

# 程序设计基础大作业-飞行棋 实验报告

221220028 金文泽

## 概述



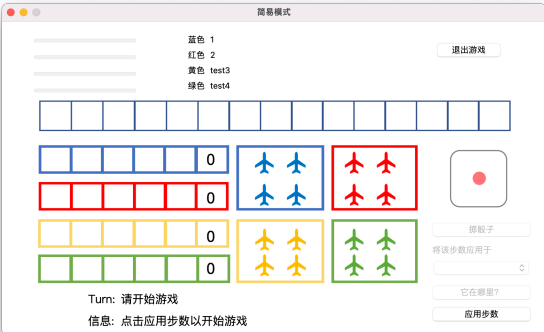
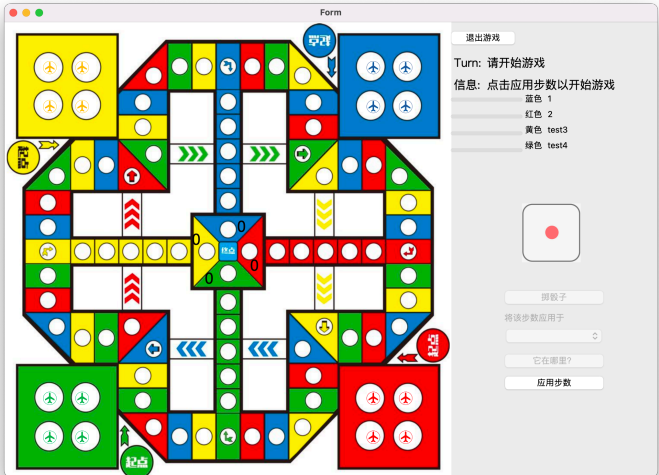
本次作业是用C/C++开发一个飞行棋小游戏。本作品完成了所有基本要求，并采用面向对象编程范式，基于Qt的GUI库，完整的实现了一个带界面的标准棋盘、规则的飞行棋。下面将从几个方面介绍本作品

**编译执行方式：**工程在arm64, macOS Ventura, Qt 6.4.0, clang 环境下开发。不保证在windows x86/x86\_64下编译运行不出现bug，没有提供在x86 windows下编译生成的exe文件。具体编译执行将由我在展示时完成。但随文件附带有在 飞行棋.zip 中的app文件，可在上述 arm64 macOS Ventura （浅色模式）下正常运行。代码源文件为 new.zip

## 1. 文件结构

```
1  - new 项目文件夹
2  |- new.pro qmake makefile文件
3  |
4  |- 头文件
5  ||- gamefinish.h 结算界面QWidget窗口类的头文件
6  ||- place.h Place类的属性声明
7  ||- plane.h Plane类的属性声明
8  ||- players.h Player类的属性声明
9  ||- settings.h 玩家设置界面QWidget类的头文件
10 ||- standard.h 标准模式QWidget类的头文件
11 ||- widget.h 主界面的QWidget类的头文件
12 |
13 |- 源文件
14 ||- gamefinish.cpp 结算界面的实现
15 ||- main.cpp 主函数
16 ||- place.cpp Place类的实现
17 ||- plane.cpp Plane类的实现
18 ||- players.cpp Player类的实现
19 ||- playersvariables.cpp 有关游戏玩家的一些全局变量
20 ||- settings.cpp 设置界面的实现
21 ||- standard.cpp 标准模式的实现
22 ||- widget.cpp 主界面的实现
23 |
24 |- 界面文件(略)
25 |
26 |- 资源(略)
```

## 2. 实现效果UI展示

主界面	玩家设置界面
	
简易模式界面（已移除）	标准模式界面
	

### 3. 基本思路与数据抽象

本部分主要说明该作品开发的基本思路，由于Qt GUI库不是本课程的关注内容故略去相关内容。

#### 游戏前玩家设置

在实现游戏前设置玩家数量、玩家信息、电脑玩家等功能时，我采用的是全局变量的形式，并放在一个独立的 `playersvariables.cpp` 文件中。同时，通过设置界面，可以使用下拉复选框选择电脑玩家，或者是自由地输入玩家昵称；可以使用数字输入栏（输入范围2~4），动态地调整玩家数量，多余玩家数量的复选框将不可用。

```
1 //playersvariables.cpp
2 #include <QString>
3 #include <QQueue>
4 bool hasset = false; //该变量存储玩家信息有没有被正常设置完成
5 QString Players[4] = {}; //QString类的数组用于储存玩家昵称
```

```

6  int playerNumber = 4; //用于储存玩家数量
7  QQueue<QString> * finish_queue = new QQueue<QString>; //利用Qt内部实现的队列数据结构，实现到达
   终点玩家的FIFO，以呈现最终的排名
8  int sizes = 41; //用于控制棋子的大小：标准模式和简易模式中棋子的大小是不一样的
9
10 //settings.cpp
11 .....
12 void Settings::on_applyButton_clicked()
13 {
14     .....
15     hasset = true;
16     for(int i = 0; i < playerNumber; i++){
17         if(Players[i] == "--请输入玩家昵称或者选择电脑玩家--") hasset = false;
18     }
19     .....
20 }
21 .....

