

DEPARTAMENTO DE ELETRÓNICA, TELECOMUNICAÇÕES E INFORMÁTICA

**Simulação de jogo de futebol: Mecanismos associados à execução e
sincronização de processos e *threads***

Sistemas Operativos

Diogo Teixeira - 113042



Prof.: Nuno Lau (nunolau@ua.pt)

2 de janeiro de 2025

Índice

1	Introdução	1
2	Comportamento dos semáforos	2
3	Referee	3
3.1	arrive()	3
3.2	waitForTeams()	3
3.3	startGame()	4
3.4	play()	4
3.5	endGame()	5
4	Goalie	6
4.1	arrive()	6
4.2	goalieConstituteTeam()	6
4.3	waitReferee()	8
4.4	playUntilEnd()	8
5	Player	10
5.1	arrive()	10
5.2	playerConstituteTeam()	10
5.3	waitReferee()	12
5.4	playUntilEnd()	13
6	Testes	14
6.1	Avaliação de <i>Deadlock</i>	14
6.2	Confirmação de Resultados	15
7	Conclusão	16

Capítulo 1

Introdução

No âmbito da cadeira Sistemas Operativos foi proposto um desafio prático que incide sobre a compreensão e implementação de processos de mecanismos de execução e sincronização de processos e *threads*. Este desafio consiste na simulação de um jogo de futebol envolvendo 3 entidades: *Referee*, *Goalies* e *Players*.

Este projeto demonstra uma certa complexidade na coordenação e interação entre os diferentes elementos do jogo, onde cada um é representado por um processo independente. A utilização de semáforos e de memória partilhada é essencial para a sincronização destes processos.

- **Árbitro (*Referee*)**

- O árbitro demora um tempo aleatório (**A**) a chegar ao campo.
- Quando chega, fica em estado de espera (**W**) até que as duas equipas estejam formadas.
- Caso as duas equipas já estejam formadas quando chega, passa apenas brevemente por esse estado.
- Com ambas as equipas formadas, dá início ao jogo (**S**), notifica todos os *players* e *goalies* e aguarda confirmação para garantir que todos estejam prontos para começar.
- Durante o jogo, entra no estado de arbitragem (**R**) e passado um tempo encerra a partida (**E**), notificando todos os participantes.

- **Jogador/Guarda-Redes (*Player/Goalie*)**

- Todos os participantes demoram um tempo aleatório (**A**) a chegar ao campo.
- Ao chegar, verificam se existem vagas para formar equipa.
- Caso cheguem e não haja elementos suficientes para formar equipa, ficam em estado de espera (**W**).
- Os primeiros quatro *players* e um *goalie* a chegar formam a primeira equipa, e aguardam o início da partida (**s**).
- Os próximos 4 *players* e um *goalie* formam a segunda equipa (**S**).
- Se existirem participantes a mais, são informados que chegaram atrasados e não entram no jogo (**L**).
- Com o início do jogo, a primeira e segunda equipa entram nos estados, respetivamente, (**p**) e (**P**).

Capítulo 2

Comportamento dos semáforos

ID do Semáforo	semUp	#Up	semDown	#Down
playersWaitTeam	Player: <ul style="list-style-type: none">• playerConstituteTeam ou Goalie: <ul style="list-style-type: none">• goalieConstituteTeam	3 ou 4	Player: <ul style="list-style-type: none">• playerConstituteTeam	1
goaliesWaitTeam	Player: <ul style="list-style-type: none">• playerConstituteTeam ou Goalie: <ul style="list-style-type: none">• goalieConstituteTeam	1 ou 0	Goalie: <ul style="list-style-type: none">• goalieConstituteTeam	1
playersWaitReferee	Referee: <ul style="list-style-type: none">• startGame	10	Player/Goalie: <ul style="list-style-type: none">• waitReferee	1 cada
playersWaitEnd	Referee: <ul style="list-style-type: none">• endGame	10	Player/Goalie: <ul style="list-style-type: none">• playUntilEnd	1 cada
refereeWaitTeams	Player: <ul style="list-style-type: none">• playerConstituteTeam ou Goalie: <ul style="list-style-type: none">• goalieConstituteTeam	1 ou 1	Referee: <ul style="list-style-type: none">• waitForTeams	2
playerRegistered	Player: <ul style="list-style-type: none">• playerConstituteTeam Goalie: <ul style="list-style-type: none">• goalieConstituteTeam	1 cada	Player: <ul style="list-style-type: none">• playerConstituteTeam ou Goalie: <ul style="list-style-type: none">• goalieConstituteTeam	4 ou 4
playing	Player/Goalie: <ul style="list-style-type: none">• playUntilEnd	1 cada	Referee: <ul style="list-style-type: none">• startGame	10

