

Open Source for Video Games - A Shortlist of Game Engines

Authors: Johanna Pirker, Marco Bertini, Mathias Lux;

Affiliation: Graz University of Technology, University of Florence, University of Klagenfurt;

Editors: Mathias Lux and Marco Bertini

Published in: ACM SIGMM Records, Issue 3, 2020

URL: <https://records.sigmm.org/?open-source-item=open-source-for-video-games-a-shortlist-of-game-engines>

Open-source software is a relevant topic in video game development. Taking a look at the most frequently employed game engines for developing Android games [1] we can see that seven out of ten ranked engines are OSS. Over the last decade, more and more game studios and individual developers switched to open-source software. Oliver Franzke from Double Fine Productions described the phenomenon from his point of view at GDC Europe 2013 [2]. For him developing game engines from scratch is too costly, and re-using self-made game engines often involves too much repurposing as they were developed with one specific game in mind. Existing third-party game engines might not support all features needed, and requesting new features or APIs from the developers is a tedious effort. In contrast to that, an open-source game engine is typically stable, extendable, and - in the best case - has a lively community accepting patches and helping out problems. Best of all, if a bug in the game engine is found or a feature is needed, one can fix it and compile it instead of filing a bug report or feature request and hoping for the next release of the engine being in time for the release of the own project.

Epic Games was one of the first AAA game engine developers to publish the source code of their Unreal Engine [3] in 2015 to let game developers investigate critical parts and send in patches and file bug reports. However, they did not change their model to OSS. Amazon (with the Lumberyard engine), Unity, and Crytek (with the CryEngine) follow similar models, where the source code is available for developers, but no open-source license is used. Other prominent source code releases included the famous id Tech engines, which powered games like Doom and Quake, and the Serious Sam Engine from Crotech [4, 5]. In both cases, they were made available GPL license for historical and educational purposes and developers never released the most recent version of their engines.

