Projeto e desenvolvimento de aplicações para dispositivos móveis híbridas a partir de tecnologias para Web: um estudo de caso em jogos digitais

Gabriel do N. Fernandes[1], Igor O. Knop[2], Bárbara M. Quintela[1]

[1] Centro de Ensino Superior de Juiz de Fora (CES/JF)

36.016-000 – Juiz de Fora – MG – Brazil

[2] Departamento de Ciência da Computação - Universidade Federal de Juiz de Fora

Juiz de Fora - MG - Brazil

gabriel_fernandes12@msn.com, igorknop@ice.ufjf.br, barbaraquintela@cesjf.br

Abstract. *There are several tools in the market today for producing applications that generate native code for mobile devices. An important issue that arises from creating applications is software portability. As the number of smartphones is growing in the world, a difficulty arises in developing an application that meets all platforms. This article proposes, as a case study, the creation of a dice game application using HTML, CSS and JavaScript languages and a tool for the creation of a hybrid application aiming to avoid some of the problems found in multiplatform development, called PhoneGap. As a conclusion, we present some positive and negative aspects of developing hybrid applications in relation to natives.*

Resumo. *Atualmente existem no mercado várias ferramentas para produção de aplicativos que geram códigos nativos para dispositivos móveis. Um dos grandes temas sobre a criação de aplicativos diz respeito à portabilidade do software. Como o número de smartphones vem crescendo no mundo, surge a dificuldade em desenvolver um aplicativo que atenda a todas as plataformas. Este artigo propõe, como um estudo de caso, a criação de um jogo de dados utilizando as linguagens HTML, CSS e JavaScript e a ferramenta PhoneGap para a criação de um aplicativo híbrido visando contornar alguns dos problemas encontrados no desenvolvimento multiplataforma. Ao final do artigo são apresentados alguns pontos positivos e negativos encontrados ao se desenvolver aplicativos híbridos em relação à nativos.*

1. Introdução

De acordo com a companhia *Statista* (2017), empresa especializada em análises estatísticas para empresas e universidades, com análises feitas em mais de 20 setores do mercado, ao final de 2017 existirão aproximadamente 4.7 bilhões de usuários de telefones no mundo, chegando à mais de 5 bilhões no ano de 2019. A estimativa no Brasil é que em 2017 o número de usuários de *smartphones* chegue a 65 milhões.

Para Silva et. al. (2015), as ramificações do mercado de dispositivos móveis são feitas por diferentes fabricantes, o que acaba gerando uma série de plataformas de desenvolvimento, além de diferentes sistemas operacionais móveis, *software* e *hardware*. Com a existência de múltiplas plataformas, cria-se uma necessidade de reescrever o mesmo aplicativo múltiplas vezes, cada um devendo ser codificado de maneira diferente para que possa ser executado sob a sua arquitetura específica.

Atualmente os principais sistemas operacionais vendidos no mercado brasileiro são o *Android*, o *IOS* e o *Windows Phone*, que juntos detinham aproximadamente 98% dos sistemas vendidos no Brasil [Statista 2017].

Com essa grande variedade de dispositivos e de sistemas operacionais, torna-se complicada a tarefa de desenvolver um aplicativo que atenda uma grande parcela de usuários e consequentemente do mercado, pois como cada sistema operacional tem a sua própria linguagem para desenvolvimento capacidade de portabilidade do código se torna baixa, aumentando o prazo e o custo para o desenvolvimento de um único aplicativo para diferentes plataformas [Andrade 2015].

Uma das formas de facilitar a portabilidade dos aplicativos é a utilização de *frameworks* para a produção de aplicativos híbridos, dessa forma, com um único código, com pouca ou nenhuma alteração, o desenvolvedor pode realizar a portabilidade para várias plataformas sem ter a necessidade de desenvolver para cada linguagem específica de cada sistema operacional [Tavares 2016].

Neste trabalho será apresentado um estudo de caso utilizando o *framework PhoneGap* com as linguagens *Web HTML5*, *CSS3* e *JavaScript* para a criação de um aplicativo multiplataforma para verificar os benefícios e as desvantagens na criação de aplicativos híbridos. O aplicativo a ser criado para o estudo de caso será um jogo de dados chamado *Liar's Dice*.

O presente artigo está organizado da seguinte forma: na Seção 2 será apresentada uma fundamentação teórica sobre o desenvolvimento para dispositivos móveis e mais especificamente sobre o desenvolvimento de jogos; a Seção 3 apresenta a descrição do estudo de caso do jogo Liar’s Dice, incluindo detalhamento da implementação e discussão dos resultados obtidos. Por fim, a Seção 4 encerra o trabalho com as considerações finais.

2. Desenvolvimento para Dispositivos Móveis

Na criação de aplicativos, um ponto extremamente importante é a escolha do tipo a ser desenvolvido. Dentre os principais tipos estão o nativo, os *Web Apps* e os híbridos. Nesta

seção serão descritos os principais pontos dos tipos de aplicativos citados e um Quadro resumindo esses pontos baseado no trabalho de [SILVA et. al. 2015].

2.1 Desenvolvimento Nativo

Segundo Silva et. al. (2015), aplicativos nativos são normalmente encontradas nas lojas de aplicativos de cada plataforma. São normalmente desenvolvidas a partir de linguagens de programação de alto nível, como o Java para o *Android*, o *Swift* para o *IOS*, ou o C# para o *Windows Phone*. As APIs nativas de dispositivo são fornecidas ao desenvolvedor como uma parte do SDK de cada plataforma. Execução nativa consiste essencialmente em um aplicativo desenvolvido para um dispositivo móvel específico e instalado diretamente sobre o próprio dispositivo com interpretação direta pelo sistema operacional.

