universidade de aveiro
deti
departamento de eletrónica,
telecomunicações e informática

Licenciatura em Engenharia Informática
2º Ano
Ano Letivo 2024/2025

Trabalho 2:
Simulação de jogo de futebol

RELATÓRIO

Janeiro 2025

Sistemas Operativos

Alexandre Andrade 119279
Miguel Neto 119302

Conteúdo

1 Introdução

Frequentemente em programação concorrente deparamo-nos com a necessidade de diversos programas, processos, *threads*, etc., trocarem informação entre si em prol do desempenho de algum tipo de tarefa, por exemplo melhorando o tempo que tal requer. Inevitavelmente, este conceito é análogo a diferentes exemplos, como o de comunicação entre "indivíduos". Os semáforos são ideais para estes casos, já que servem de meio de comunicação entre estes, que são na realidade processos e *threads*. Associar estes semáforos aos semáforos de trânsito pode não ser 100% correto, pois os semáforos que serão abordados e utilizados são lançados como mensagens entre estes processos e *threads* (*Normalmente* os carros não têm dizem aos outros quando é que o sinal mudou!).

Da perspetiva de um programador, são variáveis implementadas já com um grau de abstração (providenciado para a execução deste trabalho) suficiente para o conceito base de um semáforo: além das operações de inicialização e término da vida de um semáforo, possui as operações **P** e **V**, frequentemente ditas *wait* e *signal*. Respetivamente às últimas 2 operações, **P** decrementa o valor do semáforo, e se já possui o valor zero, **aguarda** até poder decrementar, e **V** incrementa a variável, ou seja, **liberta** o semáforo para algum outro processo, quer já esteja a aguardar ou não, posso utilizá-lo para aceder a uma determinada secção do programa de forma segura.

Para este trabalho considera-se um grupo de amigos que pretende juntar-se para jogar uma partida de futebol. Cada amigo tem os seus papéis já definidos *a priori* **para cada equipa**:

4 jogadores de campo (***Players***) e
1 guarda-redes (***Goalies***),

com então **10** jogadores no total, no máximo, na partida. Além destes 10 jogadores, temos

1 amigo que será o árbitro (***Referee***),

responsável por coordenar o jogo: iniciar a partida, terminá-la, etc., assim como as condições envolventes. No entanto é possível chegarem amigos em excesso para o jogo, sendo então avisados que chegaram atrasados e que não vão entrar jogo.

O objetivo foi desenvolver em C, a partir do código-base que foi fornecido, uma simulação deste encontro de amigos e da consequente preparação da partida e o seu decorrer. Estas 3 entidades (cargos dos amigos) serão processos independentes, que comunicam entre si através de semáforos e memória partilhada de forma a determinar quais estão bloqueados a aguardar a sua libertação por uma outra entidade. Nesta circunstância um semáforo não precisa de ser associado ao semáforo de uma rodovia! Em vez disso, vemo-los como o referido meio de comunicação direta sobre vários aspetos do estado do jogo, através por múltiplos semáforos.

Começar-se-á por uma enunciação mais a fundo de cada uma das entidades em jogo, passando depois para uma breve exposição à necessidade de semáforos para este problema em específico, cujos estados das entidades e dos semáforos serão escrutinados a fundo no terceiro capítulo. Existe também um capítulo dedicado à demonstração de funcionamento do trabalho, juntamente com percalços tidos em alguns momentos e respetivos *outputs* considerados pertinentes para expor.

1.1 Abordagem Efetuada

Para as 3 entidades existentes existem 3 ficheiros que foram solicitados para completar para implementar a simulação:

- `semSharedMem`**`Player`**`.c`
- `semSharedMem`**`Goalie`**`.c`
- `semSharedMem`**`Referee`**`.c`

Cada um destes ficheiros irá reger o comportamento da respetiva entidade, que serão abordadas de seguida, pelo que foi necessário bastante rigor no seu desenvolvimento.

Apesar de ambos os membros do grupo terem contribuído de forma semelhante no todo do projeto, fez-se a seguinte divisão das tarefas realizadas, por questões logísticas:

Alexandre Andrade (119279): Funcionamento dos players (`semSharedMemPlayer.c`); Funcionamento dos goalies (`semSharedMemGoalie.c`);

Miguel Neto (119302): Funcionamento dos referees (`semSharedMemReferee.c`);

Quanto à manutenção do código e consequente testagem, foi um trabalho realizado por ambos durante o prazo dado.

1.2 Entidades envolvidas

Como foi referido anteriormente, o grande objetivo deste programa é a simulação de um jogo de futebol. Para tal, vão ser precisas 3 entidades (jogadores de campo (**players**), guarda-redes (**goalies**) e árbitros (**referees**)), que interagem entre si de diversas formas ao longo do tempo.

Para se desenrolar o jogo, são precisas 2 equipas, com 5 elementos cada, e um árbitro. Todos os membros que chegarem em excesso serão "descartados", isto é, impedidos de jogar.

