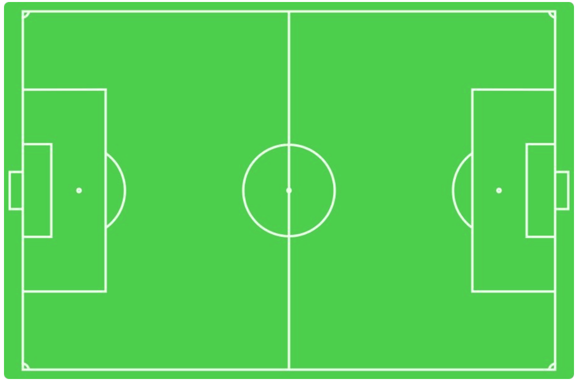
CS3211 Midterm Presentation

Individual Project: Pan Yongjing



Applying PMC in Soccer Analytics

Project source code

https://github.com/yjpan47/soccer-pmc

Contents

1. The Model of a Soccer Game

2. Manage Probabilities

3. Future Improvements

Began with a Simplified Street Soccer

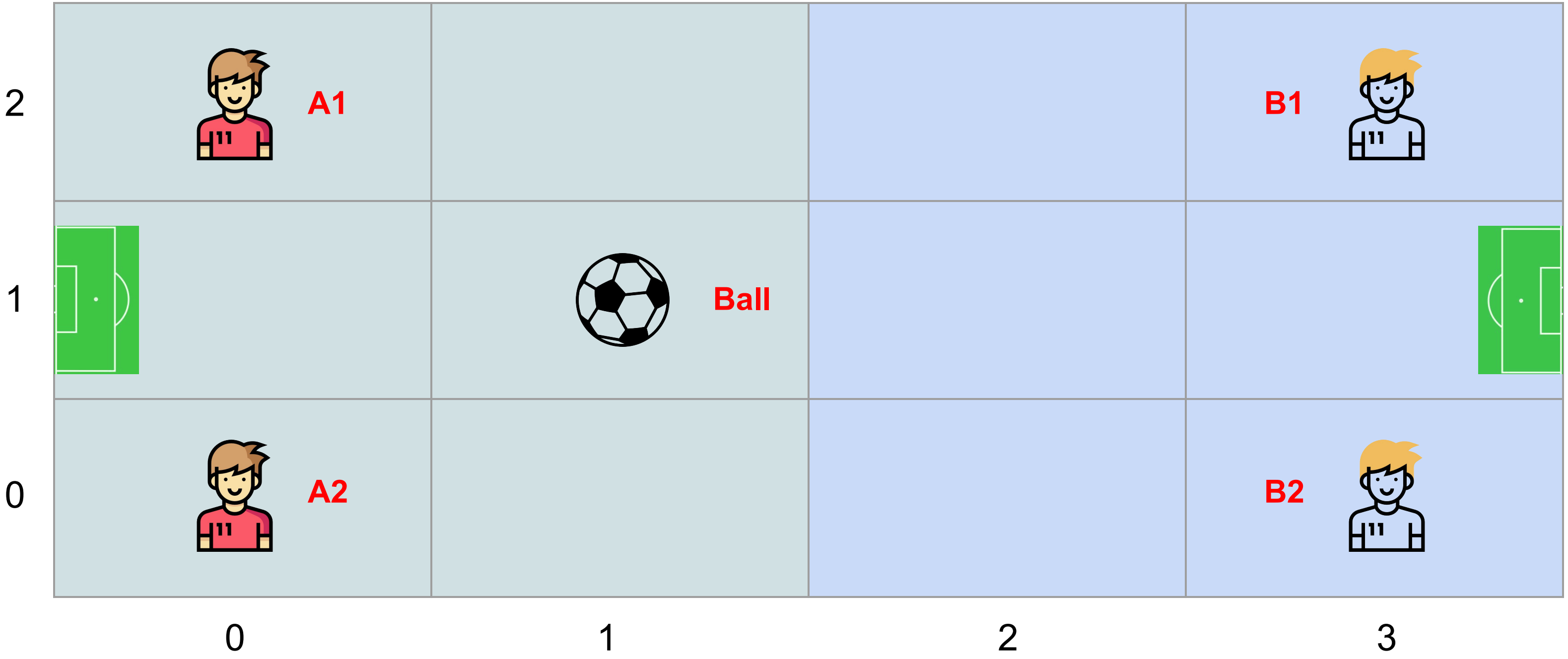
How I model the soccer game?

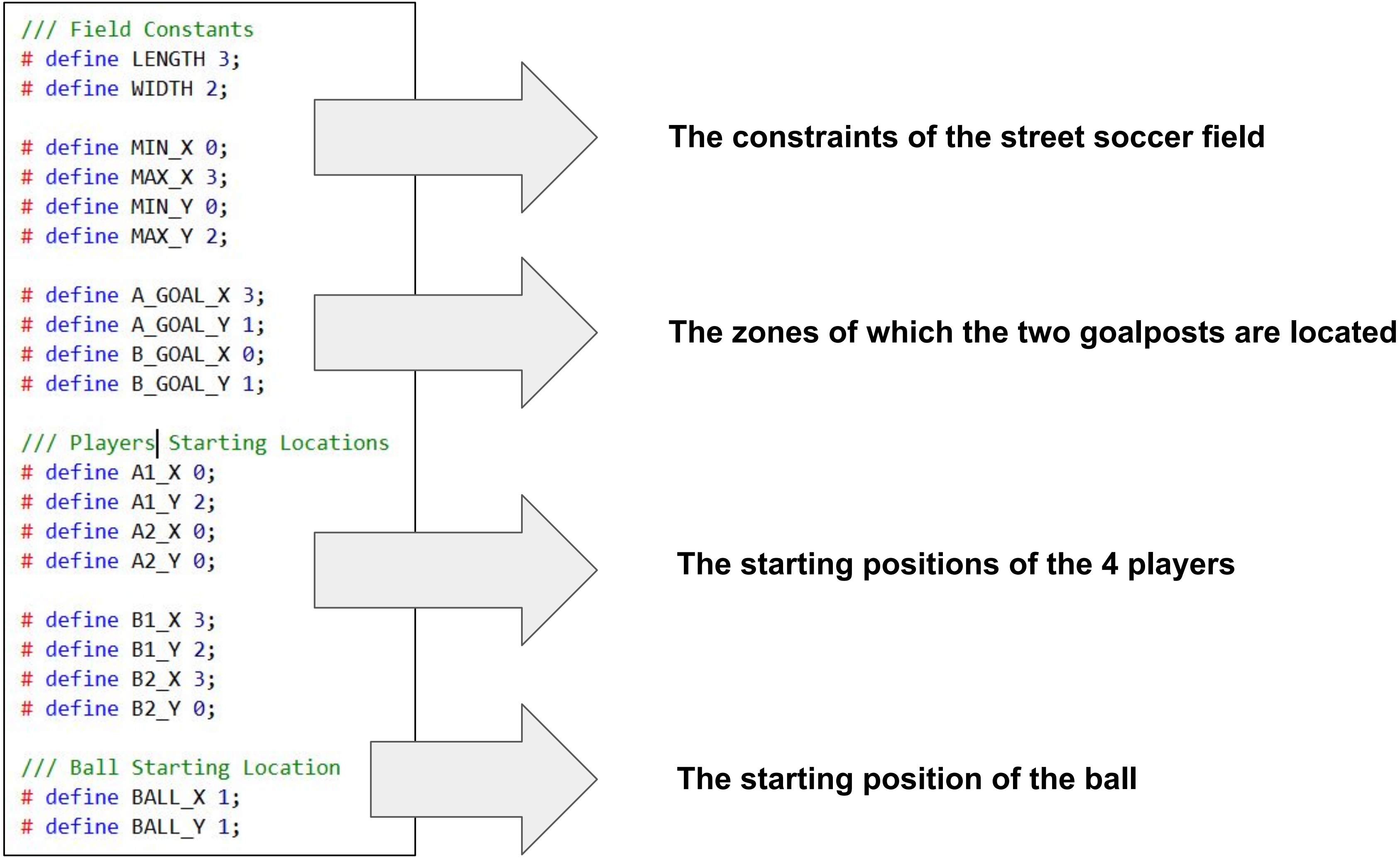
- The soccer field is divided into **4 x 3 = 12** zones.

