Оглавление

Задание на курсовую работу

Описание возможностей и алгоритма работы программы

В данной курсовой работе реализована программа, симулирующая ИИ для игры в Техасский холдем с одним или несколькими управляемыми программой игроками. Для написания данной программы были спроектированы и реализованы классы игрока, стола, или, крупье, и игральной карты. Все классы были выделены в отельные h файлы.

Класс карты «card»

Класс карты имеет специализированный конструктор, заполняющий поля масть (suit) и номинал (nominal) и ведущий реестр карт, находящихся в игре. При создании карты ее номинал и масть сравниваются с картами находящимися в игре, чтобы исключить возможность появления в игре одинаковых карт. При уничтожении карты (вызове деструктора) данная карта изымается из реестра, и она вновь может выпасть.

Класс игрока «player»

В классе игрока реализованы функции подсчета вероятности сбора комбинаций (используется упрощенная формула вычисления математического ожидания карт улучшающих руку, не учитываются шансы сбора комбинаций противником) и возможность управления действиями играющим.

Класс – обертка «party»

Данный класс реализует интерфейс для группы игроков, его функции простое переключение между игроками, проверка эквивалентности ставок игроков в игре, очистка списка от игроков не способных продолжать игру (со значением член-данного cash = 0).

Класс стол «desk» (крупье)

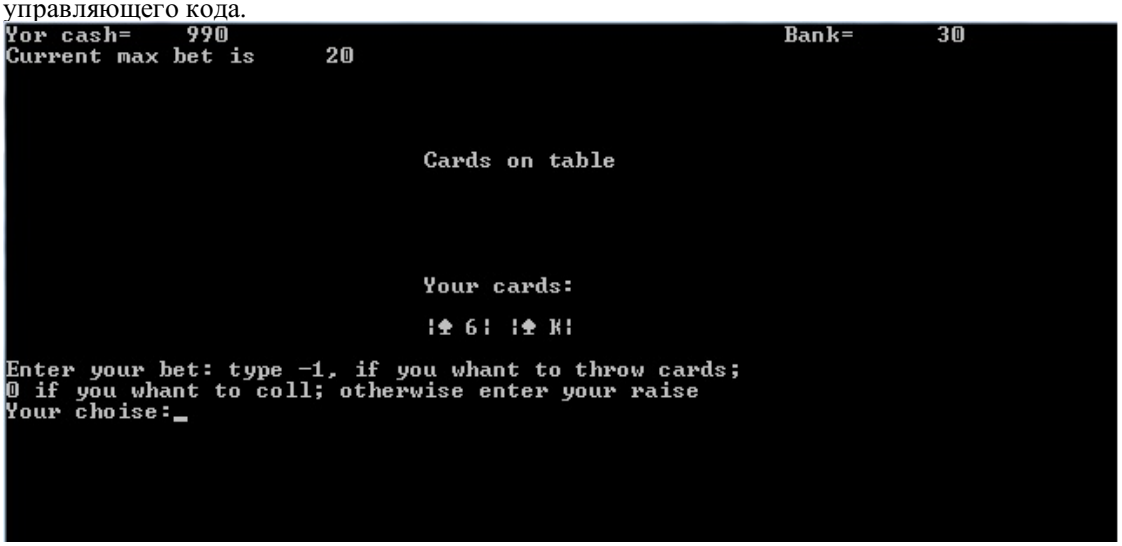
Данный класс реализует управление игрой (сбор обязательных ставок, выбор выигравшего игрока, сбор ставок и распределение банка между победителями). Кроме того данный класс управляет видимостью карт на столе и предоставляет их игрокам.

Руководство по запуску программы

При запуске исполняемого файла Holdem.exe в командной строке появится приглашение ввести количество игроков (суммарное количество игроков, учитывая, что 1 из них - играющий)

Число игроков программно ограничено: минимальное число игроков – 2, максимальное – 10.

После ввода числа игроков программа создаст стол с заданным числом игроков, раздаст карты и покажет информацию о игроке, управляемом пользователем, с приглашением ввода управляющего кода.



