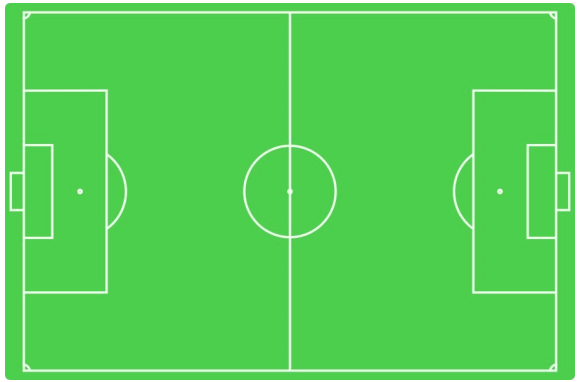
CS3211 Final Presentation

Individual Project: Pan Yongjing



Applying PMC in Soccer Analytics

Project source code

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

Contents

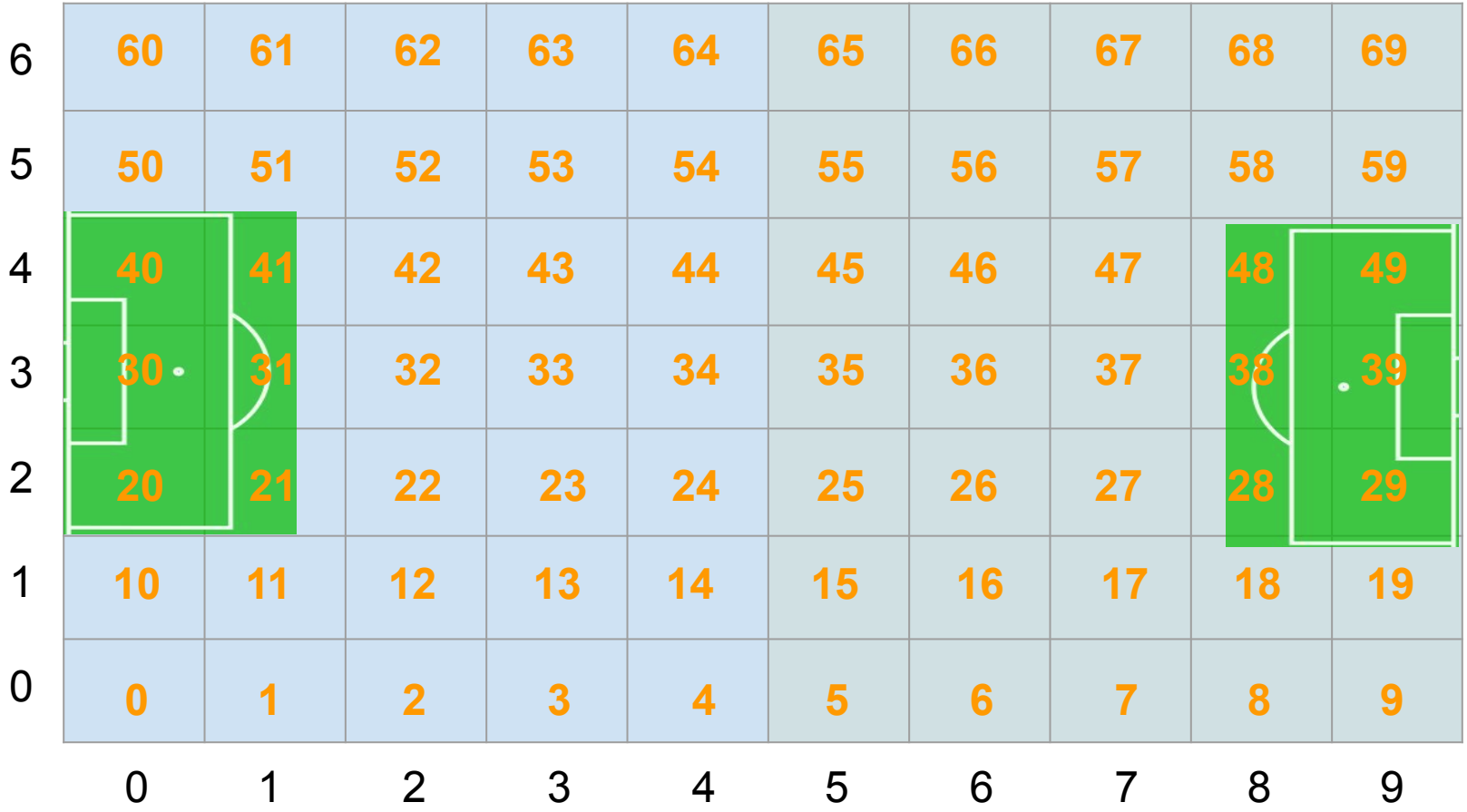
A month ago...

- 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.

A month ago...