- There are two teams - **Team A** and **Team B**.
- Each team has two players.

- There is one ball.
- The ball can be **free** or **possessed** by a player.

Simplified Street Soccer Game





```
/// Field Constants
# define LENGTH 3;
# define WIDTH 2;

# define MIN_X 0;
# define MAX_X 3;
# define MIN_Y 0;
# define MAX_Y 2;

# define A_GOAL_X 3;
# define A_GOAL_Y 1;
# define B_GOAL_X 0;
# define B_GOAL_Y 1;

/// Players Starting Locations
# define A1_X 0;
# define A1_Y 2;
# define A2_X 0;
# define A2_Y 0;

# define B1_X 3;
# define B1_Y 2;
# define B2_X 3;
# define B2_Y 0;

/// Ball Starting Location
# define BALL_X 1;
# define BALL_Y 1;
```

The constraints of the street soccer field

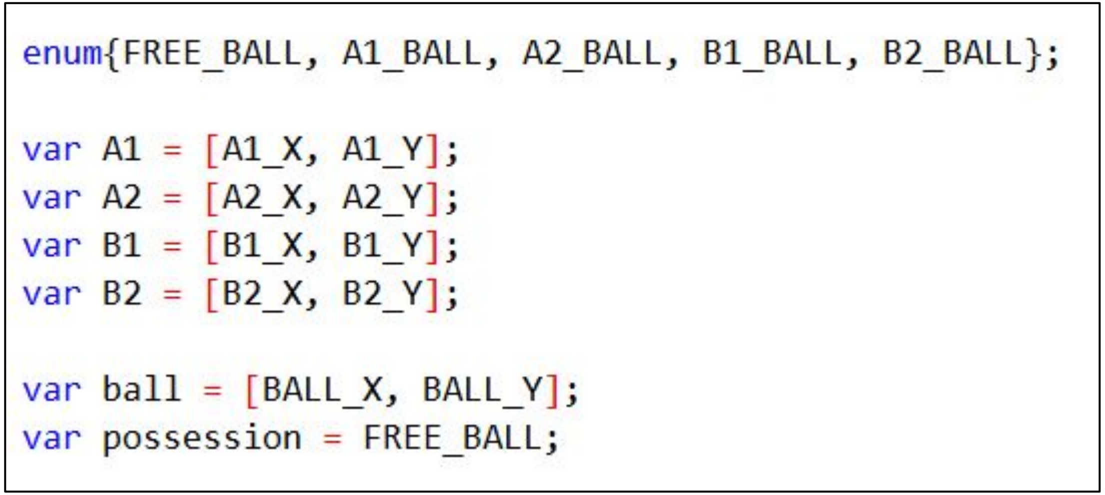
The zones of which the two goalposts are located

The starting positions of the 4 players

The starting position of the ball

The States of the Model

1. The locations (xy-coordinates) of each of the 4 players
2. The location (xy-coordinate) of the ball
3. The possession of the ball

