Expanding a Motion Controlled Game With Focus on Maintainability

Deniz Ayar
Andreas Hedbäck

Handledare/Tutor, Erik Berglund
Examinator, Anders Fröberg
Expanding a Motion Controlled Game With Focus on Maintainability

Deniz Ayar  
Linköpings University  
581 83 Linköping  
denay478@student.liu.se

Andreas Hedbäck  
Linköpings University  
581 83 Linköping  
andhe808@student.liu.se

ABSTRACT

Motion controlled games can be a good physical activity for children, but the game has to be fun and engaging. We have, with a starting point in an existing base game, developed an achievement module which follows certain code standards to make it easier to understand, and to make handovers of the code smoother. More work on the rest of the game has also been done to make it more engaging, while clean up of the existing code to follow the same standards.

Author Keywords

flow; game; motion control; achievements; code cleaning; smells; code standards; javascript; phaser; maintainability; extendability; ECMA;

INTRODUCTION

In 2006 Nintendo started a new era of motion-driven games. Nintendo’s console, Wii, sold over 100 million units \(^1\) and with this milestone it was one of the most successful consoles in modern time. Due to the success of the Wii, where body motion is used to control the game, the rest of the industry followed suite and Microsoft released their version, Kinect, bundled with the Xbox 360 in the end of 2010. But the interest for motion-driven games quickly started to fade and Microsoft stopped bundling Kinect with their console as early as 2014\(^2\), only 4 years after their initial release. Other, new gadgets started to take its place, like virtual and augmented reality.

Today there are lots of different ways to control games, not just a hand controller, however our belief is that the games need something else than just a different way to control them. To be able to keep engagement and flow in the game we need something that is interesting in the gameplay itself.

---

after only shorter sessions. Trials have been made where e.g. different enemies have been implemented, themes and styles have been changed and achievements have been tried to some smaller extent. This has lead to some mixed results, some of the achievements were too hard and made the players quit and some achievements did not give the feeling of accomplishment.

Our system needs to balance the challenge of the game with the abilities of the player. While implementing our system we will also clean up the rest of the code to the best of our abilities and common coding practices.

**Research Questions**

- How do we design an achievement system with focus on maintainability and expandability in an already existing game?

**Limitations**

The current code base uses older versions of both Phaser and ECMA, which are deprecated to some extent.

**BACKGROUND**

The work in this thesis has its starting point in an already existing project. The owners of the project wanted us to implement a system that could be integrated in the existing game. We were required to use the language and frameworks already chosen for the project, and we also needed to make sure that our implementation did not break the flow of the game.

**LITERATURE REVIEW**

**JavaScript**

Javascript is a powerful scripting language widely used on the web. It was designed by Brendan Eich, co-founder of Mozilla, and made its first appearance in 1995. Javascript is an interpreted language that is weakly typed and is run by the web browsers Javascript engine. The language is mainly used for client side applications and usually paired with HTML and CSS to make powerful and dynamic web pages, but the rise of libraries like Node.js has made it easy for developers to use Javascript in server side applications.³

In the beginning JavaScript had several independent implementations, but since 1997 it is standardised by ECMA International. Eight ECMA editions of JavaScript have been released as of 2018, some with minor editorial changes and some with significant syntax and structure reworks.⁴

While working in JavaScript, the data format JSON can be used to save data. The JSON format can convert JavaScript objects to a string for easy saving, either locally or on a remote server. The resulting .json files can then be read by a parse function in JavaScript to assemble the original objects when they are needed in the application.⁵

**Phaser**

Phaser is a framework for making 2D games in HTML5. It is developed and maintained by Photon Storm and game development is done with either JavaScript or TypeScript. Phaser can swap between a canvas and WebGL renderer to enhance browser support for desktop and mobile devices. Both canvas and WebGL are JavaScript APIs used to draw objects to the screen. While also containing support for 3D, WebGL is faster in its execution than canvas, although canvas has better support for older browser versions.

