Contents

Formal specification

This is the formal specification of the Battlecode 2022 game. Current version: *2022.2.2.0*

Welcome to Battlecode 2022: Mutation.

This is a high-level overview of this year's game. It is highly recommended to read this entire document before you begin writing bots.

This document and the game it describes may be tweaked as the competition progresses. We'll try to keep changes to a minimum, but may have to make modifications to keep the game balanced. Any changes will be announced in the official Discord channel. You may find a Changelog at the bottom of this document.

Background

The dust settles on the scorched landscape, the sun's glare harsh and unforgiving. The world is a radioactive wasteland, overrun with mutations and anomalies. Buildings have learned to walk, transmutation has given birth to the blooming science of alchemy, and even the robots have become sentient.

The arduous task of restoring civilization is yet to begin, but amidst the uncertainty, two warring factions of robots have emerged. History is being written before our eyes, for only one faction will rise above the turmoil to secure the future of the world.

Objective

In **Battlecode: Mutation**, you will write code to control your faction of robots, in a dangerous battle to conquer the enemy team.

The game world is subject to frequent Anomalies that can inflict long-lasting damage to both teams. Each player will begin with several Archons, and the player who loses all Archons first loses the game. Players must create robots that can tolerate onslaughts of Anomalies as they fight the opposing player faction.

Good luck!

Robots

The Battlecode world involves many robots of different kinds. These robots can perform actions such as attacking, moving, and communicating with each other. In each battle, your robots will face an enemy faction on the game map.

The game is turn-based and divided into **rounds**. In each round, every robot gets a **turn** in which it gets a chance to run code and take actions. Code that a robot runs costs **bytecodes**, a measure of computational resources. A robot only has a predetermined amount of bytecodes available per turn, after which the robot's turn is immediately ended and computations are resumed on its next turn. If your robot has finished its turn, it should call `Clock.yield()` to wait for the next turn to begin.

All robots have a certain amount of HP (also known as hitpoints, health, life, or such). When a robot's HP reaches zero, the robot is immediately removed from the game.

Robots are assigned unique random IDs no smaller than 10,000, except for your Archons.

Robots interact with only their nearby surroundings through sensing, moving, and special abilities. Each robot runs an independent copy of your code. Robots will be unable to share static variables (they will each have their own copy), because they are run in separate JVMs.

There are two types of robots: Droids and Buildings. Droids are robots which can freely move, forming your team's mobile combat forces; on the other hand, Buildings are generally immobile robots. You can create new robots by spending its associated resource cost. Different types of robots specialize in different tasks, and you should aim to use each type to your advantage.

Buildings overview

Buildings are generally immobile robots. When first created, they are Prototypes that cannot move or act; after they are fully built, they become Turrets that can perform actions.

There are three types of Buildings.

- Archons are headquarter Buildings that cannot be constructed. Each team starts with between 1 to 4 Archons inclusive as specified by the map, all at maximum health. Archons cannot attack, but they can build new Droids, and repair nearby Droids to increase their health.
- Laboratories house alchemists who convert Lead into Gold. Laboratories cannot attack, but can choose to produce exactly 1 Au per turn by spending Lead at a variable conversion rate. Alchemists prefer working in solitude and spend less Lead when fewer friendly robots are in their vision range.
- Watchtowers are defensive Buildings that reinforce an area. Watchtowers can deal

powerful attacks to robots in a large range.

As a special mechanic, Buildings are allowed to transform between Turret and Portable mode. Each transformation takes place over 10 turns, during which the Building will be unable to take any actions.

- In Turret mode, they can perform their regular action, but not move.
- In Portable mode, they can move, but not perform their regular action.

Finally, Buildings can also be Mutated to increase their mutation level. Mutating a building causes it to become stronger, more powerful, and more formidable. Buildings begin at mutation level 1, and there are a total of 3 mutation levels.

Droids overview

Droids are highly mobile robots that form the foundation of your strategic and offensive capability.

- Miners are resource-farming Droids that collect resources scattered on the game map. Collected resources are instantly added to your team's resource bank, and are available for immediate use.
- Builders are worker Droids that create new Buildings, and repair existing ones. New Buildings are initially in Prototype mode and must be repaired to full health before becoming Turrets.
- Soldiers are general-purpose ranged attacking Droids.
- Sages are Droids that can cause Anomalies. Anomalies caused by Sages affect a much smaller area than naturally-occurring ones, but have much more concentrated effects.

