MultiplayerKit: Um Framework para o Desenvolvimento de Jogos Multiplayer para Dispositivos iOS

João Paulo de Oliveira Sabino, Alysson Diniz dos Santos
Instituto Universidade Virtual - Sistemas e Mídias Digitais
Universidade Federal do Ceará
Fortaleza, Brasil
j.paulo_os@hotmail.com, alysson@virtual.ufc.br

***Resumo*—Este trabalho apresenta o MultiplayerKit, um *framework* que auxilia no desenvolvimento de jogos *multiplayer* para dispositivos iOS. São detalhados o projeto, a arquitetura e as principais funcionalidades implementadas para a ferramenta. São apresentados ainda os resultados dos testes de usabilidade com usuários que indicaram um razoável sucesso na utilização da ferramenta e dos quais foi possível extrair uma lista com melhoramentos que podem tornar a interação mais eficaz. Por se tratar de uma ferramenta grátis, espera-se que o MultiplayerKit possa ser uma alternativa viável para equipes de tamanho reduzido e projetos de escopo limitado.**

***Palavras-chave*-jogo; multijogador; framework; iOS;**

I. Introdução

Desenvolver jogos digitais costuma ser um trabalho complexo por exigir profissionais de diferentes áreas como programação, ilustração, narrativa, sonorização, *game design*, modelagem de personagens, entre outros [1]. Para o desenvolvimento de jogos *multiplayer* (no contexto deste trabalho este termo refere-se a jogos com dois ou mais jogadores conectados através da rede) ainda há um nível maior de complexidade por conta da necessidade de resolver questões relacionadas à rede, como: a definição de uma arquitetura adequada, o desenvolvimento de métodos que possibilitem a troca de informações entre os jogadores, a elaboração de estratégias para tratar as eventuais falhas na rede, entre outras questões.

Considerando tais dificuldades, pesquisadores e profissionais propuseram diversos *frameworks* - ferramentas que auxiliam o desenvolvimento de aplicações [2] - para jogos. Alguns destes são voltados especificamente para o desenvolvimento de jogos *multiplayer* como Photon [3], Unet [4] e Streamworks [5]. No entanto, apesar de facilitarem o desenvolvimento de jogos *multiplayer*, tais ferramentas podem ser muito complexas, especialmente se consideradas equipes inexperientes e projetos de escopo pouco abrangente. Neste contexto, uma ferramenta mais simples é o GameKit, que é focada no desenvolvimento de jogos nativos para plataformas iOS. GameKit facilita a interação entre jogadores e possibilita a comunicação entre dispositivos para criação de partidas em tempo real. No entanto, o GameKit padece de um problema oposto ao das ferramentas anteriormente citadas (Photon, Unet e Steamworks): apesar de ser uma alternativa simples, carece de funcionalidades que são básicas para o desenvolvimento de jogos *multiplayer*, como por exemplo, abstração do fluxo de dados.

O presente trabalho detalha o MultiplayerKit, um *framework* para facilitar o desenvolvimento de jogos *multiplayer* nativos para dispositivos iOS. A escolha por esta plataforma específica deu-se devido ao fato do MultiplayerKit adicionar funcionalidades ao GameKit (que é específico para iOS). As principais funcionalidades do MultiplayerKit são: (i) facilitar a manipulação das trocas de dados entre os dispositivos, (ii) desenvolver métodos para auxiliar a criação de casamentos entre os jogadores (*matchmakings*), e (iii) simplificar as etapas de configurações necessárias para comunicação em rede.

II. Referencial Teórico

Esta Seção detalha três conceitos que são fundamentais para este trabalho: *frameworks* (Seção A), jogos digitais *multiplayer* (Seções B e C) e as ferramentas de desenvolvimento associadas ao sistema operacional iOS (Seção D).

A. Frameworks

Um *framework* é uma aplicação "semi-completa" que auxilia o desenvolvimento de outras aplicações [2]. Um *framework* é, portanto, um conjunto de classes que descreve a arquitetura de um sistema, os tipos de objetos tratados e as possíveis interações entre eles [6]. Há vários benefícios na utilização de *frameworks* [2], dentre eles:

- Reutilização – definição de componentes genéricos que podem ser replicados para criar novas aplicações.
- Modularização – encapsulamento dos detalhes sensíveis de implementação através de interfaces estáveis.
- Extensibilidade – permissão para que as aplicações-cliente estendam as interfaces estáveis do *framework*. É essencial para garantir personalização de novos serviços e recursos.

B. Jogos digitais multiplayer

Jogos digitais (doravante, jogos) são aqueles que comunicam informações digitais aos canais sensoriais do jogador [7]. Por sua característica digital, estes jogos estão necessariamente atrelados a uma plataforma computacional, tais quais: computadores pessoais, consoles, ou dispositivos portáteis [8]. Além de incluírem diversos aspectos (como narrativa, mecânicas, ludicidade, imersão e etc), os jogos ainda podem envolver múltiplos jogadores, sendo denominados jogos *multiplayer*. Segundo Glazer, jogos *multiplayer* podem ser classificados em dois tipos: locais e em rede [9].

Jogos *multiplayer* locais são jogos que foram projetados para dois ou mais jogadores interagindo em uma única plataforma. Do ponto de vista da programação, além de lidar com todos os aspectos inerentes a jogos *single-player* (jogador único), jogos *multiplayer* locais tratam vários pontos de vista e/ou suporte à múltiplas entradas de diferentes dispositivos.