Players

No jogo de futebol em questão, cada equipa necessita de 4 jogadores de campo (players), o que totaliza 8 em jogo. Pelos testes que nos foram dados, testaremos o programa com 10 players, pelo que em teoria 2 terão que ficar de fora, sendo esses dois os que chegarem por último.

O seguinte diagrama descreve os estados que cada player pode assumir ao longo do seu "tempo de vida":

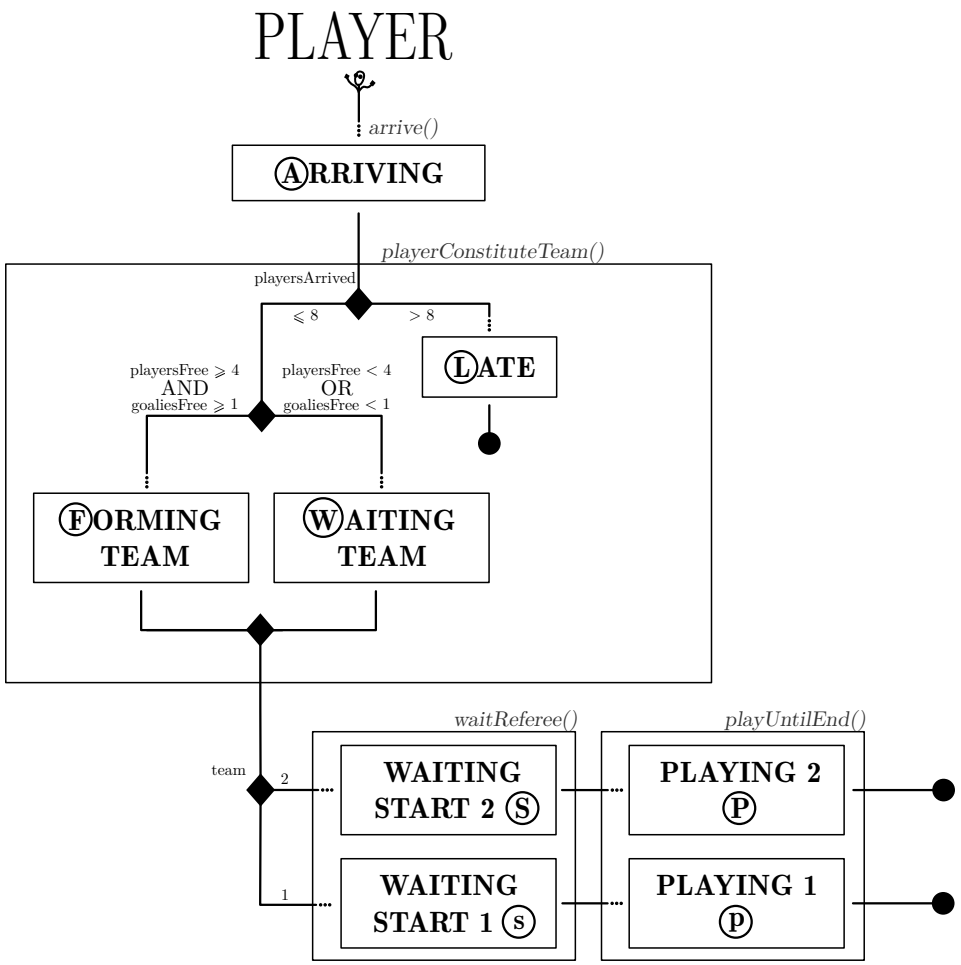

Figura 1

Goalies

Cada equipa em jogo irá precisar igualmente de um guarda-redes, totalizando as tais 5 pessoas por equipa. Estando 2 equipas a jogar, serão necessários 2 goalies. Nos testes que nos foram dados, testaremos o programa com 3 goalies, pelo que o que chegar por último ficará de fora.

O seguinte diagrama descreve os estados que cada goalie pode assumir ao longo do seu "tempo de vida":