Anomalies

The world is unstable. Anomalies are extraordinary events that may occur periodically, and can have long-lasting effects on your robots. Your advanced technology means you can access the forecast of all impending Anomalies; to thrive, your robots should be robust and designed to withstand them.

The schedule of Anomalies is predetermined for each map and can be sensed by your robots. You can generally expect them to occur naturally approximately once every 200 rounds, although this can vary. There are five different types of anomalies, as listed below.

- Abyss: Matter is lost into the abyss of spacetime. 10% of resources in all squares, as well as team reserves, are lost.
- Charge: The air begins to conduct electricity, and tight packs of Droids short-circuit. The top 5% Droids with the most friendly robots in sensor radius are destroyed.
- Fury: A searing solar flare erupts from the sun. All buildings in turret mode have 5% of their max health burned off.
- Vortex: The world is upended and the terrain is reoriented based on the symmetry of the map.
    - Horizontally Symmetric Map: Terrain is reflected across a vertical central line
    - Vertically Symmetric Map: Terrain is reflected across a horizontal central line
    - Rotationally Symmetric Map: Terrain change is chosen uniformly randomly between
        - Reflection across central vertical line
        - Reflection across central horizontal line
        - Rotation of 90° clockwise (Only for square maps)
- Singularity: A black hole finds the weaker team and consumes it. This is guaranteed to happen on, and only on, turn 2000, and the team is selected via the tiebreakers listed in the Victory section.

Map overview

Each Battlecode game will be played on a map. The map is a discrete 2-dimensional rectangular grid, of size ranging between 20×20 and 60×60 inclusive. The bottom-left corner of the map will have coordinates (0, 0); coordinates increase East (right) and North (up).

Coordinates on the map are represented as `MapLocation` objects holding the `x` and `y` coordinates of the location.

Each square of the map may contain a certain nonnegative integer amount of **rubble**. Walking through rubble will slow down your robots; you cannot move rubble. It is guaranteed that each square contains no more than 100 rubble.

Map squares may also initially contain a certain amount of **Lead**. Lead is one of the key resource types in the game, and can be collected by Miners. It is guaranteed that at least one of your Archons will initially have a Lead deposit within its vision range.

Each team will start with between 1 and 4 Archons on the map.

Each map will also have a “schedule” for Anomalies, which describes the round and type of all naturally-occurring Anomalies. Players can sense this schedule in order to vary their strategy!

In order to prevent maps from favoring one player over another, it is guaranteed that the world is symmetric either by rotation or reflection.

Metals

The core resources are Lead (symbol: Pb) and Gold (symbol: Au). Each team has a stockpile of these metals, and constructing robots subtracts from this stockpile. Earning more resources enables you to amass a larger and more powerful army of robots.

There are Lead deposits scattered on the map, which can be collected by Miners. Whereas Lead is comparatively plentiful, Gold is not found naturally. Building a Laboratory will allow you to discover the science of alchemy; the Gold you synthesize will unlock the most powerful mechanics available in the world.

Every round, each team will gain a passive income of 2 Pb. Additionally, every 20 rounds, any square of the map containing at least 1 Pb will generate an additional 5 Pb.

Both teams start the game with 200 Pb and 0 Au. Resource costs for robot construction can be found in the in-depth section further below.

Rubble

Each square on the map has some non-negative integer amount of rubble. Rubble can slow down robots acting or moving onto the square.

Whenever a robot moves, attacks, or performs any other action, any cooldown it incurs is multiplied by a factor proportional to the rubble. Specifically, when incurring a base cooldown penalty of c, the actual cooldown applied is $\left\lfloor \left(1 + \frac{r}{10}\right) \cdot c \right\rfloor$ if the robot is located on a square with r rubble at the conclusion of the action.

Resource reclaim

Whenever a robot is destroyed, 20% of its build cost is dropped on the square it last occupied. This includes its initial build cost, as well as the cost of any Mutations applied to it. Although Archons cannot be built, they have a nominal cost of 100 Au to be used in calculating the reclaim value.

Notice that resource reclaim is the only way in which Gold will enter the game map and be available for collection by Miners.

Communication

Robots can only see their immediate surroundings and are independently controlled by copies of your code, making coordination very challenging. You will be unable to share any variables between them; note that even static variables will not be shared, as each robot will receive its own copy.

