beam search

　　　　山内 拓磨

昨年度のSamurAI Jockeyで本戦8位のチームのプログラム(<https://trap.jp/post/291/>)を調査

全体的に変数名等が短くて読みづらい

* struct
  + DistNode : bfsのコストマップ作製用のNode
    - x : x座標、y : y座標、d : コスト(距離)　０に近いほどゴールに近い
  + OpponentState : 相手の状態
    - x : x座標、y : y座標、vx,vy : それぞれの方向への速さ
  + Node :
    - x : x座標、y : y座標、vx,vy : それぞれの方向への速さ
    - from : そこへ行くために選択された加速度、id : 紐づけのための値　1つ前の深さの何番手から派生したかを示す、coli : 衝突するかどうか、d : コスト？
* 変数
  + t,s,w,l,d : 制限考慮時間、制限ステップ数、width、lenght、視界
  + mp : 0,1で障害物の有無を格納　１が障害物
  + dist : 各座標のコストを格納　視界内の情報のみ
  + beam : ビーム幅
  + pre : 探索深さ
  + maxSpeed : 最高速度
  + hx,hy : 4の位置に注目した時の8近傍の相対座標
    - 1の位置が(x,y)=(1,0)、4の位置が(x,y)=(0,0)、8の位置が(x,y)=(-1,-1)といった感じに格納されている

hx[1]=1,hy[1]=0

|  |  |  |
| --- | --- | --- |
| 6 | 3 | 0 |
| 7 | 4 | 1 |
| 8 | 5 | 2 |

* function
  + gcd() : 最大公約数を返す
  + getDist() : 視界内の座標のコストを返す
    - y<0(視界より後ろ)であれば1000000を、y>2\*d(視界より先)であればd\*2-yを返す
  + innnerCollision() : 互いに疎なvx,vyに対して衝突判定を行う
    - 座標x,yからvx,vyで移動した際に障害物との衝突が起こるかを判定する
    - 障害物と衝突する場合trueを返す
    - 具体的な動作はよくわからない
  + collision() : 障害物との衝突を判定
    - 徐々に動線を伸ばしていき衝突が起こるかどうかを判定する
    - vx,vyどちらかが０ならば、0でないほうを1ずつ増やしていく
    - どちらも0でなければそれぞれを最大公約数で割った値ずつ増やしていく
  + collideOpponent() : 相手との衝突判定
    - collision()と同じように徐々に動線を伸ばしていき判定を行う
  + collideMovingOpponent() : 相手の動線との衝突判定
    - おそらくccw関数を用いた交差判定
    - 優先権の判定は移動開始位置のみの判定
    - 交差するとき優先権がある場合に1、ない場合に-1を返す
    - 交差しない場合には0を返す
  + h() : Hash関数
  + main() : メイン関数
    - コースの情報で初期化した後に、mpをy方向に100の余裕を持たせてリサイズ。distは(d\*2+1)×wにリサイズ
    - 最高速度はmin(maxSpeed,d-1)で決定　視界が短い場合には調整されるようになっている
    - while内がステップ毎の処理
    - お決まりの入力処理
    - distに初期値として10^7を代入
    - pseudo\_alphabetaのbfsのように視界の先端の障害物がない座標をコスト0とし、視界の最後方まで幅有線探索を行う

異なる点としては斜めの座標に対しても、その隣接する座標がどちらか一つでも障害物でなければ+1のコストで上書きする

* + - 相手が視界内にいれば相手の位置からbeamSearchを行う
    - その後、相手の動きも考慮しつつ自身に対してbeamSearchを行う
    - 探索深さまで達したら、最後に解の候補をソートして、そのNodeから順々にidをたどっていき、最終的にfromを取得し、それをもとに加速度を決定する
    - beamSearchの詳しい動作
      * 探索開始地点から深さ１の全探索を行う
      * それぞれについて衝突判定を行い、衝突が起こる場合には座標は動かさない
      * その結果の座標、速さ、選択された加速度、派生元を指す値、衝突したどうか、評価値を格納したNodeを生成し、保持する
      * 全探索し終えたのちに、生成したNodeを評価値順にソートする
        + 詳しいソートの基準はopperator<(Node,Node)を参照
      * これらのNodeをもちいて、また探索を行いということを探索深さに達するまで繰り返す

