　　　　　　　　　　サンプル調査(pseudo\_alphabeta/c++)

　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　山内 拓磨

　田中 健人

サンプルとして配布されているアルファ・ベータ法で探索する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、pseudo\_alphabet.cppの3つのソースから構成される。

raceState.hpp、raceState.cppは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。実際にc++でAIを開発する場合も、そのまま利用できる。

* raceState.hpp

主にraceState.cppで利用するヘッダーファイル。共通の構造体などをまとめている。

* + IntVec

ベクトルを表現する構造体。

オペレータがいろいろと定義されている。

“<”では比較するIntVecよりyの値が小さければtrueを返す。

yの値が同じでxの値が小さければtrueを返す。

* + LineSegment

線分を表現する構造体。

goseThru()ではその線分がある点を通過するかどうかを判定する。

通過する場合がtrue。

intersects()ではある線分と交わるかどうかを判定する。

交わる場合がtrue。

* + ObstState

ある点の障害物の有無を表現するための構造体。

障害物があればOBSTACLE、なければNONE。

UNKNOWNは視界外。

* + ObstacleCol、Obstacle

障害物の情報を持つクラス。

詳しくは見ていないが、Colが1列分の情報を持っていて、それをObstacleがまとめて保持する感じだと思う。

* + Course

コースの情報の構造体

width,length : コースの幅、全長

vision : 視界の範囲。ルールドキュメントのdにあたる値。

thinkTime : 考慮時間

stepLimit : 制限ステップ数

obstacle : 障害物の情報

obstacled() : 衝突判定。衝突する場合がtrue。

始点と終点の情報から障害物に衝突するかを判定する。

* + RaceState

レースの進行情報の構造体

step : ステップ数

timeLeft : 経過時間

position, oppPosition : 座標

velocity, oppVelocity : 速度

collides() : 衝突判定に使う用？か開発者の意図は分からないが、少なくとも宣言だけで未定義。機能していない。

RaceState() : レースに必要な情報が更新される。

　　 ここで視界内の情報が与えられる。

* raceState.cpp

raceState.hppで定義したクラスや構造体の持つ関数の中身を定義。

機能は上で述べた通り。

dot関数やcross関数,ccw関数(<http://www.prefield.com/algorithm/geometry/ccw.html>)はgoseThru()で使われる。

* pseudo\_alphabeta.cpp

使われているのはαβ法(<https://drive.google.com/drive/folders/17nye-lPOF621UMRxJ6sooZ6KnRh4xeXL>)

(上の資料では葉ノードがMinノードだけど、このプログラムでは葉ノードはMaxノードだと思う。Maxノードは枝3でMinノードは枝9)

αβ法はMinMax法の一つで、Maxノードにα値、Minノードにβ値を設定し、その2つの値を基に最適解になりえない枝を刈り取っていく。

* + bfs() : 視界の端側から幅優先探索を行い、視界内の座標に対する評価値をつける。  
     視界が更新されない場合も、bfsを初期化して計算しなおすのは無駄。  
    　　　視界が更新されない(y軸に進まなかった時、評価値は全て同じなのにやり直　　　　す。)
  + hist : 探索記録を残すための変数。SEARCH\_DEPTHで深さを決定。

vector<pair<int,int>> 座標(2つのintのpair)を格納したvector

評価に使う

* + memo : Stateをkeyにもち、long longとIntVecのpairを格納

探索途中のメモ

同一局面を何度も計算しないためのもの

* + State : keyがintで、2つのPlayerStateのpairを持つ
  + cal : 評価関数。サンプルではy軸だけで評価しているらしい。
  + alpha\_beta() : alpha\_beta法のメインの部分。評価値と最適手のpairを返す。

　　　再帰的に呼び出される。

* + - 探索深さがSEARCH\_DEPTHと同じになるかゴールを超えた場合終了

このとき立ち往生していた場合、評価値を-INFにして返す。

* + - できる限りy方向の速度が視界の半分の値より大きくならないように加速度を決定
    - x方向への加速度を考えるときに相手の動きを考慮する

2重ループで相手が設定する加速度を全通り考慮する

* + - 自分と相手それぞれについて次ステップの座標、速度を更新

衝突があった場合の処理もこのタイミング

* + - histにdepthをkeyにして自分と相手の座標を格納する
    - alpha\_beta()を再帰的に呼び出す

この時にdepthとalpha値、beta値を渡す

* + - 再帰がいったん止まったところが葉ノードであるから、そこでその葉ノードの評価を行う。評価が終われば記録は必要なくなるので０で初期化する。
    - 再帰的に呼ばれたalpha\_beta()でそれまでに同じ状況での探索を行っていないかを確認し、もし行っているのならその探索をスキップ
    - 2重ループの部分がMinノードで、その中でgammaを更新

gammaは暫定的なbetaの値

* + - そのMinノードでの処理が終わると1つ上のMaxノードの処理に戻って、alphaを更新する
    - その後自分の別の手を探索する際にgammaをもう1つ上のMinノードから取得して、未探索のMinノードに移る。
    - これを繰り返してalphaとbetaを更新し、それをもとに枝刈りを実行
    - こうして最終的に最上位のMaxノードのalphaと最善手を返す
  + play()
    - bfs()は貪欲法に使う。alpha\_betaには使わない。
    - alpha\_beta()での解がその場から動けないものであれば貪欲法を使う
    - おそらく深さ制限ありの全探索

サンプルでは評価関数がy軸についてのみの評価なのでここをうまくいじればより良い結果は得られるだろうとは思うけど、具体的な案は今はない。

αβ法なので探索順序を次第で計算量が変わってくる。基本的にコースは蛇行することが多いと思うので、現在のxの値がかなり大きい値であればx方向への加速度を減らす手から優先的に探索する、みたいなことをすれば充分に枝刈りが行われるかもしれないとも考えた。

基本的にはどんなコースにでも対応できるアルゴリズムだと思うけど、考慮時間がネックになりそう。あまり深い探索は難しいと思う。

そこで、どれくらいの深さでどの程度考慮時間を消費するか計測した。  
実行環境やその他いろいろな要因で前後するが、  
サンプルコースの中の、sample-course(設定されている考慮時間は20000、視野は8)  
でgreedyに対して計測を行った。Player1.αβ、Player2.greedyの時。  
  
defaultの深さ5の時、時間17589残し、49ターン目でゴール。  
深さ6の時、時間900残し、50ターン目でゴール。  
深さ7の時、8ターン目で時間使いきって失格。  
  
→視野8だから、深さ5と6じゃ性能変わりそうだけど変わらない。  
  
深さ4の時、時間19474残し、53ターン目でゴール。  
深さ3の時、時間19918残し、54ターン目でゴール。  
  
  
アルファベータの持ち味、このステージじゃあまり活かしきれてないか、  
どちらにせよステージによっては、これ以上深く探索しても意味がないラインが存在するので、このあたりも詳細に考察する必要がある。