BỘ GIÁO DỤC VÀ ĐÀO TẠO  
TRƯỜNG ĐẠI HỌC MỞ THÀNH PHỐ HỒ CHÍ MINH

BÁO CÁO

ROBOCODE

Nhóm thực hiện: Chấm

Ngành: Công Nghệ Thông Tin (IT)

Nguyễn Thành Nam (1951052126 - IT93)  
Nguyễn Trung Kiên (1951052091 - IT93)

TP. Hồ Chí Minh, tháng 5, năm 2021

MỤC LỤC

[I. GIỚI THIỆU VỀ ROBOCODE 2](#_Toc70764757)

[II. HIỆN THỰC HOÁ ROBOCODE 4](#_Toc70764758)

[1. CÁC PHƯƠNG THỨC TỪ MÃ NGUỒN ROBOCODE 4](#_Toc70764759)

[2. GIỚI THIỆU ROBOT CỦA NHÓM 5](#_Toc70764760)

[2.1 Ý tưởng robot 5](#_Toc70764761)

[3. THỰC HIỆN Ý TƯỞNG 6](#_Toc70764762)

[3.1 Thuật toán GuessFactor Targeting 6](#_Toc70764763)

[3.2 Thuật toán đánh cận chiến (Melee Strategy Algorithm) 8](#_Toc70764764)

[3.3 Lớp Robot 13](#_Toc70764765)

[3.4 Lớp Utility 14](#_Toc70764766)

[3.5 Lớp Movement\_1VS1 15](#_Toc70764767)

[3.6 Lớp Wave 17](#_Toc70764768)

[3.7 Các phương thức trong lớp Wave 18](#_Toc70764769)

[3.8 Các thuộc tính cần thiết của robot để thực thi 20](#_Toc70764770)

[3.9 Phương thức bắn (shooting) 21](#_Toc70764771)

[3.10 Phương thức cập nhật danh sách toạ độ (updateListLocation) 24](#_Toc70764772)

[3.11 Phương thức đánh giá toạ độ (evaluatePoint) 24](#_Toc70764773)

[3.12 Phương thức di chuyển (movement) 27](#_Toc70764774)

[3.13 Phương thức hoạt động sau khi bị tiêu diệt (onRobotDeath) 28](#_Toc70764775)

[3.14 Phương thức khi quét robot khác (onScannedRobot) 28](#_Toc70764776)

[3.15 Phương thức thực thi robot (run) 32](#_Toc70764777)

[3.16 Phương thức đổi màu robot (changeColor) 34](#_Toc70764778)

[III. TÀI LIỆU THAM KHẢO 35](#_Toc70764779)

# GIỚI THIỆU VỀ ROBOCODE

Theo wikipedia (<http://vi.wikipedia.org/wiki/Robocode>), Robocode là một chương trình trò chơi mà mục tiêu là tạo ra một robot, robot mô phỏng xe tăng trong một đấu trường chiến đấu để tìm và tiêu diệt các robot khác, nó được trang bị với hệ thống radar.

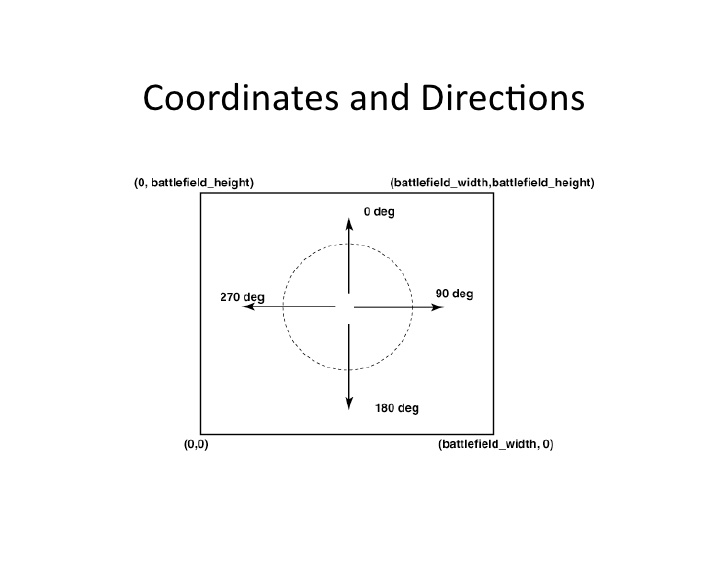
Một robot có thể di chuyển tới, lui ở các tốc độ khác nhau và di chuyển qua trái, qua phải. Các radar và tháp pháo có thể được chuyển sang trái hoặc sang phải độc lập với nhau so với phần còn lại của xe tăng. Khẩu súng có thể bắn. Khi cài đặt một trận đấu, ta có thể xem được trận đấu từ bên ngoài màn hình chơi. Người chơi là các lập trình viên, những người này sẽ không ảnh hưởng trực tiếp vào trò chơi mà thay vào đó người chơi phải lập trình làm sao cho robot thông minh có thể xử lý và phản ứng tất cả các sự kiện xảy ra trong lĩnh vực chiến đấu.

Khi một robot đối thủ lọt vào radar, một sự kiện sẽ được sinh ra và một hành động phù hợp sẽ được robot thực hiện. Robot có thể lấy được các thông tin về một đối thủ đang trong vùng quét của radar như tốc độ, heading, năng lượng còn lại, tên và gốc giữa heading của nó với robot đối thủ, khoảng cách giữa nó với đối thủ. Dựa vào các thông tin này mà robot sẽ đưa ra các hành động phù hợp. Ví dụ, khi phát hiện một đối thủ trong vòng radar, robot có thể dùng súng để bắn.

Các robot được viết bằng ngôn ngữ lập trình Java và trò chơi Robocode có thể chạy trên bất kỳ hệ điều hành có hỗ trợ nền tảng Java, bao gồm tất cả các hệ thống phổ biến như Windows, Mac OS, Linux ...

Chiến trường Robocode (battle field) là nơi mà tất cả các trận đấu diễn ra. Mỗi điểm có tọa độ x là tọa độ theo chiều ngang (battle field width) và y là tọa độ theo chiều dọc (battle field height) cặp tọa độ này sẽ thay đổi tùy theo vị trí của chiến trường. Vị trí ban đầu (0, 0) của chiến trường được đặt ở dưới cùng bên trái của chiến trường, một tọa độ (x, y) trên chiến trường luôn luôn là một cặp số thực dương.

**Giao diện Robocode**

****

**Toạ độ và vị trí của sàn đấu**

# HIỆN THỰC HOÁ ROBOCODE

Robot của nhóm sử dụng được kế thừa từ lớp **AdvancedRobot**.

## CÁC PHƯƠNG THỨC TỪ MÃ NGUỒN ROBOCODE

* **ahead(distance):** Di chuyển robot về phía trước tính theo đơn vị pixel.
* **back(distance):** Di chuyển robot về phía sau tính theo đơn vị pixel.
* **fire(power):** Sát thương đạn gây ra.
* **setAdjustGunForRobotTurn(independent):** Thiết lập cho súng quay độc lập với thân robot, nếu independent = true thì súng sẽ quay độc lập với thân và ngược lại.
* **setAdjustRadarForGunTurn(independent):** Thiết lập cho radar quay độc lập với súng robot, nếu independent = true thì radar sẽ quay độc lập với súng và ngược lại.
* **setAdjustRadarForRobotTurn(independent):** Thiết lập cho radar quay độc lập với thân robot, nếu independent = true thì radar sẽ quay độc lập với thân và ngược lại.
* **setColors(bodyColor, gunColor, radarColor, bulletColor, scanArcColor):** Thiết lập màu sắc cho các bộ phận trên robot các bộ phận gồm: thân robot, súng, radar, đạn, vùng radar quét.
* **getX():** Lấy tọa độ X của robot.
* **getY():** Lấy tọa độ Y của robot.
* **getBattleFieldWidth():** Lấy kích thước chiều ngang của chiến trường (sàn đấu).
* **getBattleFieldHeight():** Lấy kích thước chiều cao của chiến trường (sàn đấu).  
  **Ghi chú**: kích thước chiến trường mặc định là 800x600.
* **setTurnRadarRight(degree):** Thiết lập cho radar quay bên phải với một góc là degree. Degree nằm từ 0 đến 360 độ, nếu degree là số âm thì setTurnRadarRight đổi thành setTurnRadarLeft.
* **getRadarTurnRemaining():** Kiểm tra xem radar có quay không, nếu không quay thì có thể thực hiện hành động quay như trình bày ở trên.
* **execute():** Thực thi các lệnh trong đã đưa ra trong vòng lặp while(true).
* **getEnergy():** Lấy năng lượng hiện tại của robot mình.
* **getHeading():** Lấy góc của body robot mình, góc của robot nằm trong khoảng từ theo chiều kim đồng hồ.
* **getGunHeading():** Lấy góc của súng robot mình,giá trị nằm trong khoảng tương tự như getHeading().
* **getRadarHeading():** Lấy góc radar robot của mình, giá trị nằm trong khoảng tương tự như getHeading().
* **getVelocity():** Lấy vận tốc hiện tại của robot mình.
* **setAhead(distance):** Điều khiển robot đi thẳng với khoảng cách distance.
* **setBack(distance):** Điều khiển robot đi lùi với khoảng cách là distance.
* **setTurnRight(rotate):** Điều khiển body của robot xoay bên phải với góc rotate có kiểu double, rotate nhận giá trị là độ.
* **setTurnGunRight(rotate):** Điều khiển súng của robot xoay bên phải với góc rotate có kiểu double, rotate nhận giá trị là độ.
* **setFire(power):** Thiết lập bắn đạn cho robot,với viên đạn có kích thước là power. Power nằm trong khoảng từ
* **normalRelativeAngle(angle):** chuẩn hóa 1 góc từ 1 góc tương đối. Các góc được chuẩn hóa nằm trong phạm vi
* **Trong scannedRobotEvent e**:
  + **e.getBearing()** có giá trị từ . Nếu robot đối phương nằm bên trái của robot bạn thì e.getBearing() sẽ là số âm (0 đến -180). Ngược lại nếu robot đối phương nằm bên phải robot thì e.getBearing() sẽ là số dương (0 đến 180)
  + **Lưu ý** : so sánh robot đối phương nằm bên trái hay phải phụ thuộc vào  
    bodyHeading (getHeading()),có nghĩa là bodyHeading() làm chuẩn. Rồi xét robot bên trái hay bên phải)
  + **e.getHeading():** Lấy góc của body robot đối phương.
  + **e.getDistance():** Lấy khoảng cách giữa mình và đối phương.
  + **e.getVelocity():** Lấy vận tốc của đối phương.
* **onScannedRobot(ScannedRobotEvent event)** sự kiện xảy ra khi radar quét trúng đối thủ.
* **onHitWall(HitWallEvent event)** sự kiện xảy ra khi va vào tường.
* **onWin(WinEvent event)** sự khiện xảy ra khi thắng.
* Các phương thức khác các bạn có thể tham khảo [tại đây](https://robocode.sourceforge.io/docs/robocode/index.html?robocode/Robot.html)

## GIỚI THIỆU ROBOT CỦA NHÓM

### Ý tưởng robot

Ở thể thức 1v1 tạo 1 con robot có khả năng đuổi bám đối thủ nhưng vẫn giữ được khoảng cách cố định để tránh sự nguy hiểm (phân tích độ nguy hiểm – rick) từ đó đưa ra khoảng cách phù hợp, có khả năng tính toán đường đạn, vận tốc, di chuyển của địch và tự điều chỉnh đạn bắn ra phù hợp với năng lượng của mình.

