Trinity College Dublin

SCU33D06 - Software Design Analysis

Analysis and Contribution of an Open Source Project: Mindustry

Anna-Lena Rinke - 23377958
Julian Dezarnaud - 23359595
Tom Lavigne - 23377770
Cleomene Nelli - 20328320

Professor:
Anthony Ventresque

April 17, 2024

Contents

List of Figures

List of Tables

Listings

1 Introduction

On April 28, 2017, Anuke, a developer fan of gaming, participated in a Game Jam and produced a game that would impact his future years : ”Mindustry.” Mindustry is a hybrid tower-defense and factory-building game where the primary goal is to defend your base against waves of enemy attacks while also expanding and optimizing your industrial infrastructure at the same time. Players need to gather resources, build and upgrade various types of machinery, such as drills, conveyors, turrets, and factories, to produce ammunition, power, and other necessary items. The overall goal is to survive increasingly difficult waves of enemies while building an efficient production chain to support your defenses.

After getting very positive feedback from the people around him, Anuke decided to develop the game further to sell it via popular market platforms. Seeing that his game was doing well and to keep it alive, he decided to promote it as an open source project to build up a large community of developers and players who would help him maintain and improve the game. Today, Minustry has become a very large open-source project followed by thousands of people.

In this software design analysis project, our main aim is to assess the factors contributing to Mindustry’s longevity and success. To do this, we will start by examining the stakeholders of the project, namely mainly Anuken and his community and their relationship. Next, we will look at the overall composition of the game through different perspectives to understand how its different elements work together to create a game of this quality. This includes the social analysis of the structure and context, as well as the more technical analysis of the code. Then, we will assess the strengths and weaknesses of the project by computing different metrics like the test coverage and code complexity, as they tell us about non-functional requirements like the maintainability. Finally, after highlighting the weak points of the project, we will propose contributions including one in particular: the writing of an introductory tutorial for new contributors. By conducting a comprehensive analysis, we hope to gain valuable insights into the elements that have made Mindustry a successful open-source project and use these findings in future game development projects.

2 Stakeholders

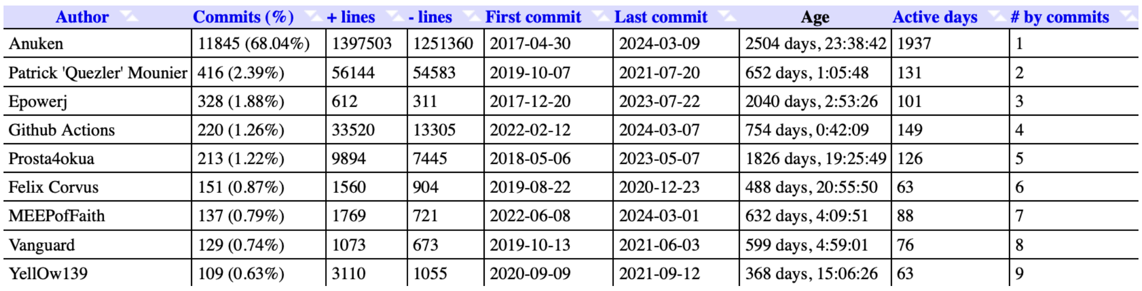
2.1 A one man open source project

The stakeholder analysis for the Mindustry project reveals a unique structure due to its nature as a one-man open-source project. The creator ‘Anuken’ of the project holds multiple roles, shortening the decision-making process and project execution.

Anuken officially fulfills multiple different roles that are typically distributed among several people in larger projects. These roles include the project developer, maintainer, communicator, support team, tester, and production engineer. This unification of roles allows for a clear, unified vision and consistent implementation, which is evident in the fact that Anuken is responsible for producing the majority of the code. This can be seen in Figure 1, which clearly shows that Anuken contributed 68% of the commits.

Anuken sold his product thanks to Steam and the iOS/Apple store, which handle payment transactions. This arrangement provides a reliable platform for users to access the project and ensures secure financial transactions.

However, In the case of assessors, suppliers, and system administrators, these roles are not applicable to the Mindustry project. This is due to the absence of specific standards or regulations to conform to, and the lack of need for backend servers or additional resources.



Author	Commits (%)	+ lines	- lines	First commit	Last commit	Age	Active days	# by commits
Anuken	11845 (68.04%)	1397503	1251360	2017-04-30	2024-03-09	2504 days, 23:38:42	1937	1
Patrick 'Quezler' Mounier	416 (2.39%)	56144	54583	2019-10-07	2021-07-20	652 days, 1:05:48	131	2
Epowerj	328 (1.88%)	612	311	2017-12-20	2023-07-22	2040 days, 2:53:26	101	3
Github Actions	220 (1.26%)	33520	13305	2022-02-12	2024-03-07	754 days, 0:42:09	149	4
Prosta4okua	213 (1.22%)	9894	7445	2018-05-06	2023-05-07	1826 days, 19:25:49	126	5
Felix Corvus	151 (0.87%)	1560	904	2019-08-22	2020-12-23	488 days, 20:55:50	63	6
MEEPofFaith	137 (0.79%)	1769	721	2022-06-08	2024-03-01	632 days, 4:09:51	88	7
Vanguard	129 (0.74%)	1073	673	2019-10-13	2021-06-03	599 days, 4:59:01	76	8
YellOw139	109 (0.63%)	3110	1055	2020-09-09	2021-09-12	368 days, 15:06:26	63	9

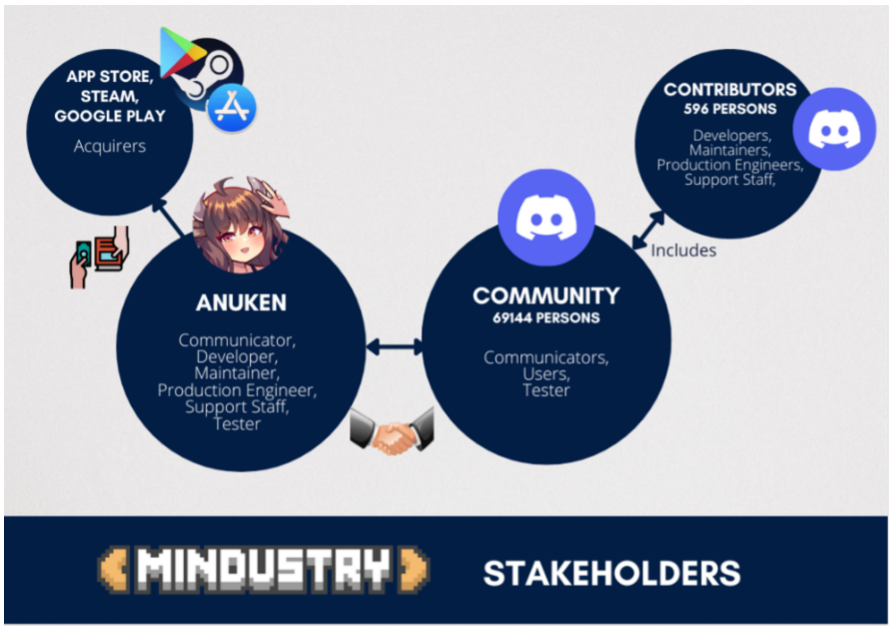
Figure 1: Listing of how much the different authors contributed to the source code. Made with gitstats

2.2 Impact of the community

After creating the root of the game, Anuken wanted to ensure that players make the game their own, so he wanted to add features from the community. However, this presents a substantial challenge in terms of workload. What better solution than an open-source project to ensure that players take ownership of the game and see it evolve, while at the same time doing as little as possible so that he can spend time on other projects? While Anuken is responsible for the majority of the work, his community is highly active and assists him in many tasks through GitHub and a large Discord server. The community, among other tasks, contributes over 30% of the code, proposes numerous updates, identifies and reports bugs, initiates the development of new features on their branches, and enhances support.

A closer look at Discord reveals its importance as a vital component of the project, with 69,144 members. The server has 34 separate channels dedicated to various topics such as design, development, adding features, modding and so on. A specific part of the discord is dedicated to modes, so that players can modify the game and even develop modifications to play as they see fit. Although English is the main language used in chat, there are also language-specific channels to facilitate discussions between contributors in their native language. The channels are very active, with over 300 messages exchanged every day and an impressive average response time of 4 minutes for 50 questions asked. The creator himself takes an active part in the discussions and responds promptly. This active participation can be attributed to the fact that many of the programmers are also players of the game, promoting a deeper connection and commitment to the project.

This community-centered approach enables real-life testing and feedback, replacing the classic suggestion system with a new one that invites users to collaborate and improve the game. By involving these users and taking their needs and comments into account, Anuken can guarantee the success of the project. He can then concentrate on maintaining consistency and planning new updates, rather than constantly fixing bugs and reviewing all suggestions. The impact of the community is evident in the number of contributors who have modified the code based on Discord recommendations, with 596 contributors to date. For example, in March 2024 alone, not counting mergers, 42 authors have made 78 changes to the master branch (including 24 by Anuken). The master branch saw changes in 109 files, with 3,964 additions and 2,423 deletions, underlining the important role of the community in the development of the project.



