Proposta de Simulador Monte Carlo para o Andar do Bêbado

Mayara Fernandes[1], **Lucas Tonussi**[1]

[1]Ciência da Computação – Universidade Federal de Santa Catarina
Florianópolis – SC – Brasil

{mayara.fernandes,lucas.tonussi}@grad.ufsc.br

Resumo. *Este documento traz a documentação sobre uma proposta de simulador, em C#, para criar casos de simulação do "Andar do Bêbado"(em inglês Random Walk), o software simula na forma de gráfico o caminho dos passos utilizando números aleatórios para os próximos ângulos de cada passo na caminhada aleatória em um plano cartesiano (2D). A principal vantagem desse simulador é a interface gráfica que permite o usuário criar dois gráficos, um para visualizar a caminhada e outro para visualizar o histograma de comparação com a situação ideal de distâncias percorridas.*

1. Introdução

O simulador em questão foi feito com o auxílio do: *Visual Studio* (Ambiente Integrado de Desenvolvimento de Software), da linguagem C#, e de recursos disponíveis pelo framework **.NET 4.5**.

A ideia essencial do uso do método de Monte Carlo é o uso de aleatoriedade para resolver um problema [Wikipédia 2017], no nosso sistema estamos utilizando o gerador de números aleatórios da própria linguagem C#, que é encontrado na classe *Random*, conforme consta no manual da Microsoft para a classe Random: "A implementação da classe Random é baseada em uma versão modificada do algoritmo de Donald E. Knuth, chamado *Subtractive Random Number Generator Algorithm* [Knuth 1997]. Uma descrição completa sobre esse algoritmo pode ser encontrada nesse site.

2. Referencial Teórico

O referencial teórico para esse trabalho está no documento de proposta para esse trabalho, sendo necessário apenas extrair os requisitos de sistema. Além do documento, o nosso referencial teórico foram as aulas de Modelagem e Simulação ministradas pelo professor Dr. Paulo Freitas Filho.

3. Manual de como utiliza o software

A figura 1, logo abaixo, mostra como utilizar o formulário. Basta você digitar uma quantidade de passos e uma quantidade de repetições. Se você não digitar repetições então irá mostrar duas Abas: (1) Caminho, e (2) Distância. Na Aba Caminho mostra um gráfico do caminho aleatório percorrido. Utiliza-se um gerador próprio do C# para gerar, **uniformemente**, números aleatórios de 0 graus à 360 graus e depois esse valor é convertido para radianos e assim pode ser utilizado no algoritmo normalmente.

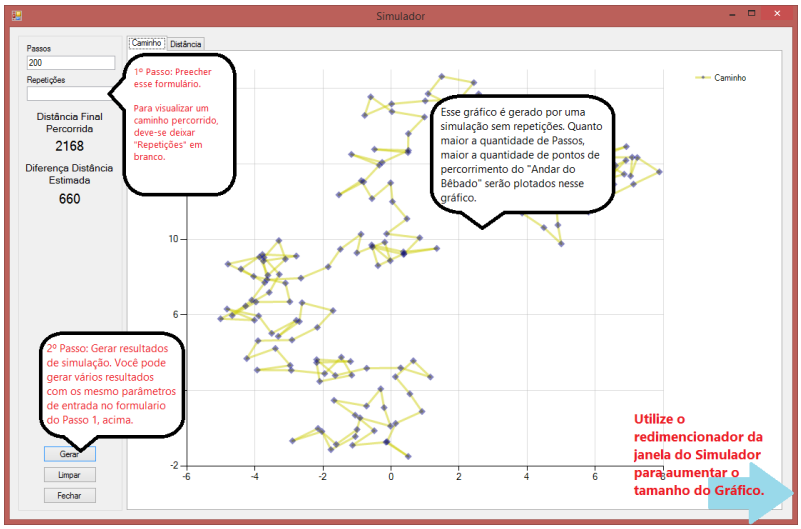


Figura 1. Tutorial Como Utilizar

Na figura 2 é onde o usuário pode avaliar o gráfico de comparação com a distância estimada e a distância percorrida. O eixo horizontal é o eixo da distância percorrida, e a linha vermelha seria a distância ideal estimada.

