**BattleCode Command and Variable Cheat Sheet**

**Units**

**Workers** - these are the only units that you start with. They can clone themselves and lay down Factories or Rockets.

**Factories** - these are "buildings" that produce any type of unit (except Rockets).

**Rangers** - the "shooting" unit - these units can deal damage at long range.

**Knights** - fast, durable melee units.

**Mages** - "Glass cannons." Extremely low health, but can deal devastating splash damage if they get close to the enemy. Can also unlock an ability that lets them teleport.

**Healers** - aptly named, they heal units.

**Rockets** - these are the vehicles that can get you to Mars. They're capable of holding 8 (or more, if you upgrade them) units, and can take off to bring them over to Mars

**Unit Commands:**

1. \_\_init\_\_(self)
   * 1. Use environment variables to connect to the manager.
     2. :type self: GameController
     3. :rtype: GameController
2. apply\_turn(self, turn)
   * 1. :type self: GameController
     2. :type turn: TurnMessage
     3. :rtype: TurnApplication
3. asteroid\_pattern(self)
   * 1. The asteroid strike pattern on Mars.
     2. :type self: GameController
     3. :rtype: AsteroidPattern
4. attack(self, robot\_id, target\_unit\_id)
   * 1. Attacks the robot, dealing the unit's standard amount of damage.
     2. GameError::NoSuchUnit - the unit does not exist (inside the vision range).
     3. GameError::TeamNotAllowed - the unit is not on the current player's team.
     4. GameError::InappropriateUnitType - the unit is a healer, or not a robot.
     5. GameError::InvalidAction - the robot cannot attack that location.
     6. :type self: GameController
     7. :type robot\_id: int
     8. :type target\_unit\_id: int
     9. :rtype: None
5. begin\_snipe(self, ranger\_id, location)
   * 1. Begins the countdown to snipe a given location.
     2. Maximizes the units attack and movement heats until the ranger has sniped.
     3. The ranger may begin the countdown at any time, including resetting the countdown to snipe a different location.
6. blink(self, mage\_id, location)
   * 1. Blinks the mage to the given location.
7. blueprint(self, worker\_id, structure\_type, direction)
   * 1. Blueprints a unit of the given type in the given direction.
     2. Subtract cost of that unit from the team's resource pool.
8. build(self, worker\_id, blueprint\_id)
   * 1. Builds a given blueprint, increasing its health by the worker's build

Amount. If raised to maximum health, the blueprint becomes a completed

structure.

1. can\_attack(self, robot\_id, target\_unit\_id)
   * 1. Whether the robot can attack the given unit, without taking into

account the unit's attack heat. Takes into account only the unit's

attack range, and the location of the unit.

1. can\_blink(self, mage\_id, location)
   * 1. Whether the mage can blink to the given location, without taking into

account the mage's ability heat. Takes into account only the mage's

ability range, the map terrain, positions of other units, and the edge

of the game map.

1. can\_blueprint(self, worker\_id, unit\_type, direction)
   * 1. Whether the worker can blueprint a unit of the given type. The worker

can only blueprint factories, and rockets if Rocketry has been

researched. The team must have sufficient karbonite in its resource

pool. The worker cannot already have performed an action this round.

1. can\_build(self, worker\_id, blueprint\_id)
   * 1. Whether the worker can build a blueprint with the given ID. The worker

and the blueprint must be adjacent to each other. The worker cannot

already have performed an action this round.

1. can\_harvest(self, worker\_id, direction)
   * 1. Whether the worker is ready to harvest, and the given direction contains

karbonite to harvest. The worker cannot already have performed an action

this round.

1. can\_heal(self, healer\_id, target\_robot\_id)
   * 1. Whether the healer can heal the given robot, without taking into

account the healer's attack heat. Takes into account only the healer's

attack range, and the location of the robot.

1. can\_javelin(self, knight\_id, target\_unit\_id)
   * 1. Whether the knight can javelin the given robot, without taking into

account the knight's ability heat. Takes into account only the knight's

ability range, and the location of the robot.

