فصل پنجم

جستجوهای رقابتی (games)

در محیط های چند عامله، هر عامل داده شده ای نیاز به در نظر گرفتن عمل های دیگر عامل ها و چگونگی اثر آنها بر خودش داشت. محیط های رقابتی که در آن هدف های عامل ها در تناقض قرار دارد، به مسائل جستجوی رقابتی منجر می شود که اغلب **بازی** نامیده می شود. وجود حریف در بازی باعث مشکل شدن تصمیم گیری و عدم قطعیت می شود.

نظریه بازی ریاضی، یک شاخه از علم اقتصاد است که هر محیط چند عامله را به عنوان بازی تجسم می کند و در آن هر عاملی حضور مؤثری بر دیگر عوامل چه به صورت همیاری و چه به صورت رقابتی دارد. در AI، بازی ها نوع خاصی از شکل کلی هستند که نظریه پردازان بازی آنها را قطعی، با تغییر نوبت، دو نفره، بازی های جمع- صفر می نامند. در مفاهیم مطرح شده ما، این به معنی محیط قطعی کاملاً قابل مشاهده است که در آن دو عامل وجود دارد که اعمال آنها می بایست تعویض شوند و در آن مقادیر سودمندی در انتهای بازی همواره برابر و برخلاف هم هستند. برای مثال، اگر یک بازیکن بازی شطرنج را ببرد (+۱) دیگری حتماً بازنده (-۱) است. این تقابل بین توابع سودمندی عامل هاست که موقعیت را رقابتی می سازد.

حالت یک بازی به سادگی قابل نمایش است و عامل ها معمولاً به تعداد عمل کمی محدود می شوند که خروجی آنها توسط قوانین دقیقی تعریف شده است. بازی های فیزیکی مثل کریکت یا هاکی روی یخ توصیف بسیار پیچیده تری دارند، مجموعه اعمال ممکن گسترده ای دارند، و قوانین ناکافی برای تعریف مجاز بودن اعمال در آنها وجود دارد.

بازی ها، برخلاف مسائل ابتدایی مطرح شده در فصل 2 برای حل بسیار دشوارند. برای مثال، بازی شطرنج فاکتور انشعاب متوسط در حدود ۳۵ گره دارد و بازی معمولاً تا ۵۰ حرکت برای هر بازیکن ادامه می

یابد، بنابراین درخت جستجوی آن در حدود ۳۵^۱۰۰ یا ۱۰^۱۵۴ گره دارد. اگر چه گراف جستجو در حدود ۱۰^۴۰ گره متمایز دارد. بازی ها، همانند دنیای واقعی هستند، بنابراین نیازمند توانایی ساختن برخی تصمیمات در زمانی دارند که محاسبه تصمیم بهینه امکان پذیر نیست. بازی ها همچنین به شدت در حالت عدم کارایی جریمه می شوند.

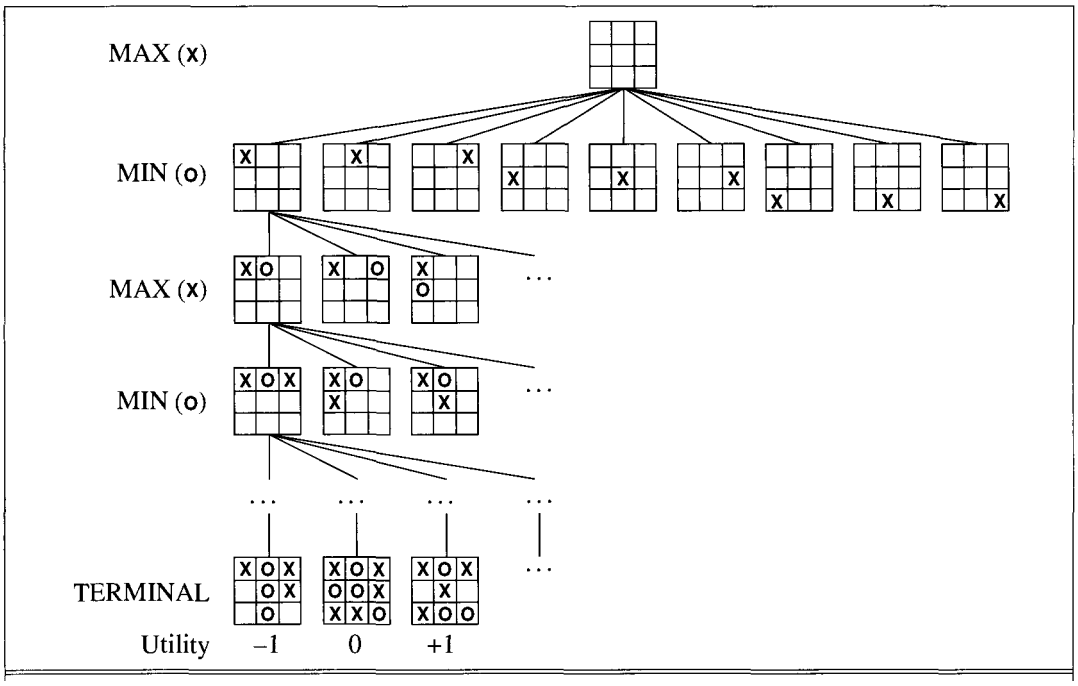
تصمیمات بهینه در بازی ها

بازی هایی را در نظر می گیریم که دو نفره اند و یکی را MAX و دیگری را MIN به دلایلی می نامیم که بعداً مشخص خواهد شد. MAX ابتدا حرکت می کند، سپس نوبت عوض شده تا بازی به پایان رسد. در پایان بازی، به بازیکن برنده جایزه داده می شود و جریمه ها به بازنده تعلق خواهد داشت. یک بازی می تواند به عنوان نوعی مسئلهٔ جستجو با مؤلفه های زیر تعریف شود:

- حالت آغازین که شامل موقعیت صفحه بازی و مشخص نمودن حرکت بازیکن است.
- تابع مابعد که لیستی از جفت (حرکت، حالت) بر می گرداند، و هر کدام شامل یک حرکت مجاز و حالت حاصل است.
- آزمون پایانی (Terminal Test) که مشخص می کند کی بازی تمام می شود. حالاتی که بازی در آنها تمام می شود، حالات پایانی نام دارند.
- تابع سودمندی (Utility Function) اغلب تابع هدف یا تابع امتیاز می نامند، که مقدار عددی برای حالات پایانی می دهد. در شطرنج خروجی برنده، مساوی با مقادیر ۱+ یا ۱- یا صفر است. برخی بازی ها خروجی های با بازهٔ وسیع تری دارند، مثلا امتیازات تختهٔ نرد از ۱۹۲+ تا ۱۹۲- است. این فصل عمدتاً به بازی های جمع- صفر می پردازد.

حالت اولیه و حرکات مجاز برای هر طرف **درخت بازی** را تعریف می کند. شکل ۵-۱ بخشی از درخت بازی تیک-تاک-تو (O-X) را نشان می دهد. از حالت اولیه، MAX نه حرکت مجاز دارد. بازی یک در میان بین جایگزین یک X توسط MAX و یک O توسط MIN ادامه یافته تا به گره های برگی برسیم که

معرف حالات پایانی هستند. حالت پایانی وقتی است که یک بازیکن سه مهره در یک سطر داشته باشد و یا تمامی خانه ها پر شده باشد. مقدار مشخص شده در هر گره برگی معرف مقدار سودمندی حالت پایانی از منظر MAX است. مقادیر بیشتر برای MAX خوب هستند ولی برای MIN بد خواهد بود. (این دلیل نام آنهاست) کار MAX استفاده از درخت جستجو (عملاً سودمندی حالات پایانی) برای تعیین بهترین حرکت است.



