Relatório de SO

Simulação de jogo de futebol

Tomás Gameiro nº119586 Francisco Silva nº118716

Dezembro 2024

Conteúdo

1 Introdução

Este relatório descreve o desenvolvimento de uma solução para a simulação de um jogo de futebol envolvendo um grupo de amigos. O objetivo principal é implementar um programa em linguagem C que utiliza semáforos e memória partilhada para sincronizar e comunicar processos independentes, representando os papéis dos jogadores de campo, guarda-redes e árbitro.

2 Variáveis Gerais e de Estado

No ficheiro probConst.h estão definidas as diversas variáveis que temos que utilizar durante a realização do projeto.

As variáveis representam os possíveis estados e valores máximos para o número de *players*, *goalies* e de *referees*. Os *players*/*goalies* podem ter vários estados possíveis: *ARRIVING*, *WAITING_TEAM*, *FORMING_TEAM*, *WAITING_START_1*/*WAITING_START_2*, *PLAYING_1*/*PLAYING_2* e *LATE*. O mesmo acontece com o árbitro: *ARRIVING*, *WAITING_TEAMS*, *STARTING_GAME*, *REFEREEING* e, por último, *ENDING_GAME*.

Variável	**Valor**
NUMPLAYERS	10
NUMGOALIES	3
NUMREFEREES	1
NUMTEAMPLAYERS	4
NUMTEAMGOALIES	1

Tabela 1: Total de Jogadores, Guarda-redes e Árbitros

Variável	**Código**
ARRIVING	A
WAITING_TEAM	W
FORMING_TEAM	F
WAITING_START_1	s
WAITING_START_2	S
PLAYING_1	p
PLAYING_2	P
LATE	L

Tabela 2: Estados de Jogadores e Guarda-redes

Variável	Código
ARRIVINGR	A
WAITING_TEAMS	W
STARTING_GAME	S
REFEREEING	R
ENDING_GAME	E

Tabela 3: Estados do Árbitro

3 Estruturas de dados internos

Outro ficheiro igualmente importante é o probDataStruct, uma vez que contém as estruturas de dados essenciais para o programa.

O tipo *STAT* é uma estrutura que serve para armazenar os estados dos 3 grupos, *player*, *goalie* e *referee*. Assim, este possui para os *players* um *array* de inteiros *unsigned playerStat* de tamanho 10, para os *goalies* um *array* de inteiros *unsigned playerStat* de tamanho 3 e um inteiro para guardar o estado do único árbitro presente.

```
typedef struct {
    /** \brief players state */
    unsigned int playerStat[NUMPLAYERS];
    /** \brief goalies state */
    unsigned int goalieStat[NUMGOALIES];
    /** \brief referees state */
    unsigned int refereeStat;

} STAT;
```

Figura 1: *Struct STAT*.

Por outro lado o tipo *FULL_STAT* é uma estrutura que serve para armazenar informação sobre praticamente todas as variáveis intrínsecas ao problema dos 3 grupos, *player*, *goalie* e *referee*. Este guarda a estrutura de dados STAT variável st, tendo informação sobre todos os estados atuais, guarda o número total de *players*, *goalies* e *referees*, o número de *players* e *goalies* que já chegaram, bem como o número de *players* e *goalies* disponíveis, e, por último, armazena o ID da próxima equipa a ser formada (inicializado com valor 1).

```
typedef struct
{   /** \brief state of all intervening entities */
    STAT st;

    /** \brief total number of players */
    int nPlayers;

    /** \brief total number of goalies */
    int nGoalies;

    /** \brief total number of referees */
    int nReferees;

    /** \brief number of players that already arrived */
    int playersArrived;
    /** \brief number of goalies that already arrived */
    int goaliesArrived;
    /** \brief number of players that arrived and are free (no team) */
    int playersFree;
    /** \brief number of goalies that arrived and are free (no team) */
    int goaliesFree;

    /** \brief id of team that will be formed next - initial value=1 */
    int teamId;

} FULL_STAT;
```

Figura 2: *Struct FULL_STAT.*

4 Semáforos

Os semáforos foram essencias para a realização deste trabalho. Pois alguns processos tinham que ser executados antes que outros. Os semáforos permitem, assim, bloquear processos quando necessário.

