« به نام او »

آموزش شبیه سازی دو بعدی فوتبال

نویسنده : محمد علی میرزایی - علی یعقوبی

مرجع روبوکاپ ایران

www.iranrcss.com

جلسه یازدهم

مباحث این جلسه :

۱- آموزش بیس UVA_Trilearn Base

۲- آموزش های هوش مصنوعی – جستجوی آگاهانه و اکتشاف

در جلسه قبل تا خط شماره ۱۳ را برسی کردیم . به ادامه بحث می پردازیم :

```
۱ - SoccerCommand BasicPlayer::mark( ObjectT o, double dDist, MarkT mark )
۲ - {
۳ - VecPosition posMark = getMarkingPosition( o, dDist, mark );
۴ - VecPosition posAgent = WM->getAgentGlobalPosition();
۵ - VecPosition posBall = WM->getGlobalPosition( OBJECT_BALL );
۶ - // AngDeg angBody = WM->getAgentGlobalBodyAngle();
۷ - if( o == OBJECT_BALL )
۸ - {
۹ - if( posMark.getDistanceTo( posAgent ) < ۱.۵ )
۱۰ - return turnBodyToObject( OBJECT_BALL );
۱۱ - else
۱۲ - return moveToPos( posMark, ۳۱.۱, ۳.۱, false );
۱۳ - }
```

```
١٤ - if( posAgent.getDistanceTo( posMark ) < ٢.١ )
١٥ - {
١٦ - AngDeg angOpp = (WM->getGlobalPosition( o ) - posAgent).getDirection();
١٧ - AngDeg angBall = (posBall - posAgent).getDirection();
١٨ - if( isAngInInterval( angBall, angOpp,
١٩ - VecPosition::normalizeAngle( angOpp + ١٨١ ) ) )
٢٠ - angOpp += ٨١;
٢١ - else
٢٢ - angOpp -= ٨١;
٢٣ - angOpp = VecPosition::normalizeAngle( angOpp );
٢٤ - Log.log( ٥١٣, "mark: turn body to ang %f", angOpp );
٢٥ - return turnBodyToPoint( posAgent + VecPosition( ١.١, angOpp, POLAR ) );
٢٦ - }
٢٧ - Log.log( ٥١٣, "move to marking position" );
٢٨ - return moveToPos( posMark, ٢٥, ٣.١, false );
٢٩ - }
```

ابتدا خطوط زیر را با هم بررسی می کنیم :

```
۱۴ - if( posAgent.getDistanceTo( posMark ) < ۲.۱ )
۱۵ - {
۱۶ - AngDeg angOpp = (WM->getGlobalPosition( o ) - posAgent).getDirection();
۱۷ - AngDeg angBall = (posBall - posAgent).getDirection();
۱۸ - if( isAngInInterval( angBall, angOpp,
۱۹ - VecPosition::normalizeAngle( angOpp + ۱۸۱ ) ) )
۲۰ - angOpp += ۸۱;
۲۱ - else
۲۲ - angOpp -= ۸۱;
۲۳ - angOpp = VecPosition::normalizeAngle( angOpp );
۲۴ - Log.log( ۵۱۳, "mark: turn body to ang %f", angOpp );
۲۵ - return turnBodyToPoint( posAgent + VecPosition( ۱.۱, angOpp, POLAR ) );
۲۶ - }
```

در خط ۱۹ تابعی به نام `normalizeAngle` به کار رفته که وظیفه ی این رو داره که زاویه ی دریافتی را به رنج ۱۸۰- تا ۱۸۰ ببرد . در زیر تابع را آوردیم :

```
/*! This method normalizes an angle. This means that the resulting
    angle lies between -۱۸۰ and ۱۸۰ degrees.
    \param angle the angle which must be normalized
    \return the result of normalizing the given angle */
AngDeg VecPosition::normalizeAngle( AngDeg angle )
{
  while( angle > ۱۸۰.۰  ) angle -= ۳۶۰.۰;
  while( angle < -۱۸۰.۰ ) angle += ۳۶۰.۰;
  return ( angle );
}
```

حال به توضیح تابع getMagnitude می پردازیم :

```
/*! This method determines the magnitude (length) of the vector
    corresponding with the current VecPosition using the formula of
    Pythagoras.

    \return the length of the vector corresponding with the current
    VecPosition */
double VecPosition::getMagnitude( ) const
{
  return ( sqrt( m_x * m_x + m_y * m_y ) );
}
```

این تابع طول بردار ی که با آن تابع را فراخوانده ایم محاسبه میکند .

