# WEB BASED GAME ENGINE DESIGN

Lappeenranta–Lahti University of Technology LUT

Master's Programme in Software Engineering and Digital Transformation, Master's Thesis

2023

Mikko Mustonen

Examiners: Associate professor Jussi Kasurinen

D.Sc. (in tech) Jani Rönkkönen

ABSTRACT

Lappeenranta–Lahti University of Technology LUT

LUT School of Engineering Science

Software Engineering

Mikko Mustonen

**Web Based Game Engine Design**

Master's thesis

2023

43 pages, 1 figures, 0 tables and 0 appendices

Examiners: Associate professor Jussi Kasurinen and D.Sc. (in tech) Jani Rönkkönen

Keywords: Game engine, HTML5, web, design science

This thesis is a study about what is needed to develop a web-based game engine with some unique requirements of Seepia Playables. It goes through what the requirements for the engine are and what existing solutions there are and why the existing solutions don't fit the requirements. Components for building a new solution are explored. Where possible existing solutions are recommended, and their potential issues are described, and solutions for these issues are suggested. Some of the issues are caused by the limitation of the web platform and others are by the solutions itself. For the requirements that don't have existing solutions, one is designed and proposed.

TIIVISTELMÄ

Lappeenrannan–Lahden teknillinen yliopisto LUT

LUT Teknis-luonnontieteellinen

Tietotekniikka

Mikko Mustonen

**Web pohjaisen pelimoottorin suunnittelu**

Tietotekniikan diplomityö

2023

43 sivua, 1 kuvaa, 0 taulukkoa ja 0 liitettä

Tarkastajat: Apulaisprofessori Jussi Kasurinen ja TkT Jani Rönkkönen

Avainsanat: Pelimoottori, HTML5, web, suunnittelutiede

Tämä opinnäytetyö on tutkimus siitä mitä tarvitaan kehittääkseen web pohjainen pelimoottori Seepia Playbles yrityksen tarpeiden mukaisesti. Työssä käydään läpi mitä vaatimuksia pelimoottirille on, mitä olemassa olevia ratkaisuja on ja miksi ne eivät ole vaatimuksiin sopivia. Uuteen ratkaisuun vaadittavia komponentteja tutkitaan. Milloin mahdollista olemassa olevia ratkaisuja suositellaan ja niiden mahdollisista ongelmista kerrotaan sekä niihin ehdotetaan ratkaisua. Jotkin näistä ongelmista johtuvat web alustasta ja muut varsinaisesta ratkaisusta itsestään. Niille vaatimuksille, joille ei ole olemassa ratkaisua suunnitellaan ja ehdotetaan uutta ratkaisua.

ABBREVIATIONS

| | |
|---|---|
| API | Application Programming Interface |
| GUI | Graphical User Interface |
| UI | User Interface |
| OOP | Object Oriented Programming |

**Table of contents**

# 1 Introduction

Making games is complex and time-consuming task. Some of it is because creating something new and unique is hard but there are also many tedious repetitive tasks. Some tasks also are easier to accomplish using textual editing and others benefit from a visual editor. That is why game engines, in this case referring to the development tools, are so popular. At Seepia Playables however playable ads, which are usually small demos of games, are still mostly made manually. Some tasks are automated, but many aren't, and visual tools are barely used for constructing the games. This makes making the games unnecessarily tedious and time consuming. Some examples are having to manually add asset loading for every asset and editing game levels and animations by changing values in code and then reloading the game. These can be greatly simplified with a visual editor and asset manager. So, a game engine is needed.

There are many existing game engines out there but many of them are not suitable for simple reasons like wrong target platform, which in this case is web, or because the company has some rather unique requirements. The company's requirements for the game engine are to:

- Output very small builds
- Be easy to use for non-coders
- Support lots of build variations and their easy development
- Support running games and building them in a web browser
- Support any game aspect ratio and the possibility of it changing dynamically

These unique requirements come from the fact that Seepia is mostly making playable ads. A more detailed and descriptive list of requirements is further in the thesis in the implementation chapter.

## 1.1 Scope and limitations

As the task of making a game engine from scratch is quite a large one this project makes heavy use of pre-existing software libraries. Things like physics engine or rendering engine

are some of the bigger examples which would be an entire thesis on their own but since there are suitable open-source implementations for them there is no need to reinvent the wheel. The available implementations will be compared and the most suitable selected. This project focuses on the editor and user experience part of a game engine by integrating necessary libraries together and designing what doesn't yet exist or can be improved relatively easily. During this project only 2D graphics support will be considered because the company doesn't need 3D that much, but it will be designed in such a way that later implementing 3D graphics shouldn't be difficult.

As it is supposed to be possible for non-coders to make games just by combining components in the editor some components need to be designed. Only some of the most commonly needed and basic components will be designed during this project and the creation of more game specific components will be left to the game projects.

## 1.2 Goals of this project

The goal for this project is to find out what is needed for making a game engine with specific requirements and what problems there are. This will require research into the existing solutions and libraries and their integration into a functional piece of software. New solutions will need to be suggested for any requirements that don't have viable existing solutions.

## 1.3 Structure of the thesis

The thesis starts with an introduction. Then the game engines as a concept are explained. Third chapter covers the research method chosen for this thesis. Fourth chapter is all about the research and design. It includes the requirements for the engine, research into existing solutions and designs for new solutions. Finally, the thesis ends with conclusions.

# 2 Game engines

Game engines are software used to make creating games easier. The term can be used both for the framework consisting of core libraries for making the games but also the combination of a framework and a visual editor for it. (Christopoulou and Xinogalos, 2017; “Common game development terms and definitions | Game design vocabulary | Unity,” 2022; Cowan and Kapralos, 2014; Valencia-García et al., 2016, p. 146) The terms game engine and framework are somewhat interchangeable and are used differently in different sources. In (Cowan and Kapralos, 2014) game engine is defined as the core that is part of the completed game but in (“Common game development terms and definitions | Game design vocabulary | Unity,” 2022) game engine is the suit of tools used for game development. This study uses the latter definition and the artifact that will be created will be a suite of tools with emphasis on the editor part. The frameworks consist of low-level components of games like rendering and physics engine leaving the higher-level game logic for the actual game developer to do (Cowan and Kapralos, 2014). Games are of course very diverse, and it is hard to make an engine that could support every type of game optimally. It is much easier to make a game engine that is optimized to one specific type of game. There are of course some very popular game engines like Unreal and Unity that do a good enough job in being a general-purpose game engine that they have become the most used engines in Steam marketplace according to information scavenged from games in Steam (Doucet et al., 2021). These are commercial game engines available for anyone to use with a certain license agreement (“Plans and pricing,” 2022; “Unreal Engine (UE5) licensing options - Unreal Engine,” 2022). There are also many in-house game engines that are not publicly available and open-source game engines that are freely available. There is a huge number of game engines programmed in different languages, for different platforms, with different designs, and with different licenses. A list of engines in Wikipedia has 190 of them. (“List of game engines,” 2022)

In addition to making games for entertainment, game engines can be used for serious games or simulators for education and learning and in architecture visualization (Cowan and Kapralos, 2014; Valencia-García et al., 2016, pp. 143–146). The game engines are also being used in film production such the use of Unreal in the making of The Mandalorian series (Farris, 2020; “Unreal Engine Powers Film & Television Production,” 2022).

