TRABALHO DE GRADUAÇÃO

Uso de sensores no ajuste dinâmico de dificuldade híbrido em jogos

Guilherme Victor Ramalho Natal

Brasília, Setembro de 2018

ENGENHARIA
MECATRÔNICA
UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia
Curso de Graduação em Engenharia de Controle e Automação

TRABALHO DE GRADUAÇÃO

Uso de sensores no ajuste dinâmico de dificuldade híbrido em jogos

Guilherme Victor Ramalho Natal

Relatório submetido como requisito parcial de obtenção de grau de Engenheiro de Controle e Automação

Banca Examinadora

Prof. Carla Denise Castanho, CIC/UnB _____
Orientador

Prof. Dr. Tiago Barros Pontes e Silva, DIN/UnB _____
Examinador interno

Prof. Dr. Mauricio Miranda Sarmet, IFPB _____
Examinador externo

Brasília, Setembro de 2018

FICHA CATALOGRÁFICA

GUILHERME, VICTOR R NATAL
Uso de sensores no ajuste dinâmico de dificuldade híbrido em jogos

[Distrito Federal] 2018.

viii, 44p., 297 mm (FT/UnB, Engenheiro, Controle e Automação, 2018). Trabalho de Graduação – Universidade de Brasília.Faculdade de Tecnologia.

1. Ajuste dinâmico 2.Jogos eletrônicos
3. Dados fisiológicos.

I. Mecatrônica/FT/UnB II. Controle e Automação

REFERÊNCIA BIBLIOGRÁFICA

NATAL, GUILHERME VICTOR, (2018). Uso de sensores no ajuste dinâmico de dificuldade híbrido em jogos . Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-$n°$08, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 52p.

CESSÃO DE DIREITOS

AUTOR: Guilherme Victor R Natal

Uso de sensores no ajuste dinâmico de dificuldade híbrido em jogos

GRAU: Engenheiro ANO: 2018

Guilherme Victor R Natal

Universidade de Brasília, Campus Universitário Darcy Ribeiro — Asa Norte

CEP 70910-900 Brasília–DF – Brasil.

RESUMO

A dificuldade percebida em jogos digitais é um fator pessoal que gera um campo para novos mecanismos de ajuste de dificuldade para organizar e adaptar a dificuldade do jogo para diferentes jogadores. Atualmente, muitos jogos ajustam sua dificuldade com base na performance do jogador, mas vários pesquisadores mostram modelos que ajustam a dificuldade de acordo com o estado afetivo do jogador ou com base na performance e no estado afetivo. Portanto, esse trabalho tem como objetivo investigar os elementos em um jogo *shooter* 2D que podem melhorar a chance de levar o jogador do estado de fluxo utilizando o modelo híbrido de ajuste dinâmico de dificuldade (ADD). Para fazer isso, uma nova versão do jogo *Asteroids* foi adaptada usando uma placa de aquisição de dados fisiológicos e algoritmos para extrair os dados do jogo. Experimentos foram conduzidos com o objetivo de determinar quais seriam as variáveis relevantes para implementar o ADD (velocidade e densidade) e para confirmar se o jogador entrou em estado de fluxo com e sem os dados sobre o estado afetivo no ADD. Além disso, cada participante também respondeu a um questionário no qual verifica o quando o estado de fluxo foi percebido. Foi observado como as variáveis influenciaram a percepção do jogador sobre o jogo,conclui-se que velocidade tem uma importância maior do que a da densidade.

Palavras-chave: Ajuste dinâmico de dificuldade, jogos eletrônicos, dados fisiológicos.

ABSTRACT

The difficulty perceived in digital games is a personal factor that generates a field for new mechanisms of adjustment of difficulty to arise and adapt to different players. Currently, most games adapt difficulty based on the player's performance, but several researches show models that adjust the difficulty according to the affective state of the player or the conjunction of both models. Therefore, this work aims to investigate elements in 2D shooters that may improve the perception of the flow state of a player through the hybrid model of Dynamic Adjustment of Difficulty (DDA). For that, a version of the game *Asteroids* was adapted using a physiological data acquisition board and algorithms to extract data from the game. Experiments were conducted in order to analyze the objects of the DDA to be implemented (speed and density) and to verify the perception of the flow with and without affective data being considered in the DDA. Besides, each participant also answered to a questionnaire which quantifies how much the state of flow was perceived. It was observed how the variables influenced the perception of the player about the game, with the speed having a greater impact than the density.

Key-words: Dynamic difficult adjustment, games, Physiological data.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

Capítulo 1

Introdução

A relação entre a dificuldade e o nível de habilidade em um jogo tem influência direta no prazer sentido pelo jogador durante uma sessão de jogos. Um jogo mais difícil do que um jogador pode superar torna a experiência de jogo frustrante, enquanto um jogo fácil demais para um certo jogador pode levar ao tédio.

Para adequar a dificuldade do jogo ao jogador foi criado o conceito de ajuste dinâmico de dificuldade (ADD) [6]. Um ADD utiliza métodos para coletar dados e quantificar a performance do jogador a partir dos dados coletados ele tenta ajustar a dificuldade para melhor se adequar ao jogador.

O objetivo principal do ADD é maximizar a diversão do jogador no intuito de fazer o jogador sentir-se imerso no jogo. A teoria do fluxo (ou *flow*) [10], oriunda na psicologia, pode ser usada para verificar o nível de imersão do jogador. O Fluxo é um estado psíquico em que o indivíduo se mantém completamente concentrado em uma tarefa na qual sua habilidade está no nível do desafio.

Atualmente existem muitos gêneros de jogos digitais, para cada um deles está associado um tipo de dificuldade. Para cada tipo de jogo existe um desafio associado, o nível do desafio varia de acordo com a característica de cada jogo. Devido a esse desafio ser muito diferente de um jogo para outro, a implementação do ADD deve ser única para cada jogo.

Quando os primeiros ADD foram implementados, eles utilizavam apenas a coleta de dados do jogo para avaliar a performance do jogador – chamados também de dados *in-game*. Um exemplo desse tipo de ADD foi feito para o jogo *half-life* [6], no qual foi utilizado o conceito de economia de itens para balancear o jogo, mantendo o estoque de itens do jogador adequado para cada situação.

Alguns autores como [7] perceberam que pode se utilizar o estado afetivo do jogador para inferir e avaliar a dificuldade percebida por ele. Esses autores criaram o conceito de ADD afetivo, que faz uso de sensores para capturar dados fisiológicos do usuário. Tais dados são processados posteriormente para o sistema utilizar como parâmetro para alterar a dificuldade em tempo de execução.

1.1 Definição do problema

Em geral o ADD é feito utilizando apenas os dados do jogo ou dados fisiológicos, como mencionado anteriormente. Essas abordagens se mostraram eficazes em manter o jogador em estado de fluxo, entretanto, não foram encontrados na base de dados do IEEE Explore Digital Library, da ACM (Association for Computing Machinery) e do Google Scholar, trabalhos que quantifique e ajuste a dificuldade dinamicamente com base em ambas fontes de dados, do jogo e fisiológicas do jogador em conjunto. Em outras palavras, não foi encontrado na literatura um ADD híbrido, e portanto, é desconhecido se esta abordagem poderia trazer benefícios em comparação com as já difundidas.

1.2 Objetivo

O objetivo principal deste trabalho é verificar se um ADD híbrido é mais eficiente do que os ADDs tradicionais, ou seja, aqueles baseados somente em dados do jogo ou somente em dados fisiológicos do jogador. Para tanto, este trabalho envolve a criação de uma ferramenta capaz de coletar dados sobre jogo de forma que seja possível quantificar a influência de um parâmetro (como a velocidade de algum elemento, por exemplo) na percepção de dificuldade. Um ADD híbrido se faz combinando dados coletados do jogo (dados *in-game*) e dados fisiológicos do jogador.

Para o ADD híbrido a ser implementado neste trabalho, os dados fisiológicos serão coletados por meio de sensores de condutância (EDA) de pele e de eletrocardiografia (ECG), esses dois sensores tem relação direta com o estado afetivo do jogador. Além disso, pretende-se averiguar quais medidas pode aferir de maneira mais confiável o estado afetivo do jogador. Os dados coletados pelos sensores fisiológicos devem ser usados em conjunto com os dados do jogo para qualificar o nível de frustração do jogador, e assim ajustar a dificuldade do jogo de modo que o jogador entre no estado de fluxo.

1.3 Metodologia

Para se alcançar os objetivos propostos serão utilizadas adaptações do jogo *Asteroids*, desenvolvido em Unity. Este jogo é um *shooter* 2D. O objetivo do jogo é encontrar uma pílula azul sem que o jogador se colida com os asteroides.

Na primeira adaptação do jogo temos apenas uma fase com questionários sobre a dificuldade percebida ao final de um intervalo de tempo. Nesta adaptação objetivo do jogo é sair do campo de asteroides. Já a segunda adaptação consiste de 6 fases com diversos tipos de asteroides e uma mudança de dificuldade de acordo com o desempenho do jogador e seu estado afetivo. Nessa adaptação são incluídos vários tipos de asteroides mas o objetivo do jogo continua o mesmo.

O objetivo da primeira adaptação do jogo é apenas quantificar a influência de algumas variáveis do jogo na percepção de dificuldade do jogador. Os resultados da primeira adaptação serão

utilizados no ADD da segunda adaptação, que será a implementação do ADD híbrido de fato.

Para cada adaptação do jogo são conduzidos testes com usuários, mas apenas a segunda adaptação é utilizada para quantificar o nível de fluxo por parte do jogador. Essa quantificação é feita por meio de um questionário descrito no Capítulo 5. A Figura 1.1 mostra uma visão geral das etapas conduzidas neste trabalho.

Figura 1.1: Fluxograma do trabalho

O restante desta monografia está organizada da seguinte forma. No Capítulo 2 é apresentada a fundamentação teórica, isto é, uma breve descrição dos conceitos abordados neste trabalho. O Capítulo 3 traz alguns trabalhos correlatos e o Capítulo 4 foi dedicado a apresentação do DDA Híbrido proposto neste trabalho. O relato dos testes conduzidos durante esta investigação, assim como os resultados obtidos estão no Capítulo 5. Por fim, o Capítulo 6 apresenta a conclusão juntamente com algumas sugestões de trabalhos futuros.

Capítulo 2

Conceitos Fundamentais

Este capítulo fornece o embasamento teórico necessário para a compreensão deste trabalho, incluindo os conceitos chaves, tais como: jogo, jogador, taxonomia de jogos, *biofeedback*, estado de fluxo e nível de dificuldade.

2.1 Jogo

Existem várias definições para jogo, mas a definição dada por Juul [11] é aquela que tem melhor adequação ao que será exposto. Segundo ele, o jogo é um sistema formal baseado em regras com resultado variável e quantificável, no qual a diferentes resultados são atribuídos diferentes valores. Além disso, o jogador se esforça para influenciar o resultado, o jogador se sente emocionalmente ligado ao jogo e as consequências da atividade são opcionais e negociáveis.

Essa definição é adequada para jogos digitais, pois os mesmos possuem um conjunto de regras não negociáveis, mesmo o uso de *cheats* já estando incluso nessa definição.

Outras definições importantes para esse trabalho são: jogabilidade e *gameplay*. Em [12] o conceito de jogabilidade é descrito como sendo a virtude que um jogo possui de ser fácil e intuitivo de jogar. Outro conceito de jogabilidade é como o jogador interage com a mecânica do jogo.

Segundo [12] *gameplay* é definido como o ponto de encontro entre o jogador e o jogo e o entendem como um espectro de possíveis ações e reações que são geradas tanto pelo jogador quanto pelo jogo. Esse conceito tem muita semelhança com o conceito de jogabilidade, mas ainda assim são diferentes. *Gameplay* esta relacionado ao uso do jogo enquanto jogabilidade esta relacionada a interface do jogo.

A partir dos conceitos apresentados anteriormente é desenvolvido um jogo com uma determinada jogabilidade para implementar um ADD.

Tabela 2.1: Classificação dos jogos

Categoria	Sub categoria	Representantes
Ação	Plataforma 2D	Plataforma 2D: Donkey kong
	Luta competitiva	Luta competitiva: Street fighter
	First-person-shooter	First-person-shooter: Half-life
Estratégia	Massively Multiplayer Online Role-playing Games (MMORPG)	MMORRPG: Ragnarok
	Real-time strategy	Real-time: Warcraft III
	Turn based strategy	Turn based strategy:Final fantasy
Aventura	2D	2D:Maniac mansion
	3D	3D:UltimaTextual:Zork
	Textual	
Simulação	EA sports	EA sports: FIFA
	EA race	EA race:NASCAR
Puzzle	Construção	Visual Matching:Tetris
	Visual Matching	Construção:The Incredible Machine series
	Hidden objects	Hidden objects:Minesweeper
Educativos	Educação de crianças	Educação:Urban jungle
	Programação	Programação:Code wa
	Jogos sérios	Sérios: America's army

2.2 Tipos de jogos e suas classificações

Existem diversos tipos de jogos, e nem sempre um jogo está completamente inserido em uma categoria ou em outra. Para este trabalho o entendimento básico da taxonomia de jogos se faz necessário, pois cada jogo requer de um ADD diferente.

