5
Metodologia

Neste capítulo, o jogo especialmente desenvolvido para os testes é descrito, o algoritmo de jogo adaptativo é proposto e os principais aspectos de implementação são apresentados. Este capítulo também apresenta o método de avaliação da experiência do jogo.

5.1
Tipo do jogo

Para a avaliação do uso de dificuldade adaptativa em jogos, foi desenvolvido um jogo do tipo *shoot'em up* classificado pela taxonomia de Gularte apresentada na seção 2.1.2 como um jogo de tiro de ação *single player*.

Os jogos *shoot'em up* também são conhecidos como *shmups* ou coloquialmente como "jogos de navinha". Eles constituem um gênero de jogos popular em *arcades* e em especial no Japão, sendo tradicionalmente em *scroll* horizontal ou vertical, tipicamente em 2D. Representantes populares do gênero[1] são Space Invaders, R-Type, Gradius, Ikaruga e Dodonpachi.

A Figura 5.1 ilustra o aspecto do jogo proposto.

5.2
Algoritmo para jogo adaptativo

A modelagem do grau de dificuldade do jogo proposto e o processo de ajuste do jogo são baseados na hipótese de que o comportamento do NPC é determinado por um conjunto V de variáveis, chamadas de variáveis de comportamento do NPC. No jogo *shoot'em up* proposto, temos:

$$V = \{speed, shotDelay, halfRange\} \tag{5-1}$$

[1]Para mais informações sobre o gênero, recomendamos como leitura: `http://www.racketboy.com/retro/shooters/shmups-101-a-beginners-guide-to-2d-shooters` e `http://www.racketboy.com/retro/shooters/games-that-defined-the-shmups-genre`.

Figura 5.1: Exemplo de situação de jogo

Os valores destas variáveis nascem iguais aos seus valores de referência:

$$speed = speed_0 \tag{5-2a}$$

$$shotDelay = shotDelay_0 \tag{5-2b}$$

$$halfRange = halfRange_0 \tag{5-2c}$$

Os valores de referência são estabelecidos pelo programador. No jogo em questão, os valores escolhidos foram: $speed_0 = 300$, $shotDelay_0 = 900$ e $halfRange_0 = 200$.

Estas variáveis são a base para os comportamentos de mais alto nível de um NPC, tais como agilidade e precisão. A variável *speed* tem efeito direto no cálculo de deslocamento do NPC, de maneira que quanto mais alto for esse valor, mais experiente o jogador terá que ser para enfrentar este NPC mais ágil. O atraso entre cada tiro (*shotDelay*) afeta a taxa de tiro (*rate of fire*) do NPC e, portanto, um valor baixo desta variável significa um NPC mais eficiente. A variável *halfRange*[2] determina a área de ameaça, que estabelece a precisão do tiro do NPC. Quanto menor for esta área, mais preciso será o tiro

[2] *halfRange* tem interpretações diferentes dependendo do tipo do jogo. Por exemplo, em jogos FPS (*First-Person Shooter*) *halfRange* está associada ao decaimento de dano (*damage falloff*) de uma dada arma que é geralmente governado por duas variáveis: máximo alcance (*maxRange*) e percentagem mínima de dano (p). Neste caso, a arma causa 100% de dano para os primeiros $p \times maxRange$ e a partir deste ponto cai linearmente para 0% de dano ao longo da trajetória até alcançar a distância igual a *maxRange*. Consequentemente, o dano em $halfRange = \frac{maxRange}{2}$ é $\frac{50}{(100-p)} \approx 56\%$.

e, consequentemente, mais experiente deverá ser o jogador para enfrentar este NPC mais letal.

Uma maneira de ajustar o jogo para um determinado grau de dificuldade é criar um fator multiplicador (associado a cada tipo de jogador) que afeta as variáveis de comportamento. No nosso jogo, consideramos o seguinte conjunto de tipos de jogador:

$$Tipos = easy, medium, hard \tag{5-3}$$

e definimos o multiplicador de dificuldade $m(tipo)$, que é neutro (i.e. igual a 1.0) para o tipo *medium*. No nosso jogo, consideramos o seguinte:

$$m(easy) = 0.85 \tag{5-4a}$$

$$m(medium) = 1.0 \tag{5-4b}$$

$$m(hard) = 1.2 \tag{5-4c}$$

Com base nestes conceitos, o ajuste do tipo (ou modelo[3]) de jogador é feito aplicando o multiplicador sobre as variáveis de comportamento, como no Algoritmo 5.2. Exploramos essa aplicação em detalhes na seção 5.3.4.

Algoritmo 5.1 function ajusta(tipo) return valores de V

$speed \leftarrow speed_0 \times m(tipo)$
$shotDelay \leftarrow \frac{shotDelay_0}{m(tipo)}$
$halfRange \leftarrow \frac{halfRange_0}{m(tipo)}$
$newV \leftarrow \{speed, shotDealy, halfRange\}$
return $newV$

A remodelagem do tipo de jogador é baseada na definição de um conjunto C de n características de desempenho do jogador, também chamadas de *traits*, tais que:

$$c_i \in C / c_i \in [0, 1], i = 1, n \tag{5-5}$$

E cujos valores são calculados no final de cada *wave*[4] de inimigos. A remodelagem de jogador implementada é explicada em detalhes na seção 5.5.

A relação entre as características de desempenho do jogador e o tipo de jogador é dada por valores mínimos e máximos de desempenho padrão

[3]Neste trabalho, usamos livremente os termos "tipo de jogador" e "modelo de jogador" como sinônimos, apesar de que, rigorosamente falando, "modelo de jogador" e "modelagem de jogador" referem-se a todo o conjunto da proposta.

[4] *Wave* é o conjunto de inimigos que surgem juntos. É uma terminologia típica de jogos de tiro com formação de naves inimigas (e.g. Space Invaders).

associados a cada tipo: $c_{i,min}^{tipo}$ e $c_{i,max}^{tipo}$. Por exemplo, $c_{2,min}^{medium} = 0.3$ e $c_{2,max}^{medium} = 0.6$, conforme ilustra a Tabela 5.5.1.

Desta maneira, definimos os valores mínimos e máximos que identificam um determinado tipo de jogador:

$$MIN^{tipo} = \sum_{i=1}^{n} c_{i,min}^{tipo} \tag{5-6a}$$

$$MAX^{tipo} = \sum_{i=1}^{n} c_{i,max}^{tipo} \tag{5-6b}$$

A Tabela 5.5.1 mostra, por exemplo, que $MIN^{medium} = 1.2$ e $MAX^{medium} = 2.4$.

