Sports Simulation – Simple Soccer

Artificial Intelligence for Interactive Media and Games

Professor Charles Rich
Computer Science Department
rich@wpi.edu

[Based on Buckland, Chapter 4 and lecture by Robin Burke]

Plan for next three weeks (re Soccer only)

Read Chapter 3 on Steering!

- **Mon/Tue/Thu**: Simple Soccer Anatomy
- **Sun midnite**: “My Team” homework due [3 pt]
  - set up code to make modifications
  - study game play carefully to look for improvements
- **Mon**: In-class brainstorming
- **Wed midnite**: “Team Design” homework due [3 pt]
- **Wed (two weeks later)**: “Tournament Team” due [10 pt]
  - requires adding substantial new strategy
- **Fri, Nov 18**: Soccer tournament (HL 230)
  - final grade bonus points for winner and runner-up
The Road to Tournament...

<table>
<thead>
<tr>
<th>Day</th>
<th>Book</th>
<th>Lecture (with links to notes)</th>
<th>Homework (with links)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tue, Oct 25</td>
<td></td>
<td>Intro/Overview/Admin</td>
<td></td>
</tr>
<tr>
<td>Wed, Oct 26</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thu, Oct 27</td>
<td>Chapter 2</td>
<td>State Machines</td>
<td></td>
</tr>
<tr>
<td>Fri, Oct 28</td>
<td>Chapter 2</td>
<td>Event Messages</td>
<td></td>
</tr>
<tr>
<td>Sun, Oct 30</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mon, Oct 31</td>
<td>Chapter 4</td>
<td>Simple Soccer Anatomy</td>
<td></td>
</tr>
<tr>
<td>Tue, Nov 1</td>
<td>Chapter 4</td>
<td>Simple Soccer Anatomy</td>
<td></td>
</tr>
<tr>
<td>Wed, Nov 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thu, Nov 3</td>
<td>Chapter 4</td>
<td>Simple Soccer Anatomy</td>
<td></td>
</tr>
<tr>
<td>Fri, Nov 4</td>
<td></td>
<td>Research: AIIDE Conference</td>
<td></td>
</tr>
<tr>
<td>Sun, Nov 6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mon, Nov 7</td>
<td>Brainstorming: Simple Soccer Strategy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tue, Nov 8</td>
<td>Chapter 6</td>
<td>LUA Scripting</td>
<td></td>
</tr>
<tr>
<td>Wed, Nov 9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thu, Nov 10</td>
<td>Chapter 6</td>
<td>LUA Scripting</td>
<td></td>
</tr>
<tr>
<td>Fri, Nov 11</td>
<td></td>
<td>Procedure Content Generation</td>
<td></td>
</tr>
<tr>
<td>Sun, Nov 13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mon, Nov 14</td>
<td>Procedure Content Generation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tue, Nov 15</td>
<td>Chapter 7</td>
<td>Raven Anatomy</td>
<td></td>
</tr>
<tr>
<td>Wed, Nov 16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thu, Nov 17</td>
<td>Chapter 7</td>
<td>Raven Anatomy</td>
<td></td>
</tr>
<tr>
<td>Fri, Nov 18</td>
<td></td>
<td>Soccer Tournament (HL 230 1)</td>
<td></td>
</tr>
</tbody>
</table>

Simple Soccer

- 2D sports simulation (*no interactive player*)
- 2 teams (“red” and “blue”)
- 5 autonomous agents per team
  - 4 field players
  - 1 goal keeper
- 1 field (“pitch”)
- 2 goals
- 1 ball
Simple Soccer Demo

- **Red Team**: BucklandTeam
- **Blue Team**: BurkeTeam
- Keyboard controls
  - P for pause/resume
  - R for reset (new match)
- Frame (update) rate
  - default 60 Hz (FrameRate in Params.ini)
  - can slow down to study behavior more closely
- Match
  - default 5 min (TimeLimit in Params.ini)
  - scoring bonus for using less CPU time (details later)

Why?

