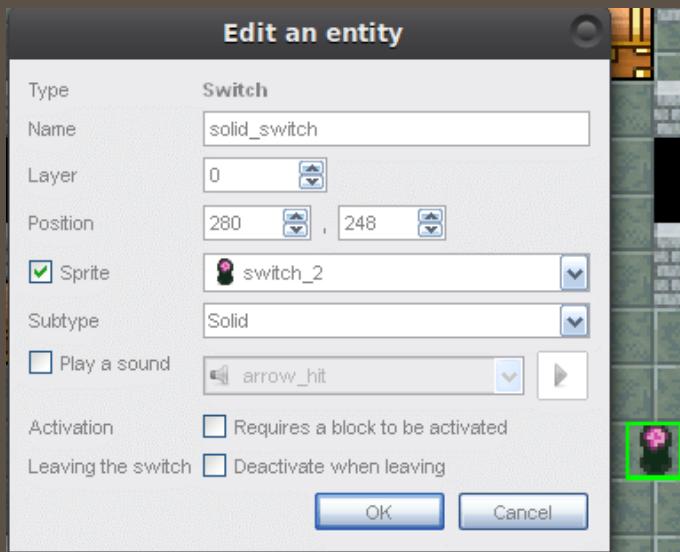
```

if not chest_2:is_open() then
    chest_2:set_enabled(false)
end
end

```

## Solid Switch Coding

The same as the walkable switch. Just set it to `solid` and press `C` to activate it. There will be an example in the dynamic Tile section.

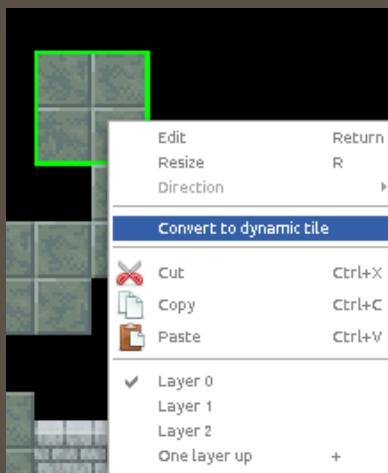


## Dynamic Tile

Dynamic tiles are entities, and they can be manipulated like an entity once a name is given to them. You can script it and all that jazz. Tiles are originally static meaning they cannot be used with scripts until converted to Dynamic.

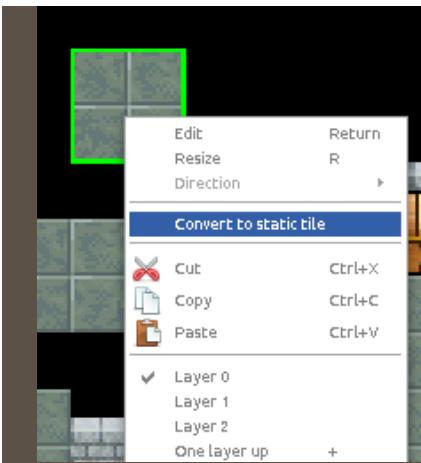
### Convert to Dynamic

Right click > Convert to dynamic tile



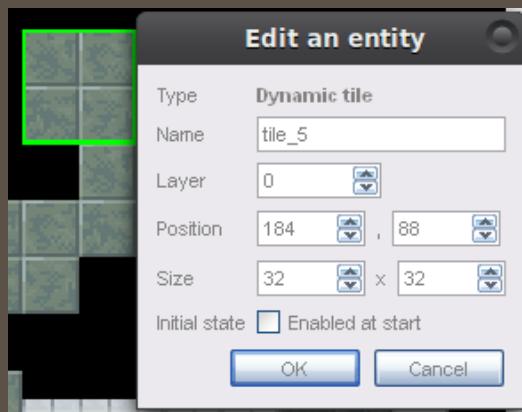
### Convert to Static

Right click > Convert to static tile



## Dynamic Properties

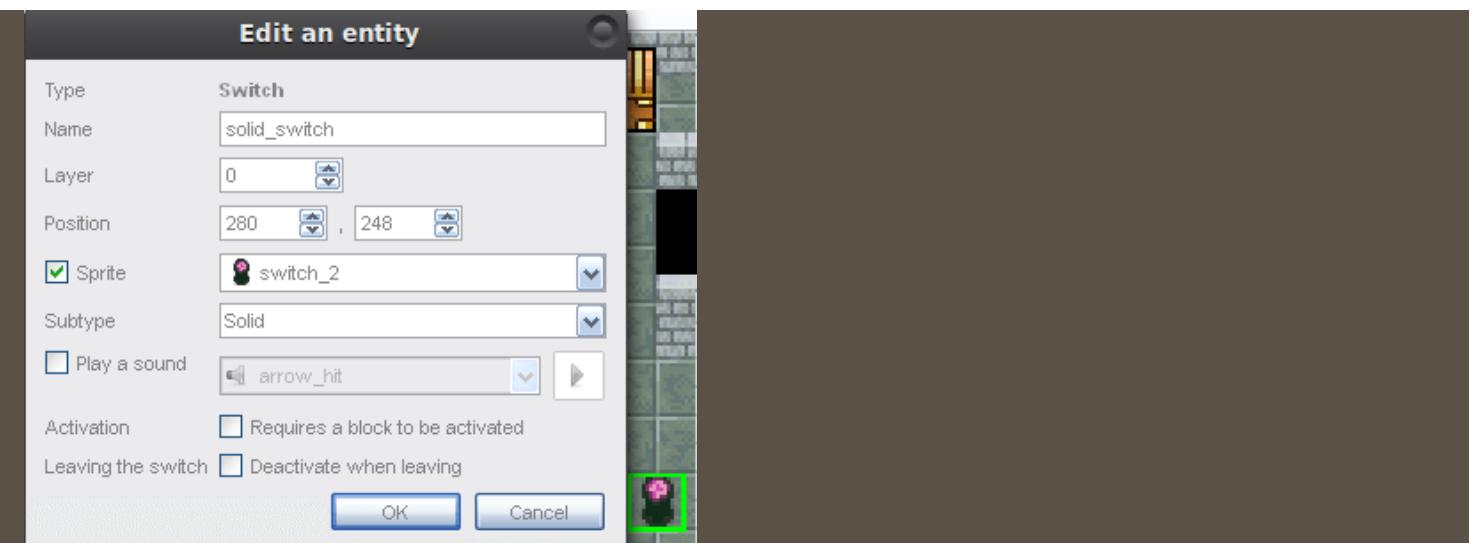
Type	Option
Name	A name is needed for scripting reasons. You can leave it blank, but unless you plan to have it there just for show, then a name is needed.
Layer	The layer you want the entity on.
Position	The coordinates the tile is at. You can manually change them or move the entity with the cursor.
Size	The dimension size of the tile.
Initial State	Display or do not display at start of game.



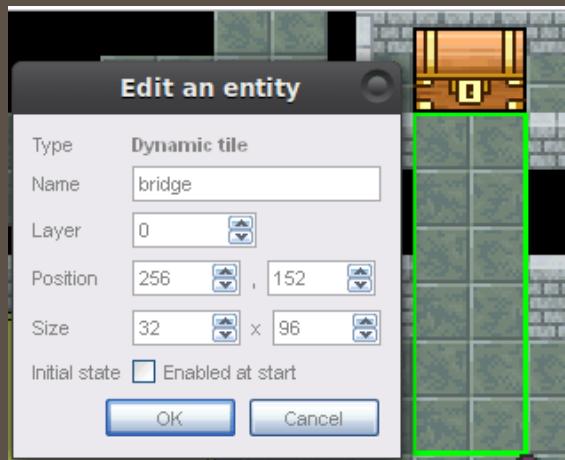
## Dynamic Scripting

Dynamic scripting is not really different from scripting with entities.

Give a name to a switch. I am using a solid switch in this case and gave it the name `solid_switch`.



The next step is to convert a normal tile to dynamic, so we can script it. Make sure that `enabled at start` is unchecked.



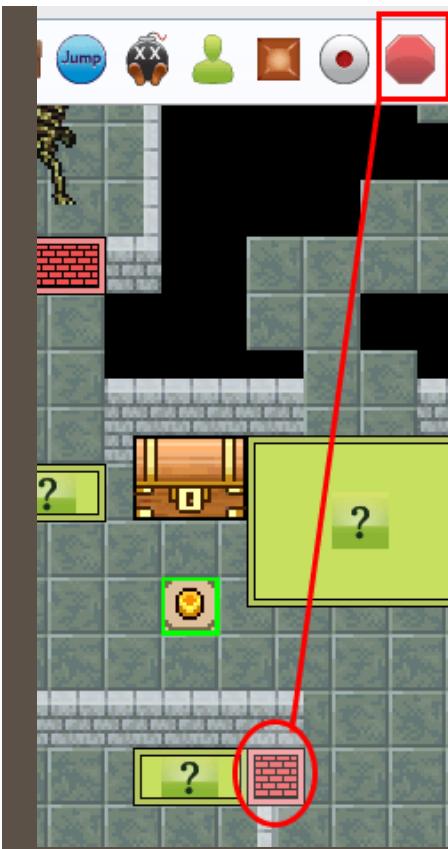
Now we will make the bridge to appear when the solid switch is hit. We will do this with the following script. Not really anything to explain because we already covered these functions.

```
--Solid Switch bridge
function solid_switch:on_activated()
    bridge:set_enabled(true)
end
```

## Wall Entity

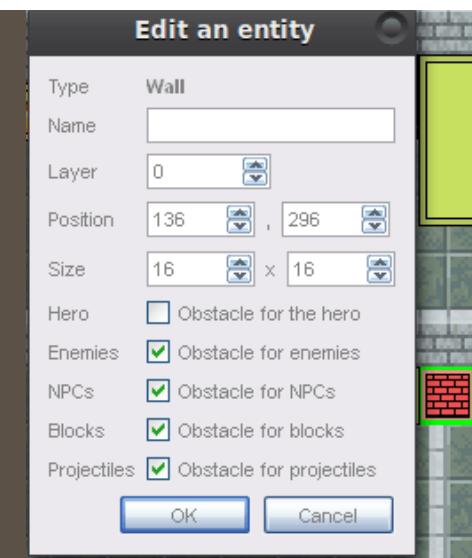
### Add Wall Entity

Click on the stop sign like image by the switch entity to add the wall entity. It can be stretched to fit a large area and blocks paths for certain obstacles.



## Wall Entity Properties

Type	Option
Name	A name is needed for scripting reasons. You can leave it blank, but unless you plan to have it there just for show, then a name is needed.
Layer	The layer you want the entity on.
Position	The coordinates the entity is at. You can manually change them or move the entity with the cursor.
Size	The dimension size of the entity.
	<b>Hero</b> - Prevents the hero from passing. <b>Enemies</b> - Prevents the enemies from passing. <b>NPCs</b> - Prevents the NPCs from passing. <b>Blocks</b> - Prevents the blocks from passing. <b>Projectiles</b> - Prevents the projectiles from passing.
Obstacles	



## Wall Entity Scripting

The scripting is no different from any other entity. There will be an example in the sensor section, but let me cover another function before that.

The function is `[remove()]`. This function deletes or erases an entity from the map. One must be really careful when using this function because annoying messages can appear if not taken care of properly.

The reason why I am covering this function is that one will want the wall to vanish at a certain point.

Let us pretend that the wall is blocking the enemy. On some condition, like stepping on a switch, the wall can be removed.

```
enemy_wall:remove()
```

Now every time that switch is activated an annoying message will appear in the terminal because the entity was removed and no longer exists (`nil`).

The following script is a way to avoid the annoying message. Of course, there are many other ways and this way may not be very professional. One could just show an entity as activated instead of leaving the space blank or remove the entity too. Removing the switch avoids the error message all together.

```
if enemy_wall == nil then
--nothing happens
else
    enemy_wall:remove()
end
```

## Sensor Entity

### Add Sensor Entity

The sensor entity is right next to the wall Entity. It is a green circle with a question mark on it. It can be stretched to fit a large area and can detect the hero when it gets into the boundary.



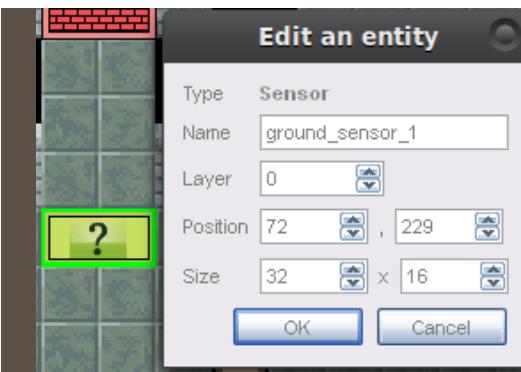
## Sensor Properties

Type	Option
Name	A name is needed for scripting reasons. You can leave it blank, but unless you plan to have it there just for show, then a name is needed.
Layer	The layer you want the entity on.
Position	The coordinates the entity is at. You can manually change them or move the entity with the cursor.
Size	The dimension size of the entity.

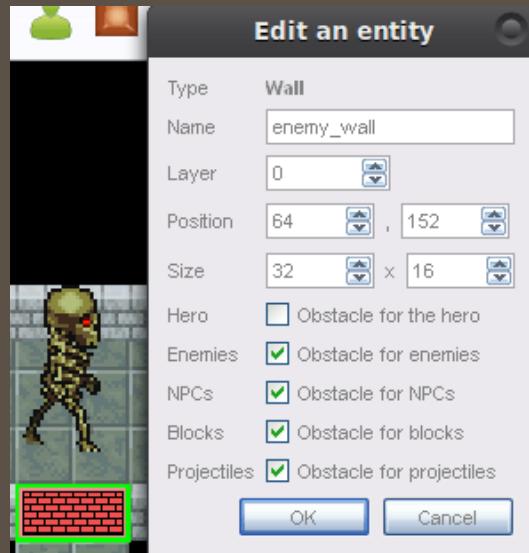


## Sensor Scripting

This is an example of wall and sensor scripting. First off, do not name your sensor entity the name `sensor`. You can, but it will interfere with a shortcut, at least when I tried it. Instead, we will be using the name `ground_sensor`.



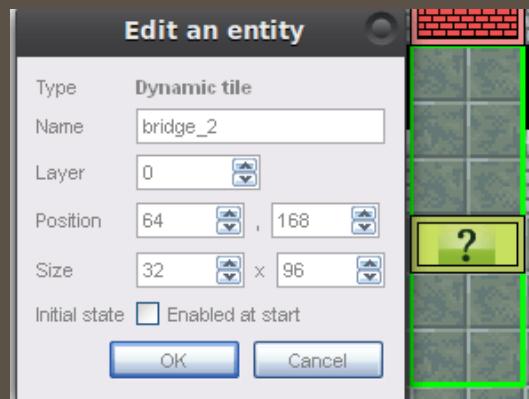
Name the wall `enemy_wall` or any name you want.



Now let us go over the scripting part. There is a good function that goes good with sensors. The function is `hero:save_solid_ground`. This function teleports the hero back to the sensor in this case. That means if the hero falls into a hole he/she/it will go back to the sensor instead of near the falling point.

```
function ground_sensor_1:onActivated()
    hero:saveSolidGround()
```

Enable the bridge with the following script. Make sure that `enabled at start` is unchecked.



```
bridge_2:setEnabled(true)
```

Remove the wall when the sensor is touched and prevent annoying message because the wall entity will no longer exists (nil).

generated by haroopad

```

if enemy_wall == nil then
--nothing happens
else
    enemy_wall:remove()
end

```

The final script:

```

--Bridge sensor
function ground_sensor_1:on_activated()
    hero:save_solid_ground()
    bridge_2:set_enabled(true)

    if enemy_wall == nil then
--nothing happens
    else
        enemy_wall:remove()
    end
end

```

Tip: There is a shortcut for `hero:save_solid_ground()`. Instead of typing `hero:save_solid_ground()` over and over again for each new sensor, you could use the following script.

```

--Do not name your sensor, "sensor." Do not give it the same name. Use a unique name like "ground_
for sensor in map:get_entities("ground_sensor_") do

    function sensor:on_activated()
        hero:save_solid_ground()
    end
end

```

To start, the “for sensor” part refers to the sensors on the map. The `get_entities()` function refers to the names of the sensors. The names are all `ground_sensor_` with some number index. For example, `ground_sensor_1`. The `get_entities()` functions grabs or gets the sensors with the same name, but with a number difference (number index).

```
for sensor in map:get_entities("ground_sensor_") do
```

Now we will make a function for the sensors on the map. We will `save_solid_ground()` with this function. The “sensor” in the `on_activated` function refers to all the sensors on the map.

```

function sensor:on_activated()
    hero:save_solid_ground()
end

```

The resulting script.

```

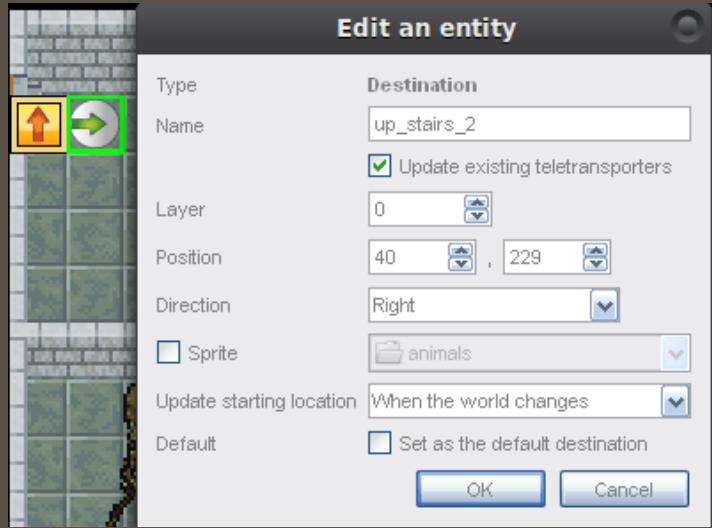
--Do not name your sensor, "sensor." Do not give it the same name. Use a unique name like "ground_
for sensor in map:get_entities("ground_sensor_") do

    function sensor:on_activated()
        hero:save_solid_ground()
    end
end

```

We covered the `set_position` function, but we can get the position of an entity as well. The function for getting an entity is `get_position()`.

Using the `get_position()` function is quite easy. For example, if you wanted the hero to teleport to an entity to trick or torture the person playing your game, then you could have the player go to a destination entity. I used the name `up_stairs_2`.



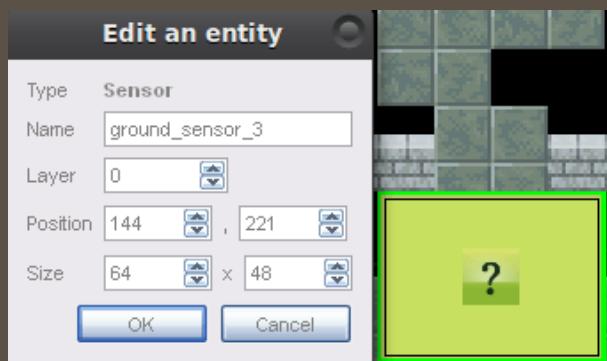
```
hero:set_position(up_stairs_2:get_position())
```



```
function ground_sensor_2:onActivated()
    hero:set_position(up_stairs_2:get_position())
end
```

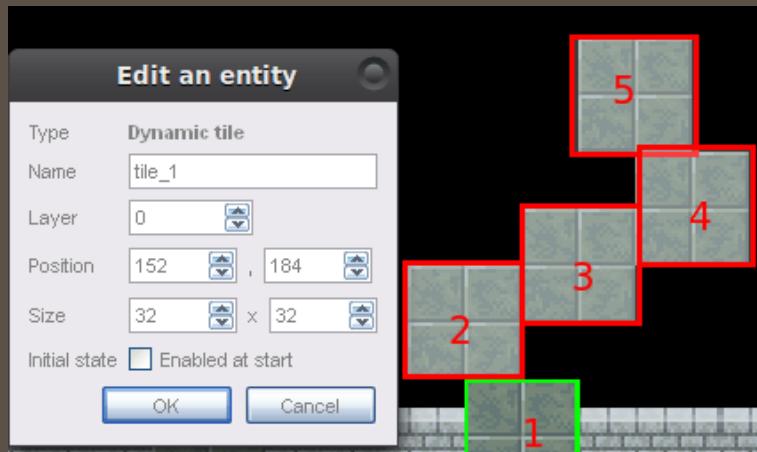
## Sensor Multiple Dynamic Tiles

This a shortcut for bulk numbers of dynamic tiles with the same name. The first thing we will do is add a sensor. I named it `ground_sensor_3`.



generated by haroopad

The next step is to add a dynamic tile and name it. Make sure to set up the properties before copying. The number index in the name will change as you copy it. We do not want the dynamic tiles enabled at start, so uncheck `enabled at start`.



We will use a timer for a delay with each dynamic tile. One will appear after the other. You could shorten the timer for an instant result.

The first thing that needs to be done is to make a variable called `tile_index`. We will be concatenating this with the name of the dynamic tiles, `tile_` with some number index. We will assign the value `1` to `tile_index`.

```
local tile_index = 1
```

The second step is to make a timer. You can assign the time delay you want.

```
sol.timer.start(400, function()
```

The third step is to make a variable called `previous_tile`. This will activate the first tile.

Now let us set up the `previous_tile` variable.

```
local previous_tile =
```

We will assign it to a function called `map:get_entity`. We will use this to get the names of all the dynamic tiles we want. One wants the names of all the `tile_` dynamic tiles with some number index. For example, `tile_1`.

```
local previous_tile = map:get_entity("tile_"
```

Lastly, we will concatenate the `tile_index`. That will name it `tile_1` because the value of `tile_index` is `1`.

```
local previous_tile = map:get_entity("tile_" .. tile_index)
```

The fourth step `next_tile` is almost the same as `previous_tile`. The only difference is that we will be adding `1` to the `tile_index`.

```
local next_tile = map:get_entity("tile_" .. (tile_index + 1))
```

The fifth step is to set up an increment for `next_tile`. This is needed for `tile_` indexes 2, 3, 4, and 5.

Tip: Increment is adding a value to itself.

```
tile_index = tile_index + 1
```

The seventh step is to enable the `previous_tile`. If it is set to false, then the tiles will vanish one after the other. That could be useful for timed puzzles.

### False example

```
previous_tile:set_enabled(false)
```

However, we want it to be true.

```
previous_tile:set_enabled(true)
```

Step 8 will be to stop the index from going beyond the number of tiles we have. The number of dynamic tiles is five for me. We will return `next_tile` false if it becomes nil. For instance, `tile_6` will be nil because it does not exist.

```
if next_tile == nil then
    --finished
    return false
end
```

The 9th step is to set `next_tile` to true, but in this case it really does not matter if it is true or false. There may be transition differences.

```
next_tile:set_enabled(true)
--Return true or it will stop after the second tile.
return true
end
end
```

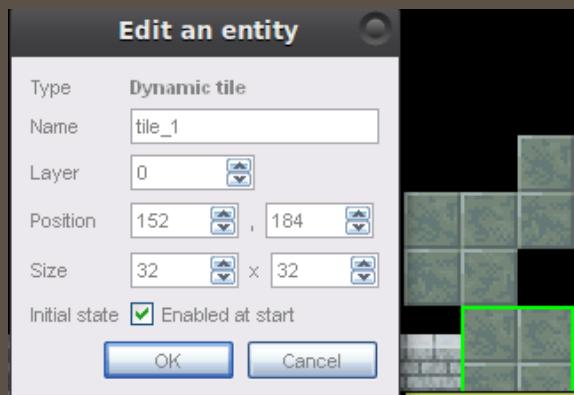
The final script.

```
function ground_sensor_3:on_activated()
hero:save_solid_ground()

local tile_index = 1
sol.timer.start(400, function()
    local previous_tile = map:get_entity("tile_" .. tile_index)
    local next_tile = map:get_entity("tile_" .. (tile_index + 1))
    tile_index = tile_index + 1
    previous_tile:set_enabled(true)
    if next_tile == nil then
        --finished
        return false
    end
    next_tile:set_enabled(true)
    --Return true or it will stop after the second tile.
    return true
end)
end
```

Lastly, if you check `enabled at start` for the initial state of all the `tile` dynamic tiles, then set `previous_tile` and `next_tile` enabled to false, the dynamic tiles will vanish one after the other instead of appearing on after another.

```
previous_tile:set_enabled(false)
next_tile:set_enabled(false)
```



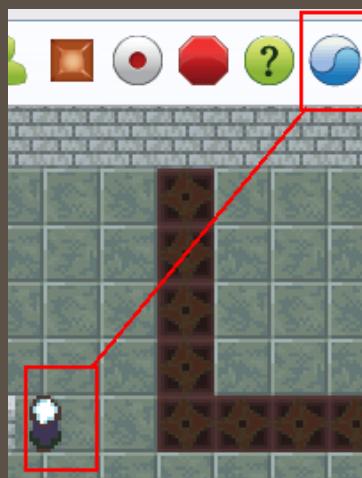
```
function ground_sensor_3:onActivated()
    hero:saveSolidGround()

    local tile_index = 1
    sol.timer.start(400, function()
        local previous_tile = map:getEntity("tile_" .. tile_index)
        local next_tile = map:getEntity("tile_" .. (tile_index + 1))
        tile_index = tile_index + 1
        previous_tile:setEnabled(false)
        if next_tile == nil then
            --finished
            return false
        end
        next_tile:setEnabled(false)
        --Return true or it will stop after the second tile.
        return true
    end)
end
```

## Crystal Switch Entity

### Add Crystal Switch

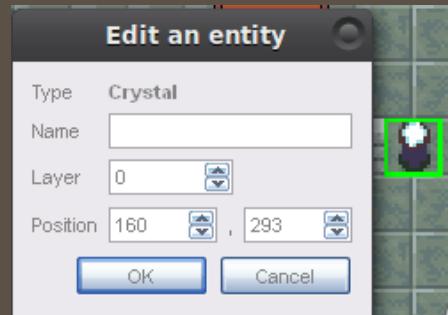
The crystal switch works with the crystal block. A crystal block rises or lowers when a crystal switch is hit with key **C**. For example, the block will lower if the initial state of the block is set to lowered.



### Crystal Switch Properties

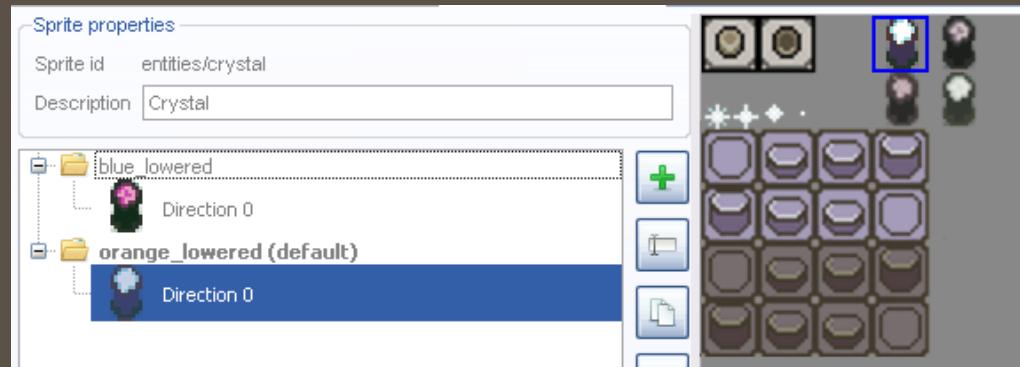
generated by haroopad

Type	Option
Name	A name is needed for scripting reasons. You can leave it blank, but unless you plan to have it there just for show, then a name is needed.
Layer	The layer you want the entity on.
Position	The coordinates the entity is at. You can manually change them or move the entity with the cursor.



## Default Crystal Switch Graphic Names

As far as I know the animation keywords for switches are `blue_lowered` and `orange_lowered`. The default animation is the color that one wants to see first.



## Crystal Block Entities

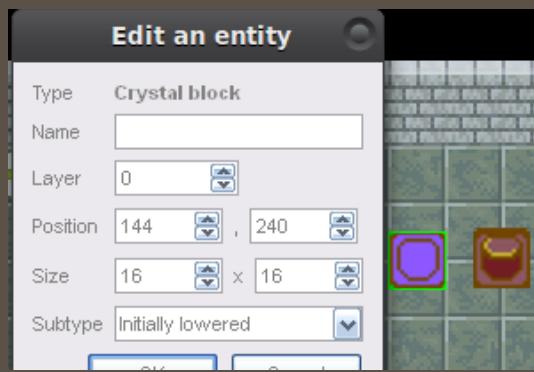
The crystal block works with the crystal switch. A crystal block rises or lowers when a crystal switch is hit with key `c`. For example, the block will lower if the initial state of the block is set to lowered.

### Add Crystal Block



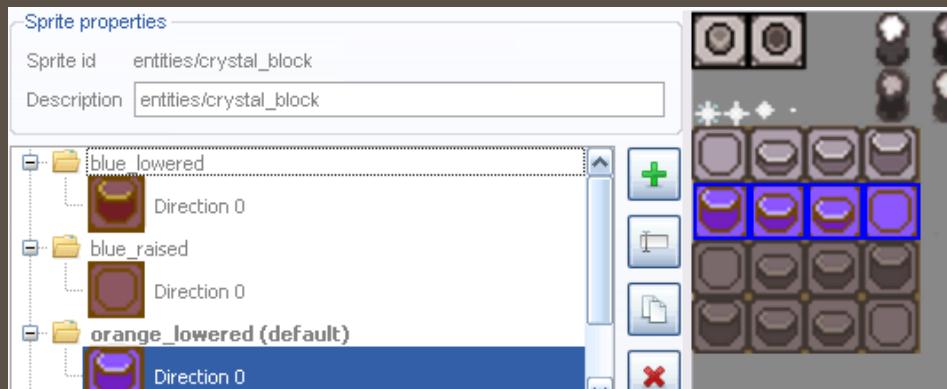
## Crystal Block Properties

Type	Option
Name	A name is needed for scripting reasons. You can leave it blank, but unless you plan to have it there just for show, then a name is needed.
Layer	The layer you want the entity on.
Position	The coordinates the entity is at. You can manually change them or move the entity with the cursor.
Size	The dimension size of the entity.
Subtype	Initially lowered or raised. Basically, start the block lowered or raised at the start of the map.



## Default Crystal Block Graphic Names

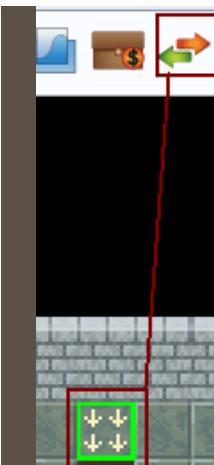
As far as I know the keywords are `blue_lowered`, `blue_raised`, `orange_lowered`, and `orange_raised`. I tried different keywords for the animations, but the blocks failed to show up on the map when playtesting the game.



## Stream Entity

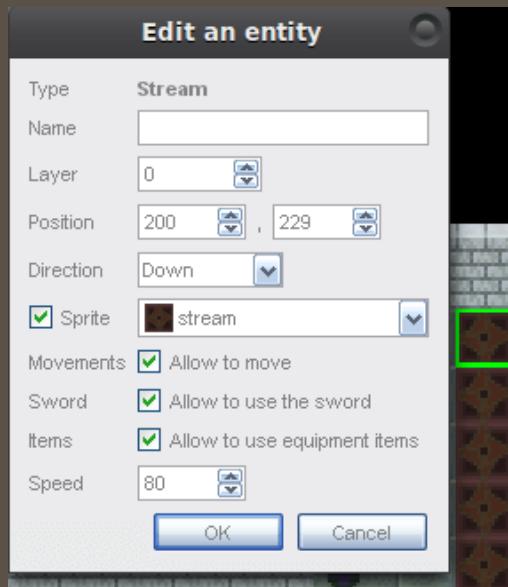
The stream entity pushes the player or moves him/her/it in the desired direction. It can also be used to slow the player down. For example, gravity, spinning wheel, and/or a sandstorm. A developer might want to have some force to slow down the player.

### Add Stream



## Stream Properties

Type	Option
Name	A name is needed for scripting reasons. You can leave it blank, but unless you plan to have it there just for show, then a name is needed.
Layer	The layer you want the entity on.
Position	The coordinates the entity is at. You can manually change them or move the entity with the cursor.
Direction	You can pick the direction you want the stream to go or push the player. Up, down, left, left-up, right-up, and right.
Sprite	Pick the sprite image for the switch.
Movement	Allow the player to move when on the stream.
Sword	Allow the player to use the sword during movement.
Items	Allow the usage of items when on the stream.
Speed	The speed at which the stream will move the player.



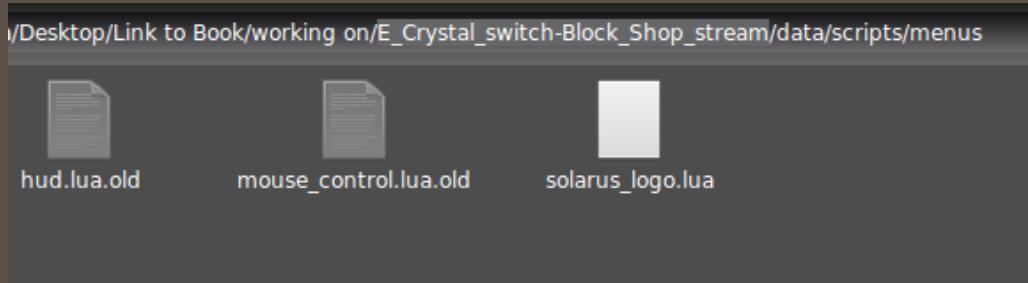
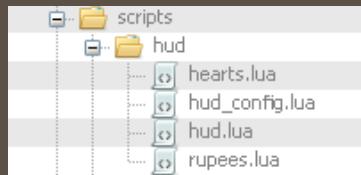
## HUD

HUD stands for [heads up display]. It is the health bar, money counter, etc.

There are many scripts from Christopho's games that can be used. You can check his [YouTube Tutorial](#) for that information. I will be covering the heart display and money counter.

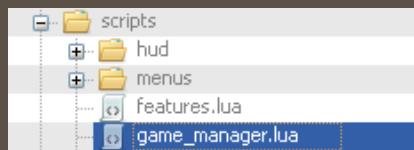
## Setup HUD Scripts

The 1st step will be to add the HUD scripts folder (Get them from sample > [E\\_Crystal\\_switch-Block\\_Shop\\_stream.zip](#).) and remove the current HUD in the menu directory or rename the scripts to [.old](#).



Remember to comment out the old hud in the [game\\_manager.lua](#) and require the HUD from the manager.

```
require("scripts/hud/hud")
```



```
game:register_event("on_started", function()
-- [
--   -- HUD menu.
local hud = require("scripts/menu/hud")
sol.menu.start(game, hud)
hud:create(game)

-- Mouse control.
local mouse_control = require("scripts/menu/mouse_control")
sol.menu.start(game, mouse_control)
mouse_control:create(game, hud)
-- ]
local hero = game:get_hero()

require("scripts/menu/alttp_dialog_box")
require("scripts/multi_events")
require("scripts/hud/hud")

local game_manager = {}
```

Add the [green\\_digits](#) and [white\\_digits](#) fonts to the font directory. Get them from sample > [E\\_Crystal\\_switch-Block\\_Shop\\_stream.zip](#). Remember to add them in the manager too.

generated by haroopad



## Hud Configurations `hud_config.lua`

You can change the coordinates of the counter and heart health display from this script. Change the x,y coordinates until you like what you see.

```
-- Hearts meter.
{
    menu_script = "scripts/hud/hearts",
    x = -88,
    y = 0,
},
-- Rupee counter.
{
    menu_script = "scripts/hud/rupees",
    x = 121,
    y = 10,
}
```

## Health Heart Display `hearts.lua`

Health is the life points the player has until he dies or faints. The way this script works is that it draws a surface of your heart image. The image will have full, half empty, and empty graphics.



