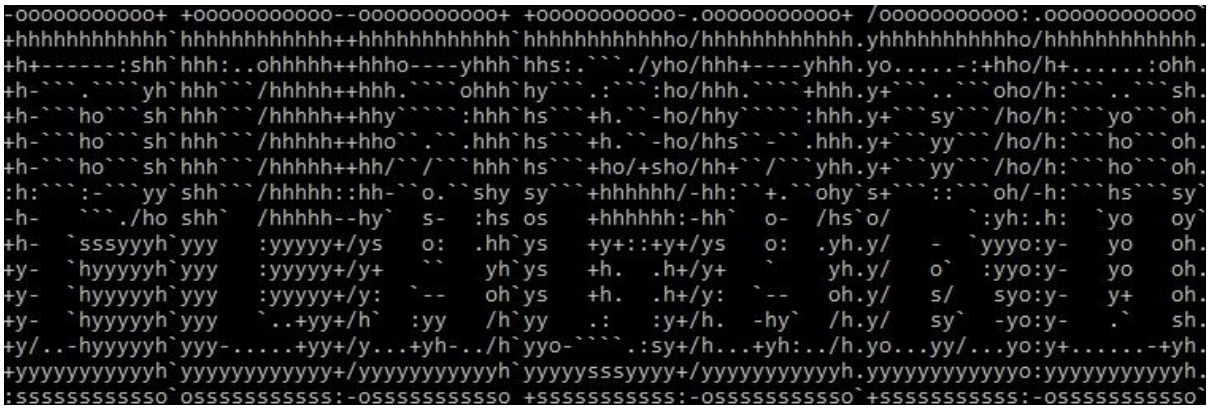
Relatório de Projecto

Curso: Engenharia e Desenvolvimento de Jogos Digitais
UC: Algoritmos e Estruturas de Dados

Ano Lectivo: 2016/2017



Docente: Duarte Duque

Elementos do grupo:

Tiago Mariano 13211

Diogo Portela 13236

Pedro Silva 14281

Introdução

Placard é um jogo de apostas desportivas gerida pelos Jogos SantaCasa, e tem como base odds atribuídas a resultados de eventos desportivos reais.

O objectivo deste projecto é criar um jogo similar ao Placard mas com simulação virtual de jogos, de forma a podermos fazer infinitos jogos e ter resultados de imediato.

Este relatório está estruturado de forma a explicar as partes mais complexas do código, facilitando a compreensão do mesmo, tanto para o utilizador como para aplicar melhorias ou revisão posterior..

Estratégia e Organização

Para a realização do trabalho em grupo usámos, para além do IDE Microsoft Visual Studio, o GitHub, para manter as alterações feitas no código por cada um de nós, actualizada. Para além disso, usamos também o GitKraken de forma a podermos ter vários stages de desenvolvimento guardados, podendo regressar a um stage antigo a qualquer momento.

Explicação do código

```c
#define maximoClubes 20
#define maximoJogos 380
```

```c
typedef struct
{
    char nome[60];
    float ataque_casa;
    float defesa_casa;
    float ataque_fora;
    float defesa_fora;
}clube;
```

```c
typedef struct
{
    clube *casa;
    clube *visitante;
    float PoissonCasa[120];
    float PoissonFora[120];
    float oddCasa, oddEmpate, oddVisitante;
    int resultado[2];
}jogo;
```

```c
typedef struct
{
    char nome[60];
    clube listaClubes[maximoClubes];
    jogo listaJogos[maximoJogos];
    int listaClubesCount;
    int listaJogosCount;
    int maxpts;
    float mediapts_casa;
    float mediapts_fora;
}modalidade;
```

Para melhor organização deste projecto, optámos por criar uma rede de estruturas que se completam mutuamente, de forma a facilitar todo o desenvolvimento do mesmo. Desta forma, a grande maioria das funções criadas recebem apenas uma modalidade e através dela conseguem aceder a tudo o que está associado a esta, inclusivé os jogos criados, bem como os clubes, máximo de pontos dessa modalidade (hardcoded), etc…

```c
/*HEADER FILE COM AS FUNCOES NECESSARIAS PARA TODO O CODIGO DO PROJETO*/
void            Definicoes              (modalidade *mod, int *quantidade);
void            FicheiroLeData          (modalidade *mod, int *quantidadeMods);
void            GereJogo                (modalidade *mod, int *quantidade, int *saldo);
void            GerirSaldo              (int *saldo);
void            ListarTudo              (modalidade *mod, int *quantidade);
void            LimpaEcra               (void);
int             SeedAleatoria           (void);
```

