COMMUNICATIONS AND INNOVATIONS GAZETTE

Do C à Web: um relato da criação de uma ferramenta de apoio para game design de jogos de mesa em uma plataforma *online*

Lucas Paiva dos Santos[1], **Celso Gabriel Malosto**[1], **Augusto Castilho**[1], **Igor de Oliveira Knop**[1], **Bárbara de Melo Quintela**[1]

[1]Grupo de Ensino Tutorial em Sistemas de Informação,
Universidade Federal de Juiz de Fora (UFJF)
Rua José Lourenço Kelmer – S/N – Juiz de Fora – MG – Brasil

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

Abstract. *Tabletop games, such as board games and RPGs, are a rising industry, which has attracted investment and new authors every year. Due to restrictions imposed by social distancing, the developing process of these games has become hybrid, based on online tools, which imposes, to the authors, limitations related to the manipulation of existing components in simulation environments available in the market. This work presents an experience report of a solution implemented by tutored students in their initial years. The ideation of the prototype began with modeling in a programming language and has evolved into a real-time web application with three-dimensional elements. The created prototype makes it possible for many players to interact virtually with a Rubik's cube, which is unavailable to game designers using the tools of the market.*

Resumo. *Os jogos de mesa, como tabuleiros e RPGs, são uma indústria em franca ascensão, o que tem atraído investimento e novos autores a cada ano. Com as restrições impostas pelo isolamento social, o processo de desenvolvimento desses jogos se tornou um processo híbrido, baseado em ferramentas online, o que impõe aos autores limitações quanto à manipulação de componentes existentes nos ambientes de simulação do mercado. Este trabalho apresenta o relato de uma solução implementada por alunos tutorados em períodos iniciais. A ideação do protótipo teve início com a primeira linguagem de programação e foi evoluído até uma aplicação web em tempo real e com elementos tridimensionais. O protótipo criado torna possível uma iteração multijogador em um cubo mágico, não disponível para os projetistas de jogos nas ferramentas do mercado.*

1. Introdução

Os jogos são atividades que acompanham a humanidade desde a antiguidade, se se integram na cultura e interagem com a tecnologia da época (HUIZINGA, 1971). Mesmo com a massiva popularização dos jogos digitais, os jogos conseguem, ao mesmo tempo, ligar a tradição à tecnologia: jogos de tabuleiro milenares e competições desportivas são um campo fértil para o emprego de tecnologias modernas, como inteligência e visão computacional, ciência de dados, eletrônica e desenvolvimento de software.

Os jogos de mesa modernos são primariamente atividades sociais e a indústria teve que se readaptar completamente durante o período de isolamento social. Antes utilizadas

como alternativas tecnológicas, as plataformas para partidas *online* se tornaram ferramentas para manter o processo criativo, para manter o número de lançamentos e produção de conteúdo.

Essas plataformas usam uma abordagem diferente do que seria uma pura implementação digital dos jogos: elas replicam os visuais e comportamentos de componentes dos jogos, não as regras. Os jogadores manipulam baralhos, peões e miniaturas, mas ainda são responsáveis por interpretar e impor as regras. Essas plataformas permitiram que autores do mundo todo pudessem criar, desenvolver e apresentar os jogos para as empresas desenvolvedoras de jogos e público.

Entretanto, como são plataformas de uso geral, elas possuem interações com componentes comuns e o processo criativo não tem essa restrição. Vários autores tiveram que mudar um projeto promissor para se adequar ou suspender o desenvolvimento.

O Grupo de Ensino Tutorial de Sistemas de Informação foi procurado por um autor de jogos para a criação de um complemento para o desenvolvimento de um jogo de mesa, que precisaria ser *online*, para permitir se comunicar com várias empresas desenvolvedoras de jogos e jogadores de teste espalhados geograficamente. O projeto do autor exigiria a confecção de um cubo mágico (cubo de Rubik), multijogador. Com as interações básicas, mas com estado compartilhado entre todos. O escopo do trabalho seria desafiador, pois os tutorados envolvidos estavam ainda nos períodos iniciais, concluindo a primeira disciplina de programação.

