Sistemas Operativos

2020/2021

Trabalho Prático 2 - Simulação de Jogo de Futebol

Trabalho feito por:
Camila Fonseca - 97880
Diogo Monteiro - 97606

Índice

Índice **2**

Introdução **3**

Análise do problema **4**

Implementação da solução **7**

Goalie e Player 8

Arrive() 9

playerConstituteTeam() / goalieConstituteTeam() 10

Formar Equipa 10

waitReferee() 16

playUntilEnd() 17

Árbitro 18

arrive() 18

waitForTeams() 19

startGame() 19

play() 20

endGame() 21

Testes de funcionalidade **22**

Introdução

Para este trabalho prático foram fornecidos diversos ficheiros de código fonte C para um programa que simula o decorrer de um jogo de futebol. A simulação é feita através da criação de vários processos, que são análogos às várias entidades presentes no jogo - os jogadores e o árbitro. Cada entidade tem um conjunto de estados entre os quais pode transitar, estando apenas um dos estados ativos a qualquer momento.

Para além do código fonte, são também dados ficheiros binários pré-compilados que exemplificam o funcionamento pretendido do programa, e que podem também ser usados para testar as diferentes entidades.

O objetivo deste trabalho prático é completar o código fonte das três entidades (os ficheiros ‘semSharedMemPlayer.c’, ‘semSharedMemGoalie.c’, e ‘semSharedMemReferee.c’) de forma a que estes atualizem devidamente o seu estado no decorrer do jogo, e que o seu processo bloqueie quando devem esperar por outro. Para tal, devem ser usados as estruturas e semáforos já definidos no código dado.

No final do trabalho, os *scripts* completados devem produzir resultados semelhantes aos binários pré-compilados, sem ocorrer nenhum *deadlock* na execução.

Análise do problema

Começamos por olhar para as constantes definidas no ficheiro 'probConst.h', entre as quais estão definidos os diferentes estados dos jogadores e do árbitro. De seguida, com o auxílio dos binários pré-compilados, identificamos quais eram as possíveis transições entre eles, para ter uma ideia de como os estados das entidades devem evoluir ao longo do jogo. Para a fácil visualização das transições de estados, criamos um diagrama para a sua representação.

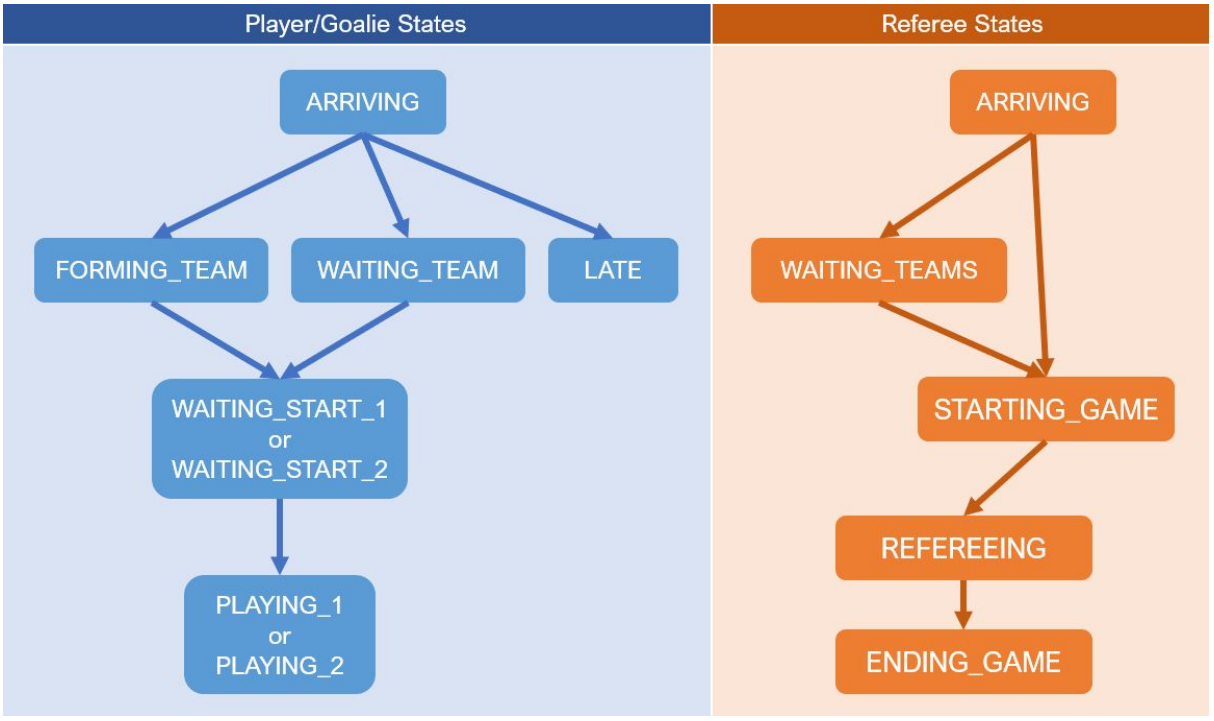


Diagrama de estados para os *players*/*goalies* e para o *referee*

Todas as entidades começam no estado ARRIVING, representativo de quando a entidade ainda não chegou, e permanecem nesse estado por uma duração aleatória. Quando uma entidade chega, dependendo dos estados das outras entidades, há diferentes transições de estado possíveis.