- Why should we learn all this complicated, detailed soccer strategy?
  - after all, this is a course about general techniques for game AI, not soccer specifically...
- **Answer**:  
  1. Because there is no other way to appreciate the real issues in building a game AI without mastering something reasonably complicated.  
  2. Actually, this is only a start and has lots of room for improvement---a platform for your own ideas!
Design Issues in Simple Soccer

- Geometry and Physics
  - steering
  - navigation

- Tiered AI
  - overall team state machine
  - each player has state machine
  - each player has (changing) role in team
    - e.g., pass receiver
  - messaging between players
    - e.g., “pass me the ball”

- Soccer-specific strategy and design

Avoiding Perfection

- Like many other genres (e.g., FPS), AI opponents in sports simulations must be beatable
  - AI’s may have inherently weak strategies (e.g., no defensive plays in Simple Soccer)
  - explicit fudge factors (e.g., n% of shots go wild)

- Inaccurate (approximate) physics modeling
  - saves compute time, but causes AI’s to make mistakes
  - e.g., circles instead of ellipses to calculate interception in Simple Soccer
  - bug = feature 😊
How about “Stats-Driven” Play?

- not illustrated in Simple Soccer
- individual AI performance governed by “stats” (e.g., speed, shooting accuracy)
- interactions between AI’s calculated based on stat comparisons and random factors
- typical in reality-based sports games (NBA, etc.)

Soccer Rule Simplifications

- ball cannot go over heads of players
- ball rebounds off walls
- ball does not collide with players’ feet
- no corners or throw-ins
- no off-side rules
As aid to implementing strategies, pitch divided into 18 regions (numbered 0-17 as above)

Each player has a “home region”
- starting point for match
- may change during play
**Simple Soccer Physics**

- **Physics** simulation or “game” simulation?

- **Answer:** A balance between them
  - enough (approximate) physics to look ok
  - plus other (completely unrealistic) information to make it easier to implement game, e.g.,
    - m_pOwner: a field of ball that points to player that currently “owns” the ball

---

**Soccer Ball Physics**

- Three elementary kinematic equations
  - $v = u + at$
  - $d = ut + \frac{1}{2} at^2$
  - $v^2 = u^2 - 2ad$

- **Dynamics:** $F = ma$
- Acceleration (a) is Friction in Params.ini
- Soccer ball only checks for collision with pitch boundaries
  - angle of incidence equals angle of reflection
  - ball moves freely through players “feet”
Reality vs. Approximation

- **Kicking a soccer ball...**

- **In reality:**
  - player swings foot toward moving ball
  - force on ball at moment of collision with foot changes current velocity of ball

- **Approximation in game:**
  - pretend ball stopped at moment of kick
  - player gives ball fixed initial velocity
  - easier to calculate
  - looks ok

---

(player roles)

CS/IMGD 4100 (B 16)
**Player Roles in Soccer Team Class**

- **Closest Player to the Ball**
  - updated every tick
  - never null

- **Receiving Player**
  - waiting to receive kicked ball
  - may be null

- **Controlling Player**
  - more contextual: passer, receiver, attacker
  - may be null

- **Supporting Player**
  - works with controlling player
  - always defined when controlling player defined

*May need to extend these to improve team strategy!*

**Scoring Support (Sweet) Spots**

SupportSpotCalculator instance for each team

“Heat Map”

- possible destinations for supporting player in opposing half-pitch (default 13x6)
- each *dynamically* rated for (weights in Params.ini)
  - safe passing (Spot_PassSafeScore)
  - goal scoring potential (Spot_CanScoreFromPositionScore)
  - distance from controlling player (Spot_DistFromControllingPlayerScore)
Non-agent entities:

- Soccer pitch
- Goal
- Soccer ball
- Support spot calculator
- Main loop

Team States (Upper Tier AI)

- Attacking
  - our team possession
  - goal scored
  - other team possession

- Defending
  - goal scored

- Prepare..Kickoff
  - play starts

[Demo]
TeamStates::PrepareForKickoff

- entered
  - start of match
  - after goal scored
- sends “GoHome” message to all players
- waits until all players are home
- transitions to Defending state

TeamStates::Defending

- change home regions to defending set
- steers all field players to homes
- if team gets control, transition to Attacking
TeamStates::Attacking

- change home regions to attacking set
- choose supporting player / spot
- if team loses control, transition to Defending

FieldPlayerStates::GlobalPlayerState

- handles messages between players
  - Msg_SupportAttacker
  - Msg_ReceiveBall
  - Msg_GoHome
  - Msg_Wait (not used)

- and from team to players
  - Msg_GoHome
  - Msg_PassToMe

- no messages from players to team in this implementation (could add!)
Field Players (4)

