Background

I've always planned for there to be mobile apps for Bot Land, but I didn't know exactly how I was going to accomplish it. This document talks about different approaches that I can take and will hopefully eventually contain prototyping/research information.

Reasons to even have a mobile version

I think a lot of this comes down to a gut feeling since I don't have any Bot-Land-specific mobile research (other than this Strawpoll): http://www.strawpoll.me/14748349

- Gameplay style
    - Bot Land is a casual game wrapped in a hardcore learning curve. I think that almost every aspect of it lends itself well to mobile gaming.
    - The length of matches within Bot Land are especially conducive to mobile play sessions. Most matches will last a minute or so in total and you can play a bunch of them.
- There are a lot of gamers I could target by having mobile apps. Apparently 51% of the world's Internet traffic is done from a mobile device.
    - Even on Twitch, it seems like 20-30% of my viewers are watching from non-desktop platforms.
- There are many people who have tried playing Bot Land on mobile or who have asked about it. I think that if I didn't have mobile apps that it would be very hard to convert these kinds of people into players if they only got a message saying "sorry, you have to play this from a computer".
- I don't know if any web games (even the ".io" games like slither.io or agar.io) have gotten *super* popular. They've definitely had at least hundreds of concurrent players, but how many players keep returning?
- Playing on mobile can be more social too, e.g. sharing app installations ("have you heard of this game?"), seeing someone else play it on a bus, etc.

For me, these reasons are convincing enough to want to make mobile apps.

Reasons to have an app in the app stores as opposed to just a web site

- Discoverability
- Ratings/reviews
- Reach

Requirements from a mobile Bot Land app

- Performance - the game should run at least 30 FPS

- Fullscreen - the game shouldn't have any browser chrome around it
- Code base - I would like any mobile-specific code to be relatively contained rather than being a completely separate codebase. For example, making a native app from the ground up in Kotlin or Swift is out of the question given the resources I have.

These are explicitly *not* requirements:

- Getting 100% of the payments from players. I don't mind if Google or Apple takes their fair cut for listing an app in their store.

Mobile scenarios

I expect people to play Bot Land on mobile *almost exactly* how they would on the web with one exception: scripting. Even if I make the greatest interface in the world, I don't think people are going to want to write scripts from scratch on mobile. I expect most players will fit into one of the following categories:

- Making scripts on desktop and never modifying them on mobile
- Making scripts on desktop and slightly modifying them on mobile
- Using premade scripts from mobile

I do expect people to watch replays, buy items, attack other players, edit their defense, etc. from mobile.

Electron (for desktop)

This doesn't fit into mobile, but it keeps coming up on-stream, so I want to address it here. Regardless of my mobile plans, I will almost certainly end up wrapping Bot Land into Electron and distributing it on Steam. The high-level reasons why:

- Access to all of the Steam playerbase
- Ratings/reviews in the store so that people can trust Bot Land

As for the particulars about using Electron or being on the Steam platform, I haven't looked into this and I don't know if it's necessary for launch, but these are the kinds of things that I would want to do eventually.

Approaches

9:57 HiDeoo: Well regarding performances, for everything you listed except "game engine" & "RN", all performances will be equal to the performance of your game using webGL, nothing else

[X] Just implement responsive design

No matter what mobile solution I end up going with, I'm going to need to spend time changing the UI so that it's mobile-friendly (e.g. fewer rows for things on some screens, completely different layouts on others). This is non-trivial in itself, but it's possible that I *only* do this work and just have people go to play.bot.land to play the game.

Conclusion: this is probably not a good idea since there's not too much difference between doing this and making some kind of app or more reconcilable approach for mobile users.

[X] Progressive Web App (reference)

You can pin sites from Chrome on Android by clicking the three dots at the upper right and choosing "Add to Home screen". After that, you get a full-screen experience without any browser chrome.

Also, you can prompt your users to do this with an install banner.

There's a brief tutorial on PWAs here.

Support on Safari is apparently not great. HiDeoo says you can't have a "real" PWA. To achieve this, you would probably need a native-app wrapper. Note that service workers are fully implemented in the Safari preview, and the manifests are in development, meaning this *will* happen eventually.

It sounds like I could do this for Android and have a web-view wrapper for iOS. Then, when iOS fully supports PWAs, I could just remove the web-view wrapper.