You might have to adjust the surface for your image if you have super huge hearts or other graphics related to this system. This part can be found at the beginning of the `hearts.lua` script.

```
hearts.surface = sol.surface.create(80, 16)
```

If your hearts or graphics are like mine (beyond 8x8 in dimensions), then you will have to adjust them. At first mine turned out like this.



The script is currently set up for 8x8 hearts, but mine is 16x16 hearts. You will have to adjust the `function hearts:rebuild_surface()`.

```
function hearts:rebuild_surface()

    hearts.surface:clear()

    local life = hearts.current_life_displayed
    local max_life = hearts.max_life_displayed
    for j = 1, max_life do
        if j % 2 == 0 then
            local x, y
            if j <= 20 then
                x = 4 * (j - 2)
                y = 0
            else
                x = 4 * (j - 20)
                y = 16
            end
            hearts.surface:drawImage(heart, x, y)
        end
    end
end
```

```

        x = 4 * (j - 22)
        y = 8
    end
    if life >= j then
        hearts_img:draw_region(0, 0, 8, 8, hearts.surface, x, y)
    else
        hearts_img:draw_region(16, 0, 8, 8, hearts.surface, x, y)
    end
end
if life % 2 == 1 then
    local x, y
    if life <= 20 then
        x = 4 * (life - 1)
        y = 0
    else
        x = 4 * (life - 21)
        y = 8
    end
    hearts_img:draw_region(8, 0, 8, 8, hearts.surface, x, y)
end
end

```

To be specific this part:

```

if j <= 20 then
    x = 4 * (j - 2)
    y = 0
else
    x = 4 * (j - 22)
    y = 8
end
if life >= j then
    hearts_img:draw_region(0, 0, 8, 8, hearts.surface, x, y)
else
    hearts_img:draw_region(16, 0, 8, 8, hearts.surface, x, y)
end
end
if life % 2 == 1 then
    local x, y
    if life <= 20 then
        x = 4 * (life - 1)
        y = 0
    else
        x = 4 * (life - 21)
        y = 8
    end
end

```

In order to fix this I have to adjust `draw_region` functions. `Draw_region` draws certain locations on a surface.

Note: You can set up your graphics in the sprite editor to get coordinates easier.

Location 1 is at `(0, 0, 16, 16,)`. The 2 zeros are the positions. The y position does not change (`px, py, sx, sy,`) and the next 2 spots `16, 16` are the sprite dimensions.

`px,py` = position

`sx,sy` = sprite dimensions

```
hearts_img:draw_region(0, 0, 16, 16, hearts.surface, x, y)
```

generated by haroopad

Location 2 is at `(16, 0, 16, 16, )`.

```
hearts_img:draw_region(16, 0, 16, 16, hearts.surface, x, y)
```

Location 3 is at `(32, 0, 16, 16, )`.

```
hearts_img:draw_region(32, 0, 16, 16, hearts.surface, x, y)
```

The script result is the hearts are almost fixed. Notice that spacing issue?



```
if life >= j then
    for j = 1, max_life do
        if j % 2 == 0 then
            local x, y
            if j <= 20 then
                x = 4 * (j - 2)
                y = 0
            else
                x = 4 * (j - 22)
                y = 8
            end
            if life >= j then
                hearts_img:draw_region(0, 0, 16, 16, hearts.surface, x, y)
            else
                hearts_img:draw_region(32, 0, 16, 16, hearts.surface, x, y)
            end
        end
    end
    if life % 2 == 1 then
        local x, y
        if life <= 20 then
            x = 4 * (life - 1)
            y = 0
        else
            x = 4 * (life - 21)
            y = 8
        end
        hearts_img:draw_region(16, 0, 16, 16, hearts.surface, x, y)
    end
end
```

One must adjust the variable `x` to fix the spacing issue. I changed them all to

`x = 5`.



```
if life >= j then
    for j = 1, max_life do
        if j % 2 == 0 then
            local x, y
            if j <= 20 then
                x = 5 * (j - 2)
                y = 0
            else
                x = 5 * (j - 22)
                y = 8
            end
            hearts_img:draw_region(0, 0, 16, 16, hearts.surface, x, y)
        end
    end
    if life % 2 == 1 then
        local x, y
        if life <= 20 then
            x = 5 * (life - 1)
            y = 0
        else
            x = 5 * (life - 21)
            y = 8
        end
        hearts_img:draw_region(16, 0, 16, 16, hearts.surface, x, y)
    end
end
```

```

        end
        if life >= j then
            hearts_img:draw_region(0, 0, 16, 16, hearts.surface, x, y)
        else
            hearts_img:draw_region(32, 0, 16, 16, hearts.surface, x, y)
        end
    end
end
if life % 2 == 1 then
    local x, y
    if life <= 20 then
        x = 5 * (life - 1)
        y = 0
    else
        x = 5 * (life - 21)
        y = 8
    end
    hearts_img:draw_region(16, 0, 16, 16, hearts.surface, x, y)
end

```

## Money System Rupee Style

I will explain on how to adjust it to different size hearts and coordinate changes. You can change the counter's location from [hud\\_config.lua](#).

The HUD uses an image file called [rupee\\_icon.png](#). I made up a simple gem for this purpose. This needs to be added to [sprites > hud](#) directory.

The counter should just show up after adding the scripts. You can adjust the gem's location in [rupees.lua](#). Look for [function rupees:on\\_draw](#).

This line changes the x-axis. The [-2](#) location.

```
rupee_icon_img:draw(dst_surface, x + -2, y)
```

This line changes the y-axis. The [11](#) location.

```
digits_text:draw(dst_surface, x, y + 11)
```

```

function rupees:on_draw(dst_surface)

    local x, y = dst_x, dst_y
    local width, height = dst_surface:get_size()
    if x < 0 then
        x = width + x
    end
    if y < 0 then
        y = height + y
    end

    rupee_icon_img:draw(dst_surface, x + -2, y)
    digits_text:draw(dst_surface, x, y + 11)
end

```

The counter shows up green and nothing happens. In order to use the counter one must set the limit. By default, it is set to 0. That means it is full. Green font means the counter is full.

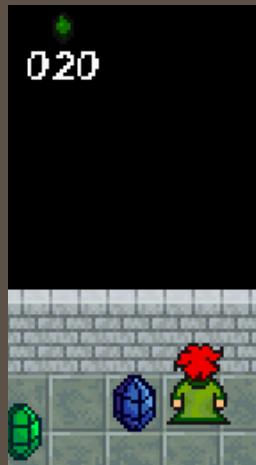


In `game_manager.lua` under function `function game_manager:start_game()` add the following line of code.

```
game:set_max_money(100)
```

```
function game_manager:start_game()

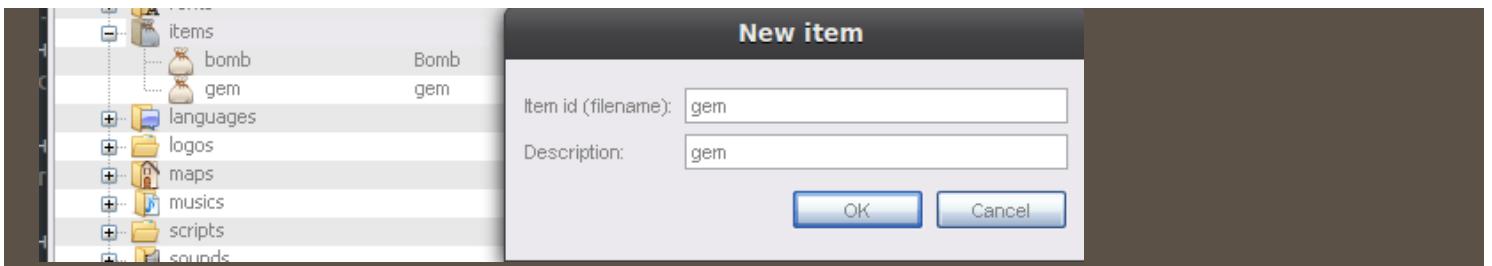
local exists = sol.game.exists("save1.dat")
local game = sol.game.load("save1.dat")
if not exists then
    -- Initialize a new savegame.
    game:set_max_life(12)
    game:set_life(game:get_max_life())
    game:set_ability("lift", 2)
    game:set_max_money(100)
    game:set_ability("sword", 1)
    game:set_starting_location("Map_4") -- Starting location.
end
```



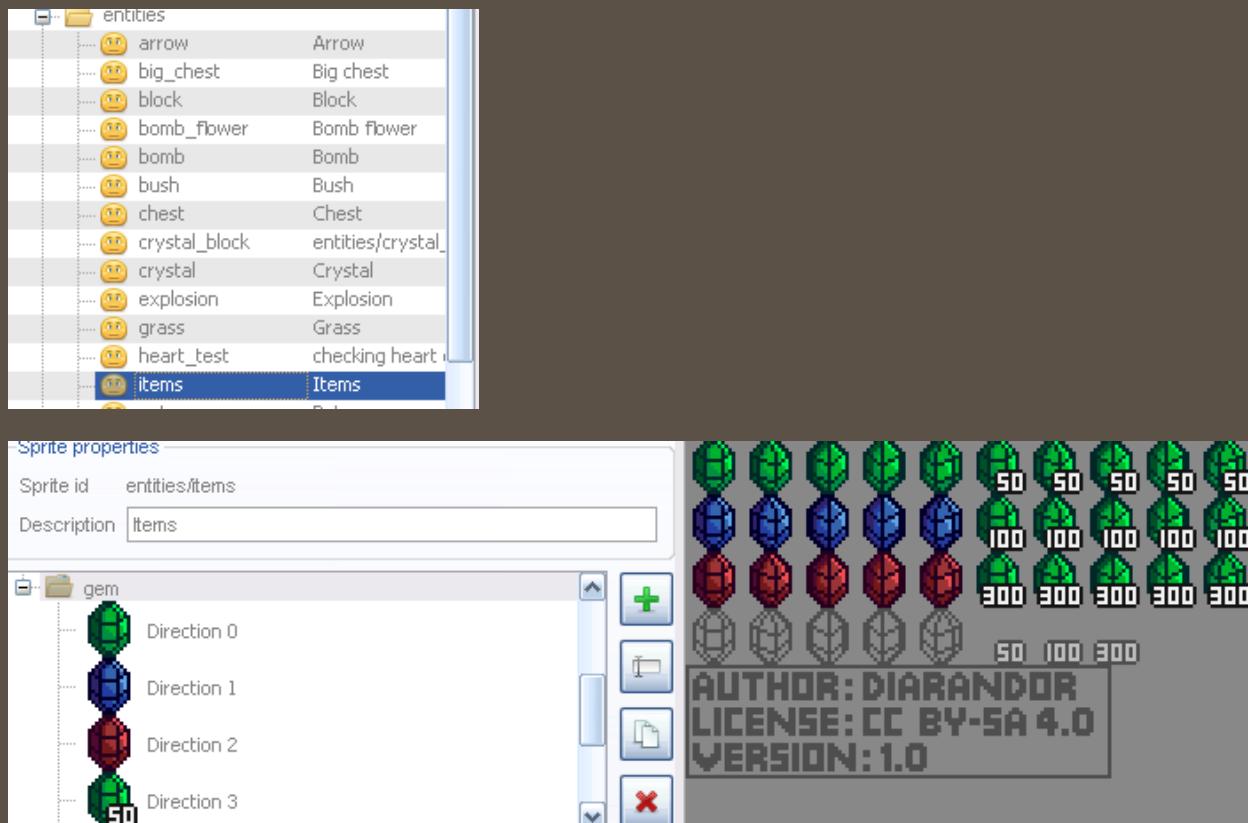
## Money (Gem) Setup

The gem already exists for you. You can thank Diarandor for that, but I will explain how to set up the gem. This script is a little cleaner.

1. Make an item script called `gem`. You probably want to delete the old one first or use the name `gem2`.



1. Go to `sprites > entities > items`. The item script name must match, the item animation game.



1. By default, a script template is made when making an item.

```
-- Lua script of item gem.
-- This script is executed only once for the whole game.

-- Feel free to modify the code below.
-- You can add more events and remove the ones you don't need.

-- See the Solarus Lua API documentation for the full specification
-- of types, events and methods:
-- http://www.solarus-games.org/doc/latest

local item = ...
local game = item:get_game()

-- Event called when the game is initialized.
function item:on_started()

    -- Initialize the properties of your item here,
    -- like whether it can be saved, whether it has an amount
    -- and whether it can be assigned.
end
```

```
-- Event called when the hero is using this item.
function item:on_using()

    -- Define here what happens when using this item
    -- and call item:set_finished() to release the hero when you have finished.
    item:set_finished()
end

-- Event called when a pickable treasure representing this item
-- is created on the map.
function item:on_pickable_created(pickable)

    -- You can set a particular movement here if you don't like the default one.
end
```

1. The function `function item:on_using()` will not be used because you buy things with money (gems), but if you plan to attack with gems, then you would use this function. That means you can delete this part.

```
-- Event called when the hero is using this item.
function item:on_using()

    -- Define here what happens when using this item
    -- and call item:set_finished() to release the hero when you have finished.
    item:set_finished()
end
```

1. The function `function item:on_pickable_created(pickable)` is used for when an item is created on a map. The movement animation can be changed. For example, have it spin differently (default one works fine in my opinion) or have the gem move slowly away from the player. That means you can delete this part of the script as well.

```
-- Event called when a pickable treasure representing this item
-- is created on the map.
function item:on_pickable_created(pickable)

    -- You can set a particular movement here if you don't like the default one.
end
```

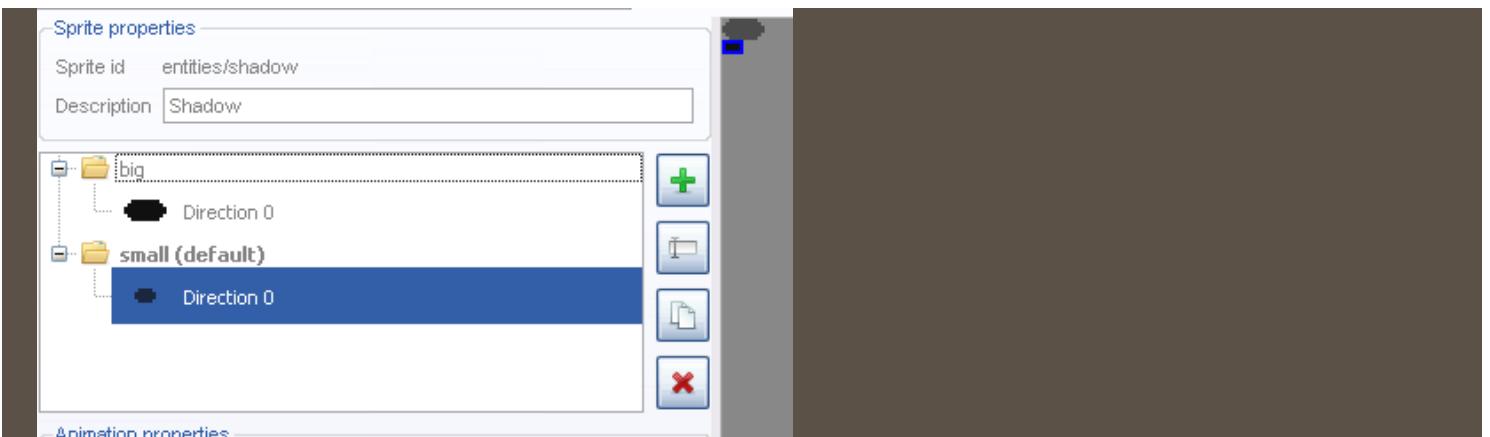
1. We are left with one function `function item:on_started()`. This function is used to set up the properties for your item. For example, shadow size, and sound when picked up. There are a lot of functions that can be used and I will cover the basic ones.

```
local item = ...
local game = item:get_game()

-- Event called when the game is initialized.
function item:on_started()

    -- Initialize the properties of your item here,
    -- like whether it can be saved, whether it has an amount
    -- and whether it can be assigned.
end
```

1. Set the shadow size with the function `self:set_shadow("animation_name")`.



```
function item:on_created()
```

```
-- Define the properties of rupees.  
self:set_shadow("small")  
end
```

1. Set whether it can disappear or not with the function `[self:set_can_disappear(true/false)]`.

```
function item:on_created()
```

```
-- Define the properties of rupees.  
self:set_can_disappear(true)  
end
```

1. Set the "ta da da daa" brandish sound when picking up the item. This will be false because it would be strange for this to happen on a simple gem. Normally, this happens when a chest is open. The function for that is `[self:set_brandish_when_picked(false/true)]`.

```
function item:on_created()
```

```
-- Define the properties of rupees.  
self:set_brandish_when_picked(false)  
end
```

1. Set the sound for when the item is picked up with function

`[self:set_sound_when_picked("name_of_sound")]`.

```
function item:on_created()
```

```
-- Define the properties of rupees.  
self:set_sound_when_picked("picked_rupee")  
end
```

The result:

```
function item:on_created()
```

```
-- Define the properties of rupees.  
self:set_shadow("small")  
self:set_can_disappear(true)  
self:set_brandish_when_picked(false)  
self:set_sound_when_picked("picked_rupee")  
end
```

## Function item:on\_obtaining

The last function will be for setting up the value for all the variants and so that money can be added to the counter. We do this with the function `function item:on_obtaining(variant, savegame_variable)`.

1. It is best to start this with a table. The table will hold the money amount for 3 variants. For example, value 2 = variant 1, value 10 = variant 3, etc.

```
function item:on_obtaining(variant, savegame_variable)
    local amounts = { 2, 6, 10 }
end
```

1. Next we will use the function `game:add_money()`. This is used to add value to the money counter. We will use the variable `amounts` and `variant` from the function `item:on_obtaining(variant, savegame_variable)` to add values for the gem `amounts[variant]`.

```
function item:on_obtaining(variant, savegame_variable)
    local amounts = { 2, 6, 10 }
    game:add_money(amounts[variant])
end
```

The result is the gem script.

```
local item = ...
local game = item:get_game()

function item:on_created()
    -- Define the properties of rupees.
    self:set_shadow("small")
    self:set_can_disappear(true)
    self:set_brandish_when_picked(false)
    self:set_sound_when_picked("picked_rupee")
end

function item:on_obtaining(variant, savegame_variable)
    local amounts = { 2, 6, 10 }
    game:add_money(amounts[variant])
end
```

## Shop Entity

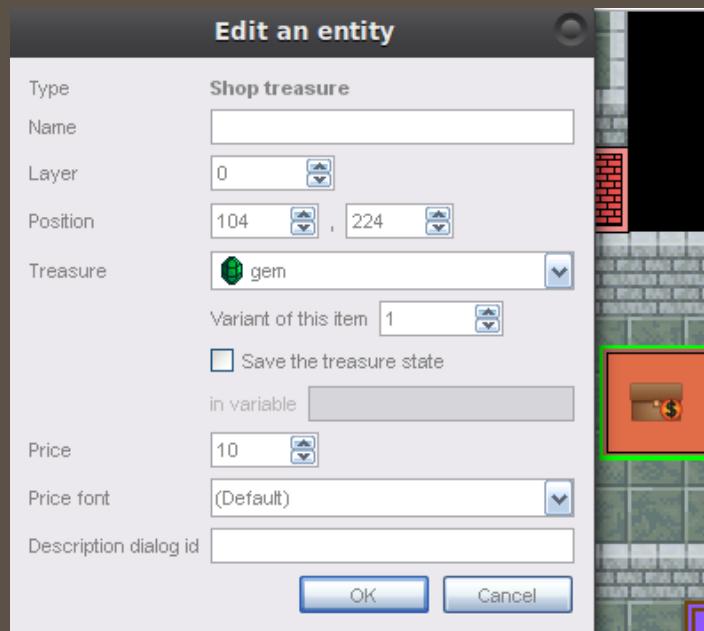
The shop entity is a quick shop setup. The shop entity shows the price, an icon, and the treasure. It uses an image file called `rupee_icon.png`.

Add Shop Entity



## Shop Entity Properties

Type	Option
Name	A name is needed for scripting reasons. You can leave it blank, but unless you plan to have it there just for show, then a name is needed.
Layer	The layer you want the entity on.
Position	The coordinates the entity is at. You can manually change them or move the entity with the cursor.
Treasure	The treasure item to choose. You can pick a variant and save state.
Price	The cost of the item.
Price font	The font for the shop.
Description dialog id	Dialog to use when buying the item.



## Shop Setup

generated by haroopad

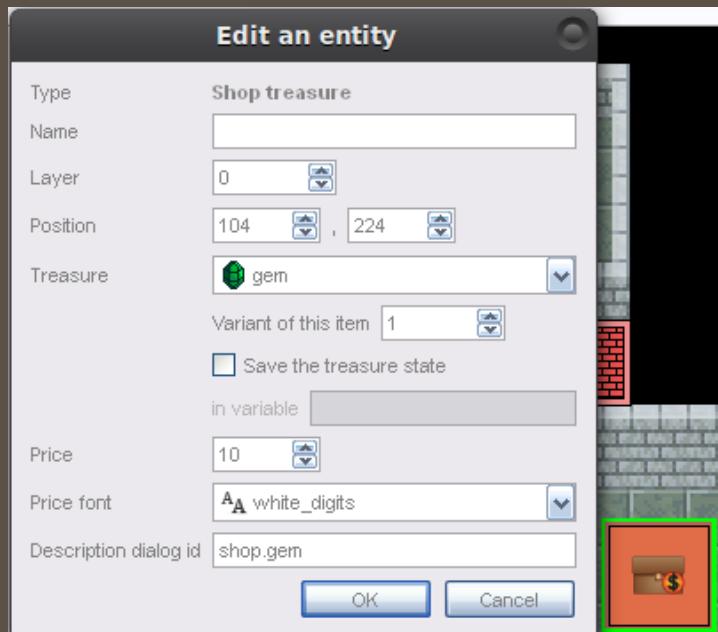
This is what the basic shop set up looks like.



## Shop Entity Dialogs

The shop requires some dialogs.

The first is in the dialog properties. The name can be whatever you want in this case.



`shop.gem`

Do you want to buy a gem?

The rest of the dialog go by default built in names. They will all be under, “\_shop.”

`_shop.amount_full`

You can't carry any more!

`_shop.not_enough_money`

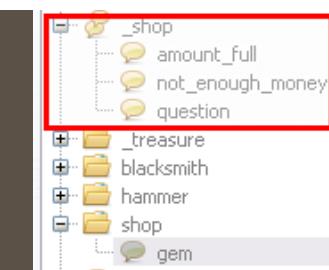
Sorry, you don't have enough Rupees!

`_shop.question`

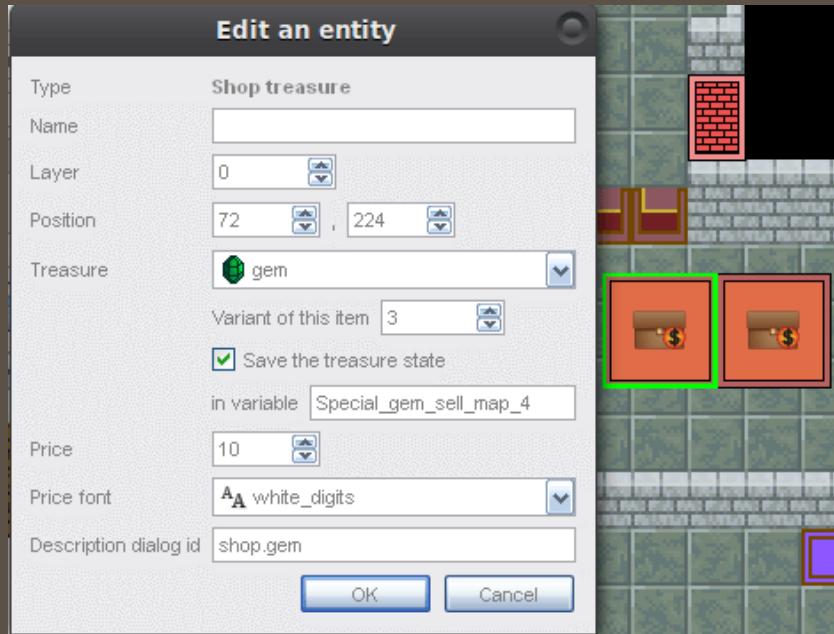
Price: \$v Rupees.

\$? Buy

\$? Don't Buy



Lastly, if you save the state of the treasure from the shop, then you will only be able to buy that item once.



## Door Entity

The purpose of this entity is for opening doors.

You can find the sample for this section in the directory [Lessons > Chapter\\_13\\_Entities > F\\_door\\_stairs\\_seperator\\_custom.zip](#).

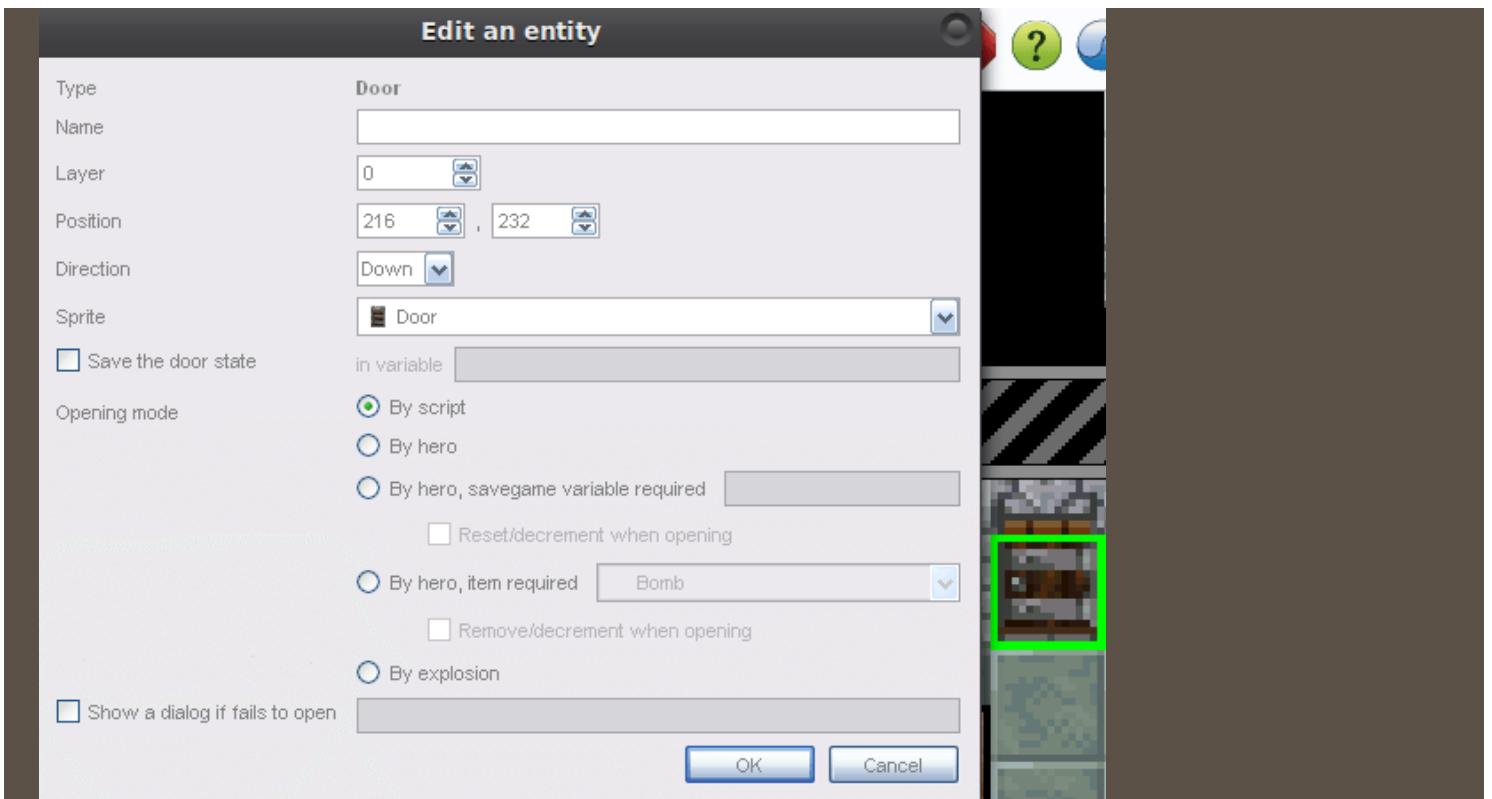
### Add Door

The door entity is by the stream entity.



## Door Properties

Type	Option
Name	A name is needed for scripting reasons. You can leave it blank, but unless you plan to have it there just for show, then a name is needed.
Layer	The layer you want the entity on.
Position	The coordinates the entity is at. You can manually change them or move the entity with the cursor.
Direction	You can pick the sprite direction of the door if that graphic exists in an animation setup. The options are up, down, left, and right.
Sprite	Pick the sprite image for the door.
Door State	If you want the door to stay open when leaving the map, then you must check the <b>save the door state</b> .
Opening mode	<p>There are bunches of opening modes for a door.</p> <p><b>By script</b> - That means a script is required for opening a door.</p> <p><b>By hero</b> - The hero opens the door.</p> <p><b>By hero, savegame variable</b> - This means the hero can open this door if the savegame variable exists.</p> <p><b>By hero, item</b> - An item is required for the hero to open the door.</p> <p><b>By explosion</b> - A bomb can open the door or some other item with an explosion ability.</p>
Dialog	If the door fails to open, then a dialog can be shown. For example, "you need a small key to open this door."

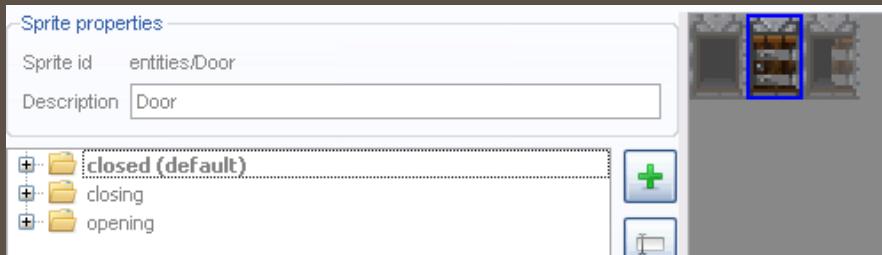


## Door Graphic Setup

For a door to work 3 default animation keywords are needed.

- closed - Door is closed. Not open.
- closing - Door is opening or closing
- opening - Door is open.

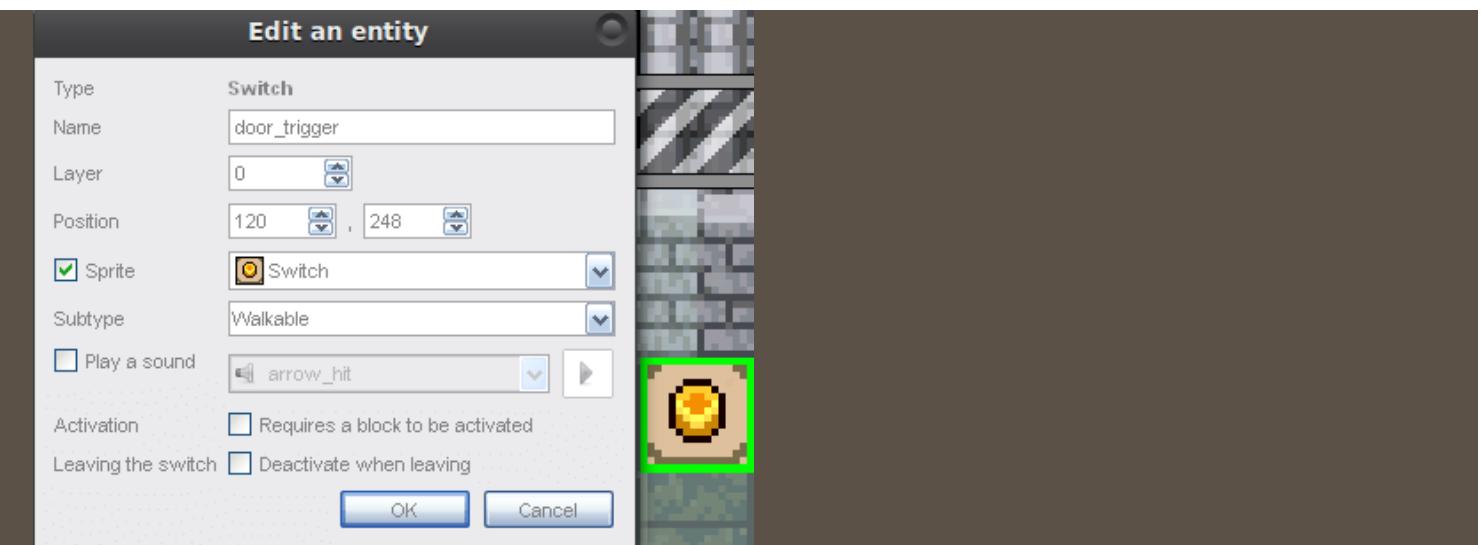
The closing time determines whether the graphic shows or not. Give the door entity a frame delay of **0 ms** for animation **closed** if you do not want the door graphic to vanish when playtesting.



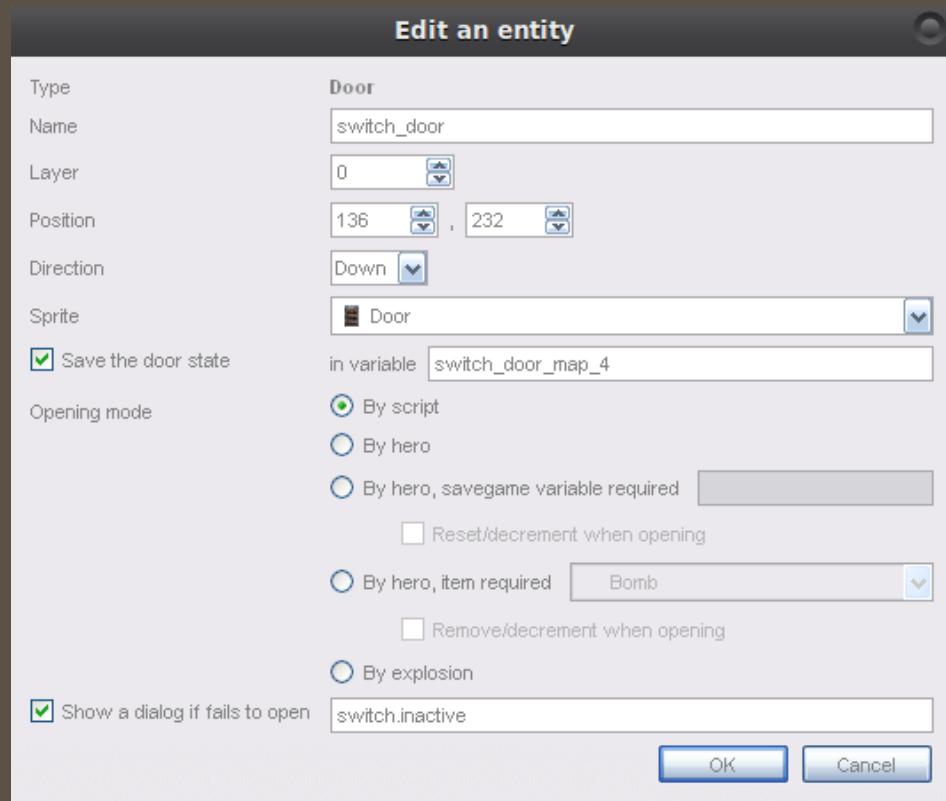
## Door Script

We are going to open two doors at the same time with a switch entity.

