学　　号 _____
密　　级 _____

中 山 大 学 本 科 论 文

2018 世界杯 “预演” 和文字直播

院 (系) 名 称：　数据科学与计算机学院

专 业 名 称：　计算机科学与技术

学 生 姓 名：　李新锐, 朱祎康

指 导 教 师：　乔海燕　教授

2017 年 11 月 29 号

BACHELOR'S DEGREE THESIS
OF Sun Yat-Sun UNIVERSITY

FIFA 2018 Simulation

School (Department): School of Date and Computer Science

Major: Computing Science

Candidate: Li Xinrui,Zhu Yikang

Supervisor: Prof. Qiao Hai Yan

Sun Yat-Sun University

Nov, 2017

郑 重 声 明

本人呈交的作业论文, 是在导师的指导下, 独立进行研究工作所取得的成果, 所有数据、图片资料真实可靠. 尽我所知, 除文中已经注明引用的内容外, 本学位论文的研究成果不包含他人享有著作权的内容. 对本论文所涉及的研究工作做出贡献的其他个人和集体, 均已在文中以明确的方式标明. 本作业论文的知识产权归属于培养单位.

本人签名: _____ 日期: _____

摘　　要

本文主要说明了如何对 2018 年世界杯进行预演和文字直播。主要问题包括：如何获取、加工 2018 年世界杯的各种数据，如何对足球比赛进行数学建模，如何模拟和展示整个世界杯的过程。本文采用了 Python 爬虫爬取数据，关键值排序处理数据，对足球的运动进行数学建模等方法。

关键词: 2018 年世界杯; 爬虫; 数学建模; 关键值排序

ABSTRACT

This pasage illustrates how to simulate and report FIFA 2018. The key problem contains how to get and deal with data about FIFA 2018, how to build a mathematical model of a football match, how to simulate and display the process of the whole FIFA 2018 and so on. We use web crawler to get data, sorting by rank to deal with it and build the mathematical model about movement of the football.

Key words: FIFA 2018; web crawler; mathematical model; sorting by rank

目　　录

1 问题说明

1.1 需要解决的问题

为了完成本次世界杯预演和文字直播的编程任务，有以下问题需要解决：

1. 如何获取并加工 2018 年世界杯的时间地点、出线国家、球员数据等数据
2. 如何良好地设计类和对象
3. 如何依据世界杯比赛规则，模拟和展示整个世界杯比赛的过程
4. 如何对足球比赛进行数学建模，模拟一场比赛的赛况

1.2 解决方法

对应上述每个问题，我们采用了以下方法加以解决：

1. 使用 Python 爬虫将 fifa 官网和 sofifa.com 在线球员数据库上的有关信息爬取到本地，处理缺失值，加以筛选整合，转换为适合程序读取的 json 格式。
2. 主要设计了 Player、NationalTeam、Match、Game 这 4 个类，Player 表示一名球员，NationalTeam 队表示参赛国家队信息，与 Player 类是组合（contains-a）关系。Match 类表示世界杯进程中的一场比赛，与 NationalTeam 类是关联关系（association）。Game 类表示储存整场世界杯比赛的信息，并对外提供执行整场比赛模拟进程的接口，与 Match 和 NationalTeam 类是组合关系。此外采用单例模式设计了 RandLib 随机数发生库，用以产生比赛模拟过程中要用到的各种分布的随机数。
3. 通过按照 Project 说明文档的要求，将世界杯比赛过程分解为分组、小组赛、各轮淘汰赛、决赛等阶段，分而治之，依照各阶段的规则在 Game 类中设计不同的私有方法，最后在公有的模拟整场比赛的函数中顺序调用，从而完成了整个世界杯比赛的模拟和展示过程。
4. 下面的关键模块说明 process 和 oneAction 将详细讨论。

2　设计与实现

2.1　系统框架

在问题说明部分已经简述本次程序中类的设计。如图2.1的 UML 图中直观地展示了这一关系。每个框图中是一个类，框中自上而下三栏分别显示了类的名称，包含的其他类的成员变量，该类的公有接口。

