Battlecode 2026: Uneasy Alliances

Table of contents

*Note: **Rat** refers to a generic rat of any type, if baby rat or rat king is not specified.*

Background (lore)

(All animals mentioned are robots. No animals were harmed in the making of this game.)

Deep beneath the abandoned dorms of MIT, thanks to a student letting their failed final project loose on campus, a robotic rat society consisting of clans of baby rats led by noble rat kings has formed. Like all developing societies, there is, of course, conflict. It is not chromatic, but it is dangerous. You have heard tales of many large, hungry robot cats (someone else's failed project, probably) that are on the prowl for sustenance. As such, your society and a nearby society have formed an uneasy alliance.

Before your clan's baby rats begin fighting the cat, you must remember the task you were born with: *protect your noble rat kings.*

Objective

Collect cheese, stay alive, defeat the cats, and choose whether or not to stay cooperating with your enemy. Mindgame your opponent by scoring more points than them to win the match.

Each **match** is split into **3 games**, each of which is split into **2000 rounds** (i.e. robot turns).

An even number of cats will spawn in the map at the start of a game, and they will try to kill the rats of both teams. Every game will start in cooperation mode, where you will work with the other team to attack the cats. If all the cats are defeated, teams will earn points in a way that depends heavily on their contribution to the victory (i.e. how much damage each team did to the cats).

However, at any point in each game, you may also choose to backstab your opponent, ending the cooperation immediately. After a backstab, you must not only defend your rat kings against the cats but also the other rat team. In backstabbing mode, a team's points will be more heavily dependent on the survival of their rat kings and less on their contribution to the defeat of the cats.

Good luck!

Cooperation & Backstabbing

The game has 2 possible phases: cooperation and backstabbing. Each game begins in cooperation. Teams may choose to betray one another over the course of a game. Either team can backstab at any time. A

*Note: **Rat** refers to a generic rat of any type, if baby rat or rat king is not specified.*

backstab is initiated once one team causes any rat from the other team to lose health or when a ratnapping* occurs.

In other words, a backstab is initiated when:

1. A rat attacks (bites) an enemy rat
2. A rat triggers a trap set by the enemy team
3. A rat ratnaps an enemy rat

For example, if a baby rat triggers an enemy trap, the game state changes to backstabbing in the same round, immediately following that rat's turn. (The game state will reset to cooperation at the start of the next game of the match.)

*See "Rat" section for more details

Win conditions

Each match is played as a best of 3 games. For any game, regardless of the state (cooperation or backstabbing), if all rat kings of any team die for any reason, the game ends at that round and that team auto-loses the game. *The only exception is if all rat kings of both teams die in the **same round***, in which case the point system (see below) is used to calculate the winner

If all cats are defeated in cooperation mode, the game ends at that round. The winning team is determined by the team with more points, where points will be awarded to the point system below.

If all cats are defeated in backstabbing mode, the game does not end, as you are not only up against the cats but also the enemy team. You must keep fighting until one team loses all their rat kings (at which point that team auto-loses that game) or the end of the game (round 2000) is reached.
If both teams make it to the end of the game (i.e. both teams have at least 1 surviving rat king for all 2000 rounds), points are awarded to each team based on the point system below.

Point system

When using the point system, the winner of a game is determined by the team with more points. Points are awarded based on the game state (cooperation or backstabbing) and the following factors: damage dealt to cats, number of living rat kings, and amount of cheese transferred by baby rats to the rat king.

