Introduction to WrightEagleBase

陈荣亚

cry@mail.ustc.edu.cn

(Base on WrightEagleBase4.0)
Outline

- About World State
- About Decision
- Source Code Structure
- Some Useful Functions and Tools
About World State

Server

Observe Information String

Parser\{.h .cpp\}

Observe Information

Observer\{.h .cpp\}

Client (WEBase)

Perception

Maintenance
About World State

- Perception
- Observer
- History
- Server Model
- WorldState{.h .cpp}
- BallState{.h .cpp}
- PlayerState {.h .cpp}
Info State

WorldState {.h .cpp} → InfoState {.h .cpp} → InterceptInfo {.h .cpp} → PositionInfo {.h .cpp}
Decision Data

- History
- InfoState{.h .cpp}
- Strategy{.h .cpp}
- Analyser{.h .cpp}

Used by defense and offense
Only Used by defense
About Decision

Know Formation {.h .cpp} & All of the above Information

Inverse

WorldState {.h .cpp}
InfoState {.h .cpp}
Decision Data

Do action by using ActionEffector{.h .cpp}

Agent {.h .cpp}
Do Decision

Agent

Behavior*Planner
(In Behavior*{.h .cpp})

ActiveBehavior(s)

Behavior*Executer
(In Behavior*{.h .cpp})

BehaviorBase{.h .cpp}
Decision Tree

- DecisionTree (.h .cpp)
  - BehaviorPenalty (.h .cpp)
    - BehaviorSetPlay (.h .cpp)
      - BehaviorAttack (.h .cpp)
        - BehaviorDefense (.h .cpp)
  - BehaviorDribble (.h .cpp)
  - BehaviorPass (.h .cpp)
  - BehaviorShoot (.h .cpp)

Active Behavior

Best

Behavior*Executer
Structure of Source Code

- **conf/**
  - player.conf, server.conf and other file of configuration
- **data/**
  - some data generated by offline calculation
- **formations/**
  - files of formation
- **src/**
  - C++ source code
- **Logfiles/**
  - log files (created by the class "Logger")
- **Debug/**
  - Debug version Makefile
- **Release/**
  - Release version Makefile
Files

- **dbg, dd** dynamic debug tools
- **genlog** generate log files
- **showlog** show sight log
- **memcheck** check the fault about memory
- **initrc** the sharing shell, used by all of tools above
- **dynamicdebug.txt** the text file used to dynamic debug.
- **start.sh** start the team
- **Makefile** Makefile
Files (cont.)

- Types.{h, cpp}  
  some basic class, some marco
- Geometry.{h, cpp}  
  about the geometric computing
- Utilities.{h, cpp}  
  some useful tools and data structure such as PythonArray
- Dasher.{h, cpp}  
  about dashing
- Kicker.{h, cpp}  
  about kicking
- Tackler.{h, cpp}  
  about tackling
- Behavior*.{h, cpp}  
  about planning and executing all the behavior.  
  You can change the Behavior*::Plan() to change strategy.
- CommunicationSystem.{h, cpp}  
  the subsystem about communication
- VisualSystem.{h, cpp}  
  the subsystem about visual.
- Coach.{h,cpp}  
  the online coach, you can change the types of player in this file.
Useful Functions

- Dasher {.h .cpp}
- Kicker {.h .cpp}
- Tackler {.h .cpp}
- VisualSystem {.h .cpp}
- CommuniteSystem {.h .cpp}
Dasher

- GetBall(⋯) Get ball in specified cycle or as fastest as possible
- GoToPoint(⋯) Go to point in specified cycle or as fastest as possible
- CycleNeedToPoint(⋯) Calculation the cycle needed running to a specified point
- RealCycleNeedToPoint(⋯) Calculation the real cycle needed running to a specified point
Kicker

- KickBall(⋯)  Kick ball in the specified way
- GetMaxSpeed(⋯)  Maximum speed that can be kicked to the specified direction
- GetStopBallAction(⋯)
- GetAccelerateBallAction(⋯)
- GetKickBallToAngleAction(⋯)
Tackler

- TackleStopBall(⋯)
- CanTackleStopBall(⋯)
- TackleToDir(⋯)
- CanTackleToDir(⋯)
- GetBallVelAfterTackleToDir(⋯)
Visual System

- RaiseBall(⋯)  Attention to the ball by assigning weights
- RaisePlayer(⋯)  Attention to the player by assigning weights
- SetForceSeeBall(⋯)  (If you can)
- SetForceSeePlayer(⋯)
- SetCritical(⋯)  Set whether use the narrow view width
- ForbidDecision(⋯)
- SetCanTurn(⋯)  Set whether consider “turn” action while doing decision
- ChangeViewWidth(⋯)
Communicate System

- SendBallStatus(⋯)
- SendTeammateStatus(⋯)
- SendOpponentStatus(⋯)  (All of above is Boardcast)
- ParseReceivedTeammateMsg(⋯)
About dynamic debug

Server → Observe Information String → Logger{.h .cpp} → BINARY-*-msg.log

save_server_message = on

gdb/ cgdb → Attach & Control → Client
Process of dynamic debug

- Modify the name of BINARY in initrc
- Modify the team_name in conf/player.conf
- Set save_server_message = on
- Start the match normally (server_message will be recorded in Logfiles/ as BINARY-*-msg.log)
- ./dd unum, run the Client in dynamic debug mode
- ./dbg, make gdb attach to the Client process, then debug.
About Logger

- Record Log files
- TextLogger: record logs in the text form
- SightLogger: record logs in the rcg form
- ./genlog unum: generate log files using server message log
- ./showlog: show sight log using rcsslogplayer
Hint on Offense Decision

- Assort Position
- Ahead Pass
- And so on
Hint on Defense Decision

- BehaviorFormation (.h .cpp)
- BehaviorMark (.h .cpp)
- BehaviorMark (.h .cpp)
- BehaviorBlock (.h .cpp)

Same Line Teammate

Self

Best

Behavior* Executor