# 3 Related work

While research this topic it became evident that there isn't much prior research on the subject of game engine architecture or game engines as a whole. Game engine related research is mostly about specific parts of the engine like rendering or physics. This has also been noticed by some prior studies (Maggiorini et al., 2016, p. 279). Nevertheless, some works with value were found.

Warren (Warren, 2019) studied small purpose built online game engines. The game engines studied were Twine, Bitsy and PuzzleScript. All of their current versions run in web browser, and they are made for easily creating very specific types of games. Twine is for creating interactive stories and it supports some simple logic and images and sounds. Bitsy is for creating The Legend of Zelda like top-down adventure games with very limited graphics. PuzzleScript is for creating different kinds of puzzle games like the name would suggest. Their logic is made with PuzzleScripts own simple scripting language. The study gather some data form itch.io marketplace from 5001 games created using these engines. The data revealed that the games created using these engines are mostly free and short. More interestingly a survey was conducted where 161 developers that had made some of the 5001 games answered. This showed that Twine had 45.9%, Bitsy 59.3% and PuzzleScript 93% male users. The rest were female and non-binary. PuzzleScript users also had the most programming experience which would make sense since it is the only engine that needs programming.

Campos et al. (Campos et al., 2022) studied, design and developed a small 2D specific game engine. Their goal was to make it flexible, easy to extend and easy to use. They achieved the flexibility by constructing the engine from modules that can be easily customized and plugged into the game. They also used Entity Component System architecture for the code. They tested the project by implementing a platform game and also asked 12 developers, 2 experts and 10 newbies to create a short game in 2 days. They were given manual, a video tutorial and assets. A survey was then conducted on their experience using the engine. 91.6% of the respondents were satisfied with the design of the application, 75% found the functionalities useful for what they were meant for, "85% of the participants noticed a

reduction of time and increase of efficiency … with our solution vs the tools that they usually use such as Unity, Unreal Engine or Godot" and 75% considered using it for future projects.

Park and Baek (Park and Baek, 2020) design and developed a small light-weight 3D game engine prototype. Their goal was to develop an engine where they can quickly implement and test new features. Like Campos et al. they used a modular approach. Their game logic uses event-based system where functions are automatically called when a certain event is detected.

Maggiorini et al. (Maggiorini et al., 2016) studied game engines as a whole and identified 3 possible problems and proposed a solution for them and an alternative approach. The problems they identified are monolithic software, centralized software and platform dependency. Monolithic software is problematic because of build times since even small changes could lead to big rebuilds. Their solution is to use plug-in approach where there are smaller standalone modules. Centralized software is problematic because it's hard to scale up if more computational power is needed. In their example they talk about online games where currently the games need to be specifically made with distributed computation in mind if it is needed. Their solution is to create a high-performance messaging system between modules in engine so that they can be transparently distributed. Platform dependency is problematic because the engines need to have multiple lower-level implementations for different platforms. The implementations may not be work equally well. Their solution again is to use modularity to make the implementation easier and possibly use the messaging system to create platform independent communication between modules. Their proposed alternative approach is to create a new engine where everything is modules, and they communicate using a messaging system. They likened it to a microkernel.

Chover et al. (Chover et al., 2020) studied 2D game engines and designed and developed a simplified game engine. They analyzed some game engines capable of creating 2D games and classified into a table their platform, scripting method, behavior specification and how many predefined functions or behaviors the engines have. Their architecture consists of physics, input, logic, sound and render modules. Their games use a very simplified representation of scenes which have actors, and these actors have properties and rules. Every actor has same properties but can have also custom properties. The rules defined the game logic, and they are constructed using decision trees. The decision trees can use a set of fifteen predefined actions and six conditions. They created a Candy Crush like match-three

mechanic to test the engine. They also evaluated the engine on a summer camp. There were 120 participants aged between 7 and 14. 74 were boys and 46 girls. Groups of children were each assigned on of 9 different kinds of arcade games to create. All of them were able to complete the assignments. A survey was then conducted about the engine's ease of use and user satisfaction and the results were very positive. 23 three of the children were also asked to create a game using Scratch and they were then asked to compare their simplified game engine experience to Scratch experience. Only creating loops was considered harder in the simplified game engine since there isn't a predefined way to do it.

Charrieras and Ivanova (Charrieras and Ivanova, 2016) study game engines from a very philosophical standpoint. The most relevant part is their analysis of game engine architectures where they describe how game engines have evolved from more game specific to more generalized. They give discuss in more detail the programming paradigms used for game engines and more specifically object-oriented programming (OOP) and inheritance-based design, and component-based data driven design. While noting that OOP is more prevalent, they point out that it is inflexible once the inheritance hierarchy is designed. They do note though that it works well with smaller systems but as the amount of content increases it becomes increasingly harder to work with. Component-based design on the other hand is described as more flexible since it is possible to compose new game objects of any number of components. They also describe it as more content creator friendly since new objects can be made without recompiling the game.

# 4 Research method

Design science is a research method that produces an artifact to solve a practical problem (Hevner et al., 2004). It also closely resembles a normal software development workflow of designing, building, and iterating based on feedback. Thus, it is a good fit for this study. Hevner et al. have made guidelines for this type of research in the field of information systems. The 7 guidelines are for design science research are:

1. **Design as an artifact:** Design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation.
2. **Problem relevance:** The objective of design-science research is to develop technology-based solutions to important and relevant business problems.
3. **Design evaluation:** The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.
4. **Research contributions:** Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies.
5. **Research rigor:** Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.
6. **Design as a search process:** The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.
7. **Communication of research:** Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.

The first guideline of producing an artifact is fulfilled by the resulting list of found libraries and made designs that will be produced during this study. The second guideline of problem relevancy is fulfilled by the company's need for improved game making tools and workflows. The third guideline of design evaluation is fulfilled by making sure that the requirements are met. The fourth guideline of research contributions is fulfilled by the designed game engine's improved and unique features compared to other available solutions.

The fifth and sixth guidelines are closely linked and are about the research and design process and then iterating on the design to improve it. The seventh guideline of communication of research is fulfilled is fulfilled by explaining the research in this thesis sufficiently clearly and detailedly.

# 5 Research and design

This chapter has the different steps of the game engine's design process. First there are requirements for the game engine which include both basic requirements for any game engine and unique requirements for the company's use case. These requirements are based on experience of working on various projects and discovering difficult or time-consuming tasks that could be solved with a well-made tool. After requirements there is a look into existing game engines and libraries to see if there are any that could be used. There will also be suggestions for any found issues and designs for new solutions.

## 5.1 Requirements

As the target build platform is web and the developers at the company are already familiar with web technologies it makes sense to use them also for the game engine and its editor part. The game engine and its editor should be made using typescript and other common web project tooling. The editor should run in a web browser and be able to run the games and build them. While the target platform is web it should be possible to build into various slightly different formats for different publishing services.

