

**Sistemas Operativos**

**Trabalho 2**

**Simulação de um Jogo de Futebol**

**Professor:**

Nuno Lau ([nunolau@ua.pt](mailto:nunolau@ua.pt))

**Pedro** **Sobral**, 98491, P3

**Daniel Figueiredo**, 98498, P5

**As distribuições de percentagem são iguais para os membros do grupo**

21/01/2021

­

**Índice**

[**1 - Introdução** 3](#_Toc61971073)

[**2 - Introdução ao Problema** 4](#_Toc61971074)

[2.1 - Compilação e Execução 4](#_Toc61971075)

[2.2 – O problema 4](#_Toc61971076)

[**3 – Implementação** 6](#_Toc61971077)

[3.1 - Referee 6](#_Toc61971078)

[3.1.1 - *arrive()* 7](#_Toc61971079)

[3.1.2 - *waitForTeams()* 8](#_Toc61971080)

[3.1.3 - *startGame()* 9](#_Toc61971081)

[3.1.4 - play() 10](#_Toc61971082)

[3.1.5 - *endGame()* 11](#_Toc61971083)

[3.2 - Goalie 12](#_Toc61971084)

[3.2.1 - *arrive()* 13](#_Toc61971085)

[3.2.2 - *goaliesConstituteTeam()* 14](#_Toc61971086)

[3.2.3 - *waitReferee()* 17](#_Toc61971087)

[3.2.4 - *playUntilEnd()* 18](#_Toc61971088)

[3.3 - Player 19](#_Toc61971089)

[3.3.1 - *arrive()* 20](#_Toc61971090)

[3.3.2 - *playerConstituteTeam()* 21](#_Toc61971091)

[3.3.3 - *waitReferee()* 24](#_Toc61971092)

[3.3.4 - *playUntilEnd()* 25](#_Toc61971093)

[**4 - Resultados** 26](#_Toc61971094)

[4.1 - Confirmação dos resultados 26](#_Toc61971095)

[4.1.1 - Avaliação de *deadlocks* 26](#_Toc61971096)

[4.1.2 - Confirmação dos resultados 27](#_Toc61971097)

[**5- Conclusão** 28](#_Toc61971098)

[**6 - Bibliografia** 29](#_Toc61971099)

# **1 - Introdução**

No âmbito da unidade curricular de Sistemas Operativos, foi-nos apresentado a realização deste trabalho prático, que consiste na realização de uma simulação de um jogo de futebol, com 3 envolventes, *player, goalie, referee*, todos os envolventes são processos independentes, sendo que a sua sincronização e comunicação é realizada através de semáforos e de memória partilhada.

A simulação é constituída por duas equipas, sendo que cada uma terá 5 jogadores, 4 de campo (*player*), e um guarda-redes (*goalie*), as equipas vão sendo formadas à medida que os jogadores vão chegando, se ambas as equipas já estiverem formadas, os jogadores que chegarem irão ser informados que as equipas se encontram completas e que por consequência não entraram no jogo. Por jogo, existe um *referee*, este envolvente é bastante importante pois é ele que dita o início da partida, e o fim da mesma. De maneira a que não ocorram *deadlocks*, vamos usar semáforos para que não ocorra “choque” de informação, e deste modo a simulação correr da maneira correta.

A implementação do código será feita através do IDE *VSCode*, pois é um editor com que ambos os membros do grupo já estão familiarizados derivados a unidades curriculares e projetos passados. A execução de todo o trabalho foi suportada através de um [repositório no GitHub](https://github.com/TheScorpoi/SO_Trabalho02)1, o que facilita em muito todo o *workflow* da realização do mesmo.

Com a realização deste trabalho prático, esperamos conseguir cumprir todos os pontos essenciais que o guião propõe, e veemente alargar os nossos horizontes no que toca a programar com semáforos em C, pois são uma variável muito importante no que toca a controlar a acesso a determinadas regiões por parte de vários processos

1 – O repositório encontra-se privado, pelo menos até ao dia da entrega do trabalho (21/01/2021)

# **2 - Introdução ao Problema**

## 2.1 - Compilação e Execução

Para compilar o programa é necessário à partida ter um compilador de C instalado na máquina, por exemplo o gcc. Posto isto, para compilar basta executar, estando na pasta /semaphore\_soccergame/src/, o comando:

*make all*

De seguida, temos de entrar na pasta /semaphore\_soccergame/run/, para que possamos “simular o jogo de futebol”, fazendo:

*./probSemSharedMemSoccerGame*

Para averiguar se havia a existência de algum *dead lock,* executamos o código *run.sh,* que nos permite executar um determinado número de vezes o ./*probSemSharedMemSoccerGame*

**Nota :** Para se correr o programa *run.sh,* pela primeira vez, teremos de dar permissões ao utilizador, assim, das próximas vezes em que for necessário correr esse programa, não teremos de repetir esse passo de novo.

## ­2.2 – O problema

Sendo direto e conciso, o que se pretende obter é uma maneira eficaz computacionalmente que lide com uma gestão de recursos, neste caso envolvendo 3 entidades.

FALAR MAIS DISTO

O grande problema a resolver, é fazer com que as entidades façam o que lhes está destinado no momento certo,

FALAR MAIS DISTO

Para que tudo ocorra como esperado há algumas orientações que são obrigatórias cumprir:

* 1
* 1
* 1
* 1
* 1
* 1

Como base de implementação para este trabalho prático usamos o código-fonte fornecido pelo Professor, código esse onde já se encontravam as variáveis e os semáforos inicializados, entre outras funções.

O estado que cada individuo pode abarcar está definido no ficheiro *probConst.h*, os estados serão abordados mais à frente.