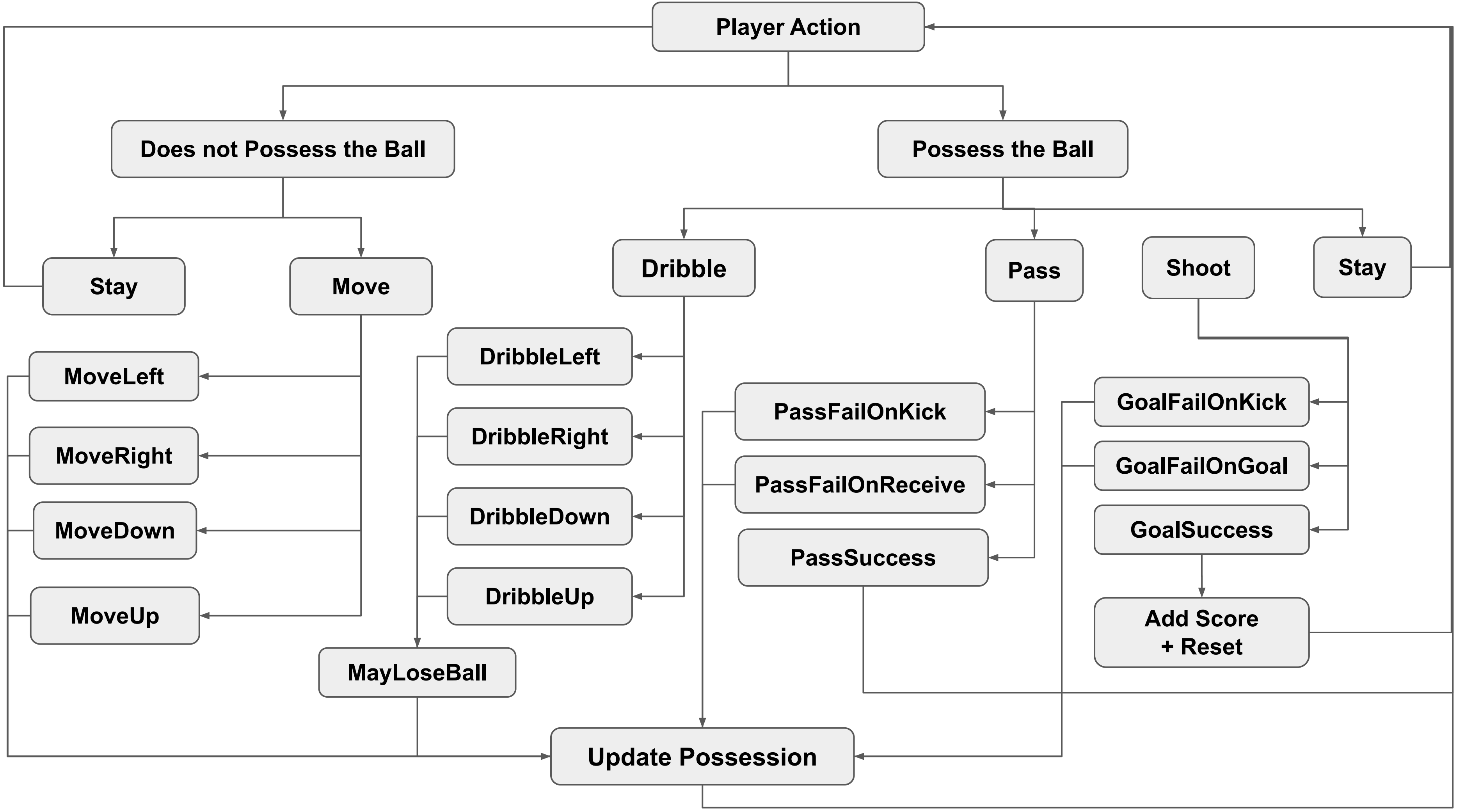


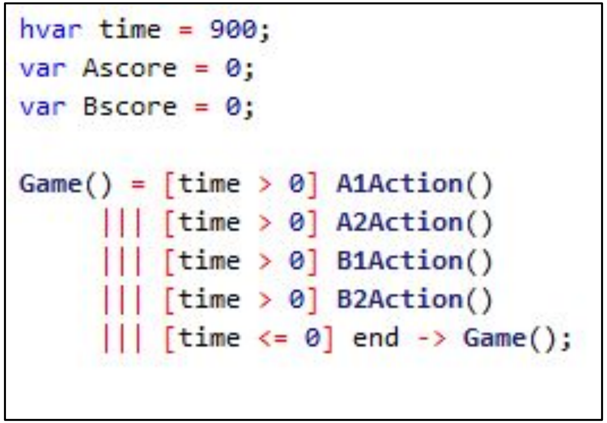
```
enum{FREE_BALL, A1_BALL, A2_BALL, B1_BALL, B2_BALL};

var A1 = [A1_X, A1_Y];
var A2 = [A2_X, A2_Y];
var B1 = [B1_X, B1_Y];
var B2 = [B2_X, B2_Y];

var ball = [BALL_X, BALL_Y];
var possession = FREE_BALL;
```

The Events of the Model

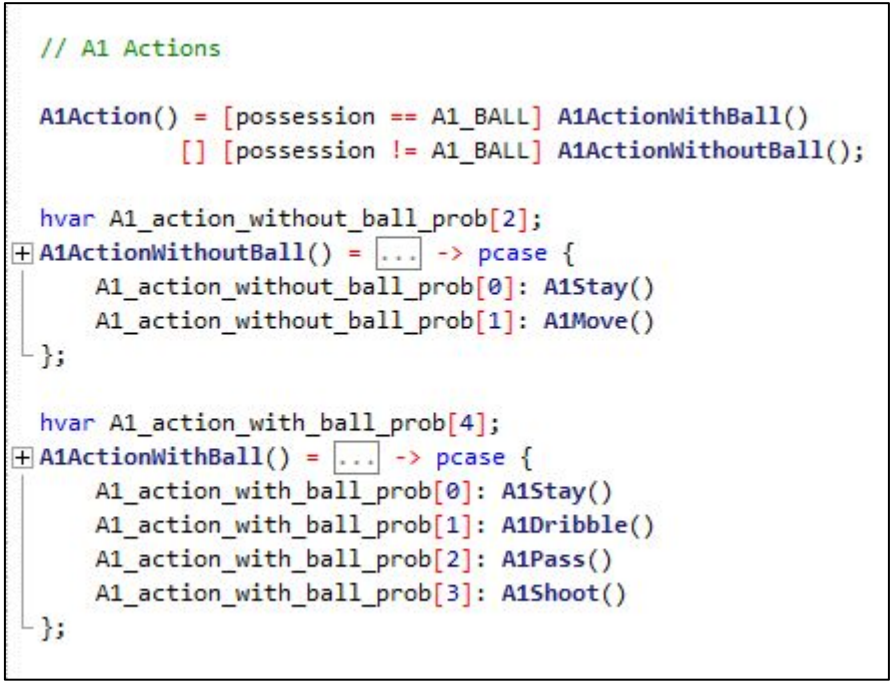
Actions of each player

- When not possessing the ball
  - Stay in the zone
  - Move to adjacent zones (left/right/up/down)
    - Potentially gain possession of the ball
- When possessing the ball
  - Stay in the zone
  - Dribble the ball to adjacent zones (left/right/up/down)
    - Potentially lose possession of the ball
  - Pass the ball to teammate
    - Potentially lose possession of the ball
  - Shoot the ball
    - Potentially lose possession of the ball
    - Potentially score a goal