Neste cenário, teve início um projeto de desenvolvimento com dois objetivos principais: prover uma solução tecnológica para o caso particular e; em paralelo, preparar os tutorados para o uso de tecnologias emergentes do desenvolvimento *web* moderno. Neste texto, segue o relato da implementação completa, realizada ao longo dos cinco primeiros meses de 2022, na Universidade Federal de Juiz de Fora.

2. Fundamentação

O cubo mágico foi inventado em 1974 por Ernő Rubik, e em sua forma mais popular, é composto por um mecanismo de 27 pequenos cubos, chamados *cubelets*, dispostos em um arranjo 3 por 3. As faces aparentes são coloridas e o desafio é embaralhá-las e depois colocá-las novamente em ordem, com restrição de tempo em algumas competições. O cubo mágico, ou cubo de Rubik, é um brinquedo de imenso sucesso se seu uso tem sido estudado em diversas áreas, como combinatória, visão e inteligência computacional, psicoterapia e educação matemática (ZENG et al., 2018).

Os jogos modernos, *designer's games* ou *hobby games* tiveram sua ascensão no mercado alemão entre 1982 e 1996 e, logo se popularizaram no resto do mundo (WOODS, 2012). Esses jogos são caracterizados por dar ênfase em decisões táticas e planejamento estratégico, diminuindo a influência da sorte e da aleatoriedade.

O processo para se criar um jogo de tabuleiro é complexo, exigindo equilíbrio matemático, sessões de teste de jogo com pessoas nos diversos estágios e muitas modificações em função das observações (MARCELO; PESCUITE, 2009). Esse processo foi impactado diretamente pelo isolamento social, exigindo que os autores e empresas desenvolvedoras de jogos utilizassem ferramentas *online* para testar seus jogos pela internet durante a pandemia por COVID-19.

O jogo de tabuleiro é trabalhado pela equipe de criação em ciclos iterativos de testes e modificações. Uma ferramenta para realizar partidas *online* é o Tabletopia[1], que é uma plataforma *online* para publicação de jogos de mesa. Alternativamente, o software Tabletop Simulator[2] também é utilizado, permitindo edição durante uma partida, antes da simulação. O primeiro permite expor o protótipo digital a mais pessoas, pois roda direto no navegador e não exige que os testadores paguem pelo uso. Já o segundo, exige que cada jogador tenha sua própria chave do jogo sendo utilizado, mais frequentemente só entre membros da indústria.

Ambas as plataformas não permitem uma extensão mais complexa, o que levou os tutorados a experimentar alternativas no desenvolvimento *web* e ambientes de execução. A primeira delas, o ReplIt é uma plataforma para desenvolvimento e compartilhamento de software, muito utilizada como ferramenta para aprendizado de programação (REPLIT, 2022). Ele permite a edição e execução compartilhada de código entre várias pessoas.

A linguagem padrão para programação na *web* é o JavaScript (PLURALSIGHT, 2022), uma linguagem de programação interpretada estruturada, de alto nível, com tipagem dinâmica e multiparadigma. Com HTML e CSS, o JavaScript é uma das três principais tecnologias da *World Wide Web*.

O React é uma biblioteca JavaScript de código aberto com foco em criar interfaces de usuário em páginas web (META, 2022). Com o React o desenvolvimento passa a ser realizado em componentes, permitindo reuso e controle do aumento da complexidade da aplicação.

A arquitetura padrão da *web* é a cliente-servidor, na qual o cliente abre uma conexão com o servidor, faz sua solicitação e após uma resposta, esta conexão é interrompida. A Socket.IO é uma biblioteca orientada a eventos para aplicativos da *Web* em tempo real. Ela permite a comunicação bidirecional entre clientes e servidores da *Web* (SOCKET.IO, 2022), garantindo uma conexão permanente quando necessário.