12:46 KubaTheBest: @Adam13531 For PWA to appear in apps list it needs to use HTTPS, Service Worker, Web App manifest and your browser needs to agree (based on your usage of the PWA). Source: https://developers.google.com/web/fundamentals/app-install-banners/#what_are_the_criteria. As a bonus of creating PWA, Microsoft can add your PWA do their Windows Store, in case people will start using their store someday (sorry Microsoft).

Examples:

- https://spacerage.hmans.io/

Questions:

- What do you do if you don't have Chrome on Android? How would you pin a site?
    - Need to figure out if Firefox can do this.
- How important will code-splitting be?

    - Apparently Lighthouse is a validation tool that Google provides for catching basic problems like load time, HTTP vs. HTTPS, etc., and it won't validate if I don't split the code up for login. Right now, bundle.js is just a massive blob that's needed to even be able to log in.

Downsides of using PWAs:

- I don't get all of the appstore "goodness" like ratings, reviews, familiarity, discoverability, etc.
- I think that users are going to be bound to a particular browser, meaning I'm not sure what's going to happen if you try using Chrome on iOS or Firefox on Android. I'll have to look into that eventually.
- I'm not sure if there's a way to remove the browser chrome or force landscape mode on the first visit or if the game needs to be launched via the homescreen for that.

Conclusion: users just aren't used to PWAs (as of 2018). This means a couple of things:

- The user has to pin the app to their home screen and then launch the app through that pin. Until then, I can't necessarily control the orientation of their screen, so I have to pop up a message saying that the game doesn't work in portrait mode.
- When someone wants to download a game, they don't think to go to a website right now; they go to the app store. If they don't find it there, I think they'll just give up.

Web-view wrapper app (more specific Android notes here)

On iOS, it's a lightweight version of the Safari webview. On Android, it uses whatever the default webview renderer is, which is usually the engine that Chrome uses ("blink").

HiDeoo: Adam13531 Just checked, on Android, as long as you're the owner of the website you're wrapping, it's allowed even tho PWA make it easier than a native wrapper for a website https://play.google.com/about/spam-min-functionality/spam/webviews-affiliate/

HiDeoo says that asset-loading shouldn't be as big of a problem as I'd originally thought. I currently load "assetmap.json" from the web, then I use that to point at all of the other resources. HiDeoo says that "[...] you don't have to care about that, in your WKWebView or Android equivalent, you intercept with the handler they provide all requests for example with assets/* and if it match, you cancel and serve the bundled image in your app".

- Pros
    - I get a "real" app that can be downloaded from the store. This means I get reviews, ratings, discoverability, ease of installation, ease of updates, etc.
    - Relative to some of the other solutions, this one may not be too hard to do.
- Cons
    - iOS store acceptance criteria - "Your app should include features, content, and UI that elevate it beyond a repackaged website. If your app is not particularly useful,

unique, or “app-like,” it doesn’t belong on the App Store." (reference). This means I may need to put in some work that would otherwise be pointless just so that I can be in the store. However, some people report that they’ve had to do very little to get accepted into the App Store (or nothing at all: mlnck: All meteor (https://www.meteor.com/) exports do is wrap the page in a webkit view. I've had to send more than 1 to the app store that was approved.). Given that, I’m leaning toward this as a solution right now: add a single native feature so that we can say we followed the ToS, but don’t spend a lot of time or money on it.

- Or maybe, if I can talk to a human in the app-curation process, I could argue that just having it be full-screen is something that’s not possible via PWAs on Safari yet? HiDeoo says you need native notifications, in-app payments, or something else given his experience at work (where their apps would get accepted but then later rejected).
- Could look into this more via these resources: https://developer.apple.com/contact/
- If I’m going to go the route of push notifications, I need to have a server / service running somewhere. I could use something like the free tier of this. Keep in mind though that OneSignal sells your data to advertisers on the free tier ("Best of all, OneSignal is a free service that supports unlimited devices and notifications. OneSignal makes money by selling data to advertisers and research companies. We also offer paid service options for clients that require increased data privacy.").
- [09:13] darkysharky: another feature that you could use to pass the bar is Universal Links (be able to link to a place in your app)

