Tetris
(решение)

Задачата Тетрис предостави на състезателите възможност да се сблъскат с по-нестандартна задача – такава, за която няма известно полиномиално решение. Те трябваше да напишат изкуствен интелект, който играе известната игра Тетрис.

Като цяло правилата бяха почти същите като на оригиналната игра, с разликата, че играчът знае предварително ВСИЧКИ блокчета, а не само следващото, както и при оценяването (което беше направено с цел да се опрости и без друго дългото и сложно условие).

Първото нещо, което състезателите със сигурност са забелязали е, че не може да се напише „кратко" решение за повече от 20 точки. Наистина, да хванем тестовете с по 20 фигури е много лесно – всяка от фигурите има ротация, при която е широка най-много 2 колони. Това ни позволява да разделим дъската по колони на 5 части. Във всяка от тях имаме по 20 реда, като най-дългата фигура заема 4. Тоест във всяка част можем да сложим поне по 5 фигури. Това означава, че имаме място за поне 25 фигури.

Ако искаме да хванем повече точки, обаче, трябва да напишем значително по-сложно и дълго решение. Най-малкото трябва да напишем функции, които да се справят с падането на фигурите и последвалите изтривания. За щастие, състезателите можеха да ползват дадения във визуализатора код за това, което спестява значително писането (но си има и своите недостатъци, което ще коментираме малко по-късно).

Използвайки кода за слагане на фигурите съществуват няколко прости решения, които биха могли да хванат немалко точки (между 40 и 60).

За всеки от следващите разгледани подхода ще ни е нужна „оценъчна" (евристична) функция, която ни казва колко добра е дадена дъска. Трите най-очевидни неща кога дъска е „лоша" са:

1. Да има „дупки", тоест полета, в които няма блокче, но над тях има сложени други блокчета. Дупките пречат на премахванията на редове (което е основната идея на играта) и са много нежелани.
2. Да има твърде много блокчета – това означава, че дъската е почти пълна и няма много място за нови фигури.
3. Да има много блокчета на високо – това означава, че някои части от дъската са почти запълнени, докато други могат да са почти празни.

Разбира се, в зависимост от стратегията на изкуствения интелект, оценката на дъската може да се определя по други начини.

Аз лично използвах следната схема за оценяване в моите решения: всяка дупка допринася с по 10 точки към оценката, а всяко блокче – с височината, на която се намира. Колкото по-малка е оценката на дадена дъска, толкова по-добре.

След като сме определили тази евристична функция (която обикновено се нарича фукнция на финес, или fitness()), нека разгледаме няколко различни подхода.

Greedy
За всяка фигура пробваме всяка ротация и всяка позиция (до 4 ротации и 10 позиции, тоест само 40 възможности за фигура!). Избираме тази от тях, която води до „най-хубава" нова дъска. Този метод е много лесен за писане, много бърз, иии... сравнително неефективен. Въпреки това би хванал доста повече от 20 точки (около 35, до 50 ако е имплементирано добре).

Backtrack
Същото като грийдито, но пробваме не само всички възможни слагания на следващата фигура, а и на следващите няколко след нея. На всяко ниво броят конфигурации, които трябва да разглеждаме, расте експоненциално (40 на степен броя нива), така че не можем да си позволим много дълбока рекурсия. В интерес на истината, не можем да си позволим повече от 2 нива (ползвайки кода от визуализатора)! Въпреки това този метод води до значително подобрение в точките (около 50-60).

Backtrack with pruning
Някои от клоните на рекурсията очевидно не водят към нищо добро (тоест може и да водят, но не и в обозримо близкото бъдеще, което можем да изчерпим). Следователно можем да ги „изрежем" и да не ги разглеждаме. Това може да доведе до значително подобрение откъм време, но също така и да не намери оптималният отговор. Една от възможните стратегии е да ползваме Алфа-Бета отсичане или някое по-радикално рязане, които биха довели точките ни до... около 60. Хм...

Защо? Ами просто 2 нива не са достатъчно за адекватно подрязване на дървото. Затова трябва да забързаме алгоритъма си по някакъв начин.