Como um aplicativo nativo é instalado diretamente no sistema operacional do *smartphone*, o usuário pode ter acesso à aplicação mesmo estando *off-line*, ao contrário do que acontece com os *Web Apps*, que necessitam de acesso a internet durante todo o tempo.

2.2 Desenvolvimento Web

Web Apps são basicamente sites com o layout e algumas funcionalidades feitas especificamente para dispositivos móveis, fazendo com que a experiência que o usuário tenha seja parecida com a de um aplicativo nativo, adicionando botões e animações que se aproximem dos nativos [TAVARES 2016].

Os *Web Apps* são desenvolvidos com as linguagens padrão da *Web*, HTML5, CSS3, podendo ainda ter *JavaScript* na programação dos mesmos. Pelo fato de serem basicamente sites que serão interpretados pelos navegadores dos dispositivos móveis, os *Web Apps* possuem nenhum ou quase nenhum acesso aos recursos de *hardware* do dispositivo, como a câmera por exemplo, além da necessidade contínua de acesso a internet para utilizá-los [TAVARES 2016].

Um outra forma de se desenvolver um *Web App* é através do conceito de *Progressive Web Apps* (*PWAs*) introduzido pelo *Google* em 2015. Seguindo esse conceito, um *Web App* utiliza as tecnologias *Web* mais modernas para diminuir as diferenças encontradas entre os aplicativos *Web* e os Híbridos e nativos, adicionando funcionalidades como a de notificação do *smartphone* e a possibilidade de acessar o conteúdo do aplicativo enquanto estiver *offline* [WAHLSTRÖM 2017].

2.3 Desenvolvimento Híbrido

Aplicativos híbridos ficam no meio termo entre aplicativos nativos e *Web*. Para Silva et. al. (2015), os aplicativos híbridos são executados dentro de um ambiente de processo nativo na plataforma do dispositivo, assim como os aplicativos nativos. Os aplicativos híbridos comumente são desenvolvidos com um conteúdo HTML5 que é executado por um navegador em tela cheia, que não possui barra de endereços ou os demais controles normalmente imbuídos no navegador. Esses aplicativos usam uma camada de abstração que permite o

acesso as APIs nativas de cada plataforma através da linguagem *JavaScript*, que normalmente não seriam acessíveis a partir de um navegador *Web* móvel padrão.

Quadro 1. Quadro comparativo entre os 3 tipos de desenvolvimento de apps apresentados.

	Vantagens	Desvantagens
Nativo	Acesso à todas as APIs e controles nativos. Interação com o usuário ocorre de maneira mais fluida. Independente de acesso à internet.	Distribuição dependente das lojas de cada plataforma. Utilização de uma nova linguagem para cada plataforma diferente.
Web App	Não precisa de aprovação das lojas de cada plataforma. Roda em vários aparelhos com diferentes tamanhos de tela e especificações. Distribuição e atualização ocorrem de maneira mais rápida.	Não pode ser acessado enquanto estiver offline. Tem menos acessos as funcionalidades nativas do dispositivo. Mais lentos.
Híbrido	Rodam enquanto offline. Acesso as APIs nativas (câmera, GPS, acelerômetro, etc.) Basicamente um único código para múltiplas plataformas Roda em vários aparelhos com diferentes tamanhos de tela e especificações.	Necessitam de aprovação das lojas; Limitação de design. Performance limitada.

Fonte: (Silva et. al. 2015)

2.4 *Frameworks* para desenvolvimento híbrido.

Conforme apresentado em Kasperbauer (2013), os *frameworks* que se destacam dentre os que suportam mais de uma plataforma e aplicam tecnologias web são o *PhoneGap*, *Rhomobile* e *Titanium*. A seguir serão apresentadas algumas características desses *frameworks*.

2.4.1 PhoneGap

PhoneGap é o *framework* desenvolvida pela Adobe, também conhecido como *Cordova*. Utiliza as linguagens web padrão HTML5, CSS3, *JavaScrip*t, *Prototype*, além de suporte para Jquery [PhoneGap 2017].

O PhoneGap tem repositório com *plugins* criados pelos próprios desenvolvedores da ferramenta que dão acesso a várias APIs das plataformas, como a câmera, contatos, status da bateria, acelerômetro, compasso, vibração, geolocalização, transferência de arquivos, informações de rede, etc [PhoneGap 2017].

2.4.2 Titanium

Appcelerator Titanium é um *framework* de desenvolvimento de aplicativos que permite criar aplicações móveis utilizando a simplicidade da linguagem *JavaScript*. Os aplicativos desenvolvidos através do *Appcelerator Titanium* são executados como aplicativos nativos em plataforma *iOS*, *Android*, entre outras [Costa 2012].

Dentre as linguagens que podem ser utilizadas para se desenvolver utilizando o *Titanium* estão o *JavaScript*, *HTML* e *CSS*. Além de utilizar essas linguagens para criar aplicativos móveis híbridos, também pode-se utilizar o Titanium para desenvolver aplicações desktop [Titanium 2017].

2.4.3 Rhomobile

Assim como o *Titanium* o *Rhomobile* também usa as linguagens *web* padrões como *HTML*, *CSS*, *JavaScript* e *Ruby* para criar aplicativos híbridos para as plataformas *IOS*, *Android* e *Windows Phone* [Rhomobile 2017].

Porém, para ter acesso à algumas funcionalidades e *APIs* de cada plataforma, como o acesso a bateria para o *Android* e *IOS*, o desenvolvedor precisa pagar por uma licença.

