Package dev.robocode.tankroyale.botapi

Class BaseBot

java.lang.Object
    dev.robocode.tankroyale.botapi.BaseBot

All Implemented Interfaces:

IBaseBot

Direct Known Subclasses:

Bot

```
public abstract class BaseBot
extends java.lang.Object
implements IBaseBot
```

Abstract bot class that takes care of communication between the bot and the server and sends notifications through the event handlers. Most bots can inherit from this class to get access to basic methods.

Field Summary

Fields inherited from interface dev.robocode.tankroyale.botapi.IBaseBot

`MAX_NUMBER_OF_TEAM_MESSAGES_PER_TURN, TEAM_MESSAGE_MAX_SIZE`

Constructor Summary

Constructors

Modifier	Constructor	Description
protected	`BaseBot()`	Constructor for initializing a new instance of the BaseBot class.
protected	`BaseBot(BotInfo botInfo)`	Constructor for initializing a new instance of the BaseBot class.
protected	`BaseBot(BotInfo botInfo, java.net.URI serverUrl)`	Constructor for initializing a new instance of the BaseBot class.
protected	`BaseBot(BotInfo botInfo, java.net.URI serverUrl, java.lang.String serverSecret)`	Constructor for initializing a new instance of the BaseBot class.

Method Summary

All Methods Instance Methods Concrete Methods

Modifier and Type	Method	Description
boolean	`addCustomEvent(Condition condition)`	Adds an event handler that will be automatically triggered `IBaseBot.onCustomEvent(dev.robocode.tankroyale.botapi` when the `Condition.test()` returns `true`.
void	`broadcastTeamMessage (java.lang.Object message)`	Broadcasts a message to all teammates. When the message is send, it is serialized into a JSON represer all public fields, and only public fields, are being serialized into representation as a DTO (data transfer object). The maximum team message size limit is defined by `IBaseBot.TEAM_MESSAGE_MAX_SIZE`, which is set to 32768 byte

double	**calcMaxTurnRate**(double speed)	Calculates the maximum turn rate for a specific speed.
void	**clearEvents**()	Clears out any pending events in the bot's event queue immedi
int	**getArenaHeight**()	Height of the arena measured in units.
int	**getArenaWidth**()	Width of the arena measured in units.
java.awt.Color	**getBodyColor**()	Returns the color of the body.
java.awt.Color	**getBulletColor**()	Returns the color of the fired bullets.
java.util.Collection<**BulletState**>	**getBulletStates**()	Current bullet states.
double	**getDirection**()	Current driving direction of the bot in degrees.
int	**getEnemyCount**()	Number of enemies left in the round.
double	**getEnergy**()	Current energy level.
int	**getEventPriority** (java.lang.Class<**BotEvent**> eventClass)	Returns the event priority for a specific event class.
java.util.List<**BotEvent**>	**getEvents**()	Returns an ordered list containing all events currently in the b
double	**getFirepower**()	Returns the firepower.
java.lang.String	**getGameType**()	Game type, e.g.
java.awt.Graphics2D	**getGraphics**()	Gets a graphics object that the bot can paint debug informatio
java.awt.Color	**getGunColor**()	Returns the color of the gun.
double	**getGunCoolingRate**()	Gun cooling rate.
double	**getGunDirection**()	Current direction of the gun in degrees.
double	**getGunHeat**()	Current gun heat.
double	**getGunTurnRate**()	Returns the gun turn rate in degrees per turn.
double	**getMaxGunTurnRate**()	Returns the maximum gun turn rate in degrees per turn.
int	**getMaxInactivityTurns**()	The maximum number of inactive turns allowed the bot will be game for being inactive.
double	**getMaxRadarTurnRate**()	Returns the maximum radar turn rate in degrees per turn.
double	**getMaxSpeed**()	Returns the maximum speed in units per turn.
double	**getMaxTurnRate**()	Returns the maximum turn rate of the bot in degrees per turn.
int	**getMyId**()	Unique id of this bot, which is available when the game has sta
int	**getNumberOfRounds**()	The number of rounds in a battle.
java.awt.Color	**getRadarColor**()	Returns the color of the radar.
double	**getRadarDirection**()	Current direction of the radar in degrees.
double	**getRadarTurnRate**()	Returns the radar turn rate in degrees per turn.
int	**getRoundNumber**()	Current round number.
java.awt.Color	**getScanColor**()	Returns the color of the scan arc.
double	**getSpeed**()	The current speed measured in units per turn.
double	**getTargetSpeed**()	Returns the target speed in units per turn.
java.util.Set<java.lang.Integer>	**getTeammateIds**()	Returns the ids of all teammates.
int	**getTimeLeft**()	The number of microseconds left of this turn before the bot wil
java.awt.Color	**getTracksColor**()	Returns the color of the tracks.

int	getTurnTimeout()	The turn timeout is important as the bot needs to take action by IBaseBot.go() before the turn timeout occurs.
java.awt.Color	**getTurretColor**()	Returns the color of the gun turret.
java.lang.String	**getVariant**()	The game variant, which is "Tank Royale".
java.lang.String	**getVersion**()	Game version, e.g.
double	**getX**()	Current X coordinate of the center of the bot.
double	**getY**()	Current Y coordinate of the center of the bot.
void	**go**()	Commits the current commands (actions), which finalizes the c
boolean	**isAdjustGunForBodyTurn**()	Checks if the gun is set to adjust for the bot turning, i.e.
boolean	**isAdjustRadarForBodyTurn**()	Checks if the radar is set to adjust for the body turning, i.e.
boolean	**isAdjustRadarForGunTurn**()	Checks if the radar is set to adjust for the gun turning, i.e.
boolean	**isDebuggingEnabled**()	Flag indicating if graphical debugging is enabled and hence if IBaseBot.getGraphics() can be used for debug painting.
boolean	**isDisabled**()	Specifies if the bot is disabled, i.e., when the energy is zero.
boolean	**isStopped**()	Checks if the movement has been stopped.
boolean	**isTeammate**(int botId)	Checks if the provided bot id is a teammate or not.
boolean	**removeCustomEvent**(**Condition** condition)	Removes triggering a custom event handler for a specific condi previously added with IBaseBot.addCustomEvent(dev.robocode.tankroyale.botap
void	**sendTeamMessage**(int teammateId, java.lang.Object message)	Sends a message to a specific teammate. When the message is sent, it is serialized into a JSON represen all public fields, and only public fields, are being serialized into representation as a DTO (data transfer object). The maximum team message size limit is defined by IBaseBot.TEAM_MESSAGE_MAX_SIZE, which is set to 32768 byte
void	**setAdjustGunForBodyTurn**(boolean adjust)	Sets the gun to adjust for the botÂ´s turn when setting the gun
void	**setAdjustRadarForBodyTurn**(boolean adjust)	Sets the radar to adjust for the body's turn when setting the ra
void	**setAdjustRadarForGunTurn**(boolean adjust)	Sets the radar to adjust for the gun's turn when setting the rad
void	**setBodyColor**(java.awt.Color color)	Sets the color of the body.
void	**setBulletColor**(java.awt.Color color)	Sets the color of the fired bullets.
void	**setEventPriority**(java.lang.Class<**BotEvent**> eventClass, int priority)	Changes the event priority for an event class.
boolean	**setFire**(double firepower)	Sets the gun to fire in the direction that the gun is pointing wit firepower.
void	**setFireAssist**(boolean enable)	Enables or disables fire assistance explicitly.
void	**setGunColor**(java.awt.Color color)	Sets the color of the gun.
void	**setGunTurnRate**(double gunTurnRate)	Sets the turn rate of the gun, which can be positive and negativ
void	**setInterruptible**(boolean interruptible)	Call this method during an event handler to control continuing handler, when a new event occurs again for the same event har an earlier event.
void	**setMaxGunTurnRate**(double maxGunTurnRate)	Sets the maximum turn rate which applies to turn the gun to th