APP STORE, STEAM, GOOGLE PLAY
Acquirers
ANUKEN
Communicator, Developer, Maintainer, Production Engineer, Support Staff, Tester
COMMUNITY
69144 PERSONS
Communicators, Users, Tester
CONTRIBUTORS
596 PERSONS
Developers, Maintainers, Production Engineers, Support Staff,
Includes
MINDUSTRY
STAKEHOLDERS

Figure 2: Identified stakeholders of Mindustry

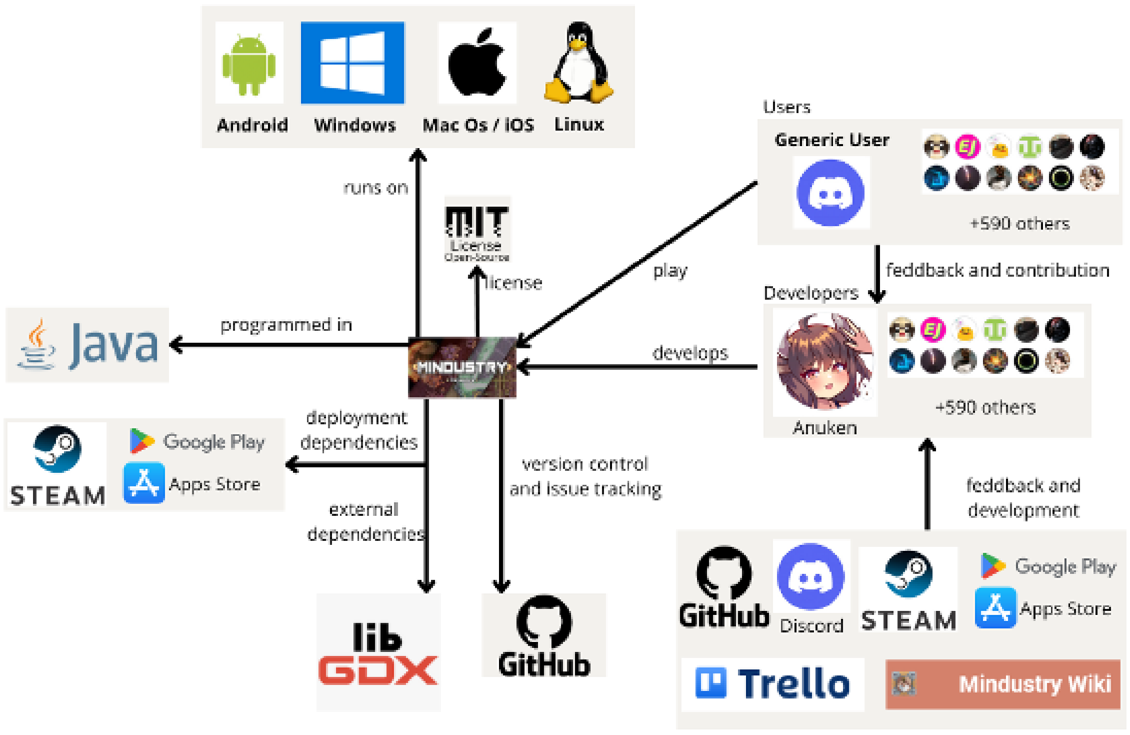


Figure 3: Context View of Mindustry

3 Architectural View

3.1 Context View

To construct the context view, which can be seen in Figure 3, I looked into the GitHub repository to get information about the number of developers and the main contributors of Mindustry. With the source code I could get the dependencies and with what was programmed this project. Inside the source code, there is already information in the ReadMe file about how the project works, but to truly see how they get feedback and proper contribution I looked into the Discord server.

Mindustry is available for a lot of different devices, allowing it to access huge amounts of potential players!

By creating this view it became clear that the whole project revolves around the Discord server of Mindustry. It's where everyone talks to each other about any issues, ideas or feedback. You can get information and help quickly and even get replies from the owner himself. There is truly a strong sense of community. Players and developers created a Wiki to quickly get knowledge about any foes or block of the games. The Trello is used to get the needed advancements and is used as a roadmap.

GitHub takes the role of version control, issue tracking and it's where the most trusted contributors will look at the commits of the developers.

But everything is centralized in the Discord. All the activity of the GiHhub can be seen in the "github" channel of the Discord, the advancement in the roadmap is in "announcement" and every other aspect of the code, ideas and activities about Mindustry have their own channels.

Mindustry has very little dependencies because even packages and collections are self-made. I found that it is really impressive for a game to be made up to 68% by only one man. Unfortunately, there is no documentation or further information about them. What I think is important to highlight, is this positive feedback loop from the contributors. As most of them are players, they truly have an idea on how to make the game better. Some ideas may be only for their own pleasure, e.g. doing a mod, but even these ideas can be a good starting point for bigger changes and can be integrated into the real game, making everyone able to take Mindustry a step further.

3.2 Development View

This UML Development View diagram, which can be seen in Figure 4, shows the software architecture of the Mindustry project. Packages have been grouped into three layers : Access Layer, Interface Layer and Core Layer.

- The Access Layer contains packages that provide access to external resources such as files, databases and network services. Here, however, it is very small, as the game contains no databases as such, only locally stored files and local servers.
- The Interface Layer contains the packages that provide a user interface for the game. All interactions with the user pass through the Interface Layer.
- The Core Layer contains the packages that provide the main game logic and functionality. This is the actual content of the game, whether it's the sequence of events or the game elements.

The arrows on the diagram represent strong dependencies between packages. Due to their high number, not all the classes and their interactions are represented in the diagram, as we wanted to focus on the global architecture.