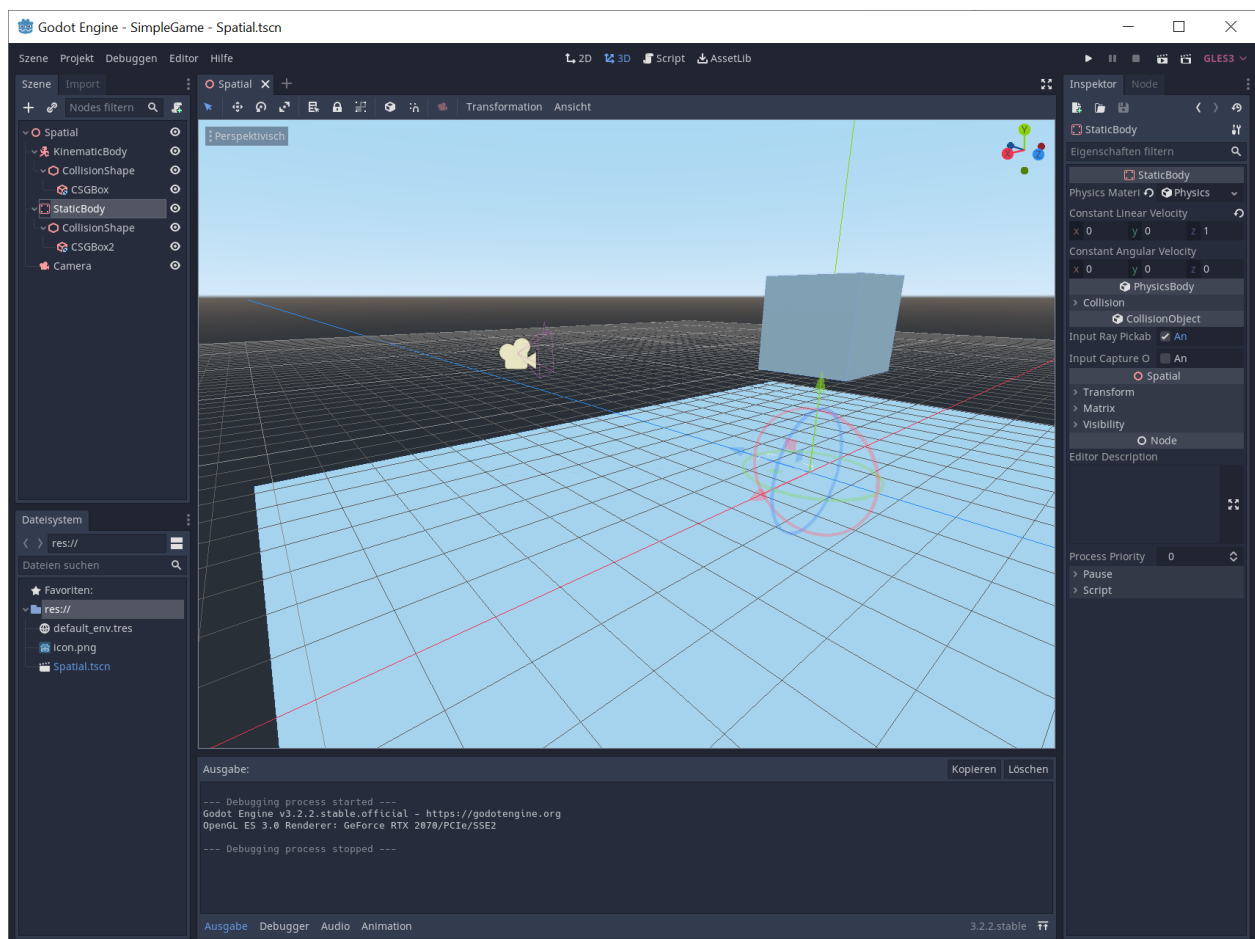
Open Source Game Engines

A game engine, often also referred to as a *game framework* as the lines are blurry there, covers the most basic aspects of video games, and tries to take many of the menial tasks away from game developers. They can be very basic, only offering a way to implement the game loop with additional support for timing and assets, or they can be fully-fledged with in-built high-level editors or even marketplaces to share, sell, and download content. Moreover, some game engines are genre-specific and focus on very particular game mechanics, e.g. non-linear storytelling, puzzles, or dungeon exploration, by providing specialized tools and workflows to make game creation in this particular genre easier. For most of them, platform independence is a big thing, so that a game can be written on a platform, typically a PC or a Mac, and deployed to many others, e.g. mobile phones, tablets, the web, etc. Especially among indie developers, open source engines provide a great opportunity. On the indie game hosting website

itch.io, 5 out of the top 15 used game engines are open-source engines [8] including Twine, Godot, Ren'Py, LÖVE, and libGDX. In the following sections, we will take a closer look at these engines.

Godot Engine

A popular choice for 2D as well as 3D game development for multiple platforms is the Godot Engine. It follows a scene based approach, where a game is based on a scene graph, or more precisely a tree, and nodes of the tree depend on each other. Godot comes with its own editor running on the Godot Engine. Language of choice has been GDScript for a long time, which is similar to Python. But lately, Microsoft has sponsored the integration of C# in the Godot Engine and there is an additional release with Mono included, supporting C# 8. Moreover, an extensive catalogue of community provided plugins adds to the in-built features with other programming languages, support for VR, and much more.



The screenshot shows a simple scene in Godot with a box, a plane, and a camera in the main editor. The scene manager is shown on the left hand side, just above the resource management tab.

Godot is available under the MIT License, also a common and permissive open source software license. Godot offers cross compilation and export to Windows, Mac OS X, Linux, Android, iOS, Nintendo Switch, HTML, and WebAssembly. With 28 MB of download for the Windows 10 executable, Godot has a small footprint. However, the packages for distribution on different platforms require extra

downloads per platform. The development of Godot has seen more than 30,000 commits by more than 1000 contributors on Github to date. The user and developer community is active throughout various channels including Discord, web forums, Facebook, Steam, and YouTube.

Godot is a little bit harder to learn compared to LÖVE as the GDScript language as the primary method of programming is something new, but it is similar enough to Python. Through the scene based approach many elementary tasks, e.g. controlling the game loop or loading assets, are done by the engine.