If the game concludes in **cooperation** state, each team will be rewarded points based on the following formula:

*Note: **Rat** refers to a generic rat of any type, if baby rat or rat king is not specified.*

round(0.5 * % damage to cat + 0.3 * % of living rat kings + 0.2 * % cheese transferred) [Eq. 1]

If instead the game concludes in **backstabbing** state, each team will be rewarded points based on the following formula:

round(0.3 * % damage done to cat + 0.5 * % of living rat kings + 0.2 * % cheese transferred) [Eq. 2]

Note the following calculations:

- % damage to cat is the amount of damage (i.e. through biting or cat trap triggers) your team did to the cats divided by the total amount of damage both teams did to the cat; 0 if no damage was done to the cats across both teams
- % living rat kings is the number of living rat kings you have at the end of the game divided by the total number of living rat kings across both teams at the end of the game
- % cheese is the amount of cheese your team transferred (via baby rats) to the rat king divided by the total amount of cheese transferred (via baby rats) to the rat king across both teams

Tiebreakers: If both teams end with the same number of points, the following tiebreakers will be applied, in order of priority:

1. Sum of the amount of global cheese the team has at the end of the game
2. Sum of number of total rats (baby rats and rat kings) alive at the end of the game

If teams are still tied after applying tiebreakers, a uniformly random team will be selected.

Map overview

Each Battlecode game will be played on a map. The map is a discrete 2-dimensional rectangular grid, of size ranging between **30×30** and **60×60** inclusive, with **all sides being even**. 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.

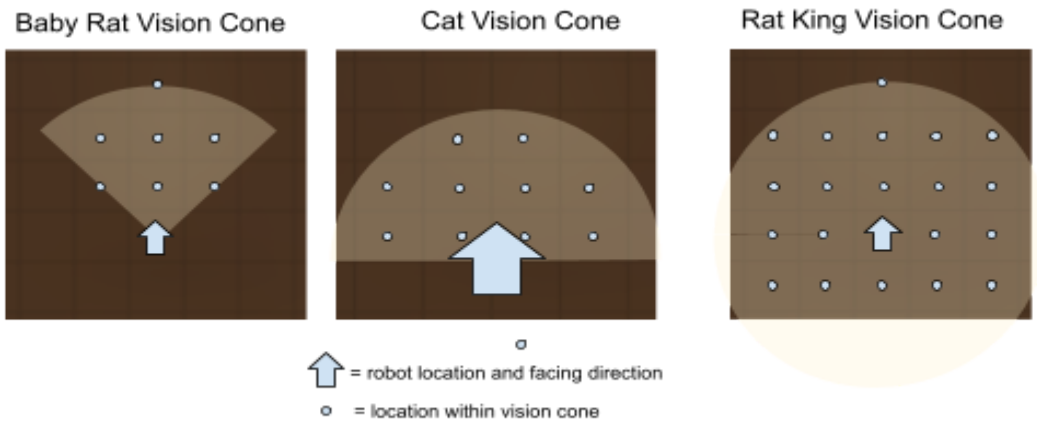
In order to prevent maps from favoring one player over another, it is guaranteed that the world is symmetric either by rotation or horizontal/vertical reflection.

Visibility and Passability

All robots will have a facing direction that is one of the following 8 directions. A robot's vision cone will be centered on the robot's facing direction, and it defines the region in which it can sense map features and other robots. You can sense a multi-tile robot by sensing any of its occupied robot locations, not just the center.

*Note: **Rat** refers to a generic rat of any type, if baby rat or rat king is not specified.*

Baby rats can sense map features (i.e. walls, dirt, cheese, traps, center of cheesemines) and other robots up to $\sqrt{20}$ units away in the 90 degree cone that they are facing towards. Rat kings have a $\sqrt{25}$ unit vision radius in all directions (i.e. 360 degree vision cone).



Note, these examples are with radius = 3, not the actual vision cone radii of the robots in the game

Walls and Dirt

Map walls and dirt are impassable for all unit types. No more than **20%** of a map will be walls. No more than **50%** of a map will be dirt.

Rats and cats may dig (i.e. remove) or place dirt that it owns onto the map but cannot modify walls. Dirt that is removed by a rat is added to a team's global dirt stash, from which any rats on the same team may use to place dirt on an unoccupied tile.

Digging and placing dirt can only be done **within the rat's vision cone** and in tiles that are **directly adjacent** to the robot. Effectively, this means rat kings can dig/place dirt on any tiles within a distance of (3/2 + √2) of their center, and baby rats can dig/place dirt on any tiles within a distance of (0.5 + √2) of their center.

