COMMUNICATIONS AND INNOVATIONS GAZETTE

Integração de modelos matemáticos e simulação com desenvolvimento Web moderno

Celso Gabriel D. A. Malosto[1], **Bárbara de Melo Quintela**[1], **Igor de Oliveira Knop**[1]

[1]Grupo de Ensino Tutorial em Sistemas de Informação,
Universidade Federal de Juiz de Fora (UFJF)
Juiz de Fora – MG – Brasil

celsogabriel@ice.ufjf.br, barbara@ice.ufjf.br,
igorknop@ice.ufjf.br

Abstract. *Computational modeling is a computer-aided approach that uses mathematical models to perform in silico experiments in a safe and relatively fast manner, to reproduce distinct behaviors of several natural and social phenomena. The present work describes the construction of a real-time web-based simulator aiming the educational use to the study of the dynamics of organisms such as Physarum polycephalum, using mathematical models to represent reaction, diffusion, and chemotaxis. The software was developed using modern tools to allow real-time interaction with the mathematical model. Partial results are shown as well as the limitations, current state of implementation and the next steps.*

Resumo. *Modelagem computacional é uma abordagem auxiliada por computador, que usa modelos matemáticos para, de forma segura e relativamente rápida, realizar experimentos in silico para reproduzir comportamentos de diversos fenômenos naturais e sociais. O presente trabalho descreve a construção de um simulador em tempo real para a web para uso educacional a fim de estudar dinâmicas de organismos, como o Physarum polycephalum, utilizando modelos matemáticos para representar reação, difusão e quimiotaxia. O software foi desenvolvido com ferramentas modernas para permitir interação com os modelos matemáticos em tempo real. Os resultados parciais são apresentados junto das limitações da implementação atual e próximos passos.*

1. Introdução

As simulações computacionais que utilizam modelos matemáticos para representar fenômenos físicos já são largamente utilizadas em diversas áreas das ciências e engenharias (QUARTERONI, 2009). Na área da saúde, diversos modelos matemáticos já foram utilizados para representar as dinâmicas de células e moléculas dos sistemas biológicos para auxiliar a compreensão dos mecanismos de infecção e identificar alvos terapêuticos (PERELSON, 2002; QUINTELA et al., 2018).

Muitos desses modelos são desenvolvidos como aplicações para serem executadas por linha de comando ou *scripts* dificultando o seu uso por usuários que não sejam programadores. Além disso, os modelos matemáticos permitem que um número grande de cenários e variações de parâmetros sejam simulados, o que requer alteração desses valores

de entrada a cada execução. Dessa forma, a existência de uma interface web que permita a alteração dos parâmetros durante a execução facilita a interação e realização de diversos tipos de experimentos de forma simplificada do ponto de vista do usuário.

O cenário de desenvolvimento Web recebe contribuições constantes a fim de melhorar a segurança, a eficiência e a acessibilidade das aplicações. Entre elas, estão as bibliotecas e *frameworks* de interface de usuário, que facilitam o processo de construção das páginas, como o *React* (ROVEDA, 2020). Essa biblioteca visa à criação de sites por meio da declaração de componentes reutilizáveis, que se atualizam na tela conforme há mudança nos dados a eles vinculados, o que garante segurança e eficiência.

Este projeto tem como principal objetivo desenvolver uma aplicação para a Web, com propósitos de visualização e educação e direcionada ao usuário comum, na qual esse pode interagir com a movimentação de um protista como numa placa de Petri, alterando parâmetros e depositando substâncias no ambiente simulado, para assim compreender a influência dos valores iniciais no resultado e o processo de reação dos compostos. Dado esse propósito, é necessário garantir rapidez na execução do aplicativo, fácil interação com a interface, e flexibilidade para utilizar o modelo em outros contextos de simulação.

O projeto será construído a partir do uso da biblioteca *React* para a elaboração da página e a conexão entre seus componentes, e pela implementação de equações de decaimento, difusão e reação entre compostos, que atualizam os valores das substâncias simuladas, e cujas representação e manipulação são feitas por uma escala de cores crescente entre verde e vermelho num elemento *Canvas* da lingugagem HTML.

Na perspectiva de educação tutorial, o projeto visa a desenvolver habilidades do discente nas tecnologias utilizadas, aliando o aprendizado a um projeto prático. Outrossim, são desenvolvidas habilidades interpessoais, ou *soft skills*, tais como responsabilidade com prazos, *networking*, e contato com diferentes áreas do saber.