LÖVE

A low-level, simplistic, and elegant way of creating 2D games is provided by LÖVE [6]. Focusing on the implementation of the game loop, LÖVE provides a framework for creating games in Lua, that can be run on Windows, Mac OS X, Linux, Android, and iOS. Fully developed in C++ and C it has very small space requirements and takes less than 10MB for its Windows version. Lua as a programming language is a C-like, interpreted language, and easy to learn for people who know Java, C#, or C.

```
1  -- initialization, run once
2  function love.load()
3      jump_sfx = love.audio.newSource("sfx/jump.ogg", "static")
4      button = love.graphics.newImage("img/button.png")
5      button_sprite = {x=300, y=200, w=button:getWidth(), h=button:getHeight()}
6  end
7
8  -- run once each game loop iteration, for updating the game world
9  function love.update()
10     if love.mouse.isDown(1) then -- check if inside button
11         x = love.mouse.getX()
12         y = love.mouse.getY()
13         if x > button_sprite.x and x < button_sprite.x + button_sprite.w and
14             y > button_sprite.y and y < button_sprite.y + button_sprite.h then
15             jump_sfx:play() -- play sound
16         end
17     end
18 end
19
20 -- run once each game loop iteration, for creating output
21 function love.draw()
22     love.graphics.draw(button, button_sprite.x, button_sprite.y)
23 end
```

The above figure gives a simple example where a sound effect is played when a button is clicked. Asset loading is done in the function *love.load()*, the input analysis is done in *love.update()*, and the visual output is created in *love.draw()*. LÖVE does not offer higher-level data structures like sprites or an event system, where handlers for interaction can be registered. However, there are a lot of pre-defined Lua

data structures and additional libraries created from the community and of course one can build an own implementation of a sprite.

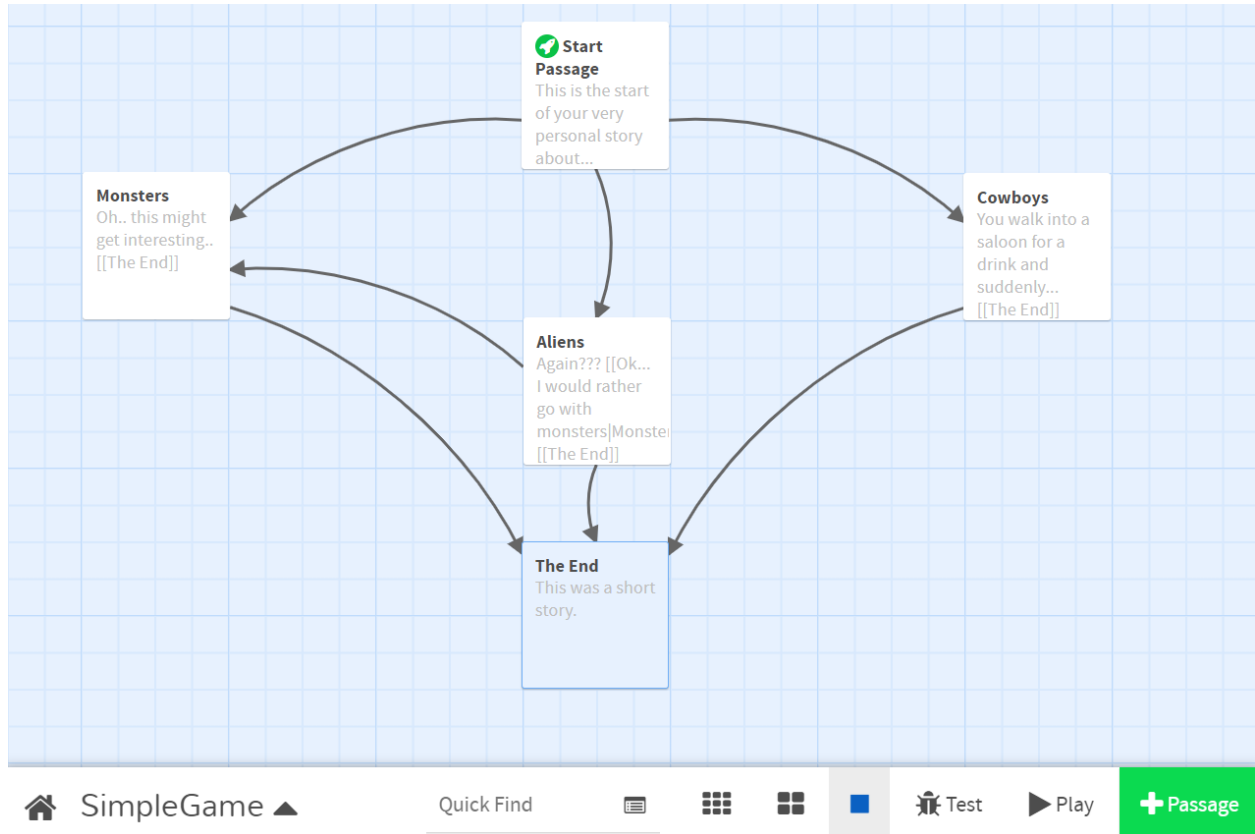
For installation, one only has to download the executables for the respective platform and run a *main.lua* file with the *love* executable. Alternatively, a game and all its code and assets can be packaged in a ZIP-file with the file ending *.love* and the *main.lua* file at its root.

