Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Uma investigação sobre a percepção do jogador em relação a diferentes mecanismos de entrada em um jogo para reabilitação controlado por movimento

Luísa Bontempo Nunes

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Orientadora
Profa. Dra. Carla Denise Castanho

Brasília
2016

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Uma investigação sobre a percepção do jogador em relação a diferentes mecanismos de entrada em um jogo para reabilitação controlado por movimento

Luísa Bontempo Nunes

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Profa. Dra. Carla Denise Castanho (Orientadora)
CIC/UnB

Prof. Dr. Edson Alves da Costa Júnior
FGA/UnB

Prof. Dr. Tiago Barros Pontes e Silva
DiN/UnB

Prof. Dr. Ricardo Pezzuol Jacobi
Coordenador do Curso de Engenharia da Computação

Brasília, 12 de Agosto de 2016

Dedicatória

Dedico este trabalho a todos aqueles que, ainda que em face da adversidade, continuam caminhando em direção aos seus sonhos.

Agradecimentos

Primeiro, agradeço à professora Carla Castanho, a qual conheci sendo aluna em uma de suas disciplinas e que desde então tem sido não só uma professora e minha orientadora, mas uma grande amiga com quem pude contar. Se não tivéssemos nos conhecido, tenho certeza que minha vida teria seguido um curso muito diferente, não só no que tange a este trabalho.

Agradeço, também, a todos os voluntários que nos ajudaram a validar nossa proposta. Obrigada por comparecerem à UnB em pleno julho, e cederem um pouquinho do seu tempo para nos ajudar.

O jogo que foi objeto deste trabalho nasceu como parte de um projeto maior, que agregava conhecimentos de várias áreas. Seria impossível para mim, uma única aluna de um único curso, ter chegado aqui sem a ajuda de muitos outros. Agradeço ao Marcos Akira, da Psicologia, pela grande ajuda na preparação e realização dos testes - me mostrou o quanto este tipo de experimento não é simples; ao Lucas Fonseca, do Departamento de Engenharia Elétrica, por ter cedido os sensores IMU e nos orientado em relação ao seu uso; ao Luciano Santos e ao Fabrício Buzeto, por terem construído a infraestrutura de software na qual este trabalho foi desenvolvido; e ao Pedro Cataldi, por ter dado vida ao jogo com sua arte. A todos vocês, já mestres e doutores, obrigada por aceitarem uma aluna de graduação em seu projeto.

Mas, certamente, a vida não é só a acadêmica. Cada um de nós encontra pessoas que nos motivam a continuar. Agradeço, portanto, aos meus amigos de dentro e de fora da universidade, que são muitos para listar, mas que sabem quem são, e sabem que, sem o seu apoio, eu não estaria de pé; agradeço aos meus pais, que tiveram fé nas minhas capacidades e me sustentaram até aqui; e agradeço aos alunos e monitores de Introdução ao Desenvolvimento de Jogos, disciplina da qual fui monitora quatro vezes, e que foi uma constante fonte de alegria para mim.

Quando olhamos para trás, e pensamos na graduação, é fácil se perder entre aulas e trabalhos e esquecer que uma universidade não se faz somente de alunos e professores. Deixo aqui o meu reconhecimento ao trabalho dos funcionários dos departamentos, secretarias e postos avançados da Secretaria de Administração Acadêmica, que estão aqui

todos os semestres nos ajudando a lidar com a burocracia, e que me tiraram de situações complicadas mais de uma vez.

Por último, mas não menos importante, gostaria de agradecer aos professores Edson da Costa e Tiago Barros, que aceitaram o convite para participar da banca de avaliação deste trabalho. Obrigada por compartilharem sua experiência e seus conhecimentos comigo.

Resumo

Neste trabalho, desenvolveu-se um jogo denominado *Fisiogame*. Ele foi concebido no âmbito da terapia de reabilitação e pode receber entrada de mais de um dispositivo de captura de movimento. Para testar se diferentes métodos de entrada causariam mudanças na experiência do jogador, o jogo foi testado num experimento com participantes voluntários em que um grupo usou um sensor IMU para executar os movimentos e outro, um *smartphone*. Após jogarem o *Fisiogame*, os participantes responderam a uma entrevista baseada em conceitos da Teoria do *Flow* e a análise das respostas dadas mostrou que houve pouca ou nenhuma diferença na experiência dos jogadores entre os dois grupos, indicando que a substituição do mecanismo de entrada é viável.

Palavras-chave: input de jogos. jogos ubíquos. *flow*. terapia de reabilitação

Abstract

For this research, a game called *Fisiogame* was developed. It is was conceived for use in rehabilitation therapy, and is special in that it can use different types of input devices. To assess the possibility of a change in the input method impacting the player's experience, the game was put to test in an experiment with voluntary participants in which one group used an IMU sensor to capture their own motions and another group used a smartphone. After playing *Fisiogame*, each participant then answered an interview whose questions were based on Flow Theory concepts, and an analysis of the responses given revealed little to no difference between the experience of players in both groups, indicating that the substitution of the input method is viable.

Keywords: game input. ubiquitous games. flow. rehabilitation therapy

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

CIC Departamento de Ciência da Computação.

DTW Dynamic Time Warping.

ENE Departamento de Engenharia Elétrica.

GDD Game Design Document.

IMU Inertial Measurement Unit.

JSON JavaScript Object Notation.

SAA Secretaria de Administração Acadêmica.

UnB Universidade de Brasília.

Capítulo 1

Introdução

A indústria de jogos eletrônicos existe há pouco mais de quarenta anos e é a maior dentre as indústrias de entretenimento do mundo [29]. Essa imensa popularidade se deve tanto aos esforços dos próprios desenvolvedores de jogos em constantemente tentar inovar e melhorar seus produtos, mas também à vontade insaciável de fãs em todo o mundo de jogá-los.

O ato de jogar não é recente, tendo estado presente no decorrer do desenvolvimento das várias civilizações humanas. São conhecidos jogos com centenas ou até milhares de anos [24], e há estudos sobre o que torna essas atividades tão imersivas, como, por exemplo, a Teoria do *Flow* [10].

Conhecendo-se o poder dos jogos de atrair a atenção das pessoas, passou-se a considerar a sua utilização para auxiliá-las a atingir não objetivos lúdicos, mas sim tarefas que exigissem foco e dedicação para serem completadas. Dentre as áreas que hoje fazem uso de jogos está a terapia de reabilitação, que trabalha para restaurar as habilidades de pessoas que sofreram lesões físicas e precisa manter o paciente motivado para que a terapia proceda [18]. Jogos desenvolvidos para esse fim usam o próprio movimento do paciente como entrada para o jogo.

A questão de qual dispositivo de entrada deve ser utilizado, no entanto, não tem uma resposta definida. Diferentes tratamentos e situações podem requerer mecanismos distintos, por exemplo, um sensor pode estar disponível em uma clínica, mas o seu custo pode ser proibitivo para um paciente tê-lo em casa. Ou então, um dispositivo pode ser apropriado para observar pacientes com lesões em certas partes do corpo, mas não funcionar bem para a observação das demais.

Essa necessidade de adaptabilidade remete à Computação Ubíqua, ramo da computação que busca tornar a tecnologia onipresente e integrada, cada dispositivo utilizando informações do contexto em que está inserido de forma a se adaptar e facilitar a realização de tarefas por seres humanos [32]. Isso resultou na pulverização de dispositivos no

ambiente, trazendo à tona novas formas de interação com a tecnologia. Surgiram, então, jogos que faziam uso dessas inovações, os chamados jogos ubíquos.

É possível desenvolver um jogo ubíquo para reabilitação que se adapte ao dispositivo de captura de movimentos disponível no momento em que é jogado. No entanto, deve-se atentar para o fato de que dispositivos de entrada distintos podem proporcionar diferentes experiências. Especificamente, se a experiência do jogador for afetada negativamente, a sua recuperação pode ser prejudicada. É de vital importância garantir que isso não aconteça.

Para tanto, o objetivo deste trabalho é o desenvolvimento de um jogo para reabilitação que utiliza os dados de *Inertial Measurement Units*[1], ou *IMU*s, como entrada. Tendo em vista que estes são equipamentos especializados, foi analisada a sua substituição por um dispositivo que faz parte do cotidiano das pessoas, como, por exemplo, um *smartphone*.

Foram realizados testes em que um grupo de usuários utilizou sensores IMU, e o outro, um *smartphone*, como métodos de entrada para o jogo. Os participantes jogavam por um tempo limitado, após o qual respondiam a uma entrevista semi-estruturada cujo objetivo era avaliar características de imersão, controle, concentração, dentre outras.

O restante desta monografia está organizada da seguinte forma: O Capítulo 2 apresenta alguns dos conceitos fundamentais envolvidos neste trabalho. O *Fisiogame*, jogo desenvolvido no âmbito deste projeto, está descrito conceitualmente no Capítulo 3 e arquiteturalmente no Capítulo 4. O Capítulo 5 descreve o roteiro dos testes realizados e apresenta os resultados obtidos. Finalmente, o Capítulo 6 traz a conclusão e algumas possibilidades de trabalhos futuros.

[1]Sensor composto por uma combinação de giroscópios, acelerômetros e magnetometros que pode medir sua rotação absoluta em *quaternions*, que, por sua vez, são elementos matemáticos usados para representar uma rotação ou orientação num espaço tridimensional.

Capítulo 2

Jogos Eletrônicos

Neste capítulo, é feita uma breve apresentação de conceitos envolvendo Jogos Eletrônicos, dentre eles, os Jogos para Reabilitação e os Jogos Ubíquos.

2.1 Breve histórico

O hábito de jogar é muito antigo nas civilizações humanas, havendo registros de jogos que existiam há mais de três mil anos [24] e de uma miríade de jogos que surgiram nos milênios que seguem, dentre eles jogos de cartas, tabuleiro e até de interpretação (*role play*). Dada a persistência deste costume, era natural esperar que o avanço da tecnologia levasse ao advento de novos tipos de jogos.

Primeiro, apareceram os jogos baseados em dispositivos mecânicos. Um dos maiores exemplos foram as máquinas de *pinball*, que se popularizaram no começo do século XX. Elas consistiam, basicamente, de gabinetes de madeira dentro dos quais havia uma rampa repleta de pinos, cada um com uma pontuação atribuída, onde os jogadores lançavam uma bola e contavam seus pontos. As primeiras versões do jogo tinham pouco a ver com as atuais, que incorporaram novos elementos não apenas para se tornarem mais interessantes para os jogadores, como os componentes elétricos que respondiam a eventos do jogo, mas também para se legitimarem como uma atividade que requeria habilidade e não apenas sorte. Esta foi uma discussão essencial, visto que as máquinas chegaram a ser proibidas em alguns lugares por serem entendidas como jogos de azar. A luta por legitimidade já existia na indústria do entretenimento, e é possível que os *video games*, que viriam a surgir, tenham se beneficiado destes avanços que o *pinball* conquistou [19].

Os jogos eletrônicos propriamente ditos começaram a surgir apenas entre o fim da década de 50 e o começo da de 60. Dois jogos desenvolvidos nesta época são amplamente

considerados os primeiros jogos eletrônicos[1] da história: *Spacewar*, criado em 1962 por Steve Russell e aperfeiçoado por seus colegas, era um duelo entre espaçonaves que se moviam num plano 2D e atiravam torpedos uma contra a outra [19]; *Tennis for Two*, desenvolvido em 1958 por William Higinbotham e apresentado na mostra anual de tecnologias do *Brookhaven National Laboratory*, trazia uma visão simplificada de uma quadra de tênis em um osciloscópio e permitia a um par de jogadores rebater a bola de um lado para o outro da quadra, regulando a força da jogada [1].

O sucesso destes jogos demonstrou que havia interesse do público. Em 1972, dez anos após a criação da versão original do *Spacewar*, Ralph Baer, em parceria com a Magnavox, lançou o Magnavox Odyssey, o primeiro console de *video games* [19]. No mesmo ano, a Atari, companhia de Nolan Bushnell, um grande fã de *Spacewar*, viria a criar máquinas de *arcade* com o jogo Pong, que possibilitaria à companhia investir em novas criações, inclusive em consoles próprios [19]. Paralelamente ao desenvolvimento destes aparelhos, empresas como Apple, Commodore e IBM faziam fortes investimentos para tornar o computador pessoal cada vez mais popular [16]. Seus produtos eram computadores de propósito geral e, portanto, permitiam que jogos fossem desenvolvidos e vendidos, aplicando um duro golpe na indústria de consoles que, em meados de 1983, estava saturada e criativamente esgotada, e só viria a se recuperar com a chegada do Famicom, ou Nintendo Entertainment System, fabricado pela Nintendo [19].

Todos estes acontecimentos contribuíram para a fundação do que hoje é uma indústria multibilionária, um mercado cujo tamanho supera até mesmo o da indústria cinematográfica [29]. Nos dias atuais, 40% dos americanos possuem um console de videogame, 73% têm *PCs*, 68%, *smartphones*, e 45%, tablets [25]. Com isso, 155 milhões de pessoas naquele país, de alguma forma, consomem jogos eletrônicos [2], indicando que o costume de milênios atrás se mantêm forte, ainda que transformado pela tecnologia. Pergunta-se, então, por que esse hábito existe?