1. can\_launch\_rocket(self, rocket\_id, destination)
   * 1. Whether the rocket can launch into space. The rocket can launch if the

it has never been used before.

1. can\_load(self, structure\_id, robot\_id)
   * 1. Whether the robot can be loaded into the given structure's garrison. The robot

must be ready to move and must be adjacent to the structure. The structure

and the robot must be on the same team, and the structure must have space.

1. can\_move(self, robot\_id, direction)
   * 1. Whether the robot can move in the given direction, without taking into

account the unit's movement heat. Takes into account only the map

terrain, positions of other robots, and the edge of the game map.

1. can\_overcharge(self, healer\_id, target\_robot\_id)
   * 1. Whether the healer can overcharge the given robot, without taking into

account the healer's ability heat. Takes into account only the healer's

ability range, and the location of the robot.

1. can\_produce\_robot(self, factory\_id, robot\_type)
   * 1. Whether the factory can produce a robot of the given type.
     2. The factory must not currently be producing a robot, and the team must have

sufficient resources in its resource pool.

1. can\_repair(self, worker\_id, structure\_id)
   * 1. Whether the given worker can repair the given strucutre.
     2. Tests that the worker is able to execute a worker action, that the structure is built, and that the structure is within range.
2. can\_replicate(self, worker\_id, direction)
   * 1. Whether the worker is ready to replicate.
     2. Tests that the worker's ability heat is sufficiently low, that the team has sufficient

karbonite in its resource pool, and that the square in the given

direction is empty.

1. can\_sense\_location(self, location)
   * 1. Whether the location is within the vision range.
2. can\_sense\_unit(self, id)
   * 1. Whether there is a unit with this ID within the vision range.
3. can\_unload(self, structure\_id, direction)
   * 1. Tests whether the given structure is able to unload a unit in the

given direction.

* + 1. There must be space in that direction, and the unit must be ready to move.

1. current\_duration\_of\_flight(self)
   * 1. The current duration of flight if a rocket were to be launched this

round.

* + 1. Does not take into account any research done on rockets.

1. disintegrate\_unit(self, unit\_id)
   * 1. Disintegrates the unit and removes it from the map.
     2. If the unit is a factory or a rocket, also disintegrates any units garrisoned inside it.
2. end\_turn(self)
   * 1. :type self: GameController
     2. :rtype: TurnMessage
3. harvest(self, worker\_id, direction)
   * 1. Harvests up to the worker's harvest amount of karbonite from the given

location, adding it to the team's resource pool.

