Sports Simulation – Simple Soccer

Artificial Intelligence for
Interactive Media and Games

Professor Charles Rich
Computer Science Department
rich@wpi.edu

Plan for next three weeks (re Soccer)

- *Mon/Tues/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:* Discussion of Team Improvements
- *Weds midnite:* "Team Design" homework due [3 pt]
- *Weds midnite:* "Tournament Team" due [10 pt]
  - full credit for beating Buckland's team
- *Fri Nov. 21:* Soccer tournament (IMGD Lab)
  - bonus points for winning elimination matches

Week	Day	Book	Lecture	Homework
1	Tue, Oct 28		Intro/Overview/Admin	
	Wed, Oct 29			1 - Hello West World [3%]
	Thu, Oct 30	Chapter 2	State Machines	
	Fri, Oct 31	Chapter 2	Event Messages	
	Sun, Nov 2			2 - Bar Fly [5%]
2	Mon, Nov 3	Chapter 4	Simple Soccer Anatomy	
	Tue, Nov 4	Chapter 4	Simple Soccer Anatomy	
	Wed, Nov 5			3 - Tank States [5%]
	Thu, Nov 6	Chapter 4	Simple Soccer Anatomy	
	Fri, Nov 7		**Futures:** AIIDE Conference Report	
	Sun, Nov 9			4 - My Team [3%]
3	Mon, Nov 10		**Discussion:** Soccer Team Improvements	
	Tue, Nov 11	Chapter 6	LUA Scripting	
	Wed, Nov 12			5 - Team Design [3%]
	Thu, Nov 13	Chapter 6	LUA Scripting	
	Fri, Nov 14		**Futures:** TBD	
	Sun, Nov 16			6 - Scripting [5%]
4	Mon, Nov 17	Chapter 7	Raven Anatomy	
	Tue, Nov 18	Chapter 7	Raven Anatomy	
	Wed, Nov 19			7 - Tournament Team [10%]
	Thu, Nov 20	Chapter 7	Raven Anatomy	
	Fri, Nov 21		**Soccer Tournament** (IMGD Lab)	
	Sun, Nov 23			8 - Add Weapon [3%]

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?
  - this is a course about general techniques for game AI, not soccer specifically
  - some students asked the same question about chess in IMGD 4000
- *Answer:* Because there is no other way to appreciate the complexity of building a game AI and the software issues it forces without mastering something complex.

Issues in Simple Soccer

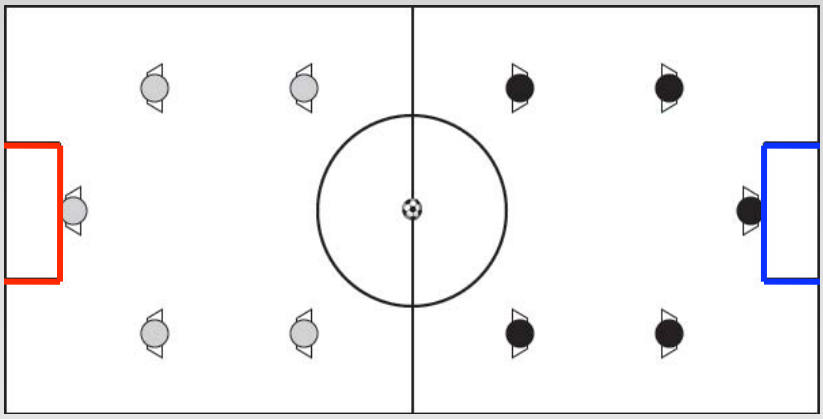
- Geometry and Physics
  - steering
  - navigation
- Tiered AI
  - overall team state machine
  - each player has state machine
  - each player has (changing) role in team
    - e.g., pass receiver
  - messaging between players
    - e.g., "pass me the ball"
- *Soccer-specific strategy and design*