Jogos *multiplayer* em rede são os que têm duas ou mais plataformas conectadas umas às outras durante uma sessão de jogo. Esta conexão pode acontecer via porta-serial, Ethernet (protocolo de conexão de rede local) ou Internet. Neste último podendo haver conexão entre dispositivos geograficamente distantes.

No contexto deste trabalho, trataremos do desenvolvimento de jogos *multiplayer* em rede.

C. Arquiteturas de jogos multiplayer em rede

O desenvolvimento de um jogo *multiplayer* em rede (assim como qualquer aplicação em rede) requer a definição de uma arquitetura que permita a comunicação entre os múltiplos dispositivos envolvidos. As arquiteturas mais utilizadas em aplicações modernas no geral são cliente-servidor e par-a-par [10].

1) Cliente-servidor: Em uma aplicação cliente-servidor há um dispositivo central, denominado servidor, que é responsável por atender as requisições dos outros dispositivos, denominados clientes. Desta forma, como pode ser observado na Figura 1, os clientes não se comunicam diretamente entre si, apenas com a entidade servidor, que possui um endereço IP fixo (identificador único de um dispositivo conectado à uma rede). As aplicações mais conhecidas que utilizam a arquitetura cliente-servidor são Web, FTP, Telnet e e-mail [10].

No caso particular dos jogos em rede, os clientes são os jogadores e toda a comunicação é intermediada por um dispositivo específico que funciona como servidor. Por exemplo, se um jogador deseja enviar uma mensagem de texto para outro jogador, a mensagem será enviada primeiro para o servidor e em seguida o servidor envia para o jogador destinatário. Este tipo de comunicação melhora a segurança, uma vez que o servidor precisa validar cada passo que os clientes fazem, podendo negar determinadas requisições. No entanto, dada a grande quantidade de dados envolvida em uma aplicação multi-usuário que executa a pelo menos 30 *frames* por segundo, o servidor deve possuir uma largura de banda alta a fim de evitar atrasos na comunicação que prejudiquem a experiência dos jogadores [11].

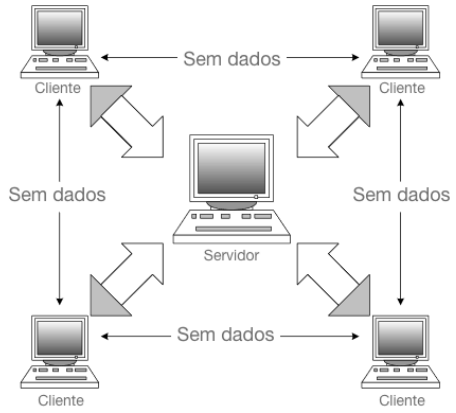

Figura 1. Arquitetura cliente-servidor. Imagem adaptada de [11].

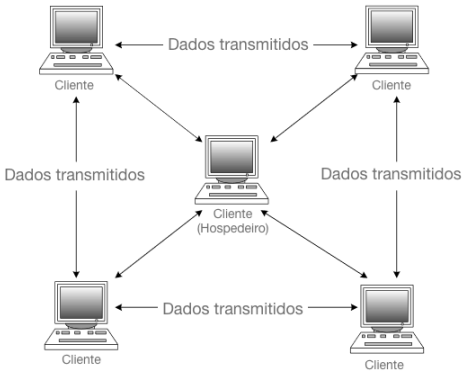

Figura 2. Arquitetura par-a-par. Imagem adaptada de [11].

2) Par-a-par: Na arquitetura pra-a-par (*peer-to-peer* - P2P) duplas de dispositivos conectados alternadamente (denominados pares) comunicam-se diretamente (sem o intermédio de um servidor, ver Figura 2). Os tipos de aplicações mais conhecidos que utilizam esta arquitetura são aplicação de compartilhamento de arquivo (por exemplo, BitTorrent), telefonia por internet (por exemplo Skype) e IPTV (por exemplo, KanKan e PPstream) [10].

No caso específico dos jogos, as arquiteturas P2P possuem um bom custo-benefício, dado que no geral não necessitam de infraestrutura e largura de banda de servidor significativas, diferentemente de arquiteturas cliente-servidor [10]. No entanto, como cada jogador é responsável por enviar requisições e respostas para todos os outros jogadores, os jogos *multiplayer* P2P possuem limitações no número máximo de jogadores [11]. Além disso, na arquitetura P2P os jogadores não precisam esperar permissão para completar

a ação. Deste modo, precisam ser inseridas verificações de segurança, a fim de evitar vulnerabilidade no jogo [11].

D. Tecnologias de desenvolvimento: o sistema operacional iOS, a a rede social Game Center e o framework GameKit

O iOS é um sistema operacional móvel para dispositivos fabricados pela Apple que roda em iPhone, iPad e iPod Touch. O iOS é o segundo sistema operacional mais presente mundialmente, atrás apenas do Android [12]. Ele possui ainda ferramentas de desenvolvimento que apoiam o desenvolvimento de jogos *multiplayer*, tais quais: Game Center e GameKit.

O Game Center é uma rede social para jogos em dispositivos Apple. Game Center implementa diferentes recursos, entre eles: (i) conquistas, que fornecem a capacidade de acompanhar as conquistas de um jogador em seu jogo; (ii) tabela de classificação, que permite o ordenamento das pontuações dos jogadores; e (iii) desafios, que permitem que um jogador desafie outros jogadores para completar uma conquista ou para alcançar uma pontuação na tabela de classificação. O Game Center possui um *framework* específico para auxiliar o desenvolvimento de jogos *multiplayer*, denominado GameKit.