3. Método

O projeto foi dividido em seis etapas: protótipo em linha de comando na primeira linguagem de programação; tradução para JavaScript; desenho dos componentes na *web*; desenho em 3D; comunicação e implantação em tempo real e; validação final. Quando o projeto teve início, os tutorados tinham acabado de concluir a primeira disciplina de programação, desenvolvida na UFJF em ANSI-C e estavam realizando a disciplina de desenvolvimento *web* em JavaScript.

O processo teve início com a modelagem dos movimentos padrão do cubo mágico, realizada com os construtores básicos da linguagem C. Essa etapa concluiu com um protótipo executado no computador, em linha de comando e terminal, sem interface gráfica. Em seguida, o código foi traduzido para a linguagem *JavaScript*. O objetivo era estudar a nova linguagem, mas ainda sem mudar radicalmente o projeto.

Na terceira etapa, passou-se a desenvolver uma interface para navegadores *web*. O cubo passou, então, a ser representado na forma planificada, com as faces dos *cubelets*

[1]Disponível em ⟨http://tabletopia.com⟩.
[2]Disponível em ⟨https://store.steampowered.com/app/286160/Tabletop_Simulator/⟩

coloridas. Foi o momento de alterar a interface com o usuário para realizar os movimentos usando botões. A quarta etapa foi a adição de uma visualização em 3D para o cubo mágico, para permitir girá-lo livremente. Essa etapa deveria usar as tecnologias disponíveis no navegador para aumentar a imersão durante a partida.

Na quinta etapa, foi implementado o sistema multijogador, que consiste em um servidor para a definição de salas e compartilhamento de estado. Desta forma, cada movimento reflete na alteração do estado para os demais. Por fim, a etapa de validação, ainda não realizada, consiste em utilizar o aplicativo em um teste real, durante uma partida com pessoas que não estão envolvidas no projeto do jogo ou ferramenta.

4. Desenvolvimento e Resultados

A modelagem do cubo mágico teve início na linguagem C, em que se visava estabelecer como os dados se relacionam, e implementar a funções de movimentos padrão. As faces dos *cubelets* guardam valores inteiros de 0 a 5, que representam cada uma das cores. Foi definida uma matriz bidimensional de três linhas e três colunas, que representa uma face do cubo mágico. Essa matriz guarda, então, os valores dos *cubelets*, que estão posicionadas num dado momento de tempo. Em seguida, foi definido um vetor de seis elementos, que guarda cada uma das matrizes representando as faces. Dessa forma, o cubo mágico é completamente representado por uma matriz tridimensional de inteiros.

A Figura 1 apresenta as operações de movimento sobre um cubo mágico utilizando a representação por faces. Assumindo que o jogador segura o cubo com a face vermelha apontada para si (figura na superior esquerda), uma operação de girar o topo da caixa faz as faces da linha se moverem em um sentido (imagem superior direita). O movimento de girar a face da frente, gira as faces e as linhas associadas (inferior esquerda) e o giro de um lado gira a face lateral e as colunas respectivas (inferior direita). Estas operações foram decompostas nas funções: *slideRow*, *slideRowInverted*, *slideCol*, *slideColInverted*, *slideSideCol*, *slideSideColInverted*, *rotateFaceClockwise* e *rotateFaceCounterClockwise*.

A função *slideRow*, apresentada no Algoritmo 1, recebe como parâmetros: o índice da linha na matriz bidimensional de face que representa a linha a ser deslizada, e as quatro matrizes de face que terão suas linhas deslizadas em ordem da direita para a esquerda, iniciando da face voltada para o jogador, denotada *front*. Essa operação copia os dados de cada uma das colunas de uma linha, e os sobrescreve sobre cada uma das colunas da linha de mesmo índice na face à esquerda. Uma vez que a face oposta ao jogador, denotada *back* estará conectada ao cubo para inverter a ordem de seus elementos, o índice das colunas a serem alteradas no processo é invertido. No Algoritmo 1, DI é uma constante que representa o número de linhas (e também colunas) das matrizes de face.