1. Add a switch entity. I named the switch **door\_trigger**.



1. Add a door entity. I named it `switch_door`.
2. You can save the state of the door.
3. Select the opening mode `By script`.
4. You can add a dialog as well.



1. Copy the door entity you just made and set it somewhere. Do not rename it because the number index at the end is needed to open both doors at the same time.



1. The function `map:open_doors("name_without_number_index")` will open both doors at the same time.

```

local map = ...
local game = map:get_game()

function door_trigger:onActivated()
    map:open_doors("switch_door")
    sol.audio.play_sound("door_open")
end

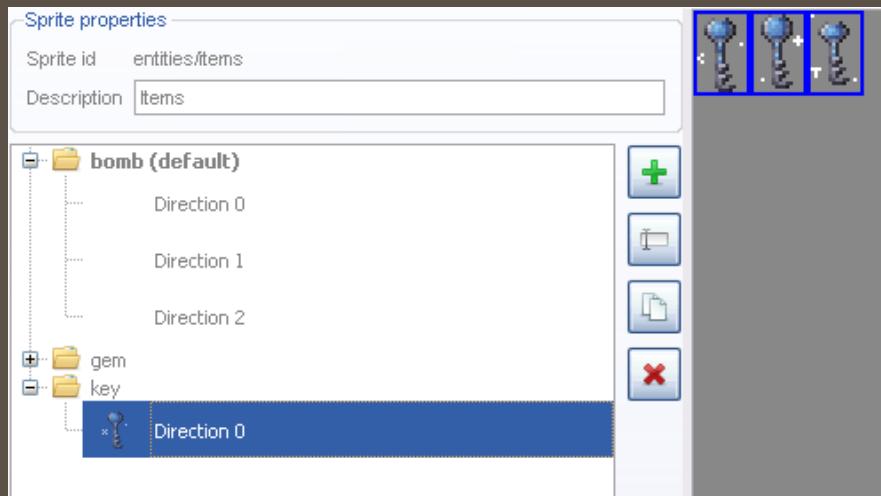
```

## Opening Door With Key

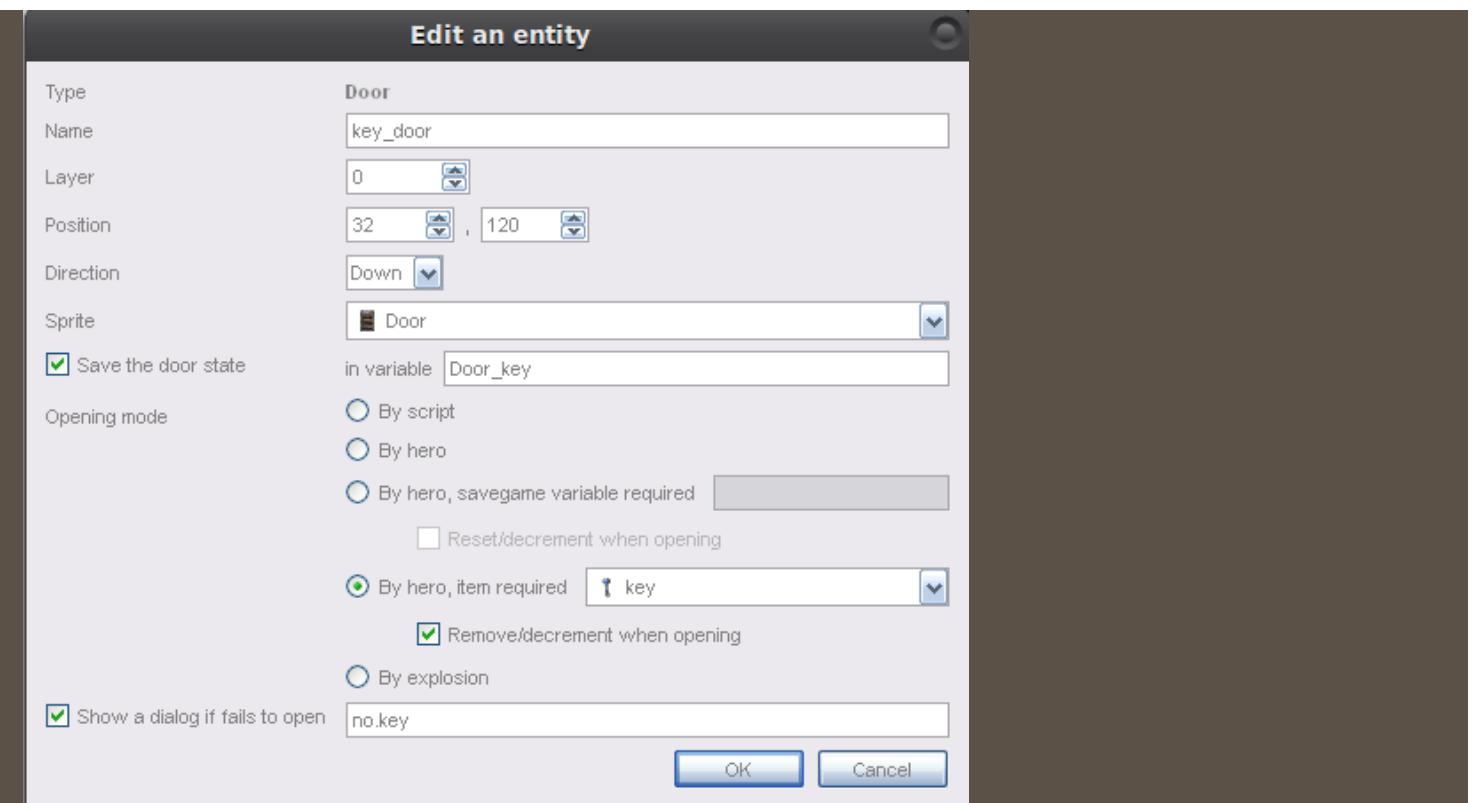
1. Create an item named **key**.



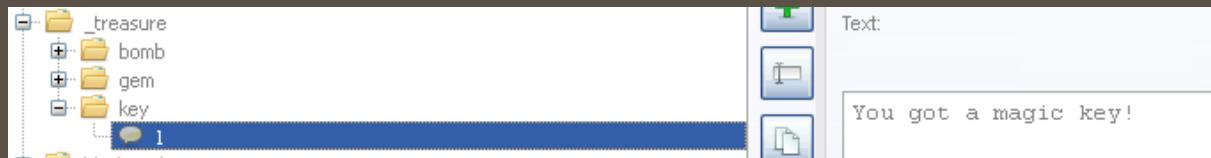
1. Make an animation for the key in **sprites > entities > items**.



1. Setup the door and require the key.



1. Make sure to create a dialog for the item `key`.



1. Saving the item `key`. The item must be saved in order for it to work. Double click on the item `key` to open its item script. On creating the key use the function `self:set_savegame_variable("name_of_item")`.

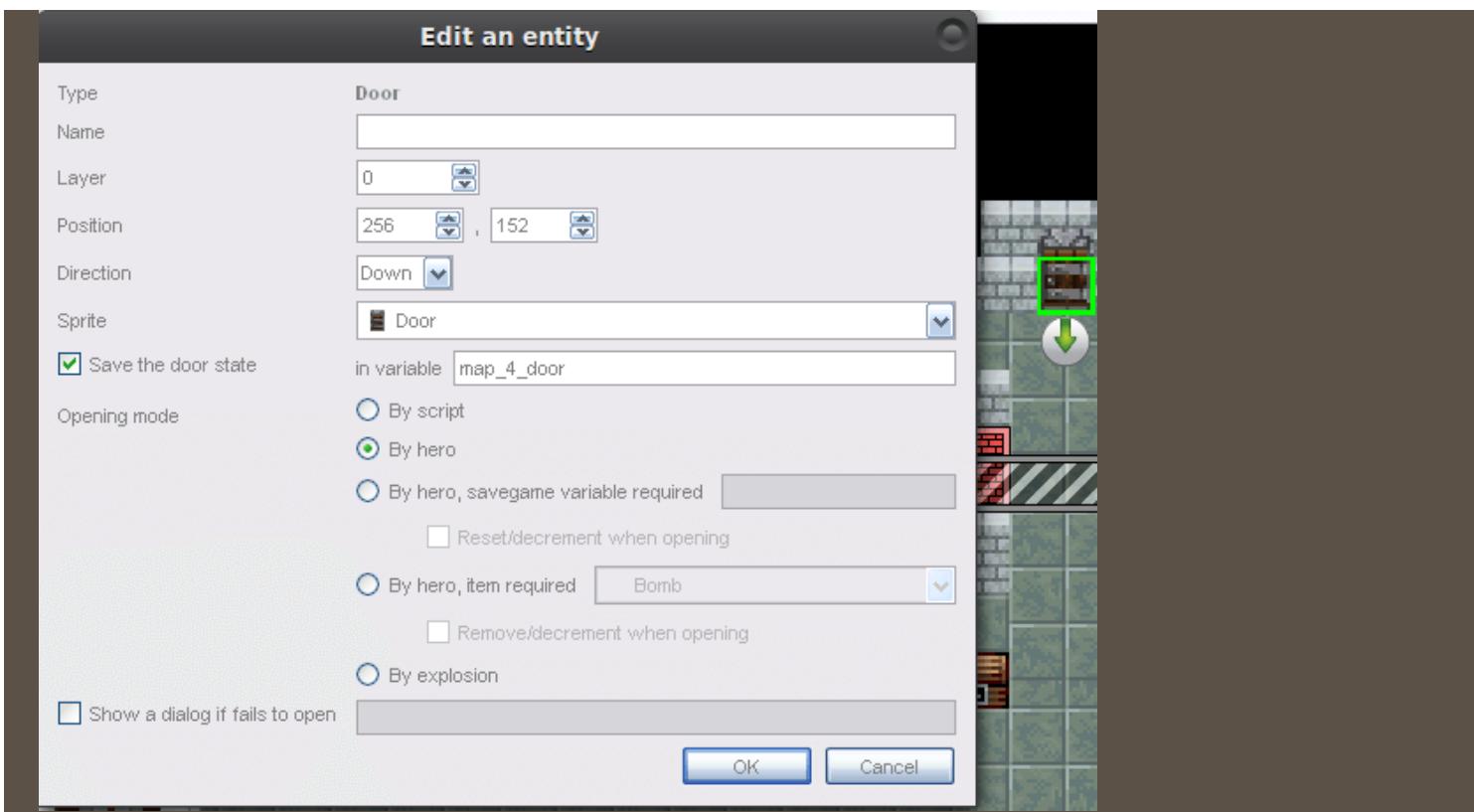
## Door Key

```
local item = ...
local game = item:get_game()

function item:on_created()
    -- Define the properties of key.
    self:set_savegame_variable("map_4_key")
end
```

## Door Opening By Hero

The hero is able to open the door.

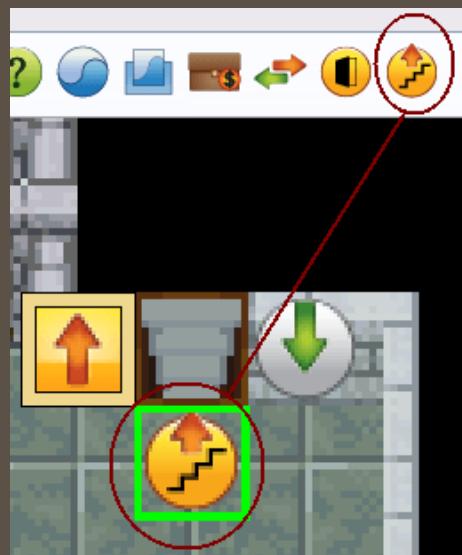


## Stairs Entity

The purpose of the stair entity is to create stair walking animation.

### Add Stairs

The stair entity is by the door entity.



### Stairs Properties

Type	Option
Name	A name is needed for scripting reasons. You can leave it blank, but unless you plan to have it there just for show, then a name is needed.

Type	Option
Layer	The layer you want the entity on.
Position	The coordinates the entity is at. You can manually change them or move the entity with the cursor.
Direction	You can pick the hero direction for when the hero goes up the stairs. The options are up, down, left, and right.
Subtype	The subtypes are: <b>Spiral staircase (going upstairs)</b> - Upward spiral stairs walking animation. <b>Spiral staircase (going downstairs)</b> - Downward spiral stairs walking animation. <b>Straight staircase (going upstairs)</b> - Straight upstairs walking animation. <b>Straight staircase (going downstairs)</b> - Straight down stairs walking animation. <b>Platform stairs (same map)</b> - Walking up platform animation. It basically walks up a layer. Trying the "Straight staircase" would make the hero go behind the layer.



## Stairs Setup

Set the destination and teletransporter onto the stair entity. The player will do a walking stair animation and teleport to the desired destination.



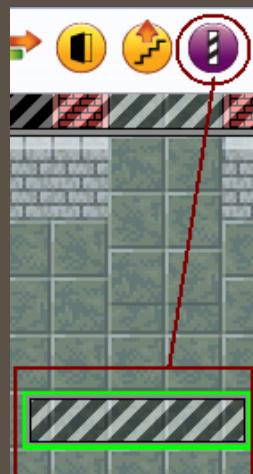
## Separator Entity

The purpose of this is block the view of the player or enemy. For example, if you have many sections in a room and do not want to see the other sections, then use the separator to block the view. The blocking has to be reasonable though. The sections should all be even or the separator will not work properly.

### Add Separator

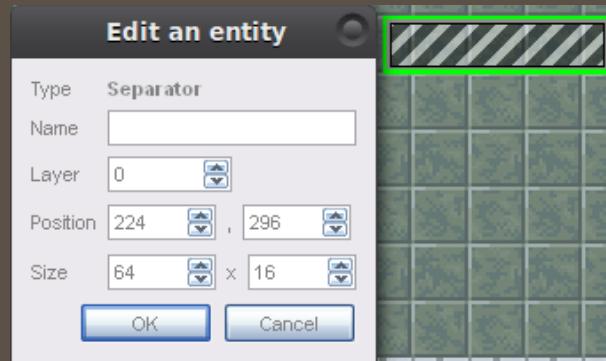
generated by haroopad

The separator entity is by the stair entity. It is the purple icon.



### Separator Properties

Type	Option
Name	A name is needed for scripting reasons. You can leave it blank, but unless you plan to have it there just for show, then a name is needed.
Layer	The layer you want the entity on.
Position	The coordinates the entity is at. You can manually change them or move the entity with the cursor.
size	One is able to expand, stretch, and/or resize the separator.



### Separator viewpoint

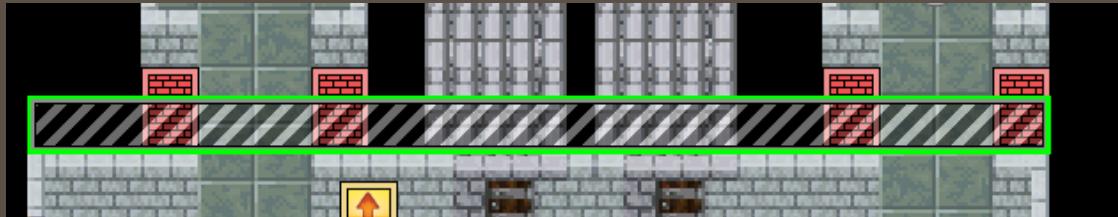
1. View without separator.

**Sample quest - Solarus 1.5.0**

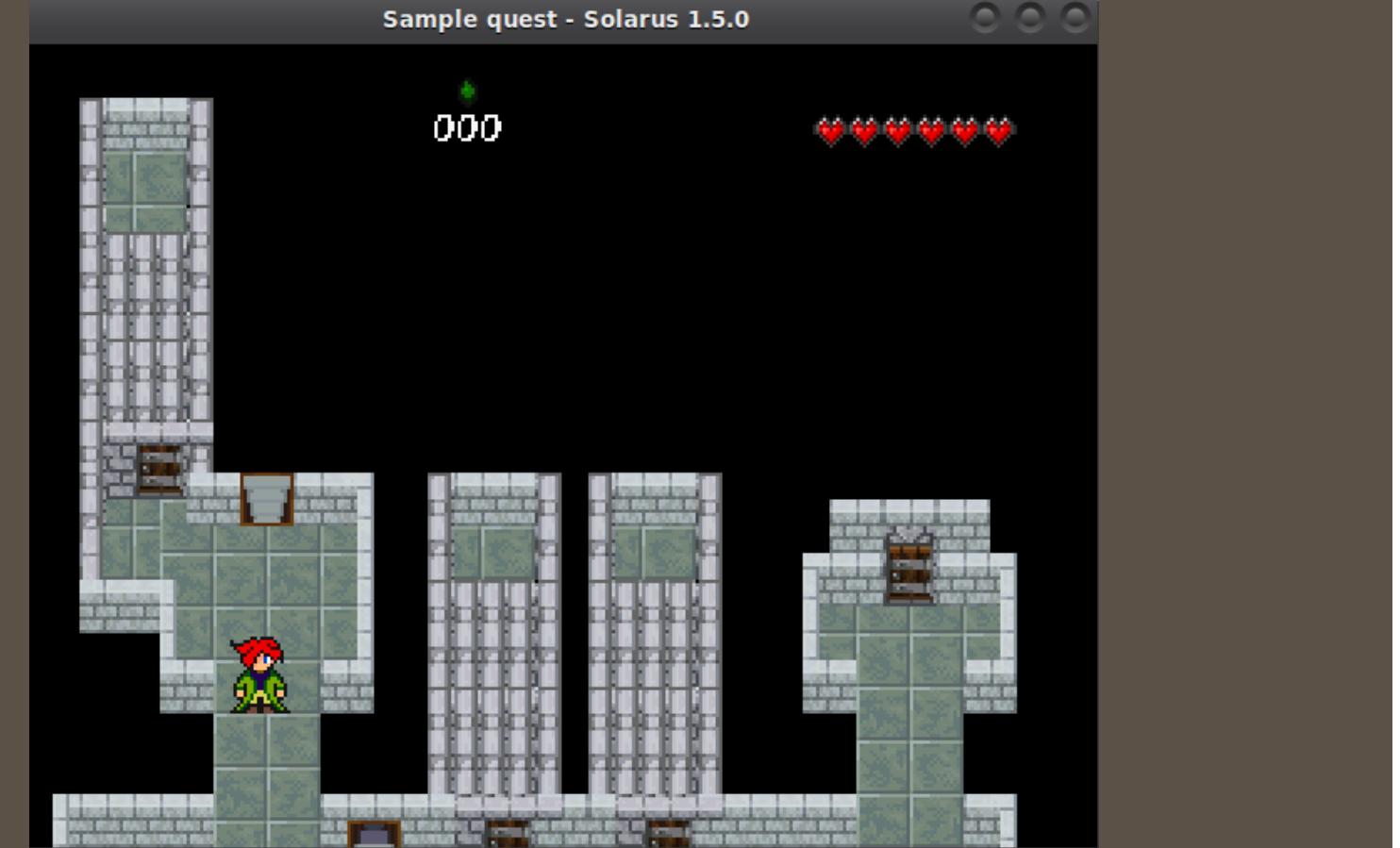
000



1. Adding separator.



1. View with separator.

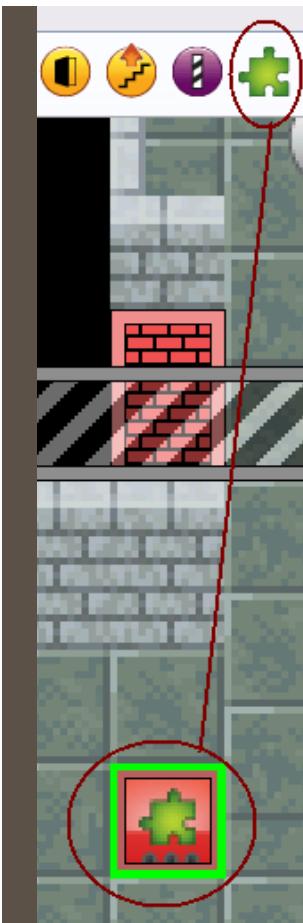
**Sample quest - Solarus 1.5.0**

## Custom Entities

The custom entity is for custom scripts.

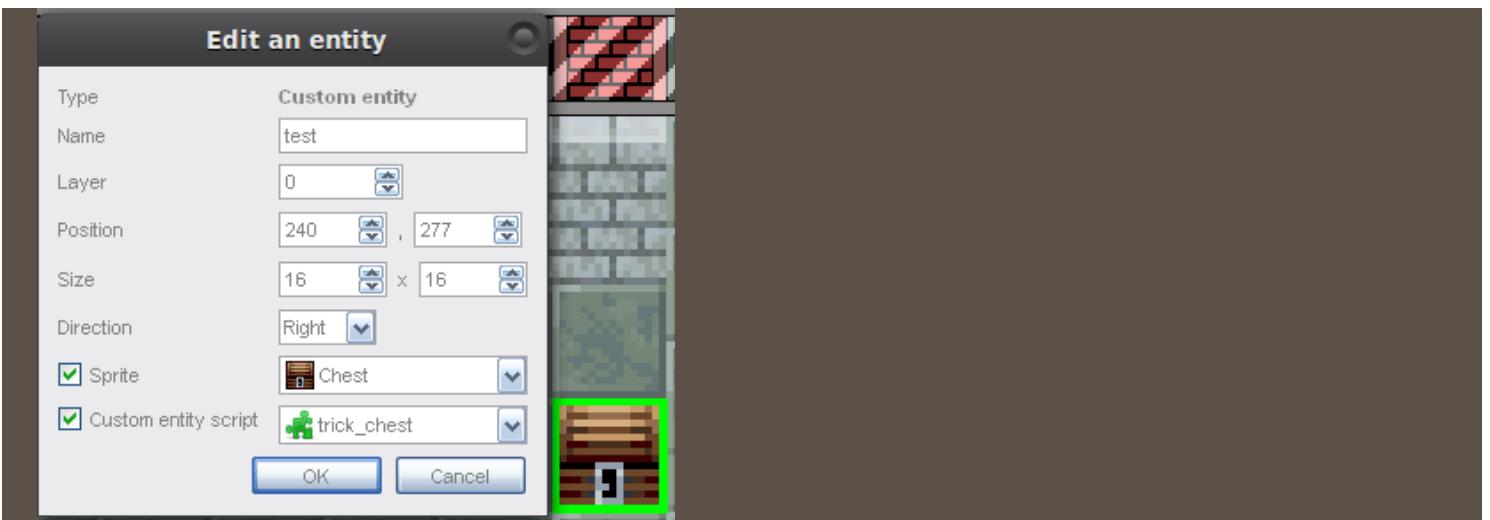
### Add Custom Entity

The custom entity is by the purple separator entity. It is the green puzzle piece icon.



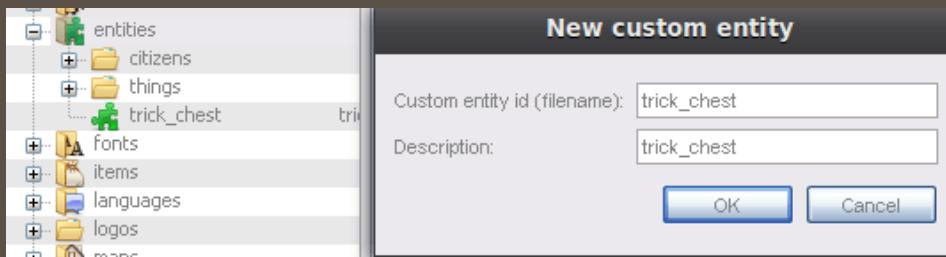
## Custom Entity Properties

Type	Option
Name	A name is needed for scripting reasons. You can leave it blank, but unless you plan to have it there just for show, then a name is needed.
Layer	The layer you want the entity on.
Position	The coordinates the entity is at. You can manually change them or move the entity with the cursor.
size	One is able to expand, stretch, and/or resize the custom entity.
Direction	The direction of the sprite if it exists in an animation. - Up - Down - Left - Right
Sprite	Pick the sprite image for the entity.
Custom entity script	Call a custom entity from the entity folder from the resource manager. You will notice the green puzzle icon. 

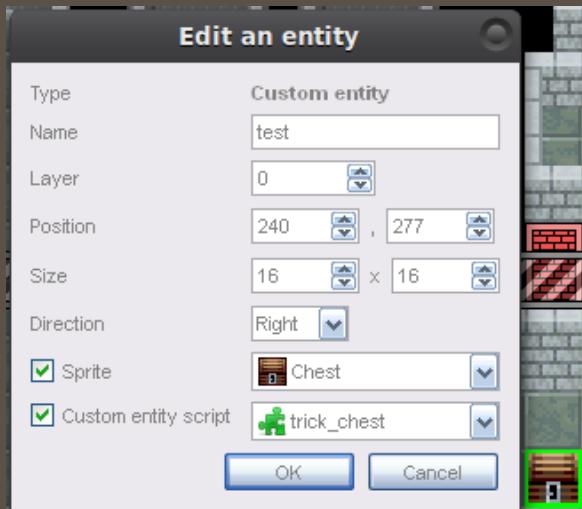


## Create Custom Entity Script

Go to entities and right click to add an entity script.



You can pick an entity script from the custom entity.



## Trick Chest Entity Scripting

We are going to script a chest that hurts the player.

1. Double click on the `trick_chest` entity.
2. Erase everything, but the `function entity:on_created()` and the entity signifier `local entity = ...`. By signifier, I mean it just tells that its entity related.
3. Setup the entity, so it works on the map. `local map = entity:get_map()`.
4. Setup the entity, so it affects the hero `local hero = map:get_hero()`.

```
local entity = ...
local map = entity:get_map()
local hero = map:get_hero()
```

1. Okay, we got the basics set up. Now we can give the entity `trick_chest` properties when it is created `function entity:on_created()`. These properties are not really needed if you set it up in the editor.
2. The only function that is truly needed is `self:set_traversable_by(true/false)`. You will not go through the entity if the is set to false. Furthermore, I demonstrated how to set the size and origin, but, you can do this in the editor.

```
function entity:on_created()
    self:set_size(16, 16)
    self:set_origin(8, 13)
    self:set_traversable_by(false)
end
```

1. Now we get to find out why we needed to `local hero = map:get_hero()`. We needed to get the hero in order to cause damage using the function `on_interaction()`. There are a few ways to cause damage.

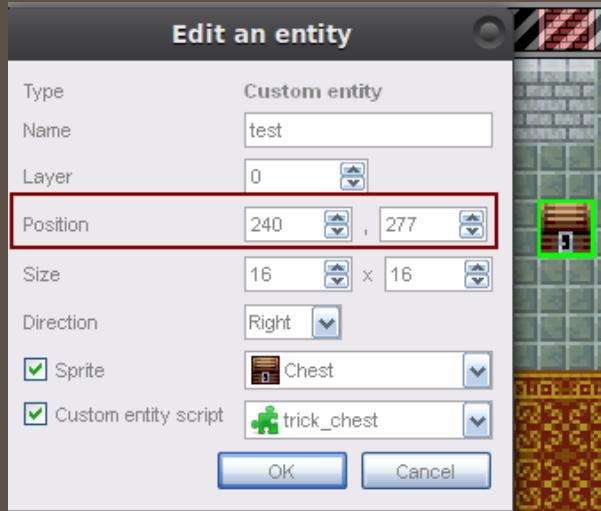
One can use game function `game:remove_life(total_life - damage_amount)`. That would require one to get the game `local game = entity:get_game()`, but there is not indication that damage was caused.

```
local entity = ...
local map = entity:get_map()
local hero = map:get_hero()
local game = entity:get_game()

function entity:on_created()
    self:set_traversable_by(false)
end

function entity:on_interaction()
    game:remove_life(12 - 6)
end
```

Another way is to use the function `hero:start_hurt(x,y,damage_amount)`. The hero will be thrown back and damage will be inflicted. The `x` and `y` are the positions of the entity.



```

local entity = ...
local map = entity:get_map()
local hero = map:get_hero()

function entity:on_created()
    self:set_traversable_by(false)
end

function entity:on_interaction()
    hero:start_hurt(240, 277, 6)
end

```

- Lastly, you can remove the entity if you want after it causes damage. I mentioned the `remove()` before.

```

local entity = ...
local map = entity:get_map()
local hero = map:get_hero()

function entity:on_created()
    self:set_size(16, 16)
    self:set_origin(8, 13)
    self:set_traversable_by(false)
end

function entity:on_interaction()
    --game:remove_life(12 - 6)
    hero:start_hurt(240, 277, 6)
    entity:remove()
end

```

## Custom Entity Switch

There is no function for `on_activated` with a custom entity, but one can easily make one with Lua using a timer.

```

local hero = map:get_hero()

--Create on_activated
sol.timer.start(500, function()
    if entity.on_activated ~= nil then
        entity:on_activated()
        return true
    end
end)

```

The easiest way to detect an entity overlapping another is with the `overlaps` function.

`entity:overlaps(other_entity, [collision_mode])`

Collision mode	Description
"overlapping"	Collision if the bounding box of both entities overlap. This is the default value.
"containing"	Collision if the bounding box of the other entity is fully inside the bounding box of this entity.

Collision mode	Description
"origin"	Collision if the origin point or the other entity is inside the bounding box of this entity.
"center"	Collision if the center point of the other entity is inside the bounding box of this entity.
"facing"	Collision if the facing position of the other entity's bounding box is touching this entity's bounding box. Bounding boxes don't necessarily overlap, but they are in contact: there is no space between them. When you consider the bounding box of an entity, which is a rectangle with four sides, the facing point is the middle point of the side the entity is oriented to. This "facing" collision test is useful when the other entity cannot traverse your custom entity. For instance, if the other entity has direction "east", there is a collision if the middle of the east side of its bounding box touches (but does not necessarily overlap) this entity's bounding box. This is typically what you need to let the hero interact with this entity when he is looking at it.
"touching"	Like "facing", but accepts all four sides of the other entity's bounding box, no matter its direction.
"sprite"	Collision if a sprite of the other entity overlaps a sprite of this entity. The collision test is pixel precise.

```
local hero = map:get_hero()

function entity:on_activated()
    if hero:overlaps(entity) then
        sol.audio.play_sound("secret")
        hero:start_hurt(240, 277, 6)
    end
end
```

Another way to detect the hero is by doing it manually with the `get_position()` function, but overlaps function is easier.

```
--if hero's position is equal to entity's position, then hero is damaged when stepping on entity.
function entity:on_activated()
local x1,y1 = hero:get_position()
print("Hero: "..x1.." , "..y1)
if (x1 > 240 and x1 < 256) and (y1 > 272 and y1 < 288) then
    sol.audio.play_sound("secret")
    hero:start_hurt(240, 277, 6)
end
end
```

## Custom Animation

You can change the animation for the custom entity. Let us say you want to do a sword swing animation. All you would have to do is the following line of code.

```
entity:get_sprite():set_animation("sword_swing")
```

You must make the `sword_swing` animation in the sprite editor.

For the hero you will not have to `get_sprite()` because the function exists for the hero and it does not have to be retrieved from the sprites methods, so it would be like the following line of code.

```
hero:set_animation("dead")
```

## Get Distance to Entity

The function `entity:get_distance(other_entity)` checks the distance in pixels to another entity. It is super useful.

### Example:

If the entities distance is less than 20 pixels, then print "Entity encounters hero!"

```
local hero = map:get_hero()

--Create on_activated
sol.timer.start(500, function()
    if map.on_activated ~= nil then
        map:on_activated()
    return true
    end
end)

function map:on_activated()
    print("-----", hero:get_distance(sprite))
    local distance_between = hero:get_distance(sprite)
    if distance_between < 20 then
        print("Entity encounters hero!")
    end
end
```

## Make Entity Turn to Entity

Then function `set_direction` and `entity:get_direction4_to(other_entity)` are needed for this to work.

The `entity:get_direction4_to(other_entity)` function gets the direction of another entity and `set_direction` makes an entity face a direction.

The following code makes the two sprites look at each other.

```
--set_direction
npc_2:get_sprite():set_direction(npc_2:get_direction4_to(npc))
npc:get_sprite():set_direction(npc:get_direction4_to(npc_2))
```

## Obstacle Detection

Let us say a sprite is using the `target` movement and gets stuck. The best way to solve this is with the function `entity:on_obstacle_reached()`. Basically, if the entity hits an obstacle, then something happens.

### Example:

In the following script the entity uses the `path_finding` movement until the distance to the hero is less than 25 pixels. At that time it uses the `target` movement again.

```
function entity:on_obstacle_reached()

    movement = sol.movement.create("path_finding")
    movement:set_target(hero)
    movement:set_speed(60)
```

generated by haroopad

```
movement:start(entity)

local distance_between = hero:get_distance(entity)
if distance_between < 25 then
    movement = sol.movement.create("target")
    movement:set_target(hero)
    movement:set_speed(60)
    movement:start(entity)
end
end
```

## On Hero State

The hero has many built in states.

States
"back to solid ground"
"boomerang"
"bow"
"carrying"
"falling"
"forced walking"
"free"
"frozen"
"grabbing"
"hookshot"
"hurt"
"jumping"
"lifting"
"plunging"
"pulling"
"pushing"
"running"
"stairs"
"stream"
"swimming"
"sword loading"
"sword spin attack"
"sword swinging"
"sword tapping"
"treasure"

**States**

“using item”
“victory”

The following script print “spin attack” when the hero does a sword spin attack.

```
function hero:on_state_changed(state)

    if state == "sword spin attack" then
        print("spin attack")
    end
end
```

**Samples:**

You can find samples of [custom switch] to [on hero state] in:

[Lessons > Chapter\\_13\\_14\\_custom\\_entity\\_game\\_over.zip](#)

## Hero Entity

This section is about “some” hero methods examples and you can check the documentation for them all. I have already demonstrated some throughout the book and will not repeat those. You can check chapter 14 section “abilities” for more and everything is in the [Solarus documentation](#).