Tabela 2.1: Comportamento dos semáforos

Capítulo 3

Referee

O árbitro percorre cinco estados durante a simulação do jogo de futebol. Após demorar um tempo aleatório a chegar ao campo, espera que ambas as equipas se formem e sinaliza todos os participantes que está pronto para dar início ao jogo. Depois de iniciar, arbitra durante um determinado tempo e encerra o jogo, voltando a notificar os participantes.

3.1 arrive()

A primeira função serve para simular a chegada do *referee* ao campo. É executado um acesso à zona crítica para atualizar o seu estado para *ARRIVINGR* e é guardado na memória partilhada através de *saveState*. No final, é utilizado um pequeno *delay* para simular o tempo de chegada.

```
static void arrive ()
{
    if (semDown (semgid, sh->mutex) == -1) {                                /* enter critical region */
        perror ("error on the up operation for semaphore access (RF)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
    sh->fSt.st.refereeStat = ARRIVINGR;                                       /* update referee state*/
    saveState(nFic, &sh->fSt);

    if (semUp (semgid, sh->mutex) == -1) {                                    /* leave critical region */
        perror ("error on the up operation for semaphore access (RF)");
        exit (EXIT_FAILURE);
    }

    usleep((100.0*random())/(RAND_MAX+1.0)+10.0);
}
```

Figura 3.1: Função *arrive()* do *Referee*

3.2 waitForTeams()

Após a chegada, o árbitro tem que esperar que ambas as equipas sejam formadas, por isso é feito um acesso à zona crítica e atualizado o seu estado para *WAITING_TEAMS*. Importante realçar que sempre que é alterado o estado de uma entidade, é necessário que esteja seja guardado através de *saveState*. De seguida, é decrementado o semáforo *refereeWaitTeams*, uma vez por cada equipa, bloqueando-o até obter a confirmação dos últimos membros de que a equipas estão formadas.

```
static void waitForTeams ()
{
    if (semDown (semgid, sh->mutex) == -1) { /* enter critical region */
        perror ("error on the up operation for semaphore access (RF)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
    sh->fSt.st.refereeStat = WAITING_TEAMS; /*update referee state*/
    saveState(nFic, &sh->fSt);

    if (semUp (semgid, sh->mutex) == -1) { /* leave critical region */
        perror ("error on the up operation for semaphore access (RF)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
    for(int i = 0; i < (NUMPLAYERS/(NUMTEAMPLAYERS + NUMTEAMGOALIES)); i++){ /*waits for both teams*/
        if (semDown(semgid, sh->refereeWaitTeams) == -1){
            perror("error on the up operation for semaphore access (RF)");
            exit (EXIT_FAILURE);
        }
    }
}
```

Figura 3.2: Função *waitForTeams()*

3.3 startGame()

