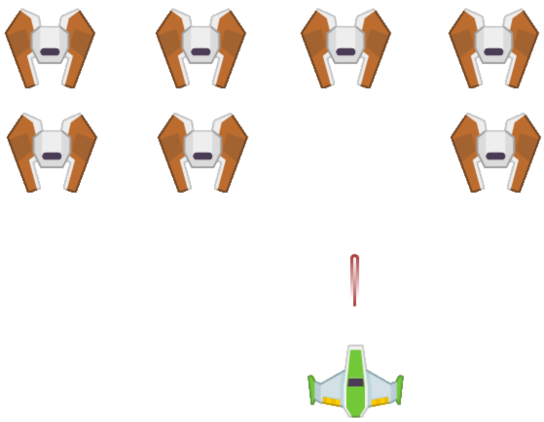
# Has Functional Got Game?

Evaluating the Benefits of Functional Programming Languages in Game Development



Master thesis - 10th Semester Computer Science

PT102F19

Aalborg University
Department of Computer Science, SICT

Written in LaTeX from a template made by Jesper Kjær Nielsen.

**AALBORG UNIVERSITY**
STUDENT REPORT

**Computer Science & Software Engineering**
Aalborg University
http://www.cs.aau.dk/

**Title:**
Evaluating the Benefits of Functional Programming Languages in Game Development

**Theme:**
Scientific Theme: Language evaluation, program benchmarking, game development, functional programming, usability evaluation

**Project Period:**
Spring Semester 2019

**Project Group:**
PT102F19

**Participant(s):**
Malte Emil Rosenbjerg Andersen
Thor Steen Jensen
Daniel Dirk Albert van Bolhuis

**Supervisor(s):**
Bent Thomsen & Lone Leth Thomsen

**Copies:** Digital distribution only

**Page Numbers:** 73

**Date of Completion:**
June 6, 2019

**Abstract:**

For many years the functional paradigm has been praised for improving correctness of programs and decreasing development time. Even major figures in the game development community has argued for the use of the functional paradigm in game development. Still, most game development done today is using object-oriented imperative programming languages such as C++ and C#. In this report, the performance cost of using a functional, object-oriented programming language, F#, instead of an imperative, object-oriented programming language, C#, for game development is evaluated. The evaluation consists of a benchmark suite focusing on numerical- and vector performance, and a usability study focusing on the transition from F# to C#, and how easy it is for developers to transfer their existing knowledge of the Unity game engine. Through this evaluation, we show that the performance penalty of using a functional programming language is negligible, and that the usability of the Unity game engine with F# is promising.

# Preface

This report builds on a prior report [1]. In section section 1.1 we give a summary of the points from that report. We recommend reading the full report, but the summary should make due.

We would like to thank Stefan Nordborg Eriksen for his valuable help in finding participants, both medialogy students and *Coding Pirates* children. We would also like to thank all of the participants that took part in our usability evaluation.

Thanks to Tobias Morell and Thomas McCollin for collaboration on creating the F# introductory material, and for their help in testing the Unity extension and pilot-testing our tasks.

Finally we would like to thank our supervisor, Bent Thomsen, for invaluable insights throughout this project.

# Summary

In this section we provide a brief summary of this report.

In Chapter 1 the problem area is introduced and arguments from major figures in the video game development community are presented. After these arguments, the hypothesis that this report attempts to answer is presented. As this report builds on previous work, a summary of that report is given in Section 1.1, including a sample of the results and the cognitive dimensions that were applied.

In Chapter 2, the current work of two of the co-authors of the previous report is presented, as we have worked together with them for some aspects of this report as well, though they are not co-authors on this report. Afterwards, some other related projects with similar focus are presented.

Chapter 3 has sections describing the functional languages and game engines that were candidates for use in this report. The process of selection is also described here.

In Chapter 4, a description of the current support for F# in Unity and of our extension to Unity to provide better support for F#, is given.

The benchmarks performed for this report are described in Chapter 5 along with a description of the test setup. First a description of the microbenchmark suite is given along with the results and an analysis for a number of platforms. Then the macrobenchmark, which is an implementation of an autonomous Artificial Intelligence (AI) game. Finally the results of the macrobenchmark along with an analysis of those results are presented.

Chapter 6 introduces the usability study by giving a description of the participants, tasks and protocol. The results are then shown and significant results are explained in depth. Finally, the threats to the validity of the usability study is discussed.

In Chapter 7, the qualitative measures of the usability study are presented and applied. Several of the dimensions from the Cognitive Dimensions framework are applied, and the results are discussed using code snippets from the solutions submitted by the participants.

Chapter 8 contains a discussion of both the validity and other considerations related to this report. Interesting results, such as the difference between the results from the two different groups of usability study participants are discussed.

Finally, in Chapter 9, we conclude on the results from the performance benchmarks presented throughout this report, and on the results from the usability study.

Chapter 10 some relevant future work are presented, as there were many interesting

directions this project could have taken. Some of these are further refinements to the F# extension to Unity, others are further investigations into the results from working with Unity. The experiment of integrating `dotnet` into Unity is also suggested as future work, as our results indicate that this could give a massive performance improvement. Finally, a study of whether the (slow) adoption rate of the functional paradigm could be a social issue rather than a technical is suggested, since our results show that both the performance and the usability, with Unity at least, is promising.

# Contents

# Chapter 1

# Introduction

There has long been many arguments for adopting the use of functional programming languages, such as lower development time and improved correctness [2]–[4]. Popular developers from the video game development community, Tim Sweeney and John Carmack, has argued that the adoption of elements from the functional paradigm could help develop better games by improving developer performance [5], [6].

The main argument against the use of functional programming languages has mostly been about performance, and more specifically, how difficult it is to achieve performance that is comparable to something implemented using an imperative language. In [7], the author argues that functional programming "sucks" and shows this through aligning a series of test done in Node.js, which uses the V8 JavaScript runtime, showing that functional constructs have not gotten faster, but imperative constructs have become slower, such that their speed have almost aligned. In [8] a question was asked about functional programming on StackOverflow and the highest rated answer argues that functional programming is slower than its imperative counter-part because of a historical commitment to allocating objects on the heap and focusing on fast garbage collectors, rather than figuring out which objects could be stack allocated and which can be heap allocated. Furthermore he argues that functional languages are slower than their C counter-part, because implementing large-scale multicore benchmarks, in which he claims that functional languages will dominate, is not possible in C.

Epic Games' founder and CEO, Tim Sweeney, expresses that he would willingly trade a 10% decrease in performance for a 10% increase in productivity [6], strengthening the argument for using functional programming in game development further.

We showed in [1] that it is possible to get performance from a C# program, running in the `dotnet` runtime, that is comparable to the performance of C++ programs, in the context of gameplay programming. Adopting C# could increase developer performance by removing the concern of memory management.

In this report, we will continue the investigation by looking into whether a move to the functional paradigm will have consequences for the performance of the developed games and applications. The investigation builds upon the results from [1], where existing video game-engines with a functional approach are evaluated; Nu [9] and Helm [10], and a functional paradigm gameplay language developed for use in the Unity, Arcadia[11].

This evaluation showed that these solutions still require a lot more maturing before being an actual alternative to major game engines, such as Unity and Unreal Engine. Arcadia was interesting because it extends Unity to support the functional programming language of the same name, which was developed specifically for game development with Unity [11].

This report attempts to answer the hypothesis:

> *Can functional programming languages, and the use of the functional programming paradigm, be used with existing modern game engines, without sacrificing performance and without increasing the cognitive load for the developers too much?*

It was formulated in such a way performance for long has been the counter-argument to using the functional paradigm. Also the many game developers should not have to start over with getting to know the game engine Application Programming Interface (API)

## 1.1 Summary of *An Analysis of Gameplay Programming Languages in Free-to-use Game Engines*

In [1], our goal was to investigate the field of game development. In our research we came across some interesting findings; one of the report's points was to see how prominent the functional programming paradigm was, since some famous game developers had given the paradigm praise, saying that it should be used in game development. Another discovery was a recent trend to abandon the old gameplay languages in different engines in favour of C#. Languages like Lua, JavaScript and proprietary solutions got deprecated in many game engines.

In addition to these discoveries, different game engines and their available languages were compared through a number of microbenchmarks. To even out the advantages of using Just-in-Time (JIT)-compilation a bigger macrobenchmark was performed, where the goal was to test the engines themselves, especially the parts utilising the gameplay API. The macrobenchmark was a simple autonomous game called Wumpus World. One of the major discoveries that came from this, was that C# running on the `dotnet`-runtime could compete with C++ in the microbenchmark suite.

As part of the research we wanted to see how far functional programming had come in game development, since certain game developers truly believed that it could help. We found 3 examples of game engines that used functional programming. We found engines that fully supported functional programming: *Nu* and *Helm*. They had been developed from the ground up with a functional language, *Nu* uses F# and *Helm* is a Haskell "functionally reactive game engine". It is required for these engines that the rest of the game also is developed in the given language. The other solution we found, was a group that had developed an extension to Unity called *Arcadia*. *Arcadia* made it possible to use Clojure as a gameplay language. That was done by utilising a .NET implementation of the Clojure runtime, so all of Unity classes are available.

In our investigation working with these engines, we found that the biggest issue was the lack of documentation and other learning material. This meant that the investigation was less about exploring the different features and caveats of each engine, but instead a fruitless search for documentation.

When researching what gameplay languages was used in different engines, we discovered that engines such as Unity and CryEngine had replaced their gameplay languages, JavaScript and Lua respectively, with C# and Godot was adding support for C#. We also saw that Unreal Engine had deprecated their old scripting language UnrealScript [12], and Kismet [13], their first integrated Visual Programming Language (VPL), which was also JIT-compiled.

We decided to test the different ways you could program the gameplay in the different engines. We wanted to test Unreal Engine with C++, Unity with C# and *Arcadia*, CryEngine with C# and C++ and Godot with C# and GDScript; a language with a lot of similarities to Python. These languages were put through a suite of microbenchmarks, inspired by the work of former research. We focused on four areas, divided into 12 benchmarks, revolving around different mathematical problems that had to be solved. The benchmark suite focused on Vector Math, Sestoft's Multiply, Primes100 and Array Allocation.

To our surprise we discovered that `dotnet` C# outperformed C++ with a significant margin, in execution time Figure 1.

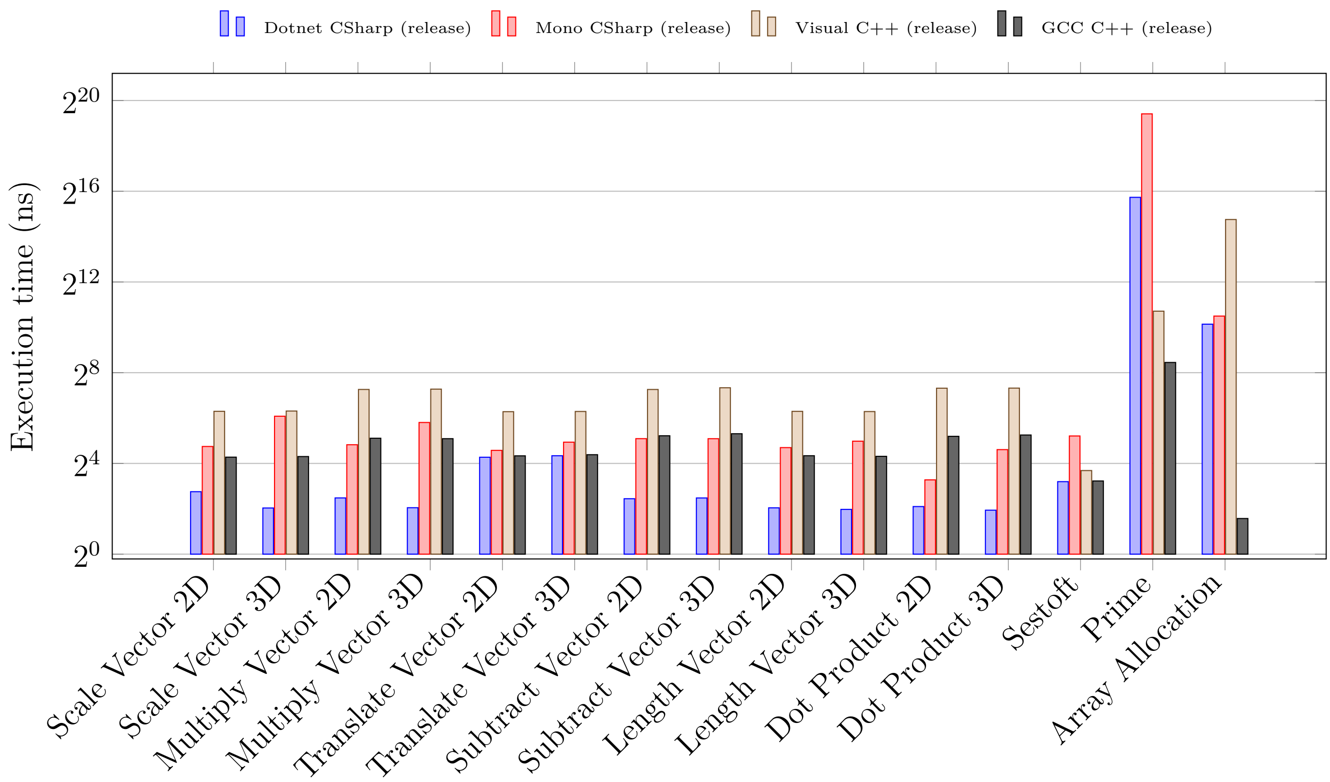



**Figure 1:** Graph over execution times in C++ and C#.

We also saw in the benchmarks `Primes` and `Array Allocation`, how effective GNU Compiler Collection (GCC) and to our surprise Unreal Engine C++ was at array allocation and primes, even when trying to force the compiler into not optimising away all the work.

After the microbenchmarks, a bigger test was needed to test the actual engines. For this we made a macrobenchmark, using an already existing AI game called Wumpus world.

The goal of the game is for the AI agent to find the gold and deliver it back to the start. The agent is blind and have to sense its way around the map. On their map there is obstacles, there is a "Wumpus" that wants to eat the agent and pits the agent can fall into. The Wumpus gives out a stench and the pits have a breeze that the agent can sense when standing one tile away. The must navigate trough the map, relying on the ability to percept the stench and breezes and logical reasoning about where the obstacles are.

The two engines tested was Unreal Engine and Unity, because both did the best with their respective languages. So a C++ and a C# version of Wumpus was made in their respective engines and tested.

For the test we choose to use the metric of how much time does one world iteration take, as our baseline for comparison. Each map would be run 10 times and where one iteration would be from the agent starts moving, till he has delivered the gold back to his start position.

We can see the results of the test, shown on Figure 2. The spikes in the performance of Unreal Engine (iteration 14, 28 etc.) correspond to clearing the agent's state and beginning the next iteration. In the case of Unity, the execution time for the agent's first tour through the map is much higher than that of Unreal Engine. The execution time in Unity stabilises after the thirteenth iteration, lying well over the execution time in Unreal Engine. Interestingly, the spikes that correspond to restarting the map (every thirteenth iteration) are higher in Unreal Engine than they are in Unity. This comes at the cost of uneven execution time in Unity, which could be caused by garbage collection. Generally Unreal Engine execution times are more stable and predictable than those of Unity.

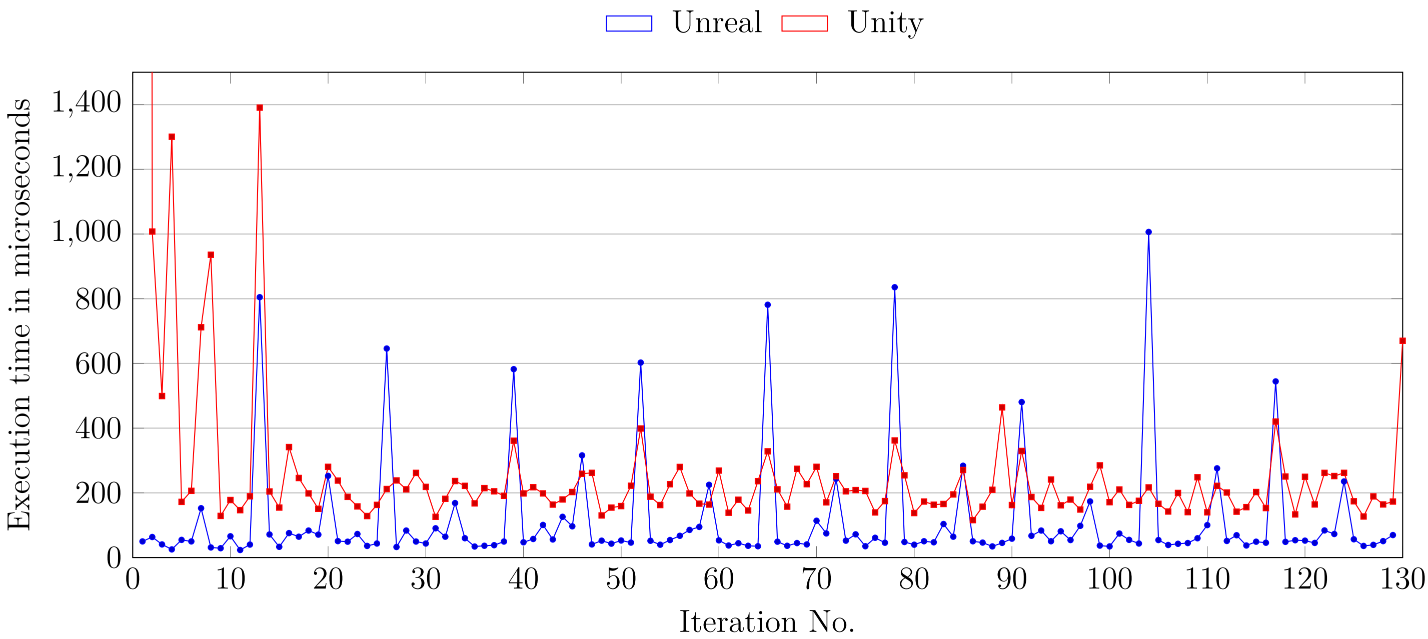



**Figure 2:** Wall clock-time for each invocation of the `World.Iterate` method

We were interested in researching the usability of languages used in engines. For this we used cognitive dimensions [14] to get a better understanding of the languages. Cognitive dimensions is a framework that utilises dimensions that are used to evaluate a language [14]. We then use these evaluations of the different languages to put them up against each other in a comparison. We chose to omit some of the dimensions because they where not fitting for game development, so the specific dimensions we used were:

- Diffuseness/Terseness
- Hidden dependencies
- Premature Commitment

- Progressive Evaluation
- Role-expressiveness
- Viscosity

These dimensions were chosen because they favour fast development, problem mitigation [15] and seemed the most relevant. This is especially important in game development due to the complex nature [16].

We concluded that the languages are both members of the C-family and therefore share many features and keywords. This means that most differences are in the details, one of which is memory management. This is not covered by any of the dimensions presented here, but could fall under the hard mental operations dimension.

In conclusion, C# assists the programmer and steers them around pitfalls. C++, on the other hand, allows the programmer to do any action, even when that action is not advisable. Examples of this are the macros and templates of the C++ language, which allow the programmer to increase code reuse at the risk of creating code that is hard to reason about. Furthermore, traits such as progressive evaluation, which only C# supports, allows programmers to find errors early and avoid costly mistakes.

# Chapter 2

# Related work

## 2.1 A Game of Paradigms

A piece of closely related work is that of Morell and McCollin [17]. In their report they explore Functional Reactive Programming (FRP) using F# and Unity. For their research they perform a user study, which evaluates professional Unity developers' ability to use F# with FRP with Unity. The user test they evaulated, they wanted to see how well experienced expert developers would take to F#. To do this, they made a test, where they had 8 test cases they would put their users through. They had to do these cases in both C# and F#, and they had 20 minutes for the C# part and 40 minutes for the F# part. The test cases was as follow:

- FPS Controller
- 3rd Person Controller
- Talent Tree Walker
- Armour Graph
- Dialogue Tree
- Concurrency
- Unit Management (RTS)
- Magnetic objects

The report also covers the benchmarking of the concurrency in C#, F# and Unity, because the lenient evaluation strategy they made use off was lacking. So they decided to research the issue and they discovered that there was a overhead using F# in Unity, but they would not say anything with certainty. They also conclude that users even though struggled with various aspects of F#, they could still could produce F# code that had certain qualities that where lacking in C#. But the users where still hesitant to switch to F# but they could recognise the benefits of the language.

We have also worked together with them on developing the F# introduction page, as they also performed tests with users and needed material for introducing the participants to the use of F# with Unity [17], [18].

## 2.2 Language Support in Unity

There have been several projects which aimed to add another language as the gameplay programming language to Unity, such as Arcadia and Casanova [11], [19].

These two projects have two different approaches to adding a new gameplay language.

Casanova is both declarative and procedural language, for writing gameplay code for Unity games [20]. It is transpiled to C# for use with Unity, but one of the goals of the language is to have F# as a compile target [19], [21]. Casanova is a much more high level language than most of the other gameplay languages used in free-to-play game engines. The language describes entities and the rules these entities follow.

Arcadia adds Clojure support to Unity via Clojure CLR: "a native implementation of Clojure on the Common Language Runtime"[1]. Clojure is integrated as an interpreted language on top of Unity. This means the code can be modified while a game is running in the Unity Editor. It is mostly functional (and side-effect free), but has some specific functions that affect the state of the game.

One of the nice features of Clojure is Software Transactional Memory (STM), which is a way of handling concurrency. It is inspired from database transactions and a variant of transactional memory. While there is no official page that states Clojure CLR has support for STM there is some evidence, such as the STM tests on their github page[2]. Arcadia and STM were explored in a previous report, and the find was that STM was not the go-to method of managing memory in Unity [1].

## 2.3 Game development tools

In the academic, there has been made a few game engines. One of those are JOT, JOT is a specialised modular multipurpose massively multiplayer online game engine [22]. JOT had a interesting and academic approach of making a paper that specified a design and a architecture for the game engine. JOT was implemented in Java, due to Java's multiplatform abilities and the available third

[1] https://clojure.org/about/clojureclr

[2] https://github.com/clojure/clojure-clr/blob/master/Clojure/Clojure.Source/clojure/samples/stm/

party applications and libraries. The paper helps get a clearer picture of how a game engine can be designed and how a architecture can look.

Another paper is "Leveling Up: Could Functional Programming Be a Game Changer?", where the authors describe how FRP can be used for game development [4]. They show how games can be developed with the tool Yampa Arcade a "Haskell-embedded incarnataion" as the developers describe it them self [23]. They go through and show examples of games made in Yampa and then they go into what FRP is and the different principles modelling with FRP.

# Chapter 3

# Research

In this chapter we will research the technologies we want to use, making arguments and reasoning to why we choose the technologies, we will be going over what languages we want to use and what game engine we want to work with.

## 3.1 Functional Languages

In [1] we investigated three solution for functional programming in game engines, two of them being full engines that could be used with F# and Haskell, Helm and Nu. We also tested Arcadia which allowed for functional programming with Clojure in Unity. All three solutions had their own issues, but the biggest issue that was consistent was the lack of documentation and examples to help actually produce games and in the engines we could use, there were also performance issues. Arcadia was substantially slower than any of the other programming languages, always being slower than its `mono` C# counter part [1].

Haskell, F# and Clojure are some of the more popular functional programming languages today, with it we also have languages like Prolog and Scala that also are popular functional programming options [24]. What we find attractive for our test, is a readable syntax, this of course is a subjective topic, but we would want a syntax that is not too intimidating for less experienced or no experience programmer. While still being able to resemble the more popular gameplay programming languages that already exists.

**Haskell** is a statically typed purely functional programming language with type inference and lazy evaluation. Type classes, which enable type-safe operator overloading. Its main implementation is the Glasgow Haskell Compiler (GHC) [25].

**Clojure** is a dynamically typed mostly functional general purpose programming language. Since it is a Lisp dialect, it also includes a macro system, and makes use of the "code-as-data" philosophy. Clojure is compiled to run in the Java Virtual Machine (JVM), so Java libraries are accessible [26].

**F#** is a strongly typed multi-paradigm programming language that encompasses functional, imperative, object-oriented programming methods. F# is a member of the ML language family and originated as a .NET Framework implementation of a core of the programming language OCaml [27].

We chose to proceed with F#, since it is supported by the `mono` runtime, which several game engines use. This means that more effort is focused on applying the functional paradigm to gameplay instead of attempting to glue a functional language to an engine.

## 3.2 Game Engines

Before implementing F# support in an engine, we need to research how accessible the different engines are. We could create a engine our self, but it creates a big overhead and we need to focus time on production of parts that are not important to the actual project. We worked with four engines last project, CryEngine, Godot, Unity and Unreal Engine. How well documented are the different engines for modifications or plugin creation.

All engines either have source available or a documented APIs that makes it possible to create extensions or change modules of the engine.

**CryEngine** uses the `mono` runtime, so it should be possible to use F#. This is however not the case, as the CryEngine editor will not allow for creation of F# scripts, nor is it possible to add F# files to the C# project without getting compilation errors. It was not possible to add an F# project to the Game-solution, as this was just removed by the editor each time it reloads the solution. Adding a reference to a Dynamic Link Library (DLL) file containing some F# functions is not possible either, as this results in some other compilation error. Since the source code for the editor is available, it would be technically possible for us to implement F# support. This seems like a large task itself, so it would be more appropriate for a separate project [28].

**Godot** can support F# fairly simply, but does require some basic setup [29]. With Godot it is possible to have both the default C# project and the F# project in the same solution. Allowing for easy code integration between the two languages. Again this lets us focus on running tests, instead of spending time gluing components together [30].

**Unity** can support DLL-files created from any .NET project, including F#. The project just needs to be built as a library, so it a DLL-file is created, which can be imported into the editor and used. The library DLL can also be referenced from the C# project, so some of the codebase can be F#. Unlike in Godot, this requires two instances of the Integrated Development Environment (IDE), as the projects cannot be in the same solution due to Unity's C# solution management. So while it is possible, it is cumbersome. It is possible to extend the editor using the Unity Editor API, so there might be a way in which to implement F# such that it is less of a hassle to work with the language in Unity [31].

**Unreal Engine** is excluded as there is no official support for .NET. There is the *MonoUE* project [32], which aims to integrate `mono` into Unreal Engine. It is possible to make plug-ins with Unreal [33] and the engines source code is available [34] such that F# support could be implemented, but with no `mono` or `dotnet`, implementing F# would be, like with CryEngine, a bigger project more appropriate for a different time [35].

The choice falls down to Godot and Unity because CryEngine and Unreal Engine does not support the creation of F# scripts, and require too much work to support the usage of F#. So we choose to continue with Unity, both because of personal experience with the engine and because Unity is a more widely used engine. So it will be easier to find test participants for our user study, which is covered in Chapter 6.

## 3.3 Champagne prototyping

Used when an existing product needs a new feature added (such as a game engine requiring a new language added). The champagne prototype is a visual simulation of a feature implemented in an existing system [36]. The prototype is non-functional, but it is designed in such a way that it looks to the user like it is implemented. This gives the user the option to explore and interpret the feature. Since the champagne prototype is implemented in the existing environment, the user is able to explore and use the fully functional program as they are used to with menus and everything.

An important step in champagne prototyping is the recruitment of highly qualified participants. In their experiment, they wished to recruit participants that were sufficiently advanced. Advanced enough the point where using matrix calculations and user defined functions would make sense to the participant.

In recruiting these highly qualified people, the interviewer approached people, without an appointment, carrying a bottle of champagne. Not wishing to give anything away, they asked some questions to assess if the participant was eligible.

# Chapter 4

# Extending Unity to support F#

This chapter describes the exploration of adding F# support to Unity in a way that is user-friendly enough to enable novice Unity developers to use F# to implement their gameplay code.

## 4.1 Current Support

Currently there is no official support for using F# in Unity. Since F# projects compile to DLL-files, exactly like C# projects, a F# project can be built and the resulting DLL-file can either be referenced by the Unity C# project, or moved into the `Assets`-folder of a Unity project. The Unity Editor will then detect the change and index the file. After indexing the DLL-file, the classes inheriting from `MonoBehaviour` are available for use as normal components in Unity, where they can be dragged onto a `GameObject` to attach it.