Existem 3 entidades, sendo que dispõem das seguintes funções:

* ***Player –*** Chegar, pertencer a uma equipa e jogar.
* ***Goalie –*** Chegar, pertencer a uma equipa e jogar.
* ***Referee –*** Iniciar, arbitrar, e terminal o jogo.

Relativamente à memória partilhada, esta encontra-se no ficheiro *shareDataSync.h,* nas estruturas *FULL\_STAT*, e *STAT*.

A utilização de semáfororos serve essencialmente para controlar o acesso à memoria partilhada, evitando assim choques de informação entre os 3 individuos que compõem o trabalho prático. As notificações entre indivíduos foram feitas através de semáforos, para que desta forma o programa executasse sem problemas maiores.

Para facilitar todo o processo de implementação, dos semáforos e da memoria partilhada, elaboramos uma tabela, com todos os semáforos que foram alterados por nós, desta forma a implementação dos semáforos no código tornou-se mais simples e rápida, pois a tabela estrutura de forma bastante rigorosa, o que, e onde as alterações no código-fonte serão feitas.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| SEMAFORO | Entidade Down | Função Down | #Downs | Entidade Up | Função Up | #Ups |
| playersWaitTeam | Jogadores ou guarda-redes | playerConstituteTeam()  goalieConstituteTeam() | 1/0 | Jogador formador da equipa | playerConstituteTeam()  goalieConstituteTeam() | 3/4 |
| goaliesWaitTeam | Jogadores ou guarda-redes | playerConstituteTeam()  goalieConstituteTeam() | 1/0 | Jogador formador da equipa | playerConstituteTeam()  goalieConstituteTeam() | 1/0 |
| playersWaitReferee | Jogador formador da equipa | playerConstituteTeam()  goalieConstituteTeam() | 10 | Árbitro | startGame() | 10 |
| playersWaitEnd | Árbitro | endGame() | 10 | Jogadores ou guarda-redes | playUntilEnd() | 10 |
| refereeWaitTeams | Árbitro | waitForTeams() | 2 | Jogador formador da equipa | playerConstituteTeam()  goalieConstituteTeam() | 2 |
| playersRegistered | Jogadores ou guarda-redes | playerConstituteTeam()  goalieConstituteTeam() | 1 cada | Jogador formador da equipa | playerConstituteTeam()  goalieConstituteTeam() | 1 cada |

# **3 – Implementação**

## 3.1 - Referee

Para a nossa simulação do jogo de futebol, como já referido anteriormente, iremos necessitar de um árbitro, sendo que, o nosso árbitro, ao longo do código irá assumir cinco estados diferentes. Sendo eles (Fig.1):

* ***ARRIVING****,* tomando o valor 0, que significa que está a chegar ao campo .
* ***WAITING\_TEAMS***, tomando o valor 1, que significa que o árbitro está à espera de que se formem as duas equipas.
* ***STARTING\_GAME***, tomando o valor 2, que significa que, as equipas já estão formadas, logo o árbitro pode iniciar o jogo.
* ***REFEREEING***, tomando o valor 3, que significa que o árbitro está a arbitrar o jogo
* ***ENDING\_GAME***, tomando o valor 4, que significa que o árbitro termina o jogo.

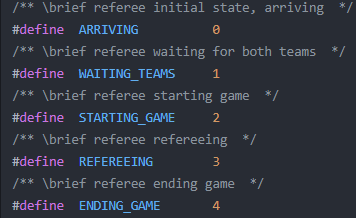


Figura - Constantes de estado do árbitro

Para que o árbitro percorra, todos estes estados mencionados, serão utilizadas cinco funções, sendo elas *arrive(), waitForTeams(), startGame(), play() e endGame()*.

### 3.1.1 - *arrive()*

Nesta função (Fig.2), é pedido para que se atualize o estado do árbitro. Logo, alteramos o estado do árbitro para *ARRIVING*, dentro da região crítica do *mutex* e em seguimos guardamos esse estado.

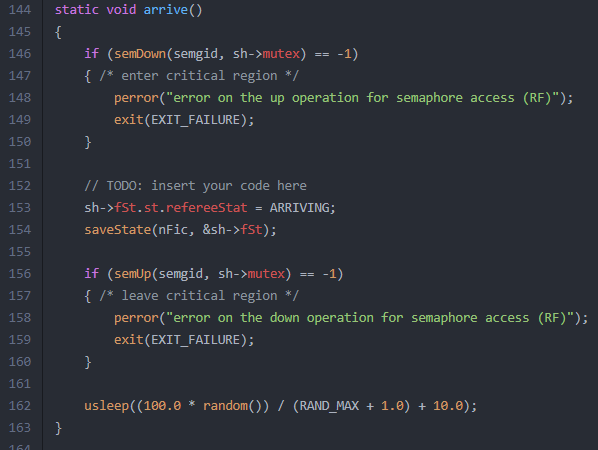


Figura – Função arrive() do árbitro

### 3.1.2 - *waitForTeams()*

Na função *waitForTeams()* (Fig.3)é pedido que o árbitro espere que as equipas estejam completamente formadas, ou seja, quatro jogadores e apenas um guarda-redes em cada equipa. Para isso, é necessário que o árbitro atualize o seu estado e que seja utilizado o semáforo *refereeWaitTeams.*