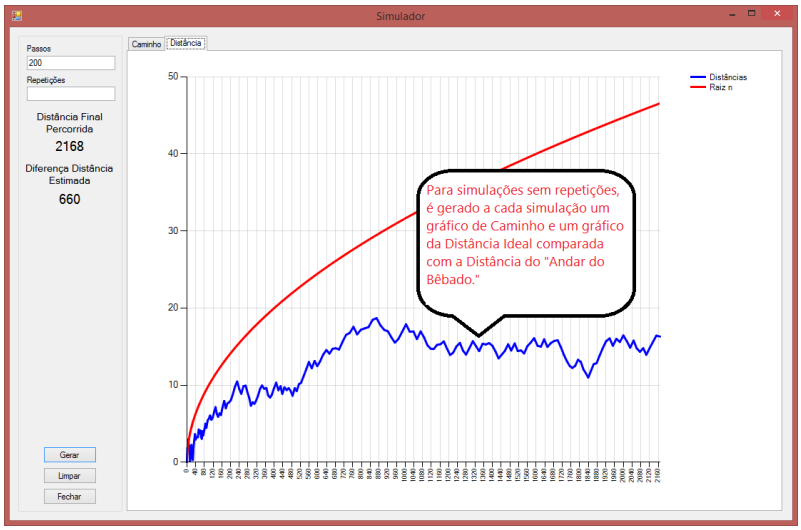


Figura 2. Tutorial Como Utilizar

A figura 3, a seguir, mostra onde o usuário pode averiguar resultados de várias simulações, o eixo vertical (1) mostra as distâncias ideais para o conjunto (classe).

O eixo horizontal mostra os conjuntos de frequências das simulações encontradas para esse classe.

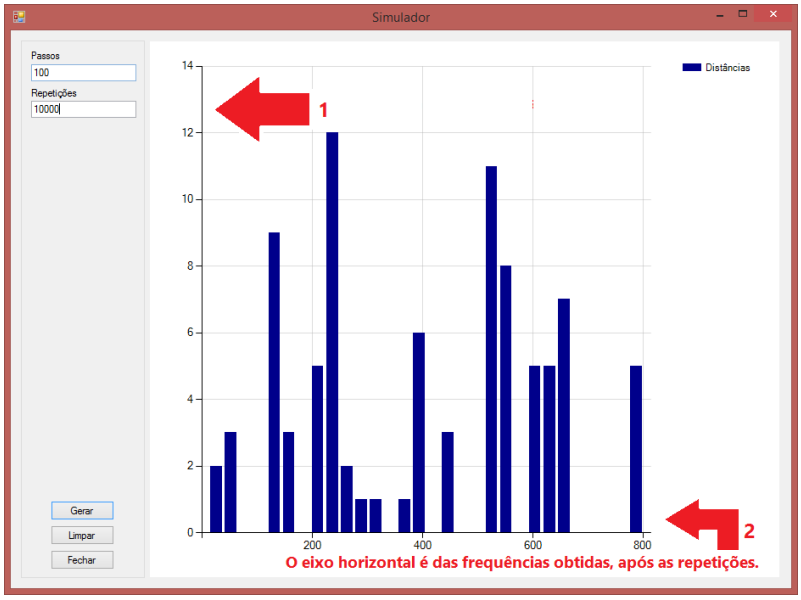


Figura 3. Tutorial Como Utilizar

4. Diagrama do Programa

A figura 4 a seguir representa o diagrama UML para as classes, métodos e propriedades do nosso modelo de Simulador Monte Carlo para o ”Andar do Bêbado”. As notas no diagrama abaixo oferecem uma breve explicação para algumas funções da classe.