The build size limitations for the playable ads by ad networks are very strict and are usually at the maximum 5MB. In practice it is better to aim even lower to enhance the ad performance since smaller size means the ad will be downloaded and shown faster. This is why 2MB is usually the target size. The worst-case scenario is when the output needs to be a single HTML file with everything embedded into it including the engine and all the assets. This is a problem because the assets need to be encoded in Base64 to be embedded in the HTML file which causes their size to grow by about 33% ("Base64 - MDN Web Docs Glossary," 2023). To help with this the used fonts should be automatically optimized by making a subset of only the used characters and using an optimized format. The sounds and textures should also be optimized as much as possible while keeping acceptable quality, and the optimization should be easily configurable.

The editor should make developing build variations easy. There should be a support for variations and sub variations or dependencies. It should also be easy to switch between the variations in the development environment and test them.

The engine should have a visual editor interface that is easy to use for even non-coders. The interface should be familiar and use common components where applicable while improving on things when possible. The interface should also have resizable and movable elements to support different workflow preferences.

The engine should support 2D graphics and physics and have easy to use editors and gizmos for them. There should also be 2D particle system component that supports complex particle behaviour and sub particles. There is also a need to support any aspect ratio and the possibility of the screen orientation changing during gameplay. For this some UI components and system is needed. They should support placing the components in different positions in different aspect ratios. Animations should also be taken into account. Normal elements are easy to animate by just interpolating between their start and end values but in the case of the UI components some extra functionality is needed. Since the start and end values can change mid-animation, the animation must update accordingly. To make managing the animations easier a timeline or similar editor is needed.

The editor itself and the games as well should support localization. In addition to building the game with a single localization to save space it should be also possible to make a build with multiple languages. The language could then be automatically detected or manually set.

To save the created game levels and user interfaces (UI) there is a need for a scene file format. This will also need an accompanying scene loader.

The engine should support sounds. Only very basic functionality is needed including playing and stopping audio and changing its volume and playback rate.

The coders may be used to various different code editors. It should be possible to use any of them in the workflow.

It should be easy to add and reuse game components. They should only be included in the builds when they are actually used.

The editor should support version control. Git is the most popular version control solution and it has many good features (Mustonen, 2019). The company is also already using Git so it makes sense to try to make it work in the editor as well.

The engine should have an input manager. The main input method is touch but other methods should be considered.

## 5.2 Existing game engines

As mentioned in the game engines chapter, there are a lot of game engines out there. This begs to question if there would already be a suitable game engine for the company to use. Thus, there is a need to find out what is available.

Using the previously mentioned game engine list as a base for the search, the requirements above quickly filter out most of the game engines. After taking out all engines that don't have web as a target platform there is only 25 engines left of 190. Then taking out all that don't use a web interface leaves 9 options with some sort of web interface. Further taking out the ones that don't support 2D leaves 6 alternatives. Then removing the ones that don't support TypeScript development technically leaves nothing but 4 of the engines have the option to use JavaScript. Since TypeScript compiles to JavaScript, it would be possible to use it through some tinkering but as it is not built into the tools it is not optimal. In Construct and GDevelop the primary programming method is their event sheets which are a sort of higher level more visual and descriptive programming. The remaining engines are:

- Construct ("Game Making Software - Construct 3," 2022)
- GDevelop ("Free and Easy Game-Making App | GDevelop," 2022)
- PlayCanvas ("PlayCanvas - The Web-First Game Engine," 2022)
- Twine ("Twine / An open-source tool for telling interactive, nonlinear stories," 2022)

Twine is tool for creating interactive stories, so it doesn't fit the use case. Construct is a commercial game engine and GDevelop is open source. PlayCanvas has open-source engine and commercial editor. PlayCanvas is 3D first but has also some 2D features whereas both Contruct and GDevelop support only 2D. Minimal PlayCanvas builds are too big as the runtime is about 1.3 MB and physics adds another 1.8 MB, so it is out. GDevelop has more

reasonable build size but its physics engine uses WebAssembly which is not allowed in the use case. GDevelop's interface is also more confusing than either PlayCanvas or Construct. As Construct is closed source it is harder to make changes to it if necessary but both GDevelop and Construct do support extensions. Neither of them has premade solutions for build variations, robustly handling dynamically changing aspect ratio, asset optimization, localization or version control and the other requirements are not fulfilled ideally.

("HTML5 Game Engines - Find Which is Right For You," 2022) and (Valencia-García et al., 2016, p. 146) also list some game engines which are specifically those with HTML5 target. The lists have many similarities and many of them were also on the Wikipedia list above. Many of the engines have discontinued development and thus are not suitable. Most of them are also just frameworks and not game engines with an editor. Interesting ones are:

- Phaser ("Phaser - A fast, fun and free open source HTML5 game framework," 2022)
- Pixi.js ("PixiJS," 2022)
- MelonJS ("melonJS," 2022)

Phaser and MelonJS are game frameworks with renderer, audio, input, physics, and other utilities already included. Pixi.js is mostly just rendering library though it comes with support for pointer events and it has some plugins available for example for sound ("PixiJS Sound," 2022). MelonJS is interesting mostly because it is quite a bit smaller than either Pixi.js or Phaser ("melonjs v14.1.2 | Bundlephobia," 2022; "phaser v3.55.2 | Bundlephobia," 2022; "pixi.js v7.0.5 | Bundlephobia," 2022). Upon further research an editor for Phaser ("Phaser Editor 2D | HTML5 Game IDE," 2022) and Pixi.js based game editors ("Ct.js — a free game editor," 2022; Kostin, 2022) was found. Phaser Editor 2D is quite good but doesn't have all the necessary features. Large part of its source code is open source, but the core is closed. Ct.js is Pixi.js based open-source game editor which has somewhat unusual user interface. Thing-editor is another Pixi.js based open-source game editor. Unfortunately, its documentation is currently only in Russian. All these editors have a web-based user interface, but they don't run purely in browser instead run a server locally that does the file operations and serves the user interface to a browser.

None of these options fill the requirements. As such it is reasonable to build a custom game engine more fitting to the use case.

## 5.3 Existing libraries

To minimize the work required to make the custom engine it makes sense to use as many existing libraries as possible provided that they are good enough. The needed functionalities that require a library or need to be implemented from scratch are:

- 2D rendering engine
- 2D physics engine
- 2D particle system
- UI library for the editor
- Audio library
- Input manager
- Scene format and loader
- Animation system
- GUI components
- Asset optimization
- Build system that support various targets
- Build variations and sub variations
- Running typescript in browser
- Localization
- Version control

### 5.3.1 2D rendering engine

There are plenty of 2D rendering engines out there. Some game frameworks, like Phaser, come with their custom renderer on top of many other libraries already integrated (“Phaser - A fast, fun and free open source HTML5 game framework,” 2022). A simple performance benchmark has multiple canvas engines compared against each other in drawing squares on

screen (“Canvas Engines Comparison,” 2022). There are 16 different implementations of the benchmark with Pixi.js being the fastest. Another benefit about Pixi.js is that the company already uses it, so it is familiar to the developers there. While Pixi.js is not the smallest of the options, it has a lot of features whereas some of the others are too minimal. To optimize the size the unnecessary features can be removed. There is even an implementation for Three.js 3D rendering engine which is good but doesn’t compete with Pixi.js in 2D. Unfortunately, Phaser framework doesn’t have an implementation there but it’s performance and features seem to be similar to that of Pixi.js.