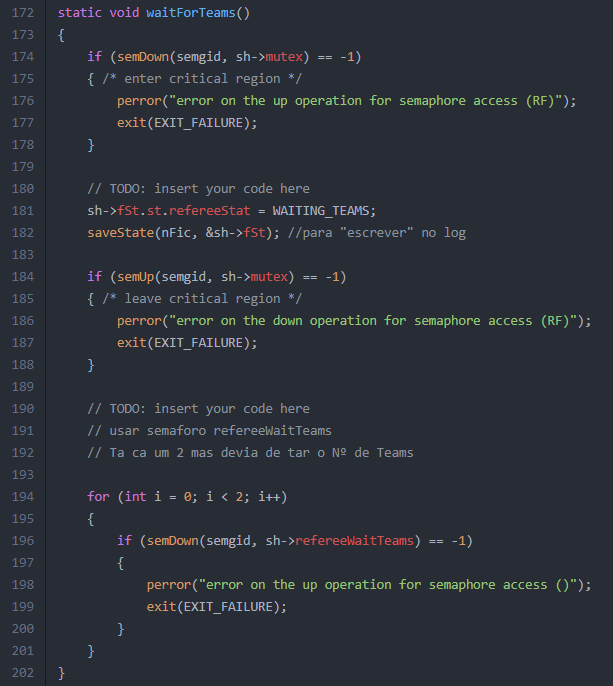


Figura - Função waitForTeams()

Para a implementação deste semáforo, utilizamos um ciclo *for*, iterando-o duas vezes, visto que existem duas equipas. Este semáforo, utilizado pelo árbitro, vai fazer dois *downs*, representando que vai estar à espera pela confirmação do último membro de cada equipa (o que formou a equipa) com a informação que as equipas estão completas.

### 3.1.3 - *startGame()*

A função *startGame()* (Fig.4), pede que o árbitro comece o jogo. Para isso, o árbitro tem de alterar o seu estado atual e terá de notificar os jogadores todos que o jogo vai começar.

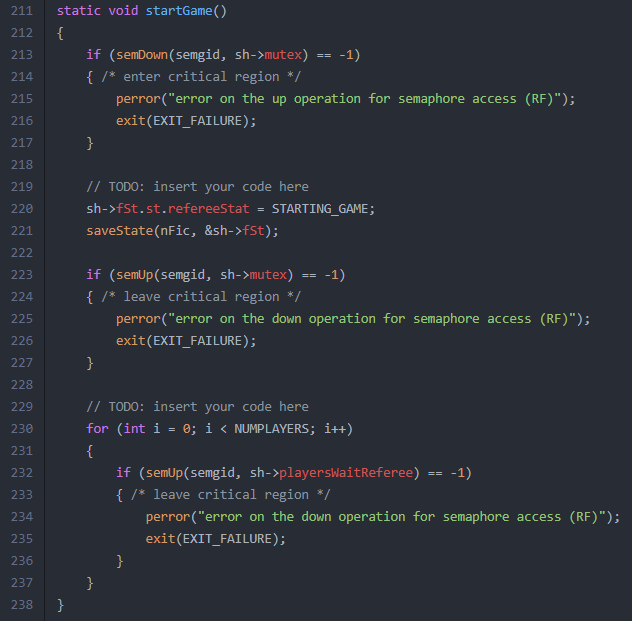


Figura - Função startGame()

Dentro da região crítica, atualizamos o estado do árbitro para *STARTING\_GAME*, significando que o jogo vai começar. Para que o árbitro pudesse avisar os jogadores que o jogo iria começar, fora da região crítica, utilizamos o semáforo *playersWaitReferee*, implementando este, dentro de um ciclo *for,* iterando *NUMPLAYERS(10)* vezes, fazendo em cada iteração *semUp,*  que permite, ao árbitro, notificar cada jogador que o jogo vai começar.

### 3.1.4 - play()

Para a função *play()* (Fig.5)*,* era pedido, apenas, que o árbitro deixasse passar algum tempo para que pudesse terminar o jogo e que alterasse o seu estado enquanto o jogo estivesse a decorrer. Para isso, dentro da região crítica, alteramos o estado do árbitro para *REFEREEING*, estando este, assim, a arbitrar o jogo.

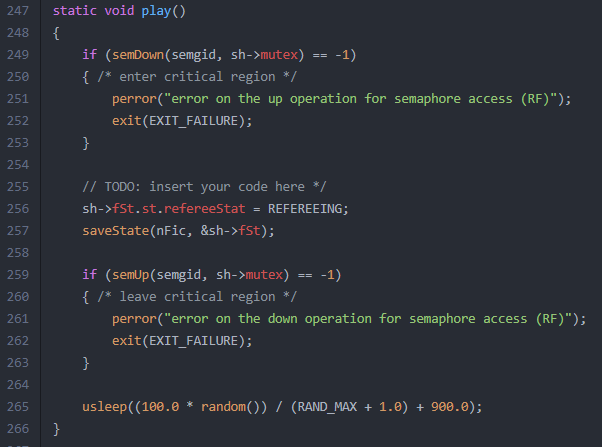


Figura - Função play()

### 3.1.5 - *endGame()*

Na função *endGame()* (Fig.6)*,* era pedido que o árbitro alterasse o seu estado e que notificasse todos os jogadores que o jogo terminou. Alteramos, dentro da região crítica, o estado do árbitro para *ENDING\_GAME*, terminando assim o jogo.



Figura – Função endGame()

Para que o árbitro pudesse notificar os jogadores acerca do término do jogo, fora da região crítica, utilizamos um ciclo *for*, iterando sobre ele *NUMPLAYERS*(10) vezes, fazendo em cada iteração *semUp,*  que permite, ao árbitro, notificar cada jogador que o jogo terminou.