Considerando todas as definições acima, podemos apresentar o nosso algoritmo de jogo adaptativo (Algoritmo 5.2), que é baseado no framework de Charles e Black (Charles04) e no método de adaptação de jogo proposto por (Houlette04):

Algoritmo 5.2 Algoritmo de jogo adaptativo

$\alpha \leftarrow$ learningRate
$tipo_0 \leftarrow$ tipo inicial informado pelo jogador
$c_i \leftarrow \frac{\left(c_{i,min}^{tipo_0} + c_{i,max}^{tipo_0}\right)}{2}$ {i.e., a média do desempenho padrão do $tipo_0$ para a característica c_i.}
$V \leftarrow$ conjunto inicial de valores de variáveis de comportamento
for all *waves* **do**
  $c_{i,obs}$ é o valor observado da característica i
  $c_i \leftarrow c_i + \alpha \times (c_{i,obs} + c_i)$ {i.e., atualiza cada característica pela regra de LMS.}
  $desempenho \leftarrow \sum_{i=1}^{n} c_i$
  if $desempenho \in \left[MIN^{tipo}, MAX^{tipo}\right]$ **then**
    $novoModelo \leftarrow tipo$
  else if $modeloCorrente \neq novoModelo$ **then**
    Remodela jogador:
    $V \leftarrow ajusta(modeloCorrente)$
  else
    Mantém modelo atual
  end if
  Armazena estatística da *wave*
end for

5.3 Definição do jogo

A linguagem escolhida para o desenvolvimento foi C++ usando Lua [5](Ierusalimschy06) como linguagem de script e o motor de jogos gratuito e de código aberto ClanLib[6]. Para a arte e som, foram usados *assets* livres, sob licença GPL[7] ou sob licença Creative Commons[8], como os gráficos disponibilizados na biblioteca SpriteLib[9], gráficos e efeitos sonoros do site OpenGameArt[10] e as músicas no site Jamendo[11].

De acordo com (Bernhaupt10, p. 5–6) e (Novak11, p. 340–352), o jogo desenvolvido pode ser considerado como estando na fase alfa de desenvolvimento, onde testes de jogabilidade são executados para realizar a avaliação do sistema desenvolvido. No caso, nossa avaliação apresentada no capítulo 6, além de estar relacionada com a corretude do sistema adaptativo desenvolvido, está relacionada também à percepção e aceitação do sistema pelos jogadores.

5.3.1 Seleção de dificuldade

Duas versões do jogo foram desenvolvidas neste trabalho: uma com adaptatividade e outra sem. Ambas as versões do jogo contam com a seleção de dificuldade através de um menu (Figura 5.2). A seleção permite uma calibragem inicial do sistema adaptativo, descrito na seção 5.4 de acordo com a preferência do jogador e para o jogo sem adaptatividade seleciona a dificuldade comum a todas as *waves* de inimigos.

5.3.2 Lua

Usamos Lua como linguagem de definição de dados, permitindo configurar parâmetros sem a necessidade de recompilar o código-fonte do jogo. A listagem 5.1 mostra o *script* Lua usado para configurar as opções de jogo, como o multiplicador de cada nível de dificuldade e demais parâmetros com seus valores conforme utilizados nos testes. Também usamos Lua para descrever a cena de testes, com as formações de naves inimigas por *wave* e seus comportamentos.

```
-- Configuration file for Adaptive Shooter
```

[5] http://www.lua.org
[6] http://clanlib.org
[7] http://www.gnu.org/copyleft/gpl.html
[8] http://creativecommons.org/
[9] http://www.widgetworx.com/widgetworx/portfolio/spritelib.html
[10] http://opengameart.org/
[11] http://www.jamendo.com/

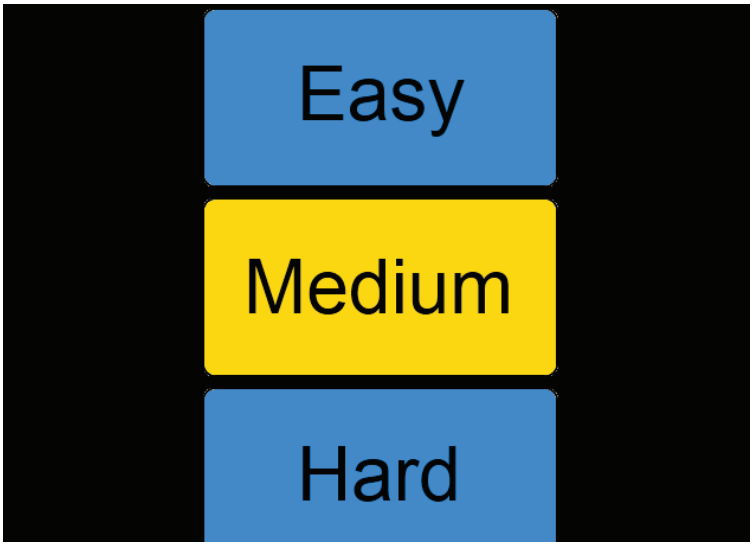

Figura 5.2: Menu de seleção de dificuldade.


```
Player = {
SpeedX = 250,
SpeedY = 250,
Resource = "sprites/rwing",
Lives = 5,
LearningRate = 0,
ShotSpeedX = 0,
ShotSpeedY = -500,
ShotDelay = 450,
HitBoxScale = 0.25
}

Enemies = {
ShotSpeedX = 0,
ShotSpeedY = 300,
ShotDelay = 900,
EasyMultiplier = 0.85,
NormalMultiplier = 1.0,
HardMultiplier = 1.2
}
```

Listagem 5.1: Configurações de teste em config.lua

5.3.3
Esquema de controle

Foi permitido aos jogadores escolherem o esquema de controle que preferissem dentre as implementações por teclado ou *gamepad*. Para os testes, usamos um *gamepad* de XBox360 por sua compatibilidade com o sistema e motor de jogo utilizados. Para o teclado, foram utilizadas as setas direcionais para movimentar o avatar do jogador pela tela e a tecla `Z` para disparar os tiros e confirmar a seleção de dificuldade. Para o *gamepad*, foi utilizada a alavanca analógica esquerda (com interpretação digital do valor de resposta) para movimentar o avatar do jogador e o botão `X` para disparar os tiros e confirmar a seleção de dificuldade.