Os semáforos utilizaveis para este projeto foram:

Semáforo	**Propósito do semáforo**
`mutex` *(valor inicial = 1)*	Utilizado na região crítica, este é decrementado através do `semDown()`, bloqueando qualquer outro processo que tente entrar numa região e protegendo a execução do processo atual. A saída desta região é marcada pelo incremento do semáforo com o `semUp()`.
`playersWaitTeam` *(valor inicial = 0)*	Utilizado para bloquear o processo do *player* enquanto alguém não tiver a formar equipa, momento em que o semáforo é incrementado e o processo é libertado.
`goaliesWaitTeam` *(valor inicial = 0)*	Utilizado para bloquear o processo do *goalie* enquanto alguém não tiver a formar equipa, momento em que o semáforo é incrementado e o processo é libertado.
`playersWaitReferee` *(valor inicial = 0)*	Utilizado para bloquear o processo dos *players* e *goalies* enquanto um árbitro não começar o jogo, momento em que o semáforo é incrementado e o processo é libertado.
`playersWaitEnd` *(valor inicial = 0)*	Utilizado para bloquear o processo dos *players* e *goalies* enquanto o árbitro não acabar o jogo, momento em que o semáforo é incrementado e o processo é libertado.
`refereeWaitTeams` *(valor inicial = 0)*	Utilizado para bloquear o processo do *referee* até quando ambas as equipas não estiverem formadas, momento em que o semáforo é incrementado e o processo é libertado.
`playerRegistered` *(valor inicial = 0)*	Utilizado para bloquear o processo dos *players* e *goalies* quando estes não estão inscritos numa equipa, momento em que o semáforo é incrementado e o processo é libertado.
`playing` *(valor inicial = 0)*	Utilizado para bloquear o processo do *referee* até quando ambas as equipas não estiverem prontas, momento em que o semáforo é incrementado e o processo é libertado.

Tabela 4: Descrição dos semáforos e seus propósitos

Durante a execução do trabalho estivemos a desenvolver esta tabela para nos poder-mos orientar nos diversos semáforos. Esta tabela foi editada algumas vezes devido a atribuições erradas aos semáforos, por exemplo o semáforo *playing*, que apesar de funcionar, se colocado na função *waitReferee* podia às vezes executar o código de modo erróneo, por isso escolhe-mos colocá- na função *playUntilEnd* depois de o novo estado ser registado e antes do semáforo de fim de jogo.

Semáforo	**Entidade down**	**Função down**	**#down**	**Entidade up**	**Função up**	**#up**
players WaitTeam	Player	playerConstituteTeam	1	Player (*foarming*)	playerConstituteTeam	3
				Goalie (*foarming*)	goalieConstituteTeam	4
goalies WaitTeam	Goalie	goalieConstituteTeam	1	Player (*foarming*)	playerConstituteTeam	3
				Goalie (*foarming*)	goalieConstituteTeam	4
players WaitReferee	Player & Goalie	waitReferee	1	Referee	startGame	10
Players WaitEnd	Player & Goalie	playUntilEnd	1	Referee	endGame	10
Referee WaitTeams	Referee	waitForTeams	2	Player (*foarming*)	playerConstituteTeam	1
				Goalie (*foarming*)	goalieConstituteTeam	1
Player Registered	Player (*foarming*)	playerConstituteTeam	4	Player & Goalie	playerConstituteTeam	1
	Goalie (*foarming*)	goalieConstituteTeam	4	Goalie	goalieConstituteTeam	1
Playing	Referee	startGame	10	Player & Goalie	playUntilEnd	1

Tabela 5: Descrição dos Semáforos e Funções Relacionadas

5 Código

Na realização deste projeto, concluímos três *scripts* (semSharedMemGoalie.c, semSharedMemPlayer.c e semSharedMemReferee.c), cada um com diversas funções para gerenciar o estado da partida, mas todos compartilham uma função em comum: a função "arrive()".

