
# ENGENHARIA DE COMPUTAÇÃO


| | |
|---|---|
| Alcides Gomes Beato Neto | 19060987 |
| Henrique Sartori Siqueira | 19240472 |
| Thiago Hideki Honda | 19310515 |


## Projeto BlackJack

**CAMPINAS**
**2019**

**Curso de Engenharia de Computação**

**07527P – Algorítmos e Programação de Computadores B**

Alcides Gomes Beato Neto 19060987
Henrique Sartori Siqueira 19240472
Thiago Hideki Honda 19310515

# Projeto BlackJack

**Trabalho em grupo apresentado para a disciplina de Algorítmos e Programação de Computadores B, solicitado pela professora Lúcia Guimarães para desenvolver as habilidades que aprendemos durante o semestre e buscar o conhecimento do que está além do ensinado em sala de aula para nos sobressairmos, futuramente, no mercado de trabalho.**

**CAMPINAS**

**2019**

# SUMÁRIO



## 1. GERAL

O jogo blackjack ou 21 foi programado com as respectivas variáveis, funções e exibições em inglês e comentado em português. A tela de início é auto explicativa, isto é, o programa espera que o usuário pressione um botão do teclado para iniciar o jogo. Todavia, antes da inicialização deve-se definir o número de jogadores e a quantidade de baralhos, logo após tais definições, há a coleta do nome de cada jogador e a respectiva quantia de dinheiro inicial a ser apostada nas rodadas. O jogo se inicia depois do usuário definir o valor que quer apostar (não podendo ultrapassar 30% do valor inicial), há a distribuição das cartas pelo banqueiro e, caso não haja nenhum jogador com a soma 21, o banqueiro distribui mais cartas caso o jogador desejar. No final da rodada, há as estatísticas prévias dos jogadores, sendo a soma das cartas de todos (incluindo o banqueiro) e o valor que cada jogador ganhou ou perdeu na rodada, isso é seguido da verificação se o usuário quer jogar outra rodada, caso seja o fim de jogo, haverá a impressão das estatísticas dos jogadores (salvo em um arquivo .txt inclusive), com a quantidade de vitórias, empates, derrotas e a soma dos mesmos, em conjunto com o montante final em que o jogador teve como retorno após ter apostado. Outrossim, o valor de ganhos ou perdas do banqueiro aparece conjuntamente com as informações dos jogadores.

Além do conteúdo visto ao decorrer do semestre, também foram implementados tópicos adicionais como a geração de números aleatórios com a função *srand*, a alocação dinâmica para a quantidade de jogadores e a utilização de um operador ternário (if ? then : else;). Também foi feita a tentativa de implementação da biblioteca allegro para a parte gráfica, mas devido a erros de código e entendimento de como ela funciona, não foi implementada.

## 2. VARIÁVEIS

Há duas variáveis cruciais do tipo estrutura (struct) para que o programa funcione bem e de forma organizada, ou seja, caso sejam realizadas modificações futuras, adicionando mais opções aos jogadores, como a opção *split* (dividir), que possibilita o jogador no meio da partida obter uma segunda mão de cartas, consequentemente realizando outra aposta, ou a opção *double* que possibilita ao usuário dobrar o valor apostado.

A estrutura relacionada ao jogador (*player*) possui duas variáveis do tipo caracter para o armazenamento do nome do respectivo jogador e dos valores das cartas que o mesmo possui. Três variáveis do tipo real para o respectivo armazenamento da quantia de dinheiro (*money*), o valor de 30% da quantia monetária declarada pelo jogador após a coleta do nome do próprio (*startbet*) e o valor apostado na respectiva rodada (*bet*). Nove

variáveis inteiras, sendo quatro para controle estatístico - com vitórias, derrotas, empates e o total de rounds jogados - uma sendo utilizada para a soma do valor das cartas que o jogador possui (*sum*) e três utilizadas como variáveis booleanas, sendo adotados os valores de 0 ou 1 para tais, isto é, uma realiza o status do Às (*Ace*) valendo 11, outra com o status de bust (estouro) do jogador caso o mesmo obtenha uma soma superior a 21, uma para contagem da quantidade de cartas na mão do jogador (utilizada no vetor de valores das cartas) e por fim, uma que indica se o jogador atingiu a soma de 21.