To facilitate communication, each team has a shared array of 64 non-negative integers strictly less than 2^{16}. Array values persist across turns; ie. they are not reset. Any robot can read from the array for a very small bytecode cost; they can also write to the array for a larger bytecode cost.

Sensing and vision

A robot can “sense” a square if the square is within the robot's sight range. A robot can “sense” another robot if it can sense the square that robot is occupying. Vision is **not** shared between robots, so a robot cannot necessarily sense information about robots in sight range of a different robot.

Sensing range is measured in squared units (Euclidean distance squared). The amount of rubble, Lead and Gold on a map square can only be sensed when that map square is within your sight range. You can also check whether a square within range is on the map using `onTheMap()`.

Victory and tiebreaks

To win **Battlecode: Mutation**, your primary objective is to destroy all enemy Archons. The first team whose final Archon is destroyed immediately loses (and conversely, the team who destroys the opponent's last Archon immediately wins).

Games must end in finite amounts of time. If games do not end by round 2000, the Singularity anomaly will occur, and destroy the weaker team. The following tiebreakers are applied in order to determine the stronger team.

1. The team with more Archons.

2. The team with greater net value in Au (alive robots and unexpended reserves).
3. The team with greater net value in Pb (alive robots and unexpended reserves).
4. A quantum phenomenon occurs and selects a uniformly random team.

Actions and cooldowns

Robots perform actions to interact with the game world, and some cannot be performed multiple times in a single turn or in a short period of time. These actions are:

- Attacking. Attacking can only be done if the robot's action cooldown is less than 10 (can check with the `isActionReady()` method), and can be performed by calling the `attack()` method. This targets a particular occupied square in range, and attacks the robot on that square. This increases the robot's action cooldown as determined by the robot type and the rubble on the robot's square.

- Moving. Moving can only be done if the robot's movement cooldown is less than 10 (can check with the `isMovementReady()` method) and if the ending square is unoccupied. Moving can be performed by calling the `move()` method. This moves the robot in the specified direction. If you wish to check whether a move is valid, you can use the `canMove()` method. This increases the robot's movement cooldown as determined by the robot type and the rubble on the ending square.

- Mining. Miners can mine resources on adjacent squares only if the robot's action cooldown is less than 10 (can check with the `isActionReady()` method), and can be performed by calling the `mineLead()` and `mineGold()` methods. This extracts one unit of metal from the targeted square, and adds it to your team's reserve. This increases the robot's action cooldown as determined by the robot type and the rubble on the robot's square.

- Constructing. Archons can construct other Droids, and Builders can construct other Buildings, but only when their action cooldown is less than 10 (can check with the `isActionReady()` method), and can be performed by calling the `buildRobot()` method.

This deducts the cost of the robot from the player's stockpile, then creates one robot of the specified type in the adjacent square of the specified direction. The `canBuild()` method can be used to check whether a construction is legal. This increases the robot's action cooldown as determined by the robot type and the rubble on the robot's square.

- Mutating. Builders can Mutate other Buildings, but only when the Builder's action cooldown is less than 10 (can check with the `isActionReady()` method), and can be performed by calling the `mutate()` method. This deducts the cost of the Mutation from the player's stockpile, then increases the mutation level of the specified Building. The `canMutation()` method can be used to check whether a Mutation is legal. This increases the action cooldown of the Builder, and also both cooldowns of the Building by 100.

- Repairing. Archons can repair other Droids, and Builders can repair other Buildings increasing the health of the robot that they are repairing, but only when their action cooldown is less than 10 (can check with the `isActionReady()` method), and can be performed by calling the `repair()` method. The `canRepair()` method can be used to check whether the action is legal. This increases the action cooldown of the robot performing a repair as determined by the robot type and the rubble on the robot's square.

- Transforming. Buildings can change modes, but only when their movement and action cooldowns are both less than 10 (can check with the `canTransform()` method), and can be performed by calling the `transform()` method. This transforms a Turret into Portable, or a Portable into a Turret, and increases both of the Building's cooldowns by 100.

- Envisioning. Sages can envision an Anomaly, but only when their action cooldown is less than 10 (can check with the `isActionReady()` method), and can be performed by calling the `envision()` method. The `canEnvision()` method can be used to check whether the action is legal. This increases the robot's action cooldown as determined by the Sage robot type and the rubble on the robot's square.