- May need a way to play without logging in
  - [09:33] thesheff17: @Adam13531 just a heads up...apple rejected our app a bunch because it didn't have a guest mode to play
  - [09:34] thesheff17: its a casino app
  - [09:37] thesheff17: I think games have other rules...its all up to the apple reviewers to say how the rules are interpreted ...also we just faked it enough so you can play a little. It doesn't have to do much.
  - Note that there are *tons* of applications that require a login at startup, e.g. Uber, Facebook, Venmo, etc.
- Processing payments
  - It would take work to support another method of processing payments (i.e. the stores themselves)
  - By processing payments through their stores, the respective companies would get a cut of the money

Conclusion: originally, I wanted to wait for iOS to just support PWAs since I figured having a PWA would affect all platforms and I wouldn’t have to do anything special. However, I still think user expectations include “games are in the app store and nowhere else”.

[X] Native apps (i.e. using Swift and Kotlin)

I don't see this happening. This violates one of the requirements of having mostly a single codebase, and I don't have the resources to write every new feature on three completely different clients. There would also be a ton of ramp-up time for any of this so that I could pursue this for even one of the main mobile platforms.

Conclusion: not doing this.

[X] Game engine

This is roughly the same to me as having native apps. Even if I took something like Unity which I've already spent a couple of months using, I'd need to learn the proper way to do everything. I'd still have two completely separate codebases. The aspect that kills this idea the most for me is reimplementing the UI; it took me long enough to make the UI the first time around with HTML/JS/CSS, and I feel like that framework is made specifically for UIs. I can't imagine doing something quickly in Unity using whatever UI approach they have that will work on all platforms.

I could try to rely on whatever web publishing the game engine has so that I ditch my current client code entirely. There are definitely some pros to doing this, but it would involve a rewrite of the entire codebase. Also, it would split the server/client codebases into multiple languages.

Conclusion: not doing this.

[X] No mobile app at launch (i.e. deferring mobile until later)

This is obviously straightforward, so I'll just list pros/cons:

- Pros
    - I would gain many months to just focus on all of the things I'm going to have to do anyway: polish, content, marketing, and features.
        - Note that even if I'm just making a web-view wrapper or something along those lines, I still need a ton of CSS for making the UI mobile-friendly.
    - If the game is popular enough, I could devote more resources to doing mobile "correctly" (whatever I determine that to be).
        - I could also inform the mobile decisions based on how players are interacting with the web version.
    - If the game isn't popular enough, I wouldn't have spent time working on mobile.
- Cons
    - Maybe the game won't be popular *because* there's no mobile version, so I may have shot myself in the foot and then I maybe wouldn't be able to pivot.
    - I wouldn't have mobile apps at launch, so I'd be losing all of those potential players.

Conclusion: I think it's important enough to launch with mobile versions as outlined at the beginning of this document (especially if having mobile versions doesn't cost too much time).

Technologies

If I'm leaning toward a thin wrapper for iOS and a PWA for Android, then I probably don't need any of these things.

[X] React Native (reference)

What I'd gain out of this is a real native app. I think this would be pretty compelling if I weren't making a game, because all of the UI would have to "feel" native. I don't care so much about that with Bot Land as long as everything is intuitive.

What I sacrifice is every bit of rendering logic that I have right now. Some of it can be very easily salvaged, e.g. most of the and stuff that I have can be converted into <View> with React Native. However, libraries that use the DOM like Blockly or CodeMirror would be a nightmare to remake. I could ditch those entirely by just not having script editors in the game (which would create other problems around players' UI expectations), but I'd still need to figure out what to do about a canvas for WebGL.

All in all, this would probably be a *ton* of work for no real pay-off. If I'd had this all in mind from the beginning, it would have been easier to transition, but I think it would have been impossible to have coded anything for the web since I wouldn't have PixiJS, Blockly, CodeMirror, or any other rendering-related library (e.g. IntroJS) unless there was already a React Native plugin for it. I think that rewriting any of those three major libraries would take at least several months on its own.

Finally, all of my components are styled with CSS, and I'd have to change to something like CSS in JS (like styled components).

Also, "React Native" and "graphics-based game" don't seem to go hand-in-hand.

10:39 HiDeoo: Adam13531 If you ever look into RN, this could be a great & small starting point as it compares React & RN, show the differences, what code would be different and show that you can have platform specific code in RN, like component/ Arena.js , component/ Arena.ios.js , component/ Arena.android.js , etc.
https://medium.com/@alexmngn/from-reactjs-to-react-native-what-are-the-main-differences-between-both-d6e8e88ebf24

Conclusion: not worth it to pursue this.

