[UNO 纸牌游戏] 设计文档

[redacted]

2024 年 6 月 7 日

1 简介

小组名称

乌诺不是优诺

小组成员

[redacted]

项目模块简介与分工

项目共两个版本, 分别是基于命令行版本的 UNO 纸牌游戏和基于图形化交互版本的 UNO 纸牌游戏.UNO 纸牌游戏是经典的纸牌游戏, 游戏的主要目标是尽快打出手中的牌. 每位玩家在自己的回合中, 可以出与桌面上的牌颜色或数字相同的牌, 或者出特殊功能的牌. 特殊功能的牌包括” 跳过”,” 反转方向”,”+2” 和”+4”. 跳过牌让下一个玩家失去回合, 反转方向改变出牌顺序,+2 让下一个玩家摸两张牌, 而 +4 则让下一个玩家摸四张牌并且可以选择改变当前牌的颜色. 若玩家没有合适的牌可出, 则需要从牌堆中摸一张牌, 若这张牌可以出, 玩家可以决定是否出这张牌并结束自己的回合. 本游戏就实现了完成 UNO 纸牌游戏需要的所有基本环节, 通过 `Application.gameloop` 实现一局多人制的 UNO 纸牌游戏

命令行版本

命令行版本的 UNO 纸牌游戏主要分为逻辑实现模块 `uno_logic.cpp` 文件和 AI 玩家智能出牌模块 `aiplayer.cpp` 文件 2 部分

图形化交互版本

图形化交互版本的 UNO 纸牌游戏主要分为逻辑实现模块 `uno_logic.cpp` 文件和图形化交互模块 `visualize.cpp` 文件 2 部分

分工

王子轩:
1. 游戏框架搭建和开发方案设计

2. 纸牌核心逻辑代码实现和图形化交互模块实现, 编写 `main.cpp` 文件,`uno_logic.cpp` 和 `uno_logic.h` 文件,`visualize.cpp` 和 `visualize.h` 文件
3. 资源文件整理和配置
4. 无 AI 玩家智能出牌功能的完整命令行版本开发调试
5. 图形化交互版本 UNO 游戏的完整开发调试和测试
6. 设计文档撰写
7. 展示 PPT 制作

赵启元:
1. 提供面向对象的参考思路
2. 负责完成 AI 玩家智能出牌模块, 编写 `aiplayer.cpp`
3. 有 AI 玩家智能出牌功能的命令行版本的调试和测试工作
4. 进行 AI 部分设计文档撰写

刘禹初:
未参与

2 设计思想与系统框架

说在前面的一些话

对于 UNO 的游戏核心逻辑, 我最初想到的是用面向过程来实现, 并且没有考虑将 UI 设计和核心逻辑完全分开来写, 尝试写了几天之后, 发现代码量非常巨大, 而且函数之间的互相调用导致之间的耦合性非常强以至于后来根本无法推进; 在和赵启元组员的讨论之后, 我们认为用面向对象的思想来写会更加简洁; 同时我决定先做一个完全基于命令行的版本, 然后再尝试基于图形化交互的版本.

核心逻辑实现花了我比较多的时间, 在完成一个完整的命令行游戏后, 我利用所有的命令行输入输出的返回值作为 UI 的接口, 实现图形化交互, 这个过程完成比较快也比较顺利

问题分析

游戏其实主要由卡牌, 抽象玩家和规则三个部分组成. 我们设计的游戏中有多个 AI 玩家, 只有一位人类玩家; 每一轮中, 玩家从抓牌, 出牌, 过牌三个动作中选择一个来做, 能出的牌根据游戏的规则取决于上一张牌的颜色, 数字和特殊功能; 游戏的信息需要显示, 包括当前玩家, 剩余牌数, 上一手出的牌等.

设计方案

核心逻辑设计框架如下:

数据结构定义

在 `uno_logic.h` 文件中定义卡牌类型 (`CardType`), 卡牌颜色 (`CardColor`), 卡牌 (`Card`), 卡牌队列 (`CardQueue`), 抽象玩家 (`AbstractPlayer`),AI 玩家 (`AiPlayer`), 人类玩家 (`Player`) 和游戏应用 (`Application`)

游戏初始化

`Application` 类的构造函数中初始化玩家数量, 创建人类玩家和 AI 玩家, 生成一副标准的 UNO 卡牌并进行洗牌.

游戏循环

游戏循环持续进行, 直到游戏结束条件达成. 每个玩家轮流进行操作, 包括选择出牌或拿牌.

出牌逻辑

玩家根据当前出的牌 (`lastCard`) 和自己的手牌选择一张合法的牌出牌.

AI 玩家逻辑

AI 玩家在选择出牌时, 会检查自己的手牌, 并选择一张可以出的牌. 如果没有可出的牌, 则抓牌.

人类玩家交互

人类玩家通过命令行提示输入选择要出的牌或选择颜色.

游戏结束条件

当所有 AI 玩家都出完牌, 或者人类玩家出完牌时, 游戏结束.

洗牌和发牌

使用 `shuffle` 函数对牌进行随机洗牌, 使用 `defaultCards` 函数生成一副标准的 UNO 卡牌.

游戏信息显示

游戏过程中, 会显示当前玩家信息, 剩余牌数, 上一手出的牌等.
基于 EasyX 的 UI 设计实现如下

数据结构定义

利用结构体数组全局变量来存储用到的图像数据, 并进行编号, 实现从命令行输出值到 UI 显示的映射关系.

图片加载和初始化

将玩家形象, 卡牌, 背景等素材加载到内存中.

游戏窗口初始化

创建游戏窗口, 设置窗口大小, 标题等, 返回玩家加入几人局的选项.

