

Senac

Simulação de Canhão, Avião e Radar

PI-V

CELSO VENANCIO LEITE
24/5/2016

Sumário

| | |
|---------------------------------|----|
| Resumo..... | 2 |
| Introdução..... | 3 |
| Revisão | 3 |
| Plano..... | 3 |
| Equações Cinemática | 3 |
| Trajetória do Avião..... | 3 |
| Trajetória Balística..... | 3 |
| Ângulo Azimute e Elevação | 4 |
| Método..... | 4 |
| Calculando a Interceptação..... | 6 |
| Resultado..... | 6 |
| Discussão..... | 10 |

Resumo

Este projeto tem como objetivo implementar um simulador de defesa antiaérea no qual um avião tenta atingir um alvo, enquanto, um canhão tenta abatê-lo. Uma aplicação cliente-servidor para simular um radar (servidor) que detecta e informa a posição exata do avião em espaço aéreo não autorizado. Outra aplicação simula o canhão (cliente) que recebe a posição do avião com algum atraso devido ao meio físico e calcula um ponto onde o avião será abatido, informando ao radar sobre o disparo para que este possa verificar se o avião é abatido ou não. O radar exibe toda a simulação em 3D utilizando *OpenTK*, uma implementação de *OpenGL* para *.NET*, e logs sobre as posições e distância entre os objetos são exibidos simultaneamente. O canhão consegue atingir o avião quando o avião é representado por uma esfera de cinco metros de raio simulado numa rede em que o *ping* entre os computadores é de dois milissegundos em média.

Introdução

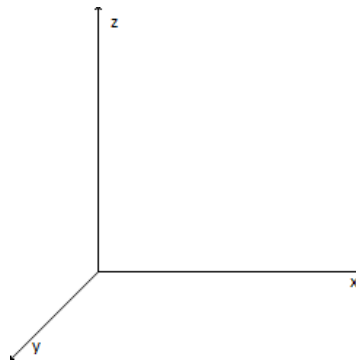
Este relatório apresenta o desenho, a implementação e simulação de um radar que detecta um avião inimigo tentando destruir um alvo. Enquanto um canhão tenta abatê-lo identificando sua trajetória e definindo o melhor momento para disparar seus quatro tiros disponíveis. O avião (uma esfera de raio igual a dois metros no espaço) é abatido quando o tiro (um ponto no espaço) o atinge. O avião destrói o alvo quando chega a menos de um quilometro do alvo. Dois aplicativos foram implementados uma para representar o radar que informa a posição do avião aproximadamente quatro vezes por segundo ao canhão. O canhão, a partir destas posições, tenta definir um ponto para tentar interceptar o avião com um tiro. Aguardando o melhor momento para informar o radar que o tiro foi disparado, considerando o tempo necessário para que a informação chegue ao radar. Como a precisão necessária para que o canhão atinja o avião é muito alta, considerar adequadamente o tempo gasto na comunicação para informar o disparo é um fator decisivo para que este acerte seu alvo.

Revisão

Para compreensão do projeto é necessário a compreensão de alguns conceitos matemáticos. Esses conceitos são apresentados a seguir.

Plano

Para aplicações que projetam objetos no espaço tridimensional é importante definir qual eixo representa altura, profundidade e largura. Todas as equações de agora em diante utilizaram os eixos como demonstrado na figura abaixo.



Podemos ver que o eixo Z é representa o eixo vertical. Isso é importante para as equações a seguir.

Equações Cinemática

Trajetoória do Avião

Falaremos muito sobre a trajetória do avião que foi definido como uma reta no espaço. A seguinte equação define esta trajetória no instante t ,

$$\vec{s}(t) = \vec{s}_0 + \vec{v}t$$

Onde \vec{s}_0 é a posição inicial do avião, \vec{v} é o vetor velocidade do avião.