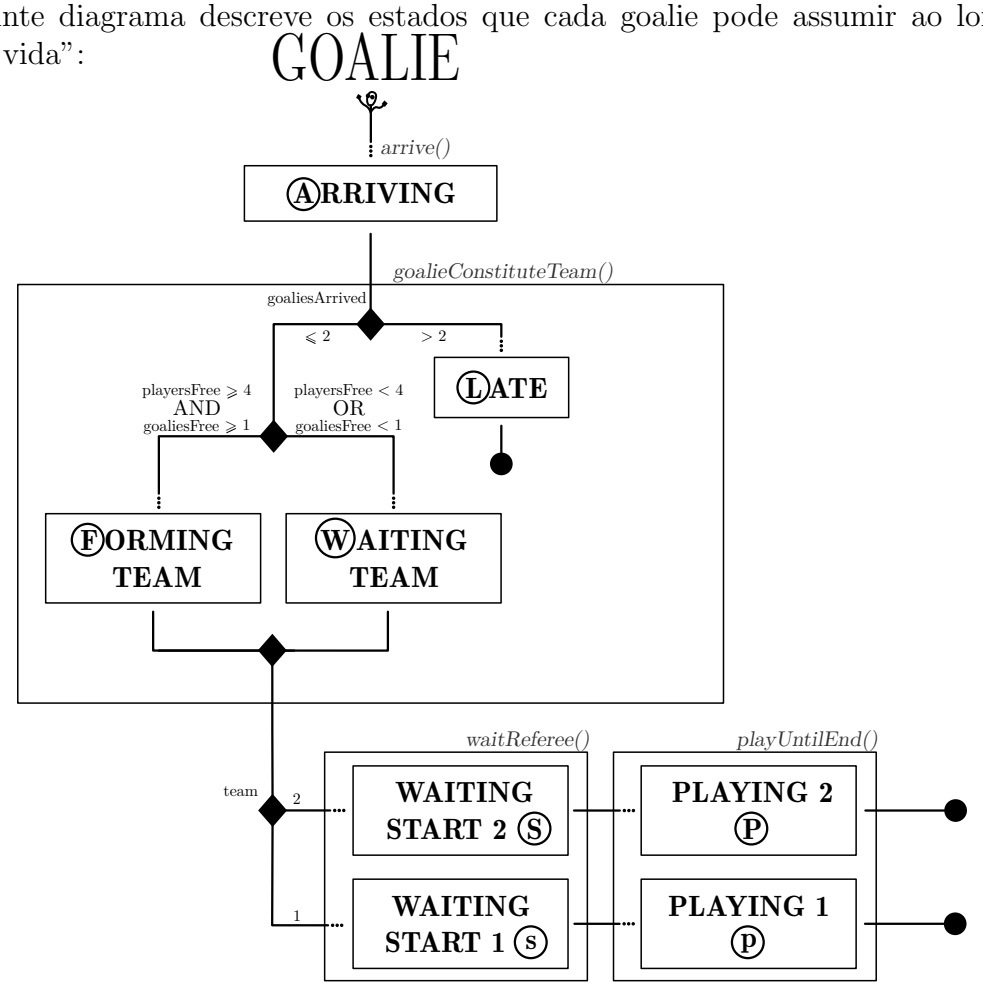

Figura 2

Referee

Por último, também é necessário um árbitro, que será aquele que irá coordenar as várias fases da partida. Consequentemente, possui comportamentos e ações diferentes daquelas das restantes entidades, dado o papel único que desempenha neste programa.

Assim, o seguinte diagrama descreve os estados que o referee pode assumir ao longo do seu "tempo de vida":

Figura 3

2 Mutex e uso de semáforos

Estas entidades mencionadas serão processos distintos no sistema, com as suas características específicas, mas precisam de comunicar (interagir) entre si para que o encontro para a futebolada possa realizar-se adequadamente. Para isto utilizamos o conceito de memória partilhada, que é permitida através dos TAD (Tipos Abstratos de Dados) `sharedDataSync` e `sharedMemory` que foram providenciados. Assim temos a `struct SHARED_DATA`, que além de possuir variáveis que representam os estados do jogo, possui também variáveis que armazenam os IDs de semáforos. Semáforos esses que são a ferramenta vital de comunicação entre entidades para o sistema funcionar corretamente.

De uma forma coloquial, como é-nos providenciado pela documentação do projeto[1], ora encontramos o termo *mutex*, ora encontramos o termo *semaphore* (semáforo). O que os difere é o facto de um semáforo permitir a múltiplas *threads* o acesso a uma região crítica em concreto, em simultâneo, ao passo que um mutex só permite o acesso a uma *thread*. Independentemente do número de threads que possam estar na região crítica, a natureza de ambos advém de permitir acessos apenas num processo (que possui essas *threads* referidas). Então, para que os semáforos (que vai ser o termo utilizado para abordar tanto um *mutex* como um semáforo, por questões simplísticas do relatório) possam reger o acesso dos diferentes processos aos dados relevantes, justifica-se que estejam na memória partilhada (indicada no primeiro parágrafo), acessível por todas as entidades a qualquer momento.

Temos as seguintes variáveis que armazenam os identificadores dos semáforos[2]:

- `playersWaitTeam`
- `goaliesWaitTeam`
- `playersWaitReferee`
- `playersWaitEnd`
- `refereeWaitTeams`
- `playerRegistered`
- `playing`

Ironicamente, estes "controladores de acesso"(pois essencialmente os semáforos acima indicados bloqueiam ou não o acesso ao próximo estado das entidades) não podem ser acedidos de forma descomunal e também têm que ser controlados. Então existe um semáforo `mutex` (binário) que serve de controlo à alteração inoportuna de determinadas semáforos em momentos em que o código *precisa* de ter apenas um *thread* a manipular propriedades, nas chamadas regiões críticas, **prevenindo uma situação de *deadlock***.