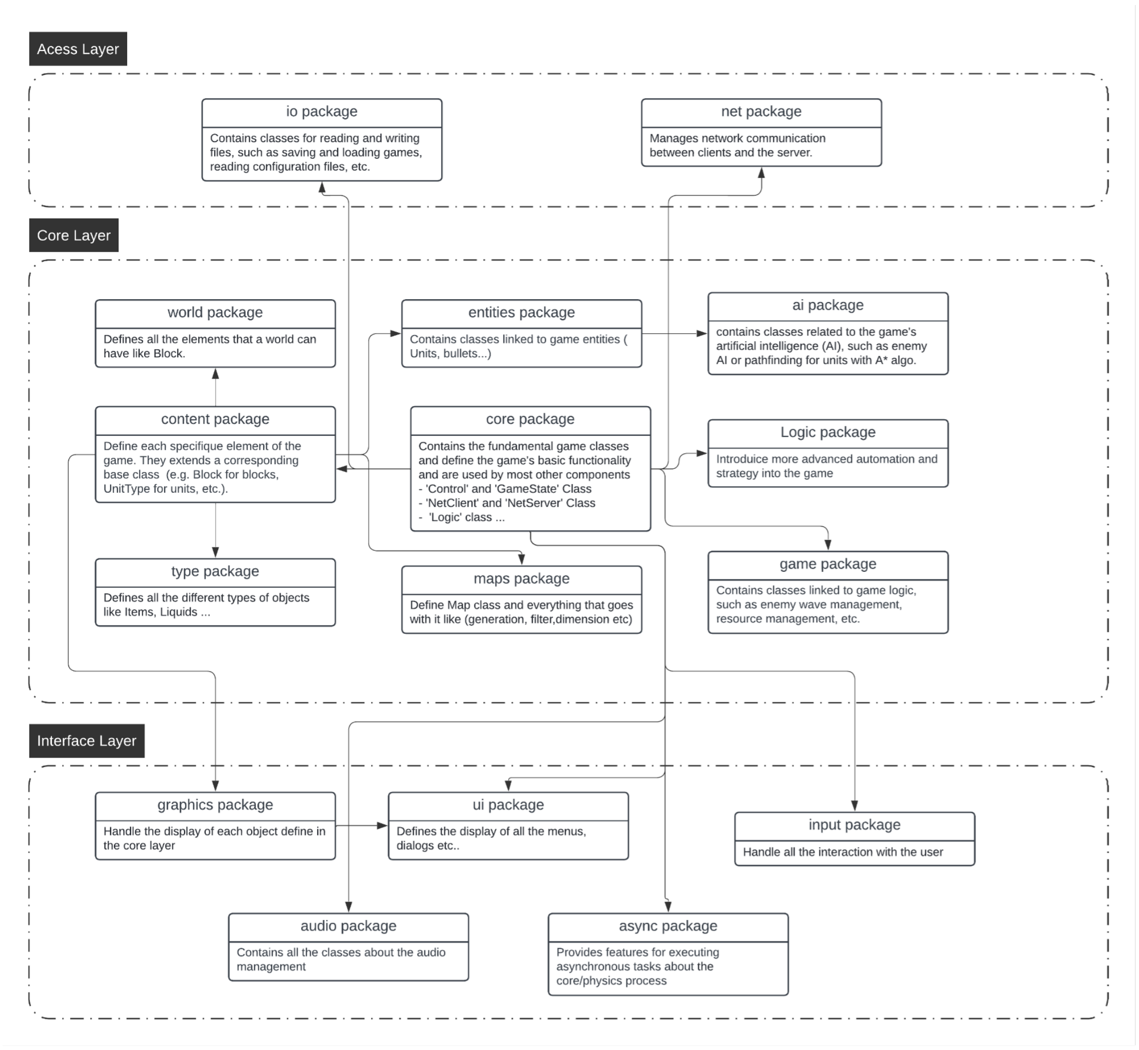

Acess Layer
io package
Contains classes for reading and writing files, such as saving and loading games, reading configuration files, etc.
net package
Manages network communication between clients and the server.
Core Layer
world package
Defines all the elements that a world can have like Block.
entities package
Contains classes linked to game entities (Units, bullets...)
ai package
contains classes related to the game's artificial intelligence (AI), such as enemy AI or pathfinding for units with A* algo.
content package
Define each specifique element of the game. They extends a corresponding base class (e.g. Block for blocks, UnitType for units, etc.).
core package
Contains the fundamental game classes and define the game's basic functionality and are used by most other components
- 'Control' and 'GameState' Class
- 'NetClient' and 'NetServer' Class
- 'Logic' class ...
Logic package
Introduice more advanced automation and strategy into the game
type package
Defines all the different types of objects like Items, Liquids ...
maps package
Define Map class and everything that goes with it like (generation, filter,dimension etc)
game package
Contains classes linked to game logic, such as enemy wave management, resource management, etc.
Interface Layer
graphics package
Handle the display of each object define in the core layer
ui package
Defines the display of all the menus, dialogs etc..
input package
Handle all the interaction with the user
audio package
Contains all the classes about the audio management
async package
Provides features for executing asynchronous tasks about the core/physics process

Figure 4: Development View

By analyzing this diagram, we can see that most of the dependencies are directed towards the Core Layer, indicating that the main game logic and core functionality are the most important elements of the system. We can also see that the Access Layer and Interface Layer have low dependencies on other packages and are quite small, making the system easy to maintain and upgrade.

3.3 Deployment View

To construct the deployment view, which can be seen in Figure 5, I used the source code and the Discord server a lot to get the needed information. This view is simpler than other Deployment views from other projects because there is no remote database server, everything is saved locally in files, and the multiplayer servers are self-hosted. What I found impressive is how this game doesn't need high requirements, making it accessible to everyone and scalable for every device, ranging from a computer to a phone. Mindustry can even Run on Raspberry devices! It shows that it was well coded, and that performance was not neglected. And it is important, because when you play you can quickly get to huge automated planets with a lot of props and conveyer belts to take care of, while being under attacks from big waves of foes.

Additionally, it doesn't have a lot of dependencies, making it more stable to change and more compatible to a lot of different and broad devices.

The self-hosted servers are a big thing, with each server getting their own fans and communities and so forth. They must be managed well by their creators, they create their own rules with their own mods, and they take care of potential cheaters. This is what can truly carry the games through the years like games such as Minecraft, Garry's Mod or Arma 3. You don't need a powerful computer to play or even host your server. This facilitates the creation of a community around the game and the potential coming of players. The game being performant, it doesn't have high bandwidth requirements, helping to get servers with high number of players. Everyone can put his hand on Mindustry, to play, to host it or even contribute to it by coding. Since there is only one main language and few dependencies, it's easier for new contributors and developers to start with it. But the fact that some collections and packages are self-made by Anuken is not helping, because he didn't do any documentation for it and there is no more information about it.

3.4 Other perspectives

3.4.1 Security and Performance

Regarding security, after talking with some developers in the Discord of Mindustry, it appears that there are currently 17 backdoors in the code. It's concerning because these backdoors can be exploited to gain unauthorized access to the game's systems or manipulate gameplay. The development team needs to address these vulnerabilities through thorough code review, testing, and implementing necessary security patches. But it's not a priority for them since there is no database with personal information or payment data which reduces the platform's attractiveness to potential attackers seeking valuable information. Additionally, the multiplayer aspect of Mindustry is solely self-hosted, and thus reduces security concerns about centralized multiplayer servers. However, it is important for players hosting their servers to implement proper security measures. The only concern for them is people cheating to win easily and progress further in multiplayer servers.

Otherwise, the Java and OS are secure, data management-wise, so it won't crash your computer or mine crypto. For the identification, UUID (Universally Unique Identifier) and USID (Unique Server Identifier) are being used for each server, which adds a layer of

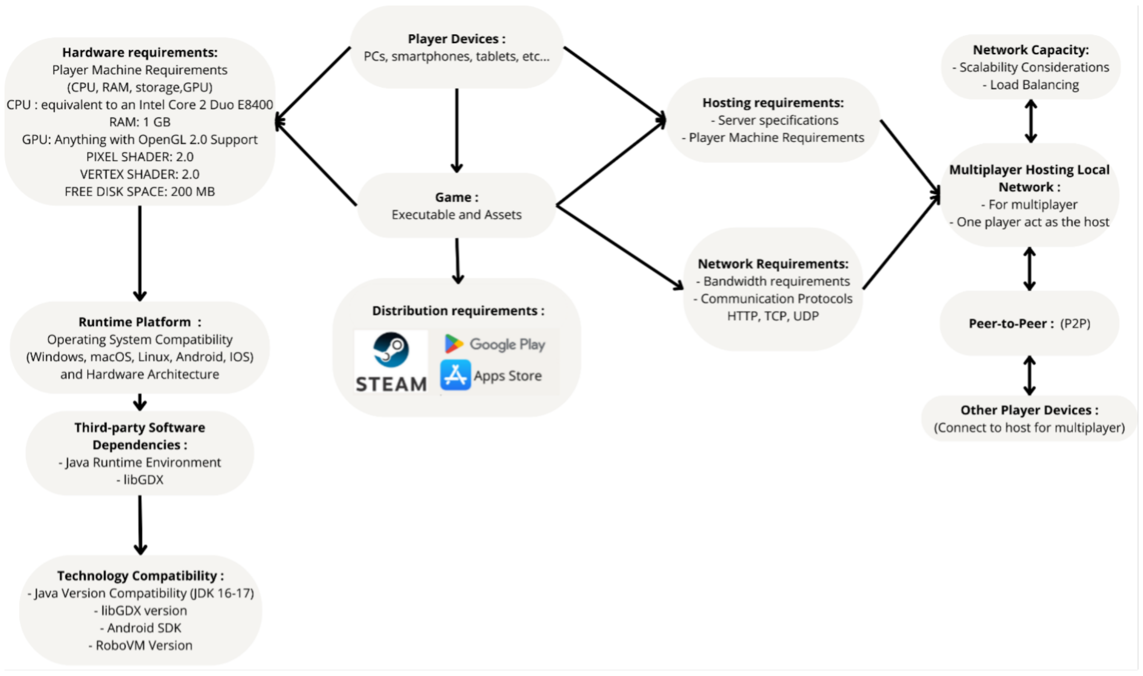

Player Devices : PCs, smartphones, tablets, etc...
Hardware requirements: Player Machine Requirements (CPU, RAM, storage,GPU) CPU : equivalent to an Intel Core 2 Duo E8400 RAM: 1 GB GPU: Anything with OpenGL 2.0 Support PIXEL SHADER: 2.0 VERTEX SHADER: 2.0 FREE DISK SPACE: 200 MB
Game : Executable and Assets
Hosting requirements: - Server specifications - Player Machine Requirements
Network Capacity: - Scalability Considerations - Load Balancing
Multiplayer Hosting Local Network : - For multiplayer - One player act as the host
Network Requirements: - Bandwidth requirements - Communication Protocols HTTP, TCP, UDP
Distribution requirements :
STEAM
Google Play
Apps Store
Peer-to-Peer : (P2P)
Other Player Devices : (Connect to host for multiplayer)
Runtime Platform : Operating System Compatibility (Windows, macOS, Linux, Android, IOS) and Hardware Architecture
Third-party Software Dependencies : - Java Runtime Environment - libGDX
Technology Compatibility : - Java Version Compatibility (JDK 16-17) - libGDX version - Android SDK - RoboVM Version

Figure 5: Deployment View of Mindustry

security. But spoofing multiplayer UUIDs is simple, so additional measures, such as IP bans, help mitigate the risk of cheating players ruining the game of the others player in multiplayer servers.

While the need for security may seem less critical, it's still vital for maintaining the integrity of the game and ensuring a positive user experience. Security breaches, even if they don't involve personal information or financial data, can still break gameplay, erode trust among the community, and damage the reputation of the game and its developers.

Also the security as to be internal of the development process and its collaborators itself. Since its an open-source project owned by just Anuke, he can't really asses every commits and pull requests so when new code come from extremely trusted contributor there often is no checking. Thus, this trusted contributor could easily add a backdoor to get console access on servers for example and then use a script to make every single community server implode.