## 3.2 - Goalie

Quanto aos *goalies*, no jogo pode haver 3, no entanto só irão jogar 2, ficando desta forma sempre um que não irá jogar. O guarda-redes, é um envolvente do jogo que pode estar em diversos estados, estados esses que estão representados no ficheiro *probConst.h* (Fig. 7).

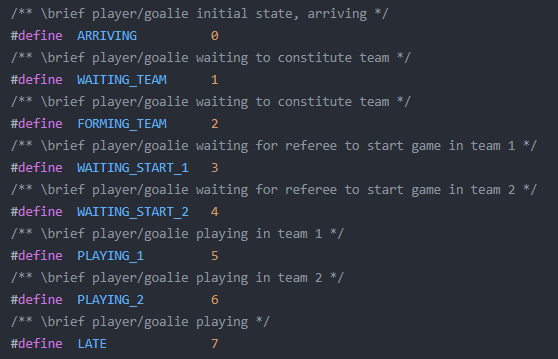


Figura - Constantes de estado do Goalie

Como a figura. 7, demonstra o *goalie* pode arcar 8 estados diferentes:

* **ARRIVING**  **0** - significa que está a chegar ao campo.
* **WAITING\_TEAM 1 -** espera que a equipa seja constituída.
* **FORMING TEAM 2 -**  *goalie* forma a equipa.
* **WAITING\_START\_1 3 -**  *goalie* da equipa 1 espera pelo início do jogo.
* **WAITING\_START\_2 4 -** *goalie* da equipa 2, espera pelo início do jogo.
* **PLAYING\_1 5 -** goalie da equipa 1, está a jogar.
* **PLAYING\_2 6 -** goalie da equipa 2, está a jogar.
* **LATE 7 -** *goalie chegou atrasado, não irá jogar.*

### 3.2.1 - *arrive()*

Na função *arrive()* (Fig.8), as alterações feitas foram minutas, uma vez que o necessário a fazer é somente alterar o estado do *goalie* em concreto, para isso usamos o *id* do mesmo para ARRIVING, esta alteração foi é feita dentro da região crítica. Fizemos uso da função *saveState()* para que o estado agora já atualizado aparecesse no terminal.

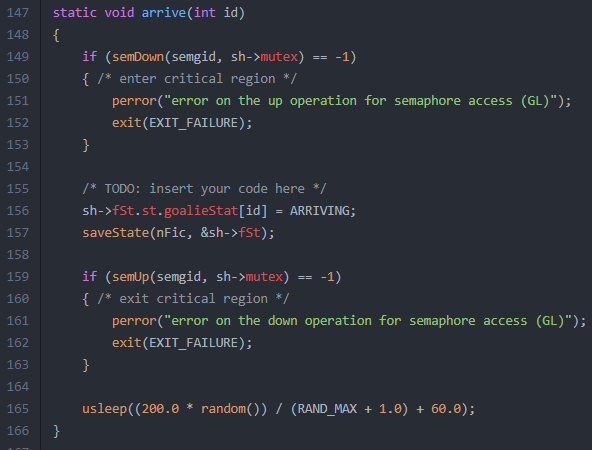


Figura - Função arrive() do Goalie

### 3.2.2 - *goaliesConstituteTeam()*

Na função *goaliesConstituteTeam()* (Fig.9) realizamos mais alterações dentro da zona crítica do *mutex* comparando com as restantes funções. Para esta função, dentro da região crítica, começamos por fazer incrementos às variáveis *goaliesFree* e *goaliesArrived,* visto que, ao chegar a este ponto da função, vamos ter mais um guarda-redes que está livre (ainda não foi colocado em nenhuma equipa) e esse mesmo é mais um guarda-redes que chegou(por isso o incremento na variável *goaliesArrived*).

Ainda dentro da região crítica, avaliamos se já chegaram pelo menos dois guarda-redes, pois para se terem duas equipas precisamos de ter dois guarda-redes (um para cada equipa) e caso já tenham chegado pelo menos dois vamos avaliar se existem quatro ou mais jogadores livres, visto que para se formar uma equipa precisamos de quatro jogadores e de um guarda-redes. Se essa condição se verificar, então alteramos o estado do guarda-redes para *FORMING\_TEAM* e fazemos um decremento da variável *goaliesFree,* pois foi um guarda-redes que passou de estar livre para pertencer a uma equipa. Para além disso, se for viável formar uma equipa, então o guarda-redes, ao iterar sobre o ciclo *for* *NUMTEAMPLAYERS(4) vezes,* faz em cada iteração um *semUp* ao semáforo *playersWaitTeam,* simbolizando a entrega da confirmação que quatro jogadores pertencem agora à equipa do guarda-redes e deste modo faz também noutro ciclo *for*, *NUMTEAMPLAYERS(4) semDowns* ao semáforo *playersRegistered*, significando que recebeu as confirmações que quatro jogadores ingressaram numa equipa. Caso ainda não haja jogadores livres ou guarda-redes livres suficientes, então irá ficar no estado WAITING\_TEAM.

Caso já tenham chegado dois guarda-redes (*goaliesArrived*), então o próximo guarda-redes a chegar irá passar para o estado *LATE*, e irá permanecer nesse estado até ao final do jogo.



Figura – Função goaliesConstituteTeam() dentro da zona crítica

Fora da região crítica (Fig.10), caso o guarda-redes se encontre no estado *FORMING\_TEAM*, ou seja este está a constituir uma equipa, iremos fazer um *semUp* ao semáforo *refereeWaitTeams,* para o guarda-redes notificar o árbitro que uma equipa foi formada.