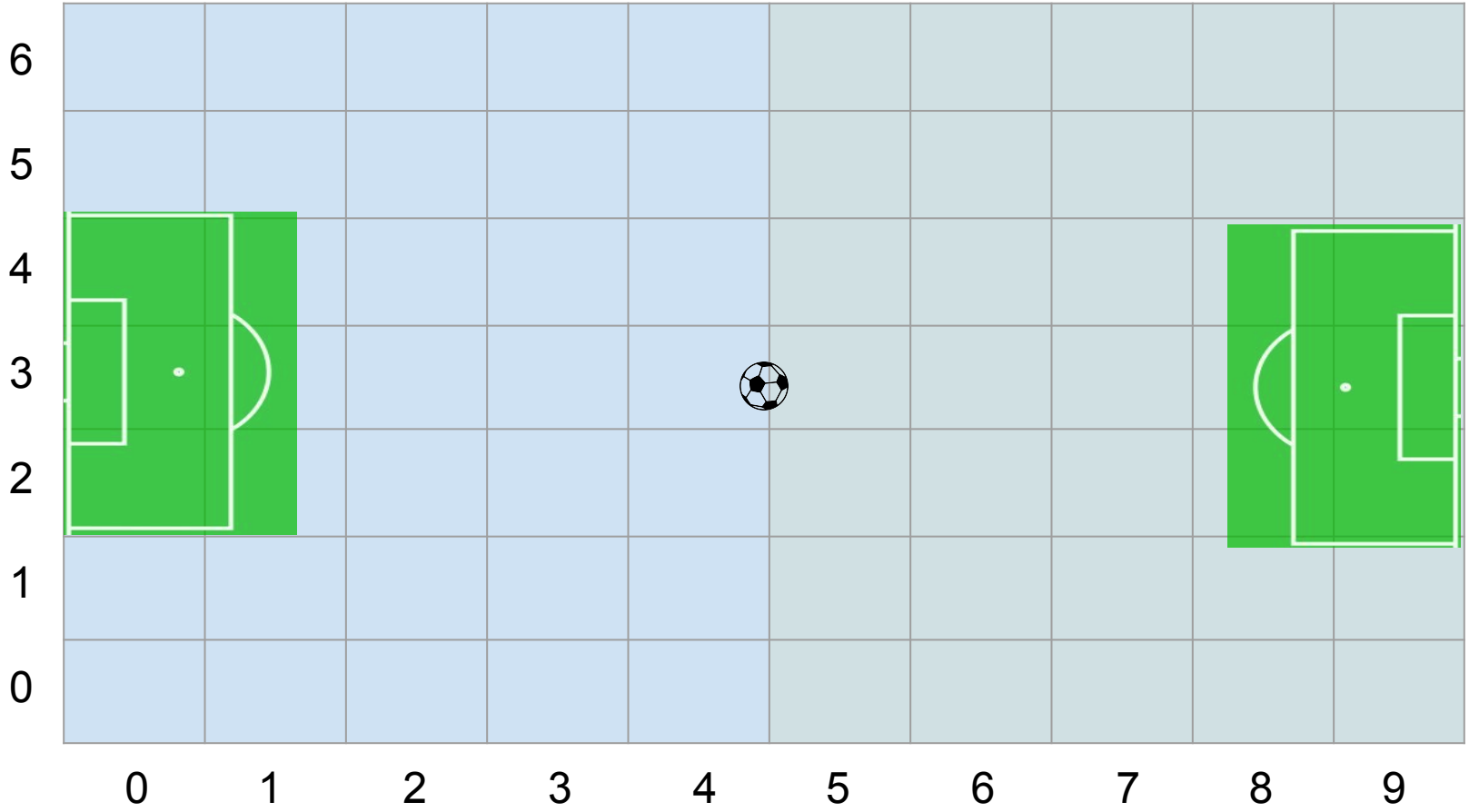
Current Model

10 x 7 = 70 coordinate locations



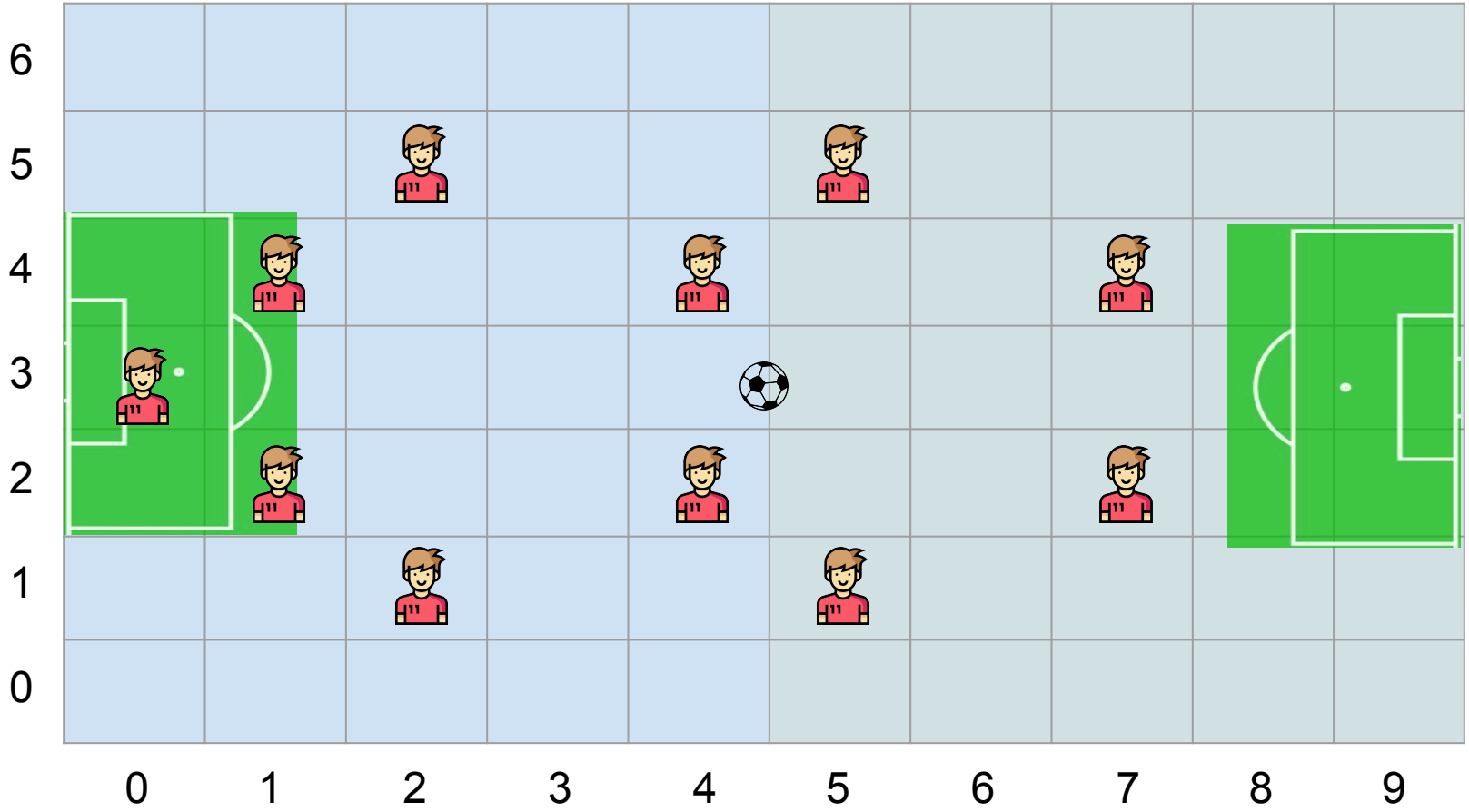
Current Model

One ball, Two teams



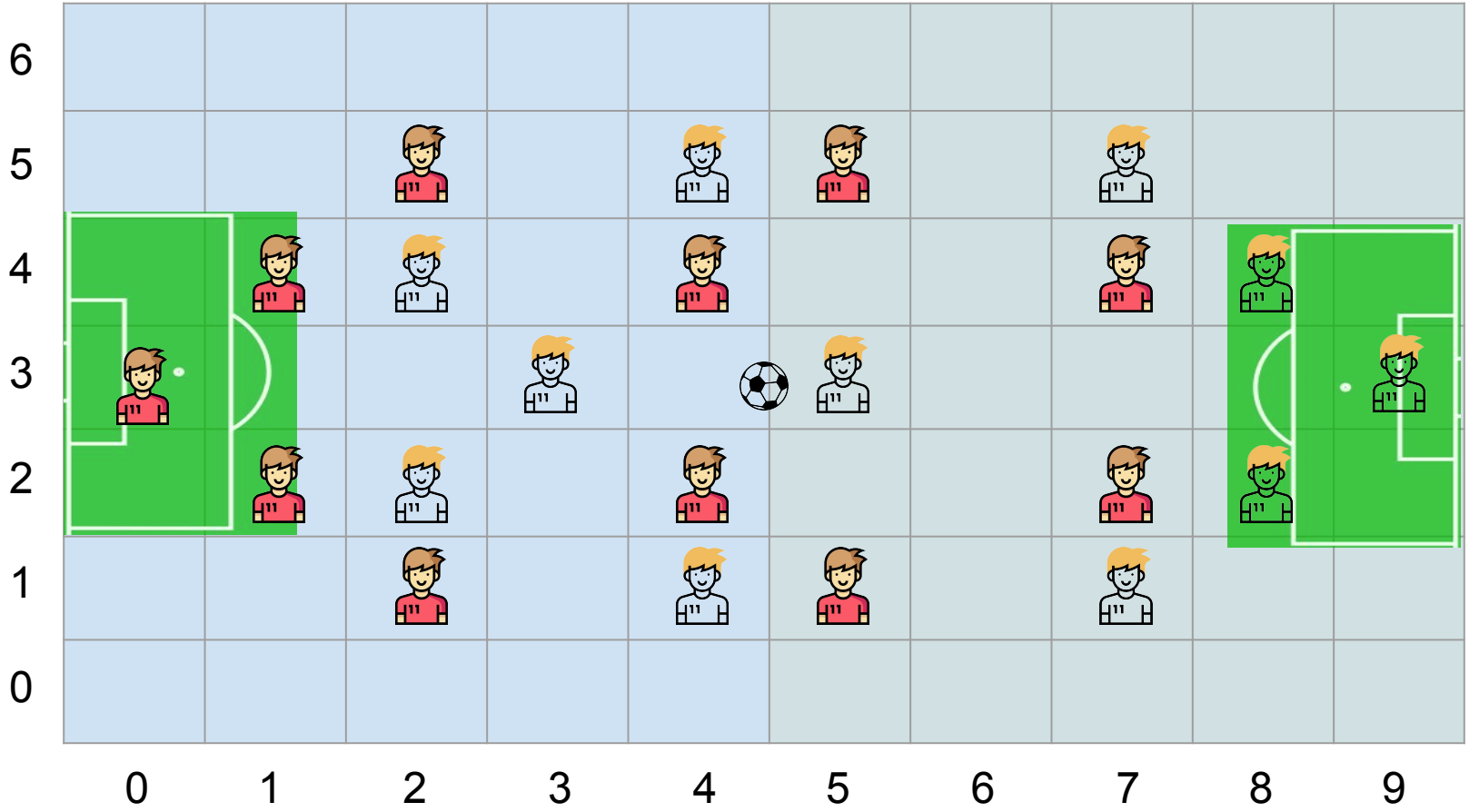
Current Model

11 players per team



Current Model

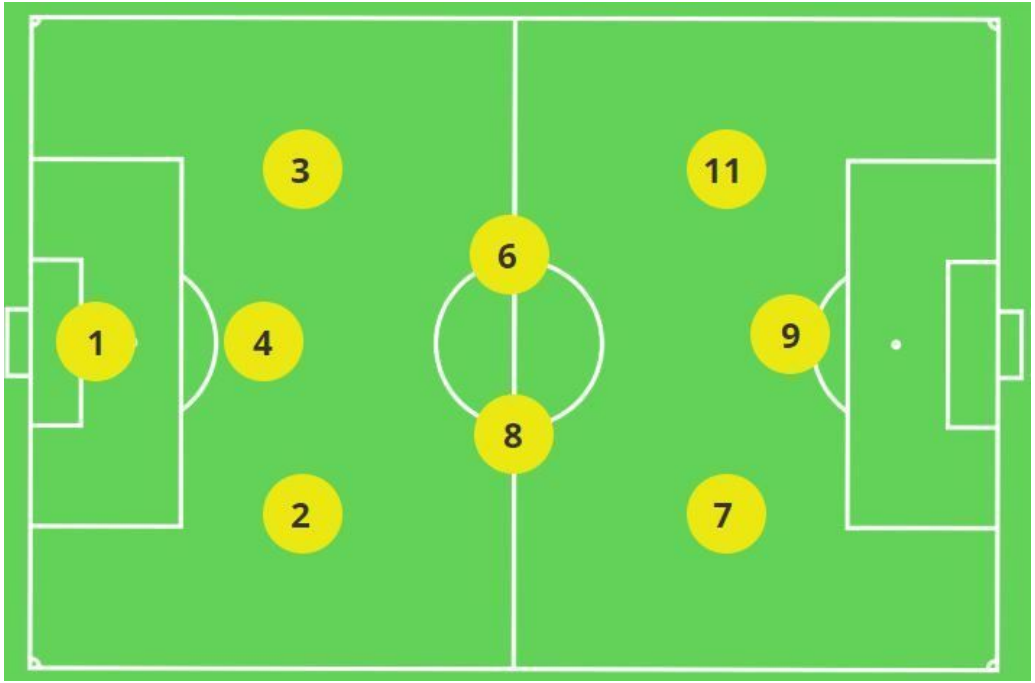
22 players in total



To container the state space

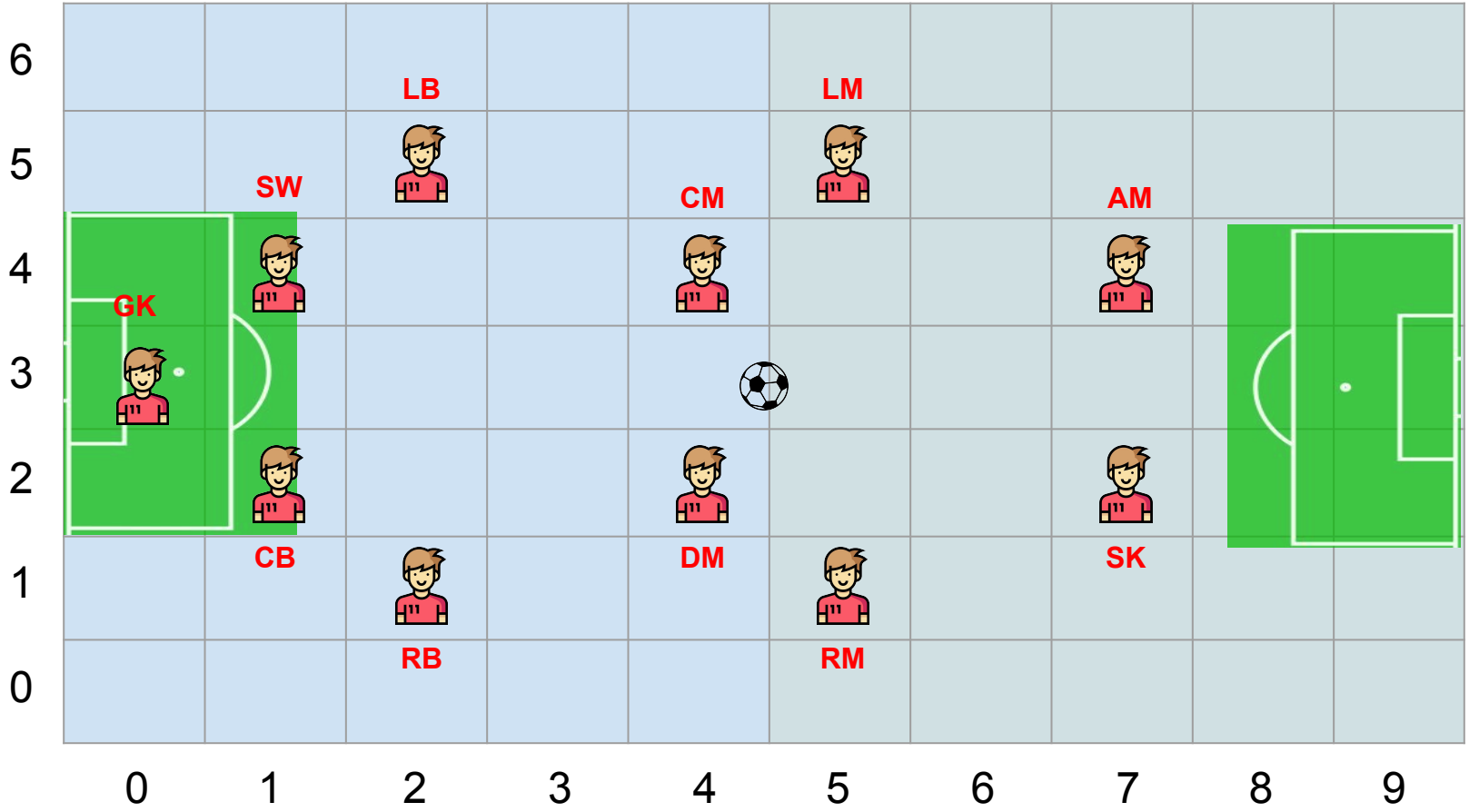
- Reduce the locations allowed for each player to move to
  - Players are assigned to specific positions and are only allowed to operate within specific zones

1	Goalkeeper
2	Right Fullback
3	Left Fullback
4	Center Back
5	Sweeper
6	Defending Midfielder
7	Right Midfielder
8	Central Midfielder
9	Striker
10	Attacking Midfielder
11	Left Midfielder