Ở thể thức hỗn chiến (battle royal) ý tưởng như thể thức PvP nhưng sẽ được nâng cấp về mặt phần quét mục tiêu, không nhắm vào 1 mục tiêu cố định để tránh tình trạng mình bắn 1 mục tiêu ở khoảng cách xa trong khi đó có 1 mục tiêu khác ở gần mình, khi các đối thủ đã bị tiêu diệt gần hết thì chuyển sang trạng thái focus (nhắm 1 mục tiêu). Đồng thời luôn chọn những vị trí tốt như các góc ở tường, luôn tránh các đối thủ khác và hạn chế di chuyển vào phần trung tâm sàn đấu.

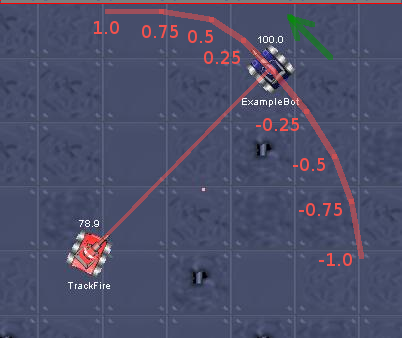
## THỰC HIỆN Ý TƯỞNG

### Thuật toán GuessFactor Targeting[[1]](#_[1]_https://robowiki.net/wiki/Guess)

GuessFactor (GF) là một cách đo góc bắn, tính đến hướng tương đối của kẻ thù tại thời điểm khai hỏa và góc thoát tối đa cho tình huống bắn cụ thể.

GuessFactor 1.0 đại diện cho độ lệch mà kẻ thù sẽ đạt được nếu nó tối đa hóa góc thoát trong khi di chuyển theo hướng hiện tại của nó so với robot bắn (tức là theo chiều kim đồng hồ hoặc ngược chiều kim đồng hồ), -1.0 đại diện cho độ lệch mà kẻ thù đạt được nếu nó tối đa hóa góc thoát của anh ta sau khi đảo ngược hướng và 0.0 đại diện cho độ lệch ổ trục hướng trực tiếp vào kẻ thù (như trong nhắm mục tiêu trực diện).

GuessFactors về cơ bản là các góc được chuẩn hóa. Chúng có liên quan đến làn sóng (wave surfing)[[2]](#_[1]_https://robowiki.net/wiki/Wave_), trong đó tất cả các GF của mỗi sóng (wave) được chỉ định một mối nguy hiểm được dự đoán trước, cho phép robot di chuyển theo hướng (True Surfing)[[3]](#_[2]_https://robowiki.net/wiki/Wave_) hoặc đến điểm (GoTo Surfing)[[4]](#_[3]_https://robowiki.net/wiki/Wave_) ít nguy hiểm nhất. Một số súng cũng sử dụng GuessFactors, trong đó chúng chọn một GF và bắn vào nó. GuessFactors cũng khả thi trong cận chiến, nhưng vì trong cận chiến, robot thường không di chuyển vuông góc với bạn nên GuessFactors thường ít hữu dụng hơn.



Những thành phần cần thiết trong GF bao gồm:

* Thông tin di chuyển (Movement Profile)[[5]](#_[5]_https://robowiki.net/wiki/Movem): Sự kết hợp của các hướng mà chúng ta có thể bắn và tần suất mà mỗi hướng là chính xác là những gì mọi người gọi là "thông tin" của robot, thực sự có thể xem thông tin của robot được lấy từ dữ liệu đã lưu của các robot khác.
* Trung bình động (Rolling Averages)[[6]](#_[6]_https://robowiki.net/wiki/Rolli): Về cơ bản là một cách ưu tiên dữ liệu mới hơn dữ liệu cũ hơn. Một khẩu súng có yếu tố đoán có thể rất tệ đối với một robot thay đổi chuyển động của nó trừ khi nó sử dụng trung bình động.
* Phân đoạn (Segementation)[[7]](#_[7]_https://robowiki.net/wiki/Segme): Đôi khi, chúng ta có thể nhận được nhiều dữ liệu có liên quan hơn bằng cách chia nhỏ dữ liệu trên một số thông số, thay vì chỉ nhìn vào thông tin chung.
* Đạn ảo (Virtual Bullets)[[8]](#_[8]_https://robowiki.net/wiki/Virtu): Một vật thể theo dõi vị trí của một viên đạn nếu nó được bắn theo một quỹ đạo nhất định và tìm hiểu xem nó có bắn trúng quỹ đạo đó hay không. Chúng được sử dụng trong tất cả các loại súng yếu tố phỏng đoán sớm nhất để phân tích MovementProfile của robot.
* Làn sóng (Wave)[[9]](#_[9]_https://robowiki.net/wiki/Waves): Hầu hết các robot hiện tại sử dụng GuessFactor Nhắm mục tiêu sử dụng sóng (wave) để tìm ổ trục mà chúng đáng lẽ phải bắn vào, như một giải pháp thay thế cho việc sử dụng một số đạn ảo (virtual bullet). Nhìn chung, chúng hiệu quả hơn một chút và cũng có thể được sửa đổi một chút để cho kết quả chính xác giống như đạn ảo (theo dõi TẤT CẢ các góc sẽ đạt được).

### Thuật toán đánh cận chiến (Melee Strategy Algorithm[[10]](#_[2]_https://robowiki.net/wiki/Melee))

#### Giới thiệu cận chiến

Cuộc cận chiến trong đó bao gồm robot của bạn và có ít nhất 2 robot khác trong sân đấu. Những robot này cần quản lý radar, mức tiêu thụ năng lượng của robot và vị trí thực địa cận chiến và hãy cẩn thận hơn so với các đối thủ 1 chọi 1. Một robot cận chiến tốt thường là một robot tốt 1 chọi 1, nhưng ngược lại điều này không nhất thiết phải đúng. Kích thước sân đấu cho các trận chiến này hầu như luôn lớn hơn mặc định và thường là khoảng 1000x1000 mặc dù các sân đấu lớn hơn có thể tạo ra các trận chiến thú vị khi các robot khác ở xa nhau đến mức chúng không thể nhìn thấy nhau. Những con robot này được luyện tập tốt sau 1v1 do các biến mới được thêm vào trận chiến.

#### Dòng thời gian của các cuộc cận chiến

|  |  |  |
| --- | --- | --- |
| Số robot trong  sân đấu | Mục tiêu | Mô tả |
| 10 | Các robot xuất hiện ở vị trí xấu thường bị tiêu diệt sớm. | Nếu có 1 robot khác ở gần mình, tốt nhất nên giết nó trước. Robot tốt sẽ giữ khoảng cách với robot khác và nếu ở trung tâm thì nên thoát ra khỏi đó ngay lập tức. |
| 6 – 7 | Tránh nhau ở các góc chết và cố gắng không bị đẩy vào thế gọng kìm. | Ta sẽ có lợi thế khi nhắm vào kẻ địch và không có kẻ địch khác nhắm vào mình. Ta có thể tấn công hầu hết các robot mà ta có thể quét được nhưng tỷ lệ trúng có thể giảm từ 25 – 30%. |
| 3 - 4 | Mỗi robot sẽ chọn 1 góc để bắn tỉa lẫn nhau. Cố gắng để những robot khác tiêu diệt lẫn nhau thay vì nhắm tới ta. | Khi số lượng robot trên sàn vừa đủ, kết hợp với tính năng nhắm mục tiêu chuẩn. Khả năng cao bạn sẽ chọn ở trong 1 góc, nếu có 1 robot khác chết (nó nằm đối diện với bạn), nguy cơ bạn sẽ là đối tượng nhắm đến tiếp theo của các robot còn lại. |
| 1 vs 1 | Trận chiến 1v1 cho đến khi kết thúc trận chiến. | Yêu cầu khả năng nhắm chính xác mục tiêu cao, sự chuẩn xác của nòng súng quyết định chiến thắng của ta. |

Trên thực tế, các giai đoạn này không phải là thứ bạn sẽ đưa vào khi tạo một robot cận chiến, chúng quan sát về trận chiến sẽ như thế nào giữa một số robot cận chiến tốt. Thực tế là thông qua các giai đoạn này ta xác định được về vị trí và chuyển động nói lên rất nhiều điều quan trọng trong cận chiến. Lý do các giai đoạn này thực sự quan trọng cần hiểu là để bạn có thể phân tích hiệu suất của robot cận chiến nếu bạn đặt nó vào trận chiến với các robot khác có cùng trình độ.

#### Cách kết hợp robot cận chiến tốt

Cách tốt nhất là di chuyển liên tục để không bị đối phương nhắm vào, nhưng không nên di chuyển tuyến tính vì rất dễ đoán hướng đi. Đồng thời luôn chọn những góc tốt nhất thể thực hiện bắn.