Avoiding Perfection

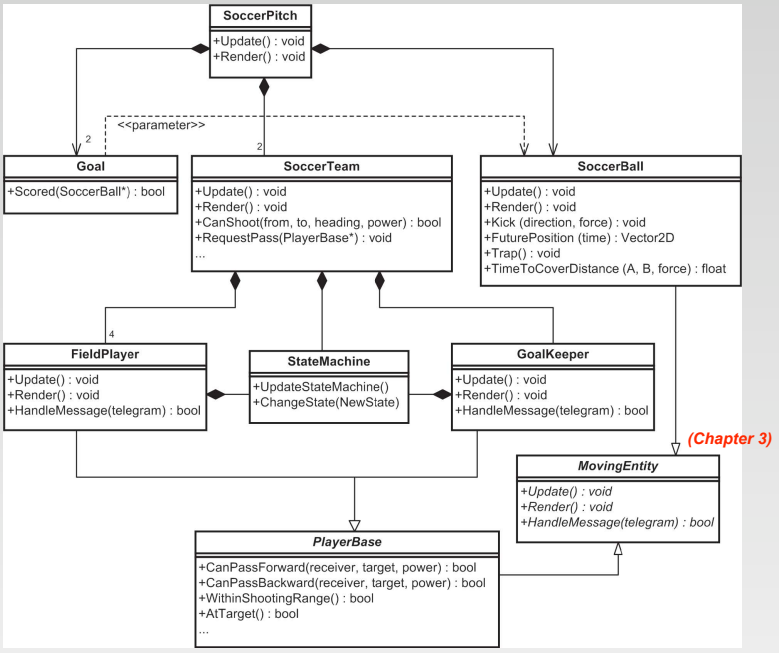
- Like many other genres (e.g., FPS), AI opponents in sports simulations must be *beatable*
  - AI's may have inherently weak strategies (e.g,. no defensive plays in Simple Soccer)
  - explicit fudge factors (e.g., n% of shots go wild)
- Inaccurate (approximate) physics modeling
  - saves compute time, but causes AI's to make mistakes
  - e.g., circles instead of ellipses to calculate interception in Simple Soccer

"Stats"-driven Play

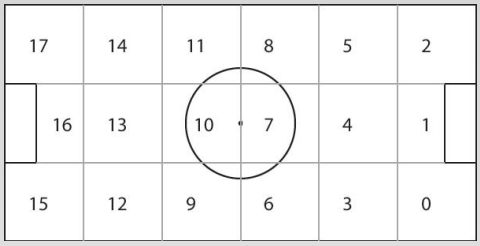
- individual AI performance governed by "stats" (e.g., speed, shooting accuracy)
- interactions between AI's calculated based on stat comparisons and random factors
- typical in reality-based sports games (NBA, etc.)
- not illustrated in Simple Soccer



- Soccer Rule Simplifications
  - ball rebounds off walls
  - no corners or throw-ins
  - no off-side rules



11



- 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

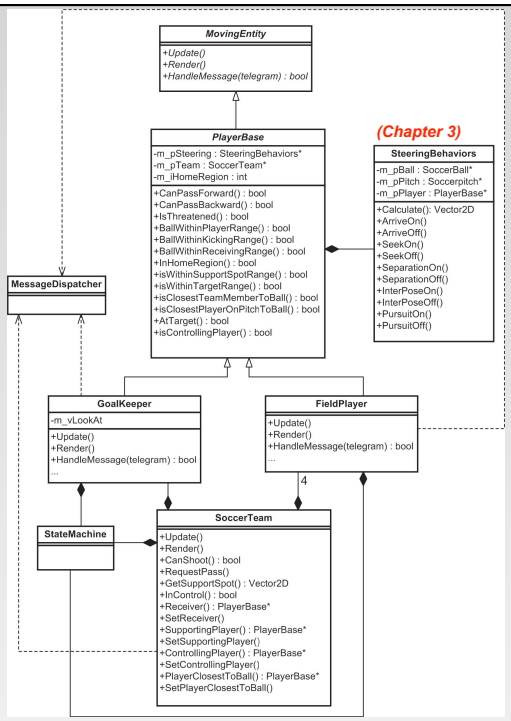
12

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"