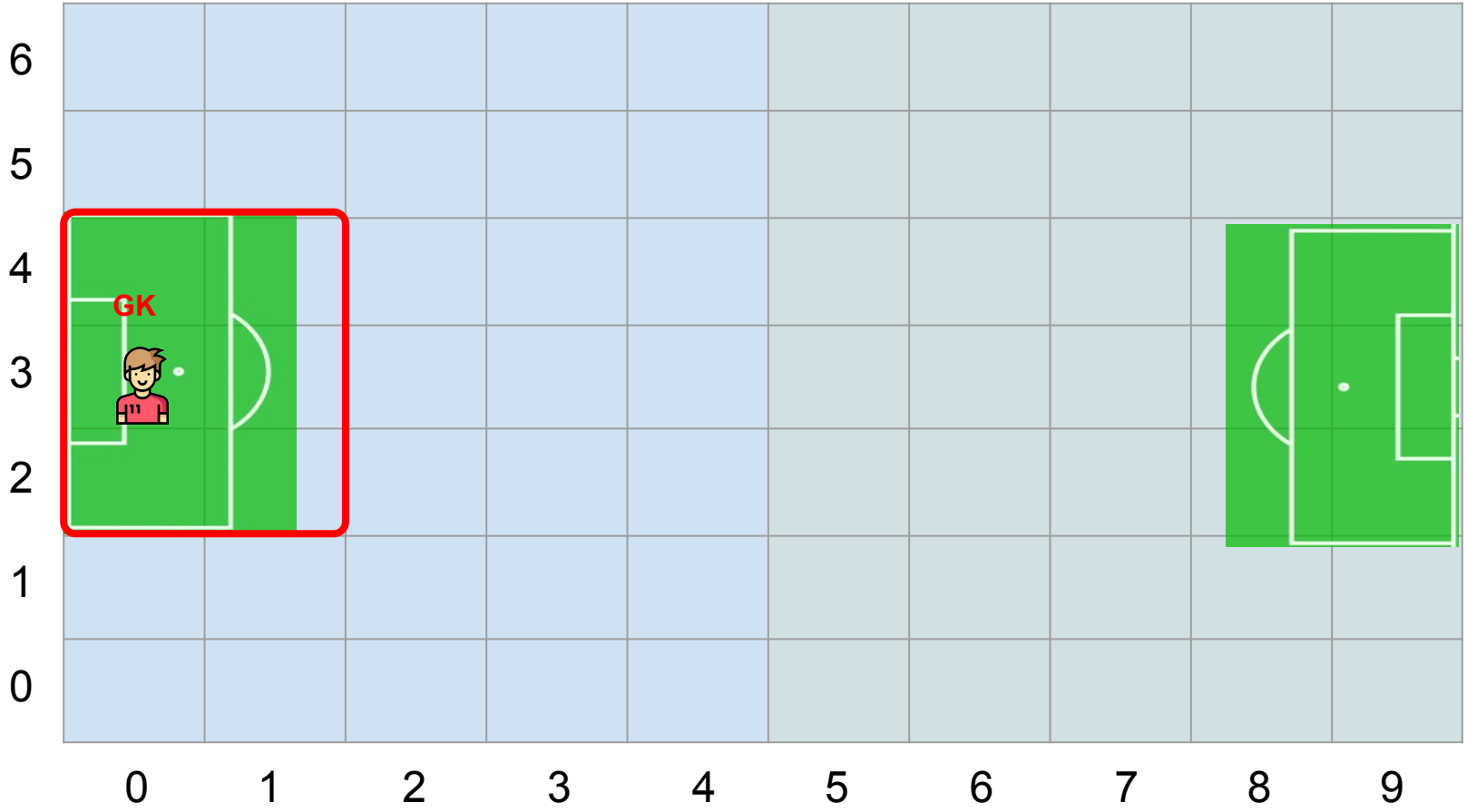
Current Model

11 players on each team



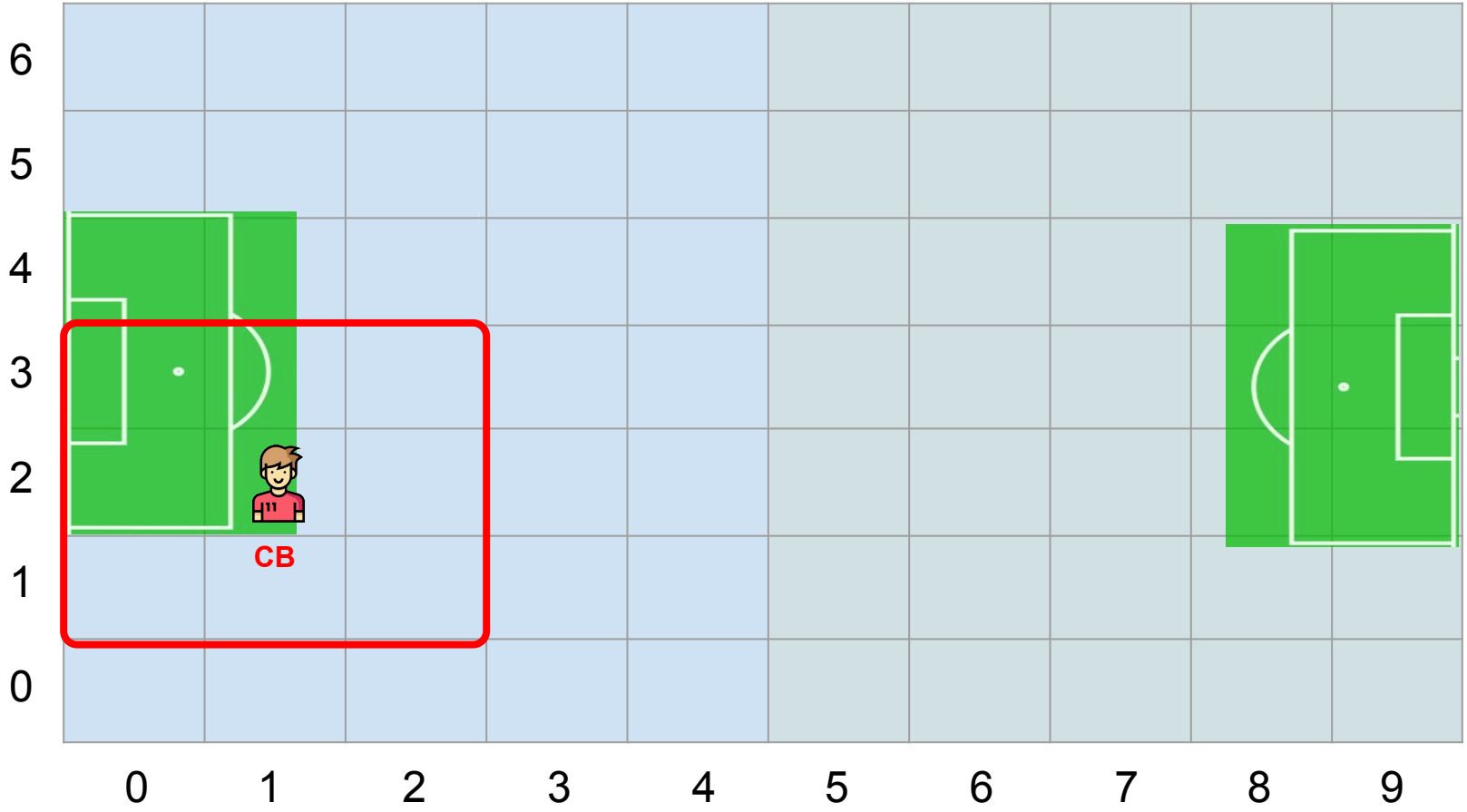
Current Model

11 players on each team



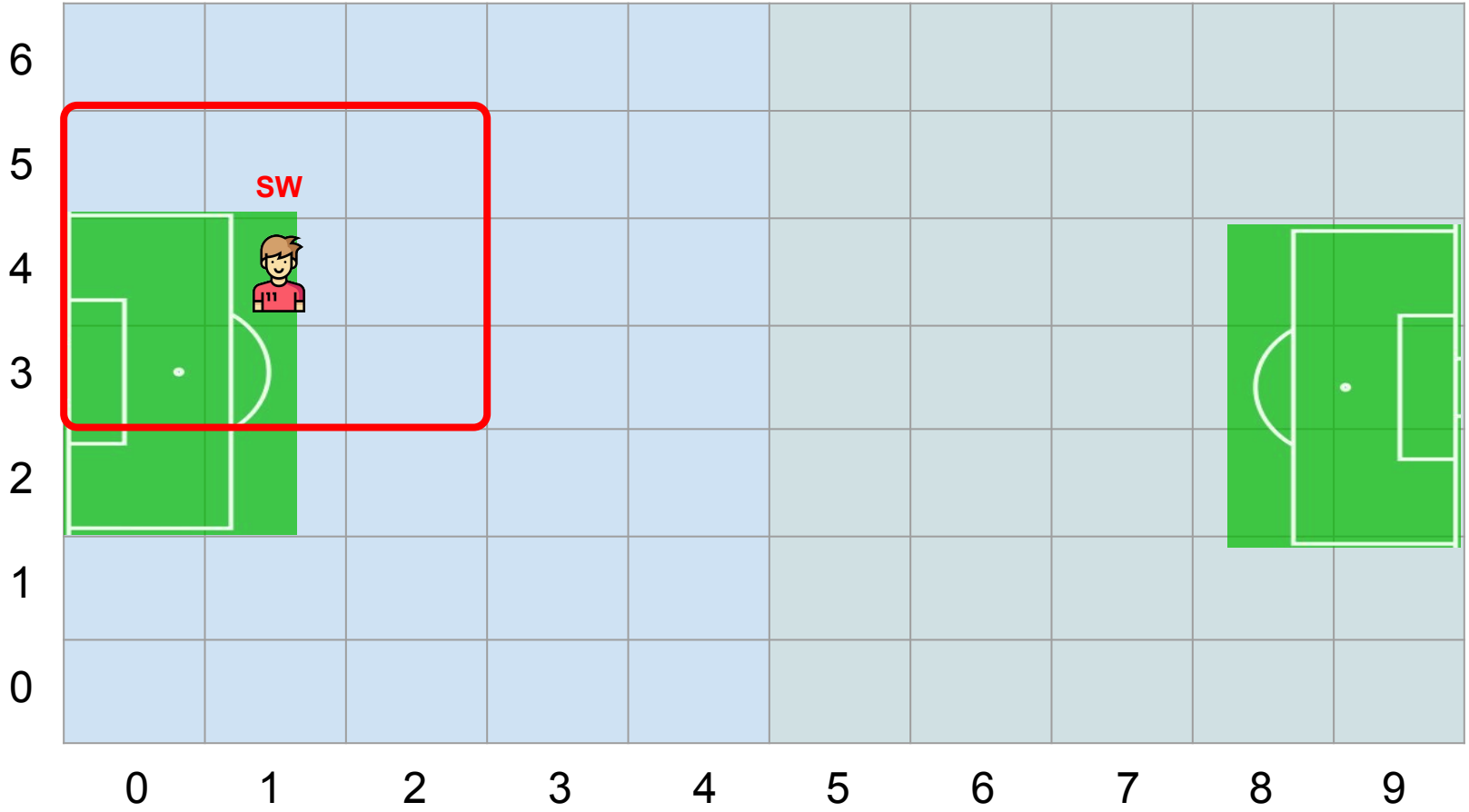
Current Model

11 players on each team



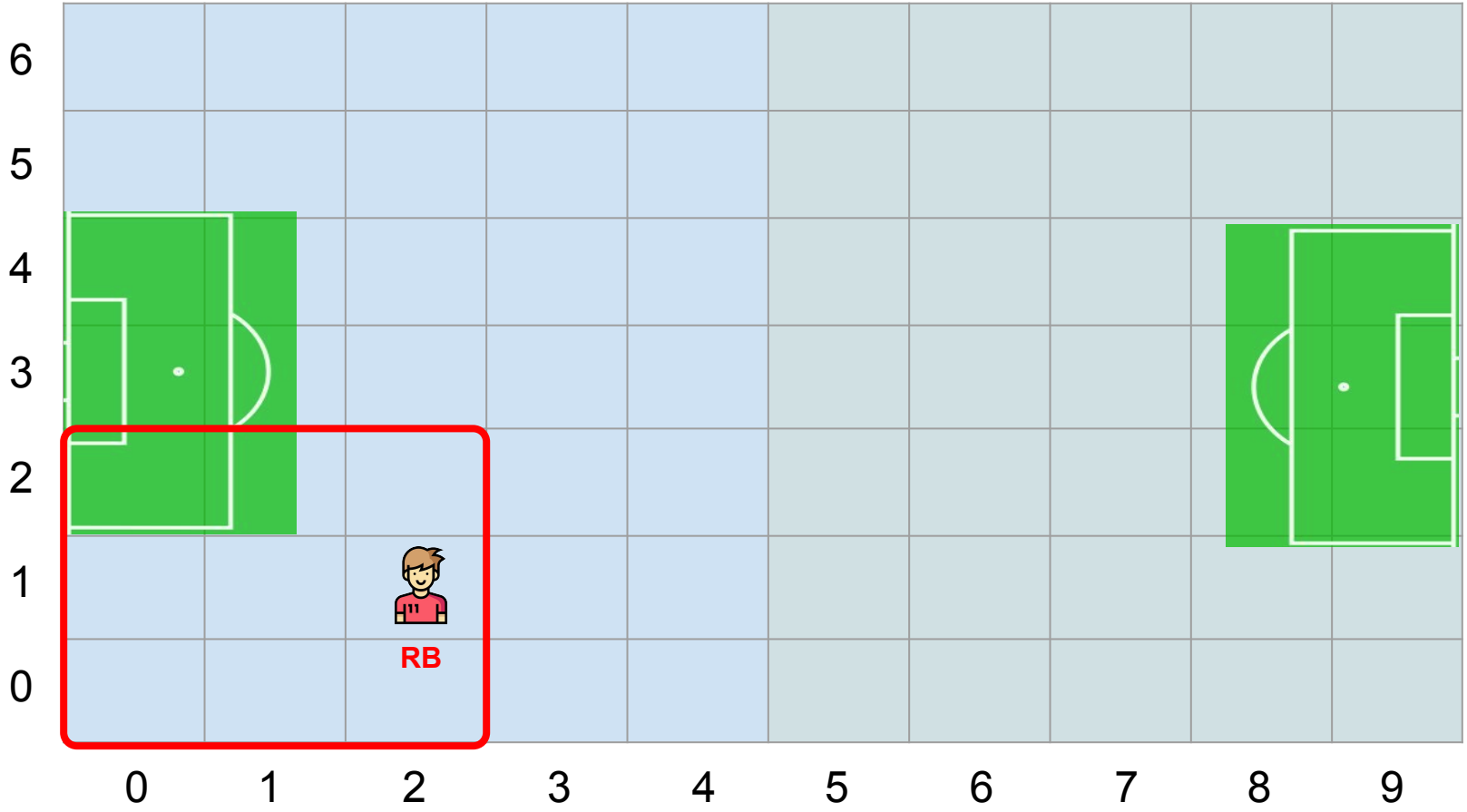
Current Model

11 players on each team



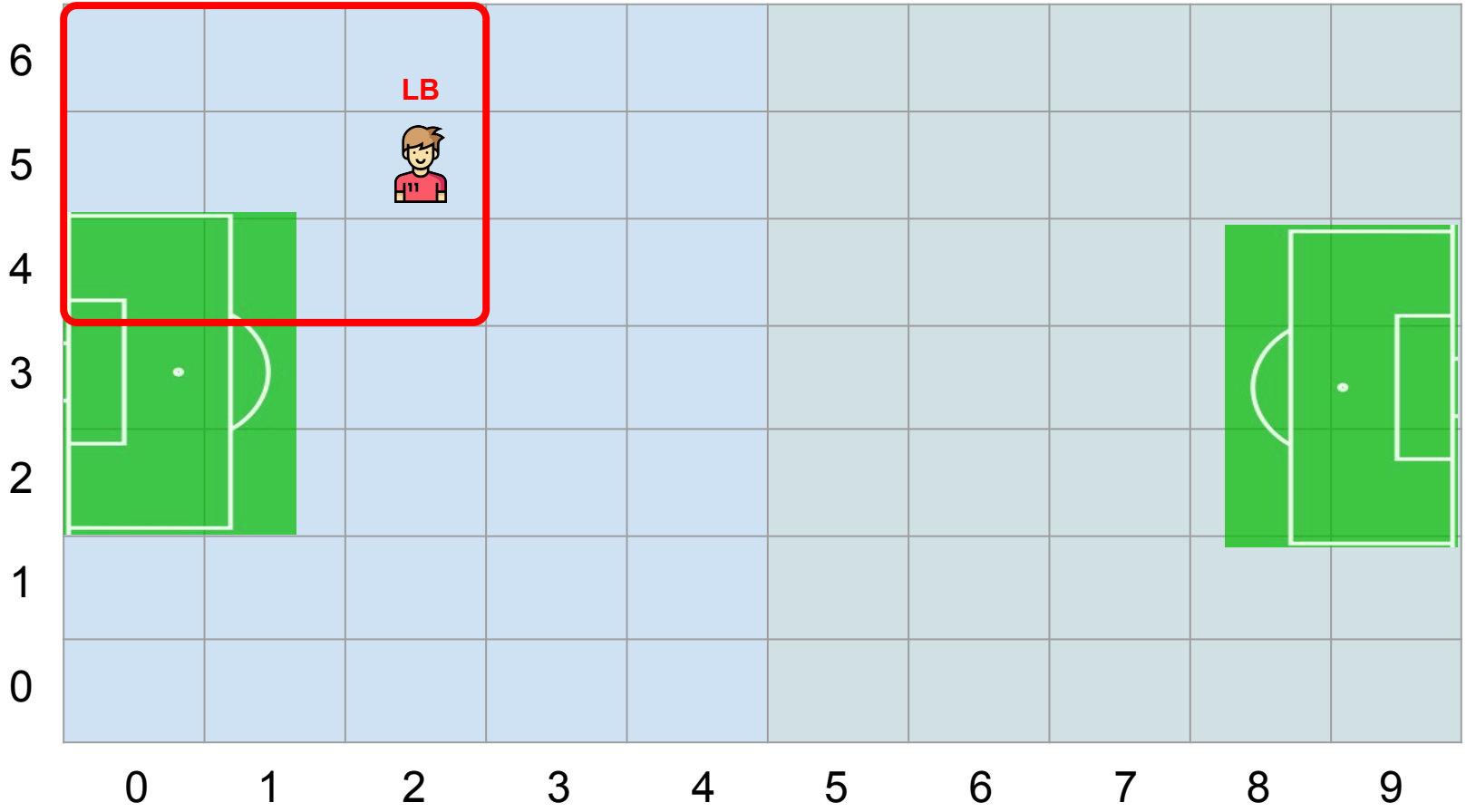
Current Model

11 players on each team



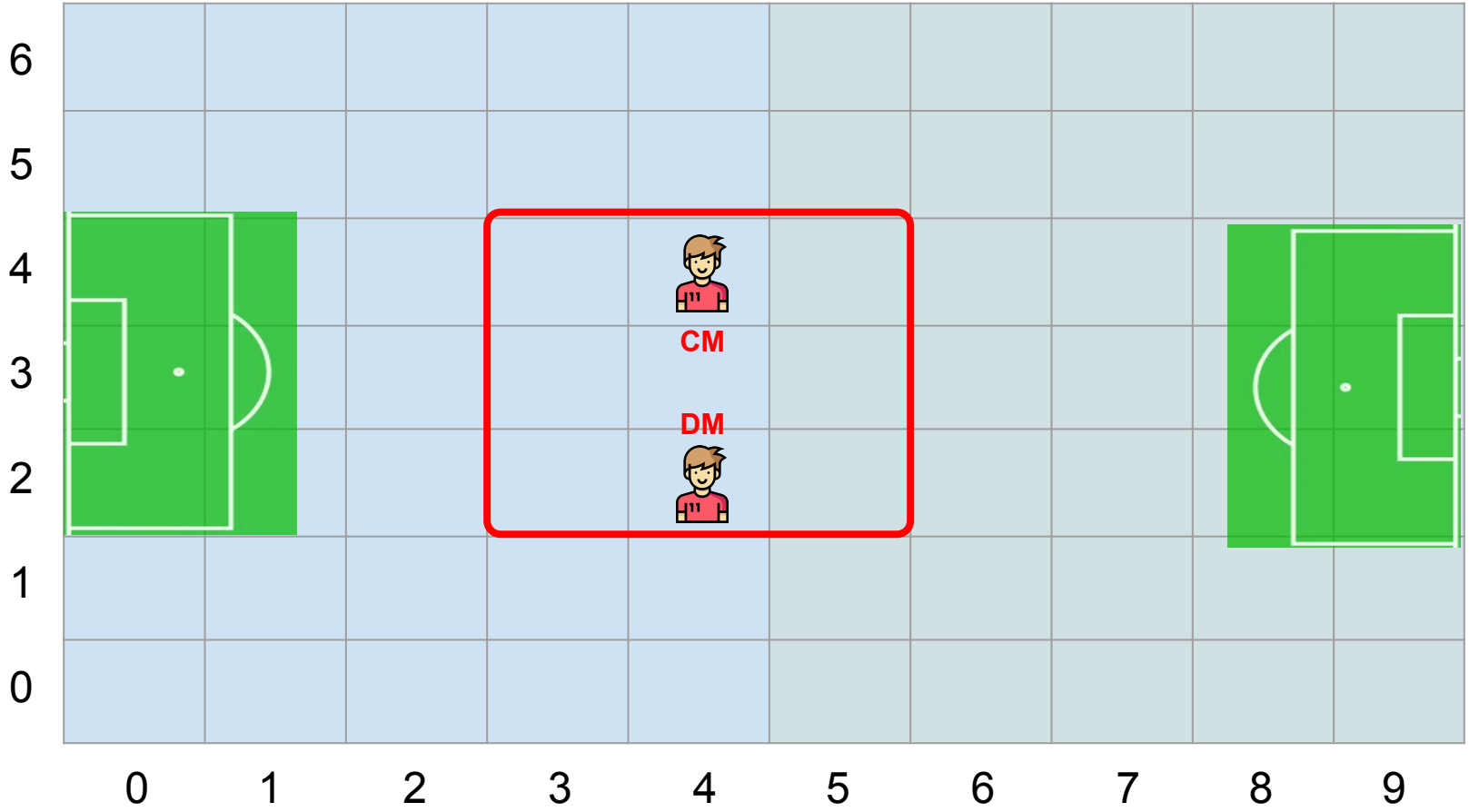
Current Model

11 players on each team



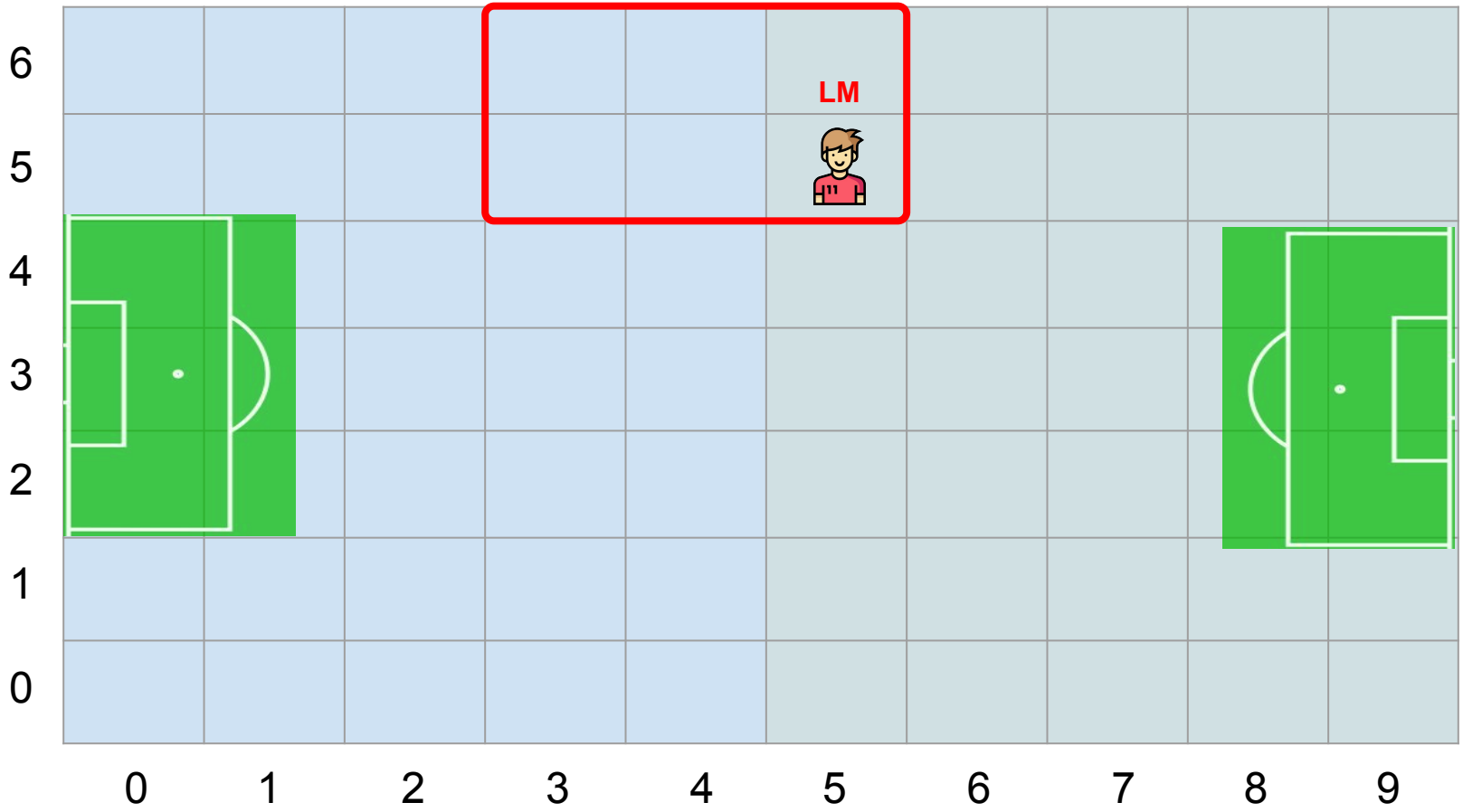
Current Model

11 players on each team



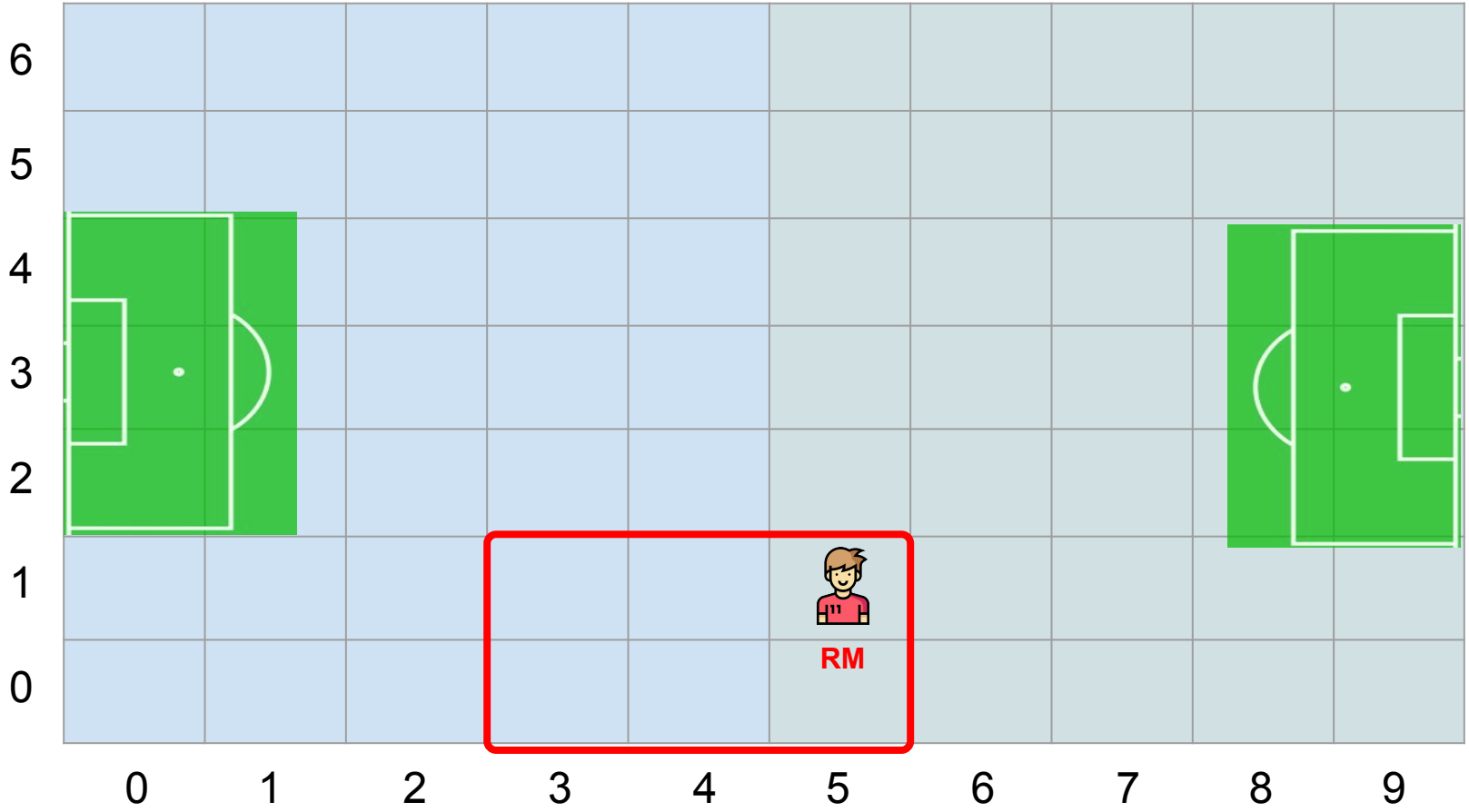
Current Model

11 players on each team