شکل ۵-۱ : درخت جستجوی(نیمه) برای بازی تیک-تاک-تو. گره بالایی حالت آغازین و MAX ابتدا حرکت می کند که X را در یک خانه خالی قرار می دهد. بخشی از درخت جستجو را نمایش داده ایم که حرکت مقابل MIN(O) و MAX را تا رسیدن به حالات پایانی نشان دهد و در این حالت پایانی می توان سودمندی ها را بر پایه قوانین بازی انتساب کرد.

استراتژی های بهینه

در یک مسئلهٔ جستجوی عادی، پاسخ دنباله ای از حرکات منجر به حالت هدف است، یعنی حالتی که منجر به برد می شود. در بازی، MIN چیزی برای گفتن دارد. از این رو MAX می بایست یک استراتژی احتمالی داشته باشد که حرکت آن را در حالت آغازین معین کند. سپس MAX در حالات حاصل از هر واکنش MIN نسبت به آن حرکات قرار گرفته و غیره. می توان گفت که استراتژی بهینه باعث بوجود آمدن

یک خروجی، حداقل به خوبی دیگر استراتژی ها خواهد بود، البته با فرض اینکه رقیب بازیکن حرفه ای باشد. ما از آنجا آغاز می کنیم که چگونه استراتژی بهینه را بیابیم حتی با آنکه محاسبه آن در بازی های پیچیده تر از تیک-تاک-تو امکان پذیر نباشد.

حتی بازی ساده ای مثل تیک-تاک-تو نیز درخت بازی بسیار مفصلی دارد. در شکل ۵-۲ بازی ساده ای را مشاهده می کنید. a_3, a_2, a_1 خروجی های MAX هستند. پاسخ های ممکن به a_1 برای MIN b_1, b_2, b_3 هستند و غیره.

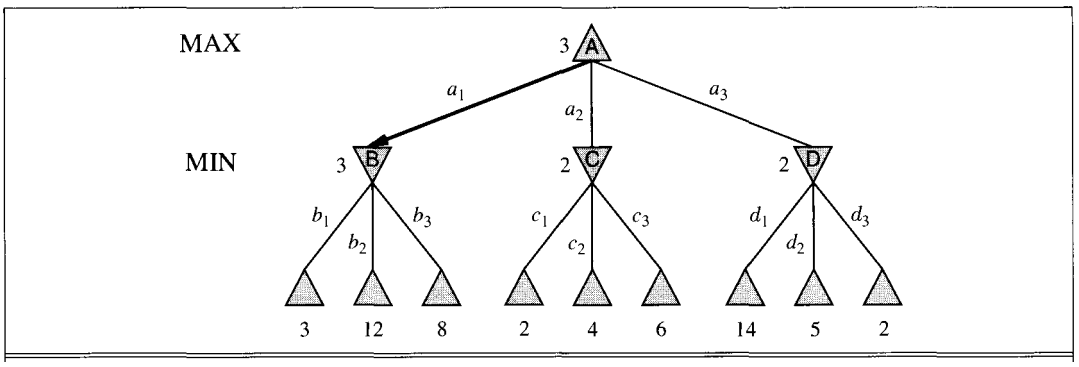
این بازی خاص پس از یک حرکت برای هر طرف MAX و MIN خاتمه می یابد. در اصطلاح بازی، می گوییم این درخت، یک حرکت، عمق دارد که شامل دو نیم حرکت است که هر کدام از آنها یک ply نام دارند. سودمندی حالات پایانی در بازهٔ 2 تا 14 است.

با درخت بازی داده شده، استراتژی بهینه می تواند با آزمون مقدار minimax هر گره تعیین شود، که آن را به صورت MiniMax-Value(n) می نویسیم. مقدار MiniMax یک گره، سودمندی برای MAX در حالت وابسته است، با فرض آنکه هر بازیکن از آنجا تا پایان بازی را بهینه بازی کنند. آشکارا، مقدار MiniMax حالت پایانی، سودمندی آن است. MAX ترجیح می دهد تا به حالت حداکثر مقدار حرکت کند، حال آنکه MIN ترجیح می دهد که به یک مقدار حداقل برسد. پس داریم:

$$\text{MiniMax-value(n)} = \begin{cases} \text{UTILITY}(n) & \text{if } n \text{ is a terminal state} \\ \max_{s \in Successors(n)} \text{MINIMAX-VALUE}(s) & \text{if } n \text{ is a MAX node} \\ \min_{s \in Successors(n)} \text{MINIMAX-VALUE}(s) & \text{if } n \text{ is a MIN node.} \end{cases}$$

اجازه دهید این تعاریف را به درخت بازی شکل ۵-۲ اعمال کنیم. گره های پایانی در سطح پایین با مقادیر سودمندیشان برچسب خورده اند. اولین گره MIN، برچسب B دارد که سه مابعد با مقادیر ۳، ۱۲ و ۸ دارد و بنابراین مقدار minimax آن ۳ خواهد بود. مشابهاً، دو گره MIN دیگر مقدار minimax ۲ دارند. گره ریشه از نوع MAX است و مقادیر minimax فرزندان آن ۳، ۲ و ۲ است، پس مقدار

minimax آن ۳ خواهد بود. می توانیم تصمیم minimax در ریشه را تعیین کنیم: عمل a_1 انتخاب بهینه برای MAX است چرا که منجر به مابعدی با بالاترین مقدار minimax خواهد شد.



شکل ۵-۲: درخت بازی دو حرکته. گره های $MAX\,\Delta$ هستند که در آن ابتدا MAX حرکت می کند و گره های $MIN\,\nabla$ هستند. گره های پایانی مقادیر سودمندی MAX را نشان می دهند و دیگر گره ها برحسب مقادیر minimax برچسب خورده اند. بهترین حرکت MAX در ریشهٔ a_1 است، زیرا منجر به مابعدی با بالاترین مقدار minimax می شود و بهترین پاسخ MIN b_1 است زیرا منجر به مابعدی با کمترین مقدار minimax خواهد شد.

الگوریتم minimax

الگوریتم minimax (شکل ۵-۳) تصمیم minimax را از حالت جاری محاسبه می کند. این محاسبه با استفاده از یک الگوریتم بازگشتی و مقادیر minimax هر حالت مابعد انجام می شود. بازگشتی تا سطح برگ های درخت پایین رفته و سپس مقادیر minimax همزمان با برگشتن از فراخوانی ها به بالا نسخه برداری می شود. برای مثال، در شکل ۵-۲ الگوریتم ابتدا به سه گره پایین- چپ به صورت بازگشتی می رسد و با کمک تابع Utility مقادیر این گره ها یعنی ۳، ۱۲ و ۸ کشف خواهد شد. سپس حداقل آنها یعنی ۳ بازگشته و در گره B کپی خواهد شد. فرآیندی مشابه برای مقادیر کپی شدهٔ ۲ برای C و ۲ برای D نیز صورت خواهد گرفت. نهایتاً، حداکثر ۳، ۲ و ۲ به عنوان مقدار بازگشتی برای ریشه که ۳ است انتخاب خواهد شد.