```

如果在开始游戏之前没有完成玩家设置，那么将会提示且无法进入游戏。

## 数据抽象

游戏的基本结构被抽象成了以下几部分

1. 棋盘格 `Place` 类：

```

1  class Place
2  {
3  public:
4      bool is_interaction = false; //该格是否是一个分流点
5      QString planeColor = "None"; //该格最新的飞机颜色
6      int position[2] = {}; //该格在UI上的坐标
7      int plane_nums = 0; //该格当前的飞机数
8      Place * exits[5] = {NULL, NULL, NULL, NULL, NULL}; //4为默认出口，当该格为分流点时，用0、
   1、2、3分别控制四个颜色的出口，若该颜色出口为NULL，则使用默认出口
9      .....
10     Place * entrance = NULL; //该格的入口
11     bool finishline = false; //该格是否是终点
12     .....
13     Place(const int pos[]); //构造方法
14     Place * nextPlace(const QString &color); //根据颜色锁定下一格的位置
15 };

```

2. 飞机棋子 `Plane` 类：

```

1  class Plane
2  {
3  public:
4      QString m_color;      //飞机的颜色
5      QString m_name;      //飞机的名称（用于显示）
6      bool is_finished = false;    //这个飞机是否已经到达终点
7      void move(bool forward);    //用于移动飞机的方法，forward代表方向，1向前
8      void jump();    //实现“跳”规则的方法
9      void fly();    //实现“飞”规则的方法
10     QLabel * m_label = NULL;    //绑定当前飞机在UI中的QLabel
11     Place * m_place = NULL;    //飞机当前坐在的Place
12     Plane(const QString &name, const QString &color, Place * place, QLabel * label);
13 };

```

3. 玩家 `Player` 类：

```

1  class Player{
2  public:
3      bool is_AI = 0;      //这名玩家是否是电脑
4      QString m_color = "";    //这名玩家的颜色
5      QString m_name = "";    //这名玩家的名称
6      Plane * m_planes[4] = {NULL, NULL, NULL, NULL}; //这名玩家的四个棋子
7      int finish_planes = 0;    //这名玩家已经到达终点的飞机数量
8      bool finished = false;    //这名玩家是否已经游戏结束
9
10     Player(const QString &name, const QString &color);
11     void finishPlane();    //用于正确操作finish_plane变量
12 };

```

## 游戏逻辑

开始游戏后，会执行 `Initialize()` 方法进行初始化，设置好棋盘棋子玩家等。

1. 点击“应用步数”按钮后，会进行回合的轮换，根据当前玩家是否是电脑执行不一样的操作。若当前玩家不是电脑：
2. 需要先掷骰子，随机生成一个1-6的步数，如果不掷骰子就点击应用步数会提示“请先掷骰子”。
3. 接着，根据本回合是否是一个起飞的后续回合，把合适的棋子加入下拉复选框中，供玩家选择棋子。如果没有可用的棋子，会在消息栏中提醒玩家。
4. 关于叠子的问题，由于我没有想到在Qt中简洁的处理方式，所以使用了一种替代的办法。如果存在棋子叠放而无法辨认的情况，或者想找到某颗棋子，都可以使用“它在哪里”按钮，找到这个棋子（棋子会放大几秒）
5. 最终，按下应用步数，棋子移动，并伴有动画。如果选择起飞一个棋子，则会提示用户下一个回合是起飞后续回合，执行相关的逻辑，并不轮换用户。

接下来，将展示棋子移动的实现。

```

1  //Standard.cpp
2  //这是普通玩家按下应用步数按钮后应该执行的方法
3  void Standard::SB_taketurn(int index, bool AI){
4      //根据复选框明确要移动的棋子
5      Plane * target = NULL;
6      for(int i = 0; i < 4; i++){
7          if(gamePlayer[index]->m_planes[i]->m_name == ui->comboBox->currentText()){

