ĐẠI HỌC QUỐC GIA TP HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
Khoa Khoa Học & Kỹ Thuật Máy Tính

TRÍ TUỆ NHÂN TẠO
Bài tập lớn số 2

ROBOCODE

GVHD:
Gs. Cao Hoàng Trụ
Ths. Vương Bá Thịnh

NHÓM Feederz:
Nguyễn Kim Trung Hiếu 51201097
Đỗ Nguyễn Khánh Hoàng 51201200

Tp. HCM, 05/2015

Mục lục

Danh sách hình vẽ

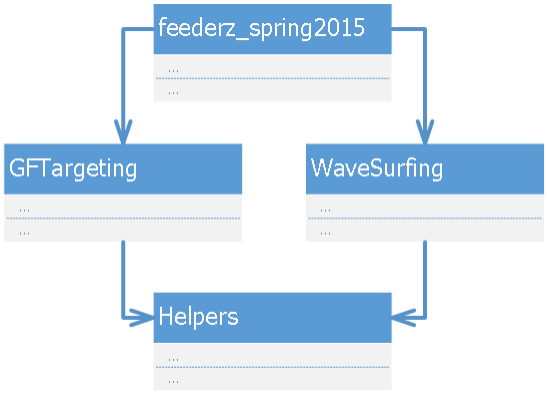
1 Robocodde

1.1 Giới thiệu

1.2 Luật chơi

2 Tổ chức Class

Chương trình được tổ chức rõ ràng với 04 class chính. Nhiệm vụ mỗi class được phân chia cụ thể và hợp lý.



Hình 1: Tổ chức class

- **feederz_spring2015**: class chính hiện thực robot. Ở đây chứa các hàm cơ bản mà hệ thống sẽ gọi trong suốt quá trình robot chạy
- **WaveSurfing**: class hiện thực kỹ thuật Wave Surfing, chịu trách nhiệm tính toán và điều khiển đường đi của robot, né đạn và né tường
- **GFTargeting**: class hiện thực kỹ thuật Guess Factor, chịu trách nhiệm tính toán và điều khiển súng của robot để nhắm chính xác mục tiêu
- **Helpers** class chứa các hàm bổ trợ được sử dụng bởi các class trên

3 Kỹ thuật Wall Smoothing

Theo luật của robocode, mỗi lần va chạm với tường robot cũng bị mất năng lượng giống như khi bị trúng đạn. Hơn nữa, nếu robot va chạm với tường và mắc kẹt ở một trong bốn góc của sân đấu thì khả năng bị tiêu diệt sớm lại càng cao hơn. Do vậy, để nâng cao khả năng sống còn của robot, việc đầu tiên cần nghĩ ngay tới đó là làm cho robot "né" được tường trong lúc chiến đấu, đừng để nó di chuyển vào những điểm chết hoặc những điểm quá gần tường.

Đầu tiên ta thiết lập một vùng an toàn cho robot. Trong suốt trận đấu, ta cố gắng điều khiển cho robot di chuyển không vượt qua giới hạn của vùng này. Cụ thể, kích thước vùng này được quy định bởi một hình chữ nhật *playingRectangle* như trong source code.

Listing 1: Thiết lập vùng an toàn

```
1 public static final int BATTLEFIELD_WIDTH = 8100;
  public static final int BATTLEFIELD_HEIGHT = 600;
3 static final int BOUNDARY_SIZE = 18;
```

```
public static Rectangle2D.Double playingRectangle = new Rectangle2D.Double(
    BOUNDARY_SIZE, BOUNDARY_SIZE,
    BATTLEFIELD_WIDTH - BOUNDARY_SIZE * 2,
    BATTLEFIELD_HEIGHT - BOUNDARY_SIZE * 2);
```

Kích thước cố định của sân đấu là 800×600. Vùng an toàn là hình chữ nhật nhỏ bên trong, cách biên của sân đấu một khoảng BOUNDARY_SIZE = 18.

Xử lý va chạm với tường được giải quyết bằng kỹ thuật Wall Smoothing. Giải thuật này cố gắng tìm ra một góc gọi là "an toàn", nghĩa là nếu robode tiến theo góc đó thì nó không thể va chạm với tường đồng thời cũng không di chuyển quá gần về phía đối thủ.

