



Pós-Graduação em Ciência da Computação

LEONARDO VIEIRA DE CARVALHO

Gameloop - Um Processo para Level Design de Jogos de Infinitos Ciclos



Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/posgraduacao

Recife
2015

LEONARDO VIEIRA DE CARVALHO

Gameloop - Um Processo para Level Design de Jogos de Infinitos Ciclos

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, como parte dos requisitos necessários à obtenção do título de Mestre em Ciência da Computação

Orientador: Geber Lisboa Ramalho

Recife
2015

Catálogo na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

C331g Carvalho, Leonardo Vieira de
Gameloop: um processo para level design de jogos de infinitos ciclos /
Leonardo Vieira de Carvalho. – 2015.
88 f.:il., fig., tab.

Orientador: Geber Lisboa Ramalho.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn,
Ciência da Computação, Recife, 2015.
Inclui referências.

1. Engenharia de software. 2. Design de jogos. I. Ramalho, Geber Lisboa
(orientador). II. Título.

005.1

CDD (23. ed.)

UFPE- MEI 2017-233

Dissertação de Mestrado apresentada por **Leonardo Vieira de Carvalho** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título “**Gameloop - Um Processo para Level Design de Jogos de Infinitos Ciclos**” orientada pelo **Prof. Geber Lisboa Ramalho** e aprovada pela Banca Examinadora formada pelos professores:

Profa. Patricia Cabral de Azevedo Restelli Tedesco
Centro de Informática/UFPE

Prof. Andre Menezes Marques das Neves
Departamento de Design / UFPE

Prof. Geber Lisboa Ramalho
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 31 de agosto de 2015.

Profa. Edna Natividade da Silva Barros
Coordenadora da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

Dedico essa dissertação, aos meus pais e irmã, que sempre estiveram comigo apoiando e incentivando. A familiares próximos que não estão mais entre nós, mas que deixam saudade. E, por fim, ao Professor Geber Lisboa Ramalho que, com sua valiosa orientação, tornou possível a concretização deste trabalho.

Agradecimentos

Agradeço primeiramente à minha família, que sempre me apoiou e tornou possível a minha presença em uma das melhores instituições de ensino do Brasil. Agradeço ao meu pai e minha mãe, que nunca mediram esforços para garantir que eu tivesse acesso a uma educação de qualidade. Agradeço à minha irmã que sempre foi minha amiga e esteve ao meu lado nos momentos mais difíceis. Agradeço também a todos os amigos que compartilharam comigo os melhores e piores momentos, prefiro não listar nomes para evitar esquecer algum, mas ninguém sabe melhor do que eles o que foi necessário para que eu chegasse até aqui. Agradeço aos professores do Centro de Informática da UFPE, que foram parte essencial da minha formação. Dentre eles gostaria de agradecer especialmente ao professor Geber Ramalho, que me orientou durante o desenvolvimento desse trabalho e que tornou possível a sua concretização. Gostaria de agradecer também aos Professores Doutores Patricia Cabral de Azevedo Restelli Tedesco e Andre Menezes Marques das Neves que aceitaram fazer parte da banca. Por fim, agradeço a todos que fizeram parte dessa longa caminhada que ainda promete muito.

*“O homem certo no lugar errado pode fazer
toda a diferença do mundo.”*

— G-Man / Half Life 2 (tradução livre)

Resumo

Jogos de infinitos ciclos são aqueles nos quais as partidas podem ter duração potencialmente infinita, com a apresentação contínua de desafios ao jogador, e tais partidas são comumente finalizadas apenas quando o jogador é afetado por um critério de derrota, como é o caso de jogos dos gêneros Endless Runner ou Roguelike. Técnicas de geração procedimental de conteúdo são usadas em jogos nas diversas etapas do processo de desenvolvimento: na criação de texturas, efeitos sonoros, músicas, personagens, mapas, níveis, entre outros. A maioria das técnicas estudadas na elaboração deste trabalho geram uma grande variedade de conteúdo com o propósito de criar níveis ou estruturas similares finitas. Essas estruturas satisfazem as demandas de jogos que são segmentados em níveis ou capítulos, mas para jogos infinitos não são o suficiente. Além disso, um elevado fator de replay é algo que aumenta significativamente o interesse do jogador, e é nesse aspecto que a geração procedimental de conteúdo pode auxiliar, por tornar possível a geração de uma grande variedade de partidas com um baixo custo de produção. O objetivo deste trabalho é gerar de forma automática partidas para jogos de infinitos ciclos que sigam restrições controláveis por game designers. Para alcançar tal objetivo, foi definido um processo em 4 etapas que visa segmentar de forma administrável a geração de conteúdo e atuar sobre cada uma destas de forma eficiente. Este processo foi então aplicado no jogo comercial Boney the Runner, e por fim foi feita a avaliação do desempenho do processo através de experimentos com grupos de jogadores.

Palavras-chave: Geração procedimental de conteúdo. Jogos. Design de jogos. Ajuste dinâmico de dificuldade. Level design automático.

Abstract

This work aims to study the application of procedural content generation techniques in the development of games of endless cycles. Games of endless cycles are those on which the sessions can be potentially infinite, with the continuous presentation of challenges to the player, as is the case with games of the Endless Runner or Rogue-like genres. Procedural content generation techniques are used in games in several aspects of the development process: in the creation of textures, sound effects, music, characters, maps, among others. Most of the techniques studied in the making of this work generate a large variety of content with the purpose of creating levels and similar finite structures. These structures satisfy the demands of games that are segmented in levels or chapters, but for games of endless cycles, they are not enough. Besides that, a good replay value is something that greatly increases the player's interest in a game, and it's on this aspect that procedural content generation can help, by making possible the creation of new content at a low production cost. The goal of this work is to generate sessions for endless games automatically while following restrictions controlled by a game designer. To reach this goal, a four-step process was defined with the purpose of segmenting the procedural content generation in a manageable way and to act on each of these steps efficiently. This process was then applied in the commercial game Boney the Runner, and an evaluation of the process was made by analyzing gameplay sessions of groups of players.

Keywords: Procedural content generation. Games. Game design. Dynamic difficulty adjustment. Automatic level design.

Lista de figuras

Figura 1 – Tipos de conteúdo de jogos que podem ser gerados de forma procedural	20
Figura 2 – Diagrama de Flow	24
Figura 3 – Exemplo de curva de progresso	26
Figura 4 – Imagem do jogo Super Mario Bros na qual é possível identificar blocos de interrogação, um inimigo Goomba e um Cano	36
Figura 5 – Um segmento do jogo Jetpack Joyride, no qual diversas moedas estão dispostas de forma a formar a sentença “COINS!!”	38
Figura 6 – Mapa de um calabouço no jogo Diablo 3. Em tal mapa é possível identificar grandes segmentos que são conectados por corredores menores	39
Figura 7 – Processo para a geração automática de conteúdo para jogos de infinitos ciclos	40
Figura 8 – Exemplo de Proporção de Segmentos	47
Figura 9 – Exemplo de Curva de Dificuldade	47
Figura 10 – Uma partida do jogo Boney the Runner Runner	51
Figura 11 – Uma partida do jogo Boney the Runner na qual o jogador cometeu um erro e por conta disso o avatar teve que escalar o obstáculo . . .	52
Figura 12 – Uma partida do jogo Boney the Runner na qual o avatar desliza sobre o musgo	53
Figura 13 – Uma partida do jogo Boney the Runner na qual o avatar anda sobre a lama	54
Figura 14 – Uma partida do jogo Boney the Runner na qual o avatar está cercado por fantasmas	55
Figura 15 – Um segmento do jogo Boney the Runner	56
Figura 16 – Três variações de tumba, da esquerda para a direita: baixa, média e alta	59
Figura 17 – Dois segmentos diferentes gerados a partir da mesma entrada . . .	60
Figura 18 – Dois exemplos de segmentos gerados no editor do jogo Boney the Runner, em cada um dos segmentos é possível identificar o mesmo elemento com larguras diferentes	62
Figura 19 – Curvas de dificuldade finita e dificuldade cíclica originais	69
Figura 20 – Distribuição dos tipos de segmento de acordo com o nível de dificuldade para a curva de Dificuldade Finita	71
Figura 21 – Distribuição dos tipos de segmento de acordo com o nível de dificuldade para a curva de Dificuldade Cíclica	72

Figura 22 – Curvas de dificuldade finita e dificuldade cíclica adaptadas	73
Figura 23 – Gráfico com resultados da análise de geração de segmentos	76
Figura 24 – Distribuição dos contadores de acerto de classificação.	78
Figura 25 – Distribuição das respostas da primeira pergunta	80

Lista de tabelas

Tabela 1 – Configurações das redes A e B	68
Tabela 2 – Resultados obtidos com redes A e B	68

Lista de abreviaturas e siglas

2D	Duas dimensões
3D	Três Dimensões
CIn	Centro de Informática
CSP	Constraint Satisfaction Problem
DIF	Dificuldade Reclassificada
EGDTO	Diferença de tempo em relação ao tempo ótimo
GPC	Geração Procedimental de Conteúdo
MLP	Multilayer Perceptron
MSE	Mean Squared Error
PSAma	Porcentagem de Segmentos de Alívio máxima
PSAmi	Porcentagem de Segmentos de Alívio mínima
PSBma	Porcentagem de Segmentos Bloqueantes máxima
PSBmi	Porcentagem de Segmentos Bloqueantes mínima
PSPma	Porcentagem de Segmentos de Preparação máxima
PSPmi	Porcentagem de Segmentos de Preparação mínima
ROC	Receiver Operating Characteristic
RPG	Role Playing Game
SA	Segmento de Alívio
SB	Segmento Bloqueante
SP	Segmento de Preparação
UFPE	Universidade Federal de Pernambuco

Sumário

1	INTRODUÇÃO	15
1.1	Considerações Iniciais	15
1.2	Motivação	17
1.3	Objetivos	18
1.4	Abordagem	18
1.5	Estrutura da Dissertação	19
2	QUESTÕES ENVOLVIDAS NA GERAÇÃO DE CONTEÚDO PARA JOGOS DE INFINITOS CICLOS	20
2.1	Introdução	20
2.2	Tipo de Conteúdo	20
2.3	Fator de Replay	23
2.4	Dificuldade	23
2.5	Curva de Progresso	25
2.6	<i>Playability</i>	26
2.7	Método de Geração do Conteúdo	27
2.8	Controle de Alto Nível	27
2.9	Geração para Jogos de Infinitos Ciclos	28
2.10	Resumo	28
3	ESTADO DA ARTE	30
3.1	Introdução	30
3.2	Técnicas Analisadas	30
3.3	Resumo	34
4	SOLUÇÃO PROPOSTA	36
4.1	Introdução	36
4.2	Definição de Elementos de Jogo e Segmento	36
4.3	O Processo	39
4.4	Geração de Segmentos	42
4.5	Classificação de Segmentos	42
4.6	Definição da Curva de Progresso	43
4.6.1	Sequenciamento de Segmentos	45
4.6.2	Proporção dos Segmentos	45
4.6.3	Conversão entre Curvas	46
4.7	Escolha do Segmento	48

4.8	Resumo	49
5	ESTUDO DE CASO	50
5.1	Introdução	50
5.2	Jogo utilizado no estudo	50
5.3	Aplicação do Processo	55
5.4	Geração de Segmentos	57
5.4.1	Geração de Segmentos Brutos	58
5.4.2	Validação de Segmentos	60
5.5	Classificação de Segmentos	61
5.5.1	Elementos Controláveis	61
5.5.2	Elementos de Gameplay	64
5.5.3	Modelagem da Dificuldade	66
5.5.4	Criação das Redes Neurais	67
5.6	Curva de Progresso	68
5.6.1	Proporção dos Segmentos	70
5.6.2	Curvas de Dificuldade	72
5.7	Escolha do Segmento	73
5.8	Resumo	74
6	AVALIAÇÃO	75
6.1	Introdução	75
6.2	Geração de Segmentos	75
6.3	Classificação de Segmentos	76
6.4	Curva de Progresso	79
6.5	Escolha dos Segmentos	81
6.6	Resumo	82
7	CONCLUSÕES E TRABALHOS FUTUROS	83
7.1	Considerações Finais	83
7.2	Contribuições	83
7.3	Trabalhos Futuros	84
	Referências	86

1 INTRODUÇÃO

O presente capítulo está organizado em seis partes: uma breve análise sobre como a geração procedimental de conteúdo tem afetado o desenvolvimento de jogos; as motivações que levaram ao estudo e desenvolvimento do trabalho em questão; o que o presente trabalho objetiva realizar; a forma como se pretende alcançar o objetivo do trabalho; os objetivos alcançados e por fim uma exposição da forma como este documento está estruturado.

1.1 Considerações Iniciais

A geração procedimental de conteúdo, ou GPC, consiste na utilização de algoritmos executados em computadores para realizar a geração automática de diversos tipos de conteúdo. No entanto, estabelecer uma definição precisa do que é “conteúdo” ainda é uma tarefa que se prova difícil. Togelius et al. (2011a) menciona a questão de que, para indivíduos de campos diferentes, conteúdo também pode ter significados diferentes: um pesquisador gráfico tem uma visão de conteúdo diferente de um *game designer* por exemplo. No entanto, mesmo com tal dificuldade em definir precisamente o que é conteúdo, Hendrikx et al. (2013) estabeleceu uma taxonomia dos tipos de conteúdo que torna clara a abrangência da geração procedural de conteúdo. Essa taxonomia será explicada em mais detalhes no capítulo 2, mas exemplos de conteúdo, geralmente criado por artistas gráficos, músicos e *game designers* (roteiristas), incluem: texturas, objetos, personagens, cenários, paisagens, sons e música, níveis, regras, textos, *quests*, etc.

Diante dos elevados investimentos de tempo e dinheiro associados ao desenvolvimento de jogos digitais (MORRIS, 2010; TAKATSUKI, 2007), em particular de seu conteúdo, é comum que diversas produtoras de jogos recorram à aplicação de técnicas de GPC em seus jogos. Não só tais técnicas costumam trazer uma maior variedade de conteúdo, como também custam significativamente menos quando comparadas à geração manual. Além disso, existem certos tipos de interações que apenas são possíveis com a utilização da GPC, como é o caso da adaptação do jogo ao jogador (SHAKER; YANNAKAKIS; TOGELIUS, 2010), o que cria experiências mais únicas e divertidas.

O início da utilização de técnicas de GPC em jogos data da década de 80, com jogos como *Rogue* (WICHMAN, 1997), um role-playing game gráfico no qual uma quantidade virtualmente infinita de níveis são gerados de forma *online*. Apesar de os níveis gerados não apresentarem uma complexidade muito elevada quando

comparados ao que é possível alcançar com a produção manual, o jogo mostrou que é possível elevar consideravelmente o fator de *replay* através da geração não-supervisionada de níveis (COMPTON; MATEAS, 2006). Por fator de *replay* entende-se o potencial que um jogador tem para continuar o jogo mesmo depois de tê-lo completado (WOLF, 2012).

Desde então, a utilização de técnicas de GPC apenas tem crescido na indústria de jogos, tanto para a geração de conteúdo *offline*, quanto na geração de conteúdo *online* (TOGELIUS et al., 2011b). Não há restrição quanto ao tipo de conteúdo que pode ser gerado de forma procedimental, texturas, sons, mapas, cidades, níveis e até mesmo sistemas inteiros podem ser gerados de forma algorítmica, mas com variados níveis de complexidade.

Alguns exemplos que ilustram bem o potencial da geração procedimental de conteúdo incluem os jogos “.kkrieger” (FARBRAUSCH, 2004) e Zettai Hero Project (NIS AMERICA, 2010). No caso do jogo “.kkrieger”, um jogo 3D de tiro em primeira pessoa, todos os artefatos do jogo são gerados de forma procedimental, o que inclui texturas, modelos tridimensionais, efeitos sonoros e cenários, entretanto nenhum aspecto de gameplay do jogo é modificado pela geração procedimental. Já no caso do jogo Zettai Hero Project, um jogo rogue-like, a geração procedimental é utilizada para aumentar o fator de replay do jogo através da geração de níveis, inimigos e equipamentos, mas não são usadas técnicas de GPC para a criação de artefatos como texturas e sons.

Levando em conta o potencial que a GPC apresenta no desenvolvimento de jogos, este trabalho tem como foco principal um segmento de jogos denominados de jogos de infinitos ciclos. A escolha em tal tipo de jogo se deve à considerável popularização de jogos desse gênero, em particular da categoria *endless runner*, a qual tem apresentado um considerável crescimento e consolidação de mercado (SCHICK, 2013). Diante de tal situação, a empresa do segmento de jogos BigHut Games (BIGHUT GAMES, 2012b), da qual o autor deste trabalho é sócio, decidiu introduzir o seu próprio jogo de tal categoria no mercado, o Boney the Runner (BIGHUT GAMES, 2012a). No entanto, durante o desenvolvimento a empresa se deparou com problemas para definir o *design* das partidas para o jogo, e ao recorrer à academia não identificou soluções que fossem diretamente aplicáveis à categoria de jogo em questão. Dado isso, foi necessário definir um processo com base nas técnicas já existentes para jogos convencionais.

Este tipo de segmento abrange diversos gêneros existentes na indústria, dentre os quais é possível destacar gêneros como *endless* e *roguelike* (KALOGIROU, 2012). Em um jogo do tipo *endless runner*, o jogador deve controlar o avatar enquanto este se desloca em um circuito infundável e uma partida apenas tem fim quando uma condição de derrota é alcançada. Já em jogos do tipo *roguelike* o jogador deve controlar o

avatar enquanto este se desloca em calabouços que normalmente são apresentados através de uma visão de cima para baixo, e enquanto explora o calabouço o jogador encontra tesouros, inimigos, aliados e caminhos que levam aos próximos pisos de tal calabouço. O que há de comum em jogos desse tipo é que em ambos uma partida de jogo pode ser potencialmente infinita, ou seja, desde que o jogador possua as habilidades necessárias para garantir que a condição de derrota não seja alcançada, ele pode continuar jogando indefinidamente. Por serem gêneros dependentes de um elevado fator de replay, é importante que as partidas em jogos desses tipos apresentem uma boa variedade, e é nesse aspecto que a GPC pode auxiliar.

1.2 Motivação

Como a popularidade de gêneros de jogos de infinitos ciclos ressurgiu recentemente, e se ampliou com jogos como Jetpack Joyride (HALFBRICK STUDIOS, 2011), materiais a respeito da geração de níveis para jogos de infinitos ciclos ainda são escassos ou proveem soluções orientadas principalmente a jogos que se utilizam do conceito de níveis, algo que normalmente não está presente em jogos desse gênero.

Muitas das soluções existentes para a geração de níveis apresentam a possibilidade de gerar quantidades virtualmente infinitas de fases para jogos, no entanto tais fases constituem de regiões desconexas e independentes, de forma que não existe um fluxo contínuo entre uma fase e outra. Em um jogo de infinitos ciclos nem sempre existe o conceito de níveis, e comumente a partida consiste em um desafio contínuo, gerado sob demanda e que respeita uma série de limitações determinadas por um *game designer*.

Dessa forma, existe a necessidade de definir um processo e criar ferramentas que auxiliem *game designers* na estruturação de partidas para jogos desse tipo.

É importante destacar que tal processo deve manter o controle do *designer* sobre o conteúdo gerado, uma vez que há grandes vantagens com a automatização da geração de conteúdo, mas é necessário que exista a capacidade de direcionar e influenciar de forma direta ou indireta o conteúdo gerado (HENDRIKX et al., 2013). Além disso, a experiência do profissional – tanto como um *designer* quanto como um jogador – lhe garante uma intuição do que torna o jogo divertido ou não (ADRIAN; LUISA, 2013).

Outro ponto relevante é a necessidade de balancear o conteúdo gerado de forma a melhorar a experiência do usuário (PEDERSEN; TOGELIUS; YANNAKAKIS, 2009). Apenas o processo de criação de níveis normalmente não é o suficiente para gerar uma experiência positiva ou divertida para o jogador, por isso é necessário que tais níveis sejam devidamente balanceados ou classificados, idealmente de forma

automática, para que se possa modelar a experiência de jogo desejada.

1.3 Objetivos

Este trabalho objetiva propor métodos e ferramentas que atuem em todas as etapas identificadas no processo de level design de jogos de infinitos ciclos de forma a automatizar tal processo, permitindo, ao mesmo tempo, que o *game designer* tenha algum controle do conteúdo a ser gerado pelas razões definidas na subseção anterior.

A automatização do processo tem dois propósitos: a redução de custo de desenvolvimento de níveis e o aumento do fator de replay durante as partidas. A redução do custo de desenvolvimento é algo que beneficia os desenvolvedores de jogos, uma vez que isto permite desenvolver jogos ricos em conteúdo com um orçamento limitado, ou mesmo para estúdios com grandes orçamentos, esta redução torna possível adicionar ainda mais variedade aos níveis de um jogo. O fator replay por sua vez é incrementado significativamente, dado que os jogadores terão acesso a uma variedade significativa de níveis de jogo a qual não seria possível se *game designers* precisassem construir cada nível manualmente.

A manutenção do controle do conteúdo por sua vez visa dar ao *game designer* a liberdade de gerar partidas que sigam certas restrições as quais fazem sentido para a experiência do usuário. Dessa forma, é possível combinar a elevada produtividade e grande variabilidade inerente a conteúdos gerados de forma procedimental à modelagem de experiência desejada por um *game designer* humano.

1.4 Abordagem

Para fins deste trabalho, a geração de níveis para jogos de infinitos ciclos foi modelada por nós como um processo com 4 etapas:

- 1) Geração de segmentos - Nesta etapa são gerados diversos segmentos jogáveis que serão inseridos no jogo. Tais segmentos podem ser encarados como pequenos blocos nos quais o jogador deve desviar de obstáculos para continuar jogando.
- 2) Classificação de segmentos - Nesta etapa, os segmentos gerados na etapa anterior são de alguma forma avaliados e separados de acordo com critérios estabelecidos, como por exemplo dificuldade. Por exemplo, podemos ter segmentos “fáceis”, “medianos” e “difíceis”.
- 3) Definição de como os segmentos serão posicionados no jogo - Nesta etapa, é definida a ordem com que as categorias de segmentos (segundo sua classificação na etapa 2) são dispostos no jogo ao longo da partida. Por exemplo, o

algoritmo pode decidir que haverá 3 segmentos fáceis seguidos de um difícil, um mediano, um fácil, um difícil, etc. É importante ressaltar que tal definição deve ser robusta o suficiente para tratar partidas longas, ou seja, ela deve estar preparada para partidas potencialmente infinitas.

- 4) Escolha do segmento a ser posicionado - Com base na definição feita na etapa 3 do processo, deve ser feita a escolha do segmento a ser posicionado dentre as opções existentes. Por exemplo, como existem diversos segmentos “difíceis”, é preciso escolher um dentre eles.

