14º Congresso de Inovação, Ciência e Tecnologia do IFSP - 2023

ESTUDO DA RELAÇÃO DAS CARACTERÍSTICAS DOS COMPONENTES DE INTERFACE GRÁFICA DE USUÁRIO E A INTERAÇÃO COM DISPOSITIVOS SENSÍVEIS AO MOVIMENTO

GUSTAVO S. ARAÚJO[1], ANDRÉ C. DA SILVA[2]

[1] Cursando Técnico em Informática Integrado ao Ensino Médio, Bolsista PIBIC-EM, IFSP, Campus Hortolândia, stedile.araujo@aluno.ifsp.edu.br .
[2] Doutor em Ciência da Computação, Docente, IFSP, Campus Hortolândia, andre.constantino@ifsp.edu.br

Área de conhecimento (Tabela CNPq): 1.03.03.04-9 Sistemas de Informação

RESUMO: Novos equipamentos e softwares de interação de usuário surgiram e se popularizaram na última década; pode-se citar telas sensíveis ao toque, acelerômetros e softwares de reconhecimento de movimento, e são empregados em diversos dispositivos, como celulares e consoles. Entretanto a interação com cada equipamento possui peculiaridades que devem ser consideradas, e a mudança do equipamento de interação acarreta em problemas de uso, surgindo assim a necessidade de investigá-los e evitar designs de interface que ocorram nesses problemas. Com essa motivação, foram realizados projetos que estudavam a navegação em páginas Web utilizando o controle sensível ao movimento do Wii e o dispositivo sensível ao movimento Kinect. Este trabalho continuou a investigar a interação por meio dessa modalidade com estudos sobre o uso do periférico sensor de movimento Kinect, periférico sensor de movimento LeapMotion e uso do controle sensível ao movimento WiiMote ao navegar em uma aplicação construída de modo que possibilite coletar dados da interação e a investigar a relação das propriedades dos componentes de interface de usuário gráfica. Como resultado, temos uma aplicação que coleta tempo e número de erros do usuário ao interagir com *checkbox*, podendo ser utilizado em futuros estudos de usabilidade por meio de testes de usuário.

PALAVRAS-CHAVE: Interação com usuário, modalidade de interação, dispositivos sensíveis ao movimento.

STUDY OF THE RELATIONSHIP OF THE CHARACTERISTICS OF GRAPHIC USER INTERFACE COMPONENTS AND THE INTERACTION WITH MOTION-SENSITIVE DEVICES

ABSTRACT: New equipment and user interaction software emerged and became popular in the last decade; we can mention touch screens, accelerometers and motion recognition software, and they are used in various devices, such as smartphones and consoles. However, the interaction with each device has peculiarities that must be considered, and changing the interaction device leads to problems of use, thus arising the need to investigate them and avoid interface designs that occur in these problems. With this motivation, projects were carried out that studied navigation on Web pages using the Wii's motion-sensitive control and the Kinect motion-sensitive device. This work continued to investigate the interaction through this modality with studies on the use of the Kinect motion sensor peripheral, the LeapMotion motion sensor peripheral, and the use of the WiiMote motion-sensitive control when navigating in an application built in a way that makes it possible to collect interaction data. and investigating the relationship of properties of graphical user interface components. As a result, we

have an application that collects time and number of user errors when interacting with a checkbox, which can be used in future usability studies through user tests.

KEYWORDS: User interaction, interaction modality, motion sensitive devices.

INTRODUÇÃO

Novas tecnologias de interação que permitem a manipulação de artefatos digitais, incluindo os dispositivos de interação gestual, são descobertas e visam facilitar e aproximar a interface do contexto natural do usuário, criando experiências únicas, mais prazerosas e memoráveis ao usuário. Não obstante, isso trás a necessidade, ainda mais do que nunca pela difusão acelerada dessas modalidades, do estudo da interação com artefatos digitais empregando essas modalidades de modo a evitar possíveis problemas de usabilidade ocasionados por escolhas de design não apropriadas para as especificidades da modalidade ou que não usufruem do potencial da modalidade.

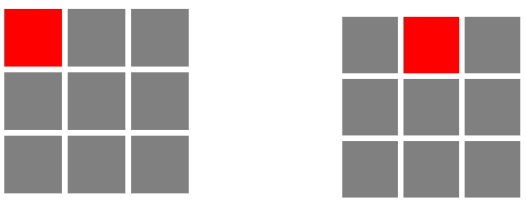
Em trabalhos anteriores do Grupo de Pesquisa Mobilidade e Novas Tecnologias de Interação, que investigaram os dispositivos de interação Kinect, da Microsoft, e Wiimote, da Nintendo, testes com usuários foram feitos para investigar a relação entre o tamanho do componente gráfico botão com o número de erros e piora (ou melhora) na usabilidade. Como resultado, os autores perceberam a clara diminuição da média de erros quando o tamanho do componente gráfico era maior.