#### Lý thuyết về chuyển động cận chiến

Robot của bạn sẽ né tránh mọi viên đạn nếu không có ai bắn vào bạn, và vì vậy hầu hết các chuyển động cận chiến đều tập trung theo cách này hay cách khác để không bị tấn công. Đây là điều làm cho chuyển động cận chiến trở nên thú vị. Nếu không thể tránh khỏi việc bạn bị nhắm đến, thì bạn có thể bắt đầu lo lắng về việc không bị đánh.

Hai mục tiêu di chuyển kết hợp thành sự cân bằng giữa "ở đúng nơi" (không bị bắn vào) và "di chuyển đúng hướng" (không bị bắn). Vì bạn chỉ có thể di chuyển theo một hướng, nên đôi khi bạn phải chọn chuyển động về phía đúng nơi hoặc đi đúng hướng. Vì các robot khác nhau ưu tiên những thứ này khác nhau, một số sẽ cực kỳ dễ đoán và dễ trúng, nhưng điều đó không quan trọng cho đến khi ai đó bắn vào chúng. Trên thực tế, về mặt lý thuyết, miễn là không có ai bắn vào bạn, lý do duy nhất bạn thực sự phải tiếp tục di chuyển là đảm bảo rằng họ tiếp tục không bắn vào bạn.

#### Các cách phổ biến để thực hiện chuyển động cận chiến

Có rất nhiều cách để mọi người hoàn thành các mục tiêu không bị nhắm đến và không bị ảnh hưởng, phổ biến gồm:

* Anti-Gravity Movement[[11]](#_[2]_https://robowiki.net/wiki/Anti-)
* Corner Movement[[12]](#_[3]_https://robowiki.net/wiki/Corne)
* Pattern Movement[[13]](#_[4]_https://robowiki.net/wiki/Patte)

#### Chuyển động rủi ro tối thiểu (Minimum Risk Movement[[14]](#_[5]))

Nhìn bề ngoài, điều này trông giống như một triển khai của Anti-GravityMovement và có thể sự khác biệt chỉ là ngữ nghĩa khó hiểu đối với một số người, nó được triển khai đủ khác nhau đến mức điểm tương đồng duy nhất thực sự là các nguyên tắc cơ bản cơ bản của chuyển động cận chiến. Nó cũng có vẻ linh hoạt hơn nhiều so với Anti-GravityMovement.

Có hai phần cơ bản đối với MinimumRiskMovement:

* Hàm rủi ro - Xác định mức độ tốt hay xấu của một điểm.
* Một chức năng tạo điểm - tạo ra các ứng viên để thử trong chức năng rủi ro.

Sau đó, bạn chỉ cần đi đến điểm có rủi ro thấp nhất. Tính ngẫu nhiên có thể hỗ trợ một trong hai chức năng một cách hiệu quả. Ngoài ra, là một điều chỉnh thú vị, bạn cần quyết định tìm kiếm một điểm để đi đến liên tục thay vì chỉ khi bạn đạt đến điểm cuối cùng mà bạn sẽ đến. Cả hai chiến lược đều có giá trị - bạn có thể gặp rắc rối nếu tiếp tục thay đổi quyết định về điểm, nhưng bạn cũng có thể phản ứng nhanh hơn với những thay đổi trên sàn đấu.

#### Khả năng cần có của chuyển động trong đánh cận chiến

|  |  |
| --- | --- |
| Khả năng | Mô tả |
| Cách xa các robot khác | Việc sử dụng bình phương nghịch đảo của khoảng cách tới mỗi robot lại hoạt động rất hiệu quả (mối đe dọa tiến gần đến 0 tại một thời điểm nào đó). Điều này có thể sẽ ngăn bạn ra khỏi hành động đến gần địch. |
| Di chuyển vuông góc | Đặc biệt là các robot ở gần bạn. |
| Đừng đập tường hoặc va chạm các robot khác quá nhiều | Đây có thể là một vấn đề lớn hơn trong cận chiến, chỉ vì bạn sẽ ở gần các bot khác và với các bức tường nhiều hơn là trong 1v1. |
| Tránh xa trung tâm sân đấu | Bạn nên ở giữa nếu trận chiến vẫn còn lớn và tất cả các robot khác đang ở bên ngoài. Tuy nhiên, nếu bạn vẫn ở giữa với 4 hoặc 5 robot còn lại, bạn có thể sẽ không trụ được lâu. |
| Đừng ở một chỗ quá lâu | Cũng giống như trong 1 chọi 1, điều này có thể dẫn đến cái chết nhanh chóng. Không quan tâm vị trí đó tuyệt vời như thế nào, ai đó sẽ đến gần nó để khiến bạn gặp nguy hiểm trong thời gian ngắn. |
| Chiến đấu thay đổi liên tục | Có khả năng điều chỉnh và thay đổi nhanh chóng, và đừng cố gắng lên kế hoạch quá xa. |
| Đoán xem ai sẽ bắn bạn | Bạn muốn ở điểm mà ít robot nhất sẽ nhắm mục tiêu bạn.  Bạn chỉ phải lo lắng về việc khó bị tấn công bởi các robot có khả năng nhắm mục tiêu bạn (nói cách khác, hướng bạn di chuyển đến một điểm nhất định chỉ quan trọng so với các robot có khả năng nhắm mục tiêu đến điểm đó hoặc của bạn vị trí hiện tại). |

#### Điều chỉnh trang bị cho robot

**Radar cận chiến:** những gì bạn làm với radar của mình thực sự phụ thuộc vào chiến lược nhắm mục tiêu của bạn là gì và bạn có muốn bắn mù hay không (bắn mù nghĩa là bạn không bắn khi bạn thực sự có thể nhìn thấy kẻ thù của mình, mà là trong vòng chạy chung của bạn hoặc thứ gì đó). Nếu bạn chỉ quay radar liên tục, bạn gần như cần phải bắn mù, nếu không thì súng của bạn sẽ không có thời gian để quay trước khi bắn. Tuy nhiên, mình nhận thấy (một lần nữa, nếu súng của bạn không phụ thuộc vào radar quay, giống như một số thiết bị ghép mẫu cận chiến) thì việc bắn mù sẽ làm giảm tỷ lệ bắn trúng của tôi một chút. Đặc biệt nếu bạn đang sử dụng súng có yếu tố đoán trong cận chiến, sử dụng khóa radar cận chiến. Khóa radar cận chiến sẽ quay radar của bạn cho đến khi bạn tiến gần đến việc bắn, và sau đó khóa mục tiêu của bạn như thể chúng là robot duy nhất trên sân trong vài tích tắc cho đến khi bạn khai hỏa. Ngoài ra, nếu bạn có khóa radar trong mã của mình, bạn cũng có thể sử dụng nó cho thể thức solo.

**Hoả lực cận chiến:** rất nhiều đòn đánh xảy ra sớm trong một trận đấu cận chiến, nhưng cuối cùng thì bạn có thể bị tiêu hao khá nhiều năng lượng. Một số người lấy yếu tố sinh tồn của cận chiến có nghĩa là họ phải luôn bắn những viên đạn công suất thấp. Điều này chỉ đúng một phần - bạn có thể sẽ bắt đầu trận chiến cận chiến gần với các robot khác. Trên thực tế, gần đến mức bạn rất có thể có được rất nhiều đòn đánh (cho hoặc chống lại bạn) trong 5 giây đầu tiên của một trận đấu. Đừng ngại bắn những viên đạn đủ mạnh để kết liễu một con bot nếu chúng thực sự ở gần bạn (vì bạn vẫn muốn kết liễu chúng nếu chúng bắn trúng bạn hoặc ai đó trước khi viên đạn của bạn chạm tới chúng).

**Chọn mục tiêu:** Ai có năng lượng thấp nhất? Có một số lý do bạn có thể muốn nhắm mục tiêu họ: Để nhận được điểm thưởng tiêu diệt. Nếu chúng ít nhất là ở gần bạn và chúng ở một phần "an toàn hơn" của chiến trường, bạn có thể thế chỗ chúng nếu chúng chết. Nếu sự di chuyển của bạn cho phép bạn đến gần hơn với những con robot thấp hơn năng lượng, bạn có thể có nhiều khả năng làm điều này hơn. Ai tiếp tục bắn vào bạn? Có hai cách để tránh bị bắn - khiến họ bắn vào người khác hoặc hạ gục họ. Tiêu diệt ai đó có thể giúp bạn tiết kiệm năng lượng! Cố gắng không lao vào giữa kẻ thù quá nhiều, đặc biệt là khi bạn đã sẵn sàng khai hỏa. Nếu bạn chỉ sử dụng khoảng cách để chọn mục tiêu, hãy thử chỉ chuyển đổi mục tiêu nếu mục tiêu mới gần hơn 100 pixel hoặc gần hơn 10% hoặc một cái gì đó.

#### Kiểm soát kẻ địch

Nếu bạn đang cố gắng chuyển đổi robot 1vs1 thành robot cận chiến, bạn sẽ nhanh chóng phát hiện ra rằng tất cả các giả định nhỏ mà bạn từng đặt ra khi mã hóa robot 1vs1 của mình đều sai. Một thách thức lớn để lưu trữ hiệu quả thông tin về kẻ thù của bạn (như vị trí hiện tại, hướng đi, vận tốc, v.v.) cũng như thông tin liên tục về kẻ thù của bạn (như thông tin nhắm mục tiêu). Một trong những thách thức lớn là làm thế nào để xác định rằng một kẻ thù nhất định đã chết. Có một cách để giải quyết là có một đối tượng (lớp EnemyInfo) trong bản đồ có vị trí, hướng, vận tốc và tất cả những thứ hiển nhiên trong đó, cũng như danh sách các đợt (lưu ý - tất cả những thứ này nên biến mất giữa các hiệp). Bản đồ này không nhất thiết phải tĩnh (mặc dù nó có thể là vì lý do mã hóa), bởi vì nó được tạo lại mỗi vòng. Trong onScannedRobot, kiểm tra xem robot mình đang quét có trong bản đồ hay không và nếu không, mình đặt nó vào đó. Sau đó, mình lấy đối tượng EnemyInfo và thao tác trên nó trên onScannedRobot. onRobotDeath loại bỏ đối tượng của robot đó khỏi bản đồ nếu chúng tồn tại (vì vậy robot không còn tồn tại khi mình không quan tâm đến nó nữa). Trong khi đó, mình có một bản đồ khác ánh xạ tên với số liệu thống kê về hệ số đoán, là bản đồ tĩnh và không được khởi động lại giữa các vòng đấu.