De acordo com [13] temos 6 grandes categorias de jogos, e várias subcategorias. A Tabela 2.1 resume as categorias, sub categorias e alguns exemplos representantes. Essa classificação não é absoluta por se tratar de um tema subjetivo e muitos jogos se encaixarem em mais de uma categoria. Ainda assim não significa que a categoria de um jogo deva ser deixada de lado, apenas deve-se levar em conta que mais de uma classificação pode estar correta.

Para cada tipo de jogo está associada uma mecânica, uma maneira de jogar e um objetivo, assim não existe uma maneira universal de obter êxito nos jogos. Como cada categoria de jogo possui suas próprias peculiaridades, não existindo também um ajuste dinâmico universal. Em jogos que simulam a realidade como FIFA e NASCAR, por exemplo, não se deve alterar a física do jogo de maneira que ela extrapole as do mundo real. Por outro lado em jogos de plataforma é possível modificar o modelo de física para permitir ao jogador pulos mais longos, conceder mais força, ou fazer qualquer mudança de modo a beneficiar ou prejudicar o jogador.

Para estudar o ajuste de dificuldade deve-se analisar especifica mente o tipo de jogo de interesse. Em jogos como *Mario world* (plataforma 2D), a dificuldade está diretamente relacionada com as seguintes variáveis: distância entre plataformas, distância máxima do pulo e quantidade de inimigos no mapa. Em jogos FPS como *Half-life* a variável que determina a dificuldade é a quantidade de estoque do jogador, então cada tipo de jogo tem suas próprias variáveis de dificuldade. No jogo deste trabalho a dificuldade está relacionada a quantidade de asteroides no campo e a velocidade dos asteroides.

Figura 2.1: Exemplo de eletrocardiograma (retirada de: [1]).

2.3 Sensores de *biofeedback*

Como [1] explica, *biofeedback* é o mecanismo de obtenção de variáveis do corpo humano em resposta a um estímulo em tempo real. Pode-se separar *biofeedback* em duas grandes categorias: biomecânico e fisiológico. Os biomecânicos estão relacionados aos movimentos e capacidades motoras, enquanto os fisiológicos se referem a variáveis do corpo humano como, temperatura, frequência cardíaca, condutância da pele, dentre outras.

É possível medir todas as variáveis citadas acima, entretanto, neste trabalho tem-se interesse em medir duas variáveis fisiológicas: os sinais elétricos cardíacos e a condutância da pele. Foram escolhidos esses dois sinais pela sua medição estar presente na placa BITalino e por poder-se inferir o estado afetivo desejável na pesquisa com eles de acordo com [1].

2.3.1 Eletrocardiografia

Eletrocardiografia (ECG) é o nome dado à coleta da forma de onda elétrica do batimento cardíaco e à plotagem em um gráfico denominado eletrocardiograma (Figura 2.1). Assim como descrito em [1], a partir do ECG podemos extrair indicativos de entusiamo ou tédio, obtidos pela frequência cardíaca ou pela amplitude da onda.

2.3.2 Sensor de condutância de pele (EDA-*Electrodermal Activity sensor*)

O Sensor de condutância de pele, ou *Electrodermal Activity sensor* (EDA), mede a resposta galvânica da pele. Geralmente, a resposta da pele está relacionada a alguma reação do sistema nervoso simpático, indicando uma mudança no nível de entusiasmo segundo [1]. O EDA detecta mudanças na condutividade da pele. Entretanto, é importante destacar que esse sensor possui algumas falhas, uma vez que diferentes tipos de peles humanas podem gerar diferentes padrões elétricos. A Figura 2.2 mostra um exemplo de medição de EDA com condutância no eixo y.

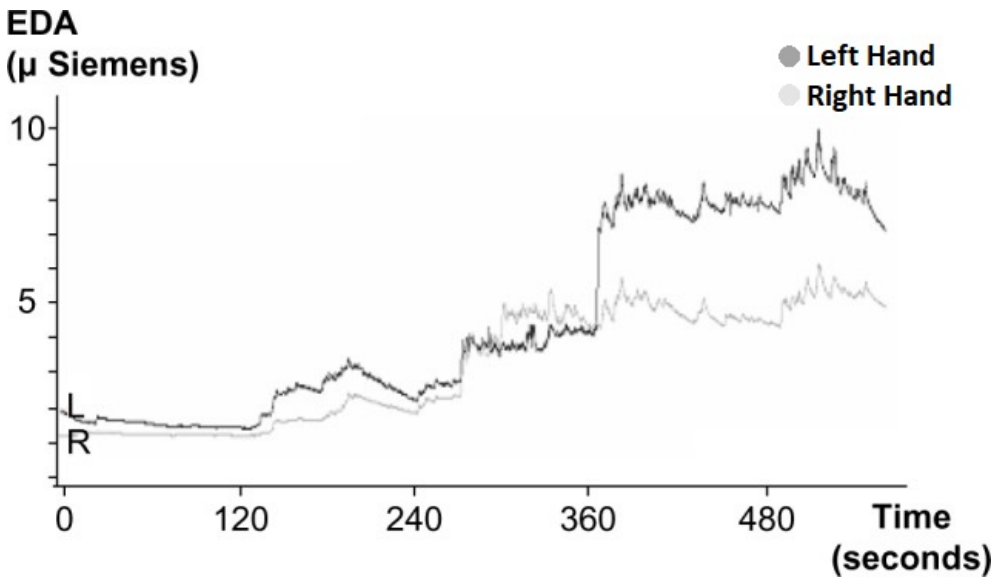

Figura 2.2: Exemplo de medição da condutância da pele pelo sensor EDA (retirada de: [1])

2.4 O jogador

Uma vez que o jogador é o agente chave neste trabalho, é fundamental entender e estabelecer como este se relaciona com o jogo. Mais precisamente, qual tipo de jogador está mais inclinado a aceitar a frustração, qual está menos inclinado e o que motiva cada tipo de jogador.

Assim como existem vários tipos de jogos também existem vários tipos de jogadores. Dentre vários aspectos, podemos caracterizar os jogadores com relação a frequência que os mesmos interagem com jogos digitais.

2.4.1 O Jogador classificado por frequência

. Para este trabalho vamos usar a classificação dada por [2], que separa os jogadores em dois grandes tipos, os casuais e os experientes.

São ditos jogadores casuais aqueles que jogam com baixa frequência, preferem jogos rápidos, como os jogos *mobile* por exemplo, que não exigem prática ou elevado grau de domínio da mecânica. Em geral buscam por uma distração momentânea. Para jogadores casuais os jogos *web* e *mobile* costumam ser os mais adequados.

São ditos jogadores experientes aqueles que jogam com alta frequência. Estes preferem jogos mais complexos, em que o domínio da mecânica é mais elaborado. Jogadores experientes, em sua maioria, dedicam-se diariamente aos jogos e têm maior tolerância à falha.

Jogadores casuais não possuem a habilidade dos jogadores experientes, logo estão mais sujeitos a falhas. Jogadores casuais ainda têm menor disposição à frustração, enquanto jogadores experientes apresentam maior paciência e tendência a explorar os jogos. Baseando-se nessas diferenças devemos considerar ajustes de dificuldade diferentes para cada perfil de jogador.

2.5 O fluxo

No dia a dia é comum ver pessoas focadas em uma atividade, em certos casos o individuo chega a perder a noção de tempo e do que tem a sua volta. Essa sensação também é comum durante uma sessão de jogos, no qual é possível que a pessoa se sinta dentro do mundo do jogo. Para explicar esse fenômeno muitos cientistas usam os termos estado de fluxo (ou *flow* em inglês) e imersão.

Segundo [14], o fluxo originalmente foi um termo utilizado para descrever o prazer encontrado na imersão nos afazeres diários. Em sua pesquisa ele percebe que, durante o período no qual alguém se encontra no estado de fluxo, o mesmo está completamente focado e desligado do mundo exterior à sua atividade.

Como citado em [15], o fluxo se refere à sensação de uma experiência ótima por parte do indivíduo. Em estado de fluxo o indivíduo tem a sensação de prazer por receber uma tarefa que aparenta ser fácil e natural, enquanto causa a sensação de foco por ser desafiadora. Uma faceta do fluxo é que existe um equilíbrio entre o desafio apresentado pela tarefa a habilidade do individuo. A partir disso um jogo que pretende trazer o jogador para o estado de fluxo deve ser balanceado adequadamente para que um jogar experiente não se sinta entediado ao jogar o jogo e um jogador inexperiente não se sinta frustrado por não conseguir.

Para algumas pessoas, jogos são uma maneira de se distrair do mundo real [2], eles permitem que o jogador se envolva no mundo do jogo, em certos casos, perdendo a noção do tempo. Nesses casos a atenção do jogador está quase totalmente voltada para a atividade em questão, resultando na sensação de que ele é o próprio personagem do jogo.

A experiência de entrar no mundo do jogo e se sentir o próprio personagem atribui-se o nome de imersão. O ser humano entra em estado de completa imersão quando lhe é apresentado uma tarefa que nem lhe exija demais, causando frustração, ou de menos causando tédio. Nesse caso, o indivíduo se encontra no estado de fluxo. A Figura 2.3 ilustra essa relação, ou seja, quanto maior for a habilidade de um jogador, maior deve ser o desafio e o contrário também é valido.

Em seu trabalho [10] demonstrou que existem oito componentes são característicos do estado de fluxo:

- Um objetivo claro;
- Uma resposta imediata do jogo;
- Foco na atividade;
- Perda da noção de tempo;
- Uma atividade desafiadora;
- Ação e percepção;
- Perda da consciência própria;

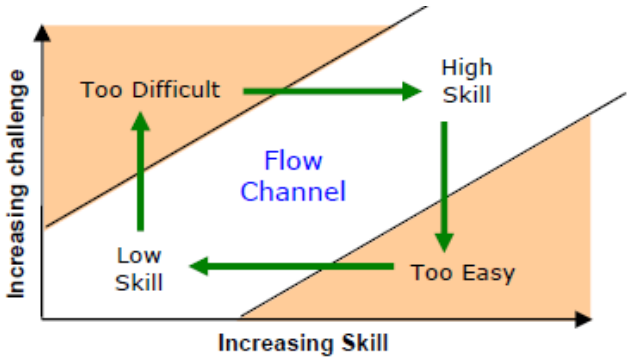

Figura 2.3: Gráfico associando dificuldade a habilidade do jogador a condição de fluxo (retirada de: [2]).

- A sensação de estar no controle.

Quando todas essas condições são atingidas a pessoa entra em estado de fluxo e consequentemente ela fica completamente imersa na atividade e perde a noção de tempo. O conceito de fluxo é muito importante para o consumo de jogos digitais, pois a qualidade do produto pode ser avaliada por sua capacidade de fazer o jogador entrar no estado de fluxo. Dessa forma pode se dizer que o objetivo final de um jogo é trazer o jogador para dentro do mundo do jogo, causando nele o sentimento de imersão.

2.6 Ajuste de dificuldade

Em geral, jogos eletrônicos têm como objetivo divertir pessoas. Não há uma fórmula para se fazer um jogo divertido, entretanto, durante o processo de criação de um jogo, alguns fatores são determinantes, dentre os quais o estabelecimento da dificuldade do jogo. Tradicionalmente, a dificuldade de um jogo é separada em níveis *easy, medium e hard* (em ordem crescente do mais fácil para o mais difícil), de maneira que o jogador pode escolher uma das três e, geralmente, permanece na mesma até o final do jogo.

Assim como dito em [2], o conceito de dificuldade está ligado à relação entre habilidade e desafio, de modo que a percepção de dificuldade é variável para cada pessoa. Dessa relação também pode se deduzir que, em termos absolutos, quanto maior a habilidade necessária para realizar uma tarefa, maior será a dificuldade do desafio.

Nos primeiros jogos, como *pong* e *Asteroids*, não haviam mecanismos de ajuste de dificuldade. Devido a sua simplicidade, o jogo tinha a dificuldade constante, portanto o desafio era constante. Os jogos de super nintendo *Mario* e *Megaman*, por outro lado, já possuem um mecanismo que aumenta a dificuldade a cada fase do jogo. Os jogos de *Playstation 2* como *God of War* possuem um mecanismo de ajuste ainda mais flexível, isto é, logo no início do jogo é possível escolher entre as seguintes dificuldades: *Easy*, *Normal*, *Hard* e *Very Hard*. Além disso, caso o jogador morra seguidas vezes em determinado ponto do jogo, é oferecido ao jogador a opção de trocar de

dificuldade. Isso não é considerado um Ajuste Dinâmico de Dificuldade (ADD), pois o ajuste é feito pelo jogador não pelo sistema do jogo em si.

A maneira como um jogo ajusta sua dificuldade, segundo [16], pode ser dividida em 3 categorias: Estáticos, Parcialmente Adaptativos e Adaptativos.

2.6.1 Jogos com dificuldade estática

Jogos com dificuldade estática não possuem mecanismos para ajustar a dificuldade ao jogador. Quando há algum tipo ajuste, trata-se de permitir ao jogador escolher uma dificuldade pré-definida para aproximar-se da dificuldade ideal àquele jogador. Exemplos desse tipo são *pong*, que não possuí nenhuma escolha da dificuldade e *God of War*, ambos mencionados anteriormente.

