Mobile Game Engines
Interviews with Mobile Game Engine Developers
Chapter 1

Mario Zechners

Libgdx

1.0.1 Background

Brownlee: Where in the world are you located?
Zechners: Graz, Styria, Austria.

Brownlee: Who do you work for and what is your current role?
Zechners: I work for a small R&D company called Know-Center, in Graz, Austria. We are mostly involved in information retrieval and extraction, bits of machine learning and visualization. I guess I could call myself senior software engineer and researcher.

Brownlee: Could you please introduce yourself?
Zechners: Education wise, I went to primary school, then something we call Gymnasium (precursor to university, takes 8 years plus a final exam in multiple fields like math, languages etc.). Started studying software engineering, got a job and kinda dropped out. There’s still some hope that I’ll finish at least my bachelor, but I’m not sure if I can find the time for it given all my activities.

My work history consists of a couple of internships in my early years at companies like Magna, a long term relationship with the place I work at now and an 8 months stay with a mobile gaming startup in San Francisco which I left to go back to my old company. Besides that I wrote a book on Android game development.

Concerning my programming and CS background: I’m mostly self-taught, although some bits of information got slammed into my head at university. I have a love for compiler engineering and graphics programming, I like to pretend I understand most state of the art machine learning
In terms of projects I did all kinds of things, from writing a C-subset compiler, to creating a C/C++ game development library called Yagl\textsuperscript{1}, similar to SFML\textsuperscript{2} but born way before it, some minor web development, a couple of games, many unfinished prototypes, and libgdx, a cross-platform Java game development framework.

**Brownlee**: How long have you been programming?
**Zechner**: Not entirely sure, but I remember getting my 486 around 1995. Soon after which I started “programming” in QBasic. Around 16-17 years I guess. My main motivation to start programming where games. I always wanted to create my own, a common motivator among programmers as far as I can tell.

**Brownlee**: What are some of the game engines you have worked on?
**Zechner**: I worked on many prototypes, without a lot of code sharing. Those could count as engines I worked on. I worked on Yagl, which is not exactly an engine but more of a framework for C/C++ and Freebasic. Libgdx is probably the biggest thing I worked on.

### 1.0.2 Your Game Engine

**Brownlee**: What is the full name of your game engine?
**Zechner**: Libgdx

**Brownlee**: Could you please describe your game engine?
**Zechner**: Libgdx is a cross-platform Java game development framework that lets you write your game once and run it on multiple platforms.

It abstracts away the differences between the different platforms and provides you with a unified modular system for file I/O, input peripherals, graphics (GL ES 1.x and 2.0), audio and application life-cycle management. On top of that it provides low to high-level helper classes, from textures, meshes, math utils to a full-blown 2D scene graph, and so on. You can pick whatever layer you want to work with at any time. Oh, and it’s not a 2D framework.

**Brownlee**: What platforms does it support?
**Zechner**: Windows, Linux, Mac OS X, WebGL/JS, Android and soonish iOS.

**Brownlee**: What programming language(s) is it written in?

\textsuperscript{1}Yet Another Game Programming Library
\textsuperscript{2}Simple and Fast Multimedia Library
Zechner: Java and C/C++ for performance intensive parts.

Brownlee: What is the current price and license structure of your game engine?
Zechner: The framework is open-source software, Apache 2 licensed. While we do not ask for money, we are always happy to receive donations that keep things like the build server alive and kicking.

Brownlee: What are the engines top 10 core features?
Zechner:

- Cross-platform.
- Fast iteration by mostly working on the desktop.
- Well-designed layers of abstraction that let you pick what you want to work with.
- Tons of utility classes that make game development a lot easier, to many to enumerate, see our website.
- Integration of various 3rd party libraries, e.g. box2d, Tremor, libmpg123, freetype and so on.
- Good documentation for an open-source project, still needs improvement though.
- Great community in form of forums and a very active IRC channel.
- Actively maintained, code is pushed every day.

Brownlee: When was it first released?

