BugWars: Game Description

1 Lore

Two insect colonies are close to each other and must fight for the small amount of resources at their disposal. Who will win the war?

2 The Game

The aim of the game is to destroy all enemy queens. If a team manages to do this, it wins the game. If none of the teams reaches this objective in 3000 rounds, the winner is decided as follows (in order):

- Team who has more queens alive.
- Team such that the sum of HP of their queens is higher.
- Team such that the sum of all their available resources plus the value to all its units is higher.
- Coin toss.

3 Economy

Food is the only resource in the game, and this resource is shared by all the units of the same team. This resource is obtained automatically at the beginning of each round (each team receives 10 units of food), but can also be obtaining by mining food deposits with ants.

4 Map

The map consists of a grid of dimensions between 20×20 and 80×80. Each unit occupies exactly one tile and all tiles of the map are accessible by every unit except if there is a rock or another unit. Rocks can be destroyed if they receive enough damage.

Each unit has a finite vision range, which means that it can only sense the tiles that are close enough to its location. The vision range depends on the unit type (Section 5).

Each tile of the map, even those that are not originally accessible, may contain a food deposit. These deposits can be mined by ants and recover one point of food each round (they never surpass their original value).

Each team starts with 1-5 queens on the map all maps are symmetric. This symmetry may be horizontal, vertical or rotational.

5 Units and Structures

Each unit has a unique identifying number (ID) chosen randomly between 1 and 10.000. Whenever a new unit is created, there is a period of 10 turns in which the unit is still in construction and it cannot be controlled. There are five types of units: Queens, Ants, Beetles, Spiders and Bees. The parameters of each of these types are as indicated in the following table. All distances are shown in squared units. For instance, a unit with 12 attack range at $(0, 0)$ can attack a unit at $(2, 1)$ since $2^2 + 1^2 \leq 12$, but it cannot attack a unit at $(2, 3)$ since $2^2 + 3^2 > 12$.

	Queen	Ant	Beetle	Spider	Bee
Cost	–	150	200	280	300
Max Health	250	15	45	10	100
Attack	–	1	4	3	1
Attack Range	–	5	5	18	5
Minimum Attack Range	–	–	–	9	–
Vision Range	36	24	24	32	36
Movement Cooldown	3	2	2	2	1
Attack Cooldown	–	2	2	2	1
Special Mechanics	Yes	Yes	No	No	No

See Special Mechanics at Section 7).

6 Movement, Attack and Cooldowns

Each unit has a movement and an attack cooldown. They can only move or attack when their respective cooldown is inferior to 1. In particular, if both cooldowns are inferior to 1, the unit can attack and move on the same turn.

Every time that the unit attacks or moves, it adds cooldown to its respective value. The amount added is the one given in the table in Section 5. If the unit moves in a diagonal direction, the movement cooldown addded is multiplied by 1.4142, which is approximately $\sqrt{2}$. At the beginning of every turn, both cooldowns decrease by 1.

Units cannot attack over obstacles or barricades. More precisely, if the segment that goes from the center of the unit's tile to the center of the attacking target passes through the interior of a tile with a rock, the attack cannot be performed. However, it is allowed that the segment passes through the boundary of such tiles. For inatance, a unit at $(0, 0)$ can attack a unit at $(1, 1)$ even if there is a rock at $(0, 1)$.

7 Special Mechanics

Queens and Ants can receive other types of instructions besides moving and attacking:

Queen:

- It can spawn Ants, Beetles, Spiders or Bees on adjacent locations if the team has enough resources to do so.
- Each round, queens can heal 1 HP to a unit at distance at most 5 (in squared units).

Ants

- Ants can mine up to three units of food from an adjacent location. This food goes directly to the team's shared resource stash.

8 Communication and Vision

Each unit can only sense tiles inside its vision radius. Units can access some of the details about these tiles using the methods at *UnitController* (check the documentation).

Each unit runs independently and they do not share memory. For instance, the tiles that are visible by a given unit might not be visible for another one. However, there is a way to communicate between units: each team has a shared array of 100000 32-bit integers initialized to 0 that can be accessed by all units of the same team. Each unit can read or write on an arbitrary amount of components using the *read* and *write* functions of *UnitController*.

9 Energy

Energy is an approximate indicator of the number of basic instructions that each unit performs. More precisely, each bytecode instruction performed by a given unit consumes one unit of energy (except internal operations of the methods provided in the documentation, these consume a constant amount of bytecode which is given in the documentation). For users not familiarized with Java, it is not necessary to know how bytecode works, however it is good to have in mind that it is somewhat proportional to the number of code instructions. The amount of energy consumed up to a certain instruction can be accessed at any time using the methods in *UnitController*. Whenever a unit surpasses the amount of energy allowed (currently set to 15000), the unit pauses and continues running the remaining instructions during its next turn.

10 User Instructions

Players must fill the *run* method of the *UnitPlayer* class, which is run independently by all units of the player's team. This function has a *UnitController* as input, which is used to give orders to the given unit and to get information of the visible tiles or the common array, among others (check the documentation for more details).

Run does the following: Whenever a new unit is created and finishes its construction period (10 turns), its *run* method is executed until either the unit finishes its turn by calling the *yield* method, or when it surpasses the amount of allowed energy. If a unit returns from its run method, it dies. Because of this it is suggested to keep its instructions inside a *while(true)* statement, and to finish each iteration with a call to *yield* (check *nullplayer* and *demoplayer*).

11 Implementation info.

You may skip this section if you're not enough familiarized with Java.

Each unit is run in an independent thread. Each of these threads gets reactivated every time the unit is scheduled (once every round, following the same relative order every round). For safety reasons, it

is forbidden to use any method or class outside *java/lang*, *java/math* and *java/util* (and some of the sub-classes of these). It is also forbidden to use static variables (which include switch statements, since they internally do so).