Para realizar a implementação e teste da ferramenta desenvolvida para este trabalho, foi utilizado o jogo para dispositivos móveis Boney the Runner, título desenvolvido pela produtora BigHut Games, o qual disponível no mercado desde outubro de 2012 e até a data da elaboração deste documento conta com mais de um milhão de *downloads* e três mil usuários ativos por dia. Para testar e validar os resultados do trabalho, foi feita a coleta de dados proveniente de partidas de jogadores com vários níveis de experiência e habilidade com o jogo.

1.5 Estrutura da Dissertação

A estrutura desse documento, após esse capítulo introdutório, tem a seguinte disposição:

Capítulo 2 – Questões Envolvidas na Geração de Conteúdo: Neste capítulo, será feita uma análise no que está envolvido e deve ser analisado na geração de conteúdo para jogos, em especial para jogos infinitos.

Capítulo 3 – Estado da Arte: Este capítulo apresenta um estudo das atuais soluções disponíveis na geração procedimental de conteúdo para jogos.

Capítulo 4 – Solução Proposta: Neste capítulo serão detalhados o processo definido para a geração de conteúdo e as técnicas utilizadas em cada uma das etapas do processo.

Capítulo 5 – Estudo de Caso: A solução proposta é aplicada em um jogo comercial com as devidas adaptações necessárias.

Capítulo 6 – Avaliação: Este capítulo detalha a avaliação das técnicas aplicadas no jogo utilizado para estudo de caso.

Capítulo 7 – Conclusões e Trabalhos Futuros: Espaço dedicado à apresentação das conclusões deste trabalho, suas contribuições e perspectiva de trabalhos futuros.

2 QUESTÕES ENVOLVIDAS NA GERAÇÃO DE CONTEÚDO PARA JOGOS DE INFINITOS CICLOS

Este capítulo tem como objetivo analisar as diversas questões que estão relacionadas à geração de conteúdo para jogos e como tais questões afetaram o desenvolvimento do trabalho.

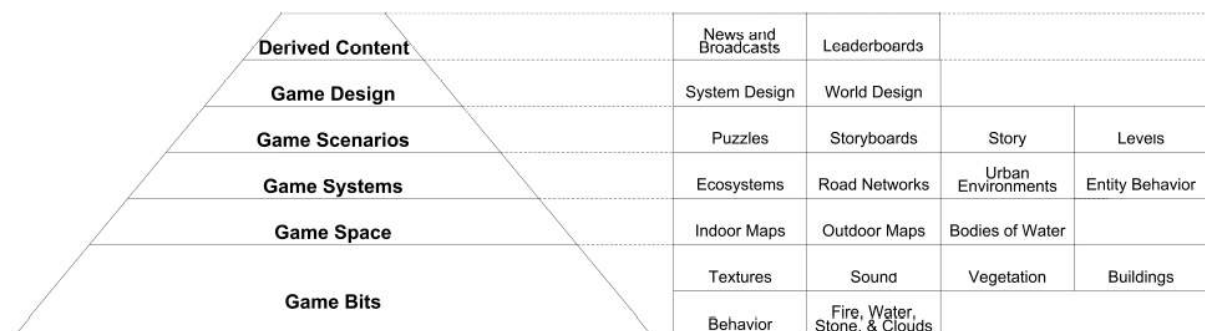
2.1 Introdução

A geração de conteúdo para jogos é uma tarefa muitas vezes complexa por precisar tratar diversos problemas relevantes que afetam diretamente a experiência do jogador e a qualidade do jogo. Neste capítulo será feita uma análise de alguns desses aspectos e de como eles impactam a geração de conteúdo.

2.2 Tipo de Conteúdo

O tipo de conteúdo a ser gerado também é o aspecto mais importante da geração procedural por afetar diretamente qual abordagem deve ser utilizada. Hendrikx et al. (2013) apresenta uma detalhada separação e classificação dos tipos de conteúdos que podem ser gerados de forma procedural, como é possível visualizar na figura abaixo.

Figura 1 – Tipos de conteúdo de jogos que podem ser gerados de forma procedural



Fonte: Hendrikx, Mark et al., 2013

Esta figura apresenta uma pirâmide composta pelos 6 tipos macro de conteúdo que podem ser gerados de forma procedural. A base da pirâmide contém os itens fundamentais utilizados na geração de conteúdo e cada nível utiliza o item imediatamente abaixo como elemento de construção. Por exemplo, para construir Game Systems (Sistemas de Jogo) são utilizados Game Space (Espaço de Jogo). A seguir, cada um desses elementos é descrito de acordo com o trabalho de Hendrikx et al. (2013), traduzido e adaptado pelo autor:

- 1) Game Bits (Fragmentos de Jogo) – São as unidades elementares do conteúdo do jogo, as quais tipicamente não engajam com o usuário quando consideradas de forma independente. Distinguimos entre os Fragmentos de Jogo os fragmentos concretos e abstratos: fragmentos completos, como árvores podem ser itens com os quais se interage no mundo simulado, enquanto fragmentos abstratos, como texturas e sons, precisam ser combinados para formar um pedaço concreto. Seis categorias principais de fragmentos de jogo foram identificados: texturas (abstrato), som (abstrato), vegetação (concreto), construção (concreto), comportamento (abstrato) e fogo, água, pedra e nuvem (concreto).
- 2) Game Space (Espaço de Jogo) – O espaço de jogo é o ambiente no qual o jogo acontece, e é parcialmente preenchido com fragmentos de jogo entre os quais o jogador navega. O espaço de jogo pode ser definido tanto de forma concreta quanto abstrata. Espaços concretos se relacionam de forma próxima com a forma como humanos percebem o local; eles podem ser florestas, labirintos, planícies, etc. O tabuleiro em um jogo de xadrez é um exemplo de espaços de jogo abstratos. O espaço de jogo tem um papel fundamental em criar um jogo interessante, uma vez que os jogadores frequentemente constroem suas interpretações do jogo a partir do espaço de jogo. Três categorias principais de espaço de jogo foram identificados: mapas interiores (abstrato ou concreto), mapas exteriores (abstrato ou concreto) e corpo de água (concreto).
- 3) Game Systems (Sistemas de Jogo) – O uso de teorias complexas de sistemas e modelagem para gerar ou simular partes de um jogo não é incomum. Sistemas de jogo podem ser abstratos, por exemplo sistemas que definem o relacionamento entre vegetação e características de mapas externos, ou concretos, como a simulação de cidades e redes de cidades. O uso de sistemas de jogo pode fazer jogos mais críveis e, por consequência, atraentes. Muitos sistemas usados em jogos são similares a sistemas e modelos complexos de livros. Quatro categorias principais de sistemas de jogo foram identificados: ecossistemas (abstrato ou concreto), sistemas de estradas (abstrato ou concreto), ambientes urbanos (abstrato ou concreto) e comportamento de entidades (abstrato).
- 4) Game Scenarios (Cenários de Jogo) – Cenários de jogo descrevem, comumente de forma transparente ao usuário, a forma e ordem na qual eventos de jogo se desdobram. É possível distinguir dois tipos de cenários de jogo, abstrato e concreto. Os cenários de jogo abstratos descrevem como outros objetos se inter-relacionam. Os cenários de jogo concretos são explicitamente apresentados no jogo, por exemplo como parte de uma narrativa de jogo. Quatro categorias principais de cenário de jogo foram identificadas: quebra-cabeças (abstrato),

storyboards (abstrato ou concreto), a história (abstrato ou concreto), o conceito de níveis (abstrato ou concreto).

- 5) Game Design (Design de Jogo) – O design de um jogo é composto por conteúdo tal como regras (o que pode ser feito no jogo?) e objetivos (o que o jogador está tentando alcançar?); um componente estético, tal como arco dramático ou tema gráfico, também são elementos importantes no design. Um jogo pode ser visto como uma instância de um design de jogo, no qual os parâmetros das regras e do objetivo foram estabelecidos. Um design de jogo pode referir a conteúdo de jogo de todos os tipos descritos até então, incluindo a referência recursiva a outros conteúdos de design de jogo. Design de jogo pode ser complementado pela geração de jogos (semi-) automática ou através do fornecimento de ferramentas que ajudam o designer a converter ideia em conteúdo de design de jogo. Duas categorias principais de design de jogo foram identificadas: o design do sistema (abstrato) e design do mundo (concreto).
- 6) Derived Content (Conteúdo Derivado) – Conteúdo derivado é definido como conteúdo que é criado como um produto secundário do mundo do jogo. Fazer isto pode aumentar tremendamente a sensação de imersão que o jogador tem com o mundo do jogo, já que os jogadores registram suas experiências dentro do jogo para revisão dentro ou fora do jogo; o conceito de “jogo além do jogo” foi a base para a noção de meta jogo. Duas categorias principais de conteúdo derivado foram identificadas: novidades e transmissões (concreto) e placares (abstrato).

Como é possível observar, praticamente qualquer tipo de conteúdo pode ser gerado de forma procedimental, seja música, efeitos sonoros, terrenos, texturas, comportamento de inteligência artificial, missões, itens, níveis, calabouços, entre outros. Cada tipo de conteúdo demanda uma abordagem diferente e apresenta complexidades diferentes.

Por conta disso, é necessário saber de antemão o que se deseja gerar para que possa se atuar de forma eficiente. E ainda mais importante: caso se pretenda gerar mais de uma forma de conteúdo simultaneamente, é necessário estabelecer uma clara separação entre cada um. Por exemplo: se um desenvolvedor deseja gerar de forma procedimental texturas para o cenário e níveis para o jogo, não se deve combinar as duas técnicas, do contrário é possível que ajustes ou modificações em um dos aspectos do jogo afete indevidamente outro aspecto.

Com isto em mente, este trabalho atua sobre o item 4, Cenários de Jogo, e mais especificamente no que diz respeito à categoria de níveis.

2.3 Fator de Replay

O fator de replay consiste no potencial que um jogo tem para ser jogado repetidas vezes enquanto ainda garante entretenimento ao usuário. A geração procedimental de conteúdo é vista como uma forma muito eficiente de aumentar o fator de replay a um baixo custo de desenvolvimento (SHAKER; YANNAKAKIS; TOGELIUS, 2010), mas ainda tem uma aplicação limitada principalmente para jogos de plataforma (COMPTON; MATEAS, 2006), um gênero muito similar ao abordado neste trabalho.

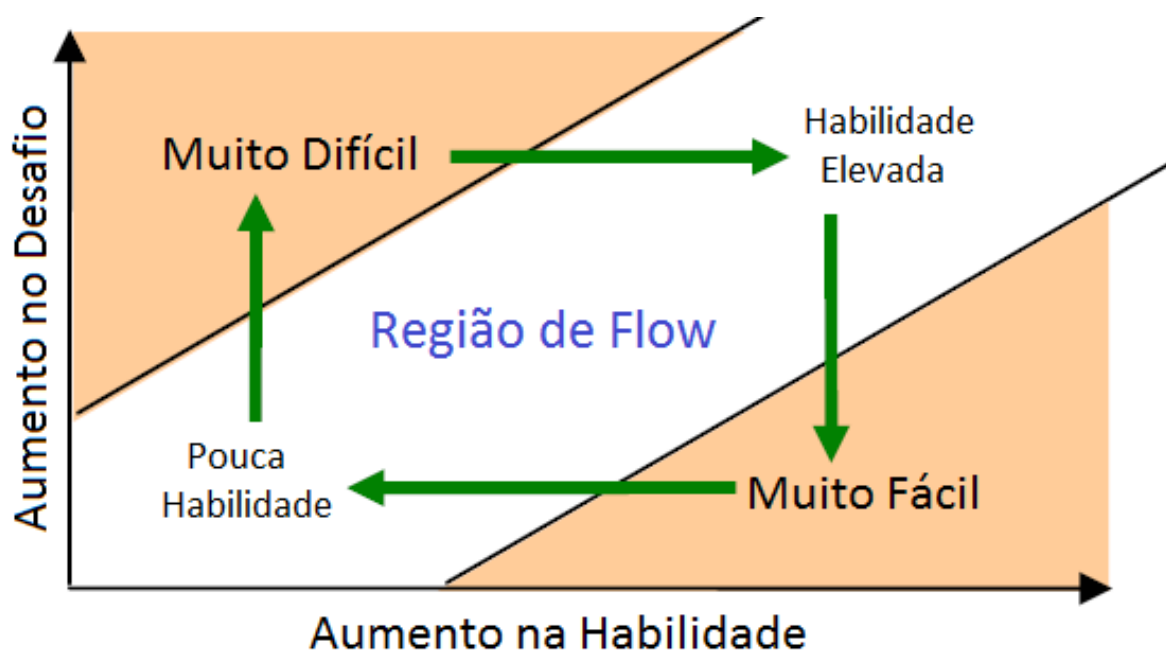
Este elemento é buscado na maioria dos jogos por manter os jogadores por mais tempo dentro dos jogos, algo desejado pelos desenvolvedores. No caso de jogos de infinitos ciclos, o fator de replay pode ser aumentado através da introdução de novos conteúdos no jogo, como elementos de personalização, algo que foge do nosso escopo, ou de uma maior variedade de desafios que o jogador pode encontrar nas partidas.

2.4 Dificuldade

A dificuldade é outro elemento muito importante na geração de conteúdo para jogos (HUNICKE; CHAPMAN, 2004) e deve ser observada atentamente. É necessário que durante o jogo exista um equilíbrio entre o nível de dificuldade apresentada ao jogador e a habilidade da qual o jogador dispõe.

Se o jogo consegue manter um equilíbrio entre esses dois elementos, é alcançado o que se chama de estado de *Flow*, como pode ser visto na Figura 2. Nesse estado, o usuário está totalmente imerso no jogo e apresenta um bom grau de aproveitamento, mas é importante ressaltar que, apesar de o conceito de Flow estar intimamente relacionado com diversão, eles não são a mesma coisa (KOSTER, 2013). Comumente diversão diz respeito a quão prazerosa é a experiência (SORENSEN; PASQUIER, 2010a), enquanto *Flow* está associado à imersão.

Figura 2 – Diagrama de Flow



Fonte: Hunicke, Robin;Chapman, Vernell, 2004

Ainda no que diz respeito à dificuldade, se esta estiver muito acima da habilidade do jogador, este poderá achar o jogo difícil e se frustrar. Se a dificuldade do jogo estiver muito abaixo da habilidade do jogador, é possível que este perceba o jogo como muito fácil, o que pode deixá-lo entediado. Em ambos os casos, representados respectivamente pelos cantos superior esquerdo e inferior direito do gráfico presente na Figura 2, o jogador pode perder o interesse pelo jogo.

Dessa forma, mesmo que não seja necessário garantir que os jogadores alcancem o estado de Flow para que um jogo seja divertido, é importante que a dificuldade do jogo esteja sempre em um nível aceitável quando comparada à habilidade do jogador. Como será visto no capítulo 4, o jogo utilizado para aplicar a ferramenta desenvolvida utilizava inicialmente um gráfico de dificuldade para definir que tipos de obstáculos o jogador irá encarar, algo que foi posteriormente substituído por uma curva de progresso automática. Além disso, é importante que a dificuldade de cada trecho de jogo seja medida corretamente de forma que a experiência desejada seja transmitida ao jogador.

Um conceito que está intimamente relacionado com a dificuldade é a Curva de Progresso, que será apresentada a seguir.

2.5 Curva de Progresso

Curvas de dificuldade são exploradas de diversas formas na academia (ADRIAN; LUISA, 2013; SORENSON; PASQUIER, 2010a; KULKARNI et al., 2015) no entanto muitas das técnicas existentes não apresentam uma boa separação entre a curva de dificuldade e a geração de conteúdo jogável, de forma que estes elementos se tornam muito acoplados. Visando atuar sobre isto, utilizamos o conceito de Curva de Progresso (STEINHOFF, 2014) que, apesar de pouco explorado na academia, é amplamente utilizado pela indústria – em particular em jogos do tipo Puzzle, como é o caso do Jelly Splash (WOOGA, 2013) ou Candy Crush Saga (KING, 2012).

Este conceito se baseia em balancear a distribuição de níveis de um jogo de forma a manter o jogador interessado e motivá-lo a continuar tentando até conseguir. Para que isso funcione, é necessária a utilização de três tipos de níveis:

- *Blocking* (Bloqueantes) – São níveis que visam bloquear o progresso do jogador até a parte seguinte do jogo. Na Figura 3, estes níveis correspondem aos pontos de pico na parte superior do gráfico.
- *Buildup* (Preparação) – São níveis que visam garantir algum progresso ao jogador enquanto o preparam para o que está por vir. Na Figura 3, estes níveis correspondem às linhas retas horizontais nos pontos intermediários do gráfico, entre os picos e vales.
- *Relief* (Alívio) – São níveis que são apresentados ao jogador logo após este passar por um momento muito difícil. O objetivo destes é oferecer alguns momentos de alívio e progresso fácil para que o jogador se sinta recompensado pelo seu feito ao passar por níveis difíceis. Na Figura 3, estes níveis correspondem às linhas retas horizontais localizadas no vale do gráfico.

Figura 3 – Exemplo de curva de progresso



Fonte: Florian Steinhoff, 2014

A Curva de Progresso está também intimamente ligada à monetização em jogos Free-to-Play (Gratuitos para Jogar), por apresentarem diversos momentos nos quais os jogadores desejarão pagar para pular momentos de dificuldade muito elevada, mas também serve como fator motivador ao apresentar aos jogadores barreiras de progresso que exigem uma grande quantidade de esforço para serem ultrapassadas.

Como em jogos de infinitos ciclos nem sempre há o conceito de níveis, a ideia apresentada pela Curva de Progresso foi adaptada para o modelo de segmentos que será devidamente explicado no capítulo 5.

2.6 Playability

Este é um fator essencial e que deve ser sempre observado em qualquer ferramenta de geração de níveis. A *playability* diz respeito à jogabilidade do conteúdo gerado (COMPTON; MATEAS, 2006), ou seja, se ao inserir no jogo o conteúdo gerado, o jogador pode interagir com este da forma desejada e se as ações do jogo são factíveis.

Se tomarmos o exemplo de um jogo de plataforma no qual o jogador se move para a esquerda ou direita e pode pular, é necessário observar se os níveis gerados

podem ser ultrapassados pelo jogador de acordo com o modelo físico do jogo, afinal é possível que seja gerado um nível com um obstáculo tão alto que o jogador não seja capaz de pular sobre este. Para evitar que tal tipo de problema ocorra, é necessário que o gerador de níveis leve em consideração limitações como a altura e distância que o jogador pode mover seu personagem no jogo.

A geração de níveis do jogo utilizado neste trabalho foi feita inicialmente de forma manual por um *game designer* que se encarregou de verificar se todos os segmentos gerados são factíveis, mas posteriormente foi substituído por um sistema que gera automaticamente os segmentos seguindo uma série de restrições que garantem que tais segmentos são ultrapassáveis.

2.7 Método de Geração do Conteúdo

A forma como o conteúdo deve ser gerado também é algo importante por envolver diversas restrições. Algumas de tais restrições abrangem se o conteúdo é gerado de forma *online* ou *offline* (TOGELIUS et al., 2011b; HENDRIKX et al., 2013). A escolha entre as duas formas está comumente associada não só ao resultado que se deseja alcançar, como a restrições da plataforma na qual o jogo será executado. Em casos onde a disponibilidade de memória de armazenamento é limitada, pode não ser possível armazenar os dados gerados a partir de uma técnica *offline*. Em contrapartida, se a plataforma dispõe de capacidade de processamento limitada – ou se parte considerável de tal capacidade de processamento já está sendo utilizada com outros aspectos do jogo – uma técnica *online* pode não ser indicada.

Além disso, o grau de interação humana que se deseja para a geração do conteúdo é um ponto relevante. É possível inserir variáveis níveis de interação humana no processo, criando-se um espectro que varia entre a total intervenção humana, na qual não ocorre nenhuma geração automática, ou nenhuma interação, no qual um sistema se encarrega de gerar o conteúdo de forma totalmente automática.

2.8 Controle de Alto Nível

É necessário que o *game designer* retenha pelo menos o controle de alto nível do processo de geração automática (HENDRIKX et al., 2013; ADRIAN; LUISA, 2013), uma vez que a experiência e conhecimento deste são necessários para indicar as diretrizes do conteúdo que se deseja gerar.

O sistema deve ser construído levando-se em conta os *guidelines* (orientações) e especificações do *game designer* para que gere o conteúdo desejado e elimine instâncias indesejáveis. Além disso, é necessário permitir o controle paramétrico do

processo de geração, de forma que seja possível modelar experiências específicas de jogo.

2.9 Geração para Jogos de Infinitos Ciclos

Apesar de amplamente utilizada na indústria, a geração para jogos de infinitos ciclos é uma questão pouco explorada na academia (ADRIAN; LUISA, 2013; KULKARNI et al., 2015). Como as partidas para jogos de infinitos ciclos podem ser potencialmente infinitas, há algumas questões relacionadas especificamente a este tipo de jogo que devem ser observadas.

É preciso definir um sistema que vá além de gerar níveis individuais, uma vez que é necessário gerar partidas contínuas e ininterruptas de forma *online*. Partes do processo de geração podem ser *offline*, mas é essencial que durante o jogo o conteúdo seja inserido de forma *online*, de forma a evitar que a experiência do usuário seja deteriorada com telas de carregamento ou momentos de travamento.

Além disso, a experiência de um usuário no caso de um jogo de ciclo infinito é diferente porque os desafios são apresentados de forma contínua. Em jogos de níveis, há intervalos claramente definidos nos quais são apresentados ao jogador momentos de relaxamento, como é o caso do fim de um nível após derrotar um *boss* (chefe de nível). Já em um jogo de infinitos ciclos é necessário introduzir tais momentos sem causar uma quebra visível na experiência do jogador durante a partida.

2.10 Resumo

Esse capítulo apresentou diversas questões que estão envolvidas na geração de conteúdo para jogos. Tais questões se mostraram essenciais por afetarem diretamente as abordagens utilizadas.

Com o entendimento das necessidades específicas existentes em jogos de infinitos ciclos, que diferem das necessidades de jogos tradicionais, e com a observação das questões aqui apresentadas, foram identificadas as propriedades necessárias para a solução desejada, são estas:

- 1) Controle pelo *Game Designer* – como foi especificado por Diaz Furlong et al. (ADRIAN; LUISA, 2013) e Hendrikx et al. (2013), é importante prover ao *game designer*, ou *level designer*, ferramentas para afetar o processo de geração de níveis para que não se perca controle do processo de design e também porquê a experiência do profissional – tanto como um designer quanto como um jogador – dá a este uma intuição do que torna um nível divertido ou não; dada a subjetividade do que é diversão e como medi-la, ter a atuação de um

designer durante o processo de geração de conteúdo é algo que enriquece demasiadamente o conteúdo.