Brownlee: Could you estimate how many games have been built and released using your engine?
Zechner: According to AppBrain\(^3\), around 1.25\% of all Android apps have been build with libgdx, that amounts to around 6k applications, given that there are currently roughly 500k Android applications around.

Brownlee: What are some well known or notable games created with the engine?
Zechner: Apparatus has probably been one of the greatest success stories. It’s been in the store for a very long time and is still among the top 25 apps on Android.

\(^3\)AppBrain [http://www.appbrain.com](http://www.appbrain.com)
Other successful games would include those by KiwiUp (Monster Park, Brightwoord Adventures) and Bams Away by Beardy Eye. AppBrain has a list of apps they think were build with libgdx.

### 1.0.3 Building Your Game Engine

**Brownlee:** Why did you start working on this game engine?

**Zechner:** I got an Android phone early (HTC Hero, in late 2009) and wanted to program a game like Super Monkey Ball for it. I faced two problems: there weren’t any frameworks or engines for Android at that point, and developing on the device was horribly slow (it still is). I thus though I could make a cross platform layer that allows me to target both the desktop and Android. The result was libgdx. The game itself never got done.

**Brownlee:** What was the first element you had working in the game engine?

**Zechner:** The application life-cycle, graphics and input were the first to work as they are essential to test other things. Files and audio came later. The first iteration was really just composed of the low-level abstractions. Once those were in place I started implementing more high-level abstractions as I needed them.

**Brownlee:** Did you initially develop alone or in a team?

**Zechner:** I initially developed alone. Contributors hopped on board a few months after the first release. I was lucky to have a small following on my blog due to my lessons learned articles concerning Android’s game dev related strange behavior and bugs.

We are now a pretty big team. At any given point in time over the last year we had at least 3-4 people commit bug fixes and new code regularly. Some people leave, others come in. Nate and myself are the heaviest contributors.

**Brownlee:** Why create a new game engine rather than extend or license an existing engine?

**Zechner:** There simply wasn’t an alternative on Android, so I created my own.

**Brownlee:** Please describe the high-level layout or structure of the game engine?

**Zechner:** Libgdx is a stack of layers, each abstracting away the nastiness of the lower layer.

The bottom layer is a bunch of simple interfaces for application lifecycle management, file I/O, audio, graphics and input. Porting libgdx to a
new platform means implementing these simple interfaces on that platform. We also have quite a bit of native C/C++ code in here to cope with a few performance issues (matrix math, memory management, custom 2D graphics library implemented in C, etc.)

Next up we have utility classes directly built on top of the bottom layer. These are mostly focused on Graphics and input, e.g. abstractions for meshes, textures, shaders and so on, or a gesture detector for pinch zoom fling, drag, you name it. A sibling layer contains math utilities like matrices, vectors, intersection testing, etc. Another sibling layer contains custom collection classes to reduce garbage collection invocations among other things.

The third layer consists of things like texture atlases (aka sprite sheets), bitmap fonts, sprites and performant sprite rendering classes (aka sprite batching), 3D model format loaders and an asset manager that let’s you asynchronously load assets and has a reference counting mechanism so you don’t load the same asset multiple times.

In the last layer we currently have a pretty powerful 2D scene graph, on top of which we implemented a capable UI toolkit to make creation of UIs easier and cross-platform. There’s also a full box2d JNI wrapper (calls native code from within Java to speed things up).

All of this is integrated in the core API. Besides the core API we offer extensions for things like audio processing and decoding, freetype integration, a bullet physics wrapper and so on. Extensions might not be available on all platforms, e.g. due to their C/C++ nature, which doesn’t work to well on the web (yes, we looked into emscripten\textsuperscript{4}).

What we currently lack is more sophisticated 3D support. What we have at the moment is sufficient for moderately complex 3D games. We have some exciting things in the pipeline though.

**Brownlee:** What are some challenges in supporting multiple mobile platforms?

**Zechner:** Coming up with abstractions that work on all platforms. The life-cycles of Android, iOS, web and desktop applications are quite a bit different for example. The same is true for input peripherals. File-systems are the biggest pain in the ass as far as I’m concerned. However, I think we managed to abstract that in a way where it’s working pretty well across all platforms, even on the web.