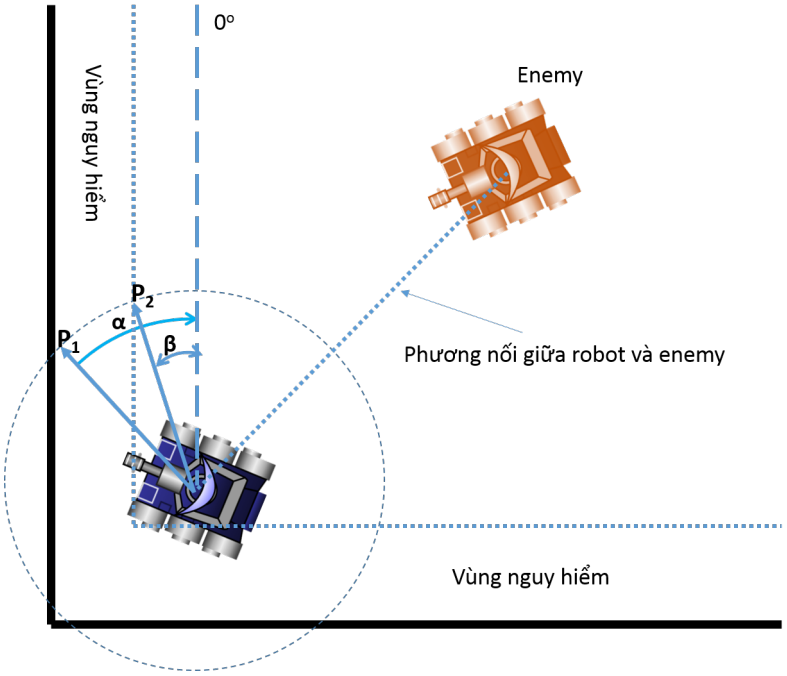
Listing 2: wallSmoothing function

```
public double wallSmoothing(Point2D.Double botLocation, double angle,int orientation) {
  Point2D.Double guesingPosition =
                  Helpers.getPositionFromAngleAndDistance(botLocation, angle, WALL_STICK);
  while (!playingRectangle.contains(guesingPosition)) {
    angle += orientation * 0.05;
    guesingPosition = Helpers.
      getPositionFromAngleAndDistance(botLocation, angle, WALL_STICK);
  }
  return angle;
}
```

Hàm wallSmoothing cần biết 03 thông tin để có thể xác định được góc đi an toàn tiếp là góc nào. Chúng là:

- Vị trí hiện tại của robot
- Góc đi vuông góc. Thật khó trình bày bằng lời góc này xác định như thế nào. Tuy nhiên nhiên nếu nhìn vào hình vẽ ta thấy, để đi vuông góc với phương nối giữa robot và enemy, robot phải đi theo hướng P_1. Và góc *angle* chính là góc tương đối giữa mũi tên P_1 với trục đứng chỉ 0^o, tức góc α. Vì góc này nằm bên trái trục 0^o nên nó sẽ có giá trị âm.
- Hướng di chuyển hiện tại. Nếu chọn enemy làm tâm thì robot có 02 hướng chính để di chuyển là theo chiều kim đồng hồ và ngược chiều kim đồng hồ tướng ứng với giá trị +1 và -1 của *orientation*. Như trong hình vẽ, robot đang di chuyển theo hướng thuận chiều kim đồng hồ nên giá trị này sẽ là +1.

Để hiểu rõ cách hoạt động của giải thuật này, ta sẽ xét trường hợp trong hình. Theo đó nếu đi theo góc α thì sau một đoạn đường WALL_STICK = 160, vị trí của robot là P_1 - tức nằm trong vùng nguy hiểm. Giải thuật sẽ cố gắng xoay mũi tên này vào sâu trong sân đấu để vị trí mới này nằm trong vùng an toàn, đồng thời góc này phải hướng ra xa enemy lớn nhất có thể. Và do đó, mũi tên này sẽ quay vào trong và dừng lại khi đạt góc β tương ứng với vị trí P_2. Cứ như vậy, giải thuật sẽ luôn hướng góc di chuyển của robot vào trong sân đấu và giữ khoảng cách an toàn đối với tường.



Hình 2: Kỹ thuật Wall Smoothing

4 Kỹ thuật Wave Surfing

4.1 Giới thiệu

Một đội chơi đến từ Bồ Đào Nha tên là ABC là những người đầu tiên mang kỹ thuật Wave Surfing vào sử dụng khi họ áp dụng để phát triển robot Shadow vào giữa năm 2004. Cho đến tháng 4 năm 2010, top 40 đội đứng đầu đều sử dụng những dạng biến thể của nó để phát triển robot cho mình. Mấu chốt của kỹ thuật này đó là việc xác định thời điểm đối phương bắn đạn để từ đó dự đoán mục tiêu của nó. Càng nhiều thông tin thu thập được sau mỗi phát bắn, khả năng dự đoán vùng nguy hiểm, tức vùng mà đối phương thường nhắm vào, càng chính xác hơn, để từ đó lựa chọn những đường đi phù hợp.

