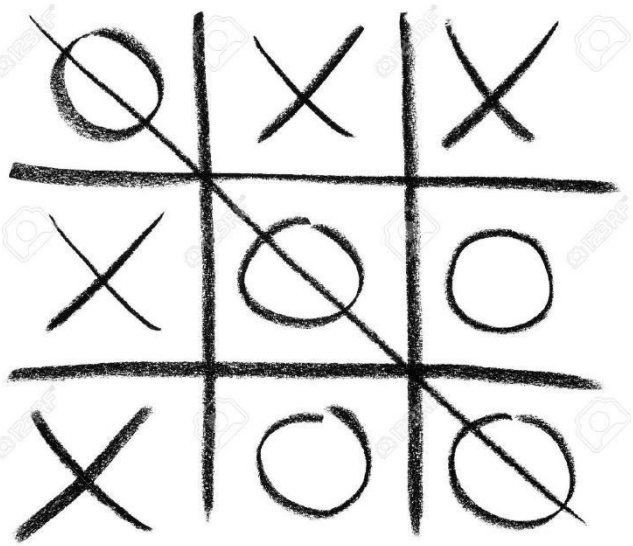
Universidade de Évora

Licenciatura Engenharia Informática

Linguagens Programação

Docente:Salvador Abreu

Jogo do Galo: Golang



Trabalho Realizado por:

André Figueira 31626

Sérgio Batista 31500

Indice

Conteúdo

Universidade de Évora

Introdução

Para esta tarefa foram disponibilizadas várias linguagens de programação possíveis sendo que após o leilão e por nosso interesse, ficamos com Go.

Golang é uma linguagem imperativa baseada em C sendo que tem sintaxe semelhante mas tem muitas outras influencias sendo que ser usada para um estilo de programação orientada a objetos apesar de não possuir classes nem hierarquia.

Em termos de enunciado houveram algumas dificuldades na interpretação inicial do mesmo apesar de termos chegado a um consenso e termos definido o rumo a seguir.

Universidade de Évora

Como compilar e correr?

Utilizadores de Windows:

1- Fazer download do MSL: https://golang.org/dl/
2- Definir a variavel de ambiente para o ficheiro de instalação anterior
3- Abrir command line no diretorio do src file
4- Fazer go run visualizador.go em 1 instancia do cmd
5- Fazer go run jogador.go em duas instancias da cmd

Se tiver bem instalado deve pedir em cada um dos terminais que não o do server para meter o nome do jogador.

Nota: Em caso de dúvida para os 2 primeiros passos recorrer a: https://golang.org/doc/install

Universidade de Évora

Utilizadores de Linux:

1- Fazer o download do arquivo correcto https://golang.org/dl/

2- tar -C /usr/local -xzf go$VERSION.$OS-$ARCH.tar.gz
   a. Exemplo: go1.2.1.linux-amd64.tar.gz

3- export PATH=$PATH./usr/local/go/bin

4- export GOPATH=$HOME/go_workspace
   a. Para criar workspace veja página seguinte