Since the start of Tree Bug Game development, Phaser has gone from version 2 to version 3 which is a completely new framework. Phaser version 3 contains almost no code from version 2. This would make a migration of the project to the newer, version 3, very difficult as everything would have to be re-implemented manually.⁶

**JavaScript Smells**

Smells are poorly considered design choices made to a programming language. Usage of these code structures can result in unintended behavior and is considered bad coding practice. Saboury, A et al.[⁷] found that code containing Smells were 65 percent more likely to fail, due to bugs or invalid inputs, than files that didn’t contain any of the bad practices. When evaluating these results Saboury, A et al. considered this list to be the points to avoid when working with JavaScript:

- Lengthy Lines: The number of characters per line.
- Chained Methods: The number chained methods in each chaining pattern.
- Long Parameter List: The number of parameters of each function in source code.
- Nested Callbacks: The number of nested functions present in the implementation of each function.
- Variable Re-assign: The uniqueness of variables in same scope.
- Assignment in Conditional Statements: The presence of assignment operator in conditional statements.

● Complex code: The cyclomatic complexity value (number of possible individual paths) of each function defined in the source code.
● Extra Bind: Whether a function is explicitly bound to a context while not using the context.
● This Assign: Whether this is assigned to another variable in a function.
● Long Methods: The number of statements in each function.
● Complex Switch Case: The number of case statements in each switch-case block in the source code.
● Depth: The maximum number of nested blocks in each function.

Good code practises
To evaluate what is considered good code we use the directions of McConnell, S. [8]. We chose this because of the ease of using the checklists McConnell, S. provides for self evaluation of good code practises regarding different topics. There were too many topics and checklist to cover in this thesis so we picked a few of them that felt most relevant for our research question:

Naming variables
Choosing variable names when working in a smaller solo project might be easy and it might not be difficult for the developer to understand their own variable name. But when collaborating in a larger project with many other developers this becomes more crucial. A standard naming convention makes it easier to see what type of variable you are working with. For example a global constant could be differentiated by being all uppercase, like MY_CONSTANT. A local variable, in that case, should be lowercase with an uppercase separating multiple words, like myVariable.

According to the checklists a variable name should:

● Fully and accurately describe what the variable represents.
● Use more meaningful names like ‘count’ or ‘index’ rather than ‘num’. Also avoid names like i, j and k when using loops.
● Be long enough that you don’t have to decipher it.
● Boolean variables should be named so their meaning is clear when true.
● Make clear if a variable is global, local or class and also distinguish among type names, named constants, enumerated types, and variables.
● Avoid abbreviations unless necessary and in that case consistent abbreviations should be used.
● Not be hard to read, not be easy to confuse with other variables and not use numerals or hard to read characters.

Good commenting technique
In more formal projects, documentation of the code would usually be put in separate files. Our project already has documentation written as comments in the source code and therefore we decided to do the same. McConnell, S. lists 30 points to consider when writing comments, some of the main ones:

● Can someone else pick the code up and start to understand it immediately? If not, it probably needs commenting to make it more clear.
● Do comments explain the code’s intent och summarize what it does, rather than just repeating it?
● Are the comments clear, up to date and correctly written? Redundant and old comments should be removed.
● Avoid end of line comments, put them on their own line. This helps prepare the reader for the code to follow.
● Code that are used to catch errors or work around some problem should be commented.
● Always comment limitations on input data, ranges of values in numeric data, global variables and control statements.

Code linting with ESLint
Code linting is a type of analyses that is used to find patterns in the code that does not meet a set of user specified rules. The rules are usually made to achieve a homogeneous code style, but also to meet common guidelines for the language used.

ESLint is a open source linter for JavaScript created by Nicholas C. Zakas in 2013. A linter is used to set a group of global code style rules for a project and ESLInt can be used as a plugin in any popular IDE (integrated development environment), like Visual Studio Code which is seen in Fig. 2. ESLInt uses a JSON file for configuration and is usually set up with a default ruleset were you can change some basic settings like what ECMA version your project is using. The user is able to define a custom ruleset with options like the “semi”-rule that either requires or disallows the use of semicolons in the source code.