[1] Os comentários fornecidos para o código do projeto possuem sintaxe que possibilita a transformação do código numa documentação *"fully-fledged"* através de ferramentas como *Doxygen*.

[2] Apesar da informação de cada semáforo estar já explícita no código, é possível visualizar a explicação de cada elemento através desta referência gerada automaticamente a partir do código dado, com a ferramenta referida na nota de rodapé anterior.

Assim, um passo crítico na implementação são as operações **P** e **V** dos semáforos, dadas pelas funções `semUp()` e `semDown()`, respetivamente[3].

Os seguintes diagramas apresentam de antemão uma lógica mais completa das entidades, agora com os semáforos (🚦) que declaram se uma entidade avança para o próximo estado ou não, assim como o que causa a espera:

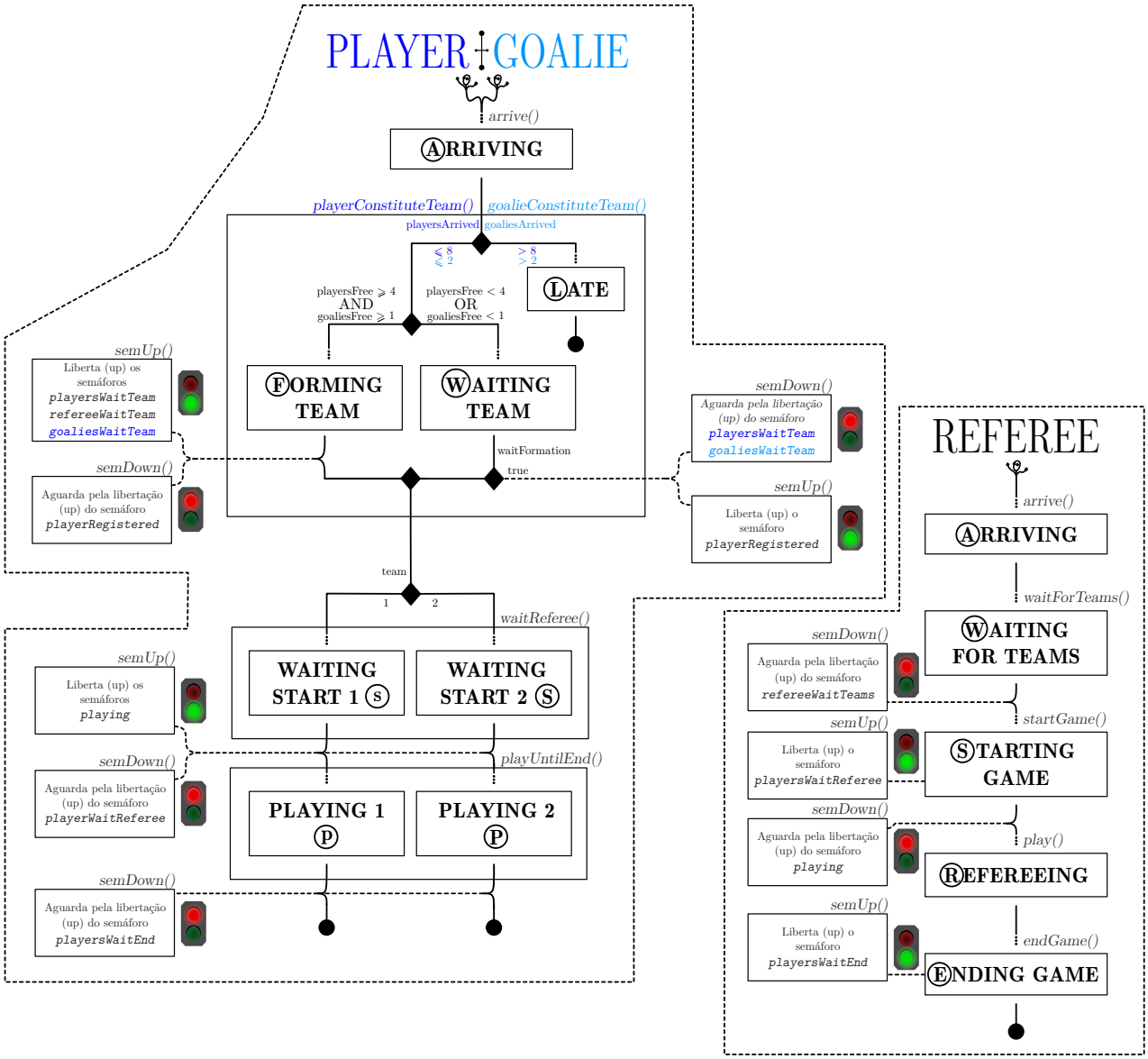

Figura 4: 2 diagramas correspondentes à interação entre as 3 entidades através dos semáforos. No primeiro (superior esquerdo), tudo o que é comum ao Player e ao Goalie é representado pela cor ■, senão usa-se a cor ■ para o Player e ■ para o Goalie.