2.2 Por que as pessoas jogam?

A palavra *jogo* tem sua raiz no latim *jocus*, que significa brincadeira, diversão. Uma origem semelhante pode ser encontrada em *game*: *gamen*, do germânico, que significa alegria, diversão. Podemos também examinar *play*, do anglo-saxão *plæga*, que, como a palavra moderna, tem vários sentidos, dentre eles, recreação. Estas mesmas raízes estão

[1] É interessante notar que a definição de jogo eletrônico utilizada por grande parte dos autores ao falar em primeiros jogos eletrônicos costuma levar em conta três fatores: o jogo deve conter processamento feito por um computador; deve ser mostrado numa tela, como os jogos atuais; e, talvez o fator mais considerado, deve ter sido desenvolvido não puramente por razões acadêmicas, mas com o propósito de entreter uma audiência. Há registro de outros jogos mais antigos, mas que não atendem a tais requisitos.

presentes em diferentes termos de idiomas atuais, e se assumirmos que não se trata de mera coincidência, temos uma resposta simples para a nossa pergunta – Jogos são percebidos, em diversas culturas, como algo prazeroso, e as pessoas jogam porque é divertido.

Esta é uma resposta válida, porém, insuficiente do ponto de vista da psicologia, já que "diversão" é um conceito abstrato, difícil de explicar ou medir. Faz-se necessário, portanto, estudar mais a fundo o que é se divertir e por que isto acontece.

Uma destas tentativas de explicar a diversão é a Teoria do *Flow*. Esta teoria é baseada em observações de seu autor, Mihaly Csikszentmihalyi, sobre pessoas que dedicavam muito tempo a realizar tarefas difíceis e com pouca recompensa externa, como músicos ou jogadores de xadrez. Além disso, foram conduzidas observações sobre tarefas que traziam satisfação a pessoas comuns durante a vida cotidiana. Ele observou que, por mais diferentes que fossem as tarefas, elas ainda eram descritas de forma similar no que diz respeito a o que nelas trazia satisfação. Concluiu-se que o prazer obtido a partir destas experiências é, de certa forma, universal [10].

Quando pessoas se encontram totalmente concentradas em uma tarefa, não pela recompensa, mas pelo simples prazer de fazê-la, dá-se o nome de "*flow*" ao seu estado mental [12]. Em [11], pode-se encontrar oito características principais de experiências que levam ao *flow*:

1. A tarefa deve ter objetivos claros;
2. A tarefa deve ter algum desafio, e o nível deve ser compatível com as habilidades pessoais;
3. O ato de fazer a tarefa deve se tornar espontâneo, perdendo-se a consciência de estar executando-a;
4. A tarefa permite concentração a ponto de se esquecer as frustrações usuais do dia a dia;
5. Há uma clara sensação de estar no controle do que acontece;
6. A tarefa causa uma suspensão da noção do eu e do mundo que se habita;
7. O tempo parece passar de forma diferente do normal;
8. A experiência é autotélica, isto é, a execução da tarefa tem fim em si mesma e é intrinsecamente recompensante.

A combinação destes elementos traz uma sensação extremamente prazerosa, suficiente para fazer com que as pessoas dediquem grande quantidade de tempo e esforço a estas atividades [30].

Nota-se que, em nenhum momento a teoria fala especificamente de jogos. Ela nasceu de observações de atividades diferentes e tenta explicar a diversão de uma forma mais abrangente. No entanto, a presença de tais características entre jogadores de videogame fez com que estudos fossem conduzidos relacionando o *flow* aos jogos, como, por exemplo, o trabalho de Jenova Chen, que apresentava um modelo de ajuste dinâmico de dificuldade para manter o *flow* [9]. A Figura 2.1, retirada desse trabalho, apresenta o *flow* como o estado entre a ansiedade e o tédio, atingido quando a dificuldade do jogo e as habilidades pessoais se equilibram.

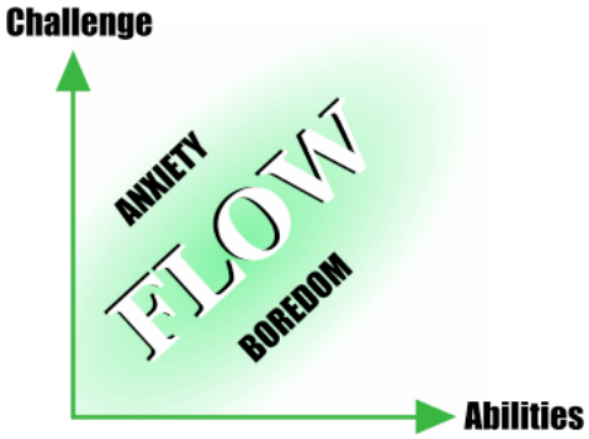

Figura 2.1: Gráfico explicativo do *flow* proposto por Jenova Chen. (Fonte: [9]).

Tais estudos abriram caminho para uma avaliação mais objetiva de questões de imersão, concentração, diversão, dentre outras, em jogos eletrônicos. Por consequência, surgiram modelos de avaliação de jogos eletrônicos baseados no *flow*, como, por exemplo, o *GameFlow*, descrito em [30].

2.3 Jogos na Terapia de Reabilitação

A teoria do *flow* nos fornece uma lista de características de tarefas divertidas, e podemos usá-la para encontrar elementos de *game design* que ajudam o jogador a alcançar o *flow*. Seria possível, então, incorporar estes elementos em outras tarefas, consideradas maçantes ou indesejáveis, para torná-las mais atraentes para as pessoas que as realizam?

Esta não é uma ideia recente. É possível encontrar exemplos bem sucedidos de uso de jogos na educação [23] e no treinamento de funcionários [3], vinculados ou não à teoria do *flow*. Em ambas as áreas, a dedicação do usuário é essencial para que os objetivos sejam atingidos.

A área de reabilitação fisioterápica é similar. Ela trata de restaurar as habilidades de pessoas que, em virtude de algum distúrbio ou lesão, perderam certas capacidades

motoras, para que estas pessoas possam retomar suas rotinas. É um processo longo e trabalhoso, durante o qual inúmeras sessões de exercícios são realizados, e a motivação do paciente é de grande importância para a sua recuperação [18].

De fato, já existem trabalhos bem sucedidos que aplicaram jogos em contextos de reabilitação. Um destes projetos [8] utilizou uma aplicação baseada no Kinect[2] para acompanhar pacientes de fisioterapia e observou neles um aumento da motivação e, consequentemente, da performance. Outros [15, 22] incluíram o Nintendo Wii[3] e o jogo Wii Fit[4] [17] (Figura 2.2) no tratamento de pacientes, e observaram melhores resultados nos pacientes que usaram o console em comparação aos que não usaram. Tais evidências de que o uso de jogos na fisioterapia pode ser benéfico propiciam a criação de jogos específicos para tal fim, como no caso do jogo objeto deste trabalho.

Figura 2.2: Imagem promocional do Wii Fit. O jogador deve pisar num acessório chamado *Balance Board* e realizar diferentes rotinas de treinamento. Fonte: `http://wiifitu.nintendo.com/`.

[2] `http://www.xbox.com/en-US/xbox-360/accessories/kinect`
[3] `http://wii.com`
[4] `http://wiifit.com`

2.4 Jogos Ubíquos

> As tecnologias mais profundas são aquelas que desaparecem. Elas se integram à vida cotidiana até se tornarem indistinguíveis dela.[5]

Assim disse Mark Weiser, ao definir a Computação Ubíqua, em 1991 [32]. O termo "ubíquo" vem do latim "*ubique*", que significa "em todos os lugares". Um dos maiores exemplos de tecnologia ubíqua é a escrita: Estamos constantemente absorvendo informação por meio da leitura, não só de livros ou revistas, que são objetos criados para armazenar grandes quantidades de informação, mas também de placas, embalagens, *outdoors*, entre outros. O desenvolvimento de sistemas de escrita, isto é, a codificação da informação em glifos padronizados, é um processo complexo que durou milhares de anos [13]. No entanto, nos dias de hoje, o ato de ler é tão natural que não atentamos a isso.

A *ubicomp* busca integrar a computação à vida cotidiana de forma similar ao que aconteceu com a escrita. Na visão de Weiser, a tecnologia seria "calma", isto é, estaria integrada ao ambiente, ampliando suas funcionalidades e ajudando os seres humanos em suas tarefas, mas sem chamar a atenção para si ou para detalhes técnicos.

Essa pulverização de dispositivos no ambiente trouxe à tona novos conceitos, novas tecnologias e, principalmente, novas formas de se interagir com o ambiente computacional. Neste contexto, surgem os jogos ubíquos, jogos os quais tentam aplicar essas inovações para criar mecânicas diferentes, com o objetivo de aumentar a imersão do jogador [28].

Por se tratar de uma área ainda muito nova e em constante evolução, a definição exata de o que torna um jogo ubíquo é uma questão em aberto, com autores apresentando diferentes ideias de o que os caracterizam. Em [7], é possível encontrar uma breve e não-exaustiva lista de definições:

- Jogos de Computação Ubíqua [4]: Esta é uma das mais antigas definições. Trata de jogos em que participantes tanto no mundo real quanto *online* podem compartilhar de uma mesma experiência. Esta visão dos jogos ubíquos foca na interação dos jogadores com o mundo, especialmente na diferença daqueles que estão no mundo real, podendo interagir diretamente com o mundo do jogo, e os que estão no mundo virtual e interagem apenas por uma determinada interface.
- Jogos de trans-realidade [14]: Estes jogos apresentam mundos paralelos ao mundo real. O jogador pode alternar entre os mundos e interagir com eles. Aqui, já observa-se uma diferença desta visão para a anterior, ou seja, o jogo ubíquo não se passa mais, necessariamente, no mundo real.

[5]Tradução livre. Texto original: "The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.

- Jogos sensíveis ao contexto [21]: São jogos que fazem uso da ideia de sensibilidade e adaptação ao contexto previamente apresentada nesta Seção. Conhecendo o ambiente em que estão inseridos, estes jogos podem se modificar de acordo com certas condições, que vão desde o hardware ou software presente no ambiente à hora e ao clima do lugar onde o jogo está sendo jogado.
- Jogos pervasivos ou Jogos baseados em localização [31]: Jogos que se aproveitam da mobilidade do jogador e usam a sua geolocalização como parte do jogo.
- Jogos de realidade misturada [5]: São aqueles que usam o mundo real como cenário para uma experiência virtual, como os jogos de realidade aumentada.

É fácil encontrar exemplos de jogos nestas categorias, e nota-se também que um jogo pode estar em mais de uma delas. É o caso, por exemplo, de Pokémon GO[6], jogo lançado recentemente com grande sucesso, que é tanto pervasivo quanto de realidade misturada. Ele usa o GPS do celular para se localizar no globo e popula áreas da cidade com Pokémons, monstros que o jogador pode capturar e colecionar. O jogador vê o mundo através da tela do celular, onde imagens da câmera são sobrepostas com Pokémons.

Pokémon GO é baseado no Ingress[7], jogo em que jogadores, também munidos de *smartphone*, se dividem em equipes e percorrem o mundo real em busca de portais posicionados em pontos de referência das cidades. Controlar estes locais requer esforço conjunto dos jogadores e recompensa sua equipe com recursos para progredir no jogo.

Outra iniciativa interessante na área de jogos ubíquos está no Xbox Smartglass[8], que permite a integração do console Xbox One com dispositivos móveis. Os jogos podem fazer uso dessa tecnologia como desejarem; Dead Rising 3[9], por exemplo, oferece informações adicionais ao jogador na tela do seu *smartphone*, como mapas, dicas, acontecimentos na cidade, dentre outros. Além disso, quando o protagonista atende o próprio celular dentro do jogo, a ligação pode ser ouvida no dispositivo no mundo real.

O jogo Space Team[10] também faz uso de dispositivos numa mesma rede, mas isso acontece de forma diferente. O jogo suporta de dois a quatro jogadores conectados localmente, os quais entram no papel de integrantes da tripulação de uma nave espacial. Cada um tem à sua disposição um painel de controle completamente diferente dos outros. Em cada fase, mensagens são mostradas na tela de cada jogador, orientando-os a realizar ações na nave, porém, frequentemente, o jogador que recebe a mensagem não tem o controle necessário no seu painel. Ele precisa comunicar o comando à "equipe" para que

[6] `http://www.pokemon.com/us/pokemon-video-games/pokemon-go/`
[7] `https://www.ingress.com/`
[8] `http://www.xbox.com/en-US/smartglass`
[9] `http://www.xbox.com/en-US/xbox-one/games/dead-rising-3`
[10] `http://www.sleepingbeastgames.com/spaceteam/`

outro jogador o faça, permanecendo atento a comandos de outros jogadores que ele possa realizar.

