송주성 | vered@hanmail.net
서울에서 병역특례를 마치고 부산대학교 전자계산학과에 복학해서 다시 학생신분으로 행복한 나날을 보내고 있다고, 다시 뭔가 재미있을 만한 것을 찾아 헤메고 있는 중인데, 지금은 Lisp, Prolog, Acla, Small Talk등의 각종 프로그래밍 언어들을 혼자틀틀이 배우는 데 푹 빠져있다.

로보코드 우승 로봇, Fermat 소스 분석

어떤 형태의 프로그램을 접하든 다른 사람들의 소스를 분석해 봄으로써 얻을 수 있는 것들이 많다는 것은 굳이 다시 말하지 않아도 다들 잘 알 것이라 생각합니다. 특히 로보코드는 인터페이스를 중요시 하는 프로그램이 아니라 알고리즘을 중요시 하는 프로그램이라 더욱 그렇습니다. 다른 사람들은 어떤 생각들을 어떻게 코딩했는지, 유명한 문서에 나온 내용들은 어떻게 코드로 구현했는지 자주 읽다보면 자기의 생각들도 차츰 구현할 수 있을 것입니다. 그런 의미에서 이번에는 작년 미국 로보코드 대회에서 중급 부문에서 우승을 차지한 Fermat라는 로봇을 분석해 보겠습니다.

작년 미국의 로보코드 대회에서 고급 부문 우승을 차지한 Yngwie라는 로봇의 성능은 이미 로보코드에 관심이 있는 사람들이라면 한 번쯤 봤을 것입니다. 물론 지금은 이 로봇보다 우수한 성능을 가진 로봇들이 많지만, 소스 코드를 쉽게 구할 수 있는 로봇 중에서는 그래도 아주 우수한 로봇에 속합니다. 하지만 Yngwie와 그 외의 우승 로봇들을 지켜보다 보면 조금 재미있는 점을 발견할 수 있습니다. Yngwie가 일대일 대전에서는 우수한 성능을 보이지만, 여러 로봇들이 한꺼번에 나와서 전투할 경우에는 그리 좋은 성능을 보이지 않는다는 것입니다. 오히려 이 경우에는 Fermat라는 로봇이 평균적으로 조금 더 나은 성적을 보입니다. 물론 Yngwie와 일대일로 붙어도 그리 나쁜 성적을 보이지 않습니다. 따라서 이번에는 Fermat라는 로봇을 한번 분석해 보겠습니다.

Fermat의 기본 정보

Fermat는 로보코드 우승자 인터뷰 기사가 나와 있는 웹 페이지에서 소스를 구할 수 있습니다. 중급 부문에서 우승 한 이후 많은 개조를 거쳐 지금은 버전 2.0까지 나와 있습니다. Fermat 버전 2.0의 경우 Yngwie보다 훨씬 월등한 성능을 보입니다. 그러나 아쉽게도 2.0 버전은 소스 코드를 구할 수가 없는 상황이라 작년 대회에서 사용된 버전의 소스를 분석하도록 하겠습니다. 작년 버전이더라도 그리 나쁜 성능은 아니고 배울 점도 많기 때문에 충분히 분석해 볼 만한 가치가 있습니다. 본격적인 설명으로 들어가기 전에 이 로봇의 이름에 대해 간단히 소개하겠습니다. Fermat는 '페르마'라고 읽으며, 우승자 인터뷰에서도 나오듯 유명한 수학자의 이름을 딴 것입니다. '페르마의 마지막 정리'로 유명한 수학자이지요. Fermat는 다음과 같이 8개의 자바 소스 코드로 구성되어 있습니다.

- **AdvancedBullet.java** : Fermat가 발사한 총알에 대한 정보를 저장함
- **DataWriter.java** : Statistic(메모리 상에)에 저장한 정보를 파일로 저장함
- **Fermat.java** : Fermat의 메인 코드, 각종 기본적인 동작들을 처리함
- **FiringStrategy.java** : 총알 발사를 위한 전략을 선택함
- **OtherBot.java** : 다른 로봇들의 정보를 가지고 있음
- **PatternAnalyzer.java** : 적 로봇의 움직임 등을 기초로 어떤 패턴을 보이는지 분석함
- **RobotBody.java** : Fermat 로봇의 몸체를 움직이기 위한 알고리즘들이 들어 있음
- **Statistic.java** : 몇 개의 총알을 쏴서 몇 발을 맞췄는지 기록함(명중률)