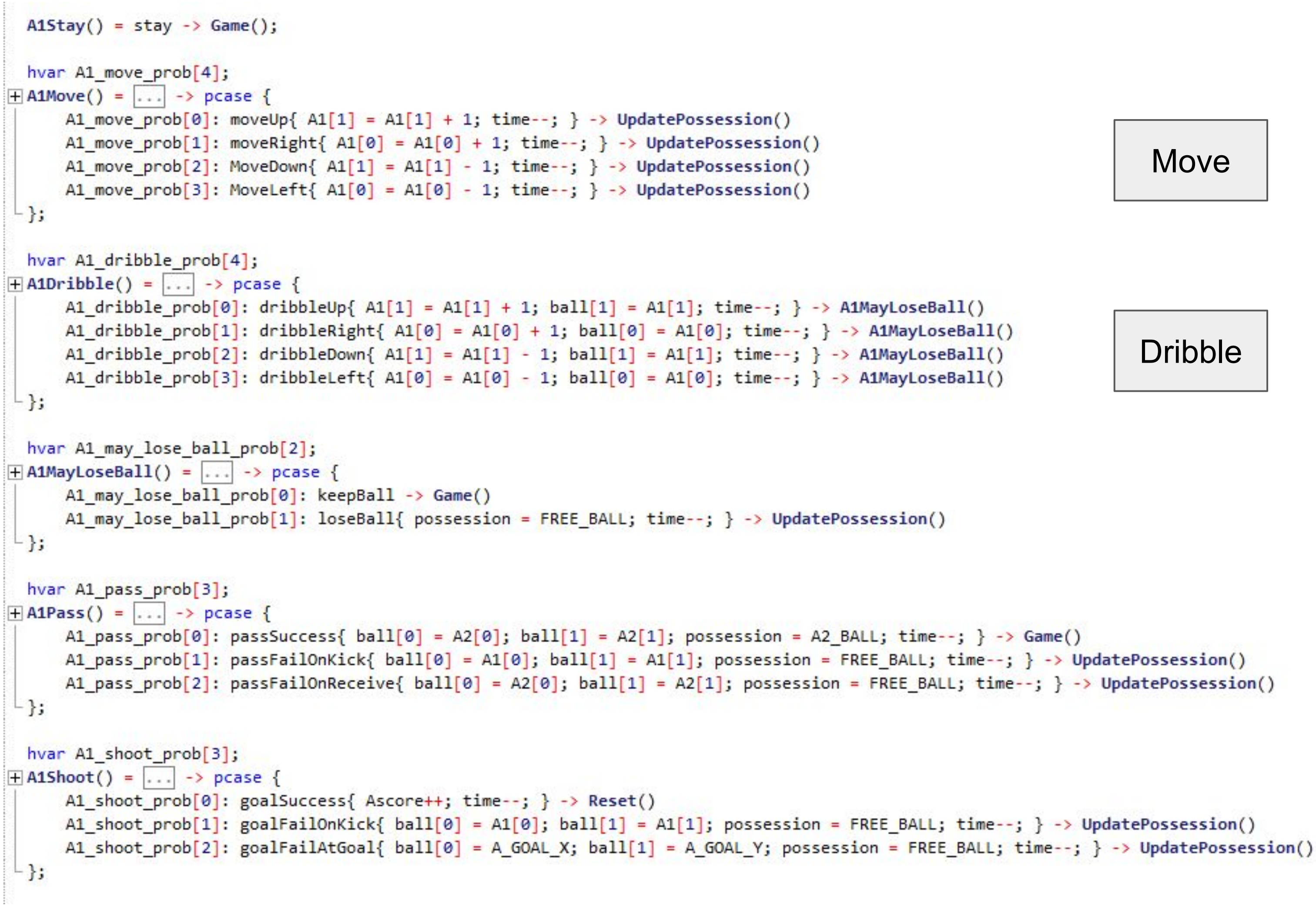

Player Action
Does not Possess the Ball
Possess the Ball
Stay
Move
Dribble
Pass
Shoot
Stay
MoveLeft
MoveRight
MoveDown
MoveUp
DribbleLeft
DribbleRight
DribbleDown
DribbleUp
MayLoseBall
PassFailOnKick
PassFailOnReceive
PassSuccess
GoalFailOnKick
GoalFailOnGoal
GoalSuccess
Add Score
+ Reset
Update Possession



```
hvar time = 900;
var Ascore = 0;
var Bscore = 0;

Game() = [time > 0] A1Action()
      ||| [time > 0] A2Action()
      ||| [time > 0] B1Action()
      ||| [time > 0] B2Action()
      ||| [time <= 0] end -> Game();
```



```
// A1 Actions

A1Action() = [possession == A1_BALL] A1ActionWithBall()
          [] [possession != A1_BALL] A1ActionWithoutBall();

hvar A1_action_without_ball_prob[2];
A1ActionWithoutBall() = ... -> pcase {
    A1_action_without_ball_prob[0]: A1Stay()
    A1_action_without_ball_prob[1]: A1Move()
};

hvar A1_action_with_ball_prob[4];
A1ActionWithBall() = ... -> pcase {
    A1_action_with_ball_prob[0]: A1Stay()
    A1_action_with_ball_prob[1]: A1Dribble()
    A1_action_with_ball_prob[2]: A1Pass()
    A1_action_with_ball_prob[3]: A1Shoot()
};
```



```
A1Stay() = stay -> Game();

hvar A1_move_prob[4];
A1Move() = ... -> pcase {
    A1_move_prob[0]: moveUp{ A1[1] = A1[1] + 1; time--; } -> UpdatePossession()
    A1_move_prob[1]: moveRight{ A1[0] = A1[0] + 1; time--; } -> UpdatePossession()
    A1_move_prob[2]: MoveDown{ A1[1] = A1[1] - 1; time--; } -> UpdatePossession()
    A1_move_prob[3]: MoveLeft{ A1[0] = A1[0] - 1; time--; } -> UpdatePossession()
};

hvar A1_dribble_prob[4];
A1Dribble() = ... -> pcase {
    A1_dribble_prob[0]: dribbleUp{ A1[1] = A1[1] + 1; ball[1] = A1[1]; time--; } -> A1MayLoseBall()
    A1_dribble_prob[1]: dribbleRight{ A1[0] = A1[0] + 1; ball[0] = A1[0]; time--; } -> A1MayLoseBall()
    A1_dribble_prob[2]: dribbleDown{ A1[1] = A1[1] - 1; ball[1] = A1[1]; time--; } -> A1MayLoseBall()
    A1_dribble_prob[3]: dribbleLeft{ A1[0] = A1[0] - 1; ball[0] = A1[0]; time--; } -> A1MayLoseBall()
};

hvar A1_may_lose_ball_prob[2];
A1MayLoseBall() = ... -> pcase {
    A1_may_lose_ball_prob[0]: keepBall -> Game()
    A1_may_lose_ball_prob[1]: loseBall{ possession = FREE_BALL; time--; } -> UpdatePossession()
};

hvar A1_pass_prob[3];
A1Pass() = ... -> pcase {
    A1_pass_prob[0]: passSuccess{ ball[0] = A2[0]; ball[1] = A2[1]; possession = A2_BALL; time--; } -> Game()
    A1_pass_prob[1]: passFailOnKick{ ball[0] = A1[0]; ball[1] = A1[1]; possession = FREE_BALL; time--; } -> UpdatePossession()
    A1_pass_prob[2]: passFailOnReceive{ ball[0] = A2[0]; ball[1] = A2[1]; possession = FREE_BALL; time--; } -> UpdatePossession()
};

hvar A1_shoot_prob[3];
A1Shoot() = ... -> pcase {
    A1_shoot_prob[0]: goalSuccess{ Ascore++; time--; } -> Reset()
    A1_shoot_prob[1]: goalFailOnKick{ ball[0] = A1[0]; ball[1] = A1[1]; possession = FREE_BALL; time--; } -> UpdatePossession()
    A1_shoot_prob[2]: goalFailAtGoal{ ball[0] = A_GOAL_X; ball[1] = A_GOAL_Y; possession = FREE_BALL; time--; } -> UpdatePossession()
};
```