[X] PhoneGap (reference)

According to their FAQ, PhoneGap is like the open-source version of Cordova. It's used to build cross-platform mobile apps using only HTML/JS/CSS.

Conclusion: see Cocoon (I didn't research this too much)

[X] Cocoon (reference)

Directly on Cordova's site, they say this about Cocoon:

"Cocoon is a Cordova based cloud service for building native HTML5 apps and games. Cocoon is focused on providing the best webview engines and features like Canvas+, JS encryption or a custom Developer App."

I can't find a whole lot of information on Cocoon. Some resources I did find:

- (reference) - Forums post from 2014 talking about how WebGL support from Cocoon doesn't seem to be great
- (reference) - A games showcase on the official site - the ones I looked at have only mobile apps, so I didn't try any out

Conclusion: *maybe* this can help, but I haven't found enough details online to compel me toward this. I would want to know exactly how this helps, what the pain points are, etc. Also, this doesn't really fit in with my PWA plan that I have now.

[X] Cordova (reference)

This can be used to take a web app and turn it into mobile apps.

Conclusion: see Cocoon (I didn't research this too much)

Prototyping

Framerate needs to be acceptable on these devices and I'm likely not going to do anything major to affect that, so I need to figure something out.

12:22 EveryJuan: my final food for thought on this - go through and optimize the heck out of all your draw calls, sprite switches, etc (I can get you a list) when it comes to webgl before doing anything on mobile so you don't get discouraged. My initial implementation on mobile was like 500 draw calls and my nexus had 5FPS LUL

9:34 HiDeoo: I mean before even thinking React Native, phonegap or w/e, it should be: is the renderer you're actually using & libs you're using to talk to that renderer even allow you to run your game on mobile at constant 30fps, if not can you achieve it or not at all and you would

need to switch to openGL ES vs webGL for example. I mean what's around the game is not really that important if the game can't even run properly in its current state.

Need to make sure the conversion itself will be manageable, so perhaps taking a small chunk of code and trying to convert it if that's how it'll look.

Buying a Mac

I need to buy some kind of Mac if I'm going to be doing mobile work at all.

Note to people who are suggesting getting a Hackintosh: I don't want to do anything illegal here, and apparently the only legal way to emulate iOS is to do so from a Mac.

I *could* use one of those rental services online where you pay for time on an iOS device, but I'd rather just own a Mac at that point.

As for which device to get, it sounds like if I'm on a budget, I should get a Mac Mini. If I want a device to use for just development or something, I could get a "real" computer like a Macbook.

Compatibility for the latest OS (reference):

List of macOS High Sierra 10.13 (last version) compatible Macs:

MacBook Pro – 2010 or later models
MacBook – Late 2009 or later models
MacBook Air – 2010 or later models
iMac – Late 2009 or later models
Mac Mini – 2010 or later models
Mac Pro – 2010 or later models

If I don't do a webview-wrapper kind of app, then I don't *need* a Mac in order to test things out, but it would greatly help for two reasons:

- Mobile debugging (apparently you don't need a developer license just to debug on a real device any longer). Also, apparently emulating was always free.
- Testing on Safari (on the Mac itself)

Marketing

The marketing plan for Bot Land may relate to my plan for mobile, so I think it's worthwhile to write some thoughts here.

High-level marketing plan:

- Make sure web and mobile versions are all available at once
- Finally leverage Twitch audience to help - spread news on social media, ask their favorite streamers to stream the game, tell their friends about it

- I'll reach out to game journalists, streamers, YouTubers
- I'll post on social media (reddit, hackernews, etc.)
- Steam
- Ads? I don't think I'm going to have the money for this unless I count preliminary salvage-pack sales from Bot Land. I may do a Kickstarter just for the marketing campaign and use the money gained from that for more marketing.
    - The ads would be for bot.land as opposed to directly playing the game. Bot.land would clearly reflect *how* you could play the game and on what devices.

I don't think having native apps really impacts this too much. If I DID have an app, I could have an advertisement link to that at the end instead of bot.land

Schedule

First week of January: flesh out this document and start to prototype and dive into more nitpicky points to research
January: figure out mobile strategy. By the end of January, I should have no more major questions about what I'm going to do going forward.

