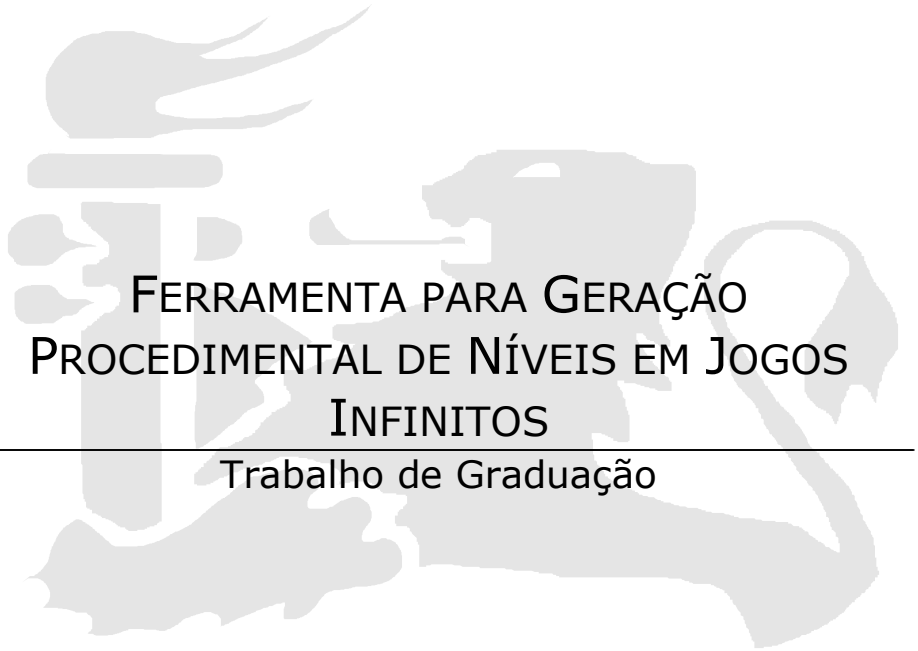
Universidade Federal de Pernambuco
Graduação em Ciência da Computação

Centro de Informática



FERRAMENTA PARA GERAÇÃO PROCEDIMENTAL DE NÍVEIS EM JOGOS INFINITOS

Trabalho de Graduação

Aluno: Alberto Rodrigues Costa Junior (arcj@cin.ufpe.br)
Orientador: Geber Lisboa Ramalho (glr@cin.ufpe.br)

7 de Março de 2014

RESUMO

O conteúdo de um jogo é um fator importante para manter os jogadores engajados, entretanto a geração manual de conteúdo é uma atividade que consome muitos recursos. Diante disto, diversas produtoras de jogos recorrem ao uso de técnicas de geração procedimental de níveis em seus jogos, visando não só a redução de custos, como também a variedade de conteúdo que as técnicas computacionais podem proporcionar. Neste contexto, o objetivo deste trabalho é criar uma aplicação baseada em restrições capaz de gerar os blocos compõem os níveis para jogos infinitos, de forma a tornar o processo mais rápido em relação à geração manual.

AGRADECIMENTOS

Em primeiro lugar agradeço à minha família, que sempre me apoiou. Agradeço ao meu pai e minha mãe, que sempre fizeram o possível para que eu tivesse acesso a uma educação de qualidade.

Agradeço também a todos os amigos que compartilharam comigo os melhores e piores momentos na universidade, mas ninguém sabe melhor do que eles o que foi necessário para chegarmos até aqui.

Agradeço ao aluno de mestrado Leonardo Vieira, que dedicou um pouco do seu tempo para me auxiliar neste projeto.

Agradeço aos professores do Centro de Informática da UFPE, dentre eles gostaria de agradecer especialmente ao professor Geber Ramalho, que me orientou durante o desenvolvimento desse trabalho, e à professora Katia Silva Guimarães Souza, com a qual muito aprendi depois de três períodos fazendo parte da monitoria da disciplina de Algoritmos e Estrutura de Dados, e ao professor Fernando Castor, com o qual também adquiri bastante conhecimento ao fazer parte da monitoria da disciplina Paradigmas de Linguagens Computacionais por mais três períodos.