游戏窗口事件处理

游戏窗口的事件处理, 通过鼠标点击, 返回人类玩家的选项, 实现从 UI 到命令行输入的映射

3 游戏流程与功能展示

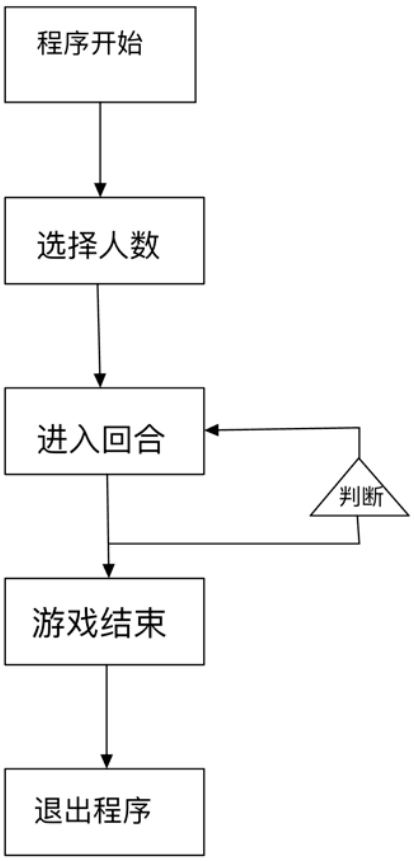

图 1: 程序流程图

图 2: 游戏开始界面

图 3: 选择人数界面

图 4: 游戏画面

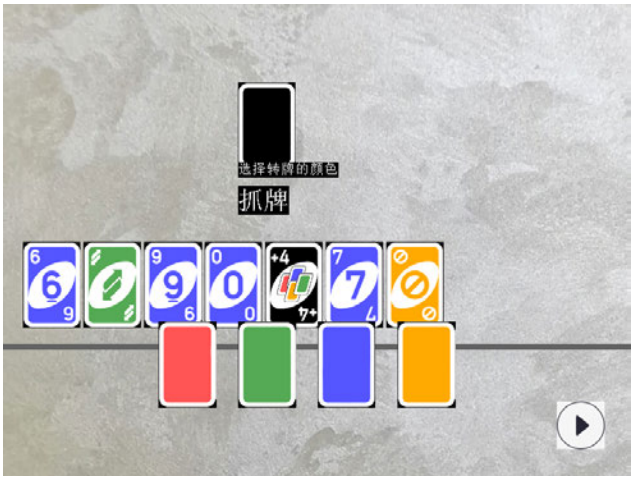

图 5: 选择加 4 的颜色牌

```
Welcome to UNO!------------------------------------------------------------
Player 1 remaining card count: 7
Player 2 remaining card count: 7
Player 3 remaining card count: 7
Player 4 remaining card count: 7

Your remaining card count: 7
Your cards:
      0 - | NUMBER      | yellow | 4 |   valid
      1 - | NUMBER      | green  | 6 |   valid
      2 - | NUMBER      | red    | 9 |   valid
      3 - | NUMBER      | yellow | 6 |   valid
      4 - | NUMBER      | blue   | 8 |   valid
      5 - | NUMBER      | red    | 1 |   valid
      6 - | NUMBER      | red    | 6 |   valid

Please select the card you want to play.
entered: 2
------------------------------------------------------------
Player 1 remaining card count: 5
Player 2 remaining card count: 7
Player 3 remaining card count: 6
Player 4 remaining card count: 7

Your remaining card count: 6
Your cards:
      0 - | NUMBER      | yellow | 4 | invalid
      1 - | NUMBER      | green  | 6 |   valid
      2 - | NUMBER      | yellow | 6 |   valid
      3 - | NUMBER      | blue   | 8 | invalid
```

图 6: 命令行同步显示该局信息

图 7: 玩家出牌

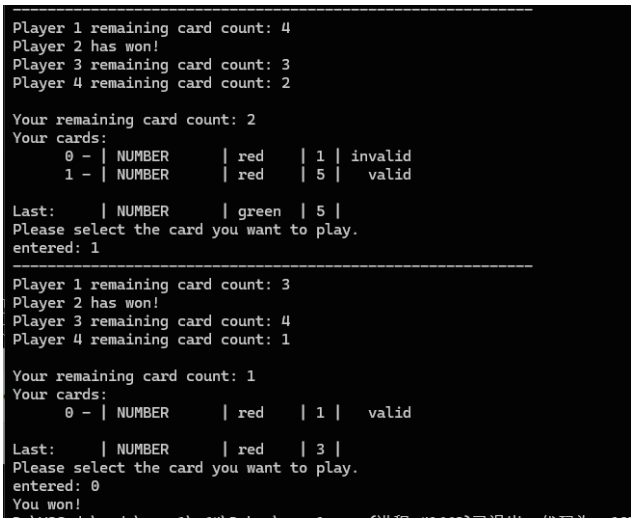

图 8: 玩家胜利

游戏的主要功能

核心逻辑

- 生成一副标准的 UNO 卡牌
- 洗牌
- 出牌逻辑
- AI 玩家逻辑
- 游戏结束条件
- 信息显示

人类玩家交互

- 选择人数
- 可视化地显示牌面
- 通过卡牌浮动提示可以出的牌
- 出牌
- 摸牌
- 打出牌
- 选择加 4 的颜色
- 通过命令行同步进行该回合游戏信息显示

AI 玩家逻辑

- 记忆上一手牌
- 从命令行获得该局游戏信息
- 选择抓牌或出牌或过牌

4 源代码

main.cpp

```cpp
# include <iostream>
# include "uno_logic.h"
# include "visualize.h"
#include <windows.h>
#pragma comment(lib,"Winmm.lib")
using namespace std;

int main() {
    printf("Welcome to UNO!");
    mciSendString(_T("open music.mp3 alias bkmusic"), NULL, 0, NULL);//播放音乐
    mciSendString(_T("play bkmusic repeat"), NULL, 0, NULL);//循环播放
    initialize_card_image();
    initialize_color_card_image();
    initialize_player_image();
    Application application = Application(initialize_startgame());
    initialize_color_card_image();
    application.gameLoop();
    system("pause");
    return 0;
}
```

visualize.h

```cpp
# include <iostream>
#include <vector>
#include <string>
#include <cmath>
#include <algorithm>
# include <graphics.h>
```

```
7  # include <time.h>
8  # include <conio.h>
9  # include <windows.h>
10 # include "uno_logic.h"
11
12 #ifndef MAX_PLAYERS
13 #define MAX_PLAYERS 7
14 #endif
15
16 #ifndef MAX_CARDS
17 #define MAX_CARDS 108
18 #endif
19
20 #ifndef _VISUALIZE_H_
21 #define _VISUALIZE_H_
22
23 typedef struct {
24         IMAGE image_card;
25         int card_id;
26 } Card_image;
27
28 extern Card_image
29 card_images[MAX_CARDS];
30
31 typedef struct {
32         IMAGE image_player;
33         int player_id;
34 } Player_image;
35
36 extern Player_image player_images[MAX_PLAYERS];
37
38 typedef struct {
39         int color_card_id;
40         IMAGE image_color;
41 } Color_card_image;
42
43 extern Color_card_image color_card_images[5];
44 int initialize_startgame();
45
```

```
46 void initialize_card_image();
47
48 void initialize_player_image();
49
50 void initialize_color_card_image();
51
52 int select_color_visualize();
53
54 void put_card_screen(Card* card, bool result, int x, int y);
55
56 void show_hand_cards(Card** temp, bool* result, int count);
57
58 int select_card_screen();
59
60 void show_last_card(Card* lastcard);
61
62 void show_grab_card();
63
64 void attention_grab_card();
65
66 void draw_line_to_hold_thecards();
67
68 void show_player(int player_id, int x, int y);
69
70 void show_players(int aiplayercount);
71 #endif
```

uno_logic.h

```
1     #ifndef UNO_LOGIC_H
2 #define UNO_LOGIC_H
3
4 enum CardType {
5     NUMBER = 0,
6     SKIP = 1,
7     REVERSE = 2,
8     DRAW_2 = 3,
9     WILD = 4,
10    WILD_DRAW_4 = 5
11 };
```

```
12
13 enum CardColor {
14     red = 0,
15     green = 1,
16     blue = 2,
17     yellow  = 3
18 };
19
20 //卡牌对象
21 struct Card {
22     //卡牌的牌面信息
23     CardType cardType = NUMBER;
24     CardColor cardColor = blue ;
25     int cardNumber = 0;
26
27     //卡牌的编号,用于到UI卡牌图像的显示映射关系
28     int card_id = 0;
29
30     //自引,指向下一张卡牌
31     Card* nextCard = nullptr;
32
33     explicit Card(CardColor cardColor, int cardNumber, int card_id);
34
35     explicit Card(CardType cardType, CardColor cardColor, int card_id);
36
37     explicit Card(CardType cardType, int card_id);
38
39     //是否需要另外的颜色选择
40     const char* info(bool enableExtraColor) const;
41 };
42
43 struct CardQueue {
44     int cardCount = 0;
45     Card* firstCard = nullptr;
46
47     explicit CardQueue(int count, Card** cards);
48
49     bool isEmpty() const;
50
```

```
51      void addCard(Card* card);
52
53      Card* takeCard();
54
55      Card** getCardsArray() const;//利用数组指针返回所有卡牌
56
57      void println() const;
58  };
59
60  struct AbstractPlayer {
61      int cardCount = 0;
62      Card* firstCard = nullptr;
63
64      virtual int selectCard(Card* lastCard) = 0;
65
66      virtual CardColor selectColor() = 0;
67
68      bool finished() const;
69
70      Card* getCardAt(int index) const;
71
72      void grabCard(CardQueue& queue, int count);
73
74      void playCard(CardQueue& queue, int index);
75
76      bool validateCard(Card* lastCard, int index) const;
77
78      Card** getCardsArray() const;//利用数组指针返回所有卡牌
79
80      void println() const;
81
82      void printValid(Card* lastCard) const;
83  };
84
85  struct AiPlayer : AbstractPlayer {
86      int selectCard(Card* lastCard) override;
87
88      CardColor selectColor() override;
89  };
```

```
90
91  struct Player : AbstractPlayer {
92      int selectCard(Card* lastCard) override;
93
94      CardColor selectColor() override;
95  };
96
97  struct Application {
98      int aiPlayerCount;
99      Player* player;
100     AiPlayer** aiPlayers;
101     CardQueue queue = CardQueue(0, nullptr);
102
103     int currentPlayer = 0;
104     AbstractPlayer** allPlayers;
105     Card* lastCard = nullptr;
106     bool reversed = false;
107
108     explicit Application(int playerCount);
109
110     void gameLoop();
111
112     int nextPlayer(bool skip) const;
113
114     int defaultCardCount();
115
116     Card** defaultCards();
117
118     void shuffle(int count, Card** cards);
119
120     void printGameInfo() const;
121 };
122
123 #endif // UNO_LOGIC_H
```