*Note: **Rat** refers to a generic rat of any type, if baby rat or rat king is not specified.*

Resources

Cheese

Cheese is the primary resource in this game. Teams can spend cheese to perform actions like building traps, modifying dirt, spawning rats, etc. Rat kings also need to consume cheese each round to maintain its health.

Cheese can either be accessed by anyone on the team (**global cheese**) or a single baby rat (**raw cheese**). When a baby rat first collects cheese, it is raw, but once it delivers it to a rat king, it becomes global cheese.

Each team starts with **2500** global cheese.

There will be an even number of cheese mines located throughout the map, with a minimum distance of 5 between them. Cheese will be spawned at random in the 9x9 square near these cheese mines. Cheese can only spawn in places that the cat can access, and rats can sense cheese once it is in their vision cone.

Each mine will spawn **5 cheese** on a random location with a probability of $(1 - (1 - 0.01)^r)$ each round, where r is the number of rounds since a cheese was last spawned at the mine. Cheese will always spawn symmetrically across the map. Cheese will not spawn on walls but can spawn on dirt.

When a baby rat collects cheese, the baby rat must bring it back to a rat king in order for the cheese to enter the global cheese pool. Any cheese collected by a rat king is directly converted to global cheese and enters the team's shared cheese pool. When rats spend cheese, their local (raw) cheese is spent first followed by global cheese.

Baby rats carrying cheese are *slowed down by their stash of raw cheese*, with a **movement and action cooldown multiplier of 0.01 * (amount of raw cheese carried)**.

If a baby rat dies, its locally held raw cheese will be dropped at its location.

Dirt

Dirt is the secondary resource in the game. The map may begin with dirt present. 1 block of dirt is gained by digging up an existing block of dirt on the map. Immediately upon digging, blocks of dirt are a global team resource.

Any dirt dug by the cat is permanently removed from the map.

*Note: **Rat** refers to a generic rat of any type, if baby rat or rat king is not specified.*

Units

The Battlecode world contains many kinds of robots. All robots can perform actions such as moving, sensing, and communicating with each other. In each battle, your robots will face one opposing enemy team as well as cat robots.

Each 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 and any cheese it is holding is dropped on the tile where it died.

Robots are assigned unique random IDs no smaller than 10,000. All units on the starting map begin facing the map center.

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.

Two or more robots may not be on the same square unless one is ratnapping the other. When their movement cooldown goes below 10, robots can move onto any of the 8 neighboring squares. All units also have facing directions, which determine the center axis of their vision cone. The 8 possible facing directions are North, Northeast, East, Southeast, South, Southwest, West, and Northwest. When a robot’s turning cooldown goes below 10, robots can turn to any of these facing directions.

To obey map symmetry, all robots (rats and cats) have a notion of **chirality** which affects the order in which they sense locations in the world and (for multi-tile robots) the order in which they retrieve their part locations. For example, for a map with vertical symmetry, one team’s robots will sense the world in increasing x and y values while the other team’s robots will sense the world in decreasing x values and increasing y values.

Cooldowns

All robots have movement, turning, and action cooldowns which determine whether they are able to move, turn, and take actions respectively on a given turn. Actions consist of any non-moving and non-turning related behaviors, not including sensing and communicating. Most actions will incur some

*Note: **Rat** refers to a generic rat of any type, if baby rat or rat king is not specified.*

amount of cooldown increase, and at the beginning of each round of a game all robot movement, turning, and action cooldowns are decreased by 10. Robots can perform movement, turning, and actions once their respective cooldowns are less than 10.

Baby Rat

Baby rats are spawned by rat kings and start with a health of **100 HP**. Once a robot's health reaches 0, it dies. Baby rats have a **forward movement cooldown of 10**, a **strafe movement cooldown of 18**, and a **turning cooldown of 10**. (Strafing is moving in 1 of the 7 non-facing directions.)

Attack

Any rat may bite any enemy rat, rat king, or cat that is at one of the *8 adjacent locations* and is *within its vision radius*. This does **10 damage** to the bitten robot if no cheese is consumed.