- 2) Geração automática do conteúdo – o conteúdo deve ser gerado com o maior nível possível de automação de forma a minimizar os custos com produção, equipe e tempo para a geração de níveis para o jogo. Além disso, a automatização permite uma maior variabilidade dos elementos de jogo.
- 3) Levar em consideração a possibilidade de partidas infinitas – em situações normais e para efeitos práticos, uma partida de jogo comumente não dura para sempre, eventualmente o jogador alcança uma condição de derrota ou comete um erro que o impede de continuar. No entanto, não é possível saber de antemão qual é a quantidade de conteúdo que um jogador é capaz de consumir até que a partida chegue ao fim, dessa forma é necessário que o sistema gere partidas sob demanda e forneça ao jogador tanto conteúdo quanto seja necessário para a continuação da sessão de jogo. Este sistema também não pode ter custos elevados de processamento ou memória, do contrário sua utilização em tempo real nos jogos se torna inaplicável.
- 4) Baseado em dificuldade – a dificuldade é algo que é mais tangível do que diversão por ser um elemento de medição mais fácil: se em situações iguais e com amostras de mesmo tamanho o inimigo X mata mais jogadores do que um inimigo Y, é porque X é um inimigo mais difícil do que Y. Além disso, é possível realizar uma associação mais direta entre os conceitos de flow explicados no capítulo 2 e a dificuldade do jogo. Dadas essas questões, é necessário que o elemento dificuldade faça parte da solução desenvolvida.

Tendo em mente tais propriedades no desenvolvimento da solução, foi criado um processo em quatro etapas que explora amplamente o conceito de “segmento” e de Curva de Progresso. Tal processo será explorado em detalhes no Capítulo 4.

3 ESTADO DA ARTE

Este capítulo apresenta um estudo do estado da arte de técnicas de geração procedimental de conteúdo para jogos digitais.

3.1 Introdução

Este capítulo visa apresentar o estado da arte no que diz respeito à geração procedimental de conteúdo. Diversas técnicas e soluções serão apresentadas e analisadas no contexto do gênero de jogo que este trabalho aborda.

3.2 Técnicas Analisadas

Compton e Mateas (2006) apresentam uma alternativa na qual é usado o conceito de ritmo na geração de níveis. Neste trabalho, componentes básicos como plataformas ou espinhos, são combinados de acordo com um tipo de padrão de ritmo, e o resultado da combinação é uma entidade denominada de “padrão”. Através da criação de diversos conjuntos de “padrões”, é possível realizar combinações que formam níveis tanto lineares quanto não-lineares. Para determinar como se dão tais combinações, o sistema usa uma estrutura de “grupo de células”, que determinam de que forma os padrões devem ser justapostos e como se dá a transição entre tais padrões.

Este modelo, apesar de bastante robusto, apresenta alguns problemas para a solução que buscamos: ele exige a definição de uma estrutura de células de antemão, algo que o torna ideal para a geração de níveis finitos, mas não resolve bem a questão da geração de partidas infinitas. Além de tal problema, o sistema usa o modelo físico do jogo como uma forma de prever a factibilidade dos padrões e menciona que podem ser usadas informações relativas às habilidades do jogador para realizar o cálculo aproximado da dificuldade, mas não indica como isso pode ser feito na plataforma em questão.

Smith et al. (2009) apresentam um trabalho similar ao descrito anteriormente, mas com mudanças substanciais na forma como o nível é gerado: ao invés de gerar blocos diretamente a partir de padrões de ritmos, é gerada uma estrutura intermediária a partir dos padrões de ritmos e a partir de tal estrutura intermediária são gerados os blocos. De acordo com o trabalho, um dos benefícios dessa abordagem é que passa a ser possível gerar uma grande quantidade de blocos para o mesmo padrão de ritmos.

Tal trabalho apresenta um problema similar ao analisado anteriormente: ele foca na criação de níveis completos com base em séries de restrições. Apesar de ser

possível gerar uma quantidade potencialmente infinita de níveis, não existe a noção de um único nível que pode ter uma extensão infinita. Além disso, o foco do trabalho está na geração de níveis com uma maior diversidade, preocupando-se apenas com a geração rítmica e sem levar em consideração a dificuldade dos níveis, o que pode afetar negativamente a experiência dos jogadores.

A abordagem para geração de conteúdo usada por Sorenson e Pasquier (2010b) consiste em um framework genérico que visa automatizar o processo de criação de níveis. Este trabalho utiliza algoritmos genéticos e blocos para construção de nível denominados de *design elements*, que correspondem às unidades básicas que são encaixadas de forma subsequente. A função de fitness do algoritmo genético é capaz de identificar o quão divertido é o nível gerado, e para isso leva em consideração a dificuldade percebida pelo jogador ao longo do tempo. Adicionalmente, é necessário especificar restrições que fornecem ao designer a capacidade de garantir certos elementos nos níveis gerados, assim como garantem que tais níveis sejam sempre factíveis pelo jogador.

Este framework apresenta um bom potencial para a geração de níveis variados e que respeitam certos parâmetros de dificuldade, mas não há garantias de que seja possível gerar níveis de forma *online* utilizando tal ferramenta. Nos experimentos feitos pelos autores, foram gerados níveis de qualidade e que respeitavam as limitações estabelecidas, mas para alcançar alguns desses níveis foi necessário algo próximo de trinta minutos de execução do algoritmo. Assim sendo, não é viável realizar a geração de níveis infinitos utilizando esta ferramenta. No entanto, esta abordagem apresentou um conceito que foi adaptado para o contexto de jogos de infinitos ciclos: os blocos. A utilização de blocos jogáveis na construção de níveis foi adaptada para o conceito de segmento em nosso trabalho, que será definido no Capítulo 4.

A técnica para geração de conteúdo apresentada por Ashlock (2010) explora a utilização de um algoritmo evolucionário para criar puzzles com níveis de dificuldade previamente especificados. Esta técnica apresenta ainda grande potencial para expansão a um custo de processamento não muito elevado, bastando que sejam feitas algumas mudanças no algoritmo de programação dinâmica usado na função de fitness

Apesar de o conteúdo apresentado no trabalho supracitado ter uma boa aproximação com o objetivo desse trabalho, o seu foco em jogo de puzzles dificulta a sua adaptação para outros tipos de jogos. A técnica apresentada é de fato bastante robusta, mas sua aplicação em gêneros diferentes de puzzle ainda não é clara.

Outro trabalho apresentado por Smith, Whitehead e Mateas (2011) introduz a ferramenta de level design Tanagra, que permite que humanos e computadores trabalhem juntamente na produção de níveis para jogos 2D. Este trabalho apresenta conceitos de geração rítmica de níveis em combinação com uma engine de satisfação

de restrições (*Constraint Satisfaction Problem* – CSP), de forma a gerar níveis divertidos e que respeitem restrições estabelecidas pelo *game designer*.

Este trabalho apresenta uma ferramenta bastante robusta e com bons resultados, no entanto ela ainda apresenta uma baixa diversidade de elementos de jogo e também se propõe a tratar apenas da questão de jogos de plataforma 2D baseados em níveis.

O trabalho desenvolvido por Pedersen, Togelius e Yannakakis (2009) consiste em uma forma de modelar a experiência do jogador no jogo Super Mario Bros. Nesse trabalho, os níveis são criados utilizando certas heurísticas que posicionam aleatoriamente elementos nos níveis, com alguns elementos que afetam a experiência do jogador sendo posicionados de forma fixa. Para realizar a modelagem da experiência, diversos jogadores são convidados a jogar conjuntos de partidas, durante as quais são coletados dados diversos sobre como se saem no jogo. Ao fim de tais partidas, pede-se que os jogadores respondam a um questionário sobre sua experiência no jogo e por fim tais dados são utilizados em uma rede neural responsável por fazer a modelagem da experiência.

Este trabalho apresenta uma abordagem bastante robusta para a modelagem da experiência do jogador, mas não abrange bem a questão de como os níveis são gerados. A heurística utilizada não é detalhada e a ferramenta foca na geração de partidas finitas, mas o conceito de modelagem é algo bastante útil e que pode ser diretamente aplicado na geração de partidas infinitas.

Kerssemakers et al. (2012) apresentam um trabalho que visa criar um gerador de gerador de levels procedurais, ou seja, um meta-PCG, de forma a minimizar o trabalho de ter que fazer um novo gerador de levels para cada novo jogo. O gerador trabalha com dois ciclos: um interno e um externo. O ciclo externo pode ser visto como o “gerador de geradores” e o ciclo interno é composto por vários “geradores de level”. O ciclo interno é auto-gerenciado e não necessita da interação humana, enquanto o ciclo externo depende de uma entrada do designer para indicar que gerador ele considerou melhor. No processo de escolha do melhor gerador, o designer tem à sua disposição diversas informações, que incluem visualização em nuvem, visualização dos níveis gerados por cada gerador, a possibilidade de jogar os níveis gerados e um score de *playability* calculado pelo sistema. Os geradores internos por sua vez são compostos por múltiplos agentes que posicionam elementos dentro do nível respeitando determinados parâmetros.

Este trabalho apresenta um grande potencial no que diz respeito à geração de conteúdo com controle pelo *game designer*. Os *game designers* podem selecionar os níveis que mais se aproximam da experiência desejada e o algoritmo trata de convergir para o resultado esperado. Um dos problemas, no entanto, é que o designer não tem controle inicial sobre o que é gerado, ele apenas pode direcionar os níveis já

criados pelo gerador na direção desejada. Além disso, a intervenção do *game designer* se mostrou fadigante, por ser necessário analisar uma grande quantidade de níveis para escolher os mais aptos – um problema que foi parcialmente resolvido através da criação de um banco de geradores de boa qualidade. Além dessa questão, o gerador de geradores preocupa-se apenas com a criação de níveis individuais, e não com a modelagem de uma experiência completa, levando em conta a dificuldade dos níveis por exemplo, e potencialmente infinita.

O trabalho de Mourato, Santos e Birra (2011) propõe uma solução para geração de níveis em jogos de plataforma 2D que apresenta uma maior expressividade e menor linearidade do que as técnicas baseadas em ritmo já existentes, exigindo o mínimo de interação humana, e se utilizando de técnicas *chunk based*. As entradas iniciais disponíveis ao *game designer* são bastante limitadas, e dizem respeito basicamente à distribuição espacial do nível e parâmetros do algoritmo genético. Já o gerador se divide em algumas etapas: a primeira etapa consiste em gerar os níveis através de um algoritmo genético, após isso o nível pode passar por uma etapa de pós-processamento na qual são adicionados inimigos ou armadilhas como uma forma de enriquecê-lo, e por fim o nível pode ser visualizado dentro da ferramenta.

Apesar de dispor de resultados bastante interessantes, este trabalho apresenta alguns problemas que dificultam a utilização da ferramenta. Primeiramente o *game designer* que lida com a ferramenta precisa ter o conhecimento de parâmetros associados a algoritmos genéticos para inserir a entrada inicial no sistema. Possivelmente a utilização de outros tipos de parâmetro de entrada tornaria a ferramenta de uso mais fácil no geral. Além disso, o processo de geração de níveis pode demorar um tempo considerável, sendo necessária a exibição de uma tela de carregamento dentro do jogo, o que impossibilita a utilização deste sistema em jogos com partidas potencialmente infinitas. Por fim, o trabalho não trata da questão de experiência ou dificuldade dos níveis gerados, ficando isso a cargo do *game designer*.

Outro trabalho apresentado por Sorenson e Pasquier (2010a) consiste de um sistema de geração de levels para jogos de plataforma 2D utilizando algoritmos genéticos. Este trabalho utiliza desafio como a entrada primária de sistema, por considerar esta como um dos principais elementos que causam prazer ao jogar um jogo. Como forma de medir o desafio, o trabalho faz uma conversão da ansiedade acumulada do jogador para o desafio baseando-se no conceito de *flow*.

Este é um trabalho bastante diferenciado por utilizar uma abordagem top-down no processo de geração e considerar a variável desafio como entrada no processo de geração, o que facilita a extensão da técnica para outros gêneros de jogo. No entanto o *game designer* possui controle muito limitado sobre o tipo de conteúdo gerado, o que dificulta a geração de um tipo específico de nível desejado. Além disso, o sistema

ainda cria níveis relativamente simples e comparativamente inferiores ao que pode ser feito por um *game designer*.

A abordagem utilizada por Adrian e Luisa (2013) apresenta conceitos bastante relevantes. Esta abordagem consiste e, utilizar uma curva de dificuldade criada pelo *game designer* e outra curva de dificuldade calculada para o conteúdo candidato, de forma que o melhor conteúdo é aquele cuja curva de dificuldade melhor se encaixa com a curva desejada.

Tal trabalho apresenta uma boa solução que pode ser adaptada para diversos jogos, no entanto o conteúdo que é inserido durante as partidas é utilizado de forma muito granular, o que pode tornar tal adaptação um pouco mais complexa. Além disso, para modelar a dificuldade o sistema utiliza de fórmulas que precisam ser definidas para o caso de cada jogo particular com base em suposições do level designer acerca do que afeta a dificuldade do jogo. Por fim, o trabalho apresenta um grande acoplamento entre curva de dificuldade, geração do conteúdo e disposição deste durante a partida, o que pode tornar ainda mais complexa a adaptação para outros tipos de jogos.

Um trabalho apresentado por Togelius et al. (2011a) envolve a geração de partidas para o jogo Super Mario Bros (NINTENDO, 1985) através da utilização de duas abordagens são utilizadas, uma *online* e uma *off-line*. Na abordagem offline os níveis são gerados com base nos comandos do jogador durante uma partida de teste. A partir daí, o próximo nível é gerado com base na entrada do usuário no nível atual. Já a abordagem *online* é bastante similar, apresentando apenas a diferença de que o conteúdo é adicionado no nível em tempo real com base nos comandos do jogador e, quanto mais moedas o jogador obtém, mais inimigos são adicionados adiante no jogo.

Este trabalho é promissor por utilizar uma abordagem que pode ser aplicada em jogos de infinitos ciclos – a geração *online*. No entanto, não é oferecido controle algum ao game designer sobre o conteúdo gerado e também não há preocupação com *flow*, dificuldade ou experiência do usuário, o conteúdo apenas é gerado de forma aleatória com base nos comandos do jogador.

3.3 Resumo

Este capítulo apresentou o estado da arte para geração procedimental de conteúdo em jogos. Muitas das técnicas apresentadas se mostram bastante eficientes em jogos que seguem a estrutura de níveis finitos, então diversas adaptações foram necessárias para utilizar conceitos apresentados nos trabalhos estudados ao que desejamos realizar.

É importante ainda ressaltar que muitos dos trabalhos existentes atuam apenas sobre alguns dos aspectos da geração de partidas, seja a criação de níveis ou a

determinação da curva de dificuldade, e mesmo os trabalhos que atuam em mais de um dos aspectos não definem de forma clara e distinta as etapas do processo de geração de conteúdo para os jogos. Além disso, são necessárias diversas adaptações aos trabalhos existentes para que as técnicas possam ser utilizadas em jogos de infinitos ciclos.

Este trabalho apresenta como diferencial a criação de um processo claramente delimitado que pode ser utilizado em jogos de infinitos ciclos, mantendo o controle do *game designer* sobre o conteúdo apresentado, mas com necessidade de um baixo nível de intervenção por parte do designer – o que aumenta a sua produtividade. Como um diferencial adicional, introduzimos ainda o conceito de Curva de Progresso, que visa tornar a experiência em jogos desse tipo ainda melhor.

4 SOLUÇÃO PROPOSTA

Neste capítulo é apresentada a solução proposta para a geração de partidas em jogos de infinitos ciclos, respeitando-se as propriedades necessárias para uma solução satisfazível.

4.1 Introdução

Este capítulo visa explicitar a solução proposta para a geração de partidas de jogos de infinitos ciclos, partindo dos conceitos fundamentais para a solução, a definição do processo proposto e por fim como tal solução pode ser implementada.

4.2 Definição de Elementos de Jogo e Segmento

Os elementos de jogo são as entidades atômicas básicas existentes em jogos. Em um jogo como Super Mario Bros (NINTENDO, 1985) por exemplo, um bloco de interrogação, um inimigo ou um cano seriam entidades atômicas, como pode ser visto na figura abaixo.

Figura 4 – Imagem do jogo Super Mario Bros na qual é possível identificar blocos de interrogação, um inimigo Goomba e um Cano



Fonte: Nintendo, 1985

Um segmento por sua vez corresponde ao agrupamento de diversos elementos de jogo em um mesmo espaço de tamanho fixo, correspondendo dessa forma à menor unidade lógica do jogo à qual o *game designer* tem acesso na ferramenta de geração de partidas. Ele corresponde ao *building block* (bloco de construção) da solução proposta e é similar ao conceito de *Design Element* (Elemento de Design) apresentado por Sorenson e Pasquier (2010b), mas como diferencial explícito de que seu tamanho é pré-definido para um tamanho que contenha uma área jogável significativa do jogo.

Tal restrição não foi feita de forma arbitrária: definimos que segmentos devem ter um tamanho fixo porque a interação entre os elementos de jogo em um determinado espaço físico é um dos principais elementos responsáveis pela definição da dificuldade de um segmento. Voltando ao exemplo do jogo Super Mario Bros: um segmento com 300 pixels de largura e apenas um inimigo poderia ser considerado fácil, já um segmento com os mesmos 300 pixels de largura e 10 inimigos seria considerado difícil. No entanto, se o *game designer* definir o tamanho do segmento como 30 pixels, um valor muito pequeno para este jogo, o gerador de níveis pode erroneamente agrupar 10 segmentos consecutivos, cada um com um inimigo, e considerar cada segmento individual como fácil, quando coletivamente eles criam uma experiência difícil.

Dessa forma, cabe ao designer definir para o seu jogo um tamanho de segmento que seja grande o suficiente para conter uma boa variedade de elementos, mas que não seja grande demais de forma a tornar difícil a medição da dificuldade. Como um princípio básico e para o caso de jogos 2D, constatamos que segmentos do tamanho de uma tela de jogo apresentam bons resultados.

A forma como os segmentos são utilizados nos jogos variam de gênero para gênero, em um jogo de corrida infinita, como Jetpack Joyride (HALFBRICK STUDIOS, 2011), os segmentos são dispostos um após o outro ao longo de uma longa linha horizontal. Já em um RPG de Ação, como Diablo 3 (BLIZZARD ENTERTAINMENT, 2012), os segmentos são dispostos em uma extensa área e são conectados através de corredores estreitos. Exemplos de segmentos para ambos os jogos podem ser vistos nas figuras abaixo.

Figura 5 – Um segmento do jogo Jetpack Joyride, no qual diversas moedas estão dispostas de forma a formar a sentença “COINS!!”



Fonte: Halfbrick Studios, 2011

Figura 6 – Mapa de um calabouço no jogo Diablo 3. Em tal mapa é possível identificar grandes segmentos que são conectados por corredores menores



Fonte: Blizzard Entertainment, 2012

4.3 O Processo

O processo em 4 etapas é uma das principais contribuições deste trabalho. O objetivo do estabelecimento de tal processo é estruturar e compartimentalizar o desenvolvimento das soluções de GPC para jogos de infinitos ciclos. Com isso, não só se torna mais fácil utilizar o mesmo processo para jogos de diversos gêneros diferentes, como também é possível modificar apenas aspectos específicos da geração procedimental sem que os outros aspectos sejam afetados.

Este processo tem sua base no conceito de segmento definido anteriormente.

O segmento foi utilizado como bloco de construção das partidas no jogo Boney the Runner e a versão inicial do processo foi definido com base na ampla experiência do time da BigHut Games na indústria, herdada da Manifesto Games (2005), empresa que atua há mais de uma década no mesmo setor e que deu origem à BigHut Games.

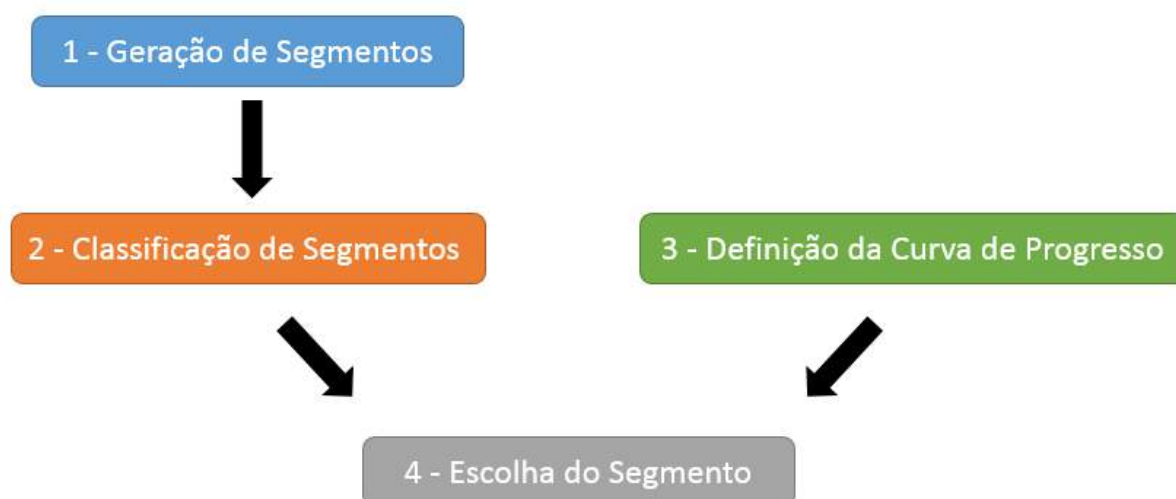
Constatou-se no mercado o sucesso da abordagem utilizada: até a data da apresentação deste trabalho, o jogo em questão conta com mais de um milhão de downloads, cinco mil usuários diários e uma base de usuários fiel e dedicada ao jogo.

Considerando-se o tamanho sucesso do jogo, o autor do trabalho em questão – que é também sócio fundador da BigHut Games – formalizou o processo para geração de partidas em jogos do tipo *endless runner* em seu trabalho de graduação no curso de Ciência da Computação da Universidade Federal de Pernambuco.

Posteriormente o autor tornou o processo genérico, de forma a não abranger apenas jogos do tipo *endless runner*, mas sim qualquer tipo de jogo de ciclos infinitos. O processo que será definido a seguir é o resultado disso, e seus resultados parciais já foram reconhecidos em publicação acadêmica (CARVALHO et al., 2013).

O processo definido pode ser visualizado de forma estruturada na figura abaixo:

Figura 7 – Processo para a geração automática de conteúdo para jogos de infinitos ciclos



Fonte: Leonardo Vieira de Carvalho et al., 2013

Cada uma das etapas será descrita a seguir:

- 1) Geração de Segmentos – Nesta etapa são gerados diversos segmentos jogáveis que serão inseridos no jogo. Este é um processo *offline* que deve ser executado durante o desenvolvimento do jogo. Nesta etapa uma grande variedade de segmentos jogáveis é gerada. Para tal geração, pode-se utilizar uma das diversas técnicas já existentes tanto na indústria quanto na academia, ou o

desenvolvedor do jogo pode desenvolver sua própria técnica se nenhuma das técnicas existentes forem adaptáveis para o jogo desejado.