2.6.2 Jogos Parcialmente adaptativos

Neste tipo de jogo a dificuldade muda em apenas uma direção, isto é, ou aumenta ou diminui. Também pode existir um mecanismo para detectar se o nível do desafio está abaixo da habilidade do jogador. Porém, o jogo não altera a dificuldade automaticamente. Nesta categoria pode-se citar jogos como *Elder Scrolls* e *Mega-man Zero*.

2.6.3 Jogos Adaptativos

Nesta categoria existe uma forma de ajuste dinâmico de dificuldade, de modo que o desempenho do jogador está sempre sendo medido e monitorado para aumentar ou diminuir a dificuldade automaticamente. Um exemplo de jogo com esta abordagem é o *Resident evil 4*.

Tendo em vista o foco deste trabalho, a próxima seção será destinada exclusivamente a este tipo de ajuste dificuldade.

2.7 Ajuste dinâmico de dificuldade

O Ajuste dinâmico de dificuldade (ADD) é uma técnica que vem se popularizando nos últimos anos. Ela procura adaptar a dificuldade de um jogo em tempo real para se adequar à habilidade do jogador. Como [17] explica, fazer o ADD é um processo complexo e custoso, e por estes motivos não têm sido aplicada a todos tipos os jogos. Essa abordagem foi desenvolvida para se opor a níveis de dificuldade pré-determinados, ou seja, para cada tipo de jogo a dificuldade pode ser alterada de um modo diferente, e portanto, e a coleta de dados e o algoritmo de ajuste empregados são diferentes.

Com a grande variedade de jogadores, os jogos com ajuste de dificuldade estáticos ou semi-adaptativos não conseguem se adequar a todos os perfis de pessoas. A não adequação à habilidade do jogador pode causar frustração ou tédio, sendo essas duas condições suficientes para tornar a experiência de jogo ruim, fazendo com que o jogador perca o interesse no jogo.

Em [17] é feito um ADD para o jogo *Half-life*. Neste tipo de jogo a medição de dificuldade tem relação direta com o inventário do jogador, pois caso o personagem tenha menos recurso do que o necessário para passar do desafio, o jogador passará por dificuldades. Nesse jogo existe um ciclo explorar, lutar, vencer e morrer. Para implementar o ADD o algoritmo calcula a probabilidade do jogador morrer e ajusta a quantidade de itens do jogador para criar uma certa sensação, que podem ser de conforto (dando uma quantidade de itens alta) ou desconforto (deixando o inventário no limite).

Para melhorar a experiência de jogo, diversas pesquisas vem sendo feitas na área de ajuste dinâmico de dificuldade. O objetivo final do ADD é adaptar em tempo real o nível do jogo para ser compatível com as habilidades que o jogador possui naquele momento. Em poucas palavras o ajuste dinâmico de dificuldade deve funcionar como uma malha de controle que monitora o jogador, monitora as variáveis do jogo e atua sobre as variáveis do jogo. A Figura 2.4 mostra um esquema de como deve funcionar o ADD.

Figura 2.4: Malha de controle geral do ADD (retirada de [3]).

Para implementar o ADD, segundo [2], geralmente existem três módulos:

- Um programa de monitoramento: responsável por salvar dados brutos do jogo (tais como, tempo para vencer a fase, número de mortes, quantidade de itens usados, etc.) e passar para o programa de análise. O programa de monitoramento também pode coletar dados sobre o jogador como nível de tensão e outros dados;
- Um programa de análise: responsável por receber dados do programa de monitoramento e computar os elementos do jogo que devem ser ajustados para se adequar ao jogador;
- Um programa de controle: responsável por receber dados do programa de análise e atuar nos elementos do jogo.

Ajustar a dificuldade consiste em alterar principalmente três elementos do jogo:

- Elementos do personagem (tais como, vida, força, velocidade, altura do pulo, etc);

- Elementos do NPC (tais como, inteligência de um inimigo, número de inimigos, dano do NPC, dentre outros elementos que tem influência na dificuldade);
- Elementos do mundo do jogo (tais como, tamanho de buraco, distância entre plataformas, tamanho da platorma (Figura 2.5), etc.). Vale destacar que a alteração de parâmetros de mundo deve ser feita de maneira muito sútil, pois estes podem ser facilmente percebidos causando a sensação de trapaça pelo jogo [2].

Figura 2.5: Megaman 7 (retirada de [4]).

2.8 O ADD e a experiência do jogador

O designer do jogos deve projetar o jogo para que o jogador tenha a melhor experiência possível. Com o objetivo de maximizar a satisfação do jogador pode-se utilizar duas abordagens, a abordagem implícita e a abordagem explícita [18]. A abordagem implícita baseia-se em alterar parâmetros que tem relação indireta com a satisfação do jogador. Abordagens explícitas usam uma função de satisfação e buscam encontrar o ponto máximo dessa função. Neste trabalho a abordagem utilizada é a implícita, e portanto, considera-se que maximizar a satisfação do jogador está diretamente relacionado, de maneira não matemática, a levá-lo ao estado de imersão, ou fluxo. Isto implica em dizer que nenhuma alteração de parâmetro do jogo terá uma relação funcional com uma função que descreve matematicamente a satisfação do jogador.

*

Capítulo 3

Trabalhos Correlatos

Neste capítulo serão expostos os trabalhos relacionados ao tema abordado no presente trabalho. Serão apresentados alguns pontos principais sobre a condução dessas pesquisas e como este trabalho se diferencia dos demais.

Ao pesquisar sobre o tema, não foram encontradas pesquisas sobre ADDs para jogos *shooter* 2D. Este estudo é uma continuação do trabalho de [2], a única referência sobre o tema em questão. Logo, os trabalhos citados a seguir não possuem ligação direta com o ADD desenvolvido, mas possuem relação com o tema investigado.

3.1 Medindo dificuldade

A primeira coisa a ser feita para implementar um ADD em um jogo é medir e quantificar a dificuldade neste jogo. No trabalho descrito em [5], a dificuldade é medida em jogos de plataforma 2D. Neste tipo de jogo, a distância entre duas plataformas é um tipo de dificuldade a ser considerado. Para medir essa dificuldade foi criado um gerador de fases com plataformas com distâncias aleatórias.

Para fazer o cálculo de dificuldade em jogos de plataforma considera-se que o jogador sabe qual é o objetivo do jogo, ou seja, chegar ao final da fase, e sabe que tarefas triviais, como andar para frente ou subir escadas, não influenciam na dificuldade. A partir disso o autor expõe cinco situações comuns em jogos de plataforma, das quais, há dois casos especiais: o caso em que o jogador não possui chance de falha e o caso em que o jogador não possui chance de falha, mas também não possui change de completar o nível. A partir desses dois casos o autor define que o jogador pode falhar por dois motivos, levar dano e morrer ou ficar frustrado e desistir.

Para simplificar o estudo o autor do trabalho insere apenas um elemento (espinhos) que pode causar dano ao jogador e apenas um obstáculo para chegar ao final da fase, isto é, uma plataforma como ilustrado na figura 3.1. A partir dessa fase teste são definidas as seguintes probabilidades:

- P(j1): probabilidade de ter sucesso no pulo e terminar a fase;

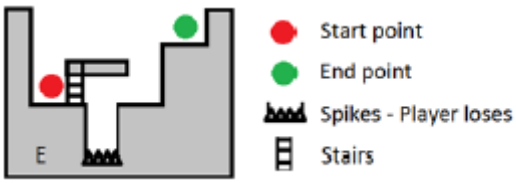

Figura 3.1: Fase de teste (retirado de: [5]).

- p(r): probabilidade de falhar e desistir do jogo.

Utilizando essas probabilidades, uma máquina de estados e equações de física cinemática, o autor define uma função de dificuldade e testa essas equações em jogos populares como: *Super Mario e Sonic*.

A equação 3.1 resume a probabilidade de sucesso do jogador, nesta equação P(j1') e P(r') são as probabilidades P(J1) e P(r) respectivamente dada a n-esima tentativa. Com essa equação o autor define a dificuldade do jogo utilizando a probabilidade do jogador obter sucesso. A máquina de estados é utilizada para representar o ciclo do jogador falhar e tentar de novo.

$$P(s) = P(j1) \sum_{i} (P(j1').P(r'))^{i} \tag{3.1}$$

3.2 O ajuste dinâmico em jogos FPS

Em [6] é descrito um ADD feito para o jogo *Half-life*. Para jogos estilo FPS *First Person Shooter* a dificuldade do jogo está diretamente relacionada à quantidade de estoque que jogador possui em determinado momento. O jogo é projetado para manipular constantemente a troca de recursos entre o jogador e o mundo. Essa relação de consumo e troca pode ser vista de como uma economia. Tradicionalmente o balanço entre consumo de inventário e fornecimento de itens era feito por *play test*, um sistema difícil e demorado de ser implementado, consequentemente custoso.

Para fazer o ADD, os autores desenvolveram uma ferramenta chamada *Hamlet*, que inclui as seguintes funções:

- Monitoramento de estatísticas do jogo pré-definidas;
- Definição de ações e políticas;
- Execução das ações e políticas;
- Painel para mostrar dados e configurações de controle do sistema;
- Geração de vestígios da sessão de jogo.

.

A ferramenta utiliza cálculos estatísticos vindos de dados do passado para estimar quando o jogador ficará sem estoque no futuro. A ferramenta fornece itens para o jogador quando prevê que o jogador ficará sem a quantidade necessária para passar do desafio. A Figura 3.2 mostra um gráfico que relaciona a quantidade de itens do jogador ao longo do tempo. A partir de gráficos como esse e da teoria do inventário foram feitos os cálculos para estimar a quantidade de itens do *player*.

A ferramenta *Hamlet* calcula a probabilidade do jogador ter uma falta de suprimentos utilizando a equação . Para isso foi utilizado a equação 3.2 sendo:

1. Ps a probabilidade do jogador ter falta de estoque;
2. X(t) a quantidade de dano recebida em determinado tempo;
3. Z a quantidade de estoque em determinado tempo;
4. F(Z) a função de distribuição Gaussiana de Z;
5. sigma a variância de F(Z);
6. mi a media de F(Z).

$$Ps = P(X(t) > Z) = 1 - P(X(t) < Z) = 1 - F(Z) = 1 - 1/(\sigma\sqrt{2\pi}) \int_{Z}^{\infty} \exp -(t-\mu)^2/2\sigma^2 dt \quad (3.2)$$

Quando o jogo detecta que vai haver falta de itens ele pode agir de duas maneiras, proativa e reativa.

Quando a ferramenta Hamlet age de maneira reativa o sistema pode tomar as seguintes atitudes: reduzir o dano causado pelos inimigos, diminuir a precisão dos inimigos, diminuir a potência das armas inimigas e reduzir a vida do inimigo.

A abordagem proativa consiste em alterar características de elementos que ainda não estão na tela como quantidade de itens de vida que serão dados ao jogador, quantidade de inimigos a nascer e precisão dos inimigos que vão aparecer.

3.3 Utilizando uma IA colaborativa para fazer o ADD

No trabalho desenvolvido por [19] está exposto que, normalmente, inteligências artificiais (IAs) são utilizadas apenas para alterar o comportamento dos inimigos. Em seu experimento ele desenvolveu uma IA que é capaz de melhorar o comportamento, tanto dos inimigos, quanto dos aliados.

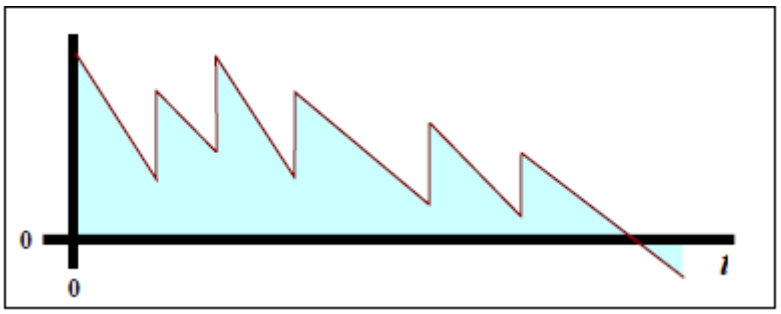

Figura 3.2: Estoque de itens do jogador pelo tempo (retirada de [6]).

Segundo o autor, a vantagem dessa abordagem é que ganha-se um eixo a mais de liberdade para alterar a dificuldade do jogo, pois pode-se tanto melhorar a inteligência dos aliados quanto a dos inimigos.

O autor também defende que melhorando a inteligência de unidades aliadas, o jogador tem uma melhor experiência de jogo interagindo com essas unidades. Nesse trabalho o autor deixa claro que a IAs dos aliados e dos inimigos devem ser desenvolvidas separadamente. Quando as IAs são desenvolvidas de forma separada é possível que dois cenários indesejados existam: a IA dos inimigos ser muito superior a dos aliados tornando o jogo virtualmente impossível para o jogador; a IA dos aliadados ser muito superior a dos inimigos tornando a participação do *player* irrelevante para o jogo.

O jogo para o qual a inteligências foi implementada é um jogo de simulação de presa e predador. O *player* é uma das presas e seu objetivo é sobreviver o máximo possível. O ADD é realizado de duas maneiras, fazendo os aliados do jogador melhores em sobreviver e fazendo os inimigos atacarem melhor, isso ocorre a cada iteração do jogo (as iterações ocorrem a cada 120 segundos).