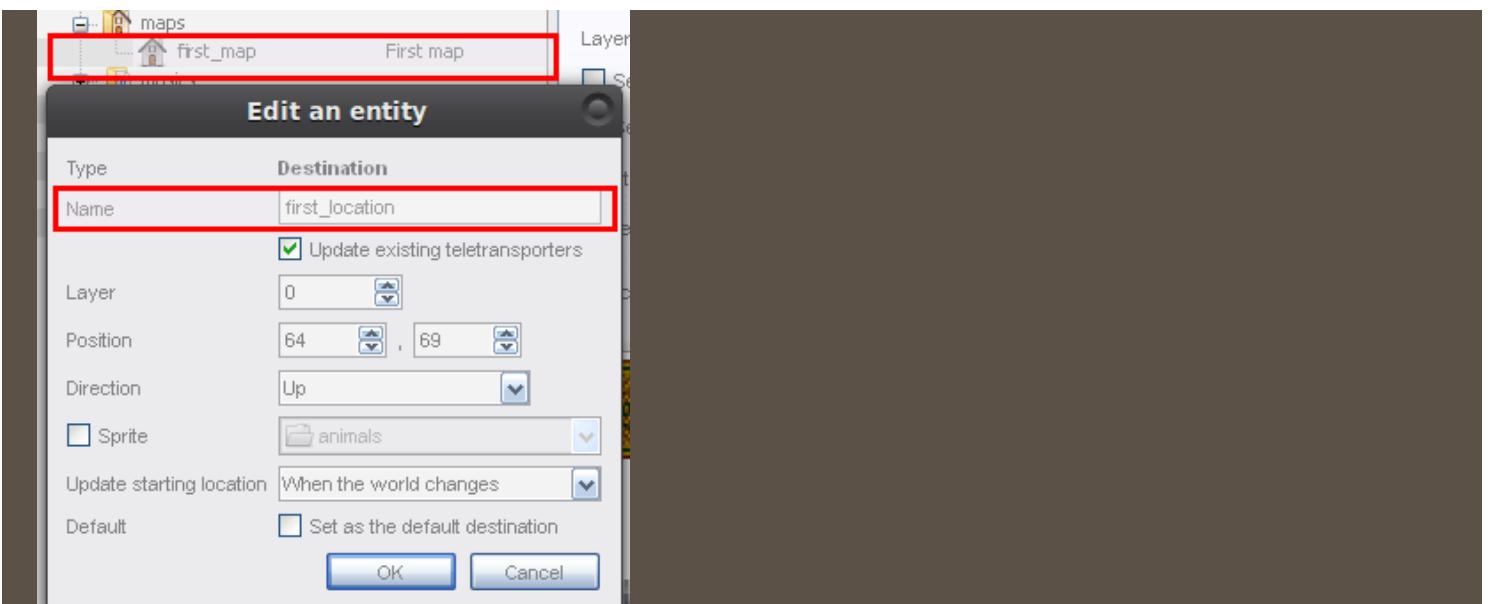
### Hero Teleport

Teleports the hero to a destination with a transition.

[hero:teleport\(map\\_id, \[destination\\_name, \[transition\\_style\]\]\)](#)

```
local map = ...
local game = map:get_game()
local hero = map:get_hero()

function map:on_started()
    hero:teleport("first_map", "first_location", "fade")
end
```



transition_style	
"immediate"	No transition effect.
"fade"	fade-out and fade-in effect.
"scrolling"	It scrolls.

## Hero Get Animation

Gets the animation the hero is currently doing.

```
hero:get_animation()
```

### Example:

```
local map = ...
local game = map:get_game()
local hero = map:get_hero()

-- Call a function every second.
sol.timer.start(1000, function()
    if hero:get_animation() == "walking" then
        print("walking")
    end
    return true -- To call the timer again (with the same delay).
end)
```

## Hero Set Animation

Sets the animation you made for the hero in the sprite editor.

```
hero:set_animation(animation, [callback])
```

### Example:

```
local map = ...
local game = map:get_game()
local hero = map:get_hero()

-- Call a function every second.
```

```

sol.timer.start(1000, function()
    if hero:get_animation() == "walking" then
        hero:set_animation("stopped")
    else
        hero:set_animation("walking")
    end
    return true -- To call the timer again (with the same delay).
end)

```

## Set Sword Sound

The sound the sword makes. By default, the sound is `sword1`.

`hero:set_sword_sound_id(sound_id)`

```

local map = ...
local game = map:get_game()
local hero = map:get_hero()

function map:on_started()
    hero:set_sword_sound_id("bomb")
end

```

## Hero Walk

The hero walk function moves the hero along a path. It can loop or ignore obstacles.

`hero:walk(path, loop, ignore_obstacles)`

### Example:

```

local map = ...
local game = map:get_game()
local hero = map:get_hero()

function map:on_started()
    hero:walk("6446600222", false, true)
end

```

Number	Direction
0	right
1	up right
2	up
3	up left
4	left
5	down left
6	down
7	down right

## Hero Jump

Makes the hero jump in a direction.

```
hero:start_jumping(direction8, distance, [ignore_obstacles])
```

### Example:

```
local map = ...
local game = map:get_game()
local hero = map:get_hero()

function map:on_started()
    hero:start_jumping(2, 100, false)
end
```

Number	Direction8
0	right
1	up right
2	up
3	up left
4	left
5	down left
6	down
7	down right

## Chapter 14: Abilities, Save Game, Quest Launcher, and Game Over

The sample for this lesson is in the following directory:

```
Lessons > Chapter_14_Abilities_Quest-launcher_savegame
```

### Abilities: Swim, Lift, Sword, Run, etc

Built-in ability levels indicate whether the hero can perform some built-in actions like attacking, swimming, and running.

Ability	Description
sword	Ability to use the sword and with which sprite.
sword_knowledge	Ability to make the super spin-attack.
tunic	Tunic (determines the sprite used for the hero's body).
shield	Protection against enemies. Determines whether the hero can avoid some kinds of attacks.
lift	Ability to lift heavy objects.
jump_over_water	Automatically jump when arriving into water without the "swim" ability.
swim	Ability to swim in deep water.

generated by haroopad

Ability	Description
run	Ability to run when pressing the action command.
detect_weak_walls	Notifies the player with a sound when a weak wall is nearby.

## Set Ability

You can set an ability with the game function `game:set_ability`.

```
game:set_ability("Ability", level)
```

### Example:

The player would gain the `sword1` ability.

```
game:set_ability("sword", 1)
```

The level at 0 turns off the ability.

```
game:set_ability("sword", 0)
```

In my opinion, I think it would be best to activate abilities in the game manager, with an NPC, and/or when obtaining an item.

### Example:

```
function item:on_obtaining()
    game:set_ability("sword", 1)
end
```

One can deactivate an ability on different maps.

### Example:

```
function map:on_started()
    game:set_ability("sword", 0)
end
```

## Get Ability

This function will get the level of the ability.

```
game:get_ability("Ability")
```

## Has Ability

This function will return true if the hero has the ability or false if the hero does not have it.

```
game:has_ability("Ability")
```

## Tunic Ability Setup

An ability needs to be set or activated before it can work.

```
game:set_ability("tunic", 1)
```

There are some default names that will need to be known for setting up the hero. The hero sprite animation is called `tunic1` by default. Many levels or variants of a tunic can be used by adding another number.



### Example:

To make a second tunic all someone has to do is create an animation file called `tunic2`.

```
game:set_ability("tunic", 2)
```

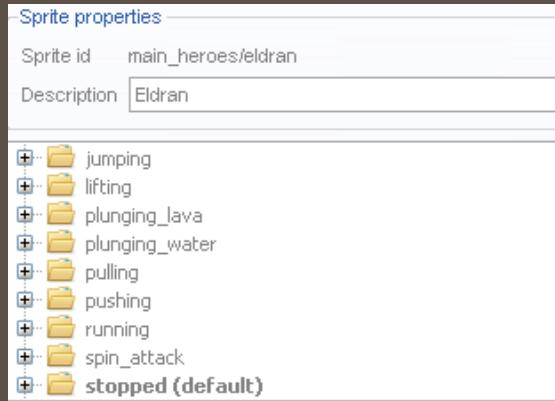
One can use a custom animation name and directory for the tunic as well.

The function `hero:set_tunic_sprite_id("directory/animation_name")` can be used to change the hero sprite.

### Example:

Diarandor uses the following line of code for his hero sprite Eldran.

```
hero:set_tunic_sprite_id("main_heroes/eldran")
```



### Tip:

By default, the "walking" animation will disable any animation. To change the default walking one could use `hero:set_tunic_sprite_id(sprite_id)` and have the animation for "walking" be "stopped." That way the hero will walk around using the "stopped" animation. Another way to do this is with the drawable sprite functions.

### Tunic Animation List:

The tunic has the following default animations and probably more. They are needed for the hero abilities.

- boomerang1
- boomerang2
- bow

- brandish
- carrying\_stopped
- carrying\_walking
- dying
- falling
- grabbing
- hookshot
- hurt
- jumping
- lifting
- plunging\_lava
- plunging\_water
- pulling
- pushing
- running
- spin\_attack
- super\_spin\_attack
- stopped
- stopped\_with\_shield
- swimming\_fast
- swimming\_slow
- swimming\_stopped
- sword
- sword\_loading\_stopped
- sword\_loading\_walking
- victory
- walking
- walking\_diagonal
- walking\_with\_shield

## Sword Ability Setup

An ability needs to be set or activated before it can work.

```
game:set_ability("sword", 1)
```

There are some default names that will need to be known for setting up the sword. The sword sprite animation is called `sword1` by default. Many levels or variants of a sword can be used by adding another number.



### Example:

To make a second sword all someone has to do is create an animation file called `sword2`. The sword will take away 2 life points at level 2.

```
game:set_ability("sword", 2)
```

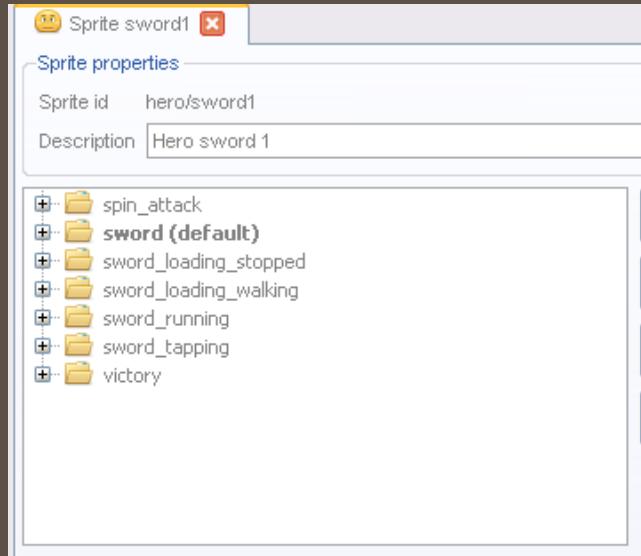
One can use a custom animation name and directory for the sword as well.

The function `hero:set_sword_sprite_id("directory/animation_name")` can be used to change the sword sprite.

### Example:

```
hero:set_sword_sprite_id("sword/red_sword")
```

The sword has the following default animations and probably more.



- spin\_attack
- super\_spin\_attack
- sword
- sword\_loading\_stopped
- sword\_loading\_walking
- sword\_running
- sword\_tapping
- victory

An animation called `sword_stars1` is needed for the charging animation during the spin attack. The `sword_stars2` and so on are needed for the other swords if they have charged attacks.



## Sword Knowledge

An ability needs to be set or activated before it can work.

```
game:set_ability("sword_knowledge", 1)
```

generated by haroopad

The `sword_knowledge` ability allows the player to do a super spin attack.

An animation called `super_spin_attack` is needed in `tunic1` and `sword1` for this to work.

## Shield Ability Setup

An ability needs to be set or activated before it can work.

```
game:set_ability("shield", 1)
```

There are some default names that will need to be known for setting up the shield. The shield sprite animation is called `shield1` by default. Many levels or variants of a shield can be used by adding another number.



### Example:

To make a second shield all someone has to do is create an animation file called `shield2`.

```
game:set_ability("shield", 2)
```

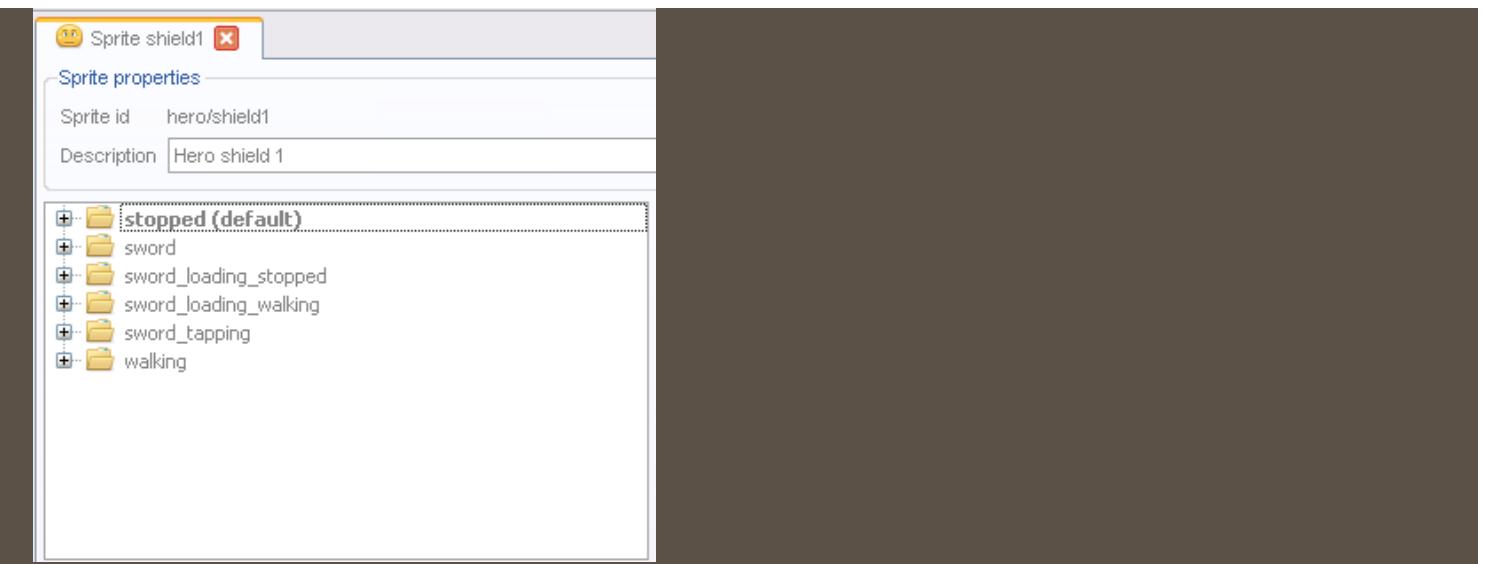
One can use a custom animation name and directory for the shield as well.

The function `hero:set_shield_sprite_id("directory/animation_name")` can be used to change the shield sprite.

### Example:

```
hero:set_shield_sprite_id("shield/red_shield")
```

The shield has the following default animations. They follow when a sword is used.



- stopped
- sword
- sword\_loading\_stopped
- sword\_loading\_walking
- sword\_tapping
- walking

## Lift Ability Setup

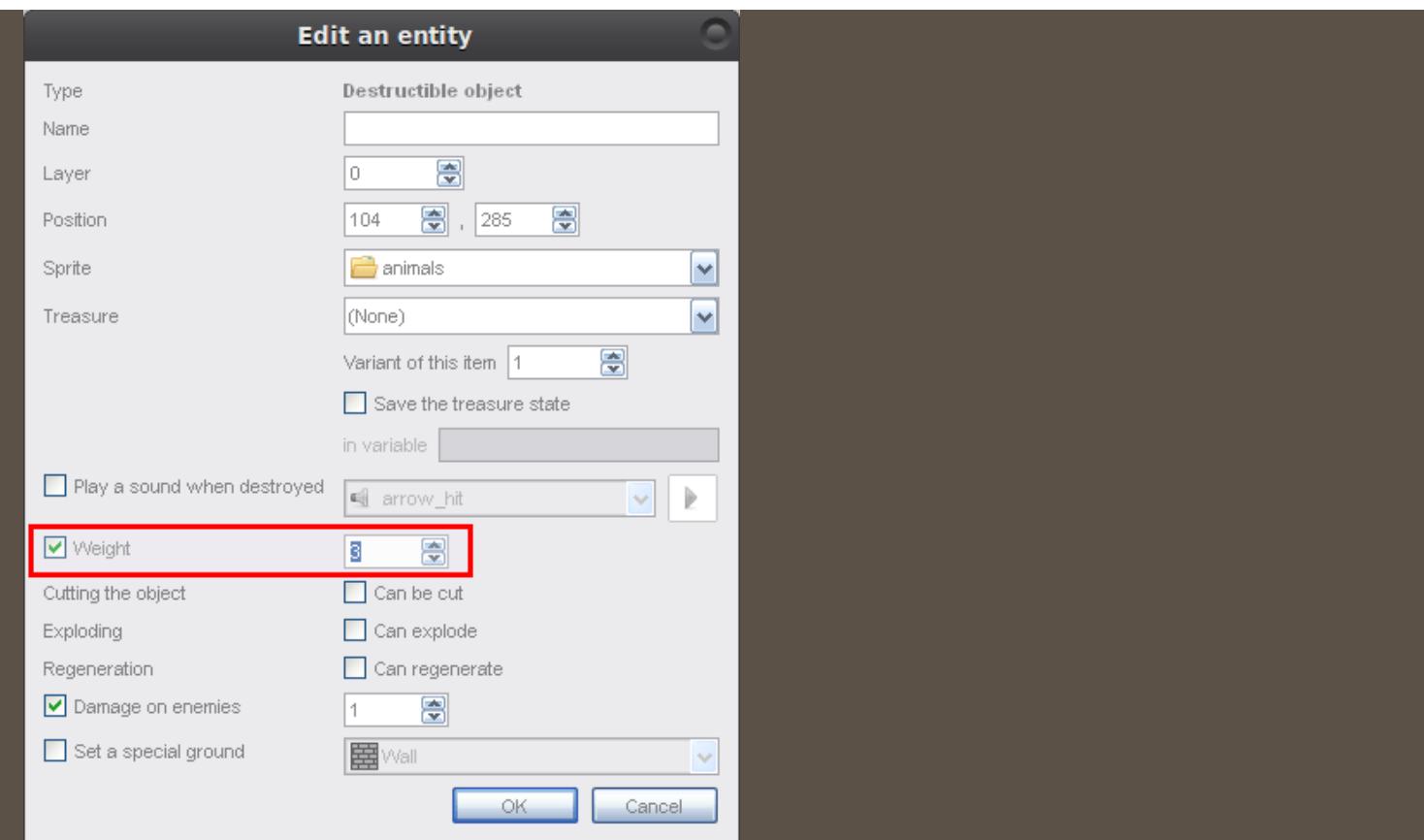
An ability needs to be set or activated before it can work.

```
game:set_ability("lift", 1)
```

The lift ability connects directly with the destructible entity weight.

The following would require lift ability level 3.

```
game:set_ability("lift", 3)
```



The `lifting`, `carrying_stopped`, and `carrying_walking` animations are required in the tunic sprite animation.

## Water Jump Ability Setup

An ability needs to be set or activated before it can work.

```
game:set_ability("jump_over_water", 1)
```

The player drowns instantly when touching deep water unless they have the swim ability. This allows the player to jump before drowning.

The `plunging_water` animation is required in the tunic sprite animation.

## Run Ability Setup

An ability needs to be set or activated before it can work.

```
game:set_ability("run", 1)
```

One must press and hold the space bar for this ability to activate.

The `running` animation is required in the tunic sprite animation.

I have encountered issues when jumping. For example, it crashes with the `jump_over_water` ability and with the jumper entity.

A solution can be a key `pressed/released` function and activate/deactivate `walking speed`. Maybe one just wants to change the hero speed anyway. The default walking speed is 88 pixels per second.

I think 150 pixels per second makes a very quick run.

```
hero:set_walking_speed(150)
```

## Swim Ability Setup

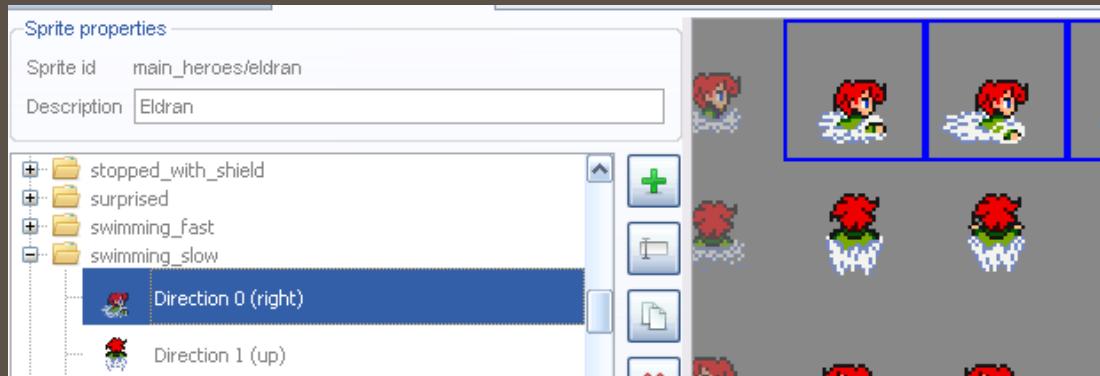
An ability needs to be set or activated before it can work.

```
game:set_ability("swim", 1)
```

The hero can swim in deep water once this ability is activated. One might want to change the hero tunic to a boat or something simple if one does not want a complex swim motion.



The `swimming_fast` and `swimming_slow` animations are required in the tunic sprite animation. You can press the action key `C` to swim faster.



## Quest Launcher

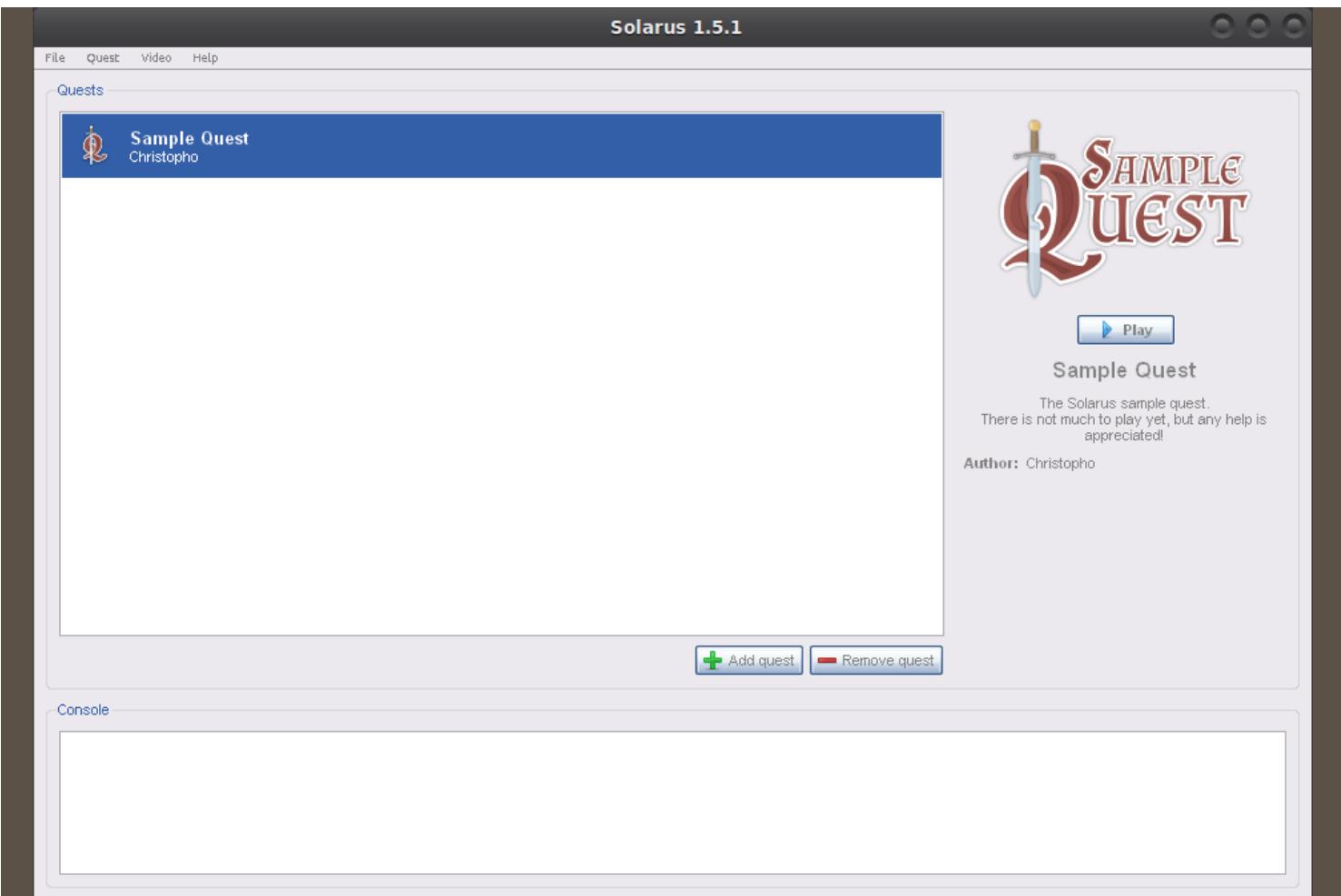
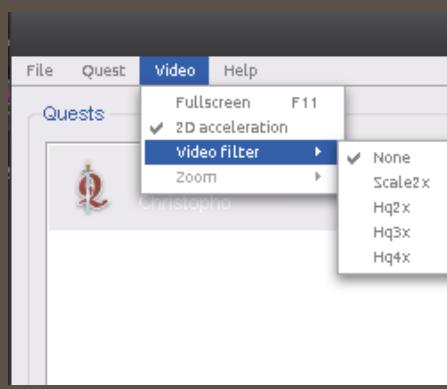
The quest launcher is a place where all Solarus games can be played from.

### Activate Quest Launcher

Activating the quest launcher.

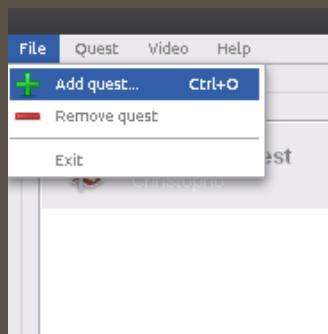


### Preview

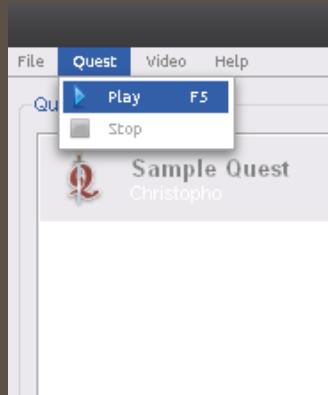
**Solarus 1.5.1****Play Game****Game - Add & Remove****Video > Fullscreen, 2D Acceleration, Window Size, & Zoom**

generated by haroopad

## Quest Launcher > File > Add, Remove, & Exit



## Game - Play & Stop



## Help > About

## About Logo & Icons:

The `logos/` directory can contain logos and icons to represent your game in the `Quest launcher`.

### Quest Logo

Christopho says, "The logo of your quest should be a PNG image of size 200x140 called `logos/logo.png`". The logo is optional."

### Quest Icons

An icon can also represent your game or quest. The icon is optional as well. Many icon sizes are allowed and every icon needs to be on a separate png. Icon sizes 16 x 16 to 1024x1024 pixels are allowed.

```
"logos/icon_16.png",
"logos/icon_24.png",
"logos/icon_32.png",
"logos/icon_48.png",
"logos/icon_64.png",
"logos/icon_128.png",
"logos/icon_256.png",
"logos/icon_512.png",
"logos/icon_1024.png".
```

The Solarus GUI will automatically choose the best size.

## Save Game

Saving the game is quite simple. The game saves at the destination entity on the maps. There are two functions that need to be known. They are `sol.game.exists(file_name)` and `sol.game.load(file_name)`. One checks if the save file exists and the other loads the save file. The best place to put the save script is in the `game_manager.lua`.

```
local exists = sol.game.exists(file_name)
local game = sol.game.load(file_name)

--check if save exists and if not, then create a save file with the following default data.
if not exists then
    --Initialize a new savegame
    --Default game data
end
```

### Example:

```
local exists = sol.game.exists("save1.dat")
local game = sol.game.load("save1.dat")
if not exists then
    --Initialize a new savegame

    game:set_max_life(12)
    game:set_life(game:get_max_life())
    game:set_ability("lift", 2)
    game:set_max_money(100)
    game:set_ability("sword", 2)
    game:set_ability("sword_knowledge", 1)
    game:set_ability("shield", 1)
    game:set_ability("swim", 1)
    game:set_ability("jump_over_water", 1)
    game:set_starting_location("Map_4", "starting_destination") -- Starting location.
end
```

The best way to save the game is by using the `d` key and a simple dialog. By now you know that by default the `d` key pauses the game. The function `game:save()` saves the game by making a save file called `save1.dat`.

```
function game:on_paused()
    game:start_dialog("pause.save_question", function(answer)
        if answer == 2 then
            game:save()
        end
        game:set_paused(false)
    end)
end
end
```

### game\_manager.lua

```
require("scripts/menus/alttp_dialog_box")
require("scripts/multi_events")
require("scripts/hud/hud")
```

```

local game_manager = {}

-- Starts the game from the given savegame file,
-- initializing it if necessary.
function game_manager:start_game()

    local exists = sol.game.exists("save1.dat")
    local game = sol.game.load("save1.dat")
    if not exists then
        --Initialize a new savegame

        game:set_max_life(12)
        game:set_life(game:get_max_life())
        game:set_ability("lift", 2)
        game:set_max_money(100)
        game:set_ability("sword", 2)
        game:set_ability("sword_knowledge", 1)
        game:set_ability("shield", 1)
        game:set_ability("swim", 1)
        game:set_ability("jump_over_water", 1)
        game:set_starting_location("Map_4", "starting_destination") -- Starting location.
    end

    game:register_event("on_started", function()

        local hero = game:get_hero()
        hero:set_tunic_sprite_id("main_heroes/eldran")
    end)
    game:start()

    function game:on_paused()
        game:start_dialog("pause.save_question", function(answer)
            if answer == 2 then
                game:save()
            end
            game:set_paused(false)
        end)
    end
end
end

return game_manager

```

You can find the save file in a folder called `.solarus` in your `users` directory. The engine creates a file called `save1.dat` by default when the game is saved. The file can be found in folder that is set in the Engine properties write directory. In my case the name is `sample_quest`.

`users/user_name/.solarus/sample_quest`

### Example:

`users/zefk/.solarus/sample_quest`

The names in `save1.dat` that have an underscore before it are automatically created by the engine.

Other save values are created by entities or with the function `game:set_value(savegame_variable_name, value)`. The value can be a string, boolean, or a number. `game:set_value()` is very useful when making custom scripts.

You can check the documentation for more information on [save data](#).

## Game Over

In this section we will be covering gameover. I will be using a quick simple script I made for this purpose. You can check out [Christopho's gameover script](#) for something more complex. He creates a hero sprite that he unpauses and sets an animation.

Here is the script we will be covering.

### game\_over.lua

```
local game_over_menu = {}

--Function to go under game:start() in save_menu.lua
function game_over_menu:start(game)

local gameover = {

    browse = 0,

    yes_hover_img = sol.surface.create("game_over/yes_hover.png"),
    no_hover_img = sol.surface.create("game_over/no_hover.png"),
    idle_bg_img = sol.surface.create("game_over/idle_bg.png"),

    yes_hover = true,
    no_hover = false,
    idle_bg = true,

    up = true,
    down = true,

    up_sound = "none",
    down_sound = "none",

}

--Set up and down sound
gameover.up_sound = "cursor"
gameover.down_sound ="cursor"

--The draw function for showing images
function game_over_menu:on_draw(screen)

--Show idle-bg-hover image when conditions are true
if gameover.idle_bg == true then
    gameover.idle_bg_img:draw(screen)
end

--Show yes-hover image when conditions are true
if gameover.yes_hover == true then
    gameover.yes_hover_img:draw(screen)
end

--Show no-hover image when conditions are true
if gameover.no_hover == true then
    gameover.no_hover_img:draw(screen)
end


```

```

end

end -- end draw function

--key function for pressing buttons
function game_over_menu:on_key_pressed(key)

--Go down
if key == "down" and gameover.up == true then
    sol.audio.play_sound(gameover.up_sound)
    if gameover/browse < 1 then
        gameover/browse = gameover/browse + 1
    end
end

--Go up
if key == "up" and gameover.down == true then
    sol.audio.play_sound(gameover.down_sound)
    if gameover/browse > 0 then
        gameover/browse = gameover/browse - 1
    end
end

--Yes hover
if gameover/browse == 0 then
    print("up")
    gameover/yes_hover = true
    gameover/no_hover = false

    local map = game:get_map()

    if key == "a" then

        game:set_starting_location(map:get_id()) -- Starting location.
        game:start()
        game:stop_game_over()
        sol.menu.stop(self, game_over_menu)
    end
end

--No hover
if gameover/browse == 1 then
    print("down")
    gameover/no_hover = true
    gameover/yes_hover = false

    if key == "a" then
        sol.main.reset()
    end
end
end -- end key pressed function
end -- end of function game_over_menu:start(game)

return game_over_menu --return menu

```

The script is called from the [game\\_manger.lua](#).

```

local game_over_menu = require("scripts/game_over.lua")

game:register_event("on_started", function()

    local hero = game:get_hero()

```

```
--Game over
print("life"..game:get_life())

function game:on_game_over_started()
    local life = game:get_life()
    if life == 0 then
        hero:set_animation("dead")
        sol.audio.play_sound("hero_dying")
        sol.menu.start(self, game_over_menu)
    end
end
end) -- end of game:register_event
```

The game parameter is passed through the `save_menu.lua` after `game:start()` just like the `game_manager.lua`.

```
game:start()

local game_manager = require("scripts/game_manager")
local game_over_menu = require("scripts/game_over.lua")

game_manager:manage(game)
game_over_menu:start(game)
```

The important functions in this script are `game:stop_game_over()` and `function game:on_game_over_started()`.

`game:stop_game_over()` stops the `function game:on_game_over_started()` function. By default, the game starts over and restores health, but `function game:on_game_over_started()` is needed to make a custom death.

I set an animation called `hero:set_animation("dead")` and the animation `dead` is only one frame. The reason for this is that the `game_over_started` function suspends the game and it will stop any complex animation before it finishes. That is why it is best to use a **sprite method** because a sprite method can be **unpaused**. That is what Christopho does in his **script**.

```
function game:on_game_over_started()
    local life = game:get_life()
    if life == 0 then
        hero:set_animation("dead")
        sol.audio.play_sound("hero_dying")
        sol.menu.start(self, game_over_menu)
    end
end
```

Okay, after starting the animation the death sound occurs and the `game_over.lua` menu is started. In the game over script you can start the game again `game:start()`.

```
--Yes hover
if gameover.browse == 0 then
    print("up")
    gameover.yes_hover = true
    gameover.no_hover = false

    local map = game:get_map()

    if key == "a" then

        game:set_starting_location(map:get_id()) -- Starting location.
        game:start()
```

```

game:stop_game_over()
sol.menu.stop(self, game_over_menu)
end
end

```

**Sample:**

You can check out the sample in:

[Lessons > Chapter\\_13\\_14\\_custom\\_entity\\_game\\_over.zip](#)

## Chapter 15: Title Screen, Save menu, Movements, Map Types, Camera, I/O

This chapter is mostly about the remaining basics that need to be covered. You can get the sample [Chapter\\_15\\_Sample.zip](#) in the [Lessons > Chapter\\_15](#) directory.