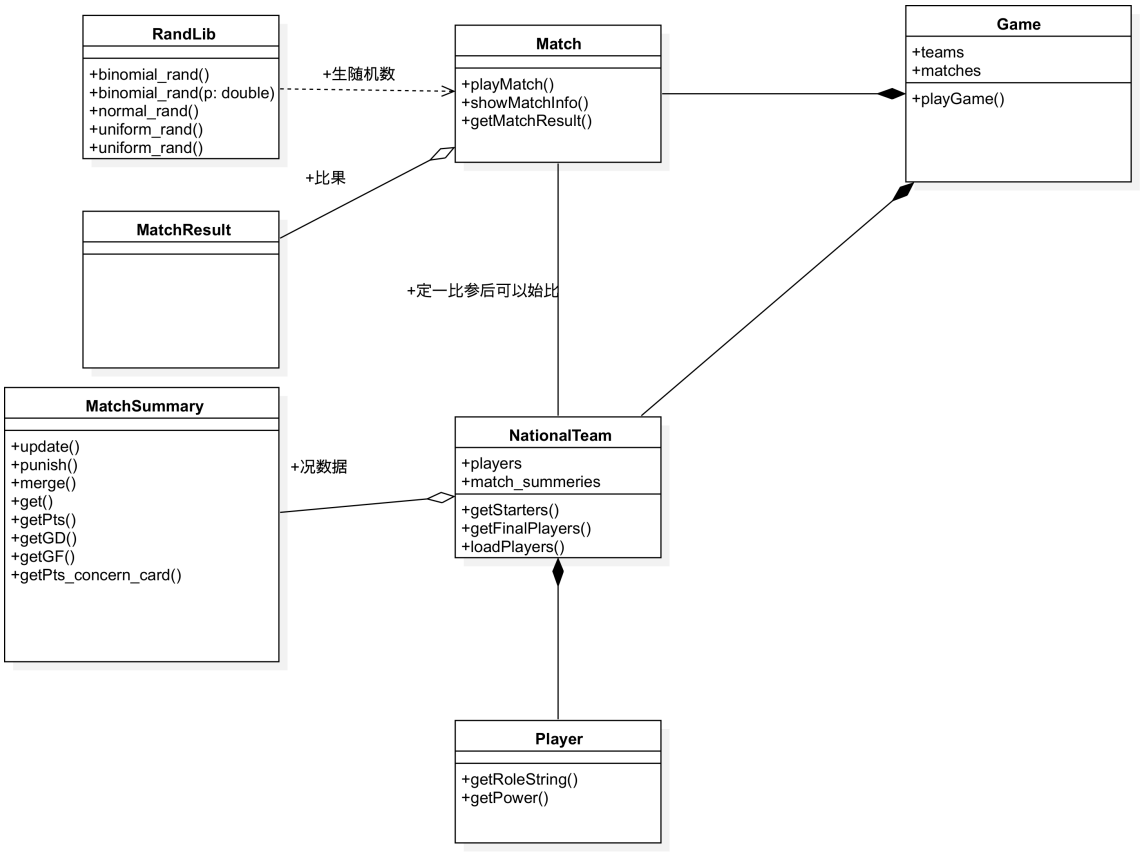

图 2.1　系统框架

2.2 关键模块说明

2.2.1 Game::sortRank(依据各个阶段比赛结果，对参赛队伍进行排名)

本函数接收两个参数：`const std::vector<NationalTeam *> &to_sort` 是待排序的参赛队伍，`stage_t stage` 是比赛阶段。返回一个`std::vector<NationalTeam*>` 表示按照世界杯排名规则排名由高到低排序的队伍数组。

世界杯排名规则如下：

In the league format, the ranking in each group is determined as follows:

a) greatest number of points obtained in all group matches;

b) goal difference in all group matches;

c) greatest number of goals scored in all group matches.

在小组赛中，由于排名规则影响小组赛出线结果，故在使用 a, b, c 规则比较之后仍然存在无法决出排名先后的情况下，将使用以下 d 到 h 规则进行对比

If two or more teams are equal on the basis of the above three criteria, their rankings shall be determined as follows:

d) greatest number of points obtained in the group matches between the teams concerned;

e) goal difference resulting from the group matches between the teams concerned;

f) greater number of goals scored in all group matches between the teams concerned;

g) the goals scored away from home count double between the teams concerned (if the tie is only between two teams).

h) fair play points system in which the number of yellow and red cards in all group matches is considered according to the following deductions:

- first yellow card: minus 1 point
- second yellow card/indirect red card: minus 3 points
- direct red card: minus 4 points
- yellow card and direct red card: minus 5 points;

如果使用 d 到 h 规则对比后仍然存在无法决出排名先后的情况，将采用抽签决定排名

i) drawing of lots by the FIFA Organising Committee.