Sumário

Capítulo 1 – Introdução:

Este capítulo apresenta cinco seções: uma breve análise sobre como a geração procedimental de conteúdo e a geração de níveis tem afetado o desenvolvimento de jogos; as motivações que levaram ao estudo e desenvolvimento deste trabalho; o objetivo que este trabalho almeja realizar; a maneira que se pretende alcançar o objetivo e por fim uma exibição da como este trabalho está estruturado.

1.1. Considerações Iniciais

A indústria de jogos digitais tem convivido com uma demanda cada vez maior por conteúdo em jogos, sendo a quantidade e a diversidade importantes fatores que mantem os jogadores engajados, entretanto a geração manual de conteúdo é uma atividade que consome muitos recursos em termos de tempo e dinheiro. Consequentemente, tem havido um aumento de interesse na produção automática, que é usualmente chamada de geração procedimental de conteúdo ou GPC (HENDRIKX et al., 2011). GPC refere-se à aplicação de algoritmos executados em computador com pouca ou nenhuma contribuição humana, para realizar a geração ou seleção automática de diferentes tipos conteúdo (TOGELIUS et al., 2013).

Técnicas de GPC têm sido usadas com sucesso em diversos tipos de jogos e sua utilização começou no inicio da década de 80, quando os jogos do gênero *Rogue* tornaram-se bastante populares, *Rogue* pode ser descrito como um *role-playing game* gráfico, este estilo trouxe como principal inovação a capacidade de criar um numero ilimitado de níveis sem supervisão humana, embora não tão complexos quanto os gerados manualmente é possível observar que a variedade de níveis proporcionou ao gênero um fator de replay considerável (COMPTON et al. 2006, CARVALHO 2012).

Atualmente diversas produtoras de jogos recorrem ao uso de técnicas de GPC em seus jogos, não só pela redução de custos e variedade de conteúdo que as técnicas computacionais podem trazer, mas também pelo motivo de que as técnicas de geração podem se adaptar as preferências individuais de cada jogador, tornando o conteúdo cada vez mais personalizado ao público alvo (SHAKER et al., 2010).

A aplicação de técnicas de GPC no desenvolvimento de jogos tem se mostrado impressionante no que diz respeito ao tipo de conteúdo gerado, para Carvalho (CARVALHO, 2012), “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”. Em razão de todo potencial da geração procedimental, este trabalho visa apresentar uma maneira de empregar esta técnica à geração de níveis, especialmente em um estilo de jogo que se tornou bastante popular através dos dispositivos móveis, o e*ndless runner* ou corrida infinita, cujo objetivo a toda rodada é alcançar uma distância cada vez maior do ponto de partida e ao mesmo tempo deve-se desviar de obstáculos (CARVALHO, 2012).