Para esse trabalho foi escolhido o *framework PhoneGap* por ser um dos mais conhecidos e permitir mais plataformas diferentes, além do fato de ser o único com funcionalidades totalmente gratuitas.

3. Estudo de Caso: Liar’s Dice

O objetivo do presente trabalho é mostrar que é possível utilizar o *framework PhoneGap* para desenvolver aplicativos para diferentes plataformas utilizando o mesmo código-fonte. Para tanto, foi escolhido um aplicativo de jogo de dados chamado Liar's Dice. Nesta seção será apresentada uma breve explicação do jogo.

3.1 Regras

Para se iniciar o jogo, necessita-se de pelo menos dois jogadores com 5 dados de seis faces cada. Em cada rodada um jogador escolhe se vai fazer uma aposta ou chamar o último jogador a apostar de mentiroso (Liar).

Cada aposta consiste em dizer quantos dados na mesa contando os dados de todos os jogadores estão com certa face voltada para cima, por exemplo, a aposta "Cinco 3" significa que o jogador apostou que existem pelo menos cinco dados com a face 3 voltada para cima, contando os dados de todos os jogadores presentes.

A cada rodada os jogadores podem ir fazendo novas suposições sempre aumentando as apostas ou interromper a rodada chamando alguém de mentiroso.

As mãos são ranqueadas primeiro pela quantidade de dados e depois pelo número da face. Assim "Dois 3" é melhor que "Dois 2", mas não que "Três 2".

As apostas nunca devem ser menores que a anterior em relação a face do dado, logo, se algum jogador apostou "Cinco 3" no início da partida, a menor aposta que poderá ser feita para a face 3 seria "Seis 3" ou uma quantidade de dados maior.

Quando algum jogador chama alguém de mentiroso a rodada de apostas é interrompida e os dados são verificados, se o jogador que chamou de mentiroso estiver certo, o jogador que fez a suposição errada perde um dado, se quem chamou de mentiroso estiver errado, ele é quem perde o dado. Quem perde todos os dados sai da mesa até restar somente um jogador com pelo menos um dado na mesa.

3.2 Arquitetura

O aplicativo desenvolvido com o *PhoneGap* age como se fosse um cliente para que o usuário possa interagir com o servidor. O servidor do aplicativo contém toda a lógica do negócio e envia somente os dados para que o aplicativo móvel possa exibi-la. A Figura 1 mostra a arquitetura utilizada pelo *PhoneGap* na criação de aplicativos. O servidor do aplicativo normalmente é um servidor *Web*, podendo ser desenvolvido em qualquer linguagem que suporte os protocolos *Web* padrões, como *Java*, *PHP .NET*, etc.

No caso do aplicativo desenvolvido para o presente trabalho, foi criado um servidor utilizando a linguagem *JavaScript* e utilizado *Websocket* com a API Node.js para as conexões entre o cliente e o servidor.

Figura 1. Esquema representativo da arquitetura básica do Phonegap.

Fonte: Phonegap (2017).

A conexão *Websocket* é feita para se criar um canal de comunicação bidirecional entre o cliente e o servidor. Para rodar o *Websocket* no servidor foi utilizada a biblioteca feita em *JavaScript* chamada *Socket.io*, que roda junto com o node.js. No lado do cliente, o *Socket.io* roda diretamente do browser do usuário. Para se ter acesso ao *Socket.io* e ao servidor por meio de um aplicativo desenvolvido pelo *PhoneGap*, deve-se acessar o arquivo 'config.xml' que é criado dentro da pasta do aplicativo pelo próprio *Phonegap* e modificar a linha <access origin="*" /> e colocar o endereço do servidor com o *Socket.io* instalado. Conforme mostrado na Figura 2, foi utilizado para esse trabalho um servidor local para rodar a aplicação, mas devendo ser alterado no futuro para um servidor externo com o *node.js* e o *Socket.io* devidamente instalados.

Figura 2. Trecho de código-fonte *HTML* contendo o acesso ao servidor pelo Phonegap.

```
<access origin="'http://192.168.0.101:*" />
<allow-intent href="http://*/*" />
<allow-intent href="https://*/*" />
<allow-intent href="tel:*" />
<allow-intent href="sms:*" />
<allow-intent href="mailto:*" />
<allow-intent href="geo:*" />
<platform name="android">
    <allow-intent href="market:*" />
</platform>
<platform name="ios">
    <allow-intent href="itms:*" />
    <allow-intent href="itms-apps:*" />
</platform>
```

Fonte: elaborado pelo autor.

Quando se cria um novo projeto através do aplicativo do *PhoneGap* para *desktop*, os arquivos e diretórios já são criados dentro do projeto conforme exibido na Figura 3. O arquivo *config.xml* bem como a pasta 'www' devem estar obrigatoriamente incluídos dentro da pasta do projeto. A pasta 'www' deve conter as páginas *HTML*, *JavaScript* e todo o *CSS* necessário para o projeto. A página 'index.html' contida na raiz do diretório 'www' é a página principal do projeto, portanto não deve ser excluída ou ter o nome alterado.

Dentro do diretório 'merges', que é opcional, deve ser incluído todo conteúdo que é específico para cada plataforma, como imagens diferentes, *layout* ou *plugins* que acessem a *API* de cada plataforma. Portanto sempre que houver algum conteúdo específico para cada plataforma ele deve ser inserido neste diretório dentro de uma pasta com o nome da plataforma em questão.

Figura 3. Estrutura do diretório do projeto.

Fonte: PhoneGap (2017).