故在 sortRank 模块的实现中，首先定义了两个排序算子`rule_a_to_c`,`rule_d_to_h`, 这些算子接收两个有序对:`const std::pair<MatchSummary, NationalTeam*>& t1`, `const std::pair<MatchSummary, NationalTeam*>& t2`，每个有序队表示进行排序的队伍和排序参照的胜负、进球和犯规数据，返回一个 bool 值表示 t1 排名是否大于 t2。

它们可以作为参数传入 std::sort 函数，使得 sort 函数依据指定的规则进行排序。

在淘汰赛各论比赛中 sortRank 模块直接依据规则 a 到 c 进行排序。

在小组赛中，该模块执行排序的算法则较为复杂：

同样首先使用规则 a 到 c 进行排名，获得一个可能仍然部分无序的排序结果`to_sort_still_equal`。之后使用规则 g 到 h 再次进行排序。

因为接下来必须知道是哪些队伍排名不分先后。故定义了算子`rule_a_to_c_side_effect`，该算子在排序之外还记录无法决出排名的队伍到集合 equal 中。

由于小组赛共 4 个队伍，可能出现队伍 X1 与 X2 无法决出排名，队伍 X3 和队伍 X4 又无法决出排名，这样存在多组相互之间无序的队伍的情况。故必须记录哪几支队伍之间排名相同，即存在哪几个无序组。

这里使用了并查集解决这一问题，使用`disjoint_union` 函数将排名相同的队伍并入一个无序组。

由于此时排序依据的对战数据不是依据小组赛 4 只队伍之间所有比赛的对战结果，而是依据一个无序组的队伍之间的对战结果，故又定义了 packer 函数将一个无序组中排名相同的各个队伍之间的对战记录息抽取出来，依据这个对战记录，执行规则 d 到 h，获取有序的队伍排名，替换`to_sort_still_equal` 中对应的无序的部分。最终获取一个完全有序的小组赛排名序列。

2.2.2 Match::process(对一场比赛的模拟)

首先要对一场比赛进行数学建模。

为了简化比赛过程，我们假定每名球员的相对位置保持不变，即保持确定的阵型。那么首先要确定每只队伍的阵型选择，实际比赛中每只队伍甚至在每一场

比赛中都会使用不同的阵型，这将难以模拟，所以为了简化处理，我们选择了足球赛场最为常见的攻守兼备的“433”阵型——4 名后卫，3 名中锋，3 名前锋。如图2.2所示：

图 2.2　433 阵型

每名球员的位置固定下来之后，我们只需要模拟球在球员脚下的移动即可（因为争抢球的过程短暂，可以认为任意时刻，球一定在某一人的脚下），那么只需要写一个球的一次移动的函数 oneAction，那么比赛的过程，就是若干个 oneAction。平均分析，大概每分钟传球有 4 次，也就是 90min 有 360 次 oneAction。

如果 90min 出现平局，需要判断是否为“淘汰赛”，如果是，进行 30min 的加时赛（120 次 oneAction）。

如果加时赛也出现平局，进行点球大战。双方各点球 5 个，如果没有分出胜负，双方一个一个的进行点球，直到比分不同，分出胜负。由于点球大战与球员能力的相关性不大，更多的取决于运气、心态，所以这一过程，用纯随机比较合适，取点球进球的概率 $p = 0.8$。

2.2.3 Match::oneAction(足球的一次移动)

oneAction 接受一个参数 pair<int, int> Start，用来表示开始时球在 Start.first 的队伍 Start.second 球员的脚下，oneAction 返回一个 pair<int, int> 表示结束时球的位置信息。如果出现进球，返回值的 first 为 -1.