A operação *slideRowInverted* é implementada de forma similar, mas o giro é realizado da esquerda para a direita. Em lógica análoga são implementadas as operações *slideCol* e *slideColInverted*, mas o índice da coluna é passado nos parâmetros, e o índice da linha é variável dentro da função, sendo as quatro faces passadas de baixo para cima.

Não basta apenas manipular os dados das linhas ou colunas, afinal, quando é realizado um deslize de face, todos os *cubelets* desta são girados em sentido horário ou anti-horário. Dessa forma, a operação *rotateFaceClockwise* foi criada. Ela recebe apenas a face a ser rotacionada como parâmetro. Dentro de si, é criada uma matriz bidimensional

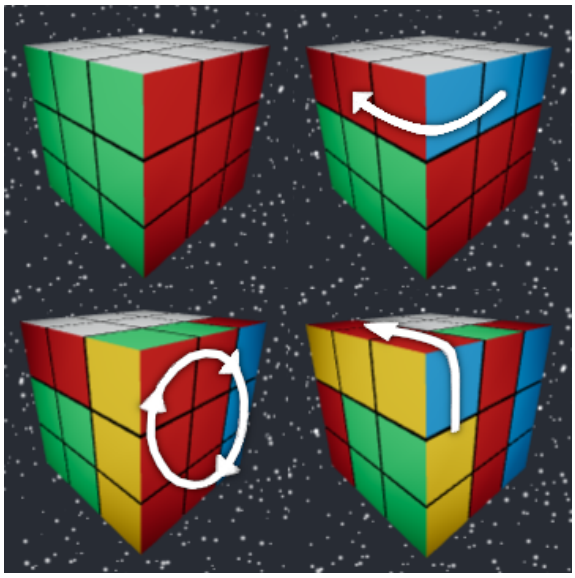

Figura 1. Representação das operações *slideRow* (superior direita), *rotateFaceClockwise* (inferior esquerda) e *slideSideCol* (inferior direita) respectivamente.

```
void slideRow(int row, int face1[][DI],
              int face2[][DI], int face3[][DI],
              int face4[][DI])
{
    int newRow2[DI];
    int newRow3[DI];
    for (int c = 0; c < DI; c++)
    {
        newRow2[c] = face1[row][c];
    }
    for (int c = 0; c < DI; c++)
    {
        face1[row][c] = face4[row][c];
        face4[row][c] = face3[row][DI - 1 - c];
        newRow3[c] = face2[row][DI - 1 - c];
    }
    for (int c = 0; c < DI; c++)
    {
        face2[row][c] = newRow2[c];
        face3[row][c] = newRow3[c];
    }
    return;
}
```

Listagem 1. Operação slideRow

auxiliar, que representa o estado futuro. No sentido horário, uma face do *cubelet* de índice de linha R e de coluna C recebe o valor da posição cujo índice de linha é C subtraído da ordem da matriz, e cujo índice de coluna é R. A operação *rotateFaceCounterClockwise* segue lógica similar, mas a subtração é realizada para encontrar o índice da coluna, e não mais da linha.

Por sua vez, a operação *slideSideCol* gira uma fatia do cubo mágico no sentido anti-horário para os movimentos que não impactam na face orientada na direção do jogador. Nesse sentido, uma linha da face de cima, por exemplo, tornar-se-á uma coluna da face direita. Assim, a manipulação de índices acaba sendo um pouco mais complexa. Essa operação recebe como parâmetros o índice da coluna que será girada da face à direita da face frontal, e quatro faces em ordem, iniciando da direita e, da perspetiva dessa, seguindo de baixo para cima. A operação *slideSideColInverted* é implementada analogamente, mas faz o giro no sentido horário.