3.3 Controle do Estado do Jogo

Para realizar o controle do estado do jogo foi utilizado o padrão de projeto *State*, que controlará o estado da partida, verificando a possibilidade de fazer apostas, a entrada de um jogador na partida, etc.

O padrão *State* permite a alteração de um objeto conforme o estado em que o mesmo se encontra. Para Figueredo (2014) as alterações e reações realizadas em objetos e eventos se tornam independentes do estado em que o objeto se encontra. O objeto *State* se torna responsável por validar o estado dos demais objetos e realizar as devidas alterações e reações.

O padrão *State* visa desacoplar os estados possíveis do jogo da classe principal, tornando mais fácil a inclusão de novos estados, como rodadas intermediárias e diminuindo a complexidade da classe que controla a partida.

O primeiro estado "*SearchingPlayers*" se dá quando o primeiro usuário se conecta ao servidor, criando o objeto partida. No estado procurando jogadores, não é possível visualizar os dados dos jogadores já conectados e nem fazer apostas.

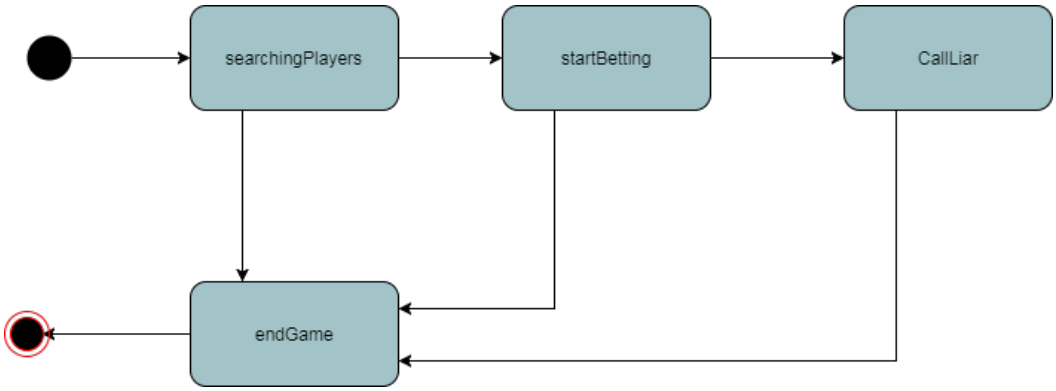
O primeiro estado é alterado para o segundo "*BeginGame*" quando a quantidade definida de quatro jogadores se conecta na partida. Nesse estado os dados são exibidos para os seus devidos jogadores e o primeiro jogador começa a fase de apostas. Nessa fase a opção de duvidar da aposta (Liar) ainda não está disponível pois a primeira aposta ainda não foi realizada.

O estado a seguir é "*startBetting*" no qual a primeira aposta já foi realizada, portanto nesse estado o jogador que estiver em sua rodada poderá realizar uma aposta maior ou duvidar da primeira aposta realizada.

Quando a partida chega ao estado "*startBetting*" o jogador pode realizar a ação de duvidar da última aposta realizada, alterando o estado para "*callLiar*". Nesse estado, nenhum jogador pode fazer mais apostas, o sistema verifica se a última aposta realizada está correta e retira um dado do jogador perdedor, conforme regra explicada na seção 4.1 do presente artigo. Nesta etapa quando um jogador perde todos os seus dados ele é retirado da partida, que prossegue caso ainda haja mais de um jogador restante. Caso só reste um jogador depois do estado "*callLiar*" a partida é alterada para o último estado disponível, "*endGame*" que exibe qual jogador venceu a partida e começa a procurar mais jogadores para a próxima partida.

Todos os 3 primeiros estados podem pular para o estado final "*endGame*" quando não houver o número mínimo de jogadores disponíveis. A Figura 4 mostra o diagrama de estados possíveis para a partida.

Figura 4. Diagrama de estados de uma partida do jogo *Liar's Dice*.



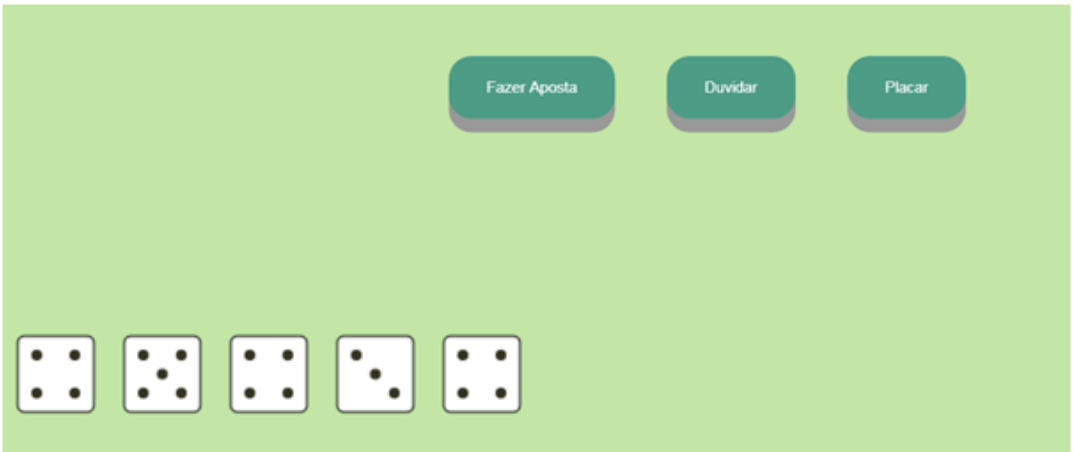
Fonte: elaborado pelo autor (2017).