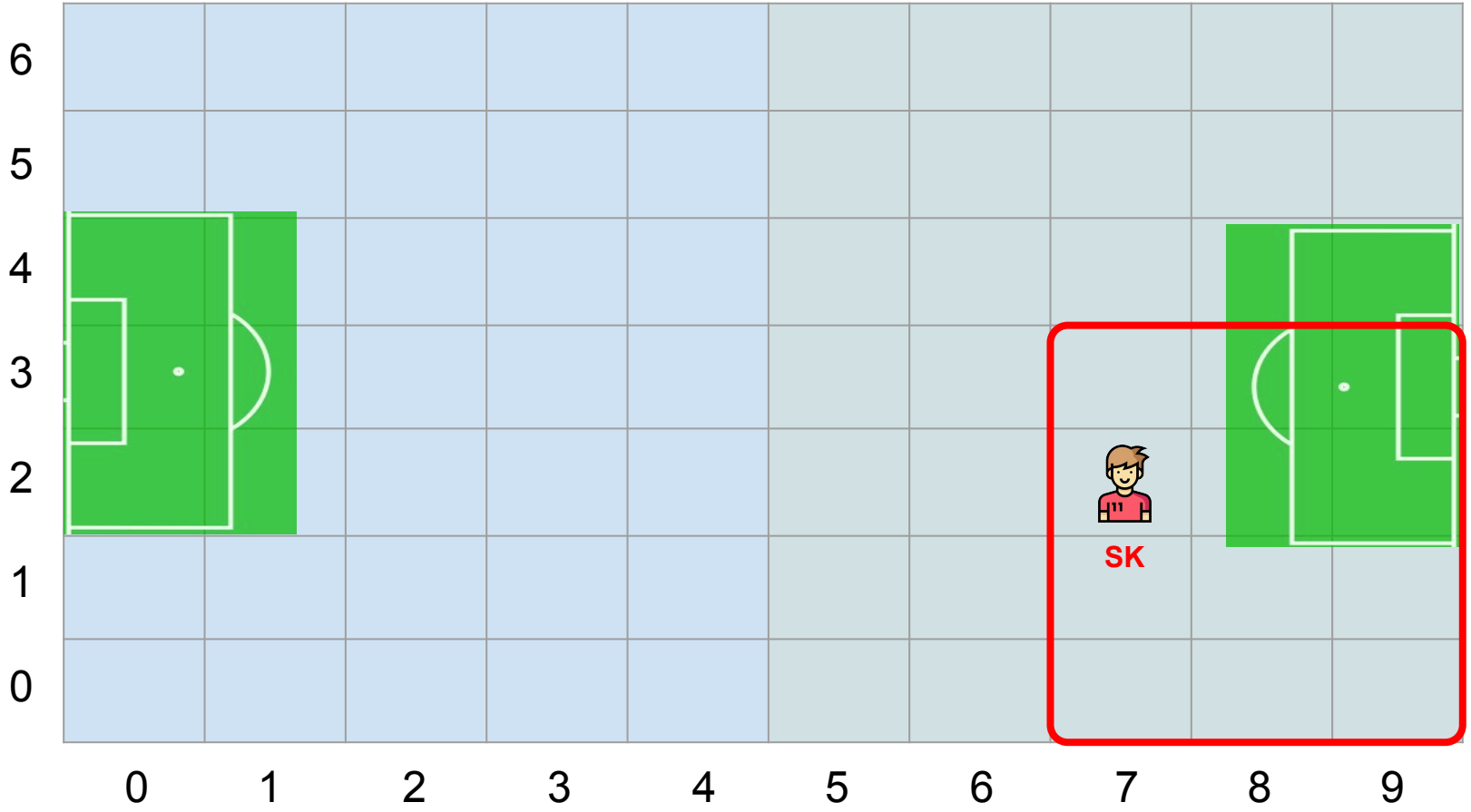
Current Model

11 players on each team



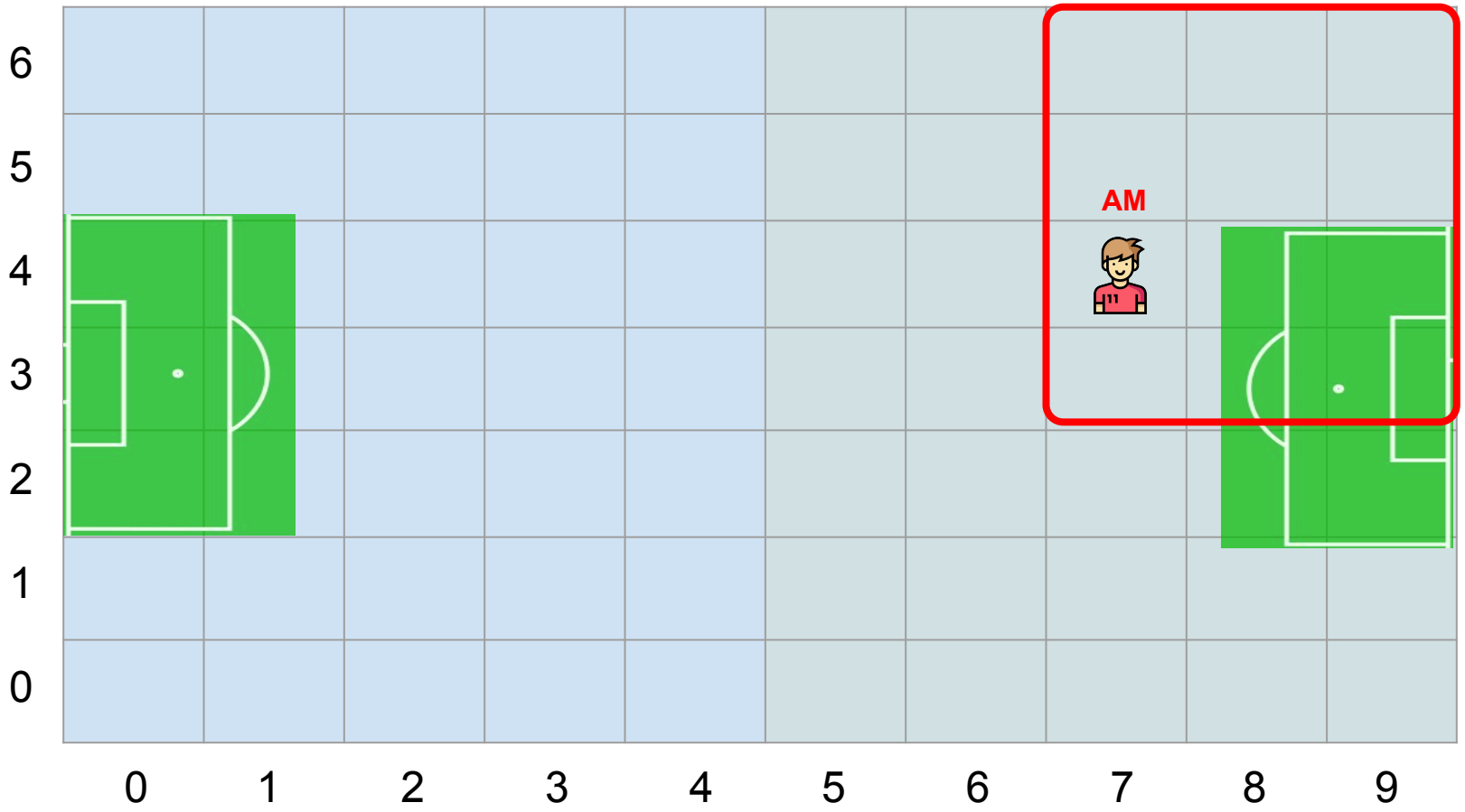
Current Model

11 players on each team



Current Model

11 players on each team



The States of the Model

1. The locations of each of the 22 players
2. The location of the ball
3. The possession of the ball

```
# import "PAT.Lib.SoccerManager";

var <GameManager> gameManager = new GameManager();
var<ManagerA> managerA = new ManagerA();
var<ManagerB> managerB = new ManagerB();

var teamA = managerA.getPlayerStartingLocations();
var teamB = managerB.getPlayerStartingLocations();

var ball = gameManager.getStartingBallLocation();