A test project was made that implement a just rotating image to compare the build sizes of Pixi.js, Phaser and MelonJS (Mustonen, 2023). It was found that MelonJS had some aliasing in the image and the image jittered while rotating so it is not a suitable library. Pixi.js and Phaser on the other hand produced nearly identical results. The difference can be seen in Figure 1. The image’s edges in the Pixi.js renderer are slightly aliased because by default Pixi.js doesn’t use anti-aliasing whereas Phaser does.

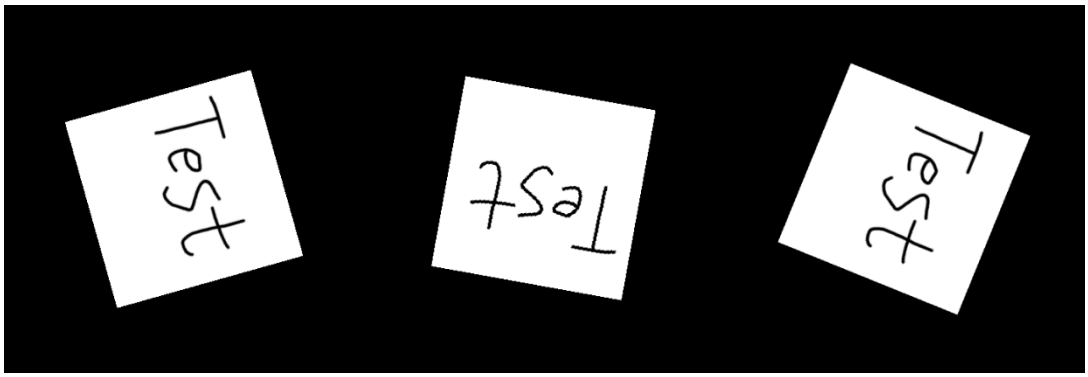


**Figure 1. From left to right Pixi.js, MelonJS and Phaser test results.**

In this very simple case Pixi.js was simpler to use since it didn’t need to use predefined way of how things should work. With the normal package imports for both libraries the final JavaScript bundle sizes for the project were 371.16 kB for Pixi.js and 1357.45 kB for Phaser. This is obviously a massive difference, but it is expected since Phaser comes with a lot more built in like physics, sound, input, camera, and particle systems. Still, it shows that out of the box Pixi is clearly better for small simple projects. Both libraries can be optimized though to include only what is needed. For Pixi this means using separate packages for features and only importing what is needed. The import for this test project went from this:

```
import { Application, Assets, Sprite } from 'pixi.js';
```

to this:

```
import { Application } from "@pixi/app";
import { Assets } from "@pixi/assets";
import { Sprite } from "@pixi/sprite";
```

Phaser has a far more tedious process to optimize it since it needs to be manually rebuilt with a configuration specifying what to include in the build (Davey, 2022). The configuration used for the test is as follows:

```
const Phaser = {
    Cameras: {
        Scene2D: require('./cameras/2d')
    },
    Events: require('./events'),
    Game: require('./core/Game'),
    GameObjects: {
        DisplayList: require('./gameobjects/DisplayList'),
        UpdateList: require('./gameobjects/UpdateList'),

        Image: require('./gameobjects/image/Image'),

        Factories: {
            Image: require('./gameobjects/image/ImageFactory')
        },

        Creators: {
            Image: require('./gameobjects/image/ImageCreator')
        }
    },
    Loader: {
        FileTypes: {
            ImageFile: require('./loader/filetypes/ImageFile')
        },
```

```
        LoaderPlugin: require('./loader/LoaderPlugin')
    },
    Scale: require('./scale'),
    ScaleModes: require('./renderer/ScaleModes'),
    Scene: require('./scene/Scene'),
}
```

After the optimizations the projects final JavaScript bundle sizes were 226.16 kB for Pixi.js and 543.79 kB for Phaser. Both are significantly smaller while Phaser still being way bigger. With Pixi.js being smaller, easier to work with and the developers are already familiar it, it is recommended as the rendering engine.

### 5.3.2 2D physics engine

For physics engine the company has previously used Matter.js (liabru, 2023) which is small and has nice documentation. It lacks some features though like a revolute joint. Revolute join is useful for example for ragdolls to limit the rotation of its limbs. The performance of Matter.js also isn't amazing. There are many other physics engines some of which use WebAssembly and others that aren't maintained or are too big to be suitable. Here are some interesting engines and their sizes are:

- Matter.js 77.7 kB
- P2-es 89.5 kB
- Cannon-es 123 kB
- Planck.js 186.5 kB
- @box2d/core 233.1 kB

Cannon-es ("cannon-es," 2023) is actually a 3D physics engine unlike the others that are 2D. It is on the list mainly because its size is surprisingly small while being fully featured. 3D engine could of course be used for 2D physics by locking an axis but a 2D engine is more optimized for its purpose. Planck.js (Shakiba, 2023) and @box2d/core ("@box2d/core," 2023) are TypeScript implementations of Box2D (Catto, 2023) a quite popular C++ 2D physics engine. There is a benchmark comparing different JavaScript or TypeScript

implementations of Box2D and these two engines are the interesting maintained engines from the list ("@box2d/benchmark," 2023). @box2D/core is over twice as fast as Planck.js in this benchmark. @box2D/core doesn't have its own API documentation and instead redirects to the Box2D documentation. Planck.js has good documentation. Both of these are however over twice as big as p2-es which also has good documentation and features. P2-es is also tree shakeable which means only the features that are used will be included in the final builds. For these reasons p2-es is recommended.

### 5.3.3 2D particle system

Since Pixi.js was chosen as a rendering engine, it could make sense to just use PixiJS Particle Emitter library ("PixiJS Particle Emitter," 2023). Another option could be modular-particle-system library that can be easily extended with modules as the name would suggest ("Lightweight Particle System for TypeScript," 2023). Both of them use their own kind of modules though in Pixi's particle emitter they are called behaviours. Since they are modular it is possible to only include what is needed saving some space. As Pixi's particle emitter is more tested and under active development, it is recommended.

### 5.3.4 Audio library

Pixi.js has a library for sound as well called PixiJS Sound ("PixiJS Sound," 2022). There is also a popular audio library called Howler (Simpson, 2023) that is also used at the company already. These libraries are about the same size but have some different features. PixiJS Sound has audio filters whereas Howler has 3D spatial sound and stereo panning. As spatial sound is quite nice for many games and Howler also handles edge cases and bugs on different platforms it is the recommended option.

### 5.3.5 UI library for the editor

There are many UI frameworks to choose from. BestOfJs site has a seach for all things JavaScript. Searching with UI framework tag results in a list of 48 projects that can be sorted based on a few different criteria including Github stars which are basically favorites on

Github used to follow a project (“Best of JavaScript,” 2023). Sorting by total number of stars gives Vue.js 2, React and Angular as the top three frameworks with 202, 200 and 86 thousand stars respectively. When sorting by stars added in the last 30 days the top three are now React, Svelte and Solid with 43.4, 28.9 and 19.8 stars added per day respectively. Vue and Angular aren’t far behind at 5th and 7th places. On the total number of stars list Svelte is 4th with 65 thousand and Solid 10th with 25 thousand stars. These results show that React and Vue are hugely popular and React is gathering popularity fast. On the other hand, there are also newer frameworks like Svelte and Solid that are also increasing in popularity.