Kicking a Soccer Ball

- In reality:
  - player swings foot toward moving ball
  - force on ball at moment of collision with foot changes velocity of ball
- Approximation in game:
  - pretend ball stopped at moment of kick
  - player gives ball fixed initial velocity
  - easier to calculate
  - looks ok



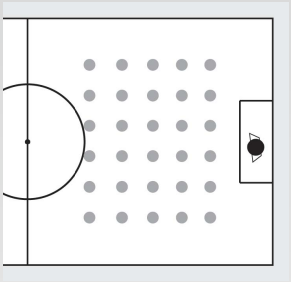
15

Player Roles in Soccer Team Class

- Receiving Player
  - waiting to receive kicked ball
  - may be null
- Closest Player to the Ball
  - updated every tick
  - never 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

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

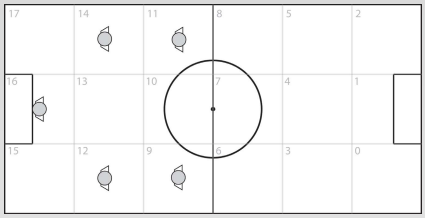
Code Walk

Non-agent entities:

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

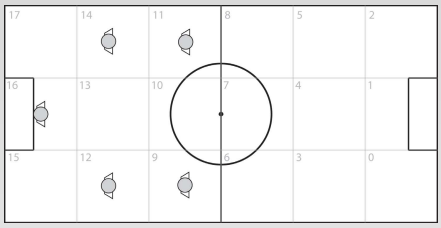
Team States (Upper Tier AI)

TeamStates::PrepareForKickoff



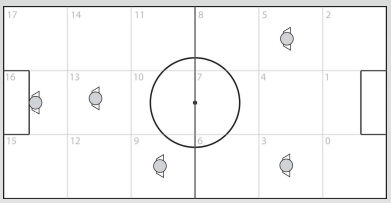
- entered
  - start of match
  - after goal scored
- sends all players to (defending) home regions
- "waits" until all players are home
- transitions to Defending state

TeamStates::Defending



- change home regions to blue (defending) set
- steers all field players to homes
- if team gets control, transition to Attacking

TeamStates::Attacking



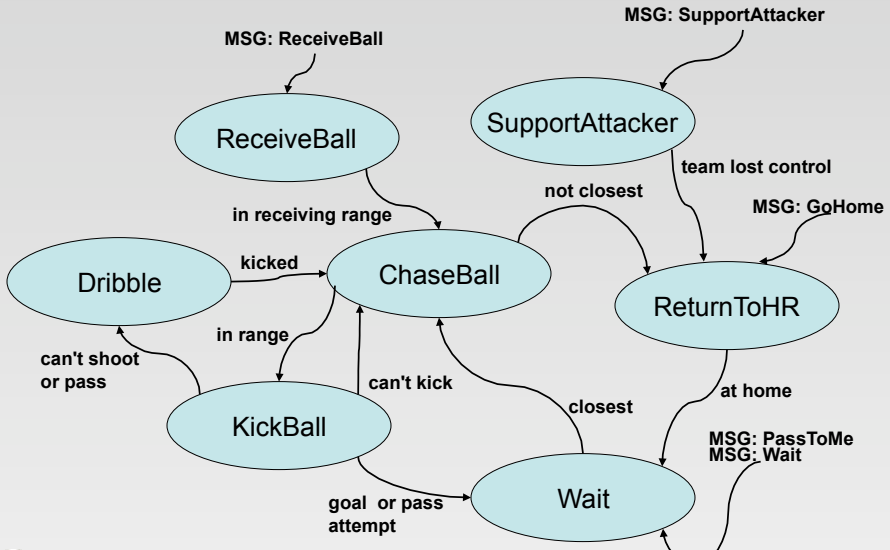
- change home regions to red (attacking) set
- choose supporting player / spot
- if team loses control, transition to Defending