uno_logic.cpp

```
1   #include <algorithm>
2   #include <cstdio>
3   #include <ctime>
```

```
4     #include <windows.h>
5     #include "uno_logic.h"
6     #include "visualize.h"
7
8
9     //卡牌类Card三种牌型的构造函数
10    Card::Card(CardColor cardColor, int cardNumber, int card_id) : cardColor(cardColor
11
12    Card::Card(CardType cardType, CardColor cardColor, int card_id) : cardType(cardTyp
13
14    Card::Card(CardType cardType, int card_id) : cardType(cardType) , card_id(card_id)
15
16
17    //卡牌类Card的info函数
18    const char* Card::info(bool enableExtraColor) const {
19        char* info = new char[20];
20        const char* color = cardColor == blue ? "blue   " : (cardColor == red ? "red
" : (cardColor == yellow ? "yellow" : "green  "));
21        if (enableExtraColor) {
22            if (cardType == WILD) sprintf(info, "| WILD         | %s |    |", color);
23            else if (cardType == WILD_DRAW_4) sprintf(info, "| WILD DRAW 4 | %s |
|", color);
24            else if (cardType == NUMBER) sprintf(info, "| NUMBER       | %s | %d |", co
25            else if (cardType == SKIP) sprintf(info, "| SKIP         | %s |
|", color);
26            else if (cardType == REVERSE) sprintf(info, "| REVERSE      | %s |
|", color);
27            else if (cardType == DRAW_2) sprintf(info, "| DRAW 2       | %s |
|", color);
28        }
29        else {
30            if (cardType == WILD) sprintf(info, "| WILD         |         |
|");
31            else if (cardType == WILD_DRAW_4) sprintf(info, "| WILD DRAW 4 |
|    |");
32            else if (cardType == NUMBER) sprintf(info, "| NUMBER       | %s | %d |", co
33            else if (cardType == SKIP) sprintf(info, "| SKIP         | %s |
|", color);
```

```
34              else if (cardType == REVERSE) sprintf(info, "| REVERSE      | %s |
|", color);
35              else if (cardType == DRAW_2) sprintf(info, "| DRAW 2       | %s |
|", color);
36          }
37          return info;
38      }
39
40      CardQueue::CardQueue(int count, Card** cards) {
41          if (count == 0) return;
42          cardCount = count;
43          firstCard = cards[0];
44          Card* current = firstCard;
45          for (int i = 1; i < count; i++) {
46              current->nextCard = cards[i];
47              current = current->nextCard;
48          }
49          current->nextCard = nullptr;
50      }
51
52      bool CardQueue::isEmpty() const {
53          return firstCard == nullptr;
54      }
55
56      void CardQueue::addCard(Card* card) {
57          cardCount++;
58          if (firstCard == nullptr) {
59              firstCard = card;
60              card->nextCard = nullptr;
61          }
62          else {
63              Card* current = firstCard;
64              while (current->nextCard != nullptr) current = current->nextCard;
65              current->nextCard = card;
66              card->nextCard = nullptr;
67          }
68      }
69
70      Card* CardQueue::takeCard() {
```

```
71          if (firstCard == nullptr) return nullptr;
72          cardCount--;
73          Card* remove = firstCard;
74          firstCard = remove->nextCard;
75          return remove;
76      }
77
78      Card** CardQueue::getCardsArray() const {
79          Card** array = new Card * [cardCount];
80          Card* current = firstCard;
81          for (int i = 0; i < cardCount; i++) {
82              array[i] = current;
83              current = current->nextCard;
84          }
85          return array;
86      }
87
88      void CardQueue::println() const {
89          printf("Queue cards [%d]:\n", cardCount);
90          Card* current = firstCard;
91          while (current != nullptr) {
92              printf("    %s\n", current->info(false));
93              if (current->nextCard == nullptr) break;
94              current = current->nextCard;
95          }
96          printf("\n");
97      }
98
99      bool AbstractPlayer::finished() const {
100         return cardCount == 0;
101     }
102
103     Card* AbstractPlayer::getCardAt(int index) const {
104         Card* current = firstCard;
105         for (int i = 0; i < index; i++) {
106             if (current == nullptr) return current;
107             current = current->nextCard;
108         }
109         return current;
```

```
110     }
111
112     void AbstractPlayer::grabCard(CardQueue& queue, int count) {
113         for (int i = 0; i < count && !queue.isEmpty(); i++) {
114             if (firstCard == nullptr) {
115                 firstCard = queue.takeCard();
116                 firstCard->nextCard = nullptr;
117             }
118             else {
119                 Card* current = firstCard;
120                 while (current->nextCard != nullptr) current = current->nextCard;
121                 Card* take = queue.takeCard();
122                 current->nextCard = take;
123                 take->nextCard = nullptr;
124             }
125             cardCount++;
126         }
127     }
128
129     void AbstractPlayer::playCard(CardQueue& queue, int index) {
130         if (firstCard == nullptr) return;
131         Card* previous = nullptr;
132         Card* played = firstCard;
133         for (int i = 0; i < index; i++) {
134             if (played->nextCard == nullptr) break;
135             previous = played;
136             played = played->nextCard;
137         }
138         if (previous != nullptr) previous->nextCard = played->nextCard;
139         else firstCard = firstCard->nextCard;
140         cardCount--;
141         queue.addCard(played);
142     }
143
144     bool AbstractPlayer::validateCard(Card* lastCard, int index) const {
145         if (lastCard == nullptr) {
146             if (index >= cardCount) return false;
147             return true;
148         }
```

```
149         Card* card = getCardAt(index);
150         switch (card->cardType) {
151         case NUMBER:
152             return card->cardColor == lastCard->cardColor || (lastCard->cardType == NU
153         case SKIP:
154         case REVERSE:
155         case DRAW_2:
156             return card->cardColor == lastCard->cardColor;
157         case WILD:
158         case WILD_DRAW_4:
159             return true;
160         }
161         return false;
162     }
163 
164     Card** AbstractPlayer::getCardsArray() const {
165         Card** array = new Card * [cardCount];
166         Card* current = firstCard;
167         for (int i = 0; i < cardCount; i++) {
168             array[i] = current;
169             current = current->nextCard;
170         }
171         return array;
172     }
173 
174     void AbstractPlayer::println() const {
175         printf("Player cards [%d]:\n", cardCount);
176         Card* current = firstCard;
177         while (current != nullptr) {
178             printf("    %s\n", current->info(false));
179             if (current->nextCard == nullptr) break;
180             current = current->nextCard;
181         }
182         printf("\n");
183     }
184 
185     void AbstractPlayer::printValid(Card* lastCard) const {
186         printf("Your remaining card count: %d\n", cardCount);
187         bool cardResult[108] = { false };
```

```
188         for (int i = 0; i < cardCount; i++) cardResult[i] = validateCard(lastCard, i);
189         printf("Your cards:\n");
190         for (int i = 0; i < cardCount; i++) {
191             printf("    %s%d - %s %s\n", i >= 100 ? "" : (i >= 10 ? " " : "
    "), i, getCardAt(i)->info(false), cardResult[i] ? "  valid" : "invalid");
192             Sleep(100);
193         }
194         printf("\n");
195 
196         // Card** temp = (Card**)malloc(cardCount * sizeof(Card*));