void	**setMaxTurnRate**(double maxTurnRate)	Sets the maximum turn rate which applies to turn the bot to th
void	**setRadarColor**(java.awt.Color color)	Sets the color of the radar.
void	**setRadarTurnRate**(double radarTurnRate)	Sets the turn rate of the radar, which can be positive and negat
void	**setRescan**()	Sets the bot to rescan with the radar.
void	**setResume**()	Sets the bot to resume movement after having been stopped, e
void	**setScanColor**(java.awt.Color color)	Sets the color of the scan arc.
void	**setStop**()	Sets the bot to stop all movement including turning the gun an
void	**setStop**(boolean overwrite)	Sets the bot to stop all movement including turning the gun an
void	**setTargetSpeed**(double targetSpeed)	Sets the new target speed for the bot in units per turn.
void	**setTracksColor**(java.awt.Color color)	Sets the color of the tracks.
void	**setTurnRate**(double turnRate)	Sets the turn rate of the bot, which can be positive and negativ
void	**setTurretColor**(java.awt.Color color)	Sets the color of the gun turret.
void	**start**()	The method used to start running the bot.

Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Methods inherited from interface dev.robocode.tankroyale.botapi.IBaseBot

`bearingTo, calcBearing, calcDeltaAngle, calcGunBearing, calcRadarBearing, directionTo, distanceTo, gunBearingTo, normalizeAbsoluteAngle, normalizeRelativeAngle, onBotDeath, onBulletFired, onBulletHit, onBulletHitBullet, onBulletHitWall, onConnected, onConnectionError, onCustomEvent, onDeath, onDisconnected, onGameEnded, onGameStarted, onHitBot, onHitByBullet, onHitWall, onRoundEnded, onRoundStarted, onScannedBot, onSkippedTurn, onTeamMessage, onTick, onWonRound, radarBearingTo`

Constructor Detail

BaseBot

`protected BaseBot()`

Constructor for initializing a new instance of the BaseBot class. This constructor should be used when both `BotInfo` and server URL is provided through environment variables, i.e., when starting up the bot using a booter. These environment variables must be set to provide the server URL and bot information and are automatically set by the booter tool for Robocode.

Example of how to set the predefined environment variables used for connecting to the server:

- `SERVER_URL=ws://localhost:7654`
- `SERVER_SECRET=xzoEeVbnBe5TGjCny0R1yQ`

Example of how to set the environment variables that covers the `BotInfo`:

- `BOT_NAME=MyBot`
- `BOT_VERSION=1.0`
- `BOT_AUTHORS=John Doe`
- `BOT_DESCRIPTION=Short description`
- `BOT_HOMEPAGE=https://somewhere.net/MyBot`
- `BOT_COUNTRY_CODES=us`
- `BOT_GAME_TYPES=classic, melee, 1v1`
- `BOT_PLATFORM=JVM`
- `BOT_PROG_LANG=Java 11`
- `BOT_INITIAL_POS=50,70, 270`

These values can take multiple values separated by a comma:

- `BOT_AUTHORS, e.g. "John Doe, Jane Doe"`
- `BOT_COUNTRY_CODES, e.g. "se, no, dk"`
- `BOT_GAME_TYPES, e.g. "classic, melee, 1v1"`

The `BOT_INITIAL_POS` variable is optional and should *only* be used for debugging.

The `SERVER_SECRET` must be set if the server requires a server secret for the bots trying to connect. Otherwise, the bot will be disconnected as soon as it attempts to connect to the server.

If the `SERVER_URL` is not set, then this default URL is used: ws://localhost:7654

BaseBot

```
protected BaseBot(BotInfo botInfo)
```

Constructor for initializing a new instance of the BaseBot class. This constructor assumes the server URL and secret are provided by the environment variables SERVER_URL and SERVER_SECRET.

Parameters:

`botInfo` - is the bot info containing information about your bot.

BaseBot

```
protected BaseBot(BotInfo botInfo,
                  java.net.URI serverUrl)
```

Constructor for initializing a new instance of the BaseBot class.

Parameters:

`botInfo` - is the bot info containing information about your bot.

`serverUrl` - is the server URL

BaseBot

```
protected BaseBot(BotInfo botInfo,
                  java.net.URI serverUrl,
                  java.lang.String serverSecret)
```

Constructor for initializing a new instance of the BaseBot class.

Parameters:

`botInfo` - is the bot info containing information about your bot.

`serverUrl` - is the server URL

`serverSecret` - is the server secret for bots

Method Detail

start

```
public final void start()
```

The method used to start running the bot. You should call this method from the main method or similar.

Example:

```
 public void main(String[] args) {
     // create myBot
```

Specified by:

`start` in interface `IBaseBot`

go

```
public void go()
```

Commits the current commands (actions), which finalizes the current turn for the bot.

This method must be called once per turn to send the bot actions to the server and must be called before the turn timeout occurs. A turn timer is started when the `GameStartedEvent` and `TickEvent` occurs. If the `go()` method is called too late, a turn timeout will occur and the `SkippedTurnEvent` will occur, which means that the bot has skipped all actions for the last turn. In this case, the server will continue executing the last actions received. This could be fatal for the bot due to loss of control over the bot. So make sure that `go()` is called before the turn ends.

The commands executed when `go()` is called are set by calling the various setter methods prior to calling the `go()` method: `IBaseBot.setTurnRate(double)`, `IBaseBot.setGunTurnRate(double)`, `IBaseBot.setRadarTurnRate(double)`, `IBaseBot.setTargetSpeed(double)`, and `IBaseBot.setFire(double)`.

Specified by:

`go` in interface `IBaseBot`

See Also:

`IBaseBot.getTurnTimeout()`

getVariant

```
public final java.lang.String getVariant()
```

The game variant, which is "Tank Royale".