To use a `MonoBehaviour`-class as a component, in a project created outside the Unity Editor, it is necessary to add references to the Unity DLL-files that Unity also references when creating a C# project through the Unity Editor. The path to these files may vary between Operating Systems (OSs), and also depends on which version of Unity is installed. Since the Unity Editor knows the correct path for a given installation, the references can be extracted from the C# project file created by Unity.

When the correct Unity references are added, it is possible to create types that inherit from `MonoBehaviour` in the F# project. After compiling this project, it is necessary to copy over the resulting DLL-file and the `FSharp.Core.dll` file to Unity's `Assets`-folder. `FSharp.Core.dll` is necessary since it contains the F#.NET Standard library classes, such as `FSharpOption`.

All these steps make Unity development with F# cumbersome, and developers would end up using a significant amount of time on all the legwork instead actually writing gameplay code in F#.

## 4.2 A Solution

The first step is to look for similar solutions for inspiration. The Casanova compiler adds support for the Casanova language, which is transpiled to C# [19]. This approach is partly different to what is needed for F# support, since F# already compiles to Common Intermediate Language (CIL), which runs on the .NET platform. Even though no transpilation is needed, the Casanova pipeline is still similar to the one required for F# support. After new code is written in Casanova, a command is executed to call the compiler, and the output from the compiler is then moved into the Unity `Assets`-folder.

Armed with this information, we have created a Unity package that adds an F# menu to automate the extraneous work. The menu has buttons and functionality for building, creating and opening F# projects, shown on Figure 3. The package handles creating an F# project, extracting and including the correct Unity references, and finally, opening the F# project in the default editor, for convenience. Finally, the package also handles copying over the needed files to the Unity `Assets`-folder.

The package is released on GitHub and is freely available under `GPLv3` license, so the license will not get in the way of developers wanting experiment with using F# in their Unity games [37].

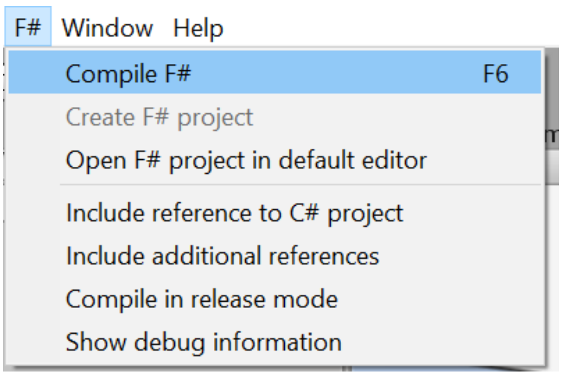



**Figure 3:** Screenshot of the F# menu in Unity Editor.

The menu additionally allows changing some build-specific settings, such as build mode and which Unity packages to references. For convenience, the menu also supports creating the F# project in a single click and opening the project in the default editor for such files. The menu finally has the `Compile F#` button, which takes care of the entire build process. The compilation/build process is handled by dotnet, depending on the number of lines of source code, the building process can take some time (about ten seconds). This can be improved by building through the IDE, since most IDEs use the *incremental build* option. The *incremental build*-option lets the compiler cache the latest build-result of each file. Next time the developer initiates building the project, the compiler can use the cached version of all the files that has not been changed since last build. This speeds up the build-time to around one to two seconds, which is much more acceptable. When the developer presses compile in the F# menu in the Unity Editor, it is detected that an IDE has created a recent build of the project. If the build is more recent than any changes to the files of the given project, building the project again is skipped and the needed DLL-files are copied over, thus saving several seconds. In Figure 4 an example of using the Unity package in action. The underlined file on the lower right is the compiled F# project. It expands to show all the different MonoBehaviours available inside.

To summarise, the developer only has to press Create F# project, then Open F# project in default editor, write some gameplay code, return to the Unity Editor and press Compile F# and wait for the project to build. All the MonoBehaviour-inheriting classes found are then made available by the Unity Editor. This streamlines the development process using F# quite a bit, but the developer still has to press compile themselves, whereas the Unity Editor automatically detects changes to C# files and compiles them.

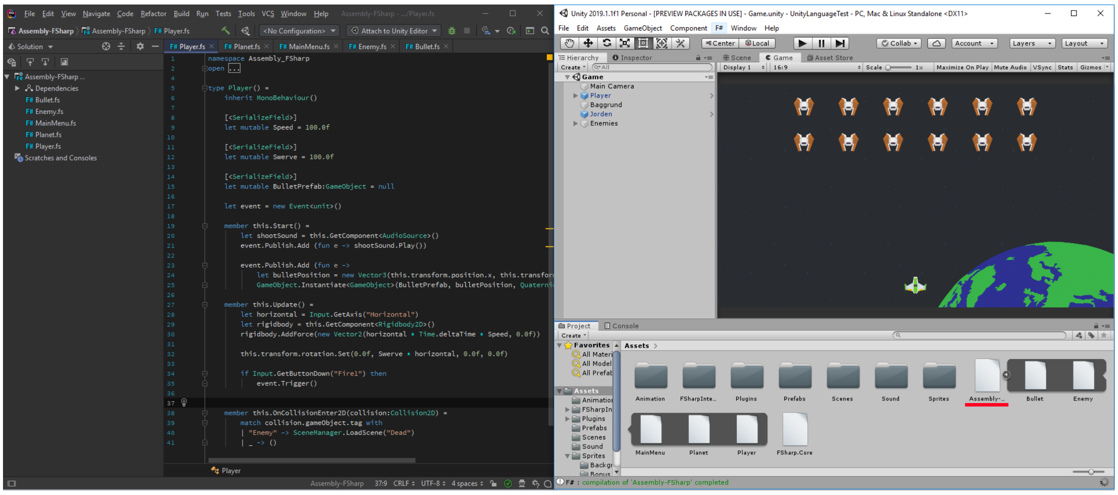


**Figure 4:** Screenshot of an example F#-Unity development environment.

# Chapter 5

# Benchmarks

In this chapter we are going to benchmark F# used in different environments so that we can see what performance the code can perform. This will be a continuation of the benchmarks done in [1], using the same benchmarking methods replicating the micro- and macrobenchmarks and expanding upon them. To these results we are going to compare with one of the more popular gameplay programming language used today, C# which was discovered in [1], which was showed to even outperform C++ in some instances when developing games [1].

## 5.1 Test Setup

In this section we discuss the foundation of the benchmarks. It will be discussed what type of benchmarks there will be used. Then our method of during the benchmarks will be discussed. Finally we will present the platforms that will be tested and what system on which the test are executed.

### 5.1.1 Types of Benchmark

We could not find a clear definition of what a micro-, macro- and application-benchmarks are, within the programming-technology field besides the one defined in [1]. In general benchmarks are tests of programs, that yield some metric. This metric may be memory usage, execution time or throughput [38]. The benchmarks are categorised by the size of the program under test. The following definitions is taken from [1].

**Microbenchmarking** is also known as component-benchmarking and is defined as *a benchmark testing a single and minimal unit of functionality and ex-*

*cluding start-up time*. In this case a unit of functionality is a single function, object, or equivalent programming construct, of small size. The goal is to test the performance of the single unit [1].

**Macrobenchmarking** is defined as *a benchmark testing multiple units of functionality and excluding start-up time*. The main difference between micro- and macrobenchmarking is the number of functional units. The goal of macrobenchmarking is to test the performance of a set of connected units [1].

**Application-benchmarking** is also known as program-benchmarking and is the broadest category of benchmarks. They are defined as *a benchmark testing a full application, consisting of multiple units of functionality and including start-up time* [1].

### 5.1.2 Method

The tests will follow the same method as in [1], where we use a benchmark that compensates for the warm-up of the virtual machine. We expect the same to be the case for game engines. In order to avoid running the tests while the engine and virtual machine is starting up, a small test-runner was written. The test-runner starts the tests when the Space button is pressed. To run the tests, the program is started followed by waiting three to five seconds before pressing space to start the tests, to allow for initial loading to complete. To the extent it was possible, the test-runner would output Comma Separated Values (CSV)-files containing the data for each language/environment. Another small script to merge and format the data was also written, which made it easier to work with the data [1].

### 5.1.3 Platform

This experiment examines F# and is going to be evaluating the language in different configurations. To have a point of comparison, C# will also be evaluated since it is one of the most widely used gameplay programming languages, available in most big free-to-use engines [1].

When evaluating F#, it of interest to test for a difference in performance when writing mutable and immutable code. The benchmarks will be performed using two different environments; Unity which uses a custom version of `mono` and `dotnet`.

### 5.1.4 System Setup

The system on which the tests were executed runs Windows 10 Pro and its specifications are listed in Table 1.

| Processor | | |
|---|---|---|
| Model | Intel Core i5 4210U | |
| Clock Frequency | 1.7 | GHz |
| Max Turbo | 2.4 | GHz |
| Physical | 2 | Cores |
| Logical[1] | 4 | Cores |
| **Memory** | | |
| Memory Size | 12 | GiB |
| Memory Speed | 1600 | MHz |
| Memory Type | DDR3L 1600 | |
| **OS** | | |
| Type | Windows 10 Education | |
| Version | Build 17134 | |

**Table 1:** System specifications

## 5.2 Microbenchmark

In this experiment we want to see how efficient F# is compared to C#. C# is the gameplay language used in most bigger free to use engines today [1], so it is a good language to compare performance to.

The investigation in the following experiment can be formulated into these questions:

- Is there a significant difference in performance, depending on what environment F# is run in?
- Can F# run as fast as C# in the same environment?
- Is there a significant difference in the mutable and immutable F# solution?

### 5.2.1 Test Case

The test cases in [1] was made to explore different aspects of game programming languages, the test cases was as follows and this is taken from [1]:

**Sestoft's Multiply** is listed in Section 5.2.1 [39]. This method is designed to prevent compilers from optimising the multiplication away with a constant

[1]Logical cores are sometimes called threads. However logical cores is used here to avoid confusion with the software concept; threads, which is distinct from hardware threads.

value as well as keeping the input relatively small. It represents a minimal computation that still has significant measurable execution time.

**Vector Math** is a series of vector operations, i.e. scaling a vector by a factor, multiplying two vectors, translating a vector, subtracting two vectors, calculating the length of a vector and calculating dot product of two vectors. This is done for vectors of two and three dimensions.

**Array Allocation** allocates and initialises an array of 100,000 elements and returns the last element.

**Primes** implements the Sieve of Eratosthenes algorithm [40] to generate all prime numbers that are lower than 100. This produces a list of numbers, the last of which is returned from the function.

```
private static double multiply(int i) {
    double x = 1.1 * (double)(i & 0xFF);
    return x * x * x * x * x * x * x * x * x * x * x * x * x * x * x * x * x * x * x
      ↪  * x;
}
```

**Listing 1:** Method proposed for benchmark by Sestoft [39].

Evaluating the microbenchmarks showed that there was too much focus on vector calculations. To compensate for that, array- and numerical benchmarks was added, besides having Primes and Sestoft's Multiply. For our array benchmarks we make use of the already established benchmarks "the Cowichan problems" [41]. Not all 13 problems will be implemented, but a select few that are relevant for game development.

For the numerical benchmarks a Fibonacci benchmark was added [42]. The Fibonacci benchmark in [42] is designed for concurrency testing of server and clients, but modified for the purpose of benchmarking the programming languages used for game development. We implement the Fibonacci algorithm in each language and measure how long it takes to calculate the nth Fibonacci number. We can then compare which language is the fastest at a given benchmark, and from that, the fastest at the benchmark suite in general.

Since the test originated in evaluating imperative and object oriented languages the benchmarks have a certain bias towards these paradigms, since they were create for these. To evaluate the functional language features, a series of functional programming benchmarks have been added, specifically making use of features from the functional paradigm. For F# that means that the data will be piped through a `map` function and then a `reduce`, whereas for C# LINQ will be used. Here `Select` for mapping and then `Sum` for reducing.

With the additions, the benchmark suite now includes:

**Numerical Math** is a series of numerical operations i.e. the new is Fibonacci but Sestoft's Multiply and Primes is also added to this category.

**Matrix Math** is a series of benchmarks implementiong some of the Cowichan problems, specifically Conway's Game of life, Invasion percolation and a Random number generator for a $N \cdot M$ matrix.

**Functional programming** A series of small benchmarks that make use of the features in functional languages, specifically MapReduce, Discriminated Unions and Pattern matching.

**Vector Math** the same series of vector math calculations as in [1].

### 5.2.2 Results

In this section the results of the test cases are presented and discussed. This is done on the basis of the research questions that were presented in the start of this section. The results from all the tests may be found in Appendix B. All tests results are listed as mean running time in nanoseconds and the graphs use logarithmic scale on the y-axis, as the running times vary wildly between the platforms.

#### Mutable and Immutable F#

In the exploration of the question "Is there a significant difference in the mutable and immutable F# solution?" we implemented a series of our microbenchmarks in F# such that they were in our opinion as equivalent as needed. Some microbenchmarks were excluded because it was not possible to our knowledge make an equivalent mutable and immutable version. Both version was compiled in `dotnet` 2.2.

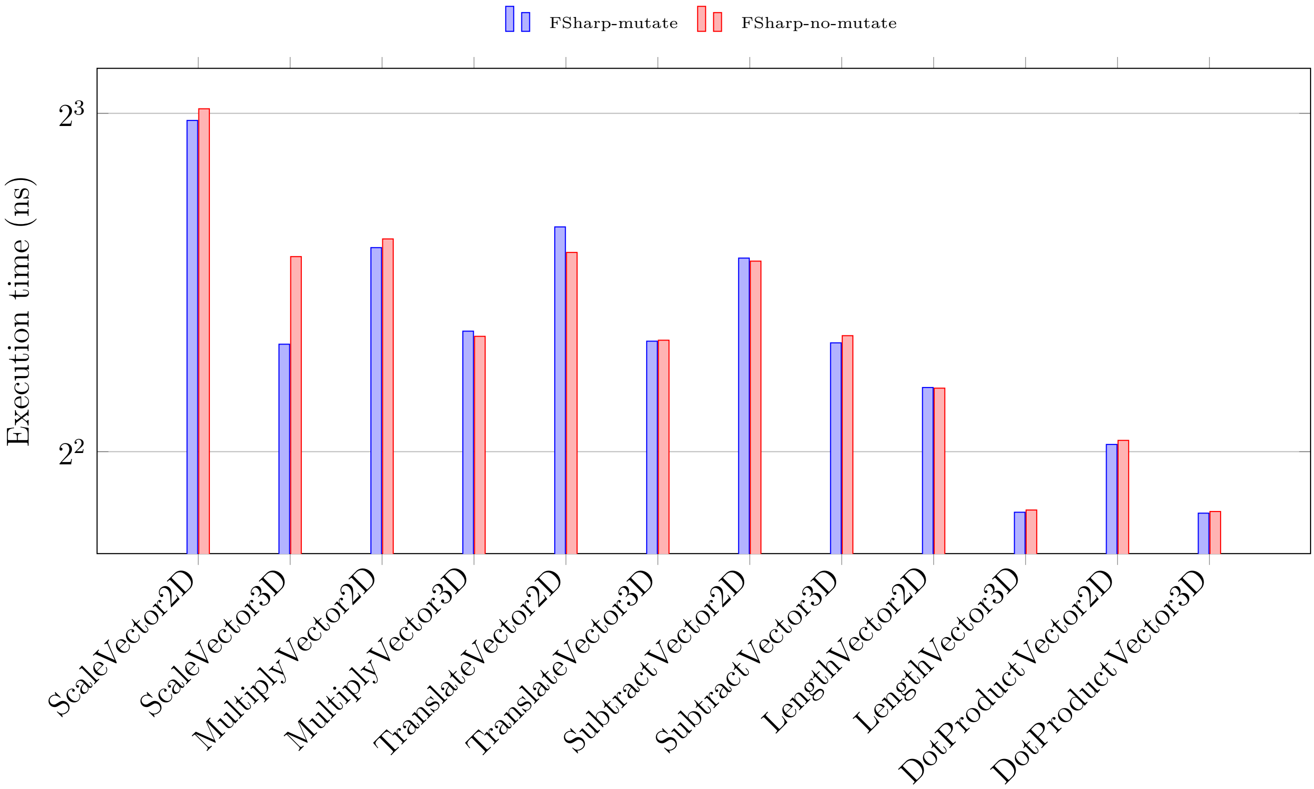



**Figure 5:** Graph over execution times in F# mutable and immutable implementations

We can see in Figure 5 that mutable edges it out and is faster in most cases, we can see that the mutable ScaleVector3D implementation outperformed immutable quite a bit. We believe we have made a proper solution, but there could always be the chance that the immutable implementation could be optimised more. It is worth mentioning, since there is such a huge difference in just one of the benchmarks.

### F# and C#

In order to evaluate the question "Can F# run as fast as C# in the same environment?" we will be taking the best results of mutable and immutable F# and compare them to C#. Both are run in the same runtime; `dotnet` v. 2.2

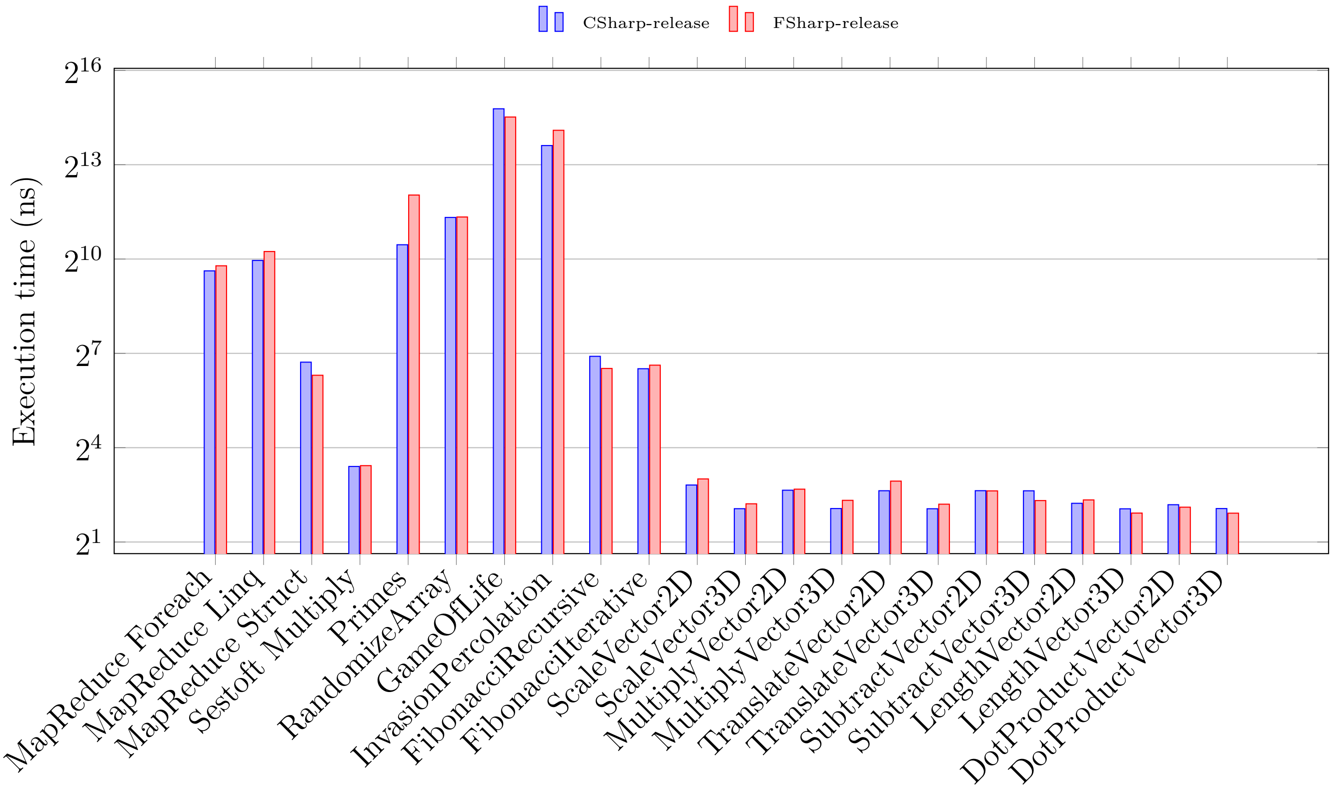



**Figure 6:** Graph over execution times in F# and C# in `dotnet`

When looking at F# and C# in Figure 6 C# slightly edges out F# in a many of the cases, all of them being very close, and F# being slightly faster in some such as the SubtractVector, LengthVector and DotProduct vector benchmarks. In F# `Array` is used instead of `foreach`, `Seq` instead of `Linq` and `Discriminated Unions` instead of a `struct` and `enum`. Where the F# version uses discriminated unions, instead of C# structs, it runs faster.

### Benchmarks in Unity

In our last test we want to answer the question "Is there a significant difference in performance, depending on what environment F# is run in?" to this we are going to run the suite of microbenchmarks in Unity, both debug and release, to see if Unity makes significant optimisations when creating a release build. The results from running the benchmark suite in Unity is then compared to the results from the `dotnet` runtime.

All benchmarks in Unity are performed with an empty scene, running in lowest possible resolution and graphical fidelity.

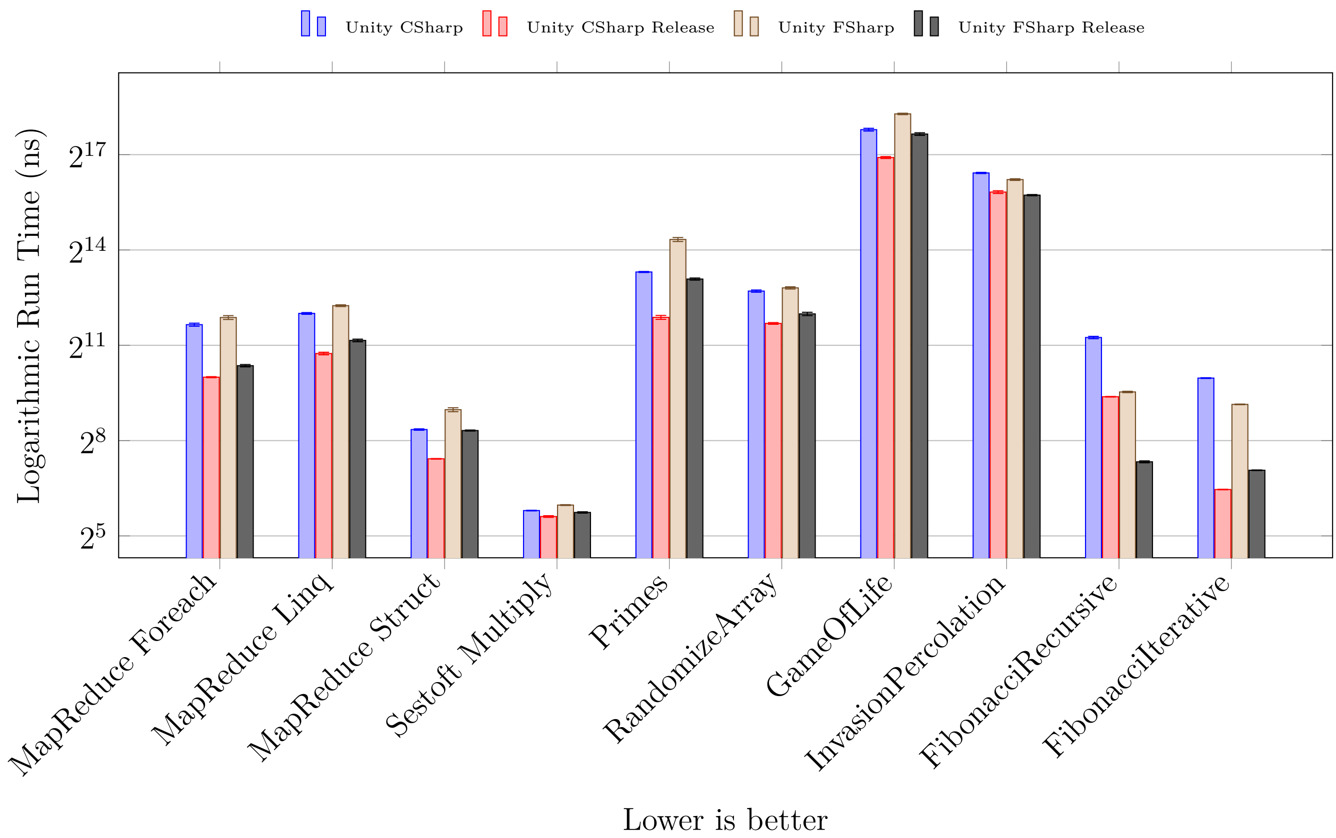



**Figure 7:** Graph over execution times in Unity C# and F# numeric tests.

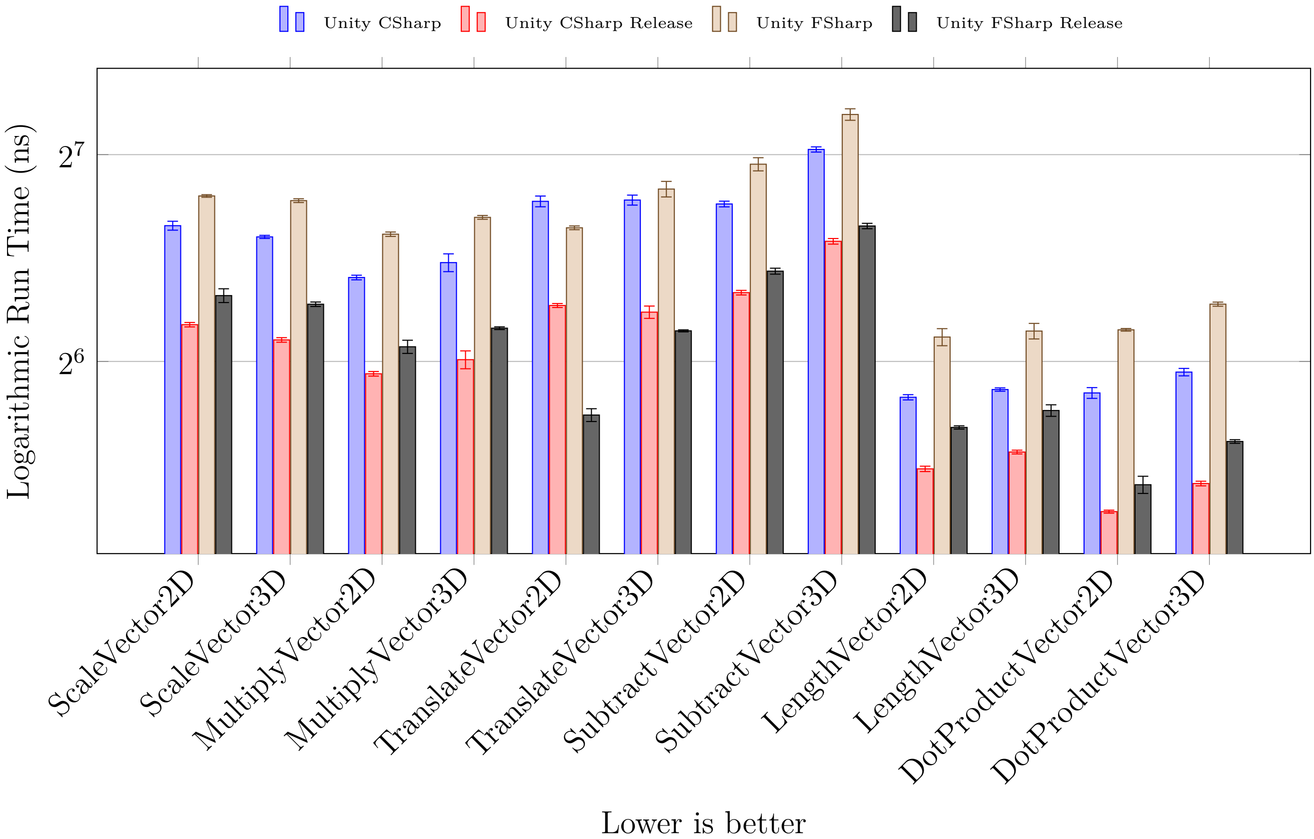



**Figure 8:** Graph over execution times in Unity C# and F# vector tests.

It is interesting that in Figure 7 and Figure 8 the results show that release F# is slower than C# in the editor and we can see a quite a optimisation from editor C# to release C#. Our guess is that the unity compiler, when building a release build does some optimisations that could be counter productive to F#. It is especially evident in the vector microbenchmarks in Figure 8.