- Transmuting. Laboratories can transmute Lead into Gold, but only when their action cooldown is less than 10 (can check with the `isActionReady()` method), and can be performed by calling the `transmute()` method. This deducts the applicable amount of

Lead from your stockpile to gain 1 Au. The `canTransmute()` method can be used to check whether there is sufficient Lead available. This increases the robot's action cooldown as determined by the Laboratory robot type and the rubble on the robot's square.

- Communicating. All robots can read the team array by calling `readSharedArray()`, for standard Java bytecode costs. Robots can also update the contents of the shared array with `writeSharedArray()`, which incurs a flat bytecode cost of 100 bytecodes per write.
- Disintegrating. Any robot can call the `disintegrate()` method. This immediately destroys the robot that calls it.

After every turn, the movement and action cooldowns of all robots are decremented by 10.

Robots in-depth

All robots can read and write to the shared team array.

Attack and vision ranges are squared Euclidean distances. A vision range of 36 means that the robot can see a `MapLocation` within a circle of radius up to 6 tiles away.

Buildings

Buildings are upgradeable robots, built by Builders. They are initially built in Prototype mode with 80% of their maximum health, and will automatically switch to Turret mode after being repaired to full health. Buildings can then switch between Turret and Portable modes by calling `transform()`, at the cost of 100 cooldown for both cooldown timers.

Buildings can also be Mutated, which upgrades their properties. Being Mutated by a Builder also incurs a cost of 100 cooldown for both of the Building's cooldown timers.

The cost of Mutating a Building depends on the Building type, as listed in the table below.

	Archon	**Laboratory**	**Watchtower**
Cost	100 Au (nominal)	180 Pb	150 Pb
Cooldown / action	10	10	10
Cooldown / move	24	24	24
Health mutations	600 → 1080 → 1944	100 → 180 → 324	150 → 270 → 486
Attack mutations	−2 → −4 → −6	N/A	4 → 8 → 12
Action radius	(repair) 20	N/A	20
Vision radius	34	53	34
Bytecode limit	20,000	5,000	10,000
Level 2 mutation cost	300 Pb	150 Pb	150 Pb
Level 3 mutation cost	80 Au	25 Au	60 Au

- Archon: the headquarters robot type. Can construct and repair other Droids. Cannot be built.
- Laboratory: the house of the alchemist. Can convert Lead into Gold. Since alchemists prefer solitude, if there are n friendly robots within the Laboratory's vision range, 1 Au can be created by expending $\lfloor A - Be^{-kn} \rfloor$ Pb, where $A = 20, B = 18$. The value of k depends on the Mutation level of the Laboratory, and is 0.02 at level 1, 0.01 at level 2, and 0.005 at level 3. This formula can be accessed with the method `getTransmutationRate()`.
- Watchtower: a defensive reinforcement. Can attack enemy robots that stray within the attack radius.

Droids

Droids are mobile robots, built by Archons.

	Miner	**Builder**	**Soldier**	**Sage**
Cost	50 Pb	40 Pb	75 Pb	20 Au
Cooldown / action	2	10	10	200
Cooldown / move	20	20	16	25
Health	40	30	50	100
Attack	N/A	−2	3	45
Action radius	(mine) 2	(repair) 5	13	25
Vision radius	20	20	20	34
Bytecode limit	10,000	7,500	10,000	10,000

- Miners: the resource collecting robot. Can mine squares for Lead and Gold.
- Builders: the construction robot. Can build new Prototypes, and repair existing Buildings.
- Soldiers: the general-purpose attack robot. Can attack nearby robots.
- Sage: the bringer of Anomalies. Sages are able to produce concentrated attacks, as well as the following Anomalies, but at a very slow rate:
  - Abyss: 99% of resources in all squares within action range are lost.
  - Charge: The air begins to conduct electricity, and all enemy Droids within action range lose health equal to 22% of their maximum health.
  - Fury: A searing solar flare erupts from the sun. All buildings within action range in turret mode have 10% of their max health burned off.
  - Vortex and Singularity are not available to Sages.

Bytecode limits

Robots are also very limited in the amount of computation they are allowed to perform per turn. Bytecodes are a convenient measure of computation in languages like Java, where one Java bytecode corresponds roughly to one basic operation such as “subtract” or “get field”, and a single line of code generally contains several bytecodes (for details see here). Because bytecodes are a feature of the compiled code itself, the same program will always compile to the same bytecodes and thus take the same amount of computation on the same inputs. This is great, because it allows us to avoid using time as a measure of computation, which leads to problems such as nondeterminism. With bytecode cutoffs, re-running the same match between the same bots produces exactly the same results - a feature you will find very useful for debugging.