Specified by:

`getVariant` in interface `IBaseBot`

Returns:

The game variant of Robocode.

getVersion

```
public final java.lang.String getVersion()
```

Game version, e.g. "1.0.0".

Specified by:

`getVersion` in interface `IBaseBot`

Returns:

The game version.

getMyId

```
public final int getMyId()
```

Unique id of this bot, which is available when the game has started.

Specified by:

`getMyId` in interface `IBaseBot`

Returns:

The unique id of this bot.

getGameType

Specified by:

`getGameType` in interface `IBaseBot`

Returns:

The game type.

getArenaWidth

```
public final int getArenaWidth()
```

Width of the arena measured in units.

First available when the game has started.

Specified by:

`getArenaWidth` in interface `IBaseBot`

Returns:

The arena width measured in units

getArenaHeight

```
public final int getArenaHeight()
```

Height of the arena measured in units.

First available when the game has started.

Specified by:

`getArenaHeight` in interface `IBaseBot`

Returns:

The arena height measured in units

getNumberOfRounds

```
public final int getNumberOfRounds()
```

The number of rounds in a battle.

First available when the game has started.

Specified by:

`getNumberOfRounds` in interface `IBaseBot`

Returns:

The number of rounds in a battle.

getGunCoolingRate

```
public final double getGunCoolingRate()
```

Gun cooling rate. The gun needs to cool down to a gun heat of zero before the gun can fire. The gun cooling rate determines how fast the gun cools down. That is, the gun cooling rate is subtracted from the gun heat each turn until the gun heat reaches zero.

First available when the game has started.

Specified by:

`getGunCoolingRate` in interface `IBaseBot`

Returns:

The gun cooling rate.

See Also:

`public final int getMaxInactivityTurns()`

The maximum number of inactive turns allowed the bot will become zapped by the game for being inactive. Inactive means that the bot has taken no action in several turns in a row.

First available when the game has started.

Specified by:

`getMaxInactivityTurns` in interface `IBaseBot`

Returns:

The maximum number of allowed inactive turns.

getTurnTimeout

`public final int getTurnTimeout()`

The turn timeout is important as the bot needs to take action by calling `IBaseBot.go()` before the turn timeout occurs. As soon as the `TickEvent` is triggered, i.e. when `IBaseBot.onTick(dev.robocode.tankroyale.botapi.events.TickEvent)` is called, you need to call `IBaseBot.go()` to take action before the turn timeout occurs. Otherwise, your bot will skip a turn and receive a `IBaseBot.onSkippedTurn(dev.robocode.tankroyale.botapi.events.SkippedTurnEvent)` for each turn where `IBaseBot.go()` is called too late.

First available when the game has started.

Specified by:

`getTurnTimeout` in interface `IBaseBot`

Returns:

The turn timeout in microseconds (1 / 1,000,000 second).

See Also:

`IBaseBot.getTimeLeft()`, `IBaseBot.go()`

getTimeLeft

`public final int getTimeLeft()`

The number of microseconds left of this turn before the bot will skip the turn. Make sure to call `IBaseBot.go()` before the time runs out.

Specified by:

`getTimeLeft` in interface `IBaseBot`

Returns:

The amount of time left in microseconds.

See Also:

`IBaseBot.getTurnTimeout()`, `IBaseBot.go()`

getRoundNumber

`public final int getRoundNumber()`

Current round number.

Specified by:

`getRoundNumber` in interface `IBaseBot`

Returns:

The current round number.

getTurnNumber

`public final int getTurnNumber()`

The current turn number.

getEnemyCount

```
public final int getEnemyCount()
```

Number of enemies left in the round.

Specified by:

`getEnemyCount` in interface `IBaseBot`

Returns:

The number of enemies left in the round.

getEnergy

```
public final double getEnergy()
```

Current energy level. When the energy level is positive, the bot is alive and active. When the energy level is 0, the bot is still alive but disabled. If the bot becomes disabled it will not be able to move or take any action. If negative, the bot has been defeated.

Specified by:

`getEnergy` in interface `IBaseBot`

Returns:

The current energy level.

isDisabled

```
public final boolean isDisabled()
```

Specifies if the bot is disabled, i.e., when the energy is zero. When the bot is disabled, it is not able to take any action like movement, turning, and firing.

Specified by:

`isDisabled` in interface `IBaseBot`

Returns:

`true` if the bot is disabled; `false` otherwise.

getX

```
public final double getX()
```

Current X coordinate of the center of the bot.

Specified by:

`getX` in interface `IBaseBot`

Returns:

The current X coordinate of the bot.

getY

```
public final double getY()
```

Current Y coordinate of the center of the bot.

Specified by:

`getY` in interface `IBaseBot`

Returns:

```
public final double getDirection()
```

Current driving direction of the bot in degrees.

Specified by:

`getDirection` in interface `IBaseBot`

Returns:

The current driving direction of the bot.

getGunDirection

```
public final double getGunDirection()
```

Current direction of the gun in degrees.

Specified by:

`getGunDirection` in interface `IBaseBot`

Returns:

The current gun direction of the bot.

getRadarDirection

```
public final double getRadarDirection()
```

Current direction of the radar in degrees.

Specified by:

`getRadarDirection` in interface `IBaseBot`

Returns:

The current radar direction of the bot.

getSpeed

```
public final double getSpeed()
```

The current speed measured in units per turn. If the speed is positive, the bot moves forward. If negative, the bot moves backward. Zero speed means that the bot is not moving from its current position.

Specified by:

`getSpeed` in interface `IBaseBot`

Returns:

The current speed.

getGunHeat

```
public final double getGunHeat()
```

Current gun heat. When the is fired it gets heated and will not be able to fire before it has been cooled down. The gun is cooled down when the gun heat is zero.

When the gun has fired the gun heat is set to 1 + (firepower / 5) and will be cooled down by the gun cooling rate.

Specified by:

`getGunHeat` in interface `IBaseBot`

Returns:

The current gun heat.

See Also:

`IBaseBot.getGunCoolingRate()`

Current bullet states. Keeps track of all the bullets fired by the bot, which are still active on the arena.

Specified by:

`getBulletStates` in interface `IBaseBot`

Returns:

The current bullet states.

getEvents

```
public final java.util.List<BotEvent> getEvents()
```

Returns an ordered list containing all events currently in the bot's event queue. You might, for example, call this while processing another event.

Specified by:

`getEvents` in interface `IBaseBot`

Returns:

an ordered list containing all events currently in the bot's event queue.

See Also:

`IBaseBot.clearEvents()`

clearEvents

```
public final void clearEvents()
```

Clears out any pending events in the bot's event queue immediately.

Specified by:

`clearEvents` in interface `IBaseBot`

See Also:

`IBaseBot.getEvents()`

getTurnRate

```
public final double getTurnRate()
```

Returns the turn rate of the bot in degrees per turn.

