Relatório do trabalho

Grupo:5

Alunos: Ricardo da Silva Correia A89156

Pedro Portovedo

Hugo Quintas A89152

Introdução/Identificação do problema

Este trabalho tem como principal objetivo a elaboração de um programa que permita jogar basquete entre vários jogadores ou apenas um. O jogador deve lançar a bola, tendo em conta a velocidade de salto, a velocidade de lançamento e o ângulo de lançamento, de forma a verificar se a bola entra ou não no cesto.

Cada jogador tem três rondas, cada ronda tem até cinco lançamentos e, em modo multijogador, os jogadores estão inicialmente em pé de igualdade, uma vez que, no início de cada ronda, a posição é a mesma para todos.

No final, o jogador que tiver mais pontos ganha, caso haja empate, os jogadores empatados vão à morte súbita.



Definição de objetivos

Na realização deste trabalho, tivemos como principais objetivos:

* Tornar o jogo realista na escolha de valores relativos aos jogadores e aos seus lançamentos. Tendo em conta a impossibilidade de valores exagerados, tanto na altura dos jogadores (o mínimo que se pode utilizar corresponde ao valor da altura da pessoa mais pequena do mundo e o valor máximo não pode superar o valor da pessoa mais alta do mundo), como no ângulo de lançamento (valores entre 0º e 80º), como na altura de salto (valores entre 0 e 6.1 m/s, sendo este último o recorde mundial que se pode saltar utilizando força inferior) e por fim colocamos um limite no que toca a velocidade de lançamento uma vez que esta depende da pessoa que lança e não achamos registos de alguém que tenha superado o valor de 30 m/s. Tudo isto tem como objetivo não permitir a utilização de valores fora do padrão normal e tornar o jogo um pouco mais realista.
* Elaborar o que era pedido em obrigatório e alguns extras quanto ao trabalho. Na parte obrigatória, criamos um Menu (em que o Novo Jogo foi separado em dois, ou seja, em multijogador e solo , além disso, existe a opção que permite a visualização da tabela de pontos e falhas, por ronda, de um determinado jogo e por fim a opção Sair),o Novo Jogo como já foi referido , foi separado em dois modos distintos, um que se chama Multijogador e outro Solo sendo que estes cumprem os requisitos no que se refere ao Novo Jogo, a função GerarAreas também foi introduzida no trabalho, a GerarPosiçao foi introduzida só não foi dita a área em que se está a realizar o valor aleatório ou seja não se colocou um printf devido á falta de estética que isso colocava no trabalho. Apesar de que sempre que era importante saber a área em que se estava, era colocado um printf nos locais que achamos mais apropriados. Além disso, esta função permite a escolha da posição aleatória tanto no início de cada ronda, como na passagem de um lançamento para o outro, ou seja, é dinâmica ao ponto de poder ser usada para esses casos. O Print não colocamos necessariamente numa função devido à simplicidade que isso apresenta na função, porém através de printfs presentes em ciclos é possível saber qual o jogador a jogar, o lançamento e a respetiva ronda. A AlturaSalto foi realizada. O Lançamento foi dividido em 3 funções, de forma a ser mais fácil a leitura do código para quem está a ler (essas funções são a criarangulo, criarvelocidadeL e criarvelocidadeS) e, por fim, também foi realizada a função ClassificaçaoFinal que grava num ficheiro texto a tabela (essa função no programa designa-se gravarjogoemtexto). Além disso, decidimos criar outras duas funções que gravam a tabela num ficheiro binário e a lê no menu opção: tabela de pontuações, sendo que assim é mais pratico e não é necessário sair do jogo para a visualizar.
* Na parte da bonificação, fizemos a função da imagem do lançamento em tempo real; fizemos a “morte súbita” e apenas esta ignorando as duas rondas extra para o jogo não ser tão demorado e ser mais apelativo para se jogar. Também fizemos a adaptação da dificuldade do nível (ponto 4).

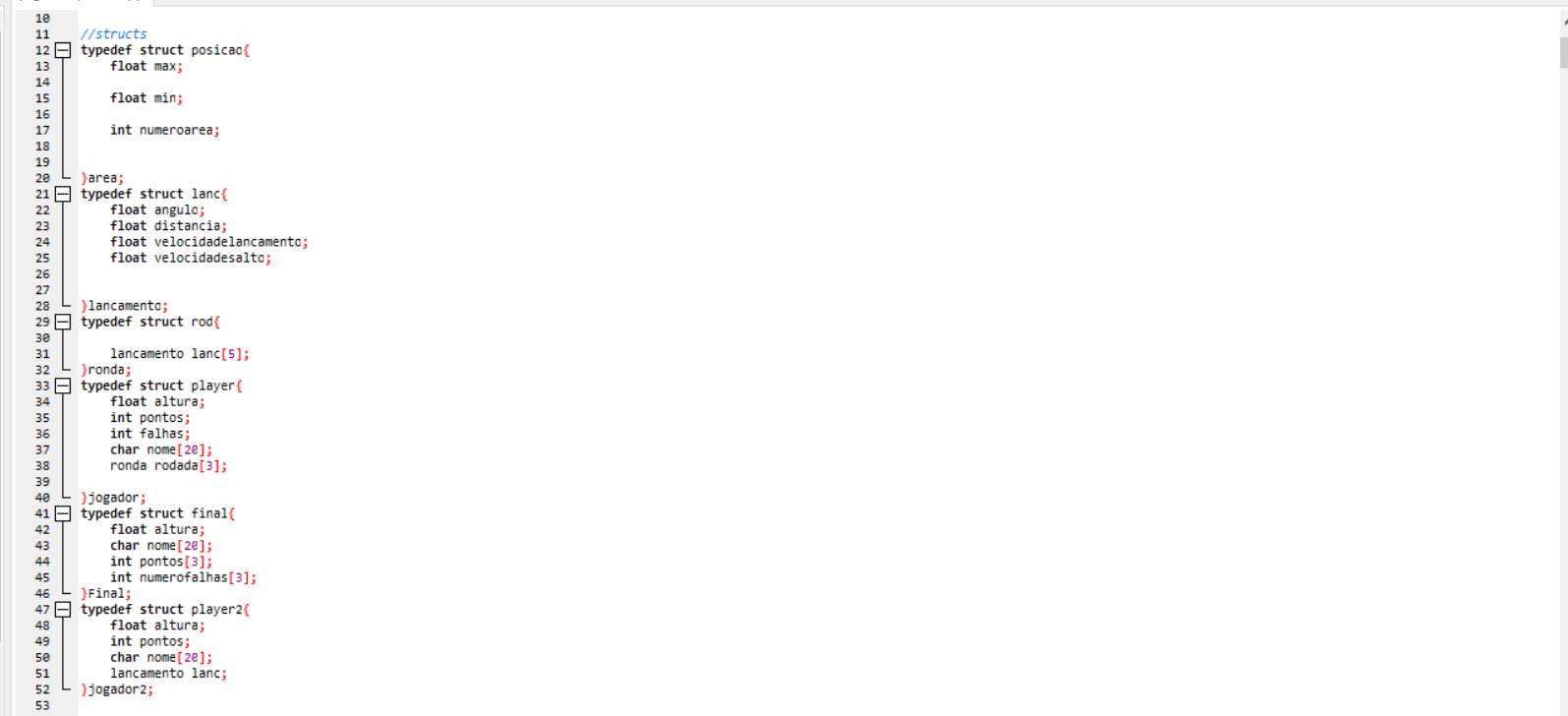
Métodos/material desenvolvido

Na realização deste trabalho os métodos que desenvolvemos foram:

* A utilização de structs, para podermos criar tipos definidos por nós (por exemplo a struct lançamento em que esta presente tudo aquilo que um lançamento precisa, como por exemplo o angulo, a velocidade de salto, …).
* A utilização de listas, que foram extremamente úteis na elaboração do programa.
* Utilização de funções cíclicas (for ,while,…)que foram de extrema importância na elaboração do menu, numero de jogadores, angulo, velocidade de salto, velocidade de lançamento, morte súbita, entre outros , de forma a criar ciclos viciados para não permitir que o programa acabe se for introduzido valores impróprios ,ou para ir filtrando valores como ocorre na morte súbita, alem disso estas funções cíclicas foram de uma extrema importância no preenchimento das structs e na realização dos lançamentos de cada jogador, entre outros.
* Utilização de funções condicionais (if…else, swicth…case) utilizadas na elaboração dos pontos, e em tudo o que exigia condições.
* Utilização de apontadores.
* Utilização de funções void e funções que retornam valores para elaborar determinadas tarefas.
* Utilização de ficheiros para gravar a tabela.
* Utilização de bibliotecas onde coloquei a animação uma vez que esta ocupava imenso espaço no trabalho, tornando-o assim mais legível.
* Utilização de funções já existentes no C como a função random, o system(“cls”), o system(“pause”) ,atoi,atof,entre outras.