197 
198         Card* temp[120];
199         bool result[120];
200          for (int i = 0; i < cardCount; i++){
201              temp[i] = getCardAt(i);
202              result[i] = cardResult[i];
203          }
204         // printf("%d\n", 80);
205          show_hand_cards(temp, result, cardCount);
206          //printf("%d\n", 90);
207         // printf("%d\n", 100);
208     }
209 
210     int AiPlayer::selectCard(Card* lastCard) {
211         for (int i = 0; i < cardCount; i++) if (validateCard(lastCard, i)) return i;
212         return -1;
213     }
214 
215     //这段AI出牌的逻辑可以重写
216     CardColor AiPlayer::selectColor() {
217         int sum[4] = { 0, 0, 0, 0 };
218         for (int i = 0; i < cardCount; i++) sum[getCardAt(i)->cardColor]++;
219         int maxSum = sum[0];
220         int result = 0;
221         for (int i = 1; i < 3; i++)
222             if (sum[i] > maxSum) {
223                 maxSum = sum[i];
224                 result = i;
225             }
```

```
226         return static_cast<CardColor>(result);
227     }
228 
229     int Player::selectCard(Card* lastCard) {
230         printf("Please select the card you want to play.\n");
231         //int card = select_card_visualize();//利用UI函数来选择卡牌
232         int card = 0;
233         card = select_card_screen();
234         //scanf_s("%d", &card);//输入选择的卡牌,
235         printf("entered: %d\n", card);
236         return card; /*>= cardCount ? cardCount - 1 : card < 0 ? -1 : card;*/
237     }
238 
239     CardColor Player::selectColor() {
240         printf("Please enter the color you want to choose.\n");
241         printf("    0 - red\n");
242         printf("    1 - green\n");
243         printf("    2 - blue\n");
244         printf("    3 - yellow\n");
245         //int color = select_color_visualize();//当UI函数,打出WILD牌或WILD_DRAW_4牌时,
246         int color = 0;
247         color = select_color_visualize();
248         /*do { scanf_s("%d", &color); } while (color < 0 || color > 3);*/
249         return static_cast<CardColor>(color);
250     }
251 
252     Application::Application(int playerCount) {
253         aiPlayerCount = playerCount > 8 ? 7 : playerCount - 1;
254         player = new Player();
255         aiPlayers = new AiPlayer * [aiPlayerCount];
256         for (int i = 0; i < aiPlayerCount; i++) aiPlayers[i] = new AiPlayer();
257 
258         allPlayers = new AbstractPlayer * [aiPlayerCount + 1];
259         allPlayers[0] = player;
260         for (int i = 0; i < aiPlayerCount; i++) allPlayers[i + 1] = aiPlayers[i];
261 
262         Card** cards = defaultCards();
263         shuffle(defaultCardCount(), cards);
264         queue = CardQueue(defaultCardCount(), cards);
```

```
265     }
266
267     void Application::gameLoop() {
268         player->grabCard(queue, 7);
269         for (int i = 0; i < aiPlayerCount; i++) aiPlayers[i]->grabCard(queue, 7);
270
271         while (true) {
272             if (currentPlayer == 0) printGameInfo();
273
274             bool end = true;
275             for (int i = 0; i < aiPlayerCount; i++)
276                 if (!aiPlayers[i]->finished()) {
277                     end = false;
278                     break;
279                 }
280             if (end) {
281                 printf("Game over!");//游戏结束,这里可以用一个UL界面来显示游戏结果
282                 break;
283             }
284
285             bool skip = false;
286             int selected = allPlayers[currentPlayer]->selectCard(lastCard);
287             if (selected < 0) {
288                 allPlayers[currentPlayer]->grabCard(queue, 1);
289             }
290             else {
291                 Card* selectedCard = allPlayers[currentPlayer]->getCardAt(selected);
292                 allPlayers[currentPlayer]->playCard(queue, selected);
293                 lastCard = selectedCard;
294
295                 if (currentPlayer == 0 && player->finished()) {
296                     printf("You won!");
297                     break;
298                 }
299
300                 switch (selectedCard->cardType) {
301                 case NUMBER:
302                     break;
303                 case SKIP:
```

```
304                     skip = true;
305                     break;
306                 case REVERSE:
307                     reversed = !reversed;
308                     break;
309                 case DRAW_2:
310                     allPlayers[nextPlayer(false)]->grabCard(queue, 2);
311                     break;
312                 case WILD:
313                     lastCard->cardColor = allPlayers[currentPlayer]->selectColor();
314                     break;
315                 case WILD_DRAW_4:
316                     lastCard->cardColor = allPlayers[currentPlayer]->selectColor();
317                     allPlayers[nextPlayer(false)]->grabCard(queue, 4);
318                     break;
319                 }
320             }
321             currentPlayer = nextPlayer(skip);
322         }
323     }
324
325     int Application::nextPlayer(bool skip) const {
326         int move = skip ? 2 : 1;
327         int current = currentPlayer;
328         while (move > 0) {
329             current += (reversed ? -1 : 1);
330             if (current < 0) current += (aiPlayerCount + 1);
331             current %= (aiPlayerCount + 1);
332             if (!allPlayers[current]->finished()) move--;
333         }
334         return current;
335     }
336
337     int Application::defaultCardCount() {
338         return 108;
339     }
340
341     Card** Application::defaultCards() {
342         Card** cards = new Card * [108];
```

```
343         int i, j, k;
344         //初始化数字卡牌
345         for (i = 0; i < 4; i++) {//4种颜色
346             for (j = 0; j < 10; j++) {//10张数字牌
347                 for (k = 0; k < 2; k++) {//每一种都有两个
348                     cards[i * 20 + j * 2 + k] = new Card(CardColor(i), j, i * 20 + j *
349                 }
350             }
351         }
352         //初始化技能牌SKIP/REVERSE/DRAW_2
353         for (i = 0; i < 4; i++) { // 最外层的逻辑,循环四种颜色
354             for (j = 0; j < 3; j++) { // 三种卡牌技能
355                 for (k = 0; k < 2; k++) { // 每张牌有两张
356                     cards[80 + i * 6 + j * 2 + k] = new Card(CardType(j + 1), CardColo
357                 }
358             }
359         }
360         //初始化WILD/WILD_DRAW_4,每个都有两张
361         for (int i = 0; i < 2; i++) {
362             cards[104 + i] = new Card(CardType::WILD, 104+ i);
363             cards[106 + i] = new Card(CardType::WILD_DRAW_4, 106 + i);
364         }
365
366
367         return cards;
368     }
369
370     void Application::shuffle(int count, Card** cards) {
371         srand(time(nullptr));
372         for (int i = 0; i < count - 1; i++) {
373             int change = (rand() % (count - i)) + i;
374             if (i != change) std::swap(cards[i], cards[change]);
375         }
376     }
377
378     void Application::printGameInfo() const {
379         printf("------------------------------------------------------------\n");
380         for (int i = 0; i < aiPlayerCount; i++) {
381             if (aiPlayers[i]->finished()) printf("Player %d has won!\n", i + 1);//可以
```

```
382             else {
383                 printf("Player %d remaining card count: %d\n", i + 1, aiPlayers[i]->ca
384                 show_players(aiPlayerCount);
385             }
386         }
387         printf("\n");
388         player->printValid(lastCard);
389         if (lastCard != nullptr) {
390             printf("Last:      %s\n", lastCard->info(true));//这里用UI显示上一手牌
391             show_last_card(lastCard);
392         }
393     }
```