O GameKit possibilita a criação de partidas *multiplayer* em tempo real utilizando arquitetura P2P. Com as funcionalidades do GameKit, jogadores podem convidar ou receber convites de outros jogadores para seus jogos. O GameKit é composto por dois protocolos: o protocolo de conexão e o protocolo de gerenciamento de partida. O *protocolo de conexão* gerencia o *matchmaking*, ou seja, o casamento entre jogadores para realizar uma partida. Também é responsável por realizar autenticação do jogador no Game Center. Por sua vez, o *protocolo de gerenciamento de partida* é responsável por direcionar os dados enviados e recebidos pela rede para os jogadores e notificar alterações de status dos jogadores.

III. METODOLOGIA

A metodologia empregada neste trabalho é embasada no Projeto Dirigido por Exemplo [14] e consiste em um fluxo de cinco etapas principais: pesquisa, planejamento, implementação, testes e avaliação. Estas etapas estão resumidas na Figura 3 e, a seguir, serão descritas em detalhes.

No início, realizou-se uma **pesquisa** exploratória a fim de coletar informações que basearam o desenvolvimento do MultiplayerKit. Nesta etapa, foi analisado e testado *frameworks* já existentes com a finalidade de: (i) elencar possíveis abordagens e técnicas úteis e (ii) identificar falhas ou características que possam ser adaptadas e melhoradas ao aplicar no MultiplayerKit. Esta etapa gerou como resultado uma lista de requisitos para a ferramenta a ser desenvolvida.

Com as informações extraídas na primeira etapa, elaboramos o **projeto** da ferramenta. Nesta etapa foi definida a arquitetura do sistema (detalhada em IV-A), e foi realizado o

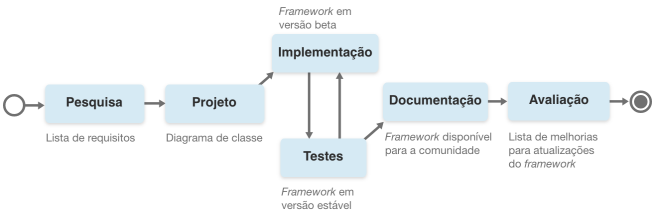

Figura 3. Etapas da metodologia.

planejamento da implementação, considerando os requisitos de reuso, modularização e extensibilidade, naturalmente associados ao desenvolvimento de um *framework*.

Em seguida, foi realizado a **implementação** do *framework* com base nos requisitos, diagramas e arquitetura definidos nas etapas anteriores. Nesta etapa, criam-se abstrações e novos métodos que possam facilitar o desenvolvimento de jogos *multiplayer* para dispositivos iOS. Esta etapa gerou o MultiplayerKit versão beta, ou seja, pronto para ser testado.

Em seguida, realizou-se os **testes** com a finalidade de elencar possíveis erros e inconsistências do MultiplayerKit. Os testes foram divididos em duas categorias: (i) testes unitários que detectem eventuais *bugs* de código, e (ii) testes-prova com desenvolvedores reais, de forma a detectar falhas que afetem o funcionamento geral da ferramenta. Durante esta etapa, fez-se retornar a etapa anterior (implementação) e realizar modificações necessárias de acordo com os resultados obtidos nos testes. Ao fim dos testes foi obtido o *framework* em versão estável (completa e livre dos *bugs* encontrados). Durante esta etapa também foi escrita a documentação, de forma a contemplar todos os aspectos do *framework*, ser de fácil compreensão e incluir bons exemplos de aplicação.

Por fim, foi realizado a **avaliação**, a fim de verificar se o *framework* cumpre o seu objetivo principal e se a compreensão por parte dos desenvolvedores é satisfatória. A avaliação gerou uma lista de melhorias para serem aplicadas nas próximas versões do *framework*.

IV. O FRAMEWORK MULTIPLAYERKIT

O MultiplayerKit adiciona funcionalidades ao GameKit que, como descrito na Seção II-D, é o *framework* de desenvolvimento associado à rede social para jogos em dispositivos Apple (*Game Center*). Ou seja, o MultiplayerKit estende funcionalidades do GameKit que, por sua vez, é um dos componentes do Game Center. Esta relação está graficamente representada na Figura 4.

Este projeto foi desenvolvido no sistema operacional MacOS e as ferramentas utilizadas foram: o ambiente de desenvolvimento Xcode e a linguagem de programação Swift em sua versão 5.0. Por esta ser uma restrição do GameKit, o MultiplayerKit trata apenas de jogos par-a-par. O *framework* em sua versão atual passou por todas as

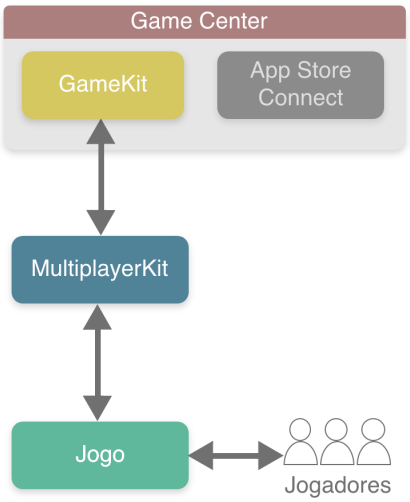

Figura 4. Relação Game Center/GameKit/MultiplayerKit. Imagem gerada pelos autores.

etapas definidas na metodologia (ver Figura 3) e encontra-se disponível em: https://github.com/jonhpol/MultiplayerKit

A. Arquitetura

A Figura 5 apresenta a arquitetura do MultiplayerKit em detalhes. Como pode-se perceber, o MultiplayerKit implementa dois módulos que estendem diretamente as funcionalidades dos protocolos do GameKit (previamente apresentados em II-D) e, por esta razão, são nomeados de maneira idêntica. São eles: o módulo de conexão e o módulo de gerenciamento de partida.