In most cases in Figure 7 and Figure 8 C# outperforms F#, but we can see that F# outperforms C# in TranslateVector, both 2D and 3D. In 2D by quite a lot. F# also outperforms when Fibonacci is done recursive, but to our surprise C# barely outperforms F# in the functional benchmarks.

The statistical analysis and ranking methods are heavily inspired by Nanz et al. [43]. The statistical analysis used is the Wilcoxon signed-rank test (two-tailed variant). Every language/configuration is compared to every other in each of two groups: *Debug/Editor* and *Release*. As in [43] we say that when $p < 0.05$ there is a significant difference, when $0.05 \leq p \leq 0.1$ then the two languages/configurations tend to be different. We also say that the two results have little tendency to be different when $0.10 \leq p \leq 0.20$.

In figure Figure 9 and Figure 10 the ranking of each language in both modes is shown. In addition to their ranking, the statistical ordering is shown as well. The

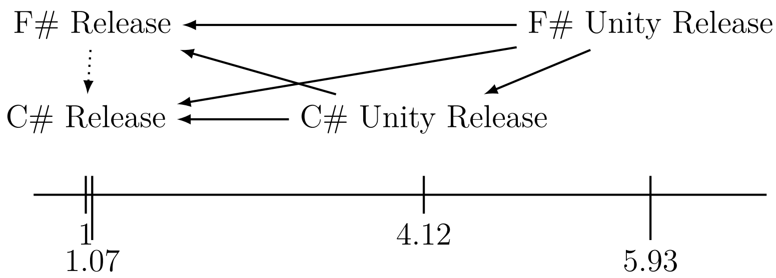



**Figure 9:** Ranking & Statistical ordering of microbenchmark tests in release modes.

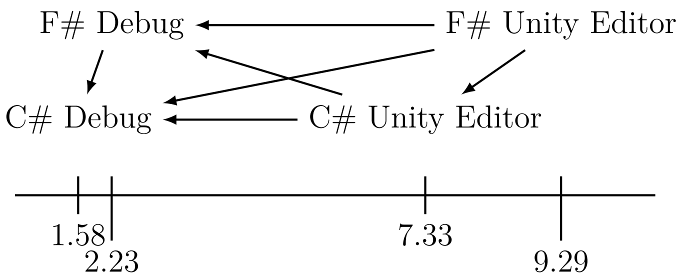



**Figure 10:** Ranking & Statistical ordering of microbenchmark tests in debug/editor modes.

statistical ordering is denoted by the arrows between configurations. A solid arrow denotes a significant statistical difference. A dotted would arrow means that the two results have little tendency to be different. As is the case of F# Release compared to C# Release (in Figure 9) we have that $0.10 < p < 0.20$. This is also visible on the ranking where F# is merely 7% slower than C#. The performance difference is of the same magnitude as the findings of the Casanova developers, who found a 5% difference between C# and F# [21].

Note that each individual microbenchmark was run 5 times, we use the average of those 5 runs to serve as the final result for each microbenchmark. The sum of those averages is the metric function $m$, for microbenchmarks. The rating function for microbenchmarks is defined in Figure 11, and is different from the one used in the original paper [43]. The language with the fastest aggregated run time will be of rank 1.0. A rank of 2.0 would have a mean run time twice as high as the fastest language. The rankings in both debug and release are based on all languages, though they are separated for readability.

One observation we made is that the difference between C# and F# is much higher in the released Unity build, than in the Unity Editor. It is difficult to see when comparing their absolute rank, but with a relative rank it is easier to visualise, shown in Table 2.

| | C# | F# |
|---|---|---|
| Release | 1 | 1.44 |
| Editor | 1 | 1.26 |

**Table 2:** Unity F# ranked with C# in same configuration.

$$rating_{micro}(L) = \frac{m(L)}{\min \ \sum m(L')_{\forall L' \in \mathbb{L}}}$$

where

| | |
|---|---|
| $L$ | a language/configuration |
| $\mathbb{L}$ | set of languages (and configurations) |
| $m : \mathbb{L} \rightarrow \{0, \infty\}$ | metric function |
| $rating_{micro} : \mathbb{L} \rightarrow \{1, \infty\}$ | rating function |

**Figure 11:** Rating function $rating_{micro}$ for microbenchmarks.

## 5.3 Macrobenchmark

For a more in-depth test of F#, and because microbenchmarks can be misleading [39], [44], a macrobenchmark was implemented so C# and F# could be evaluated in a broader scope.

As in [1], the macrobenchmark is an implementation of the AI simulation Wumpus World [45]. The same rule-set as in [1] is used, which is a simplified version of the actual Wumpus rule-set and the next section is a redone explanation of those rules. The simplifications was to remove: grab and release, turn and forward and climb. For a more detailed discussion as to why these simplifications was made, it can be read in [1].

### 5.3.1 Rules in Wumpus World

The map of Wumpus World is an N x M grid. Each cell may contain one object. The Wumpus World uses the term *neighbouring cells*. Given a cell X, the neighbouring cells are those to the north, east, south or west of X. The cells on the diagonal are not considered neighbours. A visual representation, as presented in [45], is shown in Figure 12. There is a blind agent whose mission is to find and collect the treasure without walking into a Wumpus or pit. A Wumpus gives off a stench that the agent can detect when in one of the neighbouring cells of the Wumpus. The same goes for for pits that have breezes that can be detected in neighbouring cells. Last there is Glitter which indicates that the treasure is in the current cell. The agent wins by getting back to the start position (0,0) after retrieving the treasure.

| | | | |
|---|---|---|---|
| Stench | | Breeze | Pit |
| Wumpus | Stench Glitter Breeze | Pit | Breeze |
| Stench | | Breeze | |
| Agent start | Breeze | Pit | Breeze |

**Figure 12:** A 4 x 4 Wumpus World with one Wumpus and two pits [45].

### 5.3.2 Platforms

The evaluation will be done in Unity with F# and C#. To be able to use F# in unity, we use the developed plug-in such that it is possible to compile F# and other necessary functions to make it work. To read more about the plug-in go to Chapter 4.

### 5.3.3 Metrics

Unlike microbenchmarks, there is no clear metric for how to measure time while doing macrobenchmarks. In [1] a discussion was made for which metric was best used for raw computational throughput measurement, and it was decided to measure how much time, one world iteration took. It still seems the most sensible so we decide to use the same metric or measurement for this Wumpus World evaluation.

### 5.3.4 Results

The difference between C# and F# is small, but there. As with the results in [1], there is both an immense startup time, and a repeating spike on garbage collection frames. The full dataset with all results is available in Appendix C.

l l l

In Figure 13 and Figure 14 we show the results for the macrobenchmarks, for C#

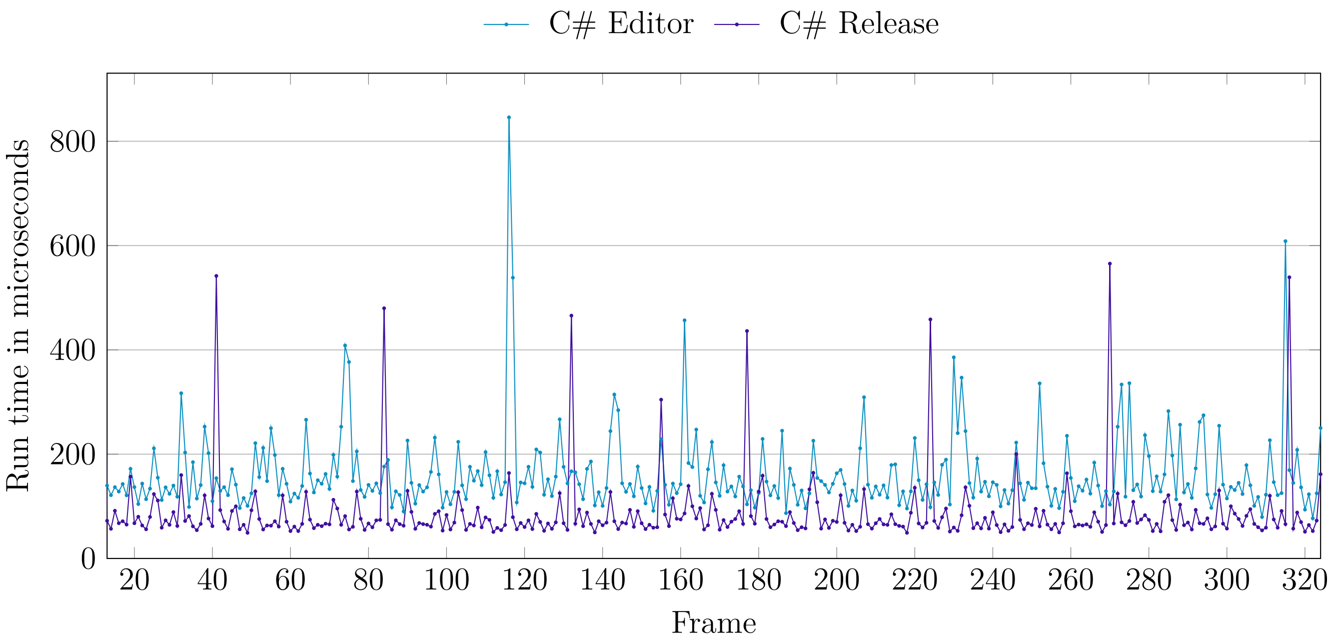



**Figure 13:** Wall clock-time for each frame of running the `World.Iterate` method.

and F# respectively, in both editor mode and in a release build. We omit the first of the 25 runs in these graphs, because the startup time is roughly 40x-260x larger than the average time for a single frame. This startup time is the highest in the release mode builds (especially C# Release), though still high in editor modes. In Figure 15 we compare C# with F# with both in release mode. The run time is similar, but in fact the difference is statistically significant (see below). A thing that stands out in Figure 15 is that F# has more smaller spikes than C#. An explanation for this may be that F# simply generates more intermediary garbage, which triggers Garbage Collection (GC) more often.

The ranking and statistical analysis for macrobenchmarks are similar to the the ones used with microbenchmarks. For the ranking and the analysis the first iteration is left out, because the startup time skews the results. The frame times for each iteration (13 individual frames) were aggregated into a sum for each iteration. The analysis and ranking was performed on the aggregated data.

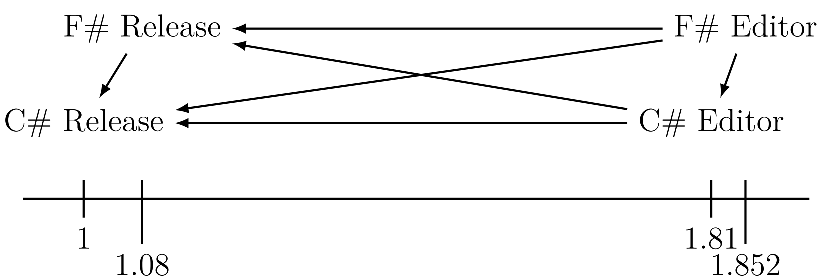



**Figure 16:** Ranking & Statistical ordering of F# and C# run times, excluding first iteration.

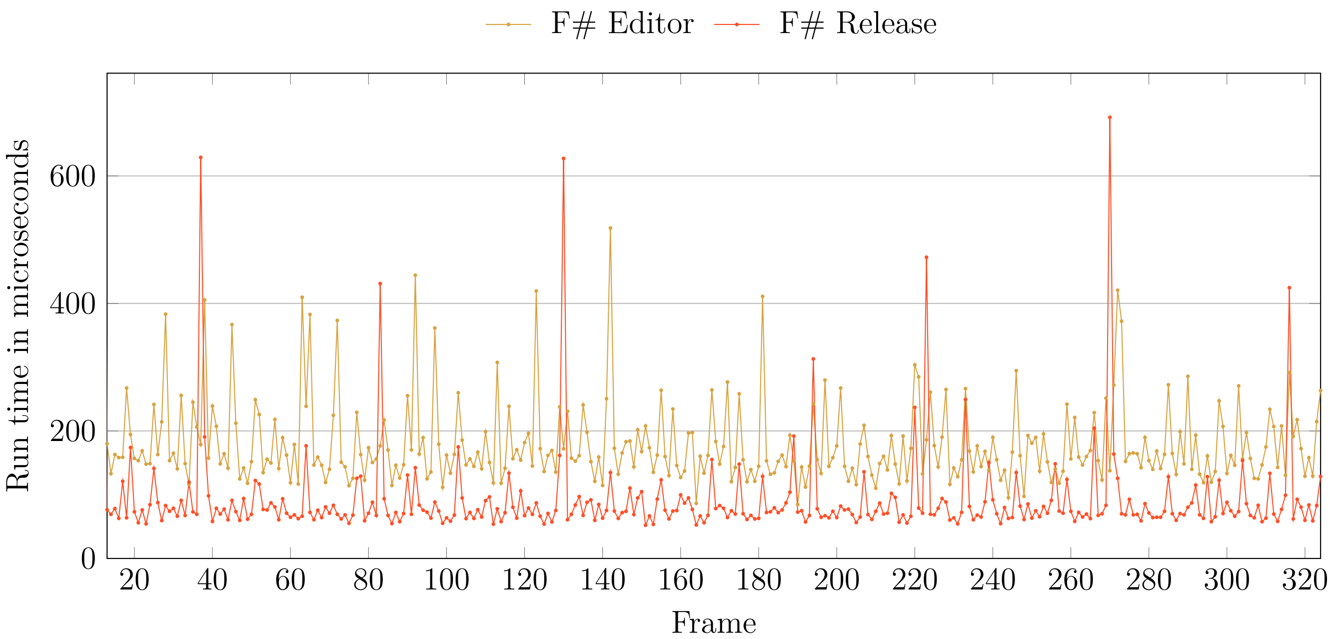



**Figure 14:** Wall clock-time for each frame of running the `World.Iterate` method.

In Figure 16 the ranking and statistical ordering of each language in both configurations is shown. The arrows have the same meaning as in Section 5.2. The analysis of the macrobenchmark did not show any occurrences of two configurations that only tended to be different, or that had little tendency to be different. If the first iteration was included in the analysis, the only thing that changed was that C#-release compared to F#-release only tend to be different, instead of a significant difference. That would make the performance gap between C# and F# even smaller. Further investigation in the startup time of both languages is considered future work (see Section 10.3).

The rating function is similar to the one in Section 5.2, although the metric function is different. The function is displayed in Figure 17. Here the mean run time of calls to the *World.Iterate*-method is used as the metric function $m$.

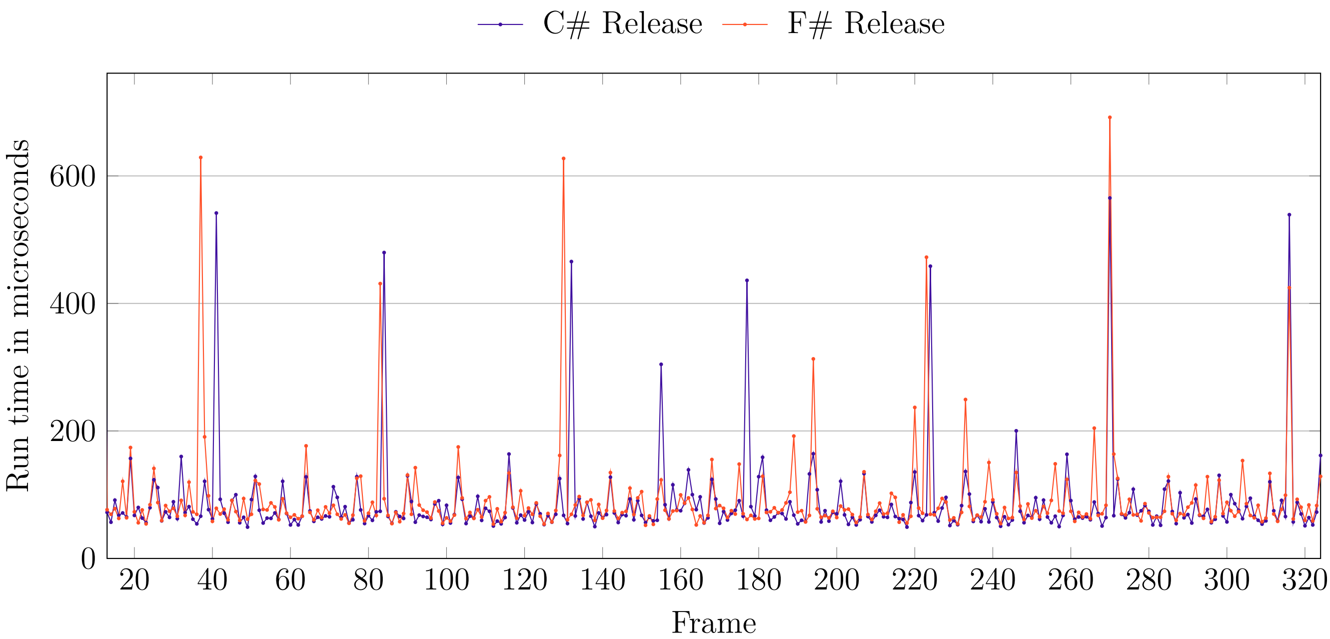



**Figure 15:** Wall clock-time for each frame of running the `World.Iterate` method.

$$rating_{macro}(L) = \frac{m(L)}{\min\ m(L')_{\forall L' \in \mathbb{L}}}$$

where

| | |
|---|---|
| $L$ | a language/configuration |
| $\mathbb{L}$ | set of languages (and configurations) |
| $m : \mathbb{L} \to \{0, \infty\}$ | metric function |
| $rating_{macro} : \mathbb{L} \to \{1, \infty\}$ | rating function |

**Figure 17:** Rating function $rating_{macro}$ for macrobenchmarks.

# Chapter 6

# Usability Study

For our usability study, we have taken inspiration from McCutchen et al. [46]. Additionally, some inspiration for our testing protocol is inspired by Champagne Prototyping as explained in Section 3.3. We designed our usability study to investigate the following:

- Does using F# versus C# programming affect the cognitive load experienced by developers?
- Does the use of F# versus C# affect the participant's experience?
- Are any of the differences observed between participants due to their level of expertise?

## 6.1 Participants and Tasks

We decided to test different levels of experience with programming by having two different test groups. One of the groups consisted of four kids age 11-16, who are learning to code games with help from an organisation called *Coding Pirates*, where volunteers teach a weekly course of 2 hours each. They use the Unity engine with C# to develop games. The second group consisted of six medialogy students on sixth and eighth semester, age 21-28. The medialogy studens have all taken at least one programming course at university level and have had some education in game development with Unity as part of their studies.

Due to the mixed levels of expertise of the participants, and the fact that neither group had used F# prior to the test, some concerns were raised:

1. The results may favour C#, since all participants have worked in it.

2. Little-to-no programming experience may mean that both languages appear more cognitively difficult than they are in reality.

3. No programming training may mean that the participants focus more on their experience with the syntax than the semantic constructs of the language(s).

4. The amount of participants is on the lower side of what we would have preferred, so results might not be a true representation of the age-groups' skill.

The tasks consists of implementing a game similar to Space Invaders [47]. This game is chosen because it is simple and can be implemented in a short amount of time. To ensure that the focus primarily is on coding, some aspects of the project is prepared in advance. Prefabs are included and ready to use, and only requires the script for a given prefab to be changed. The behaviour of the gameobjects will be written by the test subjects in a series of small self-contained tasks in C# and F#, and these tasks iteratively add the features to the game. Unlike the original Space Invaders, we omit score, multiple lives, barricades and enemies shooting back, for simplicity. So in this simplistic version, the swarm of enemies just move from side to side and when one of them has reached the side they all move forward. The player wins by shooting all enemies before they reach the player, and loses if unable to destroy them before they reach the player.

The tasks are presented in Table 3. Each task consists of a title and a description of the objectives to complete. Most tasks are divided into sub-tasks, to make it easier to understand, and easier to assess if a task is completed. The full version of the tasks can be found on the task sheet site[1], or in Appendix C.

| Task | Summary |
|---|---|
| Task 1 | Horizontal movement of the player (spaceship) |
| Task 2 | Reacting to click on mouse (fire gun) |
| Task 3 | Setting bullet prefab and sound when fired |
| Task 4 | Firing of a bullet and removing it from scene when out of bounds |
| Task 5 | Removing the enemy (and bullet) when hit |
| Task 6 | Changing scene when all enemies are dead |
| Task 7 | Synchronised movement of enemy ships and win condition |

**Table 3:** Summary of tasks.

## 6.2 Protocol

Prior to the study, the participants were briefly instructed on to how to use F# in Unity. The participants were informed they were allowed to use Google and

[1]https://sppt-2019.github.io/unity-fsharp-introduction/dmt/

any other form of documentation that could aid them in completing the task, just as if they where doing a normal coding assignment. A test session consisted of two parts, one where the participant used F# and one where they used C#. One group of participants start with F# and the other start with C#, to counter-balance the order of effects. Depending on the participants ability to understand English, they would get a different questionnaire. The older participants would get a questionnaire they would have to fill out as they completed each task, while the younger *Coding Pirates* participants would be handed a questionnaire after they had completed all the tasks they had time for. Both questionnaires were used to measure their perceived workload. The questionnaire followed the format of the NASA Task Load Index (TLX) questionnaire, a common tool used to let users self-asses their workload during a task [48]. The questionnaire consist of six sub scales: mental demand, physical demand, temporal demand, performance, effort and frustration. The participant is asked to rate their subjective workload on each scale, which ranges from 5 (low workload) to 100 (high workload) on a point 20-point scale in the original TLX questionnaire. In the version of the questionnaire used for this evaluation, the scale was modified so that the range starts from *Very Low* and ends with *Very High*, on a 9 point scale. The points between the lowest and highest option did not have any labels. These ratings can be averaged to yield the overall cognitive load. During the assignment the participants are allowed to ask questions about the syntax of F# and how to use F# with Unity, if they get stuck. Then they would receive some help or tips on how to proceed. This was primarily to counteract the issue of a new syntax being difficult to understand and use in the short amount of time availble for solving the tasks.

## 6.3 Results

There was two groups of participants, a group of 6 medialogy students and a group of 4 *Coding Pirates*. Because two of the *Coding Pirates* were very skilled they got put in the same group as the medialogy students and did the same questionnaire. Our questionnaire had physical demand included because it was part of the standard NASA TLX questionnaire, but the stat is not as important in these tests, since there there is low physical activity in programming. Some of the participants did not understand the question either, resulting in some participants not taking the question seriously. The rest rated it very low, so it was excluded. In Figure 18 we show the possible scores to the TLX scales. Green is least demand/frustration/effort and good performance. Red is the most demand/frustration/effort and poor performance.

**Figure 18:** The different colours showing the grades from most positive (left) to most negative.

All the participants progressed faster when using C#, and almost all of them got further using C# than F#. There was only a single participant that managed to complete the same number of tasks using F# as C#, though the results from the questionnaire show that the overall load was higher when using F#.

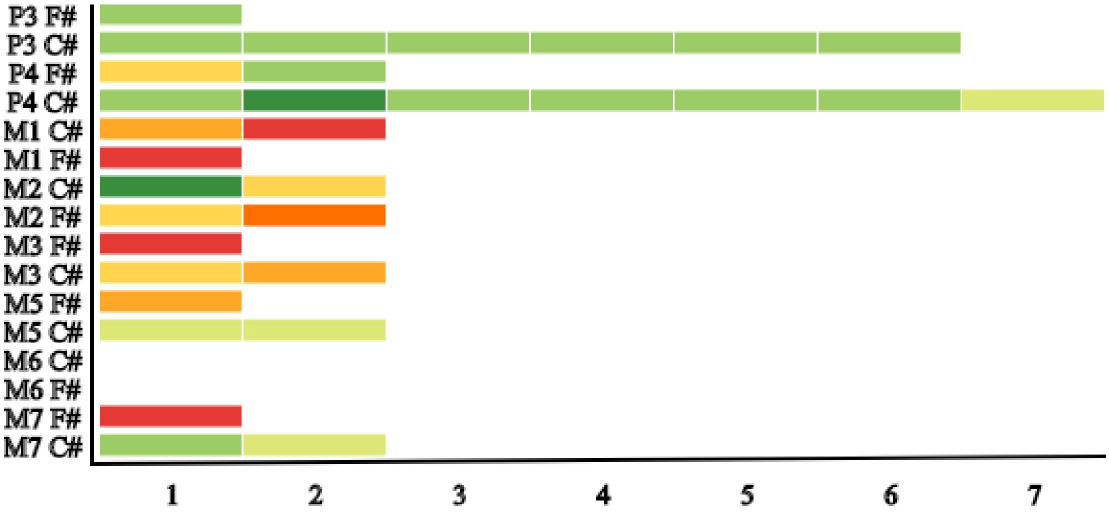



**Figure 19:** Overview of which tasks the participants completed, and how hard it was overall.

The number of tasks completed varied a lot between participants. Five out of the six medialogy students completed task 1, started working on task 2 and got some progress but did not complete it fully, using C#. As shown in Figure 19. The figure also shows the average "strain-level" for each completed task, to give an overview of how difficult the task was perceived. The language a participant started completing the tasks with is listed first. The task number is on the x axis, each box denotes that the participant has at least started on the task. The following sections go into depth with the different scales.

When looking at the progress of the *Coding Pirates* students in Figure 19, it is shown that both of them got to task 4. One completed all of the tasks and the other completed almost all the tasks and made it to task 6.

Of the 6 medialogy students, only 2 finished the first F# task and no one completed the second.

### 6.3.1 Mental Demand

When looking at the overview of the mental demand in Figure 20, it is clear that the participants experienced higher mental demand when using F# than C#, no matter what order they completed the tasks in.

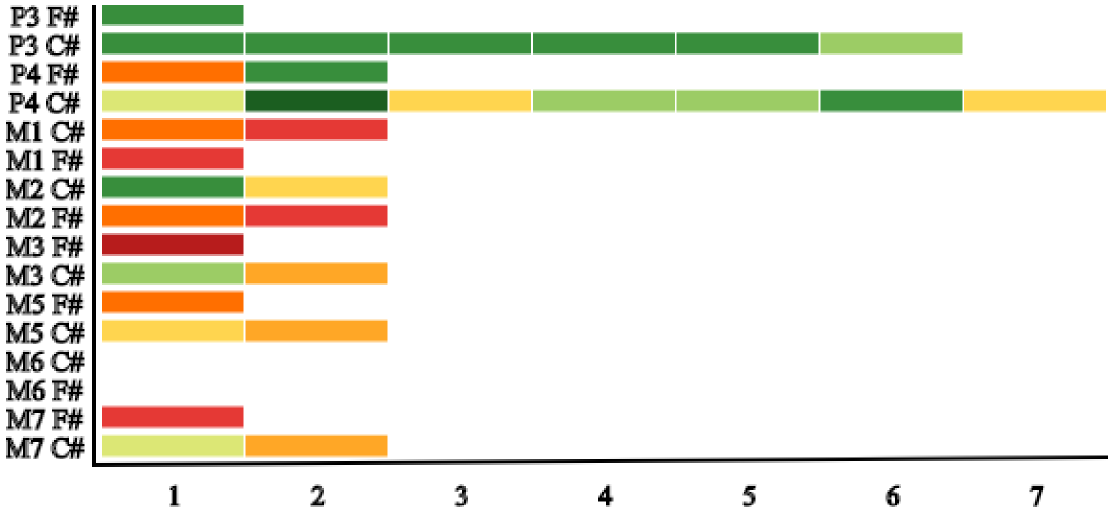



**Figure 20:** Overview of how great the mental demand was, according to the participants.

As Figure 21 shows, there is a much higher temporal demand with the F# tasks, than the C# tasks. In many of the cases the participants felt twice the temporal demand with the assignments, when using F#.

A part of it is because it is a new language and they have a limited amount of time, which puts a certain amount of stress on the participant to finish and learn the syntax of the language as fast as possible.

### 6.3.2 Temporal Demand

All participants felt a certain amount of rush as they had to do these assignments, no matter if they had to do C# or F# which makes sense since they only had 30 or 45 minutes to do each part, while also having a difficult time completing the task.