Definidas essas operações de movimento, implementou-se uma estrutura condicional para realizar as manipulações adequadas a cada um, quais sejam: *U*, *D*, *L*, *R*, *F*, *B*, e suas contrapartes na direção oposta. Para o movimento *U*, o qual gira a linha de cima para a esquerda, chamam-se as funções *slideRow*, passando como parâmetros o índice de linha 0, e as faces na ordem *front*, *left*, *back* e *right*. Ainda nessa condição, chama-se a função *rotateFaceClockwise* que recebe a face *top* como parâmetro. Já para realizar o movimento F, que gira a face frontal no sentido horário, são chamadas as funções *rotateFaceClockwise*, cujo parâmetro passado é a face *front*, e, em seguida, *slideSideColInverted*, que recebe como parâmetros o índice de coluna 0, e as seguintes faces em ordem: *right*, *top*, *left* e *bottom*. Os demais movimentos são implementados analogamente.

O cubo mágico era exibido no terminal em formato *skybox*, e as letras que representam os movimentos eram digitadas em linha de comando, de forma maiúscula para os padrões, e minúscula para os inversos. Como os valores das cores eram guardados em forma de inteiros, foi criado um vetor de caracteres, que relacionava os dados como índices deste, para assim imprimir uma letra correspondente à inicial da cor (Figura 2).

Em seguida, o código em C foi traduzido para a linguagem *JavaScript*, deixando de ser compilado, para ser interpretado pelo motor *Node.js* (NODEJS, n.d.), que manteve a execução ainda em um terminal. A modelagem de dados e operações manteve-se a mesma. Dada à característica de realizar entrada e saída não bloqueantes do ambiente, foi necessário criar uma interface de captura do movimento pela entrada padrão, e utilizar o operador *await* para aguardar a digitação do movimento antes de seguir para sua interpretação [3]. A Listagem 2 apresenta a alteração necessária para a leitura da entrada, não sendo necessários realizar mudanças mais drásticas no código já implementado em C além de uma tradução para JavaScript.

Para a implementação de uma interface *web*, foi usada a biblioteca *React*, o que exigiu a modularização do código através dos módulos ES6 e da construção de uma série de componentes, representando cada elemento visual na página.

Uma vez instalada no projeto, houve distinta separação entre aplicativo principal e seus componentes, dessa forma, subpastas foram criadas para comportar cada elemento

[3]Mais sobre o operador *await* pode ser visto em ⟨https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Operators/await⟩.

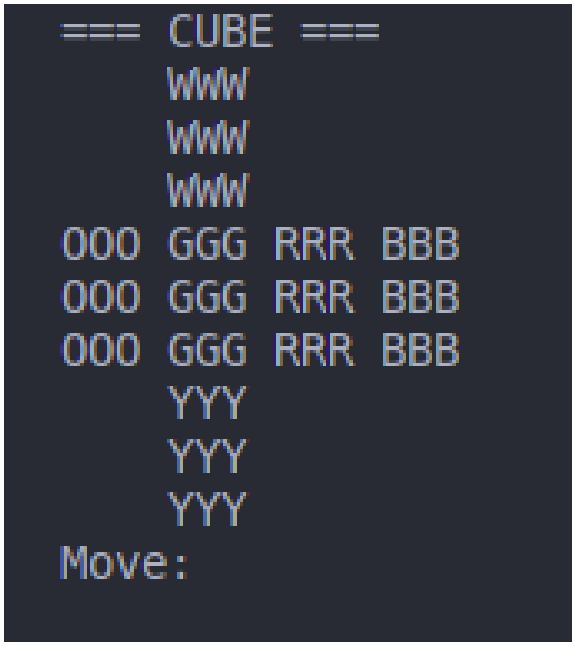

Figura 2. Exibição do cubo mágico desenvolvido em C com a saída padrão no terminal.

como um componente e sua respectiva exportação, que, passou agora, a ser importada pelo aplicativo principal.

A partir dessa conjunção, existem componentes cuja finalidade é exibir o cubo de Rubik planificado, respeitando a perspectiva, estes carecem de mais cuidados, haja visto que são reativos à proporção da tela para melhor confluência com dispositivos móveis, isto é, quando acessado o projeto posiciona as faces conforme o melhor aproveitamento do espaço disponível. Toda essa visualização foi realizada utilizando elementos HTML simples, CSS e *media queries*.