Dando continuidade aos trabalhos anteriores, apresentamos nesse artigo uma aplicação em linguagens HTML e JavaScript que registra os movimentos do usuário sobre o componente de interface de usuário *checkbox*, emitindo um relatório sobre tempo (eficiência) e acionamento do componente (eficácia), ao usar sensores de movimento Kinect e LeapMotion e o controle sensível ao movimento WiiMote para interagir com a aplicação.

MATERIAL E MÉTODOS

Como arcabouço teórico, utilizamos a definição de usabilidade proposta por Nielsen (1993 *apud* Rocha e Baranauskas, 2003, p. 29): facilidade de aprendizado (*learnability*), capacidade do usuário de atingir rapidamente um nível de desempenho razoável (eficiência), a facilidade de relembrar (*memorability*), a baixa probabilidade de erros e a satisfação subjetiva do usuário. O tempo e o número de erros são medidas para se mensurar alguns desses componentes. Decidiu-se utilizar a mesma abordagem do trabalho anterior, exibir um conjunto de elementos de interface de usuário, destacando o elemento no qual o usuário deve interagir, adotando as linguagens JavaScript, HTML e CSS ao invés da linguagem Java, adotada nos estudos anteriores. Como método, partimos com a criação da solução primeiramente elaborando um protótipo para, em seguida, produzir a aplicação final. Aqui detalharemos a construção do protótipo e a sua modificação para a construção da versão final do software. Em seguida, a aplicação de testes para averiguar o tempo de resposta com o usuário (espera-se menos de 3 milissegundos) e de corretude (dados emitidos estarem corretos). Assim, este é um estudo em laboratório sem envolvimento de usuários.

Após essa definição, elaborou-se um protótipo empregando as linguagens HTML de JavaScript. Esse protótipo (Figura 1a) consiste na apresentação de uma matriz 3x3 de retângulos (sendo que, na versão final, os retângulos serão substituídos pelo elemento de interface desejado no estudo). Ao iniciar, um dos retângulos fica em destaque e, ao usuário interagir com o elemento (por exemplo, colocar o ponteiro sobre ele e realizar o gesto de ação), o próximo retângulo a interagir é destacado (Figura 2).



(a) (b)

FIGURA 1. Tela do protótipo (a) ao iniciar, com o primeiro retângulo destacado, e (b) novo retângulo destacado após o usuário interagir com o retângulo correto.

Para se calcular o tempo que o usuário permaneceu no botão sem interagir com ele foi implementado o código exposto no Quadro 1.

```
59  function handleMouseEnter(ev) {
60      Game.timer = timer();
61      const botaoCerto = Game.isRightButton(ev.target);
62
63      if (!botaoCerto) {
64          Game.show.textContent = 'dentro do botão errado';
65          return;
66      }
67      Game.show.textContent = 'dentro do botão certo';
68  }
```

QUADRO 1. Função que emite uma saída ao passar com o ponteiro sobre um retângulo.

O objeto *Game* (linha 60 no Quadro 1) representa toda a simulação da sequência de retângulos. Um *timer*, ou seja, um contador é inicializado e guardará a informação de quando se entrou no retângulo, em outros termos, quando a função *handleMouseEnter* for chamada. Verifica-se, então, se o retângulo apertado foi o correto e, caso seja, mostra-se a mensagem apropriada (linha 64 do Quadro 1).

Então, na função *handleClick* (Quadro 2), chamada assim que o usuário interage em qualquer um dos retângulos, irá reger se deverá ou não se passar para o próximo retângulo. Ela fará isso verificando também se o retângulo está correto (linha 79 do Quadro 2) e, caso esteja, chamará o método *nextButton* que procederá ao próximo retângulo (linha 84 do Quadro 2). Em ambos os casos será encerrado o *timer* (que foi inicializado ao se entrar no retângulo) (linha 77 do Quadro 2) e, portanto, será mostrado o tempo que o usuário levou para interagir com o elemento desde que "entrou" nele.

```
75  function handleClick(ev) {
76      const botaoCerto = Game.isRightButton(ev.target);
77      const timeElapsed = Game.timer.end();
78
79      if (!botaoCerto) {
80          Game.show.textContent = `clicou no botão errado, tempo até o click: ${timeElapsed}ms`;
81          return;
82      }
83      Game.show.textContent = `clicou no botão certo, tempo até o click ${timeElapsed}ms`;
84      Game.nextButton();
85  }
```

QUADRO 2. Função que emite uma saída ao clicar com o ponteiro do mouse sobre um botão.