앞에서 설명한 것과 같이 파일은 8개로 구성되어 있지만, RobotBody.java에 DodgeBullet이라는 클래스가 하나 더 들어가 있으므로 총 클래스는 9개입니다. DodgeBullet은 연재에서 이미 소개했듯이 총알을 피하는 알고리즘입니다. Fermat에서는 이 클래스가 단순히 좌표 계산을 하는 데만 쓰이고 실제 행동은 RobotBody 클래스에서 행하기 때문에 하나의 파일에 넣은 듯 합니다. 어쨌든 이 정보들을 기초로 Fermat를 더 세부적으로 분석해 보겠습니다.

참고로 Yngwie나 Fermat의 경우 웹 페이지에서 다운받으면 zip 파일로 되어 있습니다. zip 파일은 또 소스 코드만 있기 때문에 로보코드 프로그램에서 바로 불러(import) 쓸 수가 없습니다. 그래서 이 로봇들을 다운받고도 전투에 내보내지 못하는 분들이 꽤 있습니다. 이 로봇들을 불러오려면 일단 컴파일을 해야 합니다. 우선 다운받은 파일의 압축을 풀어야 하는데, Yngwie의 경우에는 emp라는 폴더를 만들어 풀고, Fermat의 경우에는 ak라는 폴더를 만든 다음 그 안에 압축을 풀어줍니다. 이것은 소스 코드 안에서 패키지 설정을 그렇게 하고 있기 때문인데 실제로 Fermat의 경우에는 소스 코드 맨 첫 줄에 다음과 같은 코드를 볼 수 있습니다.

```
package ak;
```

이후에도 로봇을 다운받으면 이 패키지로 설정된 것과 똑같은 이름의 폴더를 만들어서 그 아래에 소스 파일들을 두고 컴파일 한다는 것을 기억해 두세요. 일단 소스 파일들의 압축을 풀었다면 로보코드 프로그램에 있는 에디터를 실행시킵니다(Robot Editor). 이 에디터에서 각 로봇들의 메인 소스 파일 하나를 불러오세요. 즉, Yngwie의 경우에는 Yngwie.java를, Fermat의 경우에는 Fermat.java를 불러오면 됩니다. 그리고 컴파일을 하면 됩니다.

컴파일이 완료되면 로보코드 메인 프로그램에서 「Robot | Import downloaded robot」 메뉴를 선택해서 로봇을 선택합니다. 그리고 전투를 위한 셋팅 창을 열면 로봇들이 목록에 올라와 있는 것을 볼 수 있습니다.

Fermat의 메인 클래스

앞서 말했듯 Fermat의 메인 클래스는 Fermat.java입니다. 메인 클래스 중에서도 핵심이 되는 부분이 run() 메쏘드라는 것을 이미 알 겁니다. 다른 대부분의 로봇들도 그렇듯이 Fermat도 핵심이 되는 부분은 전체적으로 상당히 간단한 논리로 되어 있습니다. 대강 훑어보면 다음과 같습니다.

- 적 로봇들에 대한 정보를 유지한다.
- 이 정보를 바탕으로 총알을 발사한다.
- 발사한 총알에 대해 모니터링을 한다.

이것이 거의 전부입니다. 여기서 주목할 점은 Fermat의 메인 클래스에서는 자신이 발사한 총알에 대해 집중적으로 관리를 한다는 점입니다. 그도 그럴 것이 Fermat의 전체적인 알고리즘의 핵심은 각각의 적 로봇들에게 어떤 전략을 썼을 때 명중률이 높은가를 알아내서 이 정보를 기초로 각 적들을 상대합니다. 따라서 명중률의 분석을 위한 총알 정보의 관리는 필수적인 요소이지요.

onScannedRobot() 메쏘드는 알다시피 다른 로봇을 레이더로 포착했을 때 불려지는 이벤트 메쏘드입니다. 보통 적이 포착되면 총알을 발사하는 행동을 하는 로봇들이 많지만, Fermat는 적이 포착되면 적 로봇에 대한 여러 정보만 알아내서 저장합니다. 총알을 실제로 발사하는 것은 적 로봇의 정보를 가지고 있는 OhterBot 클래스를 이용할 때 인데, 이 클래스 안에서는 이미 저장하고 있는 적 로봇들의 움직임 등을 분석해서 그에 맞는 전략을 선택하여 총알을 발사하도록 되어 있습니다.