3.4 O ADD usando sensor de resposta galvânica da pele

Assim como exposto em [7], muitos dispositivos podem ser usados para medir dados relacionados a afetividade. Independe do dispositivo utilizado para tal, sempre será necessário um processamento posterior dos dados. Devido à técnica de ajuste utilizando afetividade ser recente, não existem muitos softwares capazes de realizar essa tarefa de maneira padronizada.

Na implementação do trabalho apresentado em [7], o autor utiliza a engine Unity, para tornar o projeto multiplataforma, um Arduino e sensores para fazer a leitura da condutância da pele. Essa decisão foi baseada principalmente em baixo custo, tendo em vista que o conjunto completo custou cerca de $35 USD.

O jogo para o qual o ADD foi implementado é o *Eletroderma* (Figura 3.3), também criado pelo próprio autor do trabalho. Consiste em um *shooter* 2D no qual naves alienígenas tentam destruir a nave do jogador. Ela é destruída quando é tocada por uma nave alienígena. O jogo não possui história e dura o quanto o jogador sobreviver. Para marcar pontos deve-se destruir o maior número de naves alienígenas possíveis. Vale ressaltar que esse jogo, mesmo sendo parecido com

Figura 3.3: Jogo electroderma (retirada de [7]).

o jogo asteroides, não serve como parâmetro para dificuldade do Asteroids. Existe uma grande diferença entre as mecânicas de ambos. No Asteroids os objetos tem movimentos aleatórios, enquanto no Electroderma os movimentos dos inimigos são feitos para perseguir o personagem.

O ajuste da dificuldade foi feito utilizando as medidas dos EDAs e os algoritmos de processamento dos dados brutos para verificar se o jogador tinha um aumento ou uma diminuição na sua excitação. Esse cálculo foi feito utilizando apenas os dois últimos valores medidos da condutância da pele. Caso o segundo fosse maior que o primeiro era detectado um aumento na excitação, caso fosse menor assumia-se que houve uma redução na excitação e, por fim, caso a diferença fosse muito pequena era considerada que não houve mudança significativa na excitação do jogador. A partir dessa medida o jogo escolhe entre três dificuldades pré-definidas *easy, medium e hard*, não havendo garantia de que esse nível seja adequado para o jogador. Mesmo assim, como a mudança de dificuldade é automática, isto ainda é considerado um ADD.

O sistema realiza o ajuste baseado no estado afetivo do jogador, caso o algoritmo detecte indícios de frustração a dificuldade é reduzida, caso detecte indícios de tédio a dificuldade é aumentada. Uma abordagem similar a esta foi utilizada para realizar o ajuste no presente trabalho.

Capítulo 4

ADD Híbrido

Este capítulo se destina a apresentar a proposta deste trabalho, as ferramentas e equipamentos utilizados, bem como os desafios encontrados e soluções empregadas.

A principal contribuição deste trabalho é o desenvolvimento de um ADD híbrido para um jogo *shooter* 2D denominado Asteroids. Este ADD é denominado de híbrido, pois o ajuste dinâmico de dificuldade é baseado em dois tipos de variáveis, variáveis do jogo e dados fisiológicos do jogador. Os dados do jogo são capturados por algoritmos inseridos no código do próprio jogo, enquanto os dados do jogador são capturados por sensores de *biofeedback*. A captura de dois tipos de dados de natureza distintas tem como objetivo tornar o ADD mais robusto, ou seja, a utilização de um dados fisiológicos fornece subsídios extras, além dos dados do jogo, para determinar o ajustes a serem feitos no nível de dificuldade do jogo.

4.1 O Asteroids

Na verdade, o jogo no qual o ADD foi implementado chama-se *Asteroids: in the 2nd and 1/2th Dimension"*. Criado por Matthew Renze, este jogo é uma adaptação do jogo Asteroids de 1979. Foi utilizada a versão adaptada, pois esta possui um objetivo claro para o jogador. Na versão original do jogo, por outro lado, em geral, o jogador apenas atira em objetos enquanto o tempo passa.

O jogo Asteroids de 1979 foi um dos maiores sucessos da era dos arcades [20]. O objetivo, que não é claro, até porque o jogo não pode ser vencido, é sobreviver o máximo possível, evitando a colisão com asteroides, tiros ou outras naves. A nave pode se mover rotacionando, acelerando e desacelerando. Além disso, a única coisa que a nave pode fazer é atirar. Nesta versão do jogo, destruir naves inimigas e asteroides adicionam pontos.

O jogo *Asteroids: in the 2nd and 1/2th Dimension* é um pouco mais complexo, nele o objetivo é encontrar a pilula azul no campo de asteroides. Isso possibilita que o jogador vença o jogo, diferente da versão original em que eventualmente o jogador perderia. Neste trabalho, ao jogo *Asteroids: in the 2nd and 1/2th Dimension* de Matthew Renze, foram adicionados os seguintes elementos: uma

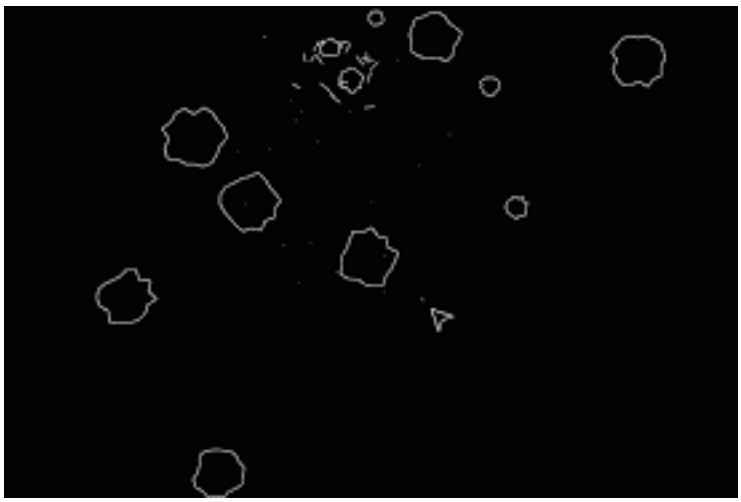

Figura 4.1: Jogo asteroides clássico (retirada de [8]).

pílula para dar um escudo, asteroides explosivos, asteroides indestrutíveis e asteroides com escudo. O jogo final possui 6 fases, em cada fase um elemento novo é adicionado ao jogo, mas em todas as fases o objetivo é sair do campo de asteroides. A distância do final do campo de asteroides fica no canto direito inferior da tela. Além disso, por fim, foi adicionado ao jogo a possibilidade de jogar em 3D. Entretanto, o jogo vem com a câmera vista de cima que faz com que o jogo fique em 2D. Outras três câmeras ortogonais também estão disponíveis. A Figura 4.2 mostra um *screenshot* do jogo.

Por questões de simplicidade, a partir daqui, o jogo *Asteroids: in the 2nd and 1/2th Dimension* adaptado neste trabalho, será referido apenas como Asteroids.

4.2 Quantificando a dificuldade para o jogo Asteroids

A primeira implementação feita neste trabalho foi a qualificação e quantificação da dificuldade em Asteroids. Nesta versão do jogo, as seguintes variáveis foram identificadas como as que podem interferir na dificuldade:

- Tamanho dos asteroides;
- Velocidade dos asteroides;
- Densidade de asteroides na tela;
- Velocidade da nave;
- Velocidade do tiro;
- Velocidade de rotação;
- *Cooldown dos tiros.*

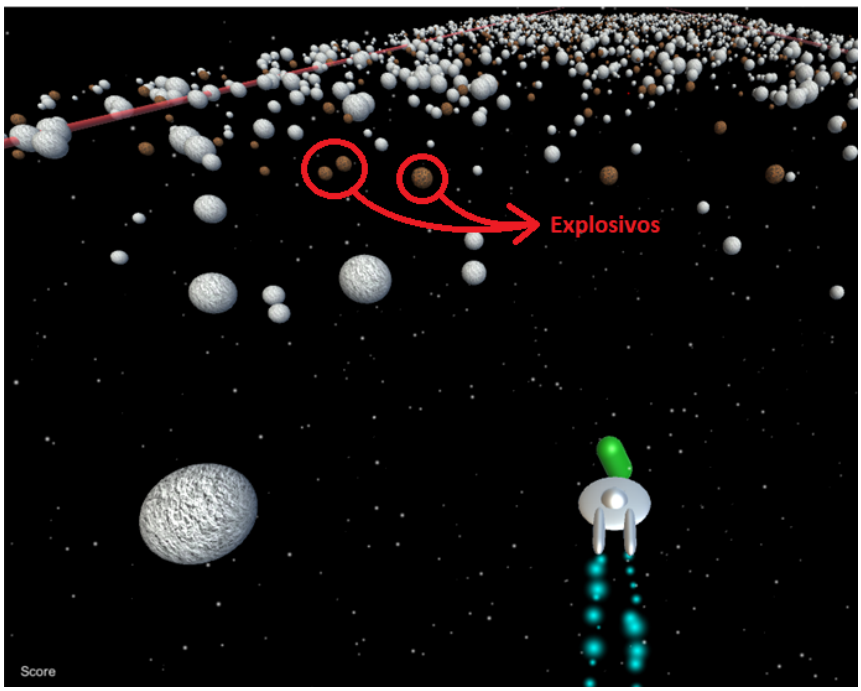

Figura 4.2: Fase número quatro do jogo com *prefabs* dos asteroides instanciados.

Neste trabalho considerou-se a variação da velocidade e densidade dos asteroides, por supor que essas teriam maior influência. Experimentos foram realizados para identificar quais dessas variáveis teriam maior ou menor influência no nível de dificuldade.

Para realizar o experimento foi desenvolvida uma versão simplificada do jogo com uma fase contendo apenas asteroides regulares (isto é, sem asteroides explosivos, gigantes, indestrutíveis ou blindados). Esta versão simplificada possui um menu de configuração, como mostrado na Figura 4.3 para escolher o tipo de variável que será controlada (velocidade e/ou densidade dos asteroides) e uma tela de instruções do jogo (Figura 4.4).

Não há níveis predefinidos de dificuldade, basicamente há um aumento nas variáveis predefinidas à medida que o tempo avança. Além disso, a cada dois minutos, a uma dada velocidade e densidade, o jogador deve avaliar em uma escala de 1 a 10 seu nível de percepção de dificuldade.

No final do experimento, espera-se que os dados revelem valores confiáveis da correlação entre o aumento de uma ou duas variáveis e a percepção de dificuldade relatada pelo jogador. Vale destacar que espera-se com este experimento apenas saber o quanto a variação dessas grandezas se relacionam, visando propor uma função para o ajuste dinâmico de dificuldade. Para poder dizer o nível exato de dificuldade, dado uma certa velocidade e uma certa densidade, seria necessário que o jogador desse notas de 1 a 10 de maneira contínua. Portanto, não será obtida uma medida exata do que são os níveis *easy, normal e hard*, mas sim uma indicação de como aumentar ou diminuir a dificuldade de modo que ela seja adequada ao jogador.

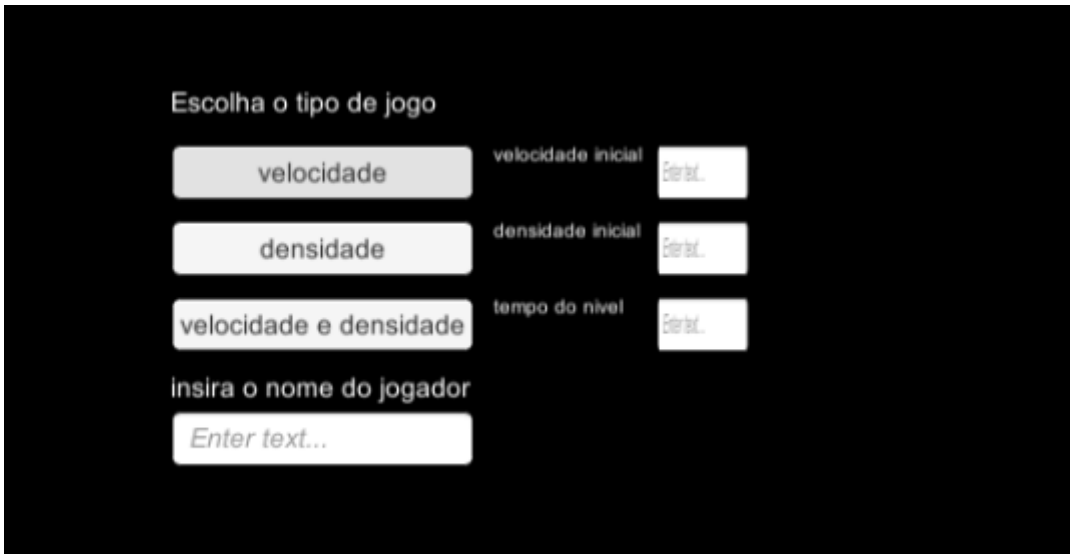

Figura 4.3: Tela de configuração do experimento.

Figura 4.4: Tela de apresentação.