Descrição das funcionalidades desenvolvidas com maior grau de importância

Explicação das funções auxiliares e structs:

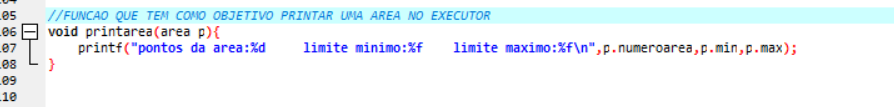
* Structs:

Verifica-se que a struct jogador contem uma lista rodada[3] pois cada jogador tem de realizar 3 rodadas, o nome do jogador, a altura, os pontos totais que o jogador obteve no final, as falhas obtidas no final. A struct ronda contem uma lista de struct lançamento[5] pois só pode haver até 5 lançamentos no máximo, e a struct lançamento contem tudo aquilo que foi preciso para a realização de um lançamento.

A struct Final é a struct que vai ser usada ao gravar um ficheiro e ao ler o mesmo.

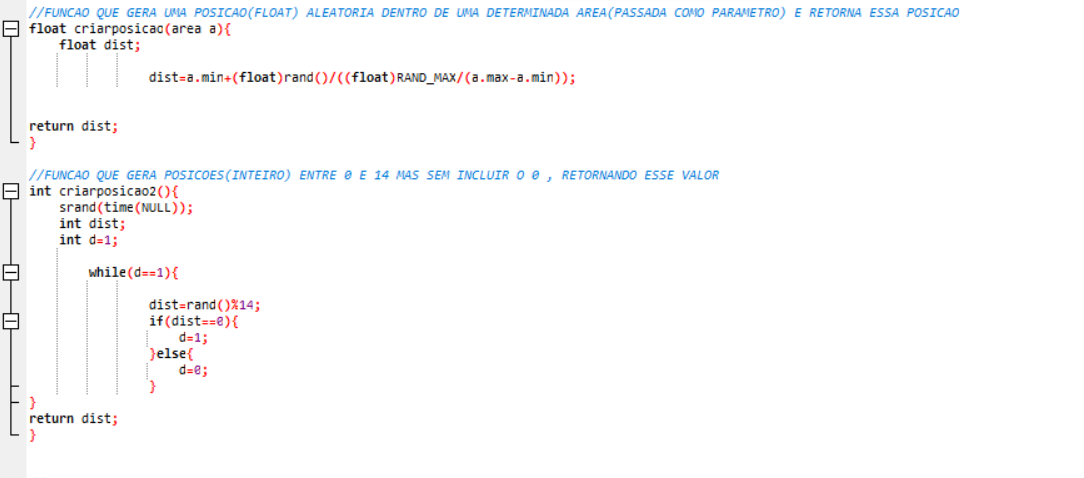
A struct jogador2 é utilizada na “morte súbita” pois tem aquilo que é preciso para a elaboração da mesma .

* Função printarea



Função que recebe uma área e imprime na consola.