visualize.cpp

```
1   #include "visualize.h"
2   #include "uno_logic.h"
3   # include <graphics.h>
4   //游戏界面最开始的界面,包括点击开始游戏,
5   Card_image card_images[MAX_CARDS];
6   Player_image player_images[MAX_PLAYERS];
7   Color_card_image color_card_images[5];
8
9   int initialize_startgame()
10  {
11      int choicetemp = 0;
12      // 初始化图形窗口,大小为800x600
13      initgraph(800, 600);
14      IMAGE backgroud_image;
15
16      // 加载背景图片并初始化IMAGE对象
17      loadimage(&backgroud_image, _T("background2.jpg"));
18      putimage(0, 0, &backgroud_image); // 显示背景图片
19      // 获取窗口和图片的大小
20      /*int window_width = getwidth();
21      int window_height = getheight();
22      int image_width = getwidth();
23      int image_height = getheight();*/
24
25      // 计算图片的位置,以确保图片居中
```

```
26          /* int x = (window_width - image_width) / 2;
27           int y = (window_height - image_height) / 2;*/
28          int x;
29          int y;
30
31          // 绘制背景图片
32          setbkcolor(0); // 设置背景色为黑色
33          setfillstyle(SOLID_FILL, BLACK); // 设置填充色为黑色
34
35
36          // 在窗口中显示"UNO Game"字样
37          settextcolor(WHITE); // 设置文本颜色为白色，以提高对比度
38          settextstyle(40, 0, _T("Consolas"));
39          outtextxy(380, 250, _T("UNO"));
40
41          // 绘制"开始游戏"按钮
42          //ctangle(300, 300, 500, 350);
43          outtextxy(330, 310, _T("开始游戏"));
44
45          // 等待用户点击"开始游戏"按钮
46          MOUSEMSG msg;  // 定义鼠标消息
47          while (true)// 循环等待用户点击
48          {
49              // 获取一条鼠标消息
50              msg = GetMouseMsg();
51
52              // 判断是否左键按下
53              if (msg.uMsg == WM_LBUTTONDOWN)
54              {
55                  // 判断是否在"开始游戏"按钮上
56                  if (msg.x >= 300 && msg.x <= 500 && msg.y >= 300 && msg.y <= 350)
57                  {
58                      cleardevice(); // 清除窗口
59
60                      putimage(0,0,&backgroud_image); // 显示背景图片
61
62                      settextcolor(WHITE); // 设置文本颜色为白色
63                      settextstyle(40, 0, _T("Consolas"));
64                      outtextxy(312, 50, _T("2 player"));
```

```
65                      outtextxy(312, 150, _T("3 player"));
66                      outtextxy(312, 250, _T("4 player"));
67                      outtextxy(312, 350, _T("5 player"));
68                      outtextxy(312, 450, _T("6 player"));
69                      outtextxy(312, 550, _T("7 player"));
70                      //判断用户选择了几人游戏
71                      while (true)
72                      {
73                          // 获取一条鼠标消息
74                          msg = GetMouseMsg();
75
76                          // 判断是否在"3 player_images"按钮上
77                          if (msg.uMsg == WM_LBUTTONDOWN)
78                          {
79                              if (msg.x >= 312 && msg.x <= 450 && msg.y >= 50 && msg.y <
80                              {
81                                  choicetemp = 2;
82                                  cleardevice();
83                                  return choicetemp;
84                              }
85                              else if (msg.x >= 312 && msg.x <= 450 && msg.y >= 150 && m
86                              {
87                                  choicetemp = 3;
88                                  cleardevice();
89                                  return choicetemp;
90                              }
91                              else if (msg.x >= 312 && msg.x <= 450 && msg.y >= 250 && m
92                              {
93                                  choicetemp = 4;
94                                  cleardevice();
95
96                                  return choicetemp;
97                              }
98                              else if (msg.x >= 312 && msg.x <= 450 && msg.y >= 350 && m
99                              {
100                                 choicetemp = 5;
101                                 cleardevice();
102
103                                 return choicetemp;
```

```
104                             }
105                             else if (msg.x >= 312 && msg.x <= 450 && msg.y >= 450 && m
106                             {
107                                 choicetemp = 6;
108                                 cleardevice();
109 
110                                 return choicetemp;
111                             }
112                             else if (msg.x >= 312 && msg.x <= 450 && msg.y >= 550 && m
113                             {
114                                 choicetemp = 7;
115                                 cleardevice();
116                                 return choicetemp;
117                             }
118                         }
119                     }
120                 }
121             }
122         }
123 
124         // 刷新图形窗口
125         // 显示背景图片
126 
127         // 清屏，准备开始游戏
128 
129     }
130 
131     //加载所有卡牌图片
132     void initialize_card_image()
133     {
134         loadimage(&card_images[0].image_card, _T("red_0.png"));
135         loadimage(&card_images[1].image_card, _T("red_0.png"));
136 
137         loadimage(&card_images[2].image_card, _T("red_1.png"));
138         loadimage(&card_images[3].image_card, _T("red_1.png"));
139 
140         loadimage(&card_images[4].image_card, _T("red_2.png"));
141         loadimage(&card_images[5].image_card, _T("red_2.png"));
142 
```

```
143         loadimage(&card_images[6].image_card, _T("red_3.png"));
144         loadimage(&card_images[7].image_card, _T("red_3.png"));
145
146         loadimage(&card_images[8].image_card, _T("red_4.png"));
147         loadimage(&card_images[9].image_card, _T("red_4.png"));
148
149         loadimage(&card_images[10].image_card, _T("red_5.png"));
150         loadimage(&card_images[11].image_card, _T("red_5.png"));
151
152         loadimage(&card_images[12].image_card, _T("red_6.png"));
153         loadimage(&card_images[13].image_card, _T("red_6.png"));
154
155         loadimage(&card_images[14].image_card, _T("red_7.png"));
156         loadimage(&card_images[15].image_card, _T("red_7.png"));
157
158         loadimage(&card_images[16].image_card, _T("red_8.png"));
159         loadimage(&card_images[17].image_card, _T("red_8.png"));
160
161         loadimage(&card_images[18].image_card, _T("red_9.png"));
162         loadimage(&card_images[19].image_card, _T("red_9.png"));
163
164         loadimage(&card_images[20].image_card, _T("green_0.png"));
165         loadimage(&card_images[21].image_card, _T("green_0.png"));
166
167         loadimage(&card_images[22].image_card, _T("green_1.png"));
168         loadimage(&card_images[23].image_card, _T("green_1.png"));
169
170         loadimage(&card_images[24].image_card, _T("green_2.png"));
171         loadimage(&card_images[25].image_card, _T("green_2.png"));
172
173         loadimage(&card_images[26].image_card, _T("green_3.png"));
