Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение
высшего образования
«Санкт-Петербургский политехнический университет Петра Великого»
(ФГАОУ ВО СПбПУ)
Институт _____
Высшая школа _____

КУРСОВАЯ РАБОТА

По дисциплине основы программирования и алгоритмизации
(семестр 2)

Игра в кости (двадцать одно)

Студент

Группы 3530203/20001 _____ Манкиев Д.Р.
подпись, дата

Оценка выполненной студентом работы:

Преподаватель _____ Муньос Н.Э.
подпись,дата

Санкт-Петербург – 2023

САНКТ-ПЕТЕРБУРСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО
Институт компьютерных наук и технологий
Высшая школа искусственного интеллекта

ЗАДАНИЕ

на выполнение курсовой работы по дисциплине
основы программирования и алгоритмизации

Студенту Манкиеву Данилу Робертовичу группа: 3530203/20001 семестр:2

1.Тема работы: кости,игра в ‘двадцать одно‘

2.Сроки сдачи студентом законченной работы: 28.05.2023

3.Исходные данные: отсутствует

4.Содержание работы: создание игры

5.Перечень графического материала:

6.Дата выдачи задания: 11.04.2023

Преподаватель _____ Муньос Н.Э.
подпись,дата

Группы 3530203/20001 _____ Манкиев Д.Р.
подпись, дата

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....4

1.Изучение механики игры в кости двадцать одно.....5

2.Логика алгоритма.....6

2.1.Реализация интерфейса игрока.....6

2.2.Создание базы данных.....10

2.3.Логика выявления победителя.....12

3.Реализация вывода результатов в отдельный файл.....14

4.Инициализация запуска программы при помощи class Program.....15

5.Создание меню при помощи curses.h.....16

6.Тестирование, результаты тестирования.....18

Заключение.....19

Введение

Данная работа предназначена для создания и дальнейшего использования игры под названием ‘Игра в кости 21‘. На заключительном этапе программа должна очуществлять игру между 2 игроками и показывать полученные результаты.

Задачи:
1) Изучить механику игры
2) Реализовать программный интерфейс игрока
3) Создать базу данных с использованием ассоциотивного массива для хранения данных о 2 игроках (виртуальных)
4) Разработать логику программы, которая четко соблюдала механики игры
5) Разработать возможность вывода результатов в отдельный файл .txt
6) Создать класс Program, который инициализировал запуск программы
7) Создать с помощью библиотеки curses.h меню, которой отображалось при запучке программы

Термины, используемые в работе:
1) Игрок - вирутальный персонаж, который будет участвовать в игре согласно логике алгоритма. Сведения об игроке: количество побежденных партий, количество очков, имя, пол (ситуативно).
2) Кость - в данной работе предмет, который способен принимать значение от 1 до 6 случайным образом
3) База данный - ассоциотивный массив с данными об игроках(количество побежденных партий, количество очков, имя, пол (ситуативно))

1.Изучение механики игры в кости двадцать одно

Игра в двадцать одно - это игра, в которой игроки поочереди выбирают бросать им кость или нет. Кость может дать значение от 1 до 6. Игрок, набравший 21 очко или ближайшее к нему число не меньшее 21, победил, игрок, набравший больше 21 очка сгорает, при чем, если все игроки набрали больше 21, выигрывает тот, у кого меньше всего очков.

Так если в игре 2 человека и они сыграт 100 партий, то распределение будет примерно 50 на 50. Для этого ведется учет того, сколько раз определенный игрок выйграл в партию. Также выбор - бросать кость или нет - определяется только самим игроком.

2.Логика алгоритма

2.1.Реализация интерфейса игрока

Реализация игрока происходит с помошью интерфейса class PlayerAbstract от которого наследуются два класса Player и NonbinaryPlayer.