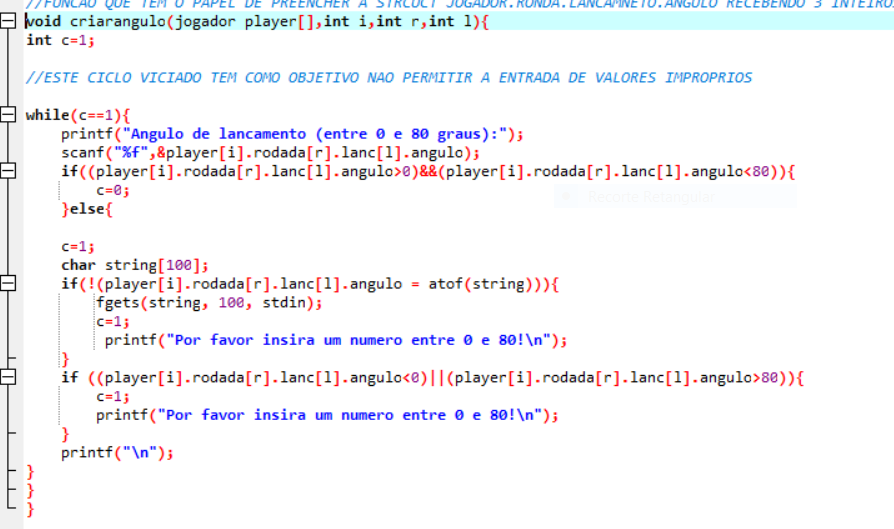
* Função criarposicao e criarposicao2

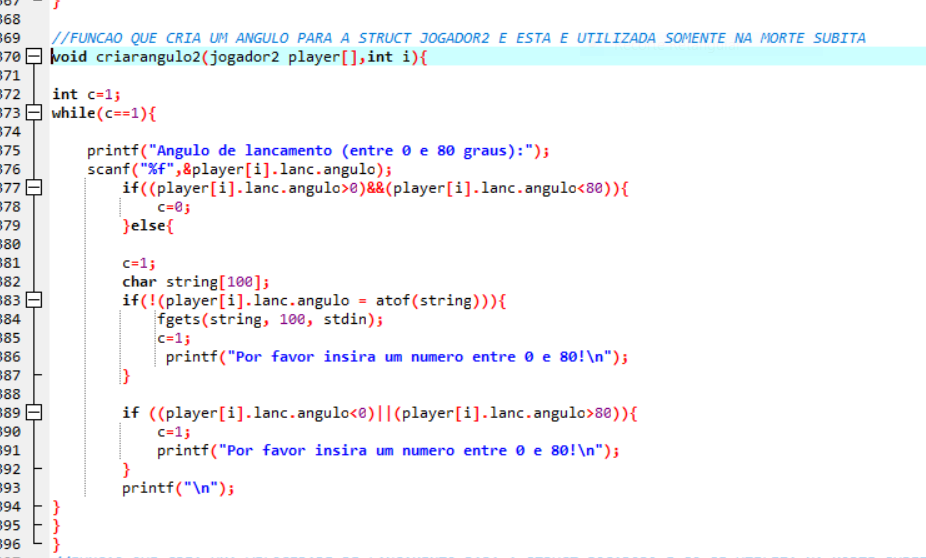


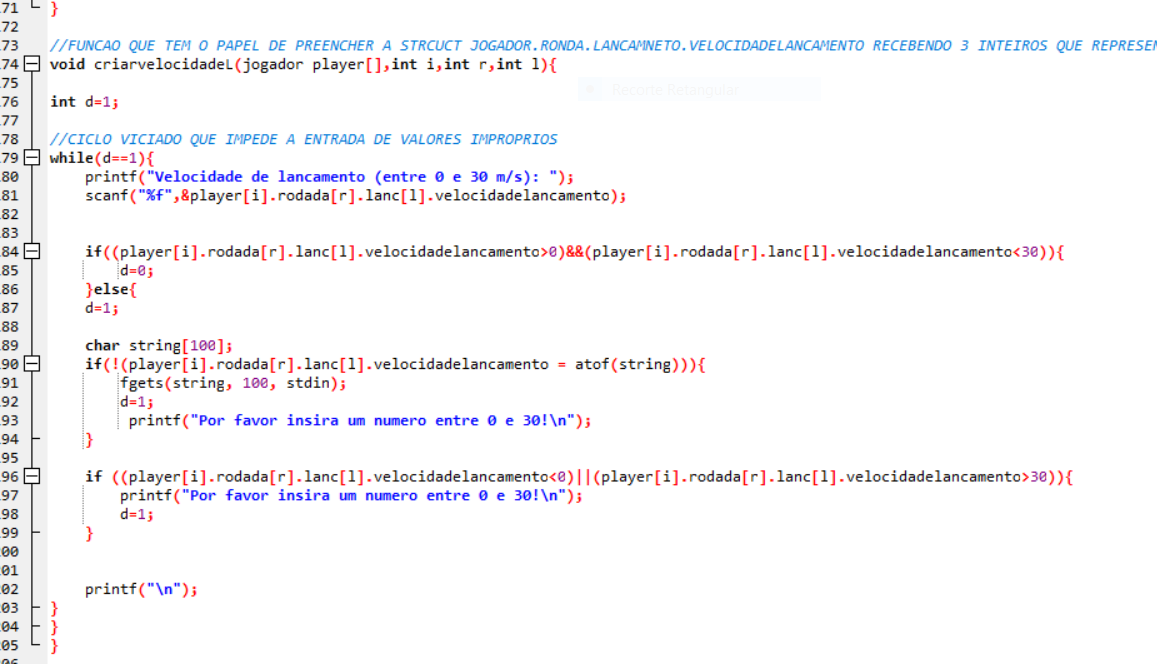
A criarposicao recebe uma área e retorna um valor aleatório em Float entre os limites máximo e mínimo da respetiva área.

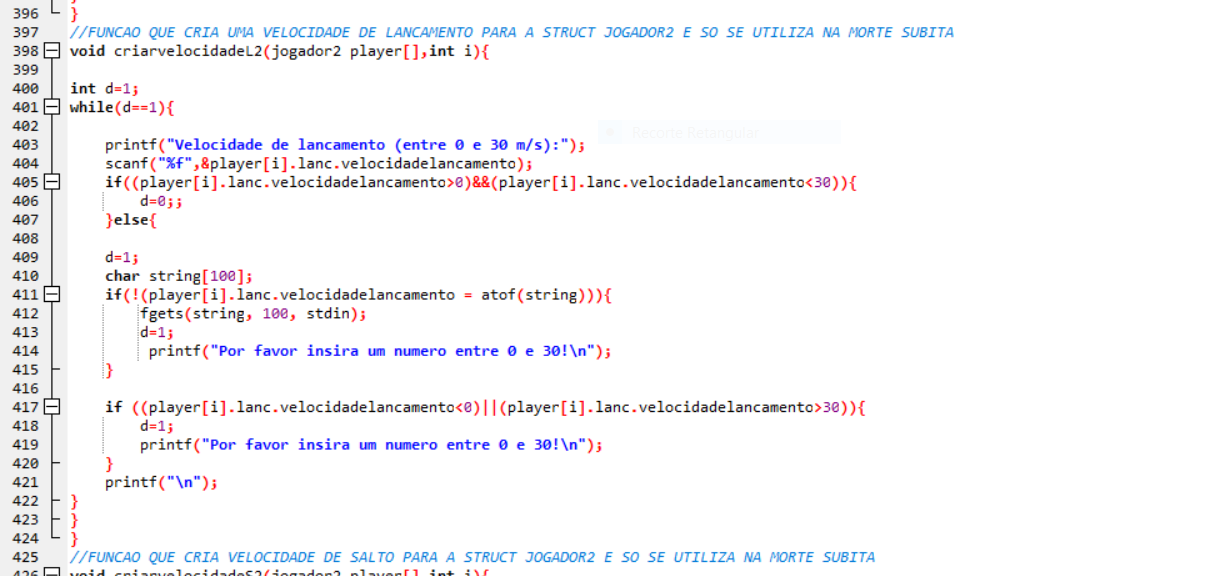
A criarposicao2 não recebe nada como parâmetro , apenas retorna um valor aleatório Inteiro entre 0 e 14 sendo estes os limites do campo total de basquete (não pode gerar o valor 0 e é utilizada na função “morte súbita”),alem disso colocamos a posição gerada por esta função ,como inteiro para simplesmente, se tornar mais fácil encestar e tornar o jogo mais apelativo.

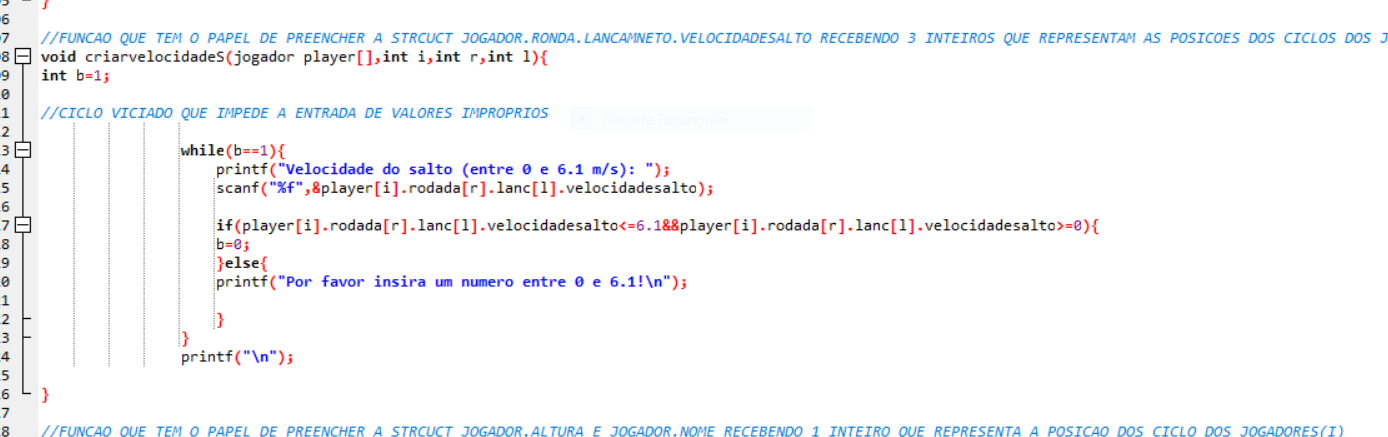
* Função criarangulo , criarangulo2, criarvelocidadeL, criarvelocidadeL2, criarvelocidadeS, criarvelocidadeS2