174         loadimage(&card_images[27].image_card, _T("green_3.png"));
175
176         loadimage(&card_images[28].image_card, _T("green_4.png"));
177         loadimage(&card_images[29].image_card, _T("green_4.png"));
178
179         loadimage(&card_images[30].image_card, _T("green_5.png"));
180         loadimage(&card_images[31].image_card, _T("green_5.png"));
181
```

```
182         loadimage(&card_images[32].image_card, _T("green_6.png"));
183         loadimage(&card_images[33].image_card, _T("green_6.png"));
184
185         loadimage(&card_images[34].image_card, _T("green_7.png"));
186         loadimage(&card_images[35].image_card, _T("green_7.png"));
187
188         loadimage(&card_images[36].image_card, _T("green_8.png"));
189         loadimage(&card_images[37].image_card, _T("green_8.png"));
190
191         loadimage(&card_images[38].image_card, _T("green_9.png"));
192         loadimage(&card_images[39].image_card, _T("green_9.png"));
193
194         loadimage(&card_images[40].image_card, _T("blue_0.png"));
195         loadimage(&card_images[41].image_card, _T("blue_0.png"));
196
197         loadimage(&card_images[42].image_card, _T("blue_1.png"));
198         loadimage(&card_images[43].image_card, _T("blue_1.png"));
199
200         loadimage(&card_images[44].image_card, _T("blue_2.png"));
201         loadimage(&card_images[45].image_card, _T("blue_2.png"));
202
203         loadimage(&card_images[46].image_card, _T("blue_3.png"));
204         loadimage(&card_images[47].image_card, _T("blue_3.png"));
205
206         loadimage(&card_images[48].image_card, _T("blue_4.png"));
207         loadimage(&card_images[49].image_card, _T("blue_4.png"));
208
209         loadimage(&card_images[50].image_card, _T("blue_5.png"));
210         loadimage(&card_images[51].image_card, _T("blue_5.png"));
211
212         loadimage(&card_images[52].image_card, _T("blue_6.png"));
213         loadimage(&card_images[53].image_card, _T("blue_6.png"));
214
215         loadimage(&card_images[54].image_card, _T("blue_7.png"));
216         loadimage(&card_images[55].image_card, _T("blue_7.png"));
217
218         loadimage(&card_images[56].image_card, _T("blue_8.png"));
219         loadimage(&card_images[57].image_card, _T("blue_8.png"));
220
```

```
221         loadimage(&card_images[58].image_card, _T("blue_9.png"));
222         loadimage(&card_images[59].image_card, _T("blue_9.png"));
223
224         loadimage(&card_images[60].image_card, _T("yellow_0.png"));
225         loadimage(&card_images[61].image_card, _T("yellow_0.png"));
226
227         loadimage(&card_images[62].image_card, _T("yellow_1.png"));
228         loadimage(&card_images[63].image_card, _T("yellow_1.png"));
229
230
231         loadimage(&card_images[64].image_card, _T("yellow_2.png"));
232         loadimage(&card_images[65].image_card, _T("yellow_2.png"));
233
234         loadimage(&card_images[66].image_card, _T("yellow_3.png"));
235         loadimage(&card_images[67].image_card, _T("yellow_3.png"));
236
237         loadimage(&card_images[68].image_card, _T("yellow_4.png"));
238         loadimage(&card_images[69].image_card, _T("yellow_4.png"));
239
240         loadimage(&card_images[70].image_card, _T("yellow_5.png"));
241         loadimage(&card_images[71].image_card, _T("yellow_5.png"));
242
243         loadimage(&card_images[72].image_card, _T("yellow_6.png"));
244         loadimage(&card_images[73].image_card, _T("yellow_6.png"));
245
246         loadimage(&card_images[74].image_card, _T("yellow_7.png"));
247         loadimage(&card_images[75].image_card, _T("yellow_7.png"));
248
249         loadimage(&card_images[76].image_card, _T("yellow_8.png"));
250         loadimage(&card_images[77].image_card, _T("yellow_8.png"));
251
252         loadimage(&card_images[78].image_card, _T("yellow_9.png"));
253         loadimage(&card_images[79].image_card, _T("yellow_9.png"));
254
255
256         //加载红色的技能卡牌,共6张,80-85,顺序为SKIP-REVERSE-PLUS2,每种功能有两张
257
258         loadimage(&card_images[80].image_card, _T("red_skip.png"));
259         loadimage(&card_images[81].image_card, _T("red_skip.png"));
```

```
260
261         loadimage(&card_images[82].image_card, _T("red_reverse.png"));
262         loadimage(&card_images[83].image_card, _T("red_reverse.png"));
263
264         loadimage(&card_images[84].image_card, _T("red_+2.png"));
265         loadimage(&card_images[85].image_card, _T("red_+2.png"));
266
267         //加载绿色的技能卡牌,共6张,86-91,顺序为SKIP-REVERSE-PLUS2,每种功能有两张
268
269         loadimage(&card_images[86].image_card, _T("green_skip.png"));
270         loadimage(&card_images[87].image_card, _T("green_skip.png"));
271
272         loadimage(&card_images[88].image_card, _T("green_reverse.png"));
273         loadimage(&card_images[89].image_card, _T("green_reverse.png"));
274
275         loadimage(&card_images[90].image_card, _T("green_+2.png"));
276         loadimage(&card_images[91].image_card, _T("green_+2.png"));
277
278         //加载蓝色的技能卡牌,共6张,92-97,顺序为SKIP-REVERSE-PLUS2,每种功能有两张
279
280         loadimage(&card_images[92].image_card, _T("blue_skip.png"));
281         loadimage(&card_images[93].image_card, _T("blue_skip.png"));
282
283         loadimage(&card_images[94].image_card, _T("blue_reverse.png"));
284         loadimage(&card_images[95].image_card, _T("blue_reverse.png"));
285
286         loadimage(&card_images[96].image_card, _T("blue_+2.png"));
287         loadimage(&card_images[97].image_card, _T("blue_+2.png"));
288
289         //加载黄色的技能卡牌,共6张,98-103,顺序为SKIP-REVERSE-PLUS2,每种功能有两张
290
291         loadimage(&card_images[98].image_card, _T("yellow_skip.png"));
292         loadimage(&card_images[99].image_card, _T("yellow_skip.png"));
293
294         loadimage(&card_images[100].image_card, _T("yellow_reverse.png"));
295         loadimage(&card_images[101].image_card, _T("yellow_reverse.png"));
296
297         loadimage(&card_images[102].image_card, _T("yellow_+2.png"));
298         loadimage(&card_images[103].image_card, _T("yellow_+2.png"));
```

```
299
300         //加载黑色的WILD和WILD+4,共8张,104-108,顺序为WILD-WILD+4,每张都有两张
301
302         loadimage(&card_images[104].image_card, _T("black_wildcard.png"));
303         loadimage(&card_images[105].image_card, _T("black_wildcard.png"));
304         loadimage(&card_images[106].image_card, _T("black_+4.png"));
305         loadimage(&card_images[107].image_card, _T("black_+4.png"));
306
307     }
308
309     void initialize_player_image()
310     {
311         //printf("%d\n", 113);
312         loadimage(&player_images[0].image_player, _T("player0.jpg"));
313
314         loadimage(&player_images[1].image_player, _T("player1.jpg"));
315
316         loadimage(&player_images[2].image_player, _T("player2.jpg"));
317
318         loadimage(&player_images[3].image_player, _T("player3.jpg"));
319
320         loadimage(&player_images[4].image_player, _T("player4.jpg"));