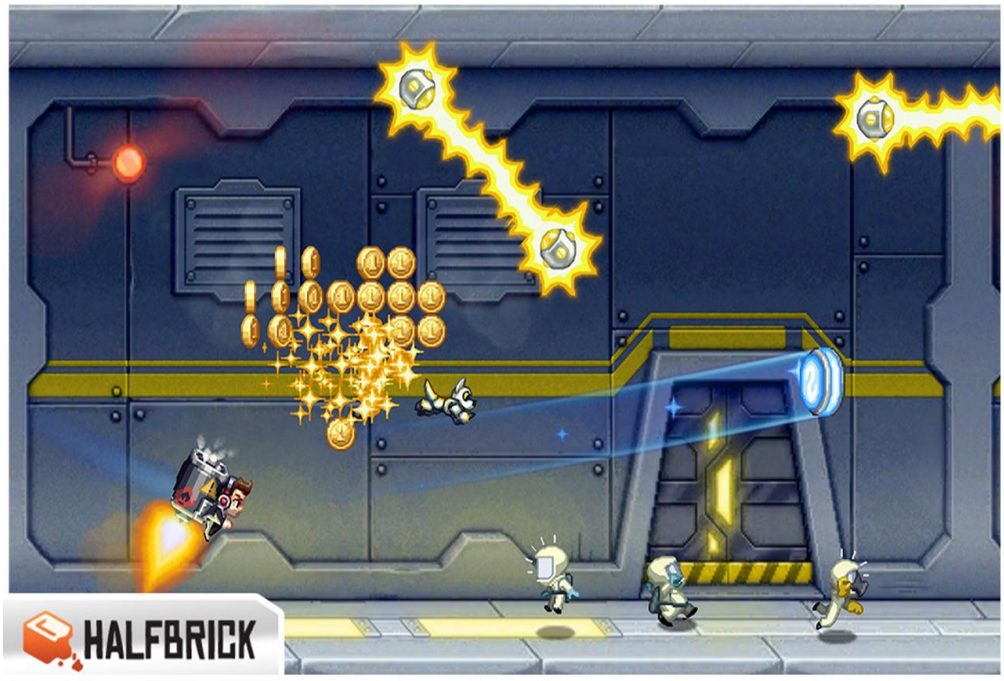


Figura 1.1. Uma partida do jogo Jetpack Joyride (HALFBRICK STUDIOS 2011) de gênero endless runner.

1.2. Motivação

O gênero de corrida infinita está diretamente ligado à geração procedimental de níveis, pois este tipo de jogo precisa de uma quantidade enorme de conteúdo, uma vez que por definição, deve ser capaz de gerar partidas potencialmente infinitas. Como forma de evitar partidas repetitivas com intuito de prolongar o engajamento do jogador por mais tempo, desenvolvedores recorrem ao uso de técnicas de geração procedimental a fim de aumentar a experiência dos jogadores sem comprometer os custos de desenvolvimento do projeto, caso lançassem mão da geração manual.

O uso de técnicas de geração procedimental de níveis se disseminou por décadas em alguns estilos de jogos, como o *Rogue* pela razão prática de que existem poucas restrições rígidas na geração de níveis e o Puzzle no qual o conceito e o entendimento dos níveis são bem delimitados. Por outro lado, os tipos de jogo corrida infinita e *2D Platform* necessitam de mais cuidado, pois até mesmo pequenas mudanças em um nível podem facilmente invalidá-lo (LASKOV, 2010). E por ser tratar de um estilo relativamente novo, poucas soluções podem ser encontradas na literatura em relação à geração de níveis para jogos infinitos, muitas delas inclusive propõem soluções dirigidas a jogos em que o conceito de níveis é bem delimitado, fator que normalmente não é observado em jogos deste tipo (CARVALHO, 2012).

Apesar da dificuldade, temos convicção de que o desenvolvimento de uma ferramenta automatizada para geração de níveis em jogos infinitos é uma tarefa que vale a pena. Com o aumento constante da complexidade gráfica e nível de expectativas de jogadores, projetos têm ficado caros em termos de tempo e dinheiro. Grandes estúdios de jogos investem milhões em design de níveis, enquanto os pequenos e independentes desenvolvedores se esforçam para entregar um jogo com um orçamento razoável. Isso faz com que uma ferramenta automatizada para auxiliar o processo de geração de níveis em jogos de corrida infinita seja uma contribuição valiosa e oportuna.

1.3. Objetivos

Alguns trabalhos existentes apresentam a geração de níveis como uma combinação de blocos independentes, esta combinação pode ser capaz de gerar uma quantidade potencialmente infinita de níveis (JENNINGS-TEATS et al, 2010). Estes blocos podem ser entendidos como segmentos de cenário de tamanho fixo. Desta forma não é somente necessário criar os segmentos, como também a maneira como serão dispostos com objetivo de criar um fluxo continuo para o jogador.

Segundo Carvalho (CARVALHO, 2012), a geração de níveis para jogos de corrida infinita pode ser tratada com um processo de quatro etapas:

1. Decisão de como os blocos ficarão dispostos no jogo – Nesta fase, é definida a forma como os segmentos serão posicionados ao longo da partida.
2. Geração de blocos – Nesta fase deve-se um gerar uma quantidade suficiente de segmentos respeitando um serie de limitações orientadas pelo designer.
3. Classificação de blocos – Os segmentos são avaliados e separados utilizando critérios preestabelecidos de posicionamento decididos na primeira etapa.
4. Escolha do bloco a ser posicionado – Tendo em vista a definição de posicionamento estabelecida na primeira etapa e a classificação dos segmentos da terceira, é preciso selecionar um segmento a ser posicionado entre as opções possíveis.