- 2 attackers
- 2 defenders
(field in PlayerBase)

Field Player States

- ReceiveBall
  - in receiving range
  - kicked
- Dribble
  - can’t shoot or pass
- ChaseBall
  - in range
  - can’t kick
- KickBall
  - goal or pass attempt
- Wait
  - at home
  - MSG: PassToMe
  - MSG: MSG, Wait
- ReturnToHR
  - closest
  - at home
  - MSG: PassToMe
  - team lost control
- SupportAttacker
  - not closest
  - MSG: GoHome
  - MSG: SupportAttacker

MSG:
- ReceiveBall
- SupportAttacker
- Dribble
- ChaseBall
- KickBall
- Wait

[Demo]
FieldPlayerStates::ChaseBall

- turn on “seek” steering to ball’s current position
- if in kicking range, transition to KickBall
- if no longer closest player, ReturnToHomeRegion
- turn off “seek” when exiting

FieldPlayerStates::Wait

- hold position at current steering target
  - turn on “arrive” steering to return if jostled by another player (collision avoidance)
- if upfield of teammate in control, send Msg_PassToMe to controlling player
- if closest to ball and no current receiver (and goalie does not have ball), transition to ChaseBall
FieldPlayerStates::ReceiveBall

- entered in response to Msg_ReceiveBall
  - telegram contains target location of ball
  - at most one player on team in this state
- choose between “arrive” vs. “pursuit” steering towards ball
  - always use “arrive” if close to goal or threatened
  - otherwise, random variation
- if close enough to ball or team loses control, transition to ChaseBall

FieldPlayerStates::KickBall

- if max kicks/sec exceeded or goalie has ball, transition to ChaseBall
- if CanShoot (see later), Ball()->Kick()
  - random noise, “pot shots”
  - transition to Wait
  - assign supporting player and send Msg_SupportAttacker
- else if threatened and CanPass (see later)
  - assign receiver and send Msg_ReceiveBall
- otherwise, transition to Dribble
  - assign supporting player and send Msg_SupportAttacker
FieldPlayerStates::Dribble

- turn upfield if necessary (maintaining control of ball)
- repeat
  - kick ball short distance
  - transition to ChaseBall
  - which will transition to KickBall
  - which will transition to Dribble

FieldPlayerStates::SupportAttacker

- steer (“arrive on”) to selected support spot
  - support spot re-evaluated every update
- if CanShoot and not threatened, then send Msg_PassToMe to controlling player (attacker)
- if cannot request pass, the remain at support spot and “track” (face) ball
- if team loses control, transition to ReturnToHomeRegion
Goal Keeper

- always faces ball
  - steering behaviors use velocity-aligned heading
  - special vector $m_{vLookAt}$

GoalKeeperStates::GlobalKeeperState

- handles two messages
  - Msg_GoHome
  - Msg_ReceiveBall
Goal Keeper States

- **ReturnHome**: msg: GoHome
  - too far from goal (unless closest to ball)

- **TendGoal**: msg: ReceiveBall
  - too far from goal
  - ball within range
  - transition to PutBallBackInPlay: has ball
  - transition to InterceptBall: pass ball

- **InterceptBall**: has ball
- **PutBallBackInPlay**: has ball

---

**GoalKeeperStates::TendGoal**

- move laterally, using "interpose" steering to keep body between ball and rear of goal
- if ball comes within control range, transition to PutBallBackInPlay
- if ball comes within intercept range, transition to InterceptBall
GoalKeeperStates::PutBallBackInPlay

- send Msg_ReturnHome to all field players (including opponents!)
- pass to teammate
- transition to TendGoal

GoalKeeperStates::InterceptBall

- steer towards ball using “pursuit”
- if close enough to trap ball transition to PutBallBackInPlay
- if move too far from goal
  - unless goalie is closest player to ball
  - transition to ReturnHome
Typical Goal Scored on Keeper

Key AI Methods in AbstSoccerTeam

- isPassSafeFromAllOpponents
- CanShoot
- GetBestPassToReceiver
- FindPass
isPassSafeFromAllOpponents

- direct pass
  - assume kicked ball speed > max player speed
  - then any player "behind" kicker is safe
  - how to calculate “behind”?

isPassSafeFromAllOpponents (cont’d)

