Performance and architecture optimization in an HTML5-based web game

Corentin Bras

Tutor, Aseel Berglund
Examinator, Henrik Eriksson
Copyright

The publishers will keep this document online on the Internet – or its possible replacement – for a period of 25 years starting from the date of publication barring exceptional circumstances.

The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/hers own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: http://www.ep.liu.se/.
Abstract

Web applications are becoming more and more complex and bigger and bigger. In the case of a web game, it can be as big as a software. As well, as these applications are run indirectly through a web browser they need to be quite well optimized. For these reasons, performance and architecture are becoming a crucial point in web development. This report covers some method to improve the architecture of a javascript application by using design patterns. It will also present some methods to optimize the performance of javascript code and some way to do some of these changes automatically. Applying these changes allows us to increase the framerate of the web game by 50% and to reduce the loading time of the game by more than 72%. The usage of design patterns such as the observer, the composite and the builder pattern permits to improve the maintainability, extendibility and flexibility of the code but also allows to implement new features.
Acknowledgments

I would like to thank Aseel and Erik Berglund for the opportunity of doing this thesis work and their help and support.
1 Introduction
1.1 Motivation
1.2 Aim
1.3 Research questions
1.4 Delimitations
1.5 Background

2 Theory
2.1 Phaser framework
2.2 Framerate
2.3 Improving a web game
  2.3.1 Improving the Javascript Architecture
  2.3.2 Improve performance of Javascript code
2.4 Code quality
2.5 Continuous integration
  2.5.1 Jenkins

3 Method
3.1 Improve the Architecture of the game
  3.1.1 Decrease the coupling between the logic and the UI of the game
  3.1.2 Create a new system for managing themes in the game
  3.1.3 Create an architecture to create easily elements in the game
3.2 Improve the performance of the game
  3.2.1 Follow the evolution of the game performance
  3.2.2 Metrics used in order to evaluate the game performance
  3.2.3 Improve the time allocation of the motion game
  3.2.4 Use reverse loops
  3.2.5 Put all the sprites in spritesheets
  3.2.6 Minify javascript code
  3.2.7 Served compress files
3.3 Automatize performance improvement
  3.3.1 Automation of the sprite sheets creation
  3.3.2 Implement a continuous integration process
4 Results

4.1 Follow the evolution of the game performance .............................................. 18
4.2 Improve the Architecture of the game ............................................................... 19
  4.2.1 Decrease the coupling between the logic and the UI of the game .......... 19
  4.2.2 Create a new system for managing themes in the game ...................... 21
  4.2.3 Create an architecture to create easily elements in the game ............... 22
4.3 Improve the performance of the game ............................................................. 23
  4.3.1 Improve the time allocation of the motion game ................................. 23
  4.3.2 Use reverse loops ......................................................................................... 24
  4.3.3 Put all the sprites in spritesheets ............................................................... 25
  4.3.4 Minify the javascript code .......................................................................... 25
  4.3.5 Served compress files .................................................................................... 26
4.4 Automatize performance improvement ............................................................ 27
  4.4.1 Automation of the sprite sheets creation ................................................. 27
  4.4.2 Implement a continuous integration process ........................................... 27

5 Discussion

5.1 Results ............................................................................................................... 29
  5.1.1 Architecture improvement ........................................................................ 29
  5.1.2 Performance improvement ......................................................................... 30
  5.1.3 Automatize the code improvement ............................................................ 32
5.2 Method .............................................................................................................. 32
  5.2.1 Architecture improvement ........................................................................ 32
  5.2.2 Performance improvement ......................................................................... 33
5.3 The work in a wider context ............................................................................ 33

6 Conclusion ........................................................................................................... 34

Bibliography ............................................................................................................ 35
# List of Figures

1.1 An image from the game ................................................. 3
1.2 The night theme ......................................................... 3

2.1 The observer pattern in "Design patterns, Elements of Reusable Object-Oriented Software" [Gamma1995] ........................................ 7
2.2 The composite pattern in "Design patterns, Elements of Reusable Object-Oriented Software" [Gamma1995] ........................................ 7
2.3 The builder pattern in "Design patterns, Elements of Reusable Object-Oriented Software" [Gamma1995] ........................................ 8

3.1 Sprites from a bug ....................................................... 13
3.2 A bug ................................................................. 13
3.3 An example of frame ................................................... 15

4.1 A frame in the standard theme ........................................ 18
4.2 A frame in the night theme ........................................... 18
4.3 A frame with the winter and the night theme ......................... 19
4.4 UML of the new UI system ........................................... 19
4.5 Drawing of the subscription process on the score subject .......... 20
4.6 Drawing of the notification process on the score subject .......... 20
4.7 UML of the new theme system ....................................... 21
4.8 The Night Theme, the Winter Theme and the Winter-Night Theme ........................................ 22
4.9 UML of the new bug creation system ................................ 22
4.10 The old readPixel call ................................................. 23
4.11 The new readPixel call ............................................... 23
4.12 A sample of code minified .......................................... 25
List of Tables

1.1 Specifications of the test computer ................................................. 2

4.1 Effects of the change of the read pixel function ................................. 24
4.2 Comparison between normal and reverse loop on different browsers ........ 24
4.3 Effects of the change of the loop by reverse loops .............................. 24
4.4 Effects of the sprite sheets on the game ......................................... 25
4.5 Effects of minifying the javascript .................................................. 26
4.6 Effects of the file compression ....................................................... 26
4.7 Effects of the improvements .......................................................... 27
1 Introduction

In the world, more and more application are online. It is possible to communicate with a friend, to share and store picture, read email using a webmail, play video games and more. All these things can be done using only a web browser. That makes web application quite different from desktop or mobile application. In effect, instead of programming the frontend of your application for an Operating System, you have to develop it for a web browser. This way when you execute a web app, the execution is done by your web browser, which is executed by your operating system. That could create some performance issues. In the case of a web game, a lack of performance can be a critical thing.

As developers want to create more web games some framework like Phaser [18] have been developed in order to help them. As well, with the development of motion gaming, more and more motion web games are created. As these games have to analyze and treat the movement of the player, they need a lot of resources.

The goal of this thesis will be to find a way to improve the performance and the architecture of a motion web game during the development phase of the game.

1.1 Motivation

Web-based games are really interesting. As you only need a web browser to play them, the user does not have to download an external app and to install it. He just has to go on the website and plays the game.

This way web based game only need a computer and an Internet access to be played. That makes them really accessible. In fact, web games can be played by a lot of people. According to the Internet live stats "Around 40% of the world population has an Internet connection" which represents about three billion people [17].

Unfortunately, web games are executed by the web browser instead of being directly executed on the processor and the graphic card. In the case of motion games, as you also have to treat the movement of the player, in some case, the processor and the graphic card can be overload. As a result, motion web games can need a lot of resources and by extrapolation a performant computer.
1.2. Aim

As people have really different hardware they may not be able to play to all these web games. This way, it should be interesting to find ways to increase performance and resources management in web based games in order to allow more people to play more web games and more complex web games.

As well, with the development of the agile method, it would be interesting to integrate these performance and architecture improvement in the Agile process.

1.2 Aim

The goal of this thesis is to find how to increase performance and architecture of a motion web game in an agile development process. The most important part of this thesis will be to find a way to improve the performance and the architecture of a web game to make it more extendable and more maintainable.