При вводе отрицательного числа игрок сбросит карты и откажется от дальнейшей от игры в данной раздаче, при вводе числа ноль игрок уравняет ставку, при вводе числа большего 0 повысит текущую максимальную ставку на данное число.

При вводе положительного числа вы играющий сможет участвовать в дальнейших торгах (при трех, четырех и пяти открытых картах на столе). При пяти открытых общих картах

идет последний заключительный круг торговли, после которого программа сравнивает комбинации всех игроков не сбросивших к данному моменту карты, определяет лучшую комбинацию и передает банк игроку собравшему лучшую комбинацию, если таковых оказалось несколько, то согласно правилам игры банк делится между всеми собравшими такую комбинацию поровну.

Окно игры на момент последнего круга торговли.

После окончания очередного кона игры программа завершается если у игрока управляемого пользователем поле cash приобрело значение 0, и выводит приглашение ко вводу управляющего символа y или n. Ввод символа y стартует новый кон игры, ввод n приводит к выходу из программы.

Приложение 1. Листинг кода программы

card.h

```
#pragma once
//#define _CRT_RAND_S in stdafx.h
//use <stdexcept>
#ifndef deck_size //default deck size
#define deck_size 52
#endif // !deck_size
std::runtime_error too_many_cards("Error: Too many cards");
class card // instances of this class cannot be copied or assigned
{
public:
	card();
	~card();
	int getsuit() const;
	int getnominal() const;
	int getcard() const;
	void printcard();
private:
	unsigned int suit;
	unsigned int nominal;
	static int cards_in_game;
	static card *  tcards[deck_size];
	card(const card&); //no copy!
	void operator=(const card&); //and no assignment
};
int card::cards_in_game=0;
card * card:: tcards[]={};
bool sdeck(const  card * a,const card * b)
{
	return a->getcard() < b->getcard();
}
bool ssuits(const card * a,const card * b)
{
	return a->getsuit()<b->getsuit();
}
bool snominals(const  card * a,const card * b)
{
	return a->getnominal()<b->getnominal();
}
card::card() //this function MUST be rewrited for LINUX version
{
	if(cards_in_game>=deck_size){throw too_many_cards;} //protection from the endless cycle
	errno_t err;
	unsigned int tmp;
	bool isdouble;
	do
	{
		isdouble=false;
		err = rand_s(&tmp);
		suit=(unsigned int)    ((double)tmp/((double) UINT_MAX + 1) * 4) + 1;
		err= rand_s(&tmp);
		nominal=(unsigned int) ((double)tmp/((double) UINT_MAX + 1) * 13) +2;
		for(int i=0;i<cards_in_game;i++)
		{
			if(this->nominal==_tcards[i]->nominal && this->suit==_tcards[i]->suit){isdouble=true;}
		}
	}
	while(isdouble);
	 tcards[cards_in_game]=this;
	cards_in_game++;
}
card::~card()
{
```

```
        for(int i=0;i<cards_in_game;i++)
        {//look for card in deck
                if(this==_tcards[i])
                {
                        for(int j=i;j<cards_in_game-1;j++){_tcards[j]=_tcards[j+1];}
                        cards_in_game--;
                         tcards[cards_in_game]=nullptr;
                        break;
                }
        }
}
int card::getsuit() const{return suit;}
int card::getnominal() const{return nominal;}
int card::getcard() const{return ((suit-1)*13+nominal);}
void card::printcard()
{
        char picnom;
        bool ispicture=true;
        switch (nominal)
        {
        case 11: picnom='V'; break;
        case 12: picnom='D'; break;
        case 13: picnom='K'; break;
        case 14: picnom='T'; break;
        default: ispicture=false; break;
        }
        std::cout<<(char)(suit+2)<<' ';
        if(ispicture) std::cout<<picnom;
        else std::cout<<nominal;
}
```