الگوریتم minimax جستجوی اول- عمق کاملی روی درخت بازی انجام می دهد. اگر عمق حداکثر درخت m باشد، و b حرکت مجاز در هر نقطه داشته باشیم، پیچیدگی زمانی الگوریتم $O(b^m)$ خواهد شد. پیچیدگی فضای مصرفی $O(bm)$ برای الگوریتمی است که تمامی مابعدها را یکجا تولید می کند و یا $O(m)$ برای الگوریتمی است که مابعدها را در هر مرحله تولید می کند. برای بازی های واقعی، هزینهٔ زمانی غیر عملی است، اما این الگوریتم پایهٔ تحلیل ریاضی بازی ها و الگوریتم های عملی تر است.

```
function MINIMAX-DECISION(state) returns an action
  inputs: state, current state in game

  v ← MAX-VALUE(state)
  return the action in SUCCESSORS(state) with value v

function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for a, s in SUCCESSORS(state) do
    v ← MAX(v, MIN-VALUE(s))
  return v

function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← ∞
  for a, s in SUCCESSORS(state) do
    v ← MIN(v, MAX-VALUE(s))
  return v
```

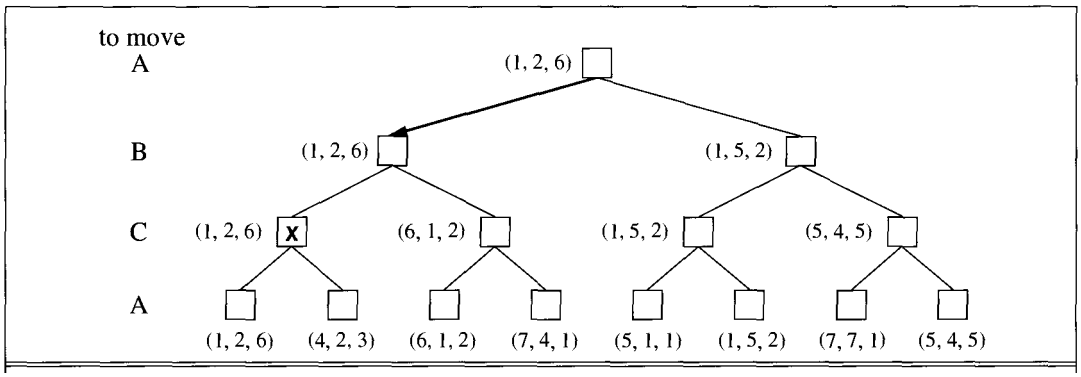
شکل ۵-۳: الگوریتم محاسبهٔ تصمیمات *minimax*. این الگوریتم عملی وابسته به بهترین حرکت ممکن بر می گرداند. یعنی حرکتی که خروجی آن بهترین سودمندی باشد و با این فرض که بازیکن رقیب سودمندی را حداقل می کند. توابع MAX_VALUE و MIN_VALUE از کل درخت بازی عبور می کند تا مقدار پشتیبانی حالت را تعیین کند.

تصمیمات بهینه در بازی های چند نفره

بسیاری از بازی های محبوب به بیش از دو بازیکن اجازهٔ بازی می دهند. بیایید ایدهٔ minimax را به بازی های چند نفره گسترش دهیم. این کار از جنبهٔ فنی سرراست است اما برخی مفاهیم جدید جالب را ایجاد می کند.

اول، نیازمندیم تا مقدار واحد برای هر گره را با بردار مقادیر جابه جا کنیم. برای مثال، در بازی سه نفره با بازیکنان A، B و C یک بردار $< V_A, V_B, V_C >$ به هر گره مربوط خواهد شد. برای حالات پایانی، این بردار سودمندی حالت را از منظر هر بازیکن می دهد. (در بازی دو نفره جمع- صفر، دو عنصر بردار می تواند به مقدار واحدی تقلیل یابد زیرا مقادیر همواره متضاد هستند). ساده ترین راه پیاده سازی این، داشتن تابع Utility است که یک بردار سودمندی را بر می گرداند.

حال به حالت غیرپایانی توجه کنید. گره مشخص شدهٔ X در درخت بازی شکل ۵-۴ را در نظر بگیرید. در این حالت، بازیکن C آنچه را می خواهد انتخاب خواهد کرد. دو شانس منجر به حالت پایانی بردارهای $< V_A = 1, V_B = 2, V_C = 6 >$ و $< V_A = 4, V_B = 2, V_C = 3 >$ را دارند. از آنجا که ۶ از ۳ بزرگتر است، C باید اولین حرکت را انتخاب کند. این بدان معنی است که اگر به حالت X برسیم بازی بعدی منجر به حالت پایانی با سودمندی $< V_A = 1, V_B = 2, V_C = 6 >$ خواهد شد. بنابراین این مقدار در X کپی شده است. در کل، مقدار کپی شدهٔ یک گره n، مقدار سودمندی هر مابعدی است که بالاترین مقدار را برای بازیکن انتخاب کننده در n دارد.



شکل ۵-۴: اولین سه حرکت درخت بازی با سه بازیکن (A,B,C). هر گره با مقادیری از دیدگاه هر بازیکن برچسب خورده است. بهترین در ریشه علامت گذاری شده است.

هر کس که بازی چند نفره همانند دیپلمات را تجربه کرده باشد، به سرعت متوجه خواهد شد نکات بیشتری از بازی دو نفره وجود دارد. بازی های چند نفره معمولاً شامل متحدینی است که یا رسمی و یا غیررسمی در بین بازیکن ها وجود دارند. اتحادها در طی فرآیند بازی ایجاد و شکسته می شود. برای مثال،

فرض کنید A,B در موقعیت ضعیفی نسبت به C هستند. تحت این شرایط اغلب راهکار بهینه برای A و B حمله به C تا به یکدیگر است چرا که C هر یک را به تنهایی نابود خواهد کرد. از این طریق، همکاری از رفتار صرفاً هر فرد تولید خواهد شد. البته هر زمان C بر اثر اتحاد ضعیف شد، اتحاد ارزش خود را از دست خواهد داد و A یا B آن را نقض خواهند کرد.

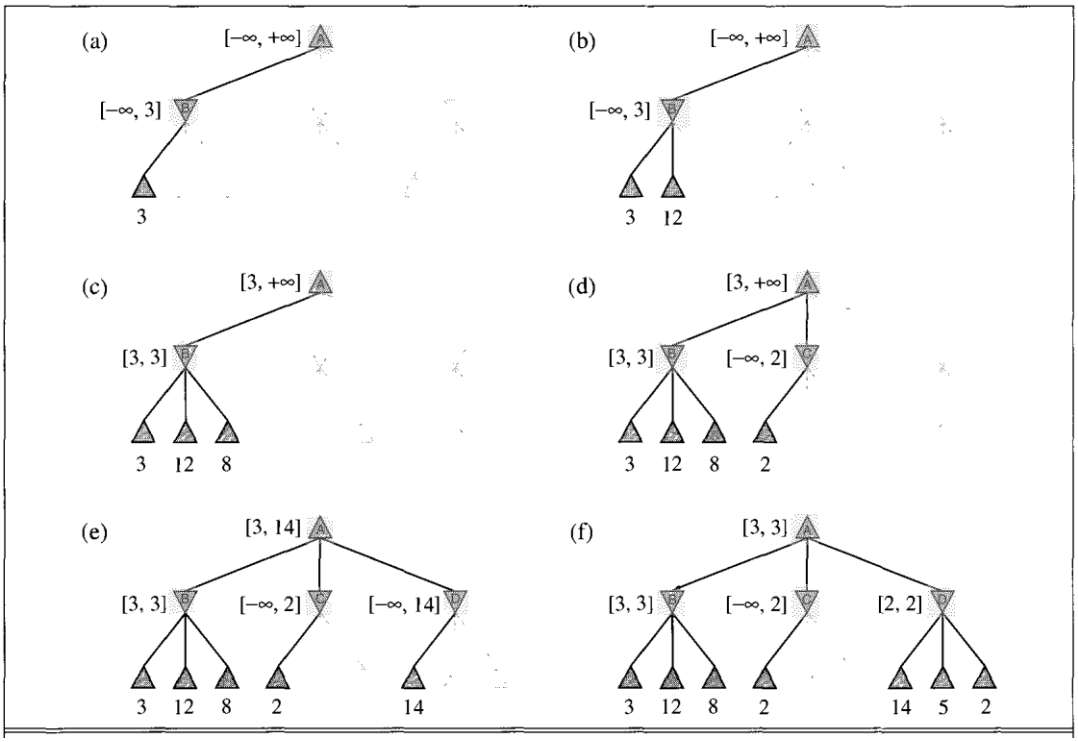
هرس آلفا- بتا $\alpha - \beta$ *Prunning*