Essa estrutura é declarada como um ponteiro para que o usuário defina a quantidade de jogadores que o jogo terá, sendo utilizada uma alocação dinâmica (*malloc*) para transformar a estrutura em um vetor de estruturas.

A variável *pla* é a responsável por definir o número de jogadores, havendo uma verificação prévia para que a mesma permaneça no intervalo esperado (1 a 4 jogadores).

Para a estrutura do banqueiro (*dealer*), foram utilizadas apenas cinco variáveis, uma do tipo real para contabilizar o total de ganhos/perdas da mesa (*bank*), quatro do tipo inteiro para status de *bust* (estouro), blackjack (soma resultando em 21), Às valendo 11 e o total do valor de cartas na mão do banqueiro.

Para que não houvesse repetições dos valores do baralho, por exemplo uma matriz de 2 a 4 linhas e 52 colunas com cada posição representada pelo valor de uma carta, foi utilizado apenas um vetor de caracteres com os diferentes valores que uma carta pode assumir, sendo declarado também um vetor de inteiros com 13 posições para contagem da quantidade de vezes que certa carta apareceu na rodada.

Outras variáveis inteiras também foram utilizadas para armazenar as decisões tomadas pelo jogador e as verificações realizadas para determinar quem perde, empata ou ganha.

Ademais, foram utilizadas outras duas cadeias de caracteres para manipulação do relatório de saída utilizando arquivo.

## 3. FUNÇÕES

A primeira função utilizada é a de coletar as informações do jogador (*getplayerinfo*), requerendo o nome e a quantia inicial para as futuras apostas, sendo realizada a verificação para que o valor seja maior que 50. Também há a inicialização com valor 0 de

todas as variáveis de controle, exceto uma, que se refere ao valor máximo que o jogador pode apostar (sendo 30% da quantia inicial).

Outra função utilizada é a de coletar as apostas dos jogadores (*placebet*), sendo verificado se o valor inserido está no intervalo de 2 até 30% do valor monetário inicial, e também, subtraindo do montante o valor escolhido.

A função que seleciona uma carta aleatória (*givecard*) é definida pela função de gerar um número aleatório de 0 a 12, o intervalo refere-se respectivamente às posições do vetor *deck* (baralho), totalizando as 13 diferentes cartas, porém para a verificação da quantidade das mesmas, o vetor *cardcheck* verifica se a mesma carta já apareceu mais vezes que o limite, isto é, a quantidade de baralhos (*de*) vezes quatro. Caso não tenha chego ao limite, há o incremento no contador para futuras verificações. Outra função que relaciona as cartas é a função cards, ela faz a parte gráfica das cartas, exibe as cartas de cada jogador por meio de for, switch e case.

A soma das cartas é realizada através de duas diferentes funções sendo uma para os jogadores e outra para o banqueiro, com a respectiva verificação do valor do às podendo valer 11, de acordo com a possibilidade da soma não ultrapassar 21, ou 1, também fazendo a verificação da soma, caso o jogador estoure, ativando o status de *bust* ou ativando o status de blackjack (*bj*) caso a soma resulte em 21. A diferença entre essas duas funções está no gerenciamento das apostas e das estatísticas de partida (derrotas e total respectivamente nesta situação), assim, o jogador possui mais informações a serem gerenciadas do que o banqueiro, este que só possui a quantia perdida ou ganha nas apostas.

Há também uma função -*blackjack*- que indica se algum jogador teve um blackjack com apenas duas cartas distribuídas logo de início.

Uma outra função (*bigger*) é responsável por indicar a maior somatória na mão de um jogador no rodada.