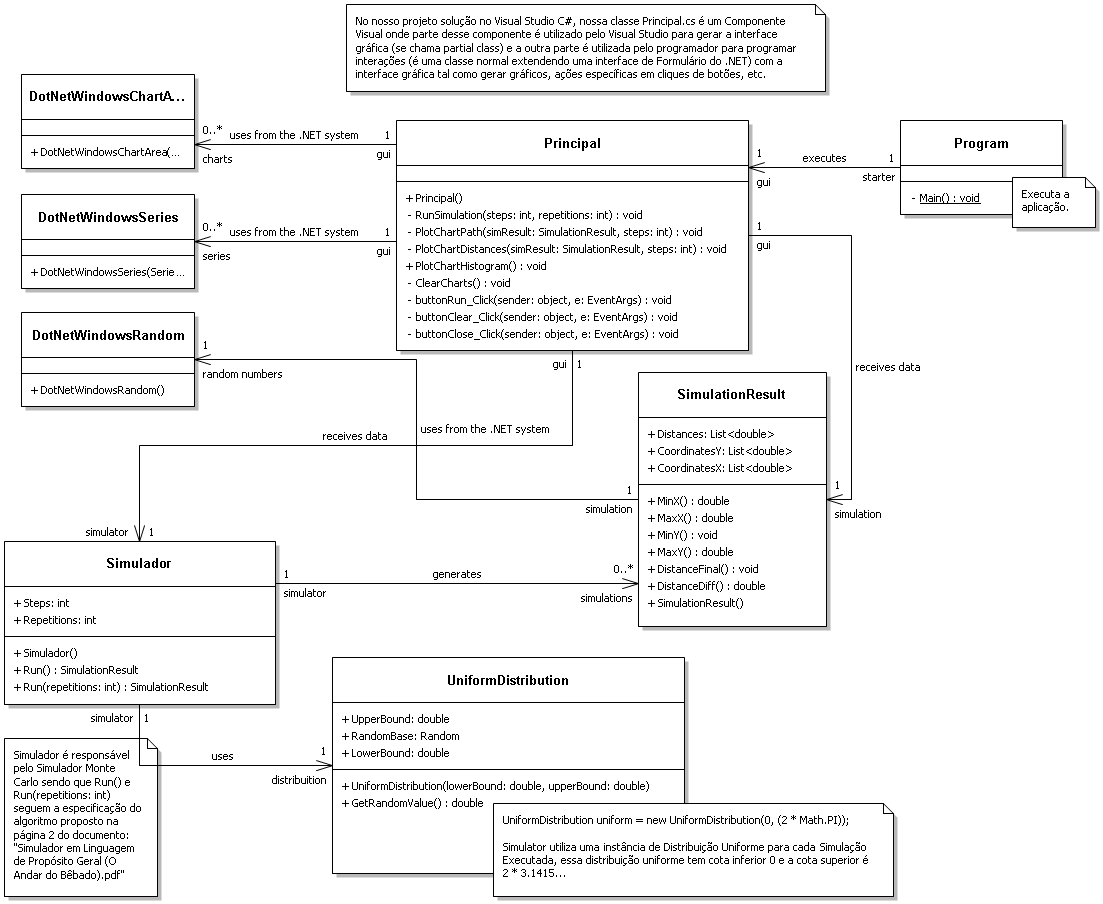


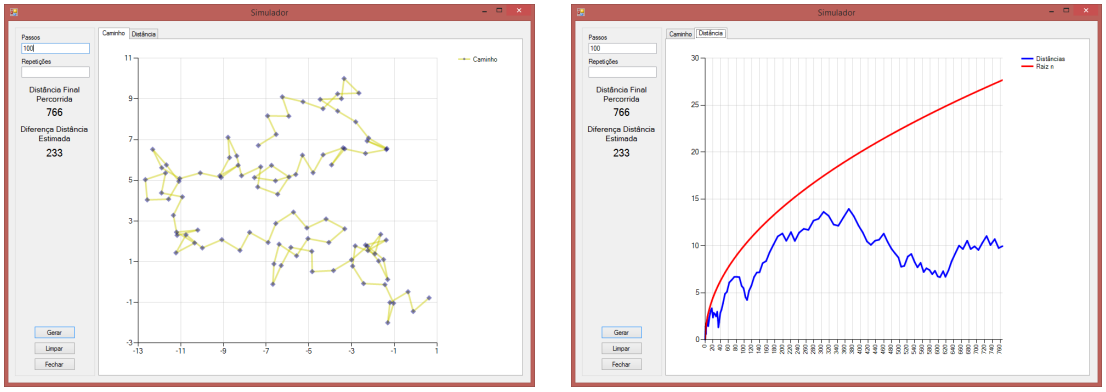
Figura 4. Diagrama UML para a nossa proposta de *Simulador Monte Carlo do ”Andar do Bêbado”*.