Specified by:

`getTurnRate` in interface `IBaseBot`

Returns:

The turn rate of the bot.

See Also:

`IBaseBot.setTurnRate(double)`

setTurnRate

```
public void setTurnRate(double turnRate)
```

Sets the turn rate of the bot, which can be positive and negative. The turn rate is measured in degrees per turn. The turn rate is added to the current direction of the bot. But it is also added to the current direction of the gun and radar. This is because the gun is mounted on the body, and hence turns with the body. The radar is mounted on the gun and hence moves with the gun. You can compensate for the turn rate of the bot by subtracting the turn rate of the bot from the turn rate of the gun and radar. But be aware that the turn limits defined for the gun and radar cannot be exceeded.

The turn rate is truncated to `Constants.MAX_TURN_RATE` if the turn rate exceeds this value.

If this property is set multiple times, the last value set before `IBaseBot.go()` counts.

getMaxTurnRate

```
public final double getMaxTurnRate()
```

Returns the maximum turn rate of the bot in degrees per turn.

Specified by:

`getMaxTurnRate` in interface `IBaseBot`

Returns:

The maximum turn rate of the bot.

See Also:

`IBaseBot.setMaxTurnRate(double)`

setMaxTurnRate

```
public final void setMaxTurnRate(double maxTurnRate)
```

Sets the maximum turn rate which applies to turn the bot to the left or right. The maximum turn rate must be an absolute value from 0 to `Constants.MAX_TURN_RATE`, both values are included. If the input turn rate is negative, the max turn rate will be cut to zero. If the input turn rate is above `Constants.MAX_TURN_RATE`, the max turn rate will be set to `Constants.MAX_TURN_RATE`.

If for example the max turn rate is set to 5, then the bot will be able to turn left or right with a turn rate down to -5 degrees per turn when turning right, and up to 5 degrees per turn when turning left.

This method will first be executed when `IBaseBot.go()` is called making it possible to call other set methods after execution. This makes it possible to set the bot to move, turn the body, radar, gun, and also fire the gun in parallel in a single turn when calling `IBaseBot.go()`. But notice that this is only possible to execute multiple methods in parallel by using **setter** methods only prior to calling `IBaseBot.go()`.

If this method is called multiple times, the last call before `IBaseBot.go()` is executed, counts.

Specified by:

`setMaxTurnRate` in interface `IBaseBot`

Parameters:

`maxTurnRate` - is the new maximum turn rate

See Also:

`IBaseBot.setTurnRate(double)`

getGunTurnRate

```
public final double getGunTurnRate()
```

Returns the gun turn rate in degrees per turn.

Specified by:

`getGunTurnRate` in interface `IBaseBot`

Returns:

The turn rate of the gun.

See Also:

`IBaseBot.setGunTurnRate(double)`

setGunTurnRate

```
public void setGunTurnRate(double gunTurnRate)
```

Sets the turn rate of the gun, which can be positive and negative. The gun turn rate is measured in degrees per turn. The turn rate is added to the current turn direction of the gun. But it is also added to the current direction of the radar. This is because the radar is mounted on the gun, and hence moves with the gun. You can compensate for the turn rate of the gun by subtracting the turn rate of the gun from the turn rate of the radar. But be aware that the turn limits defined for the radar cannot be exceeded.

setGunTurnRate in interface IBaseBot

Parameters:

`gunTurnRate` - is the new turn rate of the gun in degrees per turn.

getMaxGunTurnRate

```
public final double getMaxGunTurnRate()
```

Returns the maximum gun turn rate in degrees per turn.

Specified by:

`getMaxGunTurnRate` in interface `IBaseBot`

Returns:

The maximum turn rate of the gun.

See Also:

`IBaseBot.setMaxGunTurnRate(double)`

setMaxGunTurnRate

```
public final void setMaxGunTurnRate(double maxGunTurnRate)
```

Sets the maximum turn rate which applies to turn the gun to the left or right. The maximum turn rate must be an absolute value from 0 to `Constants.MAX_GUN_TURN_RATE`, both values are included. If the input turn rate is negative, the max turn rate will be cut to zero. If the input turn rate is above `Constants.MAX_GUN_TURN_RATE`, the max turn rate will be set to `Constants.MAX_GUN_TURN_RATE`.

If for example the max gun turn rate is set to 5, then the gun will be able to turn left or right with a turn rate down to -5 degrees per turn when turning right and up to 5 degrees per turn when turning left.

This method will first be executed when `IBaseBot.go()` is called making it possible to call other set methods after execution. This makes it possible to set the bot to move, turn the body, radar, gun, and also fire the gun in parallel in a single turn when calling `IBaseBot.go()`. But notice that this is only possible to execute multiple methods in parallel by using **setter** methods only prior to calling `IBaseBot.go()`.

If this method is called multiple times, the last call before `IBaseBot.go()` is executed, counts.

Specified by:

`setMaxGunTurnRate` in interface `IBaseBot`

Parameters:

`maxGunTurnRate` - is the new maximum gun turn rate

See Also:

`IBaseBot.setGunTurnRate(double)`

getRadarTurnRate

```
public final double getRadarTurnRate()
```

Returns the radar turn rate in degrees per turn.

Specified by:

`getRadarTurnRate` in interface `IBaseBot`

Returns:

The turn rate of the radar.

See Also:

`IBaseBot.setRadarTurnRate(double)`

setRadarTurnRate

```
public void setRadarTurnRate(double radarTurnRate)
```

The radar turn rate is truncated to `Constants.MAX_RADAR_TURN_RATE` if the radar turn rate exceeds this value.

If this property is set multiple times, the last value set before `IBaseBot.go()` counts.

Specified by:

`setRadarTurnRate` in interface `IBaseBot`

Parameters:

`radarTurnRate` - is the new turn rate of the radar in degrees per turn.

getMaxRadarTurnRate

```
public final double getMaxRadarTurnRate()
```

Returns the maximum radar turn rate in degrees per turn.

Specified by:

`getMaxRadarTurnRate` in interface `IBaseBot`

Returns:

The maximum turn rate of the radar.

See Also:

`IBaseBot.setMaxRadarTurnRate(double)`

setMaxRadarTurnRate

```
public final void setMaxRadarTurnRate(double maxRadarTurnRate)
```

Sets the maximum turn rate which applies to turn the radar to the left or right. The maximum turn rate must be an absolute value from 0 to `Constants.MAX_RADAR_TURN_RATE`, both values are included. If the input turn rate is negative, the max turn rate will be cut to zero. If the input turn rate is above `Constants.MAX_RADAR_TURN_RATE`, the max turn rate will be set to `Constants.MAX_RADAR_TURN_RATE`.

If for example the max radar turn rate is set to 5, then the radar will be able to turn left or right with a turn rate down to -5 degrees per turn when turning right and up to 5 degrees per turn when turning left.