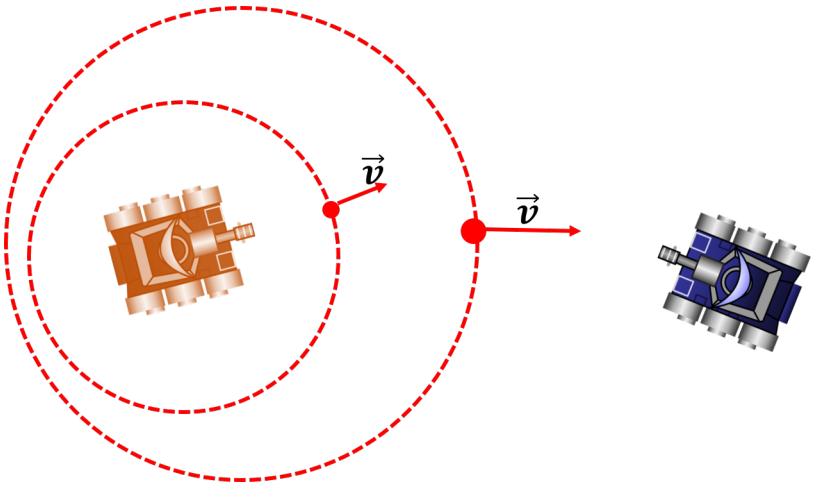
4.2 Trừu tượng hóa thông tin đạn bắn

Để tiện cho việc quản lý, những thông tin về đạn được thu thập, tổ chức và quản lý theo từng wave. EnemyWave là một đối tượng trừu tượng dùng để đóng gói thông tin một viên đạn bắn ra. Chúng bao gồm:

- fireLocation: vị trí mà ở đó viên đạn được bắn ra.
- fireTime: thời điểm bắn đạn
- bulletVelocity: vận tốc của viên đạn
- directAngle: góc bắn
- distanceTraveled: khoảng cách viên đạn đã đi được, tính từ fireLocation
- direction: hướng bắn

Listing 3: EnemyWave

```
class EnemyWave {
  Point2D.Double fireLocation;
  long fireTime;
  double bulletVelocity, directAngle, distanceTraveled;
  int direction;

  public EnemyWave() {}
}
```



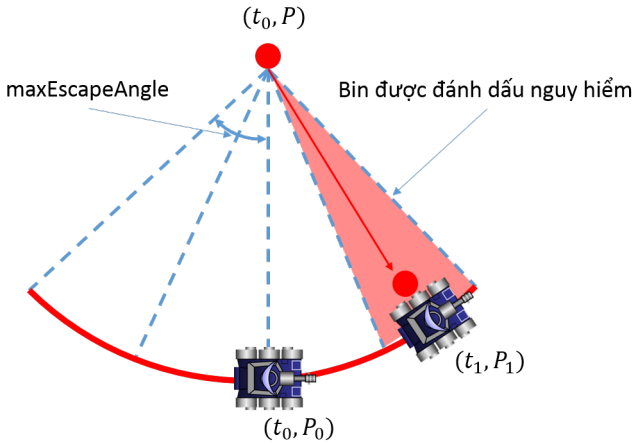
Hình 3: EnemyWave

4.3 Nhận biết thời điểm bắn đạn

API của robocode không cung cấp sự kiện nào để nhận biết việc bắn đạn. Tuy nhiên chúng ta có thể xác đinh được điều này thông qua sự sụt giảm năng lượng của đối phương. Như đã biết, mỗi lần bắn đạn robot sẽ bị mất một khoảng năng lượng và lượng thâm hụt $deltaEnergy$ này thỏa mãn $0 < deltaEnergy \leq 3.0$. Mỗi khi radar phát hiện được đối phương có dấu hiệu này thì chương trình khởi tạo một EnemyWave mới. Việc dự đoán này không phải lúc nào cũng chính xác 100% và các cải thiện sẽ được trình bày trong những phần sau.

4.4 Phân chia phạm vi nguy hiểm

Mỗi khi robot bị trúng đạn, thông tin của viên đạn được truy xuất để phục vụ cho việc phân hoạch vùng nguy hiểm. Việc phân hoạch này có thể dựa trên nhiều yếu tố bao gồm cả vận tốc đạn, khoảng cách đạn hoặc là góc bắn. Tuy nhiên vì chương trình khá đơn giản nên chúng em chỉ hiện thực phân hoạch dựa trên góc bắn.



Hình 4: Phân hoạch vùng nguy hiểm

Theo đó ứng với mỗi EnemyWave, hoặc nói theo cách khác là ứng với mỗi viên đạn được bắn ra, chúng ta sẽ xác định được một góc bắn như hình vẽ. Góc này có độ lớn bằng 2 lần góc maxEscapeAnlge[1] và được chia ra thành những phần bằng nhau được gọi là BIN. Trong chương trình sử dụng 47 BIN còn trong hình minh họa thì chia ra được 4 BIN. Một BIN được đánh dấu là nguy hiểm nếu như robot bị trúng đạn khi đang di chuyển trong BIN đó. Chẳng hạn như trong hình vẽ, tại thời điểm t_0 robot đang ở vị trí P_0 và nhận thấy đối thủ bắn ra viên đạn tại ví trí P. Cho đến thời điểm t_1 robot bị viên đạn đó đụng phải tại vị trí P_1 - thuộc BIN thứ 4. BIN này được tô đỏ trong hình vẽ và nghĩa là trong tương lai, nếu gặp trường hợp tương tự như vậy, robot sẽ hạn chế di chuyển vào BIN này. Tất cả các bị này được tổ chức và lưu trong biến statArray[].