No caso de um *player* ou *goalie*, ele pode transitar para FORMING_TEAM se existirem jogadores livres suficientes quando ele chega, senão ele muda para WAITING_TEAM, dado que ainda falta formar alguma equipa. Caso ele chegue após ambas as equipas serem formadas, transita para LATE.

Quanto ao referee, ele muda para o estado WAITING_TEAMS ao chegar se as duas equipas ainda não foram formadas, mas pode também avançar diretamente para STARTING_GAME se ele chegar após a formação das equipas.

Após a análise dos estados das entidades, dirigimos a nossa atenção aos semáforos definidos no ficheiro 'sharedDataSync.h', que irão governar o acesso à memória partilhada e serão a ferramenta principal controlar as interações entre as entidades e garantir o seu devido funcionamento.

Para orientar o nosso trabalho, procuramos perceber a função de cada semáforo e como este seria usado por cada entidade. Identificamos as entidades que deverão fazer operações 'down' e 'up', a quantidade de operações por entidade, e a função no seu código no qual as operações serão realizadas.

Segue então uma análise de cada semáforo:

playersWaitTeam:
Bloqueia *players* que ainda não estão numa equipa, e não são capazes de formar uma com os *players* e *goalies* disponíveis no momento. O bloqueio será levantado pelo *player* ou *goalie* que formar a equipa.

ENTIDADE DOWN (block):	player
FUNÇÃO down:	playerConstituteTeam()
Nº downs:	1
ENTIDADE UP (unblock):	player ou goalie (que formar equipa)
FUNÇÃO up:	<entidade>ConstituteTeam()
Nº de ups:	NUMTEAMPLAYERS (-1 se for player a formar) (3 ou 4)

goaliesWaitTeam:
Bloqueia *goalies* que ainda não estão numa equipa, e não são capazes de formar uma com os *players* disponíveis no momento. O bloqueio será levantado pelo *player* que formar a equipa.

ENTIDADE DOWN (block):	goalie
FUNÇÃO down:	goalieConstituteTeam()
Nº downs:	1
ENTIDADE UP (unblock):	player (que formar equipa)
FUNÇÃO up:	playerConstituteTeam()
Nº de ups:	NUMTEAMGOALIES (1)

playersWaitReferee:
Bloqueia *players* e *goalies* que já estão numa equipa, forçando-os a esperar que o *referee* comece o jogo.

ENTIDADE DOWN (block):	player/goalie
FUNÇÃO down:	waitReferee()
Nº downs:	1
ENTIDADE UP (unblock):	referee
FUNÇÃO up:	startGame()
Nº de ups:	(NUMTEAMPLAYERS + NUMTEAMGOALIES)*2 (10)

playersWaitEnd:

Bloqueia *players* e *goalies* enquanto estão a jogar, forçando-os a esperar que o *referee* assinale o fim do jogo.

ENTIDADE DOWN (block):	player/goalie
FUNÇÃO down:	playUntilEnd()
Nº downs:	1
ENTIDADE UP (unblock):	referee
FUNÇÃO up:	endGame()
Nº de ups:	(NUMTEAMPLAYERS + NUMTEAMGOALIES)*2 (10)

refereeWaitTeams:

Bloqueia o *referee* enquanto as duas equipas não estiverem formadas. Será desbloqueado pelos *players* ou *goalies* que formarem as equipas.

ENTIDADE DOWN (block):	referee
FUNÇÃO down:	waitForTeams()
Nº downs:	2
ENTIDADE UP (unblock):	player/goalie (que formarem equipa)
FUNÇÃO up:	<entidade>ConstituteTeam()
Nº de ups:	1

playerRegistered:

Bloqueia o player ou goalie que estiver a formar a equipa, de forma a esperar que os membros da equipa confirmem o seu registo.

ENTIDADE DOWN (block):	player/goalie (que formar a equipa)
FUNÇÃO down:	<entidade>ConstituteTeam()
Nº downs:	4
ENTIDADE UP (unblock):	player/goalie
FUNÇÃO up:	<entidade>ConstituteTeam()
Nº de ups:	1

Implementação da solução

Para começar, e ajudar a visualizar o problema, foi elaborada a análise dos semáforos incluída acima.

Como o funcionamento dos Players e dos Goalies é muito similar (Sendo que apenas diferem nas variáveis que manipulam e na condição para formar equipa), optou-se por primeiro elaborar o código para os Goalies e, depois de testado com os binários fornecidos, copiar este para o Player, com as devidas alterações.

A implementação do código do Goalie foi bastante direta, com o suporte da tabela dos semáforos.

Optou-se por se implementar estado a estado, apenas passando para a fase seguinte após se obter comportamento correto na anterior. Foi tido bastante cuidado com apenas efetuar 'Down' fora da zona crítica, (Exceto numa situação específica (semáforo playersRegistered) que não se conseguiu implementar de outro modo.) portanto os erros inesperados que surgiram foram relativamente fáceis de identificar a causa e corrigir.