### Title Screen

Almost every game has a title screen. A place to start the game. The title screen I scripted for the section is fairly simple. The script can be found in the sample mentioned above because it is too long for the book.

**About Script:**

This is a three (3) slot load save menu script. The player can make newgames and select the save slot they wish.

**Features:**

1. Change the action keys, music, and sounds in the change area(s)
2. Load saved game
3. Makes newgames

**Install Instructions: Part 1**

1. Put the `save_menu.lua` script in the directory `scripts/` (get it from the chapter 15 sample).
2. Go to `main.lua` and add the following under the title screen `on_finished` function.

**Example**

```

-- Start the game when the Solarus logo menu is finished.
solarus_logo.on_finished = function()
    -- Show the title screen after the Solarus Logo
    local title_screen_menu = require("scripts/title_screen")
    sol.menu.start(self, title_screen_menu)

    title_screen_menu.on_finished = function()

        local save_load_menu = sol.main.load_file("scripts/save_menu.lua") (game)
        sol.menu.start(self, save_load_menu)

        game_manager:start_game()
    end
end

```

1. Put the folder "save\_menu" that contains images for the menu in the directory **sprites/**

## Install Instructions: Part 2

1. Add the following in the game manager.

### Example:

```

require("scripts/menus/alttpr_dialog_box")
require("scripts/multi_events")
require("scripts/hud/hud")
local save_load_menu = sol.main.load_file("scripts/save_menu.lua") (game)

local game_manager = {}

-- Starts the game from the given savegame file,
-- initializing it if necessary.
function game_manager:manage(game)

    --Change hero sprite ID
    game:register_event("on_started", function()

        local hero = game:get_hero()
        hero:set_tunic_sprite_id("main_heroes/eldrina/eldrina")
    end)

    --This happens when the game is paused with the default pause key "D"
    --Save the game
    function game:on_paused()
        game:start_dialog("pause.save_question", function(answer)
            if answer == 2 then
                game:save()
            end
            game:set_paused(false)
        end)
    end

    function sol.main:on_key_pressed(key)

        if key == "1" then
            sol.menu.start(self, save_load_menu)
            game:set_suspended(true)
        end
    end
end

return game_manager

```

1. I suggest making a menu with the default **D** pause key. The menu would be where the player would load and save the game. At the moment the **D** key saves the game in the sample.

## Usage Instructions:

1. Use the up and down keys
2. Change the action key to what you desire in the change area. By default, it is key **A**, reset with **S**, and in game clear with **Q**.
3. Key **L** activates the save menu in game and key **Q** can clear it.

## Breaking Down The Script:

The comments in the script cover everything in the script. There are a few functions I will go over.

The function `sol.file.remove("file_name.dat")` removes a file in the write directory. For this lesson the write directory is `.users/your_name/solarus/sample_quest/`. I used this function to remove the save file for a new game.

The function `sol.game.exists("file_name.dat")` checks if a file exists in the write directory. This is normally used to check if a save files exits and if not, then set default hero settings. For example, the hero's health, item, abilities, and etc.

The function `sol.game.load("file_name.dat")` loads a saved file or creates one if none exist. It will not save it.

The function `game:start()` runs the game. Only one game can run at a time.

I do not know if I covered this, but let us say your hero dies. He will by default use the tunic animation instead of the `tunic_sprite_id` you set. You can use the function `game:register_event("on_started", function())` after starting the game to prevent this from ever happening.

```
game:register_event("on_started", function()

    local hero = game:get_hero()
    hero:set_tunic_sprite_id("main_heroes/eldran")
end)
```

## Movements

Movement	Description	String
Straight movement:	Straight trajectory in any direction.	"straight"
Random movement:	A straight movement whose direction changes randomly from time to time.	"random"
Target movement:	Straight trajectory towards a possibly moving target.	"target"
Path movement:	Predetermined path composed of steps in the 8 main directions.	"path"
Random path movement:	Like a path movement, but with random steps.	"random_path"
Path finding movement:	Like a path movement, but calculated to reach a possibly moving target.	"path_finding"
Circle movement:	Circular trajectory around a possibly moving center.	"circle"
Jump movement:	An illusion of jump above a baseline.	"jump"
Pixel movement:	A trajectory described pixel by pixel.	"pixel"

### Create Movement

This is the basic structure for creating a movement.

```
local variable = sol.movement.create("movement_name")

variable:start(entity_name)
```

generated by haroopad

There are many movement functions and one should check the documentation for more.

## Jump Movement

An illusion of jump above a baseline.

```
function map:on_started()

--Jump
local jump = sol.movement.create("jump")

jump:set_direction8(2)
jump:set_distance(100)
jump:start(elf_2)
```

## Random Path Movement

Like a path movement, but with random steps.

```
--Random_Path
local random_path = sol.movement.create("random_path")
random_path:start(elf)
```

## Target Movement

Straight trajectory towards a possibly moving target.

The default target is the hero.

```
--Target
--Default target is hero
local target = sol.movement.create("target")
target:set_speed(32) -- set speed
target:start(elf_4)
```

If one wants to target a different entity, then one has to use the function [\[:set\\_target\(entity\\_name\)\]](#).

```
--Follow sprite "elf"
local target2 = sol.movement.create("target")

target2:set_speed(32)
target2:start(elf_5)
target2:set_target(elf)
```

## Path Finding Movement

Like a path movement, but calculated to reach a possibly moving target.

```
--Path finding (Go around obstacles)
local path = sol.movement.create("path_finding")

path:set_speed(69)
path:start(elf_6)
path:set_target(hero)
```

## Random Movement

A straight movement whose direction changes randomly from time to time.

```
--Random
local straight_random = sol.movement.create("random")

straight_random:start(elf_7)
```

## Straight Movement

Straight trajectory in any direction.

You can set the angle to the following values to get basic directions, but you can use negatives and other values. I used 3 in `straight:set_angle(3)`.

Angle	Value
East	0
North	math.pi / 2
West	math.pi
South	3 * math.pi / 2

```
--Straight
local straight = sol.movement.create("straight")
straight:set_angle(3)
straight:start(elf_3)
```

## Path Movement

Predetermined path composed of steps in the 8 main directions.

```
--Path
local path = sol.movement.create("path")

path:set_path{3,0,3,0,3,0,3,0,3,0,3,0,3,0,3,0,3}
path:set_speed(80)
path:set_loop(true)
path:set_ignore_obstacles(true)
path:start(elf_8)
```

## Pixel Movement

A trajectory described pixel by pixel.

Pixel Movement needs a transition array.

### Transition Example:

```
For 1 translation: table1 = {{244,270}}
For 2 translations: table1 = {{244,270}, {244,270}}
```

### Pixel Movement Example:

```
--Pixel movement
local pixel = sol.movement.create("pixel")

pixel:set_trajectory{{5,-5},{0,-1}}
pixel:set_loop(true)
pixel:set_ignore_obstacles(true)
pixel:start(elf_9)
```

## Circle Movement

Circular trajectory around a possibly moving center.

```
--Circle movement
local circle = sol.movement.create("circle")
circle:set_center(test)
circle:set_loop_delay(1000)
circle:set_angle_speed(100)
circle:set_duration(1000000)
circle:set_max_rotations(100)
circle:set_initial_angle(100)
circle:set_clockwise(true)
circle:set_radius(22)
circle:set_radius_speed(1)
circle:set_ignore_obstacles(false)
circle:start(elf_10)
```

## Walk through Entity

Instead of bumping into entities and getting trapped. I think it is best the walk through them. You can use the function `entity:set_traversable(true/false)` to walk through NPC, etc.

```
elf_4:set_traversable(true)
elf_6:set_traversable(true)
```

## Moving Image

Moving an image is almost the same as moving an entity. Let us use a 320x240 Solarus logo.

You can find everything in the sample in the following directory.

[Lessons > Chapter\\_15 > Chapter\\_15\\_Movements\\_on\\_image.zip](#)



### Tip:

The Solarus screen zooms to 640x480, so the logo will enlarge.

## Make Menu

An easy way to set this up is with a menu.

```
local solarus_logo_menu = {}

return solarus_logo_menu
```

## Setup Image & Make Movement:

Next we need to load/create the image and movement. We apply a movement to an image by starting the movement on the created surface variable.

```
movement:start(logo_img)

local solarus_logo_menu = {}

local logo_img = sol.surface.create("menus/solarus_logo.png")

function solarus_logo_menu:on_started()
    local movement = sol.movement.create("straight")
    movement:start(logo_img)
end

function solarus_logo_menu:on_draw(screen)
    logo_img:draw(screen)
end

return solarus_logo_menu
```

## Movement properties:

We want:

- The image to move at a decent speed.

```
movement:set_speed(128)
```

- The image angle to go south. South is  $3 * \text{math.pi} / 2$ .

4 Directions	Description
East	0
North	$\text{math.pi} / 2$
West	$\text{math.pi}$
South	$3 * \text{math.pi} / 2$

```
movement:set_angle(3 * math.pi / 2)
```

- The image to not run off the screen, so we set the max distance.

```
movement:set_max_distance(240)
```

- To draw the image off screen because it is an easy way to calculate max distance. We change the image's y coordinates to -240 because we set the max distance to go down by 240. The image is 320x240.

```
logo_img:draw(screen, 0, -240)
```

```
local solarus_logo_menu = {}

local logo_img = sol.surface.create("menus/solarus_logo.png")
```

```

function solarus_logo_menu:on_started()

    local movement = sol.movement.create("straight")
    movement:set_speed(128)
    movement:set_angle(3 * math.pi / 2)
    movement:set_max_distance(240)
    movement:start(logo_img)
end

function solarus_logo_menu:on_draw(screen)
    logo_img:draw(screen, 0, -240)
end

return solarus_logo_menu

```

## Stop Menu On Key Press

- Setup a key press function with the key being `space` and stop the menu inside.
- Return true to activate function or you could change the order of your function.
- Return false to not activate the key press function for other keys, but not really needed.
- Return false and true are not really needed, but it is good practice programming wise.

```

function solarus_logo_menu:on_key_pressed(key)
    if key == "space" then
        sol.menu.stop(solarus_logo_menu)
        return true
    end
    return false
end

```

Now the menu will move down and pressing the `space` key will stop it.

```

local solarus_logo_menu = {}

local logo_img = sol.surface.create("menus/solarus_logo.png")

function solarus_logo_menu:on_started()

    local movement = sol.movement.create("straight")
    movement:set_speed(128)
    movement:set_angle(3 * math.pi / 2)
    movement:set_max_distance(240)
    movement:start(logo_img)
end

function solarus_logo_menu:on_draw(screen)
    logo_img:draw(screen, 0, -240)
end

function solarus_logo_menu:on_key_pressed(key)
    if key == "space" then
        sol.menu.stop(solarus_logo_menu)
    end
end

return solarus_logo_menu

```

## Callback Instead of Key Press:

One can remove the key press function and use a callback to stop the menu if you want the game to just start after the logo finishes.

```
movement:start(logo_img, callback())
```

```
movement:start(logo_img, function()
    sol.menu.stop(solarus_logo_menu)
end)
```

```
local solarus_logo_menu = {}

local logo_img = sol.surface.create("menus/solarus_logo.png")

function solarus_logo_menu:on_started()

    local movement = sol.movement.create("straight")
    movement:set_speed(128)
    movement:set_angle(3 * math.pi / 2)
    movement:set_max_distance(240)
    movement:start(logo_img, function()
        sol.menu.stop(solarus_logo_menu)
    end)
end

function solarus_logo_menu:on_draw(screen)
    logo_img:draw(screen, 0, -240)
end

return solarus_logo_menu
```

## Map Types

I have covered a few map types already, but here are some more useful ones that I did not cover.

```
--Name of the map
map:get_id()

--Name of the world
map:get_world()

--Floor number
map:get_floor()

--Coordinates in the world
map:get_location()

--Name of tileset
map:get_tileset()

--Name of music playing
map:get_music()
```

### Example:

This example comes directory from the sample. Press key 5 and key 6 to try it out.

```

local example = {}
local test = {}
local map_functions = false

--Name of the map
example[1] = "Map: "..map:get_id()

--Name of the world
example[2] = "World: "..map:get_world()

--Floor number
example[3] = "Floor: "..map:get_floor()

--coordinates in the world
example[4] = "Location: "..map:get_location()

--Name of tileset
example[5] = "Tiset: "..map:get_tileset()

--Name of music playing
example[6] = "Music playing: "..map:get_music()

--Display text
for rep = 1,6 do
--http://www.solarus-games.org/doc/latest/lua_api_text_surface.html
    test[rep] = sol.text_surface.create({ -- name a local variable something and assign it to the
        font = "minecraftia", -- font name
        text = example[rep], -- text you want to show
        font_size = 12, -- font size obviously
        rendering_mode = "antialiasing", -- "solid" (faster) and default
        color = {240,248,255}, -- color must be in a table RGB (http://www.rapidtables.com/web/color/convert.html)
    })
end

--Draw text to screen
function sol.main:on_draw(screen)

    if map_functions == true then
        test[1]:draw(screen,10, 15)
        test[2]:draw(screen,10,30)
        test[3]:draw(screen,10,45)
        test[4]:draw(screen,10,60)
        test[5]:draw(screen,10,75)
        test[6]:draw(screen,10,90)
    end
end --end of draw function

--Display map types using key 5 and key 6.
function map:on_key_pressed(key)

    if key == "5" then
        map_functions = true
    end

    if key == "6" then
        map_functions = false
    end
end

```

You can also get the music you set for the map. This is useful for when playing a different music and wanting to go back to the default music for the map. Maybe you do not want to write a long directory generated by haroopad

name.

```
sol.audio.play_music(map:get_music())
```

## Making a Map Entity

It is quite easy to make map entities with script. This can be useful for when spawning projectiles.

### Example:

The following script is placed in a custom entity. If an entity named `enemy` overlaps it, then a custom entity arrow will spawn or appear.

```
local entity = ...
local game = entity:get_game()
local map = entity:get_map()
local enemy = map:get_entity("enemy")

--Create on_activate
sol.timer.start(500, function()
    if entity.on_activate ~= nil then
        entity:on_activate()
    return true
    end
end)

function entity:on_activate()
    local x,y = entity:get_position()

    if entity:overlaps(enemy) then
        print("overlap")
        map:create_custom_entity({
            name = "arrow",
            direction = 2,
            layer = 0,
            width = 16,
            height = 16,
            x = x,
            y = y,
            sprite = "entities/arrow",
        })
    end
end
```

## Camera Entity

The camera is an entity like a NPC. You can check the Documentation for the functions for all entity types. You need to get the camera before you can use it.

```
local camera = map:get_camera()
```

### Camera Size

The camera size is what you can see on the screen. A size of (96,96) would be quite small compared to the default size of (320,240). The camera size function goes by 8 (8,16,24,32,40,48,56,64,72,80,88,96)

```
camera:set_size(96, 96)
```

You can set your screen back to normal with:

generated by haroopad

```
--That is the default quest screen size.
camera:set_size(320, 240)
```

## Camera Tracking

Changing the screen size can help with camera tracking when it comes to sprites.

The function `camera:start_tracking(entity_name)` is used for camera tracking. By default, the camera tracks where the hero goes.

```
camera:set_size(96, 96)
camera:start_tracking(elf_6)
```

## Camera Position

Normally, the camera position is centered at `(0,0)`. You can change the position of the screen with the function `camera:set_position_on_screen(x,y)`.

### Example:

```
camera:set_position_on_screen(120,60)
```

You can change it back to normal like this:

```
camera:set_position_on_screen(0,0)
```

## Camera & Movements

You can apply movements to the camera. The camera entity name is `map:get_camera()`.

```
local path = sol.movement.create("path")

path:set_path{2,0,2,0,3,0,3,0,3,0,3,0,3,0,3,0,3}
path:set_speed(80)
path:set_loop(false)
path:set_ignore_obstacles(true)

path:start(map:get_camera())
```

## Camera Example Script

Press the keys `1`, `2`, `3`, and `4` to test the camera. Key `2` sets everything back to default.

```
local map = ...
local game = map:get_game()
local hero = map:get_hero()
local camera = map:get_camera()

function map:on_key_pressed(key)

    if key == "1" then
        --Camera size goes by 8
        camera:set_size(96, 96)
        camera:start_tracking(elf_6)
        camera:set_position_on_screen(0,0)
    end

    if key == "2" then
        camera:set_size(320, 240)
    end
end
```

```

camera:set_position_on_screen(0,0)
camera:start_tracking(hero)
end

if key == "3" then
    camera:set_size(320, 240)
    --Camera default position is 0,0
    camera:set_position_on_screen(120,60)
    camera:start_tracking(hero)
end

local path = sol.movement.create("path")

path:set_path{2,0,2,0,3,0,3,0,3,0,3,0,3,0,3}
path:set_speed(80)
path:set_loop(false)
path:set_ignore_obstacles(true)

if key == "4" then
    --Use movements on camera
    path:start(map:get_camera())
    camera:set_position_on_screen(0,0)
end
end

```

## I/O - Input/Output

The I/O is writing and reading text from a file. The easiest way to do this is to just use the engine's functions `game:get_value` and `game:set_value`.

```
game:set_value("variable_name", value/string)
```

### Example:

```
--It might be easier to just store your values and strings in a save file.
--value
game:set_value("coordinate_z", 50)
print("Coordinate Z is: "..game:get_value("coordinate_z"))

--String
game:set_value("coordinate_question", "what")
print("Coordinate Question is: "..game:get_value("coordinate_question"))
```

However, you can write to files in other ways. You can use the function:

```
sol.file.open("file_name", "mode")
```

Modes:	Description:
"r"	read mode (the default)
"w"	write mode
"a"	append mode
"r+"	update mode, all previous data is preserved
"w+"	update mode, all previous data is erased
"a+"	append update mode, previous data is preserved writing is only allowed at the end of file.

generated by haroopad

## Opening, writing, and closing file

Writing by default creates the file. You must always remember to close a file with the function `file_variable_name:close()` or you risk memory leaks. You use the mode `"w"` to write and the function `file_variable_name:write("text to show or", variable_value_to_show)`.

```
local file_variable_name = sol.file.open("example.txt", "w")
file_variable_name:write("blah")
file_variable_name:close()
```

### Example:

```
local coordinate_x = 50
local coordinate_y = 40
local file_write = sol.file.open("example.txt", "w")
file_write:write("x:", coordinate_x, "y:", coordinate_y)
file_write:close()
```

## Reading, searching, and outputting

You can use the mode `"r"` to read a file and the function `string.find(file_variable_name, "string_to_look_for")` to check if the string exists in the file.

```
local file_read = sol.file.open("example.txt", "r")
local line = file_read:read()
print(line)

if string.find(line, "0") then
    print("zero")
end
file_read:close()
```

## Making New Lines

You can use sequences when writing to a file.

Sequence	Description
\a	bell
\b	back space
\f	form feed
\n	newline
\r	carriage return
\t	horizontal tab
\v	vertical tab
\	backslash
\"	double quote
\'	single quote
[	left square bracket
]	right square bracket

You can use formats for the function `file_variable_name:read("Format")`. I use the format `"*a"` in the example below to read all the text in the file.

Formats:	Description
<code>"*n"</code>	reads a number; this is the only format that returns a number instead of a string.
<code>"*a"</code>	reads the whole file, starting at the current position. On end of file, it returns the empty string.
<code>"*l"</code>	reads the next line (skipping the end of line), returning nil on end of file. This is the default format.

```
local file_make_file2 = sol.file.open("file2.txt", "w")
file_make_file2:write("The apple\nThe orange.\nThe limon.\nThe pear.\n")
file_make_file2:close()

local file_read_all = sol.file.open("file2.txt", "r")
local line2 = file_read_all:read("*a")
print(line2)
file_read_all:close()
```

## File Seek: Grab a Variable

You can use `file:seek` to grab a variable or string, but this is not really the best way to do it. It is probably the easiest way without custom functions.

`file_variable_name:seek (whence, offset)`

Whence	Description
<code>"set"</code>	base is position 0 (beginning of the file);
<code>"cur"</code>	base is current position;
<code>"end"</code>	base is end of file;

```
local file_seek = sol.file.open("example.txt", "r")
--x:50y:40 in example.txt. -2 would be two from the end. (40)
file_seek:seek("end",-2)

local value = file_seek:read()

--Convert string "40" to the number value 40
print("The variable is:"..tonumber(value))

--Add 5 onto the value
print("The variable is:"..value + 5)

file_seek:close()
```

## Read & Write Functions Script

This script will be very useful for managing read/write, but the easiest way to do this is to just use the engine's functions `game:get_value` and `game:set_value`.

```
--Read line function
local function readLines(sPath)
    local file = sol.file.open(sPath, "r")
```

generated by haroopad

```

local file = sol.fs.open(sPath, "r")
if file then
    local tLines = {}

    local sLine = file:read()
    while sLine do
        table.insert(tLines, sLine)
        sLine = file:read()
    end
    file.close()
    return tLines
end
return nil
end

--Write line function
local function writeLines(sPath, tLines)
    local file = sol.fs.open(sPath, "w")
    if file then
        for _, sLine in ipairs(tLines) do
            file:write(sLine)
        end
        file:close()
    end
end

local file_make_test = sol.fs.open("test.txt", "w")
file_make_test:close()

local tLines = readLines("test.txt") -- Read/open this file
table.insert(tLines, "This is the first line!\n") -- Line 1
tLines[2] = "This is line 2!\n" -- Line 2
tLines[3] = "This is line 3!\n" -- Line 3
tLines[4] = 50 -- Line 4

table.remove(tLines, 2) -- Remove line 2
writeLines("test.txt", tLines) --Write lines to this file
print("Lines in the file: ", #tLines) --Print number of lines

--Open file. You must open the file to get the value
local tLines = readLines("test.txt") -- Read this file

--Print line 3. Line 4 will not be 50 because we removed line 2. That means line 3 will be 50.
print("Line 4 value is: "..tLines[3])

```

You can ignore the functions, but it would be best to try to understand them. The only important part is after the functions.

Make the file **test.txt**.

```
--This makes the file `test.txt`.
local file_make_test = sol.fs.open("test.txt", "w")
file_make_test:close()
```

The comments in the script below tell you how to use it.

```

local file_make_test = sol.fs.open("test.txt", "w")
file_make_test:close()

local tLines = readLines("test.txt") -- Read/open this file
table.insert(tLines, "This is the first line!\n") -- Line 1
tLines[2] = "This is line 2!\n" -- Line 2
```

generated by haroopad

```
tLines[3] = "This is line 3!\n" -- Line 3
tLines[4] = 50 -- Line 4

table.remove(tLines, 2) -- Remove line 2
writeLines("test.txt", tLines) --Write lines to this file
print("Lines in the file: ", #tLines) --Print number of lines

--Open file. You must open the file to get the value
local tLines = readLines("test.txt") -- Read this file

--Print line 3. Line 4 will not be 50 because we removed line 2. That means line 3 will be 50.
print("Line 4 value is: "..tLines[3])
```

## Chapter 16: Make a Chain Quest, Bow, Boomerang, and Hookshot

### Chain Quest

The chain quest is putting together what you have learned, some new functions, and scripts are introduced. It is more like a demo than anything else.

### Sola House

Sola house is the starting point of the game. Sola has woken in a world with no memories of her past and sets out on a journey to figure out who she is.

### Sola House > F1

On the first floor there are some blockades that must be passed to exit the house. Also, an optional quest with the female armor. You will be able to perform spin attacks and have extra speed when the spirit gem is obtained.

### Disabling Spin Attack

There are no function or method as of Solarus 1.5.3 that disables the spin attack, but there is a quick hack for it that I learned from MetalZelda.

This is accomplished by registering an event using the `multi_events.lua` script. What the script does is add a callback kinda like with timers.

#### **Register event way:**

```
-- local multi_events = require("scripts/multi_events")
-- Register two callbacks for the game:on_started() event:
game:register_event("on_started", function()
    -- Some code.
end)
```

#### **Normal way:**

```
function game:on_started()
    -- Some code.
end
```

The following code disables the spin attack until the save variable "skill\_spin\_attack" is set. It should be put in the game manager.

```
function game_manager:manage(game)

game:register_event("on_started", function()

local hero = game:get_hero()

hero:register_event("on_state_changed", function(hero, state)
    if state == "sword_loading" then
        if not game:get_value("skill_spin_attack") then
            game:simulate_command_released("attack")
        end
    end
end) -- end of hero:register
end) -- end of game:register
end
```

First off, the register event with the function on\_started:

```
game:register_event("on_started", function()
end) -- end of game:register
```

Secondly, we get the hero and register it with the function on\_state\_changed:

```
game:register_event("on_started", function()

local hero = game:get_hero()

hero:register_event("on_state_changed", function(hero, state)

end) -- end of hero:register
end) -- end of game:register
```

Thirdly, we want to apply it to the state "sword\_loading" and simulate a release on "attack".

```
game:register_event("on_started", function()

local hero = game:get_hero()

hero:register_event("on_state_changed", function(hero, state)
    if state == "sword_loading" then
        if not game:get_value("skill_spin_attack") then
            game:simulate_command_released("attack")
        end
    end
end) -- end of hero:register
end) -- end of game:register
```

To enable the spin attack as I said above, just set the save variable. I normally do it in items. I put it in the spirit gem script in the chain quest.

```
function item:on_obtained()
    game:set_value("skill_spin_attack", true)
```

```
end
```

The optional quest is just a bunch of save variable, elseif, and dialogs. I explained this in a previous chapter. Chapter 13 NPC Entity section.

```
-- Event called at initialization time, as soon as this map becomes is loaded.
function map:on_started()

--Spirit quest
function woman_armor:on_interaction()

    if game:get_value("sola_house_gem_quest_finished") then
        game:start_dialog("sola_gem.done.1")
    elseif game:get_value("sola_house_yes") and not game:get_value("sola_house_gem") then
        game:start_dialog("sola_gem.find.1")
    elseif game:get_value("sola_house_yes") and game:get_value("sola_house_gem") then
        game:start_dialog("sola_gem.yes.1", function()
            hero:start_treasure("spirit_gem", 1, "sola_house_gem_quest_finished", function()
                game:start_dialog("sola_gem.wonderful_day.1")
            end)
        end)
    else
        game:start_dialog("sola_gem.hello.1", function(answer)
            if answer == 4 then --No
                game:start_dialog("sola_gem.no.1")
            elseif answer == 3 then
                game:start_dialog("sola_gem.find.1")
                game:set_value("sola_house_yes", true)
            end
        end)
    end
end
end
```

The spirit gem was hidden in a pot by a demon. I use a npc entity above the pot tile and use on\_interaction, then start\_treasure. I set the save variable, so the hero can only get the treasure once. Very basic.

```
--Get spirit gem
function pot:on_interaction()
    if not game:get_value("sola_house_gem") then
        hero:start_treasure("spirit_gem", 1, "sola_house_gem")
    end
end
```

I use a save variable again here, but with string instead of a true/false boolean. block\_pot is the NPC and flower wall is the dynamic tile. I set the skeleton disabled because I do not want the enemy to come back or respawn.

```
--The pot vanishes if the skeleton save string is "dead".
sol.timer.start(1000, function()
    if game:get_value("sola_house_skeleton") == "dead" then
        flower_wall:set_enabled(false)
        block_pot:set_enabled(false)
        skeleton:set_enabled(false)
    end
    return true -- To call the timer again (with the same delay).
end)
```

The enemy called skeleton will be disabled until the switch on the second floor is activated.

Generated by haroopad

```
--Skeleton disabled until save variable is set
if game:get_value("sola_house_switch_off") == nil then
    skeleton:set_enabled(false)
end
```

The skeleton enemy is enabled and now can attack the hero. The wall entity and the table blockade is also disabled.

```
--Skeleton is enabled. Table and wall entity are disabled.
if game:get_value("sola_house_switch_off") == true then
    skeleton:set_enabled(true)
    table_wall:set_enabled(false)
    wall:set_enabled(false)
end
```

For when the enemy dies the script sets the "sola\_house\_switch\_off" to false from the skeleton enemy script. This is to make sure that nothing respawns when the player leaves the map and returns.

```
--Disables everything
if game:get_value("sola_house_switch_off") == false then
    skeleton:set_enabled(false)
    table_wall:set_enabled(false)
    wall:set_enabled(false)
    flower_wall:set_enabled(false)
end
end -- end of map on started
```

## Skeleton.lua

```
function enemy:on_dead()
    game:set_value("sola_house_skeleton", "dead")
    game:set_value("sola_house_switch_off", false)
end
```

A timer checking if the enemy is dead. It uses the method `enemy:get_life()`, but to be honest it would be better to use the functions `entity:on_removed()` or `enemy:on_dead()`. The timer in this case can effect the enemy's explosion death animation.

```
--Create on_activated timer
sol.timer.start(2000, function()
    if map.on_activated ~= nil then
        map:on_activated()
    return true
    end
end)

--Timer that checks every 2 seconds
function map:on_activated()

    --when the skeleton's life reaches zero, the skeleton save string is set.
    if skeleton:get_life() == 0 then
        game:set_value("sola_house_skeleton", "dead")
    end

    --The pot vanishes if the skeleton save string is "dead".
    if game:get_value("sola_house_skeleton") == "dead" then
```

```

flower_wall:set_enabled(false)
block_pot:set_enabled(false)
end

end -- end of on_activated

```

## Sola House > F2

Sets npc dialog “wall\_switch\_2” disabled and makes the wall switch active on return to the map because you do not want the player to activate it again.

```

-- Event called at initialization time, as soon as this map becomes is loaded.
function map:on_started()
--Set npc dialog "wall_switch_2" disabled and makes the switch active on return
if game:get_value("sola_house_switch_off",true) then
    wall_switch_2:set_enabled(false)
    wall_switch:set_activated(true)
end

```

Sets npc dialog “wall\_switch\_2” disabled and removed the blockages on the first floor by setting the save variable.

```

--disables npc and sets a save variable
function wall_switch:on_activated()
    game:set_value("sola_house_switch_off",true)
    wall_switch_2:set_enabled(false)
end
end

```

## Soulia Forest > Part 1

### Hidden Switch

The switch is hidden under the trees with a flower on top to give a hint to its location. A sound is played when the switch is activated and the stump vanishes. Also, the check makes sure the switch cannot be activated again and to insure the stump does not respawn.

```

--Flower stump
function flower_switch:on_activated()
    flower_stump:set_enabled(false)
    sol.audio.play_sound("secret")
    game:set_value("flower_stump",true)
end

--Flower stump check
if game:get_value("flower_stump") == true then
    flower_stump:set_enabled(false)
    flower_switch:set_activated(true)
end

```

## Bomb Stump

The bomb is assigned to the `x` command slot by default in this script `game:set_item_assigned(1, self)`. Although, one could just assign it to a key press function `game:on_key_pressed(key, modifiers)`. The following script is simple and it creates a bomb at the player's location for the hero's facing direction.

### items > bomb.lua

```
function item:on_created()

    self:set_savegame_variable("bomb")
    self:set_assignable(true)
end

function item:on_obtaining()
    -- Automatically assign the item to a command slot
    -- because it is the only existing item for now.
    -- And we have no HUD.
    game:set_item_assigned(1, self)
end

-- Called when the player uses the bombs of his inventory by pressing
-- the corresponding item key.
function item:on_using()

    local hero = self:get_map():get_entity("hero")
    local x, y, layer = hero:get_position()
    local direction = hero:get_direction()
    if direction == 0 then
        x = x + 16
    elseif direction == 1 then
        y = y - 16
    elseif direction == 2 then
        x = x - 16
    elseif direction == 3 then
        y = y + 16
    end

    self:get_map():create_bomb{
        name = "bomb",
        x = x,
        y = y,
        layer = layer
    }
    self:set_finished()
end
```

The following script removes any entity that a bomb is placed on, but the entity has to be specified in the script. In this case it checks if a bomb is overlapping an entity "bomb\_stump". It uses the function `map:get_entities_by_type()` to check a list of strings with bomb name prefixes. Bombs are placed with a prefix if there is more than one on a map.

```
--Bomb stump
sol.timer.start(1000, function()
for bomb in map:get_entities_by_type("bomb") do
    if bomb:overlaps(bomb_stump) then
        function bomb:on_removed()
```

```

        bomb_stump:set_enabled(false)
        sol.audio.play_sound("secret")
        game:set_value("bomb_stump",true)
    end
end
return true
end)

--Bomb stump check
if game:get_value("bomb_stump") == true then
    bomb_stump:set_enabled(false)
end

```

Furthermore, it is good to check if the `bomb_stump` or any other entity still exists when using the `entity:on_removed()` function or you could get an annoying error message. I will explain that in more detail later.

```

function bomb:on_removed()
    if bomb_stump ~= nil and bomb_stump:exists() then
        bomb_stump:set_enabled(false)
    end
    sol.audio.play_sound("secret")
    game:set_value("bomb_stump",true)
end

```

## Trick Chest

I previously presented the trick chest script in a previous chapter, but it is a pain to get the coordinates the previous way. This time I use the `hero:start_hurt([source_entity, damage])` method. All one has to do is get the entity from the map with its name `map:get_entity("trick_chest")`.

```

local entity = ...
local map = entity:get_map()
local game = entity:get_game()
local hero = map:get_hero()

function entity:on_created()
    self:set_traversable_by(false)
    entity:get_sprite():set_animation("closed")
end

local trick_chest = map:get_entity("trick_chest")

function entity:on_interaction()
    hero:start_hurt(trick_chest, 6)
end

```

## Soulia Forest > Part 2

### Key Stump

Since, making a door entity for a dynamic tile would be a total pain. I simply used a NPC to check if the chest that contain the key is open. Remember to set a save variable for the chest.

```
--Key stump
function key:on_interaction()
    if key_chest:is_open() then
        key_stump:set_enabled(false)
        key_stump_hole:set_enabled(false)
        game:set_value("key_stump",true)
        key:set_enabled(false)
        sol.audio.play_sound("open_lock")
    end
end

--Key stump check
if game:get_value("key_stump") == true then
    key_stump:set_enabled(false)
    key_stump_hole:set_enabled(false)
end
```

## Block Switch

This is simply moving a block on a switch. Nothing special. Remember to check the block's coordinates when returning to the map.

```
--block switch
function hole_block_switch:on_activated()
    game:set_value("hole_block_switch",true)
    hole:set_enabled(false)
    sol.audio.play_sound("secret")
end

--block switch check
if game:get_value("hole_block_switch") == true then
    switch_block:set_position(856,668)
    hole_block_switch:set_activated(true)
    hole:set_enabled(false)
end
```

## Chain Village

Chain village is where almost all of its people have fallen to death due to a strange power coming over the mountains. The heroine must get a heart shield to show the mage that she is worthy of entering Zark house. The reason is that there is a slime enemy in there. Afterwards, thee heroine finds the water shield after defeating the slime and is able to cross the lake to the Water house and get the spirit shield to enter Elfa House. This is where the mini dungeon resides.