5. Metodologia

Foram feitos alguns testes da aplicação para averiguar a capacidade de processamento, e claro, avaliar também a corretude de resposta das simulações geradas. Por corretude queremos dizer se os resultados estão retornando o esperado. Todos os passos partem de um mesmo início. Esse inicio é na coordenada (0, 0). No próximo capítulo nós tentamos abordar o problema teórico das distâncias percorridas.

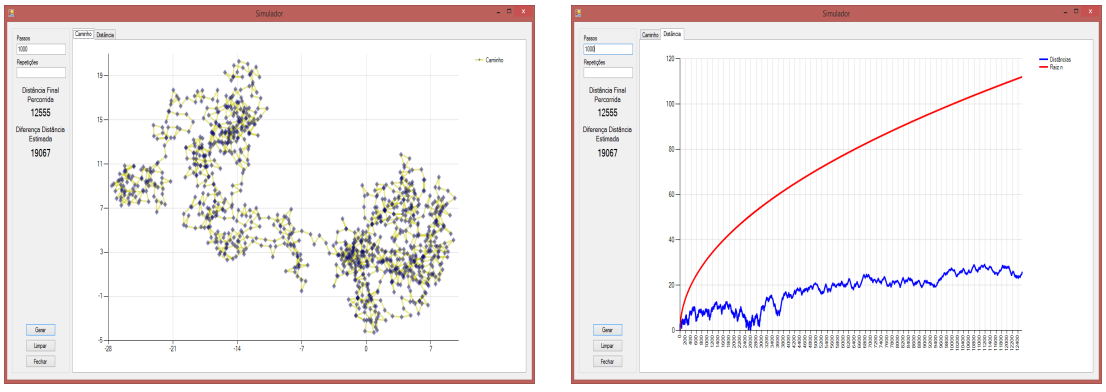
5.1. Teste de quantidade de passos e sem repetições

A figura 5 representa uma simulação com 100 passos e 0 repetições. A figura 6 representa uma simulação com 1000 passos e 0 repetições. Pelo que se observa no nosso simulador Monte Carlo do andar do bêbado é que a quantidade de passos tem influência no quanto se aproxima da distância ideal estimada, quanto maior os passos maior pode ser o erro em relação a distância estimada para a quantidade de passos k fornecida como entrada no sistema.



(a) Aba Caminho (b) Aba Distância

Figura 5. Teste com 100 Passos e 0 Repetições.

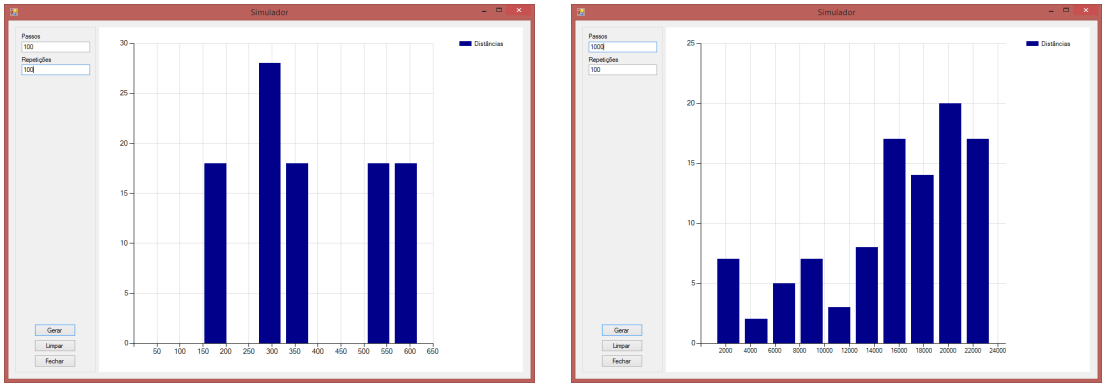


(a) Aba Caminho (b) Aba Distância

Figura 6. Teste com 1000 Passos e 0 Repetições.