Esta adaptabilidade aos dispositivos na rede, em particular, é de interesse para este trabalho. Sendo um jogo ubíquo capaz de se adaptar aos dispositivos conectados, é possível criar um jogo para reabilitação que se adapte aos mecanismos de captura de movimentos disponíveis, podendo receber entrada de qualquer um deles, independente da sua natureza. O *framework UnBiHealth* [27], utilizado na concepção do *Fisiogame*, tem este propósito. No Capítulo 4, o uso desse *framework* é discutido com mais detalhes.

Capítulo 3

Design do Jogo *Fisiogame*

Neste capítulo, são apresentadas algumas das considerações iniciais envolvidas na concepção do *Fisiogame*. Em seguida, o jogo é descrito conceitualmente, baseando-se no modelo de *Game Design Document* proposto em [26].

3.1 Considerações Iniciais

A ideia do jogo surgiu em um projeto envolvendo pacientes que sofreram de acidente vascular cerebral e, com isso, perderam parcial ou totalmente a mobilidade de um de seus braços. Observa-se que, para este grupo de pacientes em especial, os nervos responsáveis por comandar o membro não sofreram dano, apenas perdeu-se a capacidade de fazer tais comandos. Portanto, é possível haver melhora no quadro utilizando estimulação elétrica, como mostrado em estudos tais quais [33, 20].

Neste tipo de terapia, o paciente deve realizar o movimento com o auxílio da estimulação elétrica e, depois, sem ela. Daí vem a necessidade de motivação, já que é necessário que haja concentração no movimento sendo feito para que o controle do membro seja reaprendido.

Além disso, como os pacientes não necessariamente iniciam o tratamento na mesma época, elementos *multiplayer* foram eliminados do jogo, visto que acreditou-se que eles não seriam úteis. Da mesma forma, foi estabelecido que, dada a variedade de exercícios possíveis, o movimento do personagem no jogo não corresponderia ao realizado pelo jogador no mundo real, assim, possibilitando que diversos exercícios fossem realizados num mesmo contexto do jogo.

Dada a origem do jogo na terapia de pacientes com lesão nos membros inferiores, o jogo foi projetado e testado para utilização em tratamentos envolvendo estes membros. No entanto, dada a possibilidade de gravar novos movimentos para serem usados como

referência de exercício, não existem impedimentos a usar sensores nos membros inferiores e realizar exercícios desta forma.

3.2 Conceito

Trata-se de um jogo do gênero Gerenciamento, no qual os jogadores tem como objetivo obter e aplicar recursos em determinadas tarefas para progredir. Neste caso, o jogador é colocado na posição de gerente de projetos de uma pequena vila onde há personagens, alguns dos quais só aparecem após um determinado ponto na história.

O jogo foi projetado para acompanhar um paciente numa sessão de fisioterapia e recebe, como entrada, os movimentos dos exercícios realizados. Durante o *gameplay*, o jogador recebe uma série de **missões**, que consistem em pedidos de **moradores** para que um determinados **prédios** sejam construídos na vila. Para que as construções ocorram, é necessário que o jogador tenha determinados **recursos** armazenados, que são obtidos por meio dos ***minigames***.

Completar uma missão desbloqueia novas missões e novos moradores. Cada uma delas é apresentada e encerrada por uma cena em que os personagens conversam entre si e com o jogador. O prédio referente a cada missão completada aparece numa tela de mapa e tem o seu efeito ativado.

3.3 Escopo

3.3.1 Plataformas

O jogo foi projetado para computadores pessoais e é compatível com os sistemas operacionais Windows, Linux e OS X.

3.3.2 Público Alvo

O jogo é destinado a pacientes de fisioterapia, é livre para todas as idades e requer apenas habilidades de leitura.

3.3.3 Jogadores

Um jogador joga por vez, possivelmente acompanhado de seu fisioterapeuta. O progresso do jogador é gravado associado ao seu nome de usuário, podendo ter continuidade em outra sessão.

3.3.4 Gênero

O jogo é do gênero Gerenciamento, como definido na Seção 3.2. Como exemplos deste gênero, podemos citar The Simpsons: Tapped Out[1] (Figura 3.1) e Paradise Island[2].

Figura 3.1: Tela de The Simpsons: Tapped Out. Fonte: Google Play Store.

3.3.5 Objetivo

O objetivo do jogador é completar todas as missões que existirem. É importante notar, no entanto, que como o jogo deve durar, pelo menos, o tempo do tratamento fisioterápico, este objetivo é balanceado para ser muito difícil de alcançar, permitindo assim que o jogo possa ser útil durante todo o processo de reabilitação.

3.4 Mecânica

O jogo se divide em duas partes: o gerenciamento da cidade e os *minigames*.

3.4.1 Gerenciamento da Cidade

A interação do jogador nas telas da cidade ocorre exclusivamente com o *mouse*. Todas as vezes que o jogo é iniciado, é mostrado um diálogo ao jogador. Na primeira sessão, tal diálogo é longo e busca apresentar os personagens e explicar os elementos de interface. Da segunda sessão para frente, há poucas falas. A sessão sempre se inicia com a praça

[1] `http://www.ea.com/thesimpsonstappedout/`
[2] `http://www.game-insight.com/en/games/paradise-island`

da cidade sendo mostrada (ver Figura 3.2). Nesta tela, o jogador vê dois menus - um mostrando suas missões ativas (1), e outro, os *minigames* desbloqueados (2). Há também três botões: um para mostrar o mapa da cidade (3), um para mostrar o menu de opções (4) e um para mostrar a lista de recursos disponíveis (5).

Figura 3.2: Tela principal do jogo com elementos de interface destacados.

Recursos

O jogador deve estar atento aos seus recursos armazenados, pois cada missão exige uma determinada quantidade e variedade deles. Alguns recursos são mais raros que outros por dependerem de sorte para serem obtidos. O Quadro 3.1 lista todos os recursos obtíveis.

Missões

Todas as missões são decorrentes do pedido de um morador e se referem a um único prédio. Para serem completadas, é necessário que o jogador acumule quantidades determinadas de até três tipos diferentes de recursos e, então, confirme a intenção de construir o prédio em questão.

Para consultar os requisitos de uma missão, o jogador deve clicar no botão correspondente a ela no menu lateral esquerdo da tela principal, o que fará com que a tela da Figura 3.3 seja exibida. Nela, aparecem o nome do prédio que será construído (1), o progresso do jogador em conseguir todos os recursos para aquela missão (2), o morador que fez o pedido (3) com uma pequena fala (4) e um botão "Construir" (5), habilitado somente se todos os requisitos estiverem cumpridos.

Completar uma missão adiciona o prédio referente a ela à tela de mapa. Além disso, novas missões podem ser desbloqueadas tanto no início da próxima sessão, ou na atual ao

Quadro 3.1: Lista de recursos.

Ícone	Nome	Obtenção
	Madeira	Minigame de Madeira
	Pedra	Minigame de Pedra
	Metal	Minigame de Metal
	Comida	Horta, Minigame de Madeira
	Lã	Cercado de Ovelhas
	Tecido	Fábrica de Tecido
	Papel	Fábrica de Papel
	Carvão	Minigame de Pedra
	Tijolo	Olaria
	Mármore	Estúdio, Minigame de Pedra
	Ouro	Feira, Minigame de Forja

Figura 3.3: Tela de detalhamento de uma determinada missão.

final de um diálogo. A existência destas duas possibilidades confere maior controle sobre como um jogador pode progredir.

Prédios

Quando um prédio é construído, seu efeito se torna ativo. Há quatro tipos de efeitos:

- **Produção Passiva**: Uma quantidade de um determinado recurso é gerada a cada vez que um *minigame* é jogado, proporcional à quantidade de repetições do exercício.
- **Produção Ativa**: O jogador pode selecionar o prédio na tela de mapa e trocar quantidades de um recurso por outro.
- **Multiplicador Simples**: A presença do prédio melhora a eficiência da produção ou coleta de um certo recurso, aumentando sua geração em uma dada porcentagem.
- **Multiplicador Global**: O mesmo que o Multiplicador Simples, mas aplicado a todos os recursos de uma vez.

O Quadro 3.2 lista todos os prédios do jogo junto aos requisitos para construí-los, o efeito que eles trazem e, se houver, o recurso ao qual o efeito se aplica. Prédios já construídos aparecem na tela de mapa (Figura 3.4) acessível pela tela principal. O jogador pode consultar o nome e, no caso das que fornecem Produção Ativa, fazer a troca de recursos selecionando o prédio com um clique.

Figura 3.4: Tela de mapa com alguns prédios construídos. O número de casas na vila aumenta com o progresso do jogador.

3.4.2 *Minigames*

Jogar os *minigames* é a principal forma para o jogador obter recursos, e o exercício de fisioterapia está diretamente ligado a eles.

Figura 3.5: Lenhador aguardando a realização do exercício.

O *minigame* inicia-se com o personagem entrando na tela e caminhando até seu alvo, por exemplo, o lenhador andando até uma árvore. Ele prepara o seu machado e aguarda: quando o jogador realizar o exercício indicado, o seu movimento é avaliado e convertido em um nível de força para o golpe de lenhador. Finalmente, o lenhador golpeia a árvore com o machado, o que pode resultar na queda dela, dependendo da força.

Cada golpe do personagem, que, no mundo real, equivale a uma repetição do exercício, recompensa o jogador com o recurso correspondente àquele *minigame*. Além disso, há uma determinada probabilidade de recompensá-lo com outros recursos, também dependentes do *minigame*. A quantidade de recompensas aumenta na proporção em que o exercício realizado se aproxima da referência, motivando o jogador a realizá-lo com mais esforço e atenção.

Ao final do *minigame*, o total de recursos obtidos é mostrado ao jogador (Figura 3.6). Ele é, então, levado de volta à tela da praça. Neste momento, a produção referente aos prédios da cidade é calculada e adicionada às reservas do jogador.

Nota-se que não é possível perder no *minigame*. Jogadores com desempenho ruim receberão recompensas menores, mas progredirão no jogo assim como os outros jogadores. Um bom desempenho está relacionado à velocidade da progressão, mas uma performance ruim não impede que o jogador complete as missões. Essa foi uma decisão de *design* tomada para impedir que jogadores com lesões mais graves, que tivessem maior dificuldade em realizar os exercícios, não se sentissem desmotivados por repetidas derrotas. Falaremos mais da avaliação do desempenho do jogador na Seção 4.2.4.

Figura 3.6: Tela de recompensas de um *minigame*.

Quadro 3.2: Lista de Prédios com seus requerimentos e efeitos.

Nome	**Requisitos**	**Efeito**	**Recurso**
Casa	Madeira	Multiplicador Simples	Madeira
Oficina	Madeira	Multiplicador Simples	Madeira
Cercado de Ovelhas	Madeira	Produção Passiva	Lã
Horta	Madeira	Produção Passiva	Comida
Casa de Forja	Madeira, Pedra	Multiplicador Simples	Pedra
Prefeitura	Madeira, Pedra, Mármore	Multiplicador Global	–
Moinho	Madeira, Tecido	Multiplicador Simples	Comida
Feira	Madeira, Comida	Produção Passiva	Ouro
Estábulo	Madeira, Metal, Comida	Multiplicador Simples	Metal
Fábrica de Papel	Madeira, Metal	Produção Ativa	Madeira para Papel
Fábrica de Tecido	Madeira, Metal, Lã	Produção Ativa	Lã para Tecido
Olaria	Madeira, Metal, Carvão	Produção Passiva	Tijolo
Banco	Pedra, Metal, Papel	Multiplicador Simples	Ouro
Estúdio	Pedra, Tecido, Ouro	Produção Passiva	Mármore
Taverna	Pedra, Comida, Carvão	Multiplicador Global	–
Museu	Metal, Tijolo, Ouro	Multiplicador Global	–
Estátua do Fundador	Pedra, Mármore, Tijolo	Multiplicador Global	–
Biblioteca Municipal	Pedra, Mármore, Papel	Multiplicador Global	–

Capítulo 4

Projeto e Implementação

Neste capítulo, são discutidos os detalhes da implementação do jogo *Fisiogame*, que foi descrito em alto nível no capítulo anterior.

O jogo foi desenvolvido de forma a tornar possível receber entrada tanto de um *smartphone* quanto de um sensor *IMU*. Para atingir este objetivo, fez-se uso de algumas ferramentas, as quais serão discutidas na Seção 4.1 e são externas ao código do jogo, o qual é descrito da Seção 4.2 em diante.

4.1 Tecnologias Utilizadas

4.1.1 *UnBiHealth*

O *UnBiHealth* [27] é um *framework* baseado no *middleware*[1] *uOS* [7], ambos desenvolvidos por outros autores. O *uOS* foi criado com o intuito de facilitar a criação de jogos ubíquos e implementa modelos de comunicação em rede projetados para tal fim. O *UnBiHealth* estende o *middleware*, introduzindo conceitos e ferramentas específicas para o desenvolvimento de jogos ubíquos para reabilitação.