Trajetoória Balística

A trajetória dos tiros é definida pela equação,

$$\vec{s}(t) = \vec{s}_0 + \vec{v}t - \frac{\vec{g}t^2}{2}$$

Onde \vec{s}_0 e \vec{v} equivalem a posição inicial do tiro e o vetor velocidade do disparo

respectivamente assim como no caso do avião e o vetor constante $\vec{g} = \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix}$, onde g é a aceleração da gravidade.

Ângulo Azimute e Elevação

A trajetória definida pela equação acima só é útil se o vetor velocidade estiver definido. Precisaremos defini-lo a partir de um ângulo azimute e elevação com magnitude definida em 1.175m por segundo. Dado os ângulos azimute α e elevação β , calcularemos o vetor velocidade com magnitude v , como sendo

$$\vec{v} = R_z(\beta)R_y(\alpha) \begin{bmatrix} v \\ 0 \\ 0 \end{bmatrix},$$

$$R_y(\theta) = \begin{bmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{bmatrix},$$

$$R_z(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

Podemos simplificar essa multiplicação de matrizes para apenas

$$\vec{v} = \begin{bmatrix} v \cos \alpha \cos \beta \\ v \cos \alpha \sin \beta \\ -v \sin \alpha \end{bmatrix}$$

Isso dará o vetor velocidade rotacionado em ângulos anti-horário, então podemos ignorar o sinal na última componente do vetor \vec{v} para funcionar no sentido correto já que queremos a elevação. Portanto o vetor final é dado por

$$\vec{v} = \begin{bmatrix} v \cos \alpha \cos \beta \\ v \cos \alpha \sin \beta \\ v \sin \alpha \end{bmatrix}$$

Para desenvolver uma aplicação capaz de receber dados

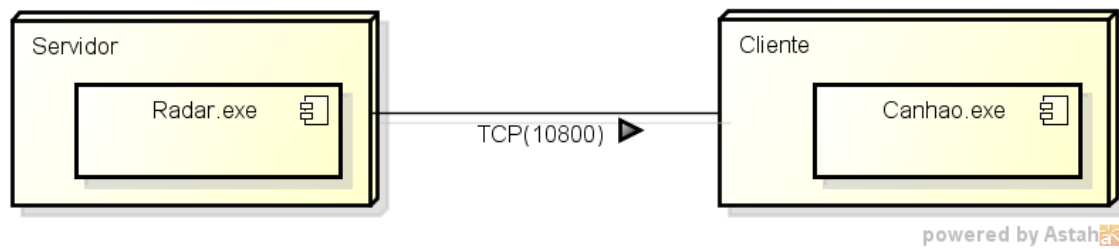
O objetivo do projeto é a experiência no desenvolvimento de aplicativos com comunicação entre computadores utilizando o protocolo TCP/IP. Além do desenvolvimento de interface gráfica em 3D para exibição da simulação

Como primeira entrega foi solicitado um aplicativo cliente (Radar) e outro servidor (Canhão), em que o servidor ao aceitar a conexão do cliente, envia um pacote contendo algum dado e espera outro de volta. O tempo de *throughput* é calculado. Este processo é repetido por 10 interações e o tempo médio é apresentado ao final.