1.3 Research questions

In order to do that we will answer to these questions:

- How can we improve the performance of a web game according to framerate and loading time measurement?
- How can we improve the architecture of a web game to gain maintainability and extensibility?

By answering these questions it will be possible to know good ways to improve the architecture and the performance of a motion web based game.

1.4 Delimitations

This thesis will be focused on studying a motion web-based game developed with the framework Phaser. The performance test will be done on Google Chrome as it’s the first web browser according to the US government[24]. As well Google Chrome offer interesting features in order to analyze the performance of a website. Also for technical reasons all the test will be done on this computer:

<table>
<thead>
<tr>
<th>Computer model</th>
<th>MSI GT70 2PC Dominator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>Intel Core i7-4810MQ CPU @ 2.80GHz</td>
</tr>
<tr>
<td>RAM</td>
<td>8GB</td>
</tr>
<tr>
<td>Graphic card</td>
<td>nvidia GeForce GTX 870M</td>
</tr>
<tr>
<td>Operating system</td>
<td>Windows 10 64-bit</td>
</tr>
</tbody>
</table>

Table 1.1: Specifications of the test computer

1.5 Background

In this thesis, the focus will be done on a 2D motion web game developed with the open source framework Phaser. In this game, the player has to squish bugs on a screen by moving his hands and his body. The bugs have to be squished before they reached the tree in order to avoid a game over. Water buckets are also going to the tree in order to replenish him. The player has to let them pass if he squishes to many buckets he will also get a game over. When you squish a bug you get points. The goal of the game is to get as many points as
possible.

Figure 1.1: An image from the game

On this image you can see the game with the tree, water buckets and bugs. You can see on the upper left corner the user interface with the number of points, the time left, the number of lives left (the number in the water drop) and the current combo (you can do combo by squishing bug of the same color). On the lower left corner there are the buttons to access to the level choice menu, the shop and the settings.

The player will also have the possibility to buy new themes in order to play the game with new game mechanics. For example here is the game with the night theme:

Figure 1.2: The night theme
On this theme, the player can only see the level around bugs, water buckets and the tree.

There are different levels in the game separated in different chapter. Depending on the chapter the difficulty of the game will change and some specials bugs will be added. For example, the freeze bug slows down the bugs and the water buckets.
2 Theory

2.1 Phaser framework

Phaser is an HTML5 game framework created in order to develop web-based games more easily [18]. In order to develop on Phaser, the language used is JavaScript. Phaser allows most of the features of a classic game engine like physics, preloading, particles, sprites, camera but also some features linked to the web development like device scaling or compatibility for a mobile web browser.

2.2 Framerate

The framerate is a metric which permits to know the refreshing rate of the screen. In order to do that, we calculate the frame rate in FPS, Frame Per Second. For example, a frame rate of 60 FPS means the screen is refreshed 60 times per second. As a result, the higher is the framerate the more the game is fluent. According to J. Chen and J. Thropp studies [4] in order to be performant enough on most of the tasks you should play at least at 15 FPS. A study about "Video acceptability and Frame Rate" [1] says that at 15 FPS a video is watchable for 80% of the population. It's only at 30 FPS that the video is watchable by everybody. This way, in order to get everybody having a good watching experience a minimal frame rate would be 30FPS. To finish with, according to M. Claypool, K. Claypool, and F.Damma study [5], their survey shows that the quality of a game increase with the frame rate but the quality of the game increase in a logarithmic way. Between 15 and 30 FPS, the difference in the score is quite significant. In parallel, the difference between 30 and 60 FPS is still present but not so important.

According to these results, 15 FPS is enough to play the game so it should be the minimal framerate we should observe on a slow computer. On the majority of the computer, a minimum of 30 FPS should be observed in order to play the game in good condition. For the best hardware configuration, a minimal Frame Rate of 60 FPS would be the best as the majority of the screen refresh 60 times per second.

2.3 Improving a web game

A complex web application is usually composed of different parts.
2.3. Improving a web game

- A markup language in order to structure the application. In the project, HTML5 defines by the W3C [12].
- A style sheet language to do the presentation of the document. In the game, CSS3 defines by W3C [8].
- A scripting language to make the web application dynamic and make it evolves depending on the user actions. Javascript is the one used in the game. This language has been developed first in 1995 by Brendan Eich. The specifications are written in the Standard ECMA-262 [13].

For a web game, there is only a few HTML and CSS code. In fact, in the studied game, a canvas HTML markup is defined and javascript is used in order to draw in the canvas and to refresh it each frame. This way, improve the web game is equivalent to improve the javascript of the game. The goal will be to improve the effectiveness, the maintainability, the extendability of the web game and reduce the potential defect of the game. To do these improvements, two aspects will be focused on, the architecture of the javascript and the javascript code.

2.3.1 Improving the Javascript Architecture

Javascript is an object oriented programming language. To improve the architecture, object oriented architecture optimization can be done.

In order to get a good object oriented design, Five basic principles also called SOLID have been defined by Robert C. Martin [22].

- Single Responsibility Principle, a class should have one and only one reason to change, if this reason is not to fulfill you shouldn’t change the class.
- Open Closed Principle, you should be able to extend a classes behavior without modifying it.
- Liskov Substitution Principle, derived classes must be substitutable for their base classes. The derived classes can be call anywhere the base classes are used.
- Interface Segregation Principle, make fine grained interfaces that are client specific. It is better to have multiple small interfaces than only a big one.
- Dependency Inversion Principle, depend on abstractions, not on concretions. A class should use abstract implementations of other classes instead of concrete implementations.

By applying these principles, the software should be easier to maintain, extend and understand. A good way to apply these principles to an existing code is to refactor the code of the existing application. As well, by refactoring the application it is possible to apply some good practices in the architecture like the use of design patterns [9]. In fact, the use of design patterns usually makes a software easier to maintain, make the code more flexible so more extendable and make the code less faulty [25].

The Observer Pattern

The goal of this design pattern is to add many dependencies to one object. If this object changes all its dependents are automatically updated.
2.3. Improving a web game

Figure 2.1: The observer pattern in ”Design patterns, Elements of Reusable Object-Oriented Software” [9]

To use this design pattern on an object, this object as to be a ConcreteSubject, the elements which want to observe the subject have to attach an observer to this subject. When the subject state changes, the notify function is called and will call the update function of all the attached Observer. This way, the observerState will always be the same as the subjectState. Some consequences of this design are [9]:

1. Abstraction of the coupling between the Subject and the Observer. The observer does not know the concrete class of the subject. This way the coupling between the observer and the subject is minimal.

2. This pattern use a broadcast system. The subject does not care about the number of Observer, it just broadcast the information to the interested observers. In that way, you can add and remove Observer without impacting the subject.

3. The last point is that observers do not know each others. At one point is pretty good as it allows to reduce quite a lot the coupling in the application. But it can also cause some issue if some observers can change a subjectState, we can create an infinite loop.

The Composite Pattern

This design pattern allows treating individual objects and compositions of objects uniformly. It also permits to create hierarchies in the object.