关键在于如何模拟球的一次移动。以开始时在 6 号球员的脚下为例，我们简化模型，认为此时 6 号球员有 3 种决策：back、forward、still，分别表示向后、向前、向本层传球。我们设置两个参数 p_1, p_2，以 p_1 为概率进行一次 0-1 随机，如果为 1，则向后传球；如果为 0，以 p_2 为概率进行一次 0-1 随机，如果为 1，则向前传球，如果为 0，则向本层传球（向自己传球视为球没有移动，即运球）。

当确定了决策之后，我们就进行传球。传球能否成功，取决于传球的球员的能力以及对位的敌方球员的能力。由于存在临场发挥等不确定因素，球员的当前能力 $power = overall + q \times potential \div 100$，其中 q 为一个随机的 $[0, 1]$ 之间的数字。传球球员的当前能力 $power_1$，对位的敌方球员的当前能力 $power_2$（所有对位敌方球员能力的平均值），根据这两个数字，得到一个传球成功（失败）的概率 p。关于这个函数 $f(power_1, power_2) = p$，其实是可以调的（修改，调参）。进行一次以 p 为概率的 0-1 随机，结果为 1，传球成功（失败），否则，传球失败（成功）。为了简化模型，如果成功，随机一个被传球的对象，如果失败，随机一个抢断球的对象。

这样一来，oneAction 函数就完成了。整个函数是可以进行调参的，其中的 p_1, p_2 还有每一个位置传球成功与否的函数，都是可以调的，这样一来，整个模型就具备了一定的“学习性”。具体来说，每一个参数的值应该取多少，这个我也不知道，我们应该让事实说话，根据事实（往届比赛的数据）来确定每一个参数的值。也就是说，先预先对每个参数设置初值，然后运行程序进行比赛模拟，根据模拟结果与往届比赛的数据，来对每一个参数进行调整。比如，比分总是 $0:0$，说明很难出现进球，那么就将一些参数调的有利于进攻方；如果比分总是出现很大数字，说明进球过于容易，那么就将一些参数调的有利于防守方。经过反复的测试、调参，我们相当于手动完成了“机器学习”的过程，整个程序预测也就趋于合理。

2.3 关键类的说明

Player 该类记录了一名球员的各项数据，提供了 getPower 函数返回综合考虑了球员综合能力和潜力的球员临场能力值。

```
1  struct Player
2  {
3      //! 构造函数，
4      Player() : scoreNumber(0), is_captain(false),
5           is_vice_captain(false),
6           role(PlayerRole::NOT_ASSIGNED) {}
7      QString name;          //! 球员姓名
8      QString nation;        //! 球员国籍
9      int ID;                //! 球员在FIFA数据库中的id
10     int index;             //! 球员编号
11     int overall;           //! 综合能力
12     int potential;         //! 球员潜力
13     int scoreNumber;       //! 生涯进球
14     bool is_captain;       //! 是否队长
15     bool is_vice_captain; //! 是否副队长
16     //! 球员角色，包括GK, DF, MF, FW
17     PlayerRole role;
18     bool can_play[4];      //! 能否胜任上述四个角色
19     //! 返回角色对应的字符串
20     static QString getRoleString(PlayerRole role);
```

```
    int getPower() const;          //! 返回球员临场能力值
};
```

NationalTeam 该类保存了一直国家队相关的各项基本信息，储存了球员指针和与该队有关的赛况数据。提供了 getStarters 和 getFinalPlayers 两个接口获取获取一场比赛的首发阵容和参赛的 23 人名单。

```
class NationalTeam
{
public:
    int world_rank;       //!国家队世界排名
    bool is_host;                 //!是否为东道主
    QString name;                 //!队名
    QString flag;                         //!国旗图片地址
    QString continent;            //!属于哪一足协
    //!获取一场比赛的首发阵容
    std::vector<Player *> getStarters();
    //!获取参赛的23人名单
    std::vector<Player *> getFinalPlayers();
    //!从文件加载队员信息
    void loadPlayers(QString player_file_name);
    //!各比赛阶段赛况汇总
    std::array<MatchSummary, 6> match_summaries;
    //!和各支队伍比赛结果汇总
    std::map<NationalTeam*, MatchSummary>
```

```
19          co_country_summaries;
20  private:
21      //!该国家队所有球员名单
22      std::vector<Player*> all_players;
23      //!参加世界杯的23人名单
24      std::vector<Player*> final_players;
25  };
```