A oferta de ambos os métodos de controle para os jogadores se justifica por permitir que o jogador escolhesse o que mais lhe agradasse, procurando favorecer as condições para que o jogador entrasse em fluxo, ao invés de obrigar o jogador a usar uma opção de controle que não lhe fosse familiar ou que não lhe agradasse.

5.3.4
Gameplay

O *gameplay* do jogo implementado consiste em o jogador, representado pela nave que controla, desviar dos tiros e das naves inimigas que se movimentam da parte superior da tela em direção à parte inferior, enquanto busca abater a maior parte possível dos inimigos para aumentar sua pontuação. Cada colisão do jogador com um tiro disparado pelos inimigos ou contra uma nave inimiga custa-lhe uma vida. Findas as cinco vidas (dada a configuração apresentada na listagem 5.1), o jogo termina com uma tela de *game over* (Figura 5.3). Caso o jogador sobreviva a todas as *waves* de inimigos, é vitorioso e apresentado à tela de vitória (Figura 5.4).

Foram implementados dois tipos de inimigos: discos voadores, que se movimentam em formação em asa; caças, que se movimentam em zig zag pela tela ou restritos a um corredor imaginário (exemplificados na Figura 5.1). A variação da inteligência artificial desses inimigos é feita através de uma variável de comportamento, com ajustes influenciados pelos multiplicadores de dificuldade apresentados na listagem 5.1 em sua velocidade, atraso entre os tiros e área de ameaça. A velocidade é modificada diretamente pelo multiplicador de dificuldade. A área de ameaça simula uma inteligência para a precisão dos disparos do inimigo, onde quanto menor a dificuldade, maior a área de ameaça (definida pelo intervalo $[SpriteCenter - halfRange, SpriteCenter + halfRange]$ onde $halfRange = \frac{200}{m(tipo)}$ e $SpriteCenter$ é a posição do centro do *sprite* do NPC).

Figura 5.3: Tela de fim de jogo.

Como o atraso entre os tiros também é modificado pela dificuldade (definido como $delay = \frac{ShotDelay}{m(tipo)}$), isso permite uma janela de reação maior (em baixa dificuldade) e menor (em alta dificuldade) para o jogador reagir ao disparo e contra-atacar. $m(tipo)$ foi definido no conjunto de equações 5-4.

5.4 Sistema adaptativo

O sistema adaptativo implementado constitui-se de um agente inteligente, o AIManager, que percebe o ambiente do jogo através dos dados desempenho do jogador e modifica o ambiente alterando variáveis dos NPCs de forma a buscar um nível de dificuldade adequado à variação de desempenho do jogador. Por nível de dificuldade adequado, queremos dizer o ajuste das variáveis dos NPCs que proporcione um desafio ao jogador sem ser fácil demais (identificado por um baixo ou nenhum consumo dos recursos do jogador, como por exemplo pontos de vida de seu avatar e bombas utilizadas) nem difícil demais (identificado por mortes constantes do avatar do jogador com baixa porcentagem de inimigos destruídos).

Escolhemos não alterar as variáveis relacionadas ao jogador pois o jogador pode perceber as alterações ao sentir as diferenças em aspectos como seu deslocamento, seu poder de fogo e sua defesa. Tais mudanças, se não explícitas ao jogador por mecânicas do jogo passivas (como evolução em níveis) ou ativas (por sua escolha, como coleta de *power ups* ou por treinamento de habilidades), podem causar quebra da imersão por violar o contrato de suspensão de descrença das regras de jogo.

Figura 5.4: Tela de vitória.

5.4.1 Modelagem e implementação

O sistema adaptativo foi modelado como uma implementação do *framework* de Charles e Black (Charles04, Charles05), apresentado na seção 3.5 e desenvolvido como uma biblioteca separada do jogo. A classe AIManager funciona como o gerente dos agentes inteligentes (implementações da classe abstrata AIAgent) que são afetados pelas mudanças percebidas do ambiente (i.e., o estado da modelagem de jogador), permitindo a atualização dos agentes e acesso ao modelo de jogador (implementação da classe abstrata PlayerModel). Cada modelo de jogador é composto por características, representadas pela estrutura Traits. O AIManager e suas classes abstratas correlatas são apresentados na Figura 5.4.1.

A atualização da modelagem de jogador é dependente da implementação do cliente da biblioteca. Apresentamos o pseudocódigo da implementação da comparação entre modelos de jogadores no algoritmo 5.5 na seção 5.5. Após a atualização do modelo de jogador observado pela instância de AIManager, a classe cliente faz a chamada ao método updateAgents. Os agentes inteligentes observados pelo AIManager recebem a chamada de seu método virtual puro `updateStats()`, implementado pelo cliente. Dessa forma, o cliente decide como a atualização dos agentes inteligentes deve proceder de acordo com suas especificidades. Esse processo é detalhado no algoritmo 5.4.1. O método `compare()` da classe PlayerModel é detalhado no algoritmo 5.5.

O uso no jogo (cliente da biblioteca AIManager) se dá através de implementações das classes abstratas da biblioteca. A Figura 5.6 mostra em