4.3 Unity 3D

A Unity[1] é uma *game engine* para desenvolvimento de jogos. *Engines* são softwares que criam abstrações para o desenvolvedor e, em geral, provêm para o funções para renderização de imagens, simulações físicas e suporte a animações e sons.

Antes das *engines* comerciais e o desenvolvimento de jogos era feito por meio do uso de bibliotecas gráficas, isto é, o programador era responsável por desenvolver toda a *engine*, ou código base, do jogo. Para cada jogo desenvolvia-se uma *engine* específica, e o programador decidia quais funções sua engine teria ou não. A Unity 3D (atualmente chamada apenas de Unity) é uma *engine* possui funções prontas para uso de renderização de imagens, áudio, física de corpos rígidos, colisões e animação de imagens.

Além disso, a Unity permite o desenvolvimento para múltiplas plataformas, possuindo em sua opção de *build* as seguintes plataformas computacionais:

- Windows 32 e 64 bits;
- Unix 32 e 64;
- Mac OS.

Além destas plataformas computacionais, a Unity também pode compilar para outras plataformas como mostra a Figura 4.5. No âmbito deste trabalho, esta possibilidade é útil para desenvolver uma versão tanto para Linux quanto para Windows, e futuramente o desenvolvimento para *mobile*.

A Unity permite códigos em CSharp (CS) e JavaScript (JS). Para desenvolver a ferramenta de coleta de dados sobre o grau de dificuldade e o jogo Asteroids com o ADD híbrido foi utilizado apenas a linguagem CSharp. Seu uso se justifica pelos seguintes motivos:

- A comunidade de CS é maior do que de JS. Logo, dúvidas e problemas são mais facilmente encontrados na internet para CS;
- A versão do jogo feita por Matthew Renze também foi feita em CS;
- Os desenvolvedores da Unity anunciaram o fim do suporte a JS.

Outra capacidade da Unity muito utilizada neste trabalho foi o *prefab*, que é um objeto (diferente dos objetos de programação orientada a objetos), que guarda as informações de determinada entidade do jogo para ser reutilizada em outro momento. Um *prefab* muito comum em jogos Unity é o *player*, isto é, uma vez criado o player no jogo os desenvolvedores criam o *prefab* dele e apenas o copiam em todas as fases.

A Unity torna a edição de variáveis do jogo trivial por meio do editor para membros (*inspector*) *public*. Todos os atributos de classe declarados como *public* na Unity ficam disponíveis para edição

[1] Site da Unity: https://unity3d.com/p-t

Figura 4.5: Opções de build da Unity.

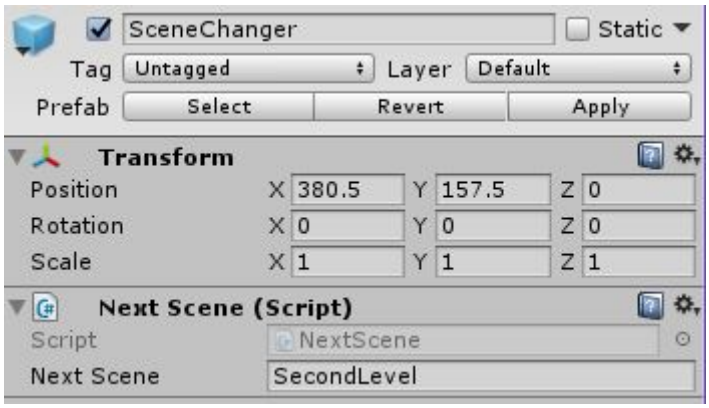

Figura 4.6: Membro público na Unity, no caso o nome da próxima fase a ser carregada.

no Unity Editor. Por exemplo, para testar a velocidade da nave, a velocidade dos asteroides e/ou o tamanho dos asteroides basta mudar o valor no editor. A Figura 4.6 mostra um exemplo de uso de um membro público no Unity.

Pode-se dizer que a Unity 3D facilita o desenvolvimento do projeto, fazendo com que problemas de programação sejam resolvidos facilmente e permite ao desenvolvedor focar no problema do jogo em questão. A escolha da Unity 3D se justifica pelos motivos já descritos e também pela sua grande quantidade de API's, incluindo uma específica para o sensor de coleta de dados fisiológicos Bitalino.

```
private void LevelPass(int level,int Difficulty){
    if (DDALearn) {
        NormalLeverSelecter (level);
    return;
    }

    if (Difficulty == 0)
        SceneManager.LoadScene("level "+ level.ToString()+ " easy",LoadSceneMode.Single);
    else if (Difficulty == 2)
        SceneManager.LoadScene("level "+ level.ToString()+ " hard",LoadSceneMode.Single);
    else
        SceneManager.LoadScene("level "+ level.ToString()+ " normal",LoadSceneMode.Single);

}
```

Figura 4.7: Código não refatorado.

4.4 Implementação do Asteroids

A implementação do Asteroids é a continuação do trabalho de [2]. Em sua implementação foram feitas 6 fases para o jogo, porém, para cada fase existiam 3 possibilidades de dificuldade: *easy, medium e hard*. Para implementar o ADD os autores não possuíam conhecimento de como as variáveis do jogo se relacionavam com a dificuldade, logo, toda a mudança de dificuldade era feita sem essa base, considerada essencial. Outro ponto do ADD de Aguiar et al. é que ele recai sobre o problema de dificuldade estática, pois caso nenhuma das três dificuldades oferecidas pelo jogo seja adequada para o jogador, o mesmo ficará frustrado.

Uma deficiência deste ADD foi a falta do uso dos paradigmas da Unity por parte autores. As técnicas de programação utilizadas para desenvolver o jogo e o ADD não utilizavam facilidades do Unity (utilizavam paradigmas de C++), dessa forma muitos códigos encontrados eram desnecessários e outros completamente fora do padrão de boas práticas do Unity. Portanto, a primeira tarefa realizada para possibilitar a continuidade deste trabalho, foi a refatoração do código, detalhada a seguir.

4.4.1 Refatoração de código

O primeiro passo na implementação deste trabalho foi refatorar os códigos do trabalho anterior. Refatorar um código significa alterá-lo para alcançar alguma melhoria, mas mantendo a funcionalidade. Um exemplo de refatoração realizada neste trabalho pode ser observado na Figura 4.7 que mostra o código anterior e na Figura 4.8 que mostra o código refatorado. Os dois códigos trocam de fase, mas o segundo utiliza uma variável pública da Unity, o uso da variável pública permite a alteração do valor da variável no inspector.

A refatoração de código é muito importante para projetos de longo prazo, códigos refatorados são melhores de ler e modificar. Quanto menos o código for refatorado, menor será a produtividade ao longo do tempo, assintoticamente chegando a zero (Figura 4.9).

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class NextScene : MonoBehaviour {

    public string nextScene;

    public void ChangeScene(){
        SceneManager.LoadScene (nextScene);
    }

}
```

Figura 4.8: Código refatorado.

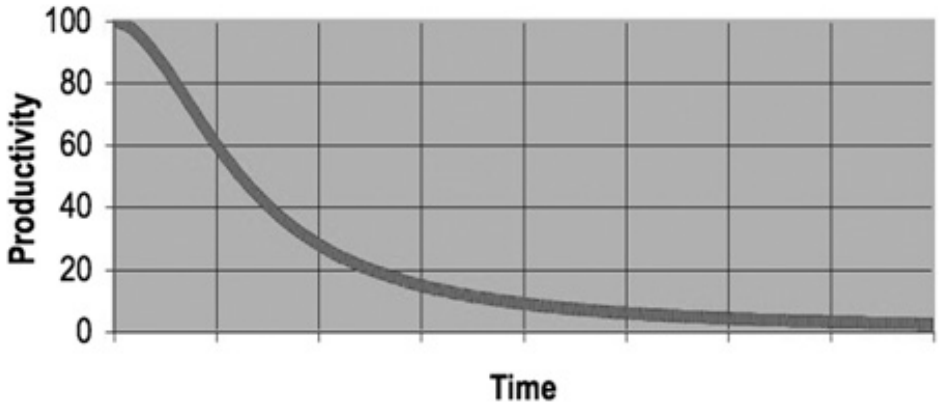

Figura 4.9: Produtividade ao longo do tempo (retirada de [9]).

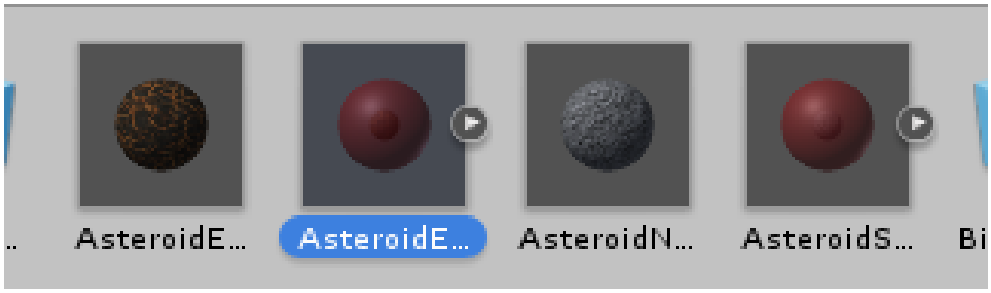

Figura 4.10: Prefabs dos asteroides usados no jogo