#### Các thuật toán trong nhắm mục tiêu trong đánh cận chiến

* Head On Targeting[[15]](#_[6]_https://robowiki.net/wiki/Head-)
* [LinearTargeting](https://robowiki.net/wiki/LinearTargeting)[[16]](#_[7]_https://robowiki.net/wiki/Linea) / [CircularTargeting](https://robowiki.net/wiki/CircularTargeting)[[17]](#_[8]_https://robowiki.net/wiki/Circu)và các biến thể
* Pattern Matching[[18]](#_[9]_https://robowiki.net/wiki/Patte) trong cận chiến
* Guess Fator Targeting[[19]](#_[10]_https://robowiki.net/wiki/Gues) trong cận chiến

#### Đánh giá robot cận chiến

Điều đầu tiên cần làm là tạo một trận đấu thử nghiệm. Tìm một nhóm robot ít nhất ở cùng khu vực lân cận với bạn theo khả năng, một số tốt hơn và một số kém hơn. Ghi lại thông tin các trận chiến và những điều chỉnh mà tôi đã thực hiện giữa chúng. Tất nhiên, đừng quá chú trọng đến điểm số, vì chúng thường khác nhau rất nhiều. Hãy nâng cấp nó dần dần.

### Lớp Robot

*class* Robot *extends* Point2D.Double {  
 *public long* scanTime;  
 *public boolean* alive = *true*;  
 *public double* energy;  
 *public* String name;  
 *public double* gunHeadingRadians;  
 *public double* absoluteBearingRadians;  
 *public double* velocity;  
 *public double* heading;  
 *public double* lastHeading;  
 *public double* shootAbleScore;  
 *public double* dist;  
}

* Lớp Robot này sẽ được kế thừa từ lớp Point2D vì trong lớp này ta có thể lấy toạ độ từ đó có thể viết cái phương thức tính toạ độ.
* Lớp này giúp ta có thể tạo robot của mình và đối thủ để lưu các dữ liệu cần thiết cho việc tính toán cụ thể.
* Các thuộc tính gồm:
  + scanTime: thời gian quét của robot
  + avlive: kiểm tra robot còn trên sàn đấu hay không
  + energy: năng lượng của robot
  + name: tên của robot
  + gunHeadingRadians: góc quay của súng
  + absoluteBearingRadians: góc va chạm của robot
  + velocity: vận tốc của robot
  + heading: lấy góc của robot
  + lastHeading: lấy góc lần cuối cùng của robot
  + shootAbleScore: năng lượng thích hợp để khai hoả
  + dist: khoảng cách giữa robot

### Lớp Utility

*public static class* Utility {  
 *static double* clamp(*double* value, *double* min, *double* max) {  
 *return* Math.*max*(min, Math.*min*(max, value));  
 }  
   
 *static* doublerandomBetween(*double* min, *double* max) {  
 *return* min + Math.*random*() \* (max - min);  
 }  
  *static* Point2D project(Point2D sourceLocation, *double* angle, *double* length) {  
 *return new* Point2D.Double(sourceLocation.getX() + Math.*sin*(angle) \* length,  
 sourceLocation.getY() + Math.*cos*(angle) \* length);  
 }  
  *static double* absoluteBearing(Point2D source, Point2D target) {  
 *return* Math.*atan2*(target.getX() - source.getX(), target.getY() - source.getY());  
 }

*static int* sign(*double* v) {  
 *return* v < 0 ? -1 : 1;  
 }  
}

* + - * Lớp Utility có các phương thức gồm:
* **clamp(value, min, max):** trả về giá trị ở giữa của 3 giá trị này khi có value cho trước, giá trị được kẹp (clamp) ở 2 đầu.
* **randomBetween(min, max):** trả về giá trị được tạo ngẫu nhiên và giá trị đó nằm trong khoảng (min, max).
* **project(sourceLocation, angle, length):** trả về toạ độ của robot đối thủ dựa vào hướng di chuyển và khoảng cách từ robot của ta đến robot địch.
* **absoluteBearing(p1, p2):** tính góc của điểm là kết quả tạo ra của 2 điểm nguồn và điểm đang nhắm tới trong hệ toạ độ cực.

x = target.getX() – source.getX()

y = target.getY() – source.getY()

* + Các hệ toạ độ được sử dụng :
  + Hệ tọa độ đề các (x, y) - rectangular coordinates system[[20]](#_[11]_https://en.wikipedia.org/wiki/)
  + Hệ toạ độ cực (r, theta) - polar coordinate system[[21]](#_[12]_https://en.wikipedia.org/wiki/)
* **sign(v):** đảo chiều di chuyển

### Lớp Movement\_1VS1

*private static final double BATTLE\_FIELD\_WIDTH = 800;  
private static final double BATTLE\_FIELD\_HEIGHT = 600;  
private static final double MAX\_TRY\_TIME = 125;  
private static final double REVERSE\_TUNER = 0.421075;  
private static final double DEFAULT\_EVASION = 1.2;  
private static final double WALL\_BOUNCE\_TUNER = 0.699484;  
private final AdvancedRobot robot;  
private final Rectangle2D fireField = new Rectangle2D.Double(WALL\_MARGIN, WALL\_MARGIN,  
 BATTLE\_FIELD\_WIDTH - WALL\_MARGIN \* 2, BATTLE\_FIELD\_HEIGHT - WALL\_MARGIN \* 2);  
private double direction = 0.4;*

* Lớp Movement\_1VS1 được tạo ra để nâng cao khả năng di chuyển, nó phù hợp trong thể thức solo 1v1.
* Các thuộc tính bao gồm:
  + BATTLE\_FIELD\_WIDTH: chiều rộng mà robot có thể di chuyển
  + BATTLE\_FIELD\_HEIGHT: chiều cao mà robot có thể di chuyển
  + MAX\_TRY\_TIME: thời gian robot di chuyển
  + REVERSE\_TUNER: điều chỉnh hướng của robot
  + DEFAULT\_EVASION: tỷ lệ né của robot
  + WALL\_BOUNCE\_TUNER: tỷ lệ tránh khỏi tường của robot
  + robot: robot muốn tạo ra làn sóng
  + fireField: tầm khai hoả của robot
* Phương thức onScannedRobot():
  + Tạo 1 con robot tượng trưng cho con robot địch và thực hiện tính các giá trị như góc bearing tuyệt đối, khoảng cách. Sau đó tính tọa độ 1 điểm đại diện cho vị trí robot của ta hiện tại, 1 điệm đại điện cho vị trí robot của địch hiện tại. Tạo thêm 1 điểm dùng để lưu trữ diểm đến của robot ta (dùng cho phía sau), 1 giá trị thử thực hiện (tryTime) = 0.

Robot enemy = new Robot();  
enemy.absoluteBearingRadians = robot.getHeadingRadians() + e.getBearingRadians();  
enemy.dist = e.getDistance();  
Point2D robotLocation = new Point2D.Double(robot.getX(), robot.getY());  
Point2D enemyLocation = Utility.project(robotLocation, enemy.absoluteBearingRadians, enemy.dist);

* + Thuật toán di chuyển
    - Bước 1: lặp tối đa MAX\_TRY\_TIME = 125 để tìm điểm đến mới của robot cả chúng ta (điều kiện là điểm đó phải nằm trong vùng bắn của robot (fireField))

Tọa độ điểm đến mới

* + - * + X= e.x + sin(e.absBearing + PI + direction ) \* e.distance \* (1.2 - tryTime /100) )
        + Y= e.y + cos(e.absBearing + PI + direction ) \* e.distance \* (1.2 - tryTime /100) )

while (!fireField.contains(robotDestination = Utility.project(enemyLocation, enemy.absoluteBearingRadians + Math.PI + direction,  
 enemy.dist \* (DEFAULT\_EVASION - tryTime / 100.0))) && tryTime < MAX\_TRY\_TIME)  
 tryTime++;

* + - Bước 2: Đổi hướng di chuyển trong 2 trường hợp. Khi lấy 1 số ngẫu nhiên trong phạm vi [0, 1) < vận tốc đạn /(0.421075 \* khoảng cách cảu địch) hoặc số lần thử (tryTime) > khoảng cách của địch / (vận tốc đạn \* 0.699484)

if ((Math.random() < (Rules.getBulletSpeed(enemyFirePower) / REVERSE\_TUNER) / enemy.dist ||  
 tryTime > (enemy.dist / Rules.getBulletSpeed(enemyFirePower) / WALL\_BOUNCE\_TUNER)))  
 direction = -direction;

* + - Bước 3: tính góc để di chuyển. Góc di chuyển là góc của điểm được tạo ra bởi 2 điểm (vị trí robot của ta hiện tại, vị trí điểm đếm mới) trong hệ tọa độ cực – góc di chuyển hiện tại của robot

double angle = Utility.absoluteBearing(robotLocation, robotDestination) - robot.getHeadingRadians();

* + - Bước 4: thiết lập di chuyển và quay radar cho robot dựa vào góc vừa tìm được

robot.setAhead(Math.cos(angle) \* 100);  
robot.setTurnRightRadians(Math.tan(angle));

### Lớp Wave

static class Wave extends Condition {  
 static Point2D targetLocation;  
 double bulletPower;  
 Point2D gunLocation;  
 double bearing;  
 double lateralDirection;  
 private static final double MAX\_DISTANCE = 900;  
 private static final int DISTANCE\_INDEXES = 5;  
 private static final int VELOCITY\_INDEXES = 5;  
 private static final int BINS = 25;  
 private static final int MIDDLE\_BIN = (BINS - 1) / 2;  
 private static final double MAX\_ESCAPE\_ANGLE = 0.7;  
 private static final double BIN\_WIDTH = MAX\_ESCAPE\_ANGLE / (double) MIDDLE\_BIN;  
 private static final int[][][][] statBuffers = new int[DISTANCE\_INDEXES][VELOCITY\_INDEXES][VELOCITY\_INDEXES][BINS];  
 private int[] buffer;  
 private final AdvancedRobot robot;  
 private double distanceTraveled;  
}