Theo cách đánh dấu như vậy, càng về sau, sự phân hoạch vùng nguy hiểm này sẽ càng chính xác, và robot sẽ học được chiến lược bắn đạn của đối phương để tìm đường đi an toàn cho mình.

4.5 Kiểm tra mức nguy hiểm

Mỗi khi nhận biết một viên đạn đang bay tới gần, chương trình sẽ tiến hành kiểm tra xem mức độ nguy hiểm của hướng mình đang đi, nghĩa là xác định xem, nếu cứ tiếp tục đi như vậy thì khả năng viện đạn này va chạm robot cao đến mức nào. Sự đánh giá này dựa trên những thông tin thu thập được từ các BIN trong statArray. Hàm checkDanger sẽ làm công việc đó.

Listing 4: checkDanger

```
public double checkDanger(EnemyWave surfWave, int direction) {
  int index = calculateIndex(surfWave,
      predictPosition(surfWave, direction));
  return statArray[index];
}
```

Từ thông tin về viên đạn sắp tới surfWave và hướng đi chuyển hiện tại direction hàm dự đoán vị trí của robot bằng predictPosition và tính toán hệ số của BIN cần tìm. Sau đó trả về giá trị của BIN này trong statArray.

4.6 Chọn hướng đi an toàn

Việc lựa chọn này chỉ dựa trên thông tin của viên đạn gần nhất comingWave. Hàm tiến hành kiểm tra mức nguy hiểm của hai hướng đi trái phải (ngược chiều, cùng chiều kim đồng hồ). Sau khi lựa chọn được một hướng đi an

[1]Là góc lớn nhất mà robot có thể di chuyển được trong một đơn vị thời gian - một tick. Việc giới hạn này là hợp lý vì mọi góc nằm ngoài khoảng này là không cần xem xét vì robot không thể nào di chuyển tới đó được

toàn nhất, góc tìm được sẽ được xử lý bởi hàm wallSmoothing để tránh trường hợp đụng tường. Góc goAngle trả về là góc cuối cùng mà robot sẽ luôn định hướng đi theo trong suốt chương trình.

Listing 5: getPerfectAngleToGo

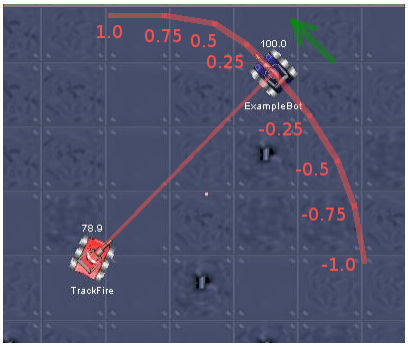
```
public double getPerfectAngleToGo() {
  EnemyWave comingWave = getClosestSurfableWave();
  if (comingWave == null) {
    return Double.POSITIVE_INFINITY;
  }
  double dangerLeft = checkDanger(comingWave, -1);
  double dangerRight = checkDanger(comingWave, 1);

  double goAngle = Helpers.getAbsoluteBearingAngle(
      comingWave.fireLocation, ourRobotPosition);
  if (dangerLeft < dangerRight) {
    goAngle = wallSmoothing(ourRobotPosition, goAngle - (Math.PI / 2),
        -1);
  } else {
    goAngle = wallSmoothing(ourRobotPosition, goAngle + (Math.PI / 2),
        1);
  }
  return goAngle;
}
```

4.7 Những biện pháp cải tiến

Giữ khoảng cách cố định
Chương trình không quan tâm đến việc giữ khoảng cách tương đối giữa robot và enemy. Nếu cải thiện được thì việc dự đoán đạn của robot sẽ chính xác hơn.
Theo dõi năng lượng của đối thủ chính xác hơn
Như đã trình bày ở phần trước, mỗi lần phát hiện ra đối thủ mất một mức năng lượng $0 < p \leq 3.0$ thì robot sẽ ghi nhận một viện đạn được bắn ra. Cách dự đoán này tuy đa phần chính xác những có một số trường hợp ngoại lệ, đặc biệt là khi gần kết thúc trận đấu - cả hai đều bắn đi những viên đạn có năng lượng nhỏ. Nói là không chính xác bởi vì không phải lúc nào đối thủ mất năng lượng cũng đều do bắn đạn. Đó có thể là do nó bị trúng đạn của ta bắn. Trong trường hợp đó, data thu thập được sẽ bị sai do những "viên đạn ảo" này, ảnh hưởng đến khả năng dự đoán.
Thay đổi chiến lược né đạn
Hiện tại ý tưởng né đạn vẫn là xác định xem viên đạn nào di chuyển đến gần robot nhất để né. Giải pháp này không hiệu quả bằng việc xem xét né viên đạn sẽ chạm robot trước thay vì viên gần nhất.