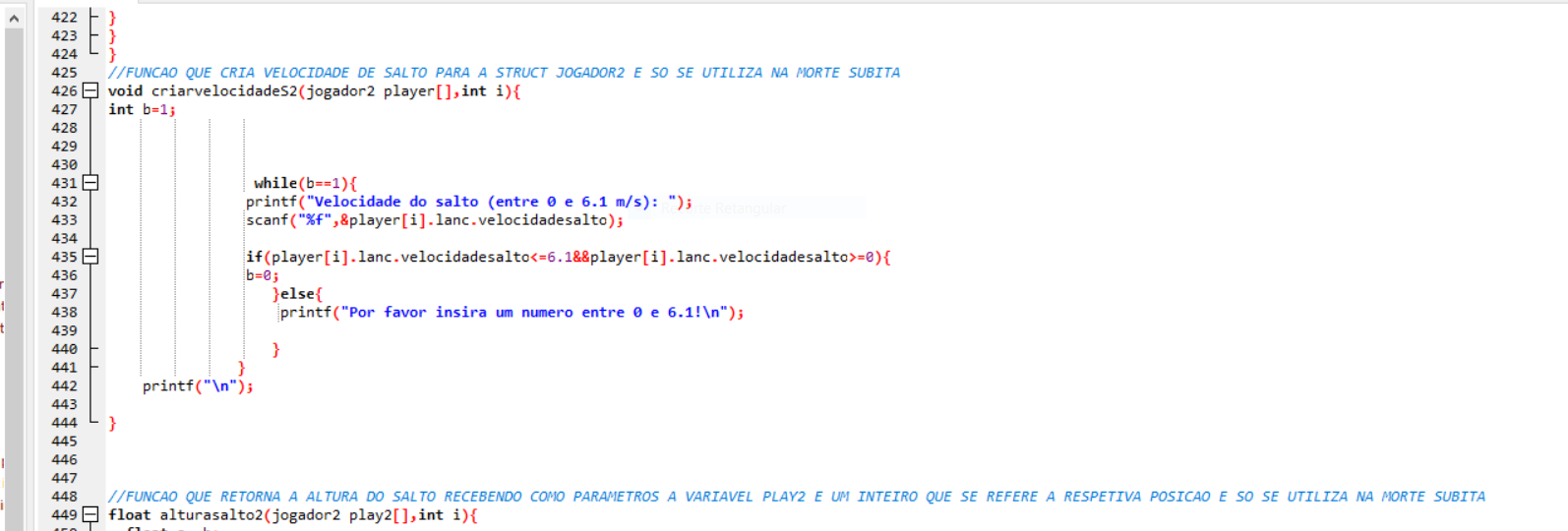












A função criarangulo e criarangulo2 servem para preencher o angulo na struct jogador e na struct jogador2 respetivamente, sendo que este preenchimento ocorre dentro de ciclos e por isso tem de ser fornecidas como parâmetro na criarangulo a respetiva struct (jogador player )e as posições onde o ciclo vai tanto para o ciclo dos jogadores como o ciclo das rondas como para o ciclo dos lançamentos.

Depois a função encarrega se por preencher esse parâmetro respetivamente ao angulo.

Alem disso impedimos a possibilidade de ser sabotado o programa caso o utilizador insira caracteres, ou valores do limite definido.

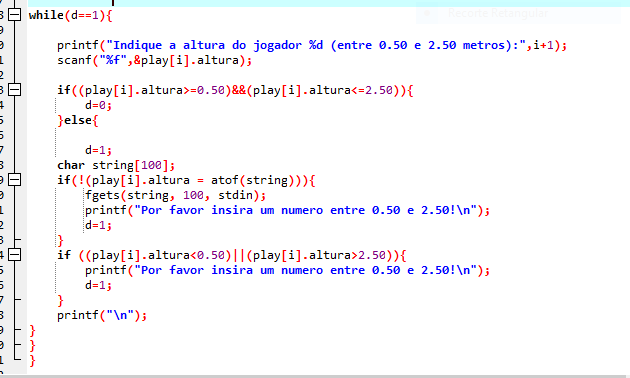
Na função criarvelocidadeL e criarvelocidadeL2 funciona da mesma maneira que as criarangulo e criarangulo2 só que agora no que diz respeito a velocidade de lançamento.

A função criarvelocidadeS e criarvelocidadeS2 funciona da mesma maneira que as anteriores só que agora no que diz respeito á velocidade de salto.

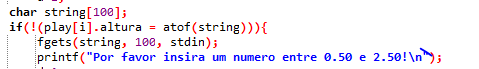
* Função caracteristicasjogador

Esta função tem como objetivo preencher a altura e nome de um jogador na função cíclica que diz respeito aos jogadores, recebendo como parâmetro o array player(struct jogador) e a posição no ciclo(i).

Na realização desta função tivemos o cuidado de evitar a sabotagem do programa, utilizando este ciclo infinito :

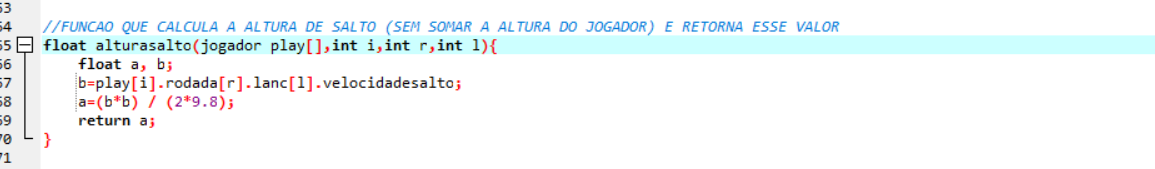


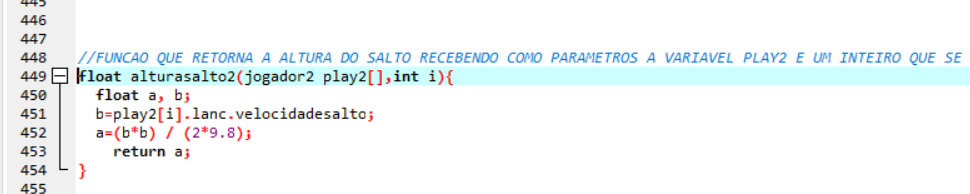
Em que só permite que o ciclo acabe caso os valores introduzidos estejam dentro dos limites estabelecidos e caso não sejam caracteres com o uso desta condição :



Que caso se verifique então não sai do ciclo pois os valores introduzidos foram caracteres.

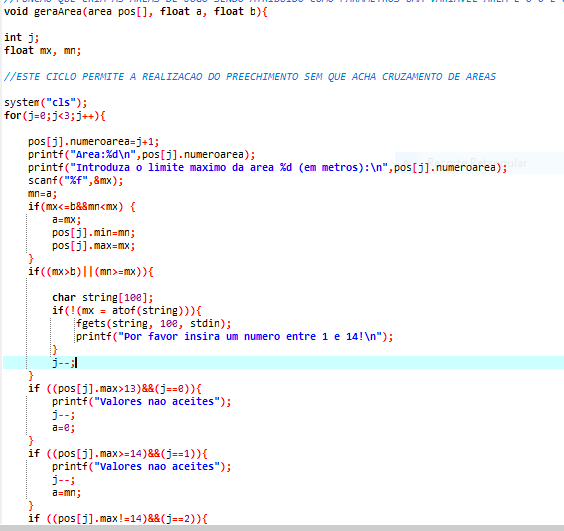
* Função alturasalto e alturasalto2

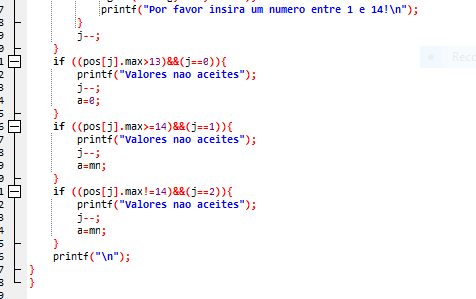




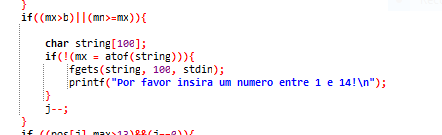
Estas duas funções retornam a altura de salto do jogador recebendo a função alturasalto, o array play(struct jogador) e as posições nos respetivos ciclos, referentes aos jogadores, ronda e lançamento , a função alturasalto2 retorna também o valor da altura do salto só que recebe como parâmetros o array play2(struct jogador2), e o valor da posição do ciclo dos jogadores (esta utiliza se na “morte súbita”).