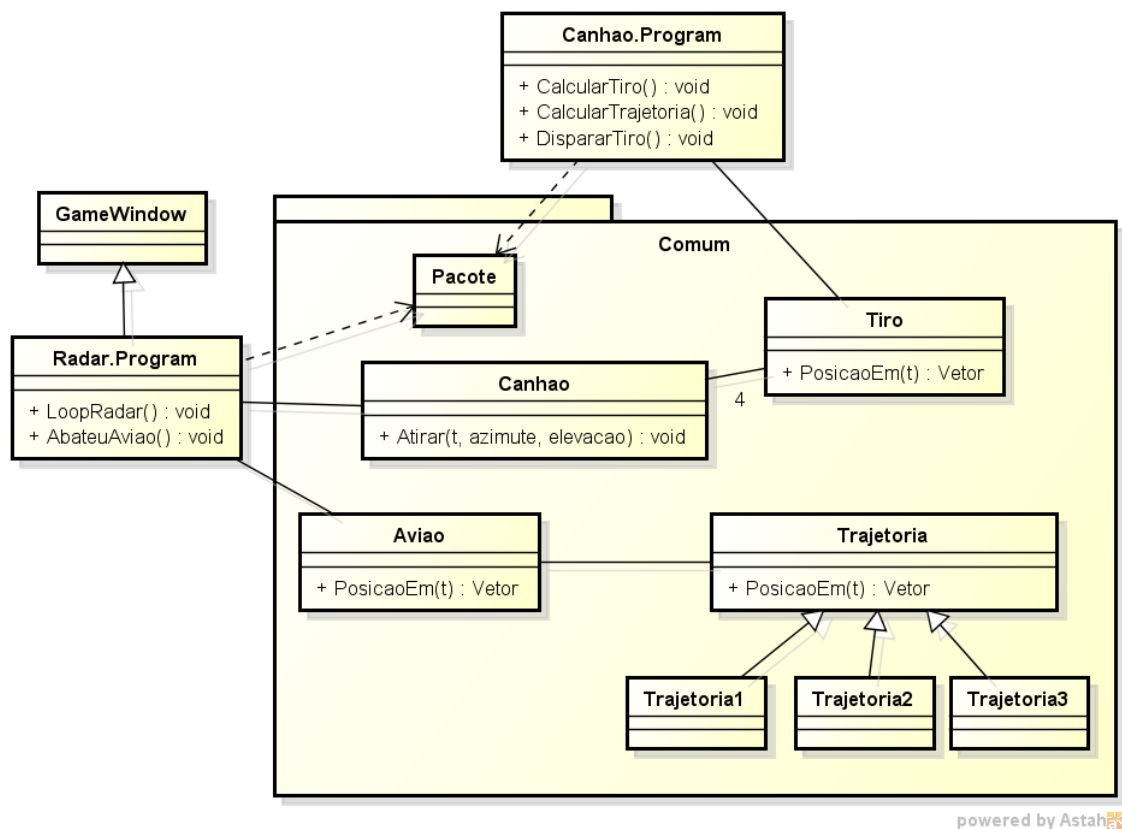
Método

Ambos aplicativo, cliente e servidor, foram desenvolvidos em C# utilizando o .NET Framework v4 no Visual Studio 2013. Para a parte gráfica foi utilizado a biblioteca OpenTK que é um *binding* para .NET do OpenGL.

As duas aplicação devem ser executadas como mostra o diagrama abaixo:



O diagrama que demonstra a estrutura das duas aplicações e a biblioteca que é comum a ambas é demonstrada abaixo:



Pacote É a estrutura que representa os possíveis pacotes enviados por ambas as aplicações. Os tipos são:

- Posição – Informa ao canhão a posição do avião num determinado momento;
- Tiro – Utilizado pelo canhão para informar ao radar que foi efetuado um disparo passando o ângulo de azimuth e elevação;
- AlvoDestruído – Informa ao canhão que o alvo foi destruído;
- AviaoAbatido – Informa ao canhão que um dos disparos acertou o avião;
- Ping – O canhão envia ao radar com o tempo;
- Pong – O radar responde ao Ping com o Pong preenchendo com o tempo recebido no Ping.