5 Kỹ thuật Guess Factor Targeting

5.1 Giới thiệu



Hình 5: Guess Factor Targeting

Guess Factor Targeting là 1 kỹ thuật ngắm bắn khá đơn giản nhưng cũng không kém phần hữu hiệu được sử dụng trong robocode. Về cơ bản, vào thời điểm bắn, ta xác định khoảng khu vực mà mục tiêu có thể di chuyển đến, chia chúng ra làm nhiều phần gọi là các BIN [2], sau đó dựa vào các thông tin thu thập được từ trước để xác định bắn vào BIN nào có khả năng cao nhất trúng mục tiêu. Guess Factor Targeting có một số nét tương đồng với WaveSurfing mặc dù 1 kỹ thuật dùng vào việc di chuyển, 1 kỹ thuật dùng vào việc ngắm bắn.

5.2 Trừu tượng hóa thông tin đạn bắn

Nếu như ở phần WaveSurfing phía trên, đối tượng EnemyWave được dùng để thu thập thông tin các viên đạn bắn ra của đối phương thì ở đây, chúng ta dùng đối tượng BulletWave để đóng gói thông tin về các viên đạn được bắn ra của chính robot của mình, bao gồm:

- targetLocation: vị trí của mục tiêu ở thời điểm viên đạn được bắn ra.
- bulletPower: mức năng lượng của viên đạn
- gunLocation: vị trí của súng (cụ thể là vị trí của robot) lúc bắn đạn.
- bearing: góc lệch giữa robot và mục tiêu
- lateralDirection: hướng di chuyển của mục tiêu (theo chiều vuông góc với robot)
- distanceTraveled: khoảng cách mà viên đạn đã di chuyển (cập nhật theo từng tick)
- robot: robot
- statBuffers: chứa kết quả thống kê của các BIN

[2]GuessFactor 0.0 (BIN ở giữa) trỏ trực tiếp tới hướng mục tiêu, GuessFactor 1.0 (BIN đầu tiên) trỏ tới vị trí xa nhất mà mục tiêu có thể tới nếu giữ nguyên hướng di chuyển hiện tại và GuessFactor -1.0 (BIN cuối cùng) trỏ tới vị trí xa nhất mà mục tiêu có thể tới nếu đổi hướng di chuyển

5.3 Phân hoạch phạm vi bắn

Như đã nói ở trên, ta chia phạm vi mà mục tiêu có thể đến thành nhiều BIN và chọn 1 BIN mà ta thấy có xác suất bắn trúng cao nhất để ngắm bắn. Mỗi khi bắn trúng mục tiêu, thông tin của viên đạn được truy xuất để phục vụ cho việc phân hoạch vùng bắn. Việc phân hoạch này có thể dựa trên nhiều yếu tố, trong bài tập lớn này, nhóm sử dụng các yếu tố distance (khoảng cách đến mục tiêu), velocity (vận tốc hiện tại của mục tiêu) và lastVelocity (vận tốc của mục tiêu ngay trước thời điểm hiện tại) để phân hoạch.

Listing 6: Phân hoạch

```
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];
}
```

Cụ thể, phạm vi mà mục tiêu có thể di chuyển đến được xác định dựa vào maximum escape angle (khác với maxEscapeAngle ở phần WaveSurfing, ở đây chỉ khoảng tối đa mà mục tiêu có thể tới được tính từ lúc bắn đến lúc viên đạn bay tới đích, giá trị này phụ thuộc vào nhiều yếu tố, để cho đơn giản, trong bài tập lớn này nhóm gán cho nó giá trị cụ thể là 0.8). Phạm vi này được chia thành 25 BIN. mỗi lần viên đạn tới đúng mục tiêu, dựa vào thông tin viên đạn bắn ra, ta sẽ truy xuất vào statsBuffer tới BIN tương ứng và tăng giá trị của nó.

Listing 7: test

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

Như vậy, trong tương lai, nếu gặp hoàn cảnh tương tự (về khoảng cách tới mục tiêu, vận tốc...) thì bắn vào BIN này có khả năng trúng mục tiêu cao hơn các BIN khác.

Theo cách đánh dấu như vậy, càng về sau, sự phân hoạch này sẽ càng chính xác, và robot sẽ học được chiến lược di chuyển của đối phương để tìm ra góc bắn thích hợp nhất.

5.4 Chọn góc bắn phù hợp và bắn

Dựa vào các kết quả thống kê thu thập được, tại mỗi thời điểm chuẩn bị bắn, ta sẽ truy xuất vào statsBuffer và chọn ra BIN tốt nhất để bắn

Listing 8: mostVisitedBin

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

Dựa vào mostVisited BIN, ta có thể tính được Bearing offset phù hợp

Listing 9: mostVisitedBearingOffset

```
double mostVisitedBearingOffset() {
  return (lateralDirection * BIN_WIDTH) * (mostVisitedBin() - MIDDLE_BIN);
}
```

Sau đó, ta tính toán bulletPower phù hợp và ngắm bắn. bulletPower được tính dựa trên khoảng cách tới mục tiêu, energy còn lại của robot.