Com ambas as equipas formadas, o *referee* está pronto para dar início à partida. É atualizado o seu estado para *STARTING_GAME* dentro da região crítica e, de seguida, é utilizado um ciclo *for*, iterando *NUMPLAYERS* (10) vezes, onde é notificado cada um dos participantes através do incremento do semáforo *playersWaitReferee* e é decrementado o semáforo *playing* com o objetivo de aguardar a confirmação dos mesmos.

```
static void startGame ()
{
    if (semDown (semgid, sh->mutex) == -1) { /* enter critical region */
        perror ("error on the up operation for semaphore access (RF)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
    sh->fSt.st.refereeStat = STARTING_GAME; /* update referee state*/
    saveState(nFic, &sh->fSt);

    if (semUp (semgid, sh->mutex) == -1) { /* leave critical region */
        perror ("error on the up operation for semaphore access (RF)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
    for(int i = 0; i < NUMPLAYERS; i++){ /*notifies players*/
        if (semUp(semgid, sh->playersWaitReferee) == -1){
            perror("error on the up operation for sempahore acess (RF)");
            exit(EXIT_FAILURE);
        }
        if (semDown(semgid, sh->playing) == -1){ /*waits players confirmation*/
            perror("error on the operation for semaphore access (RF)");
            exit(EXIT_FAILURE);
        }
    }
}
```

Figura 3.3: Função *startGame()*

3.4 play()

Nesta função é representado o decorrer do jogo. É alterado o estado do árbitro para *REFEERING*, como se estivesse a arbitrar, e introduzido um *delay* para simular a duração do jogo.

```
static void play ()
{
    if (semDown (semgid, sh->mutex) == -1) {                                /* enter critical region */
        perror ("error on the up operation for semaphore access (RF)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */                                        /*update referee state*/
    sh->fSt.st.refereeStat = REFEREEING;
    saveState(nFic, &sh->fSt);

    if (semUp (semgid, sh->mutex) == -1) {                                /* leave critical region */
        perror ("error on the up operation for semaphore access (RF)");
        exit (EXIT_FAILURE);
    }

    usleep((100.0*random())/(RAND_MAX+1.0)+900.0);
}
```

Figura 3.4: Função *play()*

3.5 endGame()

Por fim, atualiza-se o estado do *referee* para *ENDING_GAME*, o que significa que este terminou o jogo, e incrementa-se o semáforo *playersWaitEnd* através de um ciclo *for*, iterado sobre *NUMPLAYERS* (10) vezes, de forma a notificar cada elemento de cada equipa que a partida terminou.

```
static void endGame ()
{
    if (semDown (semgid, sh->mutex) == -1) {                                /* enter critical region */
        perror ("error on the up operation for semaphore access (RF)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */                                        /*update referee state*/
    sh->fSt.st.refereeStat = ENDING_GAME;
    saveState(nFic, &sh->fSt);

    if (semUp (semgid, sh->mutex) == -1) {                                /* leave critical region */
        perror ("error on the up operation for semaphore access (RF)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */

    for(int i = 0; i < NUMPLAYERS; i++){                                    /*notifies players*/
        if (semUp(semgid, sh->playersWaitEnd) == -1){
            perror("error on the up operation for sempahore acess (RF)");
            exit (EXIT_FAILURE);
        }
    }
}
```

Figura 3.5: Função *endGame()*

Capítulo 4

Goalie

O *goalie* percorre um número diferente de estados dependendo do tempo que demora a chegar ao campo. Como existem três destas entidades, um dos guarda-redes vai pertencer à primeira equipa, outro à segunda e o último a chegar é sinalizado que está atrasado e não entra no jogo.

4.1 arrive()

A primeira função é muito semelhante à do processo anterior. É executado um acesso à zona crítica e atualizado o estado de cada um dos *goalies*, identificados pelo *id*, para *ARRIVING*, e guardado na memória partilhada através de *saveState*. Novamente no final, é usado um *delay* para simular o tempo de chegada.

```
static void arrive(int id)
{
    if (semDown (semgid, sh->mutex) == -1) { /* enter critical region */
        perror ("error on the up operation for semaphore access (GL)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
    sh->fSt.goalieStat[id] = ARRIVING; /*update goalie state*/
    saveState(nFic, &sh->fSt);

    if (semUp (semgid, sh->mutex) == -1) { /* exit critical region */
        perror ("error on the down operation for semaphore access (GL)");
        exit (EXIT_FAILURE);
    }

    usleep((200.0*random())/(RAND_MAX+1.0)+60.0);
}
```