Match 表示一场比赛，储存了从 fifa 官网获取的当前已近确定的比赛的场次、时间地点、参赛队伍名称信息。这里比赛队伍的名称是指它的“别名”，例如 A3（A 组第三只队伍）、Winner Match 61（半决赛 61 冠军）。一个别名唯一确定一支队伍，例如 “A1” 一定对应俄罗斯队，但一支队伍可能有多个别名，例如俄罗斯队 ame 为 Russia 的 NationalTeam 对象可能对应 “Russia”，“A1”，“Winner Match 1” 等多个别名。这些别名对应的具体的球队是在进行抽签或某场比赛前未知的，如果确定了这些别名对应的具体队伍，就可以进行比赛。

因此该类提供的 showMatchInfo（赛前输出本场比赛的相关信息）和 playMatch 公有接口，均接受一个队伍“别名”到具体的队伍的映射关系表。

该类的私有方法 process 和 oneAction 实现了比赛过程的模拟，具体细节请见关键模块说明部分。由于比赛的进球情况影响相关队伍的赛况统计数据和球员的生涯进球数据，故实现了 recordScoreNumber 方法将有关信息写入球队和球员中。

```
1   class Match
2   {
3   public:
4       int id;                 //!比赛场次
5       QString time;           //!比赛时间
```

```
6       QString place;          //!比赛地点
7       QString team1;          //!比赛队伍一名称（别名）
8       QString team2;          //!比赛队伍二名称（别名）
9       stage_t stage;          //!在整场世界杯中所属的阶段
10      //!赛前输出本场比赛的相关信息
11      QString showMatchInfo(
12          std::map<QString, NationalTeam*> name2team);
13      //!返回比赛结果
14      MatchResult getMatchResult() const;
15      //!进行比赛
16      QString playMatch(
17          std::map<QString, NationalTeam*> name2team);
18
19  private:
20      //!存储比赛结果
21      MatchResult matchResult;
22      //!返回年月格式的比赛日期
23      QString getMonthDay(QString time);
24      //!将进球数信息写入相关结构体
25      void recordScoreNumber(int,int);
26      //!显示上场球员信息
27      QString showPaticipantInfo();
28      //!赛后输出本场比赛的结果信息
```

```
29      QString reportMatchResult();
30      //!执行比赛模拟
31      void process();
32      //!模拟比赛中的一次动作
33      pair<int, int> oneAction(pair<int, int> start);
34  };
```