مسئلهٔ موجود در جستجوی minimax تعداد حالات بازی است که بسته به تعداد حرکات نمایی خواهند بود. متأسفانه نمی توانیم نمایی بودن را حذف کنیم، اما می توانیم مؤثراً آن را نصف کنیم. حقه آن است که می توان تصمیم minimax صحیح را بدون بررسی هر گره در درخت بازی، محاسبه نمود. یعنی از ایدهٔ هرس سازی برای حذف بخش عمده درخت استفاده می کنیم. روش خاصی که در اینجا بررسی می شود، همان حرکتی را که minimax انتخاب می کند باز خواهد گرداند، اما شاخه هایی را که نمی توانند در تصمیم نهایی دخالتی داشته باشند هرس خواهند شد.

بازی دو حرکتی درخت بازی شکل ۵-۲ را در نظر بگیرید. بیایید محاسبات تصمیم بهینه را با دقت بیشتری بررسی کنیم و این بار توجه خود را معطوف هر نقطه در مراحل پیشرفت سازیم. مراحل در شکل ۵-۵ تشریح شده اند. خروجی آن چیزی است که به عنوان تصمیم minimax بدون محاسبهٔ دو برگ می توان مشخص کرد.

راه دیگر نگرش به این مطلب ساده کردن فرمول Minimax-Value است. بیایید دو مابعد ارزیابی نشده گرهٔ C در شکل ۵-۶ را با مقادیر x و y نشان دهیم و z را به عنوان حداقل X و Y در نظر بگیریم. مقدار گره ریشه از محاسبهٔ

$$\begin{aligned}
\text{MINIMAX-VALUE}(root) &= \max(\min(3,12,8), \min(2,x,y), \min(14,5,2)) \\
&= \max(3, \min(2,x,y), 2) \\
&= \max(3,z,2) \qquad \text{where } z \leq 2 \\
&= 3.
\end{aligned}$$



شکل ۵-۵: مراحل محاسبهٔ تصمیم بهینه برای درخت بازی در شکل ۵-۲. در هر نقطه، بازهٔ مقادیر ممکن برای هر گره را نشان می دهیم. (a) اولین برگ تحت B مقدار ۳ دارد. بنابراین B که یک گره Min است حداکثر مقدار ۳ به خود می گیرد. (b) برگ دوم تحت B مقدار ۱۲ دارد، Min از این حرکت اجتناب می کند، بنابراین مقدار B هنوز حداکثر ۳ است. (c) برگ سوم تحت B مقدار ۸ دارد، ما تمام مابعد B را دیده ایم، پس مقدار B دقیقاً ۳ است. حال می توانیم نتیجه گیریم که مقدار ریشه حداقل ۳ است، زیرا Max شانسی با مقدار ۳ در ریشه دارد. (d) اولین برگ C مقدار ۲ دارد. بنابراین C که یک گره Min است، حداکثر مقدار ۲ خواهد داشت. اما می دانیم که B ارزش ۳ دارد، بنابراین Max هیچ گاه C را انتخاب نخواهد کرد. به همین علت دلیلی برای بررسی گره های مابعد C وجود ندارد. این مثالی از هرس سازی آلفا- بتا است. (e) اولین برگ D مقدار ۱۴ دارد، پس حداکثر ارزش D، ۱۴ است. این بزرگتر از بهترین جایگزین برای Max است (یعنی ۳) پس باید به مابعدهای D بپردازیم. توجه کنید حال کران برای تمامی مابعدهای ریشه یافته ایم، پس ریشه حداکثر مقدار ۱۴ پیدا می کند. (f) دومین مابعد D ارزش ۵ دارد، پس باید بررسی را ادامه دهیم. سومین مابعد ارزش ۲ دارد پس D دقیقاً مقدار ۲ می یابد. تصمیم Max در ریشه حرکت به سوی B با مقدار داده شده ۳ است.

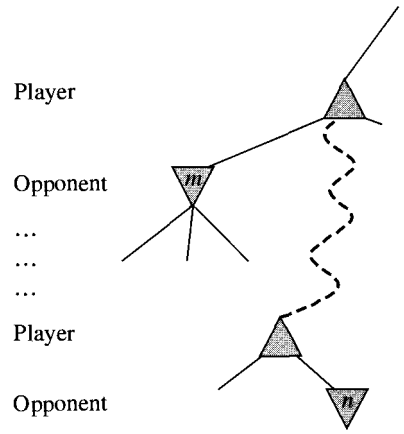
حاصل می شود. به عبارت دیگر مقدار ریشه و بنابراین تصمیمات minimax مستقل از مقادیر برگ های x و y هستند.

هرس ساز آلفا- بتا می تواند به درختی با هر عمقی اعمال شود و اغلب می توان کل زیر درخت را به جای برگ هرس کرد. اصل این است: گره n در جایی از درخت را در نظر بگیرید (شکل ۵-۶) در صورتیکه بازیکن شانس حرکت به آن گره را داشته باشد. اگر بازیکن شانس بهتری مثل m یا در گره والد n و یا هر نقطه ای شانس بالاتری را داشته باشد، n هیچ گاه در بازی واقعی قابل دسترسی نخواهد بود. بنابراین پس از آنکه به اندازهٔ کافی دربارهٔ n (از طریق آزمون اعقاب آن) دریافتیم تا به این نتیجه برسیم، آن را هرس خواهیم کرد.

یادآوری می کنیم که جستجوی minimax اول- عمق است. پس در هر زمان تنها گره های طول یک مسیر واحد در درخت را در نظر می گیریم. هرس سازی آلفا- بتا نام خود را از دو پارامتر توصیف کنندهٔ کران روی مقادیر پشتیبانی که هر جا در طول مسیر پیدا می شوند، گرفته اند:

α =مقدار بهترین(بزرگترین مقدار) انتخاب پیدا شده تاکنون، درهر نقطه انتخاب درطول مسیر برایMAX.

β =مقدار بهترین(کوچکترین مقدار) انتخاب پیدا شده تاکنون، درهر نقطه انتخاب درطول مسیر برایMIN.



شکل ۵-۶: هرس سازی آلفا- بتا: حالت کلی. اگر m از n برای بازیکن بهتر باشد، هیچ گاه در بازی به n نمی رویم.