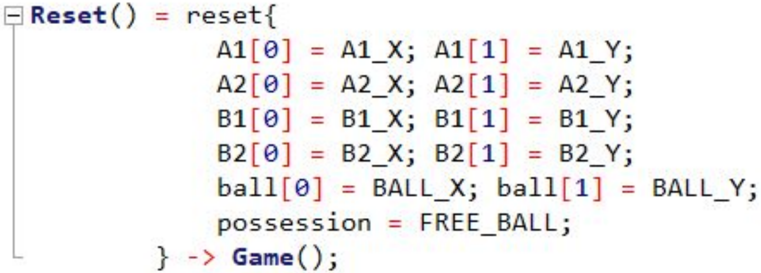
Move

Dribble

Pass

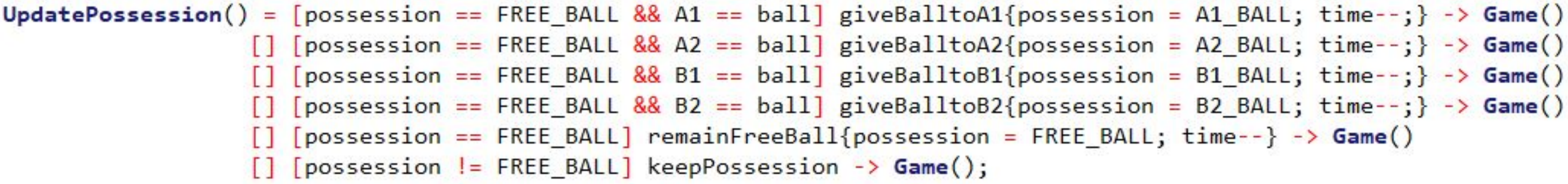
Shoot

Reset



```
Reset() = reset{
            A1[0] = A1_X; A1[1] = A1_Y;
            A2[0] = A2_X; A2[1] = A2_Y;
            B1[0] = B1_X; B1[1] = B1_Y;
            B2[0] = B2_X; B2[1] = B2_Y;
            ball[0] = BALL_X; ball[1] = BALL_Y;
            possession = FREE_BALL;
        } -> Game();
```

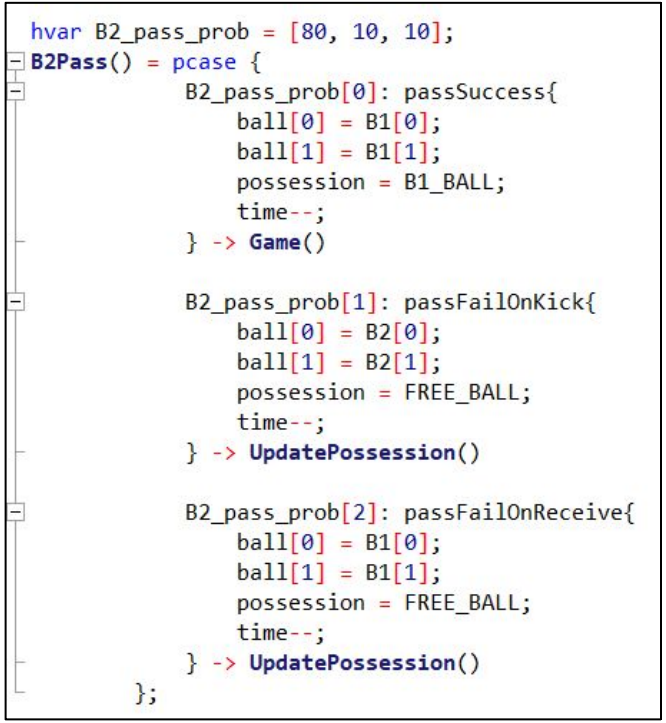
Update Possession



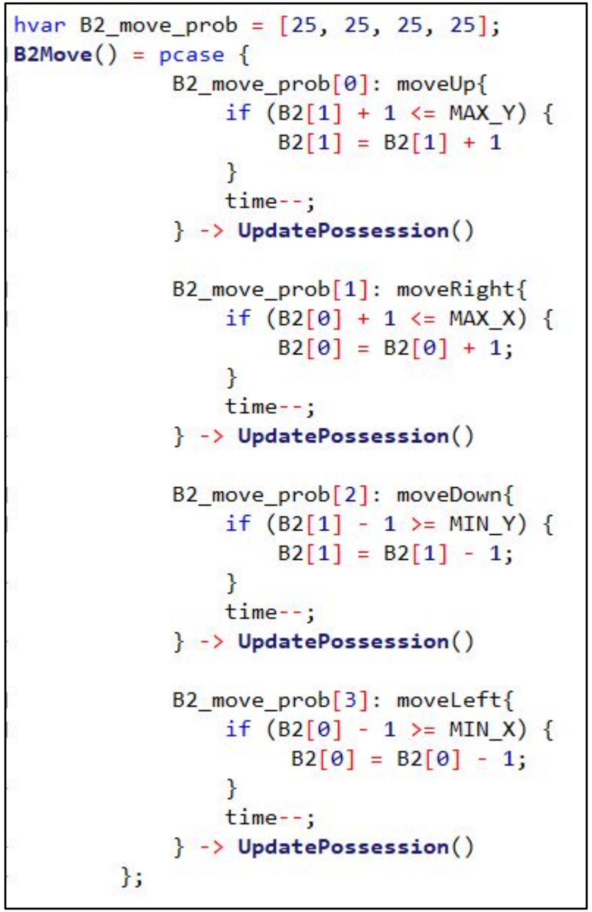
```
UpdatePossession() = [possession == FREE_BALL && A1 == ball] giveBalltoA1{possession = A1_BALL; time--;} -> Game()
                   [] [possession == FREE_BALL && A2 == ball] giveBalltoA2{possession = A2_BALL; time--;} -> Game()
                   [] [possession == FREE_BALL && B1 == ball] giveBalltoB1{possession = B1_BALL; time--;} -> Game()
                   [] [possession == FREE_BALL && B2 == ball] giveBalltoB2{possession = B2_BALL; time--;} -> Game()
                   [] [possession == FREE_BALL] remainFreeBall{possession = FREE_BALL; time--} -> Game()
                   [] [possession != FREE_BALL] keepPossession -> Game();
```