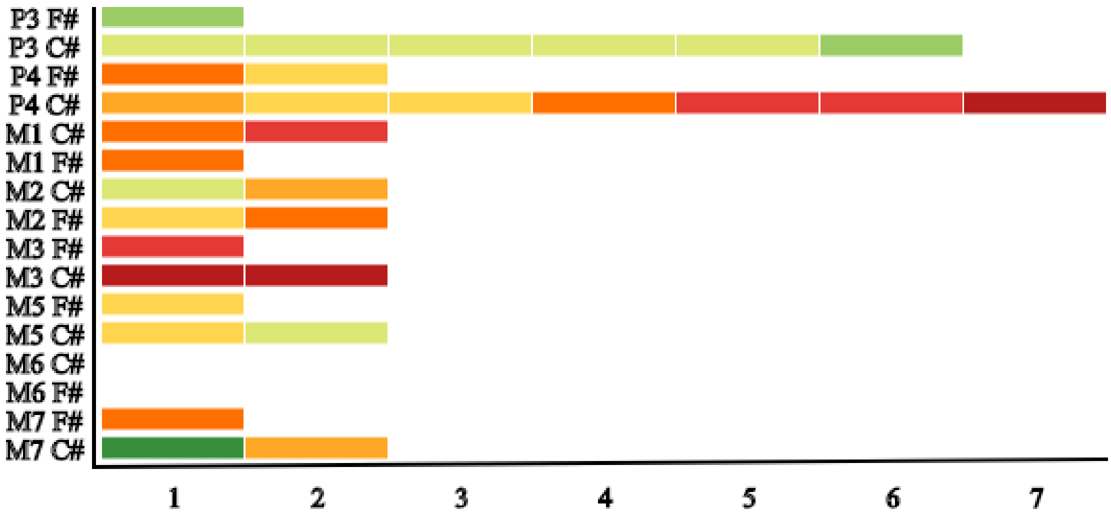



**Figure 21:** Overview of how great the temporal demand was, according to the participants.

It is very mixed whether participants felt the same rush when they worked with C# as F#, or if they felt more rushed doing F#. Something peculiar is that some participants felt less of a rush when using F#, which might have been because they accepted that it was a new language and they did not feel the same need to finish and prove themselves as much as they did with C#.

### 6.3.3 Effort

When looking at effort in Figure 22, the participants across the board rated the amount of effort used on the F# solutions to be greater than what was needed when using C#. It makes sense since it was a completely new language, where the participants also have to learn the syntax while solving the tasks.

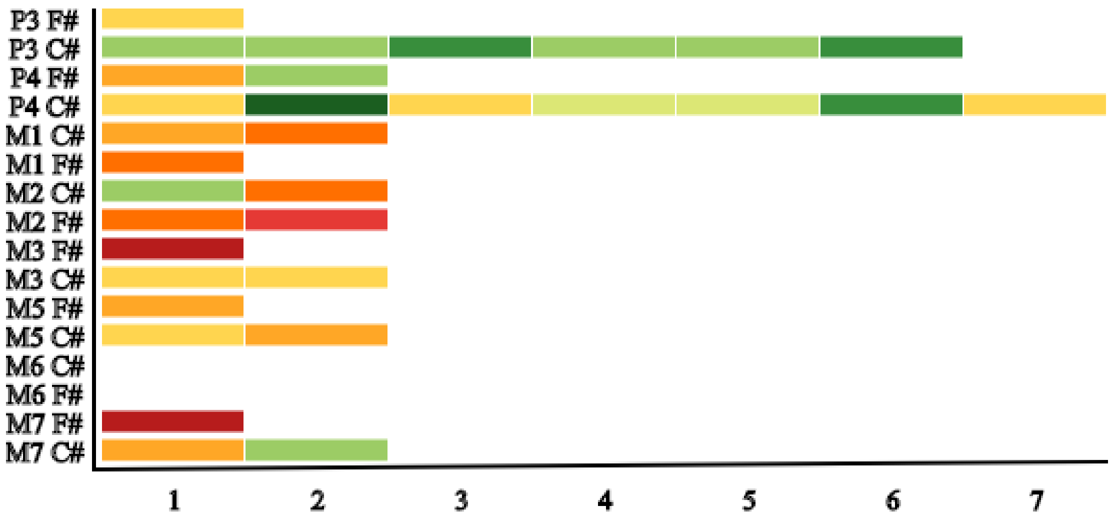



**Figure 22:** Overview of how great the participants rate their own effort.

It can also be seen that the effort increases for task number 2 for a lot of the participants, which might be caused by the task asking for a solution using events, which they might not have been using before.

### 6.3.4 Frustration

When looking at the insecurity overview show in Figure 23, it can be seen that in F# the medialogy students had a high insecurity and none of them got to task 2. While we can see that some of the *Coding Pirates* had a high insecurity with F# in task 1, but as they moved to task 2 the insecurity went down, this can be because they asked questions and got more familiar with the language.

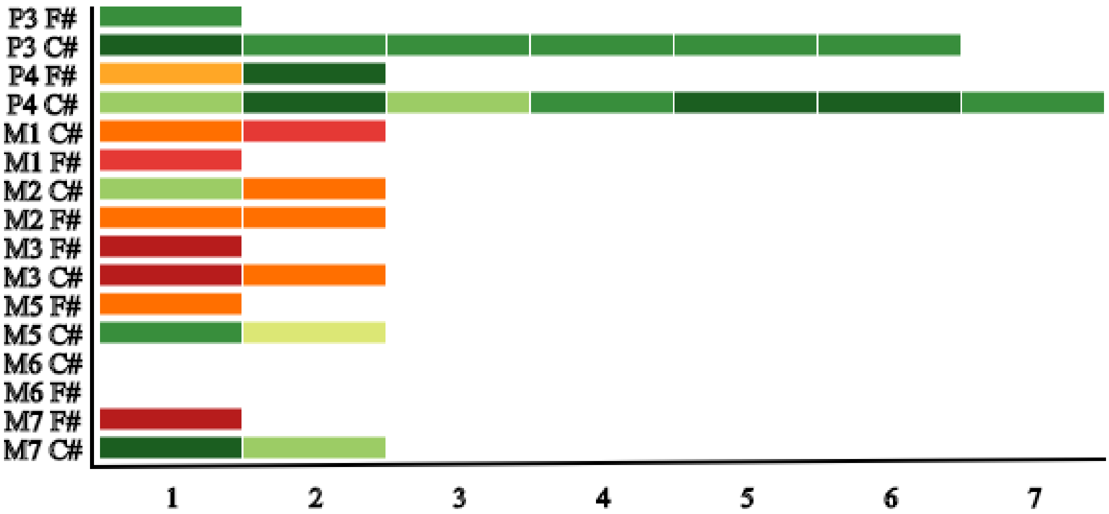



**Figure 23:** Overview of how insecure or frustrated the participants felt.

Looking at C#, the medialogy students were more insecure as they moved to task 2 and that might again be caused by them not having used events before. Whereas the *Coding Pirates* asked what events are and concluded that there were other ways to complete the task, instead of them getting stuck with completing the task using events. The fact that they asked instead of getting stuck may have caused them to progress faster. In Listing 2 is one of the participants solutions, avoiding the use of events.

```
if(Input.GetButtonDown("Fire1"))
{
    Instantiate(bullet, new Vector3(transform.position.x,
    transform.position.y + 1,transform.position.z),Quaternion.identity);
    audioSource.Play();
}
```

**Listing 2:** A participant's solution without the use of events.

This is a valid way to solve the assignment as the functionality is the same, though he did not use events as the tasks asked for.

## 6.4 Threats to Validity

An issue that was noticed while performing the usability tests with the medialogy students was the lack of questions. The tasks left them puzzled and confused, but as we were unable to view their screens while they where working, we were unable to spot if they were stuck with a problem for too long.

The medialogy students had 15 minutes less to complete each run of the tasks than the *Coding Pirates*, which can have resulted in a higher cognitive load and a more rushed solution. But because of the time available to the students, and the perceived skill of the students, we thought it would not make a big difference if they got 15 minutes less. The medialogy students were all very busy with their semester projects and could not dedicate more time than an hour for our evaluation.

Most of the problems the medialogy students experienced was understanding the F# syntax and getting the code to compile using the Unity extension.

The *Coding Pirates* children were not shy of asking questions, which helped them get started and after they has a basic understanding of the syntax, they could solve exercises and write code without our help.

It is an important threat to validity and it shows how hard it is to get started with a new language, because of the syntactical differences.

We had made two kind of surveys, one for the *Coding Pirates* and one for the students in order to avoid giving the younger participants problems with understanding the questions as they were in English, and because it might end up taking to much of their allotted time.

The *Coding Pirates* children were instead offered a Danish version of the survey in which they did not answer a page after each task, but instead only when the time was up. The two teenage *Coding Pirates* children were much more confident with the English language and opted for the English version.

The two younger *Coding Pirates* children were beginners in Unity game development and programming, so they required more help in completing the tasks and also worked together on solving the tasks because that is what they were used to and preferred. Evaluating the results of these two younger children should therefore be done differently than the results of the rest of the participants. Still, their results can provide useful insight into the problems experienced by young, novice game developers.

## 6.5 Improvements to the Extension

While conducting the tests with the participants, some problems arose. Specifically some of them had problems remembering to press compile in the F# menu in the Unity Editor. This may be because they are used to the way C# is supported in Unity, where any changes are automatically detected. A solution to this problem is considered in future work (see Chapter 10). It also seemed that the participants had a hard time understanding the build-errors that was logged to the console of the Unity Editor. The build-error output was made prettier and is now logged as an error instead of a message, to ensure that developers do notice that building the project has failed, and have a chance of understanding it.

# Chapter 7

# Qualitative Measures

In this chapter, the qualitative measures that we will be evaluating are presented. This includes the use of several of the dimensions from the Cognitive Dimensions framework. Some examples of the code submitted by participants are also shown and discussed to highlight usability problems they may have experienced, some of which could result in a higher cognitive load.

## 7.1 Usability of F# in Game Development

Using the the same dimensions as in [1], F# will be evaluated. To have some reference point it is compared with C#. Unlike the analysis in [1], two languages from different families, C and ML family, are compared. Instead of comparing two evaluations of C like languages against each other.

### Diffuseness/Terseness

First dimension we have is diffuseness/terseness, the dimension describes the amount of code is used to archive a certain goal. Where the minimum code used to archive a feature enables greater terseness.

When looking at F# it is possible to make use of the functional paradigm combined with the syntax to archive functionality on few lines, which allows greater terseness.

An example can be seen in Listing 3 taken from one of the participants, he defines the movement and the constraints of the player, the F# code is more condensed. When compared to C# in Listing 4 both implementations are short, but the F# implementation is shorter than its C# counterpart. If we look at line numbers in the two listings we can see that F# is 7 lines long and C# is 13 lines.

```
if (Input.GetAxis("Horizontal") = 0.0f)|>not then
            this.transform.position <- new Vector3(Input.GetAxis("Horizontal") *
              ↪   speed + this.transform.position.x, -4.0f, 0.0f)
if this.transform.position.x < -7.8f then
    this.transform.position <- new Vector3(-7.8f, -4.0f, 0.0f)
elif this.transform.position.x > 7.8f then
    this.transform.position <- new Vector3(7.8f, -4.0f, 0.0f)
```

**Listing 3:** A participant's F# player movement code

```
if(Input.GetAxis("Horizontal") != 0)
{
    transform.Translate (new Vector3(Input.GetAxis("Horizontal") * speed, 0, 0));
}

if(transform.position.x < -11)
{
    transform.position = new Vector3(-11, -4, 0);
}
else if (transform.position.x > 11)
{
    transform.position = new Vector3(11, -4, 0);
}
```

**Listing 4:** The same participant's C# player movement code

Both implementations are short but the C# implementation spans double the lines mostly because of C# good practice with curly braces. So both languages have good diffuseness and terseness, but it seems F# is a bit better.

### Hidden Dependencies

Hidden dependencies describes, among other things, the aspect of how certain code can depend on other code and the dependency is not entirely visible. This is a a issue that game development and game engines suffer from [16], [49], [50].

The most common examples are global values and GOTO statements. But, any subroutine is a hidden dependency since the languages provides no means to determine where it was used. The hidden subroutine issue is mitigated by most modern IDEs, which manage the code base and help developers display program flow, therefore we will focus on global variables which may affect behaviour elsewhere in the program.

An example of hidden dependencies in F# is that the order of which things are declared have an effect on the program. The order in which files are placed in the project file can make certain functions unavailable in other files, which can cause confusion and errors that can be difficult to understand in our experience. But when working with the test participants, they did not encounter any hidden dependencies in either C# or F#.

### Premature Commitment

F# presents a low premature commitment at a language level, but it is possible for the programmer to introduce premature commitments themselves through linguistic features.

In our experience the biggest issue with premature commitment when it comes to F# is that in `dotnet`, a lot of libraries are targeted for C#. The result is that when trying to use the libraries with F#, there can be complications with the API where certain keywords used in the library does not work in F# and you have to do a easy but still frustrating workaround and use extra keywords to make the library work correctly. But because of the way the task was set up, the participants did not experience any premature commitment. Since all files they had to write code in was set up, so they should just find them, write code and compile.

### Progressive Evaluation

Progressive evaluation is a feature of the programming environment more than the language itself. F# just like C# is a JIT-compiled language, which opens for the opportunity to recompile bits of a program instead of the entire program on compilation. Furthermore, the compiler utilises multithreading to recompile different parts of the code. This functionality provides modern IDEs with rich information such as type inference [51], without significant slowdowns. This enables the IDE to inform the programmer about the state of the program and advanced error detection/correction. C# has an advantage when it comes to progressive evaluation in terms of maturity, since IDE developers have had more time to refine the C# support. F#'s advantage is more integrated into the language itself. Since typical F# programs are easily split into side-effect free functions that can be run by themselves (optionally in F#-interactive). Furthermore, many game engines utilise progressive evaluation to support the data driven architecture [1]. When the test participants were doing the test, we noticed that using visual studio code, which we had recommended did not have auto-complete words and was only a text editor with highlighting which we had not noticed because our text editor of choice was Rider, by Netbrains. So it became harder for the participants to make F# code, because they would question themselves when writing the code and misspell because there was no code finishing or other helpful tools from a IDE.

### Role-Expressiveness

Role-expressiveness is often the responsibility of the programmer in textual programming languages. It is the programmers job to give variables and functions meaningful names signifying their role. Keywords are terminology that is related to languages and will thus be discussed when evaluating role-expressiveness.

When looking at F# and C# the languages have both different keywords and different syntax. This means the amount of keywords needed to express a construct vary depending on the language, even when expressing similar constructs. F# makes use of `let` as a keyword to declare both variables and functions. A variable is usually declared if there is only one name on the left of the = operator. There is an exception when returning a function bound to a single name with no parameters. A function is defined when there are more names (arguments) on the left side. Where in C# there is different syntax and keywords to define variables and methods/functions respectively.

```fsharp
let mutable v = new Vector2(1.0f)
```

```csharp
var v = new Vector2(1);
```

**Listing 5:** Definition of a mutable variable in F# and C#.

When defining a variable the keyword in C# is more clear as you are defining a variable using the `var` keyword. In F# the `let` keyword is used, then the name of the variable followed by the declaration.

In the example shown in Listing 5, F# syntax makes use of more keywords to define a mutable variable when compared to C#. When defining immutable variables, the opposite is true, that is F# is a keyword shorter than C#.

```fsharp
let v = new Vector2(1.0f)
```

```csharp
readonly Vector2 v = new Vector2(1);
```

**Listing 6:** Definition of a immutable variable in F# and C#.

If we examine function definition (shown in Listing 7), C# makes use of more keywords to to define a function. But since F# uses the same keyword for both, the role of the function/variable may not be as immediately clear as the different definitions found in C#.

The languages have two different takes on how to define variables, C# is of C style where you have `type name = value;` and F# have a style where the compiler infers the type, implicitly from the value `name = value;`.

```fsharp
let scaleVector2D scalar =
    scalar * 42.0f
```

```csharp
static float ScaleVector2D(int scalar)
{ return scalar * 42.0f; }
```

**Listing 7:** Definition of a function(/method) in F# and C#

The `static` keyword are not necessary in all cases but they help close the gap towards how the F# function works. The access modifier is omitted in the C#, which results in the method being `private`. Since both the return type and type of each parameter is inferred, there is less code to write, but the role of the written words will not be as clear as in C#. Although it should be noted that it is also possible to explicitly specify both return type and the type of all arguments in a F# function, making it optionally as expressive as C#.

Operators may be different from language to language. Even within a single language an operator can mean different things. This is the case with the = operator in F#. The operator is used to declare variables & functions, and to express boolean equality. In Listing 8 both of these cases are shown. This may cause confusion for programmers, because in many other languages, including C#, the second statement in Listing 8 would reassign the value of b to true.

```
let b = false
if (b = true) then ...
```

**Listing 8:** Assignment and boolean expression F#.

### Viscosity

When talking about viscosity in F#, the same arguments can be made as was done in [1] about C#. Every function and their implementation is done in the same file. Just like in C#, which means that arguments and the function changed in a single location. So as it was concluded in [1], C# had low viscosity and so F# must have too, and it can be further reduced by disciplined adherence to best practice standards. This thought matches the results found in [17], which is that the viscosity is low in both C# and F# and they argue the reason is because higher viscosity is found in visual programming languages. They further go to show that viscosity can be increased by the programmers.

# Chapter 8

# Discussion

In this chapter, the validity of the results of the benchmarks are discussed. The results from the usability study are explained while discussing their validity. The choice of engine will also be discussed and it will be considered if there is other engines that would have worked better or would maybe have given different results. The results of the usability study showed an interesting difference between the *Coding Pirates* children and the medialogy students, which will be discussed as well.

## 8.1 Amount of Participants

As has been mentioned in threats to validity, the amount of participants were not as high as we had hoped for, there is no right amount of participants. It has been shown in results of the usability evaluation that the best amount of participants for a quantitative evaluation is around 20 [52]. They admit that coming by 20 expert users in an area is tough and a handful (5) of users often catch most issues in a usability evaluation [53]. In [54], Skov et al. show that a usability study using 5 participants highlighted 85% of the critical usability problems, and 68% of the serious. Though we did not use the Instant Data Analysis (IDA) method described in the paper, their finding that observing users of a system will yield most of the important usability problems while requiring less time, is useful.

This of course is usability evaluation, with the goal being to catch as many usability problems in a system, whereas we just want as much data as possible, to be able to pinpoint the differences in the use of the two "systems". Having too many participants would just have resulted in many sightings of the same usability problems, which was already showing with the medialogy students, as all of them completed almost the same (low) amount of tasks.

## 8.2 Functional Paradigm or Just Functional Language

An issue with the usability evaluation is the lack of functional thinking. Even though the evaluation was performed in F#, there was not any real functional thinking involved. The same actions were performed in an imperative step-by-step manner regardless of language used. This can be seen in Listing 10 and Listing 9, where some code from one of the participants is displayed. The same 3 steps are displayed in both languages, with some minor differences:

1. Assign speed variable
2. Apply translation
3. If "Fire1" clicked, raise event

```
member this.Update() =
let mutable speed = Input.GetAxisRaw("Horizontal") * Time.deltaTime
this.transform.position <- this.transform.position + new Vector3(speed,0.0f,0.0f)
if (Input.GetButtonDown("Fire1")) then
    event.Trigger()
```

**Listing 9:** Update loop of Player.fs component in usability evaluation.

```
baseSpeed = baseSpeed * Time.deltaTime;
speed += Input.GetAxisRaw("Horizontal") * Time.deltaTime;
transform.position = new Vector3(transform.position.x + baseSpeed + speed,
  ↪   transform.position.y);
if (Input.GetButtonDown("Fire1"))
    OnClicked();
```

**Listing 10:** Update loop of Player.cs component in usability evaluation.

While this does no change anything in regards to whether F# is a useful game development tool or not, it unfortunately means that we can not say as much about the functional paradigm, besides that F# is a functional language and they performed the task in the language.

## 8.3 Performance Results

At first, two implementations of the benchmark was created; one in F# and one in C#. This raised some concern that small differences in the resulting CIL might skew the results in either direction. The results showed that the concern was valid, and needed to be mediated. The solution was to create an independent benchmark project, which could be packed into a NuGet package and installed to any .NET project.

The performance comparison between F# and C# is conducted using exactly the same benchmark, as it is installed to both projects from the same package. The measurements of mean running time and deviation should thus be directly comparable.

## 8.4 Choice of Engine

It was decided to use Unity because it was one of the easiest engines to extend to support F#, while also being one of the most popular game engines, which made it easier to find participants. The other choice would have been Godot if we wanted to implement F#, but Godot is still a new engine and is not as well known yet, so finding participants that knew of it would be difficult and putting our participants through both having to learn a new engine and a new language would further increase the cognitive load. It would be interesting to see how F# would perform in another engine that is widely used in the industry, such as Unreal Engine. That would also allow for collecting more data, as to show how slow Unity is compared to other engines; whether it is the model the Unity engine uses, or just `mono` that is slow.

## 8.5 Medialogy Students versus *Coding Pirates*

One interesting result from the evaluation, was that the *Coding Pirates* children were able to solve more tasks than the medialogy students. This came as a surprise because the medialogy students were attending 6th and 8th semester, and had been through a programming course and a Unity course. Nonetheless, the two older *Coding Pirates* children were able to solve more tasks, both using C# and F#, than any of the medialogy students. A reason for this can be that the medialogy students are attending the study because they are interested in digital creative arts such as drawing/modelling, world-building and storytelling. To them, programming might just be another chore to complete when developing a game. That could result in less focus on the craftsmanship of programming. Another source of stress for the medialogy students may have been that the duration of the evaluation was 30 minutes instead of 45 minutes like the *Coding Pirates*. On top of that, the medialogy students did not have time for a 5-minute introduction to F#, which may have helped them understand the language and syntax better.

# Chapter 9

# Conclusion

In this report, we set out to answer if it was viable to use the functional programming paradigm in game development. To test this, F# was evaluated on how well it would work as a gameplay programming language. To do this, a number of benchmarks were run to see how well the language would perform compared to already existing gameplay programming languages. Also, a usability study was performed to see how good the participants would be learning a functional programming language. F# was tested against C#, not because they both run in `dotnet`, but because C# is one of the most frequently used gameplay programming languages in free-to-use game engines [1]. F# is a functional-first multi-paradigm programming language, whereas C# is a imperative-first multi-paradigm programming language. Because of this, F# seemed like a good candidate and since they both use the same runtime, it is straight-forward to compare the performance results.

## 9.1 Performance

The results from the performance benchmarks show that the performance of F# is comparable to C# (Section 5.2& Section 5.3). The new `dotnet` runtime from Microsoft has made it trivial to create F# programs with comparable performance to C# programs. This shows that the old argument that programs written in functional languages inherently are too slow to yield acceptable performance. The mantra has always been that if you want performance, you need imperative languages [7], [8]. Many have further focused the argument by claiming that C++ is the most suitable language for applications that require high performance [5].

In [1] we used benchmarks to show that C# code running on the `dotnet` runtime could yield performance comparable to C++, and significantly better performance in some cases.

Through the benchmarks described in this report, we have demonstrated that it is possible to write F# code that yields performance comparable to C#, running in the `dotnet` runtime. The same can not be said for the `mono`-derived runtime that is used in Unity, since benchmarks show that the JIT optimiser used in that runtime is not able to optimise the code as much as the `dotnet` runtime, though the performance may still be acceptable for many types of games. From these results it would seem that the argument for enforcing the use of imperative languages, because of performance, is a moot point.

The adoption of functional languages should no longer be inhibited by the argument of performance, since we have shown that the performance is comparable.

## 9.2 Usability

An evaluation of how easy the adoption of F# would be to novice Unity developers was performed. The evaluation showed that developers were able to transfer their existing knowledge of the Unity API to F#, but also that many struggled with learning the syntax of a new language while solving the tasks, which were expected. The biggest issues for both of the two groups of participants, the medialogy students and the *Coding Pirates* children, was getting started and understanding the slight differences in syntax and how to run F# in Unity. This was the problem most commonly experienced by the medialogy students, but they did not ask for help. Instead they got stuck with syntax errors and did not manage to complete many tasks. The *Coding Pirates* children asked for help when they got confused and we could quickly tell them how to fix the syntax error or how to express something in F#. They were quick to learn the necessary F# syntax, and were able to solve several of the tasks without having problems with the language. The big difference between the two groups was the mentality towards asking for help. It would be interesting to see how far the medialogy students would have gotten, had they asked for help with syntax and compilation. Overall, the group of medialogy students seemed to find the experience stressing and confusing. Only a single student expressed an interest in learning more F# at the end of the evaluation. The *Coding Pirates* children seemed to find it interesting to learn a new language and use it to create a game. Both of the teenage *Coding Pirates* children seemed to be interested in F# at the end of the evaluation.

# Chapter 10

# Future work

This chapter presents different ideas for what future work could be interesting to research, following this project. Things that would be interesting to research is topics such as, updating the plug-in on how it compiles, because we noticed a repeating frustration with the users. While also wanting to figure out the extensive start-up time that Unity have and try using the new features Unity introduced while this paper was being written or try to use the `dotnet` prototype of Unity, engineers at Unity have worked on. It will also be proposed why it would be interesting to research the use of different engines with functional programming and how the microbenchmarks could be expanded such that the results would be even more extensive.

## 10.1 Unity Extension

As mentioned in Chapter 4, it was noticed that the participants had problems remembering to press compile in F# in the Unity Editor. A solution to this could be automatically detecting if the F# files have been changed when focusing the Unity Editor. If any changes are detected, the compiler could be invoked. A problem with this is that the hooks in the Unity API only allows for reacting to changes to files in the `Assets`-folder. An alternative to detecting the file changes when the Unity Editor is focused, is detecting the changes when the developers enters play-mode. This is also available to hook onto using the Unity API and does not require the files to be placed in the *Assets*-folder.

## 10.2 Benchmarking the Old .NET Framework Runtime

It would be interesting to see how the old Windows-only .NET Framework runtime performs compared to `mono` and the new, cross-platform `dotnet` runtime. A result that could be interesting to see is whether `mono` is slower than the old, windows-only .NET runtime. If it is, then it seems counter-intuitive to base the game engine's scripting runtime on it, though this was most likely because of `mono` being cross-platform and open-source, which the old .NET runtime was not. The new `dotnet` runtime is both cross-platform and open-source, released under MIT license.

## 10.3 Investigation on Start-Up Time

As explained in Section 5.3, the macrobenchmark suffered under an extensive startup time. Why exactly this is so extensive would be an interesting find, beyond the guess that it is just JIT warm-up. Especially so since the startup is slower in C# than it is in F#. One way to approach this would be to utilise Unity's Profiler tool, shown in Figure 24.

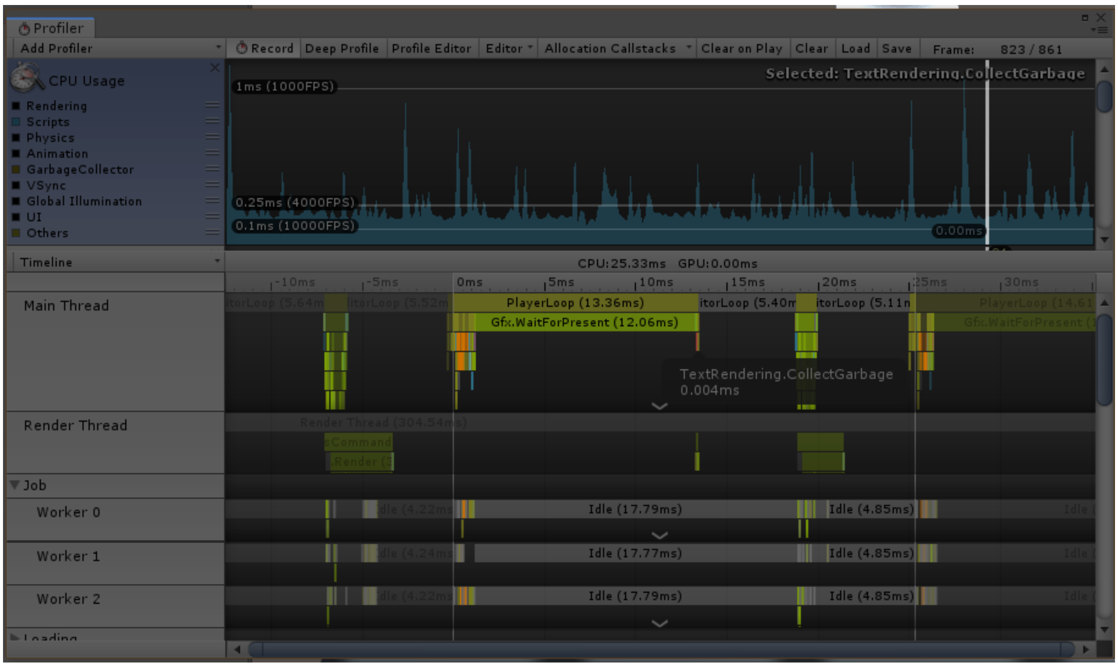



**Figure 24:** Screenshot of the Unity profiler tool.

## 10.4 Utilizing Unity's New Features

In 2019 Unity has released a new "Data-Oriented Technology Stack", which promises performance improvements, multi-threading and better readability[1]. It would be interesting to see how the new stack compares to C# running in `dotnet` Release mode, and also to the old Unity results. Both the new Entity Component System (ECS) paired with the new Burst compiler may approach or (optimistically) possibly exceed `dotnet` in performance.