تا به اینجا ما به طور کامل تابع deMeer۵ در فایل playerTeams.cpp را بررسی کردیم . در اینجا می خواهیم تابع moveToPos از توابع فایل basicPlayer.h را بررسی کنیم . اما در این جلسه تنها خود تابع را می آوریم و در جلسه بعد به طور کامل به بررسی آن می پردازیم :

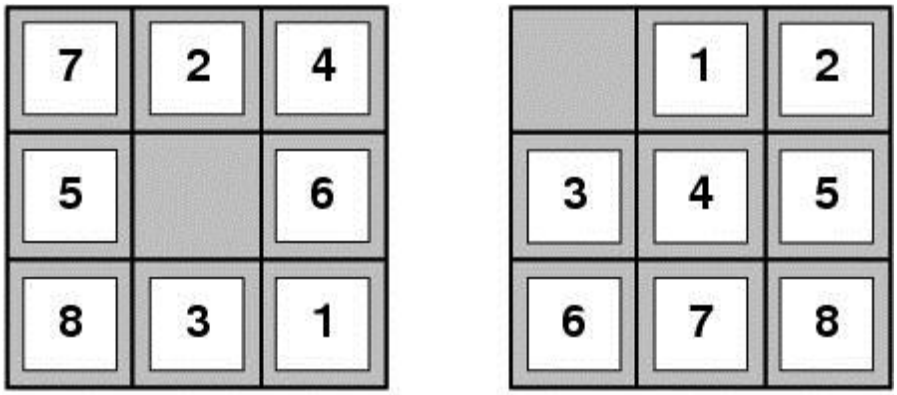
```
۱ - SoccerCommand BasicPlayer::moveToPos( VecPosition posTo, AngDeg angWhenToTurn,
                    double dDistBack, bool bMoveBack, int iCycles )

۲ - {
```

```
٣ - // previously we only turned relative to position in next cycle, now take

٤ - // angle relative to position when you're totally rolled out...

٥ - //  VecPosition posPred   = WM->predictAgentPos( ١, ٠ );

٦ - VecPosition posAgent  = WM->getAgentGlobalPosition();

٧ - VecPosition posPred    = WM->predictFinalAgentPos();

٨ - AngDeg      angBody   = WM->getAgentGlobalBodyAngle();

٩ - AngDeg      angTo     = ( posTo - posPred ).getDirection();

١٠ -            angTo     = VecPosition::normalizeAngle( angTo - angBody );

١١ - AngDeg    angBackTo = VecPosition::normalizeAngle( angTo + ١٨٠ );

١٢ - double     dDist        = posAgent.getDistanceTo( posTo );

١٣ - Log.log( ٥٠٩, "moveToPos (%f,%f): body %f to %f diff %f now %f when %f",
                posTo.getX(), posTo.getY(), angBody,
              ( posTo - posPred ).getDirection(), angTo,
              ( posTo -  WM->predictAgentPos( ١, ٠ )).getDirection(),
              angWhenToTurn );


١٤ - if( bMoveBack )

١٥ - {

١٦ - if( fabs( angBackTo ) < angWhenToTurn )
```

```
17 - return dashToPoint( posTo, iCycles );
18 - else
19 - return turnBackToPoint( posTo );
20 - }
21 - else if(  fabs( angTo     ) < angWhenToTurn ||
                (fabs( angBackTo ) < angWhenToTurn && dDist < dDistBack ) )
22 - return dashToPoint( posTo, iCycles );
23 - else
24 - return directTowards( posTo, angWhenToTurn );
25 - //return turnBodyToPoint( posTo );
26 - }
```

تابع بالا یکی از مهم ترین توابع بیس می باشد که وظیفه جا به جا کردن بازیکنان را دارد . در جلسه بعد به بررسی این تابع خواهیم پرداخت .

توابع اکتشافی



- **مثال برای معمای۸**
  - میانگِین هزینه حل تقریبا ۲۲ مرحله و فاکتور انشعاب در حدود ۳ است.
  - جست و جوی جامع تا عمق ۲۲
  - با انتخاب یک تابع اکتشافی مناسب میتوان مراحل جستجو را کاهش داد.

دو روش اکتشافی متداول برای معمای۸

۱ –

تعداد کاشیها در مکانهای نادرست $= h_1$

در حالت شروع $h_1 = 8$

h_1 اکتشاف قابل قبولی است، زیرا هر کاشی که در جای نامناسبی قرار دارد، حداقل یکبار باید جابجا شود.

۲ –

مجموعه فواصل کاشیها از موقعیتهای هدف آنها $= h_2$

در حالت شروع

$$h_2 = 3+1+2+2+2+3+3+2 = 18$$

چون کاشیها نمیتوانند در امتداد قطر جا به جا شوند, فاصله ای که محاسبه میکنیم مجموع فواصل افقی و عمودی است. این فاصله را فاصله بلوک شهر یا فاصله مانهاتان مینامند.

h_2 قابل قبول است، زیرا هر جابجایی که میتواند انجام گیرد، به اندازه یک مرحله به هدف نزدیک میشود.