O restante do trabalho apresenta a fundamentação teórica (Seção 2), os requisitos da aplicação (Seção 3) e os resultados parciais obtidos com o simulador que foi desenvolvido (Seção 4). As considerações finais sao apresentadas na Seção 5.

2. Fundamentação Teórica

Dada a complexidade que os modelos matemáticos podem alcançar, decidiu-se inicialmente representar o movimento do protista *Physarum polycephalum*, conhecido como *Blob* amarelo, e métodos de reação e difusão sobre compostos (SIMS, n.d.b). O *Physarum polycephalum* é um organismo unicelular tipo ameba que cresce formando uma rede eficiente de transporte de acordo com a disponibilidade de nutrientes que existem no ambiente. Devido a suas características, vem sendo estudado para planejamento de rotas (KAY et al., 2022; ADAMATZKY; ALONSO-SANZ, 2011).

Esse comportamento gera padrões de distribuição característicos, os quais podem ser simulados a partir de modelos matemáticos. Dada a sua magnitude macroscópica e o bom entendimento que se tem de seus mecanismos de movimento, decidiu-se por utilizá-lo como primeiro contexto de simulação deste projeto. O presente trabalho se baseia, portanto, no desenvolvimento de um simulador como o de reação-difusão desenvolvido por Karl Sims (SIMS, n.d.a). Em seu simulador, são apresentadas diversas possibilidades de formação de padrões a partir da simulação de interação entre duas substancias com

possibilidade de escolha de parâmetros para indicar como se espalham e interagem. A interface é simples e intuitiva porém o usuário não tem acesso a explicação dos conceitos diretamente a partir do simulador.

Cada vez mais os simuladores vêm sendo utilizados com objetivos educacionais. O Epidemix (MUELLNER et al., 2018) é um exemplo de aplicativo web que permite realização de simulações para uso educacional com objetivo de facilitar o uso dos modelos matemáticos sem a necessidade de aprender toda a complexidade por trás da matemática envolvida no desenvolvimento dos modelos . Essa ferramenta apresenta modelos de epidemiologia uteis para compreender como os vírus se espalham, permitindo simular diversos cenários variando os parâmetros de forma simples através da interface web. A ferramenta apresenta também os conceitos por trás da implementação.

A visão que orienta o desenvolvimento de programas para a Web atualmente busca garantir segurança na manipulação de dados e execução de *scripts*, e a acessibilidade para pessoas com diferentes necessidades e para aparelhos com características distintas. Nesse sentido, são utilizadas as tecnologias *HyperText Markup Language (HTML)* para construir a estrutura da página, *Cascading Style Sheets* (CSS) para definir o estilo de seus componentes, e *JavaScript* para manipular dados e elementos (MDN, 2022b).

A fim de facilitar o desenvolvimento e garantir esses princípios, a *Meta* criou, no ano de 2011, a biblioteca para manipulação de interface do usuário *React* (ROVEDA, 2020). Ela é projetada para criar *sites* de uma única página, que é dividida em componentes reutilizáveis. Estes são escritos em JSX, uma linguagem de programação declarativa, em que os elementos HTML que se deseja visualizar são escritos junto ao JavaScript. Assim, utilizando objetos especiais *state* e *props*, os componentes são redesenhados a cada atualização dos valores, mantendo o acesso aos dados seguro, e a exibição eficiente.

3. Requisitos da Aplicação

Parte-se do princípio que a aplicação deve ser capaz de calcular as reações em tempo real, e apresentar as concentrações de substâncias na forma de escala de cores em uma matriz bidimensional. Além disso, o código deve ser implementado de tal modo que seja escalável para permitir numerosos compostos ou populações de organismos. Dessa forma, pode ser adaptado para diversas simulações que se baseiem em reação, difusão e quimiotaxia, como os modelos infectológicos inicialmente almejados.

A interface deve ser responsiva, de modo a possibilitar a execução em aparelhos *mobile*. O usuário deve ter um papel de controle dos parâmetros da simulação. Além disso, ele pode inserir unidades de concentração dos compostos diretamente no plano de exibição via clique e arrastar do mouse ou toque na tela. Ao realizar esses movimentos, é adicionado àquela célula a quantidade de uma unidade de concentração do composto ou organismo representado na camada exibida. Dessa forma, é possível desenhar diferentes padrões de distribuição e avaliar os resultados.