Figura 4.1: Função *arrive()* do *Goalie*

4.2 goalieConstituteTeam()

A próxima função já demonstra um nível superior de complexidade comparada com as anteriores. Inicialmente, dentro da zona crítica, começamos por incrementar os contadores *goaliesArrived* e *goaliesFree*, de forma a acompanhar o número de *goalies* que chegaram e os que estão atualmente livres. Ainda dentro da região crítica, é avaliado se chegaram pelo menos 2 guarda-redes e se for o caso, são verificadas 3 condições:

- Se existirem quatro ou mais *players* livres, alteramos o estado do *goalie* para *FORMING_TEAM*, é feito um decremento nos contadores anteriores pelo número necessário para formar equipa e,

ao iterar sobre um ciclo *for* *NUMTEAMPLAYERS* (4) vezes, os *players* são sinalizados para se juntarem à equipa através de um *semUp* do semáforo *playersWaitTeam* e é aguardada a confirmação deles através de um *semDown* do semáforo *playerRegistered*. Por fim, é atribuído um *ID* à equipa, incrementado para a próxima formação e guardado na memória;

```
static int goalieConstituteTeam (int id)
{
    int ret = 0;

    if (semDown (semgid, sh->mutex) == -1){
        perror ("error on the up operation for semaphore access (GL)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
    sh->fSt.goaliesArrived++;
    sh->fSt.goaliesFree++;

    if (sh->fSt.goaliesArrived <= 2){
        if (sh->fSt.playersFree >= NUMTEAMPLAYERS){

            sh->fSt.goalieStat[id] = FORMING_TEAM;

            sh->fSt.playersFree -= NUMTEAMPLAYERS;
            sh->fSt.goaliesFree -= NUMTEAMGOALIES;

            for (int i = 0; i < NUMTEAMPLAYERS; i++){
                if (semUp (semgid, sh->playersWaitTeam) == -1) {
                    perror ("error on the up operation for semaphore access (GL)");
                    exit (EXIT_FAILURE);
                }

                if (semDown (semgid, sh->playerRegistered) == -1) {
                    perror ("error on the down operation for semaphore access (GL)");
                    exit (EXIT_FAILURE);
                }
            }
        }
    }
}
```

Figura 4.2: Função *goalieConstituteTeam()*

- Se não existirem *players* suficientes para formar equipa, o estado do *goalie* é atualizado para *WAITING_TEAM*;
- Caso as condições de cima não se verifiquem, significa que o guarda-redes chegou atrasado e o seu estado é atualizado para *LATE*.

```
ret = sh->fSt.teamId;
sh->fSt.teamId++;
saveState(nFic, &sh->fSt);

} else {
    sh->fSt.goalieStat[id] = WAITING_TEAM;
    saveState(nFic, &sh->fSt);
}

} else {
    sh->fSt.goalieStat[id] = LATE;
    saveState(nFic, &sh->fSt);
    sh->fSt.goaliesFree -= 1;
}

if (semUp (semgid, sh->mutex) == -1) {
    perror ("error on the up operation for semaphore access (PL)");
    exit (EXIT_FAILURE);
}
```

Figura 4.3: Função *goalieConstituteTeam()*

Já fora da zona crítica, caso o *goalie* se encontre no estado *WAITING_TEAM*, é realizado um *semDown* ao semáforo *goaliesWaitTeam*, o que significa que este processo é pausado até que *players* suficientes se tenham juntado para formar equipa e, depois do registo da mesma, é feito um incremento ao semáforo *playerRegistered*, o que sinaliza que o *goalie* está registado. Se o *goalie* se encontrar no estado *FORMING_TEAM*, o *referee* é notificado de que a equipa está formada através de um *semUp* ao semáforo *refereeWaitTeams*.

```
/* TODO: insert your code here */
if (sh->fSt.st.goalieStat[id] == WAITING_TEAM){                                /*if goalie is waiting a team, waits*/

    if (semDown (semgid, sh->goaliesWaitTeam) == -1) {
        perror ("error on the down operation for semaphore access (GL)");
        exit (EXIT_FAILURE);
    }

    ret = sh->fSt.teamId;                                                       /*and registers team*/

    if (semUp (semgid, sh->playerRegistered) == -1) {
        perror ("error on the up operation for semaphore access (GL)");
        exit (EXIT_FAILURE);
    }
} else if (sh->fSt.st.goalieStat[id] == FORMING_TEAM){                        /*if goalie is forming the team*/

    if (semUp (semgid, sh->refereeWaitTeams) == -1) {                          //notifies referee//
        perror ("error on the up operation for semaphore access (GL)");
        exit (EXIT_FAILURE);
    }
}
return ret;
}
```