For Performance, Anuken decided to create is own collections and packages of code such as the the package mindustry. They are self-made and designed to specifically match the game's needs and thus are great for better performances. Also there are some rules to follow if you want to contribute to the project such as:

- Do not allocate anything if possible.
- Never allocate ‘new‘ objects in the main loop. If you absolutely require new objects, use ‘Pools‘ to obtain and free object instances
- Otherwise, use the ‘Tmp‘ variables for things like vector/shape operations, or create ‘static‘ variables for re-use.
- If using a list, make it a static variable and clear it every time it is used. Re-use as much as possible.
- Do not create methods unless necessary.

- Unless a block of code is very large or used in more than 1-2 places, don't split it up into a separate method. Making unnecessary methods only creates confusion, and may slightly decrease performance.

3.4.2 Organizational Metrics

We will now see metrics depicted by Nagappan et al. (2008) that can help asses the impact of the organizational structure of Mindustry.

In the Mindustry organization, there is a high number of engineers (NOE): 592. The problem is that the more people touch the code, the higher the chance of having defective code. The reason for that is that a high number of engineers need way more coordination amongst themselves. If there are N engineers who touch a piece of code there needs to be (N*(N-1))/2 theoretical communication paths for the N engineers to communicate amongst themselves. In our case we would need: (592*(592-1))/2 = 174936 theoretical communication paths for the 592 engineers to communicate amongst themselves. This is a huge number and that shows that there may be miscommunication between those engineers leading to design mismatches, breaking another engineer's code and problems understanding design rationale.

When looking closer at that number, it can be decreased since only 20 members committed more than 50 times since the beginning of the project in 2017 and only 10 committed more than 100 times. The second most active contributor did 432 commits, and the 1st (Anuken the owner of the project) did 11,954 of the 17,445 commits, which is 68% of the whole project. So we don't need that much theoretical communications, when taking the 20 most active members we would need: (20*(20-1))/2 = 190 theoretical communications.

Still, communication is assured through this strong bonded, and connected community of developers using tools like Discord where all active developers and aspiring helpers can talk easily and discuss any problem. Also, Anuken, the owner of the project, put in place many rules to follow to be able to contribute to Mindustry, helping in reducing potential miscommunications problems.

As an open-source project, there is a high number of ex-engineers (NOEE): 572. This is the total number of unique engineers who have touched a binary and have left the company. By losing engineers, we lose knowledge of the product and thus there is a likelihood that the new person taking over the code of the leaving engineer might not be familiar with the design rationale, the reasoning behind certain bug fixes, and information about other stakeholders in the code.

And with Mindustry, most of the contributors helped around 2021 for a few commits and then left. Nowadays there are only a few contributors that are coding regularly, around 20. So, we could think that most of the knowledge disappeared but as stated before, 68% of the commits are done by only one person, so most of the knowledge is concentrated on this person so knowledge isn't lost. But that poses a big problem, if Anuken decides to stop coding and contribute to the project we lose everything, and it will be much more difficult to advance the project even further.

Since Mindustry is a project that started 7 years ago and to which a lot of people could contribute there is a high edit frequency (EF). This is the total number of times the source codea that makes up the binary, was edited. A high EF is an indicator of a lack of stability/control and even a lack of reliability and performance. In total, there were 17,445 commits since the beginning of the project. It is around 11 commits per workday (I took for each year 229 days of work). And of course, during the project, there were ups and downs in productivity and the all-time high was in November 2020 where more than 700 commits were done in a month. So, there is a high risk of control during these times, and even if Anuke wants to personally assess every commit he just can't keep up in times like that. So, rules and habits are in place to help with control and have a more stable project.

With an open-source project where there is just the creator and then all the contributors, the depth of master ownership (DMO) is really low. This metric determines the level of ownership of the binary depending on the number of edits done. And considering 68% of the project is done by the owner we can say that code quality is good and consistent. By having such a low DMO the approval/control of the project is condensed into a single point which improves intellectual control so there are no issues regarding decision-making and understanding intersecting code dependencies. Anuken can make important decisions without affecting code quality.

Also, the contributors are cohesive since the percentage of org contributing to development (PO) is up to 68%, because this is the percentage of commits that are made by Anuke. The organization is cohesive due to the fact that it is only one person who makes the decisions and most of the code.

It is the same with the level of organizational code ownership (OCO), which is the percentage of edits from the organization that contains the binary owner. So it's the same percentage: 68%. The more the development contributions belong to a single organization, the more they share a common culture, focus, and social cohesion. The more diverse the contributors to the code itself, the higher the chances of defective code, e.g., synchronization issues, mismatches, and build breaks. Here, it's not even one organization but one man, so the edits are surely cohesive. Even due to the active community on discord and its rules, there is a strong common culture and social cohesion among the contributors, limiting the chance of defective code. The overall organization ownership (OOW) is the ratio of the percentage of people at the DMO level making edits to a binary relative to the total number of engineers editing the binary. Anuken is the only owner, so this represents only 0,0003% of the total of engineers. But since he did 68% of the contributions, we can't say that this diffused contribution to the binary is lowering the quality.

We can say that only 2 organizations are working on the project, the owner Anuke and all the contributors so the organization intersection factor (OIF) is 2. The OIF is a measure of the number of different organizations that contribute greater than 10% of edits, as measured at the level of the overall org owners. So, the contribution is far from diffused and there is a strong ownership.

In summary, the analysis of Mindustry's organizational dynamics reveals both strengths and potential challenges. The organization benefits from a significant contribution of engineers, fostering a dynamic developer community. However, this abundance of contributors presents challenges in communication and coordination, creating a necessity for robust mechanisms to ensure project coherence. The concentration of contributions on a few key individuals, notably project owner Anuken, offers both advantages and vulnerabilities. While Anuken's substantial contributions ensure consistency and quality control, they also highlight a risk of a single point of failure. Despite Anuken's dominance, the organization shows cohesion and a shared culture among its contributors, mitigating risks associated with contributor diversity. So Mindustry's organizational structure is a balance between centralized control and community collaboration, with proactive measures through a lot of rules about contributions and communication in the Discord.

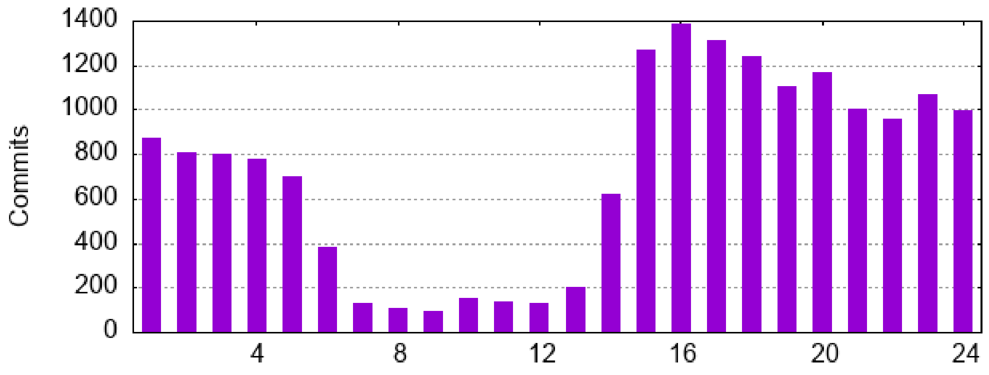


Figure 6: Number of commits by hour of day. Created with gitstats and the Mindustry GitHub-Repository.

4 Software Metrics

4.1 Repository Analysis

4.1.1 Gitstats

I used gitstats to analyze the GitHub-Repository, e.g. to get the amount of commits per author or the amount of files over time.

When looking at the number of commits by the hour of the day in Figure 6, it can be seen that the least amount of commits are between 7 am and 2 pm. In this time, people usually work in their regular jobs, which shows that this project is not maintained by people working on it full-time, but rather by people who like to contribute to this in their free time. This is supported by the fact that most commits are done between 4 pm and 12 pm, when people have time for their own interests after their workday. Another indicator for this is that the number of commits does not go down during the weekend, instead it even increases a little on Saturdays compared to some other workdays.

The statistics for the amount of commits by author seen Figure 1, show that there is one user, who commits the majority of changes. Anuken has committed about 68% of the whole project. He is also the founder of Mindustry and the one who gets notified about every pull request that is made in the repository. There are many more contributors, but the user who is responsible for the second most amount of commits is only responsible for about 2% of all commits. With that in mind it can be said that Anuken is the most important person for this project and that would not exits without him. It would probably also not be developed any longer, should he decide to stop, as he is responsible for the majority of all the code.

4.1.2 GitHub

I also had a look at the GitHub repository. What was interesting to see, is that the repository only had two active branches, the master branch and one feature branch called “async-ping”. When also taking the stale branches into consideration, there are only three feature branches more. It is not surprising that there are not more feature branches, as all contributors work on a fork of the original repository before merging it back. This is common practice in big open source project. This way, each contributor can experiment with their contribution as long as they want, without flooding the main repository with activities. It is more surprising to me that there is no development branch. The owner of the project, Anuken, merges the contributions personally, so he can make sure that they are accurate and work. The contributors also have to make sure their changes work by building and testing the application before opening a pull request. Still, I think it is surprising to directly merge a new contribution into the master branch. If something does brake, maybe because an edge case was not covered in the test suite, the whole application breaks. With an development branch, the application could be tested under real conditions without breaking everything when something goes wrong.