5.0.1 "arrive()"

Esta função tem como propósito definir o estado inicial do árbitro (*ARRIVINGR*) , guarda-redes ou jogador (*ARRIVING*). Isto é, dependendo de para qual dos indivíduos a função é chamada, esta define que este se encontra no estado 0, ou seja a chegar ao encontro. Este valor é então guardado utilizando o "saveState()".

```
static void arrive(int id)
{
    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.st.goalieStat[id] = ARRIVING;
    saveState (nFic, &sh->fSt);

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

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

Figura 3: Função "arrive()".

5.1 semSharedMemGoalie.c

Neste script foram concluídas para além de "arrive()"as funções: "goalieConstituteTeam()", "waitReferee()"e "playUntilEnd()".

5.1.1 "goalieConstituteTeam()"

Função começa por incrementar o número de guarda-redes livres e guarda-redes que chegaram.

Após isto verifica-se se a quantidade de guarda-redes que já chegaram é igual ao número de guarda-redes cuja chegada é necessária, se for o caso ,é

então, atribuído ao guarda-redes que acabou de chegar o estado de *LATE* e diminuída a quantidade de guarda-redes livres.

```
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.goaliesFree++;
    sh->fSt.goaliesArrived++;
    if(sh->fSt.goaliesArrived==NUMGOALIES){
        sh->fSt.st.goalieStat[id] = LATE;
        saveState (nFic, &sh->fSt);
        sh->fSt.goaliesFree--;
```

Figura 4: 1ª parte da função "goalieConstituteTeam()".

Caso o guarda-redes adicionado não seja o último guarda-redes a chegar, é então verificado se há jogadores e guarda-redes suficientes para criar uma nova equipa, caso isto ocorra é modificado o estado do guarda-redes, acabado de chegar, para *FORMING_TEAM* (ou seja este é o capitão), e diminuídas as quantidades de jogadores e guarda-redes livres correspondentes ás quantias requeridas para criar uma equipa. Itera-se então pelos jogadores e notifica-se a cada um que uma equipa está a ser formada, e que eles fazem parte desta, e para além disso espera-se que os jogadores confirmem o seu registo na equipa (incrementa-se também o *id* da equipa para a próxima equipa ter o *id* seguinte).

```
} else if(sh->fSt.playersFree>=NUMTEAMPLAYERS && sh->fSt.goaliesFree>=NUMTEAMGOALIES){
    sh->fSt.st.goalieStat[id] = FORMING_TEAM;                                    // u
    sh->fSt.playersFree-=NUMTEAMPLAYERS;                                        // decr
    sh->fSt.goaliesFree-=NUMTEAMGOALIES;                                       // decremen


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

    for(int i=0; i<NUMTEAMPLAYERS; i++){
        if (semDown(semgid, sh->playerRegistered) == -1) {                       // w
            perror("error on the up operation for semaphore access (PL)");
            exit(EXIT_FAILURE);
    }
    }
    ret = sh->fSt.teamId++;
    saveState (nFic, &sh->fSt);                                                   //
```

Figura 5: 2ª parte da função "goalieConstituteTeam()".

Caso não haja jogadores suficientes para criar uma equipa o guarda-redes é colocado em estado *WAITING_TEAM*.

Verifica-se então o estado do guarda-redes. Se este estiver em estado WAITING_TEAM, o semáforo *goaliesWaitTeam* é bloqueado e este fica à espera que o capitão indique que este foi incluído na equipa. Ao receber o sinal, o *id* da equipa é atribuído à variável *ret*, o que indica à equipa que o guarda-redes foi associado. Por fim, incrementa-se *playerRegistered*. No entanto, caso o guarda-redes esteja em estado *FORMING_TEAM*, este é o capitão e, então, é enviado um sinal ao árbitro através de *refereeWaitTeams* a informar que a equipa foi criada.

```
09          } else {
10              sh->fSt.st.goalieStat[id] = WAITING_TEAM;
11              saveState (nFic, &sh->fSt);
12          }
13
14          if (semUp (semgid, sh->mutex) == -1) {
15              perror ("error on the up operation for semaphore access (GL)");
16              exit (EXIT_FAILURE);
17          }
18
19          /* TODO: insert your code here */
20          if(sh->fSt.st.goalieStat[id] == WAITING_TEAM){
21              if (semDown (semgid, sh->goaliesWaitTeam) == -1)  {
22                  perror ("error on the up operation for semaphore access (GL)");
23                  exit (EXIT_FAILURE);
24              }
25              ret = sh->fSt.teamId;
26
27              if (semUp (semgid, sh->playerRegistered) == -1)  {
28                  perror ("error on the up operation for semaphore access (GL)");
29                  exit (EXIT_FAILURE);
30              }
31          } else if (sh->fSt.st.goalieStat[id] == FORMING_TEAM){
32              if (semUp (semgid, sh->refereeWaitTeams) == -1) {
33                  perror ("error on the up operation for semaphore access (GL)");
34                  exit (EXIT_FAILURE);
35              }
36          }
37          return ret;
38      }
```

Figura 6: 3ª parte da função "goalieConstituteTeam()".

5.1.2 "waitReferee()"

Verifica-se o estado do guarda-redes e atualiza-se para *WAITING_START_1* ou *WAITING_START_2*, dependendo da equipa a que este pertence. Este estado indica que o guarda-redes está à espera do início do jogo.

De seguida, o semáforo *mutex* é libertado, permitindo que outros processos acedam aos recursos partilhados.

O guarda-redes então bloqueia no semáforo *playersWaitReferee*, ficando à espera que o árbitro sinalize o início do jogo.