Interestingly enough, graphics aren’t all that hard since we put all our money on OpenGL ES 1.x and 2.0. On the desktop it’s easy to emulate through normal OpenGL, Android an iOS support OpenGL ES natively, and WebGL is pretty much OpenGL ES 2.0 with a few caveats.

**Brownlee:** What are your thoughts on the trade-off between engine per-
formance and flexibility?

**Zechner:** I don’t think speed and flexibility are really opposing metrics, quite the opposite. A flexible engine or framework will allow you to do things you know fit your requirements the best.

I think the trade-off is more about performance versus ease of use. Many engines and frameworks like Corona or Cocos2D are rather easy to use, and for many simple games that’s exactly what you want. However, if your plans are more ambitious you’ll soon run into performance problems on mobile devices. Such engines lure you in by you not having to know about things like GPU stalling, texture fetch performance and so on. Without that knowledge, and without being able to change those things in the engine, you might be shit out of luck when your game runs at a nice two frames per second.

Now, libgdx doesn’t need you to know any of those details either. But we like to think that by design we make the user learn more and more about what makes their game slower. In addition, it’s damn easy to fix things up once you identified your problem, without having to change the actual framework code itself.

**Brownlee:** How did you design the engine API?

**Zechner:** The low-level APIs were designed in a structured manner, by observing the platform APIs we wanted to target and merging them in a sensible way. That design actually held up pretty well, even with the integration of our HTML5 back-end.

Higher-level APIs grew more organically, based on needs, but with generality in mind. Good examples are all the OpenGL utility classes like meshes and textures. We like to think that those are actually pretty well done, especially considering that they work seamlessly with fixed function OpenGL ES 1.x as well as 2.0.

Since Nate joined the project, lots of the APIs were designed in lengthy Skype talks. It’s not TDD per se, but it has some resemblance in that we first write some code that shows how to use the new APIs just to get a feel of what we are missing or what doesn’t fit. From that we then derive the actual API. We iterate on that for a few days until we push it to trunk and let other people to test it. Based on their feedback we might change a few things around, after which we consider the API final. Bigger API additions will trigger a release at that point.

A crucial aspect in all of this is having nightly builds. This lets the community test new or changed APIs in advance and give us valuable feedback.

**Brownlee:** Which of graphics, physics and input handling were more of a challenge?

**Zechner:** File system abstractions were the biggest pain. Graphics was
rather easy, physics was just tedious due to our JNI wrapper being manually written (not entirely true anymore).

I had nightmares with the HTML5 back-end just due to the nature of Google Web Toolkit, which allows us to compile Java to JavaScript. GWT is an incredible piece of technology, but it has a few warts that make working with it rather tedious, e.g. compile times, missing Java features like reflection which we hacked into it.

**Brownlee:** What is your approach for staying abreast of changes to the underlying technology and device APIs?

**Zechner:** Desktop systems don’t change a lot. Our desktop back-ends only expose OpenGL ES 1.x and 2.0 - new developments like the recently release OpenGL 4.3 standard are of less concern to us.

Interestingly enough, on mobile, the number of things we had to change over the last few years in terms of new APIs were rather limited. The only thing that comes to mind are the multi-touch APIs on Android that were released with Android 2.0 IIRC. Not a lot has changed since then.

We usually stay up to date by reading release notes of the various systems and picking out those features that are interesting for us.

**Brownlee:** What is one big element of the game engine you wish you could go back and do differently?

**Zechner:** Sometimes I wished I had written it in a different language. Java’s portability is kind of a problem for us when we look at platforms like iOS or Windows Phone 8. We found a solution through Mono, so it’s not all that bad. Java as a language is also not exactly amazing. But it’s ecosystem and it’s beginner friendliness (for a statically typed language that is) are still a reason why I think it’s not the worst choice.