Caso não esteja no estado *FORMIMG\_TEAM,* vamos ver se está no estado *WAITING\_TEAM* e caso esteja faz um *semDown* do semáforo *goaliesWaitTeam,* que notifica que o guarda-redes está à espera que a equipa fique formada.De seguida, iremos à variável *ret* e colocamos o valor de *teamId++*.Outro semáforo necessário para esta função é o *playerRegistered,* fazendo um *semUp,* simbolizando que houve um jogador que se registou na equipa.



Figura - Função goaliesConstituteTeam() fora da zona crítica

### 3.2.3 - *waitReferee()*

Quanto à função *waitReferee() (*Fig.11*),* é dito que é necessário atualizar o estado do *goalies*, e esperar que o *referee* comece o jogo.

Relativamente à parte de atualizar o estado, este é feito dentro da região crítica, utilizamos um *if*, para distinguir a team a que o goalie pertence, para a *team* 1, o estado com que o *goalie* fica é WAITING\_START\_1, para a *team* 2, o estado com que o *goalie* fica é WAITING\_START\_2. Ao fim usamos a função *saveState()*, para que o estado seja atualizado.

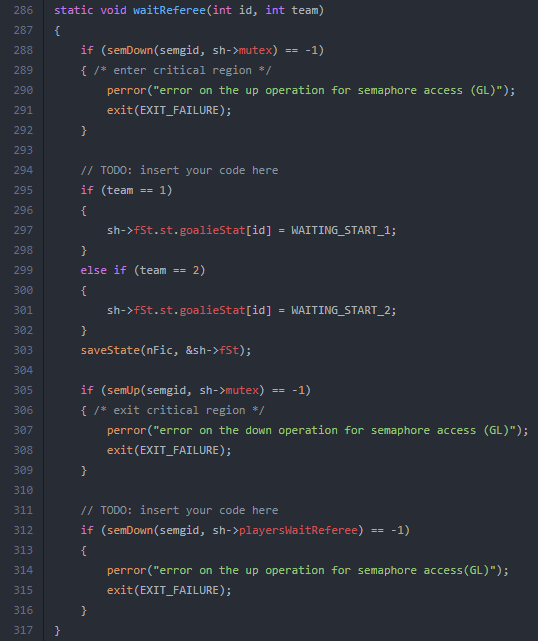


Figura - Função waitReferee() do Goalie

Agora já fora da região crítica, vamos usar o semáforo *playersWaitReferee*, através de um *Down*, pois os *goalies* vão estar à espera da confirmação do árbitro para que o possam começar a jogar.

### 3.2.4 - *playUntilEnd()*

Relativamente à função *playUntilEnd()* (Fig.12), o que fizemos foi: atualizar o estado o *goalie* e esperar que o árbitro termine o jogo.

Dentro da região crítica, vamos atualizar o estado do *goalies*, a atualização é feita com um *if*, para distinguir a team a que o goalie pertence, para a *team* 1, o estado com que o *goalie* fica é PLAYING\_1, para a *team* 2, o estado com que o *goalie* fica é PLAYING\_2. Ao fim usamos a função *saveState()*, para que o estado seja atualizado.

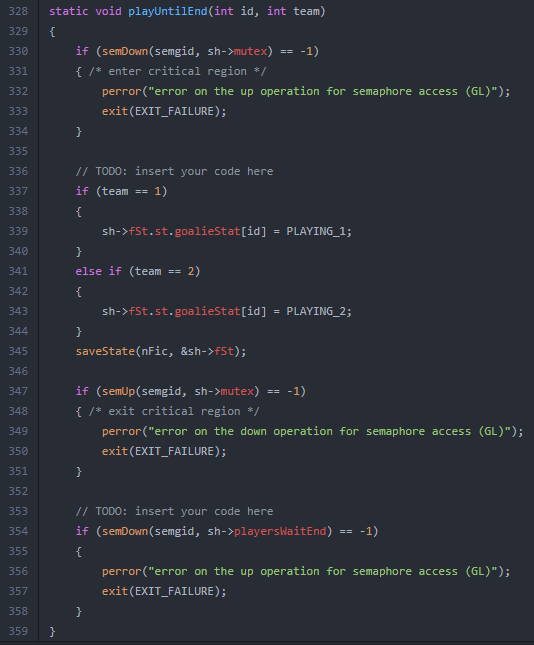


Figura - Função playUntilEnd() do Goalie

Já fora da região crítica , fazemos uso do semáforo *playersWaitEnd,* com recurso à função *semDown(),* fazemos um *Down no semáforo, pois desta forma os jogadores vão estar à espera que o jogo acabe.*

## 3.3 - Player

Quanto aos *players*, no jogo pode haver 10, no entanto só irão no jogar 8, 4 em cada equipa, ficando desta forma sempre dois que não irá jogar. O *player*, é um envolvente do jogo que pode estar em diversos estados, estados esses que estão representados no ficheiro *probConst.h* (Fig. 13).

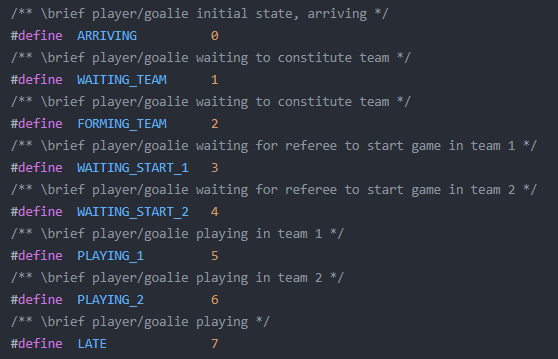


Figura - Constantes de estado do Player

Como a figura 13, demonstra o *player* pode assumir 8 estados diferentes:

* **ARRIVING**  **0** - significa que está a chegar ao campo.
* **WAITING\_TEAM 1 -** espera que a equipa seja constituída.
* **FORMING TEAM 2 -**  *player* forma a equipa.
* **WAITING\_START\_1 3 -**  *player* da equipa 1 espera pelo início do jogo.
* **WAITING\_START\_2 4 -** *player* da equipa 2, espera pelo início do jogo.
* **PLAYING\_1 5 -** *player* da equipa 1, está a jogar.
* **PLAYING\_2 6 -** *player* da equipa 2, está a jogar.
* **LATE 7 -** *player* chegou atrasado, não irá jogar.