- 2) Classificação de Segmentos – Os segmentos gerados na etapa anterior precisam ser devidamente classificados de acordo com seus respectivos níveis de dificuldade. Este é um processo *offline* que deve ser executado durante o desenvolvimento do jogo, após a geração dos segmentos. Tal classificação é necessária porque a dificuldade é o elemento que determina quando um segmento deve ser inserido durante a partida. Sem a classificação dos segmentos, estes serão posicionados de forma aleatória durante a partida, o que gera uma experiência negativa para o jogador.
- 3) Definição da Curva de Progresso – Esta etapa pode ser executada paralelamente às duas etapas anteriores e nela é definida a forma como os segmentos são dispostos no jogo ao longo da partida. A curva de progresso na verdade é um produto da curva de dificuldade gerada pelo *game designer*. A geração da curva de dificuldade é um processo *offline* que deve ser executado durante o desenvolvimento do jogo, mas a curva de progresso é gerada de forma automática durante a partida. Esta é uma curva no qual no eixo vertical há a dificuldade e no eixo horizontal há uma medida de progresso, que pode ser tempo de jogo, quantidade de níveis que o jogador obteve, quantidade de moedas, etc. Durante a partida, quando é chegado o momento de selecionar o próximo segmento a ser posicionado, o tipo de segmento a ser escolhido é consultado na curva de progresso e este então é escolhido baseado no procedimento definido na etapa anterior.
- 4) Escolha do Segmento – Com base na definição feita na etapa anterior do processo, deve ser feita a escolha do segmento a ser posicionado dentre as opções existentes. Este é um processo parcialmente *offline* e parcialmente *online*, por possuir etapas que são executadas durante o desenvolvimento e outras etapas que são executadas durante a partida. Os segmentos são escolhidos a partir do conjunto de segmentos gerados na primeira etapa, respeitando a classificação definida na segunda etapa. No entanto, a forma como os segmentos são posicionados durante a partida pode variar de acordo com o tipo de jogo. Para alguns jogos pode ser suficiente posicionar os segmentos de forma linear, já para outros pode ser necessário que estes sejam posicionados de forma adjacente em diversas direções.

Uma vez que as quatro etapas do processo foram delineadas, a seguir serão detalhadas as soluções empregadas em cada uma das etapas.

4.4 Geração de Segmentos

A geração de níveis para jogos é um problema complexo e que depende do tipo de jogo com o qual se está trabalhando. Como foi visto no capítulo 3, existem diversas técnicas aplicáveis para jogos de gêneros específicos, como jogos de plataforma 2D, e algumas outras técnicas que visam gerar conteúdo de forma genérica para jogos. No entanto, ao analisar tais técnicas genéricas de forma mais aprofundada, é possível observar que em muitos casos há limites na expressividade e extensibilidade para a riqueza de gêneros de jogos que está disponível no mercado.

Tendo tal questão em mente, o autor deste trabalho acredita que para realizar a geração de segmentos é necessário analisar de antemão o tipo de jogo e qual procedimento mais se adequa ao resultado desejado, algo que foge do escopo deste trabalho dada a grande variedade de jogos existentes no mercado. No entanto, no capítulo 5 é apresentado um estudo de caso e uma das técnicas já existentes para geração procedimental de níveis será adaptada ao jogo analisado.

Apesar dessa dificuldade, é importante reiterar que as técnicas já existentes para geração procedimental de níveis podem ser facilmente adaptadas para o conceito de segmento, bastando delimitar uma área de jogo que seja grande o suficiente para que uma variedade relevante de elementos possa ser inserida, mas que não seja demasiadamente grande, de forma a dificultar a classificação de cada segmento de acordo com a dificuldade. Aplicando-se tais restrições às técnicas já existentes, ou utilizando-as como base para uma nova técnica de geração de segmentos, é possível alcançar resultados satisfatórios.

4.5 Classificação de Segmentos

Para realizar a classificação dos segmentos, foi utilizada uma técnica similar à apresentada por Pedersen, Togelius e Yannakakis (2009): são coletados dados de partidas de diversos jogadores e tais dados são usados para modelar a dificuldade dos segmentos. No entanto, no trabalho citado esta modelagem está associada às emoções do jogador, para este trabalho foi escolhido modelar a dificuldade dos segmentos, para que o processo de classificação desses possa ser automatizado. Assim sendo, o processo para classificação dos segmentos é dividido em duas etapas.

Na primeira etapa diversos jogadores são convidados a jogar partidas do jogo, e a cada partida são coletadas métricas referentes ao desempenho do jogador em diversos segmentos. É importante deixar claro que, para que o processo de classificação dos segmentos possa ser automatizado, é necessária a coleta de dois tipos de métricas:

- 1) Elementos controláveis: dizem respeito a características do segmento que podem ser diretamente influenciadas pelo designer ou pelo gerador de segmentos, como a quantidade de inimigos existentes em um segmento por exemplo.
- 2) Elementos de gameplay: dizem respeito à forma como o jogador interage com o jogo e abrange métricas como a quantidade de tempo que o jogador passou no segmento ou quantidade de vida perdida por exemplo.

Na segunda etapa, com base no desempenho dos jogadores, faz-se uma reclassificação dos segmentos utilizando uma métrica que sirva como indicador principal de dificuldade. Esta métrica precisa ser definida pelo *game designer* e pode ser algo como a quantidade de dano sofrida pelo jogador durante o segmento por exemplo.

Apenas esta reclassificação, no entanto, não é o suficiente, é necessário que o sistema seja capaz de classificar os segmentos com base nos dados coletados anteriormente, principalmente nos dados controláveis, que podem ser modificados durante a construção dos segmentos. Por isso, são criadas duas redes neurais com propósitos diferentes: uma das redes recebe como entrada apenas os elementos controláveis, a Rede A, e outra recebe como entrada tanto os elementos controláveis quanto os elementos de gameplay, a Rede B. Ambas as redes fornecem como saída a dificuldade do segmento.

O propósito para a criação de duas redes está relacionado às necessidades existentes em diferentes fases do ciclo de vida do jogo. Durante o processo de criação de novos segmentos, não se tem acesso a dados de partidas, afinal nenhum jogador teve a oportunidade de jogar tais segmentos, e se for necessário que cada novo seja testado com um grupo de jogadores, estabelece-se um grande overhead no processo de criação de conteúdo.

Assim sendo, a Rede A se encaixa na fase inicial de desenvolvimento, na qual se tem acesso apenas a informações controláveis dos segmentos.

A Rede B por sua vez é útil para fazer ajustes periódicos no jogo após a sua disponibilização. Com o lançamento do jogo, será possível coletar dados referentes à forma como os jogadores estão interagindo e utilizar tais dados para reclassificar segmentos já existentes. Dessa forma, como mais dados passam a se tornar disponíveis com o tempo, existe uma tendência de que os segmentos fiquem cada vez mais próximos da verdadeira dificuldade que proporcionam aos usuários.

4.6 Definição da Curva de Progresso

A curva de progresso utilizada para determinar quais segmentos devem ser posicionados é baseada no conceito apresentado no capítulo 2. O preceito básico

dessa curva é que para manter o jogador interessado e motivá-lo a continuar tentando até conseguir passar de um determinado desafio, deve-se balancear a distribuição de níveis de um jogo de forma que entre momentos de pico de dificuldade, existam momentos de alívio ou de preparação para outro pico de dificuldade.

Tendo isso em mente, Steinhoff (2014) definiu três tipos básicos de níveis de jogo:

- **Blocking (Bloqueantes)** – São níveis que visam bloquear o progresso do jogador até a parte seguinte do jogo. Para o propósito deste processo, níveis bloqueantes são considerados aqueles de alta dificuldade.
- **Buildup (Preparação)** – São níveis que visam garantir algum progresso ao jogador enquanto o preparam para o que está por vir. Para o propósito deste processo, níveis bloqueantes são considerados aqueles de dificuldade intermediária e não são restritos apenas a aparecerem antes de um bloqueante.
- **Relief (Alívio)** – São níveis que são apresentados ao jogador logo após este passar por um momento muito difícil. Para o propósito deste trabalho, níveis de alívio são considerados aqueles de baixa dificuldade, e não necessariamente têm a função apenas de fornecer um momento de descanso após um nível muito difícil. Ao adotar essa abordagem, níveis de alívio podem ser usados nos primeiros momentos do jogo como uma forma de preparação para o jogador.

Exploramos este mesmo conceito, mas de uma forma diferenciada: o *game designer* define uma curva de dificuldade e, com base nesta curva, o sistema gera uma curva de progresso correspondente respeitando de forma automática os princípios apresentados anteriormente. Além disso, adaptamos o conceito de níveis bloqueantes, de preparação e de alívio para segmentos dos mesmos tipos: segmento bloqueante (SB), segmento de preparação (SP) e segmento de alívio (SA).

Para realizar a conversão entre tais curvas, são utilizados dois conceitos:

- 1) **Sequenciamento de Segmentos:** Os segmentos são apresentados no formato de sequências com uma quantidade pré-definida de segmentos, os quais seguem determinadas restrições, como por exemplo de que antes de todo SB deve haver um SP ou SA. Todas as restrições serão especificadas em mais detalhes
- 2) **Proporção dos Segmentos:** Conforme o jogador progride no jogo – sendo a forma de progresso determinada pelo *game designer* através de uma métrica como tempo de jogo ou experiência do personagem por exemplo – menos SA devem aparecer e mais SB devem existir, de forma a tornar a partida mais desafiadora com o tempo.

A forma como estes conceitos foram empregados é explicada em mais detalhes a seguir.

4.6.1 Sequenciamento de Segmentos

Para o sequenciamento de segmentos, foram definidas as seguintes restrições, baseadas nos conceitos apresentados por Steinhoff (2014):

- Toda sequência deve ter tamanho fixo
- Nenhuma sequência pode ter início com um SB
- Toda sequência deve apresentar pelo menos um SB
- Segmentos do tipo SP só pode aparecer imediatamente antes ou depois de um SB

Com tais restrições, pode-se criar uma grande variedade de sequências, todas proporcionando ao jogador momentos de alívio, preparação e dificuldade. A caráter ilustrativo, se for definido que sequências devem ter quatro segmentos, uma sequência válida seria SA-SP-SB-AS; nesta sequência o jogador primeiro joga um segmento de alívio, em seguida é apresentado um segmento de preparação, então um segmento bloqueante é apresentado, e uma vez que o segmento bloqueante é ultrapassado, mais um segmento de alívio é apresentado.

4.6.2 Proporção dos Segmentos

Já para a proporção de segmentos, há uma variação linear na quantidade percentual de cada tipo de segmento ao longo da partida, e o designer tem o controle sobre como esta variação se comporta. Para exercer tal controle são usadas seis variáveis:

- Porcentagem de Segmentos Bloqueantes mínima (PSBmi) – A porcentagem de segmentos bloqueantes que podem aparecer em uma sequência criada no ponto de dificuldade mínima da curva de dificuldade.
- Porcentagem de Segmentos de Preparação mínima (PSPmi) – A porcentagem de segmentos de preparação que podem aparecer em uma sequência criada no ponto de dificuldade mínima da curva de dificuldade.
- Porcentagem de Segmentos de Alívio mínima (PSAmi) – A porcentagem de segmentos de alívio que podem aparecer em uma sequência criada no ponto de dificuldade mínima da curva de dificuldade.

- Porcentagem de Segmentos Bloqueantes máxima (PSBma) – A porcentagem de segmentos bloqueantes que podem aparecer em uma sequência criada no ponto de dificuldade máxima da curva de dificuldade.
- Porcentagem de Segmentos de Preparação máxima (PSPma) – A porcentagem de segmentos de preparação que podem aparecer em uma sequência criada no ponto de dificuldade máxima da curva de dificuldade.
- Porcentagem de Segmentos de Alívio máxima (PSAma) – A porcentagem de segmentos de alívio que podem aparecer em uma sequência criada no ponto de dificuldade máxima da curva de dificuldade.

Através da definição da curva de dificuldade e do estabelecimento dos valores, é possível realizar a conversão da curva de dificuldade para a curva de progresso.

4.6.3 Conversão entre Curvas

Para tornar mais simples o processo de conversão entre curva de dificuldade e curva de progresso, a curva de dificuldade definida pelo *game designer* varia em um intervalo entre zero e um. A curva de progresso propriamente dita é gerada sob demanda durante a partida com base em tal curva de dificuldade e nas proporções de segmentos, respeitando as regras de sequenciamento.

Como a variação nas quantidades de segmentos é definida de forma linear, a seguinte equação é utilizada para definir a porcentagem de aparecimento de um determinado tipo de segmento em uma sequência com base na dificuldade:

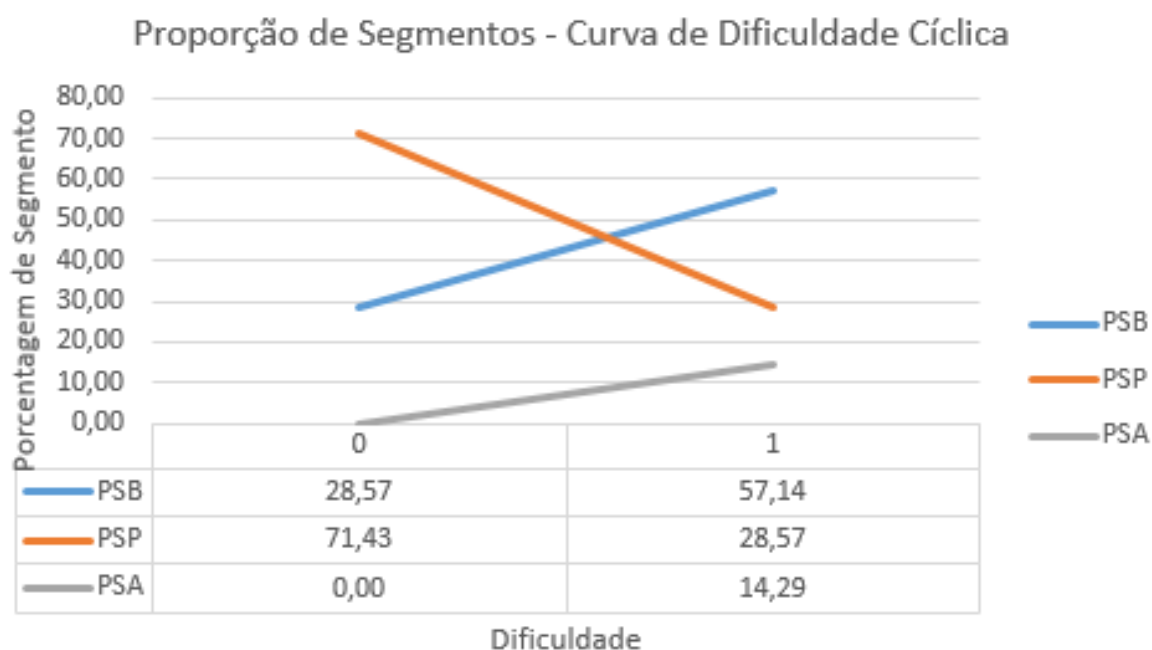
$$f(d) = (PSma - PSmi) * d + PSmi \quad (4.1)$$

Nesta fórmula, “PSma” corresponde à porcentagem mínima de segmentos de um determinado tipo, “PSmi” corresponde à porcentagem máxima de segmentos de um determinado tipo e “d” corresponde à dificuldade. Com esta fórmula é possível obter as porcentagens de cada tipo de segmento para todos os pontos da curva de dificuldade.

Como já foi especificado, a conversão da curva de dificuldade para curva de progresso é realizada durante a partida. Durante a partida, em intervalos regulares, a curva de dificuldade é consultada e a dificuldade a cada intervalo é obtida. Este valor é então aplicado na equação 1 para cada tipo de segmento – SB, SP e SA – fornecendo as porcentagens de cada tipo de segmento que devem existir na sequência seguinte. Com base no tamanho da sequência e nas porcentagens dos segmentos, define-se quantos segmentos de cada tipo devem existir na sequência, então a sequência é gerada automaticamente respeitando as restrições apresentadas anteriormente.

Com o propósito de ilustrar um exemplo da utilização desta técnica, definimos um exemplo para as proporções de segmentos na figura 8 e curva de dificuldade na figura 9.

Figura 8 – Exemplo de Proporção de Segmentos



Fonte: O autor

Figura 9 – Exemplo de Curva de Dificuldade



Fonte: O autor

No exemplo em questão, o gráfico de proporção de segmentos foi construído utilizando a equação 1 utilizando os valores 15, 25, 60, 70, 30 e 0 para as variáveis PSBmi, PSPmi, PSAmi, PSBma, PSPma e PSAmi – como pode ser visto na legenda do gráfico.

Se consultarmos a curva de dificuldade na unidade de tempo 3 o valor numérico da dificuldade é 0,4. Ao utilizarmos esse valor na equação 1 para cada um dos tipos de segmentos, obtemos os valores 37, 27 e 36 para as variáveis PSB, PSP e PSA respectivamente. Se for adotada uma sequência de tamanho 5, deverão haver 2 segmentos bloqueantes, 1 segmento de preparação e 2 segmentos de alívio (em caso de valores não inteiros, o sistema automaticamente considera o inteiro mais próximo).

Com essa informação, o sistema pode então gerar uma sequência que contenha tais quantidades e respeite as restrições indicadas no sequenciamento, como por exemplo SA-SB-SP-SA-SB.

Por fim, como estamos tratando de jogos de infinitos ciclos, fica claro que apenas uma curva de dificuldade finita não seria o suficiente para gerar partidas que possam ter duração potencialmente infinitas. Por conta disso é criada outra curva de dificuldade, denominada de curva de dificuldade cíclica. Todos os demais conceitos aplicados na curva de dificuldade finita também se aplicam a esta, a única diferença é que a curva de dificuldade cíclica começa a ser usada quando o jogador chega ao fim da curva de dificuldade finita. Caso jogador chegue ao fim da curva de dificuldade cíclica, o sistema retorna ao ponto de origem da curva e a partida prossegue. Este processo continua ocorrendo até que a partida termine.

Como o *game designer* possui o controle sobre o tamanho da sequência, as variáveis de porcentagem de segmento e as curvas de dificuldade, é possível criar diversos comportamentos enquanto se garante de forma automática a aplicação do conceito de curva de progresso.

4.7 Escolha do Segmento

A escolha do segmento é um processo simples e que apresenta um conjunto pequeno de restrições. A princípio, as únicas restrições definidas são de que a escolha do segmento deve evitar repetições, com o objetivo de aumentar a variabilidade dos elementos de jogo, e o segmento selecionado deve ter um bom nível de aproximação com a dificuldade desejada no momento.

Dado que temos um conjunto de segmentos, cada um devidamente classificado de acordo com sua dificuldade, realizamos a separação destes em *buckets* (grupos), os quais incluem segmentos dentro de níveis de dificuldade pré-estabelecidos. Por exemplo, se a dificuldade dos segmentos varia de 0 a 1, pode-se criar 4 buckets: de 0 a

0.25, de 0.26 a 0.5, de 0.51 a 0.75 e por fim de 0.76 a 1. Quão menor forem os buckets, maior precisão se terá com os níveis de dificuldade dos segmentos dentro deles, mas menores variações de segmentos estará disponível em cada um.

Este processo de separação em diferentes buckets é executado de forma *offline*, durante o desenvolvimento do jogo, e o seu resultado é a criação de várias listas de segmentos (denominadas de “listas de segmentos ativos”), uma para cada bucket. Durante a partida, os segmentos são selecionados de forma on-line com base na definição estabelecida na curva de progresso da etapa 3. Se a curva de progresso define que deve ser inserido na partida um segmento de um dado nível de dificuldade, o seletor escolhe aleatoriamente um segmento correspondente na lista de segmentos ativos.

No entanto, apenas tal método de escolha de segmentos gera um problema: é possível que o mesmo segmento apareça repetidas vezes em um curto espaço de tempo por conta da natureza aleatória da seleção. Por conta disso cria-se também um segundo grupo de listas: as listas de segmentos inativos. A relação entre essas listas é de um para um, ou seja, para cada lista de segmentos ativos, existe uma lista de segmentos inativos correspondentes.

Sempre que um segmento é selecionado a partir da lista de segmentos ativos, ele é removido desta lista e adicionado na lista de segmentos inativos. Quando eventualmente a lista de segmentos ativos fica vazia, todos os segmentos existentes na lista de segmentos inativos são readicionados na lista de segmentos ativos e removidos da lista de segmentos inativos.

Este processo garante que os segmentos existentes em um bucket apenas sejam reutilizados na partida uma vez que todas as opções existentes já tenham sido utilizadas, dessa forma aumentando a variabilidade durante cada partida.

4.8 Resumo

Este capítulo apresentou de forma detalhada o processo proposto para geração procedimental de conteúdo e os respectivos detalhamentos das etapas desse processo. Como já foi dito, um dos diferenciais deste processo está em sua capacidade de compartimentar as diversas etapas envolvidas na geração de conteúdo, de forma a tornar cada uma independente. Isto em contrapartida torna possível utilizar diversas técnicas diferentes para cada etapa e facilmente alterar as técnicas utilizadas de acordo com as necessidades do jogo.

5 ESTUDO DE CASO

Neste capítulo é feito um estudo de caso da utilização da solução proposta em um jogo comercial.