هیچ کدام از این برآوردها، هزینه واقعی راه حل نیست.

هزینه واقعی ۳۶ است.

اثر دقت اکتشاف بر کارایی

فاکتور انشعاب مؤثر b*

- اگر تعداد گره هایی که برای یک مسئله خاص توسط A* تولید میشود برابر با N و عمق راه حل برابر با d باشد، آن گاه b* **فاکتور انشعابی** است که درخت یکنواختی به عمق d باید داشته باشد تا حاوی N+۱ گره باشد.

$$N+1 = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d$$

- فاکتور انشعاب مؤثر معمولاً برای **مسئله های سخت ثابت** است.

- اندازه گیریهای تجربی b* بر روی مجموعه کوچکی از مسئله ها میتواند **راهنمای خوبی برای مفید بودن** اکتشاف باشد.
- مقدار b* در اکتشافی با طراحی خوب، نزدیک **۱** است.

اثر دقت اکتشاف بر کارایی

	Search Cost			Effective Branching Factor		
d	IDS	$A^*(h_1)$	$A^*(h_2)$	IDS	$A^*(h_1)$	$A^*(h_2)$
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	3644035	227	73	2.78	1.42	1.24
14	–	539	113	–	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

میانگین گره های بسط یافته در جستجویIDS و A* و فاکتور انشعاب مؤثر با استفاده ازh۱ و h۲.

اگر برای هر گره n داشته باشیم: h۲(n) >= h۱(n)

- h۲ بر h۱غالب است.
- غالب بودن مستقیما به کارایی ترجمه میشود.
- تعداد گره هایی که با بکارگیری h۲ بسط داده میشود، هرگز بیش از بکارگیری h۱ نیست.

همیشه بهتر است از تابع اکتشافی با مقادیر بزرگ استفاده کرد، به شرطی که زمان محاسبه اکتشاف، خیلی بزرگ نباشد.

الگوریتم های جست و جوی محلی و بهینه سازی

الگوریتم های قبلی، فضای جست و جو را به طور سیستماتیک بررسی میکنند

- تا رسیدن به هدف یک یا چند مسیر نگهداری میشوند
- مسیر رسیدن به هدف، راه حل مسئله را تشکیل میدهد

در الگوریتم های محلی مسیر رسیدن به هدف مهم نیست

- مثال: مسئله ۸ وزیر

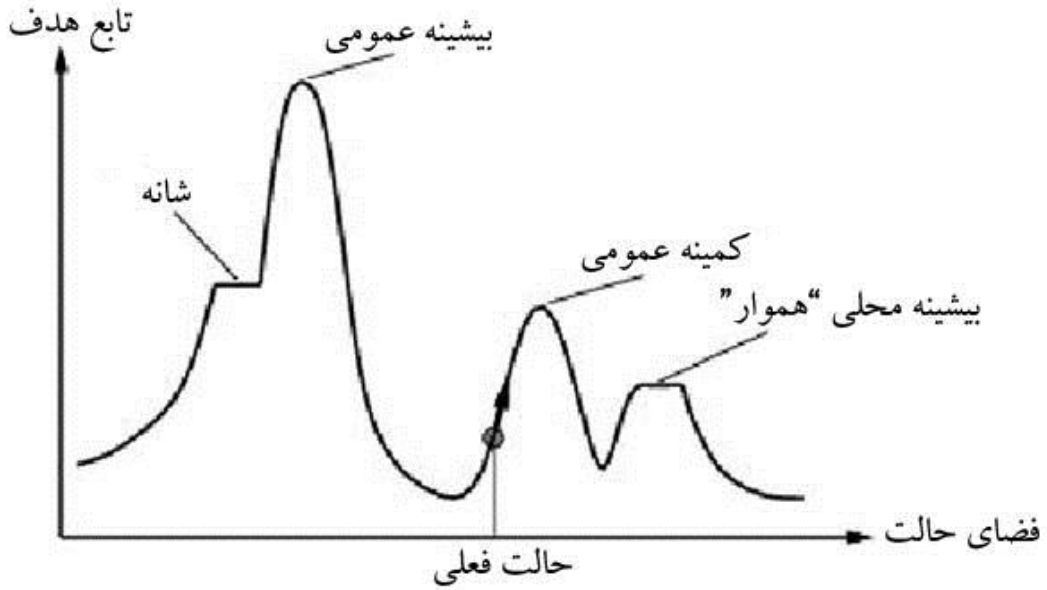
دو امتیاز عمده جست و جوهای محلی

- استفاده از حافظه کمکی
- ارائه راه حلهای منطقی در فضاهای بزرگ و نامتناهی

این الگوریتمها برای حل مسائل بهینه سازی نیز مفیدند

- یافتن بهترین حالت بر اساس تابع هدف

الگوریتم های جست و جوی محلی و بهینه سازی



جست و جوی تپه نوردی

حلقه ای که در جهت افزایش مقدار حرکت میکند(بطرف بالای تپه)

- رسیدن به بلندترین قله در همسایگی حالت فعلی، شرط خاتمه است.