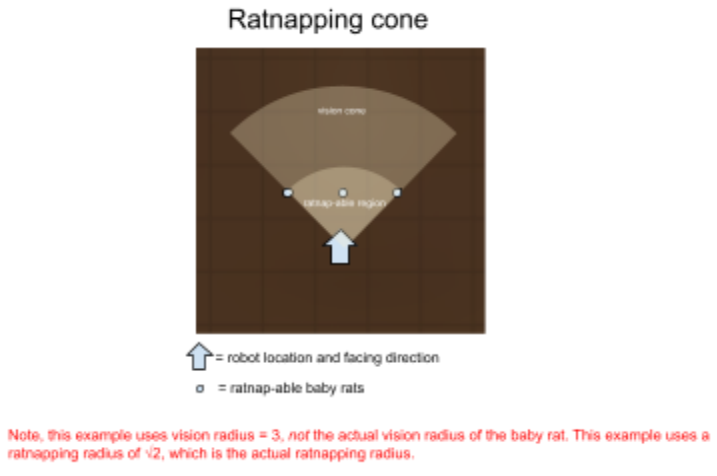
A rat may spend cheese to increase the damage done by a single bite. Spending X amount of cheese on a bite will **increase the damage of the bite by ceil(log X)** damage. Raw cheese will be consumed before global cheese.

Ratnapping / Carrying

Baby rats can ~~kid~~ratnap (i.e. carry) other baby rats around the map. A baby rat can ratnap any allied or enemy baby rat in an adjacent location that it can see, given the baby rat of interest meets one of the following criteria:

- They are facing away (i.e. the other rat cannot sense this rat)
- They have less health
- They are on your team

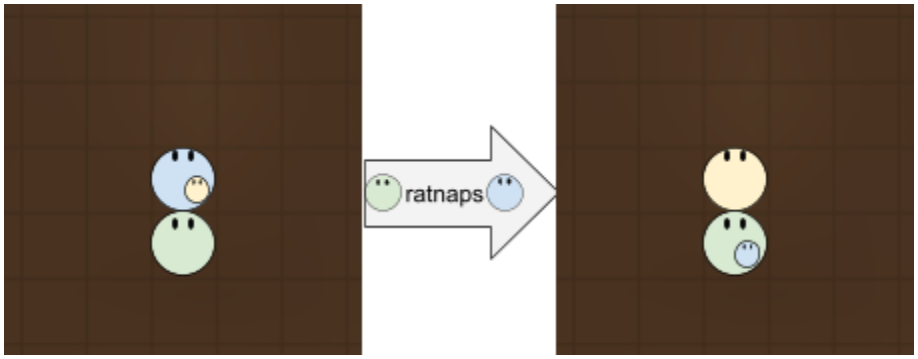
*Note: **Rat** refers to a generic rat of any type, if baby rat or rat king is not specified.*



A baby rat can drop a ratnapped rat at any time on an adjacent location that is passable and not occupied by another rat. If a ratnapped rat is not dropped within 10 rounds, it will be automatically dropped in the tile directly in front of the carrier. If the drop location is occupied or impassable, no drop will occur and instead a swap will take place where the original carrier will be carried by the originally ratnapped rat.

At any time, a rat can ratnap at most one other rat. Upon being ratnapped, a rat will immediately drop any rat it is currently carrying to occupy the spot it was just in. If, for instance, rat A is currently carrying rat B, and rat C begins carrying rat A, rat A will stop carrying rat B.

Any ratnapped rat will assume its original (pre-ratnapped) facing direction when it is dropped.



*Note: **Rat** refers to a generic rat of any type, if baby rat or rat king is not specified.*

Ratnapped rat

The carried rat occupies the same tile as the carrying rat. Rats currently being ratnapped can squeak (see communication section) and sense but cannot perform any other actions, movement, or turning. For as long as a rat is being carried, it is temporarily **stunned**, meaning it is unable to move, place traps, dig/place dirt, etc. However, the ratnapped rat is immune to attacks while it is carried; e.g. if a cat scratches at its location, only the carrier rat will be harmed. The only exception is a cat pounce, which will destroy both the carrier and ratnapped rat.