Na etapa seguinte, uma outra visualização foi adicionada, com uma os elementos sendo renderizados em três dimensões. Essa exibição tridimensional foi feita a adição de mais uma biblioteca, a *React Three Fiber*, tirando proveito dos recursos de geometria espacial de figuras sem curvas: *Box Geometry*.

Dessa forma, o projeto passou a articular duas formas de exibição de maneira que, uma vez executada uma ação pelo jogador, seus movimentos refletiam em ambas. Para isso, foi criado um componente *SingleCube*, que consiste, em um *cubelet*, cada um dos vinte e sete cubos, que juntos, compõem o cubo mágico. Porém, dada a natureza física do jogo, no máximo três faces devem ser coloridas enquanto as outras permanecem escuras.

Mais uma vez, foi mantida a coerência com o código inicial, de maneira que cada face é colorida de acordo com um valor inteiro, entre 0 e 5, relacionado a uma cor. Contudo, para que somente as faces visíveis fossem coloridas fez necessária a criação de uma função para respeitar o comportamento da biblioteca *React Three Fiber*, isto é, poliedros são renderizados a partir da ordem que suas faces são dispostas no código, a ordem seguida pela tecnologia é: posicionar a primeira cor a direita, em seguida; esquerda, cima, baixo, frente e costas. Na ausência de uma das seis cores foi adotada a cobertura em preto.

```
import readLine from "readline";

const leitor = readLine.createInterface({
    input: process.stdin,
    output: process.stdout
});
// (...)
async function main() {
    // (...)
    showCube(cube);
    process.stdout.write("Move: ");
    for await (const line of leitor) {
        doMove(line, cube);
        showCube(cube);
        process.stdout.write("Move: ");
    }
}
main();
```

Listagem 2. Uso do operador await para leitura do teclado.

A Figura 3 apresenta o cubo durante laço de montagem dos *cubelets* sendo dispostos na tela, mas interrompidos durante a depuração.

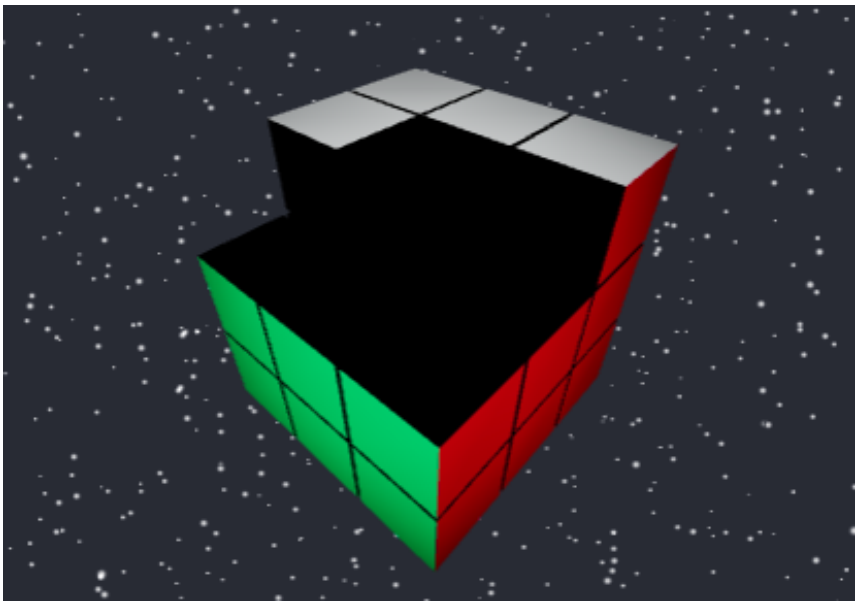

Figura 3. Depuração: interrompido durante a montagem

Em consonância com a implementação *web*, as movimentações feitas pelo jogador deixaram de ser lidas da entrada padrão para darem lugar a botões exibidos na página. Apesar da nova roupagem, o funcionamento dos movimentos não foi alterado, apenas adicionado das funcionalidades de reiniciar o cubo mágico, desfazer movimentos e também