Overall conclusions

I think that going the PWA route is probably best for now. They're not fully supported on iOS/Safari yet, so I'll have to wait until Fall 2018 for them to hopefully be supported. If they're still not supported by then, that's okay, I'll just launch it when they're ready (unless it'll be >Q1 2019). If they say they're not going to ever have support by then, then I'll fall back to the webview wrapper on iOS.

The reason I want to go with PWAs is because the cost of implementing them is not too high and they meet all of the requirements from the beginning of this document.

The reasons I'm *not* going with proper/native apps despite recognizing some of the benefits they confer:

- Cost
    - The cost of making an iOS app would be too high (probably around a month of time, but even that depends on getting accepted into the store).
    - The cost of making an Android app is probably not very high, but there's still some amount of work to do there that would probably take at least a week.
- I don't think the benefits are so great.
- I don't think that having native apps fits into my MVP. I think that having a mobile solution is good enough for the MVP even if it's not the *best* mobile solution or the one that people expect. I can always add native apps in the future when I know that it's a high priority. Also, PWAs will prove that webview-wrappers will be performant enough.

This conversation in Twitch chat from 1/3/2018 kind of sums up how I feel about native apps:

> 9:42 swattkidd: But you are right man its not worth the effort. and if it is the audience will demand it. You have a restaurant with a menu - get customers in the door and then if people demand a new dish enough, add it to the menu lol
> 9:44 EveryJuan: I look at is another way - what if BotLand doesn't catch on, but it would have if you did a mobile app
> 9:44 EveryJuan: :thinking:
> 9:45 swattkidd: @EveryJuan in that case I imagine at least a few people would demand an iOS app then adam can restart and relaunch as a brand new mobile only game

Regarding technologies like PhoneGap, Cocoon, and Cordova, I don't think they're going to provide much for me given that I'll just be bundling what I have on the web already. If I eventually want payments, notifications, etc., I can start to look into it then.

Also, given the downsides of PWAs (not forcing landscape mode and not forcing fullscreen until launched from the home screen, not allowing use of all browsers), I may actually change my plan to having webview-wrappers eventually.

Overall, the way I see it is that if Bot Land is going to work with a PWA, then it'll work with a webview wrapper, and I don't technically have to do any work toward either until the "end", so I can defer most of the detailed decisions until much later (e.g. when I have responsive web design).

Update (1/15/2018): I'm getting a *lot* of feedback saying that people don't expect to find good mobile games directly in the browser. I can *have* a PWA, but perhaps it shouldn't be the only thing at launch or even exist at launch if it's going to take time.

A bunch of people in chat essentially said that if they weren't watching the stream, they wouldn't even bother trying to play Bot Land in a mobile browser.

Rendering performance

Writing down some notes here about investigations.

Base iPad FPS on Adam account with default Edit Defense view: 17-20.

Passing in "legacy: true" to the renderer didn't seem to have much of an impact on iPad/Chrome.

Rendering tiles doesn't seem to negatively impact rendering performance that much. I turned them all off and it seemed to give the same FPS.

Cutting out shaders entirely seems to make the FPS increase by 25-50%.

Cutting out bots from rendering put the FPS at about 36.

This is a significant gain. What is it about bots that make them hard to render?

- Shaders
- Multiple textures
- Graphics (for life bars)

Cutting out EVERYTHING from rendering gives me 52-60 FPS on iPad. I did this by commenting out all of the this.stage.addChild calls in the client Arena.

If I add ONLY bots back in, the FPS eventually goes to about 21-25 FPS (which is so close to what it is by default with everything drawing that bots are the biggest portion to work on).

Rendering everything but shaders and bots gives me about 30-35 FPS. After a while, it goes up to about 40-45 for some reason.

Only rendering bots (no tiles or cpus or chips) but with no shaders gives 40-50 FPS.

Cutting out chips and CPUs eventually settles on 20 FPS.