적을 포착하자마자 총알을 발사하지 않고 일단 분석이 끝나야 총알을 발사한다는 점을 주목할 필요는 있지만, 이 부분은 수정할 여지가 많이 있습니다. 왜냐하면 Fermat로 전투를 여러 번 실행해 보면 알겠지만 가끔씩 적과 아주 가까운 거리에 붙게 되었는데도 총을 쏘지 않는 경우가 있습니다(처음 게임을 시작할 때는 거의

100% 총을 쏘지 않습니다). 우리 일상 생활에서도 논리적으로 심사숙고한 후에 행동해야 할 때도 있지만, 생각보다는 행동이 먼저 일어나야 할 때가 있습니다. 이 점을 고려해서 Fermat를 수정한다면 좀더 좋은 로봇으로 발전할 거라고 생각됩니다.

onScannedRobot() 메쏘드처럼 다른 이벤트 메쏘드들(onHitRobot(), onHitWall(), onBulletHit(), onBulletMissed() 등)에서도 이벤트가 발생하는 즉시 어떤 행동을 취하지는 않습니다. 단지 그런 이벤트가 발생했다는 정보만을 저장할 뿐입니다. 그에 따른 행동은 일단 분석을 거치고 난 다음에 어떤 행동을 다음 번에 취하는 것이 좋을지 결정되어야 실행에 옮깁니다. 그래서 Fermat가 전투 중에 자주 갈팡질팡하는 모습을 볼 수 있습니다. 이런 모습은 꽤 좋은 성적을 거두는 로봇들에게서 공통적으로 보이는 모습들이므로 이런 부분을 수정하는 것도 좋지만 이런 점을 역이용하는 것도 좋지 않을까 생각됩니다. 즉, 똑똑한 로봇들이라면 위험한 상태로 빠지는 것을 예방하는 쪽으로 움직이기 때문에 여러 위협적인 상황들이 동시에 찾아오면 한 자리에 발이 묶이는 경우가 꽤 있습니다. 따라서 똑똑한 적 로봇에게 먹힐 만한 여러 위협적인 상황들을 동시에 만들 수 있다면 적을 꼼짝없이 묶어두고 총알을 퍼부을 수 있다는 것이지요.

이벤트들에 대한 설명은 이쯤하고 다음은 메인 클래스의 findBestTarget() 메쏘드를 보겠습니다. Fermat는 이 메쏘드를 이용해서 가장 공격하기 좋은 타겟을 찾는데, 물론 이것은 일대일 상황에서는 쓰이지 않습니다. 베스트 타겟을 찾는 방법은 생각보다 간단합니다. Fermat가 생각하는 베스트 타겟의 조건은 '가까운 거리'에 있고, '총을 돌리기 쉬운 곳'에 있는 적입니다. 다시 말해 가까운 거리에 있는 적 중에서 총을 조금만 회전해도 공격할 수 있는 적을 제일 좋은 타겟으로 생각한다는 것입니다. 다음에 나온 코드가 베스트 타겟을 찾는 핵심 코드인데, 전장에 있는 적들의 정보를 하나씩 찾아보면서 베스트 타겟의 조건에 가장 적합한 적을 찾는다는 것이 주요 내용입니다.

```
for(i=0;i<otherBots.size();i++)
{
   bot = (OtherBot)otherBots.elementAt(i);
   if(bot.alive)
   {
      turn = Math.abs(findGunTurn(bot.X,bot.Y));
      value = Math.pow(((bot.X-getX())*(bot.X-getX())
         +(bot.Y-getY())*(bot.Y-getY())),2)
         * Math.pow(45+turn,0.5);
      if(value < minValue)
      {
         minValue = value;
         target = i;
      }
   }
}
```