321
322         loadimage(&player_images[5].image_player, _T("player5.jpg"));
323
324         loadimage(&player_images[6].image_player, _T("player6.jpg"));
325     }
326
327     void initialize_color_card_image() {
328             loadimage(&color_card_images[0].image_color, _T("red.png"));
329
330             loadimage(&color_card_images[1].image_color, _T("green.png"));
331
332             loadimage(&color_card_images[2].image_color, _T("blue.png"));
333
334             loadimage(&color_card_images[3].image_color, _T("yellow.png"));
335
336             loadimage(&color_card_images[4].image_color, _T("black.png"));
337     }
```

```
338
339     //从屏幕上选择WIL牌的颜色
340     int select_color_visualize(){
341         initialize_color_card_image();
342
343       /*  IMAGE backgroud_image_1;
344         loadimage(&backgroud_image_1, _T("background2.jpg"));
345         putimage(0, 0, &backgroud_image_1);*/
346
347         int color_temp = 0;
348         setbkcolor(BLACK);
349         settextcolor(WHITE);
350         settextstyle(20, 0, _T("Arial"));
351         outtextxy(300, 200, _T("选择转牌的颜色"));
352         //将颜色卡片图片显示出来
353         for (int i = 0; i < 4; i++) {
354             putimage(200 + i * 100, 400, &color_card_images[i].image_color);
355         }
356         //等待用户选择颜色,从屏幕上鼠标的位置获取颜色编号
357         MOUSEMSG msg;  // 定义鼠标消息
358         while (true)// 循环等待用户点击
359         {
360             // 获取一条鼠标消息
361             msg = GetMouseMsg();
362
363             // 判断是否左键按下
364             if (msg.uMsg == WM_LBUTTONDOWN)
365             {
366                 // 判断是否在某一个颜色卡片上
367                 for (int i = 0; i < 4; i++) {
368                     if (msg.x >= 100 + i * 100 && msg.x <= 100 + i * 100 + color_card_
369                         && msg.y >= 200 && msg.y <= 200 + color_card_images[i].image_c
370                         color_temp = i;
371                         break;
372                     }
373                 }
374                 break; // 退出循环
375             }
376         }
```

```
377         //closegraph();
378         return color_temp;
379     }
380
381     //从屏幕上获取要打出的卡牌
382     //int select_color_visualize()
383
384     //将单张卡牌显示在屏幕上
385
386     void show_hand_cards(Card** temp, bool* result, int count) {
387
388        /* if (getwidth() != 0) {
389             initgraph(800, 600);
390         }*/
391         //printf("%d\n", 85);
392         IMAGE backgroud_image_1;
393
394         loadimage(&backgroud_image_1, _T("background2.jpg"));
395
396         putimage(0, 0, &backgroud_image_1);
397        // printf("%d\n", 111);
398         attention_grab_card();
399         show_grab_card();
400         draw_line_to_hold_thecards();
401         //printf("%d\n", 221);
402         if (count > 10) {
403             for (int i = 0; i < 10; i++) {
404                // printf("%d\n", 13141);
405                 put_card_screen(temp[i], result[i], 20 + 76 * i, 320);
406                 Sleep(100);
407             }
408             for (int i = 10; i < count; i++) {
409                 //printf("%d\n", 13141);
410                 put_card_screen(temp[i], result[i], 30 + 76 * (i -10), 450);
411                 Sleep(100);
412             }
413         } else if(count < 10) {
414             for (int i = 0; i < count; i++) {
415                 //printf("%d\n", 13141);
```

```
416                 put_card_screen(temp[i], result[i], 30 + 76 * i, 320);
417                 Sleep(100);
418             }
419         }
420         //printf("%d\n", 134);
421         Sleep(500);
422     }
423
424     void put_card_screen(Card* card, bool result, int x, int y) {
425         if (result == false) {
426             putimage(x, y, &card_images[card->card_id].image_card);
427
428         }
429         else {
430             putimage(x, y-20, &card_images[card->card_id].image_card);
431         }
432         return;
433     }
434
435     //从屏幕上选择要打出的卡牌,返回卡牌的编号
436     int select_card_screen() {
437         //printf("%d\n", 2233);
438         //initgraph(800, 600);
439         while (true) {
440             //等待用户点击
441             MOUSEMSG msg;  // 定义鼠标消息
442             while (true)// 循环等待用户点击
443             {
444                 // 获取一条鼠标消息
445                 msg = GetMouseMsg();
446
447                 // 判断是否左键按下
448
449                 if (msg.uMsg == WM_LBUTTONDOWN)
450                 {
451                     // 判断是否在某一个卡片上
452                     for (int i = 0; i < 10; i++) {
453                         if (msg.x >= 30 + 76 * i && msg.x <= 30 + 76 * i + card_images
454                             && msg.y >= 320 && msg.y <= 320 + card_images[i].image_car
```

```
455                             return i;
456                         }
457                     }
458                     for (int i = 10; i < 20; i++) {
459                         if (msg.x >= 30 + 76 * (i -10) && msg.x <= 30 + 76 * (i-10) +
460                             && msg.y >= 450 && msg.y <= 450 + card_images[i].image_car
461                             return i;
462                         }
463                     }
464                     if (msg.x >= 300 && msg.x <= 300 + color_card_images[4].image_colo
465                         return -1;
466                     }
467                     break; // 退出循环
468                 }
469             }
470         }
471         return 0;
472     }
473
474     void show_last_card(Card* lastcard) {
475         putimage(600, 100, &card_images[lastcard->card_id].image_card);
476         //printf("%d\n", lastcard->card_id);
477
478     }
479
480     void show_grab_card(){
481         /*if (color_card_images[4].image_color.getwidth() == 0){
482             initialize_color_card_image();
483         }*/
484         //printf("%d\n", 123);
485
486         //printf("%d\n", 124);
487         putimage(300, 100, &color_card_images[4].image_color);
488         //printf("%d\n", 125);
489         /*rectangle(300, 230, 300 + color_card_images[4].image_color.getwidth(), 230 +
490         settextstyle(36, 0, _T("Arial"));
491         outtextxy(300, 230, _T("抓牌"));
492     }
493
```

```
494     void attention_grab_card() {
495         //printf("%d\n", 112);
496         IMAGE attention;
497         loadimage(&attention, _T("attention.jpg"));
498         putimage(700,500, &attention);
499     }
500 
501     void draw_line_to_hold_thecards() {
502         setlinecolor(RGB(100,100,100));
503         setfillcolor(RGB(100, 100, 100));
504         fillrectangle(0,320 + card_images[0].image_card.getheight(), 800, 326 + card_i
505     }
506     void show_player(int player_id,int x, int y){
507         putimage(x,y,&player_images[player_id].image_player);
508     }
509 
510     void show_players(int aiplayercount) {
511         for (int i = 1; i < aiplayercount+1; i++) {
512             show_player(i, 100 * i, 600);
513         }
514     }
```