The project uses GitHub actions to automate certain processes. For example, for every pull request an action makes sure that the unit tests are run and a jar file is build. This makes sense in every big repository on which many people are working. It ensures that certain things always happen, like the running of tests, without having to rely on every person to remember to run all of these things. It also helps to standardize all commits by having them all pass the same actions.

4.2 Metrics Analysis

The frequency and history of changes made to the different files is not the only thing that can be analyzed. The files itself can also be analyzed with various tools focusing on different metrics.

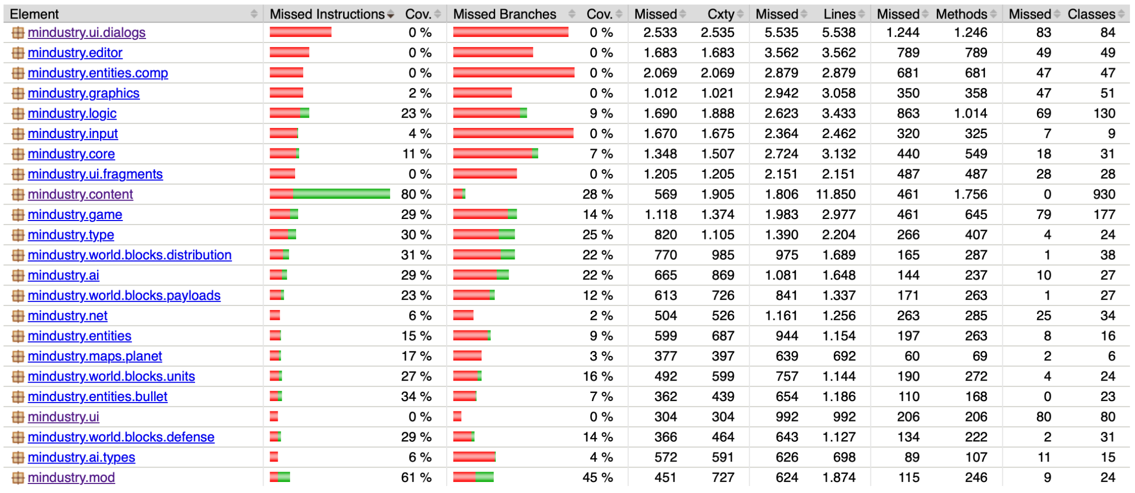
A test that is often performed on classical applications like for example websites, is a load test. This type of test is usually performed to see how the databases and APIs handle different amounts of incoming traffic. It is important to know whether a website can handle bursts of activity from its users without long waiting times for them because of scalability issues with the databases or the calls to the backend. A load test can help identify bottlenecks that need to be resolved for guaranteeing a good user experience at all times. As scalability and availability are quite an important part of a good software, I wanted to perform a load test for the Mindustry project. This proved to be very difficult.

Mindustry runs completely local, there is no extra server running in the background somewhere. This was confirmed by the main developer Anuken when one user on Discord asked about the possibility of having user accounts. Anuken's explanation as to why that will not be developed was: "Because that requires hosting a server, messing with databases, spoof prevention... so many things go into making an account system that works properly". This indicates that he does not want to deal with everything that comes along with hosting his own server and that the game itself is not designed in a way to need a dedicated server. If there is no server, it is not possible to test the scalability of the server. When there is no remote server to communicate with, there is also no API for the communication between a local frontend and a remote server. The same is true for a database. The application does not store any information about the game in a remote database. Instead, in-game structures are for examples stored in .msch-files and the game's maps and a user's saved game are saved in .msav-files, both explicitly created for these use cases in Mindustry. With that, there is also no database to load test.

As I could not perform any form of load test in this architecture, I focused on the maintainability and functionality of the code. For the maintainability, I looked at the cyclomatic complexity of different files. For the functionality I looked at the test coverage that is currently being offered by the project.

4.2.1 Cyclomatic Complexity

The Cyclomatic Complexity (CC) is a software metric developed by Thomas McCabe for describing the stability and level of confidence in a program. There are many different metrics for this out there which all use different elements of a program to compute it. The cyclomatic complexity uses the number of linearly independent paths through a program.



Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
mindustry.ui.dialogs		0 %		0 %	2.533	2.535	5.535	5.538	1.244	1.246	83	84
mindustry.editor		0 %		0 %	1.683	1.683	3.562	3.562	789	789	49	49
mindustry.entities.comp		0 %		0 %	2.069	2.069	2.879	2.879	681	681	47	47
mindustry.graphics		2 %		0 %	1.012	1.021	2.942	3.058	350	358	47	51
mindustry.logic		23 %		9 %	1.690	1.888	2.623	3.433	863	1.014	69	130
mindustry.input		4 %		0 %	1.670	1.675	2.364	2.462	320	325	7	9
mindustry.core		11 %		7 %	1.348	1.507	2.724	3.132	440	549	18	31
mindustry.ui.fragments		0 %		0 %	1.205	1.205	2.151	2.151	487	487	28	28
mindustry.content		80 %		28 %	569	1.905	1.806	11.850	461	1.756	0	930
mindustry.game		29 %		14 %	1.118	1.374	1.983	2.977	461	645	79	177
mindustry.type		30 %		25 %	820	1.105	1.390	2.204	266	407	4	24
mindustry.world.blocks.distribution		31 %		22 %	770	985	975	1.689	165	287	1	38
mindustry.ai		29 %		22 %	665	869	1.081	1.648	144	237	10	27
mindustry.world.blocks.payloads		23 %		12 %	613	726	841	1.337	171	263	1	27
mindustry.net		6 %		2 %	504	526	1.161	1.256	263	285	25	34
mindustry.entities		15 %		9 %	599	687	944	1.154	197	263	8	16
mindustry.maps.planet		17 %		3 %	377	397	639	692	60	69	2	6
mindustry.world.blocks.units		27 %		16 %	492	599	757	1.144	190	272	4	24
mindustry.entities.bullet		34 %		7 %	362	439	654	1.186	110	168	0	23
mindustry.ui		0 %		0 %	304	304	992	992	206	206	80	80
mindustry.world.blocks.defense		29 %		14 %	366	464	643	1.127	134	222	2	31
mindustry.ai.types		6 %		4 %	572	591	626	698	89	107	11	15
mindustry.mod		61 %		45 %	451	727	624	1.874	115	246	9	24

Figure 7: Exceprt from the coverage report from the jacoco plugin for gradle

This means the lower the cyclomatic complexity, the less independent paths there are, which generally makes a program easier to understand ("IBM Documentation", 2021). The ease of understanding a program is a good indicator for the maintainability of it. If it can be understood well without needing to spend days on understanding the code, there is a high probability that changes that are being made, will not break the existing functionality.

Some authors criticize the CC and state that it is not of much use. One of the reasons for that opinion is that the CC does not consider the number of lines of a program. As long as the program stays linear, the metric will not increase, even if the number of lines increases (Shepperd, 1988). Shepperd (1988) also argues that the CC increases if best practices about program structure are used, which is counterproductive. On the other hand, Ebert et al. (2016) state that the cyclomatic complexity is a great predictor for software components that will likely have a high defect rate or and will be hard to maintain. The CC gives a good starting point for a refactoring or redesign, as it displays with easy to understand reasons, which classes have a high complexity.

For these reasons I chose to compute the cyclomatic complexity for the project at hand. To compute the complexity, I used the jacoco plugin for gradle and integrated it into the already existing gradle.build file. An excerpt from the coverage report generated by jacoco can be seen in Figure 7.

What can be extracted from Figure 7, is that the cyclomatic complexity is extremely high with the highest value being 2535 in the directory mindustry/ui/dialogs. One could argue that this is the cumulated CC for the whole directory and thus not too bad. According to the report, there are 84 classes in the directory, which results in an average CC per class of about 30. This is still not ideal, as "limits over 10 should be reserved for projects that have several operational advantages over typical projects, for example experienced staff, formal design, a modern programming language, structured programming, code walkthroughs, and a comprehensive test plan" ("Code metrics - Cyclomatic complexity - Visual Studio (Windows)", 2024). Looking deeper into the individual classes, it can be discovered that there are many functions that are well over a CC of 10, sometimes even reaching the 30 and beyond.