다음으로 Fermat의 메인 소스 파일에서 집중해서 볼 만한 것은 총과 레이더를(지금 타겟으로 잡고 있는 하나의) 적 방향으로 회전하기 위해 각도를 알아내는 코드입니다. 이것과 관련된 코드는 다음과 같습니다.

```
double findGunTurn(double x,double y)
{
   double angle = normalAbsoluteHeading(
      Math.toDegrees(Math.PI/2 - Math.atan2(getY() - y, getX() - x)));
   return normalRelativeAngle(getGunHeading() - angle + 180);
}

double findRadarTurn(double x,double y)
{
   double angle = normalAbsoluteHeading(
      Math.toDegrees(Math.PI/2 - Math.atan2(getY() - y, getX() - x)));
   return normalRelativeAngle(getRadarHeading() - angle + 180);
}

public double normalAbsoluteHeading(double angle)
{
   if (angle < 0)
      return 360 + (angle % 360);

   else
      return angle % 360;
}

public double normalRelativeAngle(double angle)
{
   if (angle > 180)
      return ((angle + 180) % 360) - 180;
   if (angle < -180)
      return ((angle - 180) % 360) + 180;
   return angle;
}
```

findGunTurn()은 적이 있는 방향을 향하려면 지금 Fermat의 총이 있는 위치에서 얼마의 각도를 돌리면 되는지, 그 각도 값을 알아내는 메쏘드입니다. findRadarTurn()도 레이더를 대상으로 똑같은 기능을 합니다.

이 두개의 메쏘드에서 공통적으로 사용하는 것이 normalAbsoluteHeading()과 normalRelativeAngle() 메쏘드입니다. 전자는 화면 상단을 0도로 했을 때 적 로봇의 각도를, 후자는 Fermat의 입장에서 보는 상대적인 각도를 알아내는 코드입니다. 로봇 두개를 종이에 그려 놓고 잠시 계산해 보면 쉽게 이해할 수 있는 코드인데, 주의할 점이 있습니다. 주의할 점이란 X좌표와 Y좌표를 Fermat를 중심에 놓고 계산한 것이 아니라 적 로봇을 중심에 두고 계산한다는 점입니다. 다시 말하면 'Math. atan2(getY() - y, getX() - x))' 부분을 주의해서 보라는 것이고, 종이에 그릴 때 Fermat를 중점에 놓지 말고 적 로봇을 중점에 두라는 것입니다. 사실 필자는 계속 Fermat를 중심에 놓고 계산하다가 이 코드를 분석하는데 많은 시간을 허비했습니다. 그래서 이 코드가 그리 어려운 부분은 아니지만 논리가 좀 헛갈리게 되어 있기 때문에 여러분도 필자처럼 시간을 낭비하지 않길 바라는 마음으로 언급했습니다(이 부분만 이해를 하면 몸체를 움직이는 goTo() 메쏘드도 쉽게 이해할 수 있습니다).

OtherBot 클래스

OtherBot 클래스는 적 로봇들에 대한 정보 저장과 총알 발사를 위해 쓰입니다. 일단 이 클래스에서 주의해서 볼 것은 적 로봇에 대해 어떤 정보들을 저장하고 있는가 하는 것입니다. 변수 이름을 보면 대충 감이 오겠지만 간단히 설명하면 적 로봇의 현재 에너지, 예전 에너지, 좌표, 앞머리가 향하고 있는 각도, 속도, 평균 속도, 평균 각도, 마지막으로 공격한 시간, 스캔 한 시간, 예전에 스캔 한 시간, 살아있는지 죽어 있는지에 관한 정보, 이름, 현재 적용하고 있는 패턴 등입니다. 여기서 주의할 것은 lastWin이라는 변수는 Fermat가 이 로봇에 대해서 이긴 횟수가 아니라, 이 로봇이 Fermat를 이긴 횟수입니다. 즉, Fermat가 이 로봇에 의해 패배한 횟수를 뜻합니다.

이제 메쏘드들을 간단히 살펴보겠습니다. update() 메쏘드는 주어진 파라미터대로 로봇의 정보를 업데이트하는 간단한 역할을 합니다. fireBot() 메쏘드는 이 로봇에 대해 어떤 전략으로 총알을 발사할 것인지, 그리고 그때 총알의 세기는 어떻게 할 것인지를 판단합니다. 이 클래스에서는 특별히 중요하다거나 재활용할 수 있는 부분이 없습니다. 그러니 이 정도로 하고 다음으로 넘어가겠습니다.