player.h

```
int combination(card* c1,card* c2,card* c3,card* c4,card* c5)
{
        bool flash(false),straight(true);
        card * tested[]={c1,c2,c3,c4,c5};
        std::sort(tested,tested+5,ssuits);
        if(tested[0]->getsuit()==tested[4]->getsuit()) flash=true;
        std::sort(tested,tested+5,snominals);
        for (int i = 0; i < 4; i++)
        {
                if(tested[i]->getnominal()!=tested[i+1]->getnominal()-1)
                {
                        straight=false;
                        break;
                }
        }
        if (straight && flash) return 80000+(tested[4]->getnominal())*100;
        int seqences[2]={1,1},seqnom[2]={0,0},j=0;
        for (int i = 0,j=0; i < 4; i++)
        {
                if(tested[i]->getnominal()==tested[i+1]->getnominal())
                {
                        seqences[j]++;
                        seqnom[j]=tested[i]->getnominal();
                }
                else if (seqences[j]!=1)
                {
                        j++;
                }
        }
        if (seqences[0]==4) return 70000+seqnom[0]*100;
        if (seqences[0]==3 || seqences[1]==3)
        {
                if (seqences[1]==2)
```

```
                {
                        return 60000+seqnom[0]*100+seqnom[1];
                }
                if (seqences[0]==2)
                {
                        return 60000+seqnom[1]*100+seqnom[0];
                }
                if(!(straight || flash))
                {
                        return 30000 + seqnom[0]*100;
                }
        }
        if(flash) return 50000+100*tested[4]->getnominal();
        if(straight) return 40000+100*tested[4]->getnominal();
        if (seqences[0]==2)
        {
                if (seqences[1]==2)
                {
                        if(seqnom[0]>seqnom[1]) return 20000+100*seqnom[0]+seqnom[1];
                        else return 20000+100*seqnom[1]+seqnom[0];
                }
                return 10000+100*seqnom[0];
        }
        return 0;
}
class player
{
public:
        bool is_alive;  // true if it's a live player, 0 if computer need
        bool in_game;   //true if player didn't throw cards          need
        bool is_all_in; //true if all in                             need
        double bet;                                     //       need
        double cash;                                     //       need
        double do_bet(double cur_max_bet,int con,card * cards[],double bank);
        double check(double need);
        void get_bank(double prize);
        void reset();
        void throwcards();              //throw cards!
        void getcards();                //get cards!
        double play(double cur_max_bet,int con,card * cards[],double bank);
        double raw(int con, card * cards[],bool flag);
        void set_alive(){is_alive=true;}
        int max_nom();
        int bestcomb(card cards[]);
        player();
        ~player();
private:
        card * onhand;
        double raise(double tbet,short int mod);
        int toflash(card * cards[],int con);          //number of outs to flash
        int tostraight(card * cards[], int con);  //return 0 if you have strait, -2 if straight on table, -1 if no chance to get, and nu
outs otherwise
        int samekind(card * cards[],int con,int &current_comb); /*return number of outs to next comb & set current comb to c
        of combination that we have */
        double solt();
};

player::player()
{
        is_alive=false;
        in_game=true;
        is_all_in=false;
        bet=0;
        cash=1000;
        onhand=new card[2];
}
```

```
player::~player()
{
	delete [] onhand;
}
void player::throwcards()
{
	delete [] onhand;
	onhand=nullptr;
	in_game=false;
}
void player::getcards()
{
	onhand= new card[2];
}
void player::reset()
{
	bet=0;
	throwcards();
	is_all_in=false;
	in_game=true;
	getcards();
}
void player::get_bank(double prize)
{
	cash+=prize;
}
double player::do_bet(double cur_max_bet,int con,card * cards[],double bank) //Please write it!
{
	if (!is_alive)
	{
		double brain=raw(con,cards,false)+solt();
		int mod=(int) brain/20;
		return raise(cur_max_bet-bet,mod);
	}
	else
	{
		double code=play(cur_max_bet,con,cards,bank);
		if (code<0)
		{
			throwcards();
			return 0;
		}
		if (code==0) return check(cur_max_bet-bet);
		if (code>0)  return check(cur_max_bet+code-bet);
	}
}