1. heal(self, healer\_id, target\_robot\_id)
   * 1. Heals the robot, dealing the healer's standard amount of "damage".
2. initial\_start\_turn\_message(self)
   * 1. :type self: GameController
     2. :rtype: InitialTurnApplication
3. is\_attack\_ready(self, robot\_id)
   * 1. Whether the robot is ready to attack.
     2. Tests whether the robot's attack heat is sufficiently low.
4. is\_blink\_ready(self, mage\_id)
   * 1. Whether the mage is ready to blink.
     2. Tests whether the mage's ability heat is sufficiently low.
5. is\_heal\_ready(self, healer\_id)
   * 1. Whether the healer is ready to heal.
     2. Tests whether the healer's attack heat is sufficiently low.
6. is\_javelin\_ready(self, knight\_id)
   * 1. Whether the knight is ready to javelin.
     2. Tests whether the knight's ability heat is sufficiently low.
7. is\_move\_ready(self, robot\_id)
   * 1. Whether the robot is ready to move.
     2. Tests whether the robot's attack heat is sufficiently low.
8. is\_occupiable(self, location)
   * 1. Whether the location is clear for a unit to occupy, either by movement or by construction.
     2. GameError::InvalidLocation - the location is outside the vision range.
9. is\_over(self)
   * 1. Is the game over?
10. is\_overcharge\_ready(self, healer\_id)
    * 1. Whether the healer is ready to overcharge.
      2. Tests whether the healer's ability heat is sufficiently low.
11. javelin(self, knight\_id, target\_unit\_id)
    * 1. Javelins the robot, dealing the amount of ability damage.
12. karbonite\_at(self, location)
    * 1. The karbonite at the given location.
13. launch\_rocket(self, rocket\_id, location)
    * 1. Launches the rocket into space.
      2. If the destination is not on the map of the other planet, the rocket flies off, never to be seen again.
14. load(self, structure\_id, robot\_id)
    * 1. Loads the robot into the garrison of the structure.
15. move\_robot(self, robot\_id, direction)
    * 1. Moves the robot in the given direction.
16. next\_turn(self)
    * 1. Send the moves from the current turn and wait for the next turn.
17. orbit\_pattern(self)
    * 1. The orbit pattern that determines a rocket's flight duration.
18. overcharge(self, healer\_id, target\_robot\_id)
    * 1. Overcharges the robot, resetting the robot's cooldowns.
19. planet(self)
    * 1. :type self: GameController
      2. :rtype: Planet
20. produce\_robot(self, factory\_id, robot\_type)
    * 1. Starts producing the robot of the given type.
21. queue\_research(self, branch)
    * 1. Adds a branch to the back of the queue, if it is a valid upgrade, and

starts research if it is the first in the queue.

* + 1. Returns whether the branch was successfully added.

1. repair(self, worker\_id, structure\_id)
   * 1. Commands the worker to repair a structure, replenishing health to it.
     2. This can only be done to structures which have been fully built.
2. replicate(self, worker\_id, direction)
   * 1. Replicates a worker in the given direction.
     2. Subtracts the cost of the worker from the team's resource pool.
3. research\_info(self)
   * 1. The research info of the current team, including what branch is

currently being researched, the number of rounds left.

* + 1. Note that mutating this object by resetting or queueing research

1. reset\_research(self)
   * 1. Resets the research queue to be empty.
     2. Returns true if the queue was not empty before, and false otherwise.
2. rocket\_landings(self)
   * 1. The landing rounds and locations of rockets in space that belong to the

current team.

* + 1. Note that mutating this object does NOT have any effect on the actual

game.

* + 1. You MUST call the mutators in world!!

1. round(self)
   * 1. Returns round number
2. start\_game(self, player)
   * 1. Starts the game
3. start\_turn(self, turn)
   * 1. Starts the turn
4. unload(self, structure\_id, direction)
   * 1. Unloads a robot from the garrison of the specified structure into an

adjacent space.

* + 1. Robots are unloaded in the order they were loaded.

1. winning\_team(self)
   * 1. Returns the winning team
2. write\_team\_array(self, index, value)
   * 1. Writes the value at the index of this planet's team array.
     2. (The message teams can pass between planets)

**Unit Info:**

**Ancestors (in MRO)**

------------------

bc.Unit

builtins.object

**Static methods**

--------------

1. \_\_init\_\_(self)
   * 1. A single unit in the game and all its associated properties.
     2. :type self: Unit
     3. :rtype: Unit
2. clone(self)
   * 1. Deep-copy a Unit
     2. :type self: Unit
     3. :rtype: Unit
3. damage(self)
   * 1. The damage inflicted by the robot during a normal attack.
     2. Errors if the unit is not a robot.
     3. :type self: Unit
     4. :rtype: int
4. from\_json(s)
   * 1. Deserialize a Unit from a JSON string
     2. :type s: str
     3. :rtype: Unit
5. structure\_garrison(self)
   * 1. Returns the units in the structure's garrison.
     2. #Errors if the unit is not a structure.
     3. :type self: Unit
     4. :rtype: VecUnitID
6. to\_json(self)
   * 1. Serialize a Unit to a JSON string
     2. :type self: Unit
     3. :rtype: str