Figura 4.4: Função *goalieConstituteTeam()*

4.3 waitReferee()

Nesta função, o *goalie* espera pelo *referee* para começar o jogo depois de atualizar o seu estado. Ou seja, dentro da zona crítica, é atualizado o estado do guarda-redes através de um *if* que, dependendo do ID da equipa, esse estado é atualizado para *WAITING_START_1* (s) ou *WAITING_START_2* (S). Fora da zona crítica, é utilizado o semáforo *playersWaitReferee* para sincronizar com o sinal de início de jogo do árbitro, através do decremento do mesmo.

```
static void waitReferee (int id, int team)
{
    if (semDown (semgid, sh->mutex) == -1) {                                  /* enter critical region */
        perror ("error on the up operation for semaphore access (GL)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
    sh->fSt.st.goalieStat[id] = (team == 1) ? WAITING_START_1 : WAITING_START_2; /*update goalie state*/
    saveState(nFic, &sh->fSt);

    if (semUp (semgid, sh->mutex) == -1) {                                    /* exit critical region */
        perror ("error on the down operation for semaphore access (GL)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
    if (semDown (semgid, sh->playersWaitReferee) == -1) {
        perror ("error on the up operation for semaphore access of playersWaitReferee (GL)");
        exit (EXIT_FAILURE);
    }
}
```

Figura 4.5: Função *waitReferee()* do *Goalie*

4.4 playUntilEnd()

Por fim, o *goalie* participa no jogo e espera pelo sinal do árbitro para encerrar a partida. À semelhança da função anterior, é utilizado um *if* para distinguir o ID da equipa e atualizado o estado do guarda-redes para *PLAYING_1* (p) ou *PLAYING_2* (P). Ainda dentro da zona crítica, o *referee* é sinalizado de que o *goalie* está a participar ativamente no jogo através de um *semUp* do semáforo *playing*. Fora da zona crítica, sincroniza-se com o sinal do árbitro indicando que a partida terminou com um decremento do semáforo *playersWaitEnd*.

```
static void playUntilEnd (int id, int team)
{
    if (semDown (semgid, sh->mutex) == -1) {                                /* enter critical region */
        perror ("error on the up operation for semaphore access (GL)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
    sh->fSt.goalieStat[id] = (team == 1) ? PLAYING_1 : PLAYING_2;           /*update goalie state*/
    saveState(nFic, &sh->fSt);

    if (semUp (semgid, sh->playing) == -1) {
        perror ("error on the down operation for semaphore access (GL)");
        exit (EXIT_FAILURE);
    }

    if (semUp (semgid, sh->mutex) == -1) {                                /* exit critical region */
        perror ("error on the down operation for semaphore access (GL)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
    if (semDown (semgid, sh->playersWaitEnd) == -1) {
        perror ("error on the down operation for semaphore access (GL)");
        exit (EXIT_FAILURE);
    }
}
```

Figura 4.6: Função *playUntilEnd()* do *Goalie*

Capítulo 5

Player

O *player* segue a mesma filosofia do processo anterior, mas com um número de entidades diferente. Como existem dez para só irão jogar oito, quatro em cada equipa, os jogadores de campo percorrem diferentes estados entre si dependendo do tempo que cada um demora a chegar ao campo.