```
void Start()
{
    bitalino = new Process();
    bitalino.StartInfo.WindowStyle = ProcessWindowStyle.Hidden;/*dont open window*/
    bitalino.StartInfo.CreateNoWindow = true;/*do not show window*/
    bitalino.StartInfo.UseShellExecute = true;
    bitalino.StartInfo.FileName = Application.dataPath + "\\BitalinoReader.exe";
```

Figura 4.11: Código que roda o executável de adquirir dados fisiológicos

4.4.2 Utilizando *prefabs* para fazer o jogo

O jogo Asteroids é feito de basicamente dois tipos de elementos, a nave do jogador e os asteroides. Em sua primeira fase, apenas asteroides simples são criados, dessa forma a maneira mais fácil de implementar esse jogo foi utilizando *prefabs*. Como já mencionado, basta criar um objeto é replicá-lo quando necessário. Todos os objetos do jogo foram transformados em *prefabs*, então o código do jogo em sua maioria apenas instancia *prefabs*. A Figura 4.10 mostra os *prefabs* dos asteroides. Como todos os tipos de asteroides são *prefabs*, o algoritmo de criação de asteroides apenas instancia cada um desses objetos em posições aleatórias. A Figura 4.2 ilustra o resultado deste algoritmo.

4.4.3 Implementação da aquisição de dados fisiológicos

O código que faz a acquisição de dados fisiológicos foi feito na linguagem C++. Por outro lado, como já mencionado, o código do jogo Asteroids e do ADD foi feito em CS, portanto, para fazer os dois executarem ao mesmo tempo foi utilizado um script em CS (Figura 4.11). Para tornar o executável transparente, a linha Bitalino.StartInfo.CreateNoWindow foi usada. Essa linha faz o processo ficar escondido do usuário.

O código em C++ escreve os dados em um arquivo com extensão .txt. A leitura desse arquivo é feita toda vez que o jogador morre; neste intervalo de tempo o programa é interrompido, pois não é possível ler e escrever em um arquivo ao mesmo tempo. Após ler os dados do arquivo .txt, os dados são processados, para serem convertidos em valores físicos. A partir dos valores físicos, o jogo computa o estado afetivo do jogador.

Sabendo que o aumento na condutância da pele é um indicativo de aumento de *stress*, o

algoritmo computa a média dos valores medidos e quantos desses valores são maiores que a média. Caso o número de valores acima da média seja mais do que a metade, entende-se que o jogador está frustrado, caso seja menor que a média, entende-se que o jogador está entediado, caso seja igual entende-se que o jogador esta em seu estado normal. Cálculo equivalente foi feito para se extrair o estado afetivo da tensão gerada pelo coração (ECG).

4.5 O *BITalino*

Para fazer a acquisição de dados fisiológicos utilizou-se a placa *BITalino* [21] 4.12, um dispositivo de baixo custo que possui os seguintes sensores:

- Atividade Eletrodermal (EDA);
- Eletrocardiograma (ECG);
- Acelerомêtro;
- Eletromiografia (EMG);
- Medidor de luz.

Outro motivo para utilizar esta placa é que existem muitas API's (*Application Programming Interface*) para ela. No site do fabricante estão disponíveis as API's para as seguintes linguagens: C++, CSharp, Python, Java, MATLAB e Unity.

Embora o fabricante da placa ofereça a API em Unity, a mesma se mostrou bastante falha e difícil de usar. A implementação do programa de coleta de dados do usuário foi feita em C++, pois a API do C++ vem com um programa que mede todos os sinais possíveis do Bitalino e os coloca em um arquivo .txt. Além disso ela vem com instrução de compilação no *Visual Studio*.

Um problema encontrado nas medidas do *BITalino* é que elas são coletadas em valores discretos, ou seja, valores inteiros espaçados igualmente, na faixa de 0 a 1023 (representado por 10 bits). Portanto, é preciso um processamento dos dados para extrair os valores desejados. Felizmente, o site do fabricante fornece o *datasheet* de cada sensor disponível na placa. Assim, para transformar a medida do sensor de EDA em indicadores de condutância da pele foram utilizadas as seguintes equações:

$$EDA(\mu S) = ((ADC/2^n) * VCC)/0.123, \tag{4.1}$$

$$EDA(S) = EDA(\mu S) * 1 * 10^{-6}, \tag{4.2}$$

no qual:

- ADC é o valor medido pelo sensor;

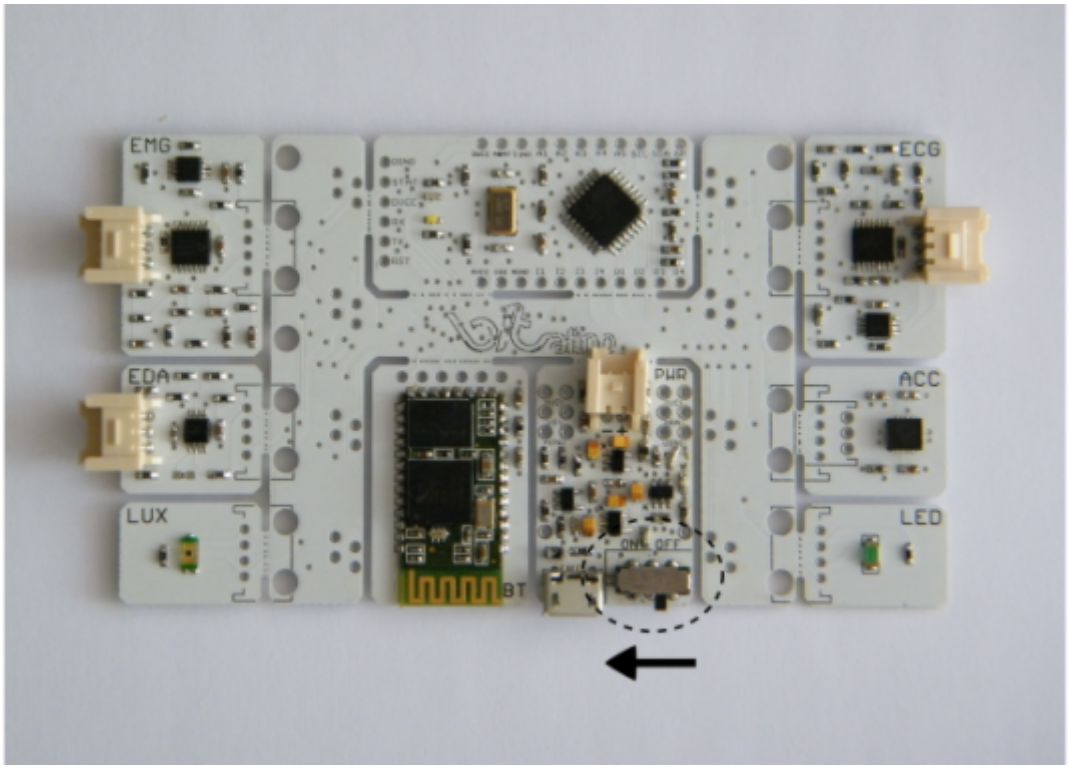

Figura 4.12: Placa BITalino com destaque para o botão de ligar (retirada de [2]).

- VCC tem valor constante 3.3V;
- n é o número de bits utilizado pelo canal (10 no caso do EDA);
- EDA(S) é o valor da condutância em Siemens.

Para transformar a medida do sensor de ECG em tensão foram utilizadas as seguintes equações:

$$ECG(V) = ((ADC/2^n - 1/2) * VCC)/Gecg, \tag{4.3}$$

$$ECG(mV) = ECG(V) * 1000, \tag{4.4}$$

No qual:

- ADC é o valor medido pelo sensor;
- VCC tem valor constante 3.3V;
- n é o número de bits utilizado pelo canal (10 no caso do ECG);
- ECG(V) é o valor em volts;
- Gecg é o ganho do sensor (1100).

O uso dessas equações serve para converter o sinal medido em valores que representam uma medida fisiológica. Como dito anteriormente, a medida do sensor vem apenas como um número digital. Esse número digital por si não possuí significado. A partir do valor digital da medida as equações determinam os valores de interesse: condutância da pele e tensão elétrica gerada pelo coração.

4.6 Proposta de ajuste dinâmico de dificuldade

A partir da quantificação de dificuldade percebeu-se que a velocidade tem relação direta com a dificuldade , enquanto a densidade não possui correlação com a dificuldade como é descrito na sessão 5.1. A proposta desse ajuste é funcionar como uma malha de controle proporcional, medindo o desempenho do jogador e seus sinais fisiológicos, a partir deses dados, ajustar a dificuldade de maneira linear.

Nesta proposta, o ajuste de dificuldade é feito em dois momentos do jogo, quando o jogador passar de fase e quando o jogador morrer. Como o intervalo de tempo entre um ajuste e outro é pequeno, pode-se dizer que o jogo ajusta a dificuldade em blocos definidos. A medida de desempenho do jogador é o número de mortes, ou seja, quanto maior este número, pior será o desempenho. Além disso, serão utilizadas duas variáveis fisiológicas para inferir o estado afetivo do jogador, a resposta galvânica da pele e o eletrocardiograma. Para simplificar o uso e interpretação das variáveis fisiológicas foram definidos apenas três estados emocionais: normal, entediado e frustrado.

Somente quando o jogador passar de fase o seu estado afetivo será computado, portanto, os ajustes que ocorrerem quando o *player* morrer durante a fase serão sempre baseados apenas em seu desempenho momentâneo. Se o jogador morrer infere-se que desempenho não foi bom, então deve-se corrigir a dificuldade (mesmo que minimamente) a cada morte. Ao passar de fase é calculada uma medida, baseando-se no número total de mortes do jogador e os dados fisiológicos, para saber como deve ser feito o ajuste para a próxima fase. A ideia principal é fazer um ajuste maior cada vez que o jogador passar de fase e pequenos ajustes toda vez que ele morrer.

O ajuste híbrido é uma expansão do ajuste baseado apenas em dados, isto é, além dos dados de desempenho, considera os dados fisiológicos do jogador. Caso o sensor seja, por algum motivo, desconectado ou simplesmente nunca tenha conseguido estabelecer conexão, o ajuste acontecerá considerando que o jogador está em um estado afetivo normal. O ajuste híbrido proposto neste trabalho considera que o jogador pode estar em três estados afetivos:

- Entediado (*BORED*);
- Normal (*NORMAL*);
- Frustrado (*STRESSED*).

Também estabeleceu-se que o estado afetivo do jogador prevalece sobre o seu desempenho no momento de realizar as mudanças no jogo, independente de quantas vezes o jogador morreu no

nível. Caso os sinais fisiológicos coletados indiquem que o jogador está frustrado a dificuldade do jogo será diminuída. Por outro lado, se estes sinais indicarem um estado de tédio, é aplicado um aumento na dificuldade. Por fim, na situação do sensor apontar um estado afetivo normal o ajuste é feito baseado apenas no desempenho do jogador.

Capítulo 5

Testes

Neste capítulo são apresentados os experimentos conduzidos, os resultados obtidos, bem como o relato de alguns desafios enfrentados durante a coleta de dados.

5.1 Qualificação e quantificação da dificuldade

A primeira etapa deste trabalho foi destinada a qualificar e quantificar o que é dificuldade para o jogo proposto. Foi criada uma *build* de teste do jogo, que consiste em uma versão reduzida para realizar este primeiro experimento. Essa *build* consiste de uma fase contendo somente asteroides normais, ou seja, não possui asteroides explosivos, gigantes, indestrutíveis ou com escudo.

Para realizar a quantificação de dificuldade foi projetado um experimento com os seguintes passos: (i) convidar pessoas voluntárias por conveniência; (ii) solicitar ao jogador voluntário para jogar o jogo e avaliar a dificuldade do mesmo a cada 2 minutos (120 segundos); e (iii) encerrar o experimento quando o jogador avaliar que o jogo se tornou frustrante. O jogador foi instruído a dar nota 10 para a dificuldade quando o jogo se tornava frustrante, nesse momento o experimento era terminado. A ideia é de que quando o jogador der nota 10 para a dificuldade seria o momento em que o jogador desistiria do jogo numa sessão de jogo real.

Foram realizados experimentos para quantificar a influência da velocidade dos asteroides na dificuldade, a influência da densidade de asteroides e por fim a da combinação da velocidade com a densidade. No experimento de quantificação da velocidade a cada 2 minutos a velocidade era aumentada em 0.2 unidades da Game Engine Unity. Além disso eram salvas as seguintes informações: número de mortes do jogador, se venceu ou não o nível, a velocidade atual, a densidade atual e o tempo decorrido para completar o nível quando este era completado. Os experimentos de densidade e da combinação de densidade e velocidade foram conduzidos de mesma maneira.