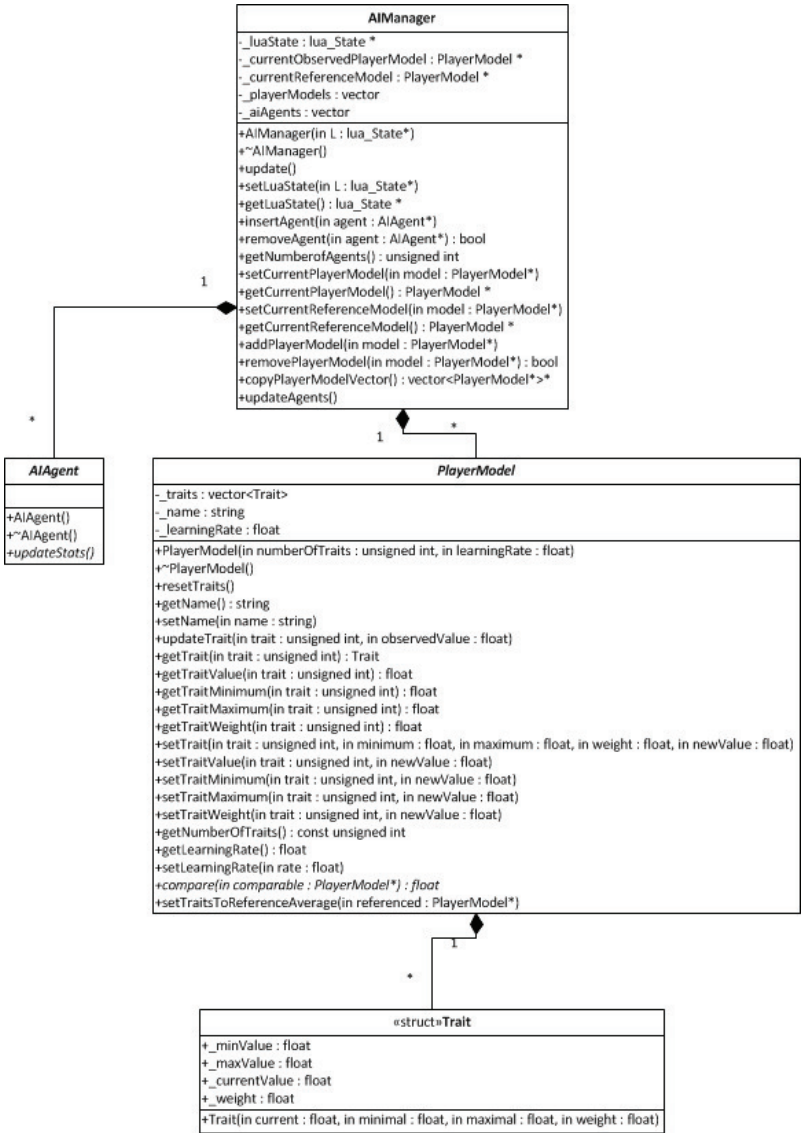

Figura 5.5: Diagrama de classes UML do AIManager

destaque sobre o *framework* de Charles e Black alguns detalhes de nossa implementação através das chamadas de métodos relativos às partes do sistema adaptativo *online*. Cabe notar que o método `waveFinish()` está fora do sistema adaptativo por ser o disparador da chamada do método `update()` do AIManager, já que atualizamos a modelagem de jogador apenas ao final de cada *wave* de inimigos.

O método `updateAgents()` no algoritmo 5.4.1 chama o método virtual `updateStats()` de cada agente inteligente controlado pelo AIManager. Em nossa implementação deste método, verificamos qual a dificuldade do modelo de jogador observado ao final de cada *wave* de inimigos e ajustamos os parâmetros do multiplicador de velocidade, da área de ameaça e do atraso

entre os tiros de cada agente inteligente. O fator de aprendizado α com valor de 0.3 (ver algoritmo 5.5 e equação 5-7), conforme observado por Houlette (Houlette04, p. 560), permitiu uma avaliação com atualização efetiva do modelo de jogador para a dificuldade correspondente aproximadamente a cada três *waves* de inimigos.

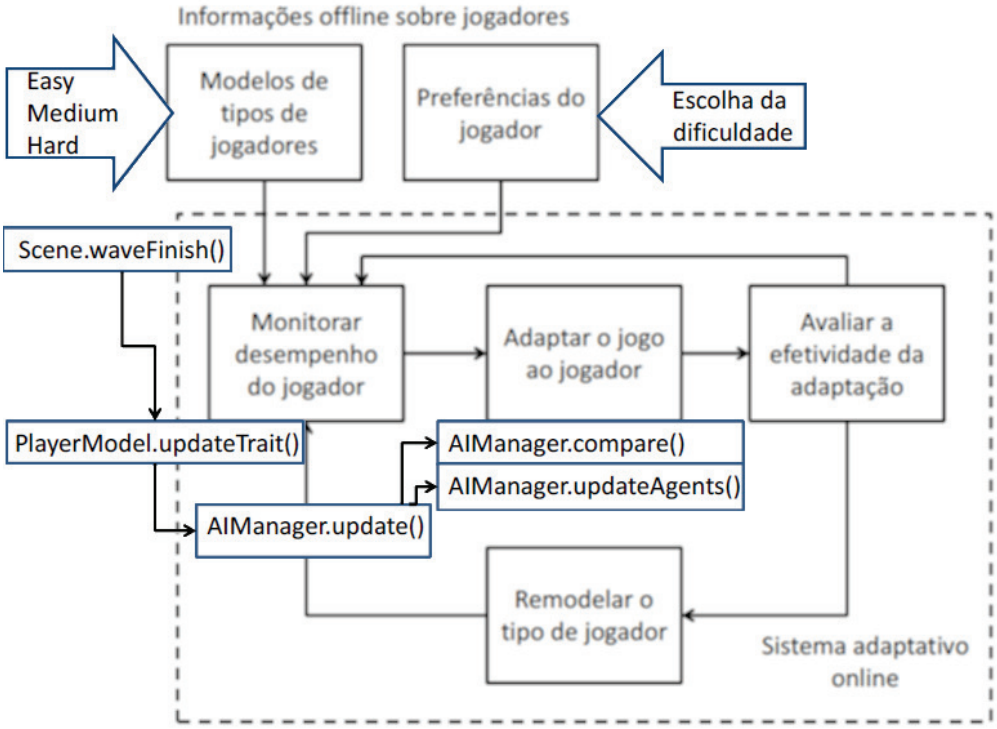

Figura 5.6: Sobreposição de partes da implementação ao *framework* de Charles e Black. Adaptado de (Charles04).

5.5 Implementação da modelagem do jogador

Para a modelagem do jogador, decidimos nos basear na modelagem proposta por Houlette (Houlette04, p.559–560). No modelo de Houlette, cada característica observada encontra-se no intervalo real de 0 a 1, onde zero significa ausência ou "o jogador nunca faz isso" e um significa presença total ou "o jogador sempre faz isso". Assim, a característica é definida como:

$$\forall x \in X / \quad x \in \mathbb{R}, 0 \leq x \leq 1$$

Onde:

X = Conjunto de características do modelo de jogador.

As características observadas foram:

Algoritmo 5.3 function AIManager.update( )

```
result ← 0
for playerModelIterator ← playerModels.begin() to playerModels.end()
do
  result ← currentObservedModel.(playerModelIterator)
  if result < 0 then
    continue
  else if result = 0 then
    currentReferenceModel ← playerModelIterator
  else
    result ← currentReferenceModel.compare(playerModelIterator)
    if result < 0 then
      currentReferenceModel ← playerModelIterator
    end if
  end if
end for
currentObservedModel.setName(currentReferenceModel.getName())
updateAgents()
```

Taxa de acerto: Definida pelo número de tiros disparados pelo jogador pelo número de tiros acertados em naves inimigas. Indica a precisão dos disparos do jogador, onde um valor 1 indica que todos os disparos foram bem sucedidos e um valor 0 indica que nenhum disparo acertou as naves inimigas.