Caso o usuário mova o ponteiro de modo a "sair de cima" do retângulo será encerrado o contador (linha 71 do Quadro 3), que, dessa vez, representará o tempo que o usuário permaneceu com o ponteiro sobre o retângulo específico (linha 72 do Quadro 3).

```
70  function handleMouseLeave(ev) {
71      const timeElapsed = Game.timer.end();
72      Game.show.textContent = `fora do botão, tempo dentro da div ${timeElapsed}ms`;
73  }
```

QUADRO 3. Função que emite uma saída quando o ponteiro do mouse sai de um botão.

Após a construção do protótipo, o ambiente de testes foi configurado considerando aplicações para traduzir movimentos coletados pelo Kinect, pelo LeapMotion e pelo WiiMote em movimento do mouse. Após refinamentos, iniciou-se a construção da versão definitiva do software, considerando a interação do usuário com componente de interface *checkbox* e incluindo uma nova

funcionalidade: transacionar entre vários tamanhos de elementos cada qual com três ordens diferentes de elementos a se pressionar.

Nesse quesito, uma estrutura sequencial foi implementada para se tornar possível essa mudança de etapa com o código do Quadro 4. Ele pede como parâmetro uma sequência arbitrária definida pelo usuário e gerencia sua continuidade pelo método next, que avançará o índice da sequência em 1, e o método *start*, que define o tempo em que a sequência foi iniciada. Há, também, vários métodos getters para expor os dados da sequência tal qual o tempo decorrido desde o início da sequência com o método *getTimeElapsed*.

```
11  function sequencia(ordem, token) {
12      let i = 0;
13      let inicio = undefined;
14      let erros = 0;
15      let finish = undefined;
16
17      return {
18          getToken() { return token; },
19          getErrors() { return erros; },
20          started() { return inicio = undefined; },
21          start() { inicio = new Date(); },
22          finish() { finish = new Date(); },
23          getTimeElapsed() { return finish - inicio; },
24          getCorrectButton() { return buttons[ordem[i]]; },
25
26          next() {
27              i++;
28              console.log(i);
29          },
30          incrementarErro() {
31              erros++;
32          },
33          finished() {
34              return i === ordem.length;
35          }
36      }
37  }
```

QUADRO 4. Função que cria uma sequência.

No Quadro 5, vê–se uma implementação de uma das etapas, nome dado à seção do programa com um tamanho de botão e várias sequência de botões a serem clicados. Ela pede um tamanho de botão e, em seu corpo, vai definir 3 sequências com a função explicitada acima, nesse caso, uma linear, uma vertical e uma pré-selecionada. É essa a função responsável por gerenciar os cliques do usuário e atualizar seu estado de maneira adequada muito semelhante ao programa anterior com a única especificidade de chamar a próxima sequência quando a atual acabar, como visto na linha 86 do Quadro 6. O método *onClick* será um *EventListener* para cliques, aplicado a cada um dos botões.

```
49  function etapa(tamanho) {
50      buttons.forEach(button => button.classList.add(tamanho));
51
52      let sequenciaAtual = 0;
53      let noBotaoCerto = false;
54
55      const sequencias = [
56          sequencia([0, 1, 2, 3, 4, 5, 6, 7, 8], 'linear'),
57          sequencia([0, 3, 6, 1, 4, 7, 2, 5, 8], 'vertical'),
58          sequencia([4, 1, 3, 8, 6, 7, 2, 5, 0], 'pré-selecionada'),
59      ];
```

QUADRO 5. Função que cria uma etapa.

```
78          onClick(ev) {
79              if (noBotaoCerto) {
80                  sequencias[sequenciaAtual].next();
81                  highlight(sequencias[sequenciaAtual].getCorrectButton());
82                  noBotaoCerto = false;
83
84              } else sequencias[sequenciaAtual].incrementarErro();
85
86              if (sequencias[sequenciaAtual].finished()) {
87                  sequencias[sequenciaAtual].finish();
88
89                  if (sequenciaAtual === sequencias.length-1) {
90                      alert(`acabou a etapa de tamanho ${tamanho}`);
91                      exibirRelatorio(sequencias, tamanho);
92                      aplicarEtapa(etapa(tamanhos[tamanhoAtual++]));
93                      return;
94                  }
95
96                  alert('acabou a sequencia');
97
98                  sequenciaAtual++;
99                  sequencias[sequenciaAtual].start();
100                 highlight(sequencias[sequenciaAtual].getCorrectButton());
101                 noBotaoCerto = false;
102             }
103         }
```

QUADRO 6. EventListener de cliques para cada elemento.