O experimento foi realizado com 20 voluntários. Esse número foi escolhido por questões de tempo e escopo do trabalho. As Figuras 5.3, 5.2 e 5.1 fornecem um resumo dos dados coletados. A partir desses gráficos e cálculos de correlação pode-se inferir que a velocidade possui uma

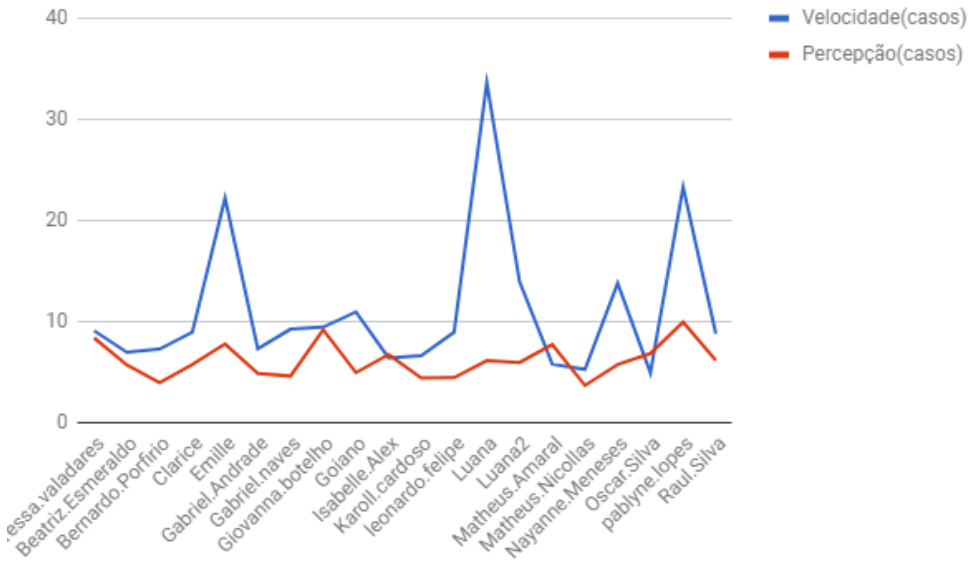

Figura 5.1: Gráfico descritivo relacionando a velocidade a dificuldade.

relação próxima a uma reta com a dificuldade, a maior evidencia disto é a correlação de 0.705 entre aumento da velocidade é a percepção de dificuldade. Já a densidade não possui correlação significativa com a dificuldade (correlação de 0.21). A Tabela 5.1 mostra o resumo dos dados coletados no experimento de qualificação da velocidade.

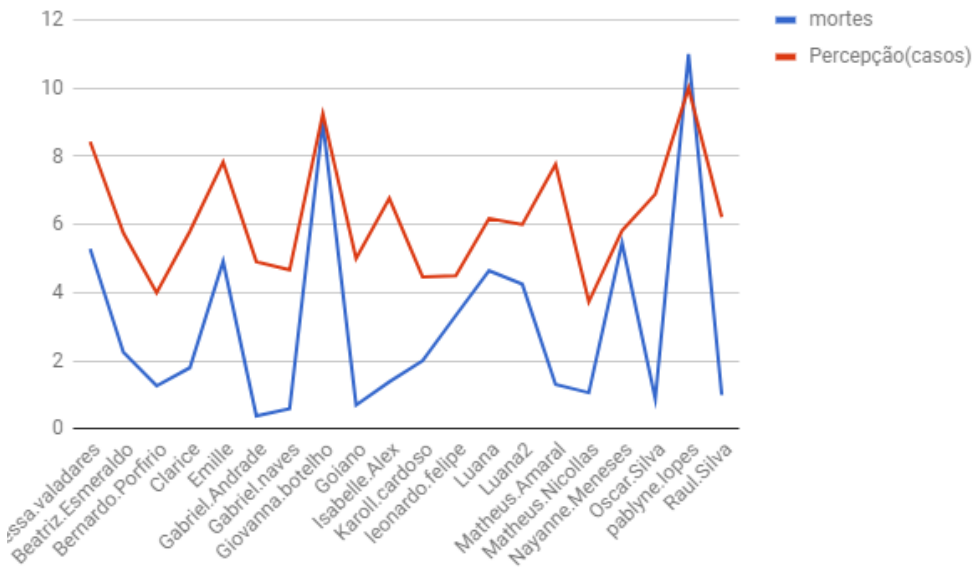

Figura 5.2: Gráfico descritivo relacionando o número de mortes a dificuldade.

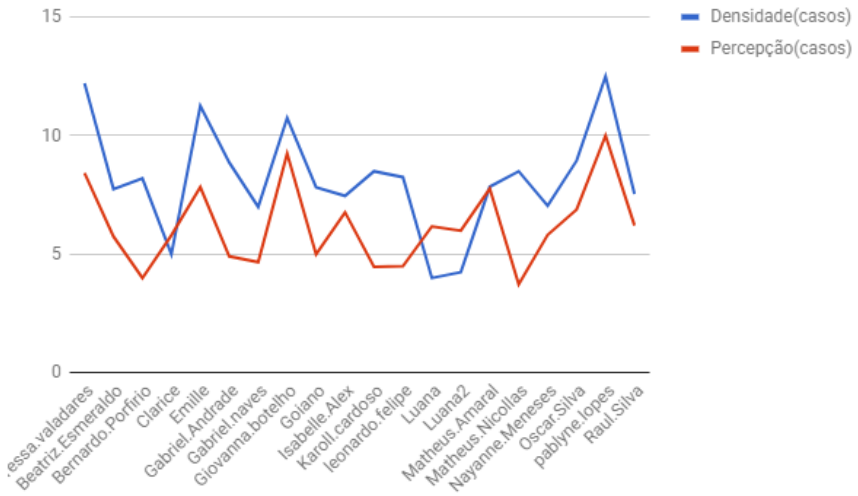

Figura 5.3: Gráfico descritivo relacionando a densidade a dificuldade.

Tabela 5.1: Resultados dos participantes, resumo.

Nome	tempo de jogo (s)	Velocidade Máxima	Total de mortes	densidade
Bernardo.Porfirio	628.27	1.4	5	700
Gabriel.Andrade	947.63	2.2	9	700
Gabriel.naves	1026.92	2.2	9	700
Goiano	816.65	2	5	700
Karoll.Cardoso	600	1.2	17	700
Matheus.Nicollas	290.38	1.2	7	700
Nayanne.Meneses	1096.65	2.2	51	700
Raul.Silva	443.2	1.8	9	700
Ana.Paula	521	1.2	8	700
Felipe.Soares	444	1	9	700
Maria.Lucia	564	1.2	7	700

5.2 A implemetação do ADD

O ADD para o jogo Asteroids, anterior ao presente trabalho, implementado por Aguiar et al. [2] foi feito de maneira estática, dado um certo desempenho do jogador. Ou seja, o jogo escolhe entre três velocidades e densidades para os asteroides, em outras palavras, este ADD é apenas a mudança automática entre *easy, normal* e *hard*. Esse tipo de ajuste tem o mesmo tipo de problema das dificuldades estáticas, quando um dos três níveis não é adequado ao jogador o ajuste não atinge sua função. O ADD de Aguiar et al. ajusta a dificuldade apenas quando o jogador passa de fase, e quando o jogador não consegue passar de fase o ajuste dinâmico de dificuldade não ocorre. Portanto, é possível que o jogador fique frustrado e nunca tenha a dificuldade corrigida ou ajustada.

A implementação do ADD proposto neste trabalho foi feita baseada na hipótese de que os experimentos iniciais para quantificação da dificuldade estavam corretos, logo, a velocidade tem relação linear com a dificuldade e a densidade não tem correlação significativa. A proposta é fazer com que a velocidade e a densidade se ajustem de acordo com o desempenho de cada jogador, fazendo com que independente do jogador a dificuldade seja sempre adequada. O ADD é realizado em dois momentos, quando o jogador morre, e quando o jogador passa de fase; isso foi feito para que o jogador não fique preso em um nível. Acredita-se que o jogador não passar da primeira fase seria uma falha fatal no algoritmo, portanto, mesmo que a dificuldade inicial seja muito pequena ainda é possível que ADD aja no sentido de diminuí-la quando o jogador tem baixo desempenho.

O desempenho do jogador, desconsiderando o estado afetivo, é quantificado da seguinte maneria: baixo para mais de cinco mortes, mediano entre duas e cinco mortes e bom para menos de duas mortes para completar a fase. Para baixos desempenhos a dificuldade é diminuída proporcionalmente ao número de mortes, para desempenhos medianos a dificuldade é aumentada proporcionalmente ao número de mortes (sempre será um pequeno aumento, pois o número de mortes estará entre 0 e 5) e para um alto desempenho a dificuldade aumenta dando um grande

aumento na velocidade e na densidade dos asteroides.

O ADD híbrido, além do desempenho do jogador, usa o estado afetivo para aumentar ou diminuir a dificuldade. Caso esteja no estado *STRESSED* a velocidade é diminuída proporcionalmente ao número de mortes (se com muitas mortes, pouco a diminui, se com poucas mortes, muito a diminui). Caso esteja no estado *BORED* a velocidade é aumentada proporcionalmente ao número de mortes (se com muitas mortes, muito diminui, se com poucas mortes, pouco a diminui). Caso esteja no estado *NORMAL* o ajuste é realizado apenas com base no desempenho (número de mortes).

Pretende-se comparar o ajuste híbrido com o ajuste baseado somente no desempenho.

5.3 A quantificação do *flow* para o ADD baseado apenas nos dados de desempenho

Após a realização dos testes de quantificação de dificuldade foram conduzidos os experimentos de quantificação de flow com o ajuste de dificuldade baseado somente nos dados do jogo. O experimento consistiu dos seguintes passos: (i) realizar uma sessão de 15 minutos com cada voluntário, sem mencionar à pessoa que a dificuldade do jogo Asteroids seria alterada; (ii) ao final dos 15 minutos o jogador deve responder um questionário (link no rodapé) projetado para avaliar se o jogador entrou ou não no estado de fluxo (*flow*) [1] .

Todos os jogadores, exceto um, chegaram ao nível 4 dos 6 níveis do jogo. Acredita-se que isso aponte um balanceamento adequado do jogo. Lembrando que no estado de *flow* a atenção do jogador esta completamente focada na tarefa e que o jogador sente que pode superar o desafio. O *score* no questionário quantifica o quanto o jogador entrou no estado do fluxo, o *score* é calculado fazendo média de todas as questões. O resultado deste *score* foi de 3.95. As Figuras 5.4 e 5.5 são exemplos de perguntas do questionário.

Além dos gráficos indicados acima, outros foram gerados, porém muitos possuem dados ou até perguntas redundantes e por isso foram omitidos. Mesmo tentando ser aleatório ao abordar as pessoas vale ressaltar que dentre os 20 voluntários, 17 são homens e apenas 3 são mulheres.

[1] https://docs.google.com/forms/d/1B9Ah4-balMA-AHtH3Idt2QB5kU1dS77qCOGeDxVV6CY/edit?usp=drive$_o penths = true$

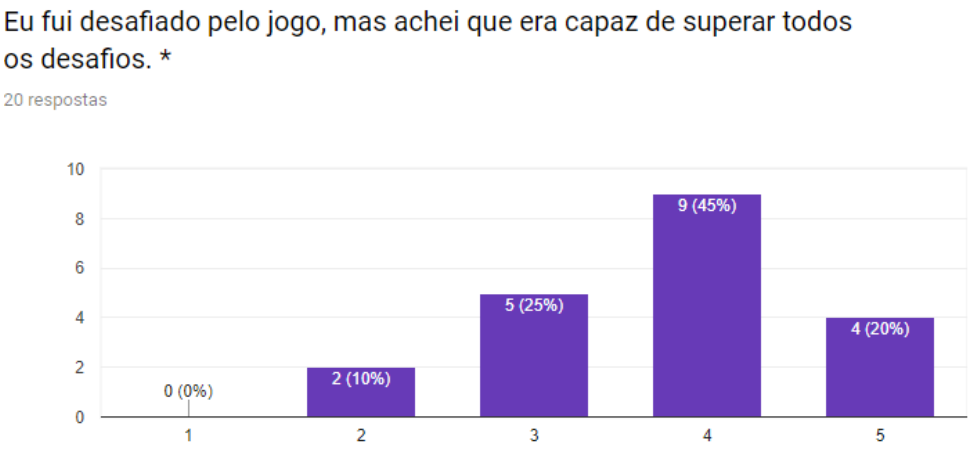

Figura 5.4: ADD baseado apenas em dados do jogo - capacidade de superar o desafio em uma escala de 1 a 5

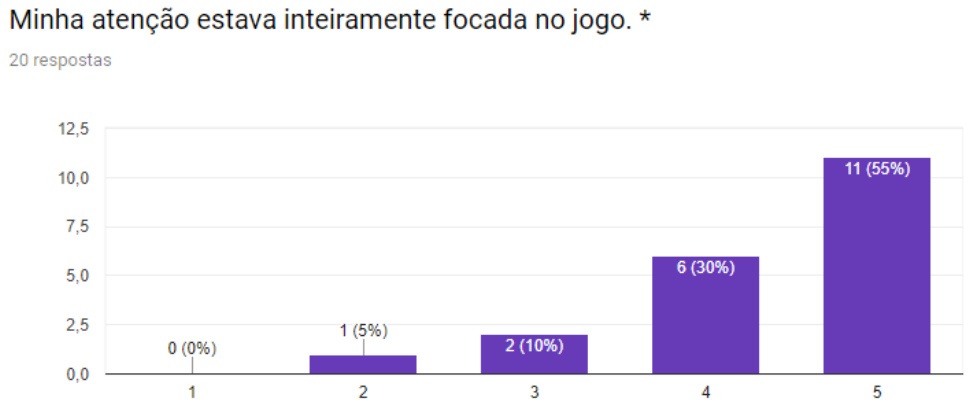

Figura 5.5: ADD baseado apenas em dados do jogo - atenção do jogador em uma escala de 1 a 5

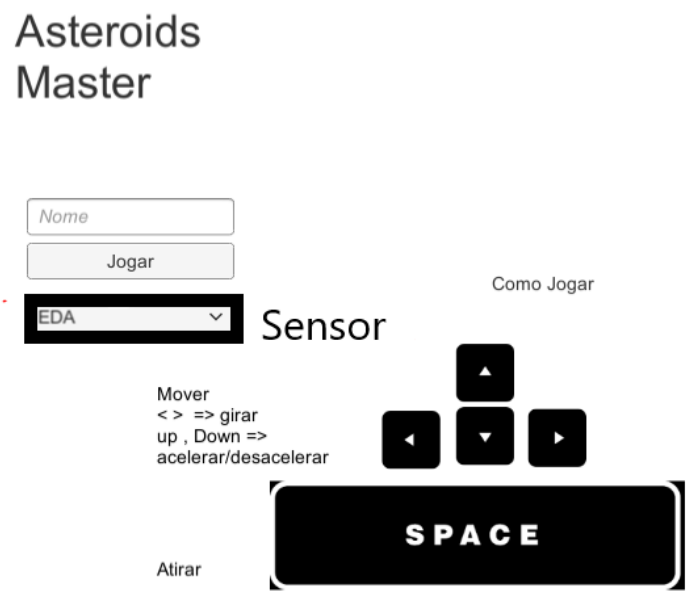

Figura 5.6: Tela de apresentação versão final do jogo

5.4 ADD utilizando dados fisiológicos obtidos por meio de sensores de *biofeedback*

O ajuste de dificuldade baseado nos dados fisiológicos do jogador foi feito utilizando a placa *BITalino* e o executável gerado no Visual Studio. O executável é exclusivo para Windows, pois o Visual Studio não possui nativamente as bibliotecas necessárias para gerar executáveis Unix. Por esse motivo a versão final do jogo com o uso de sensores está disponível apenas para Windows.

A ideia inicial era implementar o uso dos dois sensores (EDA e EMG) ao mesmo tempo, porém, devido a questões práticas como a colocação dos sensores no participante isso se tornou inviável. Então os sensores foram implementados de maneira separada, e o sensor a ser utilizado é escolhido na tela de apresentação do jogo em sua versão final. Essa *feature*, por outro lado, permite facilmente mudar a medida a ser feita no jogo. A Figura 5.6 mostra como isso pode ser feito, bastando abrir a lista *dropdown* e escolher um dos sensores. O executável que mede as saídas do Bitalino gera um arquivo em formato .txt com todos os seus sensores, logo para utilizar um sensor diferente basta olhar o *datasheet* do sensor, fornecido no site do fabricante, e fazer o cálculo para converter a saída para a unidade apropriada. A leitura do arquivo e a separação de cada medida de cada sensor já esta implementada no código.

Para fazer a comparação com o ADD baseado somente nos dados do jogo seria necessário realizar o experimento com o sensor EDA com 20 pessoas e em seguida com mais 20 pessoas utilizando o sensor de ECG. Essa é uma grande quantidade de pessoas e, devido a questões de tempo, o experimento não seguiu dessa forma. Vale deixar claro que esse experimento é particularmente problemático, devido ao tamanho dos cabos e a sudorese excessiva de alguns participantes. Algumas vezes o sensor era desconectado e isso tornava a aplicação do experimento frustrante. Outro

problema era o adaptador *Bluetooth*, pois muitas vezes o sinal era perdido no meio do experimento e os dados ficavam pela metade; em geral esse problema ocorria quando o jogador fazia movimentos bruscos, como levar as mãos a cabeça ou bater na mesa, por isso é possível dizer que a mecânica do sensor não é adequada para esse tipo de jogo. Para tentar contornar o problema a cada vez que o jogador morria, o programa de aquisição de dados fisiológicos executava novamente desde o início da fase, dessa forma o ADD levava em conta apenas o estado afetivo da tentativa mais atual.

Algumas considerações devem ser feitas sobre o experimento. Diversas pessoas se sentiram mais afetadas apenas pelo fato de estarem usando o equipamento. O efeito contrário também foi verificado, isto é, pessoas recusando-se a participar do experimento devido à necessidade de uso do equipamento. A Figura 5.7 e a Figura 5.8 mostram participantes do experimento, o primeiro participante questionou várias vezes se havia riscos de choques e outros problemas elétricos, isso pode ter influenciado seu estado afetivo. O segundo participante (um aluno de um curso de Engenharia) por outro lado não tinha problemas com o uso do sensor, para este participante acredita-se que o sensor não interferiu em seu estado afetivo.

O resultado do questionário para os testes com o EDA foi de 3.87, enquanto o resultado para os testes com a eletrocardiografia foi de 3.59, somando esses dois valores e calculando a média, temos que a média utilizando sensores é de 3.73.

