サンプル調査(greedy/c++)

　田中 健人

サンプルとして配布されている貪欲法で探索するAIのプログラムを理解するためにサンプルのソースについてまとめる。

このサンプルAIはraceState.hpp、raceState.cpp、greedy.cppの3つからソースから構成される。

* raceState.hpp、raceState.cpp

このアルゴリズムに限らず、ゲームAIとして必要な処理が記述してあり、他のc++で作成されたAIでも用いる。

詳細は下記、リンク先ドキュメント参照。<https://docs.google.com/document/d/1dvS6SH_FYOIL7SkKsJx4g5cwvWWoGLRGVwNb3pl9TJE/edit?usp=sharing>

* greedy.cpp  
  貪欲法とういうアルゴリズム名はあいまいな定義なので、このサンプルのアルゴリズムを詳細に、読み解いて、アルゴリズムとしてまとめる。  
  まずは、各メソッドでどのような処理を行っているかを記述する。  
  <mainメソッド>  
  mainはraceStateを使って、お決まりの処理を行う(送られてくる情報を各変数に格納)。  
  <PlayerState構造体>  
  Point型とIntvec型の変数をもち、operator<で、この構造体を比較可能なようにしてある。  
  このプログラムでは、自分馬のポジションと速度を格納するための構造体。  
  <Candidate構造体>  
  int型、Playstate型、IntVec型の変数step,state,howと、Candidateのポインタ\*fromをもつ。  
  このプログラムでは、候補を格納し、キューとして各候補を管理している。  
  stepはこの候補の状態に行くまでに必要なstep数、stateはこの候補がどの時に適応されるかの自分馬の「速度、ポジション」情報。  
  howにこの候補の加速度を、\*fromにこの候補に達する1つ前の候補を指すポインタを格納。  
  <playメソッド>  
    
  ①下準備  
  Candidateを格納するキューcandidatesを用意。  
  PlayerStateとCandidateを対応付けしたmapであるreachedを用意。  
  今の自分馬のポジション、速度を格納したPlayerState構造体initialを用意。  
  step=0(この候補は今の状態),state=initial,\*fromは空で、howは(0,0)の候補initialCandを用意。  
  reached(initial)はinitialCandとなるように対応づける。  
  最も良い候補を格納するポインタ\*bestを用意し、とりあえず今あるinitialCandを指し示す。  
  double型の変数goalTimeを用意し、とりあえず、このコンピュータ上でとりうる最もでかいdouble型の最大値を格納。  
  キューcandidatesにinitialCandを入れる。  
    
    
    
  ②メイン処理(全ての候補がキューからなくなるまで繰り返される。)  
  キューcandidatesの一番上にある候補を指し示すポインタc\*を用意、その候補はキューから捨てる。  
  加速度の全パターン(y方向、x方向ともに-1,0,+1のどれか)調べるために、for文で2重ループ(yは+1から優先的にcayに格納、その中でxは-1から優先的にcaxに格納)次の処理を行う。  
   [繰り返し]   
   候補cに格納しているstateの速度に、cay、caxを足してnextVeloに格納。  
   stateのポジションにnextVeloを足したポジションをnextPosに格納。  
     
   候補cのstepが0なら、敵の情報が分かるので、敵と交差するかチェック。  
   さらに、この加速度での障害物をチェック。  
   もし、どちらも問題なければPlayerState型のnextをnextVeloとnextPosで  
   用意。  
   cのstep+1、next、c、(cax,cay)でCandidate型のポインタ\*nextCandを用意。  
   [nextPosのyがゴールに到達していれば]  
   cのステップ+  
   (「コースの長さ」-「cのstateのy軸ポジション」)/(nextveloのy)  
   をtに格納。↑これはゴールに到達するまでにかかるステップ数の式

tがgoaltimeより小さければこれが調べた中で  
 nextCand一番早い候補なので、bestに格納し、  
 goaltimeにtを格納。  
   
 [それ以外の時、reachedを見て、next対応された候補が0であれば]  
 reched(next)がnextCandとなるように対応づける。   
  
 cのstepが定数searchDepth以下、かつ、nextPostのyよりコースの  
 長さが大きければ、キューcandidatesにnextCandを入れる。  
   
 nextPos.yがbestのstateのyより大きければ、bestにnextCandを

入れる。

③後仕上げ処理  
 best == &initialCandならば、(←どんな加速度でも動けなかった場合)  
 x軸、y軸ともに、ブレーキをかける。

bestの候補について、fromをたどって、&initialCandの1つの次まで戻り、  
 その候補の加速度を返す。

ということで、このサンプルは定数searchDepth(デフォルトでは7)ステップ先までをすべての加速度で試してみて、最もゴールに早く到達できるor最もゴールに近づける選択肢を選んでいく。

当然、アルファベータより強いがより時間がかかる。よね?

だけど、sample-courseではPlayer1として始めるか、player2として始めるかで勝敗が決まる。

貪欲法を使いたいけど、時間の関係で、アルファベータで枝刈りしてってことか。

実際、アルファベータのプログラムでは枝刈りして、その中で貪欲法を使ってる。よね？

まあ、ちょっと一回、各サンプルが使用する考慮じかんなどは、別ドキュメントにまとめます。