5.2. Teste de quantidade de passos e repetições

O teste da figura 7 é uma comparação entre duas simulações: (1) 100 passos e 100 repetições, (2) 1000 passos e 100 repetições, onde foi visto que ambos retornam resultados aproximados.

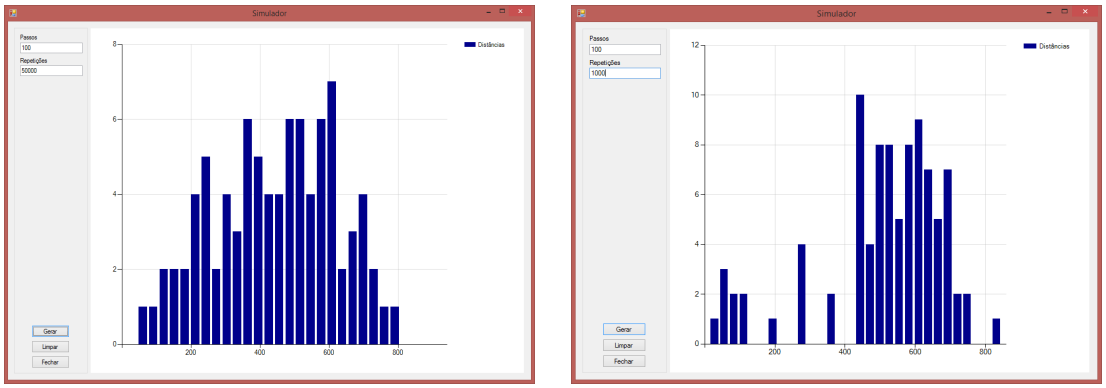


(a) 100 p 100 r (b) 1000 p 100 r

Figura 7. Comparação de 100 passos 100 repetições contra 1000 passos e 100 repetições.

5.3. Teste de desempenho e quantidade de repetições

Um teste com 1000 passos e 20000 repetições foi realizado em um Laptop Intel Core i5 4 Cores (2 Físicos, 2 Virtuais), 6 Gigabits the memória RAM DDR3 e não foi possível terminar a simulação devido a falta de memória RAM disponível. Então podemos prever que muitas repetições com uma grande quantidade de passos pode resultar em problemas por falta de memória RAM. Todavia, testes com poucos passos e muitas repetições, figura 8, irá chegar ao fim da simulação sem estourar a memória RAM disponível. Também observa-se que esse tipo de teste tende a retornar resultados aproximados para quantidade de passos fixo. a quantidade de repetições impacta diretamente no tempo de execução, quanto mais repetições maior o tempo até se alcançar o teste com todas para cada caso



(a) 100 p 50000 r (b) 100 p 1000 r

Figura 8. Comparação de 100 passos 50000 repetições contra 100 passos e 1000 repetições.

Para salientar o uso do algoritmo do fornecido no documento da proposta desse trabalho (”Simulador em Linguagem de Propósito Geral (Andar do Bêbado)”) abaixo está o algoritmo usado para gerar Simulações.

1. Começo: $x_i = y_i = 0$.
2. $l = i + 1$.
3. Gerar um número entre 0º 360º (Uniformemente Distribuído): α_i.
4. $x_{i+1} = x_{i+l} + l \times \cos(\alpha_{i+1})$.
5. $y_{i+1} = y_{i+1} + l \times \sin(\alpha_{i+1})$.
6. $d_{i+1} = \sqrt{x_{i+1}^2 + y_{i+1}^2}$.
7. Repetir Passos 2 até 6 para $i = n$.

A classe *Simulator* (ver figura 4), traz dois métodos (1): *Run() : SimulationResult* e (2): *Run(Número de Repetições) : SimulationResult*. O primeiro é executado para processar o algoritmo proposto acima. Já o segundo inclui código auxiliar para salvar os resultados de cada simulação para um número de repetições fornecido pelo usuário.

Referências

[Knuth 1997] Knuth, D. E. (1997). *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley.

[Wikipédia 2017] Wikipédia (2017). Método de monte carlo. Disponível [online]. Acessado em Julho, 2017.