[3]**Nota:** Evidentemente, estas e outras funções existem no TAD `semaphore` providenciado, mas na implementação pedida só são usadas essas 2 funções, graças ao grau de abstração que nos é oferecido.

3 Lógica do programa

Tendo em conta as várias etapas que vão desde a chegada dos vários elementos, à formação de equipas e ao jogo propriamente dito, decidimos dividir a explicação do nosso programa em 3 partes, de modo a agregar o máximo possível os estados com mais coisas em comum.

3.1 Arriving, Waiting and Late system

Nenhum jogo de futebol pode decorrer sem os necessários membros presentes. Como tal, o primeiro passo passa por tratar da chegada dos vários elementos, de modo a assegurar que a formação das equipas é bem feita.

Arriving

Esta fase é algo comum às 3 entidades, desempenhada através da função `arrive()`, como podemos ver na figura seguinte[4]:

```
138   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);
       }

       // É atribuído o estado de arriving
       (sh->fSt).st.goalieStat[id] = ARRIVING; //A
       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 5: Excerto do código do ficheiro `semSharedMemGoalie.c`.

Excetuando o mutex, que está lá para impedir que mais que várias threads alterem os dados da memória partilhada simultaneamente, esta função não utiliza semáforos. De facto, nesta função apenas se define os estados dos vários intervenientes como **ARRIVING (A)**, sendo posteriormente salvos. A cada membro é igualmente ordenado esperar por um período de tempo random, cujos intervalos variam entre os diferentes tipos de pessoa, de modo a simular a sua chegada "aleatória" à partida.

[4]Neste caso, é demonstrada a função arrive() do semSharedMemGoalie.c, mas é equivalente entre programas

Waiting and Late system

Nesta secção já começamos a ter diferenças entre o referee e os players/goalies. Começando pelo referee, uma vez que só chegará 1 árbitro, ele passará imediatamente para o estado **WAITING_TEAMS (W)** no qual, como o nome indica, irá permanecer até ser informado da formação das 2 equipas necessárias para se iniciar o jogo. Tal é conseguido na função `waitForTeams()`, onde dois 'downs' do semáforo `refereeWaitTeams` bloqueiam o processo do referee até ser libertado por algum dos players ou goalies.

Relativamente aos players e aos goalies, o mesmo é conseguido recorrendo às funções `playerConstituteTeam()` e `goalieConstituteTeam()`, respetivamente. À medida que vão chegando, cada membro vai registando a sua chegada na memória partilhada, incrementando em 1 unidade os contadores dos `playersArrived` e `playersFree` (ou `goaliesArrived` e `goaliesFree`, no caso dos guarda redes). Se antes do elemento em questão já tivessem chegado 8 players (máximo número de players numa partida), o player que acabasse de chegar seria avisado que tinha chegado atrasado e não entrava no jogo, sendo lhe mudado o estado para **LATE (L)**.

O mesmo acontece com os goalies, onde se já tivessem chegado 2 guarda-redes, o terceiro seria avisado que tinha chegado atrasado, passando então de **ARRIVING** para **LATE**.

No entanto, se ainda não se verificassem condições para a formação de uma equipa (nomeadamente o número de players livres ser menor que 4 ou o número de guarda-redes ser menor que 1), a contar com a pessoa que tivesse acabado de chegar, esse elemento passaria então ao estado **WAITING_TEAM (W)**.

Saindo agora das operações que envolvem a zona crítica, aos players e goalies que estivessem como **WAITING** (comprovado pela variável auxiliar `waitFormation`) era feito um 'down' do semáforo `playersWaitTeam` (`goaliesWaitTeam` para os guarda-redes), sinalizando que eles se encontravam à espera que fosse formada uma das equipas, bloqueando-lhes o processo nesse ponto até um outro membro (player ou goalie) ser encarregue de formar a equipa[5].

[5]Isto será abordado de seguida na secção **3.2 Forming Teams**

3.2 Forming Teams

Se, com a chegada de um novo player ou goalie, o número de players e goalies for igual ou superior ao mínimo necessário para formar uma equipa, a pessoa que chegou vai ficar então "responsável" por formar uma das equipas. Para além de atualizar os contadores referidos no ponto anterior, é lhe atribuído o estado **FORMING_TEAMS (F)**. Com isto salvo, são de seguida decrementados os contadores `(sh->fSt).playersFree` e `(sh->fSt).goaliesFree` (em 4 e 1 unidades, respetivamente), indicando que esses membros deixaram de estar disponíveis para formar equipa (uma vez que vão agora passar a formar uma).

O próximo passo varia ligeiramente dependendo do tipo de entidade que estiver a fazer a formação da equipa:

- Se for um **player**, necessita de indicar a 3 outros players e a 1 goalie que podem fazer o registo na equipa. Para tal, ocorre uma sequência de *up*'s (libertações) e *down*'s (bloqueios) em vários semáforos. Em primeiro lugar, é dado 'up' ao semáforo `playersWaitTeam` e um *down* ao `playerRegistered`, bloqueando o player que está a formar a equipa até o jogador sinalizado completar o seu registo na equipa. Este primeiro 'up' liberta os players que estavam bloqueados no estado **WAITING_TEAM**, onde eles registam o número da equipa que foram atribuídos, completando o processo de registo com um `semUp()` ao semáforo `playerRegistered` (permitindo que o formador da equipa possa resumir o seu processo). O mesmo se aplica ao goalie necessário para formar a equipa, apenas alterando as interações com o `playersWaitTeam` para `goaliesWaitTeam`. Com isto concluído, o jogador que está a formar a equipa guarda também o id dela, atualiza o id da próxima equipa e termina por dar um `semUp()` ao semáforo `refereeWaitTeams`, informando o referee que uma equipa acabou de ser formada.

- Se a pessoa que estiver a formar a equipa for um **goalie**, a lógica mantém-se igual, mudando apenas as interações com os semáforos que controlam a junção de pessoas à equipa (como quem está encarregue de formar a equipa é um goalie, deixa de ser necessário dar 'up' e 'down' do semáforo `goaliesWaitTeam`).

Com as 2 equipas formadas, os vários jogadores ficam à espera que o árbitro inicie o jogo, através da função `waitReferee()`, onde mudam o seu estado para **WAITING_START_1 (s)** ou **WAITING_START_2 (S)**, dependendo se estão na equipa 1 ou 2. Após isso, ficam bloqueados nesse estado até ao referee dar 'up' ao semáforo `playersWaitReferee`.

3.3 Game Logic

Passamos finalmente à lógica do jogo, que pode ser dividido em 3 partes: início do jogo, decorrer do jogo e final do jogo. Tudo isto é controlado pelo referee, sendo ele a entidade de foco nesta secção.

Tendo o referee chegado ao estado **ARRIVING**, procede, após um intervalo de tempo, para o estado **WAITING_TEAMS**, estado esse em que fica bloqueado até que **ambas** as equipas estejam prontas. A prontidão das duas é obtida pelo semáforo `refereeWaitTeams`, que terá que receber o `semUp` de cada equipa para que o referee consiga fazer `semDown` duas vezes.