Variação de vidas: Taxa de variação de vidas do jogador. Definido pelo número de vidas restantes ao final da *wave* pelo número no início desta. Um valor 1 indica que o jogador perdeu todas as vidas e um valor 0 indica que nenhuma vida foi perdida.

Taxa de inimigos destruídos: Definida pelo número de inimigos abatidos até o final da *wave* pelo número total de inimigos da *wave*. Um valor 1 indica que todos os inimigos foram destruídos, enquanto um valor 0 indica que nenhum inimigo foi destruído.

Taxa de inimigos destruídos total: Definida pelo número total de inimigos abatidos pelo número total de inimigos no jogo. Um valor de 1 indica que todos os inimigos foram destruídos, enquanto um valor 0 indica que nenhum inimigo foi destruído.

Para a atualização do valor das características ao final de cada *wave* de inimigos, como sugerido por Houlette (Houlette04, p. 560), usamos um método baseado na média dos mínimos quadrados (LMS, *Least Mean Squares*), como mostrado na equação 5-7.

$$traitValue = \alpha \cdot observedValue + (1 - \alpha) \cdot traitValue \quad (5\text{-}7)$$

Onde:

$$\alpha = \text{Constante de taxa de aprendizado.}$$

Esta equação é inspirada na regra de aprendizagem supervisionada conhecida por "regra de atualização LMS" (*LMS update rule*), por "regra de aprendizado de Widrow-Hoff" ou por "regra delta" (*delta rule*). Widrow e Hoff (Widrow88, pp. 123–134) treinaram redes neurais simples usando esta regra especialmente voltada para aprendizagem *online*.

Através do algoritmo implementado aplicado a cada uma das características observadas (ver algoritmo 5.5), o modelo de jogador em uma determinada seção do jogo é uma suposição do AIManager e cada chamada à função de atualização é um refino deste modelo através de uma nova evidência (o novo valor observado). A influência da função sobre a atualização da característica é limitada por uma constante de taxa de aprendizado entre 0 e 1 definida empiricamente. A necessidade de se limitar o ajuste é dada pela incerteza do jogador agir de forma completamente consistente entre cada período de atualização, o que pode levar a grandes flutuações no modelo, segundo Houlette.

Algoritmo 5.4 function Player.updateTrait(trait, observedValue)

$currentValue \leftarrow traits[trait].currentValue$
$\delta \leftarrow observedValue - currentValue$
$weighted\delta \leftarrow learningRate \times \delta$
$setTraitValue(trait, traits[trait].currentValue + weighted\delta)$

Após a atualização das características do modelo de jogador, é necessário verificar se o modelo atual ainda se encontra dentro do modelo anteriormente atribuído ou se pertence a outro modelo. A comparação das características do modelo atual com o modelo atribuído indica se a atualização do modelo é necessária e é feita como no algoritmo 5.5 e pode ser visto como um *perceptron* de uma rede neural, embora neste caso tenhamos apenas um neurônio.

5.5.1 Modelos de jogador implementados

Para o jogo, implementamos modelos de jogadores baseados em três tipos de dificuldade: *Easy*, *Medium* e *Hard*. Os modelos foram construídos baseados nas quatro características observadas citadas anteriormente, taxa de acerto, variação de vidas, taxa de inimigos destruídos por *wave* e taxa de inimigos destruídos total. Determinamos empiricamente os valores para cada modelo de jogador, apresentados na Tabela 5.5.1.

Algoritmo 5.5 function PlayerModel.compare(comparable)

```
total ← 0
totalMin ← 0
totalMax ← 0
for i ← 0 to numOfTraits do
  total ← total + getTraitValue(i)
  totalMin ← totalMin + comparable.getTraitMinimum(i)
  totalMax ← totalMax + comparable.getTraitMaximum(i)
end for
if total < totalMin then
  return -1
else if total > totalMax then
  return 1
else
  return 0
end if
```

Tabela 5.1: Modelos de jogador implementados

	Easy		Medium		Hard	
	Min	Max	Min	Max	Min	Max
Taxa de acertos	0.0	0.3	0.3	0.6	0.6	1.0
Variação de vida	0.6	1.0	0.3	0.6	0.0	0.3
Inimigos por wave	0.0	0.3	0.3	0.6	0.6	1.0
Inimigos total	0.0	0.3	0.3	0.6	0.6	1.0
Totais	0.6	1.9	1.2	2.4	1.8	3.3

Para tais intervalos de mínimo e máximo totais dos modelos de jogador, percebe-se que há intercessões entre os pares de modelos *Easy-Medium* e *Medium-Hard*, com valor de 0.6. Como os modelos de jogador estão ordenados no vetor de comparação, essa intercessão permite uma ascenção mais lenta nos níveis de dificuldade e uma queda mais rápida. O objetivo é procurar reduzir a frustração do jogador com uma dificuldade elevada.

5.6 Testes com jogadores

A literatura apresenta diversas formas de se realizar testes com jogadores para detectar diversão e a experiência de fluxo, dentre as quais podemos citar (Lankes10), (Koeffel10), (Schell11, p. 122). Nacke e Lindley (Nacke10) fazem uso de eletromiografia facial para analisar a resposta de músculos faciais de modo a detectar variações de emoção no jogador. Questionários pós-teste são comumente utilizados para avaliação da experiência de jogo e envolvimento do jogador (Brockmyer09, Calvillo-Gámez10, Ibáñez11, Machado11b).

Para poder avaliar o uso do sistema adaptativo desenvolvido, foram

realizados testes com jogadores. Os testes realizados neste trabalho envolveram os jogadores jogarem ambas as versões desenvolvidas (com e sem dificuldade adaptativa) em ordem aleatória e responderem o questionário de avaliação da experiêcia de jogo. Neste capítulo, apresentamos os conceitos abordados nos testes com os jogadores e justificamos as escolhas na metodologia de testes. Os questionários encontram-se no apêndice A.

Cabe ressaltar que nesta seção nos referimos a usuário, participante, testador e jogador como sinônimos dentro do ambiente de teste, não distinguindo entre cada nome utilizado.