Também no sentido de manter o código organizado, criámos dois ficheiros header, que ligam as funções criadas no ficheiro “PlacardFunctions.c” ao ficheiro “main.c”. Os dois ficheiros header são: “**PlacardFunctions.h**” e “**PlacardStructs.h**”, sendo que este último guarda as estruturas anteriormente referidas. Desta forma, usamos a função main apenas para organizar o código e testar as funções implementadas. Conseguimos, assim, “juntar” vinte e duas funções internas e através da recursividade, usar sete na main para organizar todo o código.

```c
void Poisson(modalidade *mod)
```

Esta função aplica a **Distribuição de Poisson**, forma escolhida por nós para calcular as probabilidades de cada jogo da forma mais realista possível.

“A **Distribuição de Poisson** é uma distribuição de probabilidade de variável aleatória discreta que expressa a probabilidade de uma série de eventos ocorrer num certo período de tempo se estes eventos ocorrem independentemente de quando ocorreu o último evento.” Segue-se uma explicação de porque faz sentido aplicar-se no nosso caso.

$$f(k;\lambda) = \frac{e^{-\lambda}\lambda^{k}}{k!},$$

Em que:

- ***e*** é base do logaritmo natural (e = 2.71828...);
- ***λ*** é um número real, igual ao número esperado de ocorrências que ocorrem num dado intervalo de tempo. Neste caso, o valor de pontos/golos esperados por parte de uma dada equipa ou jogador (se se tratar de ténis). Como é calculado será explicado mais à frente;
- ***k*** é uma ocorrência (k sendo um inteiro não negativo, k = 0, 1, 2, ...). Neste caso, k será a ocorrência “marcar k golos/pontos”.

```c
for (int i = 0; i < mod[0].listaJogosCount; i++)
{
    lambda_c = (*mod[0].listaJogos[i].casa).ataque_casa * (*mod[0].listaJogos[i].visitante).defesa_fora * mod[0].mediapts_casa;
    lambda_f = (*mod[0].listaJogos[i].visitante).ataque_fora * (*mod[0].listaJogos[i].casa).defesa_casa * mod[0].mediapts_fora;
```

Para cada jogo, calculamos o ***λ*** para a equipa da casa e para a equipa visitante.

O ***λ*** para a equipa da casa é calculada através da multiplicação entre a “capacidade de ataque”, em casa, dessa equipa e a “capacidade de defesa”, fora, da equipa visitante. Ambos os valores são multiplicados pela média de golos/pontos marcados em casa pelas equipas dessa modalidade, de forma a normalizar os valores e não chegarmos a valores exagerados de golos esperados.

Um exemplo:

BENFICA VS. SPORTING

Golos esperados do Benfica (***λ***casa) = Ataque do Benfica(c) * Defesa do Sporting(f) * média de golos(c)
λcasa = 1,35 * 0,90 * 1,10 = **1,34**
Golos esperados do Sporting (***λ***fora) = Ataque do Sporting(f) * Defesa do Benfica(c) * média de golos(f)
λfora = 1,05 * 0,85 * 1,01 = **0,90**

Como calcular a capacidade de ataque e defesa das equipas:

```
void CalculaAtteDef(modalidade *mod)
void CalculaMediaGolos(modalidade *mod, int modIndex, int *quantidade)
```

```
media = ((float)goloscasa / (float)jogoscasa);
mod[0].listaClubes[indexClube].ataque_casa = media / mod[0].mediapts_casa;
media = ((float)golosfora / (float)jogosfora);
mod[0].listaClubes[indexClube].ataque_fora = media / mod[0].mediapts_fora;
media = ((float)golossofridoscasa / (float)jogoscasa);
mod[0].listaClubes[indexClube].defesa_casa = media / mod[0].mediapts_fora;
media = ((float)golossofridosfora / (float)jogosfora);
mod[0].listaClubes[indexClube].defesa_fora = media / mod[0].mediapts_casa;
```

Após percorrer todos os jogos possíveis (guardados no ficheiro *-jogos.txt, em que * é um placeholder para o nome da modalidade), e guardar todos os golos marcados e sofridos em casa, bem como fora, de todas as equipas. Calculamos as capacidades de defesa e ataque de todas as equipas, como demonstrado no pedaço de código.