A benchmark testing the performance of many UI frameworks also includes the afore mentioned frameworks except Vue.js 2 only having Vue.js 3 (“js web frameworks benchmark,” 2023). Looking at the “keyed results” table and its geometric mean of all the factors in the table puts the frameworks from fastest to slowest in the following order:

- Solid: 1.09
- Vue (3): 1.23
- Svelte: 1.29
- Angular: 1.61
- React: 1.72

Most of the frameworks that are faster than Solid are experimental or don’t have good documentation. There are though couple interesting ones, Inferno and Mikado. Mikado is one of the fastest with score of 1.03 and Inferno is about the same with Solid with a score of 1.09. Inferno is fairly popular with 15.6 thousand stars, but Mikado only has 574.

Popularity has the benefit of having existing resources like tutorials and ready-made components, but performance and size matter as well. Solid has very similar syntax to React and Inferno even describes itself as React-like and has a compatibility layer to directly use existing React-based modules. In the end Solid is recommended because it is quite popular, similar to React while being more performant and smaller and has nice resources.

These are all HTML frameworks. Another option would be to use the 2D rendering engine to also make the editor but using HTML framework has the benefit of being able to utilize the layout and styling functionality that already exists in browsers. It also makes it easier to

later add an integrated code editor since there are already nice barebones editors that can be used like CodeMirror (“CodeMirror,” 2023).

While these frameworks make creating UI components easier they don’t have anything to aid in setting their layout. One of the requirements was for the UI to have resizable and movable elements. This means that basically a tiling window manager should be created. This can be quite complicated. Many tiling window managers make use of binary trees to store the windows (Dejean, 2023; “i3 - improved tiling wm,” 2023). The windows are stored in containers that are split in two either horizontally or vertically. This has the problem that organizing the windows is inconsistent. For example, if the screen were to be divided into four quadrants. In this configuration the tree would look like this:

- Root
    - Container
        - Window 1
        - Window 2
    - Container
        - Window 3
        - Window 4

This means that if you would try to resize a window by moving its edge it would work as expected on the edges between two windows in the same container but not on the edges of windows in different containers. Instead, it would resize the containers thus resizing all the four windows instead of only the two that intuitively should have resized. Blender 3D modelling software has quite nice tiling where this has been handled and when window corners are aligned the windows can be resized either way as one would expect (“Blender,” 2023). When resizing its application window, the sub windows are scaled which means the window sizes are a percentage of the main window size instead of a certain pixel value. This is very simple and may not be the best for user experience. For example, if the application was split horizontally into main content and an inspector, the inspector would likely only need certain width but couldn’t practically be any narrower. In this case when resizing the application window, it would make sense to only resize the width of the main content.

Another user experience issue is that when resizing a sub window its edge can only be moved within it and its neighbour. It could be nicer if the edge would push the other windows once they have reached their minimum size. Though both of these changes add quite a bit of complexity to the window management. Another nice feature would be extending windows to other windows' areas. For example, if the application was split into three and there would be two windows on the left and one on the right and you would want to have the left bottom window be at the bottom and the other two at the top, it would be handy to be able to extend the left bottom window into the right window's area. It should also be possible to move windows into another window's edge, splitting its area.

### 5.3.6 Input manager

The games only required touch input and Pixi.js has input manager which does support it. It would make sense though to implement some helper functions so that it is easier to disable multi-touch. The editor will need input manager that supports keyboard shortcuts and different contexts. Contexts would be for example 2D view editor, animation timeline and global context. With different contexts it is possible to use same shortcut for different actions depending on the contexts. The context is selected based on what part of the editor is currently focused. Hotkeys (小弟调调, 2023) is an actively developed library for exactly this purpose and may be a good option but creating a custom solution shouldn't be too hard either should it be needed. Mouse input is handled by each UI element individually and 2D view editor uses Pixi's input manager.

### 5.3.7 Animation system

There are many animation libraries available but some of them are not maintained anymore and others are too big or are meant for some specific environment. Couple small interesting libraries are Shifty and Popmotion ("Popmotion," 2023; "Shifty," 2023). Both are small and maintained but Shifty has nicer features like better playback controls. There is also available a more advanced library called Rekapi which builds upon Shifty with a timeline functionality

for more complex animations (“Rekapi,” 2023). It’s not clear whether Shifty will easily support dynamically changing targets but other libraries don’t seem to do it either so it will be discovered during integration. The animation editor which is usually called timeline needs to be made practically from scratch because while some timeline libraries exist, none were found that would be easily pluggable.

To account for dynamically changing layouts the start and end values of animation should be set for each different layout. Basically, they would be components that don’t have a visual representation but can otherwise act like any other component and be positioned in the layouts however wanted. Then the animation would lerp between the start and end components values and when the layout is changed it would change to use the corresponding components.

### 5.3.8 GUI components

The graphical user interface (GUI) components in this case means just the very basic containers that have the ability to be nicely positioned on screen by anchoring and offsets to the parent element. Other GUI components are basically just regular elements as a child of GUI container, but they could be implemented in other ways as well.

### 5.3.9 Asset optimization

Textures need to be compressed and packed into spritesheets. For this there is free-tex-packer (Norynchak, 2022). Sounds need to be converted to mp3 and WASM version of FFMPEG could be used (“ffmpeg.wasm,” 2023). Fonts need to be converted to woff2 and made into a subset of only the characters used in the localizations and there is a tool for just this called subset-font (Lind, 2023). Sound compression is just for convenience and doesn’t need an UI other than for setting the compression settings. The sound files itself should just go to one folder and all of them should automatically get converted and placed to another folder. Any other sound changes like cutting should be done in other software. The texture packer would also need UI for settings but it could also show a preview of the packed texture so the UI could be very similar to TexturePacker (“TexturePacker - Create Sprite Sheets for your game!,” 2023). The font optimization first of all needs a localization file where it gets

the used characters. The UI for font optimization should have also a place to select which localizations are used with each font so that all the required characters can be gathered.

### 5.3.10 Build variations and dependencies

So that the variations can be easily edited using the editor, variations need to be saved into the scene data. The data is further described in the next part. The unused variation data must then be removed from each build during the build process to minimize the used space. There could also be functional variations in the TypeScript which could be handled by using a code minifier like Terser to optimize out conditional expressions that evaluate to false (“API Reference · terser,” 2023). The variation data itself could be for example:

```
[
        {
                dependencyRule: “”,
                length: [“short”, ”long”]
        },
        {
                dependencyRule: “”,
                boss: [“skeleton”, “slime”]
        }
]
```

This is then used to build multiple builds. The above example would build:

- Short, skeleton
- Short, slime
- Long, skeleton
- Long, slime

Basically, a list of all the different combinations is created. As an example of dependencies the data would instead be:

```
[
        {
                dependencyRule: “”,
                length: [“short”, ”long”]
        },
        {
                dependencyRule: “long”,
                boss: [“skeleton”, “slime”]
        }
]
```

In this case only the following builds would be built:

- Short
- Long, skeleton
- Long, slime

This is because the boss variations are added only when “long” variations is true. This can be useful to not make unnecessary builds. The rule can also have multiple variations combined with logical operators. Since the boss variation depends on the length variation, the length variation must be set first. It would be easiest to implement it by having the list of variations be manually ordered so that the dependencies are met.