O foco deste trabalho será na segunda etapa descrita acima, e tem como objetivo complementar a solução proposta por Carvalho (CARVALHO, 2012), no qual seu foco era a classificação de segmentos. A solução que será apresentada neste trabalho será a criação e avaliação de uma ferramenta automatizada capaz de gerar segmentos, levando-se em conta as restrições do jogo.

1.4. Abordagem

Como estudo de caso na realização do desenvolvimento e teste da ferramenta proposta neste trabalho, foi utilizado o jogo *Boney The Runner* da produtora BigHut Games (BIGHUT GAMES 2012).

O processo de geração dos blocos deste jogo é feito de forma manual por um game designer, o qual posiciona os elementos do jogo nos segmentos. Como uma forma de tornar este processo mais rápido e diversificado, será criada uma aplicação baseada em restrição através do uso de uma biblioteca de satisfação de restrições. A aplicação criará segmentos respeitando as restrições que eram seguidas de forma implícita na geração manual, de forma a manter a corretude necessária nos blocos, entretanto será preciso especificar a quantidade e os tipos dos elementos desejados nos segmentos, assim a ferramenta irá gerar soluções de blocos corretas direcionadas as configurações almejadas.

1.5. Estrutura do documento

Além deste capítulo introdutório, este documento possui a seguinte estrutura:

Capítulo 2 – Questões Envolvidas na Geração de Níveis: Neste capítulo, será feita uma análise do que está envolvido, dos problemas a serem enfrentados e o que deve ser analisado na geração de níveis 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 níveis para jogos.

Capítulo 4 – Solução Proposta: Neste capítulo será explicitada a solução utilizada para o problema descrito, assim como detalhes sobre o jogo utilizado para testar tal solução.

Capítulo 5 – Avaliação de Resultados: Este capítulo apresenta uma avaliação dos resultados obtidos.

Capítulo 6 – Conclusão: Este capítulo exibe as conclusões acerca do trabalho apresentado e o que se espera de trabalhos futuros.

Capítulo 2 – Questões Envolvidas na Geração de Níveis

Neste capítulo será feita uma análise sobre assuntos que cercam a geração de procedimental de níveis levando-se em conta características que possam classificar os geradores e também alguns aspectos e desafios relevantes que impactam diretamente na experiência do jogador e no processo de geração níveis.

2.1. Taxonomia dos Geradores

As técnicas de geração de níveis empregam diversas estratégias no processo seu processo de criação e podem ser caracterizadas ao observarmos alguns fatores, Togelius (TOGELIUS et al., 2010) propôs um modelo de taxonomia para os processos de GPC que também podem ser aplicadas a geração de níveis, no qual os seguintes aspectos podem ser reconhecidos:

- **Online ou Offline** – Métodos de geração online consistem na produção do conteúdo durante o ciclo de jogo (ao mesmo tempo em que o jogador esta jogando a partida), o que possibilita uma quantidade de variações de geração infinita e conteúdo adaptado para cada jogador. Por outro lado, os métodos offline são aquele que são utilizados durante o desenvolvimento do jogo ou até mesmo antes do inicio das partidas, o uso de técnicas deste tipo é especialmente útil quando a complexidade do que precisa ser gerado é alta.
- **Semente aleatória ou Vetor de parâmetros** – As duas opções fazem referência o nível de controle que teremos sobre o espaço de geração em relação aos elementos controláveis do jogo, podendo ser escolhidos aleatoriamente ou retirados de conjunto parâmetros.
- **Genérico ou Adaptativo** – Conteúdo genérico é aquele que foi gerado sem levar em conta o comportamento do jogador, em contrapartida a geração adaptativa gera o conteúdo baseado na análise das interações anteriores do jogador com o jogo.
- **Estocástico ou Determinístico** – Na geração determinística é possível reviver um conteúdo (ter os mesmos resultados) dado os mesmos parâmetros de entrada, porem num processo estocástico raramente isso é possível.

- **Construtiva ou Gerar-e-testar** – Na geração construtiva o conteúdo que já foi gerado não pode ser modificado. Na estratégia gerar-e-testar uma solução pode ser rejeitada ou modificada dada necessidade.
- **Geração automática ou Autoria mista** – Classificada de acordo com o grau intervenção humana que pode ser antes, durante ou depois do processo.