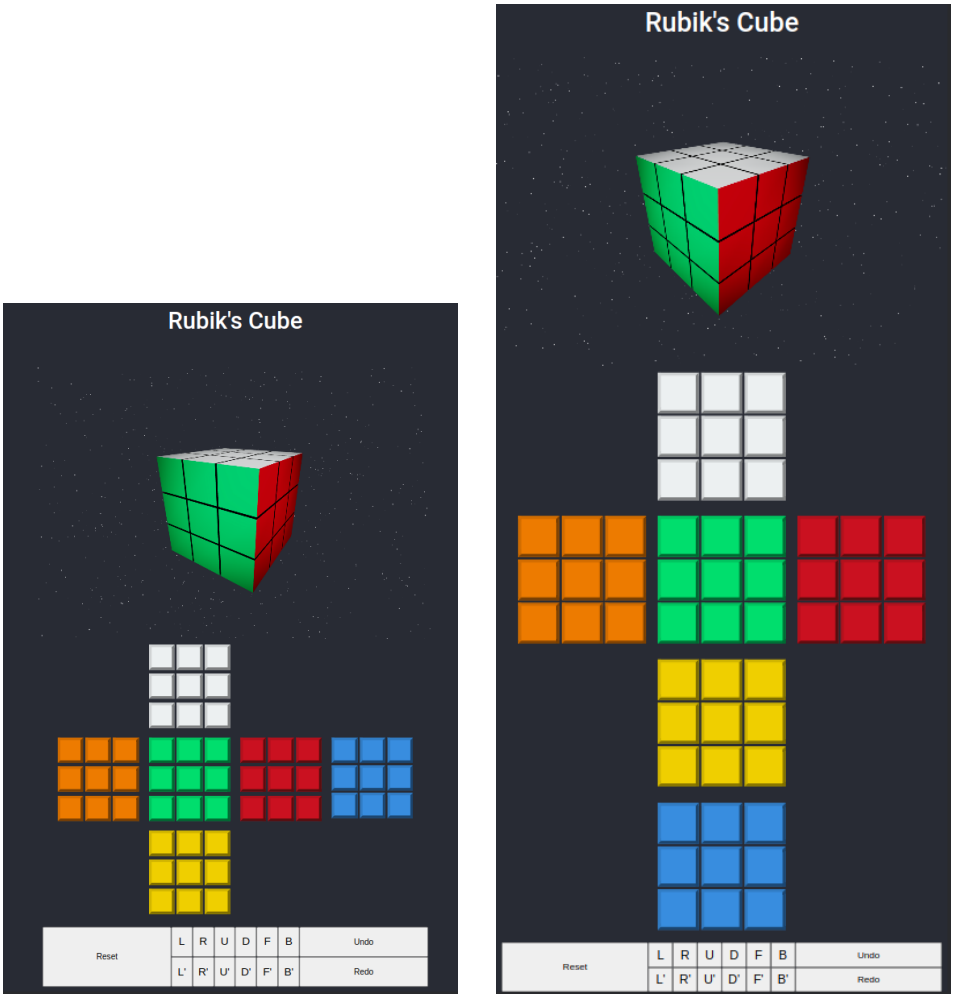

Figura 4. Captura da tela em modo desktop e mobile da aplicação final com o cubo em 3D e planificado.

de refazer movimentos desfeitos (*undo* e *redo*, do inglês). A Figura **??** apresenta a tela final no modo para dispositivos móveis e quando aberta em telas maiores.

Para que pudessem haver salas com estados de cubos diferentes, foi criado um servidor em *Node.js* com o *framework Express* (EXPRESS, n.d.), onde, a cada alteração no estado do cubo, reflete para os outros usuários na mesma sala. Essa funcionalidade se deve ao uso do *WebSocket*, que permite uma comunicação bidirecional entre o cliente e o servidor. Para tal, foi escolhida a biblioteca Socket.IO, que permitiu também a troca de mensagens entre os membros da mesma sala.