Throwing

Rats being carried can be thrown, given there is at least 1 empty (passable, non-occupied) space in front of the ratnapper where the thrown rat can land. A thrown rat is immobilized and travels forward at 2 blocks per turn for 4 turns, unless it hits an obstacle (i.e. an impassable tile or another rat) at which point it drops to the ground. Throwing a rat causes 5 * (4 - time it has been in the air) HP loss to the thrown rat and any rat that is hit by the thrown rat.

Upon landing, the flying rat is **stunned** and incurs a movement, turning, and action cooldown of **10** and **30** for the cases of dropping after four turns and for hitting a target respectively. Any rats hit as collateral are not stunned.

A rat being thrown will not trigger traps, cannot be attacked, and cannot pick up cheese or perform any actions except communicating and sensing.

If a thrown baby rat hits a cat on its path, it will get eaten by the cat and the cat will fall asleep for 2 turns.

Placing traps

Rats can place a trap on any unoccupied (i.e. no cheese mine, wall, or dirt) and passable tile. This action **costs 5 cheese** and has an **action cooldown of 5**. A maximum of **25 rat traps per team** can be active at any given time. A rat trap is triggered by an enemy rat stepping in a **radius of** $\sqrt{2}$; a triggered trap **deals 50 damage** to that rat and stuns the rat, preventing it from moving for 2 turns by adding 20 to its movement cooldown.

Your team is immune to the traps you place (you can only get hurt by your opponent's traps). Traps are visible to allied rats but hidden from enemy rats and the cat. All traps are removed after they are triggered once.

*Note: **Rat** refers to a generic rat of any type, if baby rat or rat king is not specified.*

During cooperation, if a rat on the other team triggers your trap, this will be treated as the start of a backstab.

Rats can also use placeCatTrap() on any tile during the **cooperation** phase; rats may not place cat traps after the game has switched to backstabbing state. (Previously placed cat traps will remain however). Placing a cat trap action **costs 10 cheese** and has an **action cooldown of 10**. A cat trap is triggered by a cat stepping in a **radius of** $\sqrt{2}$; a triggered trap **deals 100 damage** to the cat and stuns it, preventing it from moving for 2 turns (by adding 20 to its movement cooldown). Stuns do not stack if there are multiple cat traps triggered at once, but damage does stack. There can be at most **10 cat traps per team** on the map at once.

Rats can remove any traps that are within their vision and are directly adjacent to them. Removing a trap will not refund the cheese spent to build the trap. There is no associated cooldown increase for trap removal.

Digging/Placing Dirt

Rats can remove dirt from any immediately adjacent location that they can sense. Dirt is a global and conserved resource so dirt mined at one location can be used by other rats on the same team. For all rats, both digging and placing dirt costs **10 cheese** and incurs an **action cooldown of 25**. Rats can put dirt from their team's stash at any location that is not occupied by a robot, wall, dirt, cheese, or cheesemine.

Rat King

Every team begins with 1 rat king. It occupies a 3 by 3 square.

Each rat king consumes **3 cheese** at the end of each round, and if the supply of cheese is not enough for any rat king, it will **lose 10 HP** of its health and doesn't consume any cheese. Rat kings start consuming cheese in the order that they were created. Rat kings also start with **500 HP**. They have a **movement cooldown of 40** (in any direction) and a **turning cooldown of 10**. If all of a team's rat kings die, the opposing team immediately wins.

Rat King creation

In addition to the rat king created at the start of the game, if there are 7 or more allied rats within a baby rat's surrounding 3 by 3 square, the center baby rat can upgrade into a rat king. The surrounding 3 by 3 square cannot have any impassable tiles or intersect with existing rat kings or cats.