Goalie e Player

NOTA: Para simplicidade textual, "jogadores" refere-se a ambos players e goalies, sendo então usado player ou goalie para se referir aos tipos individualmente.

O Goalie é virtualmente idêntico ao Player no seu comportamento, exceto no que toca à verificação de se existem jogadores livres suficientes para formar equipa (No caso do player, procura por três outros players e um goalie, enquanto que o Goalie procura apenas por quatro players.) e nas variáveis/semáforos que altera.

Os Goalies e Players têm um ciclo de vida bastante direto. As funções arrive(), waitReferee() e playUntilEnd() são simples e praticamente iguais para ambos os tipos de jogador.

A função playerConstituteTeam()/goalieConstituteTeam() é onde se encontra a maioria do código elaborado, e onde players e goalies diferem mais.

```
/* simulation of the life cycle of the goalie */
  arrive(n);
  if ((team = goalieConstituteTeam(n)) != 0)
     // retorna 0 se estiver LATE, 1 se escolheu team 1
     // ou 2 se escolheu team 2
  {
      waitReferee(n, team);
      playUntilEnd(n, team);
  }
```

```
/* simulation of the life cycle of the player */
  arrive(n);
  if ((team = playerConstituteTeam(n)) != 0)
  {
      waitReferee(n, team);
      playUntilEnd(n, team);
  }
```

Arrive()

Esta função simplesmente serve para espaçar e randomizar as chegadas dos jogadores, criando alguma imprevisibilidade e garantindo que o jogo não se desenrola exatamente do mesmo modo de cada vez que é corrido.

Como o jogador aqui altera o seu próprio estado, é necessário efetuar um Down e um Up no mutex para aceder à memória partilhada sem risco de haver uma condição de corrida.

Após isto, fica à espera durante um tempo randomizado.

```
      static void arrive(int id)
      {
         if (semDown(semgid, sh->mutex) == -1)
         { /* enter critical region */
             perror("error on the down operation for semaphore access
(GL)");
             exit(EXIT_FAILURE);
         }
         sh->fSt.st.goalieStat[id] = ARRIVING;
         saveState(nFic, &sh->fSt);

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

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

playerConstituteTeam() / goalieConstituteTeam()

Nesta função é onde se define se o jogador chega atrasado, se forma equipa, ou se fica à espera que outros cheguem e depois entra numa equipa. É aqui também que se define qual a equipa que formam/integram.

O valor de retorno (a int ret) nesta função deve ser 0, se o jogador chegar atrasado, 1, se ficar na equipa um, ou 2 se ficar na equipa 2.

Para começar, entra-se na região crítica (Efetuando semDown no mutex) e incrementa-se a variável que guarda o número de goalies que já chegaram. Se este valor for superior ao dobro do número de jogadores (do tipo respectivo) necessários por equipa (o que quer dizer que já chegaram jogadores suficientes para formar duas equipas, sendo todos os que chegarem depois considerados atrasados, ficando com o estado LATE.

De modo contrário, verifica-se se existem jogadores presentes (e livres) suficientes para formar uma equipa.

- Para o goalie, é verificado se há 4 players livres.
- Para o player, é verificado se há 3 outros players e um goalie livre

Formar Equipa

Formar equipa é feito inteiramente dentro da zona crítica, apesar de se efectuar semDowns. Para tal, muda-se o estado para FORMING_TEAM, e diminui-se o número de jogadores livres.

- Para os goalies, diminui-se em NUMTEAMPLAYERS o número de players livres
- Para os players, diminui-se em NUMTEAMPLAYERS - 1 o número de players livres e em NUMTEAMGOALIES o número de goalies livres.

Nota: Como não se aumenta o número de jogadores livres logo ao chegar, não se decrementa o número de jogadores livres a contar com o próprio que está a formar equipa.

De seguida, quem está a formar equipa efetua:

- semUp 4 (NUMTEAMPLAYERS) vezes em playersWaitTeam, se for um goalie
- semUp 3 (NUMTEAMPLAYERS - 1) vezes em playersWaitTeam e 1 (NUMTEAMGOALIES) vez em goaliesWaitTeam, se for um player

e de seguida 4 semDown em playerRegistered. Devido a este down, o jogador a formar equipa fica à espera que todos os jogadores entrem na equipa, muda o valor de retorno da função para o atual teamId e de seguida incrementa este, para a outra equipa ser formada.