A implantação do *Express* cria um servidor *HTTP* com duas funções: servir os arquivos da interface gerados pelo React e fornecer uma porta para o *Socket.io* monitorar para solicitações relacionadas à comunicação em tempo real. Uma vez que um cliente esteja conectado, ele pode tomar ações como entrar em uma sala, enviar uma mensagem ou alterar o estado do cubo.

Ao usuário tomar uma ação, o *frontend* envia ao servidor uma mensagem que é transmitida para todos os outros membros da sala simultaneamente. Isso permite que o

estado seja compartilhado entre todos usuários presentes.

A avaliação final do projeto ainda não foi possível de ser realizada até o fechamento deste texto. Ela exige a observação de uma mesa real e uso da ferramenta para avaliar a usabilidade e aplicação. Entretanto, espera-se realizá-la nos próximos meses. A implantação de testes pode ser acessada livremente pelo endereço ⟨https://rubikcube.up.railway.app/⟩.

5. Considerações Finais, limitações e trabalhos futuros

Os jogos têm uma grande importância na cultura e são uma indústria que atrai e emprega pessoas das mais diversas áreas e com contínua expansão, mesmo durante o isolamento social. Prover novas técnicas, ferramentas e produtos nessa área é contribuir com a economia do entretenimento.

Este trabalho relata a construção de uma solução para um problema da indústria, realizado por um grupo de ensino tutorial de Sistemas de Informação. Mesmo em início de sua formação técnica, foi possível iniciar o trabalho com a tecnologia utilizada em meio educacional e, gradualmente, ir introduzindo novos conceitos avançados das tecnologias em uso no mercado.

Este trabalho, ainda em andamento, carece da validação com os usuários finais, principalmente em matéria de usabilidade e entendimento das operações durante uma partida real. É possível implementar uma telemetria e registro das operações no servidor para observar se o registro de partidas tem um grande número de desfazer ações, que pode ser um indício de problemas de operação ou mesmo a necessidade de melhorar o processo junto ao autor do jogo.

Como limitação de desenvolvimento, a ferramenta carece de animações nas operações. Hoje o movimento é imediato e reflete em uma movimentação instantânea na visualização planificada e 3D. Acreditamos que isso pode ser melhorado durante a fase de avaliação, com bibliotecas e técnicas do próprio CSS3 que realizam a interpolação dos movimentos. Isso contribuirá para a melhor aceitação para o usuário final.

Referências

EXPRESS. *Express - Framework web rápido, flexível e minimalista para Node.js*. n.d. Disponível em: ⟨https://expressjs.com/pt-br/⟩.

HUIZINGA, J. *Homo ludens: o jogo como elemento da cultura*. [S.l.]: Editora da Universidade de S. Paulo, Editora Perspectiva, 1971. v. 4.

MARCELO, A.; PESCUITE, J. Design de jogos: Fundamentos. *Rio de janeiro: Brasport*, 2009.

META. *React, uma biblioteca JavaScript para criar interfaces de usuário*. 2022. Disponível em: ⟨https://pt-br.reactjs.org/⟩.

NODEJS. *Sobre Node.js*. n.d. Disponível em: ⟨https://nodejs.org/pt-br/about/⟩.

PLURALSIGHT. *Celebrating 25 years of JavaScript*. 2022. Disponível em: ⟨https://www.javascript.com/⟩.

REPLIT. *Start coding instantly with Replit's browser-based IDE*. 2022. Disponível em: ⟨https://replit.com/site/ide⟩.

SOCKET.IO. *Bidirectional and low-latency communication for every platform*. 2022. Disponível em: ⟨https://socket.io/⟩.

WOODS, S. *Eurogames: The design, culture and play of modern European board games*. [S.l.]: McFarland, 2012.

ZENG, D.-X. et al. Overview of rubik's cube and reflections on its application in mechanism. *Chinese Journal of Mechanical Engineering*, Springer, v. 31, n. 1, p. 1–12, 2018.