Every round each robot sequentially takes its turn. If a robot attempts to exceed its bytecode limit (usually unexpectedly, if you have too big of a loop or something), its computation will be paused and then resumed at exactly that point next turn. The code will resume running just fine, but this can cause problems if, for example, you check if a tile is empty, then the robot is cut off and the others take their turns, and then you attempt to move into a now-occupied tile. Instead, use the `Clock.yield()` function to end a robot's turn. This will pause computation where you choose, and resume on the next line next turn.

The per-turn bytecode limits for various robots are as follows:

- Archon: 20,000
- Builder: 7,500
- Miner, Soldier, Sage: 10,000
- Watchtower: 10,000
- Laboratory: 5,000

Some standard functions such as the math library and sensing functions have fixed bytecode costs, available here. More details on this at the end of the spec.

Appendix: Other resources and utilities

Sample player

examplefuncsplayer, a simple player which performs various game actions, is included with battlecode. It includes helpful comments and is a template you can use to see what `RobotPlayer` files should look like.

If you are interested, you may find the full game engine implementation here. This is not at all required, but may be helpful if you are curious about the engine's implementation specifics.

Debugging

Debugging is extremely important. See the debugging tips to learn about our useful debug tools.

Monitoring

The `Clock` class provides a way to identify the current round (`rc.getRoundNum()`), and how many bytecodes have been executed during the current round (`Clock.getBytecodeNum()`).

GameActionExceptions

`GameActionException`s are thrown when something cannot be done. It is often the result of illegal actions such as moving onto another robot, or an unexpected round change in your code. Thus, you must write your player defensively and handle `GameActionException`s judiciously. You should also be prepared for any ability to fail and make sure that this has as little effect as possible on the control flow of your program.

Throwing any `Exception`s cause a bytecode penalty of 500 bytecodes. Unhandled exceptions may paralyze your robot.

Complete documentation

Every function you could possibly use to interact with the game can be found in our javadocs.

Appendix: Other restrictions

Java language usage

Players may use classes from any of the packages listed in `AllowedPackages.txt`, except for classes listed in `DisallowedPackages.txt`. These files can be found here.

Furthermore, the following restrictions apply:

`Object.wait`, `Object.notify`, `Object.notifyAll`, `Class.forName`, and `String.intern` are not allowed. `java.lang.System` only supports `out`, `arraycopy`, and `getProperty`. Furthermore, `getProperty` can only be used to get properties with names beginning with `"bc.testing."`. `java.io.PrintStream` may not be used to open files.

Note that violating any of the above restrictions will cause the robots to explode when run, even if the source files compile without problems.

Memory usage

Robots must keep their memory usage reasonable. If a robot uses more than 8 Mb of heap space during a tournament or scrimmage match, the robot may explode.

More information on bytecode costs

Classes in `java.util`, `java.math`, and scala and their subpackages are bytecode counted as if they were your own code. The following functions in `java.lang` are also bytecode counted as if they were your own code.

`Math.random StrictMath.random String.matches String.replaceAll String.replaceFirst String.split`

The function `System.arraycopy` costs one bytecode for each element copied. All other functions have a fixed bytecode cost. These costs are listed in the `MethodCosts.txt` file. Methods not listed are free. The bytecode costs of `battlecode.common` functions are also listed in the javadoc.

Basic operations like integer comparison and array indexing cost small numbers of bytecodes each.

Bytecodes relating to the creation of arrays (specifically `NEWARRAY`, `ANEWARRAY`, and `MULTIANEWARRAY`; see here for reference) have an effective cost greater than a single bytecode. This is because these instructions, although they are represented as a single bytecode, can be vastly more expensive than other instructions in terms of computational cost. To remedy this, these instructions have a bytecode cost equal to the total length of the instantiated array. Note that this should have minimal impact on the typical team, and is only intended to prevent teams from repeatedly instantiating excessively large arrays.

Appendix: Lingering questions and clarifications

If something is unclear, direct your questions to our Discord where other people may have the same question. We'll update this spec as the competition progresses.