3.4 Características de Implementação

A interface do aplicativo foi desenvolvida a partir de elementos *HTML*, como o elemento *canvas* para desenhar os dados do jogador na tela e botões com funções em *JavaScript* associadas à eles para realizar as interações do jogo além de exibir informações relevantes ao jogador, como as mensagens de espera de início de partida e mensagem exibindo o nome do jogador que está realizando a aposta.

A Figura 5 mostra a interface do jogo com o elemento *canvas* exibindo os dados além dos botões 'Fazer Aposta' que mostra a interface para a realização de uma nova aposta, o botão 'Duvidar', que realiza a função *'callLiar'* que finaliza a rodada para verificar se os dados da última aposta e decidir se qual jogador perderá os dados, e o botão 'Placar' que exibe todos os jogadores conectados na partida e quantos dados têm cada um deles.

Figura 5. Tela de interface de Aposta para o jogador.



Fonte: elaborado pelo autor (2017).

Para adicionar fluidez na interface foi utilizada a propriedade *transform* do *CSS* para adicionar uma animação de *'fade in'* e *'fade out'* quando houver alternância entre os elementos da página. A Figura 6 mostra um código de exemplo com uma animação para exibir um elemento *div* que exibe o placar do jogo e a classe HTML que tem uma animação para esconder o elemento com uma animação de *fade out*.

Figura 6. Exemplo de código com animação em *CSS*.

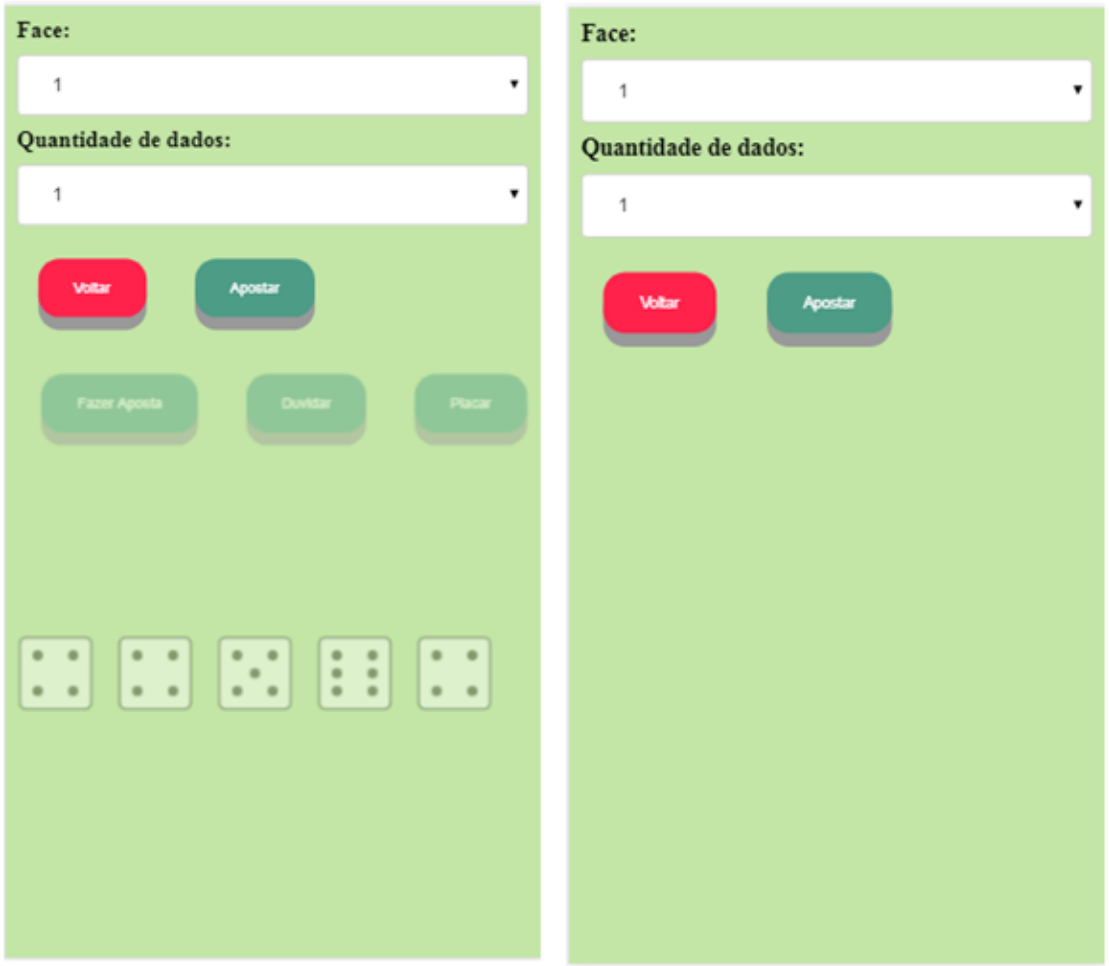
```
div.scoreBoard{
    opacity:1;
    line-height:200px;
    text-align:center;
    -webkit-transition:opacity 700ms ease 300ms,height 300ms ease ;
}
div.hide{
    height:0px;
    opacity:0;
    -webkit-transition:opacity 700ms ease,height 300ms ease 700ms;
    display: none;
}
```

Fonte: elaborado pelo autor (2017).

Nas Figura 7(a) e 7(b) podemos ver a execução do *'fade out'* na interface que exibe os dados e os botões de 'Fazer Aposta', 'Duvidar' e 'Placar' para que fique somente a interface de aposta com campos para o jogador escolher a face do dado e a quantidade para a aposta.