Listing 10: tính bulletPower

```
public double calcBulletPower(ScannedRobotEvent e) {
  double bulletPower = 0;
  bulletPower = e.getDistance() > 150 ? 1.9 : 3;
  bulletPower = Math.min(bulletPower, (e.getEnergy() + .1) / 4);
  if (bulletPower * 6 >= robot.getEnergy()) bulletPower = robot.getEnergy() / 6;
  if (bulletPower >= robot.getEnergy() - .1) bulletPower = robot.getEnergy() - .1;
  bulletPower = Math.max(Rules.MIN_BULLET_POWER,
        Math.min(Rules.MAX_BULLET_POWER, bulletPower));
  if (robot.getEnergy() < 10.0) bulletPower = 0.1;
  return bulletPower;
}
```

Listing 11: ngắm bắn

```
  robot.setTurnGunRightRadians(Utils.normalRelativeAngle(
                    enemyAbsoluteBearing -
                    robot.getGunHeadingRadians() +
                    wave.mostVisitedBearingOffset()
                  ));
  robot.setFire(wave.bulletPower);
```

6 Javadoc

6.1 Class feederz_spring2014

6.1.1 Thông tin các thuộc tính

colors	Color	Chứa các màu sắc của robot (Gồm 7 màu đỏ, da cam, vàng, lục, lam, chàm, tím)
colorNum	int	index màu sắc hiện tại của robot
waveSurfing	WaveSurfing	Đối tượng quản lý việc di chuyển
gunController	GFTargeting	Đối tượng quản lý việc điều khiển súng

6.1.2 Thông tin các hàm

---

initializeRobot

Signature public void initializeRobot()

Description Khởi tạo các giá trị ban đầu cho robot

---

changeAllColors

Signature public void changeAllColors()

Description Thay đổi màu sắc của robot (được gọi mỗi khi bị trúng đạn)

---

run
source: http://robocode.sourceforge.net/docs/robocode/

---

controllRadar

Signature public void controllRadar(ScannedRobotEvent e)

Description phụ trách việc điều khiển radar luôn hướng vào đối thủ

Parameter(s) e đối tượng thuộc lớp ScannedRobotEvent

---

controllRobot

Signature public void controllRobot()

Description phục trách việc điều khiển súng

---

onScannedRobot
source: http://robocode.sourceforge.net/docs/robocode/

---

onHitByBullet
source: http://robocode.sourceforge.net/docs/robocode/

---

onBulletHitBullet
source: http://robocode.sourceforge.net/docs/robocode/

---

onBulletHit
source: http://robocode.sourceforge.net/docs/robocode/

6.2 Class WaveSurfing

6.2.1 Thông tin các thuộc tính

ourRobot	AdvancedRobot	Đối tượng robot được truyền vào để điều khiển
ourRobotPosition	Point2D	Vị trí hiện tại của robot
enemyPosition	Point2D	Vị trí hiện tại của đối thủ
enemyWaves	ArrayList<EnemyWave>	List các đối tượng EnemyWave chứa thông tin về các viên đạn của đối thủ bắn ra
directionArray	ArrayList<Integer>	List chứa các hướng di chuyển của robot theo thời gian
BINS	int	Số lượng BIN tối đa được chia ra
statArray	double[]	Dãy chứa các giá trị của các BIN
absBearingsArray	ArrayList<Double>	Chứa các góc ngắm tuyệt đối của đối thủ (radian)
BULLET_FIRING_TIME_DELTA	int	Gia số về thời gian bắn đạn
WALL_STICK	int	Độ đai của đường đi dự đoán, sử dụng trong kỹ thuật Wall Smoothing
ROBOT_SIZE	double	Kích thước giả định của robot (có thể khác với kích thước vật lý được đặc tả)
BATTLEFIELD_WIDTH	int	Kích thước chiều ngang mặc định của sân đấu
BATTLEFIELD_HEIGHT	int	Kích thước chiều cao mặc định của sân đấu
BOUNDARY_SIZE	int	Khoảng cách nguy định vùng nguy hiểm đụng tường
playingRectangle	Rectangle2D	Hình chữ nhật nguy định vùng di chuyển an toàn
MAX_PREDICTION_TICK_NUMBER	int	Quy định thời gian tối đa cho việc dự đoán vị trí
hitTime	double	Lưu giữ thời điểm gần nhất robot bị trúng đạn

6.2.2 Thông tin các hàm

WaveSurfing

Signature public WaveSurfing(AdvancedRobot robot)

Description Khởi tạo đối tượng WaveSurfing

Parameters **robot** - đối tượng robot cần gán điều khiển di chuyển

---

getPerfectAngleToGo

Signature public double getPerfectAngleToGo()

Description Chọn góc đi phù hợp nhất, có xét đến cả sự đụng tường và né đạn

Return(s) Góc đi (radian)

---

updateData

Signature public void updateData(ScannedRobotEvent e)