Figure 2.2: The composite pattern in ”Design patterns, Elements of Reusable Object-Oriented Software” [9]

This pattern is really useful when you want to use a composition of components as a component. To use this pattern you have to create a CompositeComponent. As it inherits
from Component, you can use it as a Component. You can add the Components you want to your CompositeComponent even a CompositeComponent. If the Operation function is called on the CompositeComponent, it will call it on all the component. Some consequences of this design are [9]:

1. It allows creating hierarchies in the component by adding CompositeComponent to the CompositeComponent.
2. It creates a lot of abstraction for the client. The client can use CompositeComponent and Component in the same way without any issues.
3. Creating new components is pretty easy as they just need to inherit the Component class.
4. It creates a really general model. This way, it is pretty hard to restrict some Component to the CompositeComponent.

The Builder Pattern

This design pattern is used when you want to create a complex object which can be represented in different ways.

![Figure 2.3: The builder pattern in “Design patterns, Elements of Reusable Object-Oriented Software” [9]](image)

When a client wants to create a product, it has to create a ConcreteBuilder and he had to link it with a director. After that, the client will call the construct function so that the director will ask the builder to build the different part as it is set up. The client has just to call the GetResult function on the builder to get the product. Some consequences of this design are [9]:

1. You can create different representations of the product by creating multiple builders.
2. It encapsulates the code of the construction and the representation. The builder is only responsible for the representation when the director has just to manage the construction. The client doesn’t need to know how the product is constructed.
3. This process lets a lot of control on the object creation process as you can define the building method in a really precise step by step process.

2.3.2 Improve performance of Javascript code

Javascript optimization is one of the prominent parts of this study. In this study, the Javascript will be optimized by finding a systematic approach in order to optimize it. Some guidelines have been defined in a guide [11]. Applying these guidelines should allow improving the performance of the web application. According to this research applying these guidelines can improve the execution time of the unoptimized code between 10 and 70% depending on
the optimization which looks really good. Another way to improve javascript code is to minify it. Minify the code mean to remove all the useless characters in the code: comments, spaces, reduce the size of the name used by the variable. By doing that, the page can become a lot smaller, and so faster to download. According to Steve Souders minifying javascript can reduce its size by 20% [21]. As well by minifying javascript, all the javascript can be put in only one file which will allow doing fewer HTTP requests. A lot of tools exist in order to do this work but Google Closure Compiler looks to be a really good choice as it does some performance optimization [6] as well, Google closure compiler is maintained by a major actor in web development and used by thousands of people, this way this tool should be reliable.

**Google Chrome Developers tools**

Google Chrome DevTools are a set of web authoring and debugging tools built into Google Chrome [devtools]. This debugger is composed of multiple panels which can be used in order to debug a web application and get information about the performance of it. In this thesis, two panels will be mostly used: The network panel This panel allows observing the loading of a web page. It permits to observe which resource is loaded, how long it takes to load the resources, the size of the resource. The loading time of the page and the size of the web page are also measured by this panel. To finish, the bandwidth can also be throttled by this panel in order to do the test with a constant bandwidth. The timeline panel The timeline panel can record the javascript event of a page and display them as a timeline. For each event, you can see is execution time, when it happens but also the stack of its function call and their execution time. In this panel, an option can be enabled in order to display the framerate of the application.

**Metrics used in order to evaluate the game performance**

In order to measure the evolution of the game performance, some metrics can be useful. Frame calculation time The frame calculation time is the time needed by the game to refresh a frame. In javascript, it starts when the requestAnimationFrame function is called and end when this function is finished. This metric can be obtained using Google Chrome DevTools timeline panel 2.3.2.

**Frame rate**

The frame rate is the number of frames in one second 2.2. This metric can be obtained using Google Chrome DevTools timeline panel 2.3.2. Theoretical framerate The theoretical framerate is the framerate maximum with a given frame calculation time.

\[
\text{Theoretical framerate} = \frac{1000}{\text{Framecalculationtime}}
\]

This metrics can be obtained using the frame calculation time. Size of a file This metric is the size of a file on a data storage device. It can be obtained by looking at the properties of a file on an operating system explorer or by using Google Chrome DevTools network panel 2.3.2.

**Loading time**

The loading time is the time needed by the browser to load a file or a group of files. It can be measured by using Google Chrome DevTools network panel 2.3.2.

2.4 Code quality

Code quality is really interesting in order to evaluate an architecture. In order to evaluate the code quality, Bansiya J., and Davis C.G. have designed a model for object quality design
They defined six different quality attributes which characterize a software. We will focus on four of them.

- Understandability to make the game easier to maintain.
- Extendability to make the game easier to evolve.
- Flexibility, a part of the game can be easily modified.
- Effectiveness because the game should work well and fast.

They define also some design properties which characterize these quality attributes. In order to improve the quality attributes defined upper, the improvement should be focused on abstraction, encapsulation, cohesion, composition, coupling, inheritance and polymorphism.

### 2.5 Continuous integration

Continuous integration is a group of practices allowing to control the source code in order to verify that the modifications do not create regression in the application. The main interest of the continuous integration is to automatize some task as the compilation, unit testing and performance testing. The most common workflow for that kind of for that method is: - Developers work through a version control system like git or CVS. - Some tests are developed for each feature of the application during his development. - A continuous integration tool like Jenkins is linked to the version control system repository and have the knowledge of the test. - When a modification is done on the repository is done the continuous integration tool execute the tests. - If the test phase fails, the tool sends an alert to the developer and keep in memory the old stable version. This way, the developers are less likely to be surprised by a bug after having put the application online. As well, the continuous integration tool can also be used for automatizing some process like putting the code in production.

#### 2.5.1 Jenkins

Jenkins is an open source continuous integration tool[14]. It can be directly linked to a git repository and build it. During the build, it can apply tests and scripts. As well, the same Jenkins can manage different projects with different builds. As well, Jenkins is also a continuous delivery system[2]. That mean that Jenkins can also deliver the product directly. This way Jenkins is a really complete tool that can be used really efficiently.
3 Method

This chapter will be focused on the method used in order to improve the architecture, the performance and to implement a continuous integration system.

3.1 Improve the Architecture of the game

The game has some architectural issues, in order to add new features and to modify the game some parts need to be more flexible and more extendable. To do that in an efficient way design patterns looks to be good to use. In fact, they allow making website more usable for users and maintainable for developers [23]. As well, design patterns are elements of reusable object-oriented software [9] so a lot of developers have already used them and are used to modify or extend them. To finish design patterns define a general structure so their use will allow this study to be replicated more easily as documentation can be found in order to replicate them like in the book "Design Patterns Elements of Reusable Object-Oriented Software" [9].

So four main reasons have motivated the choice of using design patterns:

- They are tested solutions (a lot of people have used them before)
- They are common to most of the developers
- They are documented in some books and websites
- They are flexible and reusable

3.1.1 Decrease the coupling between the logic and the UI of the game

The coupling of the logic of the game and the UI (User Interface) is really strong that means that every change in the game can impact the UI and every change in the UI can impact the game.

As well, in the future:

- The UI may be totally changed.
- New information may be displayed to the user.
• Some components might be removed.

For these reasons, the architecture should be refactored. An adapted design pattern to do this should be the observer pattern. If the user interfaces observe the game, it should only be an abstract coupling between the game and them so the game logic would be separated from the UI. As well, the broadcasting system of the observer pattern should allow to add and remove UI dynamically.