LÖVE has a lively community of enthusiasts, which shows in the high number of commits (more than 3,600 to date) on Github [7] and the high frequency of forum posts. There are numerous tutorials provided by the community and a wiki is provided with API documentation and examples. LÖVE is provided under zlib license, which is very permissive and compatible with the GPL license. All in all, LÖVE provides an easy entry into low-level game programming and lets users keep their data structures and memory consumption under tight control. The downside is that LÖVE does not provide implementations for common data structures and event-based systems. However, it has a steep learning curve and a very small footprint, and as it is based on plain text Lua code, it's perfect for collaborative work on small projects using Git or any other versioning and collaboration system.

Twine

Twine is an open-source tool to create interactive stories and is hosted on GitHub [9]. While Twine v1 was written in Python, v2 is written in JavaScript; both are available under the GPL v3 license. The interactive stories are saved in a story format called *Harlowe*. The published games and interactive stories can be played in the web browser. To create such an experience no programming skills are needed. The barrier to use this tool is therefore very low and everyone can create experiences.

An important element of Twine, which can be found by many successful open-source products, is the extensive and complete user documentation and manual.



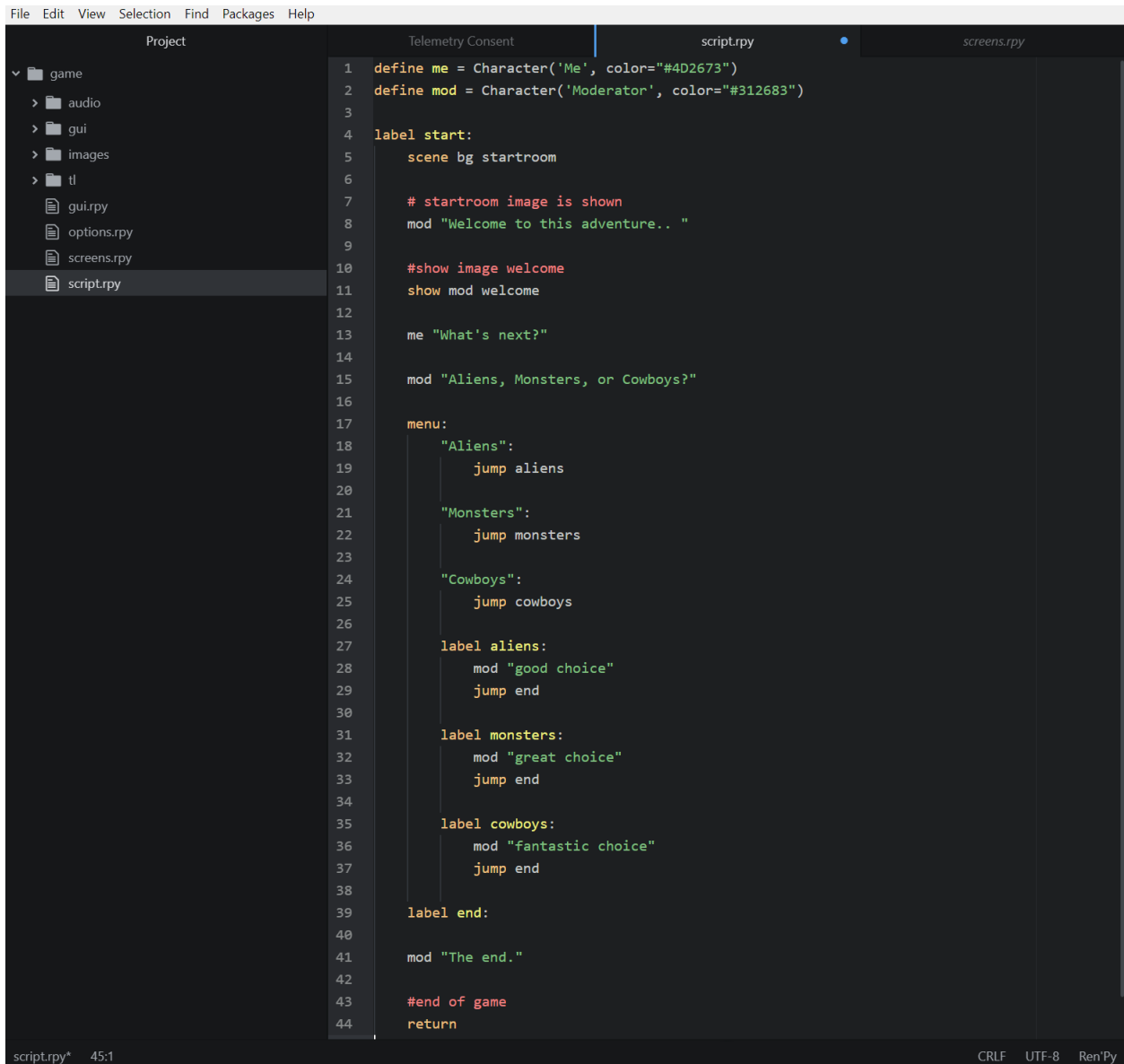
The screenshot shows a simple interactive story in the editor's "Passages View". The story is created directly in the web browser. To create a story, the developer creates different passages, which can be described as different parts of the story. These passages can be connected through links. By clicking on a link, the player is then redirected to a different passage. Often, this is used to generate a choice for the player.