Description Cập nhật thông tin cần thiết cho việc dự đoán đạn bắn như vị trí hiện tại của robot, hướng đi hiện tại, tạo một wave mới nếu phát hiện được kẻ thù vừa mới nổ súng,... Hàm được gọi gián tiếp thông qua hàm onScannedEvent() ở lớp feederz_spring2015

Parameter(s) **e** Đối tượng thuộc lớp ScannedRobotEvent chứa các thông tin gửi về từ radar

---

onBulletHit

Signature onBulletHit(BulletHitEvent e)

Description Ghi nhận thời điểm đạn của robot va chạm trúng đối thủ, phục vụ cho việc xác định chính xác thời điểm đối phương nổ súng. Hàm được gọi gián tiếp thông qua hàm cùng tên ở lớp feederz_spring2015

Parameter(s) **e** Đối tượng thuộc lớp BulletHitEvent

---

onHitByBullet

Signature public void onHitByBullet(HitByBulletEvent e)

Description Lấy thông tin từ mỗi viên đạn bị trúng để cập nhật cơ sở dữ liệu (giá trị các BIN) trong statArray. Hàm được gọi gián tiếp thông qua hàm cùng tên ở lớp feederz_spring2015

Parameter(s) **e** Đối tượng thuộc lớp HitByBulletEvent

---

getClosestSurfableWave

Signature public EnemyWave getClosestSurfableWave()

Description Tìm kiếm trong dãy các wave hiện tại wave có khoảng cách gần với robot nhất

Return(s) Đối tượng EnemyWave chứa thông tin về viên đạn gần nhất

---

checkDanger

Signature public double checkDanger(EnemyWave surfWave, int direction)

Description Kiểm tra mức nguy hiểm của một wave đối với robot

Parameter(s) **surfWave** wave cần xem xét

direction hướng di chuyển hiện tại của robot

Return(s) Giá trị chứa trong BIN tương ứng biểu thị mức độ nguy hiểm

---

wallSmoothing

Signature public double wallSmoothing(Point2D.Double botLocation, double angle,int orientation)

Description Tính toán góc đi cần thiết để tránh đụng phải tường

Parameter(s) **botLocation** Ví trị hiện tại của robot

angle Góc cần điều chỉnh (thường là góc vuông góc với phương nối giữa robot và kẻ thủ)

orientation Hướng di chuyển hiện tại +1 hoặc -1

Return(s) Giá trị góc đi (radian)

---

predictPosition

Signature public Point2D.Double predictPosition(EnemyWave surfWave, int direction)

Description Dự đoán vị trí tương lai của robot sau khi viên đạn được xét bay qua mất hoặc sau một khoảng thời gian định trước với hướng đi hiện tại

Parameter(s) **surfWave** EnemyWave chứa thông tin viện đạn đang được xét

direction Hướng đi chuyển hiện tại đang được xét

Return(s) Vị trí tương dự đoán

---

calculateIndex

Signature public static int calculateIndex(EnemyWave ew, Point2D.Double hittingPosition)

Description Tính toán index của BIN cần được cập nhật giá trị từ vị trí bị trúng đạn

Parameter(s) **ew** EnemyWave chứa thông tin của viên đạn mà robot bị trúng

hittingPosition vị trí robot bị trúng đạn

Return(s) giá trị index của BIN

---

updateWaves

Signature public void updateWaves()

Description Cập nhật thông tin cho các wave đã được lưu trữ, xóa bỏ những wave không cần thiết

---

updateStatArray

Signature public void updateStatArray(EnemyWave ew, Point2D.Double targetLocation)

Description Cập nhật các giá trị của các BIN trong statArray

Parameter(s) **ew** EnemyWave chứa thông tin viên đạn robot vừa bị bắn trúng

targetLocation Vị trí hiện tại của robot

6.3 Class GFTargeting

6.3.1 Thông tin các thuộc tính

robot	AdvancedRobot	Đối tượng robot được truyền vào để điều khiển
lateralDirection	Double	Hướng di chuyển của mục tiêu (theo chiều vuông góc với robot)
lastEnemyVelocity	Double	Tốc độ cuối cùng ghi nhận được trước thời điểm hiện tại của mục tiêu

6.3.2 Thông tin các hàm

GFTargeting

Signature public GFTargeting(AdvancedRobot robot)

Description Khởi tạo đối tượng GFTargeting

Parameters **robot** - đối tượng robot cần gán điều khiển ngắm bắn

---

updateData

Signature public void updateData(ScannedRobotEvent e)

Description Cập nhật các thông tin cần thiết đồng thời thực hiện việc điều khiển ngắm, bắn

---

calcBulletPower

Signature public double calcBulletPower(ScannedRobotEvent e)