3.1.2 Create a new system for managing themes in the game

An interesting feature of the game is to let the player be able to play different themes. Each theme can:

• add new behavior and new types of bugs to the game.
• change the graphics of the map.
• change the bugs graphics, hitbox, and speed.

These themes are currently defined in conditional structures which look like that:

```java
switch (theme) {
    case "night":
        playNight();
        break;
    case "desert":
        playDesert();
        break;
    .
    .
    .
}
```

These conditional structures are used before the launch of the game in order to prepare the theme, at each frame update and when the bug graphics and the map graphics are set up.

In the future, a lot of themes should be added. As well, the user should pay for new themes so the code of all the theme should not be needed to run the game. This way it should be interesting to refactor the theme system to reach these goals:

• create a general structure where all the themes can be called the same way.
• make the game simple. It should be able to call the theme in a simple way without checking any condition.
• make themes independent from the game. They should not be needed to play the game.
• new themes should be easy to create and integrate to the game.

The composite design pattern looks adapted to these goals. As well, the use of this design pattern may allow using multiple themes at the same time which should be a really interesting side effect in order to mix game mechanics.
3.1.3 Create an architecture to create easily elements in the game

In the game, bugs and water buckets are elements created from a sprite. In order to create various types of bugs and buckets and add more diversity to the games elements, it should be interesting to be able to create elements from multiple sprites and to be able to add variation to these sprites (different animations, some variation in the color, different type of legs and body). The base objectives would be to get sprites like that:

![Sprites from a bug](image1)

Figure 3.1: Sprites from a bug

and to be able to merge them in order to get a bug like that:

![A bug](image2)

Figure 3.2: A bug

In order to do that building process and to allow variations on the objects creation, the builder design pattern looks to be adapted to this situation. By creating multiples builder, multiple bugs can be created. As well, the structure of the builder allows adding variations on the element created and to be able to create various bugs with the same class.

3.2 Improve the performance of the game

The second part of this thesis will be focused on improving the performance of the game. To do so the performance of the game will be measured before and after each modification.

3.2.1 Follow the evolution of the game performance

Follow the evolution of the game performance is really important in order to see how modifications change it. In fact, it allows knowing if a modification improves or decreases the performance of the game. In order to observe these modifications, the easiest way is to use tools such as Google chrome developer tools after each modification to see how the software is evolving. More precisely, the profiling [19] allows following the performance of a web
application. This tool will be used a lot to observe how the game evolves by observing the evolution of the metrics.

3.2.2 Metrics used in order to evaluate the game performance

These metrics will be obtained by using Google chrome developers profiling system [19]. Each metric will be measured 10 times and an average of the results will be taken.

Frame calculation time

The frame calculation time will be obtained by using the timeline tool. It allows showing how the game is executing over time. By doing an average of multiple frames execution time, a relevant frame calculation time can be obtained. It will be measured in milliseconds (ms).

Framerate

The profiling tool allows printing the framerate. This framerate will be observed during 10 seconds in order to get an average value. The framerate will be measured in frames per seconds (fps). On a javascript application, the framerate is limited to the refreshing rate of the screen [requestFrame], on the test computer, it is 60fps.

Theoretical framerate

The theoretical framerate will be calculated using the frame calculation time:

\[ \text{Theoretical framerate} = \frac{1000}{\text{Frame calculation time}} \]

The theoretical framerate is quite interesting as it allows to see what can be the framerate without Phaser limits and the readPixels call (explained in 3.2.3).

Size of a file

The size of a file will be obtained using the operating system explorer by looking at the properties of the file.

Loading time

The loading time will be measured using the network tab of Google chrome developers tools as it allows to measure how many time is needed to load the web application but also as it allows throttling the bandwidth and this way obtain always the same loading time for a given version of the game. In order to have consistent results during the loading tests, the bandwidth will be throttled at 1.5Mb/s.

3.2.3 Improve the time allocation of the motion game

As Phaser limit the framerate at 60fps, there is some unused time between two frames.
3.2. Improve the performance of the game

Some calculations do not have to be necessarily done in the frame and can be done outside the frame. For example, the readPixels function is a WebGL function used in order to read the pixel from the camera shader to do the collision with bugs and water buckets. The execution time of this function can vary a lot between two computers. On the test computer, it takes about 30ms when on another computer it just needs 1ms. This function does not necessarily need to be called in the frame so it should be interesting to see how does the performance of the game evolve if we call it outside the frame.

3.2.4 Use reverse loops

On the Internet, a lot of people says that reversing a loop improved quite a lot the performance of javascript code [20]. In fact, when you reverse a loop you just have to check that a variable is true or false after each iteration instead of comparing it with another value. So it should be interesting to test if this optimization is relevant on the game. Reversing a loop mean that you decrement your index instead of increment it.

For example:

```javascript
var array = [];
for(var i=0;i<1000;i++)array.push(i);
```

become after reversing:

```javascript
var array = [];
var i = 1000;
while(i--)array.unshift(i);
```

The tricky part of using reverse loops is when arrays are manipulated as lists. Four functions can be used to manipulate an array as a list:

- push(element) to add an element at the end of an array.
- pop(element) to remove an element at the end of an array.
- unshift(element) to add an element at the beginning of an array.
- shift(element) to remove an element at the beginning of an array.
3.3 Automatize performance improvement

When these functions are present in a loop you want to reverse you have to be really careful about your change but usually you can replace all the push by unshift, unshift by push, all pop by shift and reciprocally all shift by pop.

3.2.5 Put all the sprites in spritesheets

Spritesheets are images composed of multiple sprites. The sprite sheet is linked in the game case to a JSON file which describes where are the sprite on the sprite sheet. The use of sprite sheet has apparently good effects on the load time but also on the framerate [7]. This way it should be interesting to watch how the use of sprite sheet impact the game performance.

3.2.6 Minify javascript code

Minify javascript should reduce its size[21]. In order to do that, a minifier have to be used. In that thesis, Google closure compiler will be used. As Google is one of the most important company in term of web development its compiler should be reliable. As well, Google closure compiler optimize code and can merge javascript files [6]. This way, it should be interested to test it on the code and to see what are the impact of this tool on the performance.

3.2.7 Served compress files

To finish on the performance optimization, the server as to serve files as compressed as possible. In fact, the smallest is a file the fastest it will be for the client to load it. To do that, it is important to check that all the assets in the game are compressed. In the game, the audio files are in WMA which is a really uncompressed format. It should be interesting to compress them in a smaller format. To be able to read them two possibilities are left mp3 or Ogg to avoid license issue in the future, Ogg has been chosen. As well, configuring the server to serve gzip files can impact a lot on the loading time [21].

For each change describe above each metric will be measured and compare between the old version the new version and the percentage of improvement will be calculated.

3.3 Automatize performance improvement

Some of the performance improvement (minify javascript and generate the spritesheets) have to be done after each modification of the code. As well, we want to be able to observe the evolution of the game performance over the time. To do that implementing an automatic process to do the performance improvement and to test the game should be interesting.