É importante informar que técnicas de geração de níveis não estão necessariamente nos extremos dos fatores citados acima, na realidade a maioria das técnicas usam estratégias híbridas.

2.2. Desafios e Propriedades Desejadas

Existem inúmeros desafios na geração procedimental de níveis por se tratar de uma atividade que esta diretamente ligada à experiência do jogador, por este motivo, são apontados alguns requisitos importantes para o processo. Entretanto, as propriedades desejáveis ou necessárias de uma solução são distintas para cada aplicação. Nesta seção serão mostradas as propriedades desejáveis mais comuns em soluções para geração de níveis.

2.2.1. Corretude e Confiabilidade

Um bom gerador de níveis precisa satisfazer as restrições e critérios de qualidade esperados. Contudo isto pode ser mais importante para alguns tipos de elementos do cenário do que para outros, por exemplo, em um jogo de corrida infinita, um obstáculo instransponível é uma falha grave de corretude, enquanto que uma flor de background que pareça estranha ou fora do lugar, sem atrapalhar o andamento do jogo caracteriza-se como uma falha leve de corretude (MAWHORTER e MATIAS, 2010).

2.2.2. Diversidade e Expressividade

Muitas vezes é necessário e interessante que o processo de geração de níveis tenha a capacidade de gerar um conteúdo diversificado, pois é a solução mais óbvia e a um baixo custo de desenvolvimento, que pode manter os jogadores interessados no jogo por mais tempo (SHAKER et al, 2010). Diante do que já foi falado no capitulo anterior, o estilo corrida

infinita demanda uma quantidade enorme conteúdo, por isso também é de fundamental importância que este conteúdo seja diversificado a fim de garantir um bom fator de replay. Fator de replay pode ser considerado como o potencial que um jogo tem de encorajar os jogadores a jogar repetidas vezes ainda mantendo o entretenimento deste jogador (CARVALHO, 2012).

A expressividade é uma propriedade que esta associada à diversidade e para facilitar seu entendimento é preciso mostrar dois exemplos extremos (TOGELIUS et al, 2014).

- Primeiro caso (Nenhuma expressividade) – As variações de um conteúdo pra outro são mínimas, por exemplo, em um jogo estilo corrida infinita os obstáculos de todos os segmentos sempre aparecem na mesma posição, mas é trocado de forma aleatória a cor deste obstáculo.
- Segundo caso (Expressividade muito alta) – Considere um gerador que organiza os elementos do jogo de forma totalmente aleatória, gerando níveis completamente não jogáveis e sem sentido.

Perante estes fatos é possível concluir que desenvolver um gerador de níveis que tenha capacidade de criar conteúdo diverso sem comprometer a qualidade do jogo é uma tarefa nada trivial.

2.2.3. Criatividade e estética

Níveis gerados por um computador correm o risco de parecer genéricos, no caso de níveis gerados manualmente, o game designer tem uma ideia ou um sentimento quando ele posiciona um elemento de jogo em algum lugar. Como exemplos disto temos as obras de arte como estátuas, pinturas e músicas, para um humano elas tem algum significado, mas para computador não (SILVA 2010, VENDRIG 2013).

É de senso comum que níveis gerados procedimentalmente produzam um conteúdo de menor qualidade do que os gerados manualmente, pois não é uma tarefa simples desenvolver um gerador de níveis que deve satisfazer restrições complexas de uma forma esteticamente agradável, coerente, original, criativa e proposital que um humano pode aplicar (LASKOV, 2010).

2.2.4. Balanceamento

O balanceamento de dificuldade é uma propriedade muito importante na geração de níveis, e seu processo consiste em modificar parâmetros, cenários, ou comportamentos com o objetivo de garantir um nível adequado de desafio ao usuário dependendo da habilidade que este dispõe (KOSTER, 2005).

Muitos trabalhos sobre balanceamento de jogos se baseiam na proposta de Csikszentmihalyi (CSIKSZENTMIHALYI, 1991) sobre o estado mental de *flow*, no qual a pessoa está completamente imersa, focada e determinada em concretizar os objetivos da tarefa que está a realizar.