3.1. Projeto da interface web

O projeto foi desenvolvido utilizando a biblioteca *React*, que é baseada na linguagem de programação *JavaScript*, aliada ao pacote de ferramentas *Create React App* (FACEBOOK, 2022), que habilitou o ambiente de desenvolvimento para a aplicação. A interface é construída a partir de componentes pequenos e autocontidos, que podem ser rede-

senhados a depender das informações do aplicativo e, assim, reutilizados em diferentes contextos.

O desenvolvimento dos componentes foi realizado de forma incremental, do nível mais alto para o mais baixo. Assim, a raiz do aplicativo, nomeada *App*, define as funções gerenciadoras de eventos, para interação do usuário com a simulação, e serve como conexão com as funções que calculam os valores das reações. Além disso, a raiz incorpora o componente *Tela de Animação*, que constrói a área de exibição por meio do elemento HTML *canvas*.

A fim de animar o *canvas*, definiu-se no *App* uma função *draw*, responsável por construir a próxima imagem a desenhar na tela. Essa função é passada como parâmetro para o componente *Tela de Animação*. Neste componente, é definida uma função *render*, que executa a função *draw*, e chama o método *window.requestAnimationFrame()*, do *JavaScript*, o qual é responsável por agendar o desenho na próxima oportunidade que o navegador fornecer. Como dentro da função *render* há uma nova requisição de desenho de quadro, isso configura um laço de animação a um número de quadros por segundo definido pelos limites definidos pelo navegador, mas muito próximo de 60 quadros por segundo em computadores modernos.

Na versão inicial do projeto, o simulador ainda apresentava os parâmetros de simulação fixos e suportava uma única camada de composto, além de não permitir a interrupção do processo. Assim, foi construído um componente para configurações, constituído por um elemento de formulário com dois campos numéricos, em que o usuário poderia definir o fator de decaimento e de difusão para o composto, e um botão de envio dos dados. Um gerenciador de evento de envio capturava os valores digitados e os atualizava nas variáveis de cálculo. Ademais, foi implementado um botão de controle de interrupção e continuidade da simulação.

Em seguida, houve a necessidade de aprimorar os campos de parâmetros, haja vista o suporte dado a um segundo composto, o qual reage com o primeiro. Assim, ao componente de configurações, foram adicionados os campos de difusão, adição ou decaimento, e padrão de distribuição inicial de cada composto. Outrossim, foram criados os campos de configuração de dimensões do plano da simulação, de forma que o número de linhas e colunas e o tamanho de cada célula puderam ser definido pelo usuário.

Dada a existência de mais um composto e o requisito de escalabilidade da aplicação, decidiu-se separar as funções de cálculo da simulação em outro arquivo, passo em que a classe *Simulador* foi construída. A responsabilidade desta é limitada a definir os valores dos compostos e construir a nova imagem a ser exibida (função *draw*).

Para melhor visualização das camadas, foi criado um componente *Informação de Concentração*, que exibe a soma de concentração de cada composto, e que se incorpora diretamente ao componente *App*. Além disso, foi construído o componente Painel, que incorpora as configurações, e também inclui um elemento de seleção de camada, que troca o composto exibido na tela.

Nesta etapa, deu-se início ao uso de *states*, um *hook* do *React* que guarda um dado variável que impacta na construção da aplicação, para controlar a interrupção da simulação. Até então, ao pausar, o elemento *canvas* era redesenhado continuamente, apenas os valores dos cálculos não mudavam. Isso custava processamento desnecessário

da máquina do usuário.

Primeiramente, a função *draw* foi encapsulada por uma nova função *doAnimationStep*, que executa a função *update*, responsável por calcular os valores da simulação, e depois chama a função *draw*, responsável apenas por montar a imagem. A partir disso, foi criado um *state booleano animate*, que define se é permitido executar um passo da simulação, o qual foi passado como propriedade para o componente Tela de Animação.

Esse componente, por sua vez, utiliza outro *hook* do *React*, chamado *effect*, que executa uma função após toda atualização nos dados de um componente. Neste caso, o *effect* recebe o *state animate* como parâmetro. Caso seu valor seja verdadeiro, a função *render* é executada, mas agora ela chama *doAnimationStep* em vez de *draw*. Por outro lado, caso o valor de *animate* seja falso, a função *draw* será executada uma única vez, e o processo de animação é finalizado.

Nesse processo, as funções *draw* e *doAnimationStep*, que estão definidas na classe *Simulador*, são chamadas, mesmo não estando no escopo do componente Tela de Animação. Isso é possível, pois no componente *App*, está instanciado um objeto da classe *Simulador*, cujas funções supracitadas são passadas como propriedades para seu componente filho *Tela de Animação*. Dessa forma, estão sendo chamadas as funções passadas como propriedades, que estão dentro do escopo do componente, ainda que sejam descritas na classe *Simulador*.