```
static int goalieConstituteTeam(int id)
{
   int ret = 0;
   if (semDown(semgid, sh->mutex) == -1)
   { /* enter critical region */
       perror("error on the down operation for semaphore access (GL)");
       exit(EXIT_FAILURE);
   }
   sh->fSt.goaliesArrived++;

   //verificar se há 4 jogadores livres
   //para os players, verificar se há mais 3 jogadores livres e um goalie
   if (sh->fSt.goaliesArrived > (NUMTEAMGOALIES * 2))
   {
       sh->fSt.st.goalieStat[id] = LATE;
       saveState(nFic, &sh->fSt);
   }
   else
   {

       if (sh->fSt.playersFree >= NUMTEAMPLAYERS)
       {
           sh->fSt.st.goalieStat[id] = FORMING_TEAM;
           sh->fSt.playersFree-=NUMTEAMPLAYERS;
           saveState(nFic, &sh->fSt);


           //goalie fica à espera que os jogadores entrem na equipa
           //down dentro da região crítica mas é necessário :(
           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);
               }
           }

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

```
            }

        }

        ret = sh->fSt.teamId;
        sh->fSt.teamId++; //team formed, now on to other team to be formed
        if (semUp(semgid, sh->refereeWaitTeams) == -1)
        {
            perror("error on the up operation for semaphore access (GL)");
            exit(EXIT_FAILURE);
        }
    }
```

Se nenhum dos casos anteriores se verificar, o jogador muda o seu estado para WAITING_TEAM, incrementa a sua variável respectiva (goaliesFree para goalies, playersFree para players) e sai da zona crítica, efetuando um semUp no mutex.

De seguida, se o seu estado for WAITING_TEAM, o jogador efetua um semDown no semáforo respectivo (goaliesWaitTeam se for goalie, playersWaitTeam se for player) e, depois de este semáforo ser levantado pelo jogador que forma a equipa, muda o seu valor de retorno para o valor atual de teamId.

Para terminar a 'inscrição' na equipa, efetua um semUp em playersRegistered, que tal como visto antes, irá eventualmente libertar o jogador a formar equipa.

```
    else
    {
        sh->fSt.st.goalieStat[id] = WAITING_TEAM;
        sh->fSt.goaliesFree++;
        saveState(nFic, &sh->fSt);
    }
}
if (semUp(semgid, sh->mutex) == -1)
{ /* exit critical region */
    perror("error on the up operation for semaphore access (GL)");
    exit(EXIT_FAILURE);
}

if (sh->fSt.st.goalieStat[id] == WAITING_TEAM)
{
    if (semDown(semgid, sh->goaliesWaitTeam) == -1)
    {
        perror("error on the down operation for semaphore access (GL)");
        exit(EXIT_FAILURE);
    }
```

```
        ret = sh->fSt.teamId;

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

    return ret;
}
```

```
static int playerConstituteTeam(int id)
{
   int ret = 0;
   if (semDown(semgid, sh->mutex) == -1)
   { /* enter critical region */
       perror("error on the down operation for semaphore access (PL)");
       exit(EXIT_FAILURE);
   }

   sh->fSt.playersArrived++;

   if (sh->fSt.playersArrived > NUMTEAMPLAYERS * 2)
   {
       sh->fSt.st.playerStat[id] = LATE;
       saveState(nFic, &sh->fSt);
   }
   else
   {

       if (sh->fSt.playersFree >= NUMTEAMPLAYERS - 1 && sh->fSt.goaliesFree
>= NUMTEAMGOALIES)
       {

           sh->fSt.st.playerStat[id] = FORMING_TEAM;

           sh->fSt.goaliesFree -= NUMTEAMGOALIES;
           sh->fSt.playersFree -= (NUMTEAMPLAYERS - 1); /* não se subtrai a
si  próprio */
           saveState(nFic, &sh->fSt);

           for (int i = 0; i < NUMTEAMPLAYERS - 1; i++)
           {
               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);
            }

           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++;
            if (semUp(semgid, sh->refereeWaitTeams) == -1)
            {
                perror("error on the up operation for semaphore access (PL)");
                exit(EXIT_FAILURE);
            }
        }
        else
        {
            sh->fSt.st.playerStat[id] = WAITING_TEAM;
            sh->fSt.playersFree++;
            saveState(nFic, &sh->fSt);
        }
    }
    if (semUp(semgid, sh->mutex) == -1)
    { /* exit critical region */
        perror("error on the up operation for semaphore access (PL)");
        exit(EXIT_FAILURE);
    }

    if (sh->fSt.st.playerStat[id] == WAITING_TEAM)
    {

        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);
        }
    }
    return ret;
```

```
}
```

waitReferee()

Esta função serve para sincronizar os jogadores e o árbito. Após entrar na zona crítica, consoante o valor guardado na variável team, o jogador muda o seu estado para WAITING_START_1, se 2, muda para WAITING_START_2. Depois de sair da zona crítica (para evitar deadlocks), é efetuado um Down em playersWaitReferee, para os jogadores esperarem que o árbitro dê início à partida.

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

   if (team == 1)
   {
       sh->fSt.st.playerStat[id] = WAITING_START_1;
       saveState(nFic, &sh->fSt);
   }
   else
   {
       sh->fSt.st.playerStat[id] = WAITING_START_2;
       saveState(nFic, &sh->fSt);
   }
   if (semUp(semgid, sh->mutex) == -1)
   { /* exit critical region */
       perror("error on the up operation for semaphore access (PL)");
       exit(EXIT_FAILURE);
   }

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

playUntilEnd()

Nesta função, os jogadores simplesmente entram na zona crítica, mudam o seu estado para PLAYING_1 ou PLAYING_2 consoante a equipa, e “jogam” até o árbito terminar o jogo, através do semaforo playersWaitEnd.