```
static void waitReferee (int id, int team)
{
    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.st.goalieStat[id] = (team == 1) ? WAITING_START_1 : WAITING_START_2;
    saveState (nFic, &sh->fSt);



    if (semUp (semgid, sh->mutex) == -1) {
        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 (GL)");
        exit (EXIT_FAILURE);
    }


}
```

Figura 7: Função "waitReferee()".

5.1.3 "playUntilEnd()"

Verifica-se o estado do guarda-redes e atualiza-se para *PLAYING_1* ou *PLAYING_2*, dependendo da equipa a que este pertence. Este estado indica que o guarda-redes está atualmente a jogar.

De seguida, o semáforo *mutex* é libertado, permitindo que outros processos acedam aos recursos partilhados, depois o guarda-redes sinaliza o semáforo *playing* com uma operação de subida (`semUp`). Esta ação serve para informar o árbitro que o guarda-redes está pronto e a participar no jogo.

O guarda-redes então bloqueia no semáforo *playersWaitEnd*, ficando à espera que o árbitro sinalize o fim do jogo.

```
static void playUntilEnd (int id, int team)
{
    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.st.goalieStat[id] = (team == 1) ? PLAYING_1 : PLAYING_2;
    saveState (nFic, &sh->fSt);


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

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

}
```

Figura 8: Função "playUntilEnd()".

5.2 semSharedMemPlayer.c

Neste script foram concluídas para além de "arrive()"as funções: "playerConstituteTeam()", "waitReferee()"e "playUntilEnd()". As implementações destas funções são praticamente idênticas às funções usadas no *script* dos guarda-redes com umas pequenas modificações, então as explicações vão se limitar a estas.

5.2.1 "playerConstituteTeam()"

As mudanças que ocorrem nesta função, limitam-se à mudança dos semáforos usados dos semáforos dos guarda-redes para semáforos associados aos jogadores e em vez de se iterar por 4 jogadores, quando o jogador é capitão, itera-se por 3 jogadores e 1 guarda-redes.

5.2.2 "waitReferee()"

Tal como na função anterior, esta é semelhantíssima à função utilizada no *script* dos guarda-redes, a única mudança são os semáforos acessados, que são os semáforos correspondentes aos jogadores e não aos guarda-redes.

5.2.3 "playUntilEnd()"

Por fim esta função, novamente, realiza as mesmas operações que a função correspondente para os guarda-redes mas com semáforos de jogadores.

5.3 semSharedMemReferee.c

Neste script foram concluídas para além de "arrive()"as funções: "waitForTeams()", "startGame()", "play()"e "endGame()".

5.3.1 "waitForTeams()"

Esta função atualiza o estado do árbitro para *WAITING_TEAMS*. Em seguida, o árbitro entra na fase de espera pela formação de duas equipas. Para isso, bloqueia o semáforo *refereeWaitTeams* duas vezes, uma por cada equipa que precisa ser formada. Cada desbloqueio desse semáforo vem de quem constitui as equipas, sincronizando assim o árbitro com os jogadores e guarda-redes que formam as equipas.

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

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

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

    /* TODO: insert your code here */
    for (int i = 0; i < 2; i++) {
        if(semDown(semgid, sh->refereeWaitTeams) == -1){
            perror("error on the up operation for semaphore access (RF)");
            exit(EXIT_FAILURE);
        }
    }

}
```

Figura 9: Função "waitForTeams()".