In addition to code optimisation, a new incremental garbage collector has been added as well. One of the claims in the release notes is "[...] support for incremental Garbage Collection to avoid GC Spikes"[2]. We suspect that the spikes that were visible in the graphs shown in Section 5.3 are these GC spikes.

## 10.5 Unity with `dotnet`

Engineers at Unity has been looking at the `dotnet` runtime for performance improvements, and has even gone as far as experimenting with creating a prototype [55]. But the Unity development team has many things on their road-map for Unity, before getting to work further on integrating the `dotnet` runtime into Unity. If and when the prototype becomes mature enough to be made publicly available, it could be interesting to run the benchmark suite and verify the performance gains that are expected from the `dotnet` runtime.

## 10.6 Exploring Different Engines

An aspect that would be interesting to look into is to see the performance other engines would have using F# compared to their native gameplay programming language. We decided to focus on Unity because of its popularity, meaning that it would be easier to find participants that had some experience with the engine. But there are other game engines used both in the industry and by amateurs. Engines such as Unreal Engine, Godot and CryEngine that are freely available for everybody to use, but also proprietary engines used in-house by companies, such as RAGE or Frostbite. To be able to do this, a new integration tool would have to be made for each individual engine. Which would require time depending on what language the engine already supports and how much API support is available.

BinSubaih et al. suggests that the future of game development lies in not just

[1] https://unity.com/dots
[2] https://unity3d.com/unity/beta/2019.1

in cross platform, but in cross engine development [56]. They talk of “G-factor portability” which is the idea that a games state, logic and model all should be game engine independent. With this in mind, some future work for this report is to port the F# plugin to different game engines. This way, the F# part of game development is game engine independent. This does not magically make different game engine APIs compatible with each other, but it removes the barrier of using different languages.

A step further, would be to more closely follow the ideas of BinSubaih et al, and unify different game engine APIs under a single API. This opens up the opportunity to develop a game without having to think of performance, graphics, engine licencing, before the game is in a state where the developers are ready to make such decisions.

## 10.7 Expanding the Microbenchmarks

It was decided that the microbenchmarks would be done mutably, because of the compatibility with C# and because the performance difference between immutable and mutable was varying and minimal in the small scope we tested it in. So we programmed the way we found most comfortable.

It would be interesting to explore if benchmarks would show a larger difference if they were programmed in a more functional way. Only using immutable variables could result in a increased need for GC, which could have an impact on performance.

## 10.8 Development time

For the usability evaluation, the participants all have experience with C# and Unity, and have never used F# before. This means that it is not possible to evaluate whether using F# had a positive impact on development time. To better be able to verify this claim, finding participants with little to no prior knowledge of F#, C# and Unity, but still some programming experience. Then performing a similar evaluation but with an hour, or perhaps even a mini-course, of learning the language they will be solving the tasks with. The results from this could help verify if game development using a functional language reduces development time. The amount of tasks solved and the average cognitive load can give an indication of whether the functional paradigm improves developer performance.

## 10.9 A Social Issue

In this report, a suite of benchmarks were performed and statistical analysis showed that F# and C# yields comparable performance. The bigger issue with functional programming is the social aspect of how the public conceive it. Functional programming is still seen by many as inherently slower than imperative programming, mostly because of the belief that since you have complete control of every single step with imperative programming, you can avoid any extraneous work.

The usability study was performed to see how novice programmers would handle and understand F#, when pitted against C# with which they had to solve the same tasks. Some participants had difficulties no matter the language, but others, specifically the teenage participants, picked up how to program in F# very quickly, after only needing help with the syntax at the beginning. So it would be interesting to see in future work, if a further study can be done testing the functional paradigm to see if it gets taught at a younger age might help them understand it easier or if it is teachable to older, more veteran C developers. Hopefully, such a study could show whether the functional paradigm improves developer performance. This might help change the view of the functional paradigm in the programming community, which in turn might improve adoption rate.

# Bibliography


[1] M. R. Andersen, T. S. Jensen, T. G. McCollin, T. Morell, and D. van Bolhuis, “An analysis of gameplay programming languages in free-to-use game engines”, Aalborg University, Tech. Rep., 2019. [Online]. Available: https://projekter.aau.dk/projekter/da/studentthesis/en-analyse-af-spilopfoerselsprogrammeringssprog-i-gratisatbruge-spilmotorer(99442369-1b23-4deb-b99d-060ae5cd5db2).html.

[2] J. Hughes, “Why functional programming matters”, *The computer journal*, vol. 32, no. 2, pp. 98–107, 1989. [Online]. Available: https://academic.oup.com/comjnl/article-pdf/32/2/98/1445644/320098.pdf.

[3] Z. Hu, J. Hughes, and M. Wang, “How functional programming mattered”, *National Science Review*, vol. 2, no. 3, pp. 349–370, Jul. 2015, ISSN: 2095-5138. DOI: 10.1093/nsr/nwv042. eprint: http://oup.prod.sis.lan/nsr/article-pdf/2/3/349/6087381/nwv042.pdf. [Online]. Available: https://doi.org/10.1093/nsr/nwv042.

[4] C. Maraffi and D. Seagal, “Leveling up : Could functional programming be a game changer ?”, 2012.

[5] J. Carmack. (Apr. 2012). In-depth: Functional programming in C++. English, [Online]. Available: https://www.gamasutra.com/view/news/169296/Indepth_Functional_programming_in_C.php (visited on Sep. 21, 2018).

[6] T. Sweeney. (2006). The next mainstream programming language: A game developer’s perspective. English, Epic Games, [Online]. Available: https://www.st.cs.uni-saarland.de/edu/seminare/2005/advanced-fp/docs/sweeny.pdf (visited on Nov. 20, 2018).

[7] zo0ok. (Jun. 2017). Functional programming sucks! (it is slow). English, TechFindings, [Online]. Available: https://techfindings.one/archives/2679 (visited on Dec. 6, 2018).

[8] J. Harrop. (Jun. 2012). Compilation - are functional languages inherently slow? English, StackOverflow, [Online]. Available: https://stackoverflow.com/questions/516301/are-functional-languages-inherently-slow (visited on Dec. 6, 2018).

[9] B. Edds. (Sep. 2018). Nu game engine. English, [Online]. Available: https://github.com/bryanedds/Nu/blob/master/Nu/Nu.Documentation/Nu%20Game%20Engine.pdf (visited on Sep. 25, 2018).

[10] Z. Corr. (Oct. 2016). Helm: A functionally reactive game engine. English, [Online]. Available: http://hackage.haskell.org/package/helm (visited on Sep. 25, 2018).

[11] R. Nasser and T. Gardner. (Feb. 2017). Arcadia: Clojure in unity. English, [Online]. Available: https://github.com/arcadia-unity/Arcadia (visited on Sep. 25, 2018).

[12] Unreal Engine. (2012). Unrealscript. English, [Online]. Available: https://api.unrealengine.com/udk/Three/UnrealScriptHome.html (visited on Feb. 19, 2019).

[13] ——, (2012). Kismet visual scripting. English, [Online]. Available: https://api.unrealengine.com/udk/Three/KismetHome.html (visited on Feb. 19, 2019).

[14] T. R. G. Green, M. Petre, *et al.*, "Usability analysis of visual programming environments: A 'cognitive dimensions' framework", *Journal of visual languages and computing*, vol. 7, no. 2, pp. 131–174, 1996.

[15] A. J. Ko and B. A. Myers, "Development and evaluation of a model of programming errors", in *Human Centric Computing Languages and Environments, 2003. Proceedings. 2003 IEEE Symposium on*, IEEE, 2003, pp. 7–14.

[16] J. Blow, "Game development: Harder than you think", *Queue*, vol. 1, no. 10, p. 28, 2004. [Online]. Available: http://faculty.salisbury.edu/~xswang/Research/papers/game/queuefeb04/blow.pdf.

[17] T. G. McCollin and T. Morell. (2019). A Game of Paradigms: A Usability Study of Functional Idioms in Gameplay Programming. English, [Online]. Available: https://morell.site/ (visited on May 27, 2019).

[18] M. R. Andersen, T. S. Jensen, T. G. McCollin, T. Morell, and D. van Bolhuis. (2019). Unity F# Introduction. Danish, [Online]. Available: https://sppt-2019.github.io/unity-fsharp-introduction/ (visited on May 27, 2019).

[19] M. Abbadi. (2015). Casanova-mk2. English, [Online]. Available: https://github.com/vs-team/casanova-mk2/blob/master/Unity/Casanova%20template%20and%20binaries/Samples/MyFirstSample/Assets/Editor/CasanovaAssetsImporter.cs (visited on Feb. 27, 2019).

[20] G. Maggiore, A. Spanò, R. Orsini, G. Costantini, M. Bugliesi, and M. Abbadi, "Designing casanova: A language for games", in *Advances in Computer Games*, H. J. van den Herik and A. Plaat, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 320–332, ISBN: 978-3-642-31866-5.

[21] G. Maggiore, A. Spanò, R. Orsini, M. Bugliesi, M. Abbadi, and E. Steffinlongo, "A formal specification for casanova, a language for computer games", Jun. 2012. DOI: 10.1145/2305484.2305533.

[22] G. N. Amador and A. J. Gomes, "Jot: A modular multi-purpose minimalistic massively multiplayer online game engine", 2014. [Online]. Available: http://vj2016.di.ubi.pt/Amador_JOT_57-62.pdf.

[23] A. Courtney, H. Nilsson, and J. Peterson, "The yampa arcade", in *Proceedings of the 2003 ACM SIGPLAN Workshop on Haskell*, ser. Haskell '03, Uppsala, Sweden: ACM, 2003, pp. 7–18, ISBN: 1-58113-758-3. DOI: 10.1145/871895.871897. [Online]. Available: http://doi.acm.org/10.1145/871895.871897.

[24] TIOBE. (2019). Tiobe index. English, [Online]. Available: www.tiobe.com (visited on May 27, 2019).

[25] Haskell. (2019). Main page. English, Haskell.org, [Online]. Available: https://wiki.haskell.org (visited on May 31, 2019).

[26] T. Hickey and R. Hickey. (2018). Clojure - rationale. English, [Online]. Available: https://clojure.org/about/rationale (visited on Nov. 29, 2018).

[27] F. S. Foundation. (2019). Clojure - main page. English, [Online]. Available: https://fsharp.org/ (visited on May 31, 2019).

[28] Crytek GmbH. (2018). Cryengine | the complete solution for next generation game development by crytek. English, Crytek GmbH, [Online]. Available: https://www.cryengine.com/ (visited on Oct. 31, 2018).

[29] L. Kokemohr. (2017). Lars kokemohr using f# in godot 3. English, [Online]. Available: www.lkokemohr.de/fsharp_godot.html (visited on Feb. 14, 2019).

[30] A. M. Juan Linietsky and the Godot Community. (2019). Godot engine - main. English, [Online]. Available: https://godotengine.org (visited on May 31, 2019).

[31] Unity Technologies. (2019). Extending the editor. English, [Online]. Available: https://docs.unity3d.com/Manual/ExtendingTheEditor.html (visited on Feb. 22, 2019).

[32] MonoUE and M. Hutchinson. (2018). MonoUE | Mono for Unreal Engine is a plugin for Unreal Engine that allows writing gameplay code with C# and F#. English, [Online]. Available: https://mono-ue.github.io (visited on Dec. 13, 2018).

[33] Unreal Engine. (2019). Plugins. English, [Online]. Available: https://docs.unrealengine.com/en-us/Programming/Plugins (visited on Feb. 22, 2019).

[34] Epic Games. (2014). Unreal engine, [Online]. Available: https://github.com/EpicGames/UnrealEngine.

[35] Unreal Engine. (2019). Main. English, [Online]. Available: https://www.unrealengine.com/en-US/ (visited on May 31, 2019).

[36] A. Blackwell, M. Burnett, and S. Peyton Jones, "Champagne prototyping: A research technique for early evaluation of complex end-user programming systems", IEEE, Sep. 2004, pp. 47–54. [Online]. Available: https://www.microsoft.com/en-us/research/publication/champagne-prototyping-research-technique-early-evaluation-complex-end-user-programming-systems/.

[37] M. R. Andersen and D. van Bolhuis. (2019). Unity F# Integration. English, [Online]. Available: https://github.com/sppt-2k19/unity-fsharp-integration (visited on Feb. 22, 2019).

[38] P. J. Fleming and J. J. Wallace, "How not to lie with statistics: The correct way to summarize benchmark results", *Commun. ACM*, vol. 29, no. 3, pp. 218–221, Mar. 1986, ISSN: 0001-0782. [Online]. Available: http://doi.acm.org/10.1145/5666.5673.

[39] P. Sestoft, "Microbenchmarks in java and c#", *Lecture Notes, Sept*, 2013. [Online]. Available: https://itu.dk/~sestoft/papers/benchmarking.pdf.

[40] Wikipedia. (Oct. 2018). Sieve of eratosthenes. English, Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes (visited on Nov. 9, 2018).

[41] G. V. Wilson and R. B. Irvin, *Assessing and comparing the usability of parallel programming systems*. Citeseer, 1995.

[42] R. Cardoso, M. Zatelli, J. Hübner, and R. Bordini, "Towards benchmarking actor- and agent-based programming languages", Oct. 2013. DOI: 10.1145/2541329.2541339.

[43] S. Nanz, S. West, K. S. Da Silveira, and B. Meyer, "Benchmarking usability and performance of multicore languages", in *Empirical Software Engineering and Measurement, 2013 ACM/IEEE International Symposium on*, IEEE, 2013, pp. 183–192.

[44] J. Y. Gil, K. Lenz, and Y. Shimron, "A microbenchmark case study and lessons learned", in *Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE! 2011, AOOPES'11, NEAT'11, and VMIL'11*, ACM, 2011, pp. 297–308.

[45] G. Ingargiola. (Feb. 2012). Cis587: The wumpus world. English, Temple University, [Online]. Available: https://cis.temple.edu/~giorgio/cis587/readings/wumpus.shtml (visited on Nov. 14, 2018).

[46] M. McCutchen, J. Borghouts, A. Gordon, S. Peyton Jones, and A. Sarkar, "Elastic sheet-defined functions: Generalising spreadsheet functions to variable-size input arrays", Nov. 2018, [Online]. Available: https://www.microsoft.com/en-us/research/publication/elastic-sheet-defined-functions-generalising-spreadsheet-functions-to-variable-size-input-arrays/.

[47] Giant Bomb. (2019). Space Invaders. English, [Online]. Available: https://www.giantbomb.com/space-invaders/3030-5099/ (visited on Feb. 22, 2019).

[48] S. G. Hart and L. E. Staveland, "Development of nasa-tlx (task load index): Results of empirical and theoretical research", in *Human Mental Workload*, ser. Advances in Psychology, P. A. Hancock and N. Meshkati, Eds., vol. 52, North-Holland, 1988, pp. 139–183. DOI: https://doi.org/10.1016/S0166-4115(08)62386-9. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166411508623869.

[49] V. Guana, E. Stroulia, and V. Nguyen, "Building a game engine: A tale of modern model-driven engineering", in *Games and Software Engineering (GAS), 2015 IEEE/ACM 4th International Workshop on*, IEEE, 2015, pp. 15–21.

[50] R. Nystrom, *Game programming patterns*. Genever Benning, 2014.

[51] K. Stolk. (2017). Anders Hejlsberg on Modern Compiler Construction, Youtube, [Online]. Available: https://youtu.be/wSdV1M7n4gQ.

[52] J. Nielsen. (2012). Quantitative studies: How many users to test? English, Nielsen Norman Group, [Online]. Available: https://www.nngroup.com/articles/quantitative-studies-how-many-users/ (visited on May 22, 2019).

[53] ——, (2012). How many test users in a usability study? English, Nielsen Norman Group, [Online]. Available: https://www.nngroup.com/articles/how-many-test-users/ (visited on May 22, 2019).

[54] J. Kjeldskov, M. B. Skov, and J. Stage, "Instant data analysis: Conducting usability evaluations in a day", in *Proceedings of the Third Nordic Conference on Human-computer Interaction*, ser. NordiCHI '04, Tampere, Finland: ACM, 2004, pp. 233–240, ISBN: 1-58113-857-1. DOI: 10.1145/1028014.1028050. [Online]. Available: http://doi.acm.org/10.1145/1028014.1028050.

[55] A. Mutel. (2018). Porting the Unity Engine to .NET CoreCLR. English, [Online]. Available: https://xoofx.com/blog/2018/04/06/porting-unity-to-coreclr/ (visited on May 27, 2019).

[56] A. BinSubaih, S. Maddock, and D. Romano, "A survey of 'game' portability", May 2019.

# Appendix A

# Acronyms

**AI** Artificial Intelligence

**API** Application Programming Interface

**CIL** Common Intermediate Language

**DLL** Dynamic Link Library

**ECS** Entity Component System

**FRP** Functional Reactive Programming

**VPL** Visual Programming Language

**GCC** GNU Compiler Collection

**GHC** Glasgow Haskell Compiler

**JVM** Java Virtual Machine

**OS** Operating System

**CSV** Comma Separated Values

**GC** Garbage Collection

**IDE** Integrated Development Environment

**JIT** Just-in-Time

**STM** Software Transactional Memory

# Appendix B

# Microbenchmark Data

| C# debug dotnet | | | | F# debug dotnet | | | | C# release dotnet | | | | F# release dotnet | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Test | Mean | Deviation | Count | Test | Mean | Deviation | Count | Test | Mean | Deviation | Count | Test | Mean | Deviation | Count |
| MapReduce Foreach | 841.087 | 17.926 | 524288 | MapReduce Array | 1024.932 | 13.612 | 262144 | MapReduce Foreach | 786.17 | 0.814 | 524288 | MapReduce Array | 878.114 | 3.001 | 524288 |
| MapReduce Foreach | 843.568 | 16.226 | 524288 | MapReduce Array | 1067.726 | 56.168 | 262144 | MapReduce Foreach | 791.497 | 5.013 | 524288 | MapReduce Array | 878.365 | 4.277 | 524288 |
| MapReduce Foreach | 831.242 | 8.698 | 524288 | MapReduce Array | 1014.008 | 7.638 | 262144 | MapReduce Foreach | 787.622 | 5.497 | 524288 | MapReduce Array | 883.425 | 13.016 | 524288 |
| MapReduce Foreach | 833.056 | 8.608 | 524288 | MapReduce Array | 1009.933 | 5.666 | 262144 | MapReduce Foreach | 790.261 | 4.573 | 524288 | MapReduce Array | 882.043 | 17.828 | 524288 |
| MapReduce Foreach | 828.946 | 11.334 | 524288 | MapReduce Array | 1006.983 | 8.939 | 262144 | MapReduce Foreach | 786.235 | 2.13 | 524288 | MapReduce Array | 885.052 | 13.902 | 524288 |
| MapReduce Linq | 1035.244 | 24.657 | 262144 | MapReduce Seq | 1204.058 | 12.089 | 262144 | MapReduce Linq | 985.219 | 15.621 | 262144 | MapReduce Seq | 1201.914 | 15.89 | 262144 |
| MapReduce Linq | 1011.744 | 10.155 | 262144 | MapReduce Seq | 1190.614 | 3.79 | 262144 | MapReduce Linq | 994.457 | 19.324 | 262144 | MapReduce Seq | 1186.597 | 1.99 | 262144 |
| MapReduce Linq | 1012.659 | 12.876 | 262144 | MapReduce Seq | 1198.995 | 12.879 | 262144 | MapReduce Linq | 982.25 | 8.225 | 262144 | MapReduce Seq | 1214.715 | 19.617 | 262144 |
| MapReduce Linq | 1008.993 | 8.106 | 262144 | MapReduce Seq | 1193.162 | 7.223 | 262144 | MapReduce Linq | 995.127 | 17.484 | 262144 | MapReduce Seq | 1211.072 | 10.033 | 262144 |
| MapReduce Linq | 1015.716 | 11.982 | 262144 | MapReduce Seq | 1197.153 | 13.158 | 262144 | MapReduce Linq | 998.691 | 18.399 | 262144 | MapReduce Seq | 1221.769 | 20.157 | 262144 |
| MapReduce Struct | 142.578 | 2.098 | 2097152 | MapReduce Unions | 281.43 | 6.545 | 1048576 | MapReduce Struct | 105.398 | 1.163 | 4194304 | MapReduce Unions | 77.993 | 1.214 | 4194304 |
| MapReduce Struct | 140.094 | 1.4 | 2097152 | MapReduce Unions | 278.606 | 1.1 | 1048576 | MapReduce Struct | 105.507 | 0.988 | 4194304 | MapReduce Unions | 78.052 | 0.448 | 4194304 |
| MapReduce Struct | 140.158 | 0.753 | 2097152 | MapReduce Unions | 280.025 | 3.283 | 1048576 | MapReduce Struct | 104.856 | 0.226 | 4194304 | MapReduce Unions | 77.105 | 0.465 | 4194304 |
| MapReduce Struct | 139.908 | 0.75 | 2097152 | MapReduce Unions | 278.342 | 1.655 | 1048576 | MapReduce Struct | 105.285 | 1.153 | 4194304 | MapReduce Unions | 77.287 | 0.842 | 4194304 |
| MapReduce Struct | 144.327 | 6.362 | 2097152 | MapReduce Unions | 279.437 | 2.449 | 1048576 | MapReduce Struct | 105.601 | 1.191 | 4194304 | MapReduce Unions | 84.184 | 8.303 | 4194304 |
| Sestoft Multiply | 16.322 | 0.083 | 16777216 | Sestoft Multiply | 25.337 | 0.073 | 16777216 | Sestoft Multiply | 10.567 | 0.079 | 33554432 | Sestoft Multiply | 10.879 | 0.13 | 33554432 |
| Sestoft Multiply | 16.659 | 0.591 | 16777216 | Sestoft Multiply | 25.97 | 0.118 | 16777216 | Sestoft Multiply | 10.61 | 0.099 | 33554432 | Sestoft Multiply | 10.756 | 0.087 | 33554432 |
| Sestoft Multiply | 16.55 | 0.159 | 16777216 | Sestoft Multiply | 26.062 | 0.418 | 16777216 | Sestoft Multiply | 10.528 | 0.052 | 33554432 | Sestoft Multiply | 10.724 | 0.083 | 33554432 |
| Sestoft Multiply | 16.589 | 0.269 | 16777216 | Sestoft Multiply | 26.196 | 0.464 | 16777216 | Sestoft Multiply | 10.515 | 0.042 | 33554432 | Sestoft Multiply | 10.703 | 0.079 | 33554432 |
| Sestoft Multiply | 16.285 | 0.107 | 16777216 | Sestoft Multiply | 26.091 | 0.256 | 16777216 | Sestoft Multiply | 10.63 | 0.108 | 33554432 | Sestoft Multiply | 10.776 | 0.227 | 33554432 |
| Primes | 2139.125 | 14.377 | 131072 | Primes | 4828.185 | 91.102 | 65536 | Primes | 1405.523 | 7.895 | 262144 | Primes | 3646.698 | 36.125 | 131072 |
| Primes | 2163.568 | 45.122 | 131072 | Primes | 4789.545 | 12.776 | 65536 | Primes | 1403.528 | 14.909 | 262144 | Primes | 4784.644 | 765.349 | 131072 |
| Primes | 2189.421 | 73.126 | 131072 | Primes | 4769.418 | 26.41 | 65536 | Primes | 1393.515 | 16.515 | 262144 | Primes | 4404.627 | 185.864 | 65536 |
| Primes | 2146.107 | 12.003 | 131072 | Primes | 4797.351 | 44.35 | 65536 | Primes | 1401.289 | 12.052 | 262144 | Primes | 4509.219 | 405.34 | 65536 |
| Primes | 2164.624 | 20.142 | 131072 | Primes | 4784.08 | 33.395 | 65536 | Primes | 1404.408 | 14.36 | 262144 | Primes | 3612.145 | 32.185 | 131072 |
| RandomizeArray | 2712.374 | 12.212 | 131072 | RandomizeArray | 2926.532 | 26.359 | 131072 | RandomizeArray | 2534.127 | 22.94 | 131072 | RandomizeArray | 2598.917 | 25.41 | 131072 |
| RandomizeArray | 2766.126 | 95.508 | 131072 | RandomizeArray | 2923.712 | 33.524 | 131072 | RandomizeArray | 2555.347 | 26.609 | 131072 | RandomizeArray | 2584.947 | 21.786 | 131072 |
| RandomizeArray | 2705.866 | 9.04 | 131072 | RandomizeArray | 2912.191 | 11.882 | 131072 | RandomizeArray | 2533.987 | 17.861 | 131072 | RandomizeArray | 2568.451 | 8.066 | 131072 |
| RandomizeArray | 2724.621 | 24.89 | 131072 | RandomizeArray | 2923.715 | 24.52 | 131072 | RandomizeArray | 2556.543 | 31.634 | 131072 | RandomizeArray | 2587.906 | 23.584 | 131072 |
| RandomizeArray | 2701.603 | 15.234 | 131072 | RandomizeArray | 2922.234 | 7.312 | 131072 | RandomizeArray | 2610.127 | 74.527 | 131072 | RandomizeArray | 2581.496 | 26.436 | 131072 |
| GameOfLife | 47912.363 | 325.375 | 8192 | GameOfLife | 68808.652 | 409.302 | 4096 | GameOfLife | 28183.036 | 133.557 | 16384 | GameOfLife | 23607.209 | 345.748 | 16384 |
| GameOfLife | 47909.119 | 161.095 | 8192 | GameOfLife | 69403.774 | 1450.791 | 4096 | GameOfLife | 28001.655 | 177.81 | 16384 | GameOfLife | 23308.954 | 75.01 | 16384 |
| GameOfLife | 48300.581 | 620.044 | 8192 | GameOfLife | 69120.977 | 339.178 | 4096 | GameOfLife | 28009.982 | 290.076 | 16384 | GameOfLife | 23302.052 | 372.771 | 16384 |
| GameOfLife | 47831.526 | 111.985 | 8192 | GameOfLife | 68734.351 | 581.746 | 4096 | GameOfLife | 27924.346 | 229.314 | 16384 | GameOfLife | 23442.487 | 121.45 | 16384 |
| GameOfLife | 48786.008 | 1220.741 | 8192 | GameOfLife | 69185.923 | 876.984 | 4096 | GameOfLife | 28202.855 | 343.77 | 16384 | GameOfLife | 23354.747 | 197.808 | 16384 |
| InvasionPercolation | 15313.796 | 231.198 | 16384 | InvasionPercolation | 23400.906 | 189.726 | 16384 | InvasionPercolation | 12535.422 | 135.108 | 32768 | InvasionPercolation | 17451.488 | 182.249 | 16384 |