جستجوی آلفا- بتا مقادیر αو β را در حین اعمال الگوریتم به هنگام نموده و شاخه های باقیمانده در یک گره را هر زمان که گره جاری بدتر از مقدار α و β گره MAX یا MIN شناخته شود، هرس خواهد کرد. الگوریتم کامل در شکل ۵-۷ داده شده است.

دو قانون برای هرس وجود دارد:

قانون اول: چنانچه مقدار آلفای یک گره از بتای پدرش یا پدرانش بیشتر شد نیازی به بررسی سایر فرزندان آن گره نیست. آن گره را هرس کنید.

قانون دوم: چنانچه بتای یک گره از آلفای پدرش یا پدرانش کمتر شد نیازی به بررسی آن گره نیست. آن گره را هرس کنید.

اگرچه که هرس آلفا- بتا باعث می شود که تنها بخشی از فضای حالت تا حالت پایانی توسعه یابد، اما این عمق هنوز هم عملی نیست. چون حرکات بازی باید در زمان منطقی انجام شوند. جستجو می بایست زودتر قطع شده و یک تابع ارزیابی اکتشافی در هنگام جستجو به حالتها اعمال شود تا از این طریق گره های غیرپایانی به صورت مؤثر گره پایانی تصور شوند. به بیان دیگر، پیشنهاد این است که minimax یا آلفا- بتا از دو طریق تغییر یابند:

۱. تابع سودمندی با یک تابع ارزیابی اکتشافی EVAL جایگزین شود که تخمینی از سودمندی وضعیت می دهد.