- transform to local coordinates of kicker
- all opponents (e.g., W) with negative x coordinate are “behind” kick (i.e., safe)
isPassSafeFromAllOpponents (cont’d)

- how about opponents beyond receiver (x coordinate > B), e.g., Q?
- if time to receiver (BQ) is greater than pass time (AB), then safe

isPassSafeFromAllOpponents (cont’d)

- how about “side passes”?
- for each side target, similarly consider times for opponents with x coordinate > target
isPassSafeFromAllOpponents (cont’d)

- how to eliminate remaining opponents?
- compute closest (perpendicular) intercept point (e.g., Xp, Yp)
- compare time for ball vs. opponent to reach intercept point
  - adjustment for ball size and capture distance
  - ignoring time for opponent to rotate

[Code Walk]

CanShoot

- choose random points along back of goal
- check that not too far (force vs. friction)
- call isPassSafeFromAllOpponents

[Code Walk]
GetBestPassToReceiver

- eliminate if receiver too far (force vs. friction)
  - doesn’t consider receiver running toward passer
- consider “side passes”

GetBestPassToReceiver (cont’d)

- compute range (dotted circle) of receiver within time duration of pass
  - using time duration to current receiver position
  - reduce range to 30% to allow safety margin (turning, etc.)
- side pass targets are ip1 and ip2
  - check that inside pitch
  - call isPassSafeFromAllOpponents

[Code Walk]
**FindPass**

- call `GetBestPassToReceiver` on each teammate beyond `MinPassingDistance`
- choose teammate who can safely receive pass that is furthest upfield

[Code Walk]

---

**Params.ini**

```ini
... 
// weights used to calculate the support spots
Spot_PassSafeScore 2.0
Spot_CanScoreFromPositionScore 1.0
Spot_DistFromControllingPlayerScore 2.0
...
```

- you might think that the name on each line identifies the variable that is set
  **WRONG**

- you might think that the variables can be listed in any order
  **WRONG**

- `ParamLoader.h`
Parameter File Loading

- We’ll see a much better version of this using Lua in Raven code
  - any order
  - add variables
  - use expressions as values

Field Player “Strategic” Parameters

// scoring values for support spots
Spot_CanPassScore 2.0
Spot_CanScoreFromPositionScore 1.0
Spot_DistFromControllingPlayerScore 2.0

// when an opponent comes within this range the player will attempt to
// pass (the higher the value, the more often players tend to pass)
PlayerComfortZone 60.0

// minimum distance a receiving player must be from the passing player
MinPassDistance 120.0

// how close the ball must be to a receiver before he starts chasing it
BallWithinReceivingRange 10.0
Goalie “Strategic” Parameters

// minimum distance a player must be from the goalkeeper before it will
// pass the ball
GoalkeeperMinPassDistance 50.0

// the distance the keeper puts between the back of the net
// and the ball when using the interpose steering behavior
GoalKeeperTendingDistance 20.0

// when the ball becomes within this distance of the goalkeeper he
// changes state to intercept the ball
GoalKeeperInterceptRange 100.0

Making Buckland’s Code “Multi-User”

- To support tournament play
- Factory pattern for teams
- Unsolved problems:
  - reusing states
  - changing parameters
Factory Pattern

- **Goal:** decide at run-time (e.g., by loading info from Params.ini) which team class to make an instance of
  - avoid directly calling “new” with class name in game initialization code
  - Params.ini
    - RedTeam
    - SmithTeam
    - BlueTeam
    - JonesTeam

- **Solution:**
  - define an abstract class (AbstSoccerTeam)
  - with a “factory method” (makeTeam)
  - use inheritance/polymorphism

---

Factory Pattern

- **singleton registry** TeamMaker->newTeam(“BurkeTeam”)

  ↓

- **singleton factory** BurkeSoccerTeamMaker->makeTeam(…)

  ↓

- **subclass AbstSoccerTeam** new BurkeSoccerTeam(…)
What’s Not Solved

- All team and player states need to be copied
  - why?
  - you are doing this in HW 4 – My Team

- Values you desire to change in Params.ini need to be replaced each place they are referenced in your team code!
  - why?
  - replace Prm.MinPassDistance by 25

Coming up...

- [Fri: Research Snapshot: AIIDE 2014]
- Sun: “My Team” Homework Due
- Mon: Brainstorming in Class
- Wed: “Team Design” Homework Due
- Wed after: “Tournament Team” HW Due