This method will first be executed when `IBaseBot.go()` is called making it possible to call other set methods after execution. This makes it possible to set the bot to move, turn the body, radar, gun, and also fire the gun in parallel in a single turn when calling `IBaseBot.go()`. But notice that this is only possible to execute multiple methods in parallel by using **setter** methods only prior to calling `IBaseBot.go()`.

If this method is called multiple times, the last call before `IBaseBot.go()` is executed, counts.

Specified by:

`setMaxRadarTurnRate` in interface `IBaseBot`

Parameters:

`maxRadarTurnRate` - is the new maximum radar turn rate

See Also:

`IBaseBot.setRadarTurnRate(double)`

getTargetSpeed

```
public final double getTargetSpeed()
```

Returns the target speed in units per turn.

Specified by:

`getTargetSpeed` in interface `IBaseBot`

Returns:

The target speed.

See Also:

`IBaseBot.setTargetSpeed(double)`

Sets the new target speed for the bot in units per turn. The target speed is the speed you want to achieve eventually, which could take one to several turns depending on the current speed. For example, if the bot is moving forward with max speed, and then must change to move backward at full speed, the bot will have to first decelerate/brake its positive speed (moving forward). When passing speed of zero, it will then have to accelerate back to achieve max negative speed.

Note that acceleration is 1 unit per turn and deceleration/braking is faster than acceleration as it is -2 unit per turn. Deceleration is negative as it is added to the speed and hence needs to be negative when slowing down.

The target speed is truncated to `Constants.MAX_SPEED` if the target speed exceeds this value.

If this property is set multiple times, the last value set before `IBaseBot.go()` counts.

Specified by:

`setTargetSpeed` in interface `IBaseBot`

Parameters:

`targetSpeed` - is the new target speed in units per turn.

getMaxSpeed

```
public final double getMaxSpeed()
```

Returns the maximum speed in units per turn.

Specified by:

`getMaxSpeed` in interface `IBaseBot`

Returns:

The maximum speed.

See Also:

`IBaseBot.setMaxSpeed(double)`

setMaxSpeed

```
public final void setMaxSpeed(double maxSpeed)
```

Sets the maximum speed which applies when moving forward and backward. The maximum speed must be an absolute value from 0 to `Constants.MAX_SPEED`, both values are included. If the input speed is negative, the max speed will be cut to zero. If the input speed is above `Constants.MAX_SPEED`, the max speed will be set to `Constants.MAX_SPEED`.

If for example the maximum speed is set to 5, then the bot will be able to move backwards with a speed down to -5 units per turn and up to 5 units per turn when moving forward.

This method will first be executed when `IBaseBot.go()` is called making it possible to call other set methods after execution. This makes it possible to set the bot to move, turn the body, radar, gun, and also fire the gun in parallel in a single turn when calling `IBaseBot.go()`. But notice that this is only possible to execute multiple methods in parallel by using **setter** methods only prior to calling `IBaseBot.go()`.

If this method is called multiple times, the last call before `IBaseBot.go()` is executed, counts.

Specified by:

`setMaxSpeed` in interface `IBaseBot`

Parameters:

`maxSpeed` - is the new maximum speed

setFire

```
public final boolean setFire(double firepower)
```

Sets the gun to fire in the direction that the gun is pointing with the specified firepower.

Firepower is the amount of energy your bot will spend on firing the gun. This means that the bot will lose power on firing the gun where the energy loss is equal to the firepower. You cannot spend more energy than available from your bot.

The bullet power must be greater than `Constants.MIN_FIREPOWER` and the gun heat zero before the gun can fire.

If the bullet hits an opponent bot, you will gain energy from the bullet hit. When hitting another bot, your bot will be rewarded and retrieve an energy boost of 3x firepower.

longer it takes to cool down the gun. The gun cooling rate can be read by calling `IBaseBot.getGunCoolingRate()`.

The amount of energy used for firing the gun is subtracted from the bots' total energy. The amount of damage dealt by a bullet hitting another bot is 4x firepower, and if the firepower is greater than 1 it will do an additional 2 x (firepower - 1) damage.

Note that the gun will automatically keep firing at any turn as soon as the gun heat reaches zero. It is possible to disable the gun firing by setting the firepower to zero.

The firepower is truncated to 0 and `Constants.MAX_FIREPOWER` if the firepower exceeds this value.

If this property is set multiple times, the last value set before go() counts.

Specified by:

`setFire` in interface `IBaseBot`

Parameters:

`firepower` - is the new firepower

Returns:

`true` if the cannon can fire, i.e. if there is no gun heat; `false` otherwise.

See Also:

`IBaseBot.onBulletFired(dev.robocode.tankroyale.botapi.events.BulletFiredEvent)`, `IBaseBot.getFirepower()`, `IBaseBot.getGunHeat()`, `IBaseBot.getGunCoolingRate()`

getFirepower

```
public final double getFirepower()
```

Returns the firepower.

Specified by:

`getFirepower` in interface `IBaseBot`

Returns:

The firepower.

See Also:

`IBaseBot.setFire(double)`

setRescan

```
public final void setRescan()
```

Sets the bot to rescan with the radar. This method is useful if the radar has not turned, and hence will not automatically scan bots. The last radar direction and sweep angle will be used for scanning for bots.

Specified by:

`setRescan` in interface `IBaseBot`

setFireAssist

```
public final void setFireAssist(boolean enable)
```

Enables or disables fire assistance explicitly. Fire assistance is useful for bots with limited aiming capabilities as it will help the bot by firing directly at a scanned bot when the gun is fired, which is a very simple aiming strategy.

When fire assistance is enabled the gun will fire towards the center of the scanned bot when all these conditions are met:

- The gun is fired (`IBaseBot.setFire(double)` and `IBot.fire(double)`)
- The radar is scanning a bot *when* firing the gun (`IBaseBot.onScannedBot(dev.robocode.tankroyale.botapi.events.ScannedBotEvent)`, `IBaseBot.setRescan()`, `IBot.rescan()`)
- The gun and radar are pointing in the exact the same direction. You can call `setAdjustRadarForGunTurn(false)` to align the gun and radar and make sure not to turn the radar beside the gun.

The fire assistance feature is provided for backwards compatibility with the original Robocode, where robots that are not an `AdvancedRobot` got fire assistance per default as the gun and radar cannot be moved independently of each other. (The `AdvancedRobot` allows the body, gun, and radar to move independent of each other).

setInterruptible

`public final void setInterruptible(boolean interruptible)`

Call this method during an event handler to control continuing or restarting the event handler, when a new event occurs again for the same event handler while processing an earlier event.