Literatura de jogos	Flow
O jogo	Uma tarefa que tem ser completada.
Concentração	Habilidade de concentração em tarefas.
Desafio e Habilidades de jogador	Habilidades devem corresponder aos desafios.
Controle	Exercer um sendo de controle sobre suas ações.
Metas apropriadas	A tarefa tem metas apropriadas.
Retorno (*Feedback*)	A tarefa tem que prover retorno imediato.
Imersão	Envolvimento profundo, mas sem esforço, redução da sensação do tempo.
Interação social	Não se aplica.

Tabela 2.1 O mapeamento dos elementos da literatura de jogos para os elementos do FLOW (SWEETSER e WYETH, 2005, adaptado pelo autor).

Ainda no que diz respeito à dificuldade, existe a necessidade de manter um equilíbrio entre o grau do desafio e a habilidade do jogador (estado de *Flow*), de forma a elevar a dificuldade na medida em que o jogador progride na partida e aumenta sua capacidade. O objetivo deste balanceamento é evitar que o jogo se torne muito difícil deixando o jogador ansioso e frustrado, enquanto que se o jogo se revelar excessivamente fácil o jogador ficará entediado e sem vontade de continuar (CARVALHO, 2012).

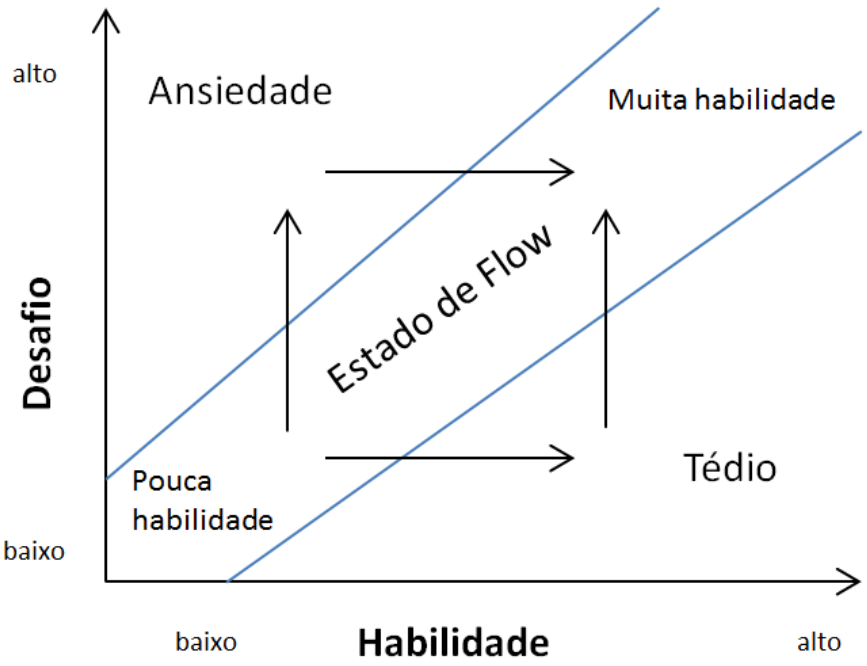


Figura 2.1 Diagrama do caminho do FLOW (CSIKSZENTMIHALYI, 1991, adaptado pelo autor).

No capítulo quatro será vista a forma que o jogo utilizado neste trabalho usa para definir que tipo de bloco será posicionado, portanto é importante que a dificuldade de cada segmento seja medida corretamente de forma que a experiência desejada seja transmitida ao jogador.

2.2.5. Rapidez

Requisitos de velocidade podem variar bastante, dependo principalmente se a geração de conteúdo é feita durante a execução do jogo ou durante seu desenvolvimento. Se a geração e feita durante a execução do jogo, haverá um maior exigência em relação a esta propriedade, pois o jogador pode não estar disposto a esperar longos períodos de carregamento (SILVA, 2010). Por outro lado, no caso da geração ser feita durante o desenvolvimento há uma maior tolerância quanto a esta propriedade, principalmente nos casos em que o conteúdo a ser gerado é de alta complexidade e necessita de um maior processamento.