Demo/Code Walk

FieldPlayerStates::GlobalPlayerState

- handles all messages between players
  - Msg_SupportAttacker
  - Msg_GoHome
  - Msg_ReceiveBall
  - Msg_PassToMe
  - Msg_Wait
- no messages between team and players in this implementation (could add!)

Field Player States



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

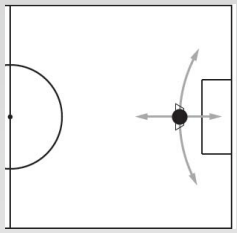
- turn upfield if necessary (maintaining control of ball)
- repeat
  - kick ball short distance
  - transition to ChaseBall
  - which will transition to KickBall
  - which will transition to Dribble

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

Demo/Code Walk

Goal Keeper

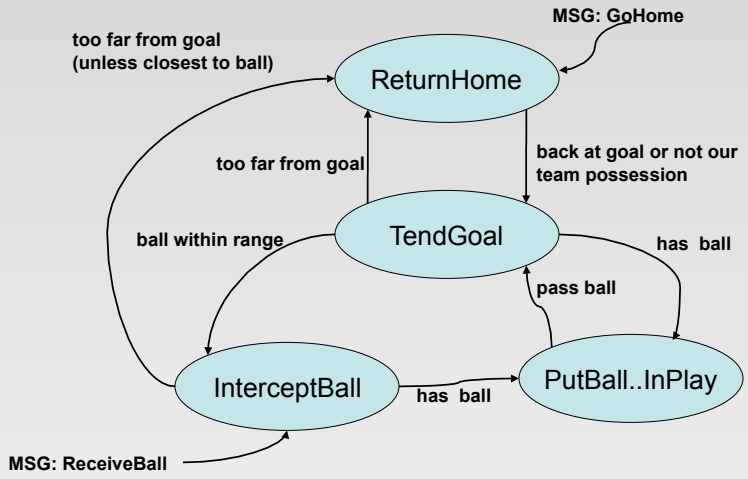


- always faces ball
  - steering behaviors use velocity-aligned heading
  - special vector m_vLookAt

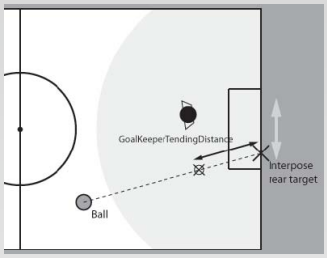
GoalKeeperStates::GlobalKeeperState

- handles two messages
  - Msg_GoHome
  - Msg_ReceiveBall

Goal Keeper States



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

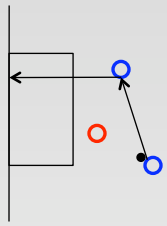
GoalKeeperStates::PutBallBackInPlay

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

GoalKeeperStates::InterceptGoal

- 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

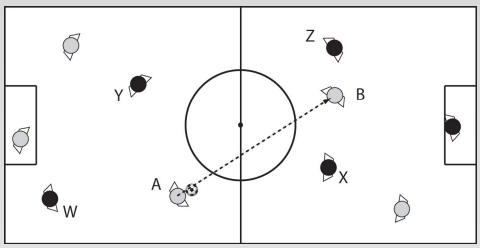


Demo/Code Walk

Key AI Methods in AbstSoccerTeam

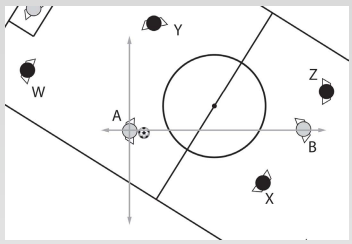
- isPassSafeFromAllOpponents
- CanShoot
- FindPass
- GetBestPasstoReceiver