double player::check(double need)
{
	if (cash<=need)
	{
		is_all_in=true;
		bet+=cash;
		double tmp=cash;
		cash=0;
		return tmp;
	}
	else
	{
		bet+=need;
		cash-=need;
		return need;
	}
}
double player::raise(double tbet,short int mod)
{
	if(cash<=tbet*2 && mod>2) mod--;
```

```
        if(mod<2)
        {
                throwcards();
                return 0;
        }
        if(mod>4) return check(tbet*2);
        switch (mod)
        {
        case 2:
                return check(tbet);
        case 3:
                return check(tbet+10);
        case 4:
                return check(tbet*1.5);
        }
}
double player::raw(int con, card * cards[],bool flag)
{
        if(con==0)
        {
                if (onhand[0].getnominal()==onhand[1].getnominal()){return 70;}
                if (onhand[0].getsuit()==onhand[1].getsuit()){return 50;}
                if (std::abs(onhand[0].getnominal() - onhand[1].getnominal())<3)
                {
                        if(std::abs(onhand[0].getnominal() - onhand[1].getnominal())<2){return 40;}
                        return 20;
                }
                else
                {
                        return 10;
                }
        }
        int current comb=0;
        int nom code=samekind(cards,con, current_comb);
        int flash code,straight code;
        flash code=toflash(cards,con);
        straight code=tostraight(cards,con);
        if(!flash code && !straight code) current_comb=8;
        if(flag) return current_comb;
        if(current_comb<8)
        {
                if(!straight code) current comb=4;
                if(!flash_code) current_comb=5;
        }
        if(current comb>3) return 90;
        if(current comb>5) return 100;
        if(current comb==2 || current comb==3) return nom_code*2*(5-con)+20;
        if(flash code>0) return flash code*2*(5-con)+30;
        if(straight code>0) return straight_code*2*(5-con)+25;
        return nom_code*(5-con);
}
int player::toflash(card * cards[],int con)
{
        int suits[4];
        int max=0;
        for (int i = 0; i < con; i++)
        {
                switch (cards[i]->getsuit())
                {
                case 1: suits[0]++; break;
                case 2: suits[1]++; break;
                case 3: suits[2]++; break;
                case 4: suits[3]++; break;
                }
        }
        for (int i = 0; i < 4; i++){if(suits[i]==5) return -2;}        //if flash on table return negative
```

```
        for (int i = 0; i < 2; i++){suits[(onhand[i].getsuit())-1]++;}
        for (int i = 0; i < 4; i++){if (suits[i]>max){max=suits[i];}}
        if(max>4){return 0;}                                    //flash olready
        if(max<4){return -1;}                                   //to little chance
        return 13-max;                                          //number of outs
}
int player::tostraight(card * cards[], int con)
{
        int * nominals=new int[con+2];
        int * nominalsT=new int[con+2];
        for (int i = 0; i < con; i++)
        {
                nominals[2+i]=cards[i]->getnominal();
                nominalsT[2+i]=cards[i]->getnominal();
        }
        if(con==5)
        {
                for(int i=2;i<con+2;i++)
                {
                        if (nominalsT[i]==14) nominalsT[i]=1;
                }
                std::sort(nominals+2,nominals+con+2);
                std::sort(nominalsT+2,nominalsT+con+2);
                int maxseq=1;
                for(int i=3;i<con+2;i++)
                {
                        if(nominals[i]==nominals[i-1]+1) maxseq++;
                }
                if(maxseq==5)
                {
                        delete [] nominals;
                        delete [] nominalsT;
                        return -2;                              //straight is on table
                }
        }
        nominalsT[0]=nominals[0]=onhand[0].getnominal();
        nominalsT[1]=nominals[1]=onhand[1].getnominal();
        for(int i=0;i<2;i++)
        {
                if (nominalsT[i]==14) nominalsT[i]=1;
        }
        std::sort(nominals,nominals+con+2);
        std::sort(nominalsT,nominalsT+con+2);
        int maxseqence[3]={0,0,0},maxTseqence[3]={0,0,0};
        int counter=1,counterT=1;
        int s[3]={-1,-1,-1},sT[3]={-1,-1,-1};
        int offset=0,offseT=0;
        for(int i=1;i<con+2;i++)                        //find seqences
