UNIVERSIDADE FEDERAL DE ALAGOAS

Instituto Federal de Alagoas - IC

Engenharia de Computação

Sistemas de Controle 2

Implementação de um controlador em espaço de estados para o jogo “Can U Control?”

Élisson França de Souza

José Renilson Almeida da Silva

Monique Moreira Moraes

Tiago Cabral Carvalho da Silva

Maceió - 2023

ÍNDICE

1. Introdução

O objetivo deste documento é relatar o desempenho obtido no jogo Can U Control nos quatro mapas disponíveis no jogo. Foram implementadas as funções de espaço de estados para a movimentação do personagem no mapa, que é um campo de vetores que direciona o personagem. O conjunto destas duas funções, por sua vez, altera o campo de vetores a fim de que o personagem alcance o alvo, fazendo com que o personagem se movimente tendendo a uma outra trajetória.

O código desenvolvido para este trabalho foi elaborado em linguagem Python, utilizando conceitos matemáticos como o método de Euler para calcular possíveis posições do personagem e jogar na saída com base nesse método, junto com uma busca exaustiva do termo de saída para o controle do personagem.

2. O jogo

“Can U Control” é um jogo com propósito educativo que visa facilitar a compreensão dos conceitos de Espaço de Estados Dinâmico. É baseado em sistemas dinâmicos e a representação do sistema é a de espaço de estados, que nada mais é do que uma equação diferencial de primeira ordem onde as variáveis de estado estão associadas a uma representação vetorial única. O valor de cada um desses vetores em um instante no tempo representa a direção que o estado irá seguir e assim para cada ponto no espaço terá uma direção diferente. O gráfico cartesiano com todos esses vetores é chamado de Campo Vetorial, que é justamente o mapa do jogo. Os elementos presentes no mapa são o personagem, o alvo, caixas de pontuação e marcas de morte. Se o personagem chegar no alvo, a fase é concluída. A movimentação do jogo é feita pela alteração do campo vetorial e não do controle do personagem.

O jogo transfere dados relacionados a vitórias, mapa, nível, posições X e Y do jogador, posições X e Y do alvo, caixas bônus e possíveis caveiras que indicam a zona que o personagem não pode passar. Com esses parâmetros X e Y do personagem, temos que controlar e mover o personagem até um objetivo, podendo ser as caixas bônus ou o objetivo do personagem, controlando apenas um parâmetro de saída com o desafio de ter duas entradas diferentes e somente uma saída para controlar ambos sinais.

3. Método de Euler

O método de Euler é um método de integração baseado em uma aproximação linear de uma função desconhecida y(x) que tem como finalidade resolver uma equação diferencial ordinária (EDO) de primeira ordem. Ele segue a ideia de que, em um intervalo pequeno o suficiente, a taxa de variação de y é aproximadamente constante. Usado para descrever ou simular processos e sistemas modelados por taxa de variação, que podem possuir até dezenas ou centenas de variáveis. Esse método também é conhecido como a série de Taylor truncada na primeira derivada. É um método de fácil implementação e compreensão, mas não é tão preciso quanto outros métodos de integração, como o runge-kutta de 4º ordem, por exemplo.

Com o método de Euler, podemos obter relações de equações diferenciais, onde podemos buscar uma certa trajetória ou tendência da trajetória para uma determinada função em um espaço definido de passo (indicado por k).

Para explicar o método de integração de Euler, vamos considerar a seguinte EDO de primeira ordem: $\frac{dy}{dx} = f(x,\ y)$.

O objetivo é encontrar uma solução aproximada para y em um intervalo de x, dado um valor inicial y_0 em x_0. A ideia básica do método de Euler é aproximar a derivada dy/dx por meio de uma diferença finita e usar essa aproximação para calcular os valores subsequentes de y.

Primeiramente para a utilização deste método é necessário definir o tamanho do passo k, que quanto menor for, melhor será a precisão da integração, mas em contrapartida resultará em mais cálculos. Seguidamente define-se o valor inicial y_0 em x_0, e a partir de x_0 calcula-se o valor do próximo ponto y para cada x_i, segundo a fórmula:

$$y_{i+1} = y_i + k * f(x_i, y_i)$$

$$x_{i+1} = x_i + k$$

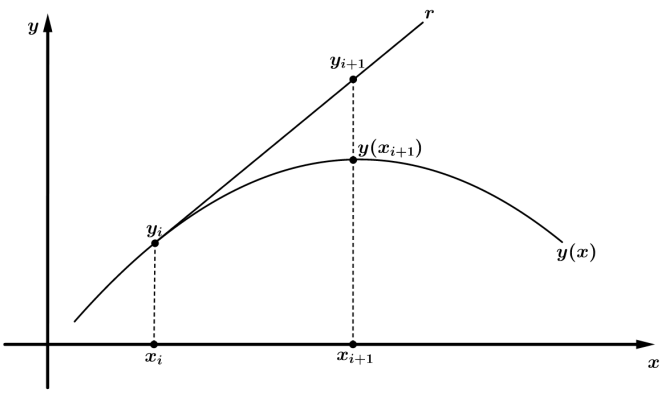


Figura 1: exemplo de função real e aproximação por método de Euler.

onde x_i é valor atual do x, y_i é o valor atual de y e $f(x_i, y_i)$ é o valor da função $f(x, y)$ no ponto (x_i, y_i).

Segue abaixo um exemplo do método de Euler para aproximação de uma função aparentemente senoidal, onde a curva em azul representa a função de fato e a em vermelho representa a aproximação de Euler para essa mesma função.

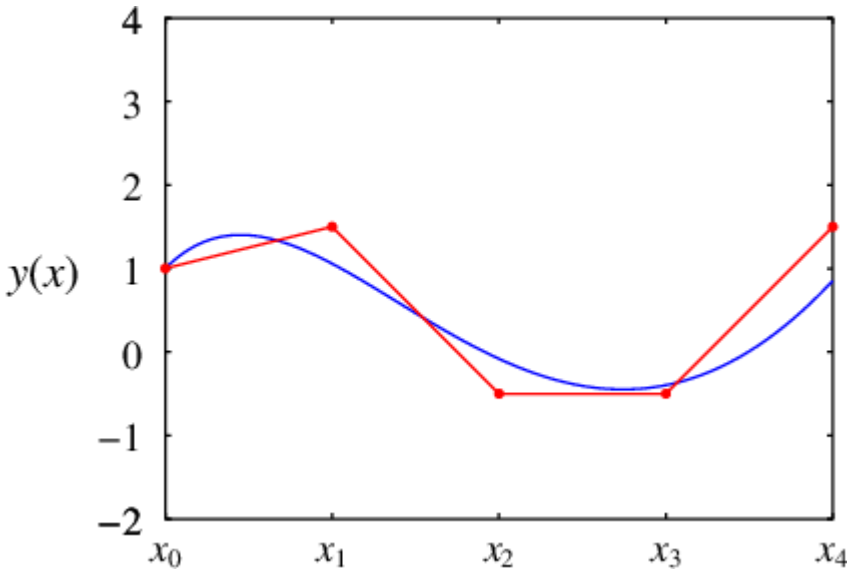


Figura 2.

É importante mencionar que o método de Euler possui algumas limitações. Ele pode introduzir erros acumulativos à medida que avança no intervalo, especialmente se o tamanho do passo k for grande. Além disso, o método pode não ser adequado para EDOs com soluções que envolvam curvas muito complexas ou com regiões de rápido crescimento ou mudanças abruptas, porém é de grande importância para a introdução e aprendizado do assunto.

4. Código Implementado

O código implementado foi feito usando os conceitos apresentados na descrição do jogo, onde se é usado o conceito do método de Euler, junto com técnicas de distância euclidiana (do alvo para com a posição relativa do personagem, atual e futuras). Foi usado também uma técnica de setar nível de saída, pois há um limite máximo e mínimo (+1 e -1, respectivamente) da saída do código. No jogo, isso remete a uma linha preta criada no local do personagem, indicando em locais prováveis onde o mesmo tenderá a se mover:

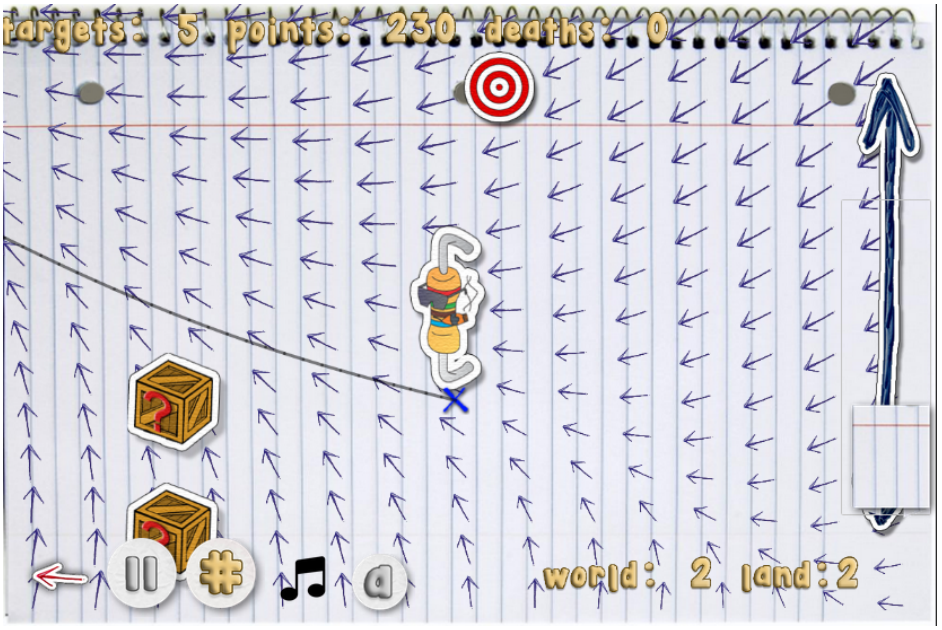


Figura 3: imagem do jogo com tendência de trajetória.

No código, temos o seguinte fluxograma de eventos:

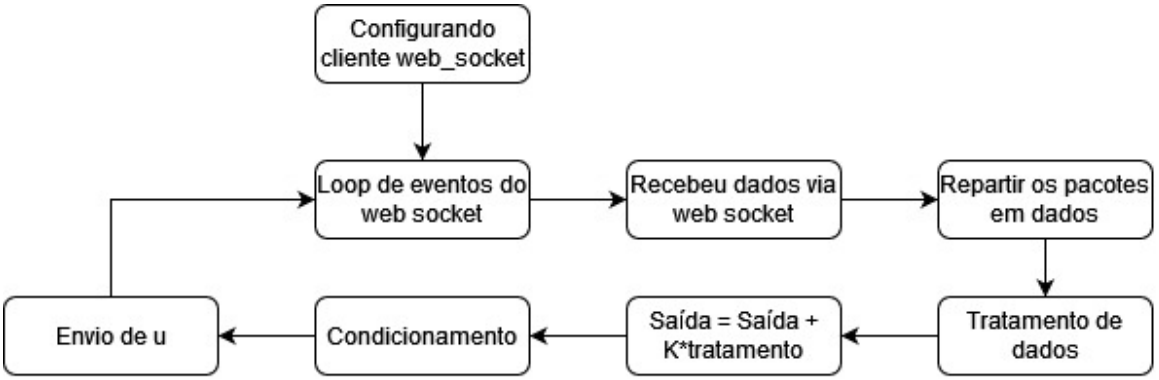


Figura 4: fluxograma do algoritmo desenvolvido.

Ao receber um pacote de dados, o código é iniciado e então reparte os dados do pacote em fragmentos que possam ser tratados e analisados, pois eles chegam como strings.


```python
def repartir_mensagens(pacote):
    vitorias = int(pacote[0])
    mapa = int(pacote[1])
    nivel = int(pacote[2])
    jogador = pacote[3], pacote[4]
    alvo = pacote[5], pacote[6]
    caixas, caveiras = [], []
    if(len(pacote) > 8):
        caixas.append((pacote[7], pacote[8]))
    if(len(pacote) > 10):
        caixas.append((pacote[9], pacote[10]))
    if(len(pacote) > 12):
        caixas.append((pacote[11], pacote[12]))
        for i in range(13, len(pacote), 2):
            caveiras.append((pacote[i], pacote[i+1]))
    return vitorias, mapa, nivel, jogador, alvo, caixas, caveiras
```

Figura 5: repartimento e divisão de dados.