O *módulo de conexão* (praticamente) automatiza o processo de autenticação do jogador. Enquanto que, utilizando apenas o GameKit, o processo de autenticação requer a implementação de várias linhas de código, com o MultiplayerKit, basta criar uma instância do *Matchmaker* e inseri-la na cena de menu para realizar a autenticação.

O *módulo de gerenciamento de partida* é onde está concentrada a funcionalidade mais relevante do MultiplayerKit. Os dados suportados nativamente pelo GameKit são do tipo *Data*, um tipo de baixo nível que é utilizado para manipular bytes na memória. No MultiplayerKit é implementado o tipo *Message*, um protocolo que pode ser implementado por qualquer estrutura que possua atributos com tipos codificáveis (por exemplo: Int, Float, Double, String, Arrays). Desta forma, o MultiplayerKit adiciona uma camada de abstração ao GameKit, com o intuito de facilitar o envio e recebimento de dados.

O MultiplayerKit possui ainda dois protocolos para fazer a interface entre os seus módulos e o jogo: *MKMenuScene* e *MKGameScene*. Estes protocolos devem ser implementados pelo programador do jogo que utilizará o MultiplayerKit nas cenas de menu e de jogo principal, respectivamente, e

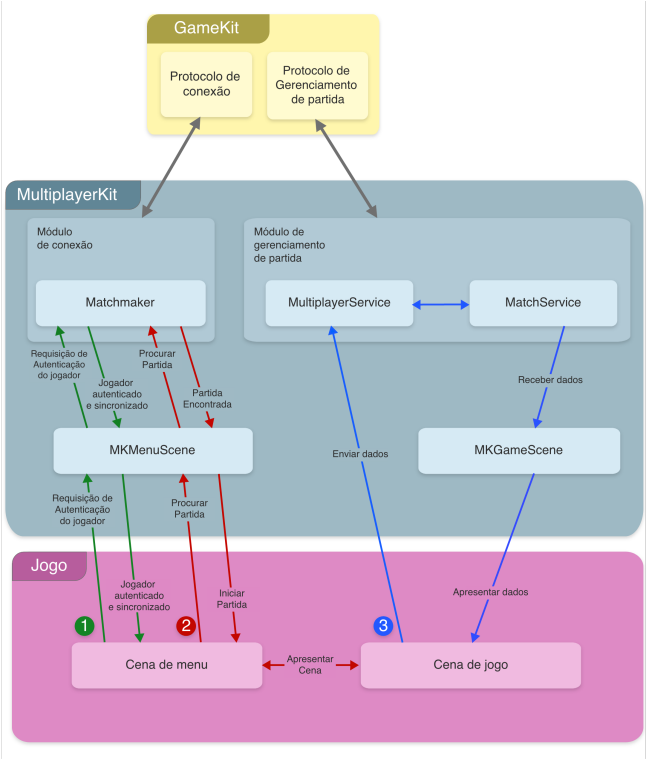

Figura 5. Arquitetura MultiplayerKit. São apresentados os três fluxos que representam as principais funcionalidades do sistema: 1 - auxiliar a criação de *matchmakings*; 2 - facilitar a manipulação das trocas de dados entre dispositivos, e 3 - simplificar as etapas de configuração necessárias para comunicação em rede. Imagem gerada pelos autores.

fazem a ligação entre o jogo e as várias funcionalidades do MultiplayerKit.

B. Funcionalidades

O MultiplayerKit possui três funcionalidades principais: (i) auxiliar a criação de *matchmakings*, (ii) facilitar a manipulação das trocas de dados entre os dispositivos e (iii) simplificar as etapas de configuração necessárias para comunicação em rede.

1) Auxiliar a criação de matchmakings: Primeiramente - como pode ser observado no fluxo indicado por 1 na Figura 5 - a cena de menu do jogo faz, através de *MKMenuScene*, uma requisição para *Matchmaker* realizar a autenticação de um jogador. Esta requisição é processada pelo *Matchmaker* que retorna para a cena de menu do jogo a confirmação de autenticação do jogador.

Após a autenticação e sincronização - como indicado no fluxo 2 na Figura 5), o jogador pode iniciar a procura de partida, ou convidar amigos a partir da cena de menu. Quando o jogador encontra uma partida disponível, o *Matchmaker* avisa a cena de menu que a partida irá começar e a cena de menu apresenta a cena de jogo quando isso for acontecer.

2) Facilitar a manipulação das trocas de dados entre os dispositivos: O fluxo 3 na Figura 5 indica o cam-

inho das trocas de mensagens utilizando MultiplayerKit. As classes envolvidas (*MultiplayerService*, *MatchService* e *MKGameScene*) adicionam o suporte ao tipo de dados *Message*, que é uma abstração do tipo de dados *Data* tratado nativamente pelo protocolo de gerenciamento de partida do GameKit. Com isto, o desenvolvedor que utiliza o MultiplayerKit trata dados de nível mais alto na sua implementação, sem preocupar-se com questões de gerenciamento de memória que podem surgir utilizando apenas o GameKit.