### **chain\_village.lua**

This is a check for leaving the map after the heroine shows that she is a true hero and removes the magic barrier from Zark house. This is so the door will not close up again.

```
--Shows that she is a true hero check
if game:get_value("shows_heroism") then
    heart_door:set_enabled(true)
else
```

```
    heart_door:set_enabled(false)
end
```

This is a check for when the heroine gets the spirit shield and is able to enter Elfa house. This is so the door will not close up again.

```
--Obtains the spirit shield check
if game:get_value("spirit_shield") then
    elfa_house_door:set_enabled(true)
else
    elfa_house_door:set_enabled(false)
end
```

This is for when the hero tries to enter a blocked door. In this case when the hero enters the door she is damaged and sent back by a evil spirit. An animation with stars over head and sound is played.

```
-- Call a function every half second
-- Damaged when overlapping elfa house doorway
sol.timer.start(500, function()
    if hero:overlaps(elfa_house_door) and elfa_house_door:is_enabled() == false then
        sol.audio.play_sound("danger")
        hero:start_hurt(elfa_house_door, 1)
        game:start_dialog("evil_spirit")
        hero:walk("66", false, true)
        hero:set_animation("bubble_stars")
    end
    return true
end)
```

The hero interacts with the mage NPC and she checks if the heroine has a heart shield. I use the `game:has_item()` method to check this without a need for a save variable.

```
local has_heart_shield = game:has_item("heart_shield")

--Checks if the hero has a heart_shield and disables the barrier if she does.
function sprite:on_interaction()
    if has_heart_shield and not game:get_value("magic_gone") == true then
        game:start_dialog("heart_shown", function()
            game:set_value("shows_heroism", true)
            heart_door:set_enabled(true)
            game:set_value("magic_gone", true)
            sol.audio.play_sound("teleporter")
        end)
    elseif not has_heart_shield then
        game:start_dialog("show_heart")
    end

    --A check for when the barrier is removed. A new dialog.
    if game:get_value("magic_gone") == true then
        game:start_dialog("heart_luck")
    end
end
```

The heroine is pushed back if she tries to enter before the barrier is vanquished.

```
-- Call a function every second.
--If the heroine tries to enter Zark house if barrier is active, then she is flung back with a sup
sol.timer.start(100, function()
    if hero:overlaps(magic_info) and not game:get_value("magic_gone") == true then
        hero:walk("66", false, true)
```

generated by haroopad

```

        hero:set_animation("bubble_surprise")
        game:start_dialog("magic_barrier")
    else
        magic_info:set_enabled(false)
    end
    return true
end)

```

## Chain Village > Shop

The shop is the only place where the heroine can obtain the heart shield. It costs 20 gems.

## Chain Village > Zark House

Okay, I mentioned above in soulia forest that I would explain more about the function `entity:on_removed()`. One must check if it is nil or exists if an entity is placed in the function `entity:on_removed()`.

```

--When slime is removed
--Bookcase vanishes and chest is revealed behind it
function slime:on_removed()
    if bookcase ~= nil and bookcase:exists() then
        bookcase:set_enabled(false)
    end

    if chest ~= nil and chest:exists() then
        if not chest:is_open() then
            sol.audio.play_sound("secret")
        end
    end

    game:set_value("zark_house_enemy_defeated",true)
end

```

Otherwise, you will get an errors like when leaving the map. For instance:

```

Error: In on_removed: [string "maps/chain_village/zark_house.lua"]:21: attempt to index global 'ch
Error: In on_removed: [string "maps/chain_village/zark_house.lua"]:17: attempt to index global 'bo

```

A better solution in this case would be function `enemy:on_dead()` because in this case the script is working with a slime enemy.

```

--When slime is dead
--Bookcase vanishes and chest is revealed behind it
function enemy:on_dead()
    bookcase:set_enabled(false)

    if not chest:is_open() then
        sol.audio.play_sound("secret")
    end

```

```
game:set_value("zark_house_enemy_defeated",true)
end
```

This is to just check to make sure the bookcase and sline do not return.

```
--Slime and bookcase return check
if game:get_value("zark_house_enemy_defeated") == true then
    bookcase:set_enabled(false)
    slime:set_enabled(false)
end
```

## Chain Village > Water House

Interacting with the book makes the bookcase vanish.

```
--Interacting with the book npc makes bookcase vanish and reveal wall switch
function book:on_interaction()
    if not game:get_value("water_house_book") then
        game:start_dialog("water_house.book", function()
            bookcase:set_enabled(false)
            sol.audio.play_sound("open_lock")
            game:set_value("water_house_book",true)
        end)
    end
end

--Checks if book is active and makes sure the bookcase stays gone when returning to the map.
if game:get_value("water_house_book") == true then
    bookcase:set_enabled(false)
end
```

Removed the wall blockade and the shadowing belonging to it, which is just a dynamic tile shadow.

```
--Removes wall and its shadow
--Sets wall switch save variable
function wall_switch:on_activated()
    blockage:set_enabled(false)
    blockage_shadow:set_enabled(false)
    game:set_value("water_house_wall_switch",true)
end

--Checks to make sure the blockage and shadow is disabled when returning to the map.
if game:get_value("water_house_wall_switch") == true then
    wall_switch:set_activated(true)
    blockage:set_enabled(false)
    blockage_shadow:set_enabled(false)
end
```

A series of switches and checks to make sure the chest appears to obtain the spirit shield. The comments say everything.

```
--switch 1 enables switch 2 and it appears
function switch_1:on_activated()
    switch_2:set_enabled(true)
end

--switch 2 makes a chest appear and plays a sound.
```

```

function switch_2:on_activate()
    chest:set_enabled(true)
    sol.audio.play_sound("secret")
    game:set_value("water_house_floor_switch",true)
end

--Checks to make sure the chest and switch_2 are not visible when entering the map
if switch_1:isActivated() == false and switch_2:isActivated() == false then
    chest:set_enabled(false)
    switch_2:set_enabled(false)
end

--Checks to make sure the switches stay active and the chest/switch_2 is visible on returning to the map
if game:get_value("water_house_floor_switch") == true then
    switch_1:set_activated(true)
    switch_2:set_activated(true)
    chest:set_enabled(true)
    switch_2:set_enabled(true)
end

```

## Chain Village > Elfa House

### Cutscene

A cut scene is just a series of movements and dialogs. Commonly, the player cannot move during these events and pictures scenes are displayed.

In this case a NPC named Elfa runs toward the hero. At the same time the hero runs toward Elfa. The `hero:walk` method freezes the hero until the movement is over. Once the `hero:walk` function is finished the hero can move again, so we must freeze her with the method `hero:freeze()`. Usually, one would not want the HUD to show when the image appears during a cut scene. In that case, we would set our HUD to false and we are using christopho's hud, so it would be `game:set_hud_enabled(false)`. It will not be found in the documentation. Once the cut scene is over we want to unfreeze the hero using the method `hero:unfreeze()` and set a save variable because we do not want the cut scene to repeat.

```

local map = ...
local game = map:get_game()

--Set a boolean for image
local elfa_separate_cutscene = false

--Load image
local night_background = sol.surface.create("background/fantasy_background.png")

-- Event called at initialization time, as soon as this map becomes loaded.
function map:on_started()

--Cutscene
if not game:get_value("end_of_cutscene") then
    hero:walk("2222222222")
    local elfa_movement = sol.movement.create("path")
    elfa_movement:set_path({6,6,6,6,6,6,6,6,6})
    elfa_movement:set_speed(88)
end

```

```

elfa_movement:start(elfa, function()
    sol.timer.start(map, 500, function()
        hero:freeze()
        game:start_dialog("elfa_house.elfa", function()
            game:set_hud_enabled(false)
            elfa_separate_cutscene = true
            sol.timer.start(map, 1000, function()
                game:start_dialog("elfa_house.night_background", function()
                    sol.timer.start(map, 1000, function()
                        game:set_hud_enabled(true)
                        elfa_separate_cutscene = false
                        hero:unfreeze()
                        game:set_value("end_of_cutscene", true)
                    end)
                end)
            end)
        end)
    end)
end

--Display image
function map:on_draw(dst_surface)
    if elfa_separate_cutscene then
        night_background:draw(dst_surface)
    end
end

```

The hero must find a secret passage to enter the dungeon and it is heavily hinted in the dialog that it is under a bookcase. A passage will be shown when the hero interacts with the bookcase. In this case, a custom entity is used for the interaction because it can be resized to fit the length of the bookcase.

```

--Dungeon entrance bookcase
function bookcase:on_interaction()
    game:start_dialog("elfa_house.bookcase", function()
        sol.audio.play_sound("switch")
        large_bookcase:set_enabled(false)
    end)
end

```

## Chain Village > Underground

### Green Orc Soldier

First off, I would like to explain the orc soldier on this map because he is a little unique. The soldier consists of two sprites. The orc and the sword to be precise. The orc is invincible and can only be disabled by hitting his sword. I set it up so when the first orc is tapped 3 times, then the bookcase vanishes.

Let me break down how I do this. I first make the orc body and sword ignore all attacks with the method `enemy:set_invincible_sprite()`. Then I gave the sword a custom consequence with the sword using method `enemy:set_attack_consequence_sprite(sword_sprite, "sword", "custom")`. Afterward, I used the `function enemy:on_custom_attack_received(attack, sprite)` to set what happens when the sword is hit by the hero. In the `green_orc_soldier.lua` I use a boolean to check when to immobilize using the method `enemy:immobilize()`, but one could just use a timer instead.

**Timer example:**

```

function enemy:on_custom_attack_received(attack, sprite)

    if attack == "sword" and sprite == sword_sprite then
        sol.audio.play_sound("sword_tapping")
        print("tapped")

        local hero = enemy:get_map():get_entity("hero")
        local angle = hero:get_angle(enemy)
        local movement = sol.movement.create("straight")
        movement:set_speed(128)
        movement:set_angle(angle)
        movement:set_max_distance(500)
        movement:set_smooth(true)
        movement:start(enemy)
        --immobilize orc
        sol.timer.start(1000, function()
            enemy:immobilize()
        end)
        tapped = tapped + 1
        print("tapped: ", tapped)
    end
end

```

**green\_orc\_soldier.lua**

```

local enemy = ...

local game = enemy:get_game()
local map = enemy:get_map()
local hero = map:get_hero()
local body_sprite
local sword_sprite
local movement
local immobilize = false
local tapped = 0

function enemy:on_created()

    body_sprite = enemy:create_sprite("enemies/green_orc_soldier")
    sword_sprite = enemy:create_sprite("enemies/green_orc_soldier_sword")
    enemy:set_life(140)
    enemy:set_damage(0)
    enemy:set_size(16, 16)
    enemy:set_origin(8, 13)

    -- Make the sword and body sprite ignore all attacks.
    enemy:set_invincible_sprite(sword_sprite)
    enemy:set_invincible_sprite(body_sprite)

    -- Except the sword.
    enemy:set_attack_consequence_sprite(sword_sprite, "sword", "custom")
end

function enemy:on_custom_attack_received(attack, sprite)

    if attack == "sword" and sprite == sword_sprite then
        sol.audio.play_sound("sword_tapping")
        print("tapped")

        local hero = enemy:get_map():get_entity("hero")
        local angle = hero:get_angle(enemy)

```

```

local movement = sol.movement.create("straight")
movement:set_speed(128)
movement:set_angle(angle)
movement:set_max_distance(500)
movement:set_smooth(true)
movement:start(enemy)
immobilize = true
tapped = tapped + 1
print("tapped: ", tapped)
end
end

--Create on_activated
sol.timer.start(2000, function()
  if enemy.on_activated ~= nil then
    enemy:on_activated()
    return true
  end
end)

function enemy:on_activated()

print("activate")

if immobilize == true then
  enemy:immobilize()
end

if tapped >= 3 then
  game:set_value("underground_tapped_3_times",true)
end

local distance_to_hero = hero:get_distance(enemy)

  if distance_to_hero < 100 then
    movement = sol.movement.create("target")
    movement:set_target(hero)
    movement:set_speed(48)
    movement:start(enemy)
    immobilize = false
  else
    movement = sol.movement.create("random")
    movement:start(enemy)
    immobilize = false
  end
end

function enemy:on_movement_changed()

  body_sprite:set_direction(movement:get_direction4())
  sword_sprite:set_direction(movement:get_direction4())
end

```

## Underground Map

I do not present anything new with this map, so nothing to explain. The only difference was with `green_orc_soldier.lua` because when it is tapped three times a save variable is set when the sword is hit three times.

```

local map = ...
local game = map:get_game()
local hero = map:get_hero()

```

```
-- Event called at initialization time, as soon as this map becomes is loaded.
function map:on_started()
    game:set_value("boss_map",true)
    game:set_value("scrolling_credits",true)

    function leaver_1:onActivated()
        wall_1:set_enabled(false)
        game:set_value("underground_leaver_1",true)
    end

    if game:get_value("underground_leaver_1") == true then
        leaver_1:set_activated(true)
    end
end

function door_1:onOpened()
    print("opened door")
    hero:teleport("chain_village/underground_miniboss", "boss_destination", "fade")
end

sol.timer.start(1000, function()
    if game:get_value("underground_tapped_3_times") == true then
        bookcase_1:set_enabled(false)
    end
    if door_1:isOpening() == true then
        --hero:teleport("chain_village/underground_miniboss", "boss_destination", "fade")
    end
    return true
end)
```

## Chain Village > Underground Boss

The underground boss was originally supposed to be the mini boss, but I decided to make it the boss because the orcs were hard enough. The script uses the `map:get_entities()` function to grab all the solid switches on the map. Thanks to Max on the Solarus forum. The skeleton boss is immortal and can only be hurt when all switches are active. The hero has 5 seconds to damage the skeleton before he reverts to being immortal.

```
local map = ...
local game = map:getGame()

-- Event called at initialization time, as soon as this map becomes is loaded.
function map:onStarted()

    local switches_on= 0

    for switch in map:getEntities("solid") do
        function switch:onActivated()
            switches_on = switches_on + 1
            if switches_on >= 4 then
                switches_on = 0
                timer_sound = sol.timer.start(5000, function() deactivate_switches() end)
                timer_sound:setWithSound(true)
            end
        end
    end
end

function deactivate_switches()
    for switch in map:getEntities("solid") do
        switch:setActivated(false)
```

```

end
end

end

```

**Sample:**

Lessons > Chapter\_16 > Chapter\_16\_Chain\_Quest.zip

## Bow

The method `hero:start_bow()` activates the bow and shoots an arrow. The `bow` animation is needed for your hero and the sounds `bow.ogg` and `arrow_hit.ogg`.

**Animation:**

Bow

**Sounds:**

`bow.ogg`  
`arrow_hit.ogg`

**Script Sample:**

A key press function can be used to activate the bow. In the sample, I put it in the `game_manager.lua` script.

```

function game:on_key_pressed(key)

    local hero = game:get_hero()

    --Bow & Arrow
    if key == "a" then
        hero:start_bow()
    end
end

```

## Boomerang

The boomerang has more detail than the bow. The boomerang will return to the hero or chase the hero if moving away from the boomerang.

`hero:start_boomerang(max_distance, speed, tunic_preparing_animation, sprite_name)`

Boomerang	Description
<code>max_distance (number)</code>	Maximum distance of the boomerang's movement in pixels.
<code>speed (number)</code>	Speed of the boomerang's movement in pixels per second.
<code>tunic_preparing_animation (string)</code>	Name of the animation that the hero's tunic sprite should take while preparing the boomerang.
<code>sprite_name (string)</code>	Sprite animation set to use to draw the boomerang then.
Source	<a href="http://www.solarus-games.org/doc/latest/lua_api_hero.html#lua_api_hero_start_boomerang">http://www.solarus-games.org/doc/latest/lua_api_hero.html#lua_api_hero_start_boomerang</a>

**Preparing Animation:**

generated by haroopad

In the sample, the `tunic_preparing_animation` is `boomerang1`.

```
hero:start_boomerang(60, 50, "boomerang1", "main_heroes/boomerang")
```

### Sprite Name:

The boomerang name in the sample is `boomerang` and I put it in the `main_heroes` directory.

```
hero:start_boomerang(60, 50, "boomerang1", "main_heroes/boomerang")
```

### Sounds:

The boomerang needs the sound `boomerang.ogg` and `sword_tapping.ogg`.

The `boomerang.ogg` sound repeats when it is activates and the `sword_tapping.ogg` actives when encountering something.

### Script Sample:

A key press function can be used to activate the boomerang. In the sample, I put it in the `game_manager.lua` script.

```
function game:on_key_pressed(key)

    local hero = game:get_hero()

    --Boomerang
    if key == "b" then
        hero:start_boomerang(60, 50, "boomerang1", "main_heroes/boomerang")
    end
end
```

## Hookshot

The method `hero:start_hookshot()` can be used to activate the hookshot.

### Animation:

The hookshot requires a sprite in the sprites entities section. The sprite name needs to be `hookshot` and needs two animations. Those animations are called `hook` and `link` for the chain.

### Sounds:

The hookshot requires the sounds `hookshot.ogg` and `sword_tapping.ogg`.

The `hookshot.ogg` sound repeats when it is activates and the `sword_tapping.ogg` actives when encountering something.

### Script Sample:

A key press function can be used to activate the hookshot. In the sample, I put it in the `game_manager.lua` script.

```

function game:on_key_pressed(key)

    local hero = game:get_hero()

    --Hookshot
    if key == "h" then
        hero:start_hookshot()
    end
end

```

**Sample:**

Lessons > Chapter\_16 > chapter\_16\_bow\_hookshot\_boomerang.zip

## Chapter 17: Game Design Walkthrough

### Enemy Types

<b>Enemy_Types</b>	<b>Description</b>
Boss	A boss is the strongest enemy in a dungeon or any other location. They are difficult to beat and sometimes are cannot be beaten unless their weakness is found.
Miniboss	They are basically weak bosses. They are strong, but not really unbeatable. It is common to gain a weapon by defeating them and that allows the player to defeat the boss in that location.
Enemy	Simple to defeat. Should not take more than a few projectiles or sword hits to take down.

### Enemy Attack Patterns

These attack patterns are normally used by all enemies. Some foes are only one of these types, but minibosses and bosses normally have more than one type.

<b>Attack_Pattern</b>	<b>Description</b>
-flation	This boss either gets bigger after each strike the player inflicts on it or smaller. The hero and boss become easier to hit when the boss inflates. When the boss deflates it takes a longer time to get to the hero and its small size might be hard to hit because it is not a big balloon.
Item	This boss is normally impossible to beat as long as he has his special item. (Staff, ring, etc)
Condition	This is a normal type of enemy. Requires one weapon to defeat it.
Multi-condition	This type of enemy is normally a boss. Sometimes the hero will have attack more than just the boss to do damage. For example, the hero will have to take out the bubbles, hit the enemy with an arrow, and strike it with the sword to make the enemy take damage.
Wack the Whistle	This enemy hides in a hole and comes out to attack.
Terrain	This type of enemy uses the terrain to attack the hero. For example, falling spikes or rocks. Also, it can spin the terrain or bump up the edges to make the hero go off balance.

generated by haroopad

Attack_Pattern	Description
Grab and throw	This type of enemy grabs and throws the hero.
One with Terrain	This type of enemy becomes one with the terrain. For example, an enemy can cover itself with water and attack. A good way to defeat this type would to know where the core is.
The Phantom	A enemy that normally floats, becomes invisible, and hides in a shadow. An item is normally needed to see it and/or light based attacks to cause damage. A phantom can use curses to hurt the hero when they try to cause damage. That curse only lasts a few seconds or a condition needs to be met to break it.
The deflectors	The projectile attacks need to be deflected and bounced back at the enemy in order for the foe to take damage. The enemy might be able to deflect back.
Projectile seeker	This enemy shoots projectiles and sometimes the projectile will follow the hero.
Hammer	This type of enemy rises in the air and slams down to the ground like a hammer.
Clone	These types of enemies can clone themselves and the only way to defeat them is to take out the original or quickly defeat them all.
limp-flation	This enemy either inflates its limps to make them bigger or ejects their limbs like a hookshot. This type of enemy normally protects its limps, so it will not take damage.
Floor mover	This type of enemy works with the floors. It can change its location and bounce off the walls at a rapid pace. For example, they can bounce off the walls by bouncing, spinning, climbing, using magic, flipping, and with their blades.
The magician	The magician is similar to the Phantom, but it normally uses magic tricks. For example, runs into a mirror, teleport, strikes with magic, turns into objects, protects itself in a hat, etc.
Dark hero	This is a dark version of the hero. Normally encountered in other worlds or dimensions.
Weapon	This type of enemy fight with weapons.
Distancer	This enemy normally hides on the ceiling, jumps on pillars, and/or flies at a distance. That way this flying foe does not take damage until it comes down to attack.
Duplicator	This type of enemy divides based on the damage that is being taken.
Color warning	This enemy display warnings that tells the hero what kind of attack it will use or how rapid its strikes will be. It does not have to be colors to be honest.
Monster commander	This type of enemy can produce weaker enemies to attack the hero.
Suicide Bomber	This type of enemy kills itself to harm the hero in an explosion.
Time traveler	This enemy can restore its health by time traveling its body. A normal effect is that it gets distorted and weakened as it keeps doing it.
The eater	This type of enemy eats the hero for a short time and spits the hero out.
Damaged by terrain only	This enemy can only be harmed by something in the terrain. A special root or statue with special powers.
Drag down	This type of enemy drags the hero down. Normally resulting in going to a different floor.
Transformer	This enemy changes and gets stronger the more it is attacked.

Check the following link for examples of bosses:

<http://zelda.wikia.com/wiki/Boss>

## Puzzles

Everyone knows that puzzles are very important in any RPG. The wonder of knowing what will happen when solving them or just the wanting to defeat the annoying bugger.

Puzzle_Type	Description
Puzzles:	A puzzle is a game, problem, or toy that tests a person's ingenuity or knowledge. In a puzzle, one is required to put the pieces together in a logical way, in order to arrive at the correct solution of the puzzle. Many beings can tell puzzles, whether they are magical or not.
Block Puzzle	The Block Puzzle involves moving shapes or plain blocks into a certain position, or the rearranging of blocks to form an image/picture. This may be to press a switch, to create a step which can be used to gain high ground, to put blocks color/colour order, move the block into a hole, etc.
Return Puzzle	The most common puzzle is the return puzzle, a puzzle in which a player must depart from a room and return again with an extra item or key, making that room passable. This can be the case with dungeon rooms with multiple exits.
Enemy Puzzle	<p>The Enemy Puzzle involves a room or area in which one or several enemies must be defeated to move on. There are three common forms. Enemy, miniboss, and boss.</p> <p>Multiple foes: This is where the player has to defeat multiple enemies in the same room. They are normally much weaker than the hero and sometimes they must be defeated in a certain order. A rewards can be gained from defeating the enemies in different orders.</p> <p>Mini Boss: The mini boss is normally encountered inside dungeons. The common reward is a new weapon or item for their defeat. Sometimes they even use an item similar to what the player will gain. They are often difficult to defeat, but not as hard as a boss.</p> <p>The boss: The player must defeat this enemy in order to pass or beat a dungeon. They are normally the most difficult foe in the dungeon to defeat. The hero normally becomes stronger in some way due to defeating them. For example, the strength gain can be more health or stronger strikes against certain types of foes.</p>
Switch Puzzle	Switches in the form of levers and buttons are used quite often and many more types can be used. For example, crystals, organic organs, and other things can be used as switches. There is really no limit on the type of switch.
Lever:	Needs to be pushed or pulled. Some inaccessible levers require use of the Hookshot (pull), Seed/nut Shooter (push), or Bombs (for timing).
Button:	Many types exist, but can be broken down into the form of weight. The weight of a block might need to be removed from a button or weight might need to be put on it, sometimes there is an item to use if the player is not heavy enough to push it. The switches can have a timer on them, meaning the player might need to hurry or the switch will have to be stepped on again.

Puzzle_Type	Description
Target Puzzle:	The Target Puzzle is normally used to open a door or in some way to help the hero reach the next floor. For example, shooting an object above a door, with a projectile weapon, can open that door. The puzzle can be made more difficult in a few ways. For example, the target can be moving or the movement might be from a platform the player is standing on. Furthermore, other objects or enemies might be moving in the hero's way! Another example, can be to activate floating objects in a certain order to reach the next floor. Weapons used are normally slingshot, bow, crossbow, and gun.
Force of Nature Puzzles:	I have noticed that wind and other forces of nature can be used to make puzzles as well. The player or objects like a bomb can move to certain points and be shot around due to the forces, like wind or maybe even plant life.
Torch puzzle:	This might be considered a force of nature, but this deserves a whole puzzle of its own. The Torch Puzzle involves the lighting of one or more torches by fire in order to receive access to another room, treasure chest, or something similar.
Riddle Puzzle:	This is normally a word based puzzle and can require doing a type of action or even wearing certain garment. Sometimes bad occurrences can happen for getting the answer wrong. The player or character can be hurt/killed, or be forced to fight a bunch of enemies. A lot of time there can be more deadly cases, but not many people do this anymore. The player's life force can decrease permanently, etc. For example, you could be cursed permanently or for a certain amount of time. The curse could have a good side too, and another riddle might be needed to break it! Do you want to?
Location Change Puzzle:	Locations can be used as puzzles. For example, the location might look exactly the same, no matter where the player goes. The player is not "always" teleporting to the same spot, but is going to a similar location. The Lost Woods in Zelda is a good example. Also, areas or dungeon rooms can switch in a different order. This might not always be a puzzle because it could be completely random. Another example, time can be used to follow a character in a location and if the player gets too far behind, then the player will not be able to pass the puzzle for getting to the next location.
Source:	<a href="https://en.wikipedia.org/wiki/Puzzle">https://en.wikipedia.org/wiki/Puzzle</a>

## Quest Types

One must think of quest types in order to make a proper story. Meeting characters can involve rescue quests, etc. Also, do not think that quest types will make your game unique every time because many characters have something unique about them. It can be a weird personality flaw, a way of living or something different about their body, but we will get more into that later. Let us start with quest types!

Quest_Type	Description
Kill quests	The character must go out and kill a specific number of creature types, or a particular non-player character and the short way to say that is NPC. These kinds of quests involve bringing back evidence of success, such as trophies or something belonging to the being. (Staff, tusks, head, etc.)
Combo quests	The player will attack certain foes or structures with a combination of attacks until the necessary number of combos has been reached. Foes in these quests are normally either immortal or infinite in amount until the player character is victorious in which the foes would be eliminated or stop appearing.

Quest_Type	Description
Delivery quests	A quest type called the delivery quest or fetch-carry quest. This is a situation where the character is sent to deliver or obtain an item from one place to another. In certain cases the character might need to collect the object first instead of being handed the item to deliver when beginning the quest. These quests are created to be difficult by asking the character to journey through different or dangerous terrain, sometimes with a time limit.
Gather quests	Gather quests, also known as collection quests, call for a player character to collect a specific amount of items. These can either be gathered from a place or environment, or need the player to kill creatures to obtain the required items. The quest could also require the character to gather a number of different items. For example, to assemble a device.
Escort quests	The Escort quest is a combination of defeating creatures to protect a non-player character(s) all while investigating an area alongside that Non-player character (NPC). A normal escort quest would have the player defending a NPC as he, it (3rd,4th, etc gender), or she moves through a monster-infested area. Most of the time the quest will force the player to kill many monsters to ensure the well-being of the NPC. Escort quests can be beneficial, in making the player's pay attention to a particular spot in order to play out a scene or reveal a section of the plot. They can also be used to transport a character from one location to another, leading the player along a route or path. However, problems with this type of quest can occur if the artificial intelligence controlling the NPC causes them to behave in unexpected or unmanageable ways. Because many of them are often done wrong, they are very unpopular among the gaming community.
Syntax quests	A phenomenon unique to text-based games, syntax quests depend on guessing the correct syntax to use to carry out a (typically simple) operation. For instance, arranging stone tablets in a certain order to make a picture or figuring out how to read the sentence and carrying out a certain action. It mostly involves arranging something in a certain order to figure out the meaning.
Hybrids	Parts of the above quest types can be put together to make more complex quests. For instance, a quest could need that the player locate the pieces needed to assemble a certain weapon or item (Gather Quest) and then use it to defeat a specific foe (Kill quest). Hybrid quests can also involve puzzles and riddles.
Quest chains	A quest chain is a group of quests that are completed in sequence they are also known as quest lines. Completion of each quest is a prerequisite to beginning the next quest in the chain. Quests usually heighten in toughness as a character goes further along the chain. The quests normally shows a single plotline in an order that explains the reason for the quests. Quest chains can also start with the opening or breadcrumb quests, in order to encourage characters to journey to a new area, where further elements of the quest chain are revealed. Through mechanisms like these, the setting of a particular location is explained to the player, with the plot or storyline being unveiled as the character progresses.
Source:	<a href="https://en.wikipedia.org/wiki/Quest_(video_gaming)">https://en.wikipedia.org/wiki/Quest_(video_gaming)</a>

## Map Design

Map design can take up a lot of time. It is best to get quick ideas from generators. [Donjon Dungeon Generator](#)

Telling you how to design your map is almost impossible because there is no right way, but I will give basic tips.

Tip	Description
	generated by haroopad

Tip	Description
Cludder	Do not put too many objects on one map unless it is like a junk yard or something like that.
Duplicate objects	The same objects all over the place would look weird. The same flower pot appearing in a room 10 times would look unnatural.
Spacing	Try to keep decent space when making a map. For example, one will want to get around the chair.
Larger Rooms	Small rooms are okay for houses, but in main locations they should be a bit more complex. For example, a square room after every door might get boring.
Outline	One should outline the map with basic tiles first because if one were to change things later after mapping, then it might look bad or a lot of things might need to be moved around. A complete map redesign can be needed in some cases when a change is decided.
Doors	Doors are always good to have because there is just that wonder of what is on the other side. Having the door locked is even better because it is like a forbidden special place.
Plant Life	Plant life (EX: Vines) around windows and other places can make maps look natural and not so plain.
Stairs	Going up to another level or floor can be as exciting as opening a door. What is up there? A stairway that is blocked can be more interesting.
Hidden Floors	Finding hidden floors can make the hero feel special and enjoy exploring the map more.
World Name	The world having a name can help you with your map organization. You do not want to get lost in over 100 maps.
Challenging	Getting through a map too easily can be a drag! Make it at least a little tough or have some locations that can be seen, so the player will not want to leave the map as soon as possible.
Heal Spots	Having spots to rest and people there to give one information is good way to not make a player mad while playing the game.

## Map Examples Legend

Border:

Room:

Down Stairs:

Up Stairs:

Closed Door:

Monster door:

generated by haroopad

No door way:

One way in door:

### Map Examples:

## Dungeon Layout

- In a dungeon you start by finding the key item.
- Paths to the item are blocked by locked doors.
- A hero diverts from the main path to get keys to those locked doors.
- The hero will find obstacles that you cannot pass without the key item.
- Once the hero gets the key items, the hero can get to the boss key and door.
- Some dungeons have places where the hero can get multiple keys at one time or require two keys for a door.
- Dungeon maps are found quite early.
- Dungeons have large areas to explore in a maze like order.
- Dungeons sometimes have optional rooms for other items.
- All other dungeons follow this pattern and repeat this pattern for more complex dungeons.

You can check the video series [Boss Keys](#) for more information.

## Sidescroller: Some Information

Sidescrollers or Platformers are normally constant action and almost no dialog. The movement normally never stops and the player stomps on or fights enemies all the way. This type differs from an ARPG in many ways. An ARPG is normally based on exploration with a top-down view (under a 3/4 perspective) and a platformer is just a straight line most of the time. Sidescroller graphics are always a side camera view. Sidescrollers can be anything someone wants them to be though. Some people mix ARPG and sidescroller or make the sidescroller a journey instead of constant action.

### Some Movements:

Jump

Crouch

Hover

Run

<b>Some Movements:</b>	
walk	
Look up	
Pound	
Push	
<b>Random Feature Dump:</b>	<b>Description</b>
Bouncers	A object that makes the player jump higher when it is jumped on.
Speed Objects	Objects that change speed on impact.
Floating Objects	They are floating objects in the air.
Growing Objects	This object grows when something is planted. Normally a seed that grows a giant plant. That way the player can climb or walk up it.
Falling Tile Breaker	Giant falling objects or NPC that break tiles.
Locked Doors	The player needs to find the key to unlock the door.
Blockades	Objects that slow down the player. For example, spikes along a path. The player would need to jump carefully onto something to avoid damage.
Secret Rooms/shortcuts	A quick shortcut on a hard to get to part of the map that allows all bonuses + beat the level without much effort or a mini game in a secret area.
Invisible Sections	These areas of the map only appear when they are hit or touched.
Object Growth	Objects that get bigger when they are hit or touched.
Floating Points	Objects that float in the air. If they are hit, then the player gets a number point value or some kind of bonus.
Flying Mode	Allows the player to fly.
Digging Mode	Allows the player to dig.
Travel Up Wall Mode	Allows the player to climb up wall.
Underwater Mode	Allows the player to swim and breath under water.
Super Mode	The ability to walk up walls, be immortal, and destroy enemies faster by just walking into them. Super mode does not last long.
Animated Backgrounds	A background that animates.
Checkpoints	The player will go back to this point if the mission is failed or death occurs.
Falling Walls	The walls slam down. Most likely causes damage.
Turning Spikes	Spikes that turn on a wheel. The player normally has to avoid this obstacle by running at the right time.
Jumping Enemies	Enemies that can jump.

<b>Random Feature Dump:</b>	<b>Description</b>
Hidden Giants	Giant creatures coming out of liquid like elements.
Balance Beam	This is a platform that the player stands on or jumps on. One edge of a platform will tilt or tilt back if the player goes to the other edge. The platform normally stays balanced when the player stands in the middle.)
Projectile Ricochet	The projectile bounces off walls or other objects.
Unlimited Ammo	No limit for projectiles that can be fired.
Scrolling Object/Scene	The scene follows behind the player preventing the player from going back or if the scene catches up with the player, then death occurs. Furthermore, this scrolling can involve an object that only makes a pass once and it will not come back after the player uses it. This prevents the player from using it again to go back.
Wheel On Wire	A wheel that moves on a wire when it is jumped on. There would normally be edges on the wheel for the player.
Falling Platform	This would be platform or bridges that falls after a certain amount of time standing on it or just by touching it.
Enemy Pole Obstacle	An enemy on an object to prevent the player character from jumping on it.
Rollers	Enemies that turn into balls and roll. Sometimes they can be used to kill other enemies.
Zigzag Flayers	Floating enemies that move in a zigzag pattern.
Distort Object	Enemies or items that make the screen wave, wiggle, and distort.
Rubbery platforms	Platforms that wiggle like rubber.
Rolling Tubes	Tubes that the player can hide in and roll over enemies. Sometimes barrels or logs are used for this.
Enemy Platform	This platform is an enemy. Normally used to trick the player when they are not paying attention.
Spiked Platforms	The platform flips once a switch is hit. Normally the platform is in a position that the player cannot jump on until the switch is flipped or the next platform will not flip.
Double Spiked Platforms	Two platforms flip at the same time once a switch is hit. Normally timed and they have spikes on one side.
Point/Bonus Trap	Objects that have enemies pop out of moving/stationary points.
Temporary Flying creatures	Flying creatures that fall once they are knocked on the head. Normally they do not regain their flight ability.
Falling Stones	Falling stones are a common trap.
Falling Block Game	Kinda like a falling block pattern game, but the blocks are huge and can kill the player.
Allied Creature	A creature that can run down and kill enemies.
Projectile Thieves	These enemies steal projectiles.