* Lớp Wave được tạo ra để sử dụng trong chuyển động hoặc nhắm mục tiêu, lớp Wave được kế thừa từ lớp Condition.
* Một làn sóng (wave) bao gồm vị trí nguồn (của robot bắn), vận tốc (thường dựa trên sức mạnh đạn của robot bắn), thời gian sóng được tạo ra và hướng đến mục tiêu tại thời điểm khai hỏa.
* Sóng "đứt" hoặc "chạm" khi khoảng cách mà sóng đã đi (vận tốc làn sóng \* (thời gian hiện tại – thời gian khai hoả)) lớn hơn khoảng cách từ nguồn sóng đến đối phương. Bạn có thể tưởng tượng làn sóng như một vòng tròn tỏa ra từ điểm mà nó được bắn ra (đó là nơi bắt nguồn của tên). Một khi sóng bị đứt, có thể dễ dàng suy ra góc bắn, độ lệch so với góc ban đầu của mục tiêu từ đó sẽ bắn trúng kẻ thù.
* Khả năng nhắm mục tiêu GuessFactor[[1]](#_[1]__https://robowiki.net/wiki/Gues) được áp dụng để sử dụng trong thể thức solo.
* Các thuộc tính gồm:
  + targetLocation: vị trí mà robot đang hướng tới
  + bulletPower: sức mạnh của đạn
  + gunLocation: vị trí của súng
  + bearing: góc va chạm
  + lateralDirection: hướng của robot
  + MAX\_DISTANCE: khoảng cách tối đa của ta và robot đối thủ
  + DISTANCE\_INDEXES: lưu trữ khoảng cách của đối thủ
  + VELOCITY\_INDEXES: lưu trữ vận tốc của đối thủ
  + BINS: số lượng thùng để lưu trữ dữ liệu

### Các phương thức trong lớp Wave

* + - * **test():** Nhằm kiểm tra điều kiện để tính toán xem viên đạn có thể bắn trúng đích hay không.

public boolean test() {  
 advance();  
 if (hasArrived()) {  
 buffer[currentBin()]++;  
 robot.removeCustomEvent(this);  
 }  
 return false;  
}

* Bước 1: tính khoảng cách đi đến của viên đạn (khoảng cách + = vận tốc di chuyển).
* Bước 2: kiểm tra xem cái khoảng cách vừa tìm được ở bước 1 có lớn hơn khoảng cách từ vị trí robot của ta tới điểm mà ta đang nhắm tới không. Nếu có thì tăng chỉ số thùng chứa dữ liệu lên 1 và xóa sự kiện bắn đã đăng kí trước đó.
* Bước 3: trả về điều kiện là false.
  + - * **mostVisitedBearingOffset():** tính độ dời của góc bearing = hướng di chuyển \* 0.05 - ( chỉ số của thùng chứa dữ liệu cao nhất -14)

double mostVisitedBearingOffset() {  
 return (lateralDirection \* BIN\_WIDTH) \* (mostVisitedBin() - MIDDLE\_BIN);  
}

* + - * **setSegmentation():** lấy dữ liệu cho các bin từ các dữ liệu đã tổng hợp được và phân tích.

void setSegmentations(double distance, double velocity, double lastVelocity) {  
 int distanceIndex = (int) (distance / (MAX\_DISTANCE / DISTANCE\_INDEXES));  
 int velocityIndex = (int) Math.abs(velocity / 2);  
 int lastVelocityIndex = (int) Math.abs(lastVelocity / 2);  
 buffer = statBuffers[distanceIndex][velocityIndex][lastVelocityIndex];  
}

* + - * **advance():** tính khoảng cách di chuyển của đạn

private void advance() {  
 distanceTraveled += Rules.getBulletSpeed(bulletPower);  
}

* + - * **hasArrived():** kiểm tra xem viên đạn bắn ra có tới mục tiêu hay không

private boolean hasArrived() {  
 return distanceTraveled > gunLocation.distance(targetLocation) - WALL\_MARGIN;  
}

* + - * **currentBin():** trả vể chỉ số của thùng chứa dữ liệu hiện tại

private int currentBin() {  
 int bin = (int) Math.round(((Utils.normalRelativeAngle  
 (Cham.Utility.absoluteBearing(gunLocation, targetLocation) - bearing)) /  
 (lateralDirection \* BIN\_WIDTH)) + MIDDLE\_BIN);  
 return (int) Utility.clamp(bin, 0, BINS - 1);  
}

* + - * **mostVisitedBin():** trả về chỉ số của thùng chứa dữ liệu cao nhất

private int mostVisitedBin() {  
 int mostVisited = MIDDLE\_BIN;  
 for (int i = 0; i < BINS; i++)  
 if (buffer[i] > buffer[mostVisited])  
 mostVisited = i;  
 return mostVisited;  
}

* + - * **onScannedRobot()**
  + Tạo 1 con robot tượng trưng cho con robot địch và thực hiện tính các giá trị như góc bearing tuyệt đối, khoảng cách. Sau đó tính tọa độ 1 điểm đại diện cho vị trí robot của ta hiện tại, 1 điệm đại điện cho vị trí robot của địch hiện tại. Tạo thêm 1 điểm dùng để lưu trữ diểm đến của robot ta (dùng cho phía sau), 1 giá trị thử thực hiện (tryTime) = 0

Robot enemy = new Robot();  
enemy.absoluteBearingRadians = robot.getHeadingRadians() + e.getBearingRadians();  
enemy.dist = e.getDistance();  
Point2D robotLocation = new Point2D.Double(robot.getX(), robot.getY());  
Point2D enemyLocation = Utility.project(robotLocation, enemy.absoluteBearingRadians, enemy.dist);

* + Thuật toán di chuyển
    - * + Bước 1: Lặp tối đa MAX\_TRY\_TIME = 125 để tìm điểm đến mới của robot cả chúng ta (điều kiện là điểm đó phải nằm trong vùng bắn của robot (fireField))

Tọa độ điểm đến mới

while (!fireField.contains(robotDestination = Utility.project(enemyLocation, enemy.absoluteBearingRadians + Math.PI + direction,  
 enemy.dist \* (DEFAULT\_EVASION - tryTime / 100.0))) && tryTime < MAX\_TRY\_TIME)  
 tryTime++;

* + - * + Bước 2: Đổi hướng di chuyển trong 2 trường hợp. Khi lấy 1 số ngẫu nhiên trong phạm vi [0, 1) < vận tốc đạn /(0.421075 \* khoảng cách cảu địch) hoặc số lần thử (tryTime) > khoảng cách của địch / (vận tốc đạn \* 0.699484)

if ((Math.random() < (Rules.getBulletSpeed(enemyFirePower) / REVERSE\_TUNER) / enemy.dist ||  
 tryTime > (enemy.dist / Rules.getBulletSpeed(enemyFirePower) / WALL\_BOUNCE\_TUNER)))  
 direction = -direction;

* + - * + Bước 3: Tính góc để di chuyển. Góc di chuyển là góc của điểm được tạo ra bởi 2 điểm (vị trí robot của ta hiện tại, vị trí điểm đếm mới) trong hệ tọa độ cực – góc di chuyển hiện tại của robot

double angle = Utility.absoluteBearing(robotLocation, robotDestination) - robot.getHeadingRadians();

* + - * + Bước 4: thiết lập di chuyển và quay radar cho robot dựa vào góc vừa tìm được

robot.setAhead(Math.cos(angle) \* 100);  
robot.setTurnRightRadians(Math.tan(angle));

### Các thuộc tính cần thiết của robot để thực thi

*static final int* AMOUNT\_PREDICTED\_POINTS = 150;  
static final int WALL\_MARGIN = 18;  
HashMap<String, Robot> enemyList = *new* HashMap<>();  
Robot myRobot = *new* Robot();  
Robot targetRobot;  
*List*<Point2D.Double> possibleLocations = *new* ArrayList<>();  
Point2D.Double targetPoint = *new* Point2D.Double(60, 60);  
Rectangle2D.Double battleField = *new* Rectangle2D.Double();  
*int* idleTime = 30;  
private static double lateralDirection;  
private static double preEnemyVelocity;  
private static Movement\_1VS1 movement1VS1;

* AMOUNT\_PREDICTED\_POINTS: là số lượng vị trí dự đoán (dự kiến) của danh sách các các điểm có thể di chuyển dùng biến static để lưu (số lượng nhóm mình chọn là 150 vị trí).
* WALL\_MARGIN: khoảng cách giữa robot với tường
* enemyList: dùng HashMap để lưu trữ dữ liệu của đối thủ (vì mỗi đối thủ là duy nhất không có trùng lắp) để tính toán chuyển động và nhắm mục tiêu của chúng.
* myRobot: tạo 1 đối tượng robot của mình để thực thi hàm.
* targetRobot: đại diện cho 1 con robot mà ta đang nhắm tới
* possibleLocations: tạo danh sách lưu toạ độ mà robot của ta thể di chuyển tới.
* targetPoint: dùng để lưu toạ độ có khả năng rủi ro thấp nhất.
* battleField: dùng để hạn chế gọi các phương thức getBattleFieldWidth và getBattleFieldHeight, đồng thời nhằm xác định kích thước sân đấu.
* idleTime: tính thời gian mà ta nhắm tới 1 vị trí mà độ rủi ro thấp nhất để di chuyển, sau 1 khoảng thời gian thì chuyển mục tiêu khác.
* lateralDireciton: hướng bên của kẻ địch.
* preEnemyVelocity: vận tốc trước đó của địch.
* movement1VS1: đối tượng trong thể thức solo.