var possession = gameManager.getStartingPossession();
```

```
The environment is:
Variables:
ball=35;
teamA=[30, 12, 52, 21, 41, 24, 5, 44, 28, 48, 55];
possession=[-1, -1];
teamB=[30, 12, 52, 21, 41, 24, 5, 44, 28, 48, 55];
```

The States of the Model

- The location of a player or the ball is stored an integer in [0, 69]
  - They are converted to xy-coordinates in the C# libraries for computations
- The order of the arrays teamA and teamB are based on the position of the player
  - For example, teamA[0] = 30 means that the Goalkeeper (Position 0) is at the location 30 which represent xy-coordinate (0, 3).

```
The environment is:
Variables:
ball=35;
teamA=[30, 12, 52, 21, 41, 24, 5, 44, 28, 48, 55];
possession=[-1, -1];
teamB=[30, 12, 52, 21, 41, 24, 5, 44, 28, 48, 55];
```

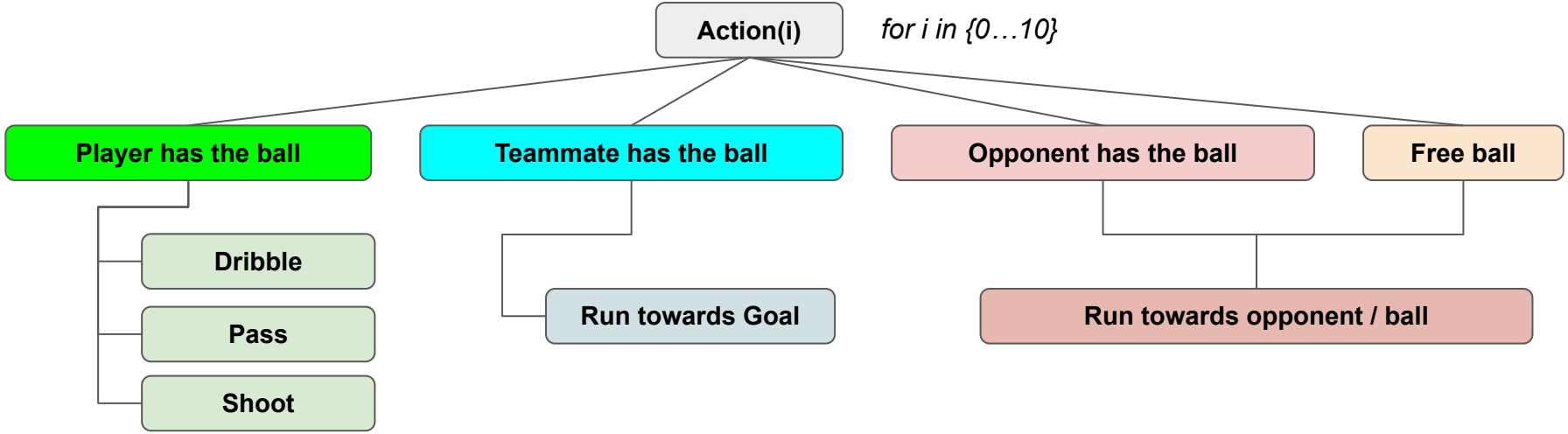
- possession[0] indicate which team has possession.
  - -1 means free ball
  - 0 means Team A
  - 1 means Team B
- possession[1] indicate position of the player which has possession.

The States of the Model

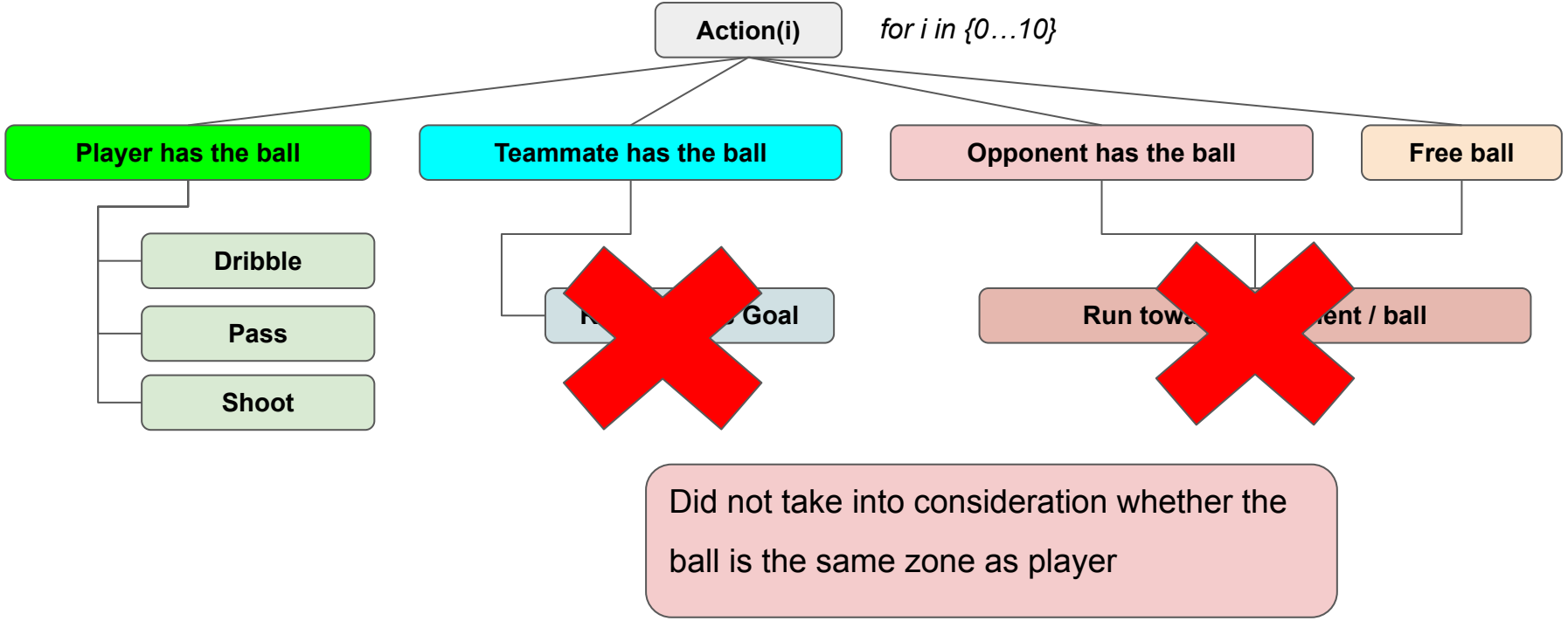
1. The locations (xy-coordinates) of each of the 4 players
2. The location (xy-coordinate) of the ball
3. The possession of the ball

```
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;
```