5- PATH=$PATH:$GOPATH/bin

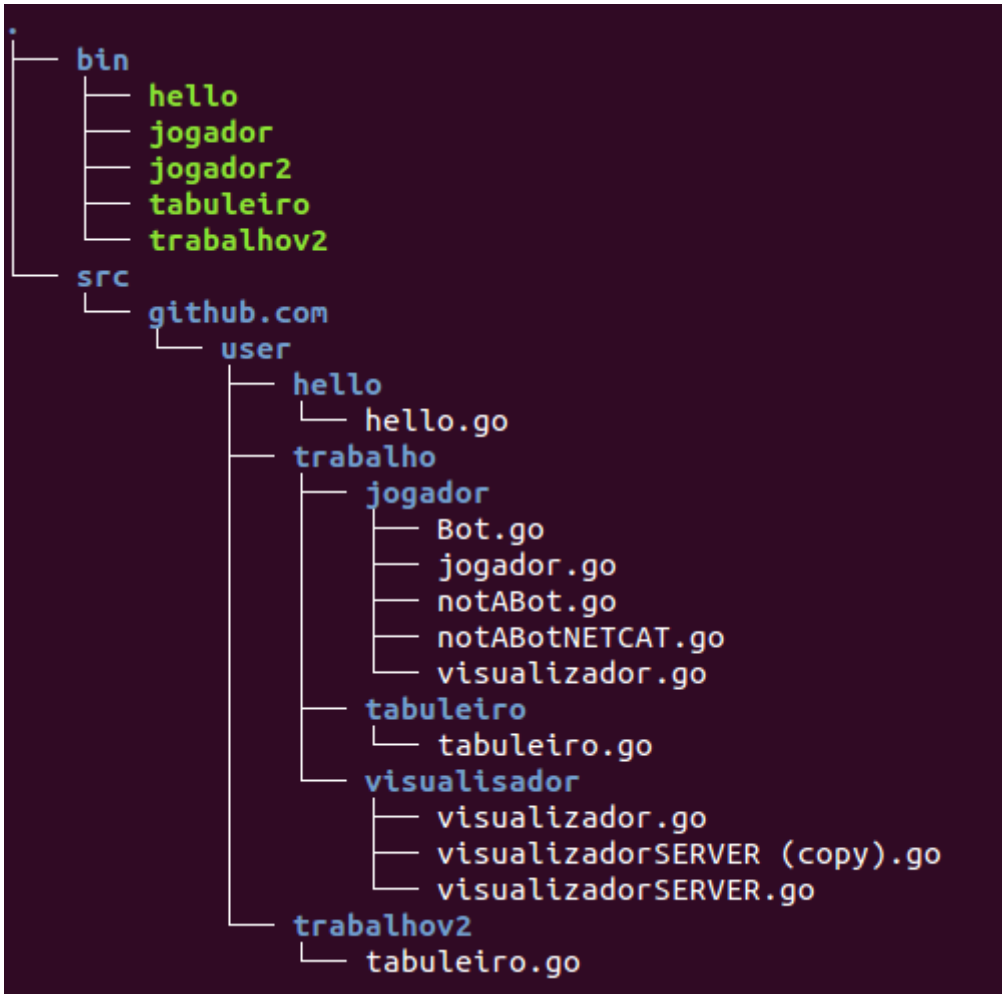
6- E na directoria dos programas correr em terminais diferentes com estes comandos por esta ordem:
   a. go run visualizador.go
   b. go run jogador.go
   c. go run jogador.go

Para criar um workspace:

mkdir $HOME/go_workspace

mkdir -p $GOPATH/src/github.com/user

mkdir $GOPATH/src/github.com/user/trabalho



Decisões Iniciais

Os primeiros passos antes da implementação do trabalho foram a busca da resposta de 2 perguntas fundamentais: Uso de Pipeline e NetCat Vs Sockets; Bots Vs UserInput.

É de realçar que para responder a estas perguntas da forma mais informada possivel foi realizada uma pesquisa extensa dentro das ferramentas disponibilizadas pelo Go para perceber realmente, o que era melhor utilizar sendo mais eficiente e menos propenso a falhas no controlo dos dados a comunicar.

No contexto de Bots Vs UserInput optámos por uma interpretação mais direcionada para o utilizador, sendo que permite uma experiencia dinamizada e mais atrativa, sendo que Go tem disponibilizadas ferramentas que facilitam a utilização de dados escritos pelo utilizador. É de referir que em estados iniciais do desenvolvimento, a implementação continha Bots mas até mesmo pela nossa experiência na utilização dos mesmos decidimos optar contra.