É dessa mesma forma que as funções gerenciadoras de eventos, declaradas em *App* são passadas para os componentes filhos. Logo, o clique no botão de envio dos dados de configurações chama uma função passada como propriedade pelo componente Painel, a qual foi passada como propriedade pelo componente *App*, em que está definida a função *handleSubmit*, que captura os dados e atualiza-os no objeto de *Simulador*.

Ademais, o *state animate* passado para *Tela de Animação* é passado novamente para configurações, em que ele é usado para decidir qual texto exibir no botão de controle, qual seja: “Pausar” ou “Continuar”. A este processo se dá o nome de “elevar o *state*”.

Tendo ficado claro o uso dos *hooks* supracitados, é mais fácil compreender a página como um encaixe de blocos estáticos, que se atualizam com a mudança nos *states*. Fundamentado nisso, foi criado o *state camadaExibida*, que guarda um caractere correspondente à camada que deve ser carregada nos componentes, A ou B.

Esse *state* foi passado como propriedade para Painel e para Configurações, sequencialmente. A este último foi anexado o componente Parâmetros da Camada, que encapsulou os campos referentes a fatores da simulação e padrão de distribuição, extraindo-os de Configurações. Isso foi feito, pois deseja-se mostrar na tela apenas os campos relativos à camada exibida. Então, o *state camadaExibida* foi passado mais uma vez para Parâmetros da Camada, em que ele é usado junto ao CSS, para definir o estilo de exibição dos elementos em que estão os campos. Para a camada exibida, nenhum estilo especial é aplicado, mas para as demais, é definido o estilo *display: none*, que esconde o elemento da tela.

Já para mudar o valor do *state camadaExibida*, foi criado um gerenciador de eventos chamado *handleChange* em *App*, que captura o valor do campo, o insere no *state*, e também atualiza essa propriedade no objeto de Simulador. Esse gerenciador é passado

como propriedade para Painel, em que foi criado o seletor de camada. Quando o usuário muda o valor desse elemento, *handleChange* é disparado, atualizando o *state*, e redesenha todos os componentes afetados por ele.

No Simulador, a propriedade *camadaExibida* é utilizada na função *draw* para decidir de qual camada selecionar os dados e construir a imagem. Além disso, foi construído no Painel um elemento que estabelece a escala de cores representada na *Tela de Animação*, e cujo valor máximo é atualizado em *draw*, a depender da camada exibida.

O próximo passo foi continuar a dividir o código em componentes pequenos, seguindo a mesma lógica. Cada campo tornou-se um componente Grupo de *Input*, e cada elemento *select* foi encapsulado em um componente Grupo de *Select*. Esses componentes são genéricos e têm seus valores e atributos definidos pelas propriedades passadas a eles. O evento de alteração de valor desses dispara um gerenciador *handleChange*, que escala até o componente *App*.

Também foram criados dois novos *states* no *App*, quais sejam: *camposParametros* e *camposDimensoes*, que guardam objetos que representam os valores dos campos referentes aos parâmetros da simulação, e daqueles relacionados às dimensões do plano simulado, respectivamente.

Esses *states* são levados até os Grupos de *Input* e *Select*, cujos elementos HTML têm valores definidos pela propriedade correspondente nos objetos de *state* supracitados. Assim, a mudança nos campos altera o *state*, fazendo o componente ser recarregado com o novo valor. A essa estrutura se dá o nome de componente controlado. Ao implementar esse modelo, o gerenciador de envio *handleSubmit* deixa de capturar o valor dos campos, mas apenas define os atributos da simulação como os valores dos *states*.

Em seguida, foi implementada mais uma camada que representará os organismos *physarum polycephalum*, o que implica mudanças nos seguintes componentes: *Painel*, mais precisamente no seletor de camada; *Informação de Concentração*; e *Parâmetros da Camada*. Além disso, os atributos do *state camposParametros* também foram atualizados para guardar os valores da camada C.

3.2. Implementação do modelo de reação e difusão

Diferentes propriedades de compostos foram modeladas neste aplicativo, quais sejam: adição irregular; adição constante; decaimento; difusão e reação entre dois compostos.

Para representar a concentração do composto, definiu-se uma matriz bidimensional de 100 células de altura e 100 células de largura, nomeada “plano”, em que cada célula recebeu um valor aleatório entre 0 e $9,\overline{9}$. A cada passo de animação, cada célula era decrementada em 1 unidade de concentração, o que representa o decaimento.