Appendix: Changelog

- Version 2022.2.1.0 (January 26, 2022)
  - Engine:
    - Fix bug impacting correct Vortex Anomaly behavior
  - Qualifying tournament maps released

- Version 2022.2.0.1 (January 20, 2022)
    - Client:
        - Release maps from Sprint II too
    - Specs:
        - Minor typographic and stylistic improvements
- Version 2022.2.0.0 (January 18, 2022)
    - Breaking changes: Sprint II Balancing changes
        - Archons:
            - Decrease level 2 Mutation cost to 300 Pb
        - Watchtowers:
            - Decrease initial cost to 150 Pb
            - Decrease level 2 Mutation cost to 150 Pb
        - Laboratories:
            - Decrease build cost to 180 Pb
            - Decrease level 2 Mutation cost to 150 Pb
            - Change exchange rate constants to $A = 20, B = 18$
            - Change exchange rate constant k to 0.02 for level 1, 0.01 for level 2, 0.005 for level 3
        - Miners:
            - Increase bytecode limit to 10,000
        - Builders:
            - Increase attack to −2 (ie. repair to 2)
        - Sages:
            - Increase action radius to 25
            - Increase Abyss depletion rate to 99%
            - Increase Charge damage rate to 22%
        - Lead regeneration:
            - Increase rate to 5 Pb every 20 turns
    - Client:
        - Visualize repair actions and Anomalies

        - Add health bar
        - Shift attack visualizations so they do not overlap
    - Sprint maps released

- Version 2022.1.0.2 (January 16, 2022)
    - Engine:
        - Trigger tiebreakers if all Archons are destroyed simultaneously during Fury.
        - Add some map validation to main server (thanks @stefangimmillaro!)

- Version 2022.1.0.1 (January 12, 2022)
    - Engine:
        - Remove initial cooldown for all newly built robots
        - Fix bug in lead and gold sensing functions

- Version 2022.1.0.0 (January 11, 2022)
    - Breaking changes: Sprint I Balancing changes
        - Archons:
            - Decrease nominal worth to 100 Au
            - Decrease level 2 Mutation cost to 400 Pb
            - Decrease level 3 Mutation cost to 80 Au
            - Change mutation health to 600 → 1080 → 1944
            - Change mutation repair to 2 → 4 → 6
        - Watchtowers:
            - Decrease level 2 Mutation cost to 200 Pb
            - Decrease level 3 Mutation cost to 60 Au
            - Charge mutation health to 150 → 270 → 486
            - Change mutation damage to 4 → 8 → 12
        - Laboratories:
            - Decrease build cost to 250 Pb
            - Decrease level 2 Mutation cost to 200 Pb
            - Decrease level 3 Mutation cost to 25 Au
            - Change mutation health to 100 → 180 → 324

        - Sages:
            - Decrease build cost to 20 Au
            - Increase action radius to 20
            - Increase vision radius to 34
        - Lead regeneration:
            - Decrease rate to 3 Pb every 40 turns
    - Sprint maps released

- Version 2022.0.3.1 (January 10, 2022)
    - Client:
        - Fix robot selection on (0, 0)
        - Fix gold income
        - Improvements to runner team selection (thanks @jmerle!)
    - Other:
        - Improvements to Javadoc deploy script

- Version 2022.0.3.0 (January 9, 2022)
    - Engine:
        - (New feature) Add overloads for sensing nearby locations with metals
        - Do not dump debug map information at start of match
    - Client:
        - Track HP change on mutations and fix DP display
        - Running averages for lead and gold income
    - Other:
        - Javadocs are 2022, not 2017

- Version 2022.0.2.0 (January 5, 2022)
    - Engine:
        - (New feature) Add `rc.senseNearbyLocationsWithLead` and `rc.senseNearbyLocationsWithGold`
        - Make passive Lead increase occur after Anomalies are processed
    - Client:

    - Fix portable Archon display
    - Fix mutation tracking
    - Update help text
    - Fix Vortex anomaly rendering
  - Other:
    - Improvements to release scripts

- Version 2022.0.1.1 (January 4, 2022)
  - Specs:
    - Clarify that Archons are not guaranteed to have ID at least 10,000
    - Update various URLs
  - Engine:
    - Fix Archon damage to always be negative (ie. repair)
    - Fix maximum value for communication array to actually be $2^{16} - 1$

- Version 2022.0.1.0 (January 3, 2022)
  - Initial release

formatted by Markdeep 1.12