Twine is a great tool to generate text-based adventures, non-linear stories, and narrative prototypes for games in a fast way and is especially a great resource to create first game projects without programming skills. While the underlying Harlowe format allows for some programming, options are limited, and debugging of complex code is tedious. In that sense Twine is limited in terms of games that can be made, but an extremely useful tool for the particular genre of narrative games. It is easy to learn and the published files (HTML 5) are easy to distribute and integrate.

Ren'Py

Ren'Py is a powerful open-source engine to create visual novels, which are interactive fiction games, often mixed media with text and images or even videos. Visual novels are created in the Ren'Py Launcher (available for Windows, Mac, and Linux). It uses the MIT license model. It is based on Python and is well supported by the community. In addition to PC and Web, Ren'Py allows building applications for Android and iOS. Ren'Py uses a Python-like scripting language. It allows the integration of multimedia assets such as graphics or audio elements. This also makes the build structures more

complex but also prevents, to some extent cracking attempts. Compared to Twine, the possibility to integrate multimedia objects makes it a powerful tool to create interactive visual experiences.



```
File Edit View Selection Find Packages Help
Project Telemetry Consent script.rpy screens.rpy
game
├── audio
├── gui
├── images
├── tl
├── gui.rpy
├── options.rpy
├── screens.rpy
└── script.rpy
1 define me = Character('Me', color="#4D2673")
2 define mod = Character('Moderator', color="#312683")
3
4 label start:
5     scene bg startroom
6
7     # startroom image is shown
8     mod "Welcome to this adventure.. "
9
10    #show image welcome
11    show mod welcome
12
13    me "What's next?"
14
15    mod "Aliens, Monsters, or Cowboys?"
16
17    menu:
18        "Aliens":
19            jump aliens
20
21        "Monsters":
22            jump monsters
23
24        "Cowboys":
25            jump cowboys
26
27        label aliens:
28            mod "good choice"
29            jump end
30
31        label monsters:
32            mod "great choice"
33            jump end
34
35        label cowboys:
36            mod "fantastic choice"
37            jump end
38
39    label end:
40
41    mod "The end."
42
43    #end of game
44    return
script.rpy* 45:1 CRLF UTF-8 Ren'Py
```

The screenshot shows a snippet of a very simple scene. In the first lines characters are defined. In the game dialogue, the names are shown in the defined color. The scene statement displays a background image; the show statement illustrates an image on the top. The images are located in the “images” folder. Ren’Py also supports player choices through the menu and jump statements. It also supports flags and if statements. This can be used, for instance, to store a player's choice and use this choice later for an in-game decision. Ren’Py also comes with a build distribution system and a testing and debugging system.