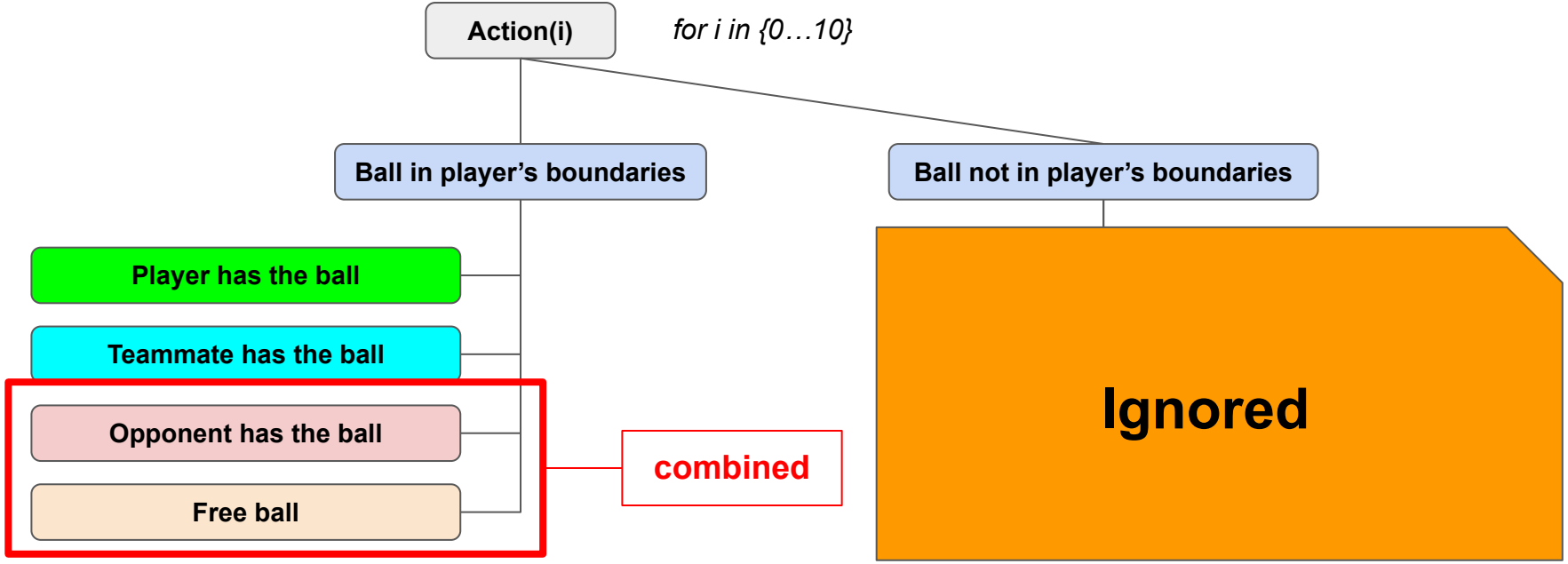
The Events of the Model



The Events of the Model



The Events of the Model

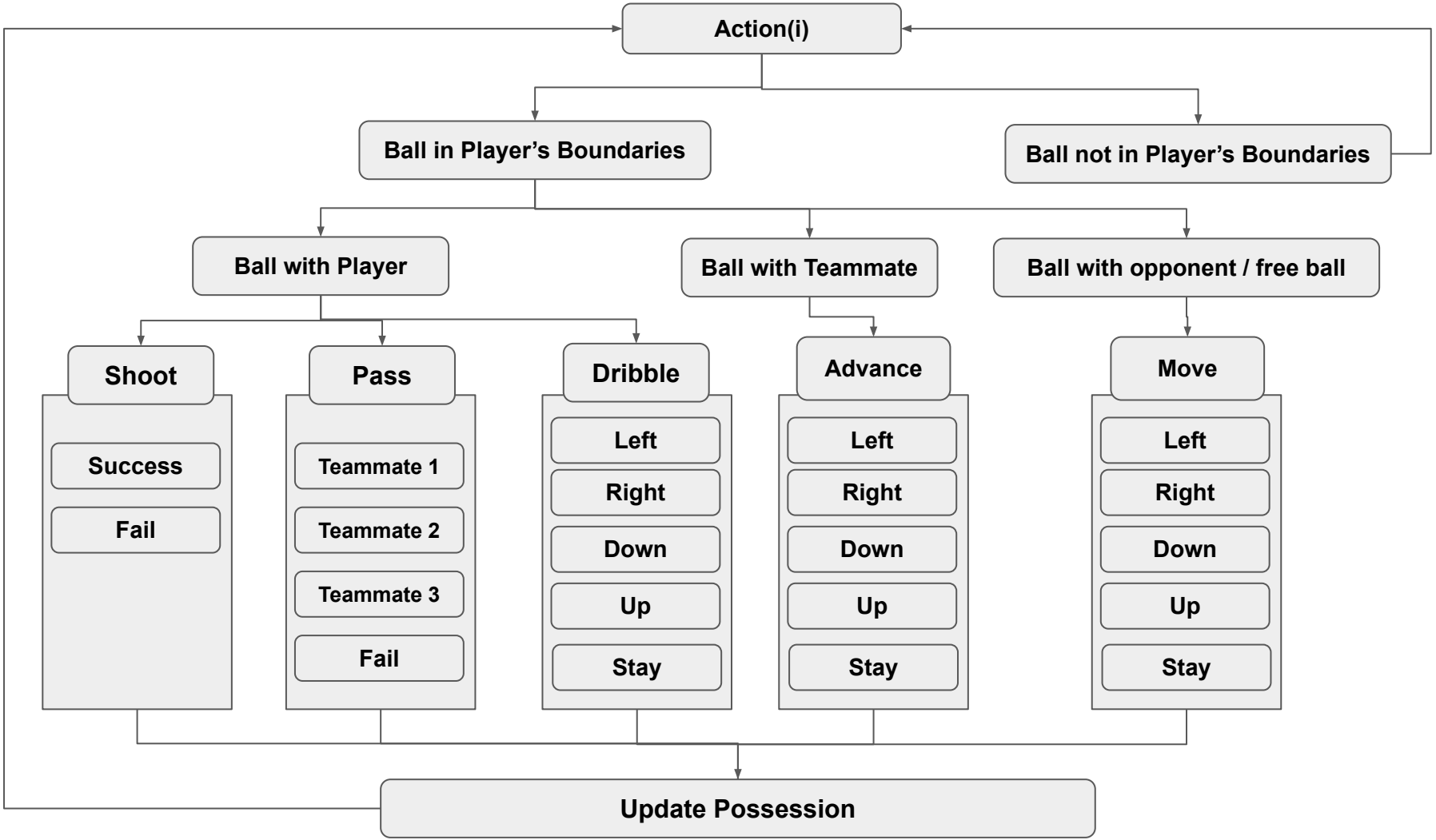


- **A player needs to be in the same zone as the ball for it to carry out any action**

The Events of the Model

Action-loop of a player

- Ball not in the player’s zone
  - Skip
- Ball in the player’s zone
  - Player possesses the ball
    - Dribble the ball (left/right/up/down)
    - Pass the ball to a teammate (can be from another zone)
    - Shoot the ball to the goal
  - Teammate possess the ball
    - Advance in the direction of the goal
  - Opponent possess the ball
    - Move in the direction of the ball


Action(i)
Ball in Player's Boundaries
Ball not in Player's Boundaries
Ball with Player
Ball with Teammate
Ball with opponent / free ball
Shoot
Success
Fail
Pass
Teammate 1
Teammate 2
Teammate 3
Fail
Dribble
Left
Right
Down
Up
Stay
Advance
Left
Right
Down
Up
Stay
Move
Left
Right
Down
Up
Stay
Update Possession

Extensive use of C# Libraries

Three main C# classes created in SoccerManager

- **ManagerA & Manager B**
  - Keep track of constants like zone boundaries for each player
  - Calculate probabilities for the **pcase**
  - Skills (**Shooting**, **Passing**, **Ball Handling**, **Defense**) of each player
    - Affects the probabilities calculated
  - Makes complex decisions
    - Whether to shoot or pass based on distance to goal or teammate
- **GameManager**
  - Complex calculations that are not the direct responsibility of the two team Managers
    - E.g. updating of possession

ManagerA and ManagerB

```
enum Position
{
    Goalkeeper,
    RightFullback,
    LeftFullback,
    CenterBack,
    Sweeper,
    DefensiveMidfielder,
    RightMidfielder,
    CentralMidfielder,
    Striker,
    AttackingMidfielder,
    LeftMidfielder
}

enum Skill
{
    Shooting,
    Passing,
    Speed,
    BallHandling,
    Defence,

}

enum Boundary
{
    minX,
    maxX,
    minY,
    maxY,
}
```

```
public class ManagerA : ExpressionValue
{
    private int fieldMinX;
    private int fieldMaxX;
    private int fieldMinY;
    private int fieldMaxY;
    private int[] goalCoordinate;
    private Dictionary<Position, Dictionary<Skill, int>> playerStats;
    private Dictionary<Position, Dictionary<Boundary, int>> playerBoundaries;
    private Dictionary<Position, List<Position>> playerPassingTargets;

    public ManagerA()
    {
        this.setGoal();
        this.setFieldBoundaries();
        this.setPlayerBoundaries();
        this.setPlayerPassingTargets();
        this.setPlayerStats();
    }

    private void setFieldBoundaries()
    {
        this.fieldMinX = 0;
        this.fieldMaxX = 9;
        this.fieldMinY = 0;
        this.fieldMaxY = 6;
    }

    private void setGoal() {
        this.goalCoordinate = new int[] {0, 3};
    }

    private void setPlayerStats()
    ...

    private void setPlayerBoundaries()
    ...

    private void setPlayerPassingTargets()
    ...

    public int getGoalLocation()
    ...

    public int[] getPlayerStartingLocations()
    ...
```

Manage the team's behaviour and characteristics

- The boundaries of the field
- The zone boundaries for each player
- The passing targets for each player
- The personal stats of each player
- The starting default location and position of each player

Team Settings

```
private void setPlayerStats()
{
    this.playerStats = new Dictionary<Position, Dictionary<Skill, int>>();

    // GoalKeeper
    this.playerStats[Position.Goalkeeper] = new Dictionary<Skill, int>();
    this.playerStats[Position.Goalkeeper][Skill.Shooting] = 20;
    this.playerStats[Position.Goalkeeper][Skill.Passing] = 90;
    this.playerStats[Position.Goalkeeper][Skill.BallHandling] = 120;
    this.playerStats[Position.Goalkeeper][Skill.Defence] = 220;
    this.playerStats[Position.Goalkeeper][Skill.Speed] = 90;

    // // Right Fullback
    this.playerStats[Position.RightFullback] = new Dictionary<Skill, int>();
    this.playerStats[Position.RightFullback][Skill.Shooting] = 60;
    this.playerStats[Position.RightFullback][Skill.Passing] = 55;
    this.playerStats[Position.RightFullback][Skill.BallHandling] = 90;
    this.playerStats[Position.RightFullback][Skill.Defence] = 100;
    this.playerStats[Position.RightFullback][Skill.Speed] = 80;

    // Center Back
    this.playerStats[Position.CenterBack] = new Dictionary<Skill, int>();
    this.playerStats[Position.CenterBack][Skill.Shooting] = 60;
    this.playerStats[Position.CenterBack][Skill.Passing] = 55;
    this.playerStats[Position.CenterBack][Skill.BallHandling] = 90;
    this.playerStats[Position.CenterBack][Skill.Defence] = 110;
    this.playerStats[Position.CenterBack][Skill.Speed] = 80;

    // Sweeper
    this.playerStats[Position.Sweeper] = new Dictionary<Skill, int>();
    this.playerStats[Position.Sweeper][Skill.Shooting] = 60;
    this.playerStats[Position.Sweeper][Skill.Passing] = 75;
    this.playerStats[Position.Sweeper][Skill.BallHandling] = 110;
    this.playerStats[Position.Sweeper][Skill.Defence] = 95;
    this.playerStats[Position.Sweeper][Skill.Speed] = 90;
```

```
private void setPlayerBoundaries()
{
    // 4-4-2 Formation

    this.playerBoundaries = new Dictionary<Position, Dictionary<Boundary, int>>();

    // GoalKeeper
    this.playerBoundaries[Position.Goalkeeper] = new Dictionary<Boundary, int>();
    this.playerBoundaries[Position.Goalkeeper][Boundary.minX] = 0;
    this.playerBoundaries[Position.Goalkeeper][Boundary.maxX] = 1;
    this.playerBoundaries[Position.Goalkeeper][Boundary.minY] = 2;
    this.playerBoundaries[Position.Goalkeeper][Boundary.maxY] = 4;

    // Right Fullback
    this.playerBoundaries[Position.RightFullback] = new Dictionary<Boundary, int>();
    this.playerBoundaries[Position.RightFullback][Boundary.minX] = 0;
    this.playerBoundaries[Position.RightFullback][Boundary.maxX] = 2;
    this.playerBoundaries[Position.RightFullback][Boundary.minY] = 0;
    this.playerBoundaries[Position.RightFullback][Boundary.maxY] = 2;

    // Center Back
    this.playerBoundaries[Position.CenterBack] = new Dictionary<Boundary, int>();
    this.playerBoundaries[Position.CenterBack][Boundary.minX] = 0;
    this.playerBoundaries[Position.CenterBack][Boundary.maxX] = 2;
    this.playerBoundaries[Position.CenterBack][Boundary.minY] = 1;
    this.playerBoundaries[Position.CenterBack][Boundary.maxY] = 3;

    // Sweeper
    this.playerBoundaries[Position.Sweeper] = new Dictionary<Boundary, int>();
    this.playerBoundaries[Position.Sweeper][Boundary.minX] = 0;
    this.playerBoundaries[Position.Sweeper][Boundary.maxX] = 2;
    this.playerBoundaries[Position.Sweeper][Boundary.minY] = 3;
    this.playerBoundaries[Position.Sweeper][Boundary.maxY] = 5;

    // Left Fullback
    this.playerBoundaries[Position.LeftFullback] = new Dictionary<Boundary, int>();
    this.playerBoundaries[Position.LeftFullback][Boundary.minX] = 0;
    this.playerBoundaries[Position.LeftFullback][Boundary.maxX] = 2;
    this.playerBoundaries[Position.LeftFullback][Boundary.minY] = 4;
    this.playerBoundaries[Position.LeftFullback][Boundary.maxY] = 6;
```