5.1 Introdução

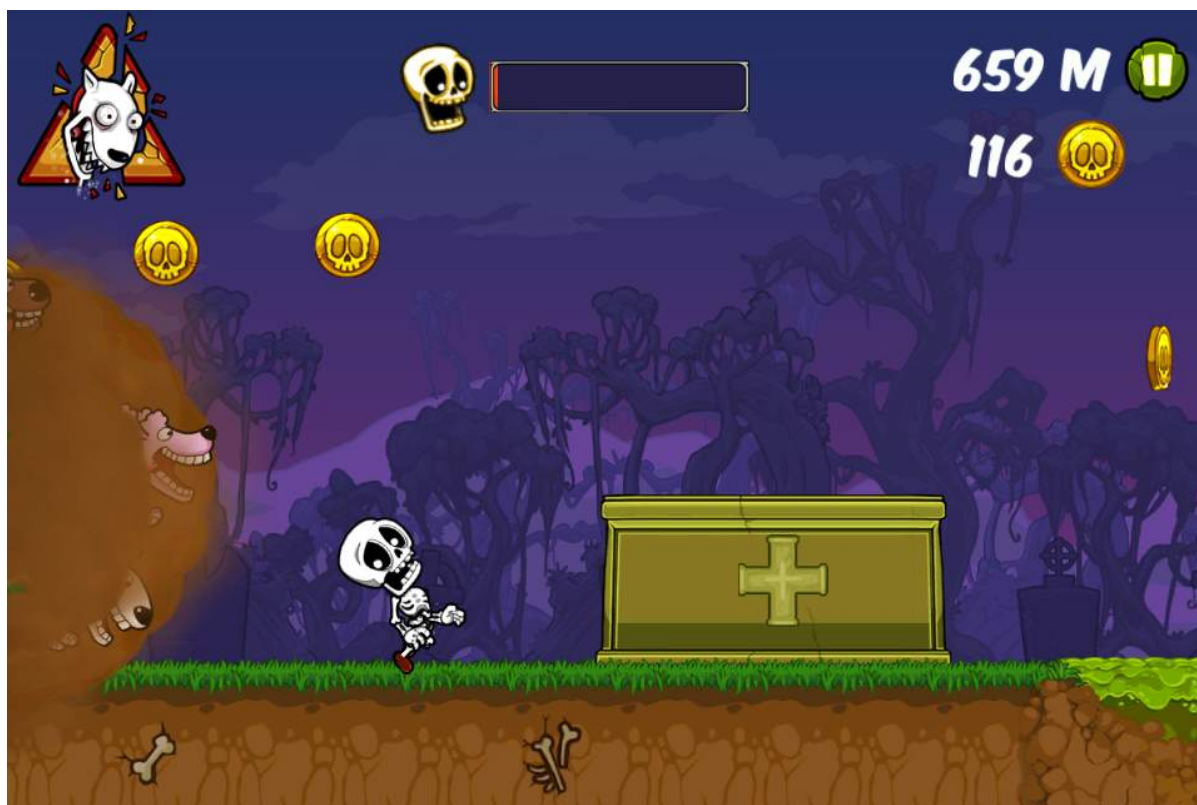
Este capítulo visa analisar a aplicação do processo proposto em um jogo de infinitos ciclos. Inicialmente é feita uma breve explicação do funcionamento do jogo e das mecânicas envolvidas neste, e em seguida as técnicas utilizadas nas etapas do processo são elicitadas.

5.2 Jogo utilizado no estudo

Para realizar este estudo foi utilizado o jogo Boney the Runner (BIGHUT GAMES, 2012a), título desenvolvido pela produtora BigHut Games (2012b). No momento da escrita desse documento, o jogo encontra-se disponível nas plataformas Google Play e iTunes e já conta com mais de 1 milhão de downloads e 5 mil usuários ativos diariamente. Todas as definições aqui apresentadas dizem respeito à versão do jogo que foi utilizada para os propósitos deste trabalho e não refletem a versão mais recente do jogo disponível na loja.

Boney the Runner é um endless runner (jogo de corrida infinita) no qual o jogador controla Boney, uma caveira animada que um dia despertou em um cemitério e foi perseguida por uma matilha de cães ferozes. A figura abaixo retrata uma partida do jogo em andamento.

Figura 10 – Uma partida do jogo Boney the Runner Runner



Fonte: BigHut Games, 2012

Na Figura 10 estão presentes alguns dos principais elementos do jogo: o avatar, os cães e um obstáculo. O avatar, Boney, se encontra à direita dos cães e está constantemente correndo da esquerda para a direita, os cães por sua vez correm em direção ao personagem com o objetivo de capturá-lo. É importante salientar que os cães correm a uma velocidade menor que o avatar, dessa forma o avatar fica cada vez mais distante dos cães enquanto corre.

Durante o caminho existem também obstáculos com formato de tumbas que devem ser evitados pelo jogador através da única ação que este pode realizar, que é o salto. Se o jogador saltar no tempo e altura certos, ele conseguirá desviar da tumba e não perderá velocidade, no entanto se o jogador cometer um erro e deixar o avatar colidir com a tumba, o avatar deverá escalá-la, o que o mantém parado em uma mesma posição por algum tempo, como pode ser visto na figura abaixo.

Figura 11 – Uma partida do jogo Boney the Runner na qual o jogador cometeu um erro e por conta disso o avatar teve que escalar o obstáculo



Fonte: BigHut Games, 2012

Enquanto o avatar escala a tumba, os cães se aproximam. Se os cães conseguirem capturar o avatar, a partida é finalizada e a pontuação do jogador é calculada. Ao longo do caminho existem também moedas que o jogador pode obter e posteriormente usar para destravar habilidades que tornam o jogo mais fácil, como escalar as tumbas mais rapidamente.

Além de tais componentes básicos, existem também componentes mais complexos, que são o musgo, a lama e os fantasmas, os quais serão apresentados a seguir.

Na figura 12 é possível identificar o musgo no chão. O musgo é um componente que faz o avatar deslizar a uma velocidade muito mais alta do que a velocidade com a qual este normalmente corre. O seu efeito dura apenas enquanto houver contato entre os pés do avatar e o chão.

Figura 12 – Uma partida do jogo Boney the Runner na qual o avatar desliza sobre o musgo



Fonte: BigHut Games, 2012

Na Figura 13 é possível identificar trechos de lama no chão com mãos saindo de tais trechos. Quando o jogador entra em contato com a lama, ele corre a uma velocidade mais baixa que a normal, mas assim que consegue fugir de tal trecho, a sua velocidade volta ao valor original.

Figura 13 – Uma partida do jogo Boney the Runner na qual o avatar anda sobre a lama



Fonte: BigHut Games, 2012

Por fim, é possível encontrar também fantasmas espalhados pelo cenário, como pode ser visto na Figura 14. Fantasmas são entidades que, ao entrar em contato com o avatar, o deixam mais lento por alguns segundos. Os fantasmas ficam em posições fixas, assim como as tumbas, e podem ser desviados pelo jogador se este pular no tempo e altura corretos.

Figura 14 – Uma partida do jogo Boney the Runner na qual o avatar está cercado por fantasmas



Fonte: BigHut Games, 2012

Estes são todos os elementos que compõem a versão do jogo utilizada para o propósito deste trabalho. A seguir será explicado como foi aplicado neste jogo o processo proposto.

5.3 Aplicação do Processo

Como já foi dito anteriormente neste trabalho, o processo para a geração de partidas para jogos de infinitos ciclos foi dividido em quatro partes distintas. No entanto, antes de elucidar como estas quatro etapas foram utilizadas no jogo em questão, é necessário definir como o conceito de segmento é aplicado no jogo Boney the Runner.

Todos os segmentos no Boney the Runner são compostos a partir dos elementos básicos apresentados anteriormente, que são túmulos, moedas, musgo, lama e fantasmas. Durante a partida, segmentos são justapostos sob demanda conforme o jogador avança da esquerda para a direita.

Na versão inicial do jogo, todas as etapas do processo foram executadas de forma manual e será descrito a seguir o processo associado a cada uma delas.

Para a definição da curva de dificuldade, um *game designer* definiu duas curvas de dificuldade em função do tempo: uma que é usada durante os primeiros minutos de

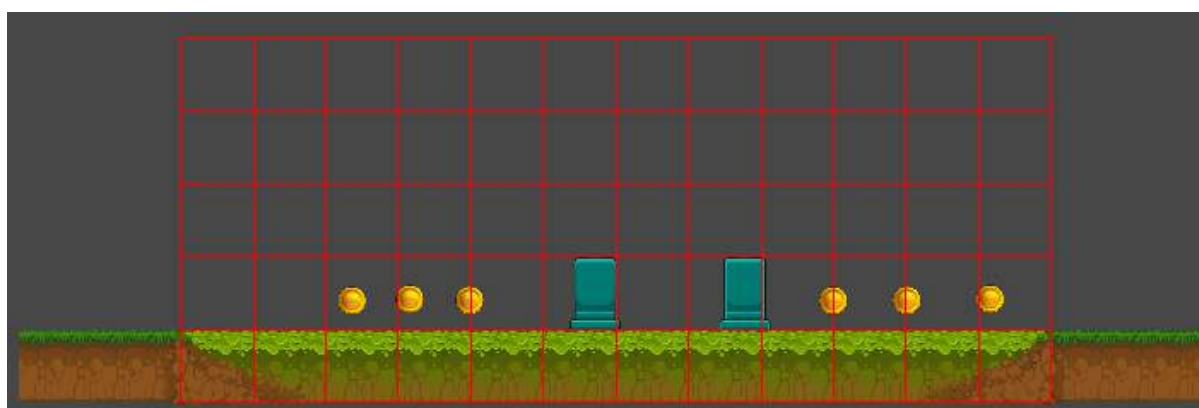
jogo e outra que é usada de forma circular depois que o jogador consegue ultrapassar determinada distância, ou seja, para jogadores que conseguem chegar a distâncias muito grandes no jogo, utiliza-se uma curva que fica em um *loop* (laço) infinito.

A criação dos segmentos é feita de forma manual por um *game designer*, e segue algumas limitações pré-determinadas:

- Todos os segmentos devem ter um tamanho de precisamente 1536 pixels, valor escolhido por ser um múltiplo do tamanho de tela que foi considerado o padrão para o desenvolvimento do jogo
- Os segmentos devem ser montados utilizando apenas componentes básicos: moedas, tumbas, musgo, lama e fantasmas

Com estas limitações garante-se uma maior coerência na etapa seguinte de classificação dos segmentos, que será vista a seguir. Na Figura 15 é possível visualizar um segmento do jogo visto dentro do editor de segmentos. É importante observar a linha quadriculada que determina os limites horizontais e verticais de tal segmento.

Figura 15 – Um segmento do jogo Boney the Runner



Fonte: BigHut Games, 2012

O processo inicial de classificação dos segmentos é feito de forma manual por *game designers*, que classificam um segmento de acordo com três níveis de dificuldade: fácil (F), médio (M) e difícil (D). Além de tais níveis, existem ainda os segmentos do tipo bônus (B) que não apresentam perigo ao jogador e apenas contém moedas. No entanto este processo apresenta falhas por ser baseado apenas na intuição do *game designer* em questão, que nem sempre está correta, principalmente quando se considera a experiência prévia que este já teve com o jogo.

Nesta versão do jogo estão disponíveis um total de 230 segmentos feitos manualmente, sendo esses divididos, de acordo com o *game designer*, da seguinte forma: 99 da dificuldade fácil, 62 da dificuldade média, 55 da dificuldade difícil e 14 segmentos

que são considerados “bônus”. Segmentos do tipo bônus não oferecem nenhum perigo ao jogador e normalmente contém apenas uma grande quantidade de moedas, como uma forma de dar um estímulo positivo ao jogador durante o jogo.

Por fim, para tratar da inserção dos segmentos durante a partida, utiliza-se um algoritmo de sorteio bastante simples: de acordo com o momento atual do jogo, busca-se na curva de dificuldade qual deve ser a dificuldade do próximo segmento, uma vez obtida a dificuldade do segmento, é sorteado um número aleatório entre um e a quantidade total de segmentos daquele tipo, o número sorteado indica qual será o segmento obtido da lista.

Com esta visão geral dos elementos que compõem a geração de níveis para o jogo em questão, podemos focar nas formas como foram otimizadas cada uma das etapas. Na lista seguinte há um breve resumo das soluções adotadas em cada uma das etapas, e nas próximas subseções cada uma dessas soluções serão detalhadas:

- 1) Criação dos segmentos – Foi realizada uma implementação simplificada da técnica descrita por Smith, Whitehead e Mateas (2011).
- 2) Classificação dos segmentos – Redes neurais foram treinadas a partir de dados coletados em diversas partidas.
- 3) Definição da curva de dificuldade – Foi realizada uma implementação direta da solução proposta no capítulo 5, com pequenas adaptações.
- 4) Inserção dos segmentos no jogo – Foi realizada uma implementação direta da solução proposta no capítulo 5, sem adaptações.

5.4 Geração de Segmentos

Para realizar a geração dos segmentos, foi utilizada uma versão simplificada da técnica proposta por Smith, Whitehead e Mateas (2011), mantendo-se os conceitos de geração rítmica e satisfação de restrições. A geração rítmica realiza um processo reverso na criação de níveis: o *game designer* especifica como deseja que o jogador pressione os botões (comumente seguindo um ritmo), e com base nisso é gerado um nível que exige que o jogador pressione os botões na mesma forma que foi indicada pelo *game designer*.

A geração de segmentos é um processo em duas etapas:

- 1) Criação de segmentos brutos – Nesta etapa dezenas de segmentos são criados usando como entrada os ritmos de toque especificados pelo *game designer*.

- 2) Validação de segmentos – Nesta etapa ocorre a verificação dos segmentos gerados para garantir que eles respeitam as restrições estabelecidas pelos *game designers*.

Estas duas etapas do processo de geração de segmentos serão detalhadas a seguir.

5.4.1 Geração de Segmentos Brutos

O gerador de segmentos utiliza uma versão simplificada das abordagens rítmicas expostas no capítulo 3 e adaptadas para o Boney the Runner. Esta geração se baseia no ritmo com qual se toca na tela do aparelho e na duração de cada toque.

Como no Boney the Runner a duração do toque na tela indica a altura que o avatar consegue pular, é necessário considerar tanto o momento do toque quanto a duração deste durante a geração de segmentos. Em testes com o jogo, constatou-se que para realizar o salto de altura mínima é necessário um toque de 0.1 segundo (o toque mínimo), e para realizar o salto de altura máxima é necessário um toque de 0.5 segundo (o toque máximo). Além disso, o tempo total que é necessário para atravessar um segmento sem que o avatar seja afetado por elementos externos é de 2.4 segundos.

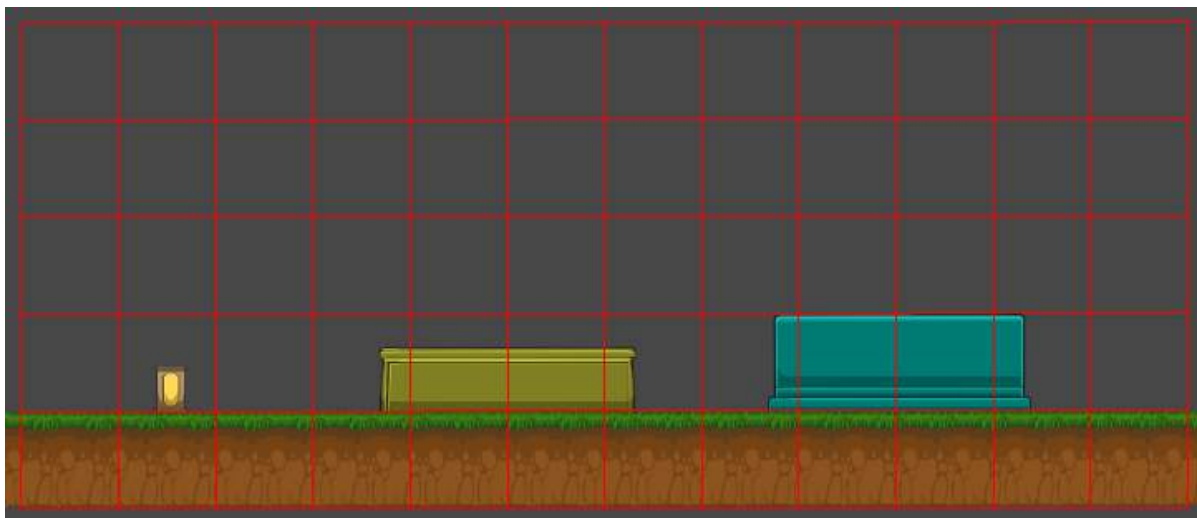
Com estas informações, é possível estabelecer uma linha do tempo de eventos que podem ser convertidos em obstáculos correspondentes. Por exemplo, um evento de toque com duração de 0.5 segundos é associado a um obstáculo alto, já um evento de 0.1 segundo corresponde a um obstáculo baixo.

Dessa forma, o *game designer* provê as seguintes entradas para o gerador:

- Elementos utilizáveis – Indica quais dos elementos de gameplay devem ser utilizados nos segmentos. As opções que podem ser selecionadas são tumba baixa, tumba média, tumba alta, lama, musgo e fantasmas.
- Lista de toques – Esta lista é apresentada no formato “X,Y”, no qual X é o momento temporal no qual o segmento deve ser inserido e Y é a duração do toque. Os números decimais devem ser separados por pontos e os elementos da lista devem ser separados por ponto e vírgula, espaços também podem ser inseridos e são ignorados. A seguinte lista de toques por exemplo seria considerada válida: “0.3, 0.25; 1, 0.5; 2, 0.1”, e sua interpretação é a seguinte: ao passar 0.3 segundos dentro do segmento, o jogador deve executar um salto que dura pelo menos 0.25 segundos; à contagem de 1 segundo, o jogador deve executar um salto da altura máxima; à contagem de 2 segundos, o jogador deve executar um salto da altura mínima.

A posição temporal do toque indica em qual posição o obstáculo deve ser posicionado: o tempo 0 equivale ao início do segmento, já o tempo 2.4 equivale ao final do segmento. Em contrapartida, a duração do toque indica qual é a altura aproximada do obstáculo em questão. As três variações de tumbas, que podem ser vistas na figura abaixo, foram utilizadas como medidas de referência para as alturas dos pulos – e consequentemente a duração dos toques.

Figura 16 – Três variações de tumba, da esquerda para a direita: baixa, média e alta

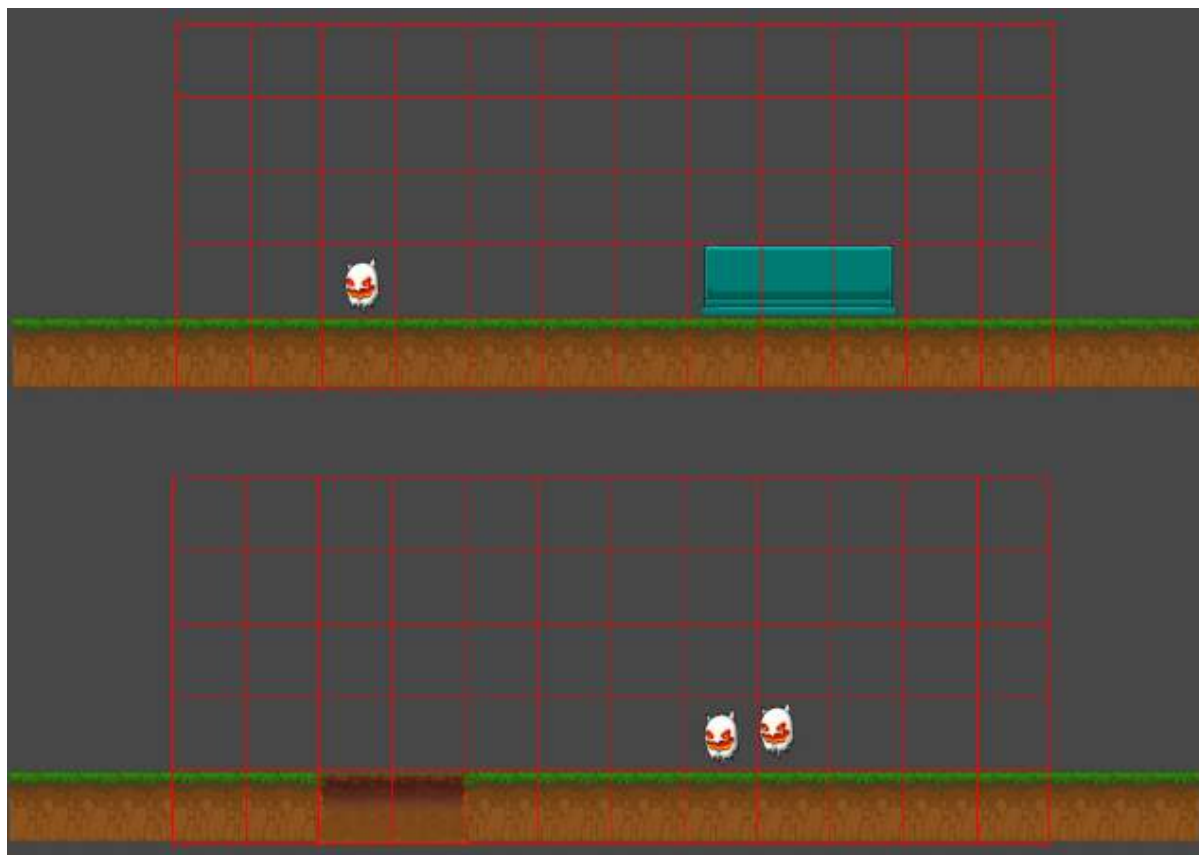


Fonte: BigHut Games, 2012

Toques de 0.1 a 0.25 segundo são considerados como saltos baixos, logo elementos são posicionados em uma altura equivalente a uma tumba baixa. Toques de 0.25 a 0.4 segundos são considerados como saltos médios, logo são posicionados elementos em uma altura equivalente às tumbas médias. Já toques de 0.4 segundos ou mais são considerados como saltos de altura elevada, logo os elementos são posicionados em uma altura equivalente às tumbas altas.

Para o caso de elementos que podem ter dimensão horizontal variável, como é o caso de tumbas, lama ou musgo, é escolhido um tamanho aleatório, adotando a restrição de que o fim do elemento deve ser antes do próximo toque ou antes do fim do segmento.

Com a lista de informações de posição temporal e duração do toque, e com os tipos de elementos que podem ser usados, o gerador é capaz de criar dezenas de segmentos diferentes para a mesma entrada. Na figura 17 é possível ver dois segmentos gerados a partir do mesmo padrão rítmico.

Figura 17 – Dois segmentos diferentes gerados a partir da mesma entrada

Fonte: BigHut Games, 2012

Apesar de o gerador ser capaz de criar uma grande variedade de segmentos, não há como garantir a *playability* de tais segmentos ou o respeito a restrições estéticas – como por exemplo a restrição de que um fantasma não pode ser posicionado dentro de uma tumba – o que torna necessária a validação dos segmentos.

5.4.2 Validação de Segmentos

Para validar os segmentos gerados, utiliza-se uma engine de satisfação de restrições. A engine utilizada é a mesma especificada por Smith, Whitehead e Mateas (2011): Choco (PRUD'HOMME; FAGES; LORCA, 2016) .

As restrições definidas e implementadas no sistema de validação estão especificadas abaixo:

- 1) Não pode haver interseção entre nenhum tipo de elemento: esta é uma restrição principalmente estética para evitar que dois elementos de jogo se sobreponham de forma indevida.
- 2) Fantasmas não podem estar posicionados imediatamente acima do início de uma tumba: se um fantasma é posicionado desta forma, cria-se uma barreira

que não pode ser ultrapassada sem que o jogador perca tempo, algo negativo porque, independente do nível de habilidade do jogador, ele não conseguirá evitar tal obstáculo

- 3) Ao fim de uma região de lama ou musgo, fantasmas não podem estar posicionados na altura de saltos médios ou altos: nesta configuração, após passar por um trecho que já reduziu bastante a sua velocidade o jogador seria obrigado a atingir o fantasma, o que causaria uma perda ainda maior de tempo.
- 4) Mais de dois fantasmas em uma mesma altura não podem estar dispostos consecutivamente ao longo de um trecho horizontal: caso isso ocorra, o jogador ficará impossibilitado de fugir dos fantasmas, sendo obrigado a atingi-los.

Aplicando tais restrições aos segmentos gerados, é feita uma filtragem dos segmentos que apresentam problemas de *playability* e somente os que respeitam as restrições são considerados apropriados para serem utilizados no jogo.