Considerando o método de Euler, temos que tratar e checar os próximos passos do personagem, onde no algoritmo, estamos fazendo com 50 passos simulados. Cada mapa tem um limite de U a ser considerado para reduzir a chance do personagem a sair do campo de visão do indivíduo, porém permitindo que ele saia rapidamente para retornar para um provável caminho mais curto, como mostra o fluxograma da figura 6. O código deve parametrizar e organizar os valores a serem enviados para a computação dos valores e seus possíveis tratamentos, como na figura 7.

```python
    if level_anterior != nivel:
        print("Quantidade de fases passadas:", passou)
        passou += 1
        level_anterior = nivel
        saida = 0

    k = mapas.define_equacoes(mapa, nivel, 0, 0, 0, False)
    CaveirasX = []
    CaveirasY = []
    ObjetivosX = []
    ObjetivosY = []
    ObjetivosX.append(alvo[0])
    ObjetivosY.append(alvo[1])
    quant = 50
    dt = 0.01
    if(len(caixas) >= 1 and nivel != 3):
        ObjetivosX.append(caixas[0][0])
        ObjetivosY.append(caixas[0][1])
        if(len(caixas) >= 2):
            ObjetivosX.append(caixas[1][0])
            ObjetivosY.append(caixas[1][1])
            if(len(caixas) >= 3):
                ObjetivosX.append(caixas[2][0])
                ObjetivosY.append(caixas[2][1])
    if(len(caveiras) >= 1):
        CaveirasX.append(caveiras[0][0])
        CaveirasY.append(caveiras[0][1])
        if(len(caveiras) >= 2):
            CaveirasX.append(caveiras[1][0])
            CaveirasY.append(caveiras[1][1])
            if(len(caveiras) >= 3):
                CaveirasX.append(caveiras[2][0])
                CaveirasY.append(caveiras[2][1])
    saida = saida + (k)*computacaoInicial(jogador[0], jogador[1], mapa, nivel, quant, dt, ObjetivosX, ObjetivosY, CaveirasX, CaveirasY)
    if abs(saida) >= 1:
        saida = (saida/abs(saida))

    servidor.send_message(cliente, str(saida))
```

Figura 6: tratamento inicial e final de dados.

No momento de salvar a menor distância, é analisada o conjunto de prováveis objetivos, tais como alvo e caixas. Quando estamos checando a distância do personagem para com o alvo, temos que o peso (f) é mínimo e sendo igual a 1, e quando não, o seu peso é 2, priorizando assim o alvo como o melhor caminho a ser considerado. Essa distância é calculada com base na distância euclidiana de um sistema em duas dimensões, porém resumida para evitar gasto de tempo desnecessário para com os cálculos, como descrita abaixo:

$$distância = f * ((x - objx)^2 + (y - objy)^2)$$

Checando as melhores distâncias para o objetivo mais próximo, temos que considerar então o cálculo das funções de espaço de estados em conjunto com as variáveis adotadas, tais como a posição do personagem, sendo eles a origem, e por uma busca exaustiva de U, buscando sempre o caminho com menor distância. O valor considerado da melhor distância é sempre o valor do loop para a primeira equação por deixar sempre mais próximo do objetivo e mais preciso o movimento.

A figura 8 mostra bem como usar o método de Euler modificado para encaixar aos requisitos do projeto de controlar somente uma saída.