The Probabilities

Started with reasonable estimations for all players



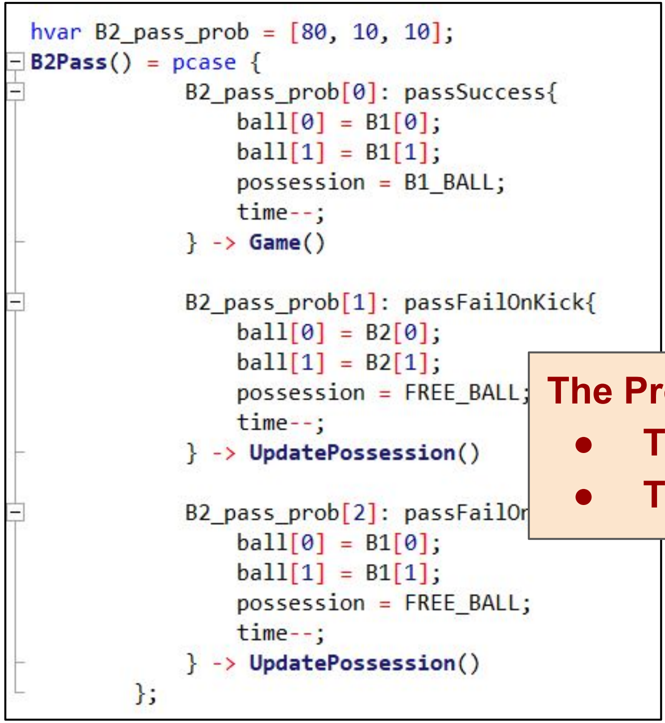
```
hvar B2_pass_prob = [80, 10, 10];
B2Pass() = pcase {
            B2_pass_prob[0]: passSuccess{
                ball[0] = B1[0];
                ball[1] = B1[1];
                possession = B1_BALL;
                time--;
            } -> Game()

            B2_pass_prob[1]: passFailOnKick{
                ball[0] = B2[0];
                ball[1] = B2[1];
                possession = FREE_BALL;
                time--;
            } -> UpdatePossession()

            B2_pass_prob[2]: passFailOnReceive{
                ball[0] = B1[0];
                ball[1] = B1[1];
                possession = FREE_BALL;
                time--;
            } -> UpdatePossession()
        };
```



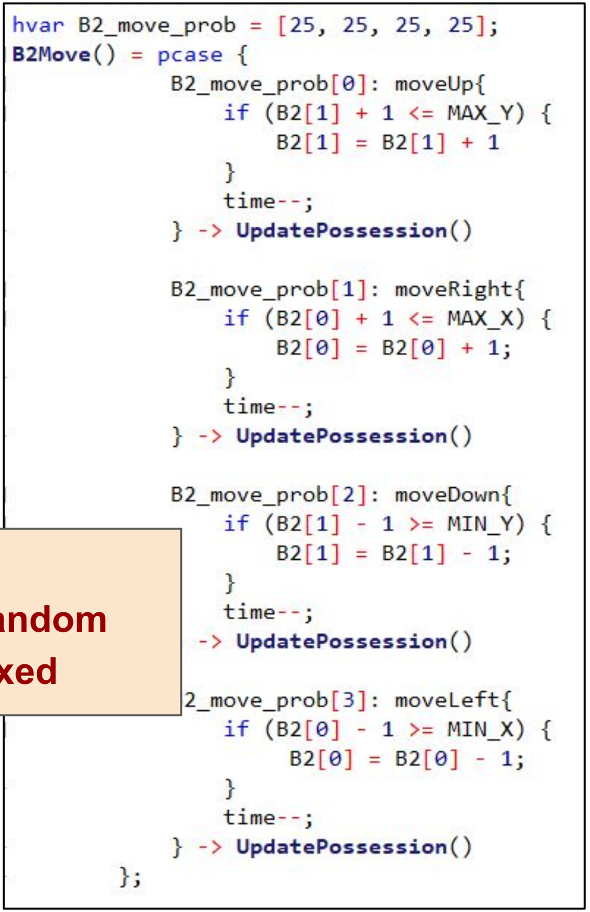
```
hvar B2_move_prob = [25, 25, 25, 25];
B2Move() = pcase {
            B2_move_prob[0]: moveUp{
                if (B2[1] + 1 <= MAX_Y) {
                    B2[1] = B2[1] + 1
                }
                time--;
            } -> UpdatePossession()

            B2_move_prob[1]: moveRight{
                if (B2[0] + 1 <= MAX_X) {
                    B2[0] = B2[0] + 1;
                }
                time--;
            } -> UpdatePossession()

            B2_move_prob[2]: moveDown{
                if (B2[1] - 1 >= MIN_Y) {
                    B2[1] = B2[1] - 1;
                }
                time--;
            } -> UpdatePossession()

            B2_move_prob[3]: moveLeft{
                if (B2[0] - 1 >= MIN_X) {
                     B2[0] = B2[0] - 1;
                }
                time--;
            } -> UpdatePossession()
        };
```

The Probabilities

Started with reasonable estimations for all players



```
hvar B2_pass_prob = [80, 10, 10];
B2Pass() = pcase {
            B2_pass_prob[0]: passSuccess{
                ball[0] = B1[0];
                ball[1] = B1[1];
                possession = B1_BALL;
                time--;
            } -> Game()

            B2_pass_prob[1]: passFailOnKick{
                ball[0] = B2[0];
                ball[1] = B2[1];
                possession = FREE_BALL;
                time--;
            } -> UpdatePossession()

            B2_pass_prob[2]: passFailOn
                ball[0] = B1[0];
                ball[1] = B1[1];
                possession = FREE_BALL;
                time--;
            } -> UpdatePossession()
        };
```



```
hvar B2_move_prob = [25, 25, 25, 25];
B2Move() = pcase {
            B2_move_prob[0]: moveUp{
                if (B2[1] + 1 <= MAX_Y) {
                    B2[1] = B2[1] + 1
                }
                time--;
            } -> UpdatePossession()

            B2_move_prob[1]: moveRight{
                if (B2[0] + 1 <= MAX_X) {
                    B2[0] = B2[0] + 1;
                }
                time--;
            } -> UpdatePossession()

            B2_move_prob[2]: moveDown{
                if (B2[1] - 1 >= MIN_Y) {
                    B2[1] = B2[1] - 1;
                }
                time--;
              -> UpdatePossession()

            2_move_prob[3]: moveLeft{
                if (B2[0] - 1 >= MIN_X) {
                     B2[0] = B2[0] - 1;
                }
                time--;
            } -> UpdatePossession()
        };
```