Only comparing the CC that is displayed in Figure 7 is not an accurate result, as was just described, as the value does not consider the amount of the different classes and number of lines per directory. To get more accurate metrics, it would be necessary to look into the report for each directory separately. But despite that, I believe that also only looking at the directory-level CC can give a good estimation for the complexity of the different directories and for which it might be necessary to have a closer look for a possible refactoring. To make these discoveries more visual and easier to understand in one glance, I also used the free trial of the CodeMR-Plugin for IntelliJ. This plugin can compute different metrics for the whole project and render very concise graphs, as can be seen in Figure 8. According to the CodeMR documentation, the "Weighted Method Count" that can be seen in Figure 8 represents the McCabe complexity, i.e. the cyclomatic complexity ("CodeMR | Static Code Analysis and Software Quality Features", 2017). It can be seen that 21% of the whole project is considered to have a very high complexity. In my opinion that is a very high number for any project, but especially for an open source project, in which many people should be able to participate. Having such a high complexity makes it hard for new people to join the project, as it makes it difficult to understand what is going on and how the different modules work together. Secondly, the possibility to break things is quite high. Potentially, there are many people working on the source code, thus also many people who could understand something wrong and break existing functionalities with their changes.

4.2.2 Test coverage

Tests are a necessary part of every software project. They make sure that the software is functionally correct and behaves the way it is supposed to behave, even in certain edge cases. They are especially important when the software is still under development and many different people are working on it. Whenever something is changed about the logic of the code or a new feature is added, tests can be used to make sure that the old features are still working as intended and that the behavior of the application has not changed in not-intentional ways. Tests can be done on different levels. Some tests, so called Unit Tests, only cover a very small and specific part of the software that doesn't interact with any outside services. Integration testing on the other hand, is responsible for testing the communication and cooperation between different components of the software ("What is Software Testing?", n.d.).

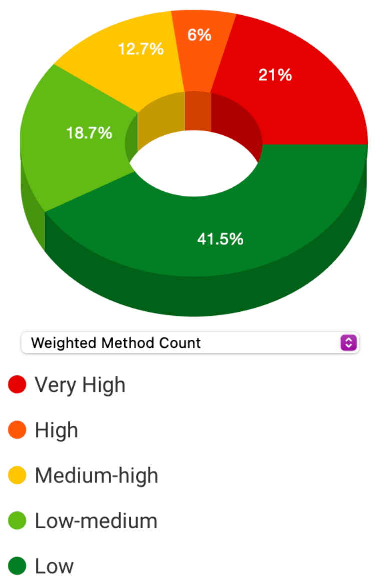


Figure 8: Report on the weighted method count, generated with CodeMr

As can be seen by the paragraph above, it is important that a software is well tested. A metric to make this comparable between many different projects is the code coverage. It shows how much of the source code is being tested by measuring how much of the code is executed during the tests. This helps to discover which critical parts of the source code are not being covered by any tests yet (Pittet, n.d.).

To compute the test coverage, I used the jacoco-plug-in for gradle, as already described in the previous section. This way the test coverage is automatically computed when running the corresponding gradle-task.

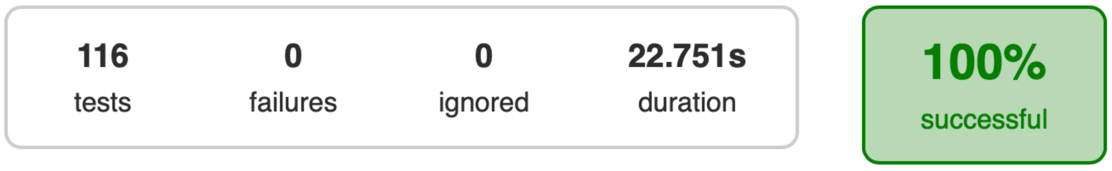

116 tests | 0 failures | 0 ignored | 22.751s duration | 100% successful

Figure 9: Report on test successes of the jacoco plugin

Apart from the detailed report on the coverage of the different directories that can be seen in Figure 7, jacoco also generates a report on the failures and successes of the performed tests, shown in Figure 9. The report demonstrates that all 116 existing tests have passed, and no test failed. 116 test cases might sound like a good number, but looking closer at the tests, it can be seen that 35 of these tests all call the same method, only with different inputs. This is also the case for some other methods. Counting only the distinct test cases, the number of tests is much smaller, i.e. 47. This is not much, considering that the whole project has an overall line count of 91261 and 1354 classes. This can also be seen when looking at the coverage report generated by jacoco (see Figure 7). The first three listed directories have a test coverage of 0%. It could be argued that this is okay, as these directories mostly deal with UI elements that maybe do not need to be tested as extensively as the game logic itself. But even when disregarding these three directories, the test coverage is very low. The aim is usually to reach about 80% of coverage (Pittet, n.d.), but in this project, only 8 out of the more than 60 directories reach a test coverage of over 50%. Most of the directories have a coverage between 10% and 30%. Additionally, these are the values for the whole directory. This means that in these directories, there will be many classes and functions with a coverage of 0% in order for the overall coverage to be so low. This is being confirmed when looking deeper into the individual directories on class and method level.

It is very hard to write test cases for a game, that cover every possible move that a player could make. There are so many different possibilities, that trying to cover them all is not possible. But certain things will always happen during a game, e.g. creating blocks, playing the background music or providing statistics about the on-going game. These are things that could be tested a lot more. There are many different people working on this project, who may not all understand each little part of the source code to its full extend. This makes it very easy to accidentally break an existing feature when implementing a new one. Because of this, it would be a good idea to have more tests, that cover a lot of the basic features of the game. Another reason why it would be important is that the project is written in Java, meaning it is object-oriented. Object-oriented projects need testing even more than other projects, as encapsulation makes it harder to find bugs and inheritance needs extensive testing of the inherited code (Bertolino, 2007).

5 Contribution

A common problem that many open-source projects face, especially far into their lifespans, is accepting newcomers into their ecosystem. When codebases have been built up over years, with hundreds of contributors and millions of lines of code, newcomers are faced with an incredible hurdle in understanding the project to the point where they may make a useful contribution. Despite this, open-source projects need a constant stream of newcomers to keep their ecosystem alive and growing.

"Despite the success of the overall FOSS movement, research has demonstrated that FOSS projects are characterized by high project abandonment rates, a lack of new developers' enrolment, and a lack of sustained participation altogether seriously endangering their survival, long-term success, and sustainability." (Carillo et al., 2017, p.323)

To try and help newcomers approach Mindustry more easily and get around the initial intimidation of an open-source project, the following will be a form of guide that will introduce them to each part of Mindustry's wonderful ecosystem. The goal is also to not just introduce them to Mindustry, but also build a template for any open-source project to use as a jumping off point for their new contributors. This may take the form of a blog, FAQ, or even a README located somewhere easy to find. The following will be broken down into two parts, "Analysing the Codespace" and "Making a Contribution". Each will also be broken down into its respective parts to make it easy to digest.

5.1 Analysing the Codespace

The first step to joining any coding project, whether it is open-source or not, is to analyse and understand how the code works. To start this process, it's smart to read any documentation on the rules that govern how the code is written. Mindustry does a good job of this by providing a file called "CONTRIBUTING.md" that lays down guidelines on how to contribute and a style guideline on what contributions should look like. The first gives helpful "dos" and "don'ts" on protocol when wanting to make a commit. This includes a preferred IDE, IntelliJ, always test your changes, discuss large changes before making them, and do not make formatting PRs. The second gives a large breakdown on the expected coding etiquette used. There are a lot of small formatting related expectations like using camel case for all variables, not putting spaces around parenthesis, never use underscores, and use short method/variable names. Most of them are common coding practices, but some are specific to either Anuken or the language, so it is imperative to learn these. A style file is provided that can be imported into IntelliJ to get the correct formatting. There are other style guidelines provided that don't have to do with formatting. These include not using incompatible Java features, using "arc" collections and classes, avoiding boxed types, not allocating anything if possible, avoiding bloated code and unnecessary getters/setters, and not creating methods unless necessary. Most of these are very specific to the project and thus are incredibly important to remember. Also, unlike the formatting expectations, these cannot be checked with a style file and left to you to remember.

Luckily the guidelines for the project are very explicit and hard to miss when looking at the GitHub, but other projects may not have these tools as visible. No matter what, learning the coding practices of the project must be done before even trying to understand the code.