Description Tính bulletPower thích hợp để bắn

6.4 Class BulletWave (Hỗ trợ cho class GFTargeting)

6.4.1 Thông tin các thuộc tính

targetLocation	Point2D	Vị trí của mục tiêu lúc bắn
bulletPower	Double	Độ mạnh của đạn
gunLocation	Point2D	Vị trí của súng (hay chính là vị trí của robot) lúc bắn
bearing	Double	Góc lệch giữa robot và mục tiêu
lateralDirection	Double	Như ở class GFTargeting
robot	AdvancedRobot	Đối tượng robot được truyền vào để điều khiển
distanceTraveled	Double	Khoảng cách mà viên đạn đã đi
buffer	Int[]	buffer phụ trỏ tới vùng phân hoạch hiện tại
MAX_DISTANCE	Double	Khoảng cách tối đa của vùng phân hoạch
DISTANCE_INDEXES	Double	Số phân đoạn phân hoạch theo khoảng cách
VELOCITY_INDEXES	Double	Số phân đoạn phân hoạch theo vận tốc
BINS	Double	Tổng số BIN
MIDDLE_BIN	Double	Index của BIN chính giữa
MAX_ESCAPE_ANGLE	Double	Khoảng góc lệch tối đa mà mục tiêu có thể đi tới
BIN_WIDTH	Double	Độ rộng mỗi BIN
statBuffers	int[][][][]	Buffer chính chứa các thông tin phân hoạch

6.4.2 Thông tin các hàm

BulletWave

Signature public BulletWave(AdvancedRobot _robot)

Description Khởi tạo đối tượng BulletWave

Parameters **_robot** - đối tượng robot cần gán điều khiển

---

setSegmentations

Signature void setSegmentations(double distance, double velocity, double lastVelocity)

Description Xác định vùng phaa hoạch (theo khoảng cách, vận tốc và vận tốc cuối)

distance - khoảng cách tới mục tiêu

velocity - Vận tốc hiện tại của mục tiêu

lastVelocity - Vận tốc cuối cùng trước thời điểm hiện tại của mục tiêu

---

test

Signature public boolean test()

Description Cập nhật khoảng cách, kiểm tra xem viên đạn đã tới mục tiêu hay chưa

---

updateDistance

Signature public boolean updateDistance()

Description Cập nhật khoảng cách đã di chuyển của viên đạn

---

hasArrived

Signature public boolean hasArrived()

Description Kiểm tra xem viên đạn đã tới mục tiêu hay chưa

---

currentBin

Signature private int currentBin()

Description Xác định BIN hiện tại

---

mostVisitedBin

Signature private int mostVisitedBin()

Description Xác định BIN bắn trúng nhiều nhất

---

mostVisitedBearingOffset

Signature mostVisitedBearingOffset()

Description Tính góc lệch dựa vào mostVisitedBin

6.5 Class Helpers

6.5.1 Thông tin các hàm

---

getAbsoluteBearingAngle

Signature public static double getAbsoluteBearingAngle(Point2D.Double from, Point2D.Double to)

Description Tính góc ngắm tyệt đối từ điểm này đến điểm khác

Parameter(s) **from** điểm đầu
to điểm cuối

Return(s) góc ngắm (radian)

---

getSuitableValueInRange

Signature public static double getSuitableValueInRange(double min, double value, double max)

Description Tìm giá trị phù hợp nằm trong khoảng cho trước, nếu giá trị tuyền vào thuộc khoảng này, trả về giá trị đó, ngược lại trả về một trong 2 đầu mút của khoảng.

Parameter(s) **min** cận dưới của khoảng

value giá trị cần xem xét

max cận trên của khoảng

Return(s) giá trị phù hợp

---

maxEscapeAngle

Signature public static double maxEscapeAngle(double velocity)

Description Tìm ra góc tối đa mà robot có thể di chuyển được trong một đơn vị thời gian với tốc độ cho trước

Parameter(s) **velocity** tốc độ hiện tại của robot

Return(s) góc (radian)

---

getPositionFromAngleAndDistance

Signature public static Point2D.Double getPositionFromAngleAndDistance(Point2D.Double from, double angle, double length)

Description Tính toán vị trí từ điểm hiện tại khi biết khoảng góc và khoảng cách của điểm đó so với điểm hiện tại.

Parameter(s) **from** điểm hiện tại

angle góc của điểm cần tính so với điểm hiện tại

length khoảng cách từ điểm cần tính so với điểm hiện tại

Return(s) điểm cần tính

---

project

Signature static Point2D project(Point2D sourceLocation, double angle, double length)

Description Xác định vị trí của mục tiêu trong hệ tọa độ Oxy

Parameter(s) **sourceLocation** vị trí của robot

angle góc lệch giữa robot và mục tiêu

length khoảng cách giữa robot và mục tiêu

Return(s) vị trí của mục tiêu

7 Tham khảo

[1] http://www.ibm.com/developerworks/library/j-robotips/index.html

[2] http://robowiki.net/wiki/Wave_Surfing_Tutorial

[3] https://ceasarjames.wordpress.com/2011/03/20/wave-surfing-explained

[4] http://robowiki.net/wiki/GuessFactor_Targeting_Tutorial