### 5.3.11 Scene format and loader

Scene formats are quitter engine specific unless compatibility with some other engine is desired. In this case since variations need to be supported a more specialized format is even more necessary. A custom format should be made with simple JSON file with the necessary object properties saved to it. It will need to take into account variations and different UI layouts in different aspect ratios. The form can be:

```
{
        type: NameOfClass,
        children: [Array of data in this same format],
        position: [
                {
                        variationRule: “variation1”,
                        data: [
                                {
                                        aspect: number,
                                        x: number,
                                        y: number
                                }
                        ]
                }
        ],
        ...
        Other class properties
        ...
```

}

In this example position property has an array of variations. The variations are checked until a variation with variationRule that evaluates to true is hit. Variation rule would be a string that has variation names and logical operators which needs to be parsed and evaluated to see if it is true. The variation then contains data which is an array of property specific data for possibly but not necessarily different aspect ratios.

### 5.3.12 Build system that support various targets

This part is similar to build variations but is more about the files used for building and the output files of the build. Different build targets need different integration to their respective publishing platform and have some unique requirements. The chosen bundler will need to be able to output single HTML builds as well as with separated assets and a JavaScript bundle. The next part includes information about bundlers.

### 5.3.13 Running typescript in browser

As TypeScript cannot be run directly in browser there needs to be some system to transpile it into JavaScript. After it has been transpiled it can be run with a simple JavaScript eval and if all the code uses EcmaScript modules it should just work. This is basically what CodeSandbox has done (“Creating a parallel, offline, extensible, browser based bundler for CodeSandbox,” 2017). Their first plan was to use Webpack JavaScript bundler because it is widely used, and they got it somewhat working but not up to their needs. A bundler has the advantage that it may be able to use other module types like CommonJs modules and it is already made and has various optimizations. A disadvantage is that they are mostly made for NodeJs environment and thus won’t work in browser out of the box and may need quite a bit of work to make it function. Divriots have made a browser version of Vite bundler (“Vite in the browser - ‹div›RIOTS,” 2022). Their version doesn’t support node_modules folder and instead a plugin has to be made that resolves module imports. Unfortunately, the Vite version they use is still 2.7.0 whereas Vite has moved to 4.0.4. Vite actually uses ESbuild bundler for development environment and RollUp bundler for production bundling

both of which have browser compatible versions (“esbuild - API,” 2023; “Frequently Asked Questions | Rollup,” 2023). There is also the option of going lower level and running NodeJs environment in browser by using WebAssembly which StackBlitz has done (“Introducing WebContainers,” 2021). This makes it possible to do almost anything one could normally do in NodeJs environment including running unmodified bundlers like Vite. Unfortunately, it isn’t available as open source. Any of these solutions may be good for this project though running a whole NodeJs environment in browser might be an overkill. It must be tested during implementation what will be the best option to go with.

One common thing is that all of them need a file system implementation, or it at least make things easier. For that there are some libraries like BrowserFS, lightning-fs and filer (“DustinBrett/BrowserFS at FileSystemAccess,” 2023; “filerjs/filer,” 2023; “lightning-fs/src at main · isomorphic-git/lightning-fs,” 2023). Filer and lightning-fs only have IndexedDB backend so the files are only accessible in browser and only the files in that filesystem can be used. BrowserFS has multiple backends and this fork of it also includes implementation for File System Access API which provides access to local file system files though only in Chromium based browsers. Other browsers only support Origin Private File System which is only accessible by from the web page that created it like IndexedDB (“File System Access API | Can I use... Support tables for HTML5, CSS3, etc,” 2023; “File System Access API - Web APIs | MDN,” 2023). While using the File System Access API allows to read and write the files outside of browser, there is no way to watch for the file changes. This means that for the changes to be updated in the engine all the files would need to be checked for changes once the engine gains focus. This is will obviously became slower as more files are added. There could also be an option to just check for changes in manually selected files.

The file system is needed for accessing the source code that is to be transpiled and built but also for serving the files to the browser since local files can’t be normally served because of cross-origin request sharing policy. To get around this a service worker needs to be implemented that acts like an in-browser server serving the game and engine files. (Samuele, 2022)

### 5.3.14 Localization

Localization can be simply accomplished with a JSON for the languages with the localized strings. For example an English localization file could look like this:

```
{
        play_now: “PLAY NOW!”,
        you_won: “YOU WON!
}
```

This object can then be queried for the needed string. There is even a handy module for it called dlv (Miller, 2023). The language variable can be either set with a build variation or in case of a multi-language build it can be automatically checked from user’s browser or there can be a switch in the game. The correct localization is then chosen based on set language. When switching language during game the changes texts won’t automatically change so it will need some sort of a function to update the text objects.

### 5.3.15 Version control

Since the company already uses Git version control it is natural to look for a solution that can utilize it. Fortunately there is a library for just that called Isomorphic-git (“isomorphic-git · A pure JavaScript implementation of git for node and browsers!,” 2023). It works both in node and in browsers which is of course required for this project. Unfortunately, there is a browser security feature which blocks cloning and pushing repositories to different website origins unless it is explicitly enabled in the Git hosts servers settings, which unfortunately the big public Git hosts don’t support (“isomorphic-git,” 2023). Isomorphic-git suggest to use a proxy server as a way to circumvent the restriction or use a service which does support it. Another option would be to use a browser extension that bypasses the restriction like Hoppscotch does (“Hoppscotch,” 2023). Isomorphic-git also needs a node fs like file system implementation and the BrowserFS mentioned previously can be used.

# 6 Discussion

In many of the related works modularity was found to be a good thing and it is also what this papers engine aims for. While making the lower-level modules easily replaceable such as changing the physics engine could be nice it is not the first priority. Making things work comes first. However higher level modularity of reusable game components is important. How granular the modularity is will depend on the implementor but for example a match-three mechanic could be a component. It would then expose configuration in the editor such as adding graphics for the puzzle pieces. It could also be composed of subcomponents like grid component with basic grid management functions.

Warren and Chover et al. (Chover et al., 2020; Warren, 2019) papers also showed that simplified programming interfaces are viable for content creation by non-programmers. While this papers engine aims to be easy to by non-programmers it is more related to creating user interfaces and level layouts. Of course, also using premade components like the aforementioned match-three component is possible. It may be possible to a compose a more complex component from simpler one but creating totally new functionality is left for programmers.

Many of the related works also showed that small light weight purpose-built engines have benefits. While this papers engine is not specific to one type of game, in fact it should be possible to create anything, it is specific to certain kind of output in this case playable ads. An engine optimized for creating small builds, with varying specifications and lots of variations is sure to speed up development.

The problems identified by Maggiorini et al. (Maggiorini et al., 2016) don’t ably in this papers engine’s design. The monolithic nature is solved by modularity, centralization issue doesn’t ably to the use case. Since this paper’s engine outputs to web platform only the platform dependency basically doesn’t ably but web is slightly fractured platform because of different browser versions and their application programming interface (API) compatibility. Also, different ad networks are sort of different platforms though they still run in browser. The API issue is handled by only using APIs that are commonly available, so the latest and greatest APIs can’t be used. The networks are abstracted away, and their specifics are handled in their own modules.