        {
                if(nominals[i]==nominals[i-1]+1) counter++;
                if(nominals[i]!=nominals[i-1]+1 || i==con+1)
                {
                        if (counter>1)
                        {
                                maxseqence[offset]=counter;
                                s[offset]=i-counter;
                                offset++;
                        }
                        counter=1;
                }
                if(nominalsT[i]==nominalsT[i-1]+1) counterT++;
                if(nominalsT[i]!=nominalsT[i-1]+1 || i==con+1)
                {
                        if (counterT>1)
                        {
                                maxTseqence[offseT]=counterT;
```

```
                        sT[offseT]=i-counterT;
                        offseT++;
                    }
                    counterT=1;
                }
            }
            if(*std::max_element(maxseqence,maxseqence+2)<2 && *std::max_element(maxTseqence,maxTseqence+2))
            {
                delete [] nominals;
                delete [] nominalsT;
                return -1;
            }
            if(*std::max_element(maxseqence,maxseqence+2)==5 || *std::max_element(maxTseqence,maxTseqence+2)==5)
            {
                delete [] nominals;
                delete [] nominalsT;
                return 0;
            }
            if(*std::max_element(maxseqence,maxseqence+2)==4 && *std::max_element(maxTseqence,maxTseqence+2)==4
            {
                delete [] nominals;
                delete [] nominalsT;
                return 8;  //2-sided straight
            }
            else
            {
                if(*std::max_element(maxseqence,maxseqence+2)==4 || *std::max_element(maxTseqence,maxTseqence+2)
                {
                    delete [] nominals;
                    delete [] nominalsT;
                    return 4;
                }
            }
            // 4-elemen sequences done!
            for (int i = 0; i < 3; i++)
            {
                if(maxseqence[i]==3)
                {
                    //3 at begin
                    if(s[i]==0 && nominals[s[i]+maxseqence[i]]==nominals[s[i]+maxseqence[i]+1]-2)
                    {
                        delete [] nominals;
                        delete [] nominalsT;
                        return 4;
                    }
                    //3 at end
                    else if(s[i]+maxseqence[i]==con+1 && nominals[s[i]]==nominals[s[i]-1]+2)
                    {
                        delete [] nominals;
                        delete [] nominalsT;
                        return 4;
                    }
                    //3 somwere else
                    else if(nominals[s[i]-1]==nominals[s[i]]-2 ||
nominals[s[i]+maxseqence[i]]==nominals[s[i]+maxseqence[i]+1]-2)
                    {
                        delete [] nominals;
                        delete [] nominalsT;
                        return 4;
                    }
                }
                if(maxTseqence[i]==3)
                {
                    if(sT[i]==0 && nominalsT[sT[i]+maxTseqence[i]]==nominalsT[s[i]+maxTseqence[i]+1]-2)
                    {
                        delete [] nominals;
```

```
                        delete [] nominalsT;
                        return 4;
                }
                else if(sT[i]+maxTseqence[i]==con+1 && nominalsT[s[i]]==nominalsT[sT[i]-1]+2)
                {
                        delete [] nominals;
                        delete [] nominalsT;
                        return 4;
                }
                else if(nominalsT[sT[i]-1]==nominalsT[sT[i]]-2 ||
nominalsT[sT[i]+maxTseqence[i]]==nominalsT[sT[i]+maxTseqence[i]+1]-2)
                {
                        delete [] nominals;
                        delete [] nominalsT;
                        return 4;
                }
        }
    }
    //3-element seqences done!
    for (int i = 0; i < offset; i++)
    {
        if(maxseqence[i]==2 && offset!=i+1)
        {
                if(nominals[s[i]+2]==nominals[s[i+1]]-2)
                {
                        delete [] nominals;
                        delete [] nominalsT;
                        return 4;
                }
        }
    }
    for (int i = 0; i < offseT; i++)
    {
        if(maxTseqence[i]==2 && offseT!=i+1)
        {