### 1.0.4 Maintaining Your Game Engine

**Brownlee:** Do you currently think of the game engine as a project or a product?

**Zechner:** I still view it as a project, given it’s OSS\(^5\) nature. Nate just recently sent me something along the lines of “if you don’t value your work, neither will anyone else” and related it to the fact that all the work I do for libgdx is free. I don’t see it like that. I value my work, but not in money.

I enjoy the fact that tons of people use our code, creating beautiful things on top of it. I like talking to people in the community, hanging out in IRC. So, keeping libgdx free and viewing it as a project makes sense to me. We still like getting donations though, our servers that build the nightlies and serve the web site cost real money.

\(^5\)Open Source Software
Brownlee: What development tools do you use in a typical week?
Zechner: Shell, Eclipse and Jenkins. My trusty Chrome to look up docs unless I have them in form of JavaDocs, integrated in Eclipse.

Brownlee: How many people are currently working on the game engine and how is the effort structured?
Zechner: We have about two dozen contributors that can modify the Wiki. Another dozen or so can commit to the trunk. Of those, about 3-4 work actively on the project’s code at any given day of the week. Nate and myself are the ones contributing the most. Everybody that is not me or Nate is usually assigned a specific task or area to work on. We give people full freedom in how they accomplish their tasks, if they follow the rules we have. For bigger features or API changes we usually have short discussions, either via IRC, a Google hangout or Skype.

Brownlee: What are some common maintenance tasks in a given week?
Zechner: Answering as many questions on the forums as possible, reacting to e-mails from the build server (which has been silent for quite some time), checking commits by others, checking and resolving issues on the issue tracker. When I have time I work on new features.

Brownlee: Who wrote the documentation for the game engine?
Zechner: JavaDocs are written by the person who wrote most of the code of a class. Video tutorials were mostly done by myself. The Wiki is a collaborative effort of all our contributors. So it’s a mix of community and maintainer driven.

Brownlee: What are the core benefits in having a development community around your game engine?
Zechner: More test devices! In all seriousness, having a large number of people test all the code paths of your code is a good thing. We now have close to 1000 issues of the tracker (with only 50 open issues!), many of which we’d have never discovered ourselves. Many people also send in patches to fix minor bugs or contribute entirely new APIs. Additionally, the community helps newcomers in the form of forum posts or by writing tutorials themselves.

Brownlee: How do you generally engage the community?
Zechner: News is announced on the blog and twitter. Day to day communication happens through the forums and IRC. A couple of times I’ve held a poll on how to move forward with a specific feature or API on the blog.

Brownlee: What are some ways that your interaction with the develop-
ment community around your engine have influenced features or the direction of your engine?

**Zechner**: Small feature requests here and there, e.g. giving more control over Audio, adding a new method here, changing the signature of something there. We didn’t exactly have big API changes due to community requests. It seems most people are happy with what they get.

**Brownlee**: How do you manage the problems around remaining backward compatible?

**Zechner**: We have releases, each have a certain amount of bug fixes or after we changed an API. That way people can stick to whatever worked for them. We however do not back-port fixed bugs to older releases.

**Brownlee**: How do you structure the release process?

**Zechner**: Rather organically. Sometimes Nate pokes me and says it’s time for a release, other times we fix a critical bug that needs to get out there. It’s mostly a gut feeling. Lots of people work with the nightlies or directly from trunk, I gather our API is stable enough that people even trust it with production code, so releases are not extremely crucial.

**Brownlee**: What processes do you use to manage code quality in your releases?

**Zechner**: I usually run through all our tests manually, since we do not have an automated test suite. Unit and integration tests work splendidly for non-interactive and non-graphical things but break down fast if you want to cover a game development framework’s API. The good guys at Gemserk started to write a few unit tests for things that can be automatically tested.

**Brownlee**: What features would you like to add to your game engine over the next twelve months?

**Zechner**: iOS support is almost complete, but still a little rough around the edges. I plan on writing an FBX importer so the dreaded 3D format nightmare is finally over. This also means reworking the 3D skeletal animation code and finishing up the 3D API for lighting, materials and rendering. Kalle stated. Those are the goals until the end of the year. I usually don’t plan further ahead than a few months.