3.3.1 Automation of the sprite sheets creation

Create sprite sheets can take a lot of time by using classic tools. The goal of this part would be to find a tool which can create a sprite sheet by using the sprite from a directory with just a command line. Two tools looks to be adapted to do that:


It should be interesting to test both of these packers and to observe which packer is the best to create the game sprite sheets.
3.3.2 Implement a continuous integration process

Executing all the actions to have a production version stable can sometimes take time. These actions are always the same so it would be interesting to automatize them. Continuous integration system like Jenkins [14] are adapted to that kind of process. For example, Jenkins can do some performance testing after each modification of the application automatically. Moreover, it can execute the script to minify the Javascript and to create the sprite sheets this way the developers will not have to execute it anymore. To finish with it can also clean the project in order to let only the needed files on the server by removing the files used for development.
4 Results

This part will present how the architectural and code performance changes have affected the game.

4.1 Follow the evolution of the game performance

Thanks to Google developers tools it’s possible to observe how long is a frame but also where the game can be improved.

Figure 4.1: A frame in the standard theme

Here you can observe a frame with the last version of the game on the standard theme. This frame takes 7ms to be calculated. On this figure, you can observe a stack of the function execution. The function call in orange represents a native call, the light oranges are Phaser framework calls and the other ones are the functions called by the game code. By observing this timeline we can see how the code is executed but also which part slow down the calculation of a frame and by extrapolation the frame rate.

For example, here is a frame for the night theme:

Figure 4.2: A frame in the night theme
This frame is 11ms longer than the standard theme. During the update, the NightTheme.update function takes quite a lot of time to be executed. This way to improve the performance of the night theme, this part should be the first to improve. Here is a frame with the night and the winter theme:

![Figure 4.3: A frame with the winter and the night theme](image)

With the combination of both themes, the frame needs 38ms to be calculated but here the NightTheme.update function (in blue on the timeline) is not the slowest part of the code. The WinterTheme.update function is really more costly. This way it should be more interesting in that case to optimize the WinterTheme first.

### 4.2 Improve the Architecture of the game

An important part of the work is to improve the architecture of the game to make it more flexible and extendible. To do that, the use of design patterns has been chosen. As they are tested and documented solutions designed in order to make an architecture more flexible and reusable, it’s interesting to know how they can affect the game.

#### 4.2.1 Decrease the coupling between the logic and the UI of the game

As explain in the method chapter, the goal of this change was to be able to change or add a UI in the game without changing the game logic but also to be able to change part of the game logic without affecting the UI. Here is the UML of the new UI.

![Figure 4.4: UML of the new UI system](image)
Before doing that changes all the UI was directly in the game class. This change reduces the size of the game class logic by one thousand lines, it represents half of the old game class. With the new UI system, the game creates only the MainUI and does not have to care about it anymore. After that, the MainUI will create all the UI for the game. The UI knows the game but the game does not know the UI. So, the user interface can be changed or removed without affecting the game behavior.

In order to make the interface evolved with the game, the observer pattern is used. The game will use a subject when an information may be used in a UI. The UI will be attached to them as observers and be notified if a change occurred. If that happen the user interface will update its components.

Here is an example about how it is working on the score subject:

![Figure 4.5: Drawing of the subscription process on the score subject](image)

When the game is launched, the interfaces which need the score will subscribe to the score subject.

![Figure 4.6: Drawing of the notification process on the score subject](image)

When the score changes, it sends automatically a notification to the interfaces which have subscribed. This way, the interfaces can update the information displayed automatically after each change in the score.

This change also affects the developers. First, it creates some code conflicts as they were working a lot on the game class but after fixing them, the UI part became easier to modify and update. In fact, as the game do not know it, a lot of change are possible without affecting the game. As well, as long you don’t change the subjects, you can change the game logic...
without affecting the UI. This way, this design fulfill the objectives fixed in the method chapter.

4.2.2 Create a new system for managing themes in the game

The goal of that part was to change the architecture to be able to create new themes easily without having to change the game class. Also, we want to get a control on the theme that the player can play. In order to do that, the composite design pattern is really adapted.

Here is the UML of the new architecture.

![UML of the new theme system](Figure 4.7: UML of the new theme system)

First, the composite design pattern is used. There are two operations: prepare and update. The prepare function is used when the game start and the update function is used on each frame of the game. To make the composite design pattern works, they are defined like that in the CompositeTheme code:

```javascript
//Update
CompositeTheme.prototype.update = function {
    this.themes.forEach(function(theme){theme.update();});
}
//Prepare
CompositeTheme.prototype.prepare = function {
    this.themes.forEach(function(theme){theme.prepare();});
}
//Where this.themes is the list of theme in the composite class
```

This way each theme in the CompositeTheme object can execute is own prepare and update function independently. This code has one side effect. It allows running multiple themes at the same time. This is not bad as it allows to play different themes at the same time and add new mechanics but sometimes one theme should be executed before one other. For this reason, each theme has a priority parameter, the theme list can be sorted by priority so the same group of themes will always be executed in the same way.
4.2. Improve the Architecture of the game

4.2.3 Create an architecture to create easily elements in the game

The builder design patterns have been used to set this functionality up. The main interest in this design pattern is that it allows creating an object step by step. In our case, the goals were to be able to create elements in the game using various sprites, to have a really precise control on the creation and to be able to create a lot of new elements easily.

By using a builder we can set up a patent for an element by coding how to build the element. Also, it allows creating some variation in the element depending on parameters. As well by creating other builders we can create new elements quite easily.

Here is the UML of the new functionality:

![UML of the new bug creation system](image)

The goal of the director would be here to launch the add sprites and add animations function when we want to build a drop graphic or a bug graphic. As well, the PathFollowerGraphics objects can have multiple sprites so we can create the element we want with the builder. We can also add animations on these different sprites. After setting this functionality up, the people working on elements creation manage to create various elements quite easily. As well if some new behaviors or new animations are needed on an element, we can easily
change the code which creates the element. Currently, it never takes more than 30 minutes to add a new behavior. As well, you can use random function in the builder in order to randomize the bug graphic. This way this architecture allow creating an infinite amount of bugs.

4.3 Improve the performance of the game

The performance improvement have been done for two main reasons. First, the game have to be more fluent. This way, it’s important to improve the framerate of the game by reducing the time needed to calculate frames. After that, the loading time has to be reduce in order to allows the player not to wait to play the game.

4.3.1 Improve the time allocation of the motion game

This modification does not affect the execution time of the code but it allows to make its time allocation better. To explain that here are the two timelines of a read pixel call before and after the change:

![Figure 4.10: The old readPixel call](image)

![Figure 4.11: The new readPixel call](image)

On the new system, the time between two frame rendering is shorter when readPixel is called. In fact, the sleep time between two frames is, with the new version, used by readPixel. On the old version people sometimes feel a small freeze when readPixel was called, now the freeze can still be felt but he is less annoying and happen less often. According to this timeline, during the freeze the game was running at 21fps, with the new version during the freeze the game is still running at 37fps which are more acceptable.
4.3. Improve the performance of the game

Between both version of the game, by changing the time allocation of the game, the frame calculation time is the same but there is a 20% improvement on the framerate. Unfortunately, this change depends on the performance of the computer. In fact if the computer is quite slow, they will be no unused time between two frames so any improvement will be noticeable in term of performance by changing the time allocation as there is no time left for the readpixels function.