5.1 arrive()

Novamente, a primeira função é muito semelhante às dos processos anteriores. É executado um acesso à zona crítica e atualizado o estado de cada um dos *players*, identificados pelo *id*, para *ARRIVING*, e guardado na memória partilhada através de *saveState*. No final, é usado um *delay* para simular o tempo de chegada.

```
static void arrive(int id)
{
    if (semDown (semgid, sh->mutex) == -1) {                               /* enter critical region */
        perror ("error on the up operation for semaphore access (PL)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
    sh->fSt.st.playerStat[id] = ARRIVING;                                   /*update player state*/
    saveState(nFic, &sh->fSt);

    if (semUp (semgid, sh->mutex) == -1) {                                  /* exit critical region */
        perror ("error on the down operation for semaphore access (PL)");
        exit (EXIT_FAILURE);
    }

    usleep(((200.0*random())/(RAND_MAX+1.0))+50.0);
}
```

Figura 5.1: Função *arrive()* do *Player*

5.2 playerConstituteTeam()

Seguindo as mesmas ideias da *goalieConstituteTeam()*, é acedida a zona crítica e incrementados os contadores *playersArrived* e *playersFree*, de forma a acompanhar o número de *players* que chegaram e os que estão atualmente livres. Ainda dentro desta zona, é avaliado se já chegaram pelo menos oito jogadores e, caso se verifique, são avaliadas 2 condições:

- Se existirem *players* e *goalies* livres suficientes para formar equipa, é atualizado o estado de cada jogador para *FORMING_TEAM* e é feito um decremento aos contadores anteriores pelo número necessário para formar equipa. Para além disso, são feitos incrementos ao semáforo *playersWaitTeam* através de um *if*, *NUMTEAMPLAYERS* (4) - 1 vezes, sinalizando os restantes

players que estão a aguardar para formar equipa. O *goalie* também é sinalizado através de um *semUp* do semáforo *goaliesWaitTeam*.

```
static int playerConstituteTeam (int id)
{
    int ret = 0;

    if (semDown (semgid, sh->mutex) == -1) { /* enter critical region */
        perror ("error on the up operation for semaphore access (PL)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
    sh->fSt.playersArrived++; /*increment counters*/
    sh->fSt.playersFree++;

    if (sh->fSt.playersArrived <= 8){
        if (sh->fSt.playersFree >= NUMTEAMPLAYERS && sh->fSt.goaliesFree >= NUMTEAMGOALIES){

            sh->fSt.st.playerStat[id] = FORMING_TEAM;
            sh->fSt.playersFree -= NUMTEAMPLAYERS;
            sh->fSt.goaliesFree -= NUMTEAMGOALIES;

            for (int i = 0; i < (NUMTEAMPLAYERS - 1); i++){ /*signals players and goalies*/
                if (semUp (semgid, sh->playersWaitTeam) == -1) {
                    perror ("error on the up operation for semaphore access (PL)");
                    exit (EXIT_FAILURE);
                }
            }

            if (semUp (semgid, sh->goaliesWaitTeam) == -1) {
                perror ("error on the up operation for semaphore access (PL)");
                exit (EXIT_FAILURE);
            }
        }
    }
}
```

Figura 5.2: Função *playerConstituteTeam()*

Por fim, é feito um *semDown* ao semáforo *playerRegistered*, através de um *if*, *NUMTEAMPLAYERS* (4) - 1 vezes, significando que aguarda confirmação dos restantes membros de que se registaram na equipa. É ainda atribuído um ID único a cada equipa (*teamId*) e incrementado o contador para preparar o sistema para a próxima equipa;

- Se não existirem *players* ou *goalies* para formar equipa, é atualizado o estado do jogador para *WAITING_TEAM*.