Capítulo 3 – Estado da Arte

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

A abordagem de Compton e Mateas (2006) é baseada na construção de padrões baseados em repetição e ritmo. Estes padrões representam uma sequência de movimentos de um jogador, como saltos (avaliados em distância e tempo). Ao criar um padrão, o sistema gera uma lista de possíveis componentes de jogo (plataformas, espinhos e outros) e define a dificuldade pretendida neste padrão. Então, o sistema passa a tentar construir um padrão ideal, que é alcançado utilizando o algoritmo de *Hill-climbing* para encontrar a dificuldade almejada. Um padrão é representado por uma célula e o nível total é representado por um conjunto de células. Eles usam esta abordagem porque uma única célula só pode criar níveis lineares e ao fazer uso de várias células, são capazes de criar níveis não lineares.

Este abordagem, apesar de bastante robusta, mostra alguns problemas para a solução que estamos buscando, pois ela exige a definição da estrutura de células de antemão, algo que não podemos ter em jogos de corrida infinita.

Smith et al. (2009) propõem algo similar à solução anterior, porem primeiro são gerados os ritmos e em seguida cria-se a geometria (estrutura) desses ritmos. Os ritmos consistem em ações do jogador, tal como mover ou saltar. Para adicionar variedade a estes padrões ainda são avaliados outras três métricas, o tipo do movimento, o comprimento e a densidade. O tipo de movimento determina como as ações são organizadas durante um ritmo. O comprimento determina a duração de um ritmo. E a densidade representa o espaçamento entre dois movimentos. Como a criação de geometria é limitada pelo modelo de físico dos movimentos, é garantido que todas as geometrias criadas são jogáveis. Um nível é criado agrupando várias geometrias, gerando algo chamado "nível base", só então componentes extras como moedas são adicionados.

Um dos benefícios dessa abordagem é que passa a ser possível gerar uma enorme quantidade de blocos para o mesmo padrão de ritmos, entretanto não existe a possibilidade de um único nível que possa ter uma extensão infinita. Outro fato importante é que esta solução é

focada na diversidade que a geração de níveis pode trazer, mas sem levar em conta a dificuldade, o que pode afetar negativamente a experiência dos jogadores.

Sorenson e Pasquier (2010) usam um algoritmo genético de otimização de restrições (*Feasible-Infeasible Two-Population Genetic Algorithm*) para maximizar os comprimentos dos caminhos de início ao fim em mapas *2D* e para conseguir o melhor arranjo possível entre os blocos que compõem o nível. A função de fitness deste algoritmo também é capaz de identificar o quão divertido é o nível gerado, e para isso leva em consideração a dificuldade percebida pelo jogador ao longo da partida.

Segundo os resultados apresentados o framework apresentado por Sorenson e Pasquier consegue gerar níveis variados respeitando os parâmetros de restrição e dificuldade dados. Contudo todo processo é feito de forma offline sem garantias de que possa ser feito de forma online visto que para criar alguns níveis foi preciso algo entorno de trinta minutos.

Pedersen et al. (2009) criou uma versão de Super Mario Bros a partir do qual gerou um modelo estatístico sobre estado emocional dos jogadores ao se depararem com os desafios do jogo. Para realizar a esta experiência, foi necessário convidar diversos jogadores para jogar um conjunto de partidas, durante as quais foram coletados dados sobre como se saiam no jogo. Ao fim das partidas, foi aplicado um questionário sobre a experiência do jogador naquele jogo, tais dados foram utilizados em uma rede neural responsável por fazer a modelagem da experiência. A partir de um algoritmo evolucionário foi possível combinar blocos para criar um nível de acordo com a emoção que se queria provocar no jogador.

O foco deste trabalho esta na modelagem da experiência do jogador, porem não tem muita clareza de como os níveis são gerados. A ferramenta foca na geração de partidas finitas, mas seu conceito de modelagem na experiência é algo muito útil e que pode ser diretamente aplicado na geração de partidas infinitas.

Existem diversas abordagens para geração procedimental de níveis, entre as quais podemos citar como mais comuns os métodos baseados em busca e os algoritmos evolutivos. Entretanto outras abordagens menos usuais vêm crescendo, como os métodos baseados em agentes, nesta abordagem um grande número de agentes é criado para gerar conteúdo de forma paralela podendo haver ou não interação entre estes agentes. Um bom exemplo dessa abordagem é o gerador de Terrenos 3D de Doran e Parberry (2010), no qual agentes de

software são separados em grupos (chamados de *Waves*) e cada grupo é responsável por criar certo tipo de elemento no terreno.