```
165   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);
      }

      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);
      }

      semDown(semgid,sh->refereeWaitTeams);
      semDown(semgid,sh->refereeWaitTeams);

  }
```

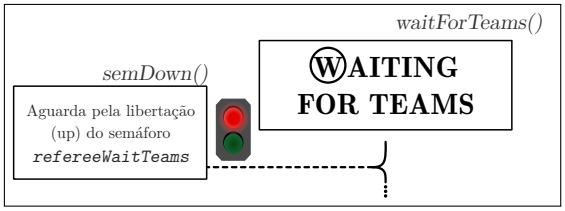

Figura 6: Segmento de código do referee para aguardar pela constituição das equipas (ou seja, pelas funções `playerConstituteTeams()` e `goalieConstituteTeams()`), com a demonstração esquemática utilizada na Figura 4.

Com as duas equipas formadas e prontas para entrar em ação, o código pode avançar para a próxima função, `startGame()`, que vai comunicar a todos que o jogo vai começar. Mais uma vez, altera-se o estado do referee para **STARTING_GAME**, liberta-se o semáforo relativo à espera dos jogadores por este sinal, o `playersWaitReferee`, e assegura-se de que os jogadores estão de facto a jogar, através de `semDown()` ao semáforo `playing`, que tem tantas interações (*up*'s e *down*'s) quantos jogadores que não estejam atrasados.

Com isso, todos os goalies e players passam para a função `playUntilEnd()` onde, como o próprio nome diz, atualizam o seu estado para **PLAYING_1 (p)** ou **PLAYING_2 (P)** com base na equipa a que pertencem, e mantêm-se nesse estado até o referee libertar o semáforo `playersWaitEnd`. Enquanto isto, o referee está na função `play()`, no estado **REFEREEING**, durante um determinado tempo até decidir que é hora de terminar o jogo.

Este término dá-se com a alteração do seu estado para **ENDING_GAME**, juntamente com a libertação da condição de espera de todos os jogadores pelo fim do jogo, dada pelo semáforo `playersWaitEnd`. Finalmente, o programa termina de forma adequada, tendo ocorrido a simulação do encontro para a futebolada no seu todo.

4 Testes realizados

A realização de testes também é uma componente chave para garantir o bom desempenho do código desenvolvido, assegurando que nenhuma das entidades fica bloqueada em alguma fase.

Para a realização dos testes, usámos principalmente os scripts da diretoria `run` que nos foram fornecidos. Dentro deles, podemos destacar o `run.sh`, que executa várias vezes uma simulação do jogo para verificar possíveis erros que nos possam ter escapado e o `filter.sh`[6], que executa igualmente uma simulação, mas dando highlight às transições entre estados. Também nos foi fornecido o script `clean.sh`, que limpa recursos da memória partilhada e semáforos relacionados ao processo em questão.