Caso um *player* chegue depois de 8 já terem chegado, é atualizado o seu estado para *LATE* e irá permanecer nesse estado até ao final do jogo.

```
for (int i = 0; i < (NUMTEAMPLAYERS - 1); i++){
    if (semDown (semgid, sh->playerRegistered) == -1) {
        perror ("error on the down operation for semaphore access (PL)");
        exit (EXIT_FAILURE);
    }
}

ret = sh->fSt.teamId;
sh->fSt.teamId++;
saveState(nFic, &sh->fSt);

} else { /*if not enough members, it waits*/
    sh->fSt.st.playerStat[id] = WAITING_TEAM;
    saveState(nFic, &sh->fSt);
}

} else {
    sh->fSt.st.playerStat[id] = LATE; /*late player*/
    saveState(nFic, &sh->fSt);
    sh->fSt.playersFree -= 1;
}

if (semUp (semgid, sh->mutex) == -1) { /* exit critical region */
    perror ("error on the up operation for semaphore access (PL)");
    exit (EXIT_FAILURE);
}
```

Figura 5.3: Função *playerConstituteTeam()*

Agora fora da zona crítica, caso o *player* se encontre no estado *WAITING_TEAM*, é realizado um *semDown* ao semáforo *playersWaitTeam*, o que significa que este processo é pausado até que jogadores suficientes se tenham juntado para formar equipa e, depois do registo da mesma, é feito um *semUp* ao semáforo *playerRegistered*, o que sinaliza que o *player* está registado. Por fim, se o jogador se encontrar no estado *FORMING_TEAM*, ou seja, está a constituir uma equipa, o árbitro é notificado de que esta está formada através de um *semUp* ao semáforo *refereeWaitTeams*.

```
/* TODO: insert your code here */
if (sh->fSt.st.playerStat[id] == WAITING_TEAM){                                /*waiting player*/

    if (semDown (semgid, sh->playersWaitTeam) == -1) {
        perror ("error on the down operation for semaphore access (PL)");
        exit (EXIT_FAILURE);
    }

    ret = sh->fSt.teamId;

    if (semUp (semgid, sh->playerRegistered) == -1) {
        perror ("error on the up operation for semaphore access (PL)");
        exit (EXIT_FAILURE);
    }

} else if (sh->fSt.st.playerStat[id] == FORMING_TEAM){

    if (semUp (semgid, sh->refereeWaitTeams) == -1) {
        perror ("error on the up operation for semaphore access (PL)");
        exit (EXIT_FAILURE);
    }

}
return ret;
}
```

Figura 5.4: Função *playerConstituteTeam()*

5.3 waitReferee()

Nesta função, o *player* espera pelo *referee* para começar o jogo depois de atualizar o seu estado. Ou seja, dentro da zona crítica, foi atualizado o estado do jogador através de um *if* que, dependendo do ID da equipa, esse estado é atualizado para *WAITING_START_1* (s) ou *WAITING_START_2* (S). Fora da zona crítica, é utilizado o semáforo *playersWaitReferee* para sincronizar com o sinal de início de jogo do árbitro, através do decremento do mesmo.

```
static void waitReferee (int id, int team)
{
    if (semDown (semgid, sh->mutex) == -1) {                                /* enter critical region */
        perror ("error on the up operation for semaphore access (PL)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
    sh->fSt.st.playerStat[id] = (team == 1) ? WAITING_START_1 : WAITING_START_2;
    saveState(nFic, &sh->fSt);

    if (semUp (semgid, sh->mutex) == -1) {                                /* exit critical region */
        perror ("error on the down operation for semaphore access (PL)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
    if (semDown (semgid, sh->playersWaitReferee) == -1) {
        perror ("error on the up operation for semaphore access of playersWaitReferee (PL)");
        exit (EXIT_FAILURE);
    }
}
```

Figura 5.5: Função *waitReferee()* do *Player*

5.4 playUntilEnd()