The Problem

- **The probabilities are random**
- **The probabilities are fixed**

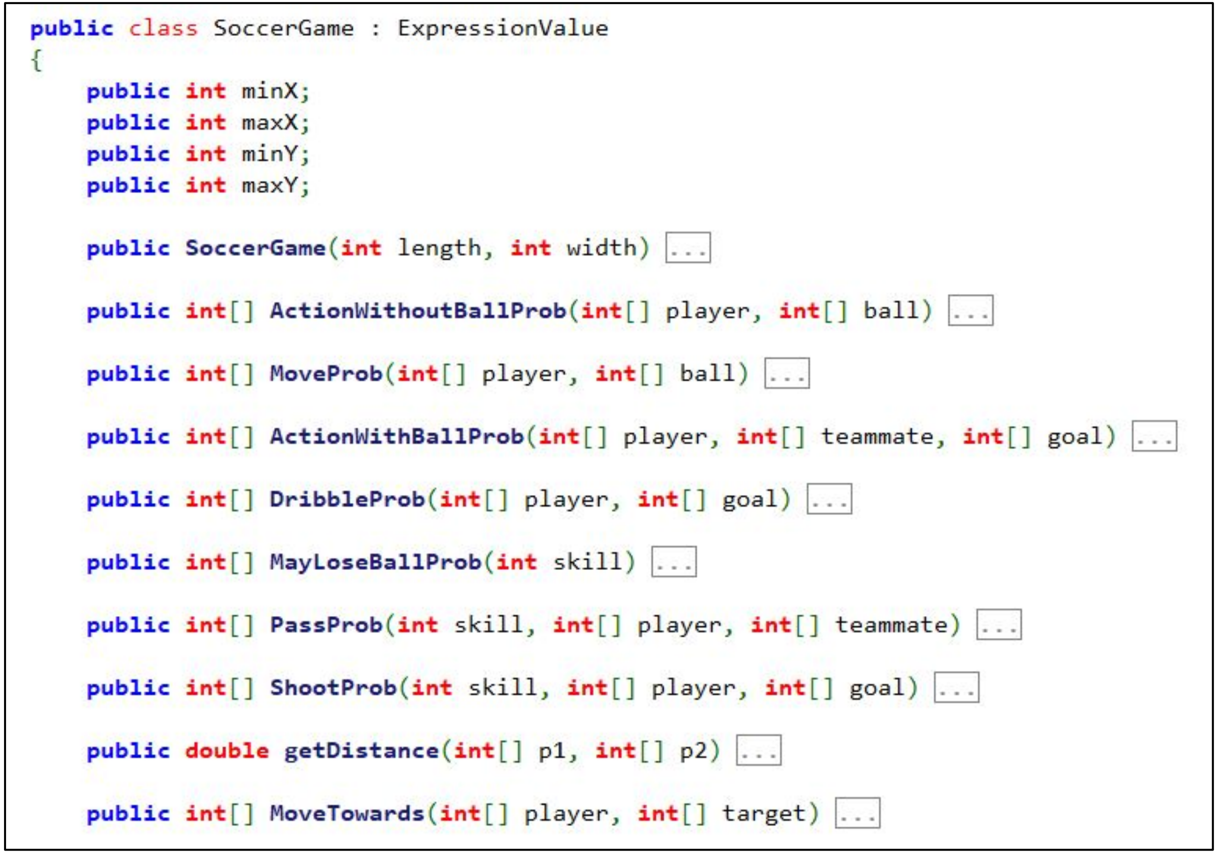
The Probabilities

Make teams and players “smarter” so that they can make non-random basic decisions

- **Build a C# library SoccerGame**
  - Contain methods that take in the current states of the model
  - Calculates and re-calculates the the probability before each **pcase**

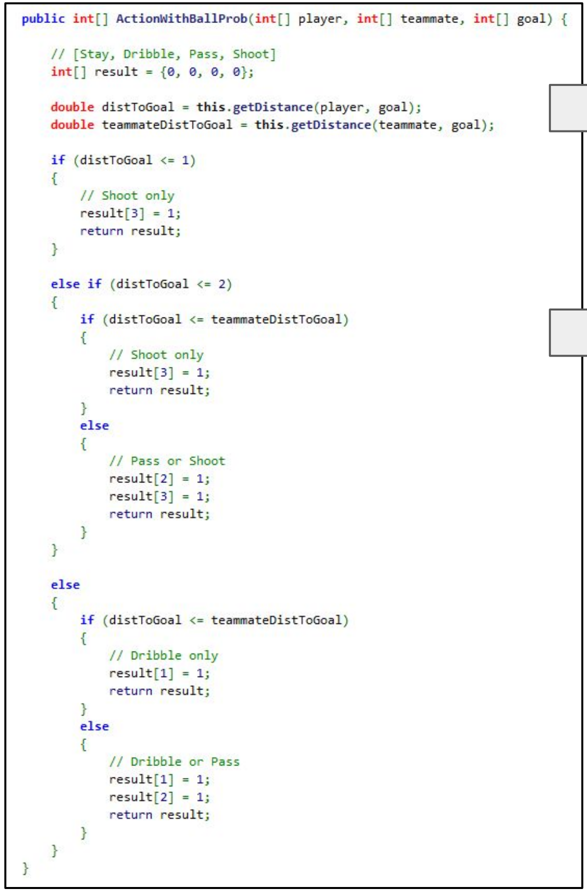
- **Introduce skill points for each player**
  - Dribble skill, passing skill, shooting skill
  - Used by **SoccerGame** library to calculate a reasonable effect that skill points could play

C# functions to manage probabilities



```
public class SoccerGame : ExpressionValue
{
    public int minX;
    public int maxX;
    public int minY;
    public int maxY;

    public SoccerGame(int length, int width) ...

    public int[] ActionWithoutBallProb(int[] player, int[] ball) ...

    public int[] MoveProb(int[] player, int[] ball) ...

    public int[] ActionWithBallProb(int[] player, int[] teammate, int[] goal) ...

    public int[] DribbleProb(int[] player, int[] goal) ...

    public int[] MayLoseBallProb(int skill) ...

    public int[] PassProb(int skill, int[] player, int[] teammate) ...

    public int[] ShootProb(int skill, int[] player, int[] goal) ...

    public double getDistance(int[] p1, int[] p2) ...

    public int[] MoveTowards(int[] player, int[] target) ...
```