- mediapts_casa, mediapts_fora referem-se à média de pontos em casa e fora, respectivamente, de todas as equipas. A função é bastante simples e de fácil compreensão.

```
for (int j = 0; j <= mod[0].maxpts; j++)
{
    mod[0].listaJogos[i].PoissonCasa[j] = (exp(-lambda_c) * pow(lambda_c, j)) / factorial(j);
    normalizador_c[i] += mod[0].listaJogos[i].PoissonCasa[j];
    mod[0].listaJogos[i].PoissonFora[j] = (exp(-lambda_f) * pow(lambda_f, j)) / factorial(j);
    normalizador_f[i] += mod[0].listaJogos[i].PoissonFora[j];
```

De seguida, percorrendo todos os ***k*** possíveis, calcula, usando a fórmula da distribuição, a probabilidade de cada um deles (***k***´s) ocorrer.

No fim da execução, para cada jogo, ficamos com um array de floats que guarda a probabilidade da equipa da casa (PoissonCasa[120]) e da de fora (PoissonFora[120]) marcarem ***k*** golos/pontos cada.

Para melhor compreensão:

- PoissonCasa (PM0c, PM1c, PM2c, ...);
- PoissonFora (PM0f, PM1f, PM2f, ...).

Sendo:

- PM- Probabilidade de Marcar;
- c- equipa da casa;
- f- equipa visitante;
- 0, 1, 2, …- pontos/golos.

Como a distribuição de Poisson calcula a probabilidade de determinadas ocorrências quando elas tendem para +∞, então o somatório de todas as probabilidades nunca será 1, logo é necessário normalizar os valores:

```
for (int i = 0; i < mod[0].listaJogosCount; i++)
{
    for (int j = 0; j <= mod[0].maxpts; j++)
    {
        aux = mod[0].listaJogos[i].PoissonCasa[j] / normalizador_c[i];
        mod[0].listaJogos[i].PoissonCasa[j] = aux;
        aux = mod[0].listaJogos[i].PoissonFora[j] / normalizador_f[i];
        mod[0].listaJogos[i].PoissonFora[j] = aux;
    }
```

Os arrays de normalizadores guardam, para cada cada jogo, a soma das probabilidades. Por exemplo, para PM4c = 0,3 golos e normalizador 0,93, a nova PM4c seria (0,3 / 0,93)*1 = 0,323.

Como calcular as odds iniciais:

```
void CalculaOddsIniciais(modalidade *mod)
for (int i = 0; i <= mod[0].maxpts; i++)
{
    for (int j = 0; j <= mod[0].maxpts; j++)
    {
        if (i == j)
        {
            empate += mod[0].listaJogos[indexJogo].PoissonCasa[i] * mod[0].listaJogos[indexJogo].PoissonFora[j];
        }
        else if (i > j)
        {
            somaaux += mod[0].listaJogos[indexJogo].PoissonFora[j];
            vitoria += mod[0].listaJogos[indexJogo].PoissonCasa[i] * somaaux;
        }
        else
        {
            somaaux2 += mod[0].listaJogos[indexJogo].PoissonCasa[i];
            derrota += mod[0].listaJogos[indexJogo].PoissonFora[j] * somaaux2;
```

Primeiramente, calculamos a probabilidade do empate:

PoissonCasa (PM0c, PM1c, PM2c, ...) PoissonFora (PM0f, PM1f, PM2f, ...)

A probabilidade de empate será:

PM0c * PM0f + PM1c + PM1f + PM2c + PM2f

De seguida, para calcular a probabilidade de vitória da equipa da casa:

PoissonCasa (PM0c, PM1c, PM2c, ...) e PoissonFora (PM0f, PM1f, PM2f, ...)

A probabilidade de vitória será:

PM1c * PM0f + PM2c * (PM1f * PM0f) + PM3c * (PM2f * PM1f * PM0f) + …

Para calcular a probabilidade de derrota da equipa da casa (ou seja, a probabilidade de vitória da equipa visitante) basta fazer o inverso que o que fizemos para a equipa da casa.