5 调试和测试过程

关于核心逻辑和 UI 的部分的调试刘老师上课讲的四大 bug 我真的是没有一个逃得过死循环不知道写出了多少个, 数组越界也是常有的事情, 野指针更是经常写出来导致异常退出; 最开始的时候写很多才调试, 真的是 bug 改不过来, 后来受刘老师提倡的增量式编程的启发, 才逐渐好了一些; 调试中最大的麻烦其实不是来源于核心逻辑实现, 而是调用 easyx 库函数的过程; 图片未正确加载是相当大的麻烦, 而且最要命的是 vs 不能支持调用 easyx 库函数时的 debug, 每次在 debug 下运行都会出现栈溢出错误导致的堆损坏, 非常要命; 最后只能是一点点用 printf 来找问题; 总之是跌跌撞撞给 debug 完了, 但是程序的鲁棒性依然比较差, 在测试中又出现了代码为-1073740940 的退出, 到现在没有解决. 大概是这些吧.

6 AI 部分的设计思路

该部分作者: 赵启元

AI 逻辑设计和 ai 部分的 debug 在 UNO 游戏中,AI 出牌逻辑的设计需要考虑多个因素, 包括出牌的优先级, 对手的可能动作, 当前的牌局状态等. 以下是设计 AI 出牌逻辑的详细思路: 在最初版本中,AI 玩家选取手牌中第一张合法的出牌, 在使用变换颜色牌时, 选取手牌中牌数最多的颜色. 这种逻辑简单且

易实现, 但是由于过于简单, 性能表现较差. 经过测试, 在由 1 名用户玩家和 4 名 AI 玩家组成的对局中, 用户玩家的胜率可以达到约 80-90 为了提升用户的游戏体验, 我们优化了 AI 的出牌逻辑, 使其更具智能. 经过查阅资料和分析实战思路, 选择赋予 AI 根据不同卡牌类型调整优先级的策略.AI 会优先使用功能牌 (加 2, 跳过, 反转), 因为它们具有较强的进攻性, 可以破坏下家的出牌节奏, 降低其胜率, 这种类型的卡牌应当优先使用. 如果没有合法的功能牌, 则使用数字牌, 在数字牌中, 优先选择手牌中牌数较多的颜色. 如果没有合法的数字牌, 则使用万能牌 (变色,+4), 因为这种牌不受颜色限制„ 具有较强的防御性, 可以在缺乏其它合法出牌选择时保护自己免于摸牌, 应当尽可能保留以备不时之需. 通过上述设计思路,AI 在出牌时会根据优先级策略选择最优的牌, 从而提高竞争力. 优先出高威胁牌可以直接影响对手的行动, 其次优先出数量最多颜色的数字牌可以增加后续出牌的灵活性, 而在没有其他选择时出万能牌, 可以确保 AI 能够在关键时刻使用它们. 这种设计使得 AI 在游戏中表现更加智能和具有挑战性. 经过测试, 在与加入优先级出牌逻辑的 AI 玩家对战时, 用户玩家的胜率只能达到约 20-30 在此基础上, 可以进一步优化 AI 的博弈性能. 我们的方案是为 AI 加入出牌历史记忆功能, 并根据历史出牌, 通过概率分析语统计推理分析其他玩家手中的牌, 并针对性地进行制约. 这种 AI 逻辑更加智能, 大幅提高了胜率. 但是由于逻辑较为复杂, 故鲁棒性相对较差,debug 较为困难. 且过强的 AI 会不可避免地导致用户玩家游玩体验下降, 因此在最终版本中并未使用这种逻辑.

7 心得体会

7.1 王子轩

嗯怎么说呢, 其实我觉得这次的大作业确实是一波三折. 有很多想说的, 但一下又不能表达清楚; 从最开始选题组队吧到现在交作业, 真的是感觉一个人抗下了所有, 刘同学全程不回消息, 完全没有做; 赵同学做了一部分的 AI, 但是因为他用的 mac 本, 有非常多的系统不兼容问题, 所以实际上最终版本的大作业 (可视化的版本) 是我一个人做的.1200 多行代码真的是多少个凌晨两三点敲出来的, 网上也几乎没有什么 uno 的 C++ 实现的资料, 有的话也读不太懂 (因为涉及到网络的部分, 包括很多人人联机对战, 技术栈我非常不熟悉), 所以我只能自己摸索. 最终还是把大作业给一点点写出来了但是 debug 的过程确实是无比的痛苦, 一点点找, 因为前期内存没有管理好, 后期只能一点点找, 找到了一个 bug, 解决了一个 bug, 然后又来一个 bug, 真的是无尽的循环. 最终的可视化版本, 我觉得还是有很多的不足, 比如说 UI 不够美观, 代码不够简洁, 功能不够完善, 但是我也没有时间再去完善了, . 玩了几局, 也记录了 4 次的测试文档, 还录了一些玩游戏的视频都放在了文件夹啊里边. 总的来说, 这次的大作业让我学到了很多, 也让我感受到了很多, 我觉得这次的大作业是我大学以来最有意义的一个大作业, 也是我最有成就感的一个大作业, 虽然最终的成果不是很完美, 但是我觉得我付出了努力, 也学到了很多, 这就够了.

7.2 赵启元

在此次编写命令行版本的 UNO 游戏过程中, 我经历了一段充满挑战和收获的旅程. 从最初的设计思路到最终实现, 我不仅提升了编程技能, 还在团队合作和解决问题的过程中积累了宝贵的经验. 编写 UNO 游戏让我对 C/C++ 编程有了更深的理解. 特别是在 OOP 方面, 我们设计了卡牌, 卡牌队列, 玩家和游戏主循环等多个类, 并通过类的继承和多态性来实现不同类型玩家的行为. 这种设计不仅使代码结构清晰, 还提高了代码的可维护性和扩展性. 在实现 AI 逻辑时, 我们从最简单的逻辑开始, 逐步优化, 最终实现了一个智能化的 AI 出牌策略. 在这个过程中, 我学会了如何分析问题, 制定策略, 并通过迭代改

进不断优化代码性能. 这种从简单到复杂, 循序渐进的编程方法, 使我对算法设计和优化有了更深刻的认识. 尽管在设计初期我们对项目进行了详细的规划, 但实际编程过程中, 程序的复杂性和出现的 Bug 还是超出了我们的预料. 最初的版本中,AI 仅能出第一张合法牌, 且变换颜色时选择数量最多的颜色. 这种逻辑简单且易实现, 但性能表现差强人意. 为提升 AI 的竞争力, 我们决定赋予 AI 根据不同卡牌类型调整优先级的策略. 这一改动大幅增加了程序的复杂性. 特别是在选择优先级高的功能牌, 处理数字牌颜色策略以及确保所有出牌都是合法的过程中, 我们遇到了许多意料之外的问题. 在调试过程中, 我们发现 AI 有时会尝试出非法的牌, 导致程序崩溃. 此外, 处理玩家输入的部分也存在一些隐藏的 bug, 比如当玩家输入非法的索引时, 程序并没有正确处理. 这些问题的解决不仅需要我们反复调试和测试, 还需要我们深入理解代码的逻辑和结构. 每次 debug 都是一次成长. 通过反复调试和分析错误记录, 我逐步学会了如何高效地定位问题, 并找到解决方案. 在优化 AI 出牌逻辑的过程中, 我学会了如何从实际游戏策略出发, 设计算法并将其转化为代码. 我们查阅了大量资料, 分析了实际游戏中的策略, 最终确定了优先使用功能牌, 其次使用数量最多颜色的数字牌, 最后使用万能牌的策略. 这种设计使得 AI 在游戏中表现更加智能和具有挑战性. 通过这次项目, 我的编程能力得到了显著提升. 特别是在处理复杂逻辑, 优化算法和调试程序方面, 我积累了许多实战经验. 此外, 我还学会了如何在团队中有效合作, 通过分工协作和互相帮助, 最终完成一个复杂的项目.