Game 储存整场世界杯的信息，并仅对外提供一个执行整场比赛模拟进程的接口 playGame()。

该类中定义了一系列私有方法用以实现 Project 说明文档中任务一到任务八的各项任务。

定义了私有方法 sortRank 对队伍排名进行排序，具体实现请见关键模块解析部分。

定义了私有方法 dualPrint，辅助用以同时向指定的文件和标准输出中输出指定文本。

定义了一系列私有变量变量如各个阶段的比赛数、各个阶段名称的字符串用以辅助任务的实现。

定义了私有成员变量 name2team 储存了队伍 “别名” 到队伍指针的映射。“别名” 的含义请见 Match 类的说明。

```
1  //分组信息的类型
2  typedef array<std::vector<NationalTeam*>, 8> groups_t;
3  //抽签罐子的类型
4  typedef array<std::vector<NationalTeam*>, 4> pots_t;
5  //球队“别名”到具体球队的映射关系的类型
6  typedef map<QString, NationalTeam *> name2team_map_t;
7  typedef QString QS;
```

```
8  class Game
9  {
10 public:
11     //!构造函数，从默认从CountryData.json中读取参赛国家，
12     // 从Players文件夹下读取各国球员信息，
13     // 从MatchTimePlace.json中读取比赛时间日期
14     Game(QS country_file_name = "./CountryData.json",
15          QS player_floder = "Players",
16          QS timeplace_file_name = "MatchTimePlace.json");
17     void playGame();      //!执行世界杯预演全过程
18 private:
19     //!记录球队“别名”到具体球队的映射关系
20     name2team_map_t name2team;
21     //!储存所有队伍
22     vector<NationalTeam*> teams;
23     //!储存晋级淘汰赛的队伍
24     vector<NationalTeam*> teams16;
25     vector<Match*> matches;//!储存所有比赛
26     groups_t groups;      //!储存小组赛分组情况
27     pots_t prepare_pots();//!准备抽签罐子
28     //!从罐子执行抽签
29     groups_t drawFromPots(name2team_map_t& name2team,
30          const pots_t& pots);
```

```
31      //!检查抽签结果是否合理
32      bool checkDraw(const groups_t& groups);
33      //!同时向文件和标准输出进行输出
34      void dualPrint(const QString& text,
35           const QString& filename,
36           bool append = false);
37
38      //!实现任务一
39      void t1_showQuaified();
40      //!实现任务二
41      void t2_divideGroups(name2team_map_t& name2team);
42      //!实现任务三的日程安排部分
43      void t3_1_showGroupMatches();
44      //!实现任务三的模拟比赛过程
45      void t3_2_playGroupMatches();
46      //!实现任务三的统计结果部分
47      void t3_3_summaryGroupMatches();
48      //!实现任务三的显示晋级部分
49      void t3_4_showNextTeams();
50      //!调用以上四个函数实现任务三
51      void t3_groupStageMatches();
52      //!实现任务四到七的日程安排部分
53      void t4_t7_showKnockoutMatches(stage_t stage);
```

```
54      //!实现任务四到七的模拟比赛过程
55      void t4_t7_playKnockoutMatches(stage_t stage);
56      //!实现任务四到七的统计结果部分
57      void t4_t7_summaryKnockoutMatches(stage_t stage);
58      //!实现任务三的显示晋级部分
59      void t4_t7_showNextTeams(stage_t stage);
60      //!调用以上四个函数实现任务四到七
61      void t4_t7_knockoutStageMatches(stage_t stage);
62      //!实现任务八
63      void t8_showWholeGameStats();
64
65      //!依据各个阶段比赛结果，对参赛队伍进行排名
66      vector<NationalTeam*> sortRank(
67          const std::vector<NationalTeam*> &to_sort,
68          stage_t stage);
69      //!各个阶段名称的首字母大写字符串
70      vector<QString> stage_name_uppercase =
71          {"Group", "Round of 16", "Quarter-finals",
72          "Semi-finals", "Finals"};
73      //!各个阶段名称的首字母小写字符串
74      vector<QString> stage_name_lowercase =
75          {"group", "round of 16", "quarter-finals",
76          "semi-finals", "finals"};
```

```
77      //!各个阶段第一次比赛的编号
78      vector<int> stage_start_match =
79          {0, 48, 56, 60, 62};
80      //!各个阶段的比赛数
81      vector<int> stage_match_num =
82          {48, 8, 4, 2, 2};
83      //!各个阶段剩余的队伍数目
84      vector<int> stage_team_num =
85          {32, 16, 8, 4, 4};
86      //!各个阶段相关的要写入的文件名中的数字
87      vector<int> stage_filename_idx =
88          {16, 8, 4, 2, 1};
89  };
```

3　程序使用和测试说明

3.1　使用说明

直接运行 FIFA.exe 即可。

3.2　测试说明

程序经过两人的多次测试，输出结果完全符合世界杯比赛规则，不存在违背常理的情况。输出格式完全符合 Project 说明文档的要求。

3.3　Bug 说明

在程序测试阶段，我们在 main 函数中使用了一个死循环反复执行整个程序。发现在一定次数的运行后，偶尔会出现读取到的球员信息无效，导致程序异常终止的情况。经检查记录该球员信息的文件没有错误，同时多数情况下该球员读取到的信息是有效的，猜测是在重复快速读取文件的过程中操作系统的文件操作出现了错误。

4 总结和讨论

4.1 完成过程中的困难

数据整合 在爬取 fifa 官网获取参赛球员姓名后，要在从 sofifa[1]下载的球员数据库中匹配到对应的球员。然而出现了大量无法匹配的问题。原因是虽然 fifa 官网和 sofifa 球员数据库均使用西欧字母表示球员姓名，然而部分国家球员姓名的对应的西欧字母转写规则不唯一；或是拼写很长，fifa 官网和 Sofifa 使用了不同的缩写（阿拉伯国家姓名在这方面的问题较为严重）。因此我试图爬取了 sofifa 的数据库，从球员详情页面获取了球员的姓名全称。然而由于球员数目过多（一万七千多人），单线程爬取速度无法接受。而使用多线程爬取则在爬取过程中出现了耗尽计算机内存的情况。经过使用 128 个爬虫线程，多次分段爬取，才最终获取了全部球员的全称，并改进姓名匹配算法后，使得最后的匹配率达到 75% 以上。