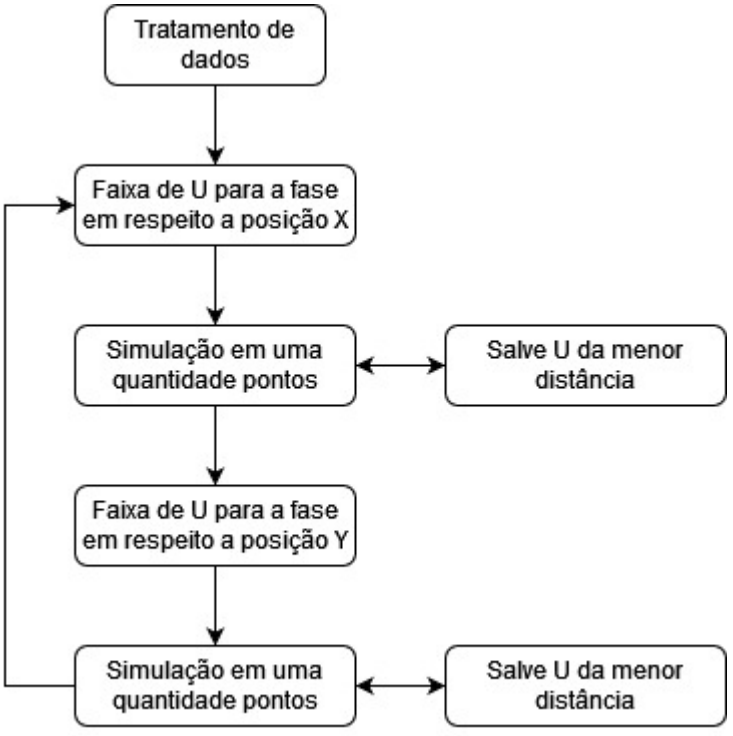


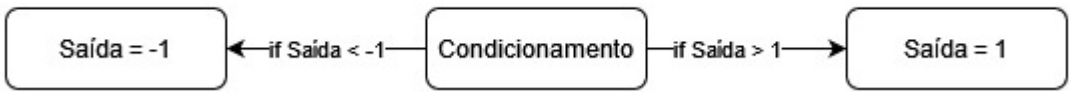
Figura 7: fluxograma de computação das coordenadas.

```python
def computacaoInicial(x, y, mapa, nivel, quant, dt, ObjetivosX, ObjetivosY, CaveirasX, CaveirasY):
    melhorDist = np.inf
    melhorI = 0
    uMax = maximo(mapa, nivel)
    for i in range(-uMax, uMax + 1):
        valorUI = i / uMax
        distMinima = np.inf
        distancia = np.inf
        x1 = x
        y1 = y
        u = valorUI
        for k in range(quant):
            for q in range(len(ObjetivosX)):
                f = 1 if q == 0 else 2
                distancia = f*((x1 - ObjetivosX[q])*(x1 - ObjetivosX[q]) + (y1 - ObjetivosY[q])*(y1 - ObjetivosY[q]))
                if (distancia < distMinima):
                    distMinima = distancia
            l12 = mapas.define_equacoes(mapa, nivel, x1, y1, u)
            x1 += l12[0]*dt
            y1 += l12[1]*dt
            if(distMinima < melhorDist):
                melhorDist = distMinima
                melhorI = valorUI
        for j in range(-uMax, uMax + 1):
            valorUJ = j / uMax
            x11 = x1
            y11 = y1
            u = valorUJ
            for k in range(quant):
                for q in range(len(ObjetivosX)):
                    f = 1 if q == 0 else 2
                    distancia = f*(x11 - ObjetivosX[q])*(x11 - ObjetivosX[q]) + (y11 - ObjetivosY[q])*(y11 - ObjetivosY[q])
                    if (distancia < distMinima):
                        distMinima = distancia
                l12 = mapas.define_equacoes(mapa, nivel, x11, y11, u)
                x11 += l12[0]*dt
                y11 += l12[1]*dt

                if(distMinima < melhorDist):
                    melhorDist = distMinima
                    melhorI = valorUI
    return melhorI
```

Figura 8: código implementado para computação.

O código, por fim, condiciona a saída para que não escreva valores indevidos na saída, limitando a saída entre o intervalo de -1 a 1.




```python
if abs(saida) >= 1:
    saida = (saida/abs(saida))
```

Figura 9: condicionamento de u.

Com isso, podemos enviar o valor de u, que controla o personagem, levando para mais próximo do objetivo que ele teve como mais próximo.

5. Conclusão

Concluímos que o algoritmo de Euler junto com outras técnicas é eficiente para controlar a trajetória do personagem no jogo “Can U Control?”, mesmo se perdendo em alguns casos, possibilitando que o algoritmo seja rápido e de certa forma confiável.

A série de Taylor ou o usando o método de Runge-Kutta, funcionaram também, expandindo as equações, usando até ordem 4, porém sua habilidade não foram bem desenvolvidas por deixar o sistema lento e gerar muito mais erros acumulativos.

Por fim, notamos que técnicas simples como Euler, busca exaustiva e passos curtos tornaram o algoritmo mais eficiente e funcional na maioria das vezes testadas.

6. Referências

Método de Euler, Wikipédia. Disponível: <https://en.wikipedia.org/wiki/Euler_method>. Acesso em 10/05/23;

Método de Euler, UFJF. Disponível em: <https://www.fisica.ufjf.br/~sjfsato/fiscomp1/node45.html>. Acesso em 10/05/23;

Distância Euclidiana, Wikipédia. Disponível em: <https://pt.wikipedia.org/wiki/Dist%C3%A2ncia_euclidiana>. Acesso em 10/05/23;

Método de Euler Explícito para Resolver EDOs, Artigo de Victoria Tanaka. Disponível em: <https://edisciplinas.usp.br/pluginfile.php/4530296/mod_resource/content/1/modelo3.pdf>. Acesso em 11/05/23;

Método de Euler, Disponível em: <https://www.ufrgs.br/reamat/CalculoNumerico/livro-py/pdvi-metodo_de_euler.html>. Acesso em 11/05/23;