Como há diversas possibilidades de verificação, ou seja, o banqueiro pode estourar com a obtenção de uma carta, pode ganhar por somar 21, pode perder para dar a vitória a outro jogador e assim por diante. Para algumas verificações o grupo notou que houve a repetição dos mesmos trechos de código, mudando apenas a condição para execução dos mesmos, assim, criou-se a função *management* (gerenciamento) em que na teoria o banqueiro gerencia o jogo com as condições contidas nas regras do jogo, lidando com o dinheiro das apostas e as variáveis estatísticas do placar.

Caso haja a escolha de jogar novamente um próximo round, a função *setzero* é responsável por zerar as variáveis de status, tanto dos jogadores quanto do banqueiro, e de controle de cartas, além da variável utilizada para a verificação de um possível empate (*winplayer*).

# 4. VERIFICAÇÕES

Como há diversas possibilidades de resultados, foram implementadas diversas condições para cada caso específico no jogo. Primeiramente, há uma verificação caso o banqueiro tenha feito blackjack, havendo o empate caso algum outro jogador também tenha feito 21 com apenas duas cartas. Logo após, há a conferência se há algum jogador que conseguiu atingir a soma de 21 com apenas duas cartas, se há, excepcionalmente, mais de um jogador, a vitória é contada para todos, não havendo empate, apenas derrotas e vitórias para este caso em específico, apesar de ser extremamente raro, mas não impossível.

Enquanto os casos anteriormente descritos não acontecerem nas rodadas, o banqueiro distribui mais cartas aos jogadores que as solicitam realizando a devida conferência se o jogador ultrapassa a soma de 21 na própria função de somatória das cartas, mostrando uma mensagem na tela caso isso aconteça. Depois, há a verificação da somatória do banqueiro, caso ele tenha feito até 17 pontos, ele tira mais cartas até que a soma ultrapasse esse número, caso a soma resulte em 17 há a verificação se há um às (valendo 11), se houver ele coleta mais cartas para si, senão ele continua o jogo comparando a somatória com os outros jogadores.

Caso haja o estouro da mesa, após obter mais cartas, os jogadores que ainda não estouraram ainda permanecem no jogo tendo como vencedor ou empate aquele/aqueles que obteve/obtiveram a maior somatória. O mesmo acontece caso a mesa obtenha um blackjack, empatando com os jogadores que também obtiveram a soma de 21 ou acrescentando um às derrotas dos jogadores sem blackjack.

Se, por ventura, nenhuma das condições anteriores forem satisfeitas, há a verificação por parte de uma função, em que esta se repete três vezes em condições diferentes, realizando a mesma verificação, ou seja, os jogadores com somatória menor perdem, caso haja igualdade de valores entre a mesa e um jogador há empate, caso haja jogadores com soma superior a da mesa há uma verificação posterior para determinar se há um vencedor ou empate e, por fim, há a possibilidade da mesa ter a maior somatória, fazendo com que os jogadores percam a rodada.

Em relação às apostas, em caso de vitória os jogadores recebem o dobro da aposta que realizaram, exceto no caso de blackjack instantâneo ou de empate por blackjack, pois os jogadores recebem três meios do que apostaram. Para os empates normais, o valor apostado é estornado. Além disso, para os perdedores, há a perda integral da aposta.

## 5. REFERÊNCIAS BIBLIOGRÁFICAS


- Disponível em: <https://pt.stackoverflow.com/questions/41692/aloca%C3%A7%C3%A3o-din%C3%A2mica-para-struct>. Acesso em 20 novembro 2019.

- Disponível em: <https://sites.google.com/site/puccprofluciaguimaraes/home/engenharia-de-computacao/algoritmos-e-programacao-de-computadores-b>. Acesso em 22 outubro 2019.

- Disponível em: <https://www.sololearn.com/Play/C>. Acesso em 20 novembro 2019.

- Disponível em: <https://www.sololearn.com/Discuss/1864628/random-numbers-in-c>. Acesso em 25 outubro 2019.

- Disponível em: <http://linguagemc.com.br/o-operador-ternario-em-c/>. Acesso em 20 novembro 2019.

- Disponível em: <http://patorjk.com/software/taag/#p=display&f=3D-ASCII&t=apc%20b>. Acesso em 23 novembro 2019.