5.3.2 "startGame()"

O árbitro atualiza o seu estado para *STARTING_GAME*. De seguida, o árbitro começa a notificar todos os jogadores e guarda-redes para começarem o jogo e fica a espera que todos comecem a jogar.

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

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

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


}
```

Figura 10: Função "startGame()".

5.3.3 "play()"

O estado do árbitro é alterado para *REFEREEING*, o que indica que este está a arbitrar o jogo. Depois é simulada a duração do jogo através do *usleep()*.

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

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

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

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

Figura 11: Função "play()".

5.3.4 "endGame()"

Nesta função que finaliza o jogo, o árbitro vai alterar o seu estado para *ENDING_GAME*. Após isto, notifica todos os jogadores para terminarem as suas atividades no jogo.

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

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

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

}
```

Figura 12: Função "endGame()".

6 Testes Realizados

Para verificar a coerência dos nossos resultados, testamos o nosso código entidade por entidade com os binários dados pelo professor. No final quando já funcionavam as 3 entidades em separado, juntámo-las, através do make all, corremos 1000 vezes e como esperado verificamos que não houve nenhum deadlock.

A verificação da lógica esperada foi realizada linha por linha analisando se havia coerência com o exemplo do professor. Aqui vamos colocar 3 exemplos de resultados analisados, o primeiro caso quem formasse a equipa fossem dois *goalies*, a segunda caso fossem dois *players* e o último caso se fosse um *goalie* e um *player*.