C# function: Determine the player’s decision

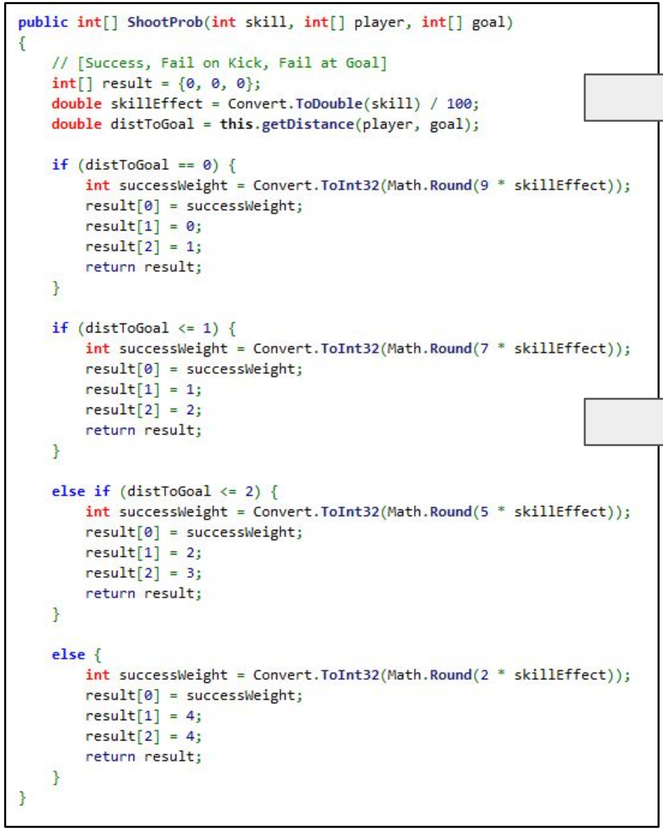


```
public int[] ActionWithBallProb(int[] player, int[] teammate, int[] goal) {

    // [Stay, Dribble, Pass, Shoot]
    int[] result = {0, 0, 0, 0};

    double distToGoal = this.getDistance(player, goal);
    double teammateDistToGoal = this.getDistance(teammate, goal);

    if (distToGoal <= 1)
    {
        // Shoot only
        result[3] = 1;
        return result;
    }

    else if (distToGoal <= 2)
    {
        if (distToGoal <= teammateDistToGoal)
        {
            // Shoot only
            result[3] = 1;
            return result;
        }
        else
        {
            // Pass or Shoot
            result[2] = 1;
            result[3] = 1;
            return result;
        }
    }

    else
    {
        if (distToGoal <= teammateDistToGoal)
        {
            // Dribble only
            result[1] = 1;
            return result;
        }
        else
        {
            // Dribble or Pass
            result[1] = 1;
            result[2] = 1;
            return result;
        }
    }
}
```

- Calculate distance to Goal
- Calculate teammate’s distance to Goal

Determine likelihood to dribble, pass or shoot

C# function: Calculate Probabilities when shooting



```
public int[] ShootProb(int skill, int[] player, int[] goal)
{
    // [Success, Fail on Kick, Fail at Goal]
    int[] result = {0, 0, 0};
    double skillEffect = Convert.ToDouble(skill) / 100;
    double distToGoal = this.getDistance(player, goal);

    if (distToGoal == 0) {
        int successWeight = Convert.ToInt32(Math.Round(9 * skillEffect));
        result[0] = successWeight;
        result[1] = 0;
        result[2] = 1;
        return result;
    }

    if (distToGoal <= 1) {
        int successWeight = Convert.ToInt32(Math.Round(7 * skillEffect));
        result[0] = successWeight;
        result[1] = 1;
        result[2] = 2;
        return result;
    }

    else if (distToGoal <= 2) {
        int successWeight = Convert.ToInt32(Math.Round(5 * skillEffect));
        result[0] = successWeight;
        result[1] = 2;
        result[2] = 3;
        return result;
    }

    else {
        int successWeight = Convert.ToInt32(Math.Round(2 * skillEffect));
        result[0] = successWeight;
        result[1] = 4;
        result[2] = 4;
        return result;
    }
}
```

- Calculate distance to Goal
- Consider the effect of player’s shooting skill

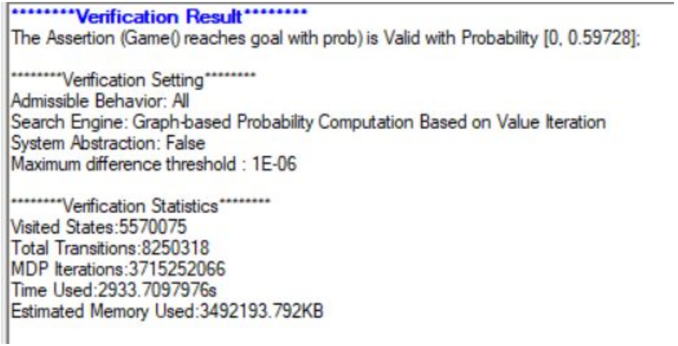
Determine the likelihood of successfully scoring versus losing the ball on kick or missing the goal

Current Situation of the Model

- Team A uses the SoccerGame C# Library to calculate the probabilities
- Team B continued with fixed reasonable estimates

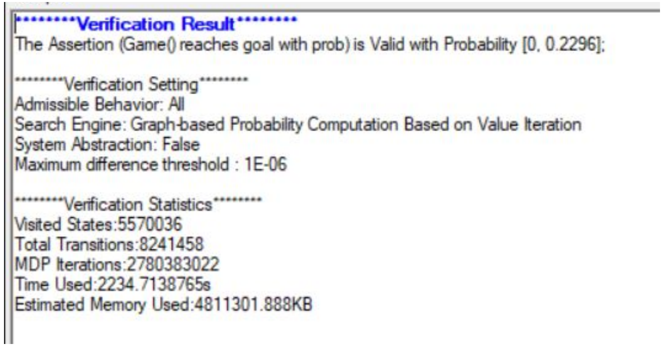
Comparing the probabilities of each team reaching 1 goal within a period of time ...

Team A



```
********Verification Result********
The Assertion (Game() reaches goal with prob) is Valid with Probability [0, 0.59728];

********Verification Setting********
Admissible Behavior: All
Search Engine: Graph-based Probability Computation Based on Value Iteration
System Abstraction: False
Maximum difference threshold : 1E-06

********Verification Statistics********
Visited States:5570075
Total Transitions:8250318
MDP Iterations:3715252066
Time Used:2933.7097976s
Estimated Memory Used:3492193.792KB
```

Team B



```
********Verification Result********
The Assertion (Game() reaches goal with prob) is Valid with Probability [0, 0.2296];

********Verification Setting********
Admissible Behavior: All
Search Engine: Graph-based Probability Computation Based on Value Iteration
System Abstraction: False
Maximum difference threshold : 1E-06

********Verification Statistics********
Visited States:5570036
Total Transitions:8241458
MDP Iterations:2780383022
Time Used:2234.7138765s
Estimated Memory Used:4811301.888KB
```

Potential Future Improvements

- Improve the estimation of the probabilities
  - Take into account more variables i.e. opponent players’ positions
  - Possible to use 2 separate C# libraries - representing each team’s differing strategies.

- Scale to more players (preferably 11 players per team)
  - Would require refactoring of the CSP# codebase.
    - The identities/location of the players need to be parameterized in process, instead of separate process for each player

Potential Future Improvements

- Add more scenarios to make it a more realistic soccer game
    - e.g. out of bound, corner kicks etc.

- Calculate some interesting statistics
    - E.g. Number of times possession changes, number of passes before goal etc.

- Reducing state space
    - Find a way to model a team as a whole instead of each player

Thank You

Project source code

https://github.com/yjpan47/soccer-pmc