No Quadro 7, vê-se a definição da função assíncrona que representa uma etapa, conjunto de sequências com um tamanho definido. Esse é o código que aplica um tamanho, passado como argumento para a função, por meio de outra função, a *applySize*, que modifica o CSS do elemento conforme o tamanho em questão. Após isso, mostra ao usuário um alerta que indica o tamanho da sessão atual e pede confirmação para continuar. Na linha logo abaixo, usa a mesma palavra reservada *await*, vista anteriormente, para pausar a execução da etapa. Mas, dessa vez, espera que toda a função sequência termine sua execução, porque, ao seu final, por ser uma função assíncrona, retorna ao final de sua execução. Com eles em mão, mostra em formato de relatório exibido no canto superior esquerdo da tela os dados coletados, repetindo isso com outras três sequências de ordem, respectivamente, crescente, decrescente e pré-definida.

```
64  //executa cada sequencia de acordo
65  const etapa = async tamanho => {
66      applySize(tamanho);
67      alert(`entrando na etapa de tamanho ${tamanho}`);
68      relatorio.innerHTML += `<h2>${tamanho}</h2>`
69
70      let dados = await sequencia(order(0, 1));
71      relatorio.innerHTML += `<h3>linear</h3>`
72      relatorio.innerHTML += `<p>duração ${dados.duration}ms<br />errors: ${dados.errors}</p>`
73
74      dados = await sequencia(order(8, 7));
75      relatorio.innerHTML += `<h3>decrescente</h3>`
76      relatorio.innerHTML += `<p>duração ${dados.duration}ms<br />errors: ${dados.errors}</p>`
77
78      dados = await sequencia(order(2, 1));
79      relatorio.innerHTML += `<h3>aleatoria</h3>`
80      relatorio.innerHTML += `<p>duração ${dados.duration}ms<br />errors: ${dados.errors}</p>`
81  }
```

QUADRO 7. Função que descreve uma etapa com um tamanho de *checkbox* e suas sequências.

RESULTADOS E DISCUSSÃO

Após concluído as mudanças no protótipo, a versão final do programa, já pensada para a execução dos testes, foi produzida. Nela, houve a substituição dos retângulos pelo elemento *checkbox* (Figura 4). Foram realizados testes com os dispositivos LeapMotion e Kinect, no qual constatamos um tempo de resposta adequado para uma ação do usuário (menos de 3 milissegundos) e a corretude funcional dos dados coletados (contabilizando manualmente os erros e comparando com os resultados emitidos pelo software).

(a) (b) (c)

FIGURA 4. Imagens do software final apresentando as três etapas com seus respectivos tamanhos do elemento *checkbox*: (a) pequeno; (b) médio e (c) grande.

CONCLUSÕES

Com o constante advento de novas tecnologias de interação é necessário investigar as propriedades dos elementos de interfaces de usuário nas diferentes modalidades de interação que emergem, em especial, considerando as características dos dispositivos inventados. O objetivo deste trabalho foi elaborar uma aplicação que permita explorar as propriedades de componentes de interface de usuário (como tamanho e distância) quando usuários utilizam sensores de movimento Kinect e LeapMotion e o controle sensível ao movimento WiiMote para interagir com a aplicação. Após o uso, a aplicação emite um relatório de tempo e de erros ao interagir com uma sequência de componentes de interface *checkbox* e de *combobox*. Testes funcionais foram realizados e os critérios de aceitação foram atendidos. Ressalta-se que este foi um estudo em laboratório, ficando como trabalhos futuros a realização de coleta de dados com usuários reais para, então, refletir sobre valores mínimos para as propriedades desses componentes de interface considerando os critérios de eficiência, eficácia e satisfação subjetiva do usuário.

CONTRIBUIÇÕES DOS AUTORES

G.S.A. contribuiu com o desenvolvimento, implementação e teste de software, e com a redação do manuscrito original. A.C.S. contribuiu com a administração do projeto e supervisão e com a redação - revisão e edição. Todos os autores contribuíram com a revisão do trabalho e aprovaram a versão submetida.

AGRADECIMENTOS

Ao IFSP e ao CNPq pela bolsa de pesquisa na modalidade PIBIC-EM edição 2021/2022.

REFERÊNCIAS

NIELSEN, J. **Usability Engineering**. EUA: Morgan Kaufmann, 1993. 362 p.

ROCHA, H.V.; BARANAUSKAS, C. **Design e Avaliação de Interfaces Humano-Computador**. Núcleo de Informática Aplicada à Educação, UNICAMP, 2003.