* Função geraarea



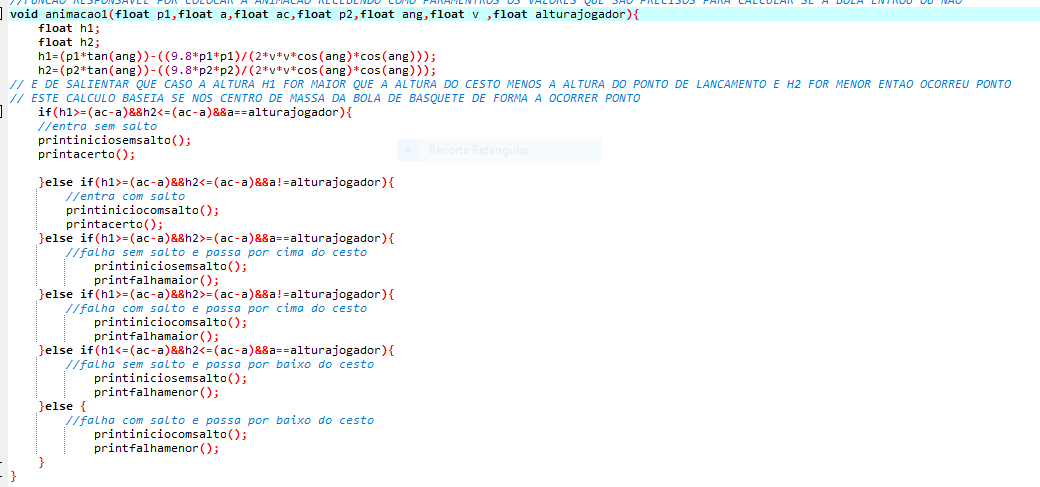


Esta função recebe como parâmetros um array designado por “pos” (struct área) e dois inteiros que são a=0 e b=14 que representam os limites mínimo e máximo que o campo pode ter, esta função vai preenchendo o array área sem que os limites das áreas se possam cruzar e os pontos de cada área são atribuídos da seguinte forma, na área[0],vale 1 ponto , na área [1] vale 2 pontos e na área[2] vale 3 pontos. Também tivemos em atenção o facto de por exemplo, ao gerar a área 1 esta não podia ter o limite máximo igual a 14 uma vez que se isso acontecesse, as restantes áreas não podiam existir dentro do limite do campo, na área 2 tivemos exactamente o mesmo cuidado uma vez que se esta tivesse o limite máximo igual a 14 então a área 3 não podia existir dentro do campo, na área 3 caso o limite máximo colocado fosse diferente de 14 então era pedido de novo esse limite até ser colocado 14.

Outro cuidado que tivemos foi evitar a sabotagem do programa quando se introduz caracteres, utilizando esta condição:

Que caso seja introduzido um caracter então é novamente pedido o limite máximo da posição em questão.

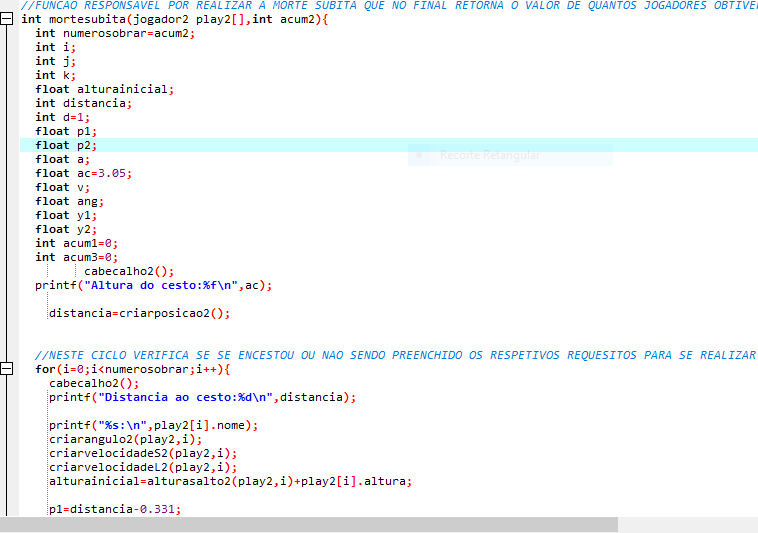
* Função animacao1

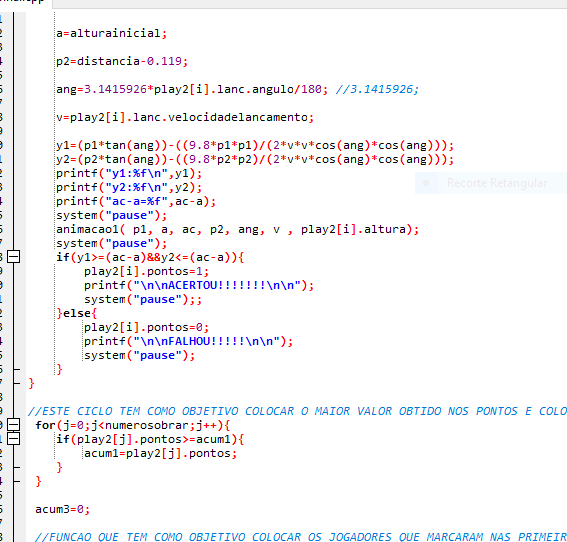


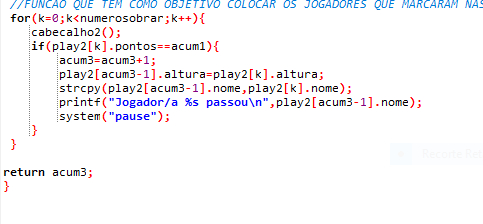
Esta função tem como objetivo introduzir uma animação durante o lançamento, recebendo como parâmetros os dados de lançamento que são necessários para colocar a animação respetiva , por exemplo, se a bola for por cima do aro e o jogador saltar então é colocada uma animação em que o jogador salta e a bola vai por cima do cesto, assim a recursividade desta função permite saber qual é o caso concreto do tal lançamento.

(NOTA: Para verificar se a bola entrou teve se em conta o centro de massa da bola , verificando que caso nos respetivos limites horizontais (p1 e p2) ,substituídos na expressão da altura, estejam para cima da altura do cesto menos a altura de lançamento em p1 e abaixo dessa altura em p2).

* Função mortesubita

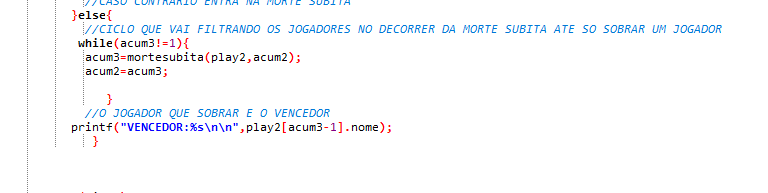






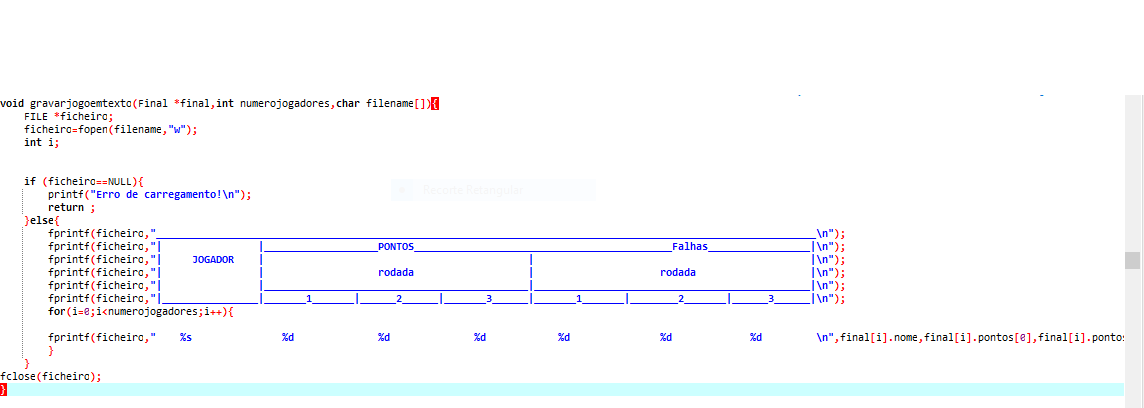
Esta função recebe um array play2(struct jogador2 play2 )e o numero de posições á qual vai percorrer nesse array e tem como objetivo retornar o numero de posições que deve percorrer no array play2 e reorganizar essa struct de forma a que os jogadores que passaram na morte súbita seja colocado nas primeiras acum3 posições ,sendo esse acum3 o valor que vai ser retornado.