3) Simplificar as etapas de configuração necessárias para comunicação em rede: Do ponto de vista do desenvolvedor, no que se refere à configuração, basta confirmar os protocolos *MKMenuScene* e *MKGameScene* na cena de menu e na(s) de jogo, respectivamente. Em seguida, deve-se criar as mensagem personalizadas específicas do seu jogo (por exemplo: movimentação, ataque, defesa, especial, *chat* de texto, dentre outros). Com estas configurações feitas, basta chamar os método de enviar dados do *MultiplayerService* e tratar os dados recebidos como quiser.

V. Avaliação

O objetivo do MultiplayerKit é facilitar o desenvolvimento de jogos *multiplayer* nativos para dispositivos iOS. Para tanto, é importante que o utilizador consiga compreender a ferramenta para que possa usá-la para estruturar seus próprios jogos. Para investigar preliminarmente se a ferramenta atende essas necessidades, decidiu-se pela realização de um teste de usabilidade.

Usabilidade pode ser definida como a extensão em que um produto pode ser utilizado por usuários para atingir metas específicas com eficácia, eficiência e satisfação em um determinado contexto de uso [17]. Portanto, os testes de usabilidade possuem o objetivo de gerar uma lista de melhorias para serem aplicadas ao *framework*, conforme planejado para a última etapa da metodologia (ver Figura 3).

A. Participantes

O perfil escolhido para a avaliação foi o de profissionais que possuem experiência no desenvolvimento de jogos e aplicativos nativos da plataforma iOS. A escolha por este público-alvo particular pretende minimizar possíveis erros causados por eventual falta de familiaridade com o ambiente iOS ou com a linguagem Swift (utilizada para desenvolvimento com iOS).

O MultiplayerKit foi testado por 5 participantes que são desenvolvedores do programa Apple Developer Academy, do sexo masculino e com idade entre 18 e 26 anos. Todos possuem experiência de aproximadamente 2 anos em desenvolvimento de jogos e aplicativos iOS.

B. Protótipo

Para a realização dos testes foi criado um protótipo de jogo mobile para iOS, disponível como ferramenta de ensino do MultiplayerKit (encontra-se disponível em: https://github.com/jonhpol/MultiplayerKit/tree/master/Demo). O protótipo é um jogo estilo *spaceshooter*, de até 4 jogadores, onde cada jogador controla uma nave. É possível mover-se utilizando o controle localizado na esquerda do dispositivo e realizar disparos utilizando o botão A, como mostrado na Figura 6.

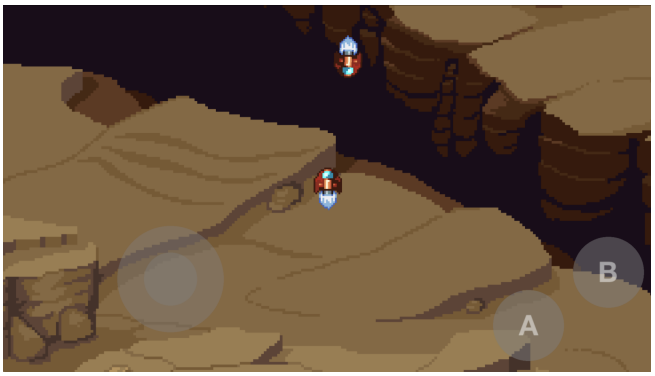

Figura 6. Protótipo desenvolvido. Neste exemplo há duas naves representando dois jogadores conectados na partida. *Sprites* criados por Luis Zuno.

Antes de iniciar o jogo, há a cena de menu, onde o usuário pode iniciar o jogo online (esperar o *matchmaking*), ou entrar no modo de treino(offline). Com a partida iniciada, todos os jogadores podem atacar uns aos outros e o objetivo do jogo é ser a última nave sobrevivente.

A ideia geral da avaliação é apresentar aos testadores uma versão offline do protótipo e pedir para que, utilizando MultiplayerKit, eles implementem a possibilidade da conexão entre 4 jogadores.

C. Instrumentos

Para a coleta de dados foi utilizado um formulário para registrar a efetividade do cumprimento de cada tarefa e realizar apontamentos que pudessem ser relevantes para o teste.

O formulário apresentava as 10 tarefas que englobam todas as funcionalidades presentes no MultiplayerKit:

- T1: Ler e compreender a documentação.
- T2: Implementar o protocolo *MKMenuScene* na cena de menu.
- T3: Criar instância da classe *Matchmaker*.
- T4: Instanciar a cena de menu com o *Matchmaker* e apresentá-la.
- T5: Criar ação para o botão de iniciar partida para exibir o *Matchmaker*.
- T6: Instanciar a cena de jogo quando a partida for encontrada.
- T7: Criar mensagens customizadas
- T8: Implementar o protocolo *MKGameScene* na cena de jogo.
- T9: Alocar todos os objetos(naves) representando os jogadores na cena.

- T10: Enviar e receber dados de pelo menos dois tipos de mensagens. As mensagens recebidas devem ter algum *feedback* visual na cena do jogo.

Em seguida, após a realização das tarefas, foi realizada uma entrevista estruturada com a finalidade de recolher dados qualitativos acerca da impressão dos participantes com relação à ferramenta. As perguntas realizadas aos usuários foram:

- Q1: Em sua opinião, a documentação foi clara e objetiva?
- Q2: Você teve dificuldade em implementar algum protocolo? Se sim, quais dificuldades?
- Q3: Foi fácil criar mensagens customizadas?
- Q4: Você teve dificuldade em realizar o envio e/ou recebimento de mensagens? Se sim quais dificuldades?
- Q5: Gostaria de comentar algo mais a respeito do *framework*?