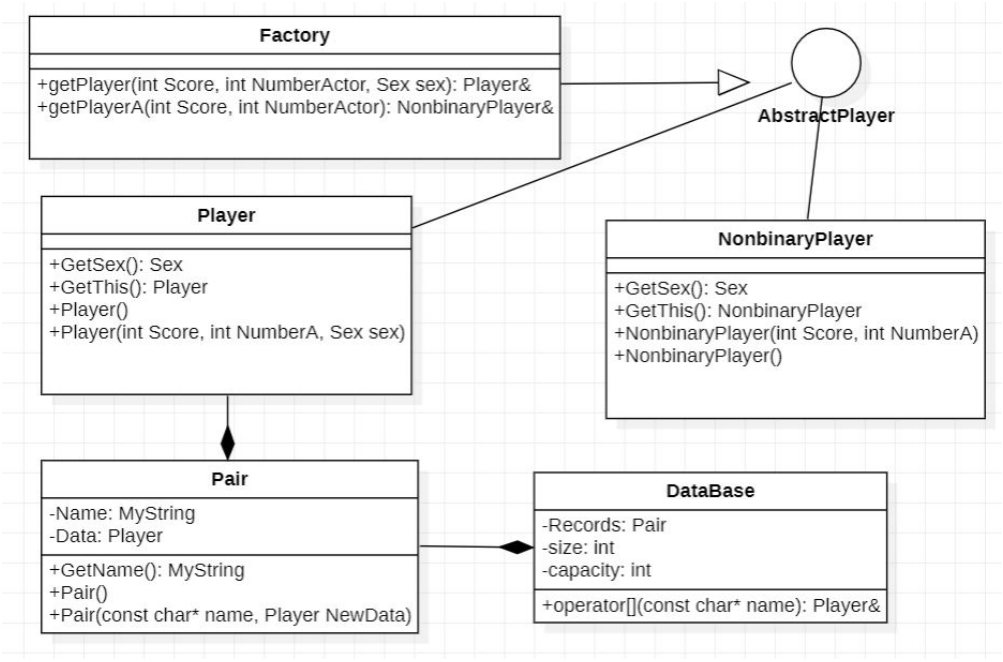

Рис. 1: Диаграмма классов для создания игроков и базы данных

```
class PlayerAbstract {
protected:
        int _Score;
        int _NumberActor;
        Sex _sex;
public:
        bool operator==(PlayerAbstract& ComparePlayer);
        void GetAnother(int dice);
        bool IsLoser();
        int& GetNumberA();
        int& GetScore();

        virtual PlayerAbstract& GetThis() = 0;
```

```
        virtual Sex GetSex() = 0;

};
```

Поле Score определяет число очков на данный момент. NumberActor определяет количество выйгранных партий. sex определяет пол игрока.
Методы PlayerAbstract:

`1) bool operator==(PlayerAbstract& ComparePlayer);`

Реализован для проверки равенства двух игроков, в частности используется наследниками.Вовращяет true, если все поля двух игроков равны(Score, sex, NumberActor).

`2) void GetAnother(int dice)`

Реализован для увеличения Score путем прибавления значение, которое выпало случайном образом от 1 до 6. Это происодит в цикле в соотвествии с вероятностями, при которых, получая еще одно значение, от 1 до 6 игрок не проигрывает. Например, если Score = 16 или Score =17, то с вероятностью 0.5 игрок получит еще одно значения от 1 до 6.

```
void PlayerAbstract::GetAnother(int dice)
{
        _Score = 0;
        srand(time(0));   // Длятого , чтобыкаждыйцикл
        random_device rd; // былослучайноезначение
        mt19937 gen(rd());
        uniform_int_distribution<> dist(1, dice); // Диапазонот
1 до 6
        uniform_int_distribution<> dist1(1, 3); //Диапазон от
1 до 3 включительно
        uniform_int_distribution<> dist2(1, 4); // Диапазонот
1 до 4
        uniform_int_distribution<> dist3(1, 10);
        while (true) {
                if (_Score <= 15) {
                        _Score += dist(gen);
```

```
                }
                else if (_Score == 16 || _Score == 17) {
                        int probability = dist1(gen);
                        if (probability == 1 || probability == 2)
                                _Score += dist(gen);
                        }
                        else {
                                break;
                        }
                }
                else if (_Score == 18 || _Score == 19) {
                        int probability = dist2(gen);
                        if (probability == 1) {
                                _Score += dist(gen);
                        }
                        else {
                                break;
                        }
                }
                else if (_Score == 20) {
                        int probability = dist3(gen);
                        if (probability == 1) {
                                _Score += dist(gen);
                        }
                        else { break; }
                }
                else if (_Score == 21) {
                        _Score += 0;
                        break;
                }
                else if (_Score > 21) {
                        break;
                }
        }
}
```

3) `bool IsLoser();`

Релизован для определения проиграл ли уже игрок (очков >21), возвращяет true, если проиграл.

4) `int& GetNumberA();`

Ревлизоан для получения данных о том, сколько раз выйграл игрок.

5) `int& GetScore();`

Реализован для получения данных о текущем количестве очков игрока.

7)`virtual Sex GetSex() = 0;`

Чисто виртуальная функция предназначенная для получения данных о поле игрока.

8)`virtual PlayerAbstract& GetThis() = 0;`

Чисто виртуальная функция, которая возвращяет указательн элемент.