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

   if (team == 1)
   {
       sh->fSt.st.goalieStat[id] = PLAYING_1;
       saveState(nFic, &sh->fSt);
   }
   else
   {
       sh->fSt.st.goalieStat[id] = PLAYING_2;
       saveState(nFic, &sh->fSt);
   }

   if (semUp(semgid, sh->mutex) == -1)
   { /* exit critical region */
       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);
   }
}
```

Árbitro

O árbitro é mais simples do que os jogadores na sua implementação, e tem o seguinte ciclo de vida:

```
      /* simulation of the life cycle of the referee */
         arrive();
         waitForTeams();
         startGame();
         play();
         endGame();
```

arrive()

A função arrive() do árbitro é idêntica à dos jogadores.

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

   sh->fSt.st.refereeStat = ARRIVING;
   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);}
```

waitForTeams()

Após entrar na região crítica, se teamId estiver com valor inferior a 3 (porque as equipas ainda não foram formadas) o árbitro muda o seu estado para WAITING_TEAMS, ficando à espera que as equipas sejam ambas formadas através de dois semDown em refereeWaitTeams.

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

   if (sh->fSt.teamId < 3) {
       sh->fSt.st.refereeStat = WAITING_TEAMS;
       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);
   }

   for (int i = 0; i < 2; i++) {       // 2 downs - 2 equipas
       if (semDown (semgid, sh->refereeWaitTeams) == -1) {
           perror ("error on the down operation for semaphore access (RF)");
           exit (EXIT_FAILURE);
       }
   }
}
```

startGame()

Nesta função, após entrar na região crítica, o árbitro muda o seu próprio estado para STARTING_GAME e volta a saír, efetuando depois 8 ((NUMTEAMGOALIES + NUMTEAMPLAYERS) *2, o que corresponde ao número total de jogadores) semUp em playersWaitReferee, permitindo assim os jogadores, que estavam à espera do árbitro, contrinuar.

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

```
    }

    sh->fSt.st.refereeStat = STARTING_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);
    }

    for (int i = 0; i < (NUMTEAMGOALIES+NUMTEAMPLAYERS)*2; i++) {
        if (semUp (semgid, sh->playersWaitReferee) == -1) {
            perror ("error on the up operation for semaphore access (RF)");
            exit (EXIT_FAILURE);
        }
    }
}
```

play()

Esta é mais uma função simples, o áribtro entra na zona crítica apenas para mudar o seu estado, e fica à espera, para dar tempo para o jogo se desenrolar.

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

    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);
}
```

endGame()

Para terminar, o árbitro volta a entrar na zona crítica, altera o seu estado para ENDING_GAME, e desbloqueia todos os outros jogadores que estão a jogar, com um semUp em playersWaitEnd. Assim efetivamente termina o jogo.