FiringStrategy 클래스

FiringStrategy는 꽤 중요하면서도 어렵고 그렇지만 꼼꼼히 봐둘 필요는 있는 그런 클래스입니다. 이 클래스는 말 그대로 적 로봇에 대해 어떤 전략으로 총알을 발사할지 결정하는 역할을 합니다. 풀어서 쓰면 이 클래스의 원리는 이렇습니다. 지금 타겟으로 잡고 있는 적에 대해 예전에 보인 행동과 지금 보이고 있는 행동들을 종합하여 어떤 패턴으로 행동하고 있는지 분석하고, 이 분석에 맞게끔 총알을 발사하는 것입니다. 즉, 적이 Linear라는 직선으로 쭉 전진하는 행동을 보이고 있다면 약간 앞쪽으로 총알을 발사해야 명중할 확률이 높을 것입니다. FiringStrategy 클래스는 바로 이런 패턴 분석을 한 다음 총알을 발사하기 위한 좌표를 넘겨주는 것입니다.

이 클래스 중에서도 다른 클래스에서 불려지는 메쏘드는 applyStrategy() 메쏘드 뿐인데, 이것은 OhterBot 클래스에서 불려집니다. 적 로봇들의 정보를 저장하는 클래스에서 총알 발사를 위한 정보도 관리한다는 것이 처음에는 논리에 맞지 않는 듯 생각됩니다. 그러나 발사한 총알의 명중률을 계산하고 이 명중률을 바탕으로 다시 총알 발사를 위한 전략을 선택한다는 것을 알게 되면 왜 이런 구조를 가지는지 이해할 수 있습니다. applyStrategy()에서는 단순히 9개의 전략 중에서 하나를 선택하고 그에 맞는 총알 발사를 위한 좌표 값을 돌려줍니다. 9개의 전략은 다음과 같습니다.

- AvgLinearStrafe
- AvgCircularStrafe
- AvgCircularStop
- AvgLinearStop
- AvgVelCircular
- AvgLinear
- Circular

◆ Linear

◆ default

이름에서도 대충 알 수 있듯이 직선 운동이나 원 운동 등의 패턴과 유사한 패턴들입니다(평균 값으로 대충 봤을 때 비슷한 패턴을 하고 있다는 등). 각각의 전략들을 모두 세세하게 분석한다는 것은 무리이고 몇 마디 말로 표현하는 것도 한계가 있기 때문에 여러분들이 직접 분석해 보라고 말할 수밖엔 없습니다. 그러니 간단히 핵심 코드만 설명해 보겠습니다.

```
nextTime = (long)Math.round(Math.sqrt(((robot.getX()-p.x)*
   (robot.getX()-p.x)+(robot.getY()-p.y)*(robot.getY()-p.y)))
      /(20-(3*firePower)));
time = robot.getTime() + nextTime;
double diff = time - bot.scanTime;
p.x = bot.X + Math.sin(Math.toRadians(normalRelativeAngle(bot.heading)))
                  *bot.velocity*diff;
p.y = bot.Y + Math.cos(Math.toRadians(normalRelativeAngle(bot.heading)))
                  *bot.velocity*diff;
```

앞의 코드가 이 클래스 전체적으로 사용되는 핵심 개념입니다. 모두 이 핵심 코드를 약간 수정하거나 덧붙인 형태이므로, 이 코드만 이해한다면 클래스를 전체적으로 이해하는데 큰 문제는 없을 것입니다. 이 코드의 분석을 위해서는 우선, '거리 = 속력×시간' 이라는 공식을 생각해야 합니다. 그리고 코드 중간에 보이는 (20-(3*firePower))라는 코드는 총알이 날아가는 속도입니다. 즉, 총알의 파워가 1이면 한 턴에 17(픽셀과 유사)이라는 속도로 날아가고, 3이면 한 턴에 11이라는 속도로 날아갑니다(총알의 파워가 클수록 날아가는 속도가 느려집니다). 이 식은 로보코드 FAQ 문서에 나와 있습니다.

따라서 앞의 코드에서는 주어진 총알의 파워에 따라서 총알이 날아가는 속도를 계산합니다. 그리고 이것을 거리와 나누면(거리/속도), 적이 위치하고 있는 곳까지 총알이 도달하는 시간을 계산할 수 있습니다. 이 시간을 기초로 Fermat가 위치할 지점을 계산합니다. 이때 예측한 지점이 전투 필드의 범위를 벗어난다면 당연히 고려할 필요가 없겠지요. 따라서 예측 지점이 잘못된 좌표라면 (-1, -1)을 넘겨주어 예측이 실패했다는 것을 알려줍니다. 이 정도만 하면 클래스 전체를 분석하는데 무리가 없을 것으로 생각됩니다.