**Instance variables**

------------------

1. ability\_cooldown
   * 1. The active ability cooldown.
     2. Errors if the unit is not a robot.
2. ability\_heat
   * 1. The active ability heat.
3. ability\_range
   * 1. The active ability range.
4. attack\_cooldown
   * 1. The attack cooldown.
5. attack\_heat
   * 1. The attack heat.
6. attack\_range
   * 1. The attack range.
7. healer\_self\_heal\_amount
   * 1. The amount of health passively restored to itself each round.
8. health
   * 1. The current health.
9. id
   * 1. The unique ID of a unit.
10. is\_ability\_unlocked
    * 1. Whether the active ability is unlocked.
11. knight\_defense
    * 1. The amount of damage resisted by a knight when attacked.
12. location
    * 1. The location of the unit.
13. max\_health
    * 1. The maximum health.
14. movement\_cooldown
    * 1. The movement cooldown.
15. movement\_heat
    * 1. The movement heat.
16. ranger\_cannot\_attack\_range
    * 1. The range within a ranger cannot attack.
17. ranger\_is\_sniping
    * 1. Whether the ranger is sniping.
18. research\_level
    * 1. The current research level.
19. rocket\_is\_used
    * 1. Whether the rocket has already been used.
20. rocket\_travel\_time\_decrease
    * 1. The number of rounds the rocket travel time is reduced by compared

to the travel time determined by the orbit of the planets.

1. structure\_is\_built
   * 1. Whether this structure has been built.
2. structure\_max\_capacity
   * 1. The max capacity of a structure.
3. team
   * 1. The team the unit belongs to.
4. unit\_type
   * 1. The unit type.
5. vision\_range
   * 1. The unit vision range.
6. worker\_build\_health
   * 1. The health restored when building or repairing a structure.
7. worker\_harvest\_amount
   * 1. The maximum amount of karbonite harvested from a deposit in one turn.
8. worker\_has\_acted
   * 1. Whether the worker has already acted (harvested, blueprinted, built, or

repaired) this round.

**Game Map**

Ancestors (in MRO)

------------------

bc.GameMap

builtins.object

**Static methods**

--------------

1. \_\_init\_\_(self)
   * 1. :type self: GameMap
     2. :rtype: GameMap
2. clone(self)
   * 1. Deep-copy a GameMap
     2. :type self: GameMap
     3. :rtype: GameMap
3. from\_json(s)
   * 1. Deserialize a GameMap from a JSON string
     2. :type s: str
     3. :rtype: GameMap
4. test\_map()
   * 1. :rtype: GameMap
5. to\_json(self)
   * 1. Serialize a GameMap to a JSON string
     2. :type self: GameMap
     3. :rtype: str

**Location**

Ancestors (in MRO)

------------------

bc.Location

builtins.object

**Static methods**

--------------

1. \_\_init\_\_(self)
   * 1. :type self: Location
     2. :rtype: Location
2. clone(self)
   * 1. Deep-copy a Location
     2. :type self: Location
     3. :rtype: Location
3. from\_json(s)
   * 1. Deserialize a Location from a JSON string
     2. :type s: str
     3. :rtype: Location
4. to\_json(self)
   * 1. Serialize a Location to a JSON string
     2. :type self: Location
     3. :rtype: str

**Map Location**

Ancestors (in MRO)

------------------

bc.MapLocation

builtins.object

**Static methods**

--------------

1. \_\_init\_\_(self, planet, x, y)
   * 1. Create a new MapLocation.
     2. :type self: MapLocation
     3. :type planet: Planet
     4. :type x: int
     5. :type y: int
     6. :rtype: MapLocation
2. add(self, direction)
   * 1. :type self: MapLocation
     2. :type direction: Direction
     3. :rtype: MapLocation
3. clone(self)
   * 1. Deep-copy a MapLocation
     2. :type self: MapLocation
     3. :rtype: MapLocation
4. from\_json(s)
   * 1. Deserialize a MapLocation from a JSON string
     2. :type s: str
     3. :rtype: MapLocation
5. to\_json(self)
   * 1. Serialize a MapLocation to a JSON string
     2. :type self: MapLocation
     3. :rtype: str