Example:

```
    public void onScannedBot(ScannedBotEvent e) {
        fire(1);
        setInterruptible(true);
        forward(100); // When a new bot is scanned while moving forward this handler will restart
                      // from the top as this event handler has been set to be interruptible
                      // right after firing. Without setInterruptible(true), new scan events
                      // would not be triggered while moving forward.
        // We'll only get here if we do not see a robot during the move.
        System.out.println("No bots were scanned");
    }
```

Specified by:

`setInterruptible` in interface `IBaseBot`

Parameters:

`interruptible` - `true` if the event handler should be interrupted and hence restart when a new event of the same event type occurs again; `false` otherwise where the event handler will continue processing.

setAdjustGunForBodyTurn

`public final void setAdjustGunForBodyTurn(boolean adjust)`

Sets the gun to adjust for the botÂ´s turn when setting the gun turn rate. So the gun behaves like it is turning independent of the botÂ´s turn.

Ok, so this needs some explanation: The gun is mounted on the botÂ´s body. So, normally, if the bot turns 90 degrees to the right, then the gun will turn with it as it is mounted on top of the botÂ´s body. To compensate for this, you can adjust the gun for the botÂ´s turn. When this is set, the gun will turn independent of the botÂ´s turn.

Note: This property is additive until you reach the maximum the gun can turn `Constants.MAX_GUN_TURN_RATE`. The "adjust" is added to the amount, you set for turning the bot by the turn rate, then capped by the physics of the game.

Note: The gun compensating this way does count as "turning the gun".

Specified by:

`setAdjustGunForBodyTurn` in interface `IBaseBot`

Parameters:

`adjust` - `true` if the gun must adjust/compensate for the body turning; `false` if the gun must turn with the body turning (default).

See Also:

`IBaseBot.setAdjustRadarForBodyTurn(boolean)`, `IBaseBot.setAdjustRadarForGunTurn(boolean)`, `IBaseBot.isAdjustGunForBodyTurn()`, `IBaseBot.isAdjustRadarForBodyTurn()`, `IBaseBot.isAdjustRadarForGunTurn()`

isAdjustGunForBodyTurn

`public final boolean isAdjustGunForBodyTurn()`

Checks if the gun is set to adjust for the bot turning, i.e. to turn independent of the botÂ´s body turn.

This call returns `true` if the gun is set to turn independent of the turn of the botÂ´s body. Otherwise, `false` is returned, meaning that the gun is set to turn with the botÂ´s body turn.

Specified by:

`isAdjustGunForBodyTurn` in interface `IBaseBot`

`IBaseBot.setAdjustRadarForGunTurn(boolean)`, `IBaseBot.isAdjustRadarForBodyTurn()`, `IBaseBot.isAdjustRadarForGunTurn()`

setAdjustRadarForBodyTurn

`public final void setAdjustRadarForBodyTurn(boolean adjust)`

Sets the radar to adjust for the body's turn when setting the radar turn rate. So the radar behaves like it is turning independent of the body's turn.

Ok, so this needs some explanation: The radar is mounted on the gun, and the gun is mounted on the botÂ´s body. So, normally, if the bot turns 90 degrees to the right, the gun turns, as does the radar. Hence, if the bot turns 90 degrees to the right, then the gun and radar will turn with it as the radar is mounted on top of the gun. To compensate for this, you can adjust the radar for the body turn. When this is set, the radar will turn independent of the body's turn.

Note: This property is additive until you reach the maximum the radar can turn (`Constants.MAX_RADAR_TURN_RATE`). The "adjust" is added to the amount, you set for turning the body by the body turn rate, then capped by the physics of the game.

Note: The radar compensating this way does count as "turning the radar".

Specified by:

`setAdjustRadarForBodyTurn` in interface `IBaseBot`

Parameters:

`adjust` - `true` if the radar must adjust/compensate for the body's turn; `false` if the radar must turn with the body turning (default).

See Also:

`IBaseBot.setAdjustGunForBodyTurn(boolean)`, `IBaseBot.setAdjustRadarForGunTurn(boolean)`, `IBaseBot.isAdjustGunForBodyTurn()`, `IBaseBot.isAdjustRadarForBodyTurn()`, `IBaseBot.isAdjustRadarForGunTurn()`

isAdjustRadarForBodyTurn

`public final boolean isAdjustRadarForBodyTurn()`

Checks if the radar is set to adjust for the body turning, i.e. to turn independent of the body's turn.

This call returns `true` if the radar is set to turn independent of the turn of the body. Otherwise, `false` is returned, meaning that the radar is set to turn with the body turning.

Specified by:

`isAdjustRadarForBodyTurn` in interface `IBaseBot`

Returns:

`true` if the radar is set to turn independent of the body turning; `false` if the radar is set to turn with the body turning (default).

See Also:

`IBaseBot.setAdjustGunForBodyTurn(boolean)`, `IBaseBot.setAdjustRadarForBodyTurn(boolean)`, `IBaseBot.setAdjustRadarForGunTurn(boolean)`, `IBaseBot.isAdjustGunForBodyTurn()`, `IBaseBot.isAdjustRadarForGunTurn()`

setAdjustRadarForGunTurn

`public final void setAdjustRadarForGunTurn(boolean adjust)`

Sets the radar to adjust for the gun's turn when setting the radar turn rate. So the radar behaves like it is turning independent of the gun's turn.

Ok, so this needs some explanation: The radar is mounted on the gun. So, normally, if the gun turns 90 degrees to the right, then the radar will turn with it as it is mounted on top of the gun. To compensate for this, you can adjust the radar for the gun turn. When this is set, the radar will turn independent of the gun's turn.

Note: This property is additive until you reach the maximum the radar can turn (`Constants.MAX_RADAR_TURN_RATE`). The "adjust" is added to the amount, you set for turning the gun by the gun turn rate, then capped by the physics of the game.

When the radar compensates this way it counts as "turning the radar", even when it is not explicitly turned by calling a method for turning the radar.

Note: This method automatically disables fire assistance when set to `true`, and automatically enables fire assistance when set to `false`. This is *not* the case for `IBaseBot.setAdjustGunForBodyTurn(boolean)` and `IBaseBot.setAdjustRadarForBodyTurn(boolean)`. Read more about fire assistance with the `IBaseBot.setFireAssist(boolean)` method.

See Also:

`IBaseBot.setAdjustGunForBodyTurn(boolean)`, `IBaseBot.setAdjustRadarForBodyTurn(boolean)`, `IBaseBot.isAdjustGunForBodyTurn()`, `IBaseBot.isAdjustRadarForBodyTurn()`, `IBaseBot.isAdjustRadarForGunTurn()`

isAdjustRadarForGunTurn

`public final boolean isAdjustRadarForGunTurn()`

Checks if the radar is set to adjust for the gun turning, i.e. to turn independent of the gun's turn.

This call returns `true` if the radar is set to turn independent of the turn of the gun. Otherwise, `false` is returned, meaning that the radar is set to turn with the gun's turn.