### 4.3.2 Use reverse loops

Reversing all javascript loops looks to be a really strange improvement so it would be interesting to benchmark this code.

```javascript
var test = [];
for(var i = 0; i<100000; i++) test.push(i);

//normal loop
var i = test.length;
var temp = Date.now();
for(i=0; i<test.length; i++) test[i]=i;
console.log(Date.now()-temp);

//reverse loop
temp = Date.now();
var i = test.length;
while(i--) test[i] = i;
console.log(Date.now()-temp);
```

Here the first log will be the time execute a normal loop and the second one will be the time needed to execute a reverse loop.

In this table you can see the results for the different browsers:

<table>
<thead>
<tr>
<th>Web browser</th>
<th>Normal loop</th>
<th>Reverse loop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google Chrome v49</td>
<td>144 ms</td>
<td>107 ms</td>
</tr>
<tr>
<td>Google Chrome v50 (test.length = 10 000 000)</td>
<td>33ms</td>
<td>33ms</td>
</tr>
<tr>
<td>Mozilla Firefox</td>
<td>138 ms</td>
<td>102 ms</td>
</tr>
<tr>
<td>Microsoft Edge</td>
<td>55 ms</td>
<td>33 ms</td>
</tr>
</tbody>
</table>

Table 4.2: Comparison between normal and reverse loop on different browsers

According to these results, the use of a reverse loop instead of a normal loop results in an improvement between 30 and 40%. After replacing all the loop in the game, there is some change on the game performance:

<table>
<thead>
<tr>
<th>Reverse loop</th>
<th>Old</th>
<th>New</th>
<th>Percentage of improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame calculation time (avg)</td>
<td>15ms</td>
<td>12ms</td>
<td>20%</td>
</tr>
<tr>
<td>Framerate</td>
<td>60</td>
<td>60</td>
<td>0%</td>
</tr>
<tr>
<td>Theoretical framerate</td>
<td>66</td>
<td>83</td>
<td>25%</td>
</tr>
</tbody>
</table>

Table 4.3: Effects of the change of the loop by reverse loops
4.3. Improve the performance of the game

The frame calculation time is improved by 25% on the new version. The framerate doesn’t change, this is caused by Phaser which limits the framerate to 60fps but if the theoretical framerate is calculated (1000 / frame calculation time), the framerate is improved by 25%.

4.3.3 Put all the sprites in spritesheets

This modification affects a lot the game performance as it divides the size of the sprite assets. It implies that all the asset calls in the game have to call sprite sheets instead of assets. They were a major issue by doing that change. Developers usually forget to change the sprite sheet when they add a new sprite. That can create some errors in production when we remove the sprite and keep only the sprite sheets. That can also result in some loss of time in order to add the sprite to the sprite sheet and in changing the call of the sprite in the code.

This change affects a lot the performance:

<table>
<thead>
<tr>
<th>Spritesheets</th>
<th>Old</th>
<th>New</th>
<th>Percentage of improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame calculation time (avg)</td>
<td>13ms</td>
<td>12ms</td>
<td>7%</td>
</tr>
<tr>
<td>Frame rate</td>
<td>60</td>
<td>60</td>
<td>0%</td>
</tr>
<tr>
<td>Theoretical framerate</td>
<td>76</td>
<td>83</td>
<td>7%</td>
</tr>
<tr>
<td>Size of the image assets</td>
<td>13Mo</td>
<td>7.6Mo</td>
<td>41%</td>
</tr>
<tr>
<td>Loading time of the image assets (1.5Mb/s)</td>
<td>51s</td>
<td>19s</td>
<td>62%</td>
</tr>
</tbody>
</table>

Table 4.4: Effects of the sprite sheets on the game

The use of sprite sheets reduces a little the frame calculation time and reduces a lot the loading time. First, using them allow to compress the assets size and reduce it by 41%. On his side, the loading time is reduced by 62% this 8% difference are due to the number of HTTP requests send. In fact, each HTTP requests need a certain amount of time to be done. By putting many sprites in one sprite sheets we have only one HTTP request instead of multiple requests.

4.3.4 Minify the javascript code

That process should only reduce the size of the javascript file and affects the loading time of the game but Google closure compiler also applies some optimization and can merge files together.

In order to see what kind of modifications Google closure compiler can do, a sample of code have been minified:

```javascript
//This is a comment
var j = 1 + 3;
for(var i = 1+3; i<6; i++) {
    console.log("test");
}
var debug = false;
if(debug)
    console.log("hello world!")
```

Figure 4.12: A sample of code minified
4.3. Improve the performance of the game

Some modifications can be observed:

- The comment line is removed
- 1+3 become 4
- j is now initialized in the loop and need less space to be initialized
- The brackets for the loop are removed
- false is replaced by !1
- the if(debug) line equivalent to if(false) is removed

So Google closure compiler do some optimizations, it removes useless parts of the code and also do some optimizations like addition and some other stuff like using !1 instead of false. Let’s see what happens on the game performance if its code is minified.

<table>
<thead>
<tr>
<th>Minifying javascript</th>
<th>Old</th>
<th>New</th>
<th>Percentage of improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame calculation time (avg)</td>
<td>18ms</td>
<td>12ms</td>
<td>33%</td>
</tr>
<tr>
<td>Framerate</td>
<td>50</td>
<td>60</td>
<td>17%</td>
</tr>
<tr>
<td>Theoretical framerate</td>
<td>55</td>
<td>83</td>
<td>51%</td>
</tr>
<tr>
<td>Size of the javascript files</td>
<td>6.19Mo</td>
<td>1.63Mo</td>
<td>73%</td>
</tr>
<tr>
<td>Loading time of the javascript (1.5Mb/s)</td>
<td>36s</td>
<td>8s</td>
<td>77%</td>
</tr>
</tbody>
</table>

Table 4.5: Effects of minifying the javascript

First, the optimization of Google closure compiler is real. On this game, we gain about 33% on the frame calculation time. As well, minifying the javascript reduce the size of the javascript by 73%. The gain is even more important on the loading time as we gain 77% of the time. This 4% difference is caused by the merging of the files, on load, the browser does fewer HTTP requests so the game load faster.

4.3.5 Served compress files

Big files are longer to load than small files. That’s why it’s better to serve compress files. For example, the use of Ogg or mp3 sounds and music instead of WMA music reduce the size of the audio content by 6. In the bug game case, it results in a reduction of the loading time of 30 sec on a 1.5Mb connection. As well, serving gzip files reduce the size of HTML, CSS and javascript files by 82% (even on minified javascript). Using gzip for images and audio is ineffective it will not reduce the size of your images and audio files but it will need some time to be uncompressed by your browser.

<table>
<thead>
<tr>
<th>Compression</th>
<th>Old</th>
<th>New</th>
<th>Percentage of improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gzip javascript and HTML size</td>
<td>1503Ko</td>
<td>262Ko</td>
<td>82%</td>
</tr>
<tr>
<td>Loading time of gzip javascript and HTML (1.5Mb/s)</td>
<td>8.3s</td>
<td>2.3s</td>
<td>72%</td>
</tr>
<tr>
<td>Size of the audio files</td>
<td>6.1Mo</td>
<td>0.9Mo</td>
<td>85%</td>
</tr>
<tr>
<td>Loading time of audio files (1.5Mb/s)</td>
<td>35.7s</td>
<td>7.8s</td>
<td>78%</td>
</tr>
</tbody>
</table>

Table 4.6: Effects of the file compression

Gzipping the javascript and HTML reduce the size of the download files by 82% but the load time by 72%. In fact, the number of HTTP request is the same between both versions so they take the same amount of time. There is a similar phenomenon when we compress the audio files. Anyway, that kind of changes reduces the load time of the files by about 70%.
All these changes allow the game to be faster and have a shorter load time:

<table>
<thead>
<tr>
<th></th>
<th>Old</th>
<th>New</th>
<th>Percentage of improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame calculation time</td>
<td>22ms</td>
<td>10ms</td>
<td>54%</td>
</tr>
<tr>
<td>Framerate</td>
<td>40fps</td>
<td>60fps</td>
<td>50%</td>
</tr>
<tr>
<td>Theoretical Framerate</td>
<td>45fps</td>
<td>100fps</td>
<td>122%</td>
</tr>
<tr>
<td>Size of the files</td>
<td>24Mo</td>
<td>7Mo</td>
<td>70%</td>
</tr>
<tr>
<td>Loading time of the files (1.5Mb/s)</td>
<td>2min 20sec</td>
<td>38s</td>
<td>72%</td>
</tr>
</tbody>
</table>

Table 4.7: Effects of the improvements

### 4.4 Automatize performance improvement

The automation of the performance improvement has for goal to reduce the number of operations needed by the developer in order to create the production version of the game. Moreover, it should mitigate the possible human errors done during the generation of the production version. To finish, it should be interesting to be able to follow automatically the evolution of the game performance.

#### 4.4.1 Automation of the sprite sheets creation

The automation of the sprite sheets creation system does not affect at all the performance but it allows developers to generate all the sprite sheets in just one command line. After that, developers became more appointed in using sprite sheets. In fact, the use of one command line is easier than to generate a whole sprite sheet manually.

Two tools have been tried on the same sprite sheet libgdx and glue:

- Libgdx does the work but the resulting sprite sheet takes quite a lot of memory (1.2Mb).
- Glue does sprite sheets three times smaller than libgdx (400kb). But does not create the JSON in the good format.

The solution used was to modify glue in order to get it generate the sprite sheets in the good format.

#### 4.4.2 Implement a continuous integration process

Executing scripts to minify javascript, to generate sprite sheets, create a version for the production and following the evolution of the game performance can take a lot of time. It takes around 20 minutes to do all these processes, 6 command lines had to be executed and a check of the performance. So if someone do that every day it represents a loss of 86 hours after one year. As well, if a human did it, he can also do error, forget to execute a script and this way create a nonworking version of the game. Setting up a continuous integration system take sometimes but it allows to control the code, control the performance, execute the scripts and put the new version in production. After setting up Jenkins the 20 minutes work needed only one click to be done. Moreover, it is possible to build the code after each modification on the git repository. This way you can have always the latest version online without doing any operation.

By implementing automation process, if a developer want to regenerate the production version he needs 3 command lines instead of two command lines and twenty operations. Moreover, if he lets the continuous integration tool running, he does not need to do any
automatize performance improvement action to have the latest production version available. The developer can also follow the performance evolution of the game after each change.
5 Discussion

On this part, I will discuss about the results and the methods used in this thesis in order to improve the architecture, the performance of the game but also to automatize some code improvement.

5.1 Results

5.1.1 Architecture improvement

Architecture changes do not look to affect a lot the performance but they are interesting at some points, by doing them you can increase the extendibility, flexibility, and maintainability of the code. Refactoring the code can also allow new features. Besides, it permits to reduce the number of lines of the main part of the code. Separating this code in different files and class make the code more easy to read and change. The other interest in using design patterns is that they are general designs. This way, the experiments done on the architecture are replicable. Depending on the implementation of the design patterns some differences can be observed but the main idea of the design will be the same.

Decrease the coupling between the logic and the UI of the game

Since the implementation of the new architecture, some developers have done changes on the UI without having any issue. In fact, they found easier to have to do a modification on a two hundred lines of code file than a two thousand one. As well, since the change, some new UI have been added without issues. The main problem with this design was in the use of the observer pattern. The developer has to be really careful to some stuff like for example not to create an infinite loop by modifying a subject value inside an observer or not to subscribe to the same subject multiple times. As well, more and more studies try to find alternate patterns to the observer pattern [16]. The choice to use it here was not done in order to do the best possible architecture but to do a good architecture that can be easily explained to any developer. This way, the use of classical design patterns is a good choice has most of the developers have at least heard about them.
Create a new system for managing themes in the game

For that part, I think, the composite design pattern is perfectly adapted. The creation of new components become really faster, all the conditional structures are not useful anymore. To create a new theme you just have to extend the AbstractTheme class and override the function you want. For example, if you do not need an update function you can just override the prepare function. As well, the ability to run multiples themes at the same times enhance the game experience. Since the implementation, two new themes have been created. The themes directly work with the others and currently it is possible to play with all the themes at the same time. Two issues have been observed:

- If the priority is not well set up, themes can be played in a wrong order and this way create a bad graphical result but the game will be working.
- If too many themes are played at the same time, it can lead to creating too many new sprites and graphics and that can slow down the game.

These two problems are minor issues as playing too many themes is more linked with a computer related issue and the priority problem can be fixed quite easily by doing some testing.

Create an architecture to create easily elements

The use of the builder design pattern is also well adapted. Doing that allow creating a lot of variation on each element we can change the color add specific sprite to the elements change the color of a sprite, make two bugs have different animations for the same behavior, etc. This way this design is adapted to that kind of change. After, this change being implemented two people was working on generating new bugs. By letting them a 30 lines documentation about how the code works they manage to create a dozen of bugs.

According to these results, design patterns can really improve the maintainability, extendibility, and flexibility of a web application which is supported by "Design Patterns Elements of Reusable Object-Oriented Software"[9].

5.1.2 Performance improvement

The metrics used to observed the difference between both versions were pretty interesting as they are really concrete. It is also possible to get them quite easily by using some tools like Google chrome developers. The main issue with these metrics is that they depend on the project and the test computer.

Improve the time allocation of the motion game

Reallocating some of the function calls after the frame rendering allows to improve quite well the frame rate. As It permits to do some calculation during the unused time after the frame rendering.

Although, the result can change a lot from a computer to an other. In the game case, some computers need a short time to calculate the readPixel. For them, reallocating the readPixel call after the frame rendering will only have a small impact on the framerate. As well, if the computer is pretty slow and need more than 17ms (frame calculation time needed to be at 60fps) to calculate a frame, they will be no more unused time so this improvement will not affect the game performance. In any case, that kind of change will not reduce the game performance.
Use reverse loops

The reverse loop improvement was not defined in any research paper, that’s why it was really interesting to benchmark it. Surprisingly, it increases the performance of loops on most browsers. As well, this change is related to the javascript engine present inside the browser so similar results should be found on another computer if you use the same javascript engine. For example, a reverse loop was faster on Google chrome v49 but are as fast as a normal loop on Google chrome v50. The improvements done by this change on the frame calculation time are really good on most of the browsers and can be seen on any computer as they are browser dependant.