## 6. ANEXOS

```
typedef struct players{
    char name[30],v[19];
    float money, startbet, bet;
    int wins, loses, ties, total, sum, A, bust, bj, cc;
}player;
```

*Imagem 1: Declaração da estrutura relacionada às informações dos jogadores.*

```
p = (player *) malloc(pla * sizeof(player));
```

*Imagem 2: Alocação dinâmica da estrutura jogadores.*

```
typedef struct table{
    int bust, sum, A, bj;
    float bank;
}dealer;
```

*Imagem 3: Declaração da estrutura relacionada às informações do banqueiro.*

```
char deck[]={"A234567891JQK\0"}
```

*Imagem 4: Declaração do vetor que representa os valores das cartas.*

```
do{
    n = (rand()%13);
}while(cardcheck[n] == de * 4);
cardcheck[n]++;
```

*Imagem 5: Geração e verificação de uma carta.*

```
case 'A':
    if((d->sum+11)>21){
        d->sum++;
        if(d->sum>21 && d->A == 1){
                d->sum -= 10;
                d->A--;
            }
            else{
                d->bust++;
            }
        }
    else{
        d->sum+=11;
        d->A++;
    }
    break;
```

*Imagem 6: Trecho da função que soma as cartas, para o caso de às.*

```
    for(int i=0; i < pla; i++){
        if(aux < p[i].sum && p[i].bust == 0)
            aux = p[i].sum;
    }

return aux;
```

*Imagem 7: Parte da função bigger, retornando o maior valor de soma.*

```
x==1 ? setzero(p,pla,cardcheck,&d,&winplayer),totalrounds++:totalrounds++;
```

*Imagem 8: Utilização de operador ternário para verificar se haverá mais uma rodada.*

```
        if(d.sum <= 17){
            if(((d.sum - 11) <= 6 && d.A == 1) || d.sum < 17){
                do{


                if(d.bust == 1){ // se a mesa faz bust
                else if(d.bj == 1){ // caso haja BJ para a mesa
                else if(d.bust == 0 && d.bj == 0){ // se não houver bust nem BJ para a mesa
            }// if - requesitos para mesa pegar +1 carta
            else if(d.sum == 17){
        } // mesa == 17 sem às || às + cartas > 17
        else{ // mesa > 17
    } // sem insta BJ de jogador
    else if(endg == 1){ // com blackjack instantâneo de jogador (se houver mais de um os dois ganham vitória)
}// if - sem blackjack instantâneo da mesa
else if(d.bj == 1){// caso haja BJ instantâneo para a mesa
```

*Imagem 9: Parte das condições que possibilitam que o jogo funcione.*

```
void cards(player p[], int j)
{
    for (int i = 0; i < p[j].cc; i++)
    {
        switch (p[j].v[i])
        {
            case 'A': printf("  ___"); break;

            case '2': printf("  ___"); break;

            case '3': printf("  ___"); break;

            case '4': printf("  ___"); break;

            case '5': printf("  ___"); break;

            case '6': printf("  ___"); break;
```

*Imagem 10: Parte da função que exibe as cartas de cada jogador*

```
#include <allegro.h> //biblioteca allegro
BITMAP* buffer,* capa; //declaração das variáveis
int main()
{ allegro_init(); //inicialização da biblioteca
   install_keyboard(); //instalação do teclado
   set_color_depth(32);
   set_window_title("Blackjack");
   set_gfx_mode(GFX_AUTODETECT_WINDOWED, 800, 600, 0, 0);
   buffer = create_bitmap(800, 600);
   capa = load_bitmap("imagem.bmp", NULL); // exibir a imagem
   while (!key[KEY_ESC]) // exibir a imagem até ser pressionado a tecla ESC
   {
       draw_sprite(buffer, imagem, 0, 0);
       draw_sprite(screen, buffer, imagem, 0, 0);
       clear(buffer);
   }
   destroy_bitmap(buffer); // parar a exibicação da imagem
   return 0; }
```

*Imagem 11: Parte gráfica do título em allegro*


*imagem 12: Título a ser implementado no allegro*