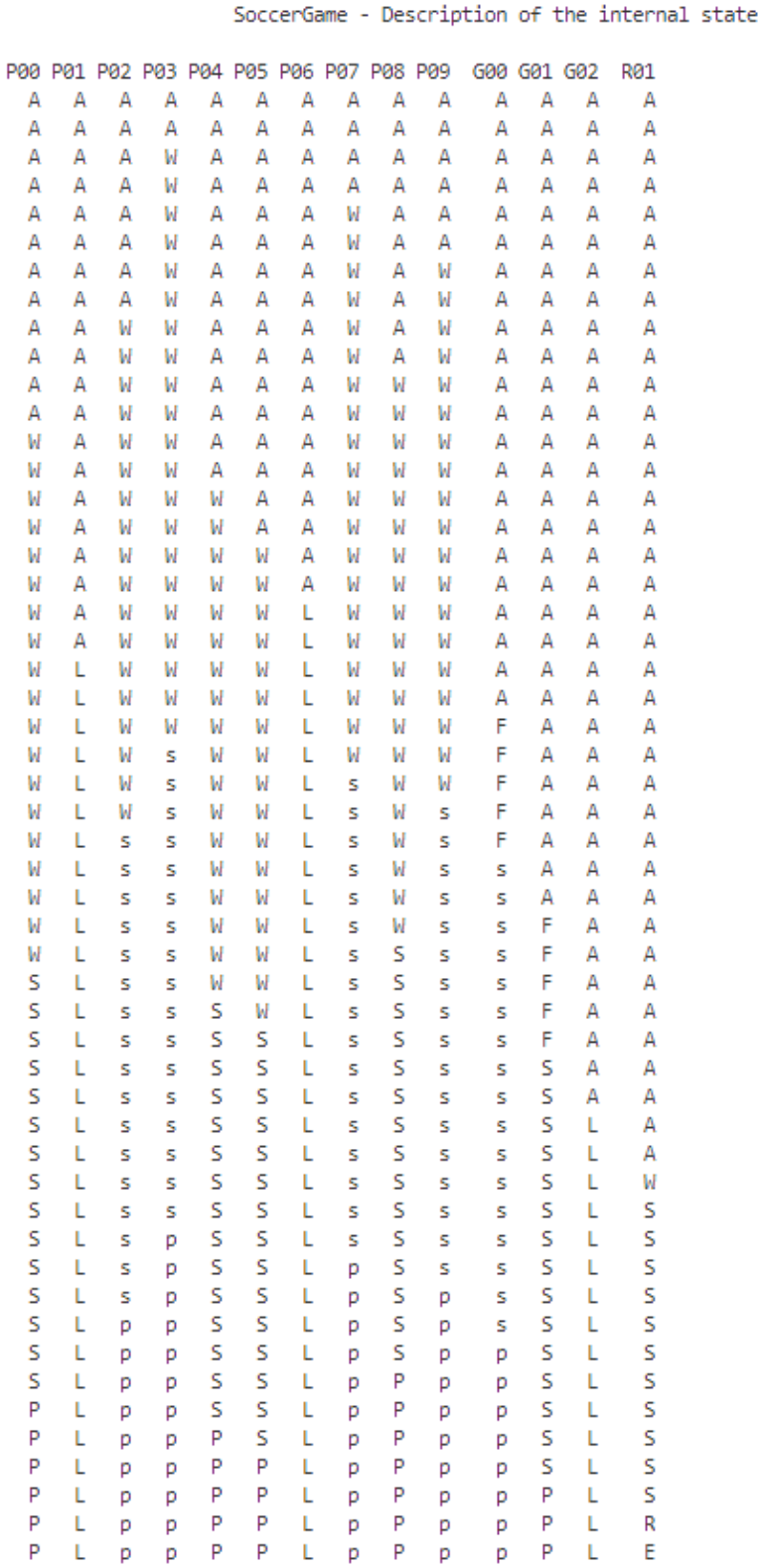

Figura 13: Dois *goalies* "F".

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	W	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	W	A
A	A	A	A	A	A	A	A	A	A	A	W	W	A
W	A	A	A	A	A	A	A	A	A	A	W	W	A
W	A	A	A	A	A	A	A	A	A	A	W	W	A
W	A	A	A	A	A	W	A	A	A	A	W	W	A
W	A	A	A	A	A	W	A	A	A	A	W	W	A
W	A	A	W	A	A	W	A	A	A	A	W	W	A
W	A	A	W	A	A	W	A	A	A	A	W	W	A
W	A	A	W	A	A	W	A	A	A	A	W	W	A
W	A	A	W	A	A	W	F	A	A	A	W	W	A
s	A	A	W	A	A	W	F	A	A	A	W	W	A
s	A	A	W	A	A	s	F	A	A	A	W	W	A
s	A	A	s	A	A	s	F	A	A	A	W	W	A
s	A	A	s	A	W	s	F	A	A	A	W	W	A
s	A	A	s	A	W	s	F	A	A	A	s	W	A
s	A	A	s	A	W	s	s	A	A	A	s	W	A
s	A	A	s	A	W	s	s	A	A	A	s	W	A
s	W	A	s	A	W	s	s	A	A	A	s	W	A
s	W	A	s	A	W	s	s	A	A	A	s	W	A
s	W	A	s	A	W	s	s	A	A	A	s	W	A
s	W	A	s	A	W	s	s	A	A	A	s	W	A
s	W	A	s	A	W	s	s	A	W	A	s	W	A
s	W	A	s	A	W	s	s	F	W	A	s	W	A
s	W	A	s	L	W	s	s	F	W	A	s	W	A
s	W	A	s	L	S	s	s	F	W	A	s	W	A
s	W	A	s	L	S	s	s	F	W	A	s	S	A
s	W	A	s	L	S	s	s	F	W	A	s	S	A
s	W	A	s	L	S	s	s	F	S	A	s	S	A
s	S	A	s	L	S	s	s	F	S	A	s	S	A
s	S	A	s	L	S	s	s	S	S	A	s	S	A
s	S	L	s	L	S	s	s	S	S	A	s	S	A
s	S	L	s	L	S	s	s	S	S	A	s	S	A
s	S	L	s	L	S	s	s	S	S	L	s	S	A
s	S	L	s	L	S	s	s	S	S	L	s	S	A
s	S	L	s	L	S	s	s	S	S	L	s	S	W
s	S	L	s	L	S	s	s	S	S	L	s	S	S
p	S	L	s	L	S	s	s	S	S	L	s	S	S
p	S	L	s	L	S	p	s	S	S	L	s	S	S
p	S	L	p	L	S	p	s	S	S	L	s	S	S
p	S	L	p	L	S	p	s	S	S	L	p	S	S
p	S	L	p	L	S	p	p	S	S	L	p	S	S
p	S	L	p	L	P	p	p	S	S	L	p	S	S
p	S	L	p	L	P	p	p	S	S	L	p	P	S
p	S	L	p	L	P	p	p	S	P	L	p	P	S
p	S	L	p	L	P	p	p	P	P	L	p	P	S
p	P	L	p	L	P	p	p	P	P	L	p	P	S
p	P	L	p	L	P	p	p	P	P	L	p	P	R
p	P	L	p	L	P	p	p	P	P	L	p	P	E

Figura 14: Dois *players* "F".