۲. آزمون پایانی با **آزمون برش** (cutofftest) جابه جا شود که تصمیم می گیرد چه زمان EVAL را اعمال کند.

```
function ALPHA-BETA-SEARCH(state) returns an action
  inputs: state, current state in game

  v ← MAX-VALUE(state, −∞, +∞)
  return the action in SUCCESSORS(state) with value v

---

function MAX-VALUE(state, α, β) returns a utility value
  inputs: state, current state in game
          α, the value of the best alternative for MAX along the path to state
          β, the value of the best alternative for MIN along the path to state

  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for a, s in SUCCESSORS(state) do
    v ← MAX(v, MIN-VALUE(s, α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v

---

function MIN-VALUE(state, α, β) returns a utility value
  inputs: state, current state in game
          α, the value of the best alternative for MAX along the path to state
          β, the value of the best alternative for MIN along the path to state

  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for a, s in SUCCESSORS(state) do
    v ← MIN(v, MAX-VALUE(s, α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

شکل ۵-۷: الگوریتم جستجوی آلفا- بتا. توجه کنید که روال ها همانند روال های MINIMAX در شکل ۵-۳ است، به جز برای دو خط در MIN-VALUE و MAX-VALUE که α و β را نگهداری می کند.

توابع ارزیابی

یک تابع ارزیابی تخمینی از سودمندی مورد انتظار بازی از موقعیت داده شده را باز می گرداند، درست همانند توابع اکتشافی فصل ۴ که تخمینی از فاصله تا هدف را برمی گردانند. کارآئی برنامه بازی وابسته به کیفیت تابع ارزیابی آن است. چگونه می توانیم یک تابع ارزیابی خوب طراحی کنیم؟ تابع ارزیابی باید:

۱) حالات پایانی را به گونه ای ترتیب بدهد که توابع سودمندی واقعی می کنند.

۲) محاسبه نباید بسیار طولانی شود.

۳) برای گره های غیرپایانی تابع ارزیابی باید با شانس برد مرتبط شود.

کسی ممکن است دربارهٔ عبارت "شانس برد" متعجب شود. مثلا بازی شطرنج شانسی نیست: ما دربارهٔ حالت جاری با قطعیت اطلاعات داریم و هیچ عامل تصادفی نیز مطرح نیست. اما اگر جستجو می بایست در حالات غیرپایانی متوقف شود، الگوریتم لزوماً دربارهٔ خروجی نهایی این حالت ها عدم قطعیت دارد.

بیایید این ایده را واقعی تر کنیم. بیشتر توابع ارزیابی از طریق محاسبهٔ ویژگی های مختلف حالت کار می کنند. این ویژگی ها کلاس های هم ارزی یا دسته های حالات متفاوت را تعریف می کنند. هر دستهٔ داده شده، دارای حالتهای منجر به برد، حالتهای منجر به مساوی و برخی حالتها که منجر به باخت می شوند، است. تابع ارزیابی نمی تواند بداند که کدام حالتها در آن وجود دارد، اما می تواند مقدار واحدی را برگرداند که بازتابی از نسبت حالتها با هر خروجی باشند. برای مثال، فرض کنید تجربهٔ ما پیشنهاد می کند ۷۲٪ حالات در یک دسته منجر به برد، (سودمندی ۱+)، ۲۰٪ منجر به باخت (۱-) و ۸٪ مساوی (۰) می شود. حال یک ارزیابی منطقی برای حالتها در این دسته می تواند متوسط وزن دار یا اصطلاحا **"مقدار مورد انتظار"** باشد. ۰/۵۲ = (۰×۰/۰۸)+((۱-)×۰/۲۰)+(۱×۰/۷۲) در کل، مقدار مورد انتظار می تواند برای هر دسته تعیین شود. در حالات پایانی، تابع ارزیابی نیازی به بازگرداندن مقادیر مورد انتظار واقعی ندارد.

در عمل، این نوع تحلیل نیازمند دسته های بسیار متنوع و بنابراین تجربیات بسیار برای تخمین تمامی حالات ممکن برای برد است. به جای آن بیشتر توابع ارزیابی عدد مجزای هر ویژگی را محاسبه کرده و آنها را ترکیب نموده تا مقدار کل حاصل شود. برای مثال، کتاب های مقدماتی شطرنج یک ارزش مادی تخمینی به هر مهره می دهند: هر پیاده ارزش ۱، یک اسب یا فیل ارزش ۳، یک رخ ارزش ۵ و وزیر ارزش ۹ دارند. دیگر ویژگی ها مثل "ساختار پیاده خوب" و "امنیت شاه" می تواند نصف یک پیاده ارزش داشته باشد. این مقادیر ویژگی ها با هم جمع شده تا ارزیابی وضعیت حاصل شود. از بعد ریاضی، این نوع تابع ارزیابی تابع خطی وزن دار نام دارد، زیرا می تواند به صورت:

$$EVAL(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) = \sum_{i=1}^{n} w_i f_i(s)$$

تعریف شود که در آن w_i وزن هر ویژگی f_i در صفحه است. برای شطرنج، f_i می تواند تعداد هر نوع مهره در صفحه و w_i مقادیر قطعات (پیاده ۱، فیل ۳ و غیره) باشند.

قطع جستجو

مرحلهٔ بعدی تغییر Alph-Beta-Search به گونه ای است که تابع اکتشافی EVAL را در زمان مناسب برای برش جستجو فراخوانی کند. از جنبهٔ پیاده سازی، دو خط در شکل ۵-۷ را که مبین Terminal-Test بود با این خط زیر عوض می کنیم:

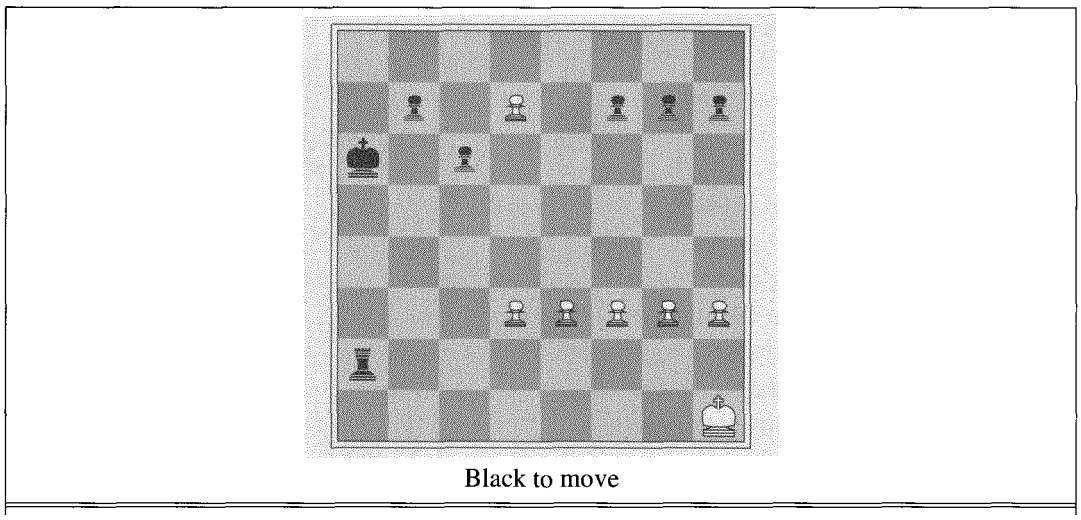
If Cutoff-Test (state,depth) then return Eval(state)

که عمق جاری با هر فراخوانی بازگشتی افزایش می یابد. سر راست ترین رهیافت کنترل مقدار جستجو، تعیین عمق ثابت است، بنابراین Cutoff-Test (state,depth)، برای تمامی عمق های بزرگتر از عمق ثابت d مقدار true برمی گرداند. (می بایست برای تمام حالات پایانی نیز همانند Terminal-Test مقدار true برگرداند). عمق d چنان انتخاب می شود که زمان مورد استفاده از قوانین بازی تجاوز نکند.