RobotBody 클래스

RobotBody는 이름 그대로 Fermat의 몸체를 컨트롤하기 위한 클래스입니다. 여기에는 반 중력 운동(Anti-gravity)과 총알 피하기(Dodge Bullet) 알고리즘이 한꺼번에 구현되어 있으므로 이런 알고리즘을 구현하는데 어려움을 겪고 있는 사람이라면 꼭 한번 분석해 보기를 권합니다. 물론 반 중력 운동이 구현되어 있으므로 벽 피하기(wall avoidance)도 어느 정도 가능합니다.

앞서 말했듯이 이 소스 파일 안에는 DodgeBullet 클래스가 하나 더 들어가 있습니다. 총알 피하기는 적 로봇의 에너지가 0.1에서 3사이의 크기로 감소했다면 총알을 발사한 것으로 간주하고, 발사된 것으로 생각되는 총알을 피하는 알고리즘입니다. 이때 DodgeBullet에서 가지고 있는 정보는 총알이 발사된 지점과 타겟이 되는 지점입니다. 정확하게는 발사되었다고 생각되는 지점인데 알다시피 적이 발사한 총알은 레이더에 포착되지 않습니다. 그래서 적의 에너지 감소를 보고 판단하는 데, 현재 레이더에 포착된 적의 상태가 S라고 하고, 바로 이전에 포착됐을 때 적의 상태를 S' 라고 가정해 봅시다. 이때, S' 와 S에서 에너지 변화가 감지되었다면, 총알은 S' 지점과 S지점 사이에서 발사되었다고 볼 수 있습니다. 총알은 Fermat를 향해 발사되었을 것이므로, 적이 S와 S' 상태일 때 Fermat가 위치했던 좌표가 타겟 지점이 됩니다. 타겟 지점도 대강 중간 정도로 가정하면, 총알을 발사한 지점과 타겟이 되는 지점 사이에 선을 하나 그릴 수 있습니다. 바로 이 선을 피하면 총알을 피할 수 있다고 보는 것입니다. 실제로 Fermat의 DodgeBullet 클래스에서도 이런 정보를 가지고 있습니다.

```
Point2D.Double source;
Point2D.Double target;
double velocity;
double startTime;
```

Fermat는 이런 단순한 논리에다가 한 가지를 더 추가했습니다. 총알을 피하더라도 벽에 부딪히지 않게 하기 위해 triangleArea()라는 메쏘드를 두어 총알을 발사한 지점과 타겟 지점, 그리고 총알을 발사한 지점과 벽 사이의 거리를 기초로 일련의 계산을 합니다. 벽에 부딪히면 에너지가 감소하기도 하지만 적에게 좋은 먹이감이 될 수도 있습니다. 넓은 곳에 있는 것보다 상대적으로 도망갈 장소가 그만큼 줄어들기 때문입니다. 따라서 그 정보를 기초로 Fermat가 이동할 방향을 결정합니다.

반 중력 운동(anti-gravity)은 적 로봇들과 벽에 중력 값을 두고 중력 값들을 가지는 물체에 너무 가까이 다가가지 않게 행동한다는 것이 핵심 원리입니다. 문서에는 좀 애매하게 나와 있는데 핵심 내용은 이것이 전부입니다. Fermat는 자주 벽이나 코너에 몰리는 모습을 볼 수 있습니다. 이것은 벽보다 적 로봇에 중력 가중치를 더 두었기 때문입니다. 즉, 적과 적이 발사한 총알을 피하는 것이 우선이기 때문에 이런 결과가 나타나는 것입니다.

여기에도 약간의 개선의 여지가 있습니다. Fermat도 총알이 도착할 시간을 대충 계산해서 피하기는 하지만, 이것을 좀더 자세하게 계산하도록 할 수도 있을 것입니다. 그리고 총알을 옆으로만 피하지 말고, 벽에 몰릴 상황이 되었다면 과감하게 뒤로 돌아 나오는 형태로 피하는 움직임을 취하게 할 수도 있을 것입니다. 제 아무리 총알 피하기 알고리즘을 사용한다 하더라도, 벽이나 코너에 몰리게 되면 총알을 맞을 확률은 높아질 수밖에 없기 때문입니다(실제로 이 부분을 해결하는 것이 반 중력 움직임의 문제이기도 합니다). RobotBody 부분은 이런 내용들을 바탕으로 대충 살펴보면 쉽게 이해할 수 있으리라 생각됩니다.