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| InvasionPercolation | 15210.797 | 176.75 | 32768 | InvasionPercolation | 23166.407 | 104.484 | 16384 | InvasionPercolation | 12515.507 | 96.858 | 32768 | InvasionPercolation | 17521.312 | 158.248 | 16384 |
| InvasionPercolation | 15131.755 | 36.055 | 32768 | InvasionPercolation | 23246.636 | 203.058 | 16384 | InvasionPercolation | 12395.677 | 34.215 | 32768 | InvasionPercolation | 17462.097 | 185.593 | 16384 |
| InvasionPercolation | 15114.775 | 34.942 | 32768 | InvasionPercolation | 23685.072 | 797.292 | 16384 | InvasionPercolation | 12464.341 | 109.683 | 32768 | InvasionPercolation | 17605.703 | 394.963 | 16384 |
| InvasionPercolation | 15205.361 | 89.946 | 32768 | InvasionPercolation | 23215.316 | 207.034 | 16384 | InvasionPercolation | 12497.908 | 110.565 | 32768 | InvasionPercolation | 17305.389 | 99.563 | 16384 |
| FibonacciRecursive | 2768.876 | 11.166 | 131072 | FibonacciRecursive | 529.299 | 3.172 | 524288 | FibonacciRecursive | 120.652 | 1.209 | 2097152 | FibonacciRecursive | 93.491 | 0.939 | 4194304 |
| FibonacciRecursive | 2798.858 | 55.522 | 131072 | FibonacciRecursive | 527.683 | 5.348 | 524288 | FibonacciRecursive | 119.845 | 1.404 | 2097152 | FibonacciRecursive | 91.438 | 0.81 | 4194304 |
| FibonacciRecursive | 2799.972 | 11.209 | 131072 | FibonacciRecursive | 527.009 | 2.112 | 524288 | FibonacciRecursive | 120.292 | 1.838 | 2097152 | FibonacciRecursive | 90.57 | 0.493 | 4194304 |
| FibonacciRecursive | 2786.775 | 25.538 | 131072 | FibonacciRecursive | 525.575 | 1.469 | 524288 | FibonacciRecursive | 118.887 | 1.487 | 4194304 | FibonacciRecursive | 91.269 | 0.856 | 4194304 |
| FibonacciRecursive | 2785.763 | 47.991 | 131072 | FibonacciRecursive | 524.536 | 1.538 | 524288 | FibonacciRecursive | 119.404 | 1.138 | 4194304 | FibonacciRecursive | 91.771 | 1.675 | 4194304 |
| FibonacciIterative | 614.935 | 1.662 | 524288 | FibonacciIterative | 692.11 | 3.065 | 524288 | FibonacciIterative | 90.881 | 0.437 | 4194304 | FibonacciIterative | 95.982 | 0.11 | 4194304 |
| FibonacciIterative | 615.412 | 4.615 | 524288 | FibonacciIterative | 694.08 | 4.78 | 524288 | FibonacciIterative | 91.11 | 0.457 | 4194304 | FibonacciIterative | 99.511 | 0.812 | 4194304 |
| FibonacciIterative | 617.042 | 5.201 | 524288 | FibonacciIterative | 695.181 | 4.406 | 524288 | FibonacciIterative | 90.576 | 0.59 | 4194304 | FibonacciIterative | 98.969 | 0.872 | 4194304 |
| FibonacciIterative | 616.262 | 3.858 | 524288 | FibonacciIterative | 691.554 | 1.987 | 524288 | FibonacciIterative | 91.074 | 0.677 | 4194304 | FibonacciIterative | 99.675 | 1.255 | 4194304 |
| FibonacciIterative | 615.43 | 1.573 | 524288 | FibonacciIterative | 692.185 | 1.147 | 524288 | FibonacciIterative | 92.241 | 3.157 | 4194304 | FibonacciIterative | 98.773 | 0.382 | 4194304 |
| ScaleVector2D | 36.754 | 0.866 | 8388608 | ScaleVector2D | 44.562 | 0.779 | 8388608 | ScaleVector2D | 6.955 | 0.024 | 67108864 | ScaleVector2D | 8.214 | 0.146 | 33554432 |
| ScaleVector2D | 36.415 | 0.1 | 8388608 | ScaleVector2D | 43.977 | 0.177 | 8388608 | ScaleVector2D | 7.031 | 0.061 | 67108864 | ScaleVector2D | 7.949 | 0.06 | 33554432 |
| ScaleVector2D | 36.209 | 0.265 | 8388608 | ScaleVector2D | 43.904 | 0.185 | 8388608 | ScaleVector2D | 7.033 | 0.038 | 67108864 | ScaleVector2D | 8.121 | 0.176 | 33554432 |
| ScaleVector2D | 36.582 | 0.338 | 8388608 | ScaleVector2D | 43.994 | 0.15 | 8388608 | ScaleVector2D | 7.106 | 0.252 | 67108864 | ScaleVector2D | 7.938 | 0.067 | 33554432 |
| ScaleVector2D | 36.557 | 0.436 | 8388608 | ScaleVector2D | 43.984 | 0.238 | 8388608 | ScaleVector2D | 6.999 | 0.054 | 67108864 | ScaleVector2D | 7.935 | 0.066 | 33554432 |
| ScaleVector3D | 41.659 | 0.155 | 8388608 | ScaleVector3D | 51.633 | 0.361 | 8388608 | ScaleVector3D | 4.18 | 0.028 | 67108864 | ScaleVector3D | 4.94 | 0.07 | 67108864 |
| ScaleVector3D | 41.774 | 0.146 | 8388608 | ScaleVector3D | 51.647 | 0.215 | 8388608 | ScaleVector3D | 4.169 | 0.043 | 67108864 | ScaleVector3D | 4.534 | 0.049 | 67108864 |
| ScaleVector3D | 41.85 | 0.525 | 8388608 | ScaleVector3D | 64.081 | 6.845 | 4194304 | ScaleVector3D | 4.158 | 0.036 | 67108864 | ScaleVector3D | 4.583 | 0.068 | 67108864 |
| ScaleVector3D | 41.862 | 0.291 | 8388608 | ScaleVector3D | 51.779 | 0.391 | 8388608 | ScaleVector3D | 4.149 | 0.043 | 67108864 | ScaleVector3D | 4.582 | 0.048 | 67108864 |
| ScaleVector3D | 41.704 | 0.175 | 8388608 | ScaleVector3D | 51.535 | 0.174 | 8388608 | ScaleVector3D | 4.186 | 0.053 | 67108864 | ScaleVector3D | 4.573 | 0.04 | 67108864 |
| MultiplyVector2D | 20.431 | 0.106 | 16777216 | MultiplyVector2D | 30.386 | 0.172 | 8388608 | MultiplyVector2D | 6.228 | 0.015 | 67108864 | MultiplyVector2D | 6.72 | 0.255 | 67108864 |
| MultiplyVector2D | 20.605 | 0.248 | 16777216 | MultiplyVector2D | 30.625 | 0.648 | 8388608 | MultiplyVector2D | 6.318 | 0.058 | 67108864 | MultiplyVector2D | 6.334 | 0.025 | 67108864 |
| MultiplyVector2D | 20.492 | 0.044 | 16777216 | MultiplyVector2D | 30.397 | 0.182 | 8388608 | MultiplyVector2D | 6.264 | 0.055 | 67108864 | MultiplyVector2D | 6.29 | 0.021 | 67108864 |
| MultiplyVector2D | 20.498 | 0.036 | 16777216 | MultiplyVector2D | 30.457 | 0.214 | 8388608 | MultiplyVector2D | 6.251 | 0.042 | 67108864 | MultiplyVector2D | 6.309 | 0.029 | 67108864 |
| MultiplyVector2D | 20.585 | 0.125 | 16777216 | MultiplyVector2D | 30.445 | 0.205 | 8388608 | MultiplyVector2D | 6.248 | 0.071 | 67108864 | MultiplyVector2D | 6.39 | 0.034 | 67108864 |
| MultiplyVector3D | 37.005 | 0.474 | 8388608 | MultiplyVector3D | 45.952 | 0.302 | 8388608 | MultiplyVector3D | 4.18 | 0.117 | 67108864 | MultiplyVector3D | 5.258 | 0.03 | 67108864 |
| MultiplyVector3D | 36.785 | 0.296 | 8388608 | MultiplyVector3D | 45.83 | 0.166 | 8388608 | MultiplyVector3D | 4.173 | 0.034 | 67108864 | MultiplyVector3D | 4.997 | 0.196 | 67108864 |
| MultiplyVector3D | 38.374 | 3.123 | 8388608 | MultiplyVector3D | 45.836 | 0.14 | 8388608 | MultiplyVector3D | 4.21 | 0.046 | 67108864 | MultiplyVector3D | 4.875 | 0.011 | 67108864 |
| MultiplyVector3D | 36.987 | 0.334 | 8388608 | MultiplyVector3D | 45.988 | 0.227 | 8388608 | MultiplyVector3D | 4.191 | 0.035 | 67108864 | MultiplyVector3D | 4.935 | 0.037 | 67108864 |
| MultiplyVector3D | 36.719 | 0.092 | 8388608 | MultiplyVector3D | 46.004 | 0.58 | 8388608 | MultiplyVector3D | 4.157 | 0.023 | 67108864 | MultiplyVector3D | 4.985 | 0.088 | 67108864 |
| TranslateVector2D | 20.088 | 0.057 | 16777216 | TranslateVector2D | 30.427 | 0.203 | 8388608 | TranslateVector2D | 6.211 | 0.03 | 67108864 | TranslateVector2D | 7.927 | 0.079 | 33554432 |
| TranslateVector2D | 20.113 | 0.042 | 16777216 | TranslateVector2D | 30.414 | 0.177 | 8388608 | TranslateVector2D | 6.179 | 0.035 | 67108864 | TranslateVector2D | 7.629 | 0.147 | 33554432 |
| TranslateVector2D | 20.206 | 0.123 | 16777216 | TranslateVector2D | 30.089 | 0.092 | 8388608 | TranslateVector2D | 6.216 | 0.089 | 67108864 | TranslateVector2D | 7.564 | 0.07 | 33554432 |
| TranslateVector2D | 20.149 | 0.076 | 16777216 | TranslateVector2D | 30.494 | 0.27 | 8388608 | TranslateVector2D | 6.16 | 0.037 | 67108864 | TranslateVector2D | 7.603 | 0.119 | 33554432 |

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TranslateVector2D | 20.162 | 0.13 | 16777216 | TranslateVector2D | 30.122 | 0.165 | 8388608 | TranslateVector2D | 6.191 | 0.052 | 67108864 | TranslateVector2D | 7.547 | 0.04 | 33554432 |
| TranslateVector3D | 35.977 | 0.081 | 8388608 | TranslateVector3D | 46.573 | 0.456 | 8388608 | TranslateVector3D | 4.147 | 0.031 | 67108864 | TranslateVector3D | 4.889 | 0.011 | 67108864 |
| TranslateVector3D | 35.988 | 0.075 | 8388608 | TranslateVector3D | 46.414 | 0.29 | 8388608 | TranslateVector3D | 4.171 | 0.009 | 67108864 | TranslateVector3D | 4.509 | 0.016 | 67108864 |
| TranslateVector3D | 36.229 | 0.331 | 8388608 | TranslateVector3D | 46.396 | 0.237 | 8388608 | TranslateVector3D | 4.128 | 0.017 | 67108864 | TranslateVector3D | 4.54 | 0.067 | 67108864 |
| TranslateVector3D | 36.561 | 0.822 | 8388608 | TranslateVector3D | 46.23 | 0.075 | 8388608 | TranslateVector3D | 4.154 | 0.028 | 67108864 | TranslateVector3D | 4.551 | 0.058 | 67108864 |
| TranslateVector3D | 36.369 | 0.656 | 8388608 | TranslateVector3D | 46.5 | 0.274 | 8388608 | TranslateVector3D | 4.213 | 0.048 | 67108864 | TranslateVector3D | 4.523 | 0.06 | 67108864 |
| SubtractVector2D | 20.174 | 0.055 | 16777216 | SubtractVector2D | 30.233 | 0.281 | 16777216 | SubtractVector2D | 6.17 | 0.055 | 67108864 | SubtractVector2D | 6.303 | 0.04 | 67108864 |
| SubtractVector2D | 20.425 | 0.255 | 16777216 | SubtractVector2D | 29.904 | 0.19 | 8388608 | SubtractVector2D | 6.164 | 0.033 | 67108864 | SubtractVector2D | 6.178 | 0.127 | 67108864 |
| SubtractVector2D | 20.285 | 0.189 | 16777216 | SubtractVector2D | 30.056 | 0.319 | 16777216 | SubtractVector2D | 6.275 | 0.078 | 67108864 | SubtractVector2D | 6.159 | 0.055 | 67108864 |
| SubtractVector2D | 20.289 | 0.165 | 16777216 | SubtractVector2D | 30.197 | 0.275 | 8388608 | SubtractVector2D | 6.184 | 0.079 | 67108864 | SubtractVector2D | 6.115 | 0.033 | 67108864 |
| SubtractVector2D | 20.106 | 0.079 | 16777216 | SubtractVector2D | 29.999 | 0.18 | 8388608 | SubtractVector2D | 6.199 | 0.058 | 67108864 | SubtractVector2D | 6.103 | 0.033 | 67108864 |
| SubtractVector3D | 20.078 | 0.051 | 16777216 | SubtractVector3D | 45.07 | 0.108 | 8388608 | SubtractVector3D | 6.179 | 0.034 | 67108864 | SubtractVector3D | 5.282 | 0.048 | 67108864 |
| SubtractVector3D | 20.115 | 0.179 | 16777216 | SubtractVector3D | 45.045 | 0.122 | 8388608 | SubtractVector3D | 6.155 | 0.021 | 67108864 | SubtractVector3D | 4.913 | 0.053 | 67108864 |
| SubtractVector3D | 20.282 | 0.448 | 16777216 | SubtractVector3D | 45.156 | 0.222 | 8388608 | SubtractVector3D | 6.204 | 0.074 | 67108864 | SubtractVector3D | 4.896 | 0.033 | 67108864 |
| SubtractVector3D | 20.177 | 0.231 | 16777216 | SubtractVector3D | 45.233 | 0.329 | 8388608 | SubtractVector3D | 6.194 | 0.131 | 67108864 | SubtractVector3D | 4.901 | 0.048 | 67108864 |
| SubtractVector3D | 20.155 | 0.107 | 16777216 | SubtractVector3D | 45.064 | 0.209 | 8388608 | SubtractVector3D | 6.178 | 0.065 | 67108864 | SubtractVector3D | 4.936 | 0.077 | 67108864 |
| LengthVector2D | 18.669 | 0.121 | 16777216 | LengthVector2D | 24.265 | 0.12 | 16777216 | LengthVector2D | 4.677 | 0.033 | 67108864 | LengthVector2D | 5.113 | 0.091 | 67108864 |
| LengthVector2D | 18.607 | 0.057 | 16777216 | LengthVector2D | 24.214 | 0.109 | 16777216 | LengthVector2D | 4.708 | 0.02 | 67108864 | LengthVector2D | 5.022 | 0.06 | 67108864 |
| LengthVector2D | 18.77 | 0.212 | 16777216 | LengthVector2D | 24.232 | 0.301 | 16777216 | LengthVector2D | 4.693 | 0.007 | 67108864 | LengthVector2D | 5.021 | 0.04 | 67108864 |
| LengthVector2D | 18.698 | 0.119 | 16777216 | LengthVector2D | 24.358 | 0.294 | 16777216 | LengthVector2D | 4.688 | 0.04 | 67108864 | LengthVector2D | 5.032 | 0.047 | 67108864 |
| LengthVector2D | 18.72 | 0.23 | 16777216 | LengthVector2D | 24.208 | 0.198 | 16777216 | LengthVector2D | 4.699 | 0.028 | 67108864 | LengthVector2D | 5.097 | 0.132 | 67108864 |
| LengthVector3D | 19.394 | 0.295 | 16777216 | LengthVector3D | 24.973 | 0.115 | 16777216 | LengthVector3D | 4.173 | 0.034 | 67108864 | LengthVector3D | 3.807 | 0.116 | 67108864 |
| LengthVector3D | 19.386 | 0.571 | 16777216 | LengthVector3D | 24.912 | 0.119 | 16777216 | LengthVector3D | 4.174 | 0.046 | 67108864 | LengthVector3D | 3.758 | 0.017 | 67108864 |
| LengthVector3D | 19.346 | 0.075 | 16777216 | LengthVector3D | 24.961 | 0.156 | 16777216 | LengthVector3D | 4.155 | 0.037 | 67108864 | LengthVector3D | 3.77 | 0.029 | 67108864 |
| LengthVector3D | 19.321 | 0.128 | 16777216 | LengthVector3D | 24.918 | 0.107 | 16777216 | LengthVector3D | 4.14 | 0.023 | 67108864 | LengthVector3D | 3.75 | 0.017 | 67108864 |
| LengthVector3D | 19.415 | 0.314 | 16777216 | LengthVector3D | 24.897 | 0.099 | 16777216 | LengthVector3D | 4.142 | 0.013 | 67108864 | LengthVector3D | 3.834 | 0.059 | 67108864 |
| DotProductVector2D | 16.475 | 0.303 | 16777216 | DotProductVector2D | 23.738 | 0.241 | 16777216 | DotProductVector2D | 4.516 | 0.028 | 67108864 | DotProductVector2D | 4.164 | 0.051 | 67108864 |
| DotProductVector2D | 16.412 | 0.186 | 16777216 | DotProductVector2D | 23.543 | 0.153 | 16777216 | DotProductVector2D | 4.586 | 0.077 | 67108864 | DotProductVector2D | 4.557 | 0.047 | 67108864 |
| DotProductVector2D | 16.352 | 0.058 | 16777216 | DotProductVector2D | 23.569 | 0.302 | 16777216 | DotProductVector2D | 4.526 | 0.023 | 67108864 | DotProductVector2D | 4.158 | 0.033 | 67108864 |
| DotProductVector2D | 16.34 | 0.124 | 16777216 | DotProductVector2D | 23.448 | 0.136 | 16777216 | DotProductVector2D | 4.54 | 0.036 | 67108864 | DotProductVector2D | 4.523 | 0.064 | 67108864 |
| DotProductVector2D | 16.362 | 0.062 | 16777216 | DotProductVector2D | 23.191 | 0.285 | 16777216 | DotProductVector2D | 4.582 | 0.065 | 67108864 | DotProductVector2D | 4.139 | 0.026 | 67108864 |
| DotProductVector3D | 16.895 | 0.1 | 16777216 | DotProductVector3D | 34.327 | 0.136 | 8388608 | DotProductVector3D | 4.185 | 0.063 | 67108864 | DotProductVector3D | 3.763 | 0.025 | 67108864 |
| DotProductVector3D | 16.913 | 0.076 | 16777216 | DotProductVector3D | 35.06 | 0.697 | 8388608 | DotProductVector3D | 4.168 | 0.054 | 67108864 | DotProductVector3D | 3.771 | 0.036 | 67108864 |
| DotProductVector3D | 16.879 | 0.082 | 16777216 | DotProductVector3D | 35.031 | 0.536 | 8388608 | DotProductVector3D | 4.144 | 0.033 | 67108864 | DotProductVector3D | 3.757 | 0.025 | 67108864 |
| DotProductVector3D | 18.937 | 2.09 | 16777216 | DotProductVector3D | 34.419 | 0.174 | 8388608 | DotProductVector3D | 4.254 | 0.11 | 67108864 | DotProductVector3D | 3.801 | 0.071 | 67108864 |
| DotProductVector3D | 16.964 | 0.081 | 16777216 | DotProductVector3D | 34.285 | 0.105 | 8388608 | DotProductVector3D | 4.153 | 0.032 | 67108864 | DotProductVector3D | 3.784 | 0.036 | 67108864 |

| C# editor unity | | | | F# editor unity | | | | C# release unity | | | | F# release unity | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Test | Mean | Deviation | Count | Test | Mean | Deviation | Count | Test | Mean | Deviation | Count | Test | Mean | Deviation | Count |
| MapReduce Foreach | 3164.587 | 15.363 | 131072 | MapReduce Array | 3811.875 | 230.127 | 131072 | MapReduce Foreach | 1018.475 | 6.841 | 262144 | MapReduce Array | 1298.441 | 19.357 | 262144 |
| MapReduce Foreach | 3193.387 | 64.181 | 131072 | MapReduce Array | 3636.335 | 40.848 | 131072 | MapReduce Foreach | 1011.843 | 5.119 | 262144 | MapReduce Array | 1378.75 | 102.925 | 262144 |
| MapReduce Foreach | 3351.121 | 420.595 | 131072 | MapReduce Array | 3661.75 | 43.04 | 131072 | MapReduce Foreach | 1018.703 | 11.604 | 262144 | MapReduce Array | 1295.649 | 12.909 | 262144 |
| MapReduce Foreach | 3165.949 | 15.65 | 131072 | MapReduce Array | 3943.603 | 420.565 | 65536 | MapReduce Foreach | 1030.809 | 28.152 | 262144 | MapReduce Array | 1288.887 | 14.162 | 262144 |
| MapReduce Foreach | 3167.574 | 19.044 | 131072 | MapReduce Array | 3701.37 | 47.298 | 131072 | MapReduce Foreach | 1021.479 | 8.234 | 262144 | MapReduce Array | 1294.176 | 17.19 | 262144 |
| MapReduce Linq | 4120.558 | 95.981 | 65536 | MapReduce Seq | 4873.857 | 78.192 | 65536 | MapReduce Linq | 1698.881 | 29.945 | 262144 | MapReduce Seq | 2228.795 | 93.004 | 131072 |
| MapReduce Linq | 4114.522 | 92.308 | 65536 | MapReduce Seq | 4858.675 | 94.379 | 65536 | MapReduce Linq | 1710.208 | 39.441 | 262144 | MapReduce Seq | 2277.074 | 64.762 | 131072 |
| MapReduce Linq | 4120.561 | 54.382 | 65536 | MapReduce Seq | 4852.566 | 80.279 | 65536 | MapReduce Linq | 1691.499 | 42.582 | 262144 | MapReduce Seq | 2320.62 | 68.792 | 131072 |
| MapReduce Linq | 4087.197 | 79.096 | 65536 | MapReduce Seq | 4822.636 | 88.193 | 65536 | MapReduce Linq | 1731.838 | 107.427 | 262144 | MapReduce Seq | 2286.958 | 42.783 | 131072 |
| MapReduce Linq | 4059.504 | 64.34 | 65536 | MapReduce Seq | 4864.76 | 114.624 | 65536 | MapReduce Linq | 1717.52 | 22.71 | 262144 | MapReduce Seq | 2259.42 | 55.52 | 131072 |
| MapReduce Struct | 325.775 | 2.066 | 1048576 | MapReduce Unions | 487.822 | 3.452 | 1048576 | MapReduce Struct | 171.425 | 1.832 | 2097152 | MapReduce Unions | 316.746 | 4.672 | 1048576 |
| MapReduce Struct | 331.116 | 18.997 | 1048576 | MapReduce Unions | 489.137 | 10.842 | 524288 | MapReduce Struct | 171.935 | 1.125 | 2097152 | MapReduce Unions | 318.839 | 2.374 | 1048576 |
| MapReduce Struct | 323.903 | 0.793 | 1048576 | MapReduce Unions | 544.872 | 68.631 | 524288 | MapReduce Struct | 172.826 | 1.312 | 2097152 | MapReduce Unions | 322.757 | 2.037 | 1048576 |
| MapReduce Struct | 323.678 | 1.439 | 1048576 | MapReduce Unions | 500.583 | 18.614 | 524288 | MapReduce Struct | 171.752 | 0.865 | 2097152 | MapReduce Unions | 316.76 | 5.884 | 1048576 |
| MapReduce Struct | 323.873 | 1.254 | 1048576 | MapReduce Unions | 487.255 | 5.887 | 1048576 | MapReduce Struct | 171.851 | 0.254 | 2097152 | MapReduce Unions | 318.087 | 2.927 | 1048576 |
| Sestoft Multiply | 55.425 | 0.321 | 8388608 | Sestoft Multiply | 62.619 | 0.515 | 4194304 | Sestoft Multiply | 48.464 | 0.275 | 8388608 | Sestoft Multiply | 53.065 | 0.231 | 8388608 |
| Sestoft Multiply | 55.429 | 0.265 | 8388608 | Sestoft Multiply | 62.968 | 0.77 | 4194304 | Sestoft Multiply | 50.564 | 3.205 | 8388608 | Sestoft Multiply | 52.946 | 0.065 | 8388608 |
| Sestoft Multiply | 55.64 | 0.42 | 8388608 | Sestoft Multiply | 62.8 | 0.402 | 4194304 | Sestoft Multiply | 48.538 | 0.322 | 8388608 | Sestoft Multiply | 53.446 | 0.332 | 8388608 |
| Sestoft Multiply | 55.689 | 0.559 | 8388608 | Sestoft Multiply | 62.657 | 0.531 | 4194304 | Sestoft Multiply | 48.3 | 0.451 | 8388608 | Sestoft Multiply | 54.352 | 2.985 | 8388608 |
| Sestoft Multiply | 55.829 | 0.434 | 8388608 | Sestoft Multiply | 62.431 | 0.556 | 4194304 | Sestoft Multiply | 48.252 | 0.169 | 8388608 | Sestoft Multiply | 53.03 | 0.276 | 8388608 |
| Primes | 10132.123 | 131.699 | 32768 | Primes | 20324.99 | 569.503 | 16384 | Primes | 3684.169 | 104.678 | 131072 | Primes | 8452.111 | 238.768 | 32768 |
| Primes | 10101.887 | 104.354 | 32768 | Primes | 20081.245 | 354.162 | 16384 | Primes | 3742.526 | 106.047 | 131072 | Primes | 8883.235 | 89.874 | 32768 |
| Primes | 10150.407 | 121.193 | 32768 | Primes | 22216.256 | 2753.199 | 16384 | Primes | 3742.986 | 93.652 | 65536 | Primes | 8803.849 | 167.795 | 32768 |
| Primes | 10132.115 | 110.725 | 32768 | Primes | 20130.051 | 288.94 | 16384 | Primes | 3895.292 | 326.877 | 131072 | Primes | 8413.988 | 361.921 | 32768 |
| Primes | 10095.747 | 93.919 | 32768 | Primes | 20141.995 | 504.021 | 16384 | Primes | 3755.088 | 170.541 | 131072 | Primes | 8829.361 | 140.399 | 32768 |
| RandomizeArray | 6575.88 | 58.54 | 65536 | RandomizeArray | 7097.305 | 99.746 | 65536 | RandomizeArray | 3201.197 | 90.282 | 131072 | RandomizeArray | 4153.095 | 179.54 | 65536 |
| RandomizeArray | 6591.145 | 44.984 | 65536 | RandomizeArray | 7060.705 | 113.015 | 65536 | RandomizeArray | 3348.754 | 27.72 | 131072 | RandomizeArray | 4244.465 | 177.662 | 65536 |
| RandomizeArray | 6966.116 | 594.971 | 32768 | RandomizeArray | 7036.956 | 34.983 | 65536 | RandomizeArray | 3275.138 | 97.445 | 131072 | RandomizeArray | 4349.842 | 133.79 | 65536 |
| RandomizeArray | 6618.575 | 52.626 | 65536 | RandomizeArray | 7109.857 | 83.25 | 65536 | RandomizeArray | 3324.304 | 32.371 | 131072 | RandomizeArray | 4284.013 | 194.005 | 65536 |
| RandomizeArray | 6615.596 | 51.122 | 65536 | RandomizeArray | 7469.491 | 542.426 | 65536 | RandomizeArray | 3332.205 | 53.62 | 131072 | RandomizeArray | 3253.89 | 42.284 | 131072 |
| GameOfLife | 222240.146 | 1843.336 | 2048 | GameOfLife | 316419.844 | 5798.59 | 1024 | GameOfLife | 126828.97 | 10026.121 | 4096 | GameOfLife | 202029.541 | 3874.481 | 2048 |
| GameOfLife | 223319.629 | 4174.32 | 2048 | GameOfLife | 318766.172 | 3818.383 | 1024 | GameOfLife | 121061.392 | 618.655 | 4096 | GameOfLife | 202328.018 | 4789.254 | 2048 |
| GameOfLife | 223711.621 | 2596.205 | 2048 | GameOfLife | 318571.289 | 5135.903 | 1024 | GameOfLife | 121759.248 | 820.012 | 2048 | GameOfLife | 207980.488 | 6170.934 | 2048 |
| GameOfLife | 224389.98 | 7229.533 | 2048 | GameOfLife | 317786.406 | 4383.526 | 1024 | GameOfLife | 122630.596 | 936.892 | 2048 | GameOfLife | 207976.807 | 6626.907 | 2048 |
| GameOfLife | 235318.037 | 18242.984 | 2048 | GameOfLife | 319738.32 | 5144.06 | 1024 | GameOfLife | 122401.445 | 559.289 | 2048 | GameOfLife | 205252.686 | 8132.406 | 2048 |
| InvasionPercolation | 86571.396 | 1111.207 | 4096 | InvasionPercolation | 75396.167 | 1550.079 | 4096 | InvasionPercolation | 57051.06 | 531.019 | 8192 | InvasionPercolation | 53659.233 | 171.562 | 8192 |