### Phương thức bắn (shooting)

Kiểm tra xem nếu ta đang nhắm tới 1 robot và robot đó còn sống thì thực hiện hành động: *if* (targetRobot != *null* && targetRobot.alive)

* Tính năng lượng của đạn bắn ra với mục tiêu đạt được lượng sát thương lớn nhất mà tốn ít năng lượng nhất
  + Bước 1: Tính năng lượng dựa vào khoảng cách
  + Bước 2: Tính năng lượng dựa trên năng lượng của đối phương và năng lượng của địch
  + Bước 3: Lấy giá trị ở giữa trong 3 giá trị (power vừa tính ở bước 2, giá trị power MIN của đạn, giá trị power MAX của đạn)

*double* dist = myRobot.distance(targetRobot);  
*double* power = (dist > 850 ? 0.1 :  
 (dist > 700 ? 0.5 : (dist > 250 ? 2.0 : 3.0)));  
power = Math.*min*(myRobot.energy / 4d,  
 Math.*min*(targetRobot.energy / 3d, power));  
power = Utility.*clamp*(power, 0.1, 3.0);

* Nhắm mục tiêu: dùng thuật toán của phương pháp Circular Targeting[[17]](#_[17]__https://robowiki.net/wiki/Cir)
  + Bước 1: Thiết lập vị trí nhắm là robot mục tiêu hiện tại, khoảng thời gian bắn trúng là 0, deltaHead (độ lệch góc giữa hướng đi mới và hướng đi cũ) của robot địch = heading hiện tại – heading cũ

long deltaHitTime;  
Point2D.Double shootAt = new Point2D.Double();  
double head, deltaHead, bulletSpeed;  
double predictX, predictY;  
predictX = targetRobot.getX();  
predictY = targetRobot.getY();  
head = targetRobot.heading;  
deltaHead = head - targetRobot.lastHeading;  
shootAt.setLocation(predictX, predictY);  
deltaHitTime = 0;

* + Bước 2: Đưa vào vòng lặp cho đến khi khoảng cách giữa robot của ta và vị trí ngắm bắn – 18)/tốc độ của viên đạn mang năng lượng power <= khoảng thời gian bắn trúng thì dừng ()

*do* {

*…condition*  
} *while* ((*int*) Math.*round*((shootAt.distance(myRobot) – WALL\_MARGIN) / Rules.*getBulletSpeed*(power)) > deltaHitTime);

* + - Dự đoán vị trí tiếp theo của robot địch dựa vào vận tốc và hướng di chuyển (phương pháp tính lateral velocity)

* + - * Trong đó:
        + head là hướng của robot địch
        + velocity: vận tốc của robot địch

predictX += Math.*sin*(head) \* targetRobot.velocity;  
predictY += Math.*cos*(head) \* targetRobot.velocity;

* + - Cập nhật lại hướng của robot địch bằng cách cộng thêm 1 góc deltaHeading
    - Tăng khoảng thời gian bắn trúng lên 1
    - Thiết lập vùng bắn thuận lợi của robot là 1 hình chữ nhật có kích thước tại tọa độ (18,18)

head += deltaHead;  
deltaHitTime++;  
Rectangle2D.Double fireField = *new* Rectangle2D.Double(WALL\_MARGIN, WALL\_MARGIN,  
 battleField.width - 36, battleField.height - 36);

* + - Kiểm tra xem tọa độ dự doán bên trên có nằm trong vị trí vùng bắn thuận lợi không. Nếu không thì tính tốc độ viên đạn dựa vào khoảng cách giữa vị trí nhắm bắn ban đầu và thời gian bắn trúng . Sau đó thiết lập năng lượng đạn phù hợp và thoát khỏi vòng lặp. Ngược lại thì thiết lập vì trí ngắm bắn là điểm đã dự đoán

*if* (!fireField.contains(predictX, predictY)) {  
 bulletSpeed = shootAt.distance(myRobot) / deltaHitTime;  
 power = Utility.*clamp*((20 - bulletSpeed) / 3.0, 0.1, 3.0);  
 *break*;  
}

* + Bước 3: Thiết lập vị trí ngắm bắn mới dựa trên vị trí dự đoán và kích thước sân đấu để đảm bảo vị trí đó luôn hợp lệ

shootAt.setLocation(Utility.*clamp*(predictX, 34, getBattleFieldWidth() - 34), Utility.*clamp*(predictY, 34, getBattleFieldHeight() - 34));

* + Bước 4: Thiết lập điều kiện để bắn

*if* ((getGunHeat() == 0.0) && (getGunTurnRemaining() ==0.0) && (power > 0.0) && (myRobot.energy > 0.1))  
 setFire(power);

* + Bước 5: Quay radar:
    - Đưa tọa độ (shootAt.x – getX(), shootAt.y – getY()) về tọa độ cực để lấy góc.
    - Sau đó lấy /2 – shootAt.x – getX(), shootAt.y – getY()) để đảm bảo radar quay một góc không quá 90. Lấy kết quả trừ đi hướng hiện tại của nòng súng rồi chuẩn hóa sang 1 góc tương đối có phạm vi từ

setTurnGunRightRadians(Utils.*normalRelativeAngle*((  
(Math.PI / 2) - Math.*atan2*(shootAt.y - myRobot.getY(),shootAt.x - myRobot.getX())) - getGunHeadingRadians()));

### Phương thức cập nhật danh sách toạ độ (updateListLocation)

* Cập nhật danh sách các điểm có thể di chuyển của robot với một số lượng điểm cụ thể
  + Bước 1: xóa toàn bộ các điểm có trong danh sách
  + Bước 2: thiết lập phạm vi di chuyển tọa độ X cho robot là phần nguyên của 125 \* 1.5
  + Bước 3: cho vào vòng lặp với số lần là số phần tử của danh sách
    - Tạo 1 số random (randXMod) của robot trong phạm vi (-xRange, xRange)
    - Tính phạm vi di chuyển của tung độ phụ thuộc vào phạm vi di chuyển của hoành độ và yRange =
    - Tạo một số random của Y trong phạm vi (-yRange, -yRange)
    - Thiết lập tọa độ điểm mà robot có thể di chuyển với điều kiện luôn nằm trong sân đấu không bị đụng tường
    - x = giá trị ở giữa (tọa độ trục hoành hiện tại + randXMod, 75, battleFieldHeight - 75)
    - y = giá trị ở giữa (tọa độ trục tung hiện tại + randYMod, 75, battleFieldHeight – 75)
    - Thêm tọa độ vừa tìm được vào danh sách tọa độ có thể di chuyển

*public void* updateListLocations(*int* n) {  
 possibleLocations.clear();  
 *final int* radius = (*int*) (125 \* 1.5);  
 *for* (*int* i = 0; i < n; i++) {  
 *double* randXMod = Utility.*randomBetween*(-radius, radius); *double* yRange = Math.*sqrt*(radius \* radius - randXMod \* randXMod);  
 *double* randYMod = Utility.*randomBetween*(-yRange, yRange);  
 *double* y = Utility.*clamp*(myRb.y + randYMod, 75, battleField.height - 75);  
 *double* x = Utility.*clamp*(myRb.x + randXMod, 75, battleField.width - 75);  
 possibleLocations.add(*new* Point2D.Double(x, y));  
 }  
}

### Phương thức đánh giá toạ độ (evaluatePoint)

Dùng để đánh giá độ nguy hiểm của một điểm nhằm mục đích đưa ra quyết định xem robot có nên di chuyển tới điểm đó không.

Trước tiên ta phải tìm hiểu về kĩ thuật Anti – Gravity Movement[[2]](#_[11]__https://robowiki.net/wiki/Ant): là một phương thức di chuyển mức độ trung bình được sử dụng chính trong cận chiến. Giống như hầu hết các phương thức di chuyển cận chiến khác, nó thì tập trung và giữ cho con robot của ta xa có thể với các con robot khác. Nó liên quan tới việc sử dụng luật của trọng lực, nhưng ngược lại. Ví dụ khi robot của bạn đến tường trong một khoảng cách nhất định thì nó sẽ bị đẩy ra.

Công thức của trọng lực là

* Bước 1: Khởi tạo giá trị dự đoán độ nguy hiểm (rickValue) bằng 1 số random trong phạm vi (0,75, 2) \* trọng lực

*double* rickValue = Utility.*randomBetween*(0.75, 2) / p.distanceSq(myRobot);

* Bước 2: Nếu trong sân đấu chỉ còn 1 vs 1 thì có thể di chuyển ở trung tâm còn ngược lại thì cho di chuyển ra một vùng mà ít bị robot địch nhắm đến nhất (di chuyển ra 1 trong 4 góc của sân đấu) bằng cách lấy rickValue cộng thêm 1 giá trị bằng với (với khoảng cách của trọng lực được tính là khoảng cách giữa điểm được đánh giá đến điểm nằm ở trung tâm sân đấu)

rickValue += (6 \* (getOthers() - 1)) /   
 p.distanceSq(battleField.width /2, battleField.height/2);

* Bước 3: Tương tự bước 2 nhưng dùng để robot tránh va chạm 4 góc của sân đấu: rickValue sẽ cộng thêm 1 giá trị bằng với conerFactor (với conerFactor là 1 số trong tập hợp {0.25, 0.5, 1} dựa vào số địch còn lại trong sân thì conerFactor sẽ là một số cụ thể trong 3 số đó) \* trọng lực (với khoảng cách trong trọng lực là khoảng cách giữa điểm đang xét và tọa độ 4 góc của sân đấu)
  + Trường hợp 1: đối với góc dưới cùng bên trái có tọa độ là (0,0)
  + Trường hợp 2: đối với góc dưới cùng bên phải có tọa độ là (chiều rộng sân đấu, 0)
  + Trường hợp 3: Đối với góc trên cùng bên trái có tọa độ là (0, chiều cao sân đấu)
  + Trường hợp 4: Đối với góc trên cùng bên phải có tọa độ là (chiểu rộng sân đấu, chiều cao sân đấu)

*double* cornerFactor = getOthers() <= 5 ? getOthers() == 1 ? 0.25 : 0.5 : 1;  
rickValue += cornerFactor / p.distanceSq(0, 0);  
rickValue += cornerFactor / p.distanceSq(battleField.width, 0);  
rickValue += cornerFactor / p.distanceSq(0, battleField.height);  
rickValue += cornerFactor / p.distanceSq(battleField.width, battleField.height);

* Bước 4: Kiểm tra xem con robot mà ta đang ngắm có còn sống hay không nếu không thì qua bước 5 ngược lại thì thực hiện các hành động sau rồi qua bước 7

*if* (targetRobot.alive)

* + Lấy góc của con robot đang nhắm so với điểm mà ta đang xét (botAngle)
  + Duyệt danh sách các robot đối thủ, và tính độ rủi ro của điểm đang xét với từng con robot địch đang trong sân đấu
  + Độ rủi ro = độ rủi ro cũ + độ nguy hiểm của robot địch \* trọng lực \* độ nguy hiểm khi di chuyển chuyển vuông góc với robot địch \* độ nguy hiểm khi di chuyển gần robot địch (với độ nguy hiểm của robot địch = enemy.energy – me.energy)
  + Độ nguy hiểm khi di chuyển vuông góc với robot địch = 1
  + Độ nguy hiểm khi di chuyển gần robot địch = 1 +