Com a diversidade de dispositivos encontrados atualmente, uma das características que devem estar presentes em qualquer site é o conceito de responsividade (Capacidade de um aplicativo se adaptar ao tamanho da tela do dispositivo que está sendo usado [Wiener 2017]). Para França (2015), projetar sites para o design responsivo não é mais um diferencial, passou a ser obrigatoriedade se quisermos alcançar todos os usuários.

Figura 7. Animação de *Fade Out* ao Apostar.



(a) Início da animação de *fade out*. (b) Final da animação de *fade out*.

Fonte: elaborado pelo autor.

A Figura 8 mostra a função 'init' que cria o elemento *canvas* e atribui a altura e largura do elemento com os mesmos do objeto *HTML 'window'*, que é criado pelo navegador com o mesmo tamanho da tela do dispositivo, seja ele um computador ou smartphone. Dentro da função 'init' também são adicionados dois eventos ao objeto '*window*', que são o *resize* e o *orientationchange* que são acionados sempre que a tela do *smartphone* é alternada entre o modo retrato e paisagem.

Figura 8. Trecho de código-fonte *JavaScript* contendo a função para Canvas Responsivo.

```
var canvas;
var ctx;
function init() {
    canvas = document.getElementById('ctx');
    if (canvas.getContext) {
        ctx = canvas.getContext("2d");
        window.addEventListener('resize', resizeCanvas, false);
        window.addEventListener('orientationchange', resizeCanvas, false);
        canvas.width = window.innerWidth;
        canvas.height = window.innerHeight;
    }
}

function resizeCanvas() {
    canvas.width = window.innerWidth;
    canvas.height = window.innerHeight;
    socket.emit('resize');
}
```

Fonte: elaborado pelo autor.

A Figura 9 mostra o funcionamento da comunicação via *WebSocket* no lado do cliente, ou seja, o código que é interpretado no dispositivo do usuário. O botão 'Apostar' tem uma função associada à ele (*makeBet*) que utiliza a função nativa do *Socket.io* chamada *emit*. Nesse caso a função *emit* enviará ao servidor a *string makeBet* além de um pacote de dados, que são os dois parâmetros utilizados nessa função. Umas das respostas que o servidor pode enviar ao cliente durante a partida pode ser a *beginMatch*, portanto usa-se a função nativa do *Socket.io* chamada *on*. Esta função por usa vez, fica escutando todas as mensagens vindas do servidor, caso a mensagem seja *beginMatch* será executada a função anônima que é o segundo parâmetro desta função neste caso, alternando as existência da classe *hide* que é responsável por alternar a visualização através do *CSS* nos elementos *ui* e *uiWaiting*.

Figura 9. Trechos de código-fonte *html* e *JavaScritp* contendo funções *Websocket* no Cliente.

```
<button class="button close-button" onclick="closeBet()">Voltar</button>
<button class="button" onclick="makeBet()">Apostar</button>
```

```
function showScore(){
    socket.emit('showScore');
    scoreBoard.classList.toggle('hide');
}
function makeBet(){
    socket.emit('makeBet', {face: idFace.value, dice: selectQtd.value})
}

socket.on('beginMatch', function(data){
    uiWaiting.classList.toggle('hide');
    ui.classList.toggle('hide');

});
```

Fonte: elaborado pelo autor

A Figura 10 mostra o código do lado do servidor varrendo a lista com todos os jogadores que estão conectados (*SOCKET_LIST*) e enviando à eles a mensagem beginMatch para iniciar a partida. Depois é verificado se está na vez do jogador ou não, enviando a mensagem corresponde à cada jogador, alterando a interface do cliente para que as apostas sejam feitas somente pelo jogador correto em cada turno.

Figura 10. trecho de código-fonte *JavaScript* de envio de mensagem à todos os jogadores.

```
for(var i in SOCKET_LIST){
    SOCKET_LIST[i].emit('beginMatch');
    if (match.playerList[match.playerTurn] == i) {
        SOCKET_LIST[i].emit('yourTurn', {bet: false});
    }else{
        SOCKET_LIST[i].emit(
            'waitingTurn',
            {
                playerName: Player.list[match.playerList[match.playerTurn]].name,
                bet: false
            }
        );
    }
}
```

Fonte: elaborado pelo autor (2017).

A Figura 11 mostra um exemplo de mensagem sendo trocada entre o cliente e o servidor, com o servidor respondendo somente ao jogador que enviou a mensagem primeiramente, ao invés de enviar a todos os jogadores. O servidor recebe a mensagem *showScore* para mostrar o placar com os dados dos jogadores e retorna somente ao socket do cliente a mensagem *onScore* com um pacote com os nomes e quantidade de dados de todos os jogadores que estão dentro da lista *Player*.

Figura 11. Trecho de código-fonte JavaScript de envio de mensagem para um jogador.

```
socket.on('showScore', function(data){
    var pack = [];
    for(var i in Player.list){
        pack.push({
            name:Player.list[i].name,
            dice:Player.list[i].diceGroup.length,
        });

    }
    socket.emit('onScore', pack);
});
```

Fonte: elaborado pelo autor (2017).

A Figura 12 mostra o construtor do objeto *Player*. O objeto *Player* é sempre instanciado com os valores do *id* do *socket* com o qual o jogador se conectou ao servidor, para o envio de mensagens, o nome de usuário inserido pelo jogador, um identificador único (variável, *number*) e o grupo de dados do jogador. Além disso uma função para retirar um dado da lista de grupo de dados caso o jogador perca a aposta e uma função para rolar os dados novamente.