<b>Random Feature Dump:</b>	<b>Description</b>
Enemy Ghosts	Enemies that sleep when you look at them (immortal at this point) and awake you are not looking. Normally a projectile needs to be bounced off a wall to kill them.
Enemy Boo!	These enemies jump out of hidden locations.
Flame Wheels	Wheels that move and have flames on them.
Bombers	Enemies that drop bombs from the sky.
Defenders	Enemies with shields. The shield has to be destroyed first or another strike is needed.
Sign Tips	Signs to help direct the player.
Popping Platforms	Platforms that blow up after being stood on for a few seconds.
Giant Bullets	Random giant bullets that fly in one direction.
Sticky ceiling	The player gets stuck on the ceiling for a few seconds.
Movable Objects	Objects that can be pushed and moved into ditches or other places to help the player cross.
Fancy Exit	There is a fancy explosion or sparkles when the player passes a level.
Multi-jump Switch	A switch that has to be jumped or slammed on multiple times in order for it to activate.
Stomp Enemies	An enemy that dies when stomped or jumped on.

## Basic Story Making\Genre

Genre is the term for any category of entertainment. Pick the genre you like most. I will only be adding genre I like to the book, so check Wiki for more and greater detail into ones I listed. I will add this to the github. [https://en.wikipedia.org/wiki/List\\_of\\_genres](https://en.wikipedia.org/wiki/List_of_genres)

<b>Genre</b>	<b>Description</b>
Action Adventure	Journeys to places and gets through obstacles along way. Normally fighting enemies, making an escape, and saving people.
Fantasy	A fantasy story is about magic or supernatural forces.
Horror	A horror story is told to scare or frighten the people. Normally with suspense, shock and/or violence.
Mystery	A mystery story is normally about someone attempting to solve a puzzle.
Science fiction	Centered around technology (computers and machines), universes space/time travel, aliens, and genetic manipulation. EX: Steampunk, cyberpunk, and clockpunk. Science fiction is mixed a lot in future eras and time travel.

## The Plot

One has to know what the genre will be and what their hero or villain does. The hero does not always have to be the good guy and can do good or bad deeds in a story.

### Quick Plot Example:

### Quick Plot Example:

The hero Froyotay Grayerv sets out to destroys a peaceful advanced alien race because their touch can make humans immortal, but result in a mutated virus that makes humans unable to breed. He must fight to the death and find a way to rid himself of the curse that has been brought on by the mutation or else he will never be able to go home to his planet Dexerath.

### Sample Plots:

The table of plots below has 36 rows, one for each of Polti's canonical plots, and 4 columns. The first column is the plot number. The second column is the plot name. The third column lists the important actors and/or elements in the plot. And the fourth column includes one or more brief plot summaries, of no more than a single paragraph.

**Note:** "PC(s)" stands for "Player Character(s)."

No.	Plot Name	Actors and Elements	Brief Plot Summary
1	Supplication	a.Persecutor b.Supplicant c.Power in Authority	Under persecution, a village sends out emissaries to the local lord, who is not known for his kindness, to beg for relief from bandits. They are harassed by the bandits along the way.
2	Deliverance	a.Unfortunate b.Threatener c.Rescuer	Bandits have been preying on a village and threaten to burn its crops. Come save the village and destroy the bandits.
3	Revenge	a.Righteous-Avenger b.Criminal	A PC's sibling has been killed accidentally in a raid. His wife refuses to accept the rule of law, which states that she should accept a weregeld for him. She involves several PCs in her plotting to totally annihilate the family of the killer, including parents, siblings, children, and siblings' children.
4	Vengeance by Family upon Family	a.Avenging-Kinsman b.Guilty Kinsman c.Relative	One of a PC's kin killed another in confused circumstances. Whether it was accidental or not is unclear. The head of the family makes the killer responsible for supporting the bereaved spouse and children. But the widowed spouse is unwilling to accept aid from a fratricide, and involves PCs in plots to destroy the new guardian.

No.	Plot Name	Actors and Elements	Brief Plot Summary
5	Pursuit	a.Fugitive b.Pursuer	A madman wanders into town, raving about the monsters that pursue him and his poor, lost love. He is obviously high born, and needs to be taken in. A few days later a squad of pursuers arrives, to find and arrest him for crimes against humanity and his own family, led by his advisor whose advice drove him quite mad, to this awful juncture. Imagine this as a sequel to Othello, with a jealous Iago pursuing tragic Othello to bring him back to justice.
6	Victim of Cruelty or Misfortune	a.Unfortunate b.Indifferent or Cruel Master	A cruel but effective noble is disgraced after an intrigue and loses his titles, possessions and lands to an equally cruel rival. He has been banished and blinded, and is wandering, alone, without followers, pitiful, when he encounters the PCs.
7	Disaster	a.Vanquished-Power b.Victorious-Power c.Messenger	The country has been invaded and the flower of this country's knighthood has been slain. The army of the enemy is advancing, burning fields and robbing and murdering the peasants. Run for your lives!
8	Revolt	a.Tyrant b.One or more Conspirators	A well-respected noble enlists the PCs to assist in a plot to overthrow his own lord and usurp his position. His intentions seem respectable at first, but grow murkier with time.
9	Daring Enterprise	a.Goal b.Bold Leader c.Adversary	Raikiela is an artist, and has promised her masterpiece to the prince. Unfortunately, her friend showed the masterpiece to a foreign noble, who desired it and took it without permission. Without starting a war, the PCs must organize an expedition to get the masterpiece back from the foreigner and to the prince.
10	Abduction	a.Abductor b.Abducted c.Guardian	Pitolso is in love with Kreta, and Kreta is in love with Pitolso. Pitolso enlists the PCs to help steal Kreta away from those who would keep Kreta for themselves. They must abduct Kreta without killing anybody and thus rousing the implacable forces of justice. Kidnapping? No, rather liberating the love from those who have imprisoned him/her.

No.	Plot Name	Actors and Elements	Brief Plot Summary
11	Mystery or Enigma	a.Interrogator b.Seeker c.Problem	Fandolio has fallen in love with Ilsentosa, the princess and heir to the throne, and she has announced that if he wants to win her love then he must promise to solve her riddle, and if he cannot solve her riddle then he must die. He has accepted the deal, and enlists the PCs to help him find the solution. The riddle is...
12	Obtaining	a.Goal b.Several Opposed Groups c.Judge or Referee	<b>1.</b> It's time for the Spring Pageant! The maidens engage in contests to see who is the most beautiful, the most religious, the most knowledgeable about lineage, and the most charming speaker. The contest is judged by former Year Queens. The winner is crowned the Year Queen and gets to marry the Great Hunter!  <b>2.</b> It's time for the Great Hunt! Unmarried men are able to enter the contest. Each contestant must go out into the wild with only a knife, a spear, and a bow and arrow, no armor. Come back with the best catch, alive if possible. The contest will be judged by former winners of the Great Hunt. The winner is crowned the Great Hunter and gets to marry the Year Queen.
13	Familial Hatred	a.Two or more Family members who hate each other	A PC's brother hates his sister's husband, whom he sees as a whimpering syncophant. The brother-in-law in turn sees the PC's brother as a cruel bully. The PC's brother has told the PC in private, after swearing him to secrecy, that he has taken a vow to kill his brother-in-law.
14	Familial Rivalry	a.Preferred Kinsman b.Rejected Kinsman c.Object	One of the player characters and his step-brother Joli, a friendly sort of guy, have both been taken as lovers by the same merchant's wife, who hides the affairs from her husband, encourages both lovers, and makes them intrigue against each other and her husband, at the same time planting seeds of suspicion in her husband, who is very jealous.

No.	Plot Name	Actors and Elements	Brief Plot Summary
15	Murderous Adultery	a.Adulterer b.Cuckold c.Adulterer's Lover	A married friend begins to have an adulterous affair with a mysterious woman. This gives him new zest for life. Then his wife disappears, and the rumors start. A week later, the PCs are accosted by her ghost, demanding proper burial for her missing body (she doesn't know where it is) and vengeance on the adulterers. Try not to get framed for her murder along the way.

No.	Plot Name	Actors and Elements	Brief Plot Summary
16	Madness	a.Madman b.Victim	<p><b>1.</b>The local hospital begins to be the site of strange, horrible, unexplained deaths. Then a PC's relative dies from a minor ailment. Octara, one of the healers at the hospital has gone mad, and sometimes, for some insane reason, slays her patients instead of healing them. Usually Octara appears to be perfectly normal and sane.</p> <p><b>2.</b>The Bloody Bull, the PCs' favorite tavern, and Azracon their favorite tavern-keeper, have been there for years, serving them good food and drink. One day Azracon changes the sign to a Happy Horse and brings two horses in and seats them at tables. Within a week, all the rooms are full of horses that he has purchased. Within another week, you cannot get a place at his tables because the entire tavern is full of horses. Then Azracon announces to everyone on the street that he is going to marry Pythalda, the most beautiful maid in the world, and is looking for a seamstress to sew a horse-sized and horse-shaped wedding dress.</p> <p>or...</p> <p><b>3.</b>The Bachelor Prince Prophoc has always been a keen horseman. Then one day he decides to seat his favorite horse at the dinner table, displacing a foreign dignitary. This is a terrible scandal. A week or so later, Prophoc holds a dinner at which he serves all sorts of roasted delicacies to over a score of horses, and in a rage he kills a horse trainer who complains that the horses won't and can't eat the food. Soon the prince's mansion and grounds are filled with horses, and taxes are assessed to pay for the drain on the treasury. The people of the city are about to rise and overthrow him, when he declares that he has fallen in love, and that he will marry. There is much rejoicing, and it seems that things have gotten better, since the number of horses in the prince's mansion falls to normal levels. Then, on the wedding day, he forces the city's high priest, with death the penalty for refusal, to marry him to his favorite white mare.</p>

No.	Plot Name	Actors and Elements	Brief Plot Summary
17	Fatal Imprudence	a.Imprudent Person b.Lost Object c.Victim	A character known to the player characters, and which we recommend be one of the PCs, has a famous item, which may or may not be magical. A curious family member was playing with it, took it out of the house, and carried it around for a while, gradually growing bored of it, and after being attracted to some unusual noise and then frightened by seeing something s/he shouldn't have seen, drops the item and runs home, where s/he is found later, shivering in fear. The guard finds the famous item at the scene of a horrifying murder, and as a result the PC becomes the main suspect.
18	Involuntary Crimes of Love	a.Lover b.Beloved c.Revealer	Bonchanto, a friendly but mischievous acquaintance of one of the PCs, introduces him to Demice, a beautiful young woman of respectable family, who has much in common with the PC, but takes some pains to make sure that the PC doesn't meet Demice's parents. At the same time he offers to act as a go-between for the lovers. He arranges for them to meet in secret several times. Soon thereafter he leaves town. Then a letter from Bonchanto arrives at Demice's home, and all hell breaks loose. Her parents come down on the PC like a ton of bricks. Here's why... Nearly 20 years ago, Demice's supposed parents couldn't bear children of their own, and so they asked the PC's parents to adopt one of their children. This wish was granted. The PC has taken his own sister as his lover!

No.	Plot Name	Actors and Elements	Brief Plot Summary
19	Kinsman Kills Unrecognised Kinsman	a.Killer b.Unrecognized Victim c.Revealer	Politics are getting nasty again, and the loyal opposition have a hooded leader who has been rabble rousing against the main PC's family interests with some effect. One of the PCs is dispatched to a meeting with the secretive leader to try to negotiate or force a solution. Negotiations are not going well, when the city forces decide on that time to raid the meeting and frame or kill the secret leader. They give the secret signal to the PC, the signal to withdraw, and when this happens the secret leader suddenly starts fighting to reach the PC, shouting loudly with his sword flashing wildly. No matter what the PC does, whether the secret leader is rescued or not, whether he is killed or arrested (if the PC does nothing the masked one will be killed in the fracas), when he is unmasked he will be found to be the PC's adoring younger brother.
20	Self Sacrifice for an Ideal	a.Ideal b.Hero c.Person or Thing Sacrificed	After declaring that only through sacrifice will our land embody IDEAL, a very well respected and loved elderly couple throw themselves onto a bonfire (with spectacular but unclear magical consequences). This would work best if the elders were either priests at a temple the PCs attend, or grandparents or parents of one or more of the PCs. The authorities, ignorant of the planned sacrifice, will have already dispatched troops to break up the gathering and arrest the leaders.
21	Self Sacrifice for Kindred	a.Hero b.Kinsman c.Person or Thing Sacrificed	Mideratho has always been a short-breathed, wimpy, genius. He can understand the workings of mechanisms like nobody's business, and has even invented his own minor sorcerous cantrip to better play with mechanical creations. To his misfortune his parents wish to have a war hero as a son. In order to make his parents happy, Mideratho joins the most hazardous duty military unit he can, and must now learn to fight. Does he lose his sight or a hand in combat? Does he get killed? Can the PCs train him well enough so he can survive? Will he ever get to work on mechanisms again?

No.	Plot Name	Actors and Elements	Brief Plot Summary
22	All Sacrificed for Passion	a.Hero b.Object of Passion c.Person or Thing Sacrificed	The Bachelor Prince Santus, the greatest general the land has had in a generation, has conquered two neighboring states by the age of 18, but has been otherwise sheltered from life. He falls in love with Oclea, a prostitute hired by the terrible King Rometradi to seduce him, and completely loses interest in war and his principality, except as a tool which he will spend freely to capture Oclea (who may like him or may despise him, signals are mixed) from Rometradi's land. And worse, he declares he will marry her and make her the queen, and if he can't then he will abdicate his throne to Aelus, his idiot brother.
23	Sacrifice of Loved One	a.Hero b.Beloved Victim c.Cause for Sacrifice	Politics are getting nasty again, and the loyal opposition have a hooded leader who has been rabble rousing against the main PC's family interests with some effect. The PC discovers, perhaps he is told by his parent, that the hooded leader is his own, adoring, younger brother Antigoly. The PC is dispatched to a meeting with Antigoly to try to negotiate or force a solution. After the initial meeting fails, the PC is told to do whatever it takes to stop the enemy movement. At the same time, city officials inquire about the location of the enemy leader, and place a bounty on his head. Antigoly refuses to meet with or speak to his family and their agents, including the PCs.

No.	Plot Name	Actors and Elements	Brief Plot Summary
24	Rivalry Between Superior and Inferior	a.Superior b.Inferior c.Object	<p><b>1.</b>Suddenly Stranoc the Guiding Hand, the most learned magical adept of the land, is at magical war with his oldest friend and former student Vulfus the Black Spider over possession of ARTIFACT. The PCs, who have dealt with both, have no choice. They are involved without their own consent, and must deal with all manner of horrible monstrosities as they attempt to survive, save their friends and families, and get out of town.</p> <p><b>2.</b>Rivalry of Two Peoples: Referring back to plot 10, the land was recently conquered by a foreign people, and the High Lord sees the opportunity to marry Karetta to Frusta, a noble of his own people, as a way to legitimize the conquest. He will not give his approval to Karetta and Pitolso, who has kidnapped Karetta heedless of the consequences. Frusta sees her abduction as an opportunity to rescue her from bandits and thus prove his worth in her eyes.</p>
25	Adultery	a.Deceived Spouse b.Adulterer c.Adulterer's Lover	<p><b>1.</b>Thossuleo sends his wife, Fonya, to visit her family, sells most of his stuff, and goes wild buying baubles for a beautiful young girl, Crotia, who may or may not care for him. She is happy to accept his gifts, however.</p> <p><b>2.</b>Thossuleo loves his wife, Fonya, very much. Then the persecution from Crotiadek, a noble and judge, begins. Thossuleo loses money, status, and finally his freedom. After he is jailed his wife divorces him and moves into Crotiadek's home.</p>
26	Crimes of Love	a.Lover b.Beloved c.theme of Dissolution	Procsoutia falls in love with her twin sons Abon and Aromed, and causes them to lay with her. The results of these matings are monstrous and terrible.

No.	Plot Name	Actors and Elements	Brief Plot Summary
27	Discovery of Dishonor of a Loved One	a.Guilty Beloved b.Discoverer	<p><b>1.</b>Lyclus and Lia are very proud of their successful daughter Linbaria. Then evidence is found that she has been an assassin working for an evil noble.</p> <p><b>2.</b>Linbaria is very proud of her parents Lyclus and Lia. Then she discovers evidence that Lia had once been a prostitute. Even worse, the evidence indicates that Lia has returned to her former profession in secret.</p>
28	Obstacles to Love	a.Lover 1 b.Obstacles c.Lover 2	<p><b>1.</b>Nymus and Teropha are terribly in love with each other. However, because of their incompatible temperaments every time they make plans to marry they get in a huge fight and end up calling off the engagement. Every time they do this they involve all their friends and end up getting mad at all of them.</p> <p><b>2.</b>Nymus and Teropha are terribly in love with each other. However, their parents do not approve, and have both placed very difficult requirements (quests) upon their children's beloved. In order to marry, Nymus must give Teropha's parents a sieve full of Styx water, and Teropha must produce a wedding dress made of purest starlight.</p>
29	An Enemy Loved	a.Beloved Enemy b.Lover c.Hateful Kinsman	Montirkul and Pytha have fallen in love despite their families being at war. The greatest warriors in each of the families are Katal and Shokalmor, whose bitter rivalry will destroy the lovers and tear their families apart.

No.	Plot Name	Actors and Elements	Brief Plot Summary
30	Ambition	a.Ambitious Person b.Coveted Object c.Adversary	<p><b>1.</b>Fleon is an ambitious general who desires to rule the city and establish a royal line. His wife, Paria, acts as his conscience and to stay his hand as he spins plots and dispatches troops to bring the mercantile and noble interests under his thumb. Finally, Fleon becomes the city chief consul, and has made enough powerful friends to change the charter of the city from a republic to a dictatorship, and Paria must act now if she would stem his ambition.</p> <p><b>2.</b>Prince Mepilus wishes to become king, but his father, King Promanast, shows no sign of weakening, and even worse his older brother, Crown Prince Aphorphon, has been declared the heir apparent. Mepilus hatches a plan to have Aphorphon slay Promanast unknowingly, upon which terrible occasion Mepilus will declare martial law, avenge his father upon his brother, and take the crown of the city for his own.</p>
31	Conflict with a God	a.Mortal b.Immortal	<p><b>1.</b>General Panmodon has been arrested, and when he is banished for his crimes he swears bloody vengeance on his home city and on the city's patron god, Lys. He goes away and begins to gather horrible artifacts with the goal of using them to bring plague, war, and famine to the city and thus weaken Lys, so that he can finally destroy Lys's temple and slay the god itself.</p> <p><b>2.</b>The Noble Sturix begins a persecution against the pacifist followers of a new god named Aes. He believes that their religion will lead to disobedience of the civic cult, rebellion against the civil authorities, famine, war, plague, and pestilence, and so he persecutes them vigorously, putting as many of them up on crosses as he can find.</p>

No.	Plot Name	Actors and Elements	Brief Plot Summary
32	Mistaken Jealousy	a.Jealous One b.Object of Jealousy c.Supposed Accomplice d.Author of Mistake	<p><b>1.</b>Medarion is baselessly jealous of his wife Clexia, and continues to grow more and more jealous of her. He persuades Hametrea to infiltrate her group of friends and search out her "many" infidelities. Clexia has two doubles in the populace, one is a poor woman, and the other is a visiting youth who has disguised himself as a woman to hide from pursuers. Medarion himself has also has doubles, a sorceror with the visiting circus and his drooling idiot twin brother.</p> <p><b>2.</b>Medarion discovers his wife Clexia is having an affair with his best friend, Promurtin. Medarion feigns ignorance, but encourages Promurtin to be jealous of Clexia by talking about how much Clexia speaks of her old friend Nantius, and how little she speaks of Promurtin. At the same time Medarion tells Clexia stories about how Promurtin has been seen about town with Nantius and a prostitute named Hametrea.</p>
33	Faulty Judgement	a.Mistaken One b.Victim of Mistake c.Author of Mistake d.Guilty Person	Oulitarth has been consorting with prostitutes and in order to protect him from being disinherited by his very conservative family, his friend Taverion takes the blame. Taverion's wife, Marnasia, hears of this and leaves him.
34	Remorse	a.Culprit b.Victim c.Interrogator	Samarand has led a bad life, and now he is haunted by the ghosts of his misdeeds, which in reminding him of his crimes cause him incredible pain and terror. He finally has remorse, and while still haunted, begins to make some amends to his victims. Please help him with amends, and with the ghosts which terrorize him.
35	Recovery of a Loved One	a.Seeker b.One Found	<p><b>1.</b>Mantraeas learns from an oracle that his father, once thought dead, is still alive. Now he will seek his father out.</p> <p><b>2.</b>Marnady learns from an oracle that she has a twin sister, and that this twin sister is still alive. She must seek her sister out and bring her home.</p>

No.	Plot Name	Actors and Elements	Brief Plot Summary
36	Loss of a Loved One	a.Kinsman Slain b.Kinsman Witness c.Executioner	Nyoquilat has taken a vow of chastity, and one day he comes home to the horrible sight of his family being slaughtered like sheep by horned raiders. He runs in to try and save them, and is beaten and left for dead, yet he lives. Now he asks for aid from any who would give it.
<b>Source:</b>	<a href="https://www.rpglibrary.org/">https://www.rpglibrary.org/</a>	The 36 Plots by Loren J. Miller	<b>CC-BY-SA 3.0</b>

## Title

The title should be made around the time of the plot. Most of the time it describes the story in few words. It should be something related to the genre in order to help people know what the story is about. I will take my quick plot from before and make a title.

### Quick Plot Example:

The hero Froyotay Grayer sets out to destroys a peaceful advanced alien race because their touch can make humans immortal, but result in a mutated virus that makes humans unable to breed. He must fight to the death and find a way to rid himself of the curse that has been brought on by the mutation or else he will never be able to go home to his planet Dexerath.

### Title:

Froyotay's Space War: The Battle for Rebirth

## Characters

Describe the hero, allies, and enemies.

### Template:

Name:

Nickname:

Occupation:

Gender:

Clothing:

Meeting:

Description:

Birth:

BMC (Before Magic Common Era) or MC (Magic Common Era)

Age:

Race:

Audio Theme:

Fight Audio:

<b>Template:</b>
Attacks:
Hair Type:
Hair Color:
Eye color:
Ear Type:
Skin Color:
Handwear:
Accessory Type:
Accessory Color:
Class:
Height:
Facial hair:
Mouth:
Skin:
Deformities:
Special marks:
Favorite food:
Favorite Drink:
Favorite Color:
Personality:
Associations:

## Goals

What are the characters trying to do?

1. Save a princess?
2. Get revenge on the creatures that ate her family?
3. Take over the world/universe/galaxy?
4. Make the most powerful AI in the known existence?
5. Create the strongest genetically modified soldier?
6. Save a race of aliens on a planet from a cyborg invasion?
7. Protect your kingdom from an invasion of spore monsters?
8. Be the greatest magic swords master that will ever live?
9. Create a dimensional parallel world machine to escape your doomed dimension?

## Story Flow & Outline

List the events that happen.

Jetarf family was eaten ----> he kills one of the aliens -----> gets the alien spaceship remodeled -----> looks for a ship called, "@#\$@#%@@@!.. word in alien tongue."

## Outline styles

### **Tips before getting into outlines:**

Outlining is a big part of game development. The second would be dialogue, but a lot of time the main character normally does not talk. The main difference between story writing and game writing is outline. What do I mean? The only thing needed to start the creation of a game is an outline and then simple dialogue or almost none at all. (Story writing is way more descriptive.) For example, an outline is needed to know how to make the artwork and what lands or kingdoms to set up. Also, names of the characters are needed and how they will look or what appeal type (Medieval or futuristic clothing?) they need. The type of outline depends on you.

## Mind Mapping Software

Bubble outlining is kinda like mind mapping I would normally do this on paper, but maybe others prefer differently. - Give thanks to Renkineko for mentioning mind mapping.

Wisemapping

Freemind

MindMaple

## Common Outline Styles

- Bubble outline
  
- Cornell Note Style
  
- Box and Arrow Style
  
- Common Bullet Outlining

## Names

To Name:
Items
Towns
Shops/buildings
Paths/roads
Forests/jungles/mountains/islands

To Name:
Characters
Worlds
Main Quests
Optional quests

## Some Fictional Beings

Being	Description
Dwarves	Short humanoid people. Usually, strong and good at crafting weapons of any kind.
Elves	A humanoid race with long ears. They live for 1000s of years, are good archers, and are skilled with magic.
Goblins	Known as short greedy ugly creatures. Many countries have conflicting information about them. Some smart, dumb, magic, etc.
Fairy	Small humanoid creatures that use magic and are very smart.
Vampire	A humanoid that is immortal. They suck human blood and have super strength. Normally their heart needs to be damaged in order to kill them, but I think losing their head will work too.
Dragon	A giant lizard with wings. Normally it will breath fire, but there are different types of dragons. EX: Ice dragon and plant Dragons. One can shoot ice and the other acid.
Demon	A being of darkness that is known for corrupting and harming humans.
Angel	A being of light that is supposed to watch over and protect humans.
Furries	Any human cross breed with an animal. EX: A human with cat ears and a tail.
Giants	Super tall beings that can squish a human like a bug, but they are not all that tall. They are taller than the tallest human though. Some are elemental humanoids. Rock giant, ice giant, lava giant, ect.
Werewolves	Half man and wolf. Turns into a wolfman at a full moon, but some can change at will.

## Personality List

### Positive Traits

Personality Traits:
Dramatic
Steady
Artful
Dreamy
Irreligious
Unhurried
Firm
Proud

<b>Personality Traits:</b>
Challenging
Maternal
Sharing
Soft
Confidential
Impressive
Benevolent
Dynamic
Dominating
Mystical
Capable
Precise
Ambitious
Steadfast
Modern
Stoic
Aspiring
Cheerful
Masculine
Secure
Hurried
Flexible
Insightful
Genuine
Stable
Discreet
Optimistic
Idealistic
Scholarly
Questioning
Contradictory
Decisive
Competitive
Businesslike

<b>Personality Traits:</b>
Gentle
Invisible
Active
Irreverent
Experimental
Extraordinary
Unpatriotic
Leaderly
Teacherly
Curious
Sophisticated
Organized
Solid
Busy
Obedient
Sporting
Strong
knowledgeable
Warm
Responsive
Selfconscious
Stubborn
Patient
Manysided
Sarcastic
Impressionable
Inoffensive
Private
Honest
Planful
Soiid
Bigthinking
Empathetic
Farsighted

<b>Personality Traits:</b>
Friendly
Reserved
Unsentimental
Religious
Honorable
Persuasive
Subtle
Predictable
Complex
Kind
Preoccupied
Folksy
Studious
Energetic
Decent
Witty
Noncompetitive
Tidy
Imaginative
Principled
Intense
Relaxed
Balanced
Tasteful
Unchanging
Goodnatured
Surprising
Stern
Sexy
Sensual
Playful
Insouciant
Deceptive
Wellrounded

<b>Personality Traits:</b>
Multileveled
Sensitive
Casual
Considerate
Incorruptible
Sweet
Chummy
Sage
Absentminded
Patriotic
Responsible
Freethinking
Skillful
Sociable
Logical
Dutiful
Breezy
Fair
Solitary
Mellow
Scrupulous
Focused
Accessible
Sober
Daring
Venturesome
Constant
Selfsufficient
Tough
Enthusiastic
Passionate
Highbred
Wellbred
Deep

<b>Personality Traits:</b>
Prudent
Progressive
Wise
Pure
Thorough
Forgiving
Elegant
Cultured
Sentimental
Innovative
Loyal
Conservative
Practical
Adaptable
Courageous
Courteous
Brilliant
Shrewd
Youthful
Soft
Outspoken
Strict
Independent
Spontaneous
Peaceful
Open
Protective
Popular
Hypnotic
Respectful
Selfless
Reflective
Objective
Impersonal

<b>Personality Traits:</b>
Quiet
Mature
Exciting
Oldfashined
Charismatic
Appreciative
Modest
Unpredicatable
Intelligent
Earthy
Creative
Formal
Impassive
Clever
Agreeable
Uncomplaining
Admirable
Unaggressive
Understanding
Glamorous
Unreligious
Faithful
Clearheaded
Effeminate
Calm
Original
Emotional
Romantic
Attractive
Winning
Punctual
Cute
Perceptive
Political

<b>Personality Traits:</b>
Ordinary
Confident
Dedicated
Clean
Neutral
Wellread
Skeptical
Healthy
Lovable
Alert
Sympathetic
Caring
Driving
Moralistic
Rational
Colorful
Ascetic
Purposeful
Trusting
Enigmatic
Amusing
Compassionate
Resourceful
Smooth
Sane
Simple
Adventurous
Undemanding
Humorous
Hardworking
Perfectionist
Gracious
Intuitive
Serious

<b>Personality Traits:</b>	
Retiring	
Upright	
Restrained	
Generous	
Charming	
Unambitious	
Foreful	
Funloving	
Athletic	
Reliable	
Cooperative	
Humble	
Boyish	
Educated	
Efficient	
Uninhibited	
Crisp	
Determined	
Aggressive	
Stylish	
Physical	
Individualistic	
Disciplined	
Gallant	
Tolerant	
Observant	
Neat	
Realistic	
Dry	
Unceremonious	
Highspirited	
Helpful	

## Negative Traits

### **Negative Traits:**

generated by haroopad

<b>Negative Traits:</b>
Thoughtless
Hidebound
Angry
Imprudent
Vulnerable
Easily Discouraged
Mechanical
Obnoxious
Gullible
Tense
Fraudulent
Deceitful
Blunt
Cautious
Insecure
Strongwilled
Grim
Crafty
Scheming
Greedy
False
Incurious
Haughty
Brutal
Insincere
Agonizing
Brittle
Forgetful
Mealy-mouthed
Unconvincing
Resentful
Repentant
Extreme
Erratic

<b>Negative Traits:</b>
Irresponsible
Confused
Highhanded
Gloomy
Delicate
Dull
Wishful
Disturbing
Suspicious
Transparent
Meddlesome
Ridiculous
Extravagant
Hateful
Inconsiderate
Superstitious
Uncreative
Disloyal
Foolish
Paranoid
Irrational
Ignorant
Troublesome
Cowardly
Uncaring
Complaintive
Inert
Aimless
Dependent
Monstrous
Argumentative
Frightening
Offhand
Obvious

<b>Negative Traits:</b>
Intolerant
Indecisive
Negativistic
Calculating
Clumsy
Rowdy
Opinionated
Sordid
Unhealthy
Naive
Unreflective
Outrageous
Repressed
Weak
Unprincipled
Mannerless
Cruel
Expedient
Cold
Unrestrained
Procrastinating
Cynical
Critical
Dirty
Softheaded
Airy
Willful
Misguided
Hesitant
Criminal
Impractical
Narcissistic
Destructive
Ritualistic

<b>Negative Traits:</b>
Obsessive
Zany
Crude
Moody
Prejudiced
Fiery
Thievish
Fearful
Smallthinking
Overimaginative
Demanding
Lazy
Insulting
Devious
Unimpressive
Melancholic
Distractible
Ruined
Disobedient
Secretive
Tactless
Bizarre
Uncharitable
Escapist
Submissive
Sedentary
Stupid
Barbaric
Bewildered
Silly
Excitable
Faithless
Envious
Selfish

<b>Negative Traits:</b>
Anxious
Rigid
Miserly
Inhibited
Abrupt
Steely
Difficult
Colorless
Sloppy
Artificial
Neglectful
Singléminded
Passive
Onesided
Possessive
Treacherous
Graceless
Morbid
Perverse
Unimaginative
Shy
Mawkish
Disorderly
Predatory
Tasteless
Irritable
Insensitive
Disrespectful
Ungrateful
Uncooperative
Messy
Moneyminded
Unstable
Odd

<b>Negative Traits:</b>
Unappreciative
Selfindulgent
Flamboyant
Asocial
Assertive
Malicious
Unpolished
Shortsighted
Slow
Stiff
Oppressed
Scornful
Grand
Unselfcritical
Fickle
Desperate
Fixed
Weakwilled
Dishonest
Impatient
Impulsive
Sly
Charmless
Fawning
Timid
Disorganized
Narrow
Uncritical
Sadistic
Vague
Undisciplined
Unreliable
Arrogant
Presumptuous

<b>Negative Traits:</b>
Bland
Mannered
Mistaken
Discouraging
Narrowminded
Careless
Childish
Indulgent
Venomous
Powerhungry
Coarse
Unfriendly
Hostile
Opportunistic
Trendy
Petty
Unlovable
Regretful
Shallow
Superficial
Crazy
Onedimensional
Miserable
Unrealistic

## Hobbies

<b>Hobbies:</b>
Scrapbooking
Dance
Sculpting
Genealogy
Jewelry making
Foreign language learning
Crocheting
Playing Musical Instruments

**Hobbies:**

- Gaming (tabletop games and roleplaying games)
- Lapidary
- Embroidery
- Cleaning
- Cosplaying
- Yoyoing
- Puzzles
- Knitting
- Magic
- Cryptography
- Computer programming
- Drama
- Yoga
- Watching Movies
- Ping Pong
- RC cars
- Amateur radio
- Digital arts
- Web surfing
- Baton twirling
- Lego Building
- Sports
- Creative writing
- Cooking
- Leather crafting
- Gambling
- Quilting
- Origami
- Video gaming
- Coloring
- Juggling
- Sewing
- Homebrewing
- Drawing

**Hobbies:**

Painting  
Drinking Coffee  
Taxidermy  
StandUp Comedy

Woodworking

Wood carving

Worldbuilding

Writing

Soapmaking

Singing

Eating

Lacemaking

Reading

Pottery

Model Building

Handball

Kiteflying

Dowsing

Rock climbing

Nordic skating

Paint Ball

Graffiti

Polo

Motor sports

Skateboarding

Bird watching

Gardening

Beekeeping

Foraging

Urban exploration

Snowboarding

Ghost Hunting

Kiteboarding

Skydiving

**Hobbies:**

- Basketball
- Taekwondo
- Hooping
- Skiing
- Running
- Sailing
- Backpacking
- Shopping
- Baseball
- Surfing
- Mountain biking
- Sculling or Rowing
- Flying
- Sand castle building
- Net Ball
- Fishing
- Metal detecting
- Machining
- Driving
- Photography
- Roller skating
- Hunting
- Football
- Astronomy
- Mushroom Hunting or Mycology
- Parkour
- Tai Chi
- Cycling
- LARPing
- Board sports
- Swimming
- BASE jumping
- Air sports
- Archery

**Hobbies:**

Kayaking  
Scuba Diving  
Inline Skating  
Geocaching  
Slacklining  
Rugby

Vehicle restoration

Shooting

Skating

Water sports

Hiking

Jogging

Tennis

Go

Insect collecting

Coin collecting

Skateboarding

Element collecting

Jukskei

Leaf collecting and pressing

Paintball

Baton Twirling

Lincoln Douglas Debate

Billiards

Judo

Fishing

Triathlon

Field Hockey

League of Legends

Chess

Auto racing

Card collecting

Radiocontrolled car racing (hobby grade)

Audiophilia

**Hobbies:**

Debate  
Birdwatching  
Cricket  
Surfing  
Cricket (Indoor)  
Touch football  
Rugby league football  
Baseball  
Basketball  
Cheerleading  
Astrology  
Tour skating  
Roller Derby  
Shooting sport  
Marbles  
Swimming  
People watching  
Racquetball  
Microscopy  
Boxing  
Traveling  
Footbag  
Amateur geology  
Videophilia (Home theater)  
World of warcraft  
Model aircraft making and flying  
Amateur astronomy  
Gaming  
Slot car racing  
Disc golf  
Gongoozling  
Seashell collecting  
Trainspotting  
Bridge

**Hobbies:**

- Color Guard
- Climbing
- Vintage Books
- Handball
- Bus spotting
- Cycling
- Table football
- Vintage cars
- Mineral collecting
- Golfing
- Aircraft spotting
- Stone collecting
- Badminton
- Jugger
- Football
- Cubing
- Shortwave listening
- College football
- Fossil hunting
- Exhibition Drill
- Reading
- Poker
- Rock stacking
- Pole dancing
- Weightlifting
- Curling
- Kart racing
- Gymnastics
- Volleyball
- Modelling
- Dancing
- Vintage clothing
- Ice hockey
- Animal showing

**Hobbies:**

Metal detecting

Book collecting

Auto Racing

Antiquing

Meteorology

Record collecting

Fencing

Pigeon racing

Equestrianism

Figure skating

Table tennis

Dog sport

Target shooting

Art collecting

Dota 2

Herping

Archery

Association football (Soccer)

Bowling

Speed skating

Squash

Boxing

Martial arts

American football

Australian Football League

Volleyball

Movie collecting

Antiquities

Geocaching

Deltiology (Postcard collecting)

Auto audiophilia

Programming

Flower collecting and pressing

Airsoft

**Hobbies:**

- Darts
- Seaglass collecting
- Stamp collecting

**Habits****Habits:**

High pitch/low raspy voice depending on mood

Smacking gum

Toying with objects before them

Tapping foot

Putting their feet on a desk or table

Rubbing eyes/chin

Speech patterns

Squinting

Glancing at watch

Clicking or tapping teeth with a nail

Talking with a full mouth

Shrugging

Playing with their glasses (spectacles) either pushing them up their nose or taking them off and twirling them by one of the arms.