Now, most newcomers are faced with the issue of looking at the massive codespace and having no idea what files do what. It was explained earlier that the project has a generally high complexity which makes this hurdle even tougher. However, tools such as gitstats, Jacoco, and those present in git can be used to organize the whole project into the most important parts. As a newcomer, it shouldn't be expected that they understand every single detail, but by using tools, the broader strokes of how everything fits together can be comprehended. However, the process of using those tools and analysing the statistics is not exactly straightforward. Instead, graphs and statistics and an analysis on what they mean will be provided for newcomers. For instance, by running a simple git command in the repository, the locations of the most edited folders and files can be obtained. The following two commands in can be used to achieve exactly that. The first command finds

Revisions	Location
14748	core
10978	core/src
7091	core/src/mindustry
6201	core/assets
4074	core/assets/bundles
3885	core/src/io
3885	core/src/io/anuke
3874	core/src/io/anuke/mindustry
2635	core/src/mindustry/world
2229	core/src/mindustry/world/blocks

Table 1: The most edited folders and their locations

Revisions	Location
1074	core/src/mindustry/content/Blocks.java
634	core/src/mindustry/content/UnitTypes.java
427	core/src/io/anuke/mindustry/core/Control.java
396	core/src/io/anuke/mindustry/world/Block.java
373	core/src/mindustry/type/UnitType.java

Table 2: The most edited files and their locations

the most edited folders with their locations, the second command does the same for the most edited files. The outputs can be found in Table 1 and 2, respectively.

```
git rev-list --objects --all | awk '$2' | sort -k2 | uniq -cf1 | sort -rn |
    head
git rev-list --objects --all | awk '$2 ~ /\.java/' | awk '{print $2}' | sort -
    k2 | uniq -c | sort -rn | head -n 5
```

Listing 1: Find most edited folders and files

The first thing they should see is that the most important code is in the core/src/-mindustry and core/assets folders. They should also notice that the "Blocks.java" and "UnitTypes.java" files tend to be revised the most and are thus important to the integrity of the game. This information is just what a newcomer needs to get off the starting block. From here, the newcomer can explore around the GitHub, using the statistics as a guide.

5.2 Making a Contribution

The process of contributing to a project can be very intimidating. "What does the project want me to contribute?" "Is my contribution too big/small?" "Is my idea good enough?" Because large open-source projects can have a very professional environment, it can be scary asking these questions. In Mindustry's special case, the community can be very welcoming and helpful. However, this will not be the case with other projects. In order to ease newcomers into their environment and explain what contributions can be made, it would be smart for the project to compile a list of different resources that they use, the capabilities of each, and how newcomers can use them to help the project.

It should be noted now that a contribution does not necessarily have to be a commit to a codespace. Open-source projects also require contributors who can help with graphic design, quality assurance, marketing, licensing, and more. This is most apparent in Mindustry's Discord where there are entire sub-communities surrounding aspects of the game like art and translating. These non-programming roles are vital for a game like "Mindustry" to operate and newcomers are encouraged to check them out if they're interested. However, here the discussion on contributing will be centred around the goal of helping with the development. It should be noted though, that this template can be used for those sub-communities in the

It may not be a resource necessarily, but when approaching a game with the goal of contributing, there is no better place to start than just playing the game. Only those who know how the game runs, what systems it uses, how the player interacts with the systems, etc. can make useful suggestions. A free and easy way to run the official release of the game is through their itch.io. The only missing bits from this release are achievements, seamless multiplayer, and map browsing/workshop which are in the paid version on Steam. However, for new players, the campaign, which is in the free version, will teach them the basics. Playing the game alone will also help newcomers form their own opinions of the games systems. By experiencing the game from an individualistic perspective, newcomers can contribute more interesting and creative suggestions. Finding a community to fit into may be a large draw for many who seek to contribute to an open-source project, but those who can take their own experience with the product and apply a new point of view will become invaluable to the team.

A very useful tool that Anuken uses to organize the development process is a website called Trello. Trello is a web-based project management and collaboration tool that organizes tasks and projects into boards. Each board typically represents a project, and within each board, users can create lists to represent different stages or categories of tasks. Within these lists, users can add cards to represent individual tasks or items. Anuken uses Trello as a way to organize where in the development process he is with different Mindustry projects. A newcomer can see what is currently in development, what has been developed but not implemented, and what has already been finished. The most important part here is the "In Progress" tab. Here, there are cards that lay out the details on what is currently being worked on. As the report is being written, Anuken is currently working on implementing Neoplasm, a new type of enemy. What's great about Trello, is anyone is able to comment on these cards and give their input, which is a great way to start getting involved. It may not be a commit, but Trello still gives a great way to contribute ideas.

Mindustry's Discord server is active daily with community members discussing all matters on the game. Some are there actively developing on the game, some are newcomers asking questions, some are there discussing mods, and some are there as casual players who enjoy the game. In general, with this much discussion happening around the game, there are a plethora of ways for the players to find bugs to report, improvements that could be suggested, and a multitude of queries. Even just reading through what other people are talking about can help inspire newcomers.

A good place to start here is the "Discussions" section of the Discord. Here, chats are organized into specific sections like mapping, logic, and art and is open to everyone. More casual conversations occur here, so it's easy for anyone to get involved. The "help" channel is perfect for players with any gameplay issues, the "design" channel is filled with players showing of their factory designs, and the "mindustry" channel has general chatter about the game. Although most of the people here are casual players, this is still a great opportunity for anyone to start getting involved.

There is also a "Development" section where more technical discussions occur. Here, people tend to talk about potential bugs, new ideas for game systems, or what Anuken is currently working on. This is the perfect place for newcomers to raise their own queries since the replies will be from experienced developers who can give reliable explanations and advice. From just a small amount of scrolling through the "development" channel, there are discussions on what length measuring equation is most optimal, discrepancies between the code that determines belt capacity and what is shown in-game, and lots of many small development related questions being answered. The admins are very strict here about keeping every message on-topic, so just make sure what is being asked or suggested belongs here.

The GitHub is the main location for reporting any bugs or making edits to the codespace. However, wasting the time of other developers by presenting a false bug or making useless edits is bad manners and should be avoided. Before ever making an issue or pull request on GitHub, the Discord should be consulted, and they should try their best to go through past ones to see if the same has been previously reported. It is also recommended to check out previous issues and pull requests posted and see how others format their submission and which kinds of suggestions are being addressed and which are being scrutinized. To streamline this process, we would provide examples of good and bad edits. Having examples to go through can help newcomers adjust to the kinds of edits that are typical of a specific project. There should also be a size range in the examples provided so that newcomers can see how big edits are expected to be. Here are a few examples of issues we would give newcomers for Mindustry.

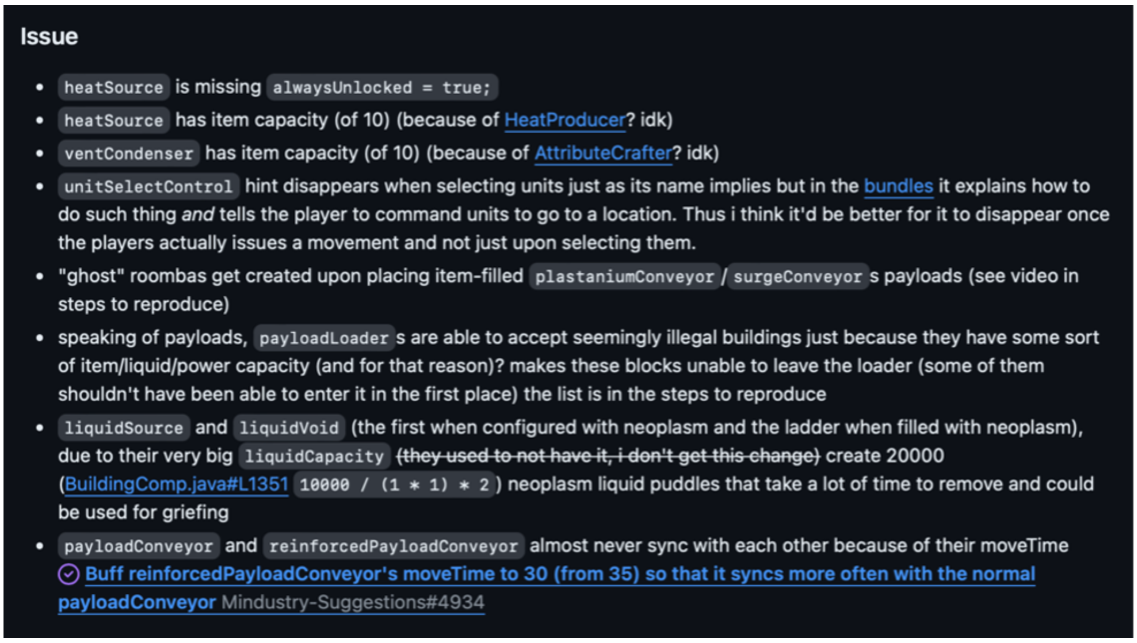
Issue #9732 is a great example of a very simple and typical suggestion from the community. The premise is very simple and even simpler fix. Two towers called the “Electrolyzer” and “Atmospheric Concentrator” both have an inaccessible “itemCapacity” variable that is set to 10 when they do not hold any items. Typically, to save lines of code, the item capacity of towers are hard-coded to have a value of 10. Since the methods that create these towers do not reference the value, they are defaulted to a value of 10. The only fix that needed to be made here is Anuken edited both towers’ methods in the “Blocks.java” file to set the “itemCapacity” to 0. In the end, the change was only two lines of code.

Issue #9731 is like the previous issue in theme, but a much larger list of suggestions. This issue was submitted by the same person as the previous one and was also done before, hence the smaller number. They are discussed in this order because the previous one is much easier for a newcomer to digest and knowing the context of that one makes this one more understandable. The issue submitted can be seen in Figure 10.