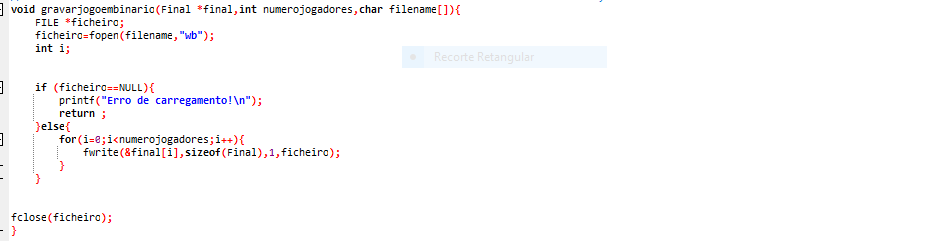
E de salientar que mal acabe o ciclo que preenche os pontos ,ou seja, se acertar o play2[i].pontos=1 e se falhar fica play2[i].falhas=0.

Depois é realizado outro ciclo que coloca no acum1 o maior valor dos pontos obtidos pelos jogadores (0 ou 1), pois vamos supor que todos os jogadores sujeitos a “morte súbita” falham o lançamento, então o acum1 fica igual a 0, e caso algum deles acertar então o acum1 fica igual a 1.Depois de saber o acum1 e realizado outro ciclo que percorre jogador a jogador e acumula no acum3 o numero dos jogadores que obtiveram a pontuação igual ao acum1,sendo colocados na posição acum3-1 do play2 a altura e nome do jogador que se encontrava na posição definida pelo ciclo , e por isso faz se a seguinte igualdade(play2[acum3-1]-altura=play2[posição no ciclo].altura) de forma a no final o play2 ser modificado, sendo que os jogadores que passaram na” mortesubita” passam para as primeiras acum3 posições do play2 e mal acabe este ciclo e retornado esse acum3 pela função “mortesubita”, isto tudo porque na main, depois de verificar que no final das 3 rondas houve empate entre jogadores, estes jogadores que foram colocados no array play2(struct jogador2) vão ficar sujeitos a um ciclo infinito:

Onde ocorre uma recursividade em que caso o acum3 seja diferente de 1 então o acum3 é igualado á função mortesubita que recebe o array play2 e o acum2 que é o numero de jogadores que empataram sendo que esses já vem colocados nas primeiras acum2 posições do array play2 (este procedimento será mais aprofundado quando for explicada a main) , depois o valor retornado pela função mortesubita fica sendo o acum3 e caso este ainda seja diferente de 1 será realizado de novo este ciclo infinito, sendo que a função mortesubita já reorganiza os jogadores que passaram na “mortesubita” nas primeiras acum3 posições do array play2, no final depois de serem “filtrados” os jogadores que foram eliminados na “mortesubita” ate sobrar um, então o ciclo acaba e temos o vencedor.

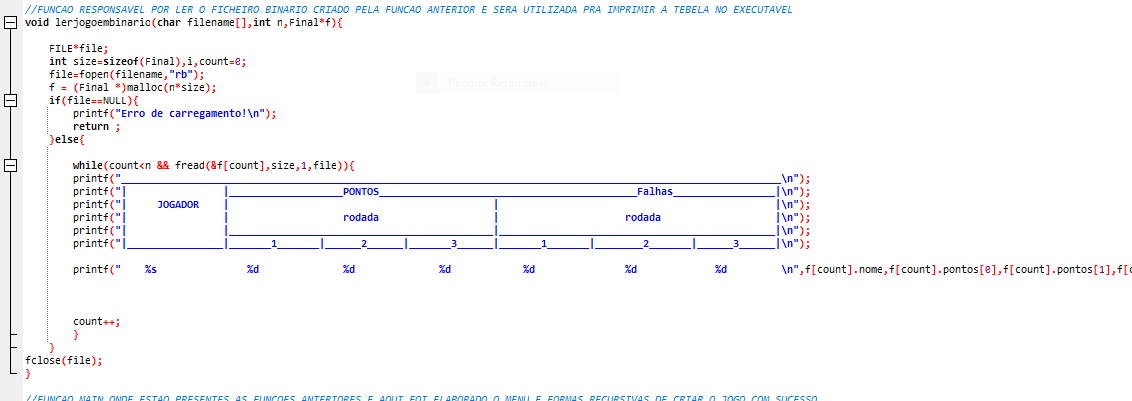
* Função gravarjogoemtexto e gravarjogoembinario





Estas duas funções têm como objectivo gravar o apontador final (struct Final), recebendo este apontador, o numero de jogadores e o nome do ficheiro e no final, a função gravarjogotexto grava num ficheiro texto a tabela com o nome , falhas e pontos por ronda de cada jogador e a função gravarjogoembinario grava o apontador final recebendo os mesmos parâmetros que a anterior .

* Função lerjogoembinario



Esta função tem como objectivo ler um arquivo binário e imprimir os seus dados numa tabela (usa se no menu da tabela de pontuações),recebendo como parâmetros o apontador onde se vai armazenar os dados lidos do ficheiro , o numero de jogadores e o nome do ficheiro. Esta função lê até uma das condições do if falhar.

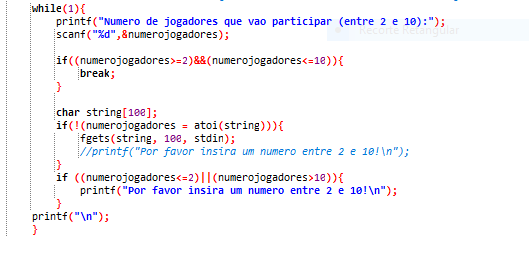
De seguida será explicado o processo realizado na main:

A main começa com um ciclo viciado para que se forem introduzidos números que não estejam associados a alguma funcionalidade, ela torna a imprimir o menu e a pedir para seleccionar uma opção, esta parte também esta protegida quando se introduzem caracteres, com a mesma estratégia que nas outras funções onde isto foi utilizado.

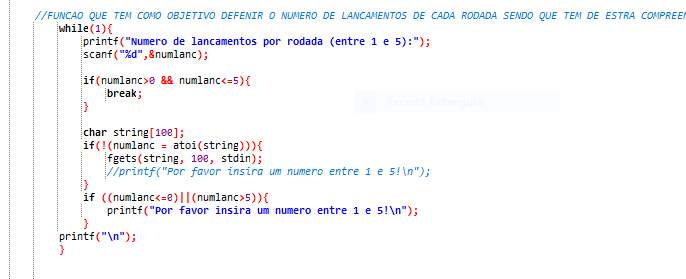
Se no menu for introduzido o valor 1 então entra no modo multijogador. Neste modo o jogo é inicializado sendo necessário preencher diversos parâmetros, o primeiro e a criação das áreas de jogo, para a sua criação é usada a função geraArea(pos,a,b) que foi explicada em cima(o a foi iniciado na main como sendo igual a zero e o b igual a 14), depois de preenchido o array pos(struct área) que foi colocado na função gerarArea, então as áreas estão definidas e serão constantes durante todo jogo.

De seguida usa se um ciclo que imprime as áreas de jogo na consola usando a função printarea.

Depois é realizado um ciclo infinito que tem o objectivo de preencher o numerojogadores, que é uma variável onde, como o próprio nome indica, é colocado o numero de jogadores que vão participar no jogo, para que o ciclo infinito termine e a variável assuma um valor , então o valor introduzido tem de estar entre 2 e 10 (limites definidos no jogo), ou seja para isso acontecer, mal é scaneado o valor pretendido, esse valor vai ser verificado , em primeiro lugar, por uma condição que só cancela o ciclo caso o valor esteja entre 2 e 10, caso os valores não correspondam a esse limite então o ciclo repete-se(os caracteres foram também impedidos de sabotar o programa utilizando o método anteriormente referido).