Passou-se então ao cálculo da difusão, o qual exigia uma função para verificar o valor das células vizinhas por meio do método laplaciano e, com base nesses valores, retornar a concentração a ser removida da ou adicionada à célula da iteração atual. Foram definidos dois pesos referentes às posições: ortogonal $p_O = 0,2$ e diagonal $p_D = 0,05$. Para uma célula afastada das bordas do plano, em que a cercam quatro células ortogonais e quatro células diagonais, a esta posição será acrescentado o valor de cada uma daquelas que a circundam multiplicado pelo peso correspondente. Ainda a essa célula, será subtraída a soma dos pesos, ou seja, 1 de concentração. Noutro caso, de uma quina, por

exemplo, em que há duas células ortogonais e uma diagonal, o acréscimo será limitado aos valores dessas três vizinhas multiplicado pelo peso vinculado, e a redução será de $0,45$. Lógica similar se segue para demais posições nas bordas.

Então, o novo valor da célula é definido por (SIMS, n.d.b):

$$A' = A + \left(D_A \nabla^2 A + AB^2 - k \cdot A\right) \Delta t$$

Em que se consideram estas variáveis, com uma restrição, para garantir que o valor final seja no mínimo 0 para as concentrações:

- A' representa o valor atualizado de concentração do composto A;
- A representa o valor, no início da iteração, de concentração do composto A;
- D_A representa o fator de difusão do composto A;
- $\nabla^2 A$ representa a função laplaciana para cálculo da diferença de concentração entre a célula e sua vizinhança;
- k representa o fator de decaimento constante do composto A;
- Δt representa a diferença de tempo desde o último passo de simulação.

Uma vez que a execução é realizada em um laço duplo, iniciando-se na célula superior mais à esquerda, e finalizando na célula inferior mais à direita, caso os valores fossem diretamente alterados, haveria influência indevida, nas próximas iterações, de fenômenos que devem ocorrer no mesmo momento.

Para solucionar esse problema, foi criada uma matriz de igual tamanho à primeira, na qual serão guardados os valores atualizados. Ao final dos laços, a referência para as matrizes são trocadas, de forma que a mais recente torna-se aquela apontada por "plano".

Convencionou-se, nesse momento, em redefinir os valores iniciais para o intervalo $[0, 1[$. Além disso, foi implementada uma função para inserir uma unidade de concentração às células sobre as quais o usuário arrastar o ponteiro do mouse ou o dedo em uma tela sensível ao toque.

Para prosseguir com a modelagem de reação, era necessário representar os valores da outra camada no plano. Dessa forma, a matriz "plano" passou a guardar, não um valor numérico, mas um objeto literal com os atributos "a" e "b", os quais, sim, guardam as concentrações de cada composto numa célula da matriz.

A reação implementada estabelece que: o composto A é adicionado ao plano numa taxa constante; duas unidades do composto B transformam uma unidade do composto A em B; o composto B sofre decaimento a uma taxa constante; as partículas não são individualmente simuladas, mas sim a concentração delas em cada célula do plano.

Com base nisso, as novas concentrações de cada célula passaram a ser definidas pelas seguintes equações (SIMS, n.d.b):

$$A' = A + \left(D_A \nabla^2 A - AB^2 + f\left(1 - A\right)\right) \Delta t$$
$$B' = B + \left(D_B \nabla^2 B + AB^2 - \left(k + f\right) B\right) \Delta t$$

Em que se consideram estas variáveis:

A' e B' representam os valores atualizados de concentração dos compostos A e B respectivamente;
A e B representam os valores, no início da iteração, de concentração dos compostos A e B respectivamente;
D_A e D_B representam os fatores de difusão dos compostos A e B respectivamente;
$\nabla^2 A$ e $\nabla^2 B$ representam as funções laplacianas para cálculo da diferença de concentração entre a célula e sua vizinhança nos compostos A e B respectivamente;
f representa o fator de adição constante do composto A;
k representa o fator de decaimento constante do composto B;
Δt representa a diferença de tempo desde o último passo de simulação.

Durante a implementação foi identificado um *bug* na equação do composto B, em que, ao adicionar progressivamente composto A a um ponto concentrado do plano, os valores de B poderiam aumentar muito rapidamente, tendendo ao infinito, de forma que não era mais possível representar computacionalmente, momento em que o *JavaScript* retornava o valor especial *Infinity*.