5.6.1 User experience em jogos

A definição de *user experience* segundo o ISO é: "as percepções e respostas de uma pessoa que resultam do uso ou uso antecipado de um produto, sistema ou serviço" (Bernhaupt10, p. 4, tradução nossa). Em jogos, trata-se do relacionamento do jogador com o jogo enquanto este acontece, tanto o processo quanto o resultado do jogar.

Calvillo-Gámez et al. (Calvillo-Gámez10, p. 50) mencionam que é possível avaliar jogos em relação à experiência, embora seja subjetiva, por serem compostos por três propriedades que permitem o foco na atividade: objetivo, ferramenta e domínio. O objetivo é o que se pretende alcançar com a atividade. As ferramentas são os meios pelos quais se atingirá o objetivo e o domínio está relacionado com o contexto da atividade. Por exemplo, em um jogo, o objetivo poderia ser a catarse (o jogador deseja uma experiência catártica, em que há divergência do seu cotidiano, um escape), a ferramenta é o jogo que ele utiliza para atingir tal objetivo e o domínio é o contexto do jogo (o seu gênero e ambientação).

Calvillo-Gámez et al. (Calvillo-Gámez10, p. 50) identificam três qualidades necessárias à aplicação para que o usuário tenha foco na atividade: a ferramenta precisa ser funcional, isto é, ser capaz de realizar a tarefa a que se destina; a ferramenta precisa ser usável, isto é, suas propriedades são compatíveis com as necessidades do usuário, relacionamento realizado através de conceitos como eficiência, eficácia e *affordance*[12]; e estéticamente atraente, de forma que dentre várias ferramentas de uso e função semelhante, o fator que levo o usuário a selecionar aquela.

Através dessas propriedades e qualidades, pode-se avaliar jogos similares e diferentes em relação à experiência de jogo proporcionada ao jogador, seja

[12]O conceito de *affordance* pode ser entendido como a relação entre as ações possíveis de se realizar com a ferramenta e as ações possíveis percebidas.

através de entrevista com o jogador usando questionários pós-jogo, seja através de avaliações usando heurísticas realizadas por peritos na área. Na seção 5.6.2 vemos como avaliar a experiência de jogo de um jogador usando um questionário pós-jogo.

5.6.2 Avaliação da experiência de jogo

Como metodologia de avaliação da experiência de jogo, optamos pelo uso de questionários em momento pré-teste, para poder classificar o usuário conforme seu conhecimento anterior em jogos, e pós-teste para avaliar os impactos da experiência de jogo.

Quanto ao uso da estratégia de *think aloud* (Fullerton08, p. 264) para avaliação da experiência, optamos pela alternativa descrita por Hoonhout (Hoonhout08, p. 71), anotando as reações do jogador e realizamos uma entrevista pós-teste, procurando usar as informações coletadas durante o teste como guia para perguntas adicionais. O uso da alternativa de Hoonhout visa evitar a quebra da imersão que pode ser provocada ao instruir o participante a verbalizar sua experiência.

Os testes com o jogo desenvolvido foram conduzidos de acordo com as recomendações de *playtesting* de Fullerton (Fullerton08, pp. 252–269). Esta subseção detalha e justifica o uso das metodologias anteriormente citadas.

O framework CEGE

O *framework* CEGE (*Core Elements of the Game Experience*) proposto por (Calvillo-Gámez10) parte do entendimento de que a experiência de jogar envolve o videogame em si e a interação entre o jogador e o videogame. Esta interação, chamada de "puppetry"[13] por Calvillo-Gámez et al. (op. cit.), provê a ligação para o resultado da experiência. A metáfora da manipulação de marionetes (*puppetry*, ou titerismo) serve para descrever esta experiência de jogar, ou, mais precisamente, serve para descrever a parte interativa da experiência. *Puppetry*, de acordo com (Calvillo-Gámez10), descreve como o jogador começa se aproximando do videogame até que, eventualmente, o jogo sendo jogado é o resultado das ações do jogador.

Procurando contribuir para um melhor entendimento do *framework* de Calvillo-Gámez et al. (op. cit.), a presente dissertação propõe que os elementos

[13] *Puppetry* significa a produção, a criação ou a manipulação de *puppets* (marionetes). A tradução mais referenciada é "fantochada", o que, entretanto, não reflete o conceito. Optamos pela tradução titerismo.

centrais (*core elements*) de Calvillo-Gámez et al. (op. cit.) sejam apresentados conforme a organização de sua hierarquia:

- Elementos Principais da experiência de Jogar (i.e., *Puppetry* e *Videogame*)
  - Elementos Construtores que permitem a percepção dos Elementos Principais
    - Elementos Observáveis do Processo que são consequências dos Elementos Construtores

A Figura 5.7 apresenta esta hierarquia, junto como a indicação das variáveis latentes e observáveis propostas por (Calvillo-Gámez10). Os questionários para os jogadores são criados usando as variáveis observáveis (que são elementos que podem ser medidos diretamente).

		Puppetry		Videogame		Elementos Principais
Variáves Latentes	**Control**	**Ownership**	**Facilitators**	**Game-Play**	**Environment**	Elementos Construtores
Variáveis Observáveis	Small Actions	Big Actions	Time	Rules	Graphics	Elementos Observáveis
	Controllers	Personal Goal	Aesthetic Value	Scenario	Sound	
	Memory	You but not You	Previous Experiences			
	Point of View	Rewards				
	Goal					
	Something to do					

Figura 5.7: Elementos centrais da Experiência de Jogo. Adaptado de (Calvillo-Gámez10)

Os elementos centrais de Calvillo-Gámez et al. (op. cit.) também têm uma estrutura de relacionamento. O principal relacionamento é a constatação de que a experiência positiva, (i.e., *enjoyment* (diversão)), é alcançada pela percepção do *videogame* pelo jogador e pela interação dele com o *videogame* (*puppetry*). Isto é, os elementos *puppetry* e *videogame* produzem diversão, assim como a ausência deles provoca a experiência negativa oposta: a frustação. *Enjoyment* (diversão) é considerada uma variável latente do *framework*, junto com os outros 5 elementos construtores. Aliás, outras variáveis latentes podem ser consideradas: "frustação", "*puppetry*", "*videogame*" e a própria experiência CEGE.