От класс PlayerAbstract наследуются два класса Player и NonbinaryPlayer, для которых реализован метод

`Sex GetSex();`

так что в классе Player данный метод вовращяет текущий пол игрока, а в классе NonbinaryPlayer всегда возвращяет none.

Все игроки создаются с помощью Factory (шаблон проектирования - Simple Factory)

```
class Factory {
public:
    // созданиеигрока  Player
        Player getPlayer(int Score, int NumberActor, Sex sex);
        // созданиеигрока  NonbinaryPlayer
        NonbinaryPlayer getPlayerA(int Score, int NumberActor);
};
```

2.2.Создание базы данных

База данных представлена ассоциативным массивом, класс реализации пар в этом массиве Pair имеет 2 поля

```
MyString _name;
Player _data;
```

В переменной name хранится имя игрока, а переменной data хранится данные этого игрока

```
class Pair
{
        MyString _name;
        Player _data;
public:
        MyString& GetName();
        bool operator==(const char* name);
        Pair();
        Pair(const char* name, Player newData);
        friend class DataBase;
};
```

Сама база данных представлен классом DataBase:

```
class DataBase {
        Pair** records;
        int size;
        int capacity;
public:
        Player& operator[](const char* name);
        DataBase() : size(0), records(nullptr), capacity(0) {};

};
```

Pair** records хранит данные об игроках, int size хранит размер массива, int capacity хранит свободное место.
Перегрузка опертора индексирования позволяет записывать данные по ключу, а ключ это имя игрока, если игрока с таким именем нет, то он автоматически создается и туда присваиваются данные.

```
Player& operator[](const char* name);
```

```
Player& DataBase::operator[](const char* name) {

        for (int i = 0; i < size; i++) {
                if (**(records + i) == name)
                        return records[i]->_data;
        }

        if (size >= capacity) {
                Pair** tmp = new Pair * [capacity + 10];
                capacity += 10;
                for (int i = 0; i < size; i++)
                {
                        tmp[i] = records[i];
                }
                delete[] records;
                records = tmp;
        }
        Player* exm = new Player;
        records[size] = new Pair(name, *exm);
        delete exm;
        return records[size++]->_data;
}
```

Пример использования:

```
Database["Ivan"] = factor.getPlayer(0, 0, male);
```

Имя в базе данных представлено классом MyString, который является аналогом char*

2.3.Логика выявления победителя

Логика выявления победителя представлена классом class Game в GameAlgorithm.h

```
class Game {
        Player* _first;
        Player* _second;
public:
        Game(Player* first, Player* second);
        Player& process();
};
```

Класс Game включает две переменные first и second, которые хранят данные об игроках.

`Game(Player* first, Player* second)`

реализует основную логику выйграша в соотвествии с механикой игры в кости(см. пункт 1.).

```
Player& Game::process() {
        Player* draw = new Player();
        _first->GetAnother(6);
        _second->GetAnother(6);
        if (_first->GetScore() <= 21 && _second->GetScore() <= 21)
        {
    return (_first->GetScore() > _second->GetScore() ?
    _first->GetThis(): _first->GetScore()==_second->GetScore() ?
    draw->GetThis() : _second->GetThis());
        }
        else if (_first->GetScore()>21 && _second->GetScore()<=21)
        {
    return _second->GetThis();
        }
        else if (_first->GetScore()<= 21&&_second->GetScore()>21)
        {
    return _first->GetThis();
        }
        else if (_first->GetScore()>21&& _second->GetScore()>21)
        {
```

```
        return (_first->GetScore()<_second->GetScore() ?
        _first->GetThis() : _first->GetScore()==_second->GetScore() ?
        draw->GetThis(): _second->GetThis());
            }
}
```

Для этго игрок first и игрок second помощью метода .GetAnother(int dice) получают определенное кол-во очков, потом конечные значения Score обоих игроков сравниваются и выявляется победитель в соотвествии с механикой(см. пункт 1.)

3.Реализация вывода результатов в отдельный файл

Реализация вывода результатов в файл происходит в файле OutIn.h посредством класса class WriteIn

```
class WriteIn {
public:
        void Out(int a, int b);
};
```

В метод

```
void Out(int a, int b);
```

передаются количество выйгранных партий 1-ого игрока и 2-ого игрока. После чего эти два числа выводятся в файл названия, которого вводит пользователь.

```
void WriteIn::Out(int a, int b)
{
        std::cout << "Enter Output File Name  - ";
        char ar[80];
        std::cin >> ar;
        std::ofstream fout(ar);
        fout<<"First:"<< a <<"Second:"<< b << " Draw: "
        << 100 - a - b << std::endl;
        fout.close();
}
```

В файл выводится кол-во попед первого, второго игрока и кол-во ничьих, когда кол-во очков совпало.