**Instance variables**

------------------

1. planet
   * 1. The planet lol.
     2. :type self: MapLocation
     3. :rtype: Planet
2. x
   * 1. The x coordinate of the map location.
     2. :type self: MapLocation
     3. :rtype: int
3. y
   * 1. The y coordinate of the map location.
     2. :type self: MapLocation
     3. :rtype: int

**Orbit Pattern**

Ancestors (in MRO)

------------------

bc.OrbitPattern

builtins.object

**Static methods**

--------------

1. \_\_init\_\_(self)
   * 1. :type self: OrbitPattern
     2. :rtype: OrbitPattern
2. from\_json(s)
   * 1. Deserialize a OrbitPattern from a JSON string
     2. :type s: str
     3. :rtype: OrbitPattern
3. to\_json(self)
   * 1. Serialize a OrbitPattern to a JSON string
     2. :type self: OrbitPattern
     3. :rtype: str

**Instance variables**

------------------

1. amplitude
   * 1. :type self: OrbitPattern
     2. :rtype: int
2. center
   * 1. :type self: OrbitPattern
     2. :rtype: int
3. period
   * 1. :type self: OrbitPattern
     2. :rtype: int

**Planet**

An enumeration.

Ancestors (in MRO)

------------------

bc.Planet

enum.IntEnum

builtins.int

enum.Enum

builtins.object

Class variables

---------------

1. Earth
2. Mars

**Player**

Ancestors (in MRO)

------------------

bc.Player

builtins.object

**Static methods**

--------------

1. \_\_init\_\_(self, team, planet)
   * 1. :type self: Player
     2. :type team: Team
     3. :type planet: Planet
     4. :rtype: Player
2. clone(self)
   * 1. Deep-copy a Player
     2. :type self: Player
     3. :rtype: Player
3. from\_json(s)
   * 1. Deserialize a Player from a JSON string
     2. :type s: str
     3. :rtype: Player
4. to\_json(self)
   * 1. Serialize a Player to a JSON string
     2. :type self: Player
     3. :rtype: str

**Instance variables**

------------------

1. planet
   * 1. :type self: Player
     2. :rtype: Planet
2. team
   * 1. :type self: Player
     2. :rtype: Team

**Research Info**

Ancestors (in MRO)

------------------

bc.ResearchInfo

builtins.object

**Static methods**

--------------

1. \_\_init\_\_(self)
   * 1. :type self: ResearchInfo
     2. :rtype: ResearchInfo

**Rocket Landing**

Ancestors (in MRO)

------------------

bc.RocketLanding

builtins.object

**Static methods**

--------------

1. \_\_init\_\_(self, rocket\_id, destination)
   * 1. :type self: RocketLanding
     2. :type rocket\_id: int
     3. :type destination: MapLocation
     4. :rtype: RocketLanding
2. clone(self)
   * 1. Deep-copy a RocketLanding
     2. :type self: RocketLanding
     3. :rtype: RocketLanding
3. from\_json(s)
   * 1. Deserialize a RocketLanding from a JSON string
     2. :type s: str
     3. :rtype: RocketLanding
4. to\_json(self)
   * 1. Serialize a RocketLanding to a JSON string
     2. :type self: RocketLanding
     3. :rtype: str

**Instance variables**

------------------

1. destination
   * 1. :type self: RocketLanding
     2. :rtype: MapLocation
2. rocket\_id
   * 1. :type self: RocketLanding
     2. :rtype: int

**Rocket Landing Info**

Ancestors (in MRO)

------------------

bc.RocketLandingInfo

builtins.object

**Static methods**

--------------

1. \_\_init\_\_(self)
   * 1. :type self: RocketLandingInfo
     2. :rtype: RocketLandingInfo