Por fim, o *player* participa no jogo e espera pelo sinal do árbitro para encerrar a partida. À semelhança da função anterior, é utilizado um *if* para distinguir o ID da equipa e atualizado o estado do jogador para *PLAYING_1* (p) ou *PLAYING_2* (P). Ainda dentro da zona crítica, o *referee* é sinalizado de que o *player* está a participar ativamente no jogo através de um *semUp* do semáforo *playing*. Fora da zona crítica, sincroniza-se com o sinal do árbitro indicando que a partida terminou com um decremento do semáforo *playersWaitEnd*.

```
static void playUntilEnd (int id, int team)
{
    if (semDown (semgid, sh->mutex) == -1) {                                /* enter critical region */
        perror ("error on the up operation for semaphore access (PL)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
    sh->fst.st.playerStat[id] = (team == 1) ? PLAYING_1 : PLAYING_2;
    saveState(nFic, &sh->fst);

    if (semUp (semgid, sh->playing) == -1) {                                /* exit critical region */
        perror ("error on the down operation for semaphore access (PL)");
        exit (EXIT_FAILURE);
    }

    if (semUp (semgid, sh->mutex) == -1) {                                    /* exit critical region */
        perror ("error on the down operation for semaphore access (PL)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */

    if (semDown (semgid, sh->playersWaitEnd) == -1) {
        perror ("error on the down operation for semaphore access (PL)");
        exit (EXIT_FAILURE);
    }
}
```

Figura 5.6: Função *playUntilEnd()* do *Player*

6.2 Confirmação de Resultados

Avaliando um *output* aleatório, é possível verificar que o resultado obtido está de acordo com as condições descritas nos capítulos anteriores: a equipa 1 tem os 5 elementos (P03, P06, P02, P08, G00), a equipa 2 tem os 5 elementos (P09, P00, P04, P08, G01), e um árbitro (R01), sendo que existem dois *players* e um *goalie* que não jogaram (P05, P01, G02).

```
zval@zval:~/Desktop/S0/Trabalho 2/senaphore_soccergame/run$ ./probSenSharedMemSoccerGame
SoccerGame - Description of the internal state

P00 P01 P02 P03 P04 P05 P06 P07 P08 P09 G00 G01 G02 R01
A A A A A A A A A A A A A A
A A A A A A A A A A A A A A
A A A A A A A A A A A A A A
A A A A A A A A A A A A A A
A A A A A A A A A A A A A A
A A A A A A A A A A A A A A
A A A A A A A A A A A A A A
A A A A A A A A A A A A A A
A A A A A A A A A A A A A A
A A A W A A A A A A A A A A
A A A W A A W A A A A A A A
A A A W A A W A A A A A A A
A A W W A A W A A A A A A A
A A W W A A W A A A A A A A
A A W W A A W A A A A A A A
A A W W A A W A A A A A A A
A A W W A A W A A A A A A A
A A W W A A W A A A A A A A
A A W W A A W W A A A A A A
A A W W A A W W A W A A A A
W A W W A A W W A W A A A A
W A W W W A W W W W A A A A
W A W W W A W W W W F A A A
W L W W W A W W W W F A A W
W L W s W A W W W W F A A W
W L W s W A s W W W F A A W
W L s s W A s W W W F A A W
W L s s W A s s W W s A A W
W L s s W L s s W W s A A W
W L s s W L s s W W s F A W
W L s s W L s s W S s F A W
S L s s W L s s W S s F A W
S L s s S L s s S S s S A W
S L s s S L s s S S s S A S
S L s s S L s s S S s S L S
S L s p S L s s S S s S L S
S L s p S L p s S S s S L S
S L p p S L p p S S s S L S
S L p p S L p p S S p S L S
S L p p S L p p S p S L S
P L p p S L p p S p S L S
P L p p P L p p P p S L S
P L p p P L p p P p P L S
P L p p P L p p P p P L R
P L p p P L p p P p P L E
```

Figura 6.2: Resultados obtidos

Capítulo 7

Conclusão

Neste projeto o objetivo era completar função de 3 ficheiros em C que representavam 3 processos distintos, *Referee*, *Goalie* e *Player*. O seu desenvolvimento foi feito através de semáforos, memória partilhada, e outros conceitos nesta linguagem de programação.

A validação da solução foi feita através de testes fornecidos pelo docente e, no geral, todos os objetivos propostos pelo desafio foram alcançados.