Ele foi desenvolvido observando as limitações que existiam em muitos outros trabalhos de jogos para reabilitação. Uma delas é o uso exclusivo de um determinado dispositivo de entrada definido pela implementação. Por causa do acoplamento muito forte entre o tipo de entrada esperado pelo jogo e o produzido pela captação de movimento, caso um paciente precise usar outro mecanismo de entrada para o jogo, é difícil substituí-lo sem dedicar grande esforço a modificar o código existente.

De forma similar, em alguns destes trabalhos, os exercícios que podem ser feitos são limitados por serem pré-determinados pela aplicação e/ou por só cobrem uma parte do

[1] Um *middleware* é uma camada de software que fica entre a plataforma e a aplicação, e que disponibiliza uma série de serviços para a segunda através de uma interface padronizada [1].

corpo. Numa clínica de fisioterapia real, há pacientes com diferentes níveis de lesões em membros superiores, membros inferiores, e na coluna. Não se pode assumir que o mesmo conjunto de exercícios é adequado para todas essas pessoas; é necessário que, para cada perfil de paciente, o fisioterapeuta possa criar exercícios apropriados.

Para resolver estes problemas, o *UnBiHealth* retira o processamento da entrada de dentro da lógica do jogo, substituindo a interface de entrada do jogo, que esperava informações heterogêneas, por uma que conta com valores normalizados. Dispositivos de entrada também passariam a ter interfaces definidas e forneceriam dados para um nó coordenador do sistema para que eles fossem normalizadas e adaptados para cada paciente. Desta forma, para cada mecanismo de entrada, só precisamos codificar a captura de movimentos e criar referências de exercício uma vez, para daí usá-los em qualquer jogo que conectemos ao sistema.

Em termos de arquitetura, a interface do jogo na rede é modelada como um conjunto de **pinos**, cada um dos quais funciona como um canal de entrada e/ou saída para um determinado tipo de dado. O significado do pino é definido pela lógica do jogo. A recepção de um dado pode se refletir numa ação do jogo, e o envio de dados serve para notificar o nó coordenador de algum acontecimento relevante.

A implementação atual do *UnBiHealth* não está completa. Existem *drivers* para sensores *IMU* e para *smartphones* Android, porém, eles são capazes apenas de fornecer os dados de um movimento realizado, não de fazer comparações. Por isso, criou-se uma nova aplicação, denominada *fisiogame-input*, também baseada no *uOS*, cujos objetivos eram gerenciar os sensores, encontrando os *drivers* na rede e controlando a captura de movimentos, e usar os dados coletados para escrever nos pinos do jogo. Na Seção 4.2.4, esta aplicação será discutida novamente, estando feita a descrição da interface de pinos disponibilizada pelo jogo e o tratamento dado a cada um.

4.1.2 Unity *uOS* Plugin

Sendo o *UnBiHealth* baseado no *uOS*, para poder interagir com suas aplicações e *drivers*, é necessário construir o jogo também baseado no *middleware* ou, pelo menos, com a capacidade de se comunicar usando o mesmo protocolo, o *uP* [6]. Estando descartada a possibilidade de criar um motor de jogo novo com estas capacidades, as alternativas eram utilizar a *uImpala*[2], motor criado com base no *uOS*, ou utilizar a Unity[3]. A segunda opção foi escolhida por se acreditar que se tratava de uma ferramenta mais poderosa.

[2] `https://github.com/matheuscscp/uImpala/`
[3] `http://unity3d.com/`

O desenvolvimento de jogos ubíquos na Unity é possível graças à existência de um *plugin* para ela, denominado Unity *uOS* Plugin [28]. Trata-se de um *port* de várias *features* do *uOS* para o C#, possibilitando a utilização delas em *scripts* da Unity.

4.2 Modelagem de Dados

4.2.1 Elementos a Modelar

Há três pontos a serem considerados aqui. No decorrer do Capítulo 3, foi descrita uma série de elementos de jogo que devem ser implementados, cada um dos quais carrega parâmetros diferentes. É desejável que seja possível tratá-los de forma genérica.

Depois, é necessário que o progresso do jogador seja salvo. Isto implica localizar elementos que compõem tal progresso e agrupá-los num conjunto de dados. Uma vez que o computador utilizado pelo jogador pode ser compartilhado com outras pessoas, como numa clínica de reabilitação, por exemplo, é preciso, também, de um mecanismo que diferencie jogos salvos de jogadores distintos.

Por último, já que o jogo faz uso do *UnBiHealth* para suportar múltiplos métodos de entrada, a sua interface de pinos também deve ser especificada, delineando o propósito de cada pino, assim como o tratamento dado a mudanças que neles ocorrem.

As soluções para as questões aqui expostas serão apresentadas nas seções a seguir.

4.2.2 Modelagem de Dados do Jogo - A classe `GameData`

O desenvolvimento do jogo foi guiado pela ideia de que é difícil prever quanto tempo durará a terapia de um paciente. Assim sendo, é necessário criar um jogo facilmente customizável e extensível, cuja manutenção seja fácil e que possa ser adaptado ao tratamento a ser feito. É preciso modularizar elementos do jogo de forma que eles possam ser retirados e acrescentados de forma rápida. Uma maneira de atingir este resultado é criar arquivos de configuração que contenham informações do jogo em formatos padronizados, e classes que possam carregar e aplicar estes dados ao *gameplay*.

A classe `GameData` (Figura 4.1), assim como uma uma série de outras auxiliares, foi criada com este propósito. A `GameData` é um *singleton* que lê arquivos de texto no formato *JSON* e armazena os dados lidos em dicionários correspondentes a cada tipo de informação, em conjunto com uma chave também lida do arquivo. Estes dados, reunidos nas instâncias das classes auxiliares, podem ser requisitados à instância estática de `GameData`, bastando fornecer a chave correspondente.

As classes `SpriteIndex`, `CharacterData` e `ResourceData` servem apenas para contornar uma limitação da Unity, que não permite, normalmente, carregar imagens em *runtime*.

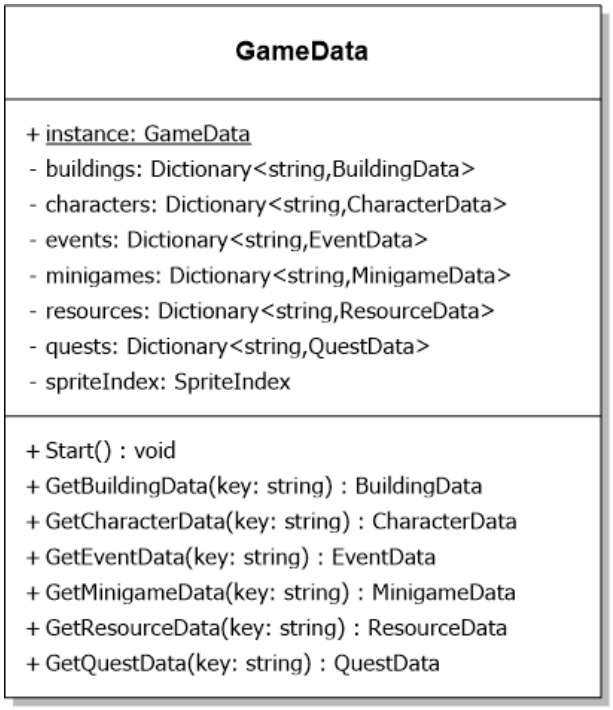

Figura 4.1: Diagrama UML da classe `GameData`.

Utilizou-se a primeira para carregar todos os *sprites* e ícones relevantes. Já para agrupar aqueles relacionados a um personagem ou recurso, foram utilizadas as duas últimas. Assim, quaisquer objetos podem usar os *sprites* em *runtime*.

Essa limitação também nos impede a adição de prédios personagens ou recursos puramente por arquivos de configuração, pois esses dependem de imagens que devem ser acrescentadas ao índice.

As outras quatro classes auxiliares, por sua vez, refletem diretamente a modelagem de elementos do jogo apresentados no Capítulo 3.

Minigames

A estrutura mais simples é a de *minigames*. Cada um recebe uma chave, que corresponde ao nome do principal recurso obtido nele. O arquivo *JSON* de *minigames* deve conter um objeto com aquela chave, que contenha duas coisas: a recompensa correspondente a cada nota dada a uma repetição do exercício e um mapa de recursos bônus.

Código 4.1: Recompensas de um *minigame* em JSON

```
"Wood": {
    "highYield": 7,
```

```
    "midYield": 6,
    "lowYield": 5,
    "bonuses": {
        "Food": {
            "yield": 8,
            "odds": 0.4
        }
    }
}
```

No Código 4.1, que corresponde à configuração do *minigame* do lenhador, tem-se os valores das recompensas (`yields`) para cada nota de exercício. Além disso, há um dicionário de bônus aninhado (`bonuses`), o qual contêm um recurso, indicado também por sua chave, associadosa um `yield` e uma probabilidade (`odds`). O dicionário de bônus pode conter quantas entradas for desejado, contanto que a soma das probabilidades seja menor ou igual a 1.

Cada par chave-valor extraído do arquivo JSON é gravado

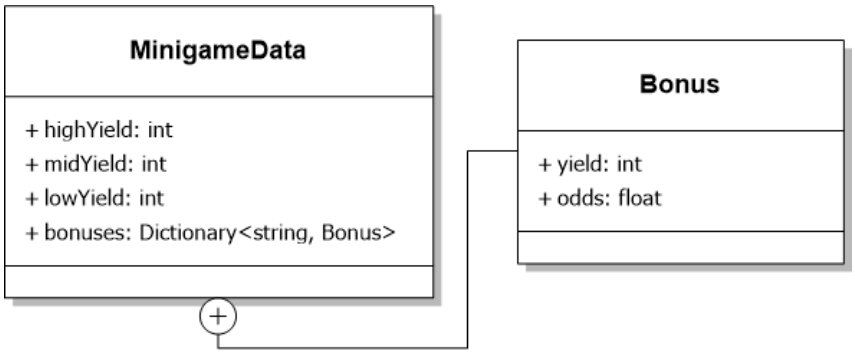

Figura 4.2: Diagrama UML da classe `MinigameData`, junto à classe aninhada `Bonus`.

Prédios

Prédios são objetos presentes na tela principal, visíveis por meio do botão de mapa, que possuem uma chave associada, que deve ser simples para cada um deles. Além disso, cada prédio precisa de um nome mais descritivo para ser mostrado para o jogador. Por último, como mostrado no Quadro 3.2, prédios podem ter quatro diferentes efeitos. Propõe-se, então, a modelagem em *JSON* vista no Código 4.2.

Código 4.2: Um prédio e seus efeitos em JSON

```
"BuildingKey": {
    "name": "UI Name",

    "yield": "Gold",
    "yieldAmount": 0.2

    "ratio": 0.5,
    "fromResource": "Wood",
    "toResource": "Paper"

    "multiplier": 1.05,
    "multipliedYield": "Metal"
}
```

Excetuando o campo `name`, todos os demais são opcionais. No entanto, se um campo de um grupo existe, todos os outros devem existir. Os grupos e suas funções são os seguintes:

- **Grupo 1**: Usado para propiciar o efeito de **Produção Passiva**. Formado por `yield`, que indica o recurso produzido, e `yieldAmount`, valor usado no cálculo da produção quando um *minigame* termina.
- **Grupo 2**: Confere ao prédio a função de **Produção Ativa**. Os recursos de origem e destino são dados por `fromResource` e `toResource`, respectivamente, enquanto a razão entre a quantidade de recursos produzidos e usados é dada por `ratio`.
- **Grupo 3**: Cria **Multiplicadores Simples** ou **Multiplicadores Globais**. O campo `multiplier` deve ser maior que 1, e dita por quanto a produção de um recurso deve ser multiplicada. Já `multipliedYield` contém a chave do recurso multiplicado, ou, no caso de um multiplicador global, contém a string `"Global"`.

Nota-se que é possível incluir os três grupos, fazendo com que um prédio tenha até três efeitos. Se um objeto *JSON* contiver mais de um grupo, todos serão lidos para a `BuildingData` (Figura 4.3) correspondente e o jogo os tratará normalmente. Porém, por questões de simplicidade de *design*, na versão atual do jogo, cada prédio só tem um efeito.

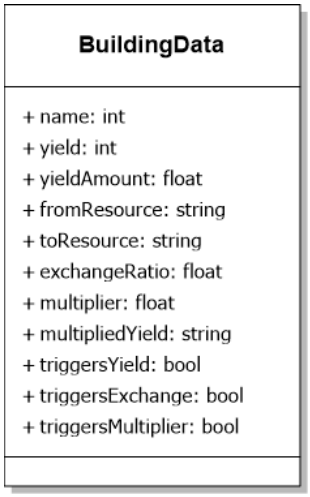

Figura 4.3: Diagrama UML da classe `BuildingData`.