                if(nominalsT[sT[i]+2]==nominalsT[sT[i+1]]-2)
                {
                        delete [] nominals;
                        delete [] nominalsT;
                        return 4;
                }
        }
    }
    delete [] nominals;
    delete [] nominalsT;
    return -1;
}
int player::samekind(card * cards[],int con,int &current_comb)
{
    int nominals[13];
    int desk_combination;
    int count_pairs=0,count_threes=0,count_fours=0;
    int player_counts[]={0,0,0};
    current_comb=0;
    for (int i = 0; i < con; i++)
    {
        nominals[(cards[i]->getnominal())-2]++;
    }
    for (int i = 0; i < 13; i++)
    {
        if(nominals[i]==2) count_pairs++;
        if(nominals[i]==3) count_threes++;
        if(nominals[i]==4) count_fours++;
    }

    if(count_pairs) desk_combination=1;
```

```
        if(count pairs>1) desk combination=2;
        if(count threes) desk combination=3;
        if(count threes && count pairs) desk_combination=4;
        if(count_fours) desk_combination=5;

        for (int i = 0; i < 2; i++)
        {
                nominals[(onhand[i].getnominal())-2]++;
        }
        if(*std::max_element(nominals,nominals+12)<2)
        {
                current comb=0;
                return 1;
        }

        for (int i = 0; i < 13; i++)
        {
                if(nominals[i]==2) player counts[0]++;
                if(nominals[i]==3) player counts[1]++;
                if(nominals[i]==4) player_counts[2]++;
        }

        if(player counts[0]>count pairs) current comb=1;
        if(player counts[0]>count pairs && player counts[0]>1) current_comb=2;
        if(player counts[1]>count threes) current comb=3;
        if((player counts[0] && player_counts[1]) && (player_counts[0]>count_pairs || player_counts[1]>count_threes) )
current_comb=6;
        if(player_counts[2]>count_fours) current_comb=7;

        if(current comb==0) return 6;
        if(current comb==1) return con*3+2;
        if(current comb==2) return 4;
        if(current comb==3) return con;
        if(current_comb==4) return 1;

}
double player::solt()
{
        errno_t err;
        unsigned int rawrand;
        err=rand s(&rawrand);
        double dice=(double) ((double)rawrand/((double) UINT_MAX + 1)*40)-20;
        return dice;
}
double player::play(double cur_max_bet,int con,card * cards[],double bank)
{
        system("cls");
        std::cout<<"Yor cash="<< std::setw(7) <<cash<<std::setw(40)<<' '<<"Bank="<<std::setw(8)<<bank<<std::endl
        std::cout<<"Current max bet is"<<std::setw(7)<<cur_max_bet;
        for (int i = 0; i < 5; i++)
        {
                std::cout<<std::endl;
        }
        std::cout<<std::setw(30)<<' '<<"Cards on table"<<std::endl;
        std::cout<<std::setw(15)<<' ';
        for (int i = 0; i < con; i++)
        {
                std::cout<<'|';
                cards[i]->printcard();
                std::cout<<'|'<<' ';
        }
        for (int i = 0; i < 5; i++)
        {
                std::cout<<std::endl;
        }
```

```
	std::cout<<std::setw(30)<<' '<<"Your cards:"<<std::endl<<std::endl;
	std::cout<<std::setw(30)<<' ';
	for (int i = 0; i < 2; i++)
	{
		std::cout<<'|';
		onhand[i].printcard();
		std::cout<<'|'<<' ';
	}
	std::cout<<std::endl<<std::endl<<"Enter your bet: type -1, if you whant to throw cards;"<<std::endl<<"0 if you wha
otherwise enter your raise";
	double code;
	std::cout<<std::endl<<"Your choise:";
	std::cin>>code;
	return code;
}
int player::max_nom()
{
	if(onhand[0].getnominal()>onhand[1].getnominal()) return onhand[0].getnominal();
	else return onhand[1].getnominal();
}
int player::bestcomb(card cards[])
{
	std::vector<int> combinations(20);
	for (int i = 0; i < 3; i++)
	{
		for (int j = i+1; j < 4; j++)
		{
			for (int k = j+1; k < 5; k++)
			{
				combinations.push_back(combination(&onhand[0],&onhand[1],&cards[i],&cards[j],&cards[k]));
			}
		}
	}
	for (int i = 0; i < 2; i++)
	{
		for (int j = 0; j < 2; j++)
		{
			for (int k = j+1; k < 3; k++)
			{
				for (int l = k+1; l < 4; l++)
				{
					for (int m = l+1; l < 5; m++)
					{

combinations.push_back(combination(&onhand[i],&cards[j],&cards[k],&cards[l],&cards[m]));
					}
				}
			}
		}
	}
	if(!(*std::max_element(combinations.begin(),combinations.end())))
	{
		return (std::max(&onhand[0],&onhand[1],snominals))->getnominal();
	}
	else
	{
		return *(std::max_element(combinations.begin(),combinations.end()));
	}
}
```