رهیافت استوارتر دیگر استفاده از عمیق شوندهٔ تکراری است که در فصل ۲ تعریف شد. مشخص است که ما نیاز به یک آزمون برش مناسب داریم. تابع ارزیابی می بایست تنها به موقعیت هایی که ساکن هستند اعمال شود، یعنی آنهایی که در آیندهٔ نزدیک مقدارشان به شدت تغییر نمی کند. در شطرنج، موقعیت های صفحه را نمی توان تنها با شمارش مهره ها در سکون ارزیابی کرد. موقعیت های ناساکن می توانند تا رسیدن به سکون گسترش یابند. این جستجوی اضافی جستجوی سکون نام دارد وگاهی شامل حرکات خاصی مثل زدن مهره هاست که منجر به عدم قطعیت در وضعیت خواهد شد.

اثر افق برای حذف دشوارتر است و آن زمانی روی می دهد که برنامه با حرکتی از رقیب رو به رو است که صدمه ای جدی می زند و تقریباً غیر قابل اجتناب است. بازی شطرنج در حالت شکل ۵-۸ در نظر بگیرید. سیاه در مهره ها جلوست، اما اگر سفید بتواند پیاده خود را از سطر هفتم با هشتم برساند، پیاده وزیر شده و برد ساده ای برای سفید مهیا می شود. سیاه می تواند تا ۱۴ حرکت از این کار با کیش دادن توسط رخ

جلوگیری می کند، اما پیاده در نهایت وزیر خواهد شد. مسئلهٔ با عمق ثابت آن است که تصور کند این حرکات تأخیری، جلوی وزیر شدن پیاده را می گیرد و ما می گوییم حرکات تأخیری وزیر شدن پیاده را «روی افق جستجو» حرکت داده تا قادر به شناسایی نشود.

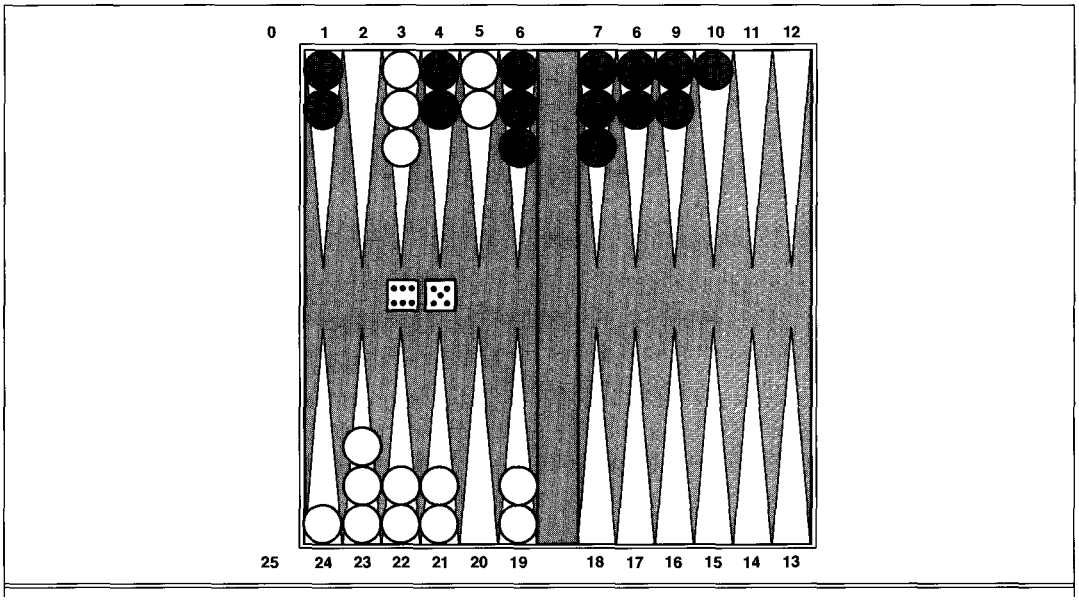


شکل ۵-۸: اثر افق. دنباله ای از کیش های رخ سیاه، حرکت ناگزیر وزیر شدن سفید را به "آن سوی افق" وادار می سازد و شرایط را شبیه برد سیاه در می آورد، در حالی که برندهٔ واقعی سفید است.

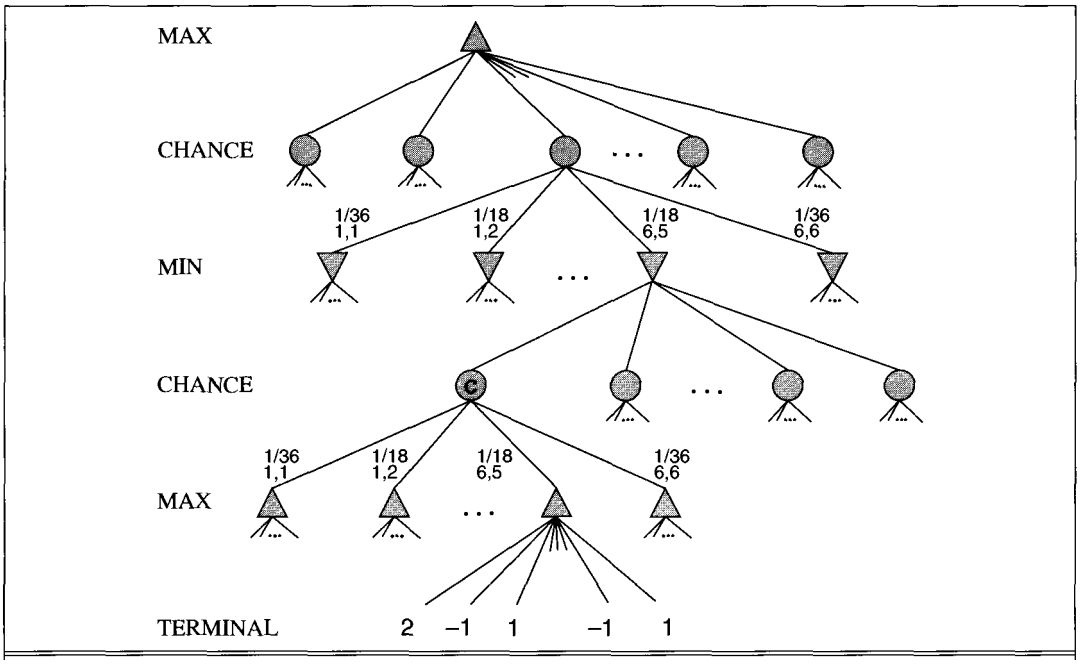
بازی هایی که شامل عامل شانس هستند

در زندگی واقعی وقایع خارجی متعدد غیرقابل پیش بینی وجود دارند که ما را به سوی وضعیت های تصور نشده می برند. بسیاری از بازی ها این غیرقابل پیش بینی بودن را از طریق افزودن عنصر تصادفی، همانند انداختن تاس دخیل می کنند. از این طریق، یک مرحله به واقعیت نزدیک تر می شوند و دیدن چگونگی این اثر در فرآیند تصمیم گیری با ارزش است.

تختهٔ نرد یک بازی نمونه است که شانس و مهارت را ترکیب کرده است. تاسها در ابتدای نوبت هر بازیکن انداخته می شوند تا حرکات مجاز را تعیین کنند. در صفحهٔ تختهٔ نرد شکل ۵-۹، برای مثال، سفید یک ۵-۶ آورده و بنابراین چهار حرکت ممکن دارد. اگرچه سفید می داند مجازش چیست، اما نمی داند سیاه چه تاسی می آورد و بنابراین حرکات مجاز سیاه را نمی داند. این بدان معنی است که سفید نمی تواند یک درخت بازی استاندارد از نوعی که در شطرنج و تیک-تا-تو دیدیم بسازد.



شکل ۵-۹: صفحهٔ نمونه بازی تخته نرد. هدف بازی خارج کردن تمامی مهره ها از صفحه است. حرکات سفید در جهت عقربه به سوی ۲۵ و حرکات سیاه در خلاف عقره به سوی صفر است. یک مهره می تواند به هر موقعیتی به جز جایی که چندین مهرهٔ رقیب در آن قرار دارند، برود. اگر یک رقیب باشد، کشته شده و باید مجدداً به چرخهٔ بازی برگردد. در صفحه نشان داده شده، سفید ۵-۶ آورده و باید بین چهار حرکت مجاز (۱۱-۵ و ۱۰-۵)، (۲۴-۱۹ و ۱۱-۵)، (۱۶-۱۰ و ۱۰-۵) و (۱۶-۱۱ و ۱۱-۵) انتخاب کند.



شکل ۵-۱۰: درخت بازی نموداری برای صفحهٔ تخته نرد

درخت بازی در تختهٔ نرد می بایست شامل گره های شانس به علاوه گره های MAX و MIN شود. گره های **شانس** به صورت دایره در شکل ۵-۱۰ نمایش داده شده اند. انشعابهایی که از هر گره شانس خارج می شوند مبین تاسهای ممکن هستند و هر کدام بر پایهٔ تاس مورد نظر و شانس موفقیت برچسب خورده اند. آنها ۳۶ راه برای انداختن دو تاس دارند که هر کدام شانس یکسانی دارند ولی چون ۵-۶ همان ۶-۵ است، پس تنها ۲۱ حالت متمایز وجود دارد. شش جفت (۱-۱ تا ۶-۶) شانس $\frac{1}{36}$ و دیگر ۱۵ حالت متمایز هر کدام $\frac{1}{18}$ شانس دارند.

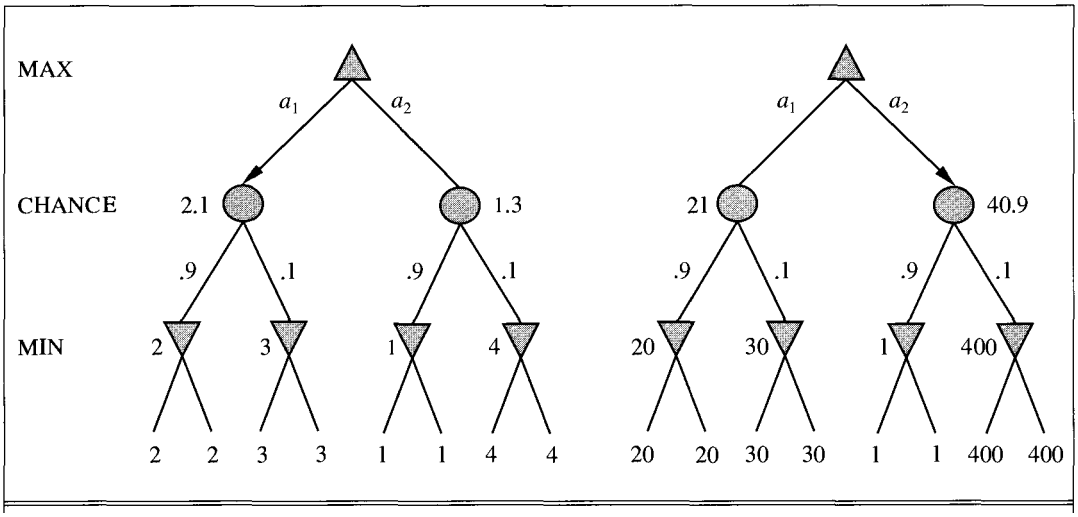
مرحلهٔ بعد تفهیم ساختن تصمیم درست است. به وضوح، می خواهیم حرکتی را انتخاب کنیم که به بهترین وضعیت منجر شود. چون وضعیت های حاصل مقادیر minimax تعریف شده ای ندارد، به جای آن، می توانیم فقط مقدار مورد انتظار یا "امید" را روی تمامی تاس های ممکن محاسبه کنیم. مقدار minimax برای بازی های قطعی مثل مقدار expectiminimax برای بازی های تصادفی می شود. گره های پایانی MAX و یا MIN هستند و همانند قبل کار می کنند. گره های شانس برحسب در نظر گرفتن وزن متوسط مقادیر حاصل از تمامی تاس های ممکنه ارزیابی می شود، یعنی:

$$\text{EXPECTIMINIMAX}(n) = \begin{cases} \text{UTILITY}(n) & \text{if } n \text{ is a terminal state} \\ \max_{s \in Successors(n)} \text{EXPECTIMINIMAX}(s) & \text{if } n \text{ is a MAX node} \\ \min_{s \in Successors(n)} \text{EXPECTIMINIMAX}(s) & \text{if } n \text{ is a MIN node} \\ \sum_{s \in Successors(n)} P(s) \cdot \text{EXPECTIMINIMAX}(s) & \text{if } n \text{ is a chance node} \end{cases}$$

که تابع مابعد برای گره شانس n، به سادگی حالت n را با هر تاس ممکن مابعدهای s متصل نموده و P(s) احتمال روی دادن تاس است. این روابط به صورت بازگشتی نسخه برداری شده تا به ریشه برسد و این درست همانند minimax است.

ارزیابی وضعیت بازی در گره های تصادفی