Finally Charrieras and Ivanova's (Charrieras and Ivanova, 2016) discussed the programming paradigms. The libraries used by this paper's engine use OOP, so it is likely easier to use that. Also, since the engine aims to be minimal it may not become problematic. The game logic and component should still be a possible to be freely made just like each developer wants to. How game object properties are exposed to the editor interface should work with any paradigm.

# 7 Conclusions

The goal of this study was to find out what is needed to make a game engine with certain requirements. Initially it was also meant to include implementation of a game engine based on the research, but it turned out to be more challenging and time consuming than what was first estimated so that part was dropped out.

In this study a list of requirements for the game engine were gathered, and based on that research into each component required to compose a game engine was done. Many good existing libraries were found, and some parts were designed. Some web platform specific challenges were identified, namely that browsers have very limited access to the local file system and browsers have a cross-origin resource sharing security policy that blocks some use cases unless it is bypassed one way or another.

This research will be valuable in the future when actually implementing a game engine, but it most definitely won’t be enough to make it without a hitch. While the solutions found in this study seem good there may be unforeseen issues and bugs in the recommended solutions. Thus, it could be even a good subject for a further study.

# References

API Reference · terser [WWW Document], 2023. URL https://terser.org/ (accessed 1.24.23).

Base64 - MDN Web Docs Glossary: Definitions of Web-related terms | MDN [WWW Document], 2023. URL https://developer.mozilla.org/en-US/docs/Glossary/Base64 (accessed 1.30.23).

Best of JavaScript [WWW Document], 2023. URL https://bestofjs.org/projects?tags=framework (accessed 1.10.23).

Blender [WWW Document], 2023. . blender.org. URL https://www.blender.org/ (accessed 4.24.23).

@box2d/benchmark [WWW Document], 2023. . npm. URL https://www.npmjs.com/package/@box2d/benchmark (accessed 1.9.23).

@box2d/core [WWW Document], 2023. . npm. URL https://www.npmjs.com/package/@box2d/core (accessed 1.9.23).

Campos, S.A.E., Morales, B.A.M., Núñez, Á.A.V., 2022. Open-Source Game Engine & Framework for 2D Game Development, in: 2022 IEEE Engineering International Research Conference (EIRCON). Presented at the 2022 IEEE Engineering International Research Conference (EIRCON), pp. 1–4. https://doi.org/10.1109/EIRCON56026.2022.9934816

cannon-es [WWW Document], 2023. URL https://github.com/pmndrs/cannon-es (accessed 1.9.23).

Canvas Engines Comparison [WWW Document], 2022. URL https://benchmarks.slaylines.io/ (accessed 12.13.22).

Catto, E., 2023. Box2D [WWW Document]. URL https://github.com/erincatto/box2d (accessed 1.9.23).

Charrieras, D., Ivanova, N., 2016. Emergence in video game production: Video game engines as technical individuals. Soc. Sci. Inf. 53, 337–356. https://doi.org/10.1177/0539018416642056

Chover, M., Marín, C., Rebollo, C., Remolar, I., 2020. A game engine designed to simplify 2D video game development. Multimed. Tools Appl. 79, 12307–12328. https://doi.org/10.1007/s11042-019-08433-z

Christopoulou, E., Xinogalos, S., 2017. Overview and Comparative Analysis of Game Engines for Desktop and Mobile Devices. Int. J. Serious Games 4. https://doi.org/10.17083/ijsg.v4i4.194

CodeMirror [WWW Document], 2023. URL http://codemirror.net/ (accessed 4.5.23).

Common game development terms and definitions | Game design vocabulary | Unity [WWW Document], 2022. URL https://unity.com/how-to/beginner/game-development-terms (accessed 12.8.22).

Cowan, B., Kapralos, B., 2014. A Survey of Frameworks and Game Engines for Serious Game Development, in: 2014 IEEE 14th International Conference on Advanced Learning Technologies. Presented at the 2014 IEEE 14th International Conference on Advanced Learning Technologies, pp. 662–664. https://doi.org/10.1109/ICALT.2014.194

Creating a parallel, offline, extensible, browser based bundler for CodeSandbox [WWW Document], 2017. . CodeSandbox. URL https://codesandbox.io/blog/creating-a-parallel-offline-extensible-browser-based-bundler-for-codesandbox (accessed 1.23.23).

Ct.js — a free game editor [WWW Document], 2022. URL https://ctjs.rocks/ (accessed 12.16.22).

Davey, R., 2022. Creating Custom Phaser 3 Builds [WWW Document]. URL https://github.com/photonstorm/phaser3-custom-build (accessed 1.9.23).

Dejean, B., 2023. baskerville/bspwm [WWW Document]. URL https://github.com/baskerville/bspwm (accessed 4.24.23).

Doucet, L., September 02, A.P., 2021, 2021. Game engines on Steam: The definitive breakdown [WWW Document]. Game Dev. URL https://www.gamedeveloper.com/business/game-engines-on-steam-the-definitive-breakdown (accessed 11.28.22).

DustinBrett/BrowserFS at FileSystemAccess [WWW Document], 2023. . GitHub. URL https://github.com/DustinBrett/BrowserFS (accessed 2.4.23).

esbuild - API [WWW Document], 2023. URL https://esbuild.github.io/api/#browser (accessed 1.24.23).

Farris, J., 2020. Forging new paths for filmmakers on The Mandalorian. Unreal Engine. URL https://www.unrealengine.com/en-US/blog/forging-new-paths-for-filmmakers-on-the-mandalorian (accessed 12.8.22).

ffmpeg.wasm [WWW Document], 2023. URL https://github.com/ffmpegwasm/ffmpeg.wasm (accessed 1.24.23).

File System Access API | Can I use... Support tables for HTML5, CSS3, etc [WWW Document], 2023. URL https://caniuse.com/native-filesystem-api (accessed 4.18.23).

File System Access API - Web APIs | MDN [WWW Document], 2023. URL https://developer.mozilla.org/en-US/docs/Web/API/File_System_Access_API (accessed 4.18.23).

filerjs/filer [WWW Document], 2023. URL https://github.com/filerjs/filer (accessed 4.18.23).

Free and Easy Game-Making App | GDevelop [WWW Document], 2022. URL https://gdevelop.io/ (accessed 12.8.22).

Frequently Asked Questions | Rollup [WWW Document], 2023. URL https://rollupjs.org/faqs/#how-do-i-run-rollup-itself-in-a-browser (accessed 1.24.23).

Game Making Software - Construct 3 [WWW Document], 2022. URL https://www.construct.net (accessed 12.8.22).

Hevner, A.R., March, S.T., Park, J., Ram, S., 2004. Design Science in Information Systems Research. MIS Q. 28, 75–105. https://doi.org/10.2307/25148625

Hoppscotch [WWW Document], 2023. URL https://docs.hoppscotch.io/documentation/features/interceptor (accessed 4.5.23).

HTML5 Game Engines - Find Which is Right For You [WWW Document], 2022. URL https://html5gameengine.com/ (accessed 12.16.22).

i3 - improved tiling wm [WWW Document], 2023. URL https://i3wm.org/ (accessed 4.24.23).