Най-очевидната оптимизация (която доста хора сигурно са се сетили) е, че не всяка фигура трябва да бъде ротирана 4 пъти за да стигне отново до себе си. Например квадратчето си остава същото колкото и пъти да го ротираме. Така за седемте фигури максималният брой смислени ротации е: 2, 4, 1, 4, 2, 2, 4. Така вместо 40 продължения имаме, средно, 19 * 10 / 7 = 27.

Друга очевидна, но много по-сложна оптимизация е да забележем адски тъпия код, който е даден във визуализатора. И макар той да е

съвсем достатъчен за проверка на резултата или за визуализация, той би бавил ужасно много решение, базирано на изчерпване.

Състезателите, борещи се за 100 точки, трябваше да пренапишат тази част (както се наложи да направя и аз, в авторското решение). Аз лично ползвах почти същия алгоритъм, само че представяйки дъската чрез 10 int-а (всяка колона като integer число, а всяко квадратче в колоната - като бит в числото). Така местенията (а и много други операции) могат да станат много много по-бързи, ползвайки битови операции. Примерно за да намерим всички редове, които са пълни (са за изтриване) са ни нужни едва 10 and-а на integer-и, вместо (до) 200 индексирания в масив.

За съжаление тази оптимизация не е „за всеки" – погледнато отстрани от неопитен състезател (че и професионален програмист - несъстезател) би изглеждало като „черни магии".

Имаше и две не толкова очевидни оптимизации, които също доведоха до голямото подобрение. Първата от тях беше, че фигурите може да падат директно, а не стъпка по стъпка. Другата беше, че по-оптимална последователност (макар и с малко дублиране на код) в стъпките на алгоритъма на поставяне е: първо сваляме фигурата (а не я разглеждаме като компонента), после изтриваме пълните редове, и само ако е имало такива разглеждаме падащи компоненти.

С тези оптимизации се постигат учудващи резултати – при мен те бяха в размер на 50 пъти подобрение на скоростта. С тях чистият бектрек може да хване с около 10-20 точки повече, докато решението с Алфа-Бета отсичане може да хване 100.

Въпреки това, моето решение не беше базирано на тази техника. Вместо това ползвах един по-екзотичен алгоритъм (поне по нашите състезания):

Simulated Annealing
Идеята е търсенето ни да стане под формата на BFS, а не DFS, като слагаме само някои от възможните продължения в опашката за следващото ниво. Тоест отново правим търсене на няколко нива, само че на всяко от тях избираме някакъв брой продължения, носещи (на текущото ниво) най-оптимален отговор. После продължаваме всяко от тях на следващото ниво. Така нашето решение има нещо като „пипала" (тези локално-оптимални продължения), които „опипват" как би се справило продължение във всяка от възможните посоки. След това отиваме само там, където ни е харесало (тоест дъската изглежда обещаващо).

Малко пояснение какво точно правим. На първото ниво пробваме всички (средно) 27 продължения. На второто отново пробваме всички 27. Това

прави 729 възможности. Оценяваме всяка от тях и елиминираме всички без най-добрите K (в моя случай K беше 50). На третото ниво строим всички продължения от K-те възможности от предходното (тоест 50 * 27 = 1350 възможности). От тези 1350 отново избираме само K. На четвъртото ниво отново получаваме 1350 възможни продължения и т.н.

Забележете, че така броят разглеждани ситуации не расте експоненциално, а линейно по броя на нивата. Така можем да си позволим да търсим 5, дори 10 нива навътре. Е, разбира се рискуваме да не намерим оптималното решение, но какво пък – ако намерим достатъчно близко до оптималното можем да сме почти сигурни, че няма да достигнем твърде лоша дъска и да загубим играта.

Резултати

Финалната версия на решението ми спечели всички игри с 10000 фигури, които пробвах (а пробвах доста). Времето за изпълнение беше между 2 и 4 секунди (на моя лаптоп). За експеримент реших да пробвам как ще се представи ако броят на фигурите е 100000. И при този тест решението успя да оцелее до края, като играта завърши за малко над половин минута (36 секунди). Извод: рано или късно хората ще станем излишни. Scary!

Автор: Александър Георгиев