Canhao Contém os disparos realizados até o momento e sabe quantos ainda estão disponíveis.

| | |
|-----------------------|--|
| Tiro | Calcula o vetor velocidade utilizando os ângulos azimute e elevação informados e o momento do disparo. Também calcula a posição da bala num momento t. |
| Aviao | Controla qual trajetória deverá ser utilizada seguindo as probabilidades do documento de especificação e a troca das mesmas. Informa também a posição do avião num momento t. |
| Trajectoria | Define o vetor velocidade, altura e etc. sobre uma determinada trajetória conforme o documento de especificação. A trajetória número 3 é a trajetória que representa o avião desistindo do ataque. |
| GameWindow | Classe base da biblioteca OpenTK para criação de telas em OpenGL. |
| Radar.Program | Implementação da lógica do programa do radar. Se comunica com o avião e canhão para saber as posições dos objetos no espaço e os desenham na tela através do OpenGL. Além de cuidar da comunicação com o canhão (TCP) enviando a posição e respondendo solicitações de ping. |
| Canhao.Program | Recebe os pacotes do radar para tentar identificar a trajetória do avião e assim poder calcular o melhor momento e direção para efetuar um disparo na tentativa de abater o avião protegendo o alvo. |

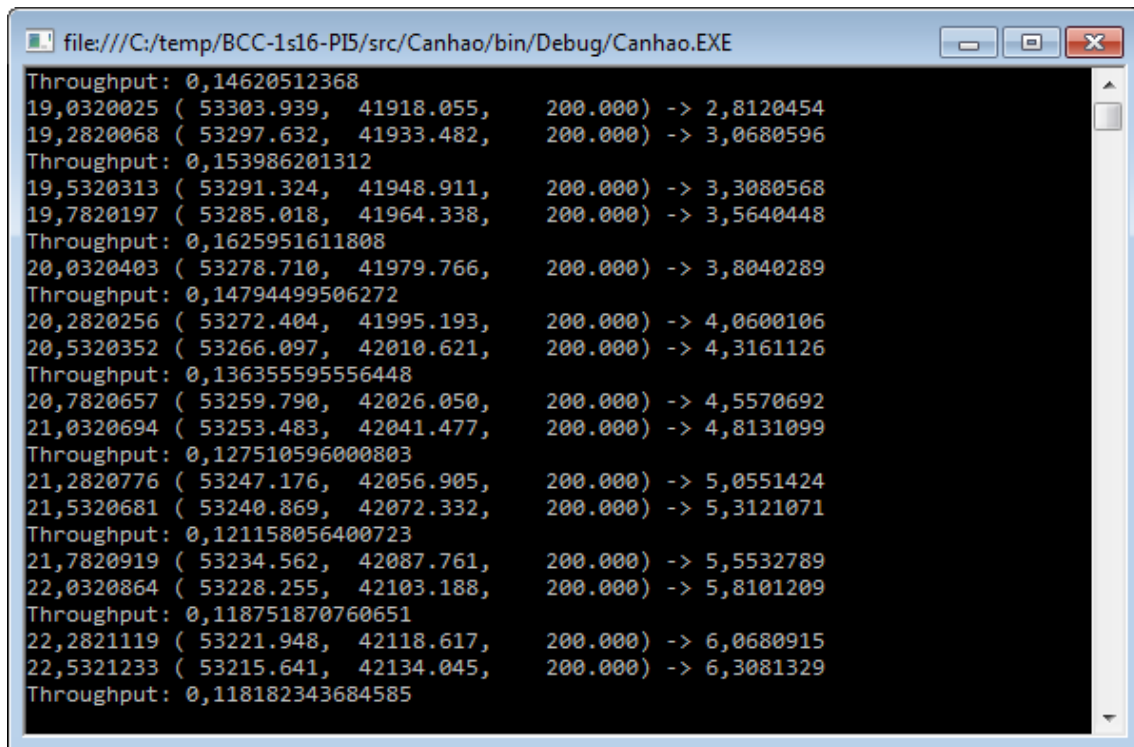
O loop principal da aplicação do radar é controlado pela classe GameWindow do OpenTK. Configurado para executar 60x por segundo. No entanto, somente a cada 4 execuções um pacote com a posição do avião é gerado e enviado a aplicação do canhão. A aplicação do canhão também executa seu loop principal 60x por segundo. Aguardando por pacotes de posição calculando trajetórias e testando o tempo médio do throughput.