11:53 clayman666: @Adam13531 i am getting around 15 fps on samsung s5 mini, also the screen is flashing constantly
11:56 clayman666: @Adam13531 shaders are off, same fps and screen is still flashing
11:57 clayman666: but then again my device is quite old

A bot itself looks like this:

- Container - this holds the whole bot to render
    - Container
        - Sprite - head sprite
    - Container
        - Sprite - torso sprite
    - Container
        - Sprite - foot sprite
    - Graphics - life bar
    - Graphics - shield bar
    - Text - name (when selected)

The reason why I have three separate sprites for head/torso/foot is so that I could eventually animate them individually if I wanted. I could maybe combine the graphics for life/shields, but the shield bar doesn't show unless shielded, and the text doesn't show unless selected

Conclusions

- Shaders are a no-go on mobile. They cut the FPS by at least 25% and I haven't really committed to them, so I could just disable them on mobile and be fine.
- I should only render life bars when a bot has taken damage. Probably the same with shield bars.
- I need to make it so that tiles aren't rendered individually.
- I need to try rasterizing bots so that there's just one draw call.
- I need to figure out a more performant way of highlighting tiles (for the crosshairs and for showing invalid placement)
- Need to figure out how to interact with getBounds less frequently.
- Set interactiveChildren to false on everything. Probably just make a function that wraps the creation of containers to set this to false immediately.
- When on mobile, allow for more execution time and CODE_EXECUTION_TIMEOUT when simulating attacks.

Ideas

Note: as of 1/12/2018, I culled the "bad" ideas or ones that I've already done, so the things remaining are potentially optimizations but that won't provide a huge FPS boost or may cost a lot of time.

Graphics like the red highlight for tiles don't change once they've been created, but I still render to them every single frame. I could cut down on that by caching it after I've created it once, but I don't know how much I'll save by doing that.

Consider destroying textures with a random delay so that a bunch of them don't get destroyed at once.

Even just doing this in SpriteComponent like this may address some stuttering during battles on slower devices:

```
setTimeout(() => {
    this.spriteToBeDestroyed.destroy({
        texture: true,
        baseTexture: true,
    });
}, _.random(1, 3000));
```

I have a feeling that MOST of the stuttering is coming from something else entirely though rather than rendering.

Update: freaktechnik suggests requestIdleCallback (when it's available)

9:38 freaktechnik: Hey adam, just looked at your mobile design doc and saw you deferring cleanup tasks with setTimeout 0, consider using https://developer.mozilla.org/en-US/docs/Web/API/Window/requestIdleCallback instead when it is available!

9:41 HiDeoo: Adam13531 On the same subject, you may have missed my previous message, it'll be available very soon (< 1 month) as a polyfill as it's core of the next version of React (https://github.com/facebook/react/issues/11330)

I could try object pooling for sprites still. Right now, when I make something like a laser or an artillery shell, I clone the base sprite. This isn't terrible since I'm reusing textures, but I still don't need to do the many allocations that occur for a Sprite:

- Sprite
    - ObservablePoint
    - Container
        - DisplayObject
            - Float32Array
            - TransformClass
                - Point
                - Point
                - ObservablePoint
                - Point
            - Bounds

I don't know what kind of performance boost I would get, but I imagine it wouldn't be too complex if I were to just modify cloneSprite to pull from a pool like how PixiGraphicsPool used to be for PIXI.Graphics before I deleted it on 1/12/2018.

RE: culling: I could use a strategy like the following:

Any entity already has a renderable property that's either false or true.

- If it's already true, then check every 5 frames to see if we should make it false
- If it's false, then check EVERY frame for whether we should make it true

That way, an entity that should be visible will ALWAYS be visible, and an entity that should be invisible will only perform the culling check every so often. This is all based on which operation is more expensive: culling or testing for culling. Depending on which one is more expensive though, we probably just always want to test or always want to draw. I would want to avoid entities NOT being visible despite them being in view due to only performing a check every so often.

I could set the spriteCache size lower on mobile. I think it was much more useful when I wasn't rasterizing sprites because I could reuse colored bot sprites. Now that I'm using RenderTextures for everything, I'm not really benefiting too much from the cache.

Another improvement could be to use the cache for rasterized sprites.

Not all of this is purely rendering performance; there's other stuff happening and I should try to decrease the load on the CPU as well as the GPU.

For example, I think getBounds is pretty resource-heavy.

If I don't rasterize bots, I can't just get rid of shaders and use tinting instead or else I run into the issue where bots have weird opacity overlapping.

Enforce a minimum zoom level on mobile so that we can cull everything not in the camera.

For optimizing the shaders, if I get rid of "if" statements and replace them with some purely mathematical logic, it would be better.