O outro relacionamento vem da constatação de que Controle (*Control*) e Facilitadores (*Facilitators*) produzem Posse (*Ownership*). Na próxima seção, as

descrições dos Elementos Construtores (variáveis latentes) e de outras variáveis são apresentadas. A hierarquia do *framework* (Figura 5.7) e a estrutura de relacionamento entre as variáveis estão graficamente apresentadas na Figura 5.8. A descrição das variáveis observáveis e a explicação de como elas são usadas para criar o questionário podem ser encontradas na tese de doutorado de Eduardo Calvillo-Gámez (Calvillo-Gámez09).

Questionário pré-teste

Em Calvillo-Gámez et. al (Calvillo-Gámez10, p. 50) temos que experiências pretéritas afetam experiências futuras. Desse modo, aplicamos um questionário anterior ao teste para obter informações qualitativas como idade, sexo, e se o jogador já teve alguma experiência com jogos antes do teste e como ele se classificaria em nível de habilidade com jogos, dividindo entre casual e dedicado (i.e., *hardcore*). Essa diferenciação foi escolhida para auxiliar na compreensão dos resultados, baseado nos critérios de diferenciação apresentados na seção 2.2.1.

A expectativa é que jogadores dedicados tenham uma facilidade maior em assimilar a experiência de jogo que jogadores casuais. Essa classificação entre casual e dedicado é reforçada pelas perguntas sobre quantas horas o jogador dedica para jogos na semana e com quais gêneros está familiarizado. Também esperamos que os jogadores que se declararam como casuais escolham a dificuldade inicial como fácil (*easy*) ou normal (*medium*) e que os jogadores que se declararam como dedicados escolham normal (*medium*) ou difícil (*hard*).

Além desses dados, coletamos também informações sobre a média semanal de horas gastas jogando jogos de *videogame* e quais gêneros de jogos está familiarizado. Os gêneros foram adaptados dos supergêneros identificados no relatório de 2012 da ESA (ESA12) e dos gêneros identificados no Australian Centre For the Moving Image[14] por se aproximarem mais das classificações usadas em outras mídias com as quais os jogadores estão mais familiarizados, como revistas e *sites* especializados, do que a taxonomia de Gularte apresentada na Tabela 4.1. Tais dados servem como apoio à confirmação da autoclassificação do jogador e esperamos que jogadores familiarizados com os gêneros Shoot'em Up e Arcade tenham maior desempenho e facilidade em assimilar os conceitos do jogo.

Questionário pós-teste

Para avaliarmos a diferença de experiência do jogador com o jogo desenvolvido em ambas as versões, usamos o questionário de Elementos Centrais da

[14] `http://www.acmi.net.au/explore_game_genres.htm`

Experiência de Jogo (CEGE) de Calvillo-Gámez et al. (Calvillo-Gámez10), que se baseia no *framework* CEGE apresentado anteriormente nesta dissertação, usando uma escala Likert de sete pontos para as questões. O questionário é aplicado posteriormente à experiência de jogo com cada uma das versões e é usado para acessar os elementos centrais do modelo CEGE.

Para avaliar o questionário pós-jogo, usamos dois conjuntos de fatores[15] do modelo de Calvillo-Gámez et al., a saber:

Diversão: A sensação de diversão ou prazer ao jogar.

Frustração: A sensação de frustração ao jogar.

ECEJ: Elementos centrais da experiência de jogo. Refere-se a titerismo[16] e video-game conjuntamente.

Titerismo: Sensação de controle e domínio do jogo.

***Video-game*:** Refere-se à jogabilidade e ao ambiente, definido pelos elementos gráficos e sonoros do jogo.

Neste conjunto de fatores, titerismo e *video-game* estão correlacionados com diversão. Se há presença de elementos centrais da experiência de jogo, então a frustração deve ser baixa e não correlacionada (Calvillo-Gámez10, p. 65) (a frustração surge da ausência ou mau funcionamento dos elementos centrais da experiência de jogo), mas não há garantia de que diversão será positiva. Identificando a presença de ECEJ, titerismo e video-game são individualmente avaliados.

Controle: Refere-se à sensação de domínio do jogador sobre as ações que o jogo oferece, fazendo o jogo responder às suas ações, manipulando-o.

Facilitadores: Elementos subjetivos da experiência de jogo, como tempo que o jogadores está disposto a gastar com o jogo, valores estéticos do jogo e experiências anteriores com jogos similares.

Posse: Sensação quando o jogador toma as ações do jogo como suas extesões, como resultado de sua ação consciente sobre o jogo.

Ambiente: A apresentação do jogo, gráficos e sons.

***Game-play*:** Refere-se às regras e ao cenário do jogo.

[15]Nesta dissertação, usamos livremente os termos "variável latente" e "fator" como sinônimos.

[16]Tradução nossa para *puppetry*.

Este conjunto de variáveis busca avaliar a percepção do jogo formada pelo jogador pelas variáveis ambiente e *game-play* que produzem diversão (do conjunto de variáveis anteriormente apresentado). Controle gera posse, que gera diversão. Posse também é produzida por facilitadores para compensar a baixa sensação de controle, através da subjetividade de fatores como experiências anteriores com jogos similares, apreciação estética do jogo e o tempo que o jogador está disposto a gastar com o jogo.

A Tabela 5.2 mostra a quais variáveis cada questão está relacionada. As questões do questionário pós-jogo estão na seção A.2. A Figura 5.8 mostra a relação entre as variáveis do questionário e os elementos centrais da experiência de jogo.

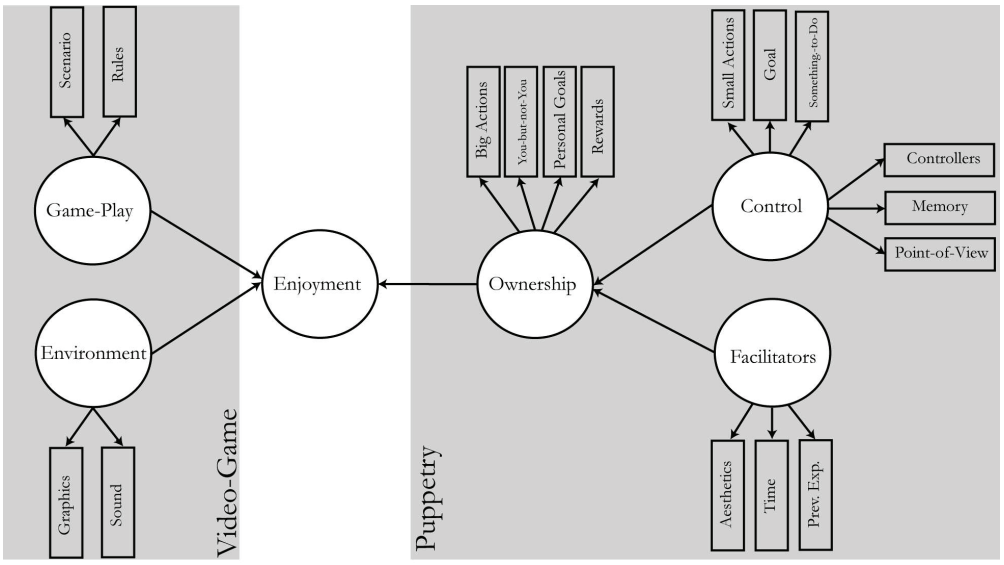

Figura 5.8: Relacionamento entre as variáveis do questionário CEGE, onde os círculos são as variáveis latentes e os retângulos as variáveis concretas e observáveis. Retirado de (Calvillo-Gámez10, p. 64).