Com a execução do processo de criação de segmentos brutos e seleção de segmentos apropriados para o jogo, é necessário classificar os segmentos para que estes possam ser usados.

5.5 Classificação de Segmentos

Para a classificação dos segmentos, foi utilizado o processo de classificação especificado no capítulo 4, dessa forma será detalhado a seguir os elementos controláveis e não controláveis e elementos de gameplay identificados, assim como o processo de coleta de dados para treinamento das redes neurais. É importante ressaltar que todas as métricas foram definidas com base em discussões com os *game designers* do jogo Boney the Runner por estes apresentarem um melhor entendimento das mecânicas do jogo.

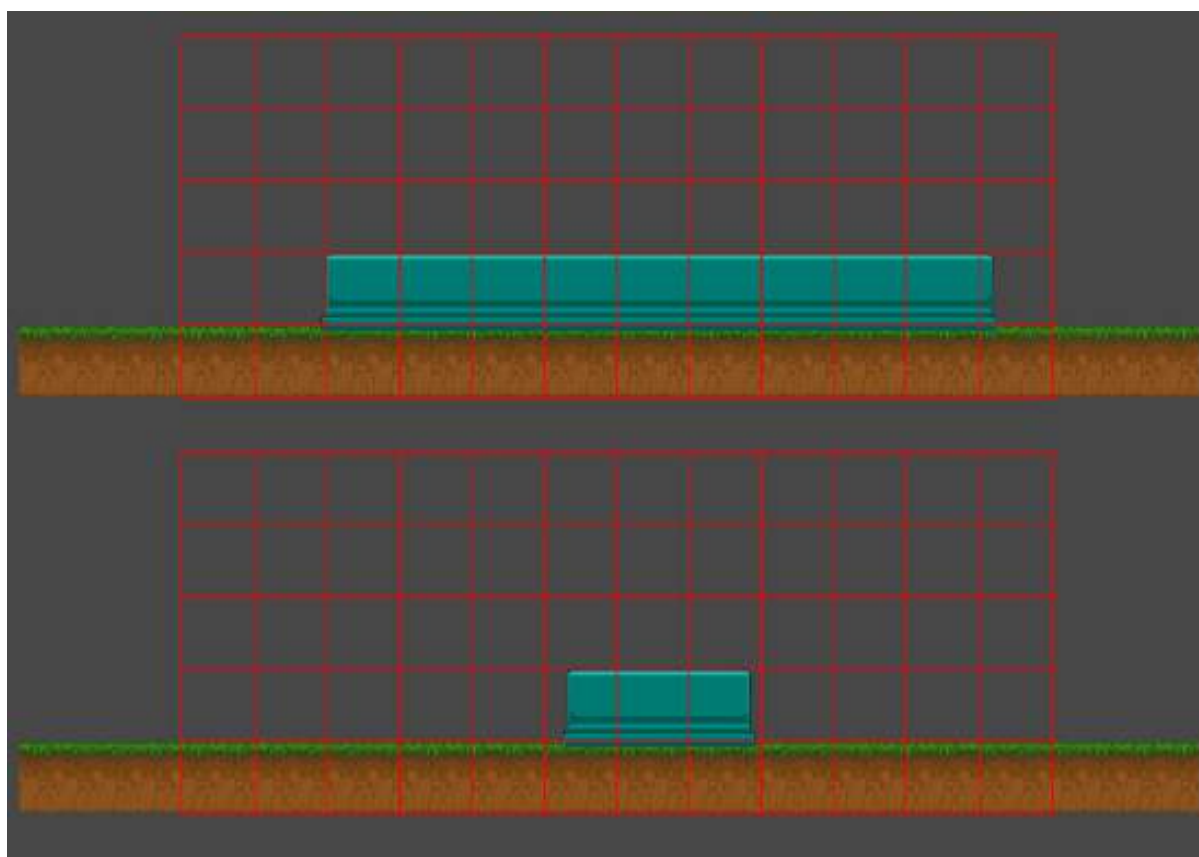
Além disso, para todo o processo de classificação de segmentos foram utilizados os segmentos feitos manualmente pelos *game designers*. O propósito disso é possibilitar a comparação da dificuldade estipulada pelo *game designer* e a dificuldade calculada pelo sistema.

5.5.1 Elementos Controláveis

Os elementos controláveis são calculados com base no posicionamento das entidades básicas no segmento. Apesar de o *game designer* não alterar diretamente tais medidas, é possível afetá-las de forma indireta através do posicionamento das entidades básicas no segmento.

É importante notar que os elementos tumba, lama e musgo podem apresentar variações de largura, como pode ser visto na Figura 18. Por conta disso, ao efetuar o cálculo das métricas relacionadas a tais elementos, antes é feita a divisão de cada um deles em uma quantidade “n” de sub-elementos com o tamanho fixo de 72 pixels, que é o tamanho mínimo no qual estes são encontrados nos segmentos. Dessa forma, uma tumba com largura 720 pixels é convertida em dez tumbas adjacentes de 72 pixels cada para efeitos de análise. Para os fins deste trabalho, cada um desses sub-elementos será denominado de “*slice*” (fatia), dessa forma no exemplo anterior teríamos 10 slices de tumba.

Figura 18 – Dois exemplos de segmentos gerados no editor do jogo Boney the Runner, em cada um dos segmentos é possível identificar o mesmo elemento com larguras diferentes



Fonte: BigHut Games, 2012

Os elementos controláveis medidos foram separados a seguir de acordo com o tipo de elemento ao qual estes correspondem, como pode ser visto a seguir:

- Tumbas - as tumbas presentes no jogo podem aparecer em três variedades de altura, alta, média e baixa, mas suas larguras podem ser modificadas livremente pelo *game designer*, como foi indicado anteriormente.

- Quantidade de tumbas altas – utilizando o método de divisão de tumbas previamente descrito, obtém-se a quantidade total de slices de tumbas altas presentes no segmento.
 - Quantidade de tumbas médias – mesmo conceito de tumbas altas, mas aplicado a tumbas médias.
 - Quantidade de tumbas baixas – mesmo conceito de tumbas altas, mas aplicado a tumbas baixas.
 - Altura média – A altura média de todos os tipos de tumbas presentes no segmento.
 - Variância da altura – A variância de altura de todos os tipos de tumbas presentes no segmento.
 - Posição média – A posição média no eixo x de todos os tipos de tumbas presentes no segmento.
 - Variância da posição – A variância da posição no eixo x de todos os tipos de tumbas presentes no segmento.
 - Porcentagem de área coberta por tumbas – Porcentagem do terreno que contém tumbas.
- Fantasma
 - Quantidade de Fantasmas – Quantidade total de fantasmas presentes no segmento.
 - Posição X Média – O ponto médio da posição dos fantasmas no eixo x.
 - Variância da posição X – A variância da posição dos fantasmas no eixo x.
 - Posição Y Média – O ponto médio da posição dos fantasmas no eixo y.
 - Variância da posição Y – A variância da posição dos fantasmas no eixo y.
- Lama
 - Quantidade de slices – A quantidade de slices de lama presentes no segmento.
 - Posição média dos slices – A posição média dos slices de lama presentes no segmento.
 - Variância da posição dos slices – A variância da posição dos slices de lama presentes no segmento.
 - Porcentagem da área coberta por lama – Porcentagem do terreno que é coberto com lama.

- Musgo – As mesmas métricas utilizadas para lama são também utilizadas para o musgo, por apresentarem uma forma de disposição no cenário muito semelhante.
 - Quantidade de slices.
 - Posição média dos slices.
 - Variância da posição dos slices.
 - Porcentagem da área coberta por musgo.
- Moeda - As mesmas métricas utilizadas para fantasma são também utilizadas para as moedas, por apresentarem uma forma de disposição muito semelhante.
 - Quantidade de Moedas.
 - Posição X Média.
 - Variância da posição X.
 - Posição Y Média.
 - Variância da posição Y.

5.5.2 Elementos de Gameplay

Os elementos de gameplay são obtidos durante as partidas do jogo de acordo com as ações executadas pelos jogadores. Para cada partida, são coletadas métricas relativas a eventos ocorridos em cada segmento inserido na partida em questão. As métricas de elementos de gameplay obtidas são descritas a seguir:

- Distância inicial entre o avatar e os cães – A distância entre o avatar controlado pelo jogador e os cães no momento em que o avatar entra no segmento.
- Distância final entre o avatar e os cães – A distância entre o avatar controlado pelo jogador e os cães no momento em que o avatar sai do segmento.
- Tempo necessário para atravessar o segmento – O tempo que o jogador levou para cruzar o segmento.
- Ocorreu morte no segmento – Assume o valor 1 se o avatar foi capturado pelos cães no segmento em questão e 0 caso contrário.
- Diferença de tempo em relação ao tempo ótimo – A diferença entre o tempo que o jogador de fato precisou para atravessar o segmento e o tempo que seria necessário para atravessar o segmento sem a influência de elementos externos. Como os segmentos possuem a largura igual e pré-determinada e o

personagem corre a uma velocidade constante, é possível calcular o tempo que o jogador levará para atravessar tal segmento sem que ele seja impulsionado ou atrasado por algum elemento presente no cenário. O valor obtido para o tempo necessário para cruzar um segmento sem que nenhum efeito seja exercido sobre o avatar é de 2,4 segundos. Se o avatar conseguir cruzar o segmento em um tempo inferior a este, é porque ele entrou em contato com o musgo, que aumenta a sua velocidade. Se o avatar cruzar o segmento em um tempo superior a este, é porque colidiu com uma plataforma, um fantasma ou entrou em contato com a lama. Assim sendo, esta métrica pode adquirir valores negativos quando o jogador atravessa um segmento mais rápido do que o esperado, valores positivos quando é atrasado por algum elemento, ou o valor zero se nada afetou o avatar enquanto este cruzava o segmento.

- Total de moedas coletadas – O total de moedas que o jogador coletou no segmento em questão.
- Porcentagem de moedas coletadas – O percentual de moedas que o jogador conseguiu coletar do total de moedas disponível no segmento.

Os dados foram coletados através de partidas jogadas por jogadores com diversos níveis de experiência e habilidade com jogos desse gênero. No total foram coletados dados referentes a 184 jogos, dos quais 53 foram considerados como jogos inválidos, restando 131 partidas. Um jogo é considerado inválido quando o avatar morre antes do quinto segmento ou durante este, e a razão para isto é que após testes foi verificado que se o jogador não pressionar nenhuma tecla durante o jogo, o avatar consegue chegar em segurança até o quinto segmento da partida, mas dificilmente consegue ultrapassar deste. Optou-se por remover partidas que tenham menos do que cinco segmentos porque essas podem ser partidas nas quais o jogador se esqueceu que o jogo estava aberto e o deixou rodando enquanto fazia outras ações.

O teste usado para descobrir a quantidade ideal de segmento a partir da qual uma partida é considerada válida é bastante simples: o jogo foi deixado em execução por 50 partidas sem nenhuma interação do usuário. Sempre que ocorria a captura do avatar, o número do segmento no qual tal captura ocorreu era salvo e outra partida era imediatamente iniciada. Após estas execuções, constatou-se que em 86% partidas o avatar foi capturado no quinto segmento, em 8% partidas no sexto segmento e em 6% partidas no quarto. Considerando-se que grande parte das capturas ocorrem no quinto segmento, este valor foi o escolhido para determinar a validade de uma partida.

Como diversas partidas foram jogadas, foram obtidos dados repetidos para diversos tipos de segmento, então foi tirada a média de todos os elementos de gameplay

para cada um dos segmentos. Por fim, todos os dados foram normalizados para um intervalo entre 0 e 1.

5.5.3 Modelagem da Dificuldade

Com base nos dados coletados, foi feita uma aproximação dos elementos controláveis e elementos de gameplay para a dificuldade de cada segmento. Através de discussões com os *game designers* responsáveis pelo jogo, chegou-se à conclusão de que seria razoável fazer uma aproximação entre a dificuldade dos segmentos e o tempo perdido em cada segmento, afinal para o tipo de jogo em questão um segmento é considerado difícil quando o jogador tem uma chance maior de cometer erros, que consequentemente o farão perder mais tempo no segmento do que é o esperado.

Tendo em mente este conceito, foram removidos os segmentos considerados como “bônus” da lista de experimentação, afinal estes não oferecem perigo algum ao jogador e são inseridos no jogo apenas como uma forma de dar um grande volume de moedas, o que causa um reforço positivo.

Para fazer a aproximação foi utilizada a métrica “Diferença de tempo em relação ao tempo ótimo” (EGDTO), que foi descrita anteriormente. Com a remoção dos segmentos considerados como bônus, restaram 216 segmentos para os quais é necessário refazer a normalização. Este conjunto foi então dividido em três sub-conjuntos de acordo com a métrica EGDTO. Para realizar tal divisão, os segmentos foram ordenados de forma crescente de acordo com esta métrica e foram obtidos conjuntos com tamanhos iguais aos determinados pelo *game designer* originalmente: 99 segmentos de dificuldade fácil, 62 segmentos de dificuldade média e 55 segmentos de dificuldade alta.

Após tal divisão, foi possível estabelecer a relação da dificuldade com a métrica EGDTO, como pode ser visto a seguir:

- Segmento fácil - valor de EGDTO inferior a 0,45
- Segmento médio - valor de EGDTO inferior a 0,52 e superior a 0,45
- Segmento difícil - valor de EGDTO superior a 0,52

Com esta reclassificação, já é possível identificar os erros cometidos pelos *game designers* ao classificar os segmentos originalmente: dos segmentos reclassificados como fáceis, 21% correspondem a segmentos classificados originalmente como médios e 7% correspondem a segmentos classificados como difíceis. Dos segmentos reclassificados como médios, 32% correspondem a segmentos classificados como fáceis e 30% correspondem a segmentos classificados como difíceis. Dos segmentos

reclassificados como difíceis, 14% correspondem a segmentos classificados como fáceis e 32% correspondem a segmentos classificados como médios. Dessa forma, pode-se concluir que o game-designer foi capaz de prever corretamente a dificuldade de 52% dos segmentos.

Após tal reclassificação, a métrica EGDTO foi removida e em seu lugar foi inserida a métrica referente à dificuldade reclassificada (DIF), que pode assumir três valores: E (easy) referente à dificuldade “fácil”, M (medium) referente à dificuldade “média” e H (hard) referente à dificuldade “alta”.

5.5.4 Criação das Redes Neurais

As redes foram criadas utilizando a ferramenta Weka (HALL et al., 2009), e tanto a Rede A quanto a Rede B são *multilayer perceptrons* (MLP) com uma camada escondida cada. A quantidade de nós (n) na camada escondida é determinada pela fórmula abaixo, que é a fórmula padrão utilizada pela ferramenta.

$$n = \frac{\text{quantidade de atributos} + \text{quantidade de classes}}{2} \quad (5.1)$$

Para treinar e validar as redes, os dados foram divididos de forma aleatória em dois conjuntos: o conjunto de treino com 1/3 dos dados e o de validação com 2/3. O número de épocas para as duas redes foi fixo em 500 e a quantidade máxima de erros para a validação foi fixa em 20. Para chegar a bons valores para a taxa de aprendizagem e momento para as duas redes, ambas foram testadas com diversas variações desses dois valores. Foi feita a combinação de 4 valores para a taxa de aprendizagem (1.5, 0.3, 0.45 e 0.6) com 4 valores para o momento (0.2, 0.4, 0.6 e 0.8). A rede com menor valor de MSE de teste foi escolhida para cada caso.

Para a Rede A, utilizou-se taxa de aprendizagem de 0.15 e momento de 0.4. O MSE de teste para esta rede foi de 0.4 e ela foi capaz de classificar corretamente 70% das instâncias.

Para a Rede B foi utilizada a taxa de aprendizagem de 0.15 e momento de 0.8. O MSE de teste para esta rede foi de 0.25 e ela foi capaz de classificar corretamente 90% das instâncias.

Na Tabela 1 é possível encontrar as configurações das duas redes, enquanto na Tabela 2 estão disponíveis os resultados obtidos por ambas.

Tabela 1 – Configurações das redes A e B

	Rede A	Rede B
Quantidade de Épocas	500	500
Quantidade máxima de erros de validação	20	20
Taxa de Aprendizagem	0.15	0.15
Momento	0.4	0.8
Quantidade de nós na camada escondida	15	18

Fonte: O autor

Tabela 2 – Resultados obtidos com redes A e B

	Rede A	Rede B
MSE de Teste	0.4	0.25
Taxa média de verdadeiros positivos	0.7	0.904
Taxa média de falsos positivos	0.18	0.041
Média da área sob a curva ROC	0.81	0.951

Fonte: O autor

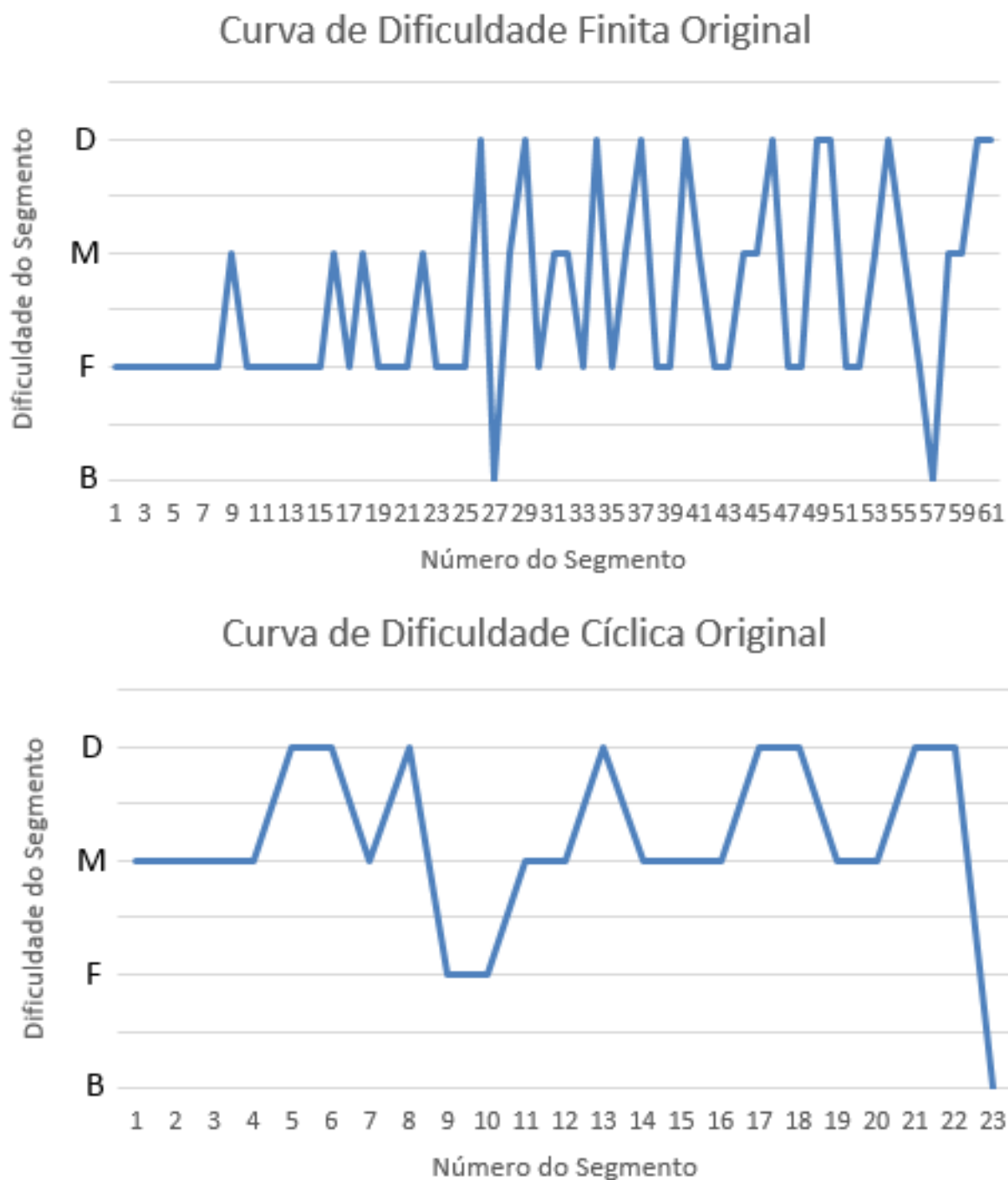
É importante notar o grande impacto que a remoção dos elementos de gameplay causou na eficiência da Rede A quando comparada à Rede B. Isto demonstra as vantagens de se coletar dados provenientes das partidas jogadas como uma forma de aproximar a dificuldade dos segmentos.

Apesar de não apresentar um desempenho tão bom quanto a Rede B, a Rede A demonstrou uma boa capacidade de prever a dificuldade dos segmentos, principalmente quando comparada à classificação original apresentada pelo *game designer*, o qual foi capaz de prever a dificuldade dos segmentos com 52% de precisão.

5.6 Curva de Progresso

O sistema de curva de progresso foi implementado de acordo com o que foi especificado no capítulo 4, sendo necessária apenas a definição dos parâmetros. No entanto, antes de fazer tais definições é necessário levar em consideração as curvas de dificuldades definidas originalmente pelos *game designers* do jogo, que podem ser vistas na figura abaixo.

Figura 19 – Curvas de dificuldade finita e dificuldade cíclica originais



Fonte: O autor

Tais curvas são do tipo “dificuldade do segmento” por “número do segmento”. A variável “número do segmento” é definida no início da partida e começa com o valor 1. Sempre que um segmento é inserido, esta variável é incrementada, dessa forma o jogo sabe qual é a dificuldade do segmento que deve buscar em seguida na curva de dificuldade.

Tendo em mente tal curva de dificuldade, para que seja feita a conversão para o

sistema de curva de progresso se faz necessária a definição de 3 parâmetros:

- 1) Tamanho da sequência
- 2) Proporção dos segmentos (tanto para a curva de dificuldade finita como para a curva de dificuldade cíclica)
- 3) Definição da curva de dificuldade (tanto para a curva de dificuldade finita como para a curva de dificuldade cíclica)

O nosso objetivo é manter a maior similaridade possível com os conceitos que os *game designers* aplicaram nas curvas de dificuldade originais durante o processo de conversão, por conta disso as curvas originais foram analisadas e usadas como base para a elaboração desses três parâmetros. Apenas o parâmetro “tamanho da sequência” foi escolhido com base em discussões com os designers, e o valor escolhido para tal tamanho foi 5.

5.6.1 Proporção dos Segmentos

Para obter as variáveis de proporção de segmentos mínimas (PSBmi, PSPmi, PSAmi) e máximas (PSBma, PSPma, PS Ama) foi feita uma conversão direta entre os tipos de segmentos:

- Segmentos Difíceis foram convertidos para Segmentos Bloqueantes
- Segmentos Médios foram convertidos para Segmentos de Preparação
- Segmentos Fáceis foram convertidos para Segmentos de Alívio

Feito isso, coletamos uma amostra de um terço dos primeiros tipos de segmentos que aparecem em cada gráfico para definir as proporções de segmentos mínimas, e um terço dos últimos tipos de segmentos para definir as proporções máximas.