Calculando a Interceptação

Resultado

O objetivo de atingir o avião calculando sua trajetória e o momento certo do disparo foi alcançado para um avião de raio igual a 5m para uma comunicação com ping médio de até 2 milissegundos, mas não para 2m como foi especificado. No código há um método implementado para tentar interpolar entre um interação do loop e outra quando for detectado que a trajetória do avião e do tiro se intersectaram, mas sem sucesso. Utilizando este método a precisa é piorada. Provavelmente com mais alguns ajustes este método ficaria bem preciso. Portanto o método utilizado é simplesmente calcular a distância entre o avião e os tiros em cada interação do radar, o que ocorre 60x por segundo.

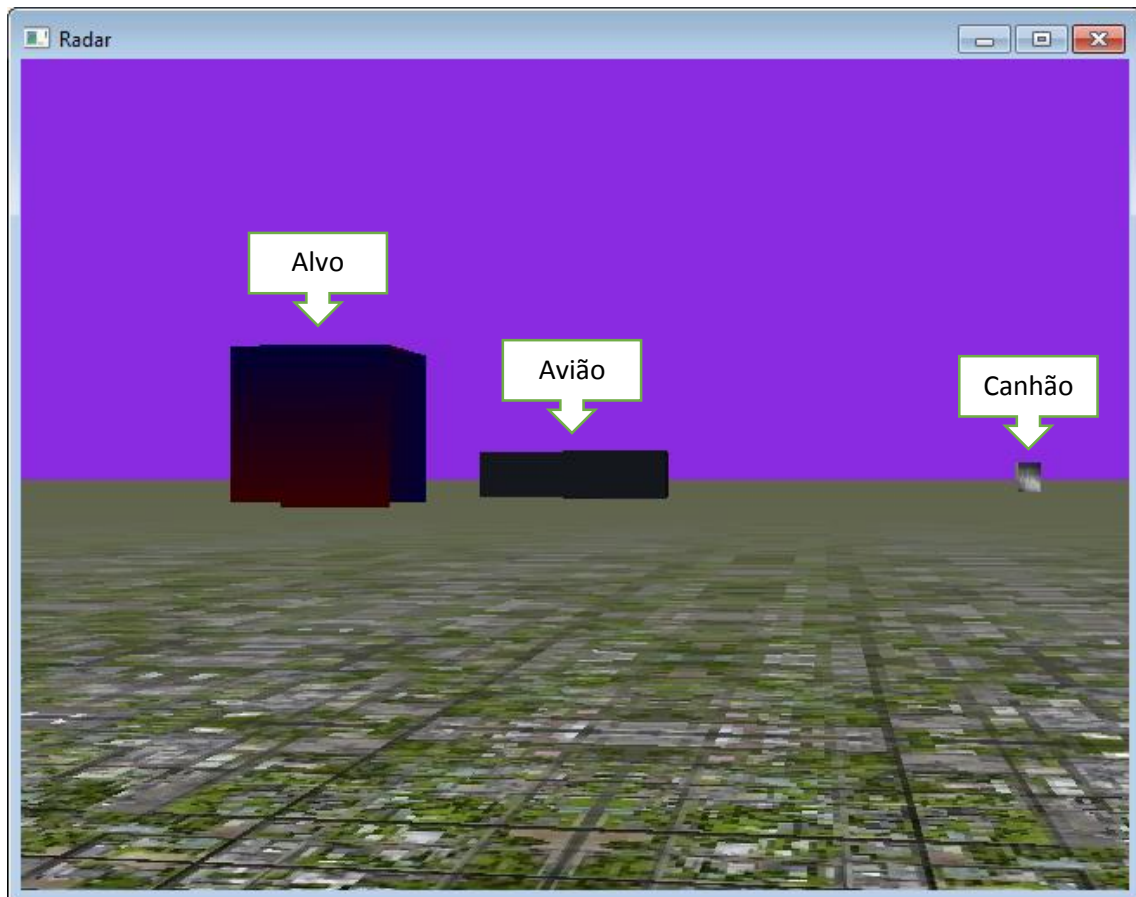
O print abaixo mostram a execução do aplicativo Canhão conectado ao aplicativo Radar