*Note: **Rat** refers to a generic rat of any type, if baby rat or rat king is not specified.*

Baby rats turning into a rat king retain their current action and movement cooldowns of the center rat. The newly formed rat king's health is the sum of the healths of the individual rats, capped at a rat king's starting health of 500. Any baby rats (allied or enemy) in the surrounding 3 by 3 square of the center baby rat will be destroyed upon rat king formation, and any raw cheese held locally by these baby rats will be incorporated into the new rat king's team's global cheese stash. Following this, any traps whose trigger radius touches the 3 by 3 square will be triggered upon rat king creation.

Upgrading to rat king requires **50 cheese**. A team *may not have more than 5 rat kings* at one time.

Spawning baby rats

Baby rats can be spawned by the rat king at any empty location directly adjacent to the rat king's. The spawned baby rat's initial facing direction will be the parent rat king's current facing direction.

Each robot spawn costs **10 cheese**, increased by *10 more cheese for every 4 baby rats alive*. For example, with 25 baby rats, the cost to spawn another rat would be 10 + 10 * floor(25/4) = 70 cheese. Spawning a baby rat incurs an **action cooldown cost of 10**.

Cats

Cats are a 2 by 2 NPC with **10,000 points of health** and a **movement cooldown of 10**. Cats can also turn to any facing direction and will not turn more than once per turn. For each game, an even number of cats will spawn. In each round, cats act last, after all existing rats and rat kings. **The following mechanics are provided to explain how the cats function, but you will not control the cat.**

Cats can sense map features and other robots up to $\sqrt{30}$ **units away** in the **180 degree cone** they are facing.

Attack

A cat can damage rats in 4 ways:

- **Pounce** - The cat can jump a distance up to 3 units away in any direction, provided there is no wall, dirt, or rat king where it will land. Cats can pounce over obstacles. Upon pouncing, all rats it lands on die immediately, and following this, the relevant cat traps are triggered. The cat only gets trapped by cat traps within the cat trap radius of the 2x2 grid of squares that the cat lands on, and does not get trapped in the middle of the pounce. Pounces incur *double the movement cooldown* of regular cat movement, i.e. a total movement cooldown of **20**.
- **Scratch** - The cat will remain in place and do **50** damage to a rat in its vision cone. This incurs an **action cooldown of 15**.

*Note: **Rat** refers to a generic rat of any type, if baby rat or rat king is not specified.*

- **Feeding** - Enemy rats can throw other rats into the space that the cat is occupying to feed the cat. The cat will immediately kill the thrown rat, then **sleep for 2 turns**. While sleeping the cat will not move or act.
- **Movement** - Cats can move onto squares currently occupied by a baby rat, immediately killing the rat. However, a cat cannot move onto a square occupied by a rat king, but it can attack and kill the rat king by scratching.

Sensing

- The cat has a 180 degree vision cone of $\sqrt{30}$ and can sense all the same things as rats (except cat traps).
- The cat cannot differentiate between rats of different teams.
- Upon hearing a squeak, the cat will know the location of its source.
- Cats know the locations of all walls on the map and can use BFS to get around walls.

Movement

The cat will spawn at the center before each game.

Each turn, the cat can move by either walking or pouncing. The cat can walk one square in the direction it is facing, provided its target destination has enough space and no dirt or rat kings.

Cats may remove dirt from any adjacent square. This has a cooldown of 30, and the removed soil is permanently gone in that game.

A cat can move when its movement cooldown is less than 10. It can also turn up to 90 degrees once per turn, either before or after moving. Cats dig dirt with an **action cooldown of 30**.

Algorithm

Every map will spawn with symmetric waypoints assigned to each cat. These waypoints are not visible to rats.
The cats will act as follows:

- **Explore mode**: Move in a straight line towards the next waypoint. It will dig away dirt if needed or attack the blocking rat; it will use BFS around walls. This is the only mode in which the cat will listen for rat squeaks.
    - If it sees a rat, it will go into **attack mode** on the rat.
    - If it hears a squeak, it will go into **chase mode** on that location.
- **Chase mode**: the cat will move towards the location of sound, pouncing if possible. Upon reaching it, it will enter **search mode**.