この時、ビーム幅によってその深さでの探索する枝の数を制限する

次にbeamSearch単独で試行した際のスコアと残り考慮時間を示す

ちなみにブログのコードをそのまま実行したところエラーが起きたため、

for (int j = 0; j < (int)min((unsigned long)beam, qs[i].size()); j++) {

の部分(beamSearchの枝刈りを行う部分)を

for (int j = 0; j < (int)min(beam, (int)qs[i].size()); j++) {

に変更した

以下に変更したコードを示す

<https://drive.google.com/file/d/1kcpHxDcXlNeHBBr_ZX2LAGVFo8ERpSef/view>

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | sample-course | course01 | course02 | course03 | course04 | course05 | course06 | course07 | course08 | course09 | course10 |
| Score | 48 | 24 | 38 | 77 | 68 | 27 | 43 | 35 | 40.5 | 18 | 33 |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | sample-course | course01 | course02 | course03 | course04 | course05 | course06 | course07 | course08 | course09 | course10 |
| greedy.cpp | 52.8 | 29.7 | 41.4 | 77.5 | 69.4 | × | 43.8 | 40 | 44.6 | 18 | 31.3 |
| LookaheadPlayer.java | 50.8 | 23.2 | 50.5 | 77 | × | 18 | × | 40.7 | × | 41.5 | 34.7 |
| pseudo\_alphabeta.cpp | 51 | 22.5 | 41.5 | × | 69.4 | 50 | 41 | 32.3 | 36.8 | 20 | 26.4 |
| py3\_sample.py | 49.5 | 31 | × | × | × | 35 | 47 | 41 | 44 | × | 35 |
| pseudo\_alphabeta\_cal1 | 49 | 24.3 | 39.5 | 77 | 68 | 50 | 41 | 32.3 | 36.8 | 19 | 26 |
| pseudo\_alphabeta\_cal2 | 49 | 23.8 | 41.5 | 77 | 68.4 | 50 | 41 | 32.3 | 36.8 | 20 | 26 |
| pseudo\_alphabeta\_cal3 | 49 | 24.3 | 41.5 | 77 | 68 | 50 | 41 | 32.3 | 36.8 | 19 | 25.8 |

単独スコアはサンプルと比べてもそれほどとびぬけているわけではなかった

<https://docs.google.com/document/d/1jHLm-mxT4_ZqVElZpzePrkBYdBp13IF4NPo3at5HB_E/edit>

↑これの結果からも言えることだが、単独スコアはあまり意味がないのかもしれない

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | sample-course | course01 | course02 | course03 | course04 | course05 | course06 | course07 | course08 | course09 | course10 |
| think time | 19973 | 5967 | 1910 | 1813 | 3909 | 19986 | 19996 | 19963 | 19994 | 1978 | 59978 |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | sample-course | course01 | course02 | course03 | course04 | course05 | course06 | course07 | course08 | course09 | course10 |
| greedy.cpp | 18584 | 5778 | 1932 | 1873 | 3695 | × | 19945 | 19350 | 19343 | 1961 | 59220 |
| LookaheadPlayer.java | 19556 | 5518 | 1192 | 1405 | × | 19555 | × | 19566 | × | 1601 | 59540 |
| pseudo\_alphabeta.cpp | 19099 | 5461 | 1419 | × | 2602 | 19805 | 19640 | 19689 | 19489 | 1700 | 59536 |
| py3\_sample.py | 14533 | 2829 | × | × | × | 18632 | 17843 | 16941 | 16638 | × | 56769 |