Specified by:

`isAdjustRadarForGunTurn` in interface `IBaseBot`

Returns:

`true` if the radar is set to turn independent of the gun turning; `false` if the radar is set to turn with the gun turning (default).

See Also:

`IBaseBot.setAdjustGunForBodyTurn(boolean)`, `IBaseBot.setAdjustRadarForBodyTurn(boolean)`, `IBaseBot.setAdjustRadarForGunTurn(boolean)`, `IBaseBot.isAdjustGunForBodyTurn()`, `IBaseBot.isAdjustRadarForBodyTurn()`

addCustomEvent

`public final boolean addCustomEvent(Condition condition)`

Adds an event handler that will be automatically triggered `IBaseBot.onCustomEvent(dev.robocode.tankroyale.botapi.events.CustomEvent)` when the `Condition.test()` returns `true`.

Specified by:

`addCustomEvent` in interface `IBaseBot`

Parameters:

`condition` - is the condition that must be met to trigger the custom event.

Returns:

`true` if the condition was not added already; `false` if the condition was already added.

See Also:

`IBaseBot.removeCustomEvent(dev.robocode.tankroyale.botapi.events.Condition)`

removeCustomEvent

`public final boolean removeCustomEvent(Condition condition)`

Removes triggering a custom event handler for a specific condition that was previously added with `IBaseBot.addCustomEvent(dev.robocode.tankroyale.botapi.events.Condition)`.

Specified by:

`removeCustomEvent` in interface `IBaseBot`

Parameters:

`condition` - is the condition that was previously added with `IBaseBot.addCustomEvent(dev.robocode.tankroyale.botapi.events.Condition)`

Returns:

`true` if the condition was found; `false` if the condition was not found.

See Also:

`IBaseBot.addCustomEvent(dev.robocode.tankroyale.botapi.events.Condition)`

setStop

`public final void setStop()`

notice that this is only possible to execute multiple methods in parallel by using **setter** methods only prior to calling `IBaseBot.go()`.

Specified by:

`setStop` in interface `IBaseBot`

See Also:

`IBaseBot.setResume()`

setStop

```
public final void setStop(boolean overwrite)
```

Sets the bot to stop all movement including turning the gun and radar. The remaining movement is saved for a call to `IBaseBot.setResume()`.

This method will first be executed when `IBaseBot.go()` is called making it possible to call other set methods before execution. This makes it possible to set the bot to move, turn the body, radar, gun, and also fire the gun in parallel in a single turn when calling `IBaseBot.go()`. But notice that this is only possible to execute multiple methods in parallel by using **setter** methods only prior to calling `IBaseBot.go()`.

Specified by:

`setStop` in interface `IBaseBot`

Parameters:

`overwrite` - is set to `true` if the movement saved by a previous call to this method or `IBaseBot.setStop()` must be overridden with the current movement. When set to `false` this method is identical to `IBaseBot.setStop()`.

See Also:

`IBaseBot.setResume()`

setResume

```
public final void setResume()
```

Sets the bot to resume movement after having been stopped, e.g. when `IBaseBot.setStop()` has been called. The last radar direction and sweep angle will be used for rescanning for bots.

This method will first be executed when `IBaseBot.go()` is called making it possible to call other set methods before execution. This makes it possible to set the bot to move, turn the body, radar, gun, and also fire the gun in parallel in a single turn when calling `IBaseBot.go()`. But notice that this is only possible to execute multiple methods in parallel by using **setter** methods only prior to calling `IBaseBot.go()`.

Specified by:

`setResume` in interface `IBaseBot`

See Also:

`IBaseBot.setStop()`, `IBaseBot.setStop(boolean)`

isStopped

```
public final boolean isStopped()
```

Checks if the movement has been stopped.

Specified by:

`isStopped` in interface `IBaseBot`

Returns:

true if the movement has been stopped by `IBaseBot.setStop()`; false otherwise.

See Also:

`IBaseBot.setResume()`, `IBaseBot.setStop()`, `IBaseBot.setStop(boolean)`

getTeammateIds

```
public final java.util.Set<java.lang.Integer> getTeammateIds()
```

Returns the ids of all teammates.

See Also:

`IBaseBot.isTeammate(int)`, `IBaseBot.sendTeamMessage(int, java.lang.Object)`

isTeammate

```
public final boolean isTeammate(int botId)
```

Checks if the provided bot id is a teammate or not.

Example:

```
    public void onScannedBot(ScannedBotEvent event) {
        if (isTeammate(event.getScannedBotId()) {
            return; // don't do anything by leaving
        }
        fire(1);
    }
```

Specified by:

`isTeammate` in interface `IBaseBot`

Parameters:

`botId` - is the id of the bot to check for.

Returns:

`true` if the provided is id an id of a teammate; `false` otherwise.

See Also:

`IBaseBot.getTeammateIds()`, `IBaseBot.sendTeamMessage(int, java.lang.Object)`

broadcastTeamMessage

```
public final void broadcastTeamMessage(java.lang.Object message)
```

Broadcasts a message to all teammates.

When the message is send, it is serialized into a JSON representation, meaning that all public fields, and only public fields, are being serialized into a JSON representation as a DTO (data transfer object).

The maximum team message size limit is defined by `IBaseBot.TEAM_MESSAGE_MAX_SIZE`, which is set to 32768 bytes. This size is the size of the message when it is serialized into a JSON representation.

The maximum number of messages that can be send/broadcast per turn is limited to 10.

Specified by:

`broadcastTeamMessage` in interface `IBaseBot`

Parameters:

`message` - is the message to broadcast.

See Also:

`IBaseBot.sendTeamMessage(int, java.lang.Object)`, `IBaseBot.getTeammateIds()`

sendTeamMessage

```
public final void sendTeamMessage(int teammateId, java.lang.Object message)
```

Sends a message to a specific teammate.

When the message is sent, it is serialized into a JSON representation, meaning that all public fields, and only public fields, are being serialized into a JSON representation as a DTO (data transfer object).

The maximum team message size limit is defined by `IBaseBot.TEAM_MESSAGE_MAX_SIZE`, which is set to 32768 bytes. This size is the size of the message when it is serialized into a JSON representation.

Parameters:

`teammateId` - is the id of the teammate to send the message to.

`message` - is the message to send.

See Also:

`IBaseBot.broadcastTeamMessage(java.lang.Object)`, `IBaseBot.getTeammateIds()`

getBodyColor

`public final java.awt.Color getBodyColor()`

Returns the color of the body.

Specified by:

`getBodyColor` in interface `IBaseBot`

Returns:

The color of the body or `null` if no color has been set yet, meaning that the default color will be used.

setBodyColor

`public final void setBodyColor(java.awt.Color color)`

Sets the color of the body. Colors can (only) be changed each turn.