The plugin notifies the developer when project files does not follow the rules and makes it easy for a multi-developer project to maintain a unified coding style. Many of the options also helps the developers avoid the coding Smells
associated with JavaScript, and most of them are defined in the default ruleset.

ESLint comes with a good amount of default rules but it is possible to customize the JSON configuration file to your own needs, even changing the parser used to evaluate the files. The process of customizing ESLint is a quite large task and won’t be covered in this paper, although all its documentation is available online.⁷

Flow

J Nakamura, M Csikszentmihalyi describes flow as a state of mind where people are fully consumed by the task they are currently doing. This state is comparable to the feeling of enjoying life where: “A good life is one that is characterized by complete absorption in what one does”[1].

Figure 2 The flow model adapted from J Nakamura, M Csikszentmihalyi. The bars show the flow zone of different groups of performers. The green is an average performer of a task, where pink is more experienced and blue more novice.

J Nakamura, M Csikszentmihalyi’s work show a correlation between the challenge of a task and the performer’s abilities. If the performer’s abilities match the challenge of the task, the enjoyment and self-esteem are peaking. This is considered a person's flow zone. A person’s flow zone can fluctuate depending on how familiar the task is and how much the performer has with it, this is indicated by the pink, green and blue bars in Fig. 2. [1]

According to J. Chen [2] there are several points to create immersion and engagement. Some more important than others, e.g. the activity needs to be challenging and require skill, as seen in Fig. 2. There also needs to be clear goals and the feedback to be immediate. This leads to states in the player that indicates flow, e.g. a sense of slowing or speeding up of time, a feeling of control and concentration on the activity being performed.

Engagement in games

Many studies have been done on engagement in video games and many different measuring methods have been used for evaluation. One of the popular ones is Csikszentmihalyi’s Flow model for measuring subjective engagement, although in more recent research it has been common to use immersion for evaluation instead. This is due to Flow often being referred to as the optimal state of mind while immersion is used to measure engagement on a scale. [6]

Some of the motives for playing video games are the fun and challenge aspects of the task. These, as well as stress relief, were the top answers when adolescents were asked to grade their reasons for playing. The reason for playing can also be seen as what makes a game engaging. These experiences correspond to the properties of the game and to make a game engaging, it has to satisfy the players need for achievement which ties into the mentioned motives. [4]

METHODS

Pre Study

Code Quality

To improve the quality of the code base, and to make our own addition to it, we first had to get acquainted with the project. This was a necessary first step for us to be able to implement anything at all since we needed to use some modules that already had been developed for the project, mainly the UI and database module. To not waste time on this task we decided to combine getting familiar with the structure and also apply some general linting rules to the code.

We used ESLint as a plugin in the IDE to easily refactor all the projects .js files, although some of the linting rules required us to fix them manually. This was done to achieve greater readability and to make sure that all developers used the same code standard to make the project more maintainable for future developers. We chose to use a semicolon standard for Javascript which among other basic rules demand the need for statements to end with a semicolon to make it more readable and easier to maintain.

We proceeded with implementing smaller parts in the game to give us insight on how to create new content, such as new achievements, bug behaviour and UI. What we quickly realised was that since the project had been ongoing for several years and developed by many people, the code standard was very mixed. Some developers had made proper comments and good code according to the guidelines of S. Mc Connell [8], while some parts of the code looked like it had been abandoned in the middle of implementation, and contained legacy code that had been commented out and were not in use.

Implementation

Ongoing Project

The work on our achievement module was a part of a bigger patch to the game. The purpose of this patch was to enhance the flow and make the game more engaging. Since the project was ongoing and worked on by many developers, we had weekly meetings with information on what was being worked on by the others and what the owners wanted us to focus on next. Among other things, we came up with a new anti-cheat system which also changed the way the bugs spawned. This was all done while we were implementing the achievement system.

A lot of collaboration had to be done between us and the other developers, not only during the weekly meetings, but on a daily basis, sometimes as soon as something new was implemented. When working in larger groups on the same project, considerations has to be taken between the different developers, we were lucky enough to sit in the same room as the others which simplified this task. The game received several large changes to core systems and mechanics while we were working on our achievement system. One such change proved to be problematic when a database was added, this is mentioned in the discussion.