**Brownlee**: On which other game engines do you keep a close eye?

**Zechner**: Unity is always fun to watch, they are doing some incredibly awesome work. I’m occasionally in contact with a few guys on the PlayN team, especially Michael Bayne who’s the number one reason IVKM now runs with MonoTouch, a necessity to run Java on iOS. We collaborated shortly on IKVM, he did the hard bits, I added JNI support. I recently started having a look at MonoGame, which seems rather promising as well.
It comes with the problem that you need to buy $400 licenses for each mobile platform, but other than that it seems to shape up fast.

**Brownlee:** What are some features of your marketing strategy for the game engine?

**Zechner:** I occasionally speak at bar camps, workshops and conferences where I mention libgdx. Other than that we do not actively market the engine since we don’t make a living off of it.

### 1.0.5 Getting Started with a Game

**Brownlee:** What would you suggest to a developer looking to start making a game with your engine?

**Zechner:** Check the online documentation\(^6\).

Watch the introduction video, then walk through the simple application on the Wiki. From there, either read the rest of the developer guide, try out the demos and tests in our code repository or follow any of the third party tutorials we link on the page above.

**Brownlee:** What suggestions do you have for developers working with your engine daily?

**Zechner:** The GPU guides by Qualcomm, Imagination Technologies and Nvidia are a nice read if you want to gain more insight into how a GPU works and what it likes best. Sadly, there aren’t really any good OpenGL ES books out there.

I’d always recommend joining IRC, following me on twitter and subscribing to the RSS feed on the blog to stay up to date with the latest developments.

**Brownlee:** What are some game ideas you have had but do not have the time to work on?

**Zechner:** I love real-time strategy games, I wrote a multiplayer RTS called Quantum 2 years ago. I have considered making it mobile friendly. There are many more ideas in the back of my mind, but I don’t want to bore anyone to death.

**Brownlee:** What do you think are the required skills for building a great game?

**Zechner:** For building a game, knowing how to program or work with something like Unity or Gamemaker or knowing a programmer is a must. Having ties to a good artist, that can work with and knows about the

\(^6\)Online Documentation [http://libgdx.badlogicgames.com/documentation.html](http://libgdx.badlogicgames.com/documentation.html)
restrictions of game assets is also a must. Having a sense of what’s already out there helps coming up with new things.

But how to make a “great” game? I guess I’d be rich if I knew.

1.0.6 With a Successful Game Engine

Brownlee: How much revenue have you generated from your game engine?
Zechner: None, it’s a non-profit OSS project.

Brownlee: What are some opportunities that you have received because of your game engine?
Zechner: A was approached by a couple of publishers, eventually signed with Apress to write “Beginning Android Games”. I got a few job offers from various mobile gaming startups in San Fransisco and London, worked for one as the teach lead for about 8 months, but realized that I’d rather work on things on my own. I still get the occasional job offer. Maybe I’ll bite again some day.

Brownlee: What elements make for a successful mobile game engine?
Zechner: That depends on what you define as successful. For me, it’s important to have a good community supporting the framework, and I think we have that. Speed and features can be easily added by a single person, a supportive community has to grow. I tend to think we have both.

Brownlee: What tactics do you think got you over the line when so many other software projects fail?
Zechner: Persistence, I just didn’t stop what I was doing. Having people join the project also kept me motivated. Finally, the growth of the community and all the stories people tell about the development experience make it a worthwhile spare time project for me to keep alive.

Brownlee: What is the most difficult module or sub-system when developing a game engine?
Zechner: It’s the combination of modules that’s difficult, I don’t think a single module itself is all that hard to create. I guess that’s the reason why you mostly only see tech demos instead of fully polished games.

Brownlee: Would you do it all again?
Zechner: Yes, I have no life.

Brownlee: What would you do differently if you were to start the project from scratch today?
**Zechner**: Make it easier for a community to gather. This means having a centralized place to go to, provide tools to easily contribute, e.g. an open Wiki, and so on.

In terms of coding, I guess I should have been more diligent in the beginning which would have kept me from the major refactoring I did over the first year.