### 3.3.1 - *arrive()*

Na função *arrive()* (Fig.14), a única alteração necessária é atualizar o estado do *player* em concreto com o seu id, para ARRIVING, esta alteração é efetuada dentro da região crítica. De seguida, usamos a função *saveState(),* para o estado ser atualizado.

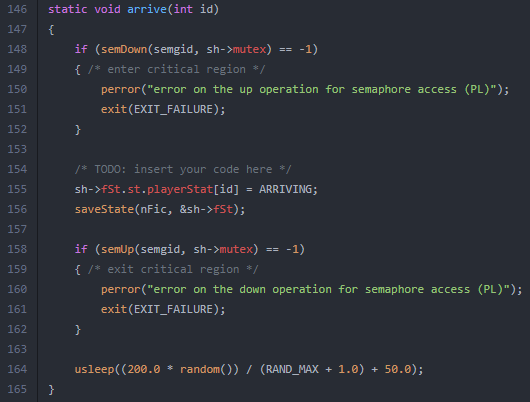


Figura - Função arrive() do Player

### 3.3.2 - *playerConstituteTeam()*

Na função *playerConstituteTeam()* (Fig.15) realizamos mais alterações dentro da zona crítica do *mutex* comparando com as restantes funções. Para esta função, dentro da região crítica, começamos por fazer incrementos às variáveis *playersFree* e *playersArrived,* visto que, ao chegar a este ponto da função, vamos ter mais um jogador que está livre (ainda não foi colocado em nenhuma equipa) e esse mesmo é mais um jogador que chegou(por isso o incremento na variável *goaliesArrived*).

Ainda dentro da região crítica, avaliamos se já chegaram pelo menos oito guarda-redes, pois para se terem duas equipas precisamos de ter oito jogadores (quatro para cada equipa) e caso já tenham chegado pelo menos oito vamos avaliar se existem quatro ou mais jogadores livres, visto que para se formar uma equipa precisamos de quatro jogadores e de um guarda-redes. Se essa condição se verificar, então alteramos o estado do jogador para *FORMING\_TEAM* e fazemos um decremento da variável *playersFree,* pois foi um jogador que passou de estar livre para pertencer a uma equipa. Para além disso, se for viável formar uma equipa, então o jogador, ao iterar sobre o ciclo *for* (*NUMTEAMPLAYERS(4) – 1) vezes (*faz menos uma vez pois ele já pertence à equipa *),* faz em cada iteração um *semUp* ao semáforo *playersWaitTeam,* simbolizando a entrega da confirmação que três jogadores pertencem agora à equipa e deste modo faz também (*NUMTEAMPLAYERS(4) – 1) semDowns* ao semáforo *playersRegistered*, significando que recebeu as confirmações que três jogadores ingressaram numa equipa , fazendo o decremento da variável *playersFree* em cada iteração pelo ciclo *for*, sendo esta parte apenas para jogadores mas ainda precisamos de colocar o guarda-redes.

Para colocar o guarda-redes na equipa, como entrou na condição *if,* então vai fazer um *semUp* ao semáforo *goaliesWaitTeam*, que simboliza que o guarda-redes está à espera que a equipa que este ingressou fique completa. Faz-se, também, um *semUp* ao semáforo *playerRegistered* para notificar que o guarda-redes faz agora parte de uma equipa. De seguida faz-se o decremento da variável *goaliesFree,* visto que o guarda-redes ingressou numa equipa e já não se encontra livre. AQUELA MERDA DO RET QUE NAO SEI QUE DIZER.

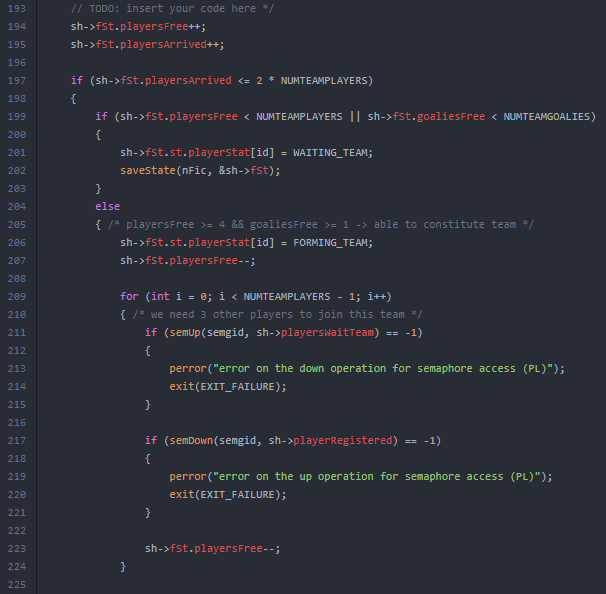


Figura – Função playerConstituteTeam() na zona crítica

Caso ainda não haja jogadores livres ou guarda-redes livres suficientes, então irá ficar no estado WAITING\_TEAM. Caso já tenham chegado oito jogadores(*playersArrived*), então os próximos jogadores a chegar irão passar para o estado *LATE*, e irão permanecer nesse estado até ao final do jogo.