*double* robotAngle = Utils.*normalRelativeAngle*(  
 Utility.*calcAngle*(p, targetRobot) - Utility.*calcAngle*(myRobot, p));  
*Iterator*<Robot> enIterator = enemyList.values().iterator();  
*while* (enIterator.hasNext()) {  
 Robot en = enIterator.next();  
 rickValue += (en.energy / myRobot.energy) \* (1 / p.distanceSq(en)) \*  
 (1.0 + ((1 - (Math.*abs*(Math.*sin*(robotAngle)))) + Math.*abs*(Math.*cos*(robotAngle))) / 2) \*  
 (1 + Math.*abs*(Math.*cos*(Utility.*calcAngle*(myRobot, p) - Utility.*calcAngle*(en, p))));  
}

* Bước 5: nếu không còn robot địch trong danh sách thì qua bước 6 ngươc lại làm các hành động sau rồi qua bước 7:
  + Lấy từng con robot ra rồi tính độ rủi ro
  + Độ rủi ro = độ rủi ro cũ + độ nguy hiểm của robot địch \* trọng lực \* độ nguy hiểm khi di chuyển gần robot địch

*else if* (enemyList.values().size() >= 1) {  
 *Iterator*<Robot> enIterator = enemyList.values().iterator();  
 *while* (enIterator.hasNext()) {  
 Robot en = enIterator.next();  
 rickValue += (en.energy / myRobot.energy) \* (1 / p.distanceSq(en)) \*  
 (1 + Math.*abs*(Math.*cos*(Utility.*calcAngle*(myRobot, p) - Utility.*calcAngle*(en, p))));  
 }  
}

* Bước 6: tính độ rủi ro = độ rủi ro cũ + độ nguy hiểm khi di chuyển đến gần điểm mà ta đang hướng đi tới
  + Với độ nguy hiểm khi di chuyển gần robot đang bị ta nhắm = 1 +

rickValue += (1 + Math.*abs*(Utility.*calcAngle*(myRobot, targetPoint) - getHeadingRadians()));

* Bước 7: trả về độ rủi ro của điểm đang xét *return* rickValue;

### Phương thức di chuyển (movement)

* Bước 1: Kiểm tra xem nếu khoảng cách từ điểm mà robot ta đang nhắm đi tới với vị trí của robot ta hiện tại < 15 hoặc là thời gian mà ta nhắm vào vị trí đó quá lâu (idleTime lớn hơn 25 tick) thì thực hiện những hành động sau, ngược lại thì sang bước 2:
  + Khởi gán lại giá trị cho idleTime = 0 nhằm chuyển qua mục tiêu mới
  + Cập nhật lại danh sách các điểm có thể di chuyển xung quanh robot
  + Duyệt danh sách các điểm đó và tìm ra điểm có độ rủi ro thấp nhất.
  + Cập nhật lại điểm mục tiêu đi đến robot của là là điểm vừa tìm được

idleTime = 0;  
 updateListLocations(AMOUNT\_PREDICTED\_POINTS);  
 Point2D.Double lowRiskP = *null*;  
 *double* lowestRisk = Double.MAX\_VALUE;  
 *for* (Point2D.Double p : possibleLocations) {  
 *double* currentRisk = evaluatePoint(p);  
 *if* (currentRisk <= lowestRisk || lowRiskP == *null*) {  
 lowestRisk = currentRisk;  
 lowRiskP = p;  
 }  
 }  
 targetPoint = lowRiskP;

* Bước 2: Tiếp tục đi đến điểm đang nhắm tới, thực hiện những hành động sau:
  + Tăng idleTime lên 1 đơn vị
  + Tính góc giữa robot của ta và điểm mà ta đang hướng đi tới (alpha)
  + Thiết lập hướng đang đi là 1
  + Kiểm tra nếu cos(alpha) < 0 thì ta đảo hướng di chuyển vì di chuyển lùi về sau và quay robot sẽ nhanh hơn di chuyển tiến lên phía trước
  + Thiết lập vận tốc cho robot của ta để số lượng quay của robot là ít nhất

(với )

* + Tiến hoặc lui 1 khoảng cách bằng với khoảng cách từ robot của ta tới vị trí điểm đang được hướng tới
  + Chuẩn hóa alpha (để alpha nằm trong phạm vi )
  + Quay radar qua phải 1 góc alpha vừa được chuẩn hóa

idleTime++;  
*double* angle = Utility.*calcAngle*(myRobot, targetPoint) - getHeadingRadians();  
*double* direction = 1;  
*if* (Math.*cos*(angle) < 0) {  
 angle += Math.PI;  
 direction \*= -1;  
}  
setMaxVelocity(10 - (4 \* Math.*abs*(getTurnRemainingRadians())));  
setAhead(myRobot.distance(targetPoint) \* direction);  
angle = Utils.*normalRelativeAngle*(angle);  
setTurnRightRadians(angle);

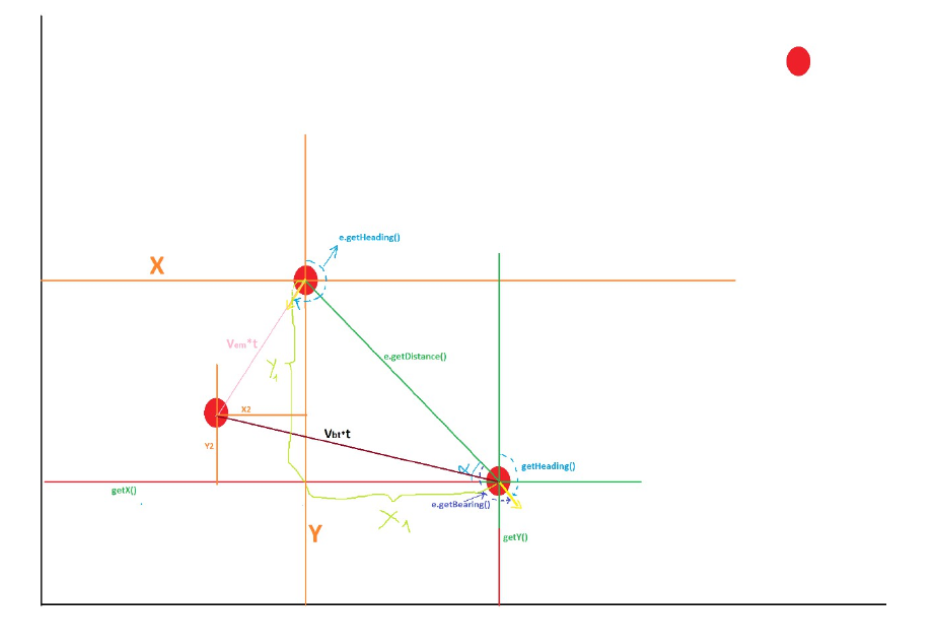
### Phương thức hoạt động sau khi bị tiêu diệt (onRobotDeath)

* + - * Kiểm tra nếu robot nào trong danh sách robot địch có tên giống với tên của robot bị tiêu diệt thì thiết lập alive của robot đó là false.
      * Nếu mà robot ta đang nhắm tới có tên giống với tên robot bị tiêu diệt thì thiết lập trạng thái robot ta đang nhắm tới là false

*public void* onRobotDeath(RobotDeathEvent event) {  
 *if* (enemyList.containsKey(event.getName()))  
 enemyList.get(event.getName()).alive = *false*;  
 *if* (event.getName().equals(targetRobot.name))  
 targetRobot.alive = *false*;  
}

### Phương thức khi quét robot khác (onScannedRobot)

* ***Nếu đang trong thể thức hỗn chiến***
  + - * Đầu tiên ta lấy thông tin con robot trong danh sách robot địch có tên giống với tên robot mà ta quét trúng. Nếu trong danh sách không có thì ta thêm nó vào trong danh sách.
      * Cập nhật các dữ liệu cần thiết cho con robot địch (bearing, heading, energy, position, shootAbleScore..)
      * Nếu trường hợp chỉ còn 1 robot địch thì ta sẽ khóa radar vào chính nó để bắn
      * Nếu như robot ta đang nhắm có trạng thái chết hoặc là có giá shootAbleScore > shootAbleScore của robot ta quét được thì sẽ chuyển mục tiêu nhắm thành robot vừa mới quét
      * Các phương pháp tính toán sử dụng:
  + Tính tọa độ robot đối phương



Theo hình ta có:

Với

Theo tính chất ta có

Vậy tọa độ robot địch ta tính được

* + Tính shootAbleTable: Dựa trên năng lượng và khoảng cách của robot địch vừa quét được tới robot của ta. Nếu năng lượng của con robot đó càng ít và khoảng cách càng gần với robot ta thì shootAbleScore càng nhỏ.