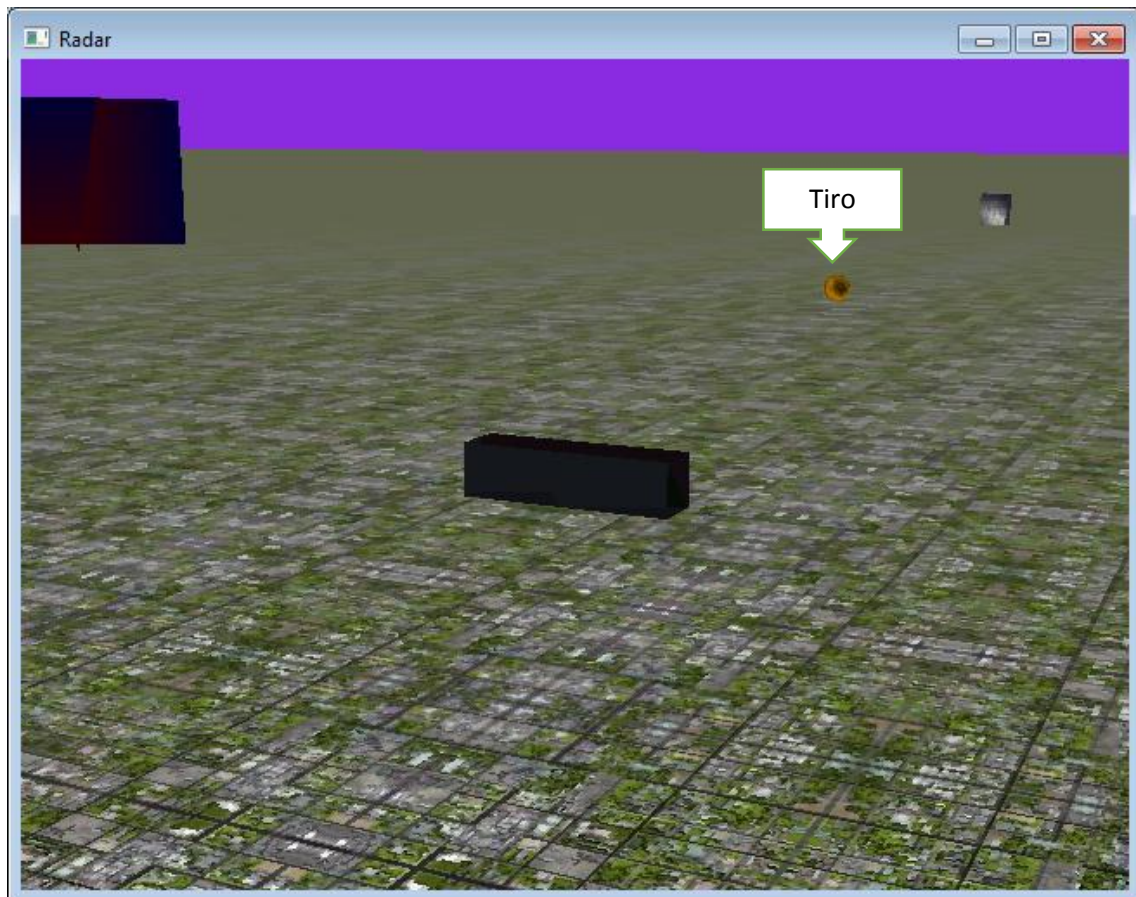
```
file:///C:/temp/BCC-1s16-PI5/src/Canhao/bin/Debug/Canhao.EXE
Throughput: 0,14620512368
19,0320025 ( 53303.939, 41918.055, 200.000) -> 2,8120454
19,2820068 ( 53297.632, 41933.482, 200.000) -> 3,0680596
Throughput: 0,153986201312
19,5320313 ( 53291.324, 41948.911, 200.000) -> 3,3080568
19,7820197 ( 53285.018, 41964.338, 200.000) -> 3,5640448
Throughput: 0,1625951611808
20,0320403 ( 53278.710, 41979.766, 200.000) -> 3,8040289
Throughput: 0,14794499506272
20,2820256 ( 53272.404, 41995.193, 200.000) -> 4,0600106
20,5320352 ( 53266.097, 42010.621, 200.000) -> 4,3161126
Throughput: 0,136355595556448
20,7820657 ( 53259.790, 42026.050, 200.000) -> 4,5570692
21,0320694 ( 53253.483, 42041.477, 200.000) -> 4,8131099
Throughput: 0,127510596000803
21,2820776 ( 53247.176, 42056.905, 200.000) -> 5,0551424
21,5320681 ( 53240.869, 42072.332, 200.000) -> 5,3121071
Throughput: 0,121158056400723
21,7820919 ( 53234.562, 42087.761, 200.000) -> 5,5532789
22,0320864 ( 53228.255, 42103.188, 200.000) -> 5,8101209
Throughput: 0,118751870760651
22,2821119 ( 53221.948, 42118.617, 200.000) -> 6,0680915
22,5321233 ( 53215.641, 42134.045, 200.000) -> 6,3081329
Throughput: 0,118182343684585
```