A partir disso, foi elaborada a hipótese de que esse comportamento ocorreria devido ao valor alto de Δt, que impacta em erros de aproximação, o que leva a concentrações extrapoladas. Ademais, o Δt usado para calcular a reação variava conforme o tempo gasto pela função de desenho da imagem na tela, o que não era o ideal, pois torna-o não determinístico. Outrossim, percebeu-se que a função de cálculo era executada mais rapidamente que a função de desenho, de forma que havia possibilidade de otimização, ao recalcular as intensidades várias vezes antes de exibir o próximo *frame*.

Dessa forma, decidiu-se criar uma função chamada *doAnimationStep()*, que calcula os valores da simulação e cria a imagem a ser impressa. Fica estabelecido um tempo fixo de $0,001\overline{6}$ segundos para todo Δt_U (diferença de tempo do cálculo da reação). Dentro da função supracitada existe um laço que executa continuamente a atualização de valores enquanto o acumulador de Δt_U for menor que o Δt_A (diferença de tempo do passo de animação). Ao fim do laço, subtrai-se do acumulador de Δt_U o Δt_A e um único Δt_U. Dessa forma, caso haja milissegundos que não foram considerados neste passo de animação, o Δt_U resultará negativo, para que o tempo restante seja então considerado no passo posterior. Após a implementação desse mecanismo, os cálculos tornaram-se mais precisos, de forma que a explosão do composto B para infinito deixou de acontecer.

4. Resultados Parciais

4.1. Visão Geral do Simulador

O simulador é implementado e publicado em uma página da Web, composta por um elemento HTML *canvas*, em que a simulação é exibida, e por um painel inferior que mostra os parâmetros de cálculo e o estado da simulação, conforme mostrado na Figura 1.

A área de exibição (Figura 2) é formada por células quadradas, que representam a concentração de determinado composto ou organismo numa posição do plano em dado momento. Elas podem variar em uma escala de cores cujos extremos mínimo e máximo são verde e vermelho, respectivamente. Abaixo da área de exibição, estão representadas as somas de concentração de compostos ou organismos em cada uma das camadas.

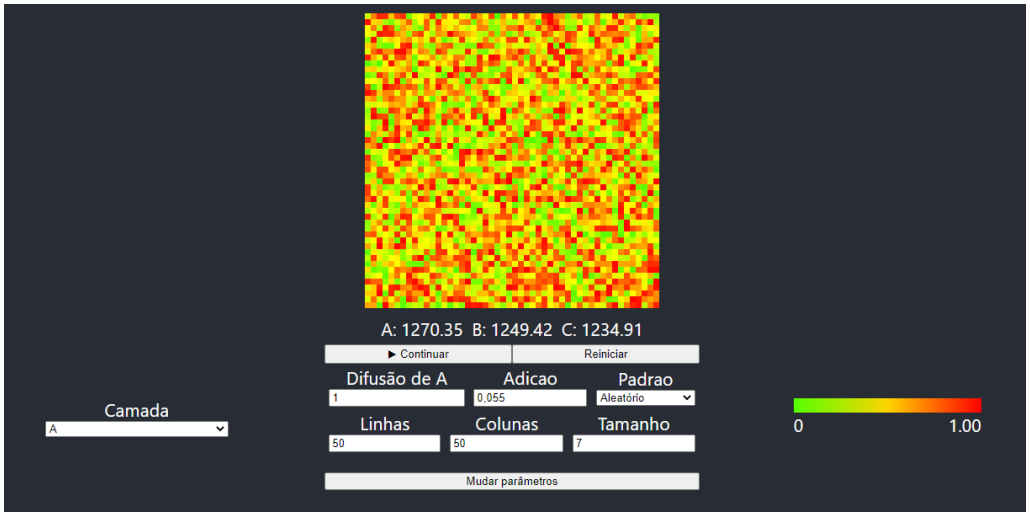
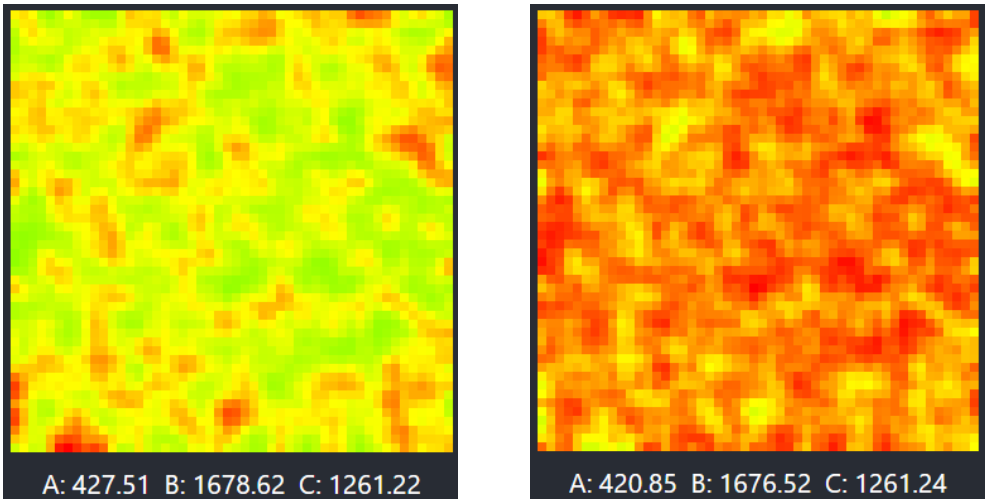