나머지 클래스들

이제 남은 나머지 클래스들은 100라인이 채 안되는 짤막한 것들입니다. 이 클래스들은 주로 정보를 저장하는 목적으로 쓰여지는 것이고, 짧은 만큼 어려운 부분도 없기 때문에 짤막하게 설명하고 넘어가겠습니다.

먼저 PatternAnalyzer는 적 로봇이 어떤 패턴을 보이는가를 분석하는 역할을 합니다. 어떻게 패턴을 분석하는지는 패턴 분석을 위해 어떤 변수를 사용하는지만 봐도 대강 알 수 있습니다. 즉, 여태까지 알아낸 적 로봇들의 위치들을 바탕으로 해서 움직인 시간의 평균이나 가속을 했는지 여부 등을 알아내고, 이것을 기초로 어떤 움직임을 보이고 있는지 추측하는 것입니다. 예를 들어, 한쪽 방향으로 등속도 운동을 하고 있다면 직선 운동을 하고 있다고 추측할 수 있고, 한쪽 방향으로 속도가 줄어들다가 다른 쪽 방향으로 곧 속도가 증가한다면 원 운동을 하고 있다고 볼 수 있습니다.

```
public int patterns;
public double avgReverseTime;
public double avgReverseTimeSqr;
public long timeLastReverse;
public boolean accelerating;
public double acceleration,lastAcceleration,lastVelocity,stddev;
public double avgMovingTime,startMovingTime,avgMovingTimeSqr,movestddev;
```

Statistic은 shots와 hits를 가지고 있는데, 이것은 각각 발사한 총알 수와 명중한 총알 수입니다. 따라서 총알의 명중률을 계산하고 보관하는 역할을 합니다. AdvancedBullet은 로보코드에서 기본으로 제공하는 Bullet 클래스를 가지고, 그에 덧붙여서 이 총알의 전략, 타겟 로봇, 발사된 지점의 정보를 가지고 있습니다. 이것은 어떤 로봇에 대해 어떤 전략으로 발사한 총알이 명중하거나 빗나갔는지 알 수 있도록 하기 위함입니다.

마지막으로 DataWriter에서는 각 로봇들의 Statistic의 정보를 파일로 저장하기 위한 클래스입니다. Fermat를 몇 번 전투에 내보낸 다음 Fermat의 패키지 폴더(ak라는 이름으로 있음)를 열어보면 폴더가 하나 더 만들어져 있는 것을 볼 수 있습니다. Fermat.data라는 이름으로 되어 있는데, 이 폴더 안을 보면 Fermat가 싸운 로봇들의 이름으로 파일들이 만들어져 있습니다. 각 파일들은 다음과 같은 형식으로 되어 있습니다(다음의 데이터는 emp.Yngwie.txt라는 이름으로 되어 있는 것을 불러 왔습니다).

3	2	3
1	1	1
2	4	2
1	1	1
2	4	4
1	2	1

총 18라인으로 되어 있는 이 파일은 9개의 전략들에 따른 명중률을 나타냅니다. Fermat가 총알을 발사하기 위한 9개의 전략을 가지고 있다는 것을 앞에서 설명했습니다. 따라서 Fermat는 9개

의 전략 중 하나를 선택해서 하나의 총알을 발사합니다. 이 총알은 앞의 AdvancedBullet으로 되어 있어서 어떤 전략으로 어떤 로봇을 향해 발사되었는지 알 수 있습니다. 그리고 이 정보를 바탕으로 Statistic의 shots와 hits 값을 계산하는 것입니다. 다시 말해 1번 전략으로 Yngwie에게 총알을 발사했는데, 몇 개의 총알을 발사해서 몇 번을 명중했느냐의 정보를 파일로 저장한 것입니다. 앞에서부터 2라인씩 하나의 전략에 대한 shots와 hits 값입니다. 일대일 전투를 했을 경우에는 '1v1'이라는 이름이 뒤에 더 붙는데, 이 파일들은 19라인으로 한 라인이 더 있습니다. 19번째 라인은 OhterBot의 lastWin 변수 값으로, Fermat가 상대 로봇에게 몇 번 패배했는지 횟수를 기록한 것입니다.

Fermat의 가장 큰 특징이 바로 이것입니다. 적 로봇에 대한 정보를 저장해 두고 계속되는 라운드에 이 정보를 활용한다는 것이지요. 이 정보를 바탕으로 어떤 로봇에게 어떤 전략으로 총알을 발사하는 것이 명중률이 높았는지 알 수 있습니다. 이것을 좀더 확장해서 유용한 정보들을 더 많이 파일로 저장해 둔다면, 예전 전투들의 기록들을 바탕으로 적에 대한 정보를 좀더 많이 알아낼 수 있을 것입니다.

2.0 버전에서 달라진 것들

Fermat 2.0 버전은 소스 코드는 공개하고 있지 않지만 어떤 부분들이 달라졌는지에 대한 설명은 있습니다. 이것은 이번에 설명한 Fermat가 어떤 부분이 취약한지 알 수 있는 자료가 됩니다. 따라서 이번에 설명한 Fermat의 소스 코드와 기본 원리를 바탕으로 다음의 2.0에서 달라진 내용들을 모두 합쳐서 생각하다보면 여러분들도 자신의 로봇을 좀더 업그레이드할 수 있으리라 봅니다.

- ◆ 좀더 많은 패턴들을 추가함
- ◆ 움직임에 관한 계산들을 개선함
- ◆ 가상 총알(Virtual Bullet)개념을 이용함
- ◆ 적절치 못한 몇 가지 코드들을 삭제함

2.0의 움직임을 보았을 때 기존 Fermat에서 총알 발사와 적 패턴 분석에 쓰이는 패턴들을 좀더 추가한 듯 합니다. 패턴은 아주 많은 것들이 있을 수 있으므로 이런 패턴들만 많이 주어진다면 로봇 성능이 상당히 좋아질 수 있습니다. 그리고 반 중력 운동과 총알 피하기에서 몇 가지 계산들을 수정한 듯 합니다. 세 번째가 생소한 사람이 있을 것입니다. 가상 총알이라는 것은 총알을 실제로는 발사하지 않고 발사했다고 가정하는 것입니다. 그래서 가상으로 발사한 총알이 명중했는지 어느 정도 근접했는지를 알아본 다음에 진짜 총알을 발사하는 방법입니다. 다양한 전략 중에 어떤 것을 선택해야할지 알 수 없는 상황에 유용하게 쓰입니다. 전략을 선택한 이후라도 선택한 전략이 과연 제대로 된 선택인지 알아보기 위해서도 쓰입니다.

훨씬 더 좋은 로봇을 기대하며

Fermat라는 로봇의 소스 코드를 분석해 보았습니다. 이번에 분석한 버전도 꽤 좋은 성능을 보이지만, 가장 최근 버전인 2.0은 감히 Yngwie가 따라올 수 없을 정도로 좋은 성능을 보여주는 로봇입니다. 머리 속에 각종 전략과 행동 패턴들은 생각해 놓았지만 막상 어떻게 구현해야할지 막막한 분들이나, 좋은 로봇을 만들기 위해 어떤 기능들을 넣어야할지 답답한 사람에게 이번 소스 분석이 도움이 되었길 바랍니다.

정리 | 조규형 | jokyu@korea.cnet.com

참 + 고 + 자 + 료

❶ 로보코드 다운로드 사이트(http://robocode.alphaworks.ibm.com/installer/) : 로보코드 프로그램을 다운받을 수 있고, 관련 API와 각종 정보를 얻을 수 있음

❷ 로보코드 우승자 인터뷰(http://www-903.ibm.com/developerworks/kr/java/library/j-robowrap.html) : 로보코드 우승자들의 인터뷰 내용, Fermat의 소스 코드를 다운받을 수 있음

❸ Robocode FAQ (http://robocode.alphaworks.ibm.com/help/robocode.faq.txt) : 로보코드에 대한 좀더 많은 정보들을 얻을 수 있음

["로봇을 소개해주세요"]

Jr.에서는 앞으로 독자 여러분이 만든 로봇을 글로 소개하고자 합니다. 자신이 만든 로봇을 Jr.에서 자랑해보세요. 신청은 다음과 같습니다.

대상 : 로봇을 만든 사람이면 누구나 가능

기간 : 7월 1일부터 항시 접수

신청 방법 : 자신이 만든 로봇의 특징을 설명한 글을 조규형 기자(jokyu@korea.cnet.com)에게 메일로 접수한다.