Apart from that, I’m mostly happy how things have developed.

**Brownlee**: How does it feel to have a game engine that developers are using to build and release games?

**Zechner**: Good.

**Brownlee**: What is one thing about your game engine with which you are not happy?

**Zechner**: Documentation could be stronger. We are doing well compared to other offerings out there, but we are still not top-notch.

Finding a way to automatically test hard to test things would be amazing. I trust our code, but having something showing others that they can trust our code as well is better.

**Brownlee**: What advice do you have for a developer thinking of creating a mobile game engine?

**Zechner**: Ask yourself what platforms you want to target, what genre(s) you want to cover. Then look around whether you find something that already covers your needs. If not, find something that makes it easier to build your specific engine, e.g. a lower level framework that abstracts away platform differences. Then write your engine. Enjoy figuring out all the platform specific bugs others already figured out before you.

**Brownlee**: What are some must-read books and resources for a developer interested in creating a mobile game engine?

**Zechner**: I wouldn’t know of something specifically for mobile game engine’s. I think my book does a good job of getting people into game programming on Android. I wouldn’t recommend it to someone who wants to write an industrial strength engine, since it’s more geared towards beginners.

Apart from books, I’d recommend checking out the best practices guides for the specific platform, Apple has multiple of those, Google has one or two on performance.

### 1.0.7 On Games

**Brownlee**: What are some games on which you have worked?

**Zechner**: Newton and Quantum are the two games I released. One is
a puzzle game inspired by gravity pods, the other is a multi-player RTS inspired by Eufloria (and born out of the frustration that Eufloria had no multiplayer mode). I worked on PennyPop in San Fransisco, a mobile game portal including a virtual world. I worked on a few small scale games for Bertelsmann. I also worked on many many unfinished prototypes.

**Brownlee:** Where do you think mobile gaming will be a few years from now?

**Zechner:** I’d hope that eventually the devices get powerful enough so I can use them as my main system, just as I use my laptop over my desktop PC now. This means it should be easy to transform it into a console or work machine. I should be able to easily hook it up to a monitor or TV, plug in one or more controllers and play. Or add a mouse and a keyboard and work. There are already a few things that come close to this, but the effort you have to put in to make it all work is still to high. Something integrated, dare I say Apple-style, might just be what finally brings together mobile and desktop/console gaming.

**Brownlee:** What is your favorite game and why?

**Zechner:** Cliche, but I still play Super Mario Brothers, like once a year. It just doesn’t get old.

Of course there are a lot more games I kinda worship. Z by the Bitmap Brothers for example, was just released on Android. Many other old DOS day games, all things id Software, Valve and Blizzard (apart from World of Warcraft). It’s hard to point at a single game and say “that’s the one I’d take with me on an island”.

**Brownlee:** What are some trends you are seeing in mobile gaming?

**Zechner:** It’s getting harder and harder for small developers to hit it big. The market is now covered by the likes of Zynga, Gameloft, Glu, even EA to some degree. It feels a lot more professional, and there’s a clear trend towards massively increased graphics quality. Just check out any games Unity uses to advertise it’s mobile licenses. This trend will clearly continue, mobile is where the money is. Let’s see how long it can be milked by the big guys.

**1.0.8 Final Questions**

**Brownlee:** Any final words on mobile game development?

**Zechner:** It’s a great way for people to get into game development. It’s easy to distribute your app to everyone (on Android), there are tons of nice engines and frameworks to chose from, and there’s real incentive to invest time into learning the required skills to maybe make some money. In a sense it reminds me of the 80’s garage game dev times. With the big
boys jumping in that time might be kinda over already, but it’s still a lot of fun. I just wished there was a way to have kids program on their iPads or Transformers directly, in a way that’s transferable to other systems.

Brownlee: How can readers best get into contact with you?
Zechner: Twitter: @badlogicgames Blog: http://www.badlogicgames.com

Don’t you dare send me an e-mail.
1.1 More Information

This was just a sample from the book “Mobile Game Engines: Interviews with Mobile Game Engine Developers”. For more information please visit http://mobilegameengines.com