The project is hosted on Github [18] and the community is very active. The project has seen almost 10,000 commits from more than 100 contributors. It is very well documented and, thanks to a

well-written manual, easy to learn. Therefore, it is not surprising that the list of popular games developed in Ren'Py is long [11].

Honorable mentions

Picking out only four open source game engines is definitely not enough, as many others are well-known, useful, and adopted in the games industry too. libGDX [13] started out as a framework to make Android game development easier and now allows for platform independent game development in Java. Games can be developed and played on Windows, Linux and Mac OS using Java or Kotlin and they can be deployed to Android, iOS, and the web.

Construct [16] is a popular HTML5-based tool to create 2D games and uses visual programming features. It started as an open-source project (Construct Classic) created by students. With Construct 2 and Construct 3, the license model moved from open source to proprietary.

Scratch [14] and PocketCode [15] feature a visual programming language, where blocks are put together like Lego to build a program. Common programming elements like variables, flow control, loops, basic math, input, output, and sound are supported and small games and fast prototypes can be created easily. Having been developed with MINT education for children and teenagers in mind, the learning curve for the visual scripting language is steep and building a game in a short time is a rewarding experience.

Besides engines and frameworks, the open-source community provides a lot of tools for game development. One well known example is Blender [17]. While most people know Blender for its 3D modeling capabilities, it can also do video editing, compositing, and animation, and is an important tool for developers who are not able or willing to use commercial tools.

Conclusion

Open-source contributions have always been an important driver of the games industry. Also, the possibility of modding and editing games has been a starting point for many programmers and game developers. For example, the first known game developers in Peru did not have the opportunity to learn how to create games anywhere. So they started cracking and decompiling published games to acquire the required game programming expertise and created their first own games built on this knowledge [12]. Thus, it is not surprising that releasing open-source tools and open-source games have provided a great boost for the game development community. Source code releases have also been used by development studios to contribute back to the community, as can be seen on the example of Croteam's Serious Engine. But they have also been used to build a community around their own studio or the game. For example, Jon Manning describes in his GDC talk how *Night in the Woods* open-source tool Yarn, a dialogue system, helped to build a community for their game. This community further improved the game and the development process by helping to fix bugs and improving the tool [10]. Based on similar positive experiences, over time many game developers have published their source code.

A further problem arising with a lot of games is archival and preservation. Proprietary engines like Unity and Unreal run well at release time, but with time underlying APIs and operating systems change,

and games can no longer be played. In this case open-source engines and games help a lot as enthusiasts and people dedicated to preservation can bring old games back to speed easier.

The list of open-source game engines is still getting longer and many open-source game engines were developed to cover special niche genres such as text-based, 2D, or web-based-games. But as we can see from the example of Godot's success, the need for open-source tools for developing complex and large games for different platforms is huge. Summarizing, open-source products are important to empower game developers not only to create games, but also to help them to build a community, to be part of a community, and to learn how to develop games.

References

- [1] <https://www.appbrain.com/stats/libraries/tag/game-framework/android-game-frameworks>, last accessed 2020-09-22
- [2] <https://www.gdcvault.com/play/1019748/Broken-Age-Rethinking-a-Classic>
- [3] <https://docs.unrealengine.com/en-US/GettingStarted/DownloadingUnrealEngine/index.html>
- [4] https://en.wikipedia.org/wiki/Doom_engine
- [5] <http://www.croteam.com/serious-sam-source-code-released/>
- [6] <https://love2d.org>
- [7] <https://github.com/love2d/love>
- [8] <https://itch.io/game-development/engines/most-projects>
- [9] <https://github.com/klembot/twinejs>
- [10] <https://www.gdcvault.com/play/1024197/Making-Night-in-the-Woods>
- [11] https://en.wikipedia.org/wiki/List_of_Ren%27Py_games
- [12] <https://www.polygon.com/features/2014/2/10/5373586/mr-byte-indie-king-of-peru>
- [13] <https://libgdx.badlogicgames.com/>
- [14] <https://scratch.mit.edu/>
- [15] <https://share.catrob.at/pocketcode/>
- [16] <https://www.construct.net/>
- [17] <https://www.blender.org/>
- [18] <https://github.com/renpy/renpy>