isPassSafeFromAllOpponents



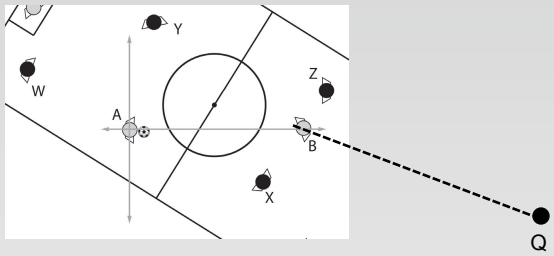
- direct pass
  - assume kicked ball speed > max player speed
  - then any player “behind” kicker is safe
  - how to calculate “behind” ?

isPassSafeFromAllOpponents (cont’d)



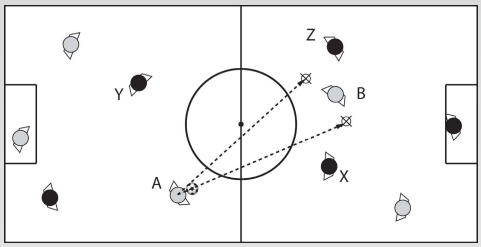
- transform to local coordinates of kicker
- all opponents (e.g., W) with negative x coordinate are “behind” kick (i.e., safe)

isPassSafeFromAllOpponents (cont'd)



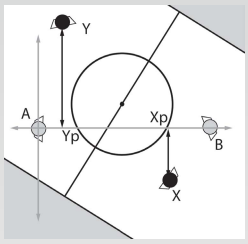
- how about opponents beyond receiver (x coordinate > B), e.g., Q ?
- if distance to receiver (BQ) is greater than pass distance (AB), then safe

isPassSafeFromAllOpponents (cont'd)



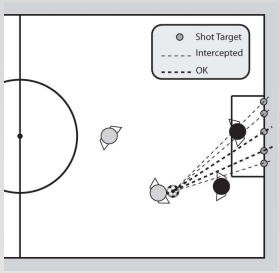
- how about "side passes" ?
- same condition for opponents "beyond receiver", except use side target points

isPassSafeFromAllOpponents (cont'd)



- how to eliminate remaining opponents?
- compute closest 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

CanShoot

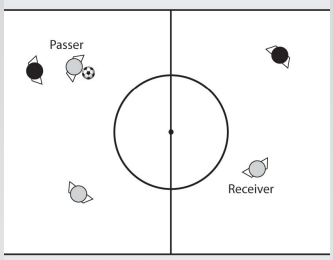


- choose random points along back of goal
- check that not too far (force vs. friction)
- call isPassSafeFromAllOpponents

FindPass

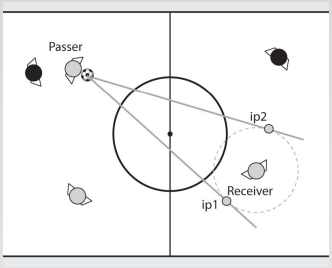
- call GetBestPassToReceiver on each teammate beyond MinPassingDistance
- choose teammate who can safely receive pass that is furthest upfield

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

- isPassSafeFromAllOpponents
- CanShoot
- FindPass
- GetBestPassToReceiver

Params.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

"Strategic" Parameters (cont'd)

```
// 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

// how close the ball must be to a receiver before he starts chasing it
BallWithinReceivingRange        10.0
```


"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
```

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
- ***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 the states need to be copied
  - why?
- Changed values in Params.ini need to be replaced at point of reference
  - why?

G.J. Sussman: "The flexibility of a unit of code is directly proportional to the number of levels of *indirection* it uses."

Homeworks

- *Sunday:* My Team
- *Weds:* Team Design
- *Weds:* Tournament Team