O contexto Pipeline e NetCat Vs Sockets era o nosso maior debate. Ambas têm vantagens sem duvida alguma, mas decidimos optar por sockets pelas dificuldades que tivemos na manutenção e criação e Pipelines com NetCat (para remoto). Sockets é um ambiente por nós conhecido e tornou a sua criação e estabilização mais eficiente.

Universidade de Évora

Comandos de Jogo

Na nossa implementação fizemo-la no sentido de incentivar a comunicação entre o visualizador e cada um dos jogadores sendo que a cada momento de jogo a realizar é pedido um comando para realizar uma determinada acção.

Comandos Visualizador:

No momento que dois jogadores se conectam ao visualizador o jogo começa o jogo.

Será enviado aos jogadores o comando “nome” e após receber resposta, será pedido ao visualizador o tamanho do jogo sendo que será apenas o input de um inteiro e será enviado o comando “tamanho N” para os jogadores em questão. Em seguida envia aos jogadores o respectivo simbolo “comeca X” e “comeca O” automaticamente. O que receber “comeca X” recebe também comando “joga” para inciar o jogo.

Cada vez que cada cada jogador joga, o visualizador tens dois comandos disponíveis, “joga” ou “espera”. Sendo “joga” permite o outro jogador jogar e “espera” faz com que o outro jogador fique a espera.

Comandos Jogador:

O primeiro pedido que o jogador recebe para começar o jogo é para escrever o seu nome, mas para tal terá de receber a mensagem do Visualizador. Para isto basta escrever “nome <nomejogador>”.

Sempre que o visualizador manda o comando “joga” para o jogador que é suposto jogar, o comando a inserir é “jogo X/Y”.

O jogador ao receber o comando “joga” pode também desistir escrevendo “desisto”. Sempre que é gerado uma Vitória, derrota ou empate é enviado para o comando apropriado automaticamente.

Implementação

O passo inicial da criação do jogo passou pelo desenvolvimento de métodos que mantivessem a integridade do jogo do galo e de todas as suas regras. Esta são então verifica victória ou derrota em linha coluna e diagonal, verificar empate (que verifica se todas as posições estão preenchidas) e verificar se uma jogada é valida (se as coordenadas são validas) quando é a vez do jogador jogar (quando recebe “joga” do visualizador).

Assim implementamos métodos que nos permitissem realizar estas condições.

O que se segue é um excerto de código.

```
func verifica_linha(tabuleiro [] string)(bool,string){
    tamanho := int(math.Sqrt(float64(len(tabuleiro))))
    size:= tamanho;
    var i int;
    n_certos:= 0;
    for i = 1; i < tamanho*tamanho; i++{

        if(strings.Compare(tabuleiro[i-1], tabuleiro[i]) == 0){
            n_certos++

            if(n_certos == tamanho-1){      // faz tamanho-1 verificacoes
                return true,tabuleiro[i];
            }
        }
        if(size-1 == i){    // mudanca de linha
            size=size+tamanho
            i += 1
            n_certos = 0
        }
    }
    return false,tabuleiro[0];
}

func verifica_coluna(tabuleiro [] string)(bool,string){
    tamanho := int(math.Sqrt(float64(len(tabuleiro))))
    var pos int;
    for i := 1; i <= tamanho; i++{
        n_certos:= 0
        for j := 2; j <= tamanho; j++{
            pos = offset(j,i,tamanho)
            MenosUmPos := offset(j-1,i,tamanho)

            if(strings.Compare(tabuleiro[MenosUmPos],tabuleiro[pos]) == 0){
                n_certos++;         // numero de certos e' igual a N-1 para verifica uma sequencia

                if(n_certos == tamanho-1){      // faz N-1 verificacoes
                    return true,tabuleiro[pos];
                }
            }
        }
    }
    return false,tabuleiro[pos];
}
```

Implementação(cont.)

Depois de implementados todos os métodos para a inicialização e concepção do jogo, implementamos métodos que permitiam a interacção Visualizador-Jogador.