Team Settings

```
private void setPlayerPassingTargets()
{

    this.playerPassingTargets = new Dictionary<Position, List<Position>>();

    // GoalKeeper
    this.playerPassingTargets[Position.Goalkeeper] = new List<Position>();
    this.playerPassingTargets[Position.Goalkeeper].Add(Position.CenterBack);
    this.playerPassingTargets[Position.Goalkeeper].Add(Position.Sweeper);
    this.playerPassingTargets[Position.Goalkeeper].Add(Position.DefensiveMidfielder);

    // Right Fullback
    this.playerPassingTargets[Position.RightFullback] = new List<Position>();
    this.playerPassingTargets[Position.RightFullback].Add(Position.CenterBack);
    this.playerPassingTargets[Position.RightFullback].Add(Position.RightMidfielder);
    this.playerPassingTargets[Position.RightFullback].Add(Position.DefensiveMidfielder);

    // Center Back
    this.playerPassingTargets[Position.CenterBack] = new List<Position>();
    this.playerPassingTargets[Position.CenterBack].Add(Position.RightFullback);
    this.playerPassingTargets[Position.CenterBack].Add(Position.Sweeper);
    this.playerPassingTargets[Position.CenterBack].Add(Position.DefensiveMidfielder);

    // Sweeper
    this.playerPassingTargets[Position.Sweeper] = new List<Position>();
    this.playerPassingTargets[Position.Sweeper].Add(Position.CenterBack);
    this.playerPassingTargets[Position.Sweeper].Add(Position.LeftFullback);
    this.playerPassingTargets[Position.Sweeper].Add(Position.DefensiveMidfielder);

    // Left Fullback
    this.playerPassingTargets[Position.LeftFullback] = new List<Position>();
    this.playerPassingTargets[Position.LeftFullback].Add(Position.Sweeper);
    this.playerPassingTargets[Position.LeftFullback].Add(Position.LeftMidfielder);
    this.playerPassingTargets[Position.LeftFullback].Add(Position.DefensiveMidfielder);
```

Complexity in handling passing of the ball

```
public int[] getPassingTargets(int playerPositionId) {
    Position playerPosition = (Position)playerPositionId;
    Position T1Position = this.playerPassingTargets[playerPosition][0];
    Position T2Position = this.playerPassingTargets[playerPosition][1];
    Position T3Position = this.playerPassingTargets[playerPosition][2];
    return new int[] { (int)T1Position, (int)T2Position, (int)T3Position };
}

public int[] probPassing(int playerPositionId, int[] teamLocationIds)
{
    Position playerPosition = (Position)playerPositionId;
    int playerLocationId = teamLocationIds[playerPositionId];
    int[] playerCoordinate = this.convertToCoordinate(playerLocationId);

    List<Position> teammates = this.playerPassingTargets[playerPosition];

    double minDist = Double.MaxValue;
    Position selectedTeammate = teammates[2];
    int selectedIndex = 2;

    for (int i = 0; i < 3; i++)
    {
        Position teammate = teammates[i];
        int positionId = (int)teammate;
        int locationId = teamLocationIds[positionId];
        int[] coordinate = this.convertToCoordinate(locationId);
        double distToGoal = this.getDistance(coordinate, this.goalCoordinate);
        double distToPlayer = this.getDistance(coordinate, playerCoordinate);
        double dist = distToPlayer + distToGoal;
        if (dist < minDist)
        {
            minDist = dist;
            selectedTeammate = teammate;
            selectedIndex = i;
        }
    }

    int[] result = {0, 0, 0};
    result[selectedIndex] = 1;
    return result;
}
```

```
public int[] probPassTo(int playerPositionId, int teammatePositionId, int[] teamLocationIds)
{
    int[] result = {0, 0};

    Position playerPosition = (Position)playerPositionId;
    int playerLocationId = teamLocationIds[playerPositionId];
    int[] playerCoordinate = this.convertToCoordinate(playerLocationId);

    Position teammatePosition = (Position)teammatePositionId;
    int teammateLocationId = teamLocationIds[teammatePositionId];
    int[] teammateCoordinate = this.convertToCoordinate(teammateLocationId);

    int passingSkill = this.playerStats[playerPosition][Skill.Passing];
    double distToTeammate = this.getDistance(playerCoordinate, teammateCoordinate);

    if (distToTeammate <= 2)
    {
        result[0] = this.updateWithSkill(9, passingSkill);
        result[1] = 1;
        return result;
    }

    if (distToTeammate <= 3)
    {
        result[0] = this.updateWithSkill(8, passingSkill);
        result[1] = 2;
        return result;
    }

    if (distToTeammate <= 4)
    {
        result[0] = this.updateWithSkill(7, passingSkill);
        result[2] = 3;
        return result;
    }

    else
    {
        result[0] = this.updateWithSkill(5, passingSkill);
        result[1] = 5;
        return result;
    }
}
```

Complexities in handling passing of the ball

```
public int[] getPassingTargets(int playerPositionId) {
    Position playerPosition = (Position)playerPositionId;
    Position T1Position = this.playerPassingTargets[playerPosition][0];
    Position T2Position = this.playerPassingTargets[playerPosition][1];
    Position T3Position = this.playerPassingTargets[playerPosition][2];
    return new int[] { (int)T1Position, (int)T2Position, (int)T3Position };
}

public int[] probPassing(int playerPositionId, int[] teamLocationIds)
{
    Position playerPosition = (Position)playerPositionId;
    int playerLocationId = teamLocationIds[playerPositionId];
    int[] playerCoordinate = this.convertToCoordinate(playerLocationId);

    List<Position> teammates = this.playerPassingTargets[playerPosition];

    double minDist = Double.MaxValue;
    Position selectedTeammate = teammates[2];
    int selectedIndex = 2;

    for (int i = 0; i < 3; i++)
    {
        Position teammate = teammates[i];
        int positionId = (int)teammate;
        int locationId = teamLocationIds[positionId];
        int[] coordinate = this.convertToCoordinate(locationId);
        double distToGoal = this.getDistance(coordinate, this.goalCoordinate);
        double distToPlayer = this.getDistance(coordinate, playerCoordinate);
        double dist = distToPlayer + distToGoal;
        if (dist < minDist)
        {
            minDist = dist;
            selectedTeammate = teammate;
            selectedIndex = i;
        }
    }

    int[] result = {0, 0, 0};
    result[selectedIndex] = 1;
    return result;
}
```

```
public int[] probPassTo(int playerPositionId, int teammatePositionId, int[] teamLocationIds)
{
    int[] result = {0, 0};

    Position playerPosition = (Position)playerPositionId;
    int playerLocationId = teamLocationIds[playerPositionId];
    int[] playerCoordinate = this.convertToCoordinate(playerLocationId);

    Position teammatePosition = (Position)teammatePositionId;
    int teammateLocationId = teamLocationIds[teammatePositionId];
    int[] teammateCoordinate = this.convertToCoordinate(teammateLocationId);

    int passingSkill = this.playerStats[playerPosition][Skill.Passing];
    double distToTeammate = this.getDistance(playerCoordinate, teammateCoordinate);

    if (distToTeammate <= 2)
    {
        result[0] = this.updateWithSkill(9, passingSkill);
        result[1] = 1;
        return result;
    }

    if (distToTeammate <= 3)
    {
        result[0] = this.updateWithSkill(8, passingSkill);
        result[1] = 2;
        return result;
    }

    if (distToTeammate <= 4)
    {
        result[0] = this.updateWithSkill(7, passingSkill);
        result[2] = 3;
        return result;
    }

    else
    {
        result[0] = this.updateWithSkill(5, passingSkill);
        result[1] = 5;
        return result;
    }
}
```

GameManager

- Handles the complex calculation and behaviour that determines the possession of the ball in game loop

- Takes into account a player's ball handling skills in comparison to a defender's defence ability.

```
public int[] updatePossession(int[] possession, int ball, int[] teamA, int[] teamB)
{
    List<Position> playersA = new List<Position>();
    List<Position> playersB = new List<Position>();
    for (int i = 0; i < 11; i++) {
        if (teamA[i] == ball) {
            playersA.Add((Position)i);
        }
        if (teamB[i] == ball) {
            playersB.Add((Position)i);
        }
    }

    int possessionTeam = possession[0];

    if (possessionTeam == -1)
    {
        Random rnd = new Random();
        bool isTeamA = rnd.Next(0, playersA.Count + playersB.Count) < playersA.Count;

        if (isTeamA)
        {
            int r = rnd.Next(playersA.Count);
            Position selected = playersA[r];
            return new int[] { 0, (int)selected };
        }

        else
        {
            int r = rnd.Next(playersB.Count);
            Position selected = playersA[r];
            return new int[] { 0, (int)selected };
        }
    }

    ManagerA managerA = new ManagerA();
    ManagerB managerB = new ManagerB();
    int playerPositionId = possession[1];
    Position playerPosition = (Position)playerPositionId;

    if (playerPosition == Position.Goalkeeper)
    {
        return possession;
    }

    if (possessionTeam == 0)
    {
        int ballHandlingSkill = managerA.getBallHandlingSkill(playerPositionId);
        foreach (Position position in playersB) {
            int defenceSkill = managerB.getDefenceSkill((int)position);
            if (defenceSkill > ballHandlingSkill) {
                return new int[] { 0, (int)position };
            }
```

Attempts at reducing state space

- Reduce FREE BALL states
  - Defenders can “steal” the ball away instead of simply waiting for the offence to lose the ball.

- Reduce the number of players that a player can pass the ball to.
- In some cases, using some calculations, give a deterministic response as to who to pass the ball to instead of a probability.

- Eliminate the losing of ball during dribbling, passing and shooting.

Potential Future Improvements

- Refactoring of C# codebase
  - Currently 1700+ LOCs etc.
  - Possibly add some tests to the codebase.

- Try more soccer strategies
  - Different formation instead of the standard 4-4-2 formation

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

Thank You

Project source code

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