Eventos

Um evento é um diálogo entre personagens, e pode acontecer em decorrência do início de uma sessão ou do destravamento ou conclusão de uma missão. Após seu término, uma nova missão ou *minigame* podem ser desbloqueados. A representação de um evento, portanto, contém dois campos opcionais, indicando missões (`questsUnlocked`) e personagens correspondentes a *minigames* (`charactersUnlocked`) a se tornarem disponíveis após o evento ser concluído. O campo `script` é obrigatório. Ele deve conter uma lista de strings num formato de script mostrado no Apêndice A, a ser lido pela classe `DialogueController` (Seção 4.3.1).

Código 4.3: Dados de um evento em JSON

```
"Teste": {
    "questsUnlocked": [4],
    "charactersUnlocked": ["stonemason"],
    "script": [
        "%frameInterval=3, %leftSpeaker=mayor",
        "Vamos ver o que os moradores precisam hoje?"
    ]
}
```

O Código 4.3 mostra um exemplo possível de evento. A chave dada é importante, pois é por ela que é possível comandar a classe controladora de diálogos a executar um evento

desejado. Também pode ser necessário que outras classes conheçam essa chave quando um evento termina, motivo pelo qual a `EventData` (Figura 4.4) a armazena em `name`.

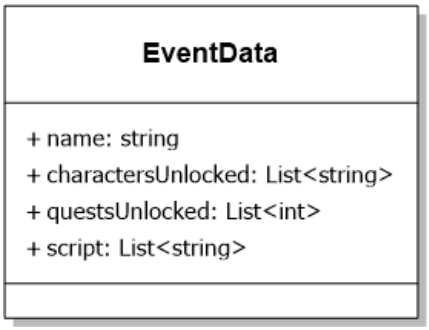

Figura 4.4: Diagrama UML da classe `EventData`.

Missões

Missões são internamente armazenadas pela classe `QuestData` (Figura 4.5), e a modelagem para elas é mostrada no Código 4.4

Código 4.4: Dados de uma missão em JSON

```
"6": {
    "name": "Representantes do Povo",
    "description": "Gracas a voce, a vila prospera.",
    "requirements": {
        "Wood": 100,
        "Stone": 100,
        "Marble": 50
    },
    "builds": "MayorHall",
    "unlockedBy": 2,
    "minimumSession": 2,
    "questOpeningEvent": "",
    "questCompletedEvent": "",
    "questGiver": "mayor"
},
```

A missão representada acima é a mesma que é mostrada na Figura 3.3. Pode ser útil para o leitor observar a figura como referência.

A chave de uma missão deve ser uma *string* numérica. A decisão por esse formato, em vez de uma lista com referências ao índice, se deu pelo fato de que a remoção de uma missão da lista causaria problemas de referência. Um dicionário pode conter um conjunto de números não sequencial.

Os campos `description` e `questGiver` têm propósitos estéticos, sendo mostrados na tela referente à missão. `name` teria o mesmo propósito, mas não é usado na implementação atual. O campo `builds` contém a chave do prédio construído, e `requirements` é um mapa dos requisitos da missão.

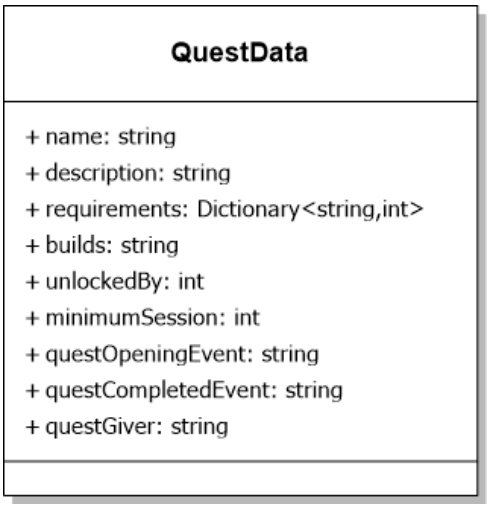

Figura 4.5: Diagrama UML da classe `QuestData`.

Os campos `unlockedBy` e `minimumSession` são, possivelmente, os mais importantes, pois são eles que fazem o encadeamento entre missões. Cada missão tem como pré-requisito uma outra e tem uma sessão mínima na qual pode aparecer. Alterando estes dois campos, é possível remodelar completamente o jogo, adicionando ou excluindo missões a serem desbloqueadas na ordem desejada.

Para manter a experiência do jogador consistente, também é necessário que os diálogos sejam coerentes com as missões dadas. Para isto, a cada missão, podem ser associadas chaves de diálogos a serem mostrados quando ela é desbloqueada (`questOpeningEvent`) e quando é completada (`questCompletedEvent`). Esses campos são opcionais, podendo ser vazios ou inexistentes.

4.2.3 Modelagem de Sessão - A classe `GameState`

É necessário que o progresso do jogador persista entre sessões e, também, que vários usuários possam compartilhar o mesmo computador, mantendo seus respectivos jogos salvos separadamente. A classe `GameState` (Figura 4.6) é outro *singleton* usado pelo jogo,

responsável por carregar jogos salvos, fornecer informações do jogador (e.g quantidade de recursos atuais) para outros objetos e gravá-las de volta em disco quando a sessão do jogador for encerrada.

Figura 4.6: Diagrama UML da classe `GameState`.

Durante sua inicialização em `Awake` e `Start`, `GameState` carrega todos os jogos salvos em `saveFiles`. Um jogo salvo pode ser extraído usando `Load` ou um novo pode ser criado usando `LoadDefault`. Quando necessário, o conjunto de jogos salvos, incluindo o atualmente carregado, é reescrito no disco por `Save`.

Apenas `sessionInitialized`, `lastGameRepetitions` e `activeMultipliers` não são gravados. Essas variáveis servem para controlar a própria sessão e são sempre carregadas com valores padrão. A primeira indica se certas inicializações já foram feitas, a segunda, quantas repetições o último *minigame* jogado teve, e a terceira é um dicionário de multiplicadores gerados pelos prédios atualmente construídos, recalculada sempre que o jogo é carregado ou uma missão é completada.

4.2.4 Modelagem de Input - A classe `GameControl`

Como mencionado na Seção 4.1, o *UnBiHealth* determina que as aplicações implementem canais de comunicação, denominados pinos, que comportam um determinado tipo e intervalo de valores e podem ser escritos interna ou externamente à aplicação. O Fisiogame expõe dois pinos: *punch*, que pode receber um número de ponto flutuante de 0 a 1, e está relacionado com os golpes do personagem no jogo, e *rec*, inteiro que pode ser 0 ou 1, e serve para notificar o jogo sobre o começo e fim de uma gravação de movimento.

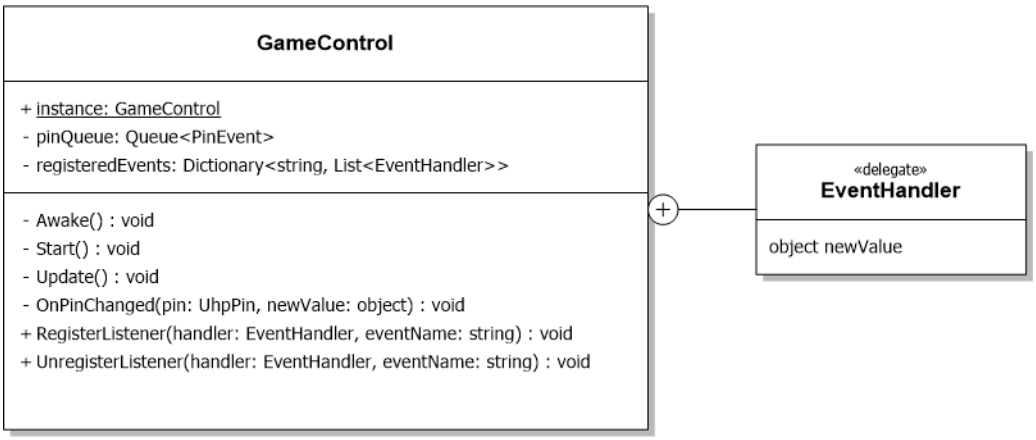

Figura 4.7: Diagrama UML da classe `GameControl` com o delegate `EventHandler`.

Na janela da *fisiogame-input*, apresentada na Seção 3.1.2, grava-se uma curva de referência com o sensor de acordo com o que é esperado no exercício. Em seguida, inicia-se a gravação da tentativa do paciente, momento no qual o pino *rec* é escrito com 1. Terminado o movimento, a curva capturada é comparada com a referência usando *Dynamic Time Warping*[4], obtendo-se um valor de distância `D` Define-se, também, um coeficiente de dificuldade para o exercício, denominada `C`. Assim, calculamos `p`, o valor a ser escrito em *punch* se dá pela seguinte equação:

$$p = \begin{cases} 0, & \text{se } D \geq C \\ 1 - \frac{D}{C}, & \text{se } D < C \end{cases} \tag{4.1}$$

Nota-se que o resultado sempre estará no intervalo [0,1] e é inversamente proporcional ao coeficiente `C`. Se `p = 0`, considera-se que o paciente executou o movimento incorretamente e nada é escrito em *punch* no jogo. Se não, o pino é escrito com o valor calculado. Em ambos os casos, terminada a captura, *rec* é escrito com zero.

[4] *Dynamic Time Warping*, ou *DTW*, é um algoritmo para analisar a semelhança entre duas sequências no tempo que podem variar em velocidade.

Qualquer componente do jogo que necessite receber notificações sobre pinos deve chamar a função `RegisterListener` de `GameControl` (Figura 4.7) com o nome do pino como o segundo argumento e um *callback* do formato `EventHandler` como o primeiro. Isso fará com que o valor do pino, sempre que alterado, seja repassado como argumento ao *callback* registrado.

4.3 Classes Auxiliares à Lógica do Jogo

4.3.1 Tratamento de Diálogos - A classe `DialogueController`

Na Seção 4.2.2, foi mostrado que `EventData` carrega consigo uma lista de strings que compõem um diálogo. Esta lista, na verdade, é um *script* a ser lido pela classe `DialogueController`.

```
%rightSpeaker=lumberjack
%leftSpeaker=mayor
%frameInterval=2
Você deveria ser mais gentil.
%do:highlightRight
Mas eu sou um <i>docinho!</i>
```

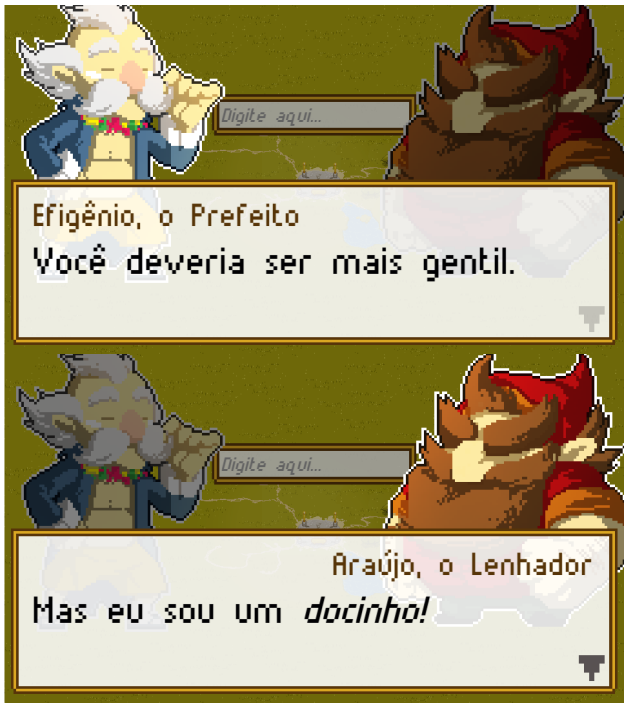

Figura 4.8: Script (à esquerda) com o diálogo gerado correspondente.

Novamente, é preciso ter em mente a necessidade de extensibilidade e rápida modificação. Num jogo em que há muitos diálogos, é pouco prático ter janelas de fala fixa. Faz-se necessário um sistema que possa ler as falas de uma lista e mostrá-las na tela junto aos falantes. Esta é a função desta classe. Ela controla um conjunto de objetos da Unity para mostrar um diálogo especificado num formato dado. Considerando a complexidade desta classe, seu funcionamento interno não será discutido em detalhes, será feita apenas uma descrição sucinta.

O controlador está sempre ativo na tela de praça, porém, invisível. Qualquer *script* pode chamar seu método `Play`, que recebe uma chave de evento a ser buscada em `GameData`. Se o evento existir, a interface de diálogo se torna visível e passa a capturar todos os cliques de mouse para si. A `DialogueController`, então, lê o *script* linha a linha. A documentação completa do formato de script criado pode ser encontrada no Anexo A. Quando chega ao final do *script*, a classe esconde seus elementos de interface e dispara um evento do C#. Qualquer outro componente do jogo pode se registrar previamente como *listener*, caso em que ele receberá a `EventData` do diálogo que terminou. A Figura 4.8 mostra um exemplo de uso da classe.

4.3.2 Visualização de Dados - A classe `ListView`

É comum, não só em jogos, mas em aplicações de modo geral, haver a necessidade de exibir um conjunto de dados na forma de lista. No entanto, a Unity não fornece uma maneira conveniente de fazer isso. Diante disso, é necessário implementar uma classe com tal função.