همانند minimax، تخمین سازی با expectiminimax_شامل قطع جستجو در نقطه ای و اعمال تابع ارزیابی به هر برگ است. در واقع وجود عامل شانس به معنی آن است که می بایست دربارهٔ معنی مقادیر ارزیابی دقت بیشتری کنیم. شکل ۵-۱۱ چنین رویدادی را نشان می دهد: با تابع ارزیابی که به برگ ها مقادیر [۴ و ۳ و ۲ و ۱] نسبت می دهد، حرکت A_1 بهترین است، با مقادیر [۴۰۰ و ۳۰ و ۲۰ و ۱] حرکت A_2 بهترین است. بنابراین برنامه در صورت تغییر در نسبت مقادیر ارزیابی رفتار متفاوتی بروز می دهد؛ پس آشکار می شود برای جلوگیری از این حساسیت تابع ارزیابی باید یک تبدیل خطی احتمال برد از یک وضعیت بازی را داشته باشد (یا کلی تر آنکه امید سودمندی وضعیت بازی را بیابیم). این یک خاصیت مهم و عمومی وضعیت ها در صورت افزودن عدم قطعیت است.



شکل ۵-۱۱: تبدیل حفظ کنندهٔ ترتیب روی مقادیر برگ ها بهترین حرکت را عوض می کند.

بازی های کارت

بازی های کارت به دلایل متعددی به دلیل ارتباط آنها با قمار مورد توجه هستند. در بین طیف گستردهٔ بازی ها، ما روی آنها یی تمرکز می کنیم که کارت ها در ابتدای بازی تصادفی پخش می شوند و هر بازیگر یک دست کارت را که برای دیگری قابل مشاهده نیست دریافت می کند. چنین بازی هایی شامل بریج، حکم و برخی انواع پوکر است.

در اولین نگاه، به نظر می رسد بازی های کارت همانند بازی های تاس هستند: کارت ها تصادفی پخش می شوند و حرکات برای هر بازیکنی معین است، اما تمامی تاس ها همان اول ریخته می شود! این نکته در عمل بسیار مفید است و از طرفی به دلایلی جالب کاملاً اشتباه است.

دو بازیکن MIN و MAX را در نظر بگیرید که چند دست گرمی بریج دو دستی چهار کارتی با همه کارت ها روباز، بازی می کنند. دست ها به صورت زیر است و اول MAX حرکت می کند:

MAX: ۶ دل، ۶ خشت، ۸ و ۹ خاج.

MIN: ۴ دل، ۲ پیک، ۵ و ۱۰ خاج

فرض کنید MAX می خواهد به ۹ خاج برسد. MIN می بایست یا ۱۰ خاج و یا ۵ خاج را بازی کند. MIN ۱۰ خاج را بازی می کند و برندهٔ این حقه است. MIN سپس حرکت کرده و به ۲ پیک می رسد. MAX چیزی برای رو کردن ندارد و باید کارتی را دور اندازد. انتخاب واضح ۶ خشت است زیرا دو کارت باقیمانده برنده هستند. حال هر کارتی MIN برای حقهٔ بعدی رو کند، MAX هر دو حقه را برنده و در دو حقه هر کدام گیر می کند. ساده است نشان دهیم، با نسخهٔ تغییر یافتهٔ مناسب minimax انتخاب ۹ خاج MAX بهینه است.

حال دست MIN را عوض کرده و به جای ۴ دل، ۴ خشت می گذاریم.

MAX: ۶ دل، ۶ خشت، ۸ و ۹ خاج.

MIN: ۴ خشت، ۲ پیک، ۵ و ۱۰ خاج.

دو حالت کاملاً متقارن هستند: بازی یکسان است جز آنکه در دومین حقهٔ MAX کارت ۶ دل را دور می اندازد. دوباره بازی با دو حقهٔ هر طرف گیر کرده و انتخاب ۹ خاج بهینه خواهد بود.

تاکنون همه چیز خوب بود. حال یکی از کارت های MIN را مخفی می کنیم: MAX می داند که MIN یا اولین دست (با ۴ دل) و یا دومین دست (با ۴ خشت) را دارد، اما ایده ای از اینکه کدام است، ندارد. MAX به صورت زیر استدلال می کند:

۹ خاج انتخاب بهینه اولین و دومین دست علیه MIN است، پس حالا هم باید بهینه باشد، زیرا می دانم که MIN یکی از این دو دست را دارد.

کلی تر، MAX از این ایده استفاده می کند که برای ارزیابی یک مجموعهٔ عملیات که شامل کارت های نادیدنی است، اول مقدار minimax آن عمل برای هر کارت ممکنی حساب شده و سپس مقدار امید روی تمامی حالات به کمک احتمال هر کارتی حساب خواهد شد. اگر فکر می کنید این منطقی است (یا حتی اگر بریج بلد نیستید) داستان زیر را مرور کنید:

روز ۱: جادهٔ A منتهی به تپهٔ طلا می شود، جادهٔ B به دو راهی می رسد. راه سمت چپ را بگیر و یک تپهٔ کوچک جواهرات را پیدا کن، اما اگر راه سمت راست را بروی با اتوبوس خارج می شوی.

روز ۲: جادهٔ A به تپهٔ طلا منتهی می شود، جادهٔ B به دو راهی می رسد. اگر سمت راست بر وی به تپهٔ کوچک جواهرات می رسی و اگر راه سمت چپ را بر وی با اتوبوس خارج می شوی.

روز ۳: جادهٔ A به تپهٔ طلا می رسد، جادهٔ B به دو راهی می رسد، درست حدس بزن و تپهٔ کوچک جواهرات را پیدا کن، اما حدس غلط تو را با اتوبوس به بیرون می برد.

به وضوح انتخاب جادهٔ B در دو روز اول منطقی نیست. هیچ کس در روز سوم جادهٔ B را انتخاب نمی کند. این همان چیزی است که متوسط گیری توصیه می کند: جادهٔ B در وضعیت های روز ۱ و ۲ بهینه است، پس برای روز ۳ نیز خواهد بود.

درسی که از این بحث گرفته شد، آن بود که وقتی اطلاعات نقصی دارد، می بایست به هر اطلاعاتی که در هر نقطه بازی کسب می شود توجه کرد. اشکال الگوریتم MAX آن است که تصور می کند در هر حرکت بازی همانند زمانی که تمامی کارت ها رو هستند جلو می رود. همان طور که مثال نشان داد، MAX مانند کسی عمل می کند که تمامی عدم قطعیت آیندهٔ خود را در همان ابتدای کار می خواهد حل کند. الگوریتم MAX همچنین هرگز تصمیم به جمع آوری اطلاعات نمی گیرد (یا اطلاعات برای شریک تهیه نمی کند)، زیرا در هر انتخاب کارت نیازی به آن ندارد. این انواع رفتارها می تواند توسط الگوریتم بهینه ای با اطلاعات ناقص به طور خودکار تولید شود. چنین الگوریتمی در فضای حالات دنیا جستجو نمی کند

(دست های بازی) بلکه در فضای باور (باور به اینکه کی کدام کارت را دارد و با کدام احتمال) جستجو می کند. در این فصل یک نکتهٔ نهایی و بسیار مهم را در پایان اشاره می کنیم: در بازی هایی با اطلاعات ناقص، بهتر است حداقل ممکن اطلاعات را به رقیب بدهیم و بهترین راه برای حصول این هدف عملکرد غیر قابل پیش بینی است. این همان علتی است که بازرسان رستوران ها سر زده کنترل انجام می دهند.