Uma das formas de ir testando o funcionamento da nossa implementação de cada uma das entidades foi através dos **testes unitários** que nos eram providenciados pelo *makefile*. Para tal podíamos executar os comandos `make pl`, `make gl` ou `make rf` e depois verificávmos o output para concluir se tinha corrido como esperado.

Quando considerássemos que os resultados dos módulos individuais se encontravam de acordo com as expectativas, era-nos dada a opção de fazer make all, que compilava os 3 ficheiros por nós completados. Com isso compilado, podíamos ou correr manualmente, executando diretamente o comando `./probSemSharedMemSoccerGame`, ou com a ajuda do ficheiro `run.sh`.

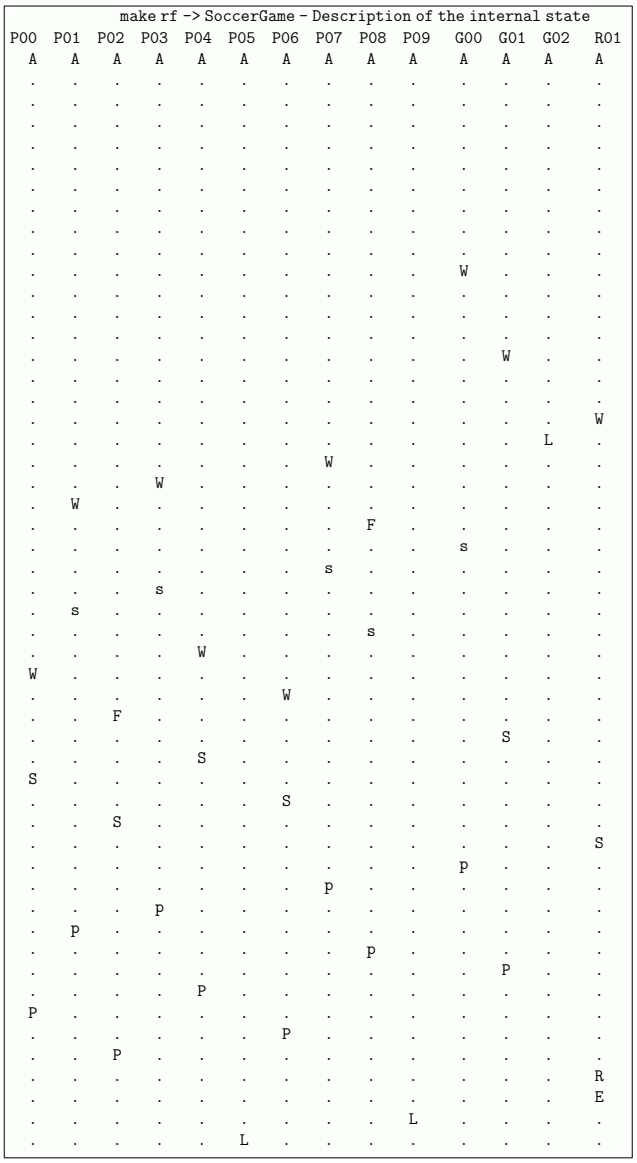

Figura 7: Exemplo de um output de um teste unitário, invocado através do `filter.sh`

[6]Foi criada também uma versão alternativa, `filter2.sh`, que permite visualizar os outputs processados também para tamanhos de equipas diferentes (**NUNCA** é alterado o número de árbitros) dos originais definidos pelos parâmetros no ficheiro `probConst.h`.
Caso seja utilizado o script, uma simples execução mostra como deve ser utilizado.
`./filter2.sh TOTALENTITIES NUMPLAYERS NUMGOALIES`,
com `TOTALENTITIES` = `NUMPLAYERS` + `NUMGOALIES` + `NUMREFEREES`

As figuras seguintes mostram a última iteração resultante da execução do comando `./run 20`, com o comando `./probSemSharedMemSoccerGame`, para a versão `make all` e `make all_bin`, respetivamente:

Versão trabalhada

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

Versão providenciada

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

Figura 8: Comparação entre *outputs* das duas versões da simulação.

Como podemos observar, o output do programa decorre como esperado, apenas havendo exporadicamente uma espécie de ”delay” aleatório na interação F - s/S e W - s/S nos players e goalies, apesar de tudo decorrer na ordem correta.

É relevante destacar que também foram feitos testes alterando os tempos de "sleep" dos vários intervenientes do jogo. Curiosamente, aumentando o tempo de sleep dos players face àquele providenciado no código inicial, verificamos que o output se assemelha mais ao exemplo-fonte fornecido. Isto pode dever-se ao facto de, sendo maior o tempo inicial de espera, mais "ordenadas" ficam as várias entidades nos processos de leitura e escrita de dados na memória partilhada, bem como nos *up*'s e *down*'s de semáforos.

Exemplo para tempos 1000 vezes superiores[7]:

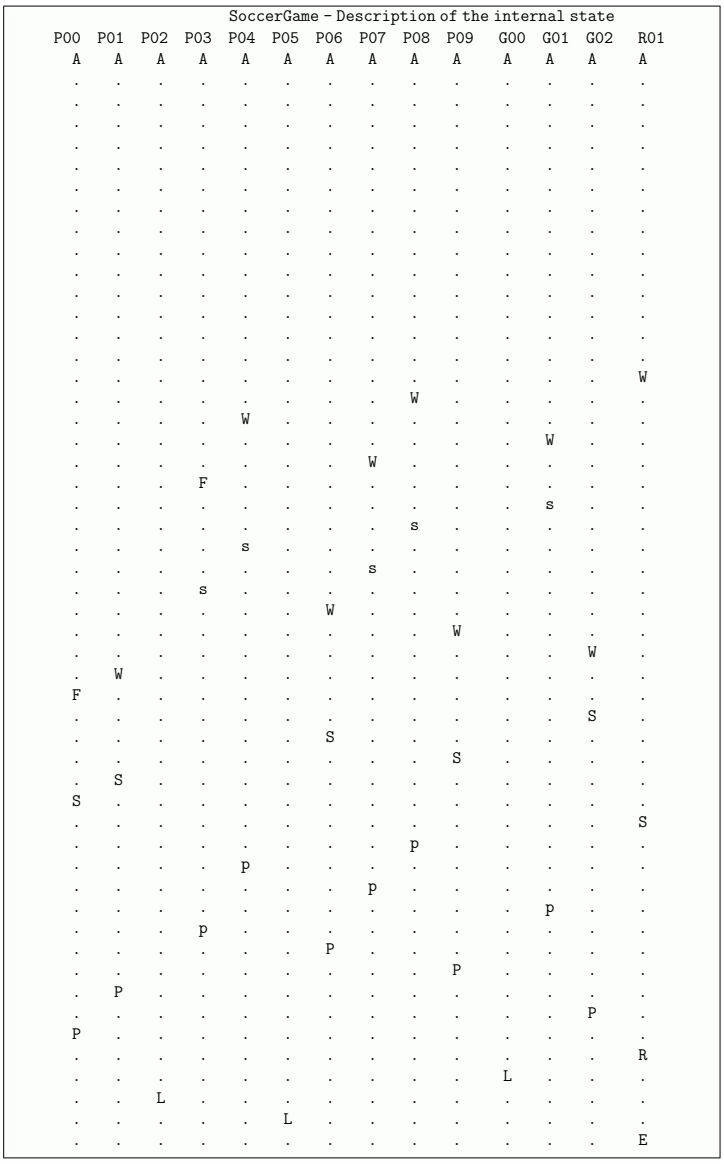

Figura 9: Exemplo de output extrapolado do comando `filter.sh`.

7