```

```

8         target = gamePlayer[index]->m_planes[i];
9     }
10 }
11
12 if(target != NULL){
13     if(takeoffTurn){
14         takeoffTurn = false;
15     }
16     else{
17         //判断下个回合是否为起飞回合
18         for(int i = 0; i < 4; i++){
19             if(target->m_place == BLport[i]) takeoffTurn = true;
20             if(target->m_place == RDport[i]) takeoffTurn = true;
21             if(target->m_place == YLport[i]) takeoffTurn = true;
22             if(target->m_place == GRport[i]) takeoffTurn = true;
23         }
24         if(takeoffTurn == true){
25             if(!AI) QMessageBox::warning(this, "提示", "你将使一个飞机起飞, 下一回合仍然是
你的回合");
26             if(!AI) ui->MessageBox->setText("请再掷一次骰子使其起飞");
27             ui->comboBox->blockSignals(true);
28             ui->comboBox->clear();
29             ui->comboBox->blockSignals(false);
30             ui->comboBox->addItem(target->m_name);
31             diceused = false;
32             return;
33         }
34     }
35     int back = 0;
36     for(int i = 0; i < dicenum; i++){
37         target->move(1); //调用飞机的move方法
38         m_sleep(200);
39         if(target->m_place->finishline == true){
40             //判断要不要返回, 返回几格
41             back = dicenum - i - 1;
42             if(back == 0){
43                 //满足以上条件时 该棋子到达终点 执行与终点相关的处理命令
44                 target->is_finished = true;
45                 gamePlayer[index]->finishPlane();
46                 if(gamePlayer[index]->finished == true){
47                     finish_players++;
48                     finish_queue->enqueue(gamePlayer[index]->m_name);
49                 };
50                 finish_labels[index]-
>setText(QString::asprintf("%d", gamePlayer[index]->finish_planes));
51                 progress[index]->setValue(gamePlayer[index]->finish_planes);
52                 target->m_label->setEnabled(false);
53             }
54             break;
55         }
56     }
57 }

```

```

58         //最后, 判断要不要跳格
59         if(target->m_color == target->m_place->placeColor && target->m_place->is_fly
== false){
60             target->jump();
61         }
62         else{
63             //要不要飞行
64             target->fly();
65         }
66         //如果不需要回程且没有到达终点, 则踢出当前区域其他色棋子
67         if(back == 0 && target->is_finished == false){
68             kickout(target->m_place);
69         }
70         //在本模式地图中, 需要回程的情况下不可能挤兑其他棋子, 故不考虑该情况
71         for(int i = 0; i < back; i++){
72             target->move(0);
73             m_sleep(200);
74         }
75     }
76 }
77
78 //Plane.cpp
79 void Plane::move(bool forward){
80     if(forward){
81         if(m_place->nextPlace(m_color) == m_place) return;
82         m_place->plane_nums--;
83         if(m_place->plane_nums <= 0){
84             m_place->plane_nums = 0;
85             m_place->planeColor = "";
86         }
87         m_place = m_place->nextPlace(m_color); //获取下一格的place
88         m_place->planeColor = m_color;
89         m_place->plane_nums++;
90         m_label->setGeometry(m_place->position[0], m_place->position[1], sizes,
sizes);
91     }
92     else{
93         if(m_place->entrance == NULL) return;
94         m_place->plane_nums--;
95         if(m_place->plane_nums <= 0){
96             m_place->plane_nums = 0;
97             m_place->planeColor = "";
98         }
99         m_place = m_place->entrance; //退格
100         m_place->planeColor = m_color;
101         m_place->plane_nums++;
102         m_label->setGeometry(m_place->position[0], m_place->position[1], sizes,
sizes);
103     }
104 }
105
106 //跳格就是向前移动四次

```

```

107 void Plane::jump(){
108     if(m_place->is_interaction){
109         return;
110     }
111     move(1);
112     move(1);
113     move(1);
114     move(1);
115 }
116
117 //已经预先定义了格子的is_fly属性，只要在条件符合时获取各自的飞行出口即可
118 void Plane::fly(){
119     if(m_place->is_fly && m_color == m_place->placeColor){
120         m_place->plane_nums --;
121         if(m_place->plane_nums <= 0){
122             m_place->plane_nums = 0;
123             m_place->planeColor = "";
124             m_place = m_place->fly;
125             m_place->planeColor = m_color;
126             m_place->plane_nums ++;
127             m_label->setGeometry(m_place->position[0], m_place->position[1], sizes,
sizes);
128         }
129     }
130 }

```

## 最后

通过本次大作业的制作，我学习了很多新知识。我自学了C++中的面向对象编程和Qt GUI库。由于在面向对象编程的设计模式方面还不够熟悉，所以可能在数据抽象和代码风格上仍有欠缺，还请各位老师、助教们批评指正。