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

   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);
   }

   for (int i = 0; i < (NUMTEAMGOALIES+NUMTEAMPLAYERS)*2; i++) {
       if (semUp (semgid, sh->playersWaitEnd) == -1) {
           perror ("error on the up operation for semaphore access (RF)");
           exit (EXIT_FAILURE);
       }
   }
}
```

Testes de funcionalidade

Ao longo da realização do trabalho, o funcionamento de cada entidade foi testado, sendo compilado apenas o código fonte da entidade em questão e usando os binários pré-compilados das outras entidades. Isto é facilitado pela configuração feita no 'Makefile', que permite compilar diferentes partes do programa dados diferentes comandos, como 'make all_bin' e 'make rf', por exemplo.

Para averiguar se os resultados obtidos estavam corretos, tivemos como base vários exemplos gerados pelos binários fornecidos, assim como o diagrama de estados.

Estando a implementação das três entidades completa, procedemos aos testes finais correndo o script bash 'run.sh' que foi disponibilizado. Verificamos que o script correu com sucesso, executando a simulação mil vezes. É assim evidente que nenhum *deadlock* ocorreu.

Quanto à correção dos resultados, fomos analisando as simulações obtidas durante os testes de cada entidade individual para certificar o seu devido funcionamento, e no final olhamos para 10 das simulações feitas com as três entidades. Nenhuma delas apresentou um erro na lógica do jogo e todas as transições de estados obedeceram às suas condições.

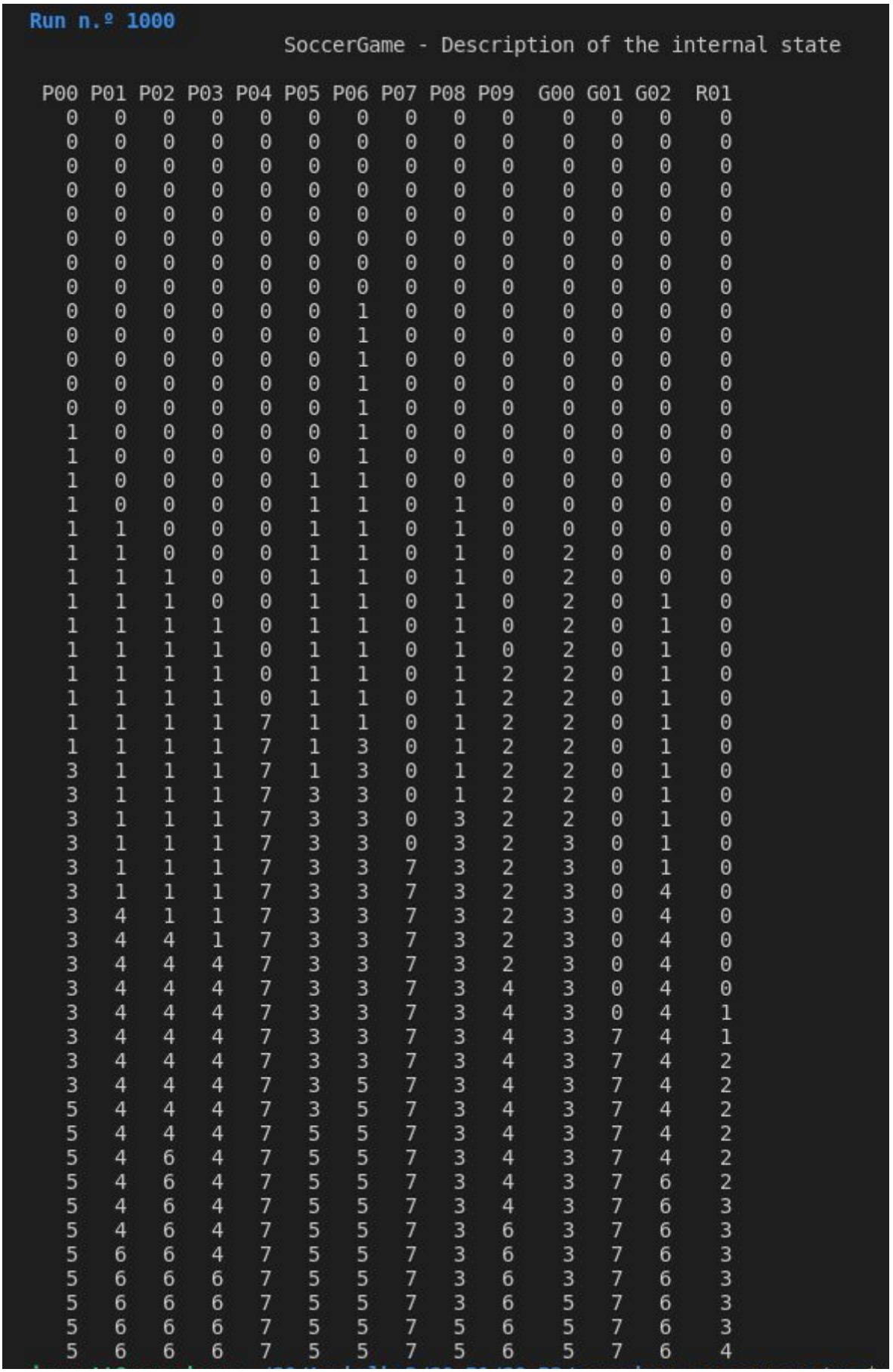
Em baixo seguem alguns exemplos de simulações, feitas para entidades individuais e para as três em conjunto.

./run.sh após make all

```
Run n.º 1000
                    SoccerGame - Description of the internal state

