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

This page intentionally left blank

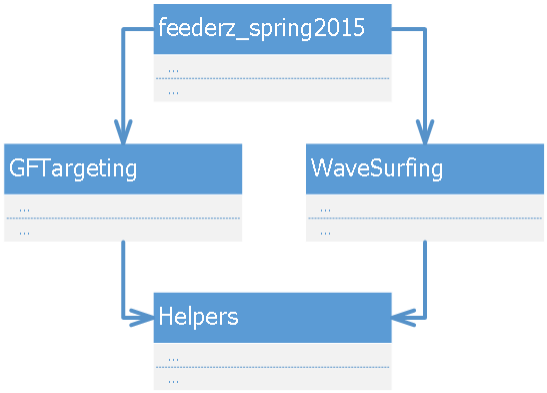
1 Robocodde

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.

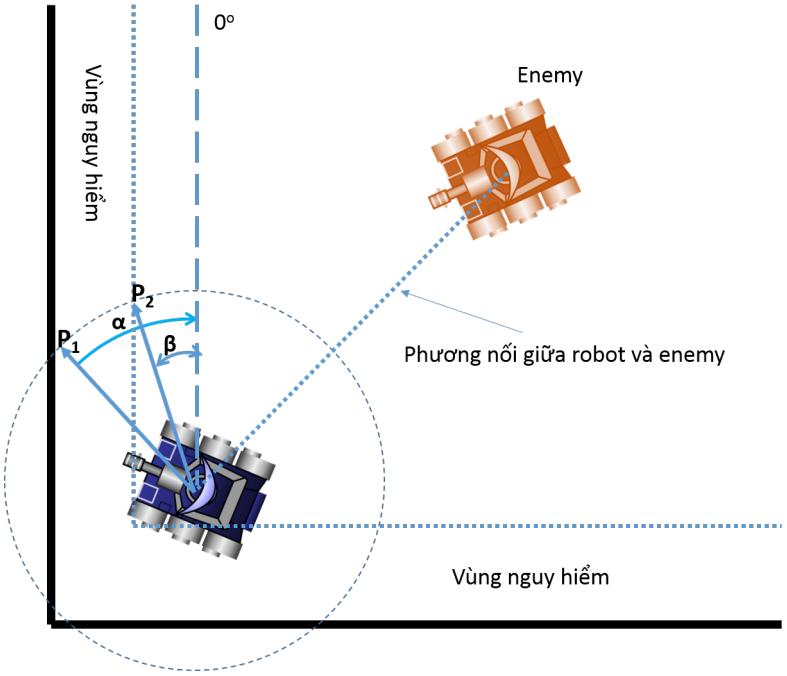
Listing 2: wallSmoothing function

```
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

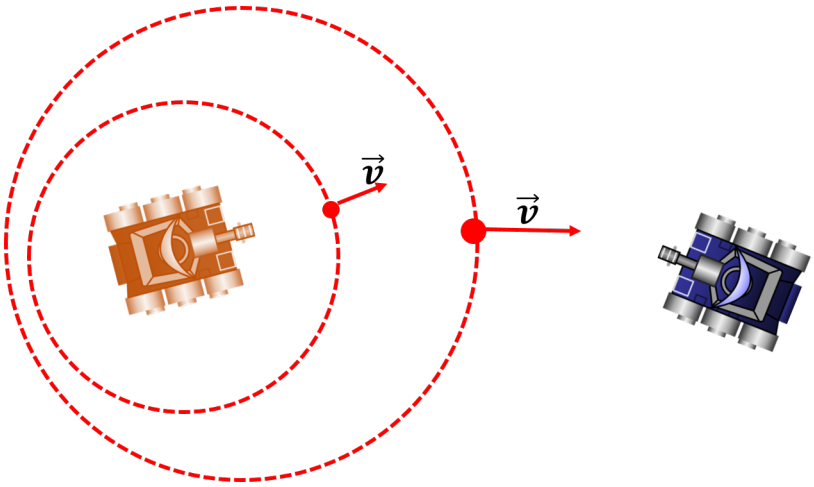
Để tiện cho việc quản lý, những thông tin về đạn được thu thập, tổ chức và quản lý theo từng wave. EnemyWave là một đối tượng trừu tượng dùng để đóng gói thông tin một viên đạn bắn ra. Chúng bao gồm:

- 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() {}
}
```



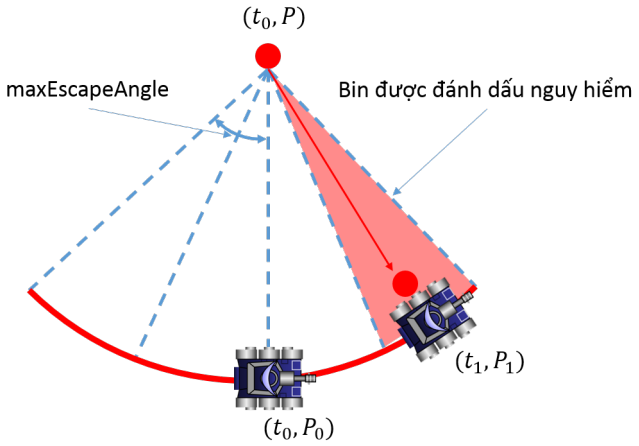
Hình 3: EnemyWave

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 Guess Factor