Example:

```
    setBodyColor(Color.RED); // the red color
    setBodyColor(new Color(255, 0, 0)); // also the red color
```

Specified by:

`setBodyColor` in interface `IBaseBot`

Parameters:

`color` - is the color of the body or `null` if the bot must use the default color instead.

getTurretColor

`public final java.awt.Color getTurretColor()`

Returns the color of the gun turret.

Specified by:

`getTurretColor` in interface `IBaseBot`

Returns:

The color of the turret or `null` if no color has been set yet, meaning that the default color will be used.

setTurretColor

`public final void setTurretColor(java.awt.Color color)`

Sets the color of the gun turret. Colors can (only) be changed each turn.

Example:

```
    setTurretColor(Color.RED); // the red color
    setTurretColor(new Color(255, 0, 0)); // also the red color
```

Specified by:

getRadarColor

```
public final java.awt.Color getRadarColor()
```

Returns the color of the radar.

Specified by:

`getRadarColor` in interface `IBaseBot`

Returns:

The color of the radar or `null` if no color has been set yet, meaning that the default color will be used.

setRadarColor

```
public final void setRadarColor(java.awt.Color color)
```

Sets the color of the radar. Colors can (only) be changed each turn.

Example:

```
    setRadarColor(Color.RED); // the red color
    setRadarColor(new Color(255, 0, 0)); // also the red color
```

Specified by:

`setRadarColor` in interface `IBaseBot`

Parameters:

`color` - is the color of the radar or `null` if the bot must use the default color instead.

getBulletColor

```
public final java.awt.Color getBulletColor()
```

Returns the color of the fired bullets.

Specified by:

`getBulletColor` in interface `IBaseBot`

Returns:

The color of the bullets or `null` if no color has been set yet, meaning that the default color will be used.

setBulletColor

```
public final void setBulletColor(java.awt.Color color)
```

Sets the color of the fired bullets. Colors can (only) be changed each turn.

Note that a fired bullet will not change is color when it has been fired. But new bullets fired after setting the bullet color will get the new color.

Example:

```
    setBulletColor(Color.RED); // the red color
    setBulletColor(new Color(255, 0, 0)); // also the red color
```

Specified by:

`setBulletColor` in interface `IBaseBot`

Parameters:

`color` - is the color of the fired bullets or `null` if the bot must use the default color instead.

Returns the color of the scan arc.

Specified by:

`getScanColor` in interface `IBaseBot`

Returns:

The color of the scan arc or `null` if no color has been set yet, meaning that the default color will be used.

setScanColor

`public final void setScanColor(java.awt.Color color)`

Sets the color of the scan arc. Colors can (only) be changed each turn.

Example:

```
    setScanColor(Color.RED); // the red color
    setScanColor(new Color(255, 0, 0)); // also the red color
```

Specified by:

`setScanColor` in interface `IBaseBot`

Parameters:

`color` - is the color of the scan arc or `null` if the bot must use the default color instead.

getTracksColor

`public final java.awt.Color getTracksColor()`

Returns the color of the tracks.

Specified by:

`getTracksColor` in interface `IBaseBot`

Returns:

The color of the tracks or `null` if no color has been set yet, meaning that the default color will be used.

setTracksColor

`public final void setTracksColor(java.awt.Color color)`

Sets the color of the tracks. Colors can (only) be changed each turn.

Example:

```
    setTracksColor(Color.RED); // the red color
    setTracksColor(new Color(255, 0, 0)); // also the red color
```

Specified by:

`setTracksColor` in interface `IBaseBot`

Parameters:

`color` - is the color of the tracks or `null` if the bot must use the default color instead.

getGunColor

`public final java.awt.Color getGunColor()`

Returns the color of the gun.

Specified by:

`getGunColor` in interface `IBaseBot`

setGunColor

```
public final void setGunColor(java.awt.Color color)
```

Sets the color of the gun. Colors can (only) be changed each turn.

Example:

```
     setGunColor(Color.RED); // the red color
     setGunColor(new Color(255, 0, 0)); // also the red color
```

Specified by:

`setGunColor` in interface `IBaseBot`

Parameters:

`color` - is the color of the gun or `null` if the bot must use the default color instead.

calcMaxTurnRate

```
public final double calcMaxTurnRate(double speed)
```

Calculates the maximum turn rate for a specific speed.

Specified by:

`calcMaxTurnRate` in interface `IBaseBot`

Parameters:

`speed` - is the speed.

Returns:

The maximum turn rate determined by the given speed.

calcBulletSpeed

```
public final double calcBulletSpeed(double firepower)
```

Calculates the bullet speed given a firepower.

Specified by:

`calcBulletSpeed` in interface `IBaseBot`

Parameters:

`firepower` - is the firepower.

Returns:

The bullet speed determined by the given firepower.

calcGunHeat

```
public final double calcGunHeat(double firepower)
```

Calculates gun heat after having fired the gun.

Specified by:

`calcGunHeat` in interface `IBaseBot`

Parameters:

`firepower` - is the firepower used when firing the gun.

Returns:

The gun heat produced when firing the gun with the given firepower.

getEventPriority

```
     int scannedBotEventPriority = getPriority(ScannedBotEvent.class);
```

Specified by:

`getEventPriority` in interface `IBaseBot`

Parameters:

`eventClass` - is the event class to get the event priority for.

Returns:

the event priority for a specific event class.

See Also:

`DefaultEventPriority`, `IBaseBot.setEventPriority(java.lang.Class<dev.robocode.tankroyale.botapi.events.BotEvent>, int)`

setEventPriority

`public final void setEventPriority(java.lang.Class<BotEvent> eventClass, int priority)`

Changes the event priority for an event class. The event priority is used for determining which event types (classes) that must be fired and handled before others. Events with higher priorities will be handled before events with lower priorities.

Note that you should normally not need to change the event priority.

Specified by:

`setEventPriority` in interface `IBaseBot`

Parameters:

`eventClass` - is the event class to change the event priority for.

`priority` - is the new priority. Typically, a positive number from 1 to 150. The greater value, the higher priority.

See Also:

`DefaultEventPriority`, `IBaseBot.getEventPriority(java.lang.Class<dev.robocode.tankroyale.botapi.events.BotEvent>)`

isDebuggingEnabled

`public final boolean isDebuggingEnabled()`

Flag indicating if graphical debugging is enabled and hence if `IBaseBot.getGraphics()` can be used for debug painting.

Specified by:

`isDebuggingEnabled` in interface `IBaseBot`

Returns:

`true` if the graphics debugging is enabled; `false` otherwise.

getGraphics

`public final java.awt.Graphics2D getGraphics()`

Gets a graphics object that the bot can paint debug information to.

Example:

```
     var g = getGraphics();
     g.setColor(Color.blue);
     g.fillRect(50, 50, 100, 100);
```

Specified by:

`getGraphics` in interface `IBaseBot`

Returns: