الگوریتمهای تعقیب و گریز

آدم کم تجربه سرش را از دست می دهد، یا موشک لیزری را هدایت می کند؟ این انتخاب شماست. الگوریتمهای جستجو کردن و گریختن بکار رفته در بازی اختلاف زیادی در نوع بازی شما بوجود می آورد، در این مورد شک نکنید. دشمنان شما می توانند همه نوع حرکات و توانایی ها را داشته باشند، اما اگر آنها نتوانند هوشمندانه موقعیت بازیگر را تعیین کنند (به منظور تعقیب و گریز)، ترس را القا نخواهند کرد و یا احساسات واقعی را بیان نمی کنند.

سراغ ریاضیات برویم! تعیین فاصله میان یک دشمن و بازیکن از مقدماتی ترین مسائل مهم می باشد. اگر بازیکن دور است، دشمنان باید مجموعه ای از رفتارهای معین را نمایش دهند (یا "وضعیتها"- مقاله ماشینهای وضعیت متناهی را مطالعه کنید). اگر بازیکن نزدیک است، مجموعه ای از رفتارهایی که کاملا متمایز است را باید نشان دهد. من یک تابع می نویسم که به سرعت فاصله بین دو نقطه را در مختصات X,Y را تعیین می کند:

```
Private Function GetDist(intX1 As Single, intY1 As Single, intX2 As Single, intY2 As Single) As Single
   GetDist = Sqr((intX1 - intX2) ^ 2 + (intY1 - intY2) ^ 2)
End Function
```

بله، بنابراین ما دانستیم بازیکن چقدر فاصله دارد، اما در کدام جهت؟ ما باید یک تابع برای این هم بنویسیم!

```
Private Function FindAngle(intX1 As Single, intY1 As Single, intX2 As Single, intY2 As Single) As Single
   Dim sngXComp As Single
   Dim sngYComp As Single

   sngXComp = intX2 - intX1
   sngYComp = intY1 - intY2
   If sngYComp > 0 Then FindAngle = Atn(sngXComp / sngYComp)
   If sngYComp < 0 Then FindAngle = Atn(sngXComp / sngYComp) + PI
End Function
```

این تابع زاویه (بر حسب رادیان) بین دو نقطه داده شده را برمی گرداند. توجه کنید که مقدار Y وارونه شده است چون که (در دنیای کامپیوتر) میزان Y همچنان که به سمت پایین صفحه می رویم افزایش می یابد.

برای بسیاری از بازیها، به این دو تابع نیاز خواهید داشت.شما می دانید بازیکن کجاست، و چگونه به طرفش بروید، پس به سادگی دشمنانتان را در آن مسیر حرکت دهید (یا مسیر عکس را برای فرار انتخاب کنید). در حالتهایی که موجودیتهای محل بازیکن و دشمن بوسیله بردارهای سرعت کنترل شده است، ما به چیزهای بیشتری نیاز داریم...

```
Private Sub AddVectors(sngMag1 As Single, sngDir1 As Single, sngMag2 As Single, sngDir2 As Single,
Optional ByRef sngMagResult As Single, Optional ByRef sngDirResult As Single)
    Dim sngXComp As Single
    Dim sngYComp As Single

    sngXComp = sngMag1 * Sin(sngDir1) + sngMag2 * Sin(sngDir2)
    sngYComp = sngMag1 * Cos(sngDir1) + sngMag2 * Cos(sngDir2)

    sngMagResult = Sqr(sngXComp ^ 2 + sngYComp ^ 2)

    If Sgn(sngYComp) > 0 Then sngDirResult = Atn(sngXComp / sngYComp)
    If Sgn(sngYComp) < 0 Then sngDirResult = Atn(sngXComp / sngYComp) + PI

End Sub
```

من این تابع را دوست دارم! آنهایی که چگونگی استفاده از بردارها را درک نکرده اند، نگران نباشند! این تابع برای شما آنها را باهم اضافه می کند. نیازی به فکر کردن نیست. به سادگی مقدار و جهت دو برداری را که می خواهید اضافه کنید، ارسال کنید، و آن مقدار و جهت بردار برآیند را برای شما باز می گرداند (sngMagResult و sngDirResult) !

ما در اینجا چه باید بکنیم؟ خوب، اجازه دهید چند لحظه در مورد مشکلمان بیندیشیم. فرض کنید که بازیکن ما و دشمن در مقابل هم هستند. درست این است که دشمن ما بازیکن را تعقیب کند، او باید ابتدا زاویه بین آنها را محاسبه کند، و در آن جهت حمله کند، درست است؟ بله، اما پس از آن به سادگی شلیک به بازیکن به خطا می رود (زیرا موقعی که

شما حمله کردن را متوقف کرده اید اشیا در فضا بسادگی توقف نمی کنند!)، و ناگزیر به تغییر زاویه به اطراف هستید. ما به بعضی از مکانیزمهای طبقه بندی ترمز کردن نیاز داریم.

و نیز اگر بازیکن متحرک باشد؟ دشمن ممکن است بتواند موقعیت فعلی بازیکن را جستجو کند، اما پیش بینی مکان آینده آن بدیهی است که مفید است (البته نه برای واقع گرایی بیشتر) .

برای انجام همه این چیزها، ما دو مقدار جدید نیاز داریم:

- تفاوت بین سرعتهای دشمن و بازیکن
- سرعت مورد نیاز برای جلوگیری از بازیکن

برای جلوگیری از خطا رفتن بازیکن، ما مجبور به تطبیق سرعتها با او در بعضی نقاط هستیم، از اینرو به مقدار سرعت اولیه نیاز داریم. و نیز در ادامه برای پیش بینی حرکت بازیکن، ما باید سرعت مورد نیاز برای برخورد را محاسبه کنیم و آن را به عنوان سرعت ثانویه فوق الذکر قرار بدهیم.

برای محاسبه تفاوت سرعت، به سادگی بردارهای سرعت بازیکن و دشمن را تفریق می کنیم! این کار را به آسانی با اضافه کردن Pi به هرکدام از بردارهای "جهت" (از این طریق آنها را معکوس می کنیم)، و سپس استفاده از تابع AddVectors انجام می دهیم.

برای محاسبه بردار برخورد کمی ترفند لازم است. جهت این بردار می تواند با استفاده از تابع FindAngle محاسبه شود؛ به آسانی مختصات بازیکن و دشمن را به آن ارسال می کنید. برای تعیین مقدار، ما باید ابتدا فاصله بین دشمن و بازیکن را تعیین (تابع GetDist)، و آن را بر مقدار تفاوت سرعت تقسیم ، و در شتاب دشمن ضرب کنیم.

چرا؟ پیدا کردن فاصله و تقسیم آن بر سرعت، زمان برخورد را به ما می دهد (چقدر مانده تا دشمن به بازیکن با سرعت معین فعلی برسد). ضرب کردن این زمان با شتاب به ما سرعت برخورد را می دهد.

فیزیک مفرح است، اینطور نیست؟

حالا، ما این بردارها را برای تشخیص اینکه دشمن ما در کدام جهت باید شلیک کند باهم جمع می کنیم (با استفاده از تابع AddVectors). می توانید کد آن را از اینجا دانلود کنید:

```
AddVectors msngSpeed, msngHeading, msngEnemySpeed, msngEnemyHeading + PI, sngMagDiff, sngDirDiff
If sngMagDiff <> 0 Then
   AddVectors ENEMY_ACCEL * GetDist(msngX, msngY, msngEnemyX, msngEnemyY) / sngMagDiff,_
              FindAngle(msngEnemyX, msngEnemyY, msngX, msngY), sngMagDiff,_
              sngDirDiff, , msngEnemyFacing
Else
   msngEnemyFacing = FindAngle(msngEnemyX, msngEnemyY, msngX, msngY)
End If
```

خط اول اختلاف سرعت را بدست می آورد. عبارت If به ما این اطمینان را می دهد که تقسیم بر صفر اتفاق نمی افتد. دومین فراخوانی AddVectors سرعت برخورد و اختلاف سرعت را پذیرفته و مقدار حرکت برای دشمن را برمی گرداند.

خوب شاید این کمی برای فهمیدن مشکل باشد، اما شما واقعا مجبور نیستید آن را درک کنید. فقط از آن استفاده کنید!!!

سورس آن را می توانید از اینجا دریافت کنید:

http://www.rookscape.com/vbgaming/intercept.zip

مترجم: محمد صافدل
منبع: http://www.rookscape.com