party.h

```
#pragma once
class party
{
public:
```

```
        party(int number);
        ~party();
        player* next_in_game();          //return pointer to next in game player
        player * nn_allin();             //return pointer to next not "all in" player
        int alive_in_party();            //return number of alive players in game
        bool bets_equality();            //return true if bets are equal
        void clean_from_poor();          //remove from party players with 0 cash
        void reset();
        player * next();                 //moves playerptr to nex player in party
        player * next(player * man);
        player * playerptr;
        int players_in_party;
private:
        player * tparty;
        int current_player_number;
};
party::party(int number)
{
        tparty= new player[number];
        playerptr=tparty;
        players_in_party=number;
        current_player_number=0;
        tparty[0].is_alive=true;
}
party::~party()
{
        delete [] tparty;
}
player * party::next()
{
        if(current_player_number<players_in_party-1)
                {
                        current_player_number++;
                        playerptr++;
                }
                else
                {
                        current_player_number=0;
                        playerptr=tparty;
                }
        return playerptr;
}
player* party::next_in_game()
{
        int i=0;
        do
        {
                i++;
                next();
                if(i==players_in_party) return nullptr;
        } while (!(playerptr->in_game));
        return playerptr;
}
player * party::nn_allin()
{
        int detector=current_player_number;
        do
        {
                next_in_game();
                if (current_player_number==detector)
                {
                        return nullptr;
                }
        } while (playerptr->is_all_in);
        return playerptr;
}
```

```
int party::alive_in_party()
{
	int souls=0;
	for (int i = 0; i < players_in_party; i++)
	{
		if(playerptr->is_alive) souls++;
		next();
	}
	next();
	return souls;
}
bool party::bets_equality()
{
	double Tbet=nn_allin()->bet;
	bool flag(true);
	for (int i = 0; i < players_in_party; i++)
	{
		if (playerptr->in_game && playerptr->bet!=Tbet)
		{
			flag=false;
		}
		next();
	}
	next();
	return flag;
}
void party::clean_from_poor()
{
	int num=0;
	for (int i = 0; i < players_in_party; i++)
	{
		if(playerptr->cash) num++;
		playerptr->throwcards();
		next();
	}
	player * tptr=new player[num];
	num=0;
	for (int i = 0; i < players_in_party; i++)
	{
		if(playerptr->cash)
		{
			tptr[num]=*playerptr;
			num++;
		}
		next();
	}
	delete[] tparty;
	tparty=tptr;
	players_in_party=num;
}
void party::reset()
{
	playerptr=tparty;
	current_player_number=0;
}
player * party::next(player * man)
{
	while (playerptr!=man) next();
	return next();
}
#pragma once
class desk
{
public:
	desk(int number);
	~desk(){}
```

```
        bool opencard(); //return 0 if succesful
        bool set_max_bet(double min_bet); //return 0 if succesful
        void renew_cards();
        bool get_bets();
private:

        card cards_on_table[5];
        card * for_players[5];
        int visible_cards;
        double minimal_bet;
        double cur_max_bet;
        party  players;
        player * button;
        double bank;      //It is cash on table
        int round;        //0 if preflop, 1 if flop, 2 if turn, 3 if reaver
};
desk::desk(int number):cards_on_table(),players(number) //initialaize massive of cards on table
{
        visible_cards=0;
        minimal_bet=10;
        cur_max_bet=10;
        bank=0;
        round=0;
        button=players.playerptr;
        for (int i = 0; i < 5; i++)
        {
                for_players[i]=&cards_on_table[i];
        }
}
bool desk::opencard()
{
        if(visible_cards==0) {visible_cards=3; return 0;}
        else if(visible_cards<5) {visible_cards++; return 0;}