Although reversing loop can have some effects, it removes some readability to the code as developers are used to using a standard loop. As well, developers can be attracted to do that two kinds of loops:

```javascript
//First bad use of reverse loop
var i = 1000;
var index = 0;
while(i--){
    index++;
    test[index] = index;
}

//second bad use of reverse loop
var max = 1000;
var i = max;
var temp = Date.now();
while(i--){
    test[max - i] = max - i;
}
```

These two ways seem to be reverse loops but can be used as a normal loop. In fact, these loops are as slow as the standard loop. The use of reverse loop is currently useful but may in the future not be as useful. With the evolution of javascript engine performance, the calculation time needed for both types of the loop may become similar. That the case for Google chrome engine. When the benchmark was first done on the version 49, the reverse loop was faster but now on the version 50 both loops are as fast.

Put sprites in sprite sheets

Putting the sprites in sprite sheets reduce the frame calculation time, the size of the image assets and even more the loading time. All these improvements are due to different reasons. The frame calculation time improvement is due to Phaser assets loading system. Phaser loads the asset when it needs it and cache it after. If it has to display a sprite of a loaded sprite sheet it will use the cached sprite sheet and it will be able to use the sprite without loading it which reduce the frame calculation time.

Moreover, when you create a sprite sheet as all the images are merged and as some compression are applied to the sprites, the size of the resulting sprite sheet is usually smaller than the size of all the sprites together. All the tests done with sprite sheets on the game always result in getting sprite sheets smaller than all the sprites in.

As well, the use of sprite sheets allows doing only one HTTP request to get all the images. That make the loading time even faster which is supported by Steve Souders paper [21]. The use of sprite sheets on an application can also result in some issues. Generating sprite sheets take some time so changing a sprite in the sprite sheet can be long for a developer who want to try different sprites.
Minify the javascript code

Minifying javascript is advised to improve the game performance [21]. As well, Google closure compiler promise a reduction of the file size but also better performances [6]. In fact, these assessments are true as the performance of the game were highly increased and the file size reduced. The other interest in using google closure compiler is that it can be used in different ways. Online if you want to test it or by command line if you want to automatize the code compilation and minify multiple javascript files in just one big file. The result of the minification would always be at least as big as the input file (if the input file is already minified) and the resulting javascript as fast as the input as the minifier just reduce some parts of the code and do performance optimization. The main issue with minifying the code is that minified code is really hard to read and modify, this way you need two versions of the code one not minified for the development and a minified one for the production.

Serve compressed files

Compressing music files do not issue any problem. It will reduce a little the quality of the music but it will improve a lot the loading time. As well, gzipping the files in the server will reduce a lot the loading time for text files (javascript, CSS, HTML, JSON) as announced in Steve Souders paper [21]. This modification has to be done on the server side but does not create any visible issue on the game. As well, this change does not affect the framerate of the game.

All these changes have good effects on the game performance and on the loading time but they can also make the code harder to change and modify. If these changes are applied to another application, the result may not be the same as the code will be different but they should impact the performance of the application in the same way.

5.1.3 Automatize the code improvement

Automatize the code improvement does not affect the performance or the loading time of the game but it will permit to do that kind of improvement automatically. As the code can change often and some process like sprite sheet generation or code minification can take some times, it is interesting to automatize these processes in order to create automatically a production version. In the game, sprite sheets and minified code need to be regenerated. As well, the development part of the code may need to be cleaned before putting the code into production. By using scripts some of these processes can be done quite easily by running one command line, this way you can create with Google closure compiler a script to minify all the javascript files [6]. As well, glue [10] can be used in order to create a sprite sheet generation script. The use of a continuous integration tool like Jenkins [14] can fully automatize this process by updating these files after each modification on the project repository. This way, it is possible to create an optimize version of the web application really fast.

All these changes allow reaching the 60fps which are the screen refreshing rate of most of the computer. Also, the 30fps objective is usually reached even with multiples themes. This framerate should make the game watchable for everybody[1]. As well, it allows having a pretty good quality game [5].

5.2 Method

5.2.1 Architecture improvement

In this thesis, the focus was to improve the architecture where some needs have been declared. This has a good impact as it increases the code quality on most of these parts but some
parts which should have been refactored was not refactored. Also, refactoring can create conflict as other developers may work on the same part of code so they need to be resolved. Design patterns as defined in "Design Patterns Elements of Reusable Object-Oriented Software"[9] have been used as much as possible. Their usage may not create the best architectures for each modification but help to create a good architecture. In fact, design patterns seem to aid to maintain software but their effectiveness is quite unsure [25]. The interest of design patterns is that most of the developers have heard about them and they are easy to find. For example, by searching "best book computer science" on a search engine, "Design Patterns Elements of Reusable Object-Oriented Software"[9] appears in most of the results. The interest of using design patterns is here not only to do a good architecture but also to make it understandable and usable by other developers.

5.2.2 Performance improvement

The focus of the performance optimization part was to improve the global performance of the game. To do that it is important to look at all the part of the game, for example if an application needs 40 seconds to load assets and 20 seconds to load javascript files, you can optimize the javascript as much as possible but you will always get a load time bigger than 40 seconds. The bug game can be divided into two parts, the code and the assets so to improve the performance of the game both parts was improved.

This approach allows to increase the framerate and decrease also the loading time. As well, the process used to improve the performance are quite easy to replicate as they are quite procedural. Moreover, by throttling the bandwidth for measuring the loading time, you can have precise results. For the framerate and the frame calculation time, for the same javascript engine and computer, it’s possible to notice a difference of about 5fps for the framerate and 5ms for the frame calculation time. These variations are due to the game itself if a lot of events happen the game may be slower, besides the processes running in background on the test machine can affect the performance. In order to be consistent, the tests have been done with a minimal number of processes running and successively. They have also been repeated in order to have average values. This way, it may be hard to get the same results if the study is replicated but the results should head to a similar conclusion.

5.3 The work in a wider context

These kinds of optimization can be really important in a world where bandwidth and the power of the computers increase more and more we sometimes forget that some users still have old computers or poor bandwidth. In march 2016, 10% of the computer are running under windows XP and the average bandwidth is only 5.6Mb/s. This way it is important to keep web applications as performant and as light as possible. As well, in a context where web applications tend to become more and more complex and bigger and bigger, it’s important to manage this complexity by using a good maintainable and extendable application but also to increase its performance.
6 Conclusion

There is a lot of improvements that can affect a web motion game. By refactoring the game architecture, you can make it easier to modify, to extend and to maintain. To do that in javascript, design patterns such as the observer, the composite, and the builder pattern can be used. As well, the game performance can be improved a lot by changing some parts of the javascript code. By applying some compression and optimization scripts or by compressing files, the framerate of the game can be improved, in the game case by 50%, and its loading time reduced, in our case by 72%. Moreover, a continuous integration system can be added in order to execute all these scripts automatically and create an optimize version of the application automatically after each change.

Another improvement on the performance optimization can be done on the web game. In particular, to display graphics. The use of shaders instead of sprites can transfer some work done by the processor to the graphic card which is fastest to render graphics. This way, working on shader can be a good way to improve the performance more the game performance.
Bibliography


[8] CSS specifications. URL: https://www.w3.org/Style/CSS/specs (visited on 02/07/2016).