Figura 12 Classe Jogador no Servidor

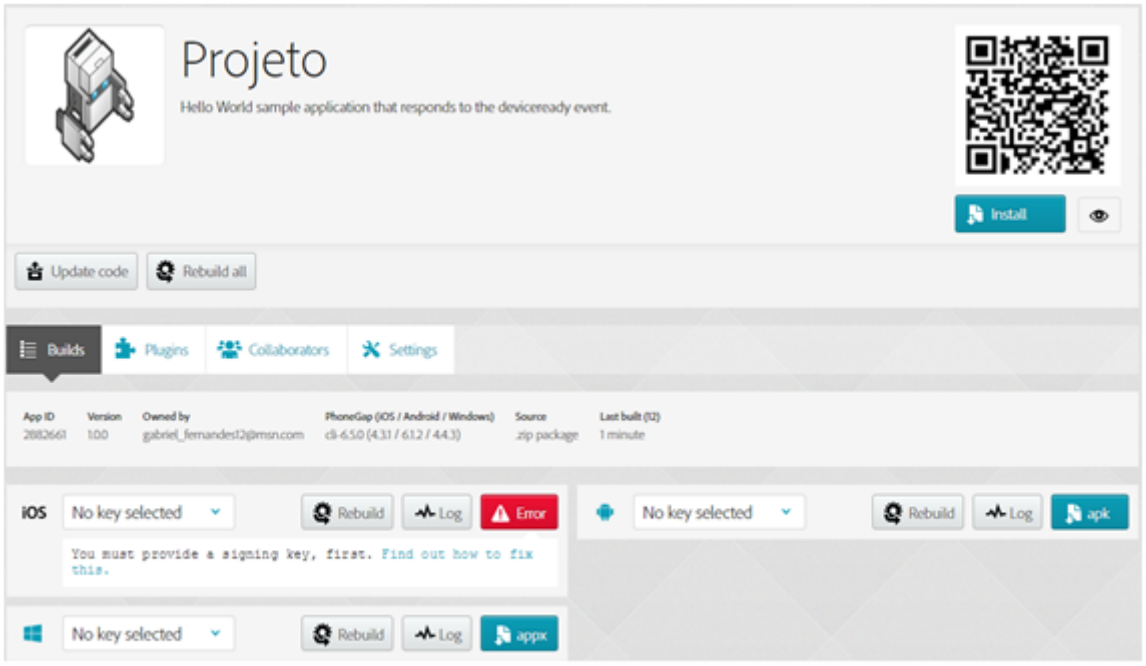
```
var Player = function(id, playerName){
    var self ={
        id: id,
        name: playerName,
        number: "" + Math.floor(10 * Math.random()),
        diceGroup: [
                    Math.floor(6 * Math.random() + 1),
                    Math.floor(6* Math.random() + 1),
                    Math.floor(6 * Math.random() + 1),
                    Math.floor(6 * Math.random() + 1),
                    Math.floor(6 * Math.random() + 1)
                    ],
    }

    self.lostDice = function(){
        self.diceGroup.pop();
        return self.diceGroup.length;
    }

    self.rollDice = function(){
        for(var i = 1; i < self.diceGroup.length; i++){
            self.diceGroup[i] = Math.floor(6 * Math.random() + 1);
        }
    }

    Player.list[id] = self;
    return self;
}
```

Fonte: elaborado pelo autor (2017).

3.5 Distribuição Multiplataforma

A desenvolvedora da ferramenta *PhoneGap* disponibiliza um serviço na nuvem chamado *PhoneGap Build* para que os desenvolvedores possam compilar os aplicativos criados para as plataformas *Windows 8*, *IOS* e *Android* sem a necessidade de instalar a SDK de cada plataforma no computador do desenvolvedor. A Figura 13 mostra o aplicativo desenvolvido para este trabalho sendo compilado para as plataformas *Windows 8* e *Android*. Ambos compilados sem a necessidade de alteração no código, evidenciando a eficiência na economia de tempo e esforço para a criação de um aplicativo para mais de uma plataforma [PhoneGap 2017].

Figura 13. PhoneGap Build para Android e Windows Phone.



Fonte: elaborado pelo autor (2017).

Como mostra a Figura 13, uma das dificuldades encontradas foi a necessidade de possuir um computador e smartphone da *Apple* para que se consiga compilar o código para a plataforma IOS, pois é necessário uma chave dada aos desenvolvedores que são membros pagos do programa de desenvolvimento *Apple Developer Program*.

4. Considerações Finais

O objetivo do presente trabalho foi desenvolver um aplicativo multiplataforma utilizando a ferramenta *PhoneGap*. Com a ferramenta foi possível desenvolver um único código que funciona nas três principais plataformas para smartphones do mercado, *Android*, *IOS* e *Windows Phone*. E como a linguagem utilizada pela ferramenta é a linguagem padrão *Web (HTML)*, o aplicativo também pode ser acessado por computadores pessoais e tablets através do navegador. Mesmo não havendo a necessidade de alterações no código-fonte para compilar e testar o aplicativo para diferentes plataformas, ainda pode existir a necessidade de realizar alterações no momento da publicação do aplicativo nas lojas oficiais de cada plataforma para que possa atender às regras de cada plataforma distinta.

O desenvolvimento de aplicativos híbridos com o uso de linguagens Web padrão se mostrou muito útil para diminuir os custos de produção do mesmo. A criação de um aplicativo com o uso de uma única linguagem mostra indícios de que é possível desenvolver um aplicativo sem a necessidade de uma equipe multidisciplinar, pois não é mais necessário o conhecimento em várias linguagens para cada plataforma utilizada, além do tempo gasto desenvolvendo o mesmo aplicativo para diferentes dispositivos.