As respostas dos testadores foram transcritas pelo moderador do teste e, posteriormente, estes resultados foram lidos e codificados em várias etapas de refino, de acordo com a metodologia descrita em [18].

D. Procedimentos

Os testes foram realizados no laboratório da Apple Developer Academy, localizado no prédio de Pesquisa do Instituto Federal de Educação, Ciência e Tecnologia do Ceará, durante o mês de Novembro de 2019. O laboratório representa um dos lugares nos quais o MultiplayerKit poderia ser de fato utilizado, o que representa que ele foi testado em um ambiente similar ao que seria o real.

Os materiais necessários foram um MacBook para os usuários realizarem os testes, e um formulário e um caderno de anotações utilizados pelo moderador. O teste ocorreu individualmente com cada usuário. Houve a necessidade de assinar um termo de consentimento, explicitando sobre o propósito da avaliação e a privacidade das suas informações. A avaliação foi dividida em três etapas: orientação inicial e apresentação do protótipo, realização do teste e, por fim, entrevista pós-teste.

Na primeira etapa, foi realizado uma breve orientação geral a fim de familiarizar os usuários antes da avaliação e a apresentação do protótipo (onde foram apresentadas as suas mecânicas básicas).

Na segunda etapa foi realizado o teste com as tarefas a serem cumpridas pelos usuários(de T1 à T10). Estas tarefas integram todo o fluxo principal do MultiplayerKit. Como todas as tarefas estão interligadas, houve a necessidade de intervenção do avaliador apenas quando alguma tarefa não foi cumprida com sucesso, pois foi essencial para dar prosseguimento à avaliação. As tarefas que foram necessárias esta interferência foram consideradas tarefas sem êxito para o fim desta avaliação.

Na terceira etapa, foi realizada a entrevista com os usuários com perguntas a respeito das impressões que tiveram ao utilizar o MultiplayerKit.

E. Ameaças à validade

Inicialmente, a coleta foi realizada com um número limitado de 5 usuários. Este número de participantes é contraditório, sendo considerado ideal por alguns autores [19], mas baixo por outros [20]. Apesar disso, considerando tratar-se de um estudo em fase inicial e de uma ferramenta de tamanho pequeno, onde 10 tarefas esgotam as possíveis interações, esperamos que o número de testadores seja suficiente para extrair interessantes informações iniciais.

Também entendemos que o perfil dos testadores (todos homens, entre 18 e 26 anos e residentes em Fortaleza) pode inserir um viés indesejado nos dados coletados. Porém, este perfil representa de fato um potencial público-alvo da ferramenta (ainda que não represente todos os possíveis públicos-alvo) e foi o possível de ser envolvido nos testes, considerando para o tempo disponível para a realização deste trabalho.

VI. Resultados e discussões

De forma geral, apesar de pequenas falhas, os usuários foram capazes de realizar a maior parte das tarefas sugeridas. Como pode ser observado na Figura 7, 2 usuários (P2 e P5) não cometeram erros, 1 usuário (P1) cometeu apenas 1 erro, enquanto 2 usuários (P3 e P4) cometeram 3 erros. Os participantes também pareceram satisfeitos ao realizarem os testes, o que pode ser observado a partir de algumas afirmações elencadas na entrevista:

- P1: “Ficou fácil fazer a comunicação dos jogadores utilizando este *framework*.”
- P2: “Não perdi tempo tendo que escrever códigos maçantes.”
- P4: “Gostei da proposta das mensagens fazerem parte de um protocolo, tornando a implementação mais genérica.”

Desta forma, pode-se considerar que o MultiplayerKit obteve um resultado razoável na avaliação. Porém, a análise dos erros por tarefa (Figura 8), permite a identificação de alguns possíveis pontos de melhoria no uso do *framework*:

A. A instanciação da cena de menu deve ser melhorada

Esta conclusão é relacionada à tarefa T4 (instanciar a *MenuScene* com o *MatchMaker* e apresentá-la) que obteve 30% de erros. Alguns usuários tiveram dificuldades nesta tarefa, pois, para o adequado funcionamento da MultiplayerKit é necessário instanciar a cena de menu juntamente com o *Matchmaker*. Entretanto, esta é uma restrição que não está posta no *framework*, ou seja, também é possível instanciar a cena de menu sem o *Matchmaker*.