Um dos benefícios desta abordagem é a sua velocidade de geração devido ao paralelismo aplicado no processo, alguns teste mostraram que solução desenvolvida por Doran e Parberry para alguns casos demorou algumas frações de segundo, enquanto outras soluções demoraram cerca de dois minutos.

Capítulo 4 – Solução Proposta e Estudo de Caso

Conforme explicito no capítulo introdutório, este trabalho tem como objetivo apresentar uma ferramenta capaz de gerar de forma automática os blocos que compõem uma partida em jogos infinitos, de modo a complementar a solução proposta por Carvalho (CARVALHO, 2012), no qual a etapa de geração de blocos ainda é feita de maneira manual por um *game designer*.

Este capítulo foca em especificar a solução proposta para a geração de partidas de jogos infinitos no contexto do jogo que foi utilizado para realizar o estudo, dessa forma faremos também uma breve introdução do funcionamento do jogo e das mecânicas envolvidas.

4.1. Problema de satisfação de restrições

Por se tratar de um problema que estabelece propriedades estruturais aos elementos de jogo que compõem um bloco, a etapa de criação de blocos foi caracterizada por nós como um problema de satisfação de restrições (PSR). Formalmente um PSR pode ser definido como uma tripla (Z, D, C), no qual Z é um conjunto finito de variáveis, D uma função que mapeia cada variável em Z para o domínio de valores (de qualquer tipo) que esta pode assumir e C é um conjunto de relações sobre um subconjunto de Z que restringem os valores que as variáveis em Z podem assumir simultaneamente. Um PSR é dito satisfatível quando existe pelo menos um conjunto de pares, na qual cada variável em Z forma um par com um valor de seu domínio, entretanto este conjunto de pares deve satisfazer todas as relações descritas em C (Rossi et al., 2006).

Diante disto, a solução proposta neste trabalho passa por descrever a criação de um bloco como um PSR, no qual as variáveis deste problema são os posicionamentos, tamanhos e tipos dos elementos que compõem o bloco, os domínios e restrições são delimitados por um game designer de maneira que seja mantido o padrão de corretude dos blocos gerados manualmente, que implicitamente eram posicionados de modo a satisfazer as restrições desejadas. Também é importante adicionar restrições baseadas no modelo físico de movimentação jogo, para que as soluções geradas sejam jogáveis.

E como maneira de representar e solucionar computacionalmente este problema, decidimos utilizar uma ferramenta que provê um ambiente em que é possível modelar de forma declarativa as restrições sobre as soluções viáveis e que também forneça estratégias eficientes para resolver problemas de busca combinatorial.

Solucionadores de restrição fazem a busca no espaço soluções de maneira sistemática, geralmente utilizando *backtracking*, *depth-first-Search* (busca em profundidade) ou *brach and bound*, alguns usam formas de busca locais o que pode não trazer todas as soluções possíveis.

O *branch and bound* foi o algoritmo de busca escolhido para esta solução, pelo motivo de ser o mais eficiente entre os oferecidos pela biblioteca. A estratégia que ele usa é a de dividir para conquistar no qual o problema é particionado em subproblemas menores, entretanto o que o faz ser mais atraente é a estratégia de *bound* (poda), de modo que o algoritmo para de percorrer um ramo da arvore de busca que não considerarado promissor. A estratégia de ramificação utilizada foi de primeiro explorar as variáveis de menor domínio e de avaliar primeiro os menores valores de domínio de cada variável.

Também é importante descrever o quão distante é desejado que uma solução deva estar da solução anterior, sendo um forma prática e eficiente de definir um limite mínimo de expressividade para o gerador, de maneira a não gerar soluções muito próximas.

4.2. Estudo de Caso

Nesta seção será apresentado o contexto em que a nossa solução foi desenvolvida e avaliada para o problema geração procedimental de níveis em jogos infinitos.

4.2.1. Jogo Utilizado no Estudo

Como estudo de caso na realização do desenvolvimento e teste da ferramenta proposta neste trabalho, foi utilizado o jogo Boney The Runner da produtora BigHut Games (BIGHUT GAMES 2012). No momento da escrita desse documento, o jogo já se encontra disponível publicamente, mas as definições aqui apresentadas dizem respeito à versão do jogo que foi utilizada para os propósitos deste trabalho e não refletem a versão atual do jogo.