RESULTS

Prestudy

The initial code cleanup and linting resulted in a much smoother development phase for us when implementing our achievement system. The large legacy code base was hard to get a grip of at first but after this initial process it was easier for us to know where and how to hook in our module to avoid breaking anything and assure it was working as intended. Our IDE of choice could apply most of the linting rules to the whole project automatically, which greatly reduced the time needed for this process. We still had some rules that did not get applied automatically so we needed to do some manual tweaking to the code where these were present.

Implementation

Anti cheat system

The mechanic used to prevent cheating before was a water bucket that would randomly spawn in and walk on the paths, if the player touched it it would fall over and the game would be over and recorded as a loss. This was introduced to prevent a player from standing close to the camera and just waving their hand over the lens resulting in the capturing of all the bugs. The cheating made the game effortless and defeated the purpose of the game which should include moving the body around.

It is however hard to have full control of every single part of your body, or having full awareness of the position of the whole body when playing. Sometimes the player’s e.g. hips moved too much and they inadvertently caught something they did not intend too, and this was happening far too often with the water bucket which became an annoyance and made the players irritated. The flow became interrupted by the induced irritation by something that the player felt that was unfair or that they had control over.

We suggested a new way to prevent cheating and to an extent keep the flow. Our suggestion was built on the premise that we should not introduce anything to the screen that would make the player immediately lose. This lead to a mechanic that we called bug ripening. The bug life cycle was divided in three stages: unripe, half-ripe and ripe, and was implemented by other developers.

Figure 3. Different stages of ripeness, unripe, half-ripe, ripe

These were shown with different tints on the bug, Fig. 3. The ripening process starts when the bug spawns at the beginning of one of the paths, which can be seen in Fig. 1, it then has a ripening period where the points that the bug gives when caught goes from negative to max over a random period of time.

This solved the problem of cheating players and made skill based movement more accentuated. The gameplay changed from catching all the bugs, all the time, to being more precision based and catch the bugs when they are ripe and give the maximum number of points. But it also had a slight negative impact on the movement, the player tended to stay more still in the middle, since all the paths leads to the tree
in the middle, and the player wants to maximize their points and catch all the bugs at their ripe stage.

This was remedied by the changes to the bug spawning system. Instead of spawning lots of slow bugs at the same time, the mechanic was changed to only spawn one bug at a time, but they were also faster than previously. The mechanic was set to spawn two bugs, but only let one bug move, the next bug will not start moving until the first bug has been caught. This made the player aware of where the next bug would start moving so they could plan their moves accordingly.

The new spawning system created a positive need to move the whole body quickly to different parts of the screen. The movement became more calculated and involving, which lead to flow, the players has to register a bug, see what stage it is in, estimate its ripening process, catch it in the right moment and then prepare for the next bug.

During implementation of our achievement system we striving for modularity and to tick as many boxes from the chosen code quality checklists designed by McConnell, S. [8] as possible. We also designed our classes and functions according to the guidelines of Saboury, A et al.[7]. Since this project will be handed over with no further instructions from us, except the comments etc., we took care when naming and declaring our system.

When developing we often tried different methods and calls to achieve what we wanted to do, the resulting names and variables, after such a session, would not follow any of the guidelines. Because of this, we often rechecked the code and the guidelines and see how we could improve them. This did cause some double work caused by the testing phase that lead to implementation with badly named methods and variables, e.g. `updateAchievements(active, completed)` which earlier was called `setUpAchievements()`. Both methods do similar changes, but the latter does changes implicitly to objects that are unknown, the former method needs explicitly stated objects to change. The naming is also clearer, while setup can be seen as something that is only done at the beginning, update is clearer and implies that it can be called on at any time.

From our understanding and with the help of the checklists from McConnell, S. [8], we believe that the code standard in our module is good enough. That means that we believe that the code could be easily interpreted, extended and maintained without our help. However, we were not able to completely avoid the mentioned JavaScript code smells due to limitations with the legacy code. The code base used the JavaScript standard ECMA 5 which lacks support for the points “This assign” and “Variable re-assign”. This was one of our biggest problems in regards to quality and has its own section in the discussion.