建模过程 建模时，什么是应该保留的，什么是应该忽略的，什么是主要的，什么是次要的，都十分让人头疼。模型如果过于简化，将十分脱离实际，没有什么意义；模型如果简化不当，十分复杂，就难以建模，难以模拟。举个例子：一次传球的成功受到太多太多因素的影响，哪些因素是主要的——球员能力，哪些因素是次要的——天气、心情、状态等等等等，所以根据球员能力来模拟，比较合理。当建模完成后，需要反复进行测试、调参，这个过程十分繁琐，工作量很大。

[1] sofifa.com

4.2 改进空间

数据处理 对于少部分未在球员数据库中匹配到的球员，我采取的补全缺失数据的策略是使用全队各项属性均值来补全这些缺失值，使得这些数据缺失的存在不影响根据有数据可查的队员的水平决定的全队平均水平。然而在时间允许的情况下，更好的办法是为更多的球员数据库编写爬虫，例如 Transfermarkt[2]，真正补全这些缺失值。

数学建模 包括以下三个方面：

改进现有模型 在传球过程中，可以增加一个“犯规”的情况，设定 2 个概率，分别表示两个人争球过程中的犯规概率。还可以增加一个出界的情况，做法与犯规类似。

机器学习，程序自动测试、调参 目前采取的是人力测试、调参，但这个过程完全可以编程解决。我们设定一个 check 函数，统计模拟结果的总进球数、平均每场比赛进球数、进球数的方差等数据，然后与往届世界杯的数据进行对比，如果相差在一定范围内，返回 0，如果大，返回 1，如果小，返回 -1。可以证明这个 check 函数关于每一个参数都是凸函数（check 实际上是这些参数的函数），那么我们可以直接进行二分，找到 check 函数的零点，也就找到了一组合适的参数。

使用更好的数学建模 现有的建模，固定了每名球员的相对位置，还是不够贴近实际，模型还是太简单。我们可以将整个赛场划分成若干个点，每平方米一个点，假定任意时刻，所有球员和球都在某一个点。这样一来，关键就在于下一个时刻，所有球员和球的位置是什么。我们将此时所有球员和球的位置放在一个向量 $\vec{x}$ 里，将下一时刻所有球员和球的位置放在一个向量 $\vec{y}$ 里，我们要找到一个矩阵 A，使得 $A\vec{x}=\vec{y}$。显然这个过程可以使用卷积神经网络来做。我们将时间按照 1s 进行分划，搜集一些过去的比赛录像，对每一秒人和球的位置进行统计，得到许多 $(\vec{x_i},\vec{y_i})$，然后用这些大数据“喂养”这个卷积神经网络，就可以得到合适的 A 了。

[2] www.transfermarkt.de

4.3 教训与收获

1. 多人合作的项目，一定要在必要的地方加上注释，解释清楚每一个函数功能、参数意义，每一个类的意义、成员的意义，以及一些语句块的作用。尤其是双方工作的接壤处，一定要说清楚需要传递的参数。
2. 要养成减少“副作用”编程的习惯，不要过于依赖全局变量，函数与函数之间不要有影响。否则，让人阅读起来非常吃力，尤其是在修改程序的时候，藕断丝连的“副作用”会使人痛不欲生。
3. 尽量使用自顶向下的程序设计思想，这样思路清晰不易混乱。使用自底向上的话，很容易写一些重复、没用的东西，而且思路混乱。
4. 锻炼了双人合作的工作模式，了解到了合作的重要性以及合作需要注意的许多事项，经历了单人工作模式所没有的诸多挑战，例如：代码沟通，任务分工，合并代码，共同编辑文稿等。
5. 锻炼了对实际问题的数学建模和分析能力，学会了如何简化问题，提炼核心要素，并用程序解决。

参考文献

[1] https://en.wikipedia.org/wiki/2018_FIFA_World_Cup

[2] http://www.fifa.com/

[3] 2018 FIFA World Cup regulations https://www.uefa.com/MultimediaFiles/Download/Regulations/uefaorg/Regulations/01/87/54/21/1875421_DOWNLOAD.pdf

[4] FIFA 18 Complete Player Dataset https://www.kaggle.com/thec03u5/fifa-18-demo-player-dataset

[5] 球员信息数据库 sofifa.com

致　　谢

感谢你，感谢我，感谢他。