考慮時間は先読み７しているにもかかわらず、サンプルよりも短くすんでいる

はじめ見たときにビーム幅600は多すぎじゃないかと思ったがこれでもまだ少し余裕がありそう

次にサンプルAIとのレース結果を示す

スコアは (beamSearch) - (対戦相手) の順で、小数点第2位を四捨五入したものを示す

SOはステップオーバー、TOは考慮時間オーバー

player1 : beamSearch

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | sample | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| greedy | 53-50.8 | 24-28.7 | 41-43 | 80-77.5 | 68-74.4 | 27-SO | 45-46.8 | 35-42 | 38.5-40.6 | 20-19 | 32-30.7 |
| alphabeta | 53.4-50 | 24-25.8 | 41-TO | 79-TO | 68-TO | 27-50 | SO-SO | 35.2-30.3 | 41-38.8 | 18-TO | 31.2-30 |
| Lookahead | 48.4-44.8 | 24-32.4 | 38-51.5 | 77-80.7 | 68-76 | 27-20 | 43-SO | 35-42.7 | 40.5-SO | 19-21 | 29-35.2 |
| py3\_sample | 51-55.5 | 24-34 | 42-TO | 79-TO | 68-TO | 27-34 | 45-51 | 37.2-43 | 39.5-46 | 19-25 | 33-39 |

player2 : beamSearch

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | sample | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| greedy | 53-50.8 | 26-26.5 | 39-43.4 | 83-81.5 | 71-69.4 | 27-SO | 41-44.8 | 38.7-39.7 | 39.5-43.6 | 19-20.3 | 29-31.3 |
| alphabeta | 55.2-54.5 | 25-22.5 | 41-TO | 78-TO | 72-TO | 26-50 | SO-SO [1] | 39.2-34.3 | 41.5-35.8 | 20-TO | 29-34 |
| Lookahead | 49-52.8 | 25.3-34.4 | 38-50.5 | 78-81.7 | 72-71 | 26-23 | 43-SO | 37.3-36.4 | 44-40 | 18-20.5 | 32-36.2 |
| py3\_sample | 53-51 | 26-32 | 39-TO | 78-TO | 70-TO | 27-35 | 43-49 | 41.2-44 | 42.5-48 | 18-TO | 33-36 |

勝敗

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | sample | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| greedy | x | o | o | x | o | o | o | o | o | o | o |
| alphabeta | x | x | o | o | o | o | - | x | x | o | o |
| Lookahead | o | o | o | o | o | x | o | o | o | o | o |
| py3\_sample | o | o | o | o | o | o | o | o | o | o | o |

勝率 (勝) - (負) ((分))

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | greedy | alphabeata | Lookahead | py3\_sample |
| beamSearch | 9-2 | 6-4 (1) | 10-1 | 11-0 |