Figura 1: Página do Simulador



(a) Composto A (b) Composto B

Figura 2: Simulação em andamento

O extremo máximo tem valor relativo, sendo definido a cada momento pelo valor da célula que possui a maior concentração. Dessa forma, a escala (Figura 3) no painel inferior mantém a informação da concentração mais precisa, e as células com valores intermediários são redesenhadas de acordo com ela.

Figura 3: Escala de concentração

Uma vez que, no plano, estão sobrepostos diferentes compostos ou organismos, o componente de seleção (Figura 4) permite ao usuário escolher qual camada será dese-

nhada na área de exibição.

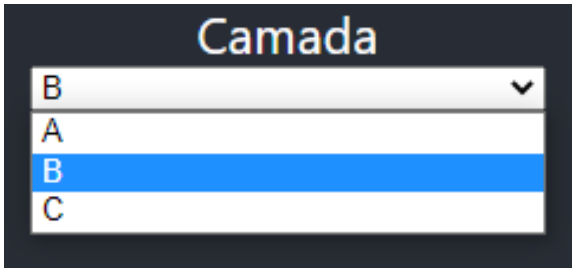


Figura 4: Seletor de camada

O painel de parâmetros (Figura 5) apresenta, na parte superior, botões de controle da simulação, que permite ao usuário interrompê-la ou dar-lhe prosseguimento, e de reinício, que redefine os valores conforme configurados.

Abaixo neste painel, encontram-se campos de configuração para os parâmetros específicos daquela determinada camada selecionada, quais sejam: fator de adição do composto A, fator de decaimento do composto B, fator de difusão dos compostos A e B e do organismo C, e padrão de distribuição de valores naquela camada ao reiniciar a simulação, quais sejam: todas as células assumem valor 0, todas as células assumem valor 1, ou cada célula assume um valor aleatório entre 0 e 1.

A segunda parte do painel de parâmetros define a quantidade de linhas e colunas da simulação, além da largura em *pixels* de cada célula. Na parte inferior, o botão de mudar parâmetros captura os valores alterados e os aplica à simulação.

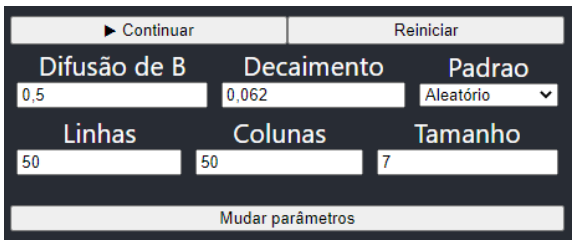


Figura 5: Configuração de parâmetros

5. Considerações Finais e Trabalhos Futuros

Este trabalho apresentou os resultados iniciais para a construção de um ambiente online educacional na forma de uma aplicação web. Modelos de reação, difusão e um método de integração foram implementados. A criação do modelo de quimiotaxia teve início. O protótipo permite observar isoladamente as variáveis do modelo, pausar a simulação, alterar as condições iniciais e as concentrações em um estágio avançado da simulação, o que atende os objetivos específicos do projeto.

No estado atual do protótipo, existem duas limitações maiores: no campo da representação dos modelos e outra em função do desempenho. Os modelos são implementados diretamente pelas suas equações via código. Não há um meio prático de trocar o modelo dinâmico sem alterar o código-fonte. Isso pode ser resolvido em trabalhos

futuros implementando um interpretador de funções matemáticas. Outra alternativa é utilizar modelos descritos em uma linguagem de intercâmbio de modelos, como o CellML (CELLML, 2022).