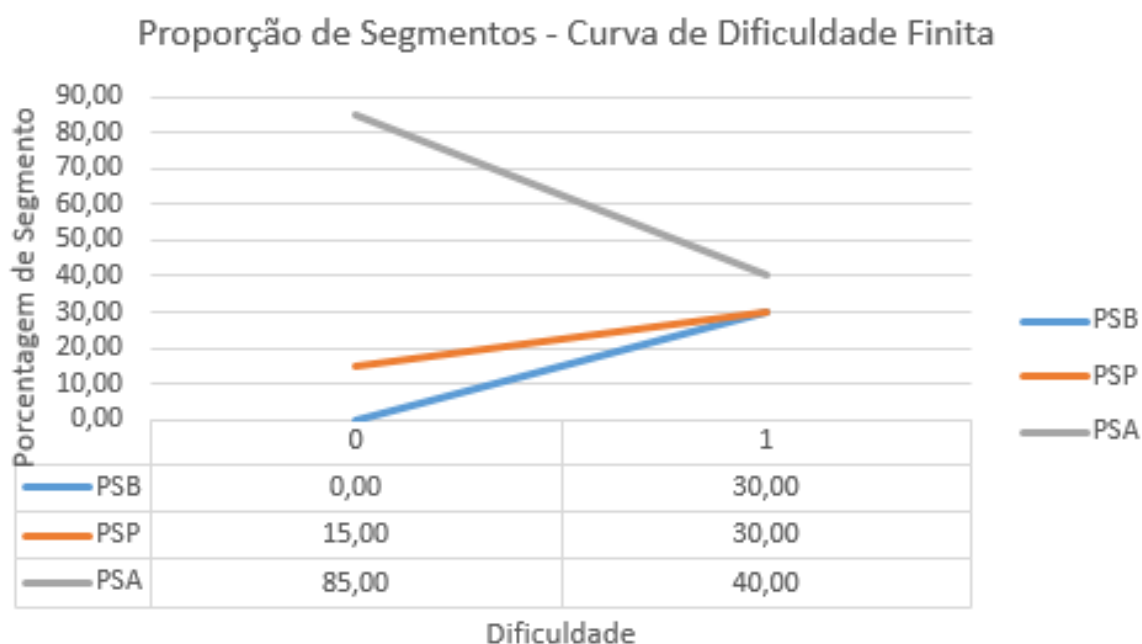
O resultado de tal conversão foram as seguintes proporções para a curva de dificuldade finita:

- Porcentagem de Segmentos Bloqueantes mínima (PSBmi): 0%
- Porcentagem de Segmentos de Preparação mínima (PSPmi): 15%
- Porcentagem de Segmentos de Alívio mínima (PSAmi): 85%
- Porcentagem de Segmentos Bloqueantes máxima (PSBma): 30%
- Porcentagem de Segmentos de Preparação máxima (PSPma): 30%

- Porcentagem de Segmentos de Alívio máxima (PSAma): 40%

A proporção dos tipos de segmentos em função da dificuldade pode ser vista na figura 20:

Figura 20 – Distribuição dos tipos de segmento de acordo com o nível de dificuldade para a curva de Dificuldade Finita



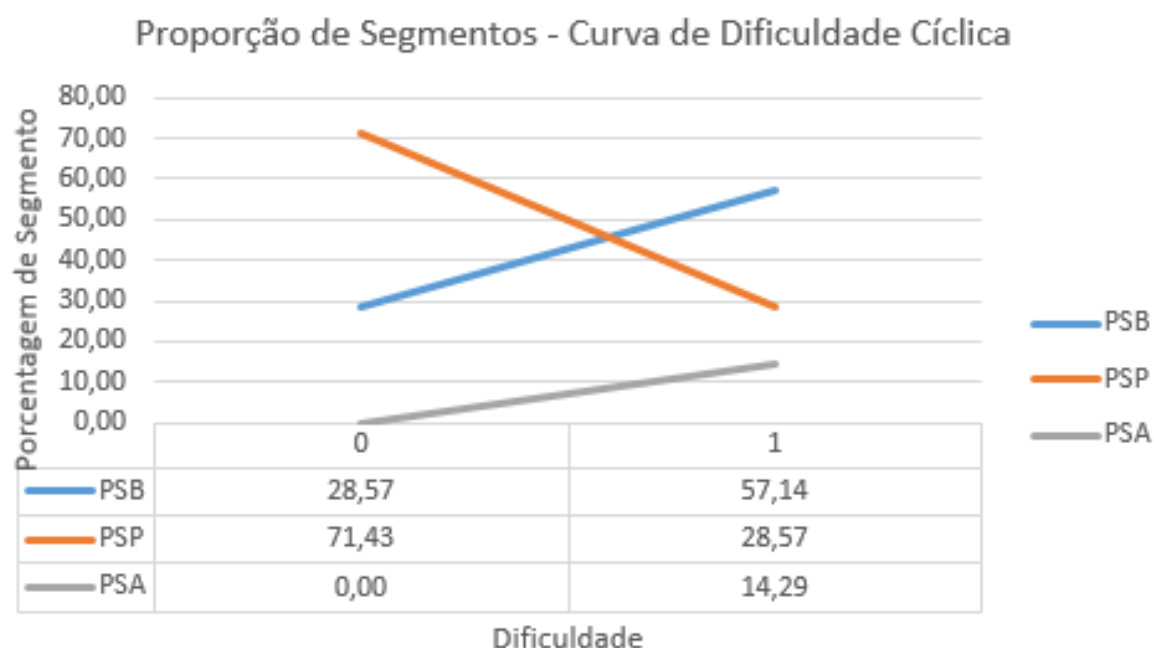
Fonte: O autor

Já para a curva de dificuldade cíclica, o resultado da conversão foi:

- Porcentagem de Segmentos Bloqueantes mínima (PSBmi): 28.57%
- Porcentagem de Segmentos de Preparação mínima (PSPmi): 71.43%
- Porcentagem de Segmentos de Alívio mínima (PSAmi): 0%
- Porcentagem de Segmentos Bloqueantes máxima (PSBma): 57.14%
- Porcentagem de Segmentos de Preparação máxima (PSPma): 28.57%
- Porcentagem de Segmentos de Alívio máxima (PSAma): 14.49%

A proporção dos tipos de segmentos em função da dificuldade pode ser vista na Figura 21:

Figura 21 – Distribuição dos tipos de segmento de acordo com o nível de dificuldade para a curva de Dificuldade Cíclica



Fonte: O autor

Visualizando as duas figuras acima, é possível visualizar claramente as intenções dos *game designers* ao projetar a distribuição dos segmentos no jogo: durante a curva de dificuldade finita, o jogador passa por um período de preparação. A partida começa com muitos segmentos fáceis e aos poucos tais segmentos são substituídos por outros de dificuldade média e alta, até que eventualmente os segmentos estão distribuídos de forma quase igual.

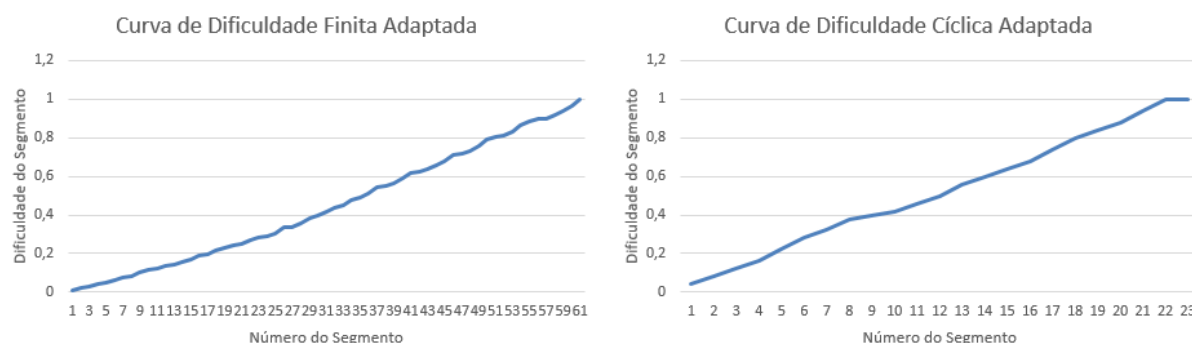
Já na proporção de segmentos da curva de dificuldade cíclica, é possível identificar uma presença consideravelmente maior de segmentos de dificuldade média, com um rápido crescimento na quantidade de segmentos difíceis. Algo interessante é que, ao fim da distribuição um segmento de dificuldade fácil é apresentado como forma de oferecer ao jogador um momento de alívio após a grande pressão apresentada neste ciclo.

5.6.2 Curvas de Dificuldade

Para a geração das curvas de dificuldade, foi feita uma soma acumulada da dificuldade em cada ponto do gráfico. Para realizar tais somas, as medidas de dificuldade foram convertidas para valores numéricos: segmentos fáceis foram convertidos para o valor 1, segmentos médios para o valor 2 e segmentos difíceis para o valor 3. O gráfico foi então normalizado para um valor entre 0 e 1.

O resultado desta conversão para as curvas de dificuldade a pode ser visto na figura 22 :

Figura 22 – Curvas de dificuldade finita e dificuldade cíclica adaptadas



Fonte: O autor

Após a adaptação das curvas, é possível notar que seus comportamentos originais são de certa forma conservados: na curva de dificuldade finita a dificuldade está frequentemente oscilando para dificuldades menores e maiores, mas sempre mantendo um sentido crescente. Já na curva de dificuldade cíclica o crescimento é mais objetivo, com o aumento na dificuldade ocorrendo de forma direta.

É importante frisar que a curva de progresso é gerada em tempo real durante as partidas e é o resultado da combinação da curva de dificuldade com a proporção dos segmentos. Ao se combinar esses dois elementos, a curva do designer seguirá o seu comportamento original, com o diferencial de que a cada sequência de segmentos sempre haverá um balanceamento entre os tipos de segmentos adequados para o momento de jogo em questão de forma a se aproximar do gráfico de flow.

Um dos maiores benefícios dessa abordagem é que o *game designer* pode abstrair a preocupação constante de modelar a curva adicionando intervalos de dificuldade maior e menor, afinal o sistema já se encarrega de encaixar os segmentos dessa forma.

5.7 Escolha do Segmento

A escolha de segmento foi implementada seguindo a técnica especificada no capítulo 4 sem mudanças ou adaptações. Para este estudo, foram utilizados três buckets de segmentos: Fácil, Médio e Difícil.

Além disso, com o propósito de otimizar a utilização de recursos do dispositivo durante a execução do jogo, quando uma lista de segmentos ativos fica vazia, os elementos não são removidos da lista de segmentos inativos correspondentes, mas sim o ponteiro para a lista de segmentos ativos muda para a lista de segmentos inativos,

e o ponteiro da lista de segmentos inativos passa a apontar para a anterior lista de segmentos ativos.

Ao realizar tal operação, evitam-se possíveis travamentos ou momentos de lentidão no jogo por conta de operações que envolvem a instanciação de objetos na memória.

5.8 Resumo

Este capítulo apresentou a implementação das técnicas definidas no processo, inseridas no contexto de um jogo comercial que a princípio havia sido balanceado totalmente de forma manual. Apesar de ter sido utilizado um jogo do tipo corrida infinita para o estudo de caso em questão, a técnica é genérica o suficiente para ser utilizada em diversos outros tipos de jogos de infinitos ciclos.

6 AVALIAÇÃO

Neste capítulo são detalhadas as formas como cada etapa do processo foi analisada, incluindo os resultados obtidos nas experimentações.

6.1 Introdução

Como o processo definido envolve uma grande diversidade de sistemas, cada um com suas próprias particularidades e implementações, foram utilizadas diversas formas de análise com o propósito de validar as técnicas empregadas no estudo de caso. Dessa forma, neste capítulo as metodologias de análise serão explicitadas para cada etapa do processo e os resultados obtidos serão analisados.

6.2 Geração de Segmentos

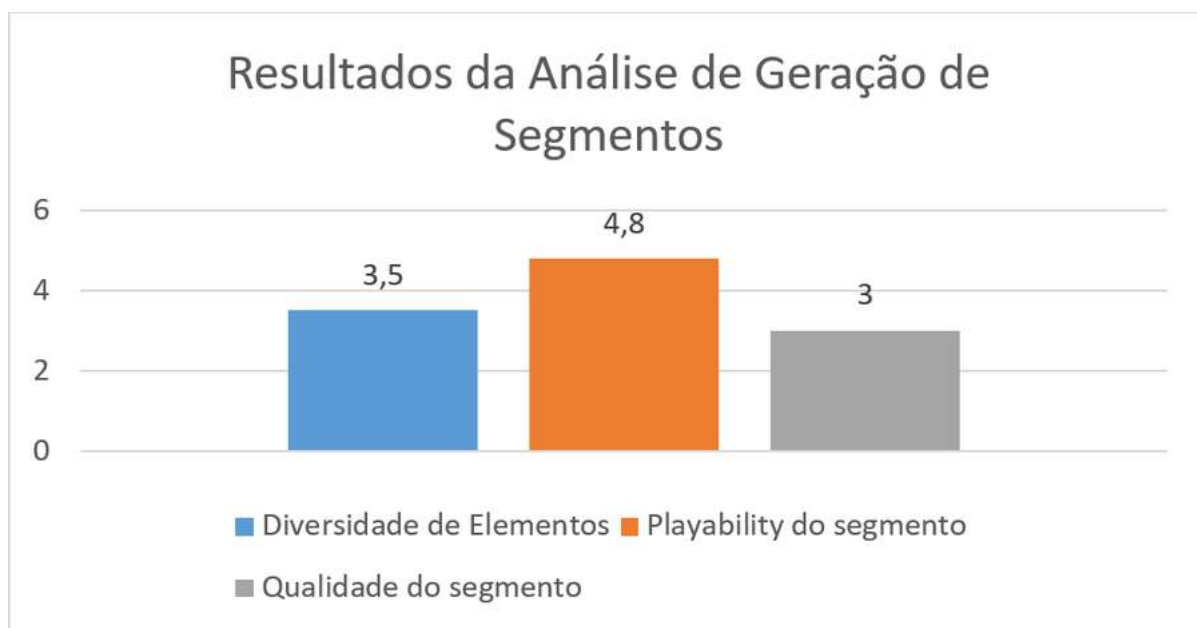
Para a geração de segmentos utilizou-se uma abordagem qualitativa na análise do conteúdo gerado: foram definidas 5 configurações de ritmo e 10 segmentos foram gerados a partir de cada configuração. Todos os elementos estavam habilitados em cada uma das configurações.

Dois *game designers* então analisaram cada um dos 50 segmentos de forma independente e para cada segmento foram apresentadas 3 perguntas na escala likert (LIKERT, 1932). As perguntas apresentadas foram:

- 1) Diversidade de elementos: há uma boa diversidade de elementos no segmento? Um elemento em particular se repete várias vezes? (1 = diversidade baixa, 5 = diversidade alta)
- 2) *Playability* do segmento: se o jogador for habilidoso, é possível passar por este segmento sem perder tempo? O jogador é obrigado a atingir algum obstáculo? (1 = o segmento apresenta vários problemas de *playability*, 5 = o segmento possui *playability* perfeita)
- 3) Qualidade do segmento: de forma geral, qual é a qualidade do segmento? Leve em consideração todos os aspectos. (1 = segmento muito ruim, 5 = segmento muito bom)

Uma vez que os dados foram coletados, eles foram combinados e obteve-se a média de cada um dos itens. Os valores das médias obtidas podem ser vistas na figura abaixo, e a análise de cada uma delas será feita a seguir:

Figura 23 – Gráfico com resultados da análise de geração de segmentos



Fonte: O autor

- 1) Diversidade de elementos: nota 3.5. Observamos que as notas da diversidade de elementos para os primeiros segmentos foram mais elevadas, mas conforme os *game designers* viam mais segmentos, ficava mais perceptível a repetitividade de alguns elementos e padrões.
- 2) *Playability* do segmento: nota 4.8. A grande maioria dos segmentos não apresentou problemas de *playability*. Os problemas identificados dizem respeito a alguns *edge cases* (casos de borda) que não foram identificados a princípio.
- 3) Qualidade do segmento: nota 0. Assim como o primeiro ponto, as notas para este item decaíram conforme os *game designers* avançavam na análise dos segmentos. Isso se deve possivelmente à relativa simplicidade como o sistema dispõe os elementos no cenário: o sistema respeita as condições estabelecidas e as restrições impostas, mas não adiciona mais variações ou experiências diferenciadas que são comuns em segmentos temáticos feitos por designers.

De forma geral, a avaliação do gerador de segmentos foi positiva, mas a simplicidade do sistema acaba por gerar segmentos mais simples e não tão empolgantes, o que pode deixar os jogadores entediados em seções mais longas.

6.3 Classificação de Segmentos

Para realizar a avaliação do classificador, seguiu-se um processo baseado no adotado por Pedersen, Togelius e Yannakakis (2009). Neste processo de avaliação,

comparou-se a capacidade de classificação manual dos *game designers* com a classificação automática do sistema com base em dados de partidas jogadas por participantes voluntários.

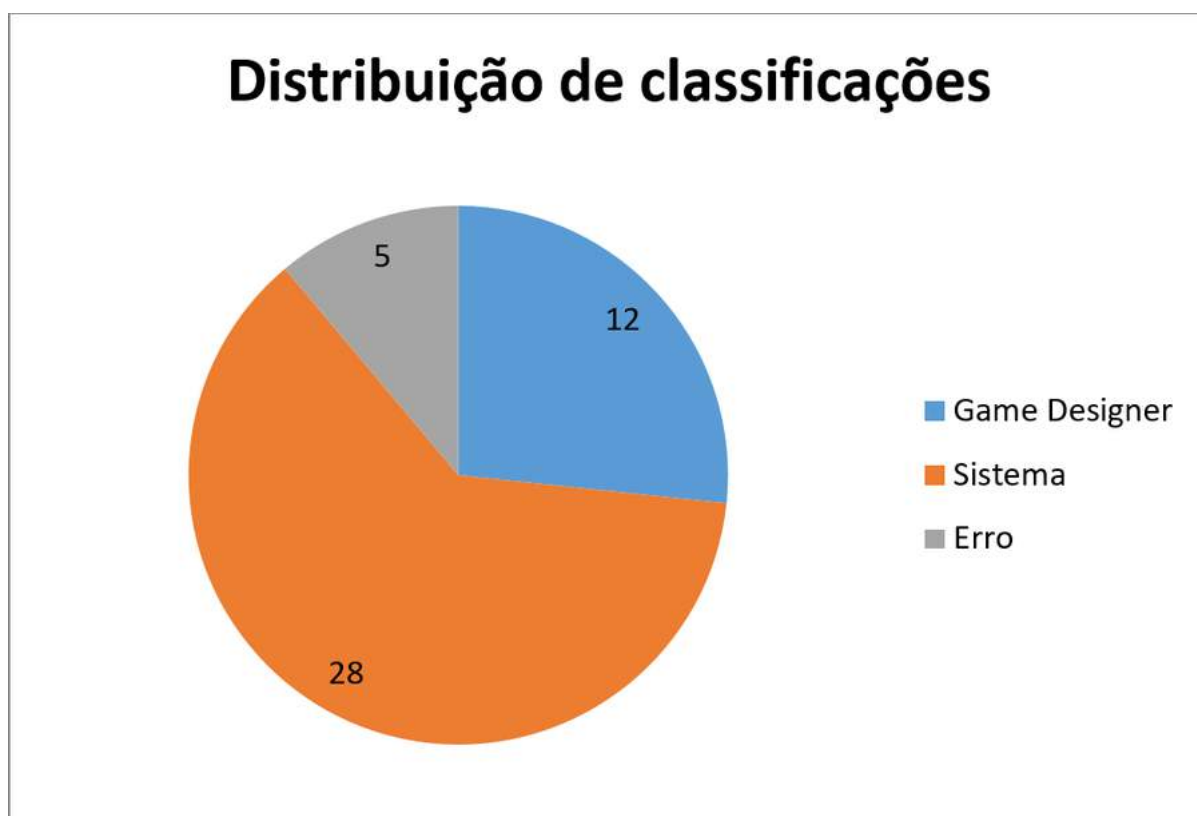
As etapas do processo são descritas a seguir:

- 1) *Game designers* criaram cinco segmentos de cada uma das três dificuldades – fácil, médio e difícil – totalizando 15 segmentos.
- 2) Cada um dos segmentos foi classificado pelo sistema utilizando-se a Rede Neural A, que já havia sido criada na etapa de desenvolvimento do sistema. Mas os segmentos não foram.
- 3) Os segmentos que apresentaram diferenças entre as duas classificações foram colocados em um conjunto denominado de “conjunto divergente”, e os classificados de forma semelhante foram colocados em um conjunto denominado de “conjunto convergente”. Com isso, temos dois grupos de segmentos separados: um de segmentos classificados de forma diferente pelos *game designers* e pelo sistema, contendo 9 segmentos, e outro que contém segmentos classificados da mesma forma pelos dois, contendo 6 segmentos.
- 4) Foi então gerada uma versão especial do jogo na qual cada partida jogada pelo participante é povoada por apenas um segmento que é inserido três vezes consecutivas. Se o voluntário ultrapassar os três segmentos ou alcançar a condição de derrota antes do fim, a partida é finalizada e é exibida uma tela de fim de jogo.
- 5) Com o propósito de comparar a eficiência de classificação do *game designer* e a do sistema, os voluntários devem jogar duas partidas consecutivas. Em uma das partidas, utiliza-se um segmento que foi classificado igualmente pelos dois, em outra são utilizados apenas segmentos nos quais houve discordância entre as duas classificações. A ordem das partidas é definida de forma aleatória, então em alguns casos o voluntário joga primeiro uma partida com um segmento do conjunto divergente, e em outros casos o voluntário joga primeiro uma partida com um segmento do conjunto convergente.
- 6) Ao fim das duas partidas, pede-se ao usuário que este responda a um questionário com uma pergunta e três alternativas. A pergunta apresentada é “Qual das duas partidas você achou mais difícil?” E as três alternativas apresentadas são as seguintes:
 - 1) A primeira partida foi mais difícil do que a segunda.
 - 2) A segunda partida foi mais difícil do que a primeira.

- 3) As duas partidas foram igualmente difíceis.
- 7) Se a resposta do voluntário favorecer a classificação do *game designer*, ou seja, se o voluntário considerou a classificação do *game designer* correta quando comparada à classificação dada pelo sistema, é incrementado um contador de acertos para o *game designer*. Se a resposta do voluntário favorecer a classificação do sistema, é incrementado um contador de acertos para o sistema. Se a resposta do voluntário não favorecer nenhum dos dois, é incrementado um contador de erro.
- 8) Ao todo foram coletados dados referentes a 46 pares de partidas e os contadores apresentaram os seguintes valores:
- *Game Designer*: 12
 - Sistema: 28
 - Erro: 5

Como pode ser visto na figura 24, 62% dos usuários concordaram com a classificação dada pelo sistema, 27% concordaram com a classificação dada pelo *game designer* e 11% discordaram das duas.