We designed the achievement system to support new achievements easily by only making changes in the JSON file. The system supports basic functionality of the different entities in the game and will work with new ones if the game expands. If the game needs more complex achievements in the future, it would require some additional structure on how to process the intended uses of the achievements, but it should only have to be done in one place due to the modular behaviour of the system.

The achievement system

During implementation we decided to divide the system into five different classes for better abstraction.

- **Achievement**: The class describing a individual achievement with properties and methods to extract information from the class, like parsing a string id for example.
- **AchievementManager**: A class containing the logic part of achievement generation which also makes calls to the player to handle the database.
- **AchievementList**: A container structure for a list of achievements. It is a regular array with some added functionality, e.g: returning a random achievement from the list, or searching through it for a specific one.
- **AchievementData**: A JSON file holding the rules for different achievements. This list is parsed to Achievement objects by the AchievementManager. The JSON file uses a structure where a type of achievement is specified like a template and the AchievementManager can build multiple achievements from every entry by using the incrementation and level properties.

![Figure 4. A simplified model of the data flow in the achievement system.](image)

When the game starts and the AchievementManager is initialized, it uses the JSON-data file to fill its AchievementList with new Achievement objects.
These achievements start at level zero and are never changed, instead acting as a base to create new achievements from within the manager.

As seen in figure 4 the manager gets notified by certain game events by the game, and has data flowing both ways with the Player class to maintain the players lists of achievements.

The player class holds two instances of the AchievementList class. One for keeping track of the currently active achievements and one containing all the achievements the player has completed. When starting a new round of the game the player send these lists to the managers updateAchievements() function. The AchievementManager checks if the player has the amount of active achievements needed, if not, it adds a random achievement given a set of basic rules which prevents the player from getting the same achievement twice. If the player has completed or already has the new achievement the manager will increment the level of the new achievement and repeat the process. When the randomized achievement is not in any of the players lists, it will get added to the players active achievements.

Completing an achievement is done by having the game notifying the manager about different events in the game, e.g. when a bug is squashed or a round is finished. These events have corresponding functions in the AchievementManager which decreases the amount property in the achievement. If the amount property reaches zero, the achievement is completed and moved from the active to the completed achievementlist in the player object.

When a round in the game is completed the player tells the class handling the database to save both its active and completed achievement lists. The same lists are retrieved by the player when a new game starts.

Example of an Achievement base in the JSON-file:

```json
{
   "description" : "Squash AMOUNT green bugs!",
   "sub_category" : "1",
   "amount": 5,
   "increase": 5,
   "type" : "GREEN",
   "theme" : "ALL"
}
```

In the JSON-file achievements are grouped by category, e.g. win rounds or squash bugs. The groups are numbered and together with the achievements sub category and level they are assigned a unique id for that specific achievement and its level.

The JSON-file is only used as a template for the different kinds of achievement that can be used by the game, the Tree Bug Game itself can create an infinite amount of achievements of different levels from the template.

The function in the AchievementManager were broken up into five different parts:

1. Parsing achievement data to achievements and putting them in the achievement list container.
2. Updating achievements, making sure that the player always has the correct amount of active achievements.
3. Handling events passed by the game to update the amount left to complete an achievement.
4. Completing an achievement, updating the players lists of active and completed achievements.
5. Updating the UI, Fig. 5, to show the new active achievements when one is completed.

These parts are all handled by separate functions to be able to track the “lifecycle” of an achievement and making it easier when expanding and maintaining the system.

DISCUSSION

Result
We spent the first weeks with familiarising ourselves with the code, applying linting rules and getting the code up to standard. With quite a large code base, around 11,000 lines of code, it was a time consuming task, but the time spent was valuable. Later on, in the implementation process, we were familiar enough with the code to know where to look or make changes without having to go through lines and lines of code to find where to make them. This saved us time during the development period.