A classe `ListView` (Figura 4.9) é baseada na ferramenta homônima do *framework* Qt[5]. A classe se baseia na existência de três componentes na visualização de dados: O **Model**, que traz o conjunto de dados, o **Delegate**, que diz como um item do conjunto de dados deve ser mostrado, e a **ListView**, que trata de instanciar Delegates de acordo com o Model que recebe e posicioná-los corretamente na tela.

Por questões de simplificação, assumimos que o Model é uma lista de arrays de `object`[6] onde estão dispostos os dados. O formato destes dados é desconhecido pela `ListView`, mas é conhecido pela especialização de `Delegate` designada para mostrá-los, de forma que, ao instanciá-la e chamar sua função `Set` com um dos itens do Model, os dados sejam extraídos corretamente.

A especialização de `Delegate` a ser usada deve ser um *script* contido no `delegatePrefab`, um `GameObject` que reune os componentes necessários (textos, imagens, etc.) para mostrar os dados de um item do Model. A `ListView`, então, tratará de instanciá-los e calcular sua posição no mundo.

A figura Figura 4.10 mostra três exemplos de uso da `ListView` no jogo: em (a), para mostrar os requisitos de uma missão e o progresso do jogador em alcançá-los; em (b), para mostrar a lista de missões que o jogador tem disponíveis; e em (c), para mostrar as reservas de recursos.

[5] `http://doc.qt.io/qt-5/qml-qtquick-listview.html`

[6] Em C#, `object` é a classe mãe de todas as outras. Isto quer dizer que um `object[]` tanto pode conter qualquer tipo, como pode guardar instâncias de tipos diferentes.

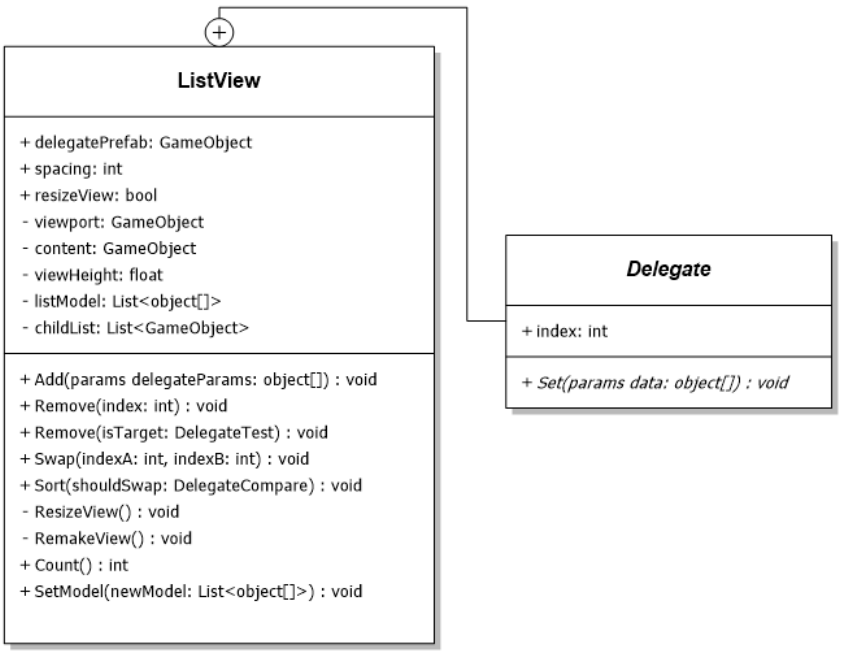

Figura 4.9: Diagrama UML da classe `ListView` e da classe abstrata `ListView.Delegate`.

Figura 4.10: Recortes de três diferentes usos de `ListView` no jogo.

Essa classe poderia ser melhorada fazendo uso de programação reflexiva e eventos, porém, isto não se mostrou necessário para os objetivos deste trabalho.

4.4 Lógica do Jogo

Com as classes auxiliares apresentadas, finalmente chega-se ao controle do jogo como um todo. Como apresentado na Seção 3.3, as mecânicas se dividem em cidade e *minigames*. Da mesma forma, há uma separação da lógica no que diz respeito a cada uma destas partes, como descrito a seguir.

4.4.1 Controle da Praça - A classe `SceneController`

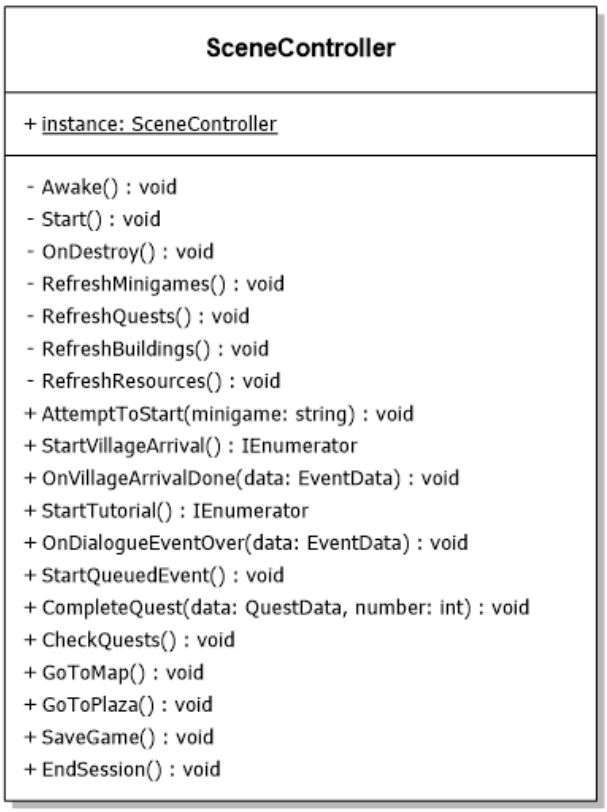

Figura 4.11: Diagrama UML da classe `SceneController`. Alguns membros internos foram omitidos para simplificação.

`SceneController` (Figura 4.11) é responsável pela inicialização e controle das funções da praça. Ela faz constantes consultas a `GameState` para avaliar o que deve ser mostrado ao jogador.

Primeiro, a classe pede ao `DialogueController` que a notifique quando qualquer evento terminar. Feito isto, `GameState` é consultada sobre a *flag* `sessionInitialized`. Quando um jogo salvo é carregado, esta variável é atribuída como `false`, o que indica para o `SceneController` que esta é a primeira vez que o jogador visita a praça. Há dois usos pra isso: se não for a primeira vez, o jogador está voltando de um minigame e o

jogo deve ser salvo. Se for, o controlador consulta se o evento `"VillageArrival"` já foi completado. Este é o tutorial, que deve ser mostrado ao jogador na primeira sessão do jogo.

Em seguida, a classe procura por missões ativas, *minigames* desbloqueados e recursos possuídos e popula as respectivas listas. Os prédios construídos são instanciados, e têm seus efeitos de multiplicador calculados. Por último, elementos decorativos, como personagens na praça e casas na tela de mapa, são posicionados.

Neste ponto, a inicialização está concluída e o jogador pode usar a interface da praça como desejar. O controlador volta a agir quando uma troca de tela é solicitada, um botão de *minigame* é clicado, uma missão é concluída ou um evento termina. O primeiro e o segundo caso são, respectivamente, exibição ou ocultação de elementos de interface e troca de cena da Unity. O terceiro caso envolve acrescentar o prédio referente à missão no mapa, computar seu efeito, subtrair os recursos utilizados e, caso exista, mostrar o evento de conclusão da missão. Finalmente, quando um evento é finalizado, o `SceneController` checa se há missões ou personagens a serem desbloqueados.

Nota-se que os únicos pontos em que o controlador busca algo específico (*hardcoded*) dentre os dados do jogo são durante o tutorial, momento no qual os eventos "`VillageArrival`" e "`Tutorial`" precisam existir, e ao gerar a lista de *minigames*, que exige não só a existência de todos eles, mas há também um nome de personagem associado associado a cada um, e ele deve estar desbloqueado para que o *minigame* apareça na lista.

Com a exceção destes casos, o tratamento de missões, recursos, eventos e prédios é totalmente genérico, demonstrando a efetividade da modelagem proposta na Seção 4.2.2. Novamente, tal tratamento apresenta a vantagem de permitir que o jogo seja expandido, encurtado e rebalanceado para se adaptar a procedimentos fisioterápicos de diferentes durações.

4.4.2 Controle de Minigames - A classe `MinigameController`

A `MinigameController` (Figura 4.12) age como uma máquina de estados, controlando as animações do personagem e dos objetos relacionados ao *minigame*, o qual é cíclico. O personagem anda até um alvo, prepara sua ferramenta, golpeia o alvo repetidamente até destruí-lo e anda até o próximo alvo.

A interação com o jogador acontece durante o estado `ATTACK_WAIT`, que é quando o personagem está preparado para golpear o alvo. Para que o ataque aconteça, o pino *punch* deve ser ativado: o método `OnPunch` atua como *listener* para a `GameControl`. Quando a ativação ocorre, uma nota é dada para o ataque dependendo do valor do pino. Valores no intervalo [0, 0.33] receberão nota **baixa**, (0.33, 0.66], nota **média** e (0.66, 1], nota **alta**.

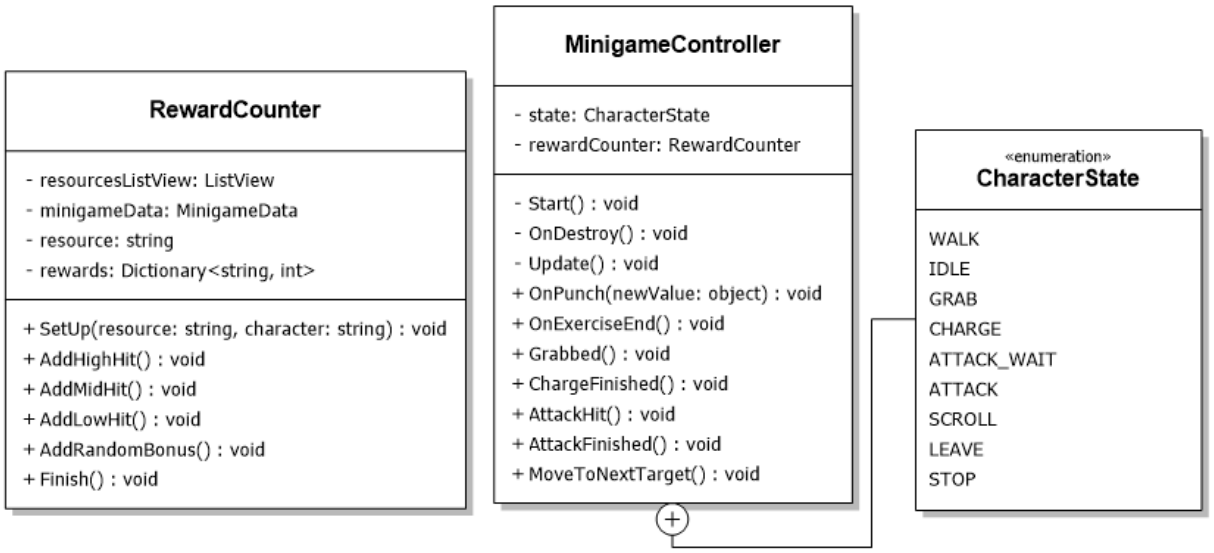

Figura 4.12: Diagramas UML das classes `MinigameController`, da enum `CharacterState` e da classe auxiliar `RewardCounter`. Novamente, alguns membros internos foram omitidos por questão de simplificação.

A nota se reflete de duas formas: primeiro, quanto mais alta ela for, mais rapidamente o alvo será destruído, sendo que a nota máxima o destrói instantaneamente. Além disso, cada golpe leva a uma chamada de método do RewardCounter, que controla os recursos coletados. `Add___Hit` é chamada a cada golpe para a nota correspondente, e resulta no acréscimo do `yield` definido em `MinigameData` às recompensas. `AddRandomBonus`, por sua vez, gera um número pseudo-aleatório e, dependendo deste valor, pode fornecer uma das recompensas bônus, também definidas em `MinigameData`.

Durante o estado `ATTACK_WAIT`, o jogador pode, em vez de fazer o movimento, clicar no botão **Sair**, que faz com que o personagem deixe a cena (estados `LEAVE` e `STOP`) e as recompensas sejam mostradas na tela (`Finish()`). Com isso, encerra-se o *minigame* e o jogador retorna para a tela de praça com os recursos que coletou e, também, os gerados por prédios com Produção Passiva, todos acrescidos às suas reservas.

Capítulo 5

Experimentos

Tendo em vista que o objetivo deste trabalho está em analisar a possibilidade de substituição de sensores *IMU* por um *smartphone* como mecanismo de entrada em um jogo controlado por movimento, é necessário avaliar o impacto dessa substituição na experiência do jogador. Para tanto, foram conduzidos testes com o *Fisiogame*, os quais estão descritos nas seções seguintes.