Lado Visualizador:

Para se começar um jogo é necessário que dois jogadores se conectem. Ao se verificar tal condição é iniciado uma Goroutine que estabelece uma ligação entre estes 3.

O visualizador começa por enviar aos jogadores um comando “nome” em que este fica a espera dos respectivos nomes. Após ter os nomes é feito um input do lado do visualizador que pede o tamanho do jogo, se este for válido (um inteiro) é enviado para os jogadores qual o tamanho do tabuleiro, seguido dos símbolos respectivos usando o comando “comeca J” (comeca X para o primeiro que se conectou, ”comeca O” para o outro). Para iniciar uma jogada, é enviado ao primeiro que se conectou o comando “joga”

```go
func handlePlayers(player1 net.Conn, player2 net.Conn){     // the goroutine

    var size string;                        // tamanho do tabuleiro
    var nomePlayer1 string;                 // nome do jogador 1
    var nomePlayer2 string;                 // nome do jogador 2
    var messagePlayer1 string;              // messagens do jogador 1
    var messagePlayer2 string;              // mensagens do jogador 2

    player1.Write([]byte("nome\n"));                            // envia ao primeiro que se conectou o nome
    nomePlayer1,_ = bufio.NewReader(player1).ReadString('\n');

    player2.Write([]byte("nome\n"));                            // apos receber resposta envia este
    nomePlayer2,_ = bufio.NewReader(player2).ReadString('\n');

    nomePlayer1,nomePlayer2 = splitName(nomePlayer1,nomePlayer2);   // faz split do comando + nome
    for true{
        fmt.Print("Tamanho do jogo: ");
        fmt.Scan(&size);                                    // scan for input
        if _, err := strconv.Atoi(size); err == nil{    // so permite input valido
            fmt.Print("\n");
            break;
        }
    }

    time.Sleep(100 * time.Millisecond);
    player1.Write([]byte("tamanho "+size+"\n"));        // envia aos jogadores o tamanho do tabuleiro
    player2.Write([]byte("tamanho "+size+"\n"));        //

    time.Sleep(100 * time.Millisecond);
    player1.Write([]byte("comeca X"+"\n"));             // envia aos jogadores quem e' quem
    player2.Write([]byte("comeca 0"+"\n"));             // o primeiro a se conectar e' sempre o X

    time.Sleep(100 * time.Millisecond);
    player1.Write([]byte("joga\n"));                // primeiro a jogar e' o player1

    var endGame = false;

    for !endGame{

        endGame = handleRequests(player1,nomePlayer1,player2,nomePlayer2,messagePlayer1);   // redireciona o output para o jogador 2
        endGame = handleRequests(player2,nomePlayer2,player1,nomePlayer1,messagePlayer2);   // redireciona o output para o jogador 1

    }

    fmt.Println("\n\nGame between "+nomePlayer1 + "and "+nomePlayer2+" has ended\n\n");

    player1.Close();        // fecha a conecao
    player2.Close();        //
}
```

Sempre que o visualizador recebe mensagem de um jogador esta é redirecionada para o outro jogador. Se for um comando jogo X/Y o outro jogador recebe a mensagem e fica ainda a espera de um outro comando do tipo “espera” ou “joga”.

Se for “espera” o jogador fica a espera novamente de outro input, se for “joga” então permite o jogador jogar.

Qualquer outro comando válido é redirecionado.

Lado Jogador:

O jogador conecta-se ao servidor e entra num ciclo infinito, em que fica espera de input do servidor, ao receber input age sobre o mesmo usando um switch.