Fora da região crítica (Fig.16), caso o jogador se encontre no estado *FORMING\_TEAM*, ou seja este está a constituir uma equipa, iremos fazer um *semUp* ao semáforo *refereeWaitTeams,* para o jogador notificar o árbitro que uma equipa foi formada.

Caso não esteja no estado *FORMIMG\_TEAM,* vamos ver se está no estado *WAITING\_TEAM* e caso esteja faz um *semDown* do semáforo *playersWaitTeam,* que notifica que o jogador está à espera que a equipa fique formada. De seguida, iremos à variável *ret* e colocamos o valor de *teamId*.Outro semáforo necessário para esta função é o *playerRegistered,* fazendo um *semUp,* simbolizando que houve um jogador que se registou na equipa.

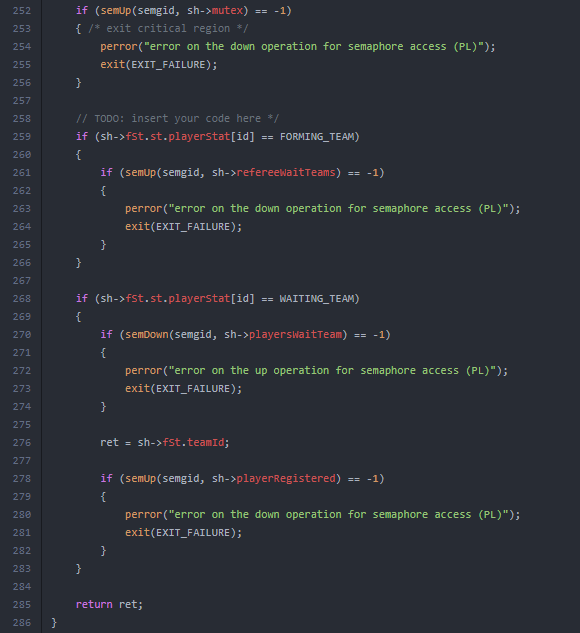


Figura - Função playerConstituteTeam fora da zona crítica

### 3.3.3 - *waitReferee()*

Em relação à função *waitReferee()* (Fig.17), é dito que esta função tem de atualizar o estado do *player*, e esperar que o árbitro dê início ao jogo.

Para atualizarmos o estado temos de fazer um *if, dentro da região critica*, para saber se estamos a tratar de um *player* da *team 1* ou da team 2, caso este seja da *team 1*, o seu estado fica WAITING\_TEAM\_1, caso seja da *team 2*, o seu estado fica WAITING\_TEAM\_2.

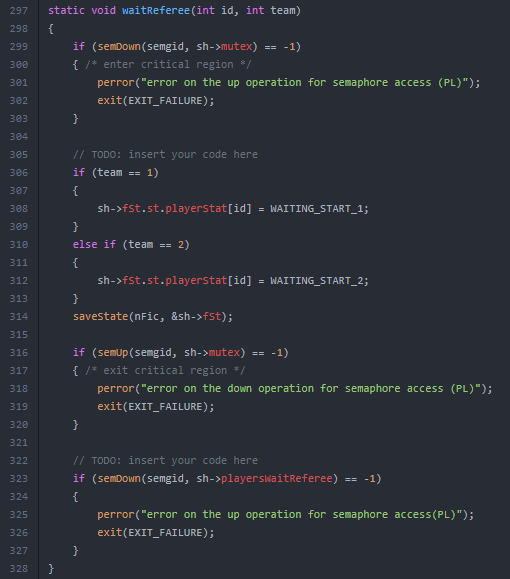


Figura - Função waitReferee() do Player

Já fora da região crítica, fazemos uso do semáforo *playersWaitReferee* através da função *semDown(),* pois os *players* vão estar à espera da confirmação do árbitro para que o possam começar a jogar.

### 3.3.4 - *playUntilEnd()*

Relativamente à função *playUntilEnd()* (Fig.18), o que fizemos foi: atualizar o estado o *player* e esperar que o árbitro termine o jogo.

Dentro da região crítica, vamos atualizar o estado do *players*, a atualização é feita com um *if*, para distinguir a team a que o player pertence, para a *team* 1, o estado com que o *goalie* fica é PLAYING\_1, para a *team* 2, o estado com que o *player* fica é PLAYING\_2. Ao fim usamos a função *saveState()*, para que o estado seja atualizado.



Figura - Função playUntilEnd()

Já fora da região crítica, fazemos uso do semáforo *playersWaitEnd,* com recurso à função *semDown(),* fazemos um Down no semáforo, pois desta forma os *players* vão estar à espera que o jogo acabe*.*

# **4 - Resultados**

Durante a implementação do código foram feitos testes (*make pl, make gl, make rf*), com a nossa implementação e com o código pré-compilado fornecido pelo Professor, para termos sempre uma noção se estávamos a ir no caminho certo para encontrar a solução.

Neste ponto do relatório, vão ser avaliados os resultados obtidos, para isso, inicialmente iremos correr o *script run.sh*, *script* esse que simula 1000 jogos, ao correr-mos o *run.sh* ficamos a saber se existiam *deadlocks* na nossa implementação.

Mais tarde, o que fizemos foi escolher arbitrariamente um desses *outputs*, e analisar detalhadamente todas as transições de estados de todos os indivíduos envolvidos.

## 4.1 - Confirmação dos resultados

### 4.1.1 - Avaliação de *deadlocks*

Ora, nesta fase decidimos correr o nosso programa 1000 vezes (Fig.19), o que é um número considerável de vezes para analisar a existência de *deadlocks*, para isso usamos o script *run.sh*, e como correr todo até ao fim, podemos concluir que não existem quaisquer *deadlocks* na nossa implementação.