Os valores apresentados são o throughput (tempo médio de ida e volta dos pacotes Ping/Pong) e a posição do avião junto com o tempo em relação ao radar e o tempo interno do canhão.

Na imagem abaixo, o grande cubo degrade velho-azul representa o alvo, o retângulo cinza escuro o avião e ao fundo está o canhão.

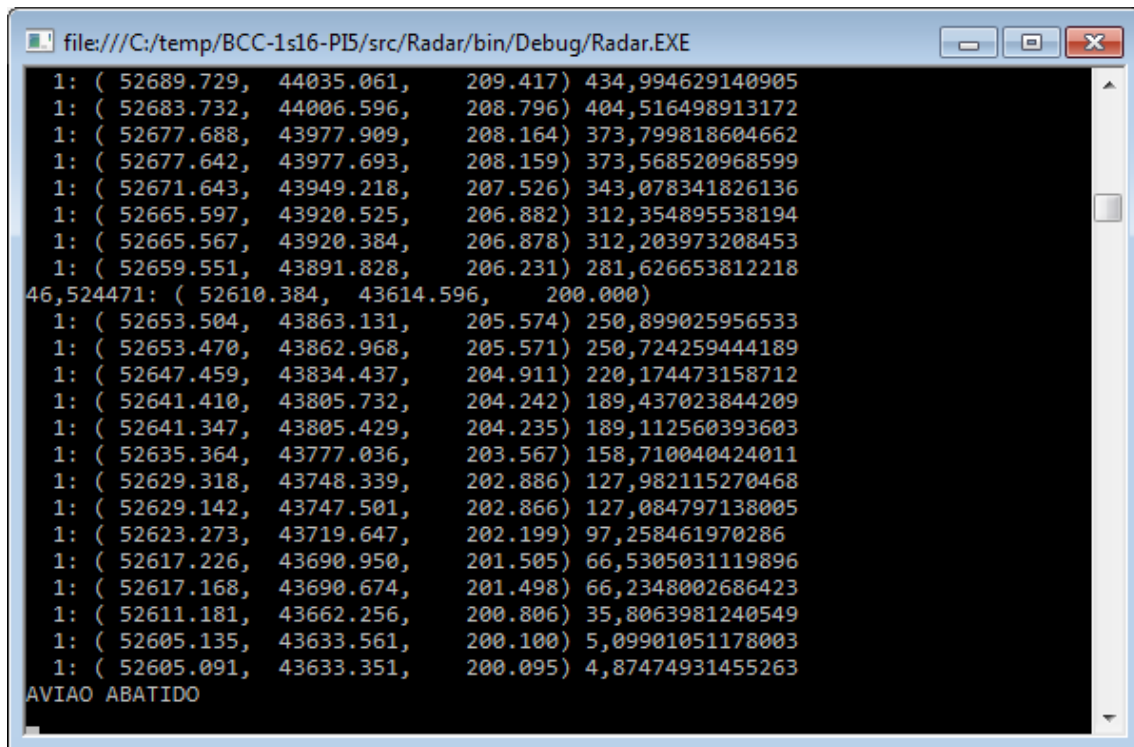


Os objetos foram projetados na tela com o tamanho bem superior ao tamanho no modelo matemático da simulação para facilitar a visualização de tudo o que está ocorrendo. Abaixo uma imagem da simulação no momento de um disparo.



Durante a simulação a câmera acompanha o retângulo que representa o avião. O ângulo de visão pode ser alterado utilizando as teclas direcionais do teclado. A tecla control mais os direcionais para cima e para baixo controlam o zoom da cena.

Junto com a representa gráfica o radar disponibiliza várias informações quantitativas sobre o que está acontecendo em cada instante da simulação em outra tela como demonstra a imagem abaixo:



```
file:///C:/temp/BCC-1s16-PI5/src/Radar/bin/Debug/Radar.EXE
1: ( 52689.729, 44035.061, 209.417) 434,994629140905
1: ( 52683.732, 44006.596, 208.796) 404,516498913172
1: ( 52677.688, 43977.909, 208.164) 373,799818604662
1: ( 52677.642, 43977.693, 208.159) 373,568520968599
1: ( 52671.643, 43949.218, 207.526) 343,078341826136
1: ( 52665.597, 43920.525, 206.882) 312,354895538194
1: ( 52665.567, 43920.384, 206.878) 312,203973208453