~~上の結果の通り未実行のLookaheadPlayer.java以外にはすべて勝ち越すことができた~~

特にgreedy.cpp、py3\_sample.py、LookaheadPlayer.javaには大きく勝ち越している

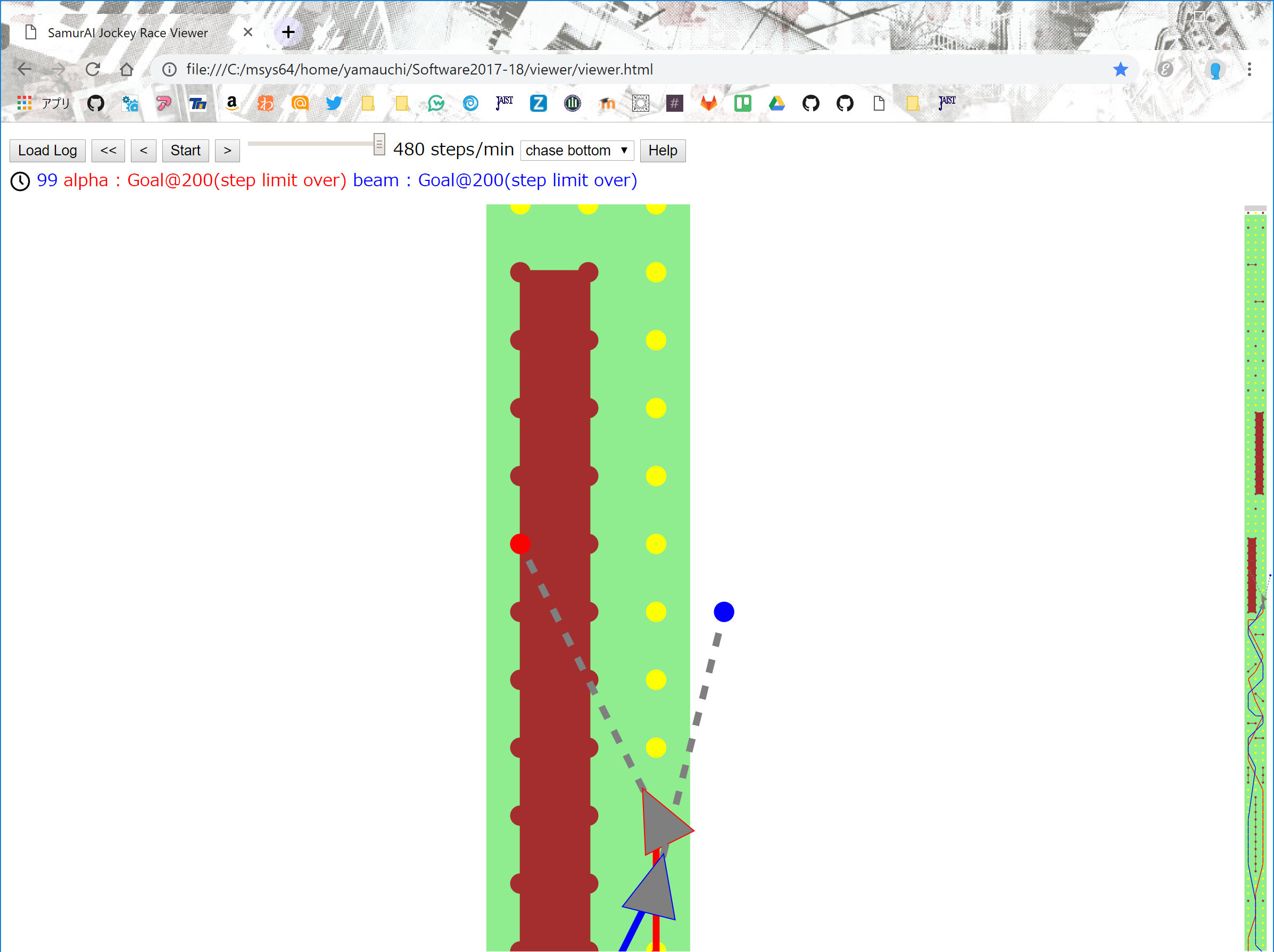
一方でpseudo\_alphabeta.cppに対しては相性のせいかそれほど勝ち越せなかった

beamSearchがalphabetaに勝ったレースの半分以上は相手のTOによるもので、お互いがTOすることなくゴールできたレースに限るとむしろ負けている

alphabetaはかなり早い段階でTOしているものが多く、beamSearchは一度もTOはしていないため、そういった意味では優秀だが、alphabetaも枝刈りさえうまくいけば、去年の本戦レベルでも戦えるAIになりうるのかもとも思った

もちろんその枝刈りが難しいのだが、、

beamSearchの総評としては、計算量が比較的少なく済むため先読みを深めやすく、安定してそれなりのスコアを出すことができるAIだと思う

alphabetaと対戦時のcourse06での出来事

　[1]　　　　　　　　両者立ち往生してSO