        else return 1;
}
bool desk::set_max_bet(double newbet)
{
        if(newbet<=0 || newbet<cur_max_bet) return true;
        else
        {
                cur_max_bet=newbet;

                return 0;
        }
}
bool desk::get_bets()
{
        int delta=0;
        delta=button->check(minimal_bet);
        bank+=delta;
        set_max_bet(delta);
        button=players.next(button);
        delta=button->check(minimal_bet*2);
        bank+=delta;
        set_max_bet(delta);
        bool ff=false;
        do
        {
                do
                {
                        if (players.nn_allin()!=nullptr)
                        {
                                bank+=players.playerptr->do_bet(cur_max_bet,visible_cards,for_players,bank);
                                set_max_bet(players.playerptr->bet);
                        }
```

```
                else
                {
                        ff=true;
                        break;
                }
        } while (players.bets_equality()!=true);
        if(ff==true) break;
} while (!opencard());
bool tie=false;
int winnercomb=0;
players.reset();
player * winner=nullptr;
for (int i = 0; i < players.players_in_party; i++)
{
        if (players.playerptr->bestcomb(cards_on_table)>winnercomb)
        {
                winnercomb=players.playerptr->bestcomb(cards_on_table);
                winner=players.playerptr;
        }
        players.next();
}
for (int i = 0; i < players.players_in_party; i++)
{
        if (players.playerptr->bestcomb(cards_on_table)==winnercomb && players.playerptr!=winner)
        {
                tie=true;
        }
}
if (tie)
{
        int number_of_winners=0;
        std::vector<player*> winners(2);
        for (int i = 0; i < players.players_in_party; i++)
        {
                if (players.playerptr->bestcomb(cards_on_table)==winnercomb)
                {
                        number_of_winners++;
                        winners.push_back(players.playerptr);
                }
        }
        for each (player* iter in winners)
        {
                iter->get_bank(bank/number_of_winners);
        }
}
else
{
        winner->get_bank(bank);
}
bank=0;
players.clean_from_poor();
if (!players.alive_in_party())
{
        std::cout<<"End Game";
        return true;
}
std::cout<<"whant to play more? y/n"<<std::endl;
char tmp;
std::cin>>tmp;
if(tmp='y')
{
        for (int i = 0; i < players.players_in_party; i++)
        {
                players.playerptr->reset();
                players.next();
        }
```

```
                round=0;
                renew_cards();
                for (int i = 0; i < 5; i++)
                {
                        for_players[i]=&cards_on_table[i];
                }
                return false;
        }
        else return true;

}
void desk::renew_cards()
{
        for (int i = 0; i < 5; i++)
                {
                        (cards_on_table[i]).~card();
                }
        for (int i = 0; i < 5; i++)
                {
                        cards_on_table[i].card::card();
                }
}
```

holdem.cpp

```
#include "stdafx.h"
#include "card.h"
#include "player.h"
#include "party.h"
#include "desk.h"


int _tmain(int argc, _TCHAR* argv[])
{
        try
        {
                int number;
                std::cout<<"Please enter number of players"<<std::endl<<"Number of players=";
                std::cin>>number;
                if (number==0) return 0;
                desk newdesk(number);
                bool flag;
                do
                {
                        flag=newdesk.get_bets();
                } while (!flag);
        }
        catch(const std::runtime_error& err)
        {
                std::cout<<err.what();
        }
}
```

Список использованных материалов

При написании данной работы использовались материалы ресурсов:
http://msdn.microsoft.com
http://stackoverflow.com
http://habrahabr.ru/
http://www.holdemworld.ru