Introducing WebContainers: Run Node.js natively in your browser [WWW Document], 2021. URL https://blog.stackblitz.com/posts/introducing-webcontainers/ (accessed 1.23.23).

isomorphic-git · A pure JavaScript implementation of git for node and browsers! [WWW Document], 2023. URL https://isomorphic-git.org/ (accessed 1.9.23).

isomorphic-git [WWW Document], 2023. URL https://github.com/isomorphic-git/isomorphic-git (accessed 1.9.23).

js web frameworks benchmark [WWW Document], 2023. URL https://krausest.github.io/js-framework-benchmark/2022/table_chrome_108.0.5359.71.html (accessed 1.10.23).

Kostin, V., 2022. Thing-editor [WWW Document]. URL https://github.com/Megabyteceer/thing-editor (accessed 12.16.22).

liabru, 2023. liabru/matter-js [WWW Document]. URL https://github.com/liabru/matter-js (accessed 1.9.23).

lightning-fs/src at main · isomorphic-git/lightning-fs [WWW Document], 2023. . GitHub. URL https://github.com/isomorphic-git/lightning-fs (accessed 4.18.23).

Lightweight Particle System for TypeScript [WWW Document], 2023. . GitHub. URL https://github.com/Risto-Paasivirta/ParticleSystem (accessed 1.10.23).

Lind, A., 2023. subset-font [WWW Document]. URL https://github.com/papandreou/subset-font (accessed 1.24.23).

List of game engines [WWW Document], 2022. . Wikipedia. URL https://en.wikipedia.org/w/index.php?title=List_of_game_engines&oldid=1124258896 (accessed 11.28.22).

Maggiorini, D., Ripamonti, L.A., Cappellini, G., 2016. About Game Engines and Their Future, in: Mandler, B., Marquez-Barja, J., Mitre Campista, M.E., Cagáňová, D., Chaouchi, H., Zeadally, S., Badra, M., Giordano, S., Fazio, M., Somov, A., Vieriu, R.-L. (Eds.), Internet of Things. IoT Infrastructures, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer International Publishing, Cham, pp. 276–283. https://doi.org/10.1007/978-3-319-47063-4_28

melonjs v14.1.2 | Bundlephobia [WWW Document], 2022. URL https://bundlephobia.com/package/melonjs (accessed 12.16.22).

melonJS [WWW Document], 2022. URL https://melonjs.org/ (accessed 12.16.22).

Miller, J., 2023. dlv: Safe deep property access in 120 bytes [WWW Document]. URL https://github.com/developit/dlv (accessed 1.23.23).

Mustonen, M., 2023. moiman100/web-game-engine-test [WWW Document]. URL https://github.com/moiman100/web-game-engine-test (accessed 1.9.23).

Mustonen, M., 2019. Parhaiten opetuskäyttöön soveltuvan versionhallintajärjestelmän löytäminen. Finding the most suitable version control system for education.

Norynchak, O., 2022. free-tex-packer-core [WWW Document]. URL https://github.com/odrick/free-tex-packer-core (accessed 1.24.23).

Park, H.C., Baek, N., 2020. Design of SelfEngine: A Lightweight Game Engine, in: Kim, K.J., Kim, H.-Y. (Eds.), Information Science and Applications, Lecture Notes in Electrical Engineering. Springer, Singapore, pp. 223–227. https://doi.org/10.1007/978-981-15-1465-4_23

Phaser - A fast, fun and free open source HTML5 game framework [WWW Document], 2022. URL https://phaser.io (accessed 12.13.22).

Phaser Editor 2D | HTML5 Game IDE [WWW Document], 2022. URL https://phasereditor2d.com/ (accessed 12.16.22).

phaser v3.55.2 | Bundlephobia [WWW Document], 2022. URL https://bundlephobia.com/package/phaser (accessed 12.16.22).

PixiJS Particle Emitter [WWW Document], 2023. URL https://github.com/pixijs/particle-emitter (accessed 1.10.23).

PixiJS Sound [WWW Document], 2022. URL https://github.com/pixijs/sound (accessed 12.16.22).

pixi.js v7.0.5 | Bundlephobia [WWW Document], 2022. URL https://bundlephobia.com/package/pixi.js (accessed 12.16.22).

PixiJS [WWW Document], 2022. URL https://pixijs.com/ (accessed 12.16.22).

Plans and pricing [WWW Document], 2022. URL https://store.unity.com/ (accessed 12.8.22).

PlayCanvas - The Web-First Game Engine [WWW Document], 2022. . PlayCanvas.com. URL https://playcanvas.com (accessed 12.9.22).

Popmotion: The animator's JavaScript toolbox [WWW Document], 2023. URL https://popmotion.io/#quick-start-animation-animate-playback-controls (accessed 1.24.23).

Rekapi [WWW Document], 2023. URL http://jeremyckahn.github.io/rekapi/doc/index.html (accessed 1.24.23).

Samuele, 2022. How To Serve a Local Folder of Files in Your Browser [WWW Document]. Strani Anelli. URL https://blog.stranianelli.com/how-to-serve-a-local-folder-of-files-in-your-browser/ (accessed 2.4.23).

Shakiba, A., 2023. Planck.js [WWW Document]. URL https://github.com/shakiba/planck.js (accessed 1.9.23).

Shifty [WWW Document], 2023. URL http://jeremyckahn.github.io/shifty/doc/index.html (accessed 1.24.23).

Simpson, J., 2023. Howler [WWW Document]. URL https://github.com/goldfire/howler.js (accessed 1.10.23).

TexturePacker - Create Sprite Sheets for your game! [WWW Document], 2023. URL https://www.codeandweb.com/texturepacker (accessed 4.18.23).

Twine / An open-source tool for telling interactive, nonlinear stories [WWW Document], 2022. URL https://twinery.org/ (accessed 12.8.22).

Unreal Engine Powers Film & Television Production [WWW Document], 2022. . Unreal Engine. URL https://www.unrealengine.com/en-US/solutions/film-television (accessed 12.8.22).

Unreal Engine (UE5) licensing options - Unreal Engine [WWW Document], 2022. URL https://www.unrealengine.com/en-US/license (accessed 12.8.22).

Valencia-García, R., Lagos-Ortiz, K., Alcaraz-Mármol, G., del Cioppo, J., Vera-Lucio, N. (Eds.), 2016. Technologies and Innovation: Second International Conference, CITI 2016, Guayaquil, Ecuador, November 23-25, 2016, Proceedings, Communications in Computer and Information Science. Springer International Publishing, Cham. https://doi.org/10.1007/978-3-319-48024-4

Vite in the browser - ‹div›RIOTS [WWW Document], 2022. URL https://divriots.com/blog/vite-in-the-browser (accessed 1.23.23).

Warren, J., 2019. Tiny online game engines, in: 2019 IEEE Games, Entertainment, Media Conference (GEM). Presented at the 2019 IEEE Games, Entertainment, Media Conference (GEM), pp. 1–7. https://doi.org/10.1109/GEM.2019.8901975

小弟调调, 2023. Hotkeys [WWW Document]. URL https://github.com/jaywcjlove/hotkeys (accessed 4.5.23).