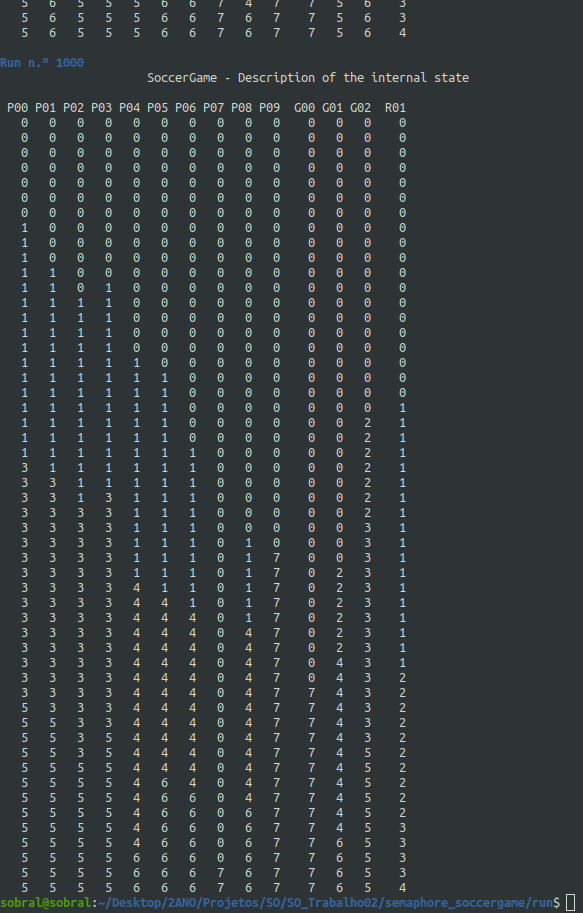


Figura 9 – Averiguação de deadlocks

Não haver *deadlock*, é uma condição necessária, mas não suficiente para um resultado correto, para isso temos de analisar um jogo e ver se todas as fases do mesmo se encontram da maneira certa.

**Nota:**  A figura (esta aqui ao lado), mostra o resultado da última execução, porém existe um ficheiro com as execuções todas que pode ser consultado [aqui](https://github.com/TheScorpoi/SO_Trabalho02/blob/main/semaphore_soccergame/run/output_1000.txt) (Por razões de tamanho é necessário fazer download do ficheiro, o GitHub não o consegue abrir).

### 4.1.2 - Confirmação dos resultados

De acordo com todas as condições que foram por nós abordadas no ponto [2.2 – O problema](#_2.2_–_O), o resultado obtido (Fig.20) mostra que todas essas condições são abordadas e tidas em atenção pelo nosso código C, dessa forma obtivemos os seguintes resultados satisfatórios, a equipa 1 tem 5 elementos (P01, P02, P03, P04, G01), a equipa 2 tem 5 elementos (P01, P05, P06, P06, G02), e um arbitro (R01), sendo que há 3 jogadores que chegaram atrasados e não jogaram (P08, P09, G00).

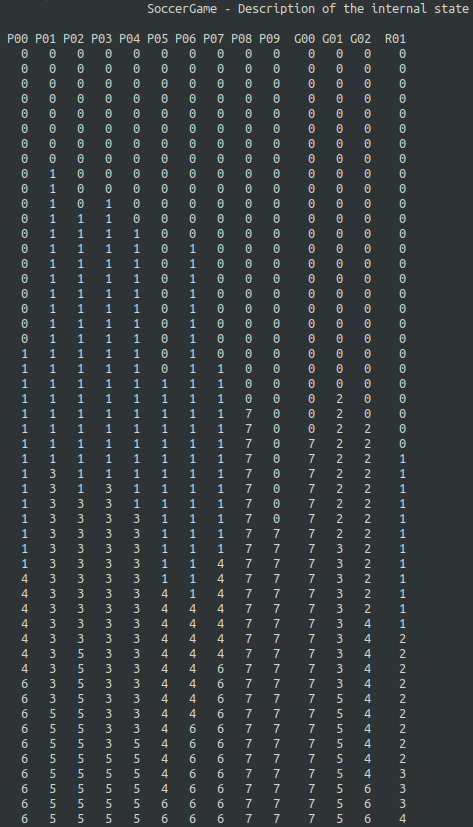


Figura 0 – Analise dos resultados

# **5- Conclusão**

Algo muito importante a reter deste trabalho prático é o conhecimento adquirido com utilização de semáforos e de memória partilhada, ficando mais claros certos pormenores sobre estas temáticas.

No início da realização deste mesmo trabalho, sentimos alguma dificuldade em analisar todo o código fornecido pelo Professor, pois não sabíamos ainda bem o que cada ficheiro continha e fazia, porém à medida que fomos trabalhando sobre os mesmos, essas dificuldades tornaram-se escassas.

Neste ponto, e em tom conclusivo, achamos que conseguimos alcançar todas as metas que o guião do trabalho prático propôs. Os resultados obtidos são nos aspetos essenciais semelhantes aos que o Professor forneceu, e nesse sentido estamos em condições de concluir que conseguimos alcançar uma solução possível e certa.

# **6 - Bibliografia**

- Para a realização deste trabalho prático consultamos os slides disponibilizados pelo Professor na página do *e-learning* da unidade curricular Sistemas Operativos.

Os sites foram consultados entre os dias 20/01/2021 e 23/01/2021:

<https://pt.stackoverflow.com/>

<https://www.geeksforgeeks.org/use-posix-semaphores-c/>

<https://riptutorial.com/c/example/31715/semaphores>