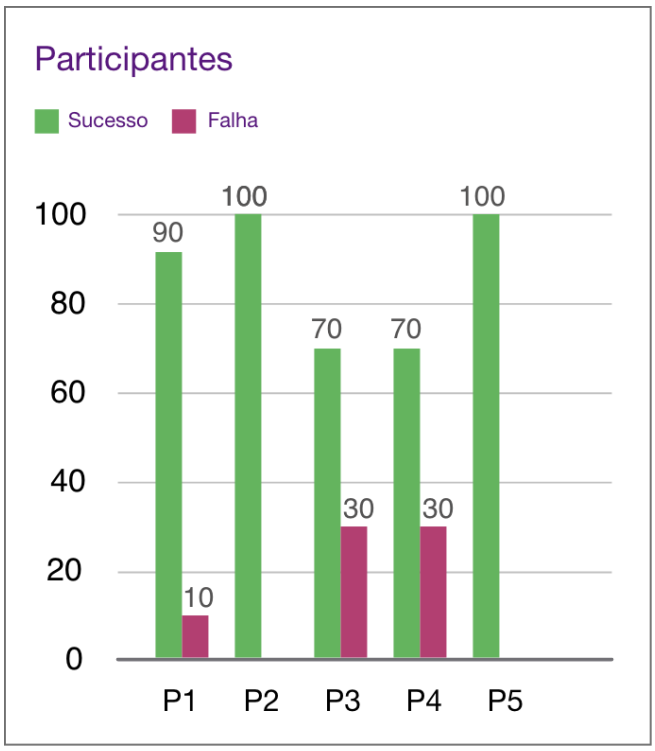

Figura 7. Taxa percentual de acertos e erros por participantes. Imagem gerada pelos autores.

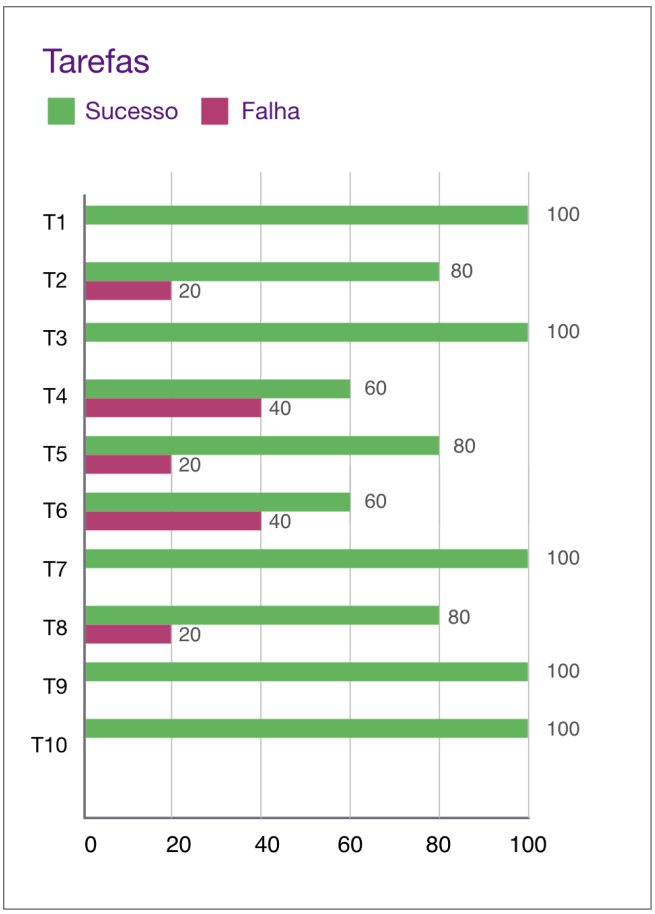

Figura 8. Taxa percentual de acertos e erros por tarefa. Número de participantes = 5. Imagem gerada pelos autores.

Para resolver este problema, sugerimos que seria necessário deixar obrigatório a inicialização com o *Matchmaker* na cena de menu após a confirmação do protocolo *MKMenuScene*.

B. A instanciação da cena de jogo deve ser mais intuitiva

Esta conclusão é relacionada à tarefa T6 (instanciar a cena de jogo quando a partida for encontrada) que obteve 30% de erros. Para apresentar a cena de jogo, é necessário chamar um método da *MKMenuScene* que, por se tratar de um método de um protocolo, fica "oculto" (ou seja, o desenvolvedor consegue visualizar a declaração deste método apenas quando olha a implementação do protocolo).

Como este problema está relacionado com uma restrição de código (métodos de protocolo ficam "ocultos"), o que sugerimos ser feito é deixar este detalhe mais evidente na documentação.

C. A documentação deve ser mais clara

Pôde-se perceber que, majoritariamente, foram encontradas dificuldades nas tarefas necessárias para a configuração inicial do MultiplayerKit, com erros em T4 (reportado na Seção VI-A), T6 (reportado na Seção VI-B), e também em T2, T5 e T8 (ver Figura 8). No entanto, nas tarefas consideradas principais, que são as de criar mensagens customizadas (T7) e enviar e receber mensagens (T10), todos os participantes obtiveram sucesso.

Apesar de todos os usuários terem obtido sucesso em T1 (ler e compreender a documentação), conjecturamos que as tarefas de configuração inicial poderiam ter sido realizadas com maior sucesso, caso a documentação fosse mais clara. Esta impressão é corroborada pelo fato de que, durante a entrevista, foi mencionado que a documentação dificultou um pouco o entendimento da ferramenta e foi sugerido que ela poderia ser mais simples e conter exemplos mais claros.

Desta forma, indicamos que, uma revisão completa da documentação, com a inserção de exemplos mais claros, pode melhorar a usabilidade do MultiplayerKit.

D. O jogo protótipo de nave utilizado no teste poderia ser mais simples

Apesar do jogo escolhido como protótipo ter uma proposta aparentemente simples - e mesmo havendo explicação prévia da organização do código pelo moderador - alguns usuários tiveram que rever o código do jogo mais vezes para compreender melhor sua estrutura, ao realizar determinadas atividades com o MultiplayerKit.

Desta forma, sugerimos criar um outro jogo de demonstração, mais simples que o atual. Estes usuários

podem buscar a compreensão inicial do *framework* a partir deste jogo, e posteriormente, avançar para o jogo de nave.