5.1 Dispositivos de entrada

Os dispositivos usados como parte do experimento foram um par de sensores *IMU* fabricados pela YostLabs[1] e um celular Samsung Galaxy S5[2]. Os *IMU*s eram presos ao braço e ao antebraço do participante por uma pulseira ajustável de velcro, enquanto participantes que usavam o celular seguravam-o com a mão direita. As dimensões dos *IMU*s (altura x largura x profundidade), em milímetros, são de 60 x 35.1 x 15 mm. O celular, por sua vez, tem 142 x 75.5 x 8.1 mm.

5.2 Delineamento

Cada voluntário recebia, primeiramente, o Termo de Consentimento Livre e Esclarecido, que explicava superficialmente a pesquisa sendo feita. Caso concordasse em prosseguir, um dos sensores era escolhido pela pesquisadora para ser usado no experimento. Participantes de número ímpar receberam os sensores *IMU* e os de número par, o *smartphone*.

O participante, então, assistia à cutscene de abertura do jogo, e era conduzido a realizar dez repetições no *minigame* do lenhador. O propósito dessa etapa estava em garantir

[1] `https://yostlabs.com/3-space-sensors/`

[2] `http://www.samsung.com/br/consumer/mobile-devices/smartphones/galaxy-s/SM-G900MZWPZTO`

que os sensores estivessem devidamente calibrados, no aprendizado do movimento a ser realizado pelo jogador (moção de bater um machado lateralmente numa árvore com o braço direito), e em terminar a primeira missão dada no jogo, a de construir uma casa, que foi balanceada para ser completável com dez repetições do primeiro *minigame*, mesmo que o jogador tivesse a pior performance possível.

A partir daí, a comunicação entre a pesquisadora e o voluntário se limitava ao essencial. O jogador era livre para completar as missões na ordem que quisesse, e realizar quantas repetições desejasse dos *minigames*. Esta etapa durava 18 minutos, tempo escolhido por ser considerado suficiente para que os jogadores desbloqueassem a missão do Moinho, que requereria várias repetições do *minigame* de pedra.

Acabando o tempo, o jogo era encerrado e o participante respondia a uma entrevista semi-estruturada. O instrumento utilizado, que está detalhado no Quadro 5.1, é uma adaptação do instrumento apresentado em [12].

Em sua forma original, trata-se de um instrumento quantitativo que mapeia as características das tarefas que levam ao *flow* em seis grupos, que podem ser descritos como Desafio, *Feedback* e Objetivos Claros, Atenção e Concentração na Tarefa, Sensação de Controle, Imersão e Autotelismo. Cada grupo tem dois ou mais itens, cada qual deve ser respondido em uma escala numérica.

A adaptação aqui proposta é um instrumento qualitativo, que transforma alguns destes itens, considerados mais relevantes, em perguntas abertas, além de inserir questionamentos sobre o uso do controle. A ordem das perguntas pode mudar, assim como mais detalhes podem ser pedidos se o entrevistador considerá-los relevantes.

Terminada a entrevista, o participante era livre para fazer perguntas sobre qualquer parte do experimento que o interessasse, com a condição de manter sigilo para não afetar os resultados de futuros voluntários.

5.3 Participantes

Ao descrever os resultados do experimento, por questões de sigilo, não são utilizados aqui os seus nomes reais, mas sim referências aos números dos participantes, e.g. “O participante #2 afirma...”. Alternativamente, pode haver referências da forma “usuário de IMU” ou “usuário de celular”. O Quadro 5.2 apresenta dados demográficos coletados para cada jogador, além do grupo do qual fizeram parte (isto é, o sensor que utilizaram).

5.4 Resultados

Nesta seção, apresenta-se uma análise das respostas dadas para cada pergunta, associadas a observações da pesquisadora.

5.4.1 Imersão

Em respeito ao tempo, apenas um usuário de IMU e outro de celular deram respostas muito distantes do tempo real, sendo estas 40 e 30 minutos, respectivamente. Os outros participantes disseram achar ter passado 15 minutos jogando, estimativa razoável e próxima do valor real. Como explicado na Seção 2.1.2, durante o *flow*, há uma perda da noção de quanto tempo se passa. As respostas discrepantes podem indicar pessoas mais engajadas no jogo. Não é possível estabelecer relação com os sensores, no entanto, dado que ambos os participantes que deram respostas discrepantes usaram métodos de entrada diferentes e, da mesma forma, há variação no grupo dos que acertaram na estimativa.

Já no que tange à noção do ambiente e inclusão no mundo do jogo, apenas um participante (#4) deu uma resposta negativa, afirmando que, no começo do jogo, se sentiu incluído, mas após um tempo, a repetição o fez perder a imersão. Existe uma hipótese de por que apenas esse participante se sentiu assim, que será discutida na Seção *Feedback* e Objetivos Claros.

5.4.2 Desafio

Todos os participantes afirmaram que o jogo era fácil, com alguns destacando a impossibilidade de perder como um fator. Isso era esperado, dado que a real dificuldade do jogo, aplicado ao seu público alvo real, está em recuperar o movimento do membro. Como nenhum dos participantes sofria de lesões motoras, é natural que não tenham achado o jogo desafiador.

No que diz respeito ao método de entrada, ambos os grupos afirmaram não ter tido dificuldade em usar o dispositivo a eles atribuído.

5.4.3 *Feedback* e Objetivos Claros

Os jogadores descreveram os objetivos em termos de acumular recursos e/ou completar missões, alguns apontando a interface e outros, o tutorial como motivo para pensarem assim. O progresso foi expresso da mesma forma. Isto indica que tanto o tutorial quanto a interface da praça estão funcionando como desejado. No entanto, esta parte da entrevista revelou dois grandes problemas com os *minigames*.

Primeiro, alguns jogadores afirmaram sentir falta de *feedback* imediato de recompensas, isto é, a exibição do recurso obtido quando o golpe é realizado. Na versão atual do jogo, as recompensas só são mostradas ao final do *minigame*, e mesmo voluntários que não levantaram essa questão na entrevista se mostraram confusos enquanto jogavam, coletando recursos a mais ou a menos do que o desejado.

O segundo problema é talvez o mais grave, tendo causado a adição de uma nova pergunta à entrevista sobre a percepção de se e como o movimento realizado podia afetar a força do golpe do personagem. Apenas dois jogadores (#4 e #5) notaram, enquanto jogavam, que conseguiam influenciar o golpe com seu movimento. Outros acharam que a variação de força era aleatória.

Mesmo entre estes dois jogadores, nenhum soube explicar o que estava sendo avaliado, ao ponto que o jogador #5 desistiu de tentar influenciar o golpe, optando por uma moção mais confortável, e o #4 tornou-se visivelmente frustrado ao longo do *gameplay* por não conseguir descobrir como deveria realizar o movimento para melhorar sua performance. Isso provavelmente causou a sua perda de imersão, reportada mais cedo no questionário.

Nota-se que, como os dois jogadores em questão usaram métodos de entrada diferentes, não há como associar essa percepção a um sensor específico. Elementos do jogo, como o tutorial e a interface dos *minigames*, devem ser revistos de forma a esclarecer as mecânicas com as quais os participantes tiveram dificuldades.

5.4.4 Atenção e Concentração na Tarefa

Todos os participantes relataram estar concentrados no jogo. O participante #3, usuário de celular, relatou distração com o equipamento em um momento em que acidentalmente desligou a tela, fazendo com que o sensor parasse de funcionar. Este mesmo participante sugeriu que fosse usada uma braçadeira no celular, já que segurá-lo na mão pode fazer com que a falha se repita.

5.4.5 Sensação de controle

Nenhum dos jogadores reportou dificuldade ou desconforto com o método de entrada. O Quadro 5.3 mostra, resumidamente, a experiência prévia de cada usuário.

5.4.6 Autotelismo

Todos os participantes afirmaram ter achado o jogo “prazeroso” ou “divertido”. Um destacou os personagens como um ponto forte, e dois afirmaram que, apesar de gostarem, o jogo era muito repetitivo. É uma avaliação esperada, dado que estamos testando o jogo fora do contexto a que ele se destina.

5.5 Desempenho

O desempenho dos jogadores, aqui definido como o progresso em completar as missões dadas, foi bastante similar. Todos os participantes atingiram a quinta missão no tempo esperado, havendo apenas uma pequena diferença entre os recursos coletados quando o *gameplay* era interrompido. O participante #1 foi quem estava mais longe de completar a missão neste momento, enquanto o #5 estava mais perto.

5.6 Conclusões

Com base nas respostas dadas, podemos observar que certos itens perdem sua relevância quando aplicados a pessoas que não são o público alvo do *Fisiogame*. Questões sobre Imersão, Autotelismo e Desafio requerem uma continuação da investigação com voluntários pacientes de fisioterapia, dado que a eficácia do jogo tem que ser provada durante o tratamento.

É interessante notar, no entanto, que não houve reclamações sobre dificuldade ou desconforto em usar os sensores (Questões sobre Sensação de controle e segunda questão de desafio). Apesar de isso, também, requerer validação com pacientes reais, há uma clara indicação de que ambos os sensores testados são perfeitamente utilizáveis com o jogo.

As perguntas sobre *Feedback* e Objetivos Claros revelaram problemas com a interface dos *minigames*, especificamente no que tange os diferentes níveis de força que um golpe pode ter e a recompensa recebida em cada caso. O segundo problema pode ser resolvido com uma simples indicação visual de quanto recurso foi recebido num momento; já o primeiro requer indicação mais clara dos níveis de força e/ou mais transparência na avaliação da entrada do jogo, possivelmente com exibição em um modelo 3D do movimento esperado.

Também é de importância a sugestão dada pelo participante #3, durante as perguntas de Atenção e Concentração na Tarefa, de usar uma braçadeira para segurar o celular. Isso pode evitar interações indesejadas com o aparelho, além de tornar acidentes (e.g. paciente deixar o celular cair) menos prováveis.

De modo geral, este trabalho resultou na concepção de tanto um jogo quanto um instrumento viáveis para utilização em pesquisas mais aprofundadas, ainda que haja problemas a ser resolvidos no primeiro. Em relação ao questionamento inicialmente apresentado, de se a substituição do *IMU* pelo celular como mecanismo de entrada é viável, o experimento realizado indica que sim, já que em nenhum momento foi possível estabelecer relação entre um determinado grupo e uma experiência pior, e todos os participantes apresentaram desempenho similar.

Quadro 5.1: Formato da Entrevista Semi-estruturada.

Características	**Perguntas**
Imersão *(parte 1)*	Sem olhar no relógio, por quanto tempo você acha que jogou o jogo?
Desafio	O que você achou da dificuldade do jogo? Foi desafiador usar este método de entrada?
Feedback e Objetivos Claros	Como você identificou seus objetivos no jogo? Como você percebia se estava progredindo?
Atenção e Concentração na Tarefa	Em algum momento você se viu distraído(a) com algo fora do jogo? *(Se o controle não for mencionado)* E o [celular/IMU]? Te distraiu?
Sensação de controle	O [celular/IMU] dificultou a sua interação? Você sentiu algum desconforto com o controle do jogo? *(Caso afirmativo)* Você pode falar um pouco mais sobre isso? Você já jogou outros jogos controlados por movimento antes? Quais?
Imersão *(parte 2)*	Você sentiu que perdeu a noção do ambiente? Você se sentiu incluído(a) no mundo do jogo?
Autotelismo	Você achou o jogo prazeroso?
Dados Demográficos	Qual a sua idade? Qual a sua experiência com *video games*?

Quadro 5.2: Dados demográficos de participantes do experimento, além do sensor usado.

Participante	**Idade**	**Experiência com jogos**	**Sensor**
#1	25	Joga casualmente	*IMU*
#2	29	Joga regularmente	*Smartphone*
#3	24	Joga regularmente	*IMU*
#4	23	Joga frequentemente	*Smartphone*
#5	56	Não joga	*IMU*

Quadro 5.3: Experiência de cada jogador com jogos controlados por movimento, por plataforma. Células contendo X indicam experiência.

Plataforma	**#1**	**#2**	**#3**	**#4**	**#5**
Wii	X	X		X	
Kinect		X	X		
Celular		X		X	

Capítulo 6

Considerações Finais

Neste trabalho, objetivou-se avaliar o impacto da substituição do dispositivo de entrada de um jogo controlado por movimento. Para tanto, foi desenvolvido um jogo, denominado *Fisiogame*, com base no *framework UnBiHealth*, que poderia usar como método de entrada tanto um sensor *IMU* quanto um *smartphone* Android. Testes foram conduzidos, nos quais os voluntários, alguns usando *IMU*s, e outros, *smartphones*, responderam a um questionário após jogar o *Fisiogame*.

A conclusão tirada, a partir de respostas dos participantes, observações feitas durante o seu *gameplay* e seu desempenho foi de que a substituição é viável e não causa prejuízos, atentando para o fato de que é recomendável usar uma braçadeira para segurar o celular, impedindo interações indesejadas com o dispositivo. Vale notar que esta é apenas uma avaliação qualitativa, não servindo como prova definitiva de que a substituição é perfeita, mas como indício. Uma possível evolução para esta pesquisa está no uso de instrumentos quantitativos e/ou na realização de experimentos com o público alvo real do jogo, isto é, pacientes de fisioterapia.