Chewing on their pencil

Chewing bottom lip

Humming

Gesturing a lot when speaking

Picking at nail polish

Stuttering

Applying Chap Stick to lips

Potty mouth

Twirling a ring around finger

Cross/uncross legs

Jiggling leg up and down

Shifts in their seat when nervous

Sniffing

Slurping

Raising eyebrows as they speak

**Habits:**

Pointing  
 Slouching  
 Cracking knuckles (cliche)  
 Jaw clenching/jaw muscle jerking (mainly a male attribute)  
 Twirling hair  
 Pulling at bottom lip  
 Snapping fingers  
 Picking at teeth  
 Rubbing hands together  
 Yawning  
 Winking  
 Running fingers through hair  
 Excessive eye blinking  
 Pinching skin  
 Picking at facial hair—eyebrows/eye lashes/moustache  
 Licking lips  
 Whistling  
 Constantly apologizing  
 Constantly checking cell phone for messages (text or voice)  
 Eye-rolling  
 Saluting  
 Biting fingernails  
 Slapping people on the back  
 Constantly touching up makeup using a compact mirror  
 Sliding the tip of one's index fingernail up and down along the side of the thumb beside it  
 Nervous cough  
 Burping

## Main Fantasy Character Classes

### Fighter Class

Fighter Class	Description
The Fighter Classes	The fighter is basically, the strong person with heavy armor, a large melee weapon, and possibly a shield or second melee weapon.
The Barbarian:	(Also known as a Berserker) focused more on damage than defense. They have higher defense than other warriors.

generated by haroopad

Fighter Class	Description
The Knight:	An experienced fighter with better armor and a lot of times he is on an armored horse.
The Swashbuckler:	A fighter that is lightweight and sometimes has little or no armor. They have great agility, cunning, and technical skills.
The Paladin:	(Also known as a Crusader, Templar, and Inquisitor) Light based knight with Healer abilities most of the time. Uses White Magic to gain higher defense.
The Dark Knight:	(Antipaladin and Death Knight.) Is the opposite of a Paladin. Their dark magic deal high amounts of damage and can lower stats of the foe.
The Dragon Knight:	(Also known as a Dragoon (Dragon rider)) Their armor is normally in a dragon style. They rarely fly, but commonly jump high and have fire breath. They can deal greater damage to dragons and most of the time have a dragon as a pet.
The Samurai:	Normally have less defense. They have Ki attacks, speed, great damage, and mobility. They are experts with swords and are known to be able to deflect bullets with their blade. It has been proven that they were able to.
Warlord:	(Also known as a General, Marshal, Commander, and Tactician) Tactical master. He can hold his own in frontline combat and gives allies around him bonuses for their stats.
Hero:	(Also known as a Lord or Protagonist) Good all around. They gain magic and unique skills. They always use a sword and a shield. It is possible that they will have no magic or shield, but that makes a really boring hero.

## Magician Class

Magician Class	Description
The Magician Classes:	(Also known as a Mage, Sorcerer, Wizard, Warlock, Witch, Magus, Magician, Sage, and Magi.) The magician uses magic against foes. There is a massive variety of different magic and one can make up an infinite number of magic users, but here are the basic magicians.
The Inherent Gift Magician:	(Known as a Sorcerer) Born with magic and do not need to study it. They normally do not know many spells unless they study.
The Theurgist Magician:	(Known as a Warlock) The magician gains magic by making a contract with a spirit.
The Summoner Magician:	(known as a Conjurer) A high level Theurgist magician. They can summon beings to do their bidding
The Vancian Magician:	(Know as a Wizard) These magicians study hard and learn magic.
The Red Mage:	They are open minded and study a variety of magic.
The Blue Mage:	(known as Mime or Mimic) They learn magic by fighting and can absorb magic or learn it by seeing their enemies use magic abilities.

<b>Magician Class</b>	<b>Description</b>
The Necromantic Magician:	They use the power of the dead, blood, and death energy. They are normally bad, but not all dark users are evil.
The Illusionist Magician:	They are only able to cast illusions.
The Nature Magician:	Have power over nature. Ex: Controlling vines, fire, water, throwing rocks, and creatures.
The Elemental Magician:	They can only control the elements. Fire, water, wind, etc.
The Druid Magician:	The can do anything related to nature. Turn into animals, trees, use fire with help of spirits, etc.
The Shamanic Magician:	Summons and makes deals with spirits.
The Elemental Magician:	They can control the elements. Plants, water, ice, fire, etc. Can be born with their power or a pack with a spirit or demon.

## Rogue Class

<b>Rogue Class</b>	<b>Description</b>
The Rogue Classes:	(Also known as a thief, ninja, assassin, shadow, pirate, scout, and gambler.). Rogues are normally thieves or treasure hunters. They are good at lockpicking, traps, disarming, sneak attacks, and attacking from the behind.
The Thief:	Ability to steal items from enemies or others. Normally the thief can steal rare items.
The Assassin:	Very stealthy and has many fighting abilities. The assassin normally poisons the enemy to easily take them down.
The Gambler:	Has some magic that relies on chance. They fight with cards or other gambling game items.
The Ninja:	Have a large range of skills. Vanishing with smoke and has most rogue like class skills. They normally hide their face in the ninja outfit, throw items like shurikens (Normally good with any weapon), and are crazy fast. The sometimes can use special powers with chakra.
The Shadow:	Ninja like rogues that hide in shadows or can hide in a shadow dimension they create.
The Pirate:	A rogue that can switch from sword and pistol. They are not normally strong, but captains normally are very good at sword play, stealing, and tricks or riddles. For example, hiding their treasure, making traps, and anything to protect their belongings. They are not always at sea, but normally have some kind of transportation. Sky pirates, water pirates, sand boat pirates, etc.
The Scout:	They move from place to place quickly with superior sensory and information gathering skills. They are normally not noticed and if they are then they are killed. Not many scouts are good at combat.

generated by haroopad

## Cleric Class

Cleric Class	Description	
The Cleric Classes:	(Also known as a medic or healer.) Unlike magicians their power comes from worshiping some kind of god.	classes, the Clerics usually draw their powers from Faith (god). They normally need to follow and practice their belief to use their power or it has a chance of backfiring on them.
The Priest:	(Known as Healer and White Mage) They are able to fight off certain types of enemies like demons or undead beings. They normally have healing powers as well.	
The Battle Priest:	They carry blessed weapons that can cause great damage to enemies. They normally fight in close combat.	
The Witch Doctor:	Their power comes from nature culture worshiping.	
The Templar:	They are a jack of all trades, but weaker than most classes and normally handle the church dirty work. Covering things up.	
The Caster:	This character is usually female and the heroine. They will be in the party because no one else can use magic. They are normally weak all around, but progress overtime.	

## Ranger Class

Ranger Class	Description
The Ranger Classes:	(Also known as a ranger, woodsmen, and hunter.) Rangers are good at archery, but can use a sword if they have to. They have skills to survive in the wilderness and can sometimes use nature magic.
The Sniper Ranger:	This type of ranger is only an archer, but does higher damage with projectiles for this reason. Their projectiles may cause status changes to their enemy. EX: Slow them down, poison them, lower their defense, etc. They are in the back of the troops because they are horrible at close combat.
The Bow and Blade Ranger:	This type of ranger uses weapons with a blade. They are good at close combat and are okay with a bow.
The Beastmaster Ranger:	This ranger can permanently take control of animals and have them destroy the enemy while they heal their allies and support with long range weapons.
The Dual Wielding Ranger:	Uses two melee weapons.

Ranger Class	Description
The Trapper Ranger:	Good at laying various traps and making it so the enemies can easily be ambushed.
The Magical Ranger:	Uses enchanted arrows. They can freeze the enemy, slow them down, or trap them in a net. Almost any kind of element.

## Rarer Class

Rarer Class	Description
Rarer Class Archetypes:	This class depends on the world. These classes have a science fiction or medieval setting thrown around in them.
Magic Knight:	Other Names: (Known as Hexblade, Spellblade, Rune Knight, Eldritch Knight.) The Magic Knight is a hybrid combat magician. Normally weaker at magic than a mage and not as good in combat as a fighter.
The Bard:	They use songs for strengthening allies, weakening enemies, status effects, and cause damage. Bards are good at diplomacy compared to most classes.
The Dancer:	A variation of the Bard, but dances instead. Using dance styles to cut up the enemy and other effects.
The Monk:	(Known as Black Belt and Martial Artist) The Monk uses their bare fists to fight or use only martial art weapons like nunchucks and staffs. They often use Ki Attacks and charge up attacks.
The Engineer:	(known as Tinker, Machinist, and Gadgeteer.) Engineers rely on technology. They are seen in most genre related to science fiction. For example, steampunk, cyberpunk, and clockpunk. They have guns and bombs as weapons. They sometimes employ stationary and mobile machines on the battlefield.
The Alchemist:	Chemist. An Alchemist combines items, magic, and science to make bombs, potions, and sometimes changing the structure of objects or creature around them. They often they use their own body in experiments.
The Psychic:	(Known ad Psion and Mentalist) Psychics use telepathy and psychokinesis to attack the enemies mind or to deal damage to his body. It differs from magic because it is done with the mind or brain. Magic just comes out of no where. Poof! Magic.
The Gunslinger:	They use a variety of guns most of the time. They normally just involve guns to be honest. Most of the time very skilled at them compared to engineers.

## Chapter 18: Upgrading, Export Project, and Making Libraries

You can get the sample [\[Chapter\\_18\\_export\\_sample.zip\]](#) in the lesson directory.

### Upgrading:

One must follow the Solarus [migration guide](#) to learn how to upgrade to a new version of Solarus. This book is for Solarus 1.5.x. It is possible that this book will be forked for 1.6+

### Export Project:

Remember to make a backup of your game before exporting.

You need to copy the Solarus Engine into a folder of your choosing, copy your quest **data** folder into that folder, and delete files you do not need.

## Deleting Files

Not all the **.dll** files are needed when exporting your game. You can delete the Solarus editor files because some of them are quite large, but you might need them for the quest launcher or if possible memory leaks happen.

Delete Files & Folders	File Size
Folder assets	50+ MB
Folder platforms	1.3 MB
icudt52.dll	23.5 MB
icuin52.dll	3.3 MB
icuuc52.dll	2.0 MB
Qt5Core.dll	4.8 MB
Qt5Gui.dll	4.6 MB
Qt5Widgets.dll	5.9 MB
solarus_editor_fr.qm	100+ KB
solarus-quest-editor.exe	4.3 MB
solarus.exe	450 + KB

Files Needed
libsolarus-gui.dll
SDL2_image.dll
libfreetype-6.dll
libstdc++-6.dll
SDL2_ttf.dll
libgcc_s_dw2-1.dll
libvorbis.dll
libmodplug-1.dll
libvorbisfile.dll
solarus_fr.qm
libogg.dll
libwinpthread-1.dll
solarus-run.exe
libphysfs.dll
lua51.dll
wrap_oal.dll

<b>Files Needed</b>
libpng16-16.dll
OpenAL32.dll
zlib1.dll
libsolarus.dll
SDL2.dll

## Setting up Data

The inside of your folder should look like this now.

You can leave your `[data]` folder out like that or you can zip it `[data.zip]`. Do not encrypt it because it will not work when activating the `[solarus-run.exe]`. You can rename the zip to `[data.solarus]`.

If you use a Linux operating system, then there might be an issue when zipping the file. The game will not play and you will get an `error.txt`.

The reason is that Linux creates certain attributes to files when they are created. I even tried using Microsoft Windows programs (7Zip) with Wine and it did not work.

I zipped it and renamed it in Windows 7 and everything worked fine. It even worked in Linux using Wine!

## Packaging or Distributing

You can zip the folder you made or use **Inno Setup**. You must provide a link to the **Solarus Engine Source code** and your game code because any code made with GPL code becomes GPL. One might want to link to the **Solarus Website** as well.

### Inno Setup

#### Step 1:

Select create a new script and press OK.

#### Step 2:

Press next.

#### Step 3:

Application Information	Description
Application name	Name of your program.
Application version	Current version of your project.
Application publisher	Your company or business name.
Application website	<a href="http://www.YourWebsite.com">www.YourWebsite.com</a>

**Step 4:**

Add the folder destination and name of your folder.

**Step 5:**

Main executable is `Solarus_run.exe`.

Other files are the `.dll` and `data` folder.

**Step 6:**

Add the name of your application start folder.

**Step 7:**

Add your text information.

- License
- Information before install
- Information after install

**Step 8:**

Select the languages you want.

**Step 9:**

Compiler Settings	Description
Filename output	The name of your installer file.
Custom Icon	Custom icon image for your installer.
Setup Password	Password for your installer.

**Step 10:**

Next

**Step 11:**

Next

**Step 12:**

generated by haroopad

Yes or no for compiling your script. Remember to [\[File > save\]](#).

### Step 13:

Edit the script if you want.

### Step 14:

You can always compile your script from [\[Build > Compile\]](#).

### Step 15:

Your installer will show up in a folder called [\[output\]](#).

## Making Libraries

The main purpose of a library is to make functions to achieve a task and/or to shorten code.

### Sample

You can grab the sample in [\[Lessons > Chapter\\_18\\_library\\_sample.zip\]](#). All the files you need are in it.

### Documentation

Documentation can help people understand what the functions in your library do and how to use them. Markdown is a format that is becoming very popular and it can help you make documentation. I recommend a markdown reader/writer like [Haroopad](#).

You can get the entity lib sample documentation in the data folder of the sample

[\[Chapter\\_18\\_library\\_sample > data > entity\\_lib\\_documentation.md\]](#).

### Basic Markdown Formatting

You can manually type the format or you can insert it with Haroopad's insert menu.

### Images:

![text to show if image is no longer exists online] (URL or link to the image)

EX:

![Fairyolica World Progress] ([http://s33.postimg.org/5vwatpy6n/Fairyolica\\_World\\_tiles.png](http://s33.postimg.org/5vwatpy6n/Fairyolica_World_tiles.png))

### Bold:

You put a double Asterisk\*\* around the text.

EX:

\*\*License:\*\*

### Headers:

Large Title Header text:

You put the Octothorp(#echo) or more commonly known as a hashtag(#echo) before the text.

EX:

#Outside/Exterior

You can have different sized headers too. Just got to add more hashtags.

EX:

###Outside/Exterior

## Table of Contents:

You can just insert with haroopad. Insert > bottom of the list > table of contents

## Embed Links:

[Go to Google] (<https://www.google.com>)

## Bullets:

Dash + space

-[space here] [text]

EX:

- This is a bullet

## Table & Nextline:

You can make a table with the vertical line "|" character. It is located by the brackets. Press shift+|

1. The first part of making a table is to set a title or category.

|Category|

2. The second part is the spacing. This requires some colons and minus signs / dashes.

Centered

|:---:|

Left

|:---|

Right

|---:|

3. The last set is to add a section.

|Section|

4. Next line in a table. It can make a messy situation look nice. Use <br> for a next line.

Line\_1: <br> Line\_2: <br> Line\_3:

EX:

|Category\_1|Category\_2|

```
| :---:|:---|
|Section| Line_1: <br> Line_2: <br> Line_3:
```

## Task List:

dash + brackets + item

An “x” in the brackets will check the item.

- [ ] Mercury
- [x] Venus
- [x] Earth
- [x] Mars
- [ ] Jupiter
- [ ] Saturn
- [ ] Uranus
- [ ] Neptune

## Types of Libraries

Solarus has certain name types for different parts. You could mix different name types in a single Lua file.

Names of Solarus userdata Lua types. These are used with the function:

```
sol.main.get_metatable("type_name")
```

<b>type_name (string)</b>
“game”
“map”
“item”
“surface”
“text_surface”
“sprite”
“timer”
“movement”
“straight_movement”
“target_movement”
“random_movement”
“path_movement”
“random_path_movement”
“path_finding_movement”
“circle_movement”
“jump_movement”
“pixel_movement”

<b>type_name (string)</b>
"hero"
"dynamic_tile"
"teletransporter"
"destination"
"pickable"
"destructible"
"carried_object"
"chest"
"shop_treasure"
"enemy"
"npc"
"block"
"jumper"
"switch"
"sensor"
"separator"
"wall"
"crystal"
"crystal_block"
"stream"
"door"
"stairs"
"bomb"
"explosion"
"fire"
"arrow"
"hookshot"
"boomerang"
"camera"
"custom_entity"

**Example:**

In a Lua file, declare the userdata type. I will use `custom_entity` in this example.

```
local metatable_entity = sol.main.get_metatable("custom_entity")
```

Make a function in that file and require this Lua file in [main.lua](#).

```
function metatable_entity:right()
--code here
end
```

You can declare this function in a custom entity script like this:

```
entity:right()
```

I will go over this in more detail in the entity library section.

## Entity Library Sample

The entity lib sample has movements related to custom entities. Adding the lib is quite simple. All one has to do is put the following line at the top of [main.lua](#) and place it in the script directory in a folder called lib to be organized.

```
require("scripts/lib/entity_lib.lua")
```

Here is the sample that I will explain.

```
--This is a library for custom entity shortcuts

local metatable_entity = sol.main.get_metatable("custom_entity")

-----
--Straight Method
-----
local move_straight = sol.movement.create("straight")

--Right
function metatable_entity:right()
    move_straight:set_angle(0)
    move_straight:start(self)
    self:set_direction(0)
    self:get_sprite():set_animation("walking")
end

--Up
function metatable_entity:up()
    move_straight:set_angle(math.pi / 2)
    move_straight:start(self)
    self:set_direction(1)
    self:get_sprite():set_animation("walking")
end

--Left
function metatable_entity:left()
    move_straight:set_angle(math.pi)
    move_straight:start(self)
    self:set_direction(2)
    self:get_sprite():set_animation("walking")
end

--Down
function metatable_entity:down()
    move_straight:set_angle(3 * math.pi / 2)
    move_straight:start(self)
    self:set_direction(3)
    self:get_sprite():set_animation("walking")
```

```

end

--Stop
function metatable_entity:stop()
    move_straight:stop()
    self:get_sprite():set_animation("stopped")
end

--Speed
function metatable_entity:speed(value)
    move_straight:set_speed(value)
end

```

## Library Setup

This library is related to the custom entity, so the metatable variable is assigned to it. This variable will be added to functions that are used in the library.

```
local metatable_entity = sol.main.get_metatable("custom_entity")
```

Adding the metatable to a function called `right()` will look like this:

```

--Right
function metatable_entity:right()
end

```

Calling the function is very easy after requiring it from `main.lua`.

### Example:

```
require("scripts/lib/entity_lib.lua")
```

## Self Function

A common function that is used when making libraries is `self` or at least I use it a lot. This refers to the entity itself.

```

local move_straight = sol.movement.create("straight")

--Right
function metatable_entity:right()
    move_straight:set_angle(0)
    move_straight:start(self)
    self:set_direction(0)
    self:get_sprite():set_animation("walking")
end

```

The function `entity:right()` would normally be called with another function or a timer in the custom entity script. For example,

```

function entity:on_created()
    entity:right()
end

```

or

```

function sol.main:on_key_pressed(key)
    if key == "1" and menu == false then
        entity:right()
    end

```

generated by haroopad

```
end
end
```

## Special Entity

Other times one might want to refer to a certain or special entity name instead of `self`. One simply needs to get the entity from the map like this:

```
local player2 = map:get_entity("player2")
```

### Example:

```
local move_straight = sol.movement.create("straight")

function metatable_entity:right()

local map = self:get_map()
local player2 = map:get_entity("player2")

move_straight:set_angle(0)
move_straight:start(player2)
player2:set_direction(0)
player2:get_sprite():set_animation("walking")
end
```

## Function Arguments

One can have arguments in the function too. In the example below the argument `value` is put in the function parenthesis or parameter.

```
--Speed
function metatable_entity:speed(value)
    move_straight:set_speed(value)
end
```

It would be called like this:

```
function entity:on_created()
    entity:speed(80)
end
```

Remember the arguments do not only have to be a number value. They can be a string, table, boolean(true/false), etc.

## Function Tables

If your function has a lot of options or you just want to make things clearer, then the use of tables for your functions might be a better setup for your library. You can change the order of the items in the table and use a name to assign them. Also, you can leave off items if you do not want to use them. Thank llamazing on the Solarus forum for this example.

```
function test(info)
    if info.a == 5 then
        print("Letter A is",info.a)
    end

    if info.b == "hey" then
        print ("Solarus welcomes you.")
    end
```

```

if info.c == true then
    local test = 50
    print(test + 10)
end
end

test{b="hey", a=5, c=true}

```

## Player 2 Movement

This script uses the keys i, l, j, & k to make an entity walk around. It is a very simple script using the functions in the library sample. The script is in an entity model called player 2 and the entity name is player2.

If have forgotten how to setup custom entities, then I have added instructions after the script below.

```

local entity = ...
local game = entity:get_game()
local map = entity:get_map()
local hero = map:get_hero()
local sprite
menu = false

local camera = map:get_camera()
camera:start_tracking(entity)

---Create sprite
function entity:on_created()
    sprite = entity:create_sprite("hero/tunic1"):set_animation("stopped")
    entity:set_can_traverse("hero", false)
    entity:speed(80)
end

--Key press function
function sol.main:on_key_pressed(key)

--right
    if key == "l" and menu == false then
        entity:right()
    end

--up
    if key == "i" and menu == false then
        entity:up()
    end

--left
    if key == "j" and menu == false then
        entity:left()
    end

--down
    if key == "k" and menu == false then
        entity:down()
    end
end

function sol.main:on_key_released(key)
    --right
    if key == "l" then
        entity:stop()
    end

```

```
--up
if key == "i" then
    entity:stop()
end

--left
if key == "j" then
    entity:stop()
end

--down
if key == "k" then
    entity:stop()
end
end
```

**Add entity:****Create entity Model:****Name Model:****Add entity script and name entity:**

The entity name must be **player2** because of the example in the library sample.

**Open Script****Paste player 2 script:**

There is no real reason to break down this script because it is very simple.

## Debugging

Lua has functions for debugging. Solarus takes care of error reports. Solarus will let you know where and what the problem is, but it does not allow you to make your own reports or warnings.

The following function uses the debug library in Lua.

```
local metatable_entity = sol.main.get_metatable("custom_entity")

function Error()
    local level_1 = debug.getinfo(1) --Error() is level 1
    local level_2 = debug.getinfo(2) --Name of function being called from is level 2
    local level_3 = debug.getinfo(3) -- Return from where ever the function was declared is level
    print ("-----")
    print(level_3.short_src.." => "..level_1.name.." was called by function [".. level_2.name.."]
    print ("-----")
end
```

It can be declared where you want the error warning to occur like this:

```
function metatable_entity:example()
    Error()
end
```

Let us say you declare this function in an entity script called `follower.lua` at line 1.

```
entity:example()
```

The output would be:

```
[string "entities/follower.lua"] => Error was called by function [example] at line 1
```

I will explain this in more detail in the sections below.

## Debug Levels

There are different function levels that `debug.getinfo` can detect. One should have at least 3 levels for debugging for libraries.

### Level 1:

`Debug.getinfo(1)` is the function that the debug script is in. In this case it is in a function called `Error()`.

```
local level_1 = debug.getinfo(1) --Error() is level 1
```

### Level 2:

`debug.getinfo(2)` is the function that `Error()` is declared in. It was called in a function named `example()` or `function metatable_entity:example()`.

```
local level_2 = debug.getinfo(2) --Name of function being called from is level 2
```

### Level 3:

`debug.getinfo(3)` is where the `example()` function was declared at. In this case it was in `follower.lua`.

```
local level_3 = debug.getinfo(3) -- Return from where ever the function was
```

## Lua Debug Library function

I will not go over all the debug library functions, but will go over the ones I use. You can check this [link](#) for a list of all the functions.

Debug Functions	Description
short_src	A short version of source for up to 60 characters, it can be useful for error messages.
name	A name for the function.

generated by haroopad

Debug Functions	Description
currentline	The line where the function was declared at.

I mentioned the levels above, so let me explain how to use the debug function with the levels.

`level_3.short_src` will give you `[string "entities/follower.lua"]`. The location where the function was declared.

`level_1.name` will give you the name of the function where the debug code is in and that is `Error()`. "`Error` was called by.....".

`level_2.name` will be the function name `example()`, "was called by function [`example`] at".

`level_3.currentline` is the line number where the function `example` was declared at. In this case it was declared at `line 1`.

```
print(level_3.short_src.." => "..level_1.name.. " was called by function [".. level_2.name.. "
```

```
[string "entities/follower.lua"] => Error was called by function [example] at line 1
```

## Debug Traceback

The function `debug.traceback()` returns a string of information.

```
function metatable_entity:right()
print(debug.traceback())

local map = self:get_map()
local player2 = map:get_entity("player2")
move_straight:set_angle(0)
move_straight:start(player2)
player2:set_direction(0)
player2:get_sprite():set_animation("walking")
end
```

```
stack traceback:
[string "scripts/lib/entity_lib.lua"]:18: in function 'right'
[string "entities/player2.lua"]:35: in function <[string "entities/player2.lua"]:31>
```

## Pcall

The `pcall` function gives you an error if the function fails.

```
function testfunction()
    t = t/nil
end

if pcall(testfunction) then
    print("Success")
else
    print("Fail")
end
```

```
Fail
```

There is another function call `xpcall` that I will not cover and you can get information about it at [tutorial point](#).

## Chapter 19: Credits

Credits so far...

### Graphics

- Diarandor (Most of his art is the sample quest) You can find a license in the sample quest. Most or all of his work is under CC-BY-SA 4.0
- Zane Kukta (My work separate from the forest and dessert tileset is CC0) See sample quest license or refer to this [issue](#) if the license file has not been updated yet.
- Christopho - His work in the sample quest and adapted work.
- Neovyse - The Solarus logo.
- Vlag - His adapted work on the hud.
- Andrew Tyler - The minecraftia font.
- Bertram - for his adapted work.
- Yusuke Kamiyamane - For the Solarus editor icons. Yusuke Kamiyamane:  
<http://p.yusukekamiyamane.com/>
- gwes - The key, heart, and skull shield.
- Reemax (Tuomo Untinen) - His skeleton scale to 32x32 and walking animation of Artisticdude's sprite.
- Artisticdude - His skeleton sprite
- interdimensional\_ <http://interdimensional.space/> - His beast sprite for the orc.
- titi\_son - For the mountain background image
- Maxs - Animated Solarus logo
- Blarumyrran - 32x32 treasure chests
- Daniel Cook - Chest by Daniel Cook ([Lostgarden.com](http://lostgarden.com))
- Benjamin Pickhardt - Daniel Cook chest modification of Benjamin Pickhardt ([hackcraft.de](http://hackcraft.de))
- Hypnosis
- Zabin
- arikel
- Viktor Hahn ([Viktor.Hahn@web.de](mailto:Viktor.Hahn@web.de))
- AlexCons
- Calciumtrice - Farming Tool Icons by Calciumtrice
- Stephen Challener (Redshrike) - Stephen Challener (Redshrike). Commissioned by OpenGameArt.org (<http://opengameart.org>)
- Casper Nilsson
- Daniel Eddeland
- Johann CHARLOT
- Skyler Robert Colladay
- Lanea Zimmerman (AKA Sharm)
- Stephen Challener (AKA Redshrike)
- Charles Sanchez (AKA CharlesGabriel)
- Manuel Riecke (AKA MrBeast)
- Daniel Armstrong (AKA HughSpectrum)

### Audio

- Diarandor (Most of the sounds and music in the sample quest.) CC-BY-SA 4.0

- Zane Kukta (Any sound I added is CC0 like the timer sounds or the license I stated in the sound directory)
- Eduardo Dueñas for his music pieces musics/village.ogg and sounds/secret.ogg
- Eric Matyas, his music from soundimage was very useful in the creation of the chain quest.  
"Closing In On the Loot\_Looping"  
"Hypnotic Orient\_Looping"  
"Magical Getaway\_Looping"  
"Pride\_v002"  
"The 8 Bit Digger"  
"Young Heroes"  
by Eric Matyas  
[www.soundimage.org](http://www.soundimage.org)

## Scripting Credits

- Christopho (All of his scripts, sample quest, and making the Solarus Game Engine) GPLv3
- Zane Kukta (Most sample scripts)
- Diarandor (Some script snippets from his assistance on the forum and sample quest)
- Ilamazing (Some script snippets from his assistance on the forum)
- Max (Some script snippets from his assistance on the forum)

## Writing Credits

- Christopho (Solarus documentation edited snippets)
- Zefk (Zane Kukta) CC-BY-SA 3.0
- Loren J. Miller [The 36 Plots](#) CC-BY-SA 3.0
- Logan M: [Differences-between-visual-novel-eroge-gal-game-and-a-dating-sim](#) (CC-BY-SA 3.0)
- Wikipedia [List\\_of\\_genres](#), [video\\_gaming](#), [PuzzleRetrogaming](#), [Mode\\_7](#), [Sprite](#), [Pixel](#), [Artificial\\_intelligence](#)), [Beat\\_em\\_up](#), CPU, Spawning, CC-BY-SA 3.0

## Solarus Team

- Christopho
- Neovyse
- Diarandor
- Maxs
- Newlink
- Renku
- std::gregwar
- Binbin

## Special Thanks

- wrightmat, I learned a lot from your scripts.
- MetalZelda, your coding suggestions were very helpful.
- Ilamazing, your assistance on the forum really helped make this book.
- Max, thanks your help on the forum.
- Claire Moore (nate-devv), thanks for the Linux version of Solarus!

## Fairyolica World Credits

I used the Fairyolica world tileset image as a preview. It is under CC-BY-SA 3.0.

About:

generated by [hadoopad](#)

- 1.The bullet “•” means it is a person and the way they want to be credited.
- 2.This is organized by license type. CC0, CC-BY 3.0, CC-BY-SA 3.0.
- 3.CC0 is not required for crediting, but I added them anyway.
- 4.Go into the original graphics for more detailed credits for each folder with related links.
- 5.Yes, all these licenses are compatible with each other, but the end result is CC-BY-SA 3.0 and you must credit everyone if you use the Fariyolica.png. That means the CC0 people too.
- 6.I refined this list a little. 11-14-16 [November 14th, 2016]

## Credit

Copyright/Attribution Notice:

[CC0 Public Domain]

- Jetrel <http://www.frogatto.com/>
- Guido Bos
- Ogresbane
- Buch (Michele Bucelli) <http://opengameart.org/users/buch>
- ‘Hyptosis’ [Maybe a free copy fo your game]
- artwork by Matthew Weekes (<https://www.facebook.com/matwekpixel/>, <https://twitter.com/matwekpixel>), but copyright (abetusk).
- CDmir
- rubberduck
- “Russpuppy <http://russpuppy.com>” [would love to hear about your game]

[CC-BY 3.0 Credit by their terms]

- Stephen Challener and the Open Surge team ( <http://opensnc.sourceforge.net>), hosted by OpenGameArt.org
- Stephen Challener (Redshrike), hosted/commissioned by OpenGameArt.org
- Leonard Pabin
- Lionx\_Dagger
- Copyright 2008 Jordan Trudgett | <http://jordan.trudgett.com/>
- knunery
- Albert Manhattan - A redrawing of Surf’s Classical Ruin Tiles (<http://opengameart.org/content/classical-ruin-tiles>) at higher resolution.
- Ivan Voirol

- credit me as Buch (Michele Bucelli) and link back to my OGA profile page.  
<http://opengameart.org/users/buch>
- Stephen Challener and the Open Surge team ( <http://opensnc.sourceforge.net>), hosted by OpenGameArt.org
- Credit Buch for the original version of this set and link to his profile page. The portcullis are based on the iron bars in <http://opengameart.org/content/lpc-dungeon-elements>, by Sharm as contributed by William.Thompsonj.
- Animated Castle Doors by Tuomo Untinen

[CC-BY-SA 3.0 Credit by their terms and share artwork under the same terms]

- interdimensional\_ <http://interdimensional.space/>
- attribute this to Jerom. I just added some colors. Put Eiyeron or @Eiyeron.
- DawnBringer ([http://www.pixeljoint.com/forum/forum\\_posts.asp?TID=12795](http://www.pixeljoint.com/forum/forum_posts.asp?TID=12795))
- (PriorBlue [Marcus Ihde])
- “420 pixel art icons” from 7Souls (<http://7soul1.deviantart.com/?rnr=74632>),
- “32x32 Fantasy Tileset” from Jerom (<http://opengameart.org/content/32x32-fantasy-tileset>).
- “Crawl Tiles” from Dungeon Crawl Stone Soup (<https://code.google.com/p/crawl-tiles/>)
- Zefk (Zane Kukta) <http://zelzec-entertainment.weebly.com/>
- Lanea Zimmerman (AKA Sharm)