ساختمان داده گره فعلی، فقط حالت و مقدار تابع هدف را نگه میدارد

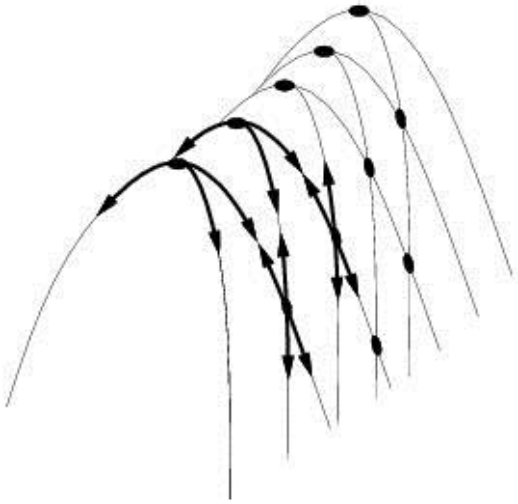
جست و جوی محلی حریصانه نیز نام دارد

- بدون فکر قبلی حالت همسایه خوبی را انتخاب میکند

تپه نوردی به دلایل زیر میتواند متوقف شود:

- بیشینه محلی

- برآمدگی ها
- فلات



انواع تپه نوردی:

- تپه نوردی غیرقطعی، تپه نوردی اولین انتخاب، تپه نوردی شروع مجدد تصادفی

مثال: مسئله ۸ وزیر

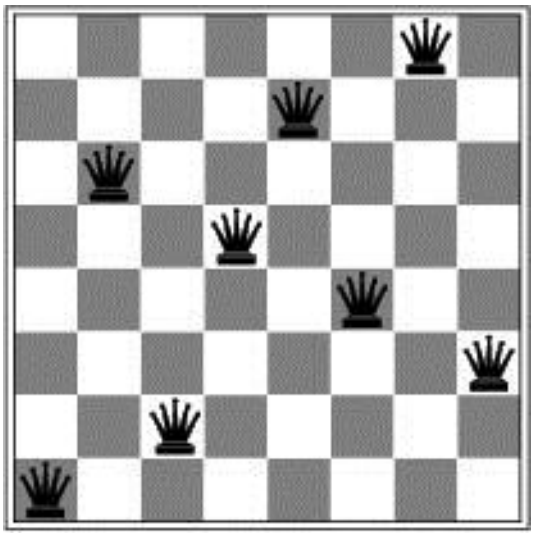
مسئله ۸ وزیر با استفاده از فرمولبندی حالت کامل

- در هر حالت ۸ وزیر در صفحه قرار دارند

تابع جانشین: انتقال یک وزیر به مربع دیگر در همان ستون

تابع اکتشاف: جفت وزیرهایی که نسبت به هم گارد دارند

- مستقیم یا غیر مستقیم



ب

الف

الف- حالت با هزینه h=۱۷ که مقدار h را برای هر جانشین نشان میدهد

ب- کمینه محلی در فضای حالت ۸ وزیر؛ h=۱

جست و جوی شبیه سازی حرارت

تپه نوردی مرکب با حرکت تصادفی

شبیه سازی حرارت: حرارت با درجه بالا و به تدریج سرد کردن

مقایسه با حرکت توپ

- توپ در فرود از تپه به عمیق ترین شکاف میرود
- با تکان دادن سطح توپ از بیشینه محلی خارج میشود
- با تکان شدید شروع(دمای زیاد)
- بتدریج تکان کاهش(به دمای پایین تر)

با کاهش زمانبندی دما به تدریج، الگوریتم یک بهینه عمومی را می یابد.

جست و جوی پرتو محلی

به جای یک حالت، k حالت را نگهداری میکند

- حالت اولیه: k حالت تصادفی
- گام بعد: جانشین همه k حالت تولید میشود
- اگر یکی از جانشین ها هدف بود، تمام میشود
- وگر نه بهترین جانشین را انتخاب کرده، تکرار میکند

تفاوت عمده با جستجوی شروع مجدد تصادفی

- در جست و جوی شروع مجدد تصادفی، هر فرایند مستقل از بقیه اجرا میشود
- در جست و جوی پرتو محلی، اطلاعات مفیدی بین k فرایند موازی مبادله میشود

جست و جوی پرتو غیرقطعی

- به جای انتخاب بهترین k از جانشینها، k جانشین تصادفی را بطوریکه احتمال انتخاب یکی تابعی صعودی از مقدارش باشد، انتخاب میکند.

در جلسه بعد به بررسی الگوریتم های ژنتیک میپردازیم.

پایان