Reconhecemos também a existência de outros questionários de avaliação da experiência de jogo, como o GEQ proposto por Brockmyer et al. (Brockmyer09) do Game Experience Lab da Eindhoven University of Technology e a proposta de Parnell (Parnell09) da London University. Entretanto, o modelo CEGE é o que melhor reflete o fato de que a experiência de interagir com um jogo é um fenômeno dual, no sentido de que é ao mesmo tempo um processo e um resultado.

Think aloud

Embora o método de *think aloud* tenha suas vantagens, como permitir compreender o que se passa na cabeça do jogador durante a experiência

Tabela 5.2: Relação das perguntas do questionário com os fatores da experiência de jogo, adaptada de (Calvillo-Gámez10, p. 65)

Itens	Fator
1, 4, 5	Diversão
2, 3	Frustração
6–38	Elementos Centrais da Experiência de Jogo
6–25, 38	Titerismo
26–37	Videogame
6–12, 25, 28	Controle
13–18	Facilitadores
19–25	Posse
26–31	Ambiente
32–37	Game-play

(*concurrent think aloud*), permitindo ao pesquisador inferir sobre o processo cognitivo do usuário a partir dos resultados observados e de suas decisões, sua maior desvantagem é interferir na experiência ao tirar o usuário do contexto da imersão para o contexto do teste, além da verbalização do usuário poder ser influenciada pelo contexto de teste (i.e., pode não se expressar naturalmente, como seria desejado) (Hoonhout08, p. 66–69). Uma alternativa é o uso de filmagem e pedir ao usuário para ver o vídeo e comentar o que pensou naquele momento (*retrospective think aloud*).

Usamos a alternativa descrita por Hoonhout em (Hoonhout08, p. 71), onde o usuário não é instruído para verbalizar sua experiência, mas seus comentários e reações são registrados. Hoonhout identifica que tal método precisa ser amparado por outros métodos de coleta de dados, como fazemos com o uso de entrevista e questionários.

Playtesting

Cada jogador jogou de forma alternada as versões com e sem dificuldade adaptativa, sem saber qual versão estava jogando. A alternância entre os jogadores de qual versão seria jogada primeiro deve-se à necessidade de se eliminar o viés no desempenho causado pelo aprendizado ao jogar uma das versões. Coletamos dados sobre o desempenho de cada jogador durante o jogo de modo a identificar e comparar o nível de dificuldade inicialmente escolhido e o nível de dificuldade percebido pela inteligência artificial, seja este ajustado pelo desempenho (na versão adaptativa) ou mantido (na versão adaptável).

Entrevista

Segundo Hoonhout (Hoonhout08, p. 72), perguntar ao usuário o que ele achou da experiência é o início e o objetivo da entrevista pós-teste.

Através da entrevista, é possível coletar dados como a opinião do participante, comparações com experiências anteriores, pensamentos, atitudes e ideias, complementando dados coletados através de outros meios durante o teste, como observação do protocolo verbal (*think aloud*) e *log* do sistema testado.

Para nossa entrevista com os participantes, escolhemos o formato semi-estruturado, onde predefinimos tópicos para discussão, mas a forma como a pergunta é feita ao participante e a ordem das perguntas é deixada em aberto, de modo a focar em determinados aspectos relativos a cada participante que foram observados durante o teste. Os tópicos encontram-se listados no apêndice A. Tomamos precauções na formulação das perguntas para evitar influenciar os participantes a determinados comportamentos ou respostas, visto que, por ser uma situação social, a entrevista pode ser afetada pelo desejo inconsciente do participante em agradar o entrevistador (Hoonhout08, p. 73).

5.6.3 Participantes

Para evitar viéses nos testes, foram selecionados participantes de ambos os sexos, de diferentes experiências anteriores com jogos e de variadas formações, entre funcionários do laboratórios TeCGraf e ICAD/Visionlab da PUC-Rio, alunos de graduação e pós-graduação dos departamentos de Informática e Artes & Design da PUC-Rio e do departamento de Informática da UFF. Ao todo, a população de testes foi de 35 participantes.

5.6.4 Roteiro dos testes

A aplicação dos testes com os usuários seguiu o seguinte roteiro:

1. Os participantes foram apresentados à pesquisa em que seriam testadores de variações de um jogo. Lembramos aos participantes que não seriam eles o objeto de teste e sim o jogo. Um termo de aceitação foi apresentado aos participantes, que foram instruídos a ler e em caso de aceite, assiná-lo. No termo, os participantes foram avisados de que dados de sua experiência de jogo seriam coletados de forma anônima para que fosse possível uma melhor avaliação do jogo sem identificá-los.

2. Foi aplicado um questionário pré-teste para coletar dados de experiência anterior do jogador.

3. Escolhido de forma alternada, cada participante jogou cada uma das versões do jogo (com ou sem adaptatividade) até que chegasse ao final

do mesmo ou que perdesse todas as vidas. Limitamos a uma única vez por versão.

4. O questionário pós-teste foi aplicado para avaliar a experiência de jogo do participante com a versão jogada.

5. Cada participante jogou a outra versão do jogo, relativa à versão anterior, até que chegasse ao final do demo ou que perdesse todas as vidas.

6. O questionário pós-teste foi aplicado para avaliar a experiência de jogo do participante com a versão jogada.

7. Foi realizada uma entrevista para coletar dados qualitativos a fim de avaliar as diferenças entre o sistema adaptativo e o sistema adaptável.

As perguntas realizadas nos questionários pré e pós teste e na entrevista podem ser conferidas no apêndice A.