Sobre o desempenho, para animações em tempo real, o simulador ainda apresenta problemas quando a malha fica maior. Um estudo para diminuir o número de instanciação de objetos e diminuir o trabalho do coletor de lixo do motor JavaScript poderia contribuir para a resolução deste gargalo. Outro caminho é utilizar estruturas mais otimizadas para manipulação de dados brutos, como os vetores tipados do JavaScript (MDN, 2022a).

Até o momento, não foi finalizada a implementação da quimiotaxia, cuja equação está em fase de elaboração. A interface foi atualizada para exibir mais uma camada, referente ao organismo *Physarum polycephalum*, como também foram alocados atributos na classe *Simulador* para guardar seus valores. Os próximos passos serão estabelecer os parâmetros de influência para a quimiotaxia, implementar e testar essa equação.

Na perspectiva de educação tutorial, este projeto explorou ferramentas modernas de desenvolvimento Web, como a biblioteca *React*, garantindo aprendizado além daquele obtido em sala de aula, e aliado à prática. Ademais, possibilitou o contato com aplicações da informática em contextos biológicos, favorecendo a experiência multidisciplinar.

Em um planejamento mais adiante, pretende-se buscar métodos de melhorar o desempenho da simulação, o que ampliará a acessibilidade a aparelhos menos potentes e, assim, permitirá atingir um número maior de usuários. Além disso, é de interesse do grupo estudar formas de simular diferentes modelos definidos via interface, e não pelo código-fonte. Essa mudança pretende facilitar a experiência de uso da aplicação, o que solidificaria a plataforma como meio de educação sobre diversos contextos.

Referências

ADAMATZKY, A.; ALONSO-SANZ, R. Rebuilding Iberian motorways with slime mould. *Biosystems*, v. 105, n. 1, p. 89–100, jul. 2011. ISSN 0303-2647. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S0303264711000608⟩.

CELLML. *The CellML project*. 2022. [Online; acesso em 29 agosto 2022]. Disponível em: ⟨https://www.cellml.org/⟩.

FACEBOOK. *Getting Started*. 2022. [Online; acesso em 31 agosto 2022]. Disponível em: ⟨https://create-react-app.dev/docs/getting-started⟩.

KAY, R. et al. Stepwise slime mould growth as a template for urban design. *Scientific Reports*, v. 12, n. 1, p. 1322, jan. 2022. ISSN 2045-2322. Number: 1 Publisher: Nature Publishing Group. Disponível em: ⟨https://www.nature.com/articles/s41598-022-05439-w⟩.

MDN. *Arrays tipados no JavaScript*. 2022. [Online; acesso em 29 agosto 2022]. Disponível em: ⟨https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Typed_arrays⟩.

MDN. *Getting started with the web*. 2022. [Online; acesso em 29 agosto 2022]. Disponível em: ⟨https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web⟩.

MUELLNER, U. et al. epidemix—An interactive multi-model application for teaching and visualizing infectious disease transmission. *Epidemics*, v. 23, p. 49–54, jun. 2018. ISSN 1755-4365. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S1755436517300270⟩.

PERELSON, A. S. Modelling viral and immune system dynamics. *Nature Reviews Immunology*, v. 2, n. 1, p. 28–36, jan. 2002. ISSN 1474-1741. Disponível em: ⟨https://doi.org/10.1038/nri700⟩.

QUARTERONI, A. Mathematical Models in Science and Engineering. v. 56, n. 1, p. 10, 2009.

QUINTELA, B. d. M. et al. A New Age-Structured Multiscale Model of the Hepatitis C Virus Life-Cycle During Infection and Therapy With Direct-Acting Antiviral Agents. *Frontiers in Microbiology*, v. 9, 2018. ISSN 1664-302X. Publisher: Frontiers. Disponível em: ⟨https://www.frontiersin.org/articles/10.3389/fmicb.2018.00601/full⟩.

ROVEDA, U. *React: o que é, como funciona e porque usar e como aprender*. 2020. [Online; acesso em 29 agosto 2022]. Disponível em: ⟨https://kenzie.com.br/blog/react/⟩.

SIMS, K. *Reaction-Diffusion Application*. n.d. [Online; acesso em 07 julho 2022]. Disponível em: ⟨http://karlsims.com/rdtool.html⟩.

SIMS, K. *Reaction-Diffusion Tutorial*. n.d. [Online; acesso em 07 julho 2022]. Disponível em: ⟨https://karlsims.com/rd.html⟩.