Robot en = enemyList.get(e.getName());  
if (en == null) {  
 en = new Robot();  
 enemyList.put(e.getName(), en);  
}  
en.absoluteBearingRadians = e.getBearingRadians();  
en.setLocation(new Point2D.Double(myRobot.x + e.getDistance() \* Math.sin(getHeadingRadians() + en.absoluteBearingRadians),  
 myRobot.y + e.getDistance() \* Math.cos(getHeadingRadians() + en.absoluteBearingRadians)));  
en.lastHeading = en.heading;  
en.name = e.getName();  
en.energy = e.getEnergy();  
en.alive = true;  
en.scanTime = getTime();  
en.velocity = e.getVelocity();  
en.heading = e.getHeadingRadians();  
en.shootAbleScore = en.energy < 25 ? (en.energy < 5 ?  
 (en.energy == 0 ? Double.MIN\_VALUE : en.distance(myRobot) \* 0.1) :  
 en.distance(myRobot) \* 0.75) : en.distance(myRobot);  
if (getOthers() == 1) {  
 // Nano Bot Lock - Very Simple  
 setTurnRadarLeftRadians(getRadarTurnRemainingRadians());  
}  
if (!targetBot.alive || en.shootAbleScore < targetBot.shootAbleScore)  
 targetBot = en;

* ***Nếu đang trong thể thức solo:***
  + - * Tạo 1 con robot mới tượng trưng cho robot địch và thiết lập các thông tin cho nó (bearing tuyệt đối, khoảng cách của robot đó tới robot của ta, vận tốc hiện tại của robot đó)

Robot enemy = new Robot();  
enemy.absoluteBearingRadians = getHeadingRadians() + e.getBearingRadians();  
enemy.dist = e.getDistance();  
enemy.velocity = e.getVelocity();

* + - * Kiểm tra xem con robot đó có di chuyển hay không (vận tốc khác 0). Nếu di chuyển thì thiết lập chiều di chuyển của nó bằng việc kiểm tra giá trị vận tốc vừa tính được. Nếu giá trị đó là số âm thì robot sẽ di chuyển lùi lại, ngược lại robot sẽ di chuyển tiến lên

if (enemy.velocity != 0) {  
 lateralDirection = Utility.sign(enemy.velocity \* Math.sin(e.getHeadingRadians() - enemy.absoluteBearingRadians));  
}

* + - * Tạo 1 làn sóng cho con robot của chúng ta và thiết lập các giá trị thích hợp cho nó (vị trí bắn của ta, tọa độ của robot địch,hướng di chuyển của địch)

Wave wave = new Wave(this);  
wave.gunLocation = new Point2D.Double(getX(), getY());  
Wave.targetLocation = Utility.project(wave.gunLocation, enemy.absoluteBearingRadians, enemy.dist);  
wave.lateralDirection = lateralDirection;

* + - * Thiết lập các giá trị segment (khoảng cách của địch, vận tốc hiện tại của địch, vận tốc, vận tốc lúc trước của địch)

wave.setSegmentations(enemy.dist, enemy.velocity, preEnemyVelocity);

* + - * Cập nhật lại vận tốc trước đó của robot địch là vận tốc hiện tại của nó sau đó thiết lập bearing của wave thành bearing tuyệt đối của robot địch. Tiếp đó ta thiết lập radar quay 1 góc sau khi đã được chuẩn hóa trong phạm vi . Khi đó góc quay của radar = giá trị bearing tuyệt đối robot địch - góc súng của robot ta + độ dời góc bearing tính được bởi wave

setTurnGunRightRadians(Utils.normalRelativeAngle(enemy.absoluteBearingRadians - getGunHeadingRadians() + wave.mostVisitedBearingOffset()));

* + - * Thiết lập năng lượng đạn bắn ra dựa vào năng lượng của ta và địch để hạn chế tiêu hao năng lượng một cách lãng phí và bắn

wave.bulletPower=Math.min(3, Math.min(this.getEnergy(), e.getEnergy()) / (double) 4);  
setFire(wave.bulletPower);

* + - * Giảm sát thương của đạn nếu năng lượng của ta còn khá ít và khoảng cách địch quá xa.

if (getEnergy() < 2 && e.getDistance() < 500)  
 wave.bulletPower = 0.1;  
else if (e.getDistance() >= 500)  
 wave.bulletPower = 1.1;  
setFire(wave.bulletPower);

* + - * Kiểm tra năng lượng còn lại của robot ta. Nếu vẫn còn đủ năng lượng thì tiếp tục thêm 1 sự kiện để tạo sóng cho đợt sau

if (getEnergy() >= wave.bulletPower)   
 addCustomEvent(wave);

* + - * Thực hiện quét robot địch khi di chuyển sau đó thiết lập lại góc quay radar tiếp tục để căn góc quét cho đợt sau

movement1VS1.onScannedRobot(e);  
setTurnRadarRightRadians(Utils.normalRelativeAngle(enemy.absoluteBearingRadians - getRadarHeadingRadians()) \* 2);

### Phương thức thực thi robot (run)

Khi bắt đầu trận đấu ta sẽ khởi gán các giá trị ban đầu:

* Kích thước sân đấu gán cho biến battleField.

battleField.height = getBattleFieldHeight();  
battleField.width = getBattleFieldWidth();

* Tọa độ, năng lượng ban đầu gán cho biến đại diện cho robot của ta.

myRobot.x = getX();  
myRobot.y = getY();  
myRobot.energy = getEnergy();

* Tọa độ vị trí hiện tại gán cho tọa độ điểm điểm rủi ro ít nhất.

targetPoint.x = myRobot.x;  
targetPoint.y = myRobot.y;

* Lấy AMOUNT\_PREDICTED\_POINTS = 150 các vị trí có thể di chuyển xung quanh robot của ta theo phạm vi hình tròn rồi đưa vào danh sách tọa độ các điểm để về sau tính toán độ rủi do từng vị trí đưa ra hướng di chuyển cho robot của mình.

updateListLocations(AMOUNT\_PREDICTED\_POINTS);

* Tạo 1 con robot tượng trưng cho con robot địch mà ta đang nhắm vào. Do vừa vào trận ta chưa nhắm vào đối tượng nào cụ thể nên để thuộc tính alive của nó là false.

targetRobot = new Robot();  
targetRobot.alive = false;

* Thiết lập quay súng không phụ thuộc vào hướng quay của robot để robot có thể di chuyển một hướng mà vẫn có thế bắn mục tiêu ở hướng khác.

setAdjustGunForRobotTurn(true);

* Thiết lập quay radar không phụ thuộc vào hướng quay của súng để robot có thể bắn mục tiêu ở một hướng mà vẫn có thế điểu chính radar quét theo hướng di chuyển của mục tiêu đó.

setAdjustRadarForGunTurn(true);

* Thiết lập radar quay phải vô tận để quét cho đến khi gặp mục tiêu cần bắn.

setTurnRadarRightRadians(Double.POSITIVE\_INFINITY);

* Nếu đang trong thể thức hỗn chiến: ở vòng lặp vô tận, qua mỗi lần lặp robot sẽ thực hiện các hành động sau:
  + Cập nhật lại các dữ liệu của robot ta (vị trí, hướng di chuyển vòng lặp trước, hướng di chuyển hiện tại, năng lượng, hướng của radar hiện tại).
  + Kiểm tra từng robot trong danh sách các robot địch, nếu như thời gian hiện tại - thời gian được quét trúng (scanTime) trước đó của nó > 25 thì ta biết được rằng thông tin của nó không được cập nhật sau 25 tick, ta sẽ gán cho trạng thái con robot đó là đã chết, tiếp đó ta kiểm tra con robot mà ta đang nhắm vào có phải là con robot đó không, nếu đúng thì ta cũng chuyển trạng thái cho con robot ta đang nhắm là đã chết .
  + Cho robot di chuyển.
  + Nếu nhắm 1 con robot mà trạng thái của nó là còn sống (alive = true) thì bắn.

while (true) {  
 myRobot.lastHeading = myRobot.heading;  
 myRobot.heading = getHeadingRadians();  
 myRobot.x = getX();  
 myRobot.y = getY();  
 myRobot.energy = getEnergy();  
 myRobot.gunHeadingRadians = getGunHeadingRadians();  
 Iterator<Robot> enIterator = enemyList.values().iterator();  
   
 while (enIterator.hasNext()) {  
 Robot r = enIterator.next();  
 if (getTime() - r.scanTime > 25) {  
 r.alive = false;  
 if (targetRobot.name != null && r.name.equals(targetRobot.name))  
 targetRobot.alive = false;  
 }  
 }  
 movement();  
 if (targetRobot.alive)  
 shooting();  
 execute();  
}

* Nếu đang trong thể thức solo:
  + Thiết lập hướng di chuyển vuông góc ban đầu là 1 (hướng tiến lên phía trước)
  + Thiết lập biến kiểm tra vận tốc lúc trước của robot địch là 0 (Ban đầu vào trận coi vận tốc trước đó của robot địch là 0)
  + Quay radar qua phải vô tận để quét robot địch xung quanh

lateralDirection = 1;  
preEnemyVelocity = 0;  
do {  
 turnRadarRightRadians(Double.POSITIVE\_INFINITY);  
} while (true);

### Phương thức đổi màu robot (changeColor)

* + - * Tạo 1 biến tĩnh sử dụng để đổi màu, setColor ngẫu nhiên trong khoảng từ 0 đến 255

static Random random = new Random();

private void changeColor() {  
 setColors(new Color(random.nextInt(255), random.nextInt(255), random.nextInt(255)),  
 new Color(random.nextInt(255), random.nextInt(255), random.nextInt(255)),  
 new Color(random.nextInt(255), random.nextInt(255), random.nextInt(255)),  
 new Color(random.nextInt(255), random.nextInt(255), random.nextInt(255)),  
 new Color(random.nextInt(255), random.nextInt(255), random.nextInt(255)));  
}

# TÀI LIỆU THAM KHẢO

##### [1] <https://robowiki.net/wiki/GuessFactor_Targeting_Tutorial>

##### [2] <https://robowiki.net/wiki/Wave_Surfing>

##### [3] <https://robowiki.net/wiki/Wave_Surfing/True_Surfing>

##### [4] <https://robowiki.net/wiki/Wave_Surfing/GoTo_Surfing>

##### [5] <https://robowiki.net/wiki/Movement_Profile>

##### [6] <https://robowiki.net/wiki/Rolling_Averages>

##### [7] <https://robowiki.net/wiki/Segmentation>

##### [8] <https://robowiki.net/wiki/Virtual_Bullet>

##### [9] <https://robowiki.net/wiki/Waves>

##### [10] <https://robowiki.net/wiki/Melee_Strategy#Evaluating_your_Melee_bot>

##### [11] <https://robowiki.net/wiki/Anti-Gravity_Movement>

##### [12] <https://robowiki.net/wiki/Corner_Movement>

##### [13] <https://robowiki.net/wiki/Pattern_Movement>

##### [14] <https://robowiki.net/wiki/Minimum_Risk_Movement>

##### [15] <https://robowiki.net/wiki/Head-On_Targeting>

##### [16] <https://robowiki.net/wiki/Linear_Targeting>

##### [17] <https://robowiki.net/wiki/Circular_Targeting>

##### [18] <https://robowiki.net/wiki/Pattern_Matching>

##### [19] <https://robowiki.net/wiki/GuessFactor_Targeting_Tutorial>

##### [20] <https://en.wikipedia.org/wiki/Cartesian_coordinate_system>

##### [21] <https://en.wikipedia.org/wiki/Polar_coordinate_system>

##### [22] <https://www.w3schools.com/java/java_hashmap.asp>