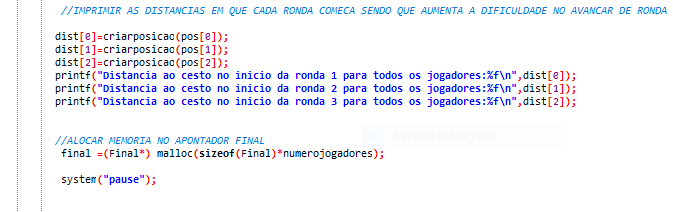


De seguida é realizado um ciclo infinito que tem o objectivo de preencher o numlanc, que é uma variável onde é colocado o numero de lançamentos que vão ser realizados por ronda, para que o ciclo infinito termine e a variável assuma um valor , então o valor introduzido tem de estar entre 1 e 5 (limites definidos no jogo), ou seja para isso acontecer, mal é scaneado o valor pretendido, esse valor vai ser verificado , em primeiro lugar, por uma condição que só cancela o ciclo caso o valor esteja entre 1 e 5, caso os valores não correspondam a esse limite então o ciclo repete-se(os caracteres foram também impedidos de sabotar o programa utilizando o método anteriormente referido).



Posteriormente, foi preenchido para o array dist[3] ( que vai ser o array que contem as posições iniciais ,ou seja , as posições onde os jogadores começam quando mudam de ronda , sendo que na primeira ronda começa numa posição aleatória da área 1 ,dist[0], na segunda ronda começam numa posição aleatória da área 2,dist[1], na terceira ronda começam numa posição aleatória na área 3,dist[2], sendo que estas distancias são as mesmas para todos os jogadores no começo de cada ronda) utilizando a função criarposicao descrita acima.

Depois, faz se uma alocação de memoria para o apontador final (struct Final),com o tamanho relativo ao numero de jogadores.



De seguida é realizado um ciclo onde percorre os jogadores do array play(struct jogador), onde é realizado o preenchimento da altura do jogador que se encontra na posição dada pelo ciclo, através da função característicasjogadores(play,i) onde i é a posição fornecida pelo ciclo.

Mal acabe a etapa do preencher características dos jogadores, é atribuído o nome do jogador e altura ao apontador final, na posição fornecida pelo ciclo, usando para o nome ,a função contida numa biblioteca do C, chamada strcpy e para o a altura bastou uma igualdade.

Depois e é iniciado o ciclo das rondas para cada jogador(o ciclo das rondas esta dentro do ciclo dos jogadores), onde para cada ronda, numa dada posição fornecida pelo ciclo das rondas, são inicializadas as variáveis np (numero de pontos) e nf(numero de falhas ) iguais a zero, depois é igualada a variável areainicial (struct área) á área pos na posição fornecida pelo ciclo das rondas, de forma a que a medida que a ronda avança, a variável assume uma área cada vez maior (esta variável areainicial vai ajudar na realização dos pontos), de seguida é atribuída á variável ai o valor da distancia presente na posição fornecida pelo ciclo das rondas do array dist(de forma a poder manusear as distancias sem nunca alterar o array dist), de seguida o acum e inicializado como zero para ao mudar de ronda não haver valores já inicializados da ronda anterior. Depois começa o ciclo dos lancamentos(que esta dentro o ciclo das rondas), onde para cada lançamento na posição dada pelo ciclo preenche o angulo do array play(play[i].rodada[r].lanc[l].angulo) em que i é a posição do ciclo dos jogadores, r posição do ciclo das rondas e l posição do ciclo dos lançamentos) utilizando a funçao criarangulo, a velocidade de salto e a velocidade de lançamento são preenchidas da mesma forma so que usando as funções criarvelocidadeS e criarvelocidadeL ,repetivamente.

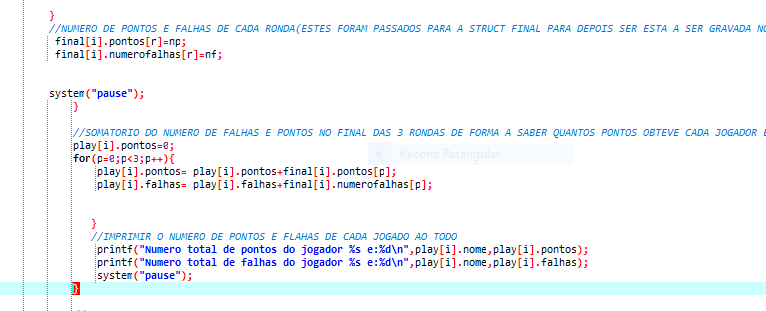
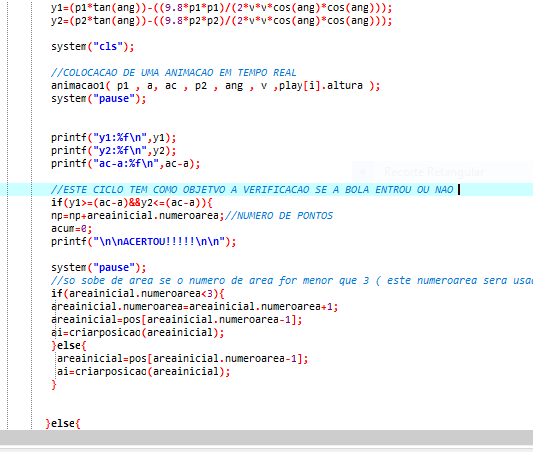
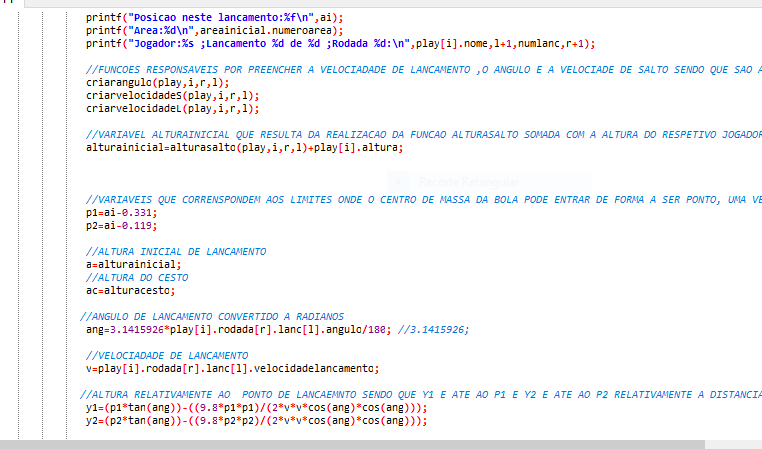
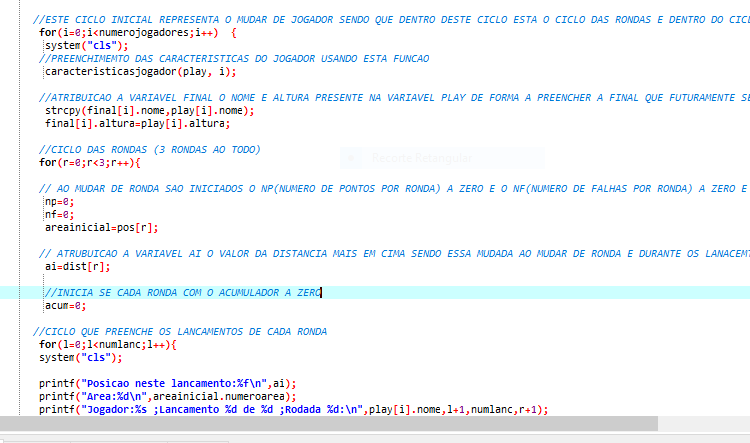
Depois é determinado o valor da variável alturainicial usando a função alturasalto(play,i,r,l) ,descrita mais acima, somando com a altura do jogador.

Depois a realização dos pontos baseia se quando o centro de massa da bola entra nos respectivos limites presentes no cesto (limite p1 e p2) ,onde pode haver ponto, e utiliza se a formula do enunciado. Para verificar se houve ponto , utilizou se a tal formula só que para a distancia p1 (substituída na formula) tem de ser maior ou igual a ac-a (ac=altura do cesto e a=altura de lançamento ) e para a p2 tem de ser menor ou igual a ac-a.