While the initial development started out good and we were able to follow our coding guidelines, we hit some problems. The code base was written while ECMA 5 was the current standard, and also the version that Phaser used. We realised that the support for assigning scope bound variables and the passing of scope was implemented in ECMA 6. Because of this we had to make a decision, break the standard used in the project, rewrite as much of the code base that was possible or to disregard the code smell guidelines. We went with the latter.

Flow
Our initial plan was to do this report on the ability to enhance flow in the game by adding a achievement system. That idea had to be abandoned due to difficulties in evaluation that would be satisfactory. We kept the original plan of building an achievement system since we were too invested in the development of it, but we decided to shift focus to the quality of the code and modularity of our system. We feel that this had a negative impact on the quality of our resulting work due to the time constraints from switching focus.

While the Flow model was rejected as a research question it impacted much of the time spent on development of the achievement system. Maintaining Flow in the game was a requirement from our employer, so while trying to build a system that is expandable and easy to maintain, we also needed to keep in mind how the achievement affects the game in general. This became somewhat obstructing for us, since other parts of the game that were not finished yet were supposed to tie in to the achievement system. Mainly the Flow increasing portion of the achievement system should be supported by the user interface in general, but there were also plans on developing a system to make the player have a profile and “collect” achievements, gaining levels and more.

In the end we decided that our implementation could easily support any of this without major rework in the future. This was done by implementing the state-like functions for the achievements were other systems can act on changes in the lifecycle of an achievement.

New Systems
Although the implementation of the new ripening system and spawning system was not in our project, a large part of our time was spent on testing and helping in the development. Since it tied in to the flow of the game, we felt that it contributed to both the project and our thesis. It also made the game more interesting to play.

Methods
The linting of the code was a good way for us to get to know the project, as there was no documentation and few comments. One benefit of deciding on a unified coding style that we did not foresee was the absence of discussions on how code should be written. When working with multiple people we feel that often there are conflicts regarding where to put brackets, spaces or tabs, but deciding on using linting rules in the IDE eliminated that and focus could be on the important parts of the project.

Many of the Javascript Smells felt like common sense and we did not have to go to great lengths to avoid them. These include the avoidance of using lengthy lines, chaining methods, and having too long parameter lists. It was however helpful to keep them in mind while writing code.

We felt frustrated by not being able to avoid a few of the Smells due to the JavaScript standard (ECMA5) used in the project that did not support them. We had many problems with scope handling because we could not follow the “this-assign” rule in ECMA5 and we got lost sometimes wondering what scope we were working with and why some variables did not exist. These problems were also magnified due to our combined experience with JavaScript not being that great.

A large part of the development process was spent trying to figure out how to get calls to the asynchronous database to arrive in order. The database was done by another developer and was implemented quite late in the project. This among other minor implementations from other developers have been delaying the achievement system and we should have been more prepared for this to happen when working with others.

We had reduced development time due to weekly meetings and random playtesting of the other developers additions to the game. This impacted our own development in a positive
way since we became more up to date with the other parts of the game, some of which in we had to use in our own system. Others also playtested iterations of the achievement system making it easier for us to get feedback and change things more frequently. These random playtests were also a great way of taking some time off coding and put ourselves in the players perspective, making it easier to evaluate certain parts of the game from the other side.

**Ethical Considerations**
Physical well being and obesity is a growing problem in the world, WHO is reporting that obesity has nearly tripled since 1975\(^8\). While we do not believe that the game will solve world obesity, we believe that games that makes the player move will in some small part help with a much larger problem.

**CONCLUSION**
With the help of the checklists and practices discussed in this thesis, we believe that we have made a module that can be further developed, maintained or extended by developers that were not present when we implemented the system. The module will also not impact the flow in a negative way, on the contrary our non-intrusive achievements, Fig. 5, should make the game more enjoyable and create a better flow.

**Future Work**
We believe that our original research question, more focused on flow, gameplay and engagement, would have been a more interesting subject for our thesis. If we would have had the time, it would be very interesting to see how different aspects and changes to the system would affect the players perception. This would have been evaluated with different questionnaires and interviews with the players which would also include psychological aspects of playing games.

---

\(^8\) [Accessed: 01-June-2018]
REFERENCES