Agora que temos as probabilidades, basta-nos transformá-las em odds para podermos apostar:

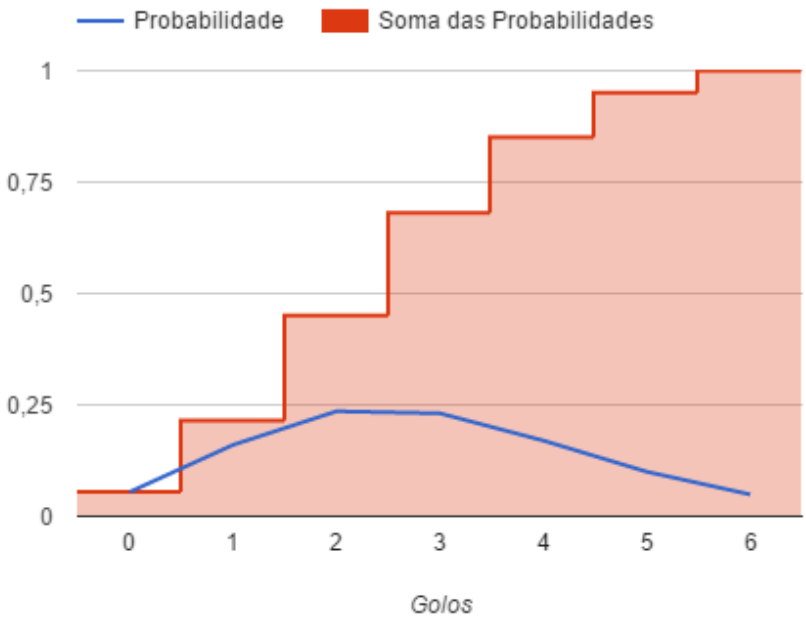
```
mod[0].listaJogos[indexJogo].oddCasa = 1 + (1 / vitoria);
mod[0].listaJogos[indexJogo].oddEmpate = 1 + (1 / empate);
mod[0].listaJogos[indexJogo].oddVisitante = 1 + (1 / derrota);
```

Para simular os resultados com base nas probabilidades calculadas vamos dar o seguinte exemplo retirado de uma base de dados gerada:

AROUCA VS. FCPORTO

Golos do Arouca:

PoissonCasa = {0,0544719994, 0,160211995, 0,235606000, 0,230985999, 0,169843003, 0,0999070033, 0,0489739999}




```
aux = rand() % 101;
aux /= 100;

int i = 0;
while (i <= mod[0].maxpts)
{
    somaaux += mod[0].listaJogos[indexJogo].PoissonCasa[i];
    if (aux < somaaux)
        return i;
    else
        i++;
```

Inicialmente geramos um **valor random no intervalo real [0,1]** e cruzamos com os valores que temos para a soma de probabilidades. Digamos que o valor que foi gerado foi **0,60**:

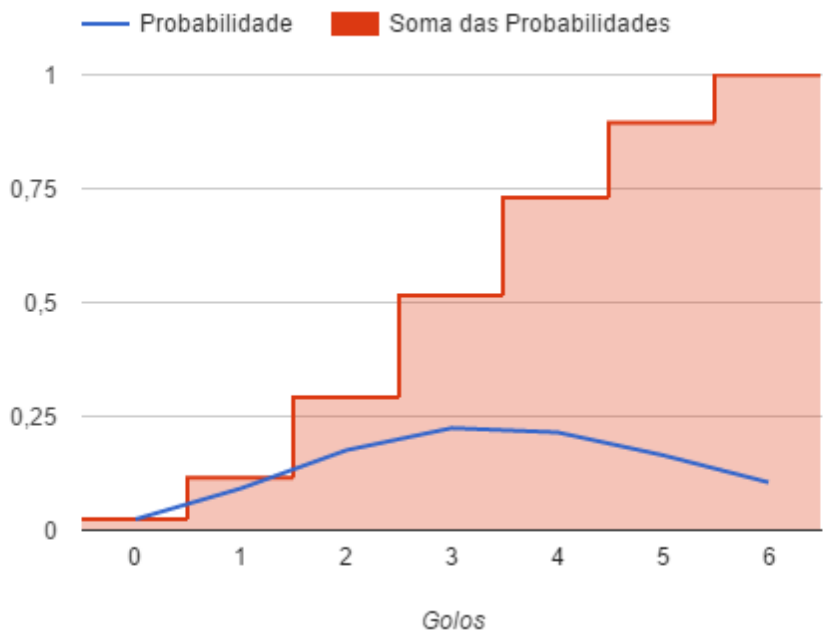
0,60 é menor que a probabilidade “equipa da casa marcar 0 golos”?
(PM0c = 0.0544719994)

- Não, logo, somamos um golo.