Uma das dificuldades encontradas no desenvolvimento do jogo, foi a de criar um *layout* fluido utilizando apenas a linguagem padrão da *Web*. Para a exibição dos dados foi utilizado o elemento canvas e algumas animações em *CSS*, o que pode causar uma experiência do usuário ruim. Outro ponto negativo em utilizar a ferramenta *PhoneGap* com o PhoneGap Build, foi a necessidade de possuir um computador pessoal da *Apple* e uma conta de desenvolvedor simplesmente para compilar o aplicativo desenvolvido para a plataforma IOS.

Este trabalho não explorou todas as funcionalidades que estão presentes na maioria dos jogos e que ficam como sugestão para trabalhos futuros. Tais como a inserção de um banco de dados para criação de contas para os usuários, assim os jogadores poderão ter acesso ao histórico de partidas e poderá ser implementado um sistema de conquistas no jogo. Outra funcionalidade que poderia ser adicionada futuramente seria adição de um chat para os jogadores se comunicarem durante a partida, além de outras funcionalidades de layout utilizando o CSS3, como por exemplo o 3D transforms poderia substituir o canvas utilizado para desenhar os dados e inserir animações de rolamento dos dados e exibição das informações na tela. Tais funcionalidades ficam aqui sugeridas como possíveis trabalhos futuros.

Como resultado final do presente trabalho foi desenvolvido um aplicativo com um único código-fonte para mais de uma plataforma e os principais pontos positivos e negativos desse desenvolvimento híbrido foram apresentados.

Referências

Andrade, Marlon M. (2015) "Projeto e desenvolvimento de jogos eletrônicos multiplataforma: um estudo de caso utilizando Cocos2d-x". Disponível em <https://seer.cesjf.br/index.php/cesi/article/view/300>. Acesso novembro de 2017.

Busso, Thiago Matias (2006) "Soluções Reutilizáveis no Domínio de Jogos Computacionais: A Aplicação de Padrões de projeto no desenvolvimento de motores de jogos". Disponível em <http://www.teses.usp.br/teses/disponiveis/3/3141/tde-07122006-142908/pt-br.php>. Acesso novembro de 2017

Costa, Carlos Humberto Lopes (2012) "Desenvolvimento de aplicativo móvel do serviço web patrool.com utilizando a tecnologia Appceleretor Titanium". Disponível em <http://repositorio.roca.utfpr.edu.br/jspui/handle/1/876>. Acesso novembro de 2017.

Figueredo, Roberto Tenorio (2014) "Padrões de Projeto GOF aplicados ao Desenvolvimento de Jogos Eletrônicos". Disponível em <http://repositorio.ufpe.br/handle/123456789/11981>. Acesso novembro de 2017.

Kasperbauer, Marcelo, Tavares, João, Batista, Marcelo, Filippetto, Alexsandro, Barcelos, Giovane, Silveira, Clóvis, Barbosa, Jorge (2013) “Chronos Mobi: uma aplicação móvel multiplataforma para o gerenciamento de projetos”. Disponível em <http://seer.upf.br/index.php/rbca/article/view/2774/2191> Acesso 27 de agosto de 2015

PhoneGap. Framework PhonGap. Disponível em <https://phonegap.com>. Acesso 8 de junho de 2017.

Rhomobile. Rhomobile API Summary. Disponível em <http://docs.rhomobile.com/en/5.4/guide/apisummary>. Acesso 8 de junho de 2017.

Sato, Adriana Kei Ohashi (2010) "Game Design e Prototipagem: Conceitos e Aplicações ao Longo do Processo Projetual". Disponível em <http://docplayer.com.br/680848-Game-design-e-prototipagem-conceitos-e-aplicacoes-ao-longo-do-processo-projetual.html>. Acesso em novembro de 2017.

Silva, Leandro Luquetti B. da, Pires, Daniel Facciolo e Neto, Silvio Carvalho (2015) "Desenvolvimento de Aplicações para Dispositivos Móveis: Tipos e Exemplo de Aplicação na plataforma iOS Alternative Title: Application Development for Mobile Devices: Types and Application Example on the iOS platform". Disponível em <http://www.lbd.dcc.ufmg.br/colecoes/wicsi/2015/004.pdf>. Acesso novembro de 2017.

Statista. Market share held by mobile operating systems in Brazil from January 2012 to December 2016. Disponível em <https://www.statista.com/statistics/262167/market-share-held-by-mobile-operating-systems-in-brazil/>. Acesso 8 junho de 2017.

Tavares, Henrique Leal (2016) “Introdução a Desenvolvimento de Aplicações Híbridas”. Disponível em <http://www.fatecgarca.edu.br/revista/Volume6/artigos_v6/artigo17.pdf>. Acesso novembro de 2017.

Titanium. Framework Appcelerator. Disponível em <https://www.appcelerator.org/>. Acesso em 8 junho de 2017.

Wahlström, Mikael (2017). “Exploring progressive web applications for health care: Developing a PWA to gather patients’ self assessments”. Disponível em <http://umu.diva-portal.org/smash/record.jsf?pid=diva2%3A1143994&dswid=9997>. Acesso em 9 de dezembro de 2017

WIENER, Lucas, EKHOLM, Tomas, HALLER, Philipp (2017). “Modular Responsive Web Design: An Experience Report”. Disponível em <https://dl.acm.org/citation.cfm?id=3079404>. Acesso em 26 de novembro de 2017.