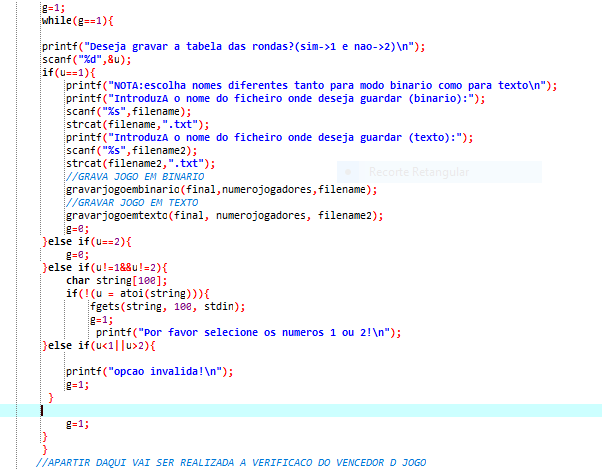
É também colocada a animação através da função animaçao1 que recebe os parâmetros necessários para saber se houve cesto ou não e coloca a animação apropriada na consola.

Para calcular o np , é incrementado ao np anterior o valor dos pontos da área de lançamento (np=np+areainicial.numeroarea) e o acum e igualado a zero,depois verifica se que caso enceste e a área seja abaixo da área 3(área 2 e 1) sobe se para a área acima somando á variável areainicial.numeroarea, o valor 1, e logo depois pega se no areainicial.numeroarea e atrubui se á variável areainicial a área adequada do array pos(areainicial=pos[areainicial.numeroarea-1]), depois atribui se um novo valor aleatório para o ai que sera usado na próxima ronda, caso esteja na área 3 não sobe de área e por isso apenas se gera um valor aleatório para o ai. No caso de falhar , o raciocínio e basicamente o mesmo , só teve em conta que se incrementa o nf com o valor 1 (nf=nf+1), caso falhe e o acumulador seja 1( sendo a área superior ou igual á área 1) então incrementa se o acumulador , se falhar e o acumulador já for 1 e a área maior que 1, então o acumulador é zerado e desce de área criando uma nova posição para o ai nessa área , caso o acumulador seja 1 e a área seja 1 entao o acum é zerado e não baixa de área, apenas e gerado um novo ai dentro dessa área. No final de todos os lançamentos é atribuído ao apontador “ final “o valor dos pontos naquela rondam (final[i].pontos[r]=np) e o valor das falhas (final[i].numerofalhas[r]=nf).

No final das rondas de cada jogador, e calculado o total de pontos e falhas realizadas nas três rondas usando um ciclo que vai somando ao play[i].pontos o valor dos pontos ganhos em cada ronda e ao play[i].falhas o valor das falhas feitas em cada ronda. Terminado o ciclo o play já contém os pontos e falhas que cada jogador obteve durante as rondas.

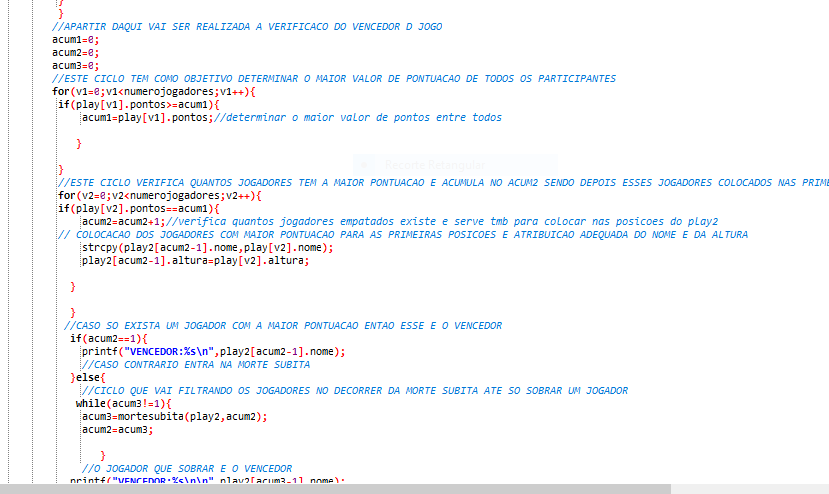


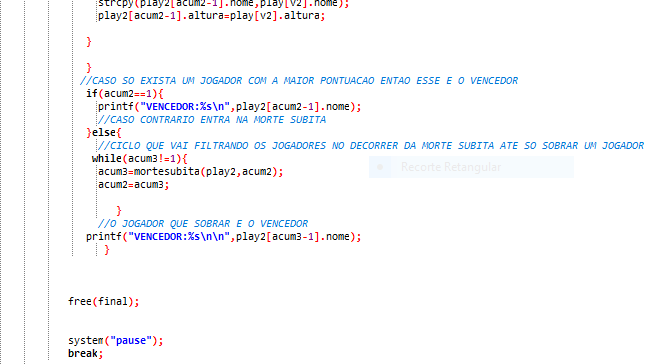
No final de todos os jogadores realizarem os lançamentos (ciclo dos jogadores acabar) é pedido se quer gravar o jogo e caso queira so tem de colocar o nome do ficheiro binário e texto que desejar(devem ser diferentes).Caso o usario digitar uma opção ou um valor improprio então o ciclo infinito onde se encontra esta parte volta ao inicio.



Depois disso é realizada a determinação do vencedor do jogo, em que inicialmente os acum1, acum2 e acum3 são zerados e é realizado um ciclo onde é colocado no acum1 o valor da maior pontuação obtida entre todos os jogadores, depois é realizado outro ciclo que verifica se o jogador ((play[v2].pontos)=acum1) obteve a maior pontuação obtida em cima. Caso obtenha, então é incrementado o acum2 e são colocadas a altura e o nome no array play2(struct jogador2) nas posições dadas pelo acum2 e esses valores provêm do play na posição dada pelo ciclo. De seguida é verificado se o acum2 é igual a 1. caso isso aconteça, é impresso o nome do vencedor, pois apenas um obteve a pontuação máxima, caso contrário entra num ciclo viciado que só acaba quando o acum3 for igual a 1 e para isso usa-se o raciocínio explicado na função “mortesubita” explicada a cima.

Quando esse ciclo acaba imprime se igualmente o vencedor que “sobreviveu” à morte súbita. Não esquecendo de no final de tudo fazer o free(final).





Encerrando o modo multijogador.

Modo solo:

O modo solo funciona exactamente com o mesmo raciocínio e métodos que o modo multijogador só que o numero de jogadores e obrigatoriamente 1 e não é determinado vencedor.

Modo tabela de pontuações:

Este tem como objectivo carregar o ficheiro binário sem ter de sair do jogo, imprimindo a tabela de pontuações. Para isso é pedido para digitar o nome do ficheiro e utiliza-se a função lerjogoembinario que recebe como parâmetros o nome do ficheiro, o valor dez, pois é o numero maximo de jogadores que podem participar num jogo e o apontador f(struct Final) onde guarda esses dados, onde guarda esses dados e imprime na consola a dada tabela (a função utilizada está explicada em cima). Sem esquecer de fazer o free(final);

Modo sair:

Cancela o programa, fechando-o;

Conclusão

Objetivos atingidos:

Neste trabalho conseguimos, utilizando as leis da física, juntamente com os nossos conhecimentos em C, criar um jogo de basquetebol realista e fácil de jogar, obedecendo à grande maioria dos critérios propostos.

Dificuldades apresentadas:

Durante a execução do trabalho deparamo-nos com imensas dificuldades no que tocava a evitar a sabotagem do programa quando se coloca caracteres quando se espera um numero, porem, com algumas pesquisas, obtivemos um método que resolve quase todas as situações, tornando o jogo mais seguro e funcionavel.

Obtivemos dificuldades na tentativa de colocar as variaves de forma dinâmica na memoria do programa, sendo que so conseguimos aplicar apontadores em certas partes do programa.

Para além disso, um dos grandes problemas é conseguir encestar devido à dificuldade que isso apresenta, pois fisicamente é difícil imaginar e calcular o angulo, a velocidade de salto e lançamento apropriados.

Na elaboração deste relatório não sabíamos muito bem o que era para fazer em alguns tópicos e por isso decidimos explicar o nosso raciocínio pormenorizado e as funções também.