Nota especial: O jogador contém uma representação do tabuleiro e sempre que é recebido um comando jogo X/Y este actualiza no seu tabuleiro.

```
func main(){

    // connection to server

    conn , _ = net.Dial("tcp","localhost:31626");                  // connecao ao servidor
    var message string;                                            // mensagem

    continueGame = true;                                           //  estado do jogo. false significa que o jogo começou

    for continueGame {

        message , _  = bufio.NewReader(conn).ReadString('\n');     // le input (output do outro jogador)

        switch{ // reage ao input

            case strings.EqualFold(message,"nome\n"):              // se for nome
                printOtherPlayerMessage("Visualizador",message);   // print da msg do outro

                for true{
                    nome,_ := bufio.NewReader(os.Stdin).ReadString('\n');   // pede input
                    if strings.EqualFold(nome[0:5],"nome "){                // so envia o comando valido
                        fmt.Fprintf(conn, nome + "\n");                     // escreve output
                        break;
                    }
                }
                break;

            case len(message) > 7 && strings.EqualFold(message[0:7],"tamanho"): //  recebe tamanho
                b := strings.Split(message, " ");                           // split da msg

                c := strings.Split(b[1],"\n");                              // split novamente por causa do \n
                size,_ := strconv.Atoi(c[0]);                               // ja tem o tamanho

                tamanho = int(size);
                tabuleiro = tabuleirovazio(tamanho);                        // gera tabuleiro vazio
                printOtherPlayerMessage("Visualizador",message);
                break;
```

```
case strings.EqualFold(message,"tamanho\n"):                        // quando so recebe o comando tamanho
    tabuleiro = tabuleirovazio(3);
    tamanho = 3;
    break;

case strings.EqualFold(message,"comeca X\n"):           // define os simbolos
    current_symbol = "X";    // o seu simbolo
    opositeSymbol = "O";     // simbolo do outro jogador
    printOtherPlayerMessage("Visualizador",message);
    break;

case strings.EqualFold(message,"comeca O\n"):           // define os simbolos
    current_symbol = "O";    // o seu simbolo
    opositeSymbol = "X";                // simbolo do outro
    printOtherPlayerMessage("Visualizador",message);
    break;

case strings.EqualFold(message,"desisto\n"):            // jogo termina porque o outro desistiu
    printOtherPlayerMessage(opositeSymbol,message);
    continueGame = false;
    break;

case strings.EqualFold(message, "ganhei\n"):            // termina porque o outro ganhou
    continueGame = false;
    printOtherPlayerMessage(opositeSymbol,message);
    break;

case strings.EqualFold(message,"empatamos\n"):          // termina porque empataram
    continueGame = false;
    printOtherPlayerMessage(opositeSymbol,message);
    break;

case strings.EqualFold(message,"espera\n"):             // fica a espera de outro comando
    printOtherPlayerMessage("Visualizador",message);
    break;

case strings.EqualFold(message,"joga\n"):               // permite jogar
    printOtherPlayerMessage("Visualizador",message);
    evalInput();        // enviar um comando
    break;

case len(message) > 4 && strings.EqualFold(message[0:4],"jogo"):    // recebe joga L/C
    printOtherPlayerMessage(opositeSymbol,message);

    message2 := strings.Split(message, " ");                    // split dos spacos
    message3 := strings.Split(message2[1],"/");                 // split por causa da /
    L := message3[0];                                           // ja tem L
```

Para enviar realizar jogadas ou desistência tem de receber o comando “joga” do visualizador.

Ao receber o utilizador pode então enviar “desisto” caso queira desistir ou “jogo X/Y” para preencher uma posição.

Esta segunda só é enviada se for válida, ou não é enviada se gerar uma Victória e nesse caso envia ao visualizador que ganhou e o jogo entre os dois termina.

Conclusão

Em suma podemos dizer que neste trabalho os nossos conhecimento nesta área de programação aumentou muito.

Entrando no desenvolvimento deste trabalho, o nosso conhecimento de relação Client/Server apesar de não ser escasso, sentimos que não era suficiente e este trabalho veio permitir aprofundarmos essa área sendo que ainda teve superior dificuldade pela imposição de uma linguagem na qual não somos muito familiares sendo que, este mesmo facto, nos permitiu alargar o leque de linguagens de programação conhecidas o que é sempre benéfico.