*Note: **Rat** refers to a generic rat of any type, if baby rat or rat king is not specified.*

- **Search mode**: The cat will take four turns to make four 90 degree turns, going into **attack mode** on the first rat it sees.
    - If no rats are seen, it will return to **explore mode**.
- **Attack mode**: The cat focuses on a single rat target and ignores all other rats. The cat will scratch at the rat it is targeting if possible. It will attempt to move towards or pounce on the rat.
    - If it can no longer see the rat, it returns to **explore mode**, going back towards the waypoint it was originally headed to.

Communication

Units 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.

Communication is done through a few methods:

1. A global array of 64 integer values between 0-1023
2. Squeaking

Global Array

Any robot may read from the global array at any time, using the readSharedArray() method. However, only rat kings may write to the global array using the writeSharedArray() method.

Squeaking

Rats may communicate with other baby rats by squeaking. Each rat can squeak **at most once per turn** by calling the squeak(int messageContent) method. This message will have the *messageContent integer, the robot's ID, current location at the time it squeaked, and the current round number* in that order. **Cats and any allied rats** within a **sqrt(16) radius** will receive this message.

Squeaks will remain available for other robots to read for 5 rounds before disappearing.

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

*Note: **Rat** refers to a generic rat of any type, if baby rat or rat king is not specified.*

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 bytecode limit for all rats is **17500**, and the bytecode limit for all rat kings is **20000.**

Crossplay between Python and Java is supported for the first time this year! Python counts bytecode differently, so the bytecode of a Python bot is multiplied by **3** to approximately convert to Java bytecode before comparing with the Java bytecode limits. This multiplier only applies to native Python operations and does not apply to the standard functions and Battlecode functions in the text file linked below. Because of the imperfect nature of this conversion factor, it is generally the case that you should work in Java if you're trying to optimize bytecode. In future years, a more accurate conversion might be used.

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.

*Note: **Rat** refers to a generic rat of any type, if baby rat or rat king is not specified.*

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

GameActionExceptions 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 GameActionExceptions 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 Exceptions causes 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.

*Note: **Rat** refers to a generic rat of any type, if baby rat or rat king is not specified.*

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.

Execution Time Limits

Robots must keep their execution time reasonable. While bytecode counting is intended for this exact purpose, some very advanced usages of Java can cause robots to run extremely slowly while remaining within bytecode limits. To reduce strain on our servers, we enforce a total execution time limit per-team. This is aggregated across all robots for the entire game, and reset between games in a single set. If a team exceeds this limit, their team will automatically resign and lose the set. These constants are accessible via game constants, and you can query how much time has elapsed via the Clock class.

This shouldn't affect the majority of players. If you find yourself running into this issue, ensure your Java files are not exceedingly large, as this can cause an issue with execution time.

More information on bytecode costs

*Note: **Rat** refers to a generic rat of any type, if baby rat or rat king is not specified.*

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

- V1.0.1
  - Initial release
- V1.0.2
  - Engine
- V1.0.3
  - Cat traps clarified.
  - Engine: Many engine bugs were fixed. Game constants containing RATKING have been renamed to use RAT_KING instead for consistency. RobotController.canBuildRobot and RobotController.buildRobot were renamed to canBuildRat and buildRat for clarity.
- V1.0.4

*Note: **Rat** refers to a generic rat of any type, if baby rat or rat king is not specified.*

- Specs clarifications:
  - Clarify order of trap triggering for rat king formation
  - Modify cat vision radius diagram
  - Clarify strafing movement cooldown
- Engine changes
  - Fix buildRat error (now computes cost before building rat)
  - Match bytecode limits for unit types
  - Fix phantom cat trap triggers
  - Allow spawned cheese to accumulate on tiles
  - Enforce action cooldowns for digging and placing traps
  - Disable triggering backstabbing state when rat attacks a cat
  - Don't trigger backstab on same-team ratnaps
  - Remove duplicate sensing of multi-tile robots
  - Prevent off-the-grid rat king formation
  - Accurately update rat king counts
  - Enforce initial number of rat kings on map (1 per team)
  - Other miscellaneous bug fixes

*Note: **Rat** refers to a generic rat of any type, if baby rat or rat king is not specified.*