 P00 P01 P02 P03 P04 P05 P06 P07 P08 P09  G00 G01 G02  R01
   0   0   0   0   0   0   0   0   0   0    0   0   0    0
   0   0   0   0   0   0   0   0   0   0    0   0   0    0
   0   0   0   0   0   0   0   0   0   0    0   0   0    0
   0   0   0   0   0   0   0   0   0   0    0   0   0    0
   0   0   0   0   0   0   0   0   0   0    0   0   0    0
   0   0   0   0   0   0   0   0   0   0    0   0   0    0
   0   0   0   0   0   0   0   0   0   0    0   0   0    0
   0   0   0   0   0   0   0   0   0   0    0   0   0    0
   0   0   1   0   0   0   0   0   0   0    0   0   0    0
   0   0   1   0   0   0   0   0   0   0    0   0   0    0
   0   0   1   1   0   0   0   0   0   0    0   0   0    0
   0   0   1   1   0   0   0   0   0   0    0   0   0    0
   0   0   1   1   0   1   0   0   0   0    0   0   0    0
   0   0   1   1   0   1   0   0   0   0    0   0   0    0
   0   0   1   1   0   1   0   0   0   0    0   0   0    0
   0   0   1   1   1   1   0   0   0   0    0   0   0    0
   0   0   1   1   1   1   0   0   0   0    0   0   0    0
   0   0   1   1   1   1   0   0   0   0    0   0   0    0
   0   0   1   1   1   1   0   0   0   0    0   0   0    0
   1   0   1   1   1   1   0   0   0   0    0   0   0    0
   1   0   1   1   1   1   1   0   0   0    0   0   0    0
   1   0   1   1   1   1   1   1   0   0    0   0   0    0
   1   0   1   1   1   1   1   1   0   1    0   0   0    0
   1   0   1   1   1   1   1   1   0   1    2   0   0    0
   1   7   1   1   1   1   1   1   0   1    2   0   0    0
   1   7   1   1   1   1   1   1   0   1    2   2   0    0
   1   7   1   1   1   1   1   1   0   1    2   2   7    0
   1   7   1   1   1   1   1   1   7   1    2   2   7    0
   1   7   3   1   1   1   1   1   7   1    2   2   7    0
   1   7   3   3   1   1   1   1   7   1    2   2   7    0
   1   7   3   3   1   3   1   1   7   1    2   2   7    0
   1   7   3   3   3   3   1   1   7   1    2   2   7    0
   1   7   3   3   3   3   1   1   7   1    3   2   7    0
   1   7   3   3   3   3   1   4   7   1    3   2   7    0
   1   7   3   3   3   3   4   4   7   1    3   2   7    0
   1   7   3   3   3   3   4   4   7   4    3   2   7    0
   4   7   3   3   3   3   4   4   7   4    3   2   7    0
   4   7   3   3   3   3   4   4   7   4    3   4   7    0
   4   7   3   3   3   3   4   4   7   4    3   4   7    2
   4   7   5   3   3   3   4   4   7   4    3   4   7    2
   4   7   5   5   3   3   4   4   7   4    3   4   7    2
   4   7   5   5   3   5   4   4   7   4    3   4   7    2
   4   7   5   5   5   5   4   4   7   4    3   4   7    2
   4   7   5   5   5   5   6   4   7   4    3   4   7    2
   4   7   5   5   5   5   6   6   7   4    3   4   7    2
   4   7   5   5   5   5   6   6   7   4    3   4   7    3
   4   7   5   5   5   5   6   6   7   6    3   4   7    3
   6   7   5   5   5   5   6   6   7   6    3   4   7    3
   6   7   5   5   5   5   6   6   7   6    3   6   7    3
   6   7   5   5   5   5   6   6   7   6    5   6   7    3
   6   7   5   5   5   5   6   6   7   6    5   6   7    4
```

./run.sh após make gl



./run após make pl

```
Run n.º 1000
                    SoccerGame - Description of the internal state

 P00 P01 P02 P03 P04 P05 P06 P07 P08 P09  G00 G01 G02  R01
   0   0   0   0   0   0   0   0   0   0    0   0   0    0
   0   0   0   0   0   0   0   0   0   0    0   0   0    0
   0   0   0   0   0   0   0   0   0   0    0   0   0    0
   0   0   0   0   0   0   0   0   0   0    0   0   0    0
   0   0   0   0   0   0   0   0   0   0    0   0   0    0
   0   0   0   0   0   0   0   0   0   0    0   0   0    0
   0   0   0   0   0   0   0   0   0   0    0   0   0    0
   0   0   0   0   0   0   0   0   0   0    0   0   0    0
   0   0   1   0   0   0   0   0   0   0    0   0   0    0
   0   1   1   0   0   0   0   0   0   0    0   0   0    0
   0   1   1   0   0   0   0   0   0   0    0   0   0    0
   0   1   1   0   0   0   0   0   0   0    0   0   0    0
   0   1   1   0   0   0   0   0   0   0    0   0   0    0
   0   1   1   1   0   0   0   0   0   0    0   0   0    0
   0   1   1   1   0   0   0   0   0   0    0   0   0    0
   0   1   1   1   0   0   0   0   0   0    0   0   0    0
   0   1   1   1   0   1   0   0   0   0    0   0   0    0
   0   1   1   1   0   1   0   1   0   0    0   0   0    0
   0   1   1   1   0   1   1   1   0   0    0   0   0    0
   0   1   1   1   0   1   1   1   1   0    0   0   0    0
   0   1   1   1   0   1   1   1   1   0    0   0   0    1
   0   1   1   1   1   1   1   1   1   0    0   0   0    1
   0   1   1   1   1   1   1   1   1   0    2   0   0    1
   0   1   3   1   1   1   1   1   1   0    2   0   0    1
   0   3   3   1   1   1   1   1   1   0    2   0   0    1
   7   3   3   1   1   1   1   1   1   0    2   0   0    1
   7   3   3   3   1   1   1   1   1   0    2   0   0    1
   7   3   3   3   1   3   1   1   1   0    2   0   0    1
   7   3   3   3   1   3   1   1   1   0    3   0   0    1
   7   3   3   3   1   3   1   1   1   7    3   0   0    1
   7   3   3   3   1   3   1   1   1   7    3   0   0    1
   7   3   3   3   1   3   1   1   1   7    3   2   0    1
   7   3   3   3   1   3   1   4   1   7    3   2   0    1
   7   3   3   3   1   3   4   4   1   7    3   2   0    1
   7   3   3   3   1   3   4   4   4   7    3   2   0    1
   7   3   3   3   4   3   4   4   4   7    3   2   0    1
   7   3   3   3   4   3   4   4   4   7    3   4   0    1
   7   3   3   3   4   3   4   4   4   7    3   4   0    2
   7   3   5   3   4   3   4   4   4   7    3   4   0    2
   7   5   5   3   4   3   4   4   4   7    3   4   0    2
   7   5   5   5   4   3   4   4   4   7    3   4   0    2
   7   5   5   5   4   5   4   4   4   7    3   4   0    2
   7   5   5   5   4   5   4   4   4   7    5   4   0    2
   7   5   5   5   4   5   4   6   4   7    5   4   0    2
   7   5   5   5   4   5   6   6   4   7    5   4   0    2
   7   5   5   5   4   5   6   6   6   7    5   4   0    2
   7   5   5   5   4   5   6   6   6   7    5   4   0    3
   7   5   5   5   4   5   6   6   6   7    5   6   0    3
   7   5   5   5   6   5   6   6   6   7    5   6   0    3
   7   5   5   5   6   5   6   6   6   7    5   6   0    3
   7   5   5   5   6   5   6   6   6   7    5   6   7    3
   7   5   5   5   6   5   6   6   6   7    5   6   7    4
```

./run.sh após make rf