4.Инициализация запуска программы при помощи class Program

Инициализация происходит при помощи класса Program, в котором есть метод void Exec(), который создаёт фабрику factor для создания 2-ух игроков, базу данных db, экземпляр класса Game с дынными первого и второго игрока. Далее в цикле от 1 до 100 происходит выявление победителя по логике, описанной в пункте 2.3. После чего полученные данные передаются параметрами в метод Out класса WriteIn, а далее выводятся в файл в соответсвии с пунктом 3.

```
 class Program {
public:
        void Exec();
};

 void Program::Exec()
{
        Factory factor;
        DataBase db;
        db["First"] = factor.getPlayer(0, 0, male);
        db["Second"] = factor.getPlayer(0, 0,none);
        Game game1(&db["First"], &db["Second"]);
        for (int i = 0; i < sto; i++) {
                game1.process().GetNumberA()++;
                db["First"].GetScore() = 0;
                db["Second"].GetScore() = 0;
        }
        WriteIn o;
        o.Out(db["First"].GetNumberA(), db["Second"].GetNumberA())
}
```

5.Создание меню при помощи curses.h

Создание меню происходит с помощью class menu, для этого в нем реализуются 2 метода WINDOW* getwin() и void window().

```
class menu {
public:
        WINDOW* getwin();
        void window();
};
```

Метод WINDOW* getwin() создает окно, в котором расположены три строки:
‘Dice Game‘
‘Start game(s)‘
‘End game(e)‘
Метод void window() создает бесконечный цикл, который считывает клавиши, нажатые пользователем. Так буква ‘s‘ запускает программу с помощью класса Program(см. пункт 4.), далее пользоветель вводит имя файла, где будут сохранятся результаты. После завершения программы окно снова появляется и ждет действия пользоваетеля. Для завершения программы требуется нажать на букву ‘e‘.

```
WINDOW* menu::getwin()
{
        initscr();
        int xMax, yMax;
        noecho();
        curs_set(0);
        getmaxyx(stdscr, yMax, xMax);
        WINDOW* window = newwin(yMax/2, xMax/2, yMax/4, xMax/4);
        box(window, 0, 0);

        mvwprintw(window, 0, 25, "Dice Game");
        mvwprintw(window, 3, 24, "Start game(s)");
        mvwprintw(window, 6, 24, "End game(e)");
        return window;
}
void menu::window()
```

```
{
        this->getwin();
        char ch;
        while (ch = wgetch(this->getwin()))
        {
                switch (ch) {
                case 's':
                        wgetch(this->getwin());
                        endwin();
                        Program tmp;
                        tmp.Exec();
                        break;
                case 'e':
                        exit(0);
                        break;
                }
        }
}
```

6.Тестирование, результаты тестирования

Рис. 2: Окно пользователя при запуске программы

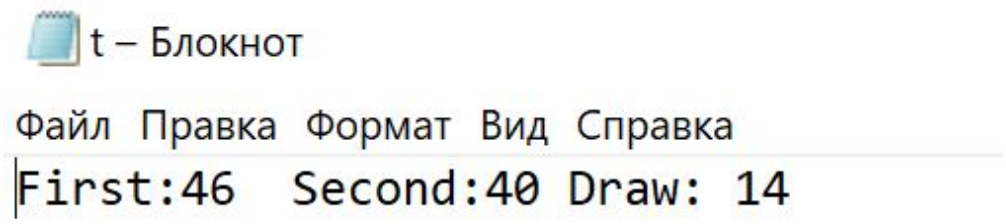

Рис. 3: Пример вывода результатов после 100 отыгранных партий

Тестирование показало хорошее соотвествие логики программы с механикой игры и распределением победителя поровну 50 на 50.

Заключение

В ходе выполнения данной курсовой работы было изучена механика игры в ксоти двадцать одно, распределение побдеителей на больших количествах партий. Было освоено, каким образом должна работать программа для корректного выполнения и соблюдения всех правил игры. На основе этого был реализован интерфейс Player с данными об количестве очков, количестве побед, поле. Для хранения данных был реализована база данных на основе ассоциотивного массива. Логика программы была описана корректно, так как тестирование дало положительную оценку. Также был предусмотрена возможность вывода результатов в файл и запуска программы через один класс Program. Пользовательский интерфейс был представлен в виде окна с использованием бибиотеки curses.h.

СПИСОК ИСПОЛЬЗВАННЫХ ИСТОЧНИКОВ

1.Павловская, Т.А. C/C++. Программирование на языке высокого уровня: учебник. / Т.А. Павловская. – М.; СПб.; Н.Новгород: Питер, 2007.- 460 с.