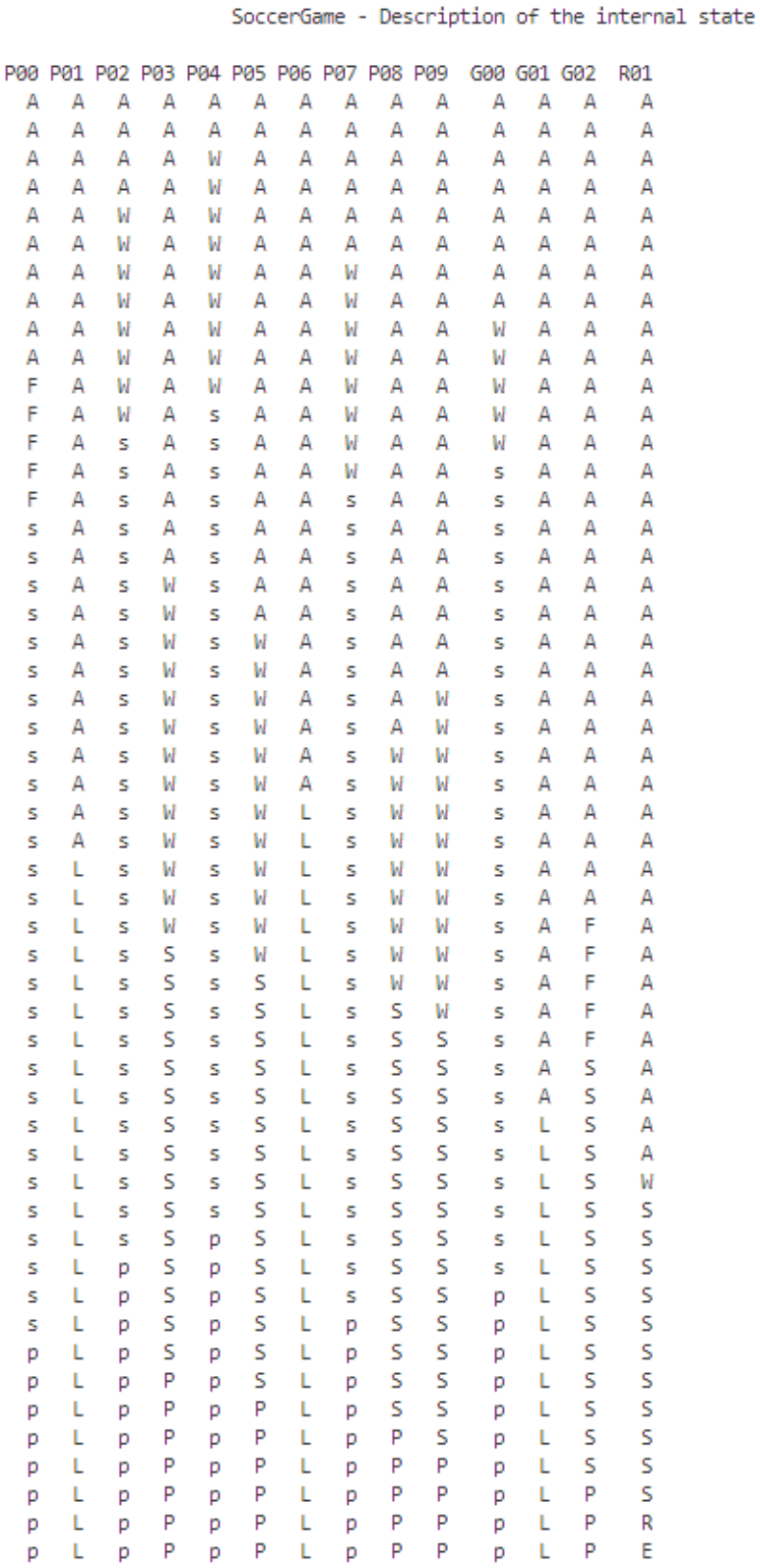

Figura 15: Um *player* e um *goalie* "F".

7 Conclusão

A realização deste projeto ampliou o nosso conhecimento teórico e prático sobre semáforos e ajudou-nos a compreender a importância que este possui na sincronização entre vários processos, garantindo que cada processo tem acesso aos recursos que necessita.