O interesse por trás de fazer esta substituição, como mencionado no Capítulo 1, está no fato de que os sensores *IMU* são um equipamento muito especializado. Também há de se considerar que são sensores muito caros, com preços em torno de 250 dólares[1]. É comum que a fisioterapia envolva exercícios também em casa, e seria possível usar o jogo neste ambiente também, porém, não é razoável esperar que o paciente de reabilitação compre um conjunto de sensores para si. *Smartphones*, no entanto, são cada vez mais comuns [25], e o paciente poderia usar o seu próprio aparelho para jogar enquanto se exercita.

Isso cria possibilidades interessantes para jogos feitos neste contexto. Atualmente, o *Fisiogame* mantem seus jogos salvos no computador local, mas seria possível conectá-lo

[1]Preço do sensor utilizado nos experimentos, que pode ser comprado em `https://yostlabs.com/product/3-space-wireless-2-4ghz-dsss/`

a um servidor que, além de controlar o progresso de cada jogador, mantivesse uma base de dados de exercícios prescritos para cada um, associados à frequência com que devem ser realizados. Assim, seria mais fácil para o profissional de fisioterapia acompanhar os exercícios realizados pelo paciente em casa. O paciente, por sua vez, poderia usufruir da diversão proporcionada pelo jogo ao realizar seu tratamento em casa.

Existem, é claro, ajustes a serem feitos no *Fisiogame*. Como apontado nos testes, há momentos em que o *feedback* provido ao jogador é ruim. Isso deve ser resolvido antes de qualquer melhoria ser implementada. Após os ajustes e com o avanço da implementação do *UnBiHealth*, novos dispositivos podem passar a ser suportados, e uma vez que a lógica de pinos implementada é universal, tais dispositivos poderiam ser imediatamente avaliados para uso no jogo.

Uma última, porém, mais complexa possibilidade de trabalho futuro, está na observação do impacto de alterações de mecânicas do jogo. O *Fisiogame* foi inspirado em jogos de celular que permitem ao jogador gerenciar cidades. Esta escolha se deu por se pensar que este tipo de mecânica seria mais familiar a uma faixa etária ampla, ao contrário de jogos ambientados em mundos de fantasia ou ficção científica, por exemplo. No entanto, não é sabido se a escolha foi realmente correta; é possível que outros tipos de jogo tivessem uma eficiência maior na terapia do paciente. Para avaliar essa possibilidade, testes precisariam ser conduzidos com pacientes reais usando várias versões diferentes do jogo.

Referências

[1] David H. Ahl. Editorial. In *Creative Computing Video & Arcade Games*, volume 1, page 4. Ziff-Davis, 1983. 4

[2] Entertainment Software Association. Essential facts about the computer and video-game industry. Technical report, Entertainment Software Association, 2015. 4

[3] Chanin Ballance. Use of games in training: interactive experiences that engage us to learn. *Industrial and Commercial Training*, 45(4):218–221, 2013. 6

[4] Staffan Björk, Jussi Holopainen, Peter Ljungstrand, e Karl-Petter Akesson. Designing ubiquitous computing games – a report from a workshop exploring ubiquitous computing entertainment. *Personal Ubiquitous Computing*, 6(5-6):443–458, January 2002. 8

[5] Elizabeth M. Bonsignore, Derek L. Hansen, Zachary O. Toups, Lennart E. Nacke, Anastasia Salter, e Wayne Lutters. Mixed reality games. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work Companion*, CSCW '12, pages 7–8, New York, NY, USA, 2012. ACM. 9

[6] F. N. Buzeto, C. D. Castanho, e R. P. Jacobi. up: A lightweight protocol for services in smart spaces. In *Ubi-Media Computing (U-Media), 2011 4th International Conference on*, pages 25–30, July 2011. 21

[7] Fabricio N. Buzeto. *Jogos Ubíquos Reconfiguráveis*. Tese (Doutorado), Universidade de Brasília, Brasília, DF, Brazil, 12 2015. 8, 20

[8] Yao-Jen Chang, Shu-Fang Chen, e Jun-Da Huang. A kinect-based system for physical rehabilitation: A pilot study for young adults with motor disabilities. *Research in Developmental Disabilities*, 32(6):2566 – 2570, 2011. 7

[9] Jenova Chen. Flow in games. Dissertação (Mestrado), University of Southern California, California, CA, United States, 2006. 6

[10] M. Csikszentmihalyi. *Flow: The Psychology of Optimal Experience*. Perennial Modern Classics. Harper & Row, 1990. 1, 5

[11] M. Csikszentmihalyi. *The evolving self: a psychology for the third millennium*. HarperCollins Publishers, 1993. 5

[12] Xiaowen Fang, Jingli Zhang, e Susy S. Chan. Development of an instrument for studying flow in computer game play. *International Journal of Human-Computer Interaction*, 29(7):456–470, 2013. 5, 38

[13] Albertine Gaur. *A History of Writing*. Cross River Press, 1992. 8

[14] Lucio Gutierrez, Eleni Stroulia, e Ioanis Nikolaidis. *fAARS: A Platform for Location-Aware Trans-reality Games*, pages 185–192. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. 8

[15] Deutsch JE, Borbely M, Filler J, Huhn K, e Guarrera-Bowlby P. Use of a low-cost, commercially available gaming console (wii) for rehabilitation of an adolescent with cerebral palsy. *Physical Therapy*, pages 1196–1207, 2008. 7

[16] Alfred D. Chandler Jr. *Inventing the Electronic Century*. Number 47 in Harvard Studies in Business History. Harvard University Press, 8 2015. 4

[17] Bieryla KA e Dold NM. Feasibility of wii fit training to improve clinical measures of balance in older adults. *Clinical Interventions in Aging*, pages 775–781, 2013. 7

[18] Bryan J. Kemp. *Motivation, Rehabilitation and Aging: A Conceptual Model*, volume 3, pages 41–51. Wolters Kluwer Health, 1988. 1, 7

[19] Steven Kent. *The Ultimate History of Video Games*. Three Rivers Press, 2001. 3, 4

[20] Teresa J Kimberley, Scott M Lewis, Edward J Auerbach, Lisa L Dorsey, Jeanne M Lojovich, e James R Carey. Electrical stimulation driving functional improvements and cortical changes in subjects with stroke. *Experimental Brain Research*, 154(4):450–460, 2004. 11

[21] K. Koskinen e R. Suomela. Rapid prototyping of context-aware games. In *Intelligent Environments, 2006. IE 06. 2nd IET International Conference on*, volume 1, pages 135–142, July 2006. 9

[22] Chen MH, Huang LL, Lee CF, Hsieh CL, Lin YC, Liu H, Chen MI, e Lu WS. A controlled pilot trial of two commercial video games for rehabilitation of arm function after stroke. *Clinical Rehabilitation*, 29(7):674–682, 2015. 7

[23] Hiroaki Ogata, Ryo Akamatsu, e Yoneo Yano. *Computer Supported Ubiquitous Learning Environment for Vocabulary Learning Using RFID Tags*, pages 121–130. Springer US, Boston, MA, 2005. 6

[24] Peter A. Piccione. In search of the meaning of senet. In *Archaeology Magazine*, volume 33, pages 55–58. Archaeological Institute of America, 1980. 1, 3

[25] Jacob Poushter. Smartphone ownership and internet usage continues to climb in emerging economies. Technical report, Pew Research Center, 2016. 4, 44

[26] Steve Rabin. *Introduction to Game Development*. Charles River Media, 2nd edition, 2009. 11

[27] Luciano H. O. Santos. Arcabouço para construção de jogos ubíquos com foco em reabilitação. Dissertação (Mestrado), Universidade de Brasília, Brasília, DF, Brazil, 4 2016. 10, 20

[28] Luciano H. O. Santos, Fabricio N. Buzeto, Lucas N. Carvalho, e Carla D. Castanho. A game engine plugin for ubigames development. In *SBGames 2014, Trilha de Computação*, 2014. 8, 22

[29] Stephen Shepard, editor. *Business Week*. Nº 3392-3405. Bloomberg L.P., 1994. 1, 4

[30] Penelope Sweetser e Peta Wyeth. Gameflow: A model for evaluating player enjoyment in games. *Computers in Entertainment*, 3(3):3–3, July 2005. 5, 6

[31] Jan-Peter Tutzschke e Olaf Zukunft. Frap: A framework for pervasive games. In *Proceedings of the 1st ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '09, pages 133–142, New York, NY, USA, 2009. ACM. 9

[32] Mark Weiser. The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.*, 3(3):3–11, July 1999. 1, 8

[33] Tiebin Yan, Christina WY Hui-Chan, e Leonard SW Li. Functional electrical stimulation improves motor recovery of the lower extremity and walking ability of subjects with first acute stroke a randomized placebo-controlled trial. *Stroke*, 36(1):80–85, 2005. 11

Apêndice A

Documentação do DialogueController

Script exemplo:

```
%frameInterval=3, %useColoredHighlight=true
%leftSpeaker=Dracula, %rightSpeaker=Richter
Die monster. You don't belong in this world!
%do:highlightLeft
It was not by my hand that I was once again given flesh.
I was called here by humans who wish to pay me tribute.
%do:highlightRight
Tribute? You steal men's souls, and make them your slaves!
%do:highlightLeft
Perhaps the same could be said of all religions...
%do:highlightRight
Your words are as empty as your soul!
Mankind ill needs a savior such as you!
%do:highlightLeft
What is a man? A miserable little pile of secrets.
But enough talk... Have at you!
```

Linhas de fala

Linhas de fala são escritas diretamente como uma linha do arquivo. Cada fala deve estar sozinha na linha, isto é, não deve ser acompanhada de outras falas, atribuições ou ações.

Falas podem conter *rich text*, mas só as quatro tags suportadas pela Unity, que são *b*, *i*, *size*, e *color*. Mais informações podem ser encontradas em `http://docs.unity3d.`

`com/Manual/StyledText.html`. Note, também, que tags faltantes ou aninhadas incorretamente causarão defeitos visuais no texto.

Atribuições

Atribuições tem a forma %var=valor. Várias atribuições podem aparecer juntas na mesma linha separadas por vírgula, mas é essencial garantir a presença do '%' em todas elas. Atribuições possíveis incluem:

- `leftSpeaker [string]`: nome do personagem a ser usado como falante à esquerda.
- `rightSpeaker [string]`: nome do personagem a ser usado como falante à direita.
- `frameInterval [int]`: número de frames entre mostrar um caracter da fala na tela e mostrar o próximo. A velocidade máxima de texto é atingida com '1'. O valor '0' desliga o *typewriting*, fazendo com que as falas sejam renderizadas instantaneamente. O valor padrão é '1'.
- `intervalTime [int]`: número de frames aguardado após mostrar um caracter de intervalo ('?', '!' ou '.'). É importante notar que, quando uma sequência desses caracteres aparece na fala (e.g. '...'), só o primeiro resultará na espera de `intervalTime`. O valor padrão é 60.
- `halfIntervalTime [int]`: número de frames aguardado após mostrar um caracter de meio intervalo (',', ';' ou ':'). Apesar do nome, não é exigido que este valor seja metade de `intervalTime`. O valor padrão é 30.
- `delay [int]`: número de frames aguardadas antes de começar a mostrar caracteres da próxima fala. Válido apenas para uma linha de fala.
- `mute [bool]`: Desabilita o som do texto. O valor padrão é *false*.
- `useColoredHighlight [bool]`: Habilita escurecer um personagem enquanto o outro está falando. O valor padrão é *true*.
- `alwaysAutoSkip [bool]`: Faz com que o texto seja pulado automaticamente, sem a necessidade de o jogador clicar na tela. Aplicado a todas as linhas de fala até ser desativado. O Valor padrão é *false*.
- `autoSkipDelay [int]`: Quando o *auto-skip* está ativado, diz o número de frames a aguardar até passar para a próxima fala. O valor padrão é '0' (pula instantaneamente).

Ações

Ações são construções da forma %do:ação. Elas podem ser agrupadas assim como as atribuições. As ações possíveis são:

- `highlightLeft`: seta o personagem à esquerda como o falante atual e, se `useColoredHighlight` estiver ativo, colore o personagem à direita com uma cor escura.
- `highlightRight`: seta o personagem à direita como o falante atual e, se `useColoredHighlight` estiver ativo, colore o personagem à esquerda com uma cor escura.
- `hideLeft`: remove o personagem à esquerda da cena.
- `hideRight`: remove o personagem à direita da cena.
- `autoSkipNext`: faz com que a linha seguinte não aguarde input do usuário para passar para a próxima, mas sim avance automaticamente após `autoSkipDelay` frames.
- `noLeftSprite`: faz com que, durante a próxima fala, o personagem à esquerda fique invisível. Não é permanente, na fala seguinte, o sprite retornará.
- `noRightSprite`: o mesmo que `noLeftSprite`, mas para o personagem da direita.