Figura 24 – Distribuição dos contadores de acerto de classificação.



Estes resultados demonstram a eficiência de classificação do sistema quando comparado ao método manual baseado na intuição. No entanto, é importante salientar o verdadeiro significado desses resultados: eles não significam que o *game designer* acertou apenas 27% das classificações ou que o sistema acertou 62%, a interpretação correta desses dados é que, em classificações nas quais ocorreram discordâncias entre as duas partes, o sistema foi capaz de prever corretamente 62% dos casos, enquanto o *game designer* foi capaz de prever 27%.

6.4 Curva de Progresso

Para avaliar a curva de progresso, utilizou-se a metodologia de teste A/B (KOHAVI; LONGBOTHAM, 2015). Voluntários foram convidados a jogar uma seção de jogo com duas partidas consecutivas do jogo, em uma é utilizada a curva de dificuldade definida manualmente (manual), e em outra é utilizado o sistema de curva de progresso empregando as curvas de dificuldades adaptadas definidas na seção 5.6.2 (automático).

A seleção de qual partida da seção é jogada primeiro é feita de forma alternada: para o voluntário 1 a primeira partida utilizou o método manual, para o voluntário 2 a primeira partida utilizou o método automático, para o voluntário 3 a primeira partida utilizou o método manual manual, e assim sucessivamente. Em ambas as partidas foram utilizados segmentos que já haviam sido feitos manualmente pelos *game designers*, mas classificados pelo sistema, e para a seleção dos segmentos foi utilizada a técnica especificada na seção 5.7. Optou-se por utilizar os segmentos feitos de forma manual pelos *game designers* por esses apresentarem estruturas mais diversificadas.

Ao fim das duas partidas, o voluntário deve responder a um questionário no qual todas as respostas são dispostas na escala likert (LIKERT, 1932). As perguntas apresentadas no questionário foram:

- 1) Do ponto de vista de diversão, como você compararia a primeira partida em relação à segunda? (1 = a primeira partida foi muito menos divertida do que a segunda, 5 = a primeira partida foi muito mais divertida do que a segunda)
- 2) Qual é o seu nível de experiência com jogos do tipo endless runner? (1 = pouco experiente, 5 = muito experiente)
- 3) Com que frequência você joga qualquer tipo de jogo semanalmente? (1 = pouca frequência, 5 = frequentemente)

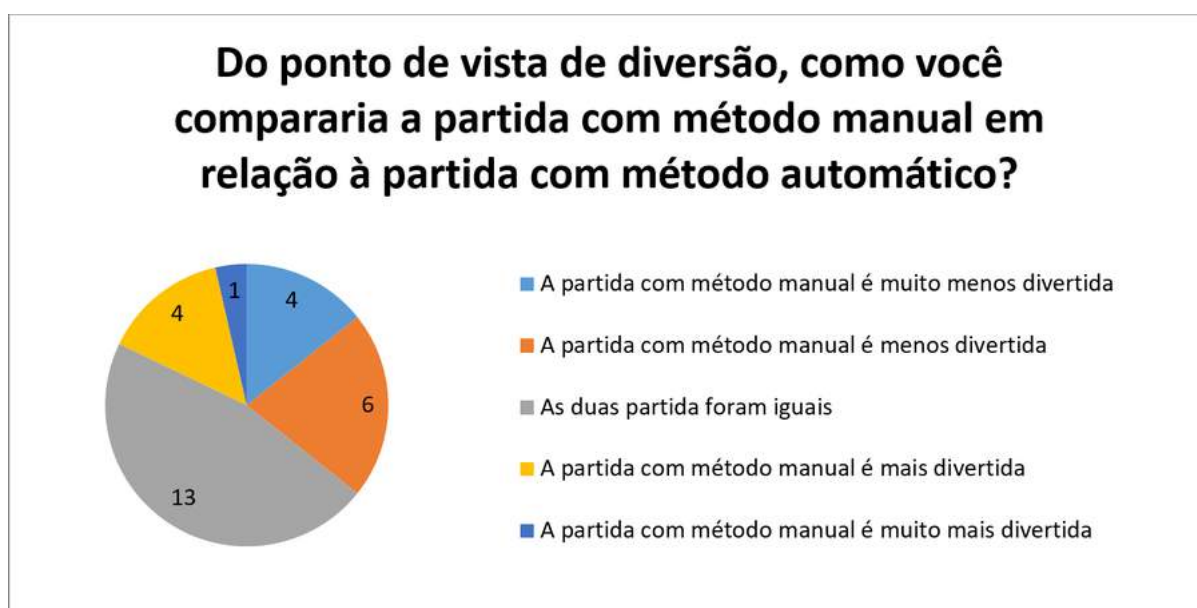
Como a curva utilizada na primeira partida de cada seção foi escolhida de forma alternada, os resultados da pergunta 1 precisam ser ajustados para que as respostas

apontem para o tipo de partida (manual ou automático) ao invés de apontar para a ordem da partida (primeira partida ou segunda partida). Para fazer isso, as respostas das seções de número par foram substituídos pelo valor oposto na escala likert. Por exemplo, se um voluntário da sexta seção respondeu que “a primeira partida foi mais divertida do que a segunda” (4), a resposta foi modificada para “a primeira partida foi menos divertida do que a segunda” (2). Dessa forma é como se em todas as seções, a primeira partida sempre utilizasse o método manual.

Com essa adaptação, a pergunta passou a ser “Do ponto de vista de diversão, como você compararia a partida com método manual em relação à partida com método automático? (1 = a partida com método manual foi muito menos divertida do que a partida com método automático, 5 = a partida com método manual foi muito mais divertida do que a partida com método automático)”.

Ao todo foram coletados dados referentes a 28 seções (56 partidas) e a divisão das respostas à versão adaptada da primeira pergunta pode ser vista na figura abaixo:

Figura 25 – Distribuição das respostas da primeira pergunta



Fonte: O autor

A princípio o resultado não aparenta ser muito positivo, dado que apenas 35% dos jogadores achou o método automático melhor do que o manual. No entanto, observamos que uma quantidade considerável de jogadores não foi capaz de perceber as diferenças entre as duas partidas. Se o método automático se equivale ao manual, o primeiro já é um sucesso. O método automático foi considerado melhor ou igual ao manual por 82% das avaliações.

Ao analisar as respostas das demais perguntas, observamos que participantes com mais experiência em endless runners preferiram o método automático: das 12

respostas dadas por participantes que possuem experiência considerável em endless runners (4 ou 5 na escala), 66% preferiram o método automático em relação ao manual.

A frequência semanal de jogo também aparentou afetar de forma levemente positiva a percepção do método manual em relação ao automático: 54% dos jogadores que jogam com frequência semanalmente (4 ou 5 na escala) preferiram o método automático em relação ao manual. No entanto, observamos uma sobreposição relevante entre frequência de jogo semanal e experiência com endless runners: 63% dos jogadores que jogam com frequência semanalmente possuem uma elevada experiência com jogos do tipo *endless runner*.

Com estas informações, podemos fazer algumas inferências:

- 1) O sistema de curva de progresso automático apresenta resultados similares à curva de dificuldade convencional manual para boa parte dos jogadores. Isto é totalmente positivo, dado que, comparado à curva de dificuldade tradicional, ele remove do *game designer* a incumbência de ter que configurar manualmente com incrementos e decrementos a dificuldade do jogo. Então conseguimos alcançar resultados melhores ao método tradicional com um sistema que elimina algumas preocupações do *game designer*.
- 2) Jogadores com mais experiência em jogos do tipo *endless runner* tiveram mais facilidade em perceber as diferenças entre as partidas e preferiram o método automático. Isso é um indicador de que o sistema cria uma experiência melhor, mas mais refinada, que só é percebida por jogadores que já sabem o que esperar de jogos do tipo *endless runner*.
- 3) Por fim, participantes que jogam com frequência semanalmente também apresentaram uma tendência maior a preferir o método automático do que o manual. O nível de preferência, no entanto, não é o suficiente para chegarmos a uma conclusão mais concreta.

De forma geral, a curva de progresso apresentou resultados positivos, mas possivelmente se o experimento for executado com um grupo maior de participantes teremos resultados mais conclusivos.

6.5 Escolha dos Segmentos

O sistema de escolha dos segmentos apresenta uma análise muito difícil por causar mudanças muito sutis em situações nas quais existe uma quantidade muito grande de segmentos disponíveis, e muito óbvias quando existem poucos segmentos.

No entanto, como este sistema esteve ativo durante a avaliação da curva de progresso, consideramos que a avaliação desta também é refletida pelo sistema de escolha de segmentos, mesmo que com um impacto menor.

6.6 Resumo

Este capítulo apresentou as avaliações das diversas técnicas utilizadas no processo definido. Mais uma vez, fica clara uma das maiores vantagens do processo: cada etapa da geração procedimental é independente e pode ser analisada e modificada de forma separada, sem afetar os demais sistemas.

De forma geral, consideramos positivos os resultados das avaliações. Mesmo em casos nos quais os resultados não foram tão significativos ou que o processo foi considerado muito similar à técnica de geração manual, ainda há a vantagem do ganho de produtividade e de diversidade ao se utilizar um sistema procedimental.

7 CONCLUSÕES E TRABALHOS FUTUROS

Este capítulo tem como objetivo apresentar algumas considerações finais sobre os principais tópicos abordados nesta dissertação, incluindo as contribuições alcançadas e indicações para trabalhos futuros.

7.1 Considerações Finais

Os objetivos do trabalho de definir um processo direcionado a jogos de infinitos ciclos, e acompanhado por técnicas funcionais para cada etapa do processo, foi alcançado. Além disso, o processo apresenta diversas características desejáveis, como a controlabilidade por um *game designer*, a garantia da *playability* do conteúdo gerado, o aumento no fator replay com a geração de conteúdo, entre outros.

Ao conhecimento do autor deste trabalho, não existem processos formalizados para geração de conteúdo para jogos de ciclos infinitos, e muitas das técnicas existentes apresentam soluções com considerável acoplamento nos sistemas.

7.2 Contribuições

A principal contribuição deste trabalho é a definição do processo em quatro etapas independentes para geração de conteúdo de forma procedimental para jogos de infinitos ciclos, o qual permite aos desenvolvedores de jogos atuar de forma isolada em cada uma. Assim sendo, é possível aplicar diversos tipos diferentes de técnicas em cada etapa do processo de geração de conteúdo. Associado a este processo há ainda o conceito de segmento, que não é algo totalmente novo por já ter sido utilizado como parte de técnicas de geração procedural rítmica, mas até então este elemento não havia sido formalizado enquanto um fator central da geração de conteúdo para jogos desta categoria em particular.

Foi apresentado também um conceito inovador de curva de progresso que utiliza uma abordagem diferente das diversas já existentes na academia. A formalização dessa abordagem abre espaço para diversas análises e possibilidades para a geração de meta-curvas de dificuldade, e certamente merece ser analisada em mais detalhes.

Por fim, uma contribuição de menor impacto é a implementação dos diversos sistemas associados ao processo para o caso particular de jogos endless runner, como é o caso do jogo que foi utilizado como plataforma de teste para este trabalho. A implementação foi apenas uma instância possível dentre as dezenas de alternativas, mas ela mostra como estas técnicas são capazes de interagir de forma harmônica mesmo

estando totalmente desacopladas. E apesar de tais técnicas estarem intimamente relacionadas com o tipo de jogo em questão, elas são expansíveis para diversos outros gêneros de jogos de infinitos ciclos.

7.3 Trabalhos Futuros

Este trabalho apresenta diversas áreas para análise e expansão. Uma análise muito interessante e que certamente interessa o autor é verificar quais resultados podem ser alcançados ao realizar diversas combinações de técnicas de geração procedimental em cada uma das etapas.

Outra adição que certamente beneficiaria o processo é a criação de uma técnica de geração de segmentos verdadeiramente genérica e que se adapte facilmente aos diversos tipos de jogo no mercado. Este certamente é um problema muito complexo, mas se uma solução for encontrada, ela pode expandir tremendamente o potencial do processo.

Além disso, o conceito de curva de progresso é algo pouco explorado e que apresenta grande potencial para melhorias e otimizações. Em particular é interessante a análise de como a utilização desta curva impactaria a monetização de jogos free-to-play, dado que o conceito foi adaptado a partir de uma técnica que visa maximizar o faturamento em jogos desse tipo.

Outro ponto relevante é a possível expansão do processo para jogos que não são de infinitos ciclos. Como este trabalho focou em resolver uma demanda não solucionada para jogos desse tipo, foi necessário manter o foco em jogos desse tipo, mas o autor acredita que o processo pode ser facilmente adaptável para jogos convencionais, o que pode tornar tal processo ainda mais útil.

O processo por sua vez não necessariamente precisa ser mantido em 4 etapas. As análises realizadas neste trabalho resultaram na divisão em quatro etapas, mas é possível que o processo possa ser ainda mais granular, permitindo um grau ainda maior de liberdade e automação, além de desacoplamento dos elementos.

Por fim, estamos estudando como utilizar o processo apresentado neste documento na versão do Boney the Runner que está disponível na loja. A princípio pretendemos fazer um teste A/B no qual uma parcela limitada dos jogadores terá acesso à versão que utiliza o processo e faremos o monitoramento constante das métricas de faturamento e retenção de usuários. Se observarmos resultados positivos, continuaremos incrementando gradualmente a quantidade de jogadores com acesso a esta versão até que, caso as métricas se mantenham positivas, esta se torne a versão padrão do jogo.

Caso tenhamos um resultado positivo no Boney the Runner, o processo passará

a ser adotado por padrão em jogos futuros da BigHut Games que se encaixem na categoria de jogos de infinitos ciclos.

Referências

- ADRIAN, D. F. H.; LUISA, S. G. C. A. An approach to level design using procedural content generation and difficulty curves. In: *2013 IEEE Conference on Computational Intelligence in Games (CIG)*. [S.l.: s.n.], 2013. p. 1 – 8. ISSN 2325-4270.
- ASHLOCK, D. Automatic generation of game elements via evolution. In: *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*. [S.l.: s.n.], 2010. p. 289 – 296. ISSN 2325-4270.
- BIGHUT GAMES. *Boney the Runner*. 2012. Disponível em: <<http://www.bighutgames.com/game.html?id=Boney:TheRunner>>. Acesso em: 25/04/2015.
- BIGHUT GAMES. *Story*. 2012. Disponível em: <<http://www.bighutgames.com/story.html>>. Acesso em: 25/04/2015.
- BLIZZARD ENTERTAINMENT. *Diablo 3*. 2012. Disponível em: <<http://us.battle.net/d3/>>. Acesso em: 20/04/2015.
- CARVALHO, L. V. de et al. A Generic Framework for Procedural Generation of Gameplay Sessions. *Proceedings of SBGames 2013*, p. 203 – 210, 2013.
- COMPTON, K.; MATEAS, M. Procedural Level Design for Platform Games. In: *Proceedings of the Second AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. AAAI Press, 2006. (AIIDE'06), p. 109 – 111. Disponível em: <<http://dl.acm.org/citation.cfm?id=3023108.3023128>>.
- FARBRAUSCH. *kkrieger*. 2004. Disponível em: <<http://www.farbrausch.de>>. Acesso em: 13/04/2015.
- HALFBRICK STUDIOS. *Jetpack Joyride*. 2011. Disponível em: <<https://halfbrick.com/our-games/jetpack-joyride/>>. Acesso em: 20/05/2012.
- HALL, M. et al. The WEKA Data Mining Software: An Update. *SIGKDD Explor. Newsl.*, ACM, New York, NY, USA, v. 11, n. 1, p. 10 – 18, nov 2009. ISSN 1931-0145. Disponível em: <<http://doi.acm.org/10.1145/1656274.1656278>>.
- HENDRIKX, M. et al. Procedural Content Generation for Games: A Survey. *ACM Trans. Multimedia Comput. Commun. Appl.*, ACM, New York, NY, USA, v. 9, n. 1, p. 1:1 – 1:22, feb 2013. ISSN 1551-6857. Disponível em: <<http://doi.acm.org/10.1145/2422956.2422957>>.
- HUNICKE, R.; CHAPMAN, V. AI for dynamic difficulty adjustment in games. v. 2, 01 2004.
- KALOGIROU, C. *Opinion: My game design hat*. 2012. Disponível em: <https://www.gamasutra.com/view/news/129539/Opinion_My_game_design_hat.php>. Acesso em: 20/05/2012.
- KERSSEMAKERS, M. et al. A procedural procedural level generator generator. In: *2012 IEEE Conference on Computational Intelligence and Games (CIG)*. [S.l.: s.n.], 2012. p. 335 – 341. ISSN 2325-4270.

KING. *Candy Crush Saga*. 2012. Disponível em: <https://king.com/pt_BR/game/candycrush>. Acesso em: 20/04/2015.

KOHAVI, R.; LONGBOTHAM, R. Online Controlled Experiments and A/B Testing. In: _____. *Encyclopedia of Machine Learning and Data Mining*. Boston, MA: Springer US, 2015. cap. Online Controlled Experiments and A/B Testing, p. 1 – 8. ISBN 978-1-4899-7502-7.

KOSTER, R. *Theory of Fun for Game Design*. 2nd. ed. [S.l.]: O'Reilly Media, Inc., 2013. ISBN 1449363210, 9781449363215.

KULKARNI, N. et al. Endless Runner using Procedural Content Generation & Real-Time Difficulty Curve Generation. *International Journal of Engineering and Technical Research (IJETR)*, v. 3, n. 5, p. 148 – 151, Maio 2015.

LIKERT, R. *A technique for the measurement of attitudes*. 1932. Tese (Doutorado).

MANIFESTO GAMES. *Expertise*. 2005. Disponível em: <<http://www.manifestogames.com.br/site/expertise.php>>. Acesso em: 25/04/2015.

MORRIS, C. *Analysis: Are Long Development Times Worth The Money?* 2010. Disponível em: <https://www.gamasutra.com/view/news/30339/Analysis_Are_Long_Development_Times_Worth_The_Money.php>. Acesso em: 13/04/2015.

MOURATO, F.; SANTOS, M. P. dos; BIRRA, F. Automatic Level Generation for Platform Videogames Using Genetic Algorithms. In: *Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology*. ACM, 2011. (ACE '11), p. 8:1 – 8:8. Disponível em: <<http://doi.acm.org/10.1145/2071423.2071433>>.

NINTENDO. *Nintendo's Official Home for Mario*. 1985. Disponível em: <<http://mario.nintendo.com>>. Acesso em: 13/04/2015.

NIS AMERICA. *Zettai Hero Project*. 2010. Disponível em: <<http://nisamerica.com/games/zhp/>>. Acesso em: 13/05/2012.

PEDERSEN, C.; TOGELIUS, J.; YANNAKAKIS, G. N. Modeling Player Experience in Super Mario Bros. In: *Proceedings of the 5th International Conference on Computational Intelligence and Games*. IEEE Press, 2009. (CIG'09), p. 132 – 139. Disponível em: <<http://dl.acm.org/citation.cfm?id=1719293.1719323>>.

PRUD'HOMME, C.; FAGES, J.; LORCA, X. *Choco Documentation*. [S.l.], 2016. Disponível em: <<http://www.choco-solver.org>>.

SCHICK, S. *The endless runner: The mobile gaming genre that just won't quit*. 2013. Disponível em: <<http://www.fiercedeveloper.com/story/endless-runner-mobile-gaming-genre-just-wont-quit/2013-07-09>>. Acesso em: 25/04/2015.

SHAKER, N.; YANNAKAKIS, G.; TOGELIUS, J. Towards Automatic Personalized Content Generation for Platform Games. In: *Proceedings of the Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. AAAI Press, 2010. (AIIDE'10), p. 63 – 68. Disponível em: <<http://dl.acm.org/citation.cfm?id=3014666.3014679>>.

- SMITH, G. et al. Rhythm-based Level Generation for 2D Platformers. In: *Proceedings of the 4th International Conference on Foundations of Digital Games*. ACM, 2009. (FDG '09), p. 175 – 182. Disponível em: <<http://doi.acm.org/10.1145/1536513.1536548>>.
- SMITH, G.; WHITEHEAD, J.; MATEAS, M. Tanagra: Reactive Planning and Constraint Solving for Mixed-Initiative Level Design. *IEEE Transactions on Computational Intelligence and AI in Games*, v. 3, n. 3, p. 201 – 215, 2011. ISSN 1943-068X.
- SORENSEN, N.; PASQUIER, P. The Evolution of Fun: Automatic Level Design Through Challenge Modeling. In: *ICCC*. [S.l.: s.n.], 2010.
- SORENSEN, N.; PASQUIER, P. Towards a Generic Framework for Automated Video Game Level Creation. In: *Proceedings of the 2010 International Conference on Applications of Evolutionary Computation - Volume Part I*. Springer-Verlag, 2010. (EvoApplications'10), p. 131 – 140. Disponível em: <http://dx.doi.org/10.1007/978-3-642-12239-2_14>.
- STEINHOFF, F. *Jelly Splash*: Puzzling your way to the top of the App Stores. 2014. Disponível em: <<http://pt.slideshare.net/wooga/jelly-splash-puzzling-your-way-to-the-top-of-the-app-stores-gdc-2014>>. Acesso em: 13/05/2012.
- TAKATSUKI, Y. *Cost headache for game developers*. 2007. Disponível em: <<http://news.bbc.co.uk/2/hi/business/7151961.stm>>. Acesso em: 13/04/2015.
- TOGELIUS, J. et al. What is procedural content generation? Mario on the borderline. In: _____. *2nd International Workshop on Procedural Content Generation in Games, PCGames 2011 - Co-located with the 6th International Conference on the Foundations of Digital Games*. [S.l.: s.n.], 2011. ISBN 9781450308724.
- TOGELIUS, J. et al. Search-Based Procedural Content Generation: A Taxonomy and Survey. *IEEE Transactions on Computational Intelligence and AI in Games*, v. 3, n. 3, p. 172 – 186, 2011. ISSN 1943-068X.
- WICHMAN, G. R. *A Brief History of Rogue*. 1997. Disponível em: <http://www.digital-eel.com/deep/A_Brief_History_of_Rogue.htm>. Acesso em: 13/04/2012.
- WOLF, M. J. P. Encyclopedia of Video Games: The Culture, Technology, and Art of Gaming. In: _____. 1. ed. [S.l.]: Greenwood Publishing Group, 2012. cap. 71, p. 524 – 524.
- WOOGA. *Jelly Splash*. 2013. Disponível em: <<https://www.wooga.com/games/jelly-splash/>>. Acesso em: 20/04/2015.