It may seem like a lot, but, in reality, the person is just making many small suggestions. Anuken is again the one to respond to this, but this time he does not use every one of the person’s suggestions. For the ones that he disagrees with, he does a great job of explaining his design choices and the reasoning behind them, as can be seen in Figure 11.

Each of the suggestions made that end up getting added are all very minor tweaks, similar to the issue before, but since so many are done, the end-product is a sizeable commit. It is clear that the person suggesting has been playing and messing around with the game for a very long since the suggestions are so well thought out. But it is only because they have played the game so much that they can have this in-depth knowledge. This exact format could then be used to demonstrate typical pull requests which, for this project, are quite similar in format. Through the use of these examples, a newcomer can learn what is typically being suggested, the formatting of a typical edit, and the size range that they can have.

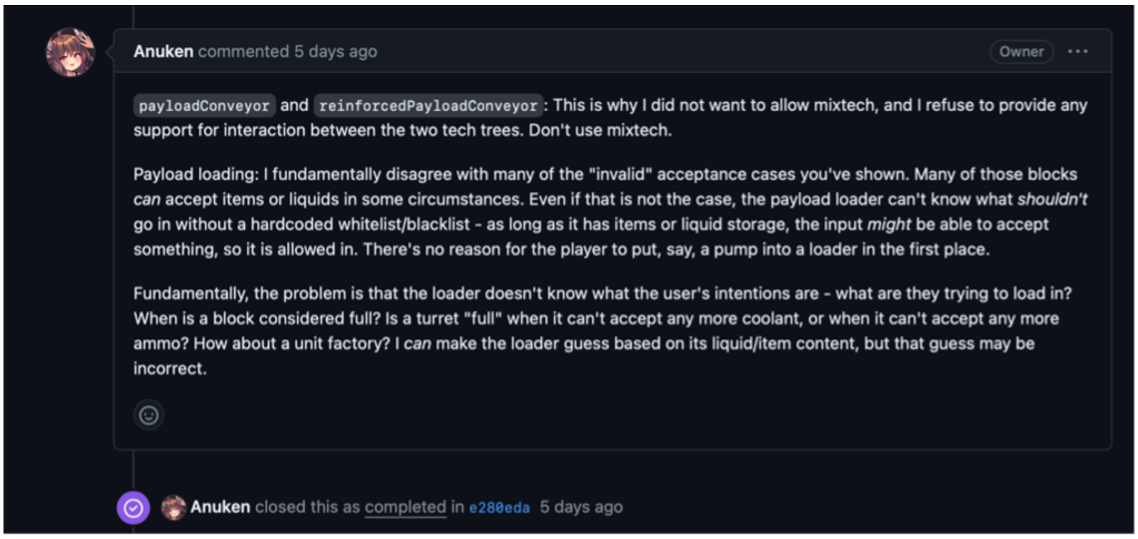
In the end, the first section 5.1 contains information on the rules that govern how the code is written, i.e. syntax and style guidelines, and a plethora of different graphs and statistics that can demonstrate anything from the testing coverage to code complexity to the files with the most revisions. This second section 5.2 contains a list of resources that newcomers can use to help and engage with the ecosystem and their specific functionalities. By providing these resources to newcomers open-source projects can finally live up to their name by having a lower barrier to entry and explicit ways for newcomers to contribute.



Issue

- `heatSource` is missing `alwaysUnlocked = true;`
- `heatSource` has item capacity (of 10) (because of HeatProducer? idk)
- `ventCondenser` has item capacity (of 10) (because of AttributeCrafter? idk)
- `unitSelectControl` hint disappears when selecting units just as its name implies but in the bundles it explains how to do such thing *and* tells the player to command units to go to a location. Thus i think it'd be better for it to disappear once the players actually issues a movement and not just upon selecting them.
- "ghost" roombas get created upon placing item-filled `plastaniumConveyor` / `surgeConveyor` s payloads (see video in steps to reproduce)
- speaking of payloads, `payloadLoader` s are able to accept seemingly illegal buildings just because they have some sort of item/liquid/power capacity (and for that reason)? makes these blocks unable to leave the loader (some of them shouldn't have been able to enter it in the first place) the list is in the steps to reproduce
- `liquidSource` and `liquidVoid` (the first when configured with neoplasm and the ladder when filled with neoplasm), due to their very big `liquidCapacity` ~~(they used to not have it, i don't get this change)~~ create 20000 (BuildingComp.java#L1351 `10000 / (1 * 1) * 2`) neoplasm liquid puddles that take a lot of time to remove and could be used for griefing
- `payloadConveyor` and `reinforcedPayloadConveyor` almost never sync with each other because of their moveTime Buff reinforcedPayloadConveyor's moveTime to 30 (from 35) so that it syncs more often with the normal payloadConveyor Mindustry-Suggestions#4934

Figure 10: Example of an issue raised on GitHub



Anuken commented 5 days ago Owner

`payloadConveyor` and `reinforcedPayloadConveyor`: This is why I did not want to allow mixtech, and I refuse to provide any support for interaction between the two tech trees. Don't use mixtech.

Payload loading: I fundamentally disagree with many of the "invalid" acceptance cases you've shown. Many of those blocks *can* accept items or liquids in some circumstances. Even if that is not the case, the payload loader can't know what *shouldn't* go in without a hardcoded whitelist/blacklist - as long as it has items or liquid storage, the input *might* be able to accept something, so it is allowed in. There's no reason for the player to put, say, a pump into a loader in the first place.

Fundamentally, the problem is that the loader doesn't know what the user's intentions are - what are they trying to load in? When is a block considered full? Is a turret "full" when it can't accept any more coolant, or when it can't accept any more ammo? How about a unit factory? I *can* make the loader guess based on its liquid/item content, but that guess may be incorrect.

Anuken closed this as completed in `e280eda` 5 days ago

Figure 11: Anuke's response to the example issue

6 Conclusion

In conclusion, our analysis revealed that Mindustry is a project with very special characteristics. It started out as a reasonably-sized personal project, but thanks to a dynamic community, it has become an open-source project followed by thousands of people. Whether it's through the views or the various metrics, we can see the desire to create a project that is close to its users. Indeed, creator Anuken invites players to participate in improving the game by helping maintain and develop it. What's more, the game is designed to be ”simple” in the sense that everything has been thought out to make it as accessible as possible to everyone, and to run on any machine. However, the game also has the shortcomings of its qualities. The fact that it was mostly coded by a single person makes it rather difficult to understand the code. That's why we have thought of adding an introductory tutorial for new contributors to explain the content of the code and guide them to an effective contribution. This would allow the game to live longer, as it is currently still very dependent on its creator.

As the gaming industry continues to evolve, the importance of open-source projects such as Mindustry is becoming increasingly apparent. By encouraging a community approach to game development, Mindustry has shown that it is possible to create a game that is both accessible and reaches a wide audience. With the addition of an introductory tutorial for new contributors, the project has the potential to attract even more talent and continue to thrive for years to come. Ultimately, Mindustry's success shows the power of collaboration and the enormous possibilities that can be achieved when passionate individuals come together to create something together.

References

Bertolino, A. (2007). Software testing research: Achievements, challenges, dreams. *Future of Software Engineering (FOSE '07)*, 85–103. https://doi.org/10.1109/FOSE.2007.25

Carillo, K., Huff, S., & Chawner, B. (2017). What makes a good contributor? understanding contributor behavior within large free/open source software projects – a socialization perspective. *The Journal of Strategic Information Systems*, *26*(4), 322–359. https://doi.org/10.1016/j.jsis.2017.03.001

Code metrics - cyclomatic complexity - visual studio (windows). (2024, March 25). Retrieved April 10, 2024, from https://learn.microsoft.com/en-us/visualstudio/code-quality/code-metrics-cyclomatic-complexity?view=vs-2022

CodeMR | static code analysis and software quality features. (2017, September 17). Retrieved April 17, 2024, from https://www.codemr.co.uk/features/

Ebert, C., Cain, J., Antoniol, G., Counsell, S., & Laplante, P. (2016). Cyclomatic complexity. *IEEE Software*, *33*(6), 27–29. https://doi.org/10.1109/MS.2016.147

IBM documentation. (2021, March 8). Retrieved April 8, 2024, from https://www.ibm.com/docs/en/raa/6.1?topic=metrics-cyclomatic-complexity

Nagappan, N., Murphy, B., & Basili, V. R. (2008). THE INFLUENCE OF ORGANIZATIONAL STRUCTURE ON SOFTWARE QUALITY: AN EMPIRICAL CASE STUDY.

Pittet, S. (n.d.). *What is code coverage?* [Atlassian]. Retrieved April 10, 2024, from https://www.atlassian.com/continuous-delivery/software-testing/code-coverage

Shepperd, M. (1988). A critique of cyclomatic complexity as a software metric. *Software Engineering Journal*, *3*, 30–36. https://doi.org/10.1049/sej.1988.0003

What is software testing? definition and types [GitHub resources]. (n.d.). Retrieved April 10, 2024, from https://resources.github.com/software-development/what-is-software-testing/