Figura 5.7: Experimento utilizando o sensor EDA

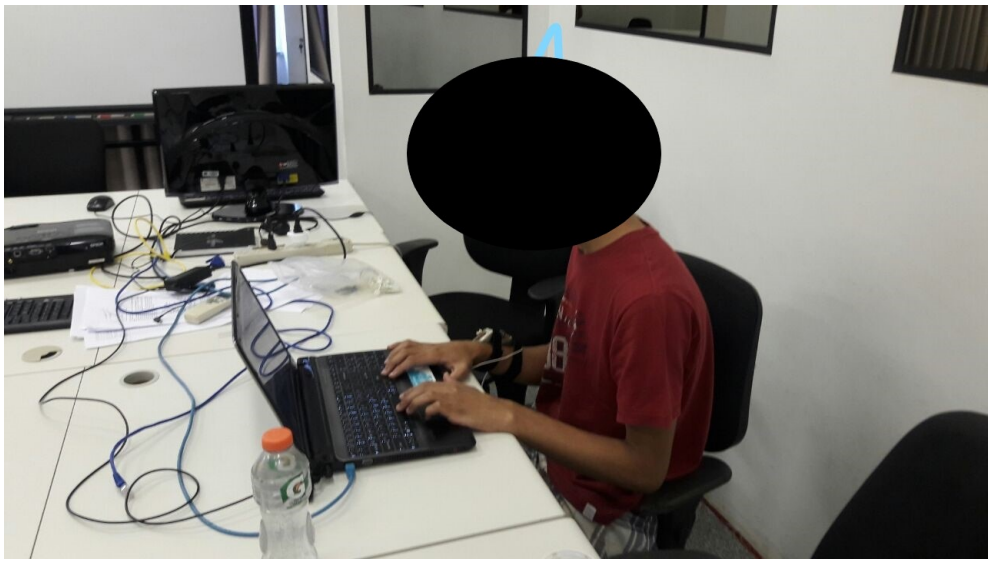

Figura 5.8: Participante do sexo masculino realizando o experimento com o sensor EDA

Capítulo 6

Conclusão

Devido a natureza do mercado dos jogos o conceito de fluxo é fundamental para o sucesso de um jogo. A partir dessa premissa o conceito de fluxo (ou *flow*) foi utilizado com o objetivo de tornar melhor a experiencia de jogo. Para fazer o jogo manter o estado de fluxo no jogador utilizou-se um ajuste dinâmico de dificuldade, essa técnica permitiu que o nível de dificuldade do jogo fosse alterado em tempo real para manter a relação dificuldade versus habilidade próxima a uma constante (pelo menos dentro de uma faixa aceitável).

Sobre a técnica de ADD em jogos, esta deve ser cada vez mais aplicada, tendendo a ser uma regra para os jogos que querem ganhar seu espaço no mercado. Como esta técnica deve ser específica para cada gênero de jogo, ainda não esta amplamente difundida e implementada em todos os jogos, entretanto, pesquisas vem sendo feitas para preencher essa lacuna. O ADD pode ser feito utilizando várias técnicas, contudo ainda não existe uma regra bem definida para implementá-lo.

É possível dizer que já existem ADDs bem definidos para jogos de plataforma 2D, FPS e outros gêneros descritos aqui. Também podemos dizer que o uso de inteligências artificiais é uma tendência nesse ramo, entretanto não se pode dizer que será sempre a melhor abordagem, como o processo de treinar e criar a IA é lento e custoso, outros métodos podem ser mais benéficos em relação a custo-benefício.

Neste trabalho, para medir o desempenho do jogador foram coletados dois tipos de dados distintos: dados do jogo e dados fisiológicos. Coletar dados fisiológicos para este fim ainda é algo em evolução, contudo, já existem placas de aquisição de dados sobre o jogador disponíveis no mercado e teorias associando sinais vitais a emoções bem definidas. Dessa forma o ADD baseado em dados fisiológicos do jogador vem crescendo e ADDs híbridos como o deste trabalho podem se mostrar ainda mais eficientes que os ADDs tradicionais que utilizam apenas um tipo de dado, no caso, dados oriundos do jogo.

Pode-se inferir que o jogo *Asteroids* de Matthew Renze permitiu a adição dos elementos necessários para poder realizar a medição de dificuldade do jogo. Após a condução de experimentos com e sem a coleta de dados fisiológicos com o sensor Bitalino, foi concluído que a velocidade do asteroide é de alta relevância para a percepção da dificuldade por parte do jogador, e que apenas

o aumento da densidade não tem relevância para, mas que o aumento das duas combinadas possui maior influência na percepção do que apenas o aumento da velocidade. É importante dizer que outras variáveis podem ter influência ainda maior do que a velocidade, sendo portanto, necessário a realização de mais testes.

Sobre a ferramenta Unity pode-se afirmar que a mesma possibilita o desenvolvimento de jogos com mais rapidez e eficiência do que a criação de uma *engine* própria. As *features* de alto nível com relação à colisão de objetos e renderização de imagens, fazem com que o desenvolvedor tenha que se preocupar apenas com a lógica do jogo, enquanto detalhes de implementação e código relacionados a problemas gráficos são deixados para a ferramenta.

Com relação à placa BITalino, com apenas um sensor, a mesma demostrou ser adequada para a realização dos experimentos. Para a utilização de mais de um sensor ao mesmo tempo recomenda-se o uso de outras placas ou alguma mecânica alternativa para os sensores. Mesmo assim, devido ao seu baixo custo e alta versatilidade esta ainda é uma boa opção para os experimentos descritos aqui. O programa desenvolvido em C++, foi capaz de coletar todos os dados necessários para a realização do experimento, ainda que os dados sejam coletados de forma bruta e precisem de processamento posterior.

Analisando os resultados sem o uso de sensores de *Biofeedback* apresentados na seção anterior, pode-se dizer que o ADD híbrido adapta pior o desafio ao jogador do que o ADD baseado apenas em dados de desempenho coletados do jogo. Com relação ao dois sensores utilizados, tendo em vista o baixo volume de dados, não é apropriado concluir que um é melhor ou mais indicado do que outro. Para tanto, é necessário a realização de mais testes e uma coleta de dados mais volumosa e significativa.

No que diz respeito ao ao algoritmo de mudança na dificuldade, conclui-se que o seu cálculo ser feito de maneira linear não causou problemas. Os testes realizados indicaram que ele foi efetivo em trazer o jogador para o estado de fluxo.

6.1 Perspectivas futuras

Este trabalho deixa várias possibilidades para trabalhos futuros. A primeira delas seria fazer os testes de percepção de dificuldade para as outras variáveis do jogo, tais como, velocidade da nave, tamanho dos asteroides, asteroides gigantes, asteroides com escudo e explosivos. Com todas essas variáveis determinadas seria possível realizar um ajuste mais fino.

Outra ponto deixado de lado neste trabalho foi a implementação de uma inteligência artificial. Seria útil um estudo para implementar uma IA utilizando a técnica de *Deep reiforcement learning*. Com essa técnica o ajuste deixaria de ser um ajuste linear, porém esse ajuste poderia melhor do que o atual, pois o mesmo supõe que o jogador esta sempre desconfortável com a dificuldade e portanto, sempre a altera (mesmo que pouco).

Um dos maiores problemas deste trabalho foi a mecânica da placa utilizada. Pode-se fazer melhor uso da placa utilizando-a no modo livre, onde seus sensores ficam fisicamente separados,

permitindo que o sensor fique mais próximo do local da medição. Isso deve acabar com o problema de desconectar a placa durante um experimento. Supondo que esta mudança seja feita na placa, ela deve ficar fixa em um lugar enquanto os sensores seriam colocados individualmente.

Para trabalhos futuros, também pode-se implementar uma ferramenta de análise de emoções por expressões faciais, utilizando as emoções detectadas para identificar falsas leituras do sensor. Esse ajuste utilizando visão computacional, teria como maior contribuição a validação dos dados vindos do sensor, mas também poderia ser usado para adicionar outros estados emocionais que os sensores não captam.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] SOARES, R. T. Sensores de biofeedback em jogos eletrônicos: um estudo teórico e prático. 2017.

[2] AGUIAR, J. V. S.; FERNANDES, L. V. Ajuste dinâmico de dificuldade em jogos a partir de variáveis do jogo e do usuário. 2017.

[3] FEEDBACK 2018. 2018. `http://www.golfwrx.com/175284/get-better-faster-with-a-tight-feedback-loop/`. Acessed: 2018-07-03.

[4] MEGAMAN Freezeman 2018. 2018. `https://strategywiki.org/wiki/Mega_Man_7/Freeze_Man`. Acessed: 2018-07-03.

[5] MOURATO, F.; SANTOS, M. P. d. Measuring difficulty in platform videogames. In: *4. ª Conferência Nacional Interacção humano-computador*. [S.l.: s.n.], 2010.

[6] HUNICKE, R. The case for dynamic difficulty adjustment in games. In: ACM. *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*. [S.l.], 2005. p. 429–433.

[7] IMRE, D. Real-time analysis of skin conductance for affective dynamic difficulty adjustment in video games. *Bachelor's dissertation) Retrieved from http://danielimre. com/wp-content/uploads/2016/03/Imre-Real-Time-Analysis-of-Skin-Conductance-for-Affective-Dynamic-Difficulty-Adjustment-in-Video-Games. pdf*, 2016.

[8] ASTEROIDS 1979. 2018. `https://www.pcworld.com/article/2606956/atari-is-rebooting-asteroids-as-a-dayz-style-survival-game.html`. Acessed: 2018-07-03.

[9] CODEPRODUCTIVIT 2018. 2018. `https://www.google.com.br/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=2ahUKEwi1v_esm4bdAhWJEJAKHZhAB2MQjRx6BAgBEAU&url=https\%3A\%2F\%2Fblog.claudiupersoiu.ro\%2F2011\%2F10\%2F04\%2Fcode-quality-and-development-time\%2Flang\%2Fen\%2F&psig=AOvVaw0TnPZXT_Wsn8aHKLhVwxVM&ust=1535217998191383`. Acessed: 2018-09-24.

[10] CSIKSZENTMIHALYI, M.; BOSE, D. K. *Flow: e Psychology of Optimal Experience*.

[11] JUUL, J. The game, the player, the world: Looking for a heart of gameness. *PLURAIS-Revista Multidisciplinar*, v. 1, n. 2, 2010.

[12] VANNUCCHI, H.; PRADO, G. Discutindo o conceito de gameplay. *Texto Digital*, v. 5, n. 2, p. 130–140, 2009.

[13] GAME Taxonomy. `http://www.ferzkopp.net/wordpress/2016/01/02/game-taxonomy/`. Accessed: 2018-05-26.

[14] SHERRY, J. L. Flow and media enjoyment. *Communication theory*, Wiley Online Library, v. 14, n. 4, p. 328–347, 2004.

[15] BAILEY, C.; KATCHABAW, M. An experimental testbed to enable auto-dynamic difficulty in modern video games. p. 18–22, 2005.

[16] ARAUJO, B. D.; O, B. F. *Um estudo sobre adaptatividade dinâmica de dificuldade em jogos*. Tese (Doutorado) — Dissertaçao de mestrado, Pontificia Universidade Católica do Rio de Janeiro-PUC-Rio, Rio de Janeiro, Brasil, 2012.

[17] LIU, C. et al. Dynamic difficulty adjustment in computer games through real-time anxiety-based affective feedback. *International Journal of Human-Computer Interaction*, Taylor & Francis, v. 25, n. 6, p. 506–529, 2009.

[18] YANNAKAKIS, G. N.; HALLAM, J. Real-time game adaptation for optimizing player satisfaction. *IEEE Transactions on Computational Intelligence and AI in Games*, IEEE, v. 1, n. 2, p. 121–133, 2009.

[19] HINTZE, A.; OLSON, R. S.; LEHMAN, J. Orthogonally evolved ai to improve difficulty adjustment in video games. In: SPRINGER. *European Conference on the Applications of Evolutionary Computation*. [S.l.], 2016. p. 525–540.

[20] ASTEROIDS wikipedia, Asteroids Game. `https://en.wikipedia.org/wiki/Asteroids_(video_game)`. Accessed: 2018-05-26.

[21] BITALINO board. `http://bitalino.com/en/plugged-kit-ble`. Accessed: 2018-06-03.