```
Run n.º 1000
                    SoccerGame - Description of the internal state

 P00 P01 P02 P03 P04 P05 P06 P07 P08 P09  G00 G01 G02  R01
   0   0   0   0   0   0   0   0   0   0    0   0   0    0
   0   0   0   0   0   0   0   0   0   0    0   0   0    0
   0   0   0   0   0   0   0   0   0   0    0   0   0    0
   0   0   0   0   0   0   0   0   0   0    0   0   0    0
   0   0   0   0   0   0   0   0   0   0    0   0   0    0
   0   0   0   0   0   0   0   0   0   0    0   0   0    0
   0   0   0   0   0   0   0   0   0   0    0   0   0    0
   0   0   0   0   0   0   0   0   0   0    0   0   0    0
   0   0   0   0   0   0   0   0   0   0    0   0   0    0
   0   0   0   0   0   1   0   0   0   0    0   0   0    0
   0   0   0   0   1   1   0   0   0   0    0   0   0    0
   0   0   0   0   1   1   0   0   0   0    0   0   0    0
   0   0   0   1   1   1   0   0   0   0    0   0   0    0
   0   0   0   1   1   1   0   0   0   0    0   0   0    0
   0   0   0   1   1   1   1   0   0   0    0   0   0    0
   0   0   0   1   1   1   1   0   0   0    0   0   0    0
   0   0   0   1   1   1   1   0   0   0    0   0   0    0
   0   1   0   1   1   1   1   0   0   0    0   0   0    0
   0   1   0   1   1   1   1   0   0   0    0   0   0    0
   0   1   0   1   1   1   1   1   0   0    0   0   0    0
   0   1   0   1   1   1   1   1   0   0    0   0   0    0
   0   1   0   1   1   1   1   1   0   1    0   0   0    0
   1   1   0   1   1   1   1   1   0   1    0   0   0    0
   1   1   0   1   1   1   1   1   0   1    0   2   0    0
   1   1   0   1   1   1   1   1   7   1    0   2   0    0
   1   1   0   1   1   1   1   1   7   1    2   2   0    0
   1   1   0   1   1   3   1   1   7   1    2   2   0    0
   1   1   0   1   3   3   1   1   7   1    2   2   0    0
   1   1   0   3   3   3   1   1   7   1    2   2   0    0
   1   1   0   3   3   3   3   1   7   1    2   2   0    0
   1   1   0   3   3   3   3   1   7   1    2   3   0    0
   1   1   7   3   3   3   3   1   7   1    2   3   0    0
   1   4   7   3   3   3   3   1   7   1    2   3   0    0
   1   4   7   3   3   3   3   4   7   1    2   3   0    0
   1   4   7   3   3   3   3   4   7   4    2   3   0    0
   4   4   7   3   3   3   3   4   7   4    2   3   0    0
   4   4   7   3   3   3   3   4   7   4    4   3   0    0
   4   4   7   3   3   3   3   4   7   4    4   3   0    2
   4   4   7   3   3   3   3   4   7   4    4   3   7    2
   4   4   7   3   3   5   3   4   7   4    4   3   7    2
   4   4   7   3   5   5   3   4   7   4    4   3   7    2
   4   4   7   3   5   5   5   4   7   4    4   3   7    2
   4   4   7   5   5   5   5   4   7   4    4   3   7    2
   4   4   7   5   5   5   5   4   7   4    4   5   7    2
   4   6   7   5   5   5   5   4   7   4    4   5   7    2
   4   6   7   5   5   5   5   6   7   4    4   5   7    2
   4   6   7   5   5   5   5   6   7   6    4   5   7    2
   4   6   7   5   5   5   5   6   7   6    4   5   7    3
   6   6   7   5   5   5   5   6   7   6    4   5   7    3
   6   6   7   5   5   5   5   6   7   6    6   5   7    3
   6   6   7   5   5   5   5   6   7   6    6   5   7    4
jnrvatt@somewhere:~/SO/trabalho2/SO_P1/SO_P2/semaphore_soccergame/run$
```