```
usleep(1000*(200.0*random())/(RAND_MAX+1.0)+50.0);
usleep(1000*(200.0*random())/(RAND_MAX+1.0)+60.0);
usleep(1000*(100.0*random())/(RAND_MAX+1.0)+10.0);
```

... para players, goalies e referee, respetivamente.

5 Conclusão

Reiterando o que foi dito na introdução, os semáforos são uma ferramenta muito útil na resolução de problemas de sincronização, como o que foi dado. Este trabalho permitiu-nos meter mãos à massa e trabalhar de perto com mecanismos associados à execução e sincronização de processos e threads, adquirindo conhecimentos de teor prático e teórico úteis. Para além disso, permitiu-nos pôr mais uma vez em prática os nossos conhecimentos da linguagem C.

Não só foi aplicado o que foi lecionado nas aulas, como também complementou-se o nosso saber com alguma pesquisa na Internet.

O trabalho por nós realizado está de acordo com os requisitos indicados no enunciado, seguindo os exemplos fornecidos. Se quiséssemos melhorar alguma coisa neste projeto, seria tentar arranjar uma forma de combater o *"delay"* de algumas das transições obtido quando os tempos do `usleep()` das entidades eram consideravelmente reduzidos.

Foi isto assim uma forma lúdica e desafiante de consolidar os conhecimentos nesta linguagem, melhorando a nossa compreensão sobre o tema abordado, mais as *nuances* intrínsecas.

6 Bibliografia/Referências

Referências

[1] GeeksforGeeks, geeksforgeeks.org.

[2] Medium, medium.com.

[3] Wikipedia, wikipedia.org.

[4] Guiões das aulas práticas.