VII. Conclusão e trabalhos futuros

Este trabalho apresentou o MultiplayerKit, um *framework* que auxilia o desenvolvimento de jogos *multiplayer* para dispositivos iOS. O *framework* objetiva ser uma solução de fácil entendimento e que possa tornar o desenvolvimento mais eficaz, impactando principalmente desenvolvedores iniciantes ou equipes pequenas.

Foram detalhadas a arquitetura proposta e as principais funcionalidades da ferramenta. A avaliação realizada indicou ajustes que devem ser realizados em futuras atualizações do *framework*. A impressão - que necessita, obviamente, ser confirmada por estudos mais robustos - e de que, apesar de pequenos erros, a ferramenta cumpre com o objetivo de tornar o processo de desenvolvimento mais prático, reduzindo a verbosidade do código, tornando o processo mais compreensível e de fácil implementação.

Para trabalhos futuros, pretende-se inicialmente implementar todas as mudanças sugeridas pela avaliação (melhorar a instanciação das cenas de menu e de jogo, e tornar a documentação mais clara, fornecendo, inclusive, um exemplo mais simplificado). Desta forma, espera-se obter uma versão mais robusta do MultiplayerKit.

Além disso, entendemos que o MultiplayerKit oferece atualmente apenas as ferramentas essenciais para o desenvolvimento de jogos *multiplayer*. Para a próxima versão da ferramenta, planeja-se, por exemplo, incluir algoritmos para tolerância de falhas (como, por exemplo, medir a latência entre os dispositivos e adicionar predição de movimento em caso de perda de dados) e adicionar a possibilidade de desafios compartilhados para os jogadores.

Referências

[1] C. E. Crooks II. Awesome 3D Game Development. [S.l.]: Charles River Media, Hingham, Massachusetts, 2004. v. 1.

[2] M. Fayad, D. Schmidt. Object-oriented application frameworks. Communications of the ACM, p. 32–33, 1997.

[3] PHOTON. Photon. 2003. Disponível em: https://www.photonengine.com/. Acesso em: 07 mai. 2019.

[4] UNET. Unet. 2005. Disponível em: https://docs.unity3d.com/Manual/UNet.html. Acesso em: 07 mai. 2019.

[5] STEAMWORKS. Steamworks. 2008. Disponível em: https://partner.steamgames.com/doc/sdk. Acesso em: 07 mai. 2019.

[6] R. E. Johnson. *Frameworks = (components + patterns)*. Communications of the ACM, p. 4, 1997.

[7] J. Kirriemuir, A. McFarlane. *Literature review in games and learning*. A NESTA Futurelab Research report, p. 6, 2004.

[8] K. Salen, E. Zimmerman. *Rules of Play: Game Design Fundamentals*. [S.l.]: MIT Press, Cambridge, Massachusetts, 2003. v. 1.

[9] J. Glazer, S. Madhav. *Multiplayer Game Programming Architecting Networked Games*. [S.l.]: Addison-Wesley Professional, Boston, 2015. v. 1. 1-14 p.

[10] J. Kurose, K. Ross. *Redes de computadores e a internet uma abordagem top-down*. [S.l.]: Pearson Education, Londres, 2013. v. 6. 62-64 p.

[11] T. Barron. *Multiplayer Game Programming*. [S.l.]: Prima Tech, Califórnia, 2001. v. 6. 52-58 p.

[12] R. van der Meulen, T. MacCall. *Gartner Says Worldwide Sales of Smartphones Recorded First Ever Decline During the Fourth Quarter of 2017*. 2018. Disponível em: http://twixar.me/cDqK. Acesso em: 07 mai. 2019.

[13] M. Rouse. Apple Swift. 2018. Disponível em: https://whatis.techtarget.com/definition/Apple-Swift. Acesso em: 07 mai. 2019.

[14] R. E. Johnson. *How to design frameworks. Object-oriented Programming Systems*. Languages and Applications Conference - OOPSLA, 1993.

[15] Swift.org. Protocols. 2015. Disponível em: https://docs.swift.org/swift-book/LanguageGuide/Protocols.html. Acesso em: 18 nov. 2019.

[16] J. Nielsen, R. L. Mack. *Usability Inspection Methods*. John Wiley & Sons, p. 5, 1994.

[17] ISO 9241-11. Ergonomic requirements for office work with visual display terminals (vdts). *The International Organization for Standardization*, 45, 1998.

[18] J. Lazar, J. Feng, H. Hochheiser. *Research Methods in Human-Computer Interaction*. Wiley Publishing, 2017. v. 2.

[19] R. A. Virzi. *Refining the Test Phase of Usability Evaluation: How Many Subjects Is Enough?*. GTE Laboratories Inc., Waltham, Massachusetts, 1992.

[20] G. Lindgaard, J. Chattratichart. *Usability testing: what have we overlooked?*. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, San Jose, California, 2007.

[21] J. S. Dumas, J. C. Redish. *A practical guide to usability testing*, Intellect books, 1999.

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S121m Sabino, João Paulo de Oliveira.
MultiplayerKit : um Framework para o Desenvolvimento de Jogos Multiplayer para Dispositivos iOS / João Paulo de Oliveira Sabino. – 2019.
8 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Instituto UFC Virtual, Curso de Sistemas e Mídias Digitais, Fortaleza, 2019.
Orientação: Prof. Dr. Alysson Diniz dos Santos.

1. Jogo. 2. Multijogador. 3. Framework. 4. iOS. I. Título.

CDD 302.23