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| InvasionPercolation | 88230.62 | 1970.792 | 4096 | InvasionPercolation | 76275.967 | 1265.009 | 4096 | InvasionPercolation | 56995.862 | 705.31 | 8192 | InvasionPercolation | 55098.845 | 1354.991 | 8192 |
| InvasionPercolation | 87839.595 | 1257.445 | 4096 | InvasionPercolation | 76517.568 | 1235.348 | 4096 | InvasionPercolation | 56945.4 | 714.778 | 8192 | InvasionPercolation | 54081.78 | 1134.522 | 8192 |
| InvasionPercolation | 88816.143 | 790.545 | 4096 | InvasionPercolation | 76128.608 | 1392.821 | 4096 | InvasionPercolation | 61647.793 | 5810.038 | 4096 | InvasionPercolation | 53786.57 | 376.806 | 8192 |
| InvasionPercolation | 87501.25 | 617.089 | 4096 | InvasionPercolation | 75628.057 | 1426.16 | 4096 | InvasionPercolation | 56659.26 | 917.26 | 8192 | InvasionPercolation | 53918.176 | 495.915 | 8192 |
| FibonacciRecursive | 2394.282 | 22.328 | 131072 | FibonacciRecursive | 739.018 | 11.388 | 524288 | FibonacciRecursive | 665.275 | 3.29 | 524288 | FibonacciRecursive | 159.921 | 0.751 | 2097152 |
| FibonacciRecursive | 2382.123 | 4.27 | 131072 | FibonacciRecursive | 737.38 | 10.007 | 524288 | FibonacciRecursive | 664.132 | 2.193 | 524288 | FibonacciRecursive | 159.547 | 0.268 | 2097152 |
| FibonacciRecursive | 2397.332 | 23.216 | 131072 | FibonacciRecursive | 740.196 | 9.623 | 524288 | FibonacciRecursive | 666.046 | 5.157 | 524288 | FibonacciRecursive | 160.214 | 1.395 | 2097152 |
| FibonacciRecursive | 2549.806 | 249.258 | 131072 | FibonacciRecursive | 745.104 | 10.814 | 524288 | FibonacciRecursive | 669.477 | 4.935 | 524288 | FibonacciRecursive | 166.486 | 13.792 | 2097152 |
| FibonacciRecursive | 2397.312 | 11.575 | 131072 | FibonacciRecursive | 735.433 | 14.049 | 524288 | FibonacciRecursive | 664.513 | 3.724 | 524288 | FibonacciRecursive | 158.896 | 0.731 | 2097152 |
| FibonacciIterative | 994.318 | 3.173 | 262144 | FibonacciIterative | 564.586 | 4.47 | 524288 | FibonacciIterative | 88.346 | 0.546 | 4194304 | FibonacciIterative | 134.874 | 0.707 | 2097152 |
| FibonacciIterative | 994.304 | 1.707 | 262144 | FibonacciIterative | 564.356 | 4.618 | 524288 | FibonacciIterative | 88.594 | 0.686 | 4194304 | FibonacciIterative | 133.604 | 1.079 | 2097152 |
| FibonacciIterative | 1001.149 | 3.38 | 262144 | FibonacciIterative | 560.857 | 1.625 | 524288 | FibonacciIterative | 87.832 | 0.362 | 4194304 | FibonacciIterative | 134.08 | 0.785 | 2097152 |
| FibonacciIterative | 1012.602 | 23.487 | 262144 | FibonacciIterative | 562.177 | 4.922 | 524288 | FibonacciIterative | 87.441 | 0.726 | 4194304 | FibonacciIterative | 134.082 | 0.916 | 2097152 |
| FibonacciIterative | 999.562 | 8.774 | 262144 | FibonacciIterative | 566.452 | 4.389 | 524288 | FibonacciIterative | 87.864 | 0.392 | 4194304 | FibonacciIterative | 133.515 | 0.691 | 2097152 |
| ScaleVector2D | 100.554 | 0.477 | 4194304 | ScaleVector2D | 111.826 | 0.778 | 4194304 | ScaleVector2D | 72.377 | 0.611 | 4194304 | ScaleVector2D | 78.82 | 0.324 | 4194304 |
| ScaleVector2D | 101.222 | 1.611 | 4194304 | ScaleVector2D | 111.305 | 0.46 | 4194304 | ScaleVector2D | 72.509 | 0.425 | 4194304 | ScaleVector2D | 79.248 | 0.592 | 4194304 |
| ScaleVector2D | 102.127 | 4.736 | 4194304 | ScaleVector2D | 111.234 | 0.319 | 4194304 | ScaleVector2D | 72.449 | 0.439 | 4194304 | ScaleVector2D | 78.62 | 0.452 | 4194304 |
| ScaleVector2D | 100.269 | 0.595 | 4194304 | ScaleVector2D | 111.297 | 0.499 | 4194304 | ScaleVector2D | 72.342 | 0.463 | 4194304 | ScaleVector2D | 83.525 | 7.301 | 4194304 |
| ScaleVector2D | 99.888 | 0.107 | 4194304 | ScaleVector2D | 111.189 | 0.366 | 4194304 | ScaleVector2D | 72.103 | 0.729 | 4194304 | ScaleVector2D | 78.624 | 0.523 | 4194304 |
| ScaleVector3D | 97.363 | 0.869 | 4194304 | ScaleVector3D | 108.895 | 0.236 | 4194304 | ScaleVector3D | 68.578 | 0.309 | 4194304 | ScaleVector3D | 77.059 | 0.358 | 4194304 |
| ScaleVector3D | 96.696 | 0.105 | 4194304 | ScaleVector3D | 110.224 | 0.931 | 4194304 | ScaleVector3D | 68.863 | 0.843 | 4194304 | ScaleVector3D | 77.153 | 0.446 | 4194304 |
| ScaleVector3D | 96.935 | 0.356 | 4194304 | ScaleVector3D | 109.808 | 1.111 | 4194304 | ScaleVector3D | 68.724 | 0.273 | 4194304 | ScaleVector3D | 77.526 | 0.62 | 4194304 |
| ScaleVector3D | 97.456 | 0.787 | 4194304 | ScaleVector3D | 109.704 | 0.86 | 4194304 | ScaleVector3D | 68.864 | 0.556 | 4194304 | ScaleVector3D | 77.677 | 0.542 | 4194304 |
| ScaleVector3D | 97.028 | 0.422 | 4194304 | ScaleVector3D | 109.576 | 0.358 | 4194304 | ScaleVector3D | 68.742 | 0.644 | 4194304 | ScaleVector3D | 78.056 | 0.96 | 4194304 |
| MultiplyVector2D | 84.877 | 1.243 | 4194304 | MultiplyVector2D | 97.394 | 0.107 | 4194304 | MultiplyVector2D | 61.167 | 0.192 | 4194304 | MultiplyVector2D | 66.29 | 0.197 | 4194304 |
| MultiplyVector2D | 84.304 | 0.199 | 4194304 | MultiplyVector2D | 98.791 | 1.127 | 4194304 | MultiplyVector2D | 61.385 | 0.524 | 4194304 | MultiplyVector2D | 66.291 | 0.262 | 4194304 |
| MultiplyVector2D | 84.59 | 0.679 | 4194304 | MultiplyVector2D | 97.74 | 0.413 | 4194304 | MultiplyVector2D | 61.478 | 0.532 | 4194304 | MultiplyVector2D | 71.389 | 6.626 | 4194304 |
| MultiplyVector2D | 85.212 | 0.721 | 4194304 | MultiplyVector2D | 97.963 | 0.729 | 4194304 | MultiplyVector2D | 61.286 | 0.43 | 4194304 | MultiplyVector2D | 66.005 | 0.168 | 4194304 |
| MultiplyVector3D | 88.164 | 0.505 | 4194304 | MultiplyVector3D | 104.129 | 0.732 | 4194304 | MultiplyVector3D | 63.24 | 0.362 | 4194304 | MultiplyVector3D | 71.389 | 0.212 | 4194304 |
| MultiplyVector3D | 88.934 | 2.548 | 4194304 | MultiplyVector3D | 103.338 | 0.64 | 4194304 | MultiplyVector3D | 63.584 | 0.519 | 4194304 | MultiplyVector3D | 71.58 | 0.213 | 4194304 |
| MultiplyVector3D | 87.544 | 0.622 | 4194304 | MultiplyVector3D | 103.582 | 0.501 | 4194304 | MultiplyVector3D | 68.628 | 8.353 | 4194304 | MultiplyVector3D | 71.675 | 0.832 | 4194304 |
| MultiplyVector3D | 92.978 | 8.79 | 4194304 | MultiplyVector3D | 103.478 | 0.581 | 4194304 | MultiplyVector3D | 63.334 | 0.199 | 4194304 | MultiplyVector3D | 71.39 | 0.129 | 4194304 |
| MultiplyVector3D | 87.783 | 0.817 | 4194304 | MultiplyVector3D | 103.808 | 0.859 | 4194304 | MultiplyVector3D | 62.954 | 0.2 | 4194304 | MultiplyVector3D | 71.532 | 0.262 | 4194304 |
| TranslateVector2D | 109.037 | 0.934 | 4194304 | TranslateVector2D | 100.363 | 0.904 | 4194304 | TranslateVector2D | 76.822 | 0.398 | 4194304 | TranslateVector2D | 53.096 | 0.416 | 8388608 |
| TranslateVector2D | 108.465 | 0.815 | 4194304 | TranslateVector2D | 100.783 | 0.57 | 4194304 | TranslateVector2D | 77.299 | 0.689 | 4194304 | TranslateVector2D | 52.787 | 0.159 | 8388608 |
| TranslateVector2D | 108.505 | 0.391 | 4194304 | TranslateVector2D | 99.571 | 0.226 | 4194304 | TranslateVector2D | 77.362 | 0.71 | 4194304 | TranslateVector2D | 55.591 | 4.608 | 8388608 |
| TranslateVector2D | 108.751 | 0.687 | 4194304 | TranslateVector2D | 100.233 | 1.102 | 4194304 | TranslateVector2D | 77.026 | 0.274 | 4194304 | TranslateVector2D | 52.708 | 0.179 | 8388608 |
| TranslateVector2D | 112.135 | 6.999 | 4194304 | TranslateVector2D | 99.558 | 0.352 | 4194304 | TranslateVector2D | 77.316 | 0.505 | 4194304 | TranslateVector2D | 53.072 | 0.369 | 8388608 |

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TranslateVector3D | 115.199 | 7.823 | 4194304 | TranslateVector3D | 114.369 | 2.945 | 4194304 | TranslateVector3D | 74.565 | 0.064 | 4194304 | TranslateVector3D | 71.368 | 0.505 | 4194304 |
| TranslateVector3D | 108.418 | 0.172 | 4194304 | TranslateVector3D | 112.517 | 0.608 | 4194304 | TranslateVector3D | 74.868 | 0.701 | 4194304 | TranslateVector3D | 70.866 | 0.213 | 4194304 |
| TranslateVector3D | 108.421 | 0.418 | 4194304 | TranslateVector3D | 112.802 | 0.431 | 4194304 | TranslateVector3D | 77.157 | 4.97 | 4194304 | TranslateVector3D | 70.722 | 0.199 | 4194304 |
| TranslateVector3D | 108.515 | 0.268 | 4194304 | TranslateVector3D | 117.855 | 10.4 | 4194304 | TranslateVector3D | 75.106 | 0.641 | 4194304 | TranslateVector3D | 70.722 | 0.104 | 4194304 |
| TranslateVector3D | 108.704 | 0.514 | 4194304 | TranslateVector3D | 112.321 | 0.459 | 4194304 | TranslateVector3D | 75.611 | 1.419 | 4194304 | TranslateVector3D | 70.722 | 0.199 | 4194304 |
| SubtractVector2D | 109.201 | 2.405 | 4194304 | SubtractVector2D | 122.721 | 0.953 | 2097152 | SubtractVector2D | 80.54 | 0.922 | 4194304 | SubtractVector2D | 85.854 | 0.196 | 4194304 |
| SubtractVector2D | 108.049 | 0.449 | 4194304 | SubtractVector2D | 122.859 | 0.797 | 2097152 | SubtractVector2D | 80.301 | 0.375 | 4194304 | SubtractVector2D | 87.267 | 2.133 | 4194304 |
| SubtractVector2D | 107.942 | 0.695 | 4194304 | SubtractVector2D | 121.797 | 0.545 | 2097152 | SubtractVector2D | 80.778 | 0.378 | 4194304 | SubtractVector2D | 86.495 | 0.608 | 4194304 |
| SubtractVector2D | 108.227 | 0.831 | 4194304 | SubtractVector2D | 122.205 | 0.256 | 2097152 | SubtractVector2D | 80.65 | 0.771 | 4194304 | SubtractVector2D | 86.405 | 0.639 | 4194304 |
| SubtractVector2D | 108.656 | 0.773 | 4194304 | SubtractVector2D | 129.787 | 11.112 | 2097152 | SubtractVector2D | 80.491 | 0.636 | 4194304 | SubtractVector2D | 86.788 | 0.747 | 4194304 |
| SubtractVector3D | 107.656 | 0.314 | 4194304 | SubtractVector3D | 122.355 | 0.433 | 2097152 | SubtractVector3D | 80.62 | 0.44 | 4194304 | SubtractVector3D | 83.795 | 0.633 | 4194304 |
| SubtractVector3D | 108.228 | 0.74 | 4194304 | SubtractVector3D | 122.778 | 0.508 | 2097152 | SubtractVector3D | 85.346 | 6.51 | 4194304 | SubtractVector3D | 83.451 | 0.265 | 4194304 |
| SubtractVector3D | 107.656 | 0.199 | 4194304 | SubtractVector3D | 122.927 | 0.755 | 2097152 | SubtractVector3D | 80.443 | 0.264 | 4194304 | SubtractVector3D | 83.685 | 0.261 | 4194304 |
| SubtractVector3D | 107.799 | 0.463 | 4194304 | SubtractVector3D | 123.242 | 1.286 | 2097152 | SubtractVector3D | 80.688 | 0.669 | 4194304 | SubtractVector3D | 83.493 | 0.211 | 4194304 |
| SubtractVector3D | 108.18 | 0.692 | 4194304 | SubtractVector3D | 123.43 | 1.388 | 2097152 | SubtractVector3D | 80.684 | 0.431 | 4194304 | SubtractVector3D | 85.924 | 6.19 | 4194304 |
| LengthVector2D | 56.497 | 0.099 | 8388608 | LengthVector2D | 69.15 | 0.543 | 4194304 | LengthVector2D | 44.44 | 0.289 | 8388608 | LengthVector2D | 51.088 | 0.285 | 8388608 |
| LengthVector2D | 56.783 | 0.481 | 8388608 | LengthVector2D | 71.58 | 6.221 | 4194304 | LengthVector2D | 44.511 | 0.105 | 8388608 | LengthVector2D | 51.517 | 0.26 | 8388608 |
| LengthVector2D | 56.592 | 0.12 | 8388608 | LengthVector2D | 68.34 | 0.201 | 4194304 | LengthVector2D | 44.583 | 0.052 | 8388608 | LengthVector2D | 51.263 | 0.502 | 8388608 |
| LengthVector2D | 57.26 | 1.566 | 8388608 | LengthVector2D | 68.625 | 0.556 | 4194304 | LengthVector2D | 44.606 | 0.4 | 8388608 | LengthVector2D | 51.209 | 0.176 | 8388608 |
| LengthVector2D | 56.52 | 0.234 | 8388608 | LengthVector2D | 69.387 | 2.389 | 4194304 | LengthVector2D | 45.011 | 1.16 | 8388608 | LengthVector2D | 51.303 | 0.183 | 8388608 |
| LengthVector3D | 58.165 | 0.272 | 8388608 | LengthVector3D | 73.342 | 7.487 | 4194304 | LengthVector3D | 47.155 | 0.308 | 8388608 | LengthVector3D | 54.209 | 0.358 | 8388608 |
| LengthVector3D | 58.307 | 0.525 | 8388608 | LengthVector3D | 69.864 | 0.258 | 4194304 | LengthVector3D | 47.061 | 0.366 | 8388608 | LengthVector3D | 55.615 | 4.215 | 8388608 |
| LengthVector3D | 58.065 | 0.22 | 8388608 | LengthVector3D | 70.332 | 0.322 | 4194304 | LengthVector3D | 47.228 | 0.136 | 8388608 | LengthVector3D | 53.956 | 0.265 | 8388608 |
| LengthVector3D | 58.212 | 0.247 | 8388608 | LengthVector3D | 70.442 | 0.585 | 4194304 | LengthVector3D | 47.394 | 0.433 | 8388608 | LengthVector3D | 53.868 | 0.298 | 8388608 |
| LengthVector3D | 58.363 | 0.375 | 8388608 | LengthVector3D | 70.137 | 0.561 | 4194304 | LengthVector3D | 47.251 | 0.246 | 8388608 | LengthVector3D | 53.701 | 0.114 | 8388608 |
| DotProductVector2D | 57.098 | 0.488 | 8388608 | DotProductVector2D | 71.038 | 0.163 | 4194304 | DotProductVector2D | 38.621 | 0.12 | 8388608 | DotProductVector2D | 41.939 | 0.436 | 8388608 |
| DotProductVector2D | 56.902 | 0.216 | 8388608 | DotProductVector2D | 71.525 | 0.53 | 4194304 | DotProductVector2D | 38.625 | 0.098 | 8388608 | DotProductVector2D | 41.843 | 0.155 | 8388608 |
| DotProductVector2D | 56.949 | 0.279 | 8388608 | DotProductVector2D | 71.105 | 0.426 | 4194304 | DotProductVector2D | 38.72 | 0.237 | 8388608 | DotProductVector2D | 41.723 | 0.271 | 8388608 |
| DotProductVector2D | 59.69 | 3.922 | 8388608 | DotProductVector2D | 71.035 | 0.325 | 4194304 | DotProductVector2D | 38.78 | 0.469 | 8388608 | DotProductVector2D | 41.938 | 0.534 | 8388608 |
| DotProductVector2D | 57.164 | 0.284 | 8388608 | DotProductVector2D | 70.921 | 0.129 | 4194304 | DotProductVector2D | 38.621 | 0.066 | 8388608 | DotProductVector2D | 44.148 | 4.695 | 8388608 |
| DotProductVector3D | 61.381 | 0.549 | 4194304 | DotProductVector3D | 77.357 | 0.589 | 4194304 | DotProductVector3D | 42.393 | 0.05 | 8388608 | DotProductVector3D | 49.087 | 0.358 | 8388608 |
| DotProductVector3D | 62.286 | 2.226 | 4194304 | DotProductVector3D | 77.474 | 0.569 | 4194304 | DotProductVector3D | 42.622 | 0.527 | 8388608 | DotProductVector3D | 48.903 | 0.242 | 8388608 |
| DotProductVector3D | 61.62 | 0.322 | 4194304 | DotProductVector3D | 77.761 | 0.541 | 4194304 | DotProductVector3D | 42.487 | 0.231 | 8388608 | DotProductVector3D | 48.927 | 0.309 | 8388608 |
| DotProductVector3D | 61.956 | 0.53 | 4194304 | DotProductVector3D | 77.526 | 0.525 | 4194304 | DotProductVector3D | 42.587 | 0.524 | 8388608 | DotProductVector3D | 48.778 | 0.29 | 8388608 |
| DotProductVector3D | 61.382 | 0.211 | 4194304 | DotProductVector3D | 77.435 | 0.464 | 4194304 | DotProductVector3D | 42.367 | 0.32 | 8388608 | DotProductVector3D | 48.885 | 0.293 | 8388608 |