Se continuarmos com este raciocínio, e com auxílio do gráfico acima, chegamos à conclusão que **0,60** está **abaixo** do valor da seguinte soma: PM0c + PM1c + PM2c + PM3c mas **acima** do valor de PM0c + PM1c + PM2c, logo, podemos concluir que na nossa simulação, o **Arouca marcou 3 golos**.

Golos do FCPorto:

PoissonFora = {0,0239439998, 0,0917190015, 0,175662994, 0,224291995, 0,214785993, 0,164546996, 0,105048999}



Imaginemos que o valor gerado desta vez foi **0,15**. Com o mesmo raciocínio, e com ajuda do gráfico, chegamos aos golos que o FCPorto marcou nesta simulação: **2**.

Nesta simulação o resultado foi, assim:

AROUCA 3 - **FCPORTO** 2

Nota:

O método usado tem várias falhas. As mais fáceis de observar são:

- A distribuição de Poisson, não tem, obviamente, em conta os elementos situacionais, como o estado do terreno de jogo, lesões de jogadores importantes, a importância do jogo em questão, etc…
- O tamanho da base de dados dita o realismo das probabilidades, e consequentemente, das odds. Para termos gráficos realistas, a base de dados teria, ela própria de ser criada com base em probabilidades reais e não de uma forma completamente aleatória, **como foi feito**. Ou seja, um gráfico realista, teria uma probabilidade de marcar 0, 1 e 2 golos muito maiores que o resto das ocorrências, logo, a probabilidade de uma equipa marcar 3 ou mais golos numa simulação seria muito mais baixa.

As restantes funções são simples leituras e escritas para ficheiro, com o intuito de guardar os valores calculados, bem como toda a informação de cada modalidade, e menus, para poder interagir com o utilizador.

Assim, há uma série de ficheiros essenciais para o bom funcionamento do programa:

- modalidades.txt
- *-clubes.txt
- *-jogos.txt
- *-resultados.txt
- *-poisson.txt

Sendo * novamente um placeholder para o nome das modalidades.

Porém, se o utilizador desejar, apenas precisa do ficheiro “modalidades.txt” para executar o programa. Tudo o resto pode ser criado, dentro do programa, por ele. Para tal, basta seleccionar a opção **“4- ALTERAR DEFINICOES”** do menu inicial. Ao criar uma nova equipa, o programa cria automaticamente os jogos possíveis. Os resultados e poisson são criados posteriormente, quando o utilizador, com as equipas e modalidades que deseja, for a opção **“3- ALTERAR COTAS”** do menu “ALTERAR DEFINICOES” e seleccionar a opção **“1- RECALCULAR COTAS”**.

A partir daí, basta jogar!

Conclusão

Após conclusão do projecto podemos assumir que complicamos bastante a leitura do enunciado. Porém, o facto de o termos feito deu-nos uma experiência de programação muito superior. Sem dúvida que a complexidade do nosso projecto deveu-se a muita pesquisa, trabalho em grupo e acaba por consolidar todos os conhecimentos que adquirimos em aula, e mais.

Deixamos também em sugestão alguns pontos que podem ser melhorados numa futura release:

- Usando as simulações com base nas probabilidades calculadas, aumentar a base de dados, melhorando assim o realismo das probabilidades e odds.
- Juntar um simulador com relatos ou movimento de pixels a simular o movimento de jogadores.
- Melhorar a capacidade de sustentar modalidades com um número grande de pontos máximos.
- Aplicar funções que ajudem modalidades como o ténis, que na realidade não têm empates, a funcionar correctamente.

Bibliografia

https://www.eecis.udel.edu/~portnoi/classroom/prob_estatistica/2007_1/lecture_slides/aula11.pdf - 23/12/2016

https://www.pinnacle.com/en/betting-articles/soccer/how-to-calculate-poisson-distribution 23/12/2016