1: ( 52659.551, 43891.828, 206.231) 281,626653812218
46,524471: ( 52610.384, 43614.596, 200.000)
1: ( 52653.504, 43863.131, 205.574) 250,899025956533
1: ( 52653.470, 43862.968, 205.571) 250,724259444189
1: ( 52647.459, 43834.437, 204.911) 220,174473158712
1: ( 52641.410, 43805.732, 204.242) 189,437023844209
1: ( 52641.347, 43805.429, 204.235) 189,112560393603
1: ( 52635.364, 43777.036, 203.567) 158,710040424011
1: ( 52629.318, 43748.339, 202.886) 127,982115270468
1: ( 52629.142, 43747.501, 202.866) 127,084797138005
1: ( 52623.273, 43719.647, 202.199) 97,258461970286
1: ( 52617.226, 43690.950, 201.505) 66,5305031119896
1: ( 52617.168, 43690.674, 201.498) 66,2348002686423
1: ( 52611.181, 43662.256, 200.806) 35,8063981240549
1: ( 52605.135, 43633.561, 200.100) 5,09901051178003
1: ( 52605.091, 43633.351, 200.095) 4,87474931455263
AVIAO ABATIDO
```

O ciclo de vida dos aplicativos é de apenas uma simulação. Após o término é necessário reiniciá-los para realização de uma nova simulação.

O código das duas aplicações pode ser obtido no *GitHub* no endereço <https://github.com/celsovlps/BCC-1s16-PI5>.

Discussão

A implementação em C# .Net permitiu uma atenção maior nos detalhes da simulação e não apenas no gerenciamento de memória e conexões que são muito mais complexos em C/C++. Mais opções podem ser trabalhadas, modelos para as equações podem ser testados permitindo um maior acerto e qualidade no trabalho final.