# Appendix C

# Macrobenchmark Data

| Unity-F#-Editor | | | Unity-F#-Release | | | Unity-C#-Editor | | | Unity-C#-Release | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Iteration no. | Time (microseco | Comment | Iteration no. | Time (microseco | Comment | Iteration no. | Time (microseco | Comment | Iteration no. | Time (microseco | Comment |
| 0 | 17531.7 | | 0 | 155972.4 | | 0 | 8818.6 | | 0 | 131205.2 | |
| 1 | 863.9 | | 1 | 919.4 | | 1 | 786.1 | | 1 | 8737.9 | |
| 2 | 414.8 | | 2 | 363.1 | | 2 | 325.4 | | 2 | 6953.2 | |
| 3 | 798 | | 3 | 1071.7 | | 3 | 810.4 | | 3 | 1834.6 | |
| 4 | 196.3 | | 4 | 71 | | 4 | 107.3 | | 4 | 102.2 | |
| 5 | 144.1 | | 5 | 86.4 | | 5 | 136 | | 5 | 89.8 | |
| 6 | 716.3 | gold | 6 | 545.3 | gold | 6 | 679.9 | gold | 6 | 16123.4 | gold |
| 7 | 644.4 | | 7 | 360.9 | | 7 | 716.7 | | 7 | 23246.9 | |
| 8 | 118 | | 8 | 54.3 | | 8 | 97.1 | | 8 | 76.6 | |
| 9 | 142.9 | | 9 | 105.6 | | 9 | 132.1 | | 9 | 401.6 | |
| 10 | 129.5 | | 10 | 70.1 | | 10 | 129.6 | | 10 | 50 | |
| 11 | 155.7 | | 11 | 91.1 | | 11 | 332.3 | | 11 | 59 | |
| 12 | 1161.1 | reset | 12 | 825.3 | reset | 12 | 1865.8 | reset | 12 | 1123.9 | reset |
| 13 | 180.1 | | 13 | 76.5 | | 13 | 139.8 | | 13 | 72.3 | |
| 14 | 133 | | 14 | 69.3 | | 14 | 121.4 | | 14 | 56.9 | |
| 15 | 163 | | 15 | 78.2 | | 15 | 136.8 | | 15 | 91.5 | |
| 16 | 158.3 | | 16 | 62.9 | | 16 | 128.3 | | 16 | 67.6 | |
| 17 | 158.7 | | 17 | 121 | | 17 | 142.8 | | 17 | 71.4 | |
| 18 | 267.3 | | 18 | 63.8 | | 18 | 121.1 | | 18 | 65 | |
| 19 | 194.5 | gold | 19 | 174 | gold | 19 | 171.9 | gold | 19 | 156.9 | gold |
| 20 | 156.9 | | 20 | 73.2 | | 20 | 136.9 | | 20 | 67.6 | |
| 21 | 153.5 | | 21 | 56 | | 21 | 104.4 | | 21 | 80 | |
| 22 | 168.9 | | 22 | 76.1 | | 22 | 143.6 | | 22 | 63.3 | |
| 23 | 147.9 | | 23 | 54.3 | | 23 | 113.7 | | 23 | 56 | |
| 24 | 148.8 | | 24 | 84.3 | | 24 | 133.9 | | 24 | 79.6 | |
| 25 | 241.6 | reset | 25 | 141.2 | reset | 25 | 211.2 | reset | 25 | 123.5 | reset |
| 26 | 163 | | 26 | 87.7 | | 26 | 154.9 | | 26 | 111.2 | |
| 27 | 214.2 | | 27 | 59.5 | | 27 | 112.1 | | 27 | 59 | |
| 28 | 383.2 | | 28 | 83 | | 28 | 136.4 | | 28 | 73.1 | |
| 29 | 153.5 | | 29 | 74 | | 29 | 124 | | 29 | 64.5 | |
| 30 | 165.1 | | 30 | 78.7 | | 30 | 139.8 | | 30 | 88.9 | |
| 31 | 140.7 | | 31 | 66.3 | | 31 | 118.5 | | 31 | 62.4 | |
| 32 | 255.8 | gold | 32 | 91.1 | gold | 32 | 316.9 | gold | 32 | 159.9 | gold |
| 33 | 148.8 | | 33 | 67.6 | | 33 | 203.2 | | 33 | 71.9 | |
| 34 | 115.9 | | 34 | 119.7 | | 34 | 98.7 | | 34 | 81.3 | |
| 35 | 245.1 | | 35 | 73.1 | | 35 | 184.7 | | 35 | 61.6 | |
| 36 | 206.1 | | 36 | 69.3 | | 36 | 115 | | 36 | 54.3 | |
| 37 | 178.7 | | 37 | 629 | | 37 | 140.7 | | 37 | 66.3 | |
| 38 | 405.5 | reset | 38 | 190.7 | reset | 38 | 252.8 | reset | 38 | 121 | reset |
| 39 | 157.4 | | 39 | 98.3 | | 39 | 201.9 | | 39 | 76.6 | |
| 40 | 239.1 | | 40 | 58.2 | | 40 | 109.9 | | 40 | 62 | |
| 41 | 207.4 | | 41 | 78.7 | | 41 | 154 | | 41 | 541.8 | |
| 42 | 148.4 | | 42 | 69.7 | | 42 | 129.6 | | 42 | 92.8 | |
| 43 | 164.3 | | 43 | 77.4 | | 43 | 136.8 | | 43 | 71 | |
| 44 | 141.6 | | 44 | 60.7 | | 44 | 121.4 | | 44 | 56.9 | |
| 45 | 366.9 | gold | 45 | 91.1 | gold | 45 | 171.1 | gold | 45 | 91.1 | gold |
| 46 | 212.2 | | 46 | 73.5 | | 46 | 141.6 | | 46 | 100 | |
| 47 | 124.9 | | 47 | 59.8 | | 47 | 96.7 | | 47 | 56 | |
| 48 | 142 | | 48 | 94 | | 48 | 115.9 | | 48 | 64.6 | |
| 49 | 118 | | 49 | 61.6 | | 49 | 100.9 | | 49 | 49.2 | |
| 50 | 151.8 | | 50 | 69.3 | | 50 | 124.9 | | 50 | 92.4 | |
| 51 | 248.9 | reset | 51 | 122.3 | reset | 51 | 221.1 | reset | 51 | 128.7 | reset |
| 52 | 225.8 | | 52 | 116.7 | | 52 | 156.1 | | 52 | 75.7 | |
| 53 | 134.7 | | 53 | 77 | | 53 | 212.1 | | 53 | 55.6 | |
| 54 | 155.7 | | 54 | 76.1 | | 54 | 148.4 | | 54 | 63.3 | |
| 55 | 149.7 | | 55 | 87.2 | | 55 | 249.8 | | 55 | 62.9 | |
| 56 | 218.1 | | 56 | 80.8 | | 56 | 198.4 | | 56 | 71.4 | |
| 57 | 141.1 | | 57 | 60.7 | | 57 | 121.5 | | 57 | 61.2 | |
| 58 | 189.4 | gold | 58 | 93.6 | gold | 58 | 171.9 | gold | 58 | 121 | gold |
| 59 | 162.1 | | 59 | 71 | | 59 | 143.2 | | 59 | 70.5 | |
| 60 | 118.9 | | 60 | 64.6 | | 60 | 109.1 | | 60 | 52.6 | |
| 61 | 178.8 | | 61 | 68.4 | | 61 | 124.5 | | 61 | 61.1 | |
| 62 | 116.7 | | 62 | 62.4 | | 62 | 116.4 | | 62 | 52.6 | |
| 63 | 409.7 | | 63 | 66.3 | | 63 | 139 | | 63 | 66.3 | |
| 64 | 238.7 | reset | 64 | 176.6 | reset | 64 | 266 | reset | 64 | 127.9 | reset |
| 65 | 382.7 | | 65 | 72.3 | | 65 | 162.9 | | 65 | 74.5 | |
| 66 | 146.6 | | 66 | 60.8 | | 66 | 126.6 | | 66 | 58.2 | |
| 67 | 159 | | 67 | 75.7 | | 67 | 150.1 | | 67 | 64.6 | |
| 68 | 146.2 | | 68 | 64.1 | | 68 | 143.3 | | 68 | 62 | |
| 69 | 119.3 | | 69 | 80.9 | | 69 | 162.1 | | 69 | 66.7 | |
| 70 | 139.9 | | 70 | 71 | | 70 | 133.4 | | 70 | 65.4 | |
| 71 | 224.6 | gold | 71 | 83.4 | gold | 71 | 198.8 | gold | 71 | 112.5 | gold |
| 72 | 373.3 | | 72 | 69.3 | | 72 | 156.9 | | 72 | 95.8 | |
| 73 | 151 | | 73 | 62 | | 73 | 252.7 | | 73 | 64.6 | |
| 74 | 143.7 | | 74 | 68.4 | | 74 | 408.4 | | 74 | 81.2 | |
| 75 | 114.2 | | 75 | 55.2 | | 75 | 376.7 | | 75 | 55.6 | |
| 76 | 125.8 | | 76 | 68 | | 76 | 148.4 | | 76 | 60.7 | |
| 77 | 229.2 | reset | 77 | 125.8 | reset | 77 | 205.3 | reset | 77 | 128.3 | reset |
| 78 | 162.9 | | 78 | 129.1 | | 78 | 131.3 | | 78 | 76.1 | |
| 79 | 122.8 | | 79 | 59.1 | | 79 | 118.1 | | 79 | 54.8 | |
| 80 | 173.7 | | 80 | 71 | | 80 | 141.1 | | 80 | 67.1 | |
| 81 | 150.5 | | 81 | 88.1 | | 81 | 129.6 | | 81 | 59.8 | |
| 82 | 156.1 | | 82 | 67.5 | | 82 | 144.1 | | 82 | 73.1 | |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 83 | 176.6 | | 83 | 431.1 | | 83 | 125.3 | | 83 | 74 | |
| 84 | 217.2 | gold | 84 | 93.7 | gold | 84 | 176.2 | gold | 84 | 479.8 | gold |
| 85 | 170.2 | | 85 | 67.6 | | 85 | 189 | | 85 | 65.9 | |
| 86 | 114.6 | | 86 | 54.7 | | 86 | 97.9 | | 86 | 55.2 | |
| 87 | 146.3 | | 87 | 72.2 | | 87 | 128.8 | | 87 | 73.2 | |
| 88 | 126.1 | | 88 | 57.7 | | 88 | 121.8 | | 88 | 65.9 | |
| 89 | 147.1 | | 89 | 70.5 | | 89 | 90.2 | | 89 | 62.8 | |
| 90 | 255.3 | reset | 90 | 130.9 | reset | 90 | 226.2 | reset | 90 | 129.6 | reset |
| 91 | 170.2 | | 91 | 69.3 | | 91 | 145 | | 91 | 89.4 | |
| 92 | 444.3 | | 92 | 142.4 | | 92 | 105.2 | | 92 | 56.9 | |
| 93 | 163.7 | | 93 | 83.8 | | 93 | 140.7 | | 93 | 68 | |
| 94 | 189.5 | | 94 | 75.7 | | 94 | 128.3 | | 94 | 65.9 | |
| 95 | 124.9 | | 95 | 72.7 | | 95 | 136.4 | | 95 | 64.6 | |
| 96 | 135.6 | | 96 | 63.3 | | 96 | 165.9 | | 96 | 61.1 | |
| 97 | 361.4 | gold | 97 | 88.5 | gold | 97 | 231.8 | gold | 97 | 85.5 | gold |
| 98 | 179.2 | | 98 | 74.4 | | 98 | 161.3 | | 98 | 90.6 | |
| 99 | 111.6 | | 99 | 55.2 | | 99 | 97.5 | | 99 | 53.5 | |
| 100 | 162.1 | | 100 | 63.7 | | 100 | 127.5 | | 100 | 83.4 | |
| 101 | 133.8 | | 101 | 58.5 | | 101 | 103.9 | | 101 | 55.6 | |
| 102 | 163.3 | | 102 | 68 | | 102 | 127.8 | | 102 | 68.8 | |
| 103 | 260 | reset | 103 | 174.9 | reset | 103 | 223.7 | reset | 103 | 127 | reset |
| 104 | 185.6 | | 104 | 94.9 | | 104 | 140.2 | | 104 | 92.8 | |
| 105 | 146.7 | | 105 | 62.5 | | 105 | 113.8 | | 105 | 54.8 | |
| 106 | 156.1 | | 106 | 72.2 | | 106 | 175.8 | | 106 | 66.3 | |
| 107 | 144.5 | | 107 | 62.8 | | 107 | 149.2 | | 107 | 63.3 | |
| 108 | 166.8 | | 108 | 76.5 | | 108 | 167.6 | | 108 | 97.5 | |
| 109 | 140.7 | | 109 | 65 | | 109 | 140.7 | | 109 | 59.8 | |
| 110 | 198.9 | gold | 110 | 90.7 | gold | 110 | 204 | gold | 110 | 78.7 | gold |
| 111 | 150.6 | | 111 | 96.6 | | 111 | 159.5 | | 111 | 74 | |
| 112 | 118.5 | | 112 | 53.9 | | 112 | 116.3 | | 112 | 50.9 | |
| 113 | 307.5 | | 113 | 77.9 | | 113 | 167.2 | | 113 | 58.6 | |
| 114 | 118.1 | | 114 | 57.8 | | 114 | 122.7 | | 114 | 54.3 | |
| 115 | 141.5 | | 115 | 71.9 | | 115 | 146.2 | | 115 | 64.2 | |
| 116 | 238.7 | reset | 116 | 133.8 | reset | 116 | 845.9 | reset | 116 | 163.8 | reset |
| 117 | 156.5 | | 117 | 80.4 | | 117 | 538.4 | | 117 | 79.2 | |
| 118 | 170.6 | | 118 | 63.3 | | 118 | 107.4 | | 118 | 56 | |
| 119 | 154.4 | | 119 | 106.1 | | 119 | 145.8 | | 119 | 68 | |
| 120 | 181.8 | | 120 | 67.2 | | 120 | 144.1 | | 120 | 60.3 | |
| 121 | 196.3 | | 121 | 79.1 | | 121 | 175.8 | | 121 | 73.1 | |
| 122 | 145 | | 122 | 69.2 | | 122 | 137.3 | | 122 | 56.4 | |
| 123 | 419.5 | gold | 123 | 87.2 | gold | 123 | 209.1 | gold | 123 | 85.5 | gold |
| 124 | 172.3 | | 124 | 66.3 | | 124 | 203.2 | | 124 | 70.1 | |
| 125 | 136.4 | | 125 | 53.9 | | 125 | 122.3 | | 125 | 53.1 | |
| 126 | 161.7 | | 126 | 70.6 | | 126 | 151.9 | | 126 | 67.1 | |
| 127 | 169.4 | | 127 | 57.3 | | 127 | 119.7 | | 127 | 56.9 | |
| 128 | 135.6 | | 128 | 75.3 | | 128 | 157 | | 128 | 69.3 | |
| 129 | 237.8 | reset | 129 | 161.6 | reset | 129 | 266.8 | reset | 129 | 125.3 | reset |
| 130 | 172 | | 130 | 627.4 | | 130 | 175.7 | | 130 | 67.6 | |
| 131 | 230.9 | | 131 | 60.7 | | 131 | 144.1 | | 131 | 54.8 | |
| 132 | 157.4 | | 132 | 69.3 | | 132 | 167.2 | | 132 | 465.7 | |
| 133 | 152.7 | | 133 | 83.9 | | 133 | 165.1 | | 133 | 66.7 | |
| 134 | 161.2 | | 134 | 97.1 | | 134 | 142.4 | | 134 | 94.1 | |
| 135 | 240.8 | | 135 | 67.6 | | 135 | 113.8 | | 135 | 62 | |
| 136 | 198 | gold | 136 | 88.1 | gold | 136 | 171.9 | gold | 136 | 88.1 | gold |
| 137 | 151.8 | | 137 | 91.9 | | 137 | 186 | | 137 | 66.3 | |
| 138 | 121 | | 138 | 59.9 | | 138 | 101.8 | | 138 | 50 | |
| 139 | 159 | | 139 | 84.7 | | 139 | 127 | | 139 | 71.4 | |
| 140 | 114.6 | | 140 | 63.2 | | 140 | 100.9 | | 140 | 63.7 | |
| 141 | 250.6 | | 141 | 75.3 | | 141 | 135.1 | | 141 | 68.8 | |
| 142 | 518.3 | reset | 142 | 134.7 | reset | 142 | 244.2 | reset | 142 | 127.5 | reset |
| 143 | 172.8 | | 143 | 74.4 | | 143 | 314.3 | | 143 | 71.4 | |
| 144 | 132.2 | | 144 | 62.8 | | 144 | 284.4 | | 144 | 56.4 | |
| 145 | 165.5 | | 145 | 71.9 | | 145 | 144.2 | | 145 | 68.9 | |
| 146 | 183.1 | | 146 | 74 | | 146 | 127.8 | | 146 | 67.1 | |
| 147 | 184.3 | | 147 | 110.3 | | 147 | 142.9 | | 147 | 93.2 | |
| 148 | 143.7 | | 148 | 68.9 | | 148 | 119.3 | | 148 | 60.7 | |
| 149 | 201.8 | gold | 149 | 95 | gold | 149 | 176.2 | gold | 149 | 90.7 | gold |
| 150 | 167.7 | | 150 | 104.8 | | 150 | 135.1 | | 150 | 67.6 | |
| 151 | 207.8 | | 151 | 52.1 | | 151 | 105.6 | | 151 | 56.4 | |
| 152 | 173.7 | | 152 | 66.7 | | 152 | 137.2 | | 152 | 64.6 | |
| 153 | 135.2 | | 153 | 53.5 | | 153 | 91.5 | | 153 | 59 | |
| 154 | 162.1 | | 154 | 93.2 | | 154 | 129.6 | | 154 | 59.9 | |
| 155 | 263.8 | reset | 155 | 123.2 | reset | 155 | 228.8 | reset | 155 | 304.5 | reset |
| 156 | 159.9 | | 156 | 75.7 | | 156 | 141.2 | | 156 | 84.3 | |
| 157 | 130 | | 157 | 62 | | 157 | 103.1 | | 157 | 62.1 | |
| 158 | 234.4 | | 158 | 74.4 | | 158 | 143.7 | | 158 | 115.5 | |
| 159 | 145.8 | | 159 | 75.2 | | 159 | 125.3 | | 159 | 76.2 | |
| 160 | 127 | | 160 | 99.7 | | 160 | 142.4 | | 160 | 74.9 | |
| 161 | 137.2 | | 161 | 86.8 | | 161 | 456.7 | | 161 | 86.4 | |
| 162 | 196.7 | gold | 162 | 95 | gold | 162 | 183 | gold | 162 | 139 | gold |
| 163 | 197.6 | | 163 | 76.9 | | 163 | 175.3 | | 163 | 100.1 | |
| 164 | 86.4 | | 164 | 52.6 | | 164 | 247.2 | | 164 | 76.6 | |
| 165 | 160.3 | | 165 | 66.7 | | 165 | 120.2 | | 165 | 96.6 | |
| 166 | 133.9 | | 166 | 56.1 | | 166 | 107.4 | | 166 | 55.6 | |
| 167 | 161.6 | | 167 | 67.6 | | 167 | 171 | | 167 | 63.7 | |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 168 | 264.3 | reset | 168 | 155.2 | reset | 168 | 223.2 | reset | 168 | 124 | reset |
| 169 | 183.4 | | 169 | 78.3 | | 169 | 145.8 | | 169 | 93.2 | |
| 170 | 148 | | 170 | 83 | | 170 | 120.2 | | 170 | 55.2 | |
| 171 | 175.7 | | 171 | 78.7 | | 171 | 178.8 | | 171 | 73.6 | |
| 172 | 276.7 | | 172 | 64.2 | | 172 | 128.3 | | 172 | 60.3 | |
| 173 | 120.6 | | 173 | 74.9 | | 173 | 138.1 | | 173 | 70.6 | |
| 174 | 142.9 | | 174 | 69.7 | | 174 | 118.9 | | 174 | 75.7 | |
| 175 | 258.3 | gold | 175 | 148 | gold | 175 | 156.9 | gold | 175 | 90.6 | gold |
| 176 | 154.8 | | 176 | 70.2 | | 176 | 138.2 | | 176 | 66.3 | |
| 177 | 120.2 | | 177 | 61.2 | | 177 | 103.9 | | 177 | 436.2 | |
| 178 | 139.4 | | 178 | 67.6 | | 178 | 130.9 | | 178 | 81.3 | |
| 179 | 121.1 | | 179 | 62 | | 179 | 97.5 | | 179 | 66.7 | |
| 180 | 144.6 | | 180 | 62.9 | | 180 | 125.3 | | 180 | 128.3 | |
| 181 | 411 | reset | 181 | 128.8 | reset | 181 | 229.2 | reset | 181 | 158.7 | reset |
| 182 | 153.1 | | 182 | 72.2 | | 182 | 147.5 | | 182 | 75.7 | |
| 183 | 132.1 | | 183 | 73.2 | | 183 | 121 | | 183 | 59.9 | |
| 184 | 134.3 | | 184 | 79.5 | | 184 | 139 | | 184 | 65.1 | |
| 185 | 152.3 | | 185 | 71.9 | | 185 | 115.9 | | 185 | 71.4 | |
| 186 | 162.1 | | 186 | 76.1 | | 186 | 245 | | 186 | 70.5 | |
| 187 | 144.1 | | 187 | 87.2 | | 187 | 86.9 | | 187 | 62 | |
| 188 | 193.3 | gold | 188 | 103.9 | gold | 188 | 172.3 | gold | 188 | 88.9 | gold |
| 189 | 153.1 | | 189 | 192 | | 189 | 141.2 | | 189 | 68 | |
| 190 | 84.7 | | 190 | 72.7 | | 190 | 103 | | 190 | 54.3 | |
| 191 | 143.3 | | 191 | 74.9 | | 191 | 130.4 | | 191 | 59.9 | |
| 192 | 112.1 | | 192 | 57.3 | | 192 | 96.2 | | 192 | 57.7 | |
| 193 | 145 | | 193 | 67.5 | | 193 | 124.9 | | 193 | 132.6 | |
| 194 | 242.4 | reset | 194 | 313 | reset | 194 | 225.8 | reset | 194 | 164.2 | reset |
| 195 | 155.2 | | 195 | 77.9 | | 195 | 154.4 | | 195 | 107.8 | |
| 196 | 133.4 | | 196 | 64.6 | | 196 | 148.4 | | 196 | 57.3 | |
| 197 | 279.7 | | 197 | 67.1 | | 197 | 141.1 | | 197 | 74.9 | |
| 198 | 144.5 | | 198 | 63.7 | | 198 | 126.6 | | 198 | 58.6 | |
| 199 | 158.2 | | 199 | 74 | | 199 | 143.2 | | 199 | 72.3 | |
| 200 | 176.6 | | 200 | 64.2 | | 200 | 163.4 | | 200 | 70.1 | |
| 201 | 267.3 | gold | 201 | 82.2 | gold | 201 | 169.8 | gold | 201 | 121.1 | gold |
| 202 | 144.5 | | 202 | 76.1 | | 202 | 142.8 | | 202 | 68.4 | |
| 203 | 121.1 | | 203 | 77.4 | | 203 | 100.9 | | 203 | 53.5 | |
| 204 | 141.6 | | 204 | 68.8 | | 204 | 130.4 | | 204 | 64.6 | |
| 205 | 115.9 | | 205 | 56.4 | | 205 | 110.8 | | 205 | 52.6 | |
| 206 | 179.6 | | 206 | 65 | | 206 | 211.3 | | 206 | 60.8 | |
| 207 | 208.7 | reset | 207 | 135.9 | reset | 207 | 309.2 | reset | 207 | 133 | reset |
| 208 | 160 | | 208 | 68.9 | | 208 | 141.5 | | 208 | 65.8 | |
| 209 | 130.8 | | 209 | 61.2 | | 209 | 116.4 | | 209 | 57.3 | |
| 210 | 110.3 | | 210 | 74 | | 210 | 138.1 | | 210 | 67.6 | |
| 211 | 148.9 | | 211 | 86.8 | | 211 | 122.3 | | 211 | 75.7 | |
| 212 | 160.3 | | 212 | 69.7 | | 212 | 140.2 | | 212 | 65.4 | |
| 213 | 138.9 | | 213 | 71 | | 213 | 116.8 | | 213 | 64.5 | |
| 214 | 192.8 | gold | 214 | 102.2 | gold | 214 | 179.2 | gold | 214 | 84.7 | gold |
| 215 | 148 | | 215 | 95.8 | | 215 | 180.5 | | 215 | 65.8 | |
| 216 | 117.2 | | 216 | 56.9 | | 216 | 102.2 | | 216 | 62.5 | |
| 217 | 192 | | 217 | 68.4 | | 217 | 128.7 | | 217 | 61.1 | |
| 218 | 121.8 | | 218 | 55.6 | | 218 | 95.8 | | 218 | 49.2 | |
| 219 | 172.8 | | 219 | 66.2 | | 219 | 128.7 | | 219 | 87.7 | |
| 220 | 303.7 | reset | 220 | 236.9 | reset | 220 | 231 | reset | 220 | 135.6 | reset |
| 221 | 284.8 | | 221 | 79.1 | | 221 | 150.1 | | 221 | 67.2 | |
| 222 | 132.6 | | 222 | 70.9 | | 222 | 112.4 | | 222 | 59.4 | |
| 223 | 186.1 | | 223 | 472.5 | | 223 | 142.4 | | 223 | 68.4 | |
| 224 | 260.8 | | 224 | 69.3 | | 224 | 98.3 | | 224 | 458.4 | |
| 225 | 177.1 | | 225 | 68.4 | | 225 | 145.8 | | 225 | 71.9 | |
| 226 | 143.6 | | 226 | 78.7 | | 226 | 121.9 | | 226 | 58.6 | |
| 227 | 190.3 | gold | 227 | 94.1 | gold | 227 | 179.6 | gold | 227 | 79.1 | gold |
| 228 | 265.1 | | 228 | 88.5 | | 228 | 189.5 | | 228 | 95.8 | |
| 229 | 116.4 | | 229 | 60.3 | | 229 | 103.5 | | 229 | 51.7 | |
| 230 | 142 | | 230 | 63.7 | | 230 | 385.7 | | 230 | 59.4 | |
| 231 | 128.3 | | 231 | 54.3 | | 231 | 240.4 | | 231 | 53 | |
| 232 | 154.8 | | 232 | 72.3 | | 232 | 346.8 | | 232 | 82.9 | |
| 233 | 266.4 | reset | 233 | 249.3 | reset | 233 | 244.2 | reset | 233 | 136.4 | reset |
| 234 | 168.5 | | 234 | 81.7 | | 234 | 144.6 | | 234 | 101 | |
| 235 | 135.9 | | 235 | 60.7 | | 235 | 116.7 | | 235 | 58.1 | |
| 236 | 176.6 | | 236 | 68 | | 236 | 191.5 | | 236 | 67.6 | |
| 237 | 144.2 | | 237 | 65 | | 237 | 128.3 | | 237 | 57.7 | |
| 238 | 168.1 | | 238 | 88.9 | | 238 | 147.1 | | 238 | 77.9 | |
| 239 | 138.9 | | 239 | 150.5 | | 239 | 119.3 | | 239 | 57.3 | |
| 240 | 189.9 | gold | 240 | 92 | gold | 240 | 146.2 | gold | 240 | 88.5 | gold |
| 241 | 154.9 | | 241 | 70.1 | | 241 | 141.1 | | 241 | 64.1 | |
| 242 | 122.7 | | 242 | 54.7 | | 242 | 100.1 | | 242 | 50.5 | |
| 243 | 138.5 | | 243 | 79.9 | | 243 | 131.2 | | 243 | 65.4 | |
| 244 | 95.3 | | 244 | 62.8 | | 244 | 105.2 | | 244 | 53 | |
| 245 | 166.8 | | 245 | 64.2 | | 245 | 130.9 | | 245 | 60.3 | |
| 246 | 294.6 | reset | 246 | 134.8 | reset | 246 | 222.3 | reset | 246 | 200.2 | reset |
| 247 | 161.2 | | 247 | 82.1 | | 247 | 144.1 | | 247 | 74 | |
| 248 | 97.5 | | 248 | 61.2 | | 248 | 112.5 | | 248 | 56.1 | |
| 249 | 192.9 | | 249 | 85.5 | | 249 | 145.4 | | 249 | 67.5 | |
| 250 | 180.9 | | 250 | 63.3 | | 250 | 134.7 | | 250 | 64.2 | |
| 251 | 189.9 | | 251 | 74.9 | | 251 | 134.7 | | 251 | 95.3 | |
| 252 | 136.9 | | 252 | 65.4 | | 252 | 335.7 | | 252 | 61.6 | |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 253 | 195.4 | gold | 253 | 81.7 | gold | 253 | 182.6 | gold | 253 | 91.5 | gold |
| 254 | 151.4 | | 254 | 71 | | 254 | 137.7 | | 254 | 65 | |
| 255 | 119.3 | | 255 | 91.1 | | 255 | 101 | | 255 | 56 | |
| 256 | 140.3 | | 256 | 148.4 | | 256 | 133.4 | | 256 | 66.3 | |
| 257 | 118.1 | | 257 | 74.4 | | 257 | 96.6 | | 257 | 50 | |
| 258 | 136.9 | | 258 | 71 | | 258 | 127.9 | | 258 | 67.5 | |
| 259 | 242 | reset | 259 | 124 | reset | 259 | 235.2 | reset | 259 | 163.4 | reset |
| 260 | 156.1 | | 260 | 74 | | 260 | 154.4 | | 260 | 90.7 | |
| 261 | 221.1 | | 261 | 58.2 | | 261 | 109.9 | | 261 | 61.6 | |
| 262 | 159.1 | | 262 | 72.3 | | 262 | 138.6 | | 262 | 65 | |
| 263 | 146.7 | | 263 | 65 | | 263 | 130 | | 263 | 63.3 | |
| 264 | 159.9 | | 264 | 69.7 | | 264 | 151.4 | | 264 | 65.5 | |
| 265 | 169.3 | | 265 | 62.5 | | 265 | 124.4 | | 265 | 60.7 | |
| 266 | 228.8 | gold | 266 | 204.4 | gold | 266 | 183.8 | gold | 266 | 88.5 | gold |
| 267 | 153.5 | | 267 | 67.6 | | 267 | 139.8 | | 267 | 70.6 | |
| 268 | 123.2 | | 268 | 70.1 | | 268 | 100.5 | | 268 | 50.9 | |
| 269 | 251.5 | | 269 | 83.4 | | 269 | 129.1 | | 269 | 64.1 | |
| 270 | 137.7 | | 270 | 691.9 | | 270 | 103.4 | | 270 | 565.4 | |
| 271 | 272 | | 271 | 163.7 | | 271 | 128.7 | | 271 | 67.1 | |
| 272 | 420.8 | reset | 272 | 125.7 | reset | 272 | 252.7 | reset | 272 | 124.4 | reset |
| 273 | 372.1 | | 273 | 70.1 | | 273 | 333.6 | | 273 | 69.3 | |
| 274 | 152.2 | | 274 | 68.4 | | 274 | 118.5 | | 274 | 63.7 | |
| 275 | 164.3 | | 275 | 92.8 | | 275 | 336.1 | | 275 | 71.9 | |
| 276 | 165.5 | | 276 | 68.4 | | 276 | 130.9 | | 276 | 108.6 | |
| 277 | 164.2 | | 277 | 69.3 | | 277 | 142 | | 277 | 68 | |
| 278 | 142.4 | | 278 | 59 | | 278 | 118.9 | | 278 | 74.8 | |
| 279 | 189.9 | gold | 279 | 86 | gold | 279 | 236.5 | gold | 279 | 83 | gold |
| 280 | 153.5 | | 280 | 71.4 | | 280 | 196.3 | | 280 | 73.9 | |
| 281 | 139.8 | | 281 | 64.1 | | 281 | 129.2 | | 281 | 52.6 | |
| 282 | 168.5 | | 282 | 64.6 | | 282 | 157.4 | | 282 | 66.3 | |
| 283 | 141.1 | | 283 | 64.6 | | 283 | 128.2 | | 283 | 52.1 | |
| 284 | 163.3 | | 284 | 74 | | 284 | 161.2 | | 284 | 108.7 | |
| 285 | 272.4 | reset | 285 | 128.3 | reset | 285 | 282.6 | reset | 285 | 121.4 | reset |
| 286 | 164.7 | | 286 | 70.2 | | 286 | 197.5 | | 286 | 73.6 | |
| 287 | 131.8 | | 287 | 59.9 | | 287 | 113.3 | | 287 | 54.7 | |
| 288 | 198.9 | | 288 | 70.5 | | 288 | 256.5 | | 288 | 103.1 | |
| 289 | 148.4 | | 289 | 68.4 | | 289 | 126.6 | | 289 | 63.8 | |
| 290 | 285.7 | | 290 | 80.4 | | 290 | 142.8 | | 290 | 68.9 | |
| 291 | 139.8 | | 291 | 87.2 | | 291 | 116.3 | | 291 | 55.6 | |
| 292 | 193.3 | gold | 292 | 115.1 | gold | 292 | 172.8 | gold | 292 | 93.2 | gold |
| 293 | 132.2 | | 293 | 68.5 | | 293 | 261.8 | | 293 | 67.6 | |
| 294 | 118.9 | | 294 | 62.9 | | 294 | 274.6 | | 294 | 66.3 | |
| 295 | 160.8 | | 295 | 128.3 | | 295 | 122.7 | | 295 | 77 | |
| 296 | 119.7 | | 296 | 57.7 | | 296 | 97 | | 296 | 56 | |
| 297 | 136.4 | | 297 | 65.4 | | 297 | 123.2 | | 297 | 61.5 | |
| 298 | 247.2 | reset | 298 | 122.7 | reset | 298 | 254.5 | reset | 298 | 130.4 | reset |
| 299 | 207 | | 299 | 70.5 | | 299 | 141.6 | | 299 | 66.7 | |
| 300 | 133.4 | | 300 | 88.1 | | 300 | 115 | | 300 | 57.3 | |
| 301 | 161.7 | | 301 | 74.9 | | 301 | 137.3 | | 301 | 100.1 | |
| 302 | 146.3 | | 302 | 66.3 | | 302 | 131.3 | | 302 | 86 | |
| 303 | 270.7 | | 303 | 73.6 | | 303 | 145 | | 303 | 75.7 | |
| 304 | 156.1 | | 304 | 153.5 | | 304 | 123.6 | | 304 | 62.4 | |
| 305 | 197.5 | gold | 305 | 86 | gold | 305 | 178.8 | gold | 305 | 81.7 | gold |
| 306 | 156.9 | | 306 | 67.5 | | 306 | 140.3 | | 306 | 94.5 | |
| 307 | 125.7 | | 307 | 63.8 | | 307 | 100.9 | | 307 | 66.3 | |
| 308 | 124.9 | | 308 | 83.4 | | 308 | 118 | | 308 | 59.9 | |
| 309 | 146.7 | | 309 | 57.7 | | 309 | 79.5 | | 309 | 53.9 | |
| 310 | 174.9 | | 310 | 63.3 | | 310 | 122.3 | | 310 | 59 | |
| 311 | 233.9 | reset | 311 | 133.5 | reset | 311 | 226.7 | reset | 311 | 120.2 | reset |
| 312 | 207 | | 312 | 69.7 | | 312 | 146.3 | | 312 | 74.8 | |
| 313 | 142.8 | | 313 | 58.2 | | 313 | 121.5 | | 313 | 59 | |
| 314 | 207.9 | | 314 | 77 | | 314 | 125.3 | | 314 | 91.1 | |
| 315 | 130.4 | | 315 | 99.2 | | 315 | 608.5 | | 315 | 65.8 | |
| 316 | 291.7 | | 316 | 424.7 | | 316 | 169.4 | | 316 | 539.2 | |
| 317 | 191.6 | | 317 | 62 | | 317 | 144.9 | | 317 | 57.3 | |
| 318 | 217.7 | gold | 318 | 92.8 | gold | 318 | 208.2 | gold | 318 | 88.1 | gold |
| 319 | 172.3 | | 319 | 80.4 | | 319 | 136.4 | | 319 | 69.8 | |
| 320 | 129.1 | | 320 | 59.9 | | 320 | 93.7 | | 320 | 51.3 | |
| 321 | 158.2 | | 321 | 83.9 | | 321 | 123.1 | | 321 | 64.1 | |
| 322 | 129.1 | | 322 | 59 | | 322 | 77 | | 322 | 52.6 | |
| 323 | 214.7 | | 323 | 83 | | 323 | 124.9 | | 323 | 72.7 | |
| 324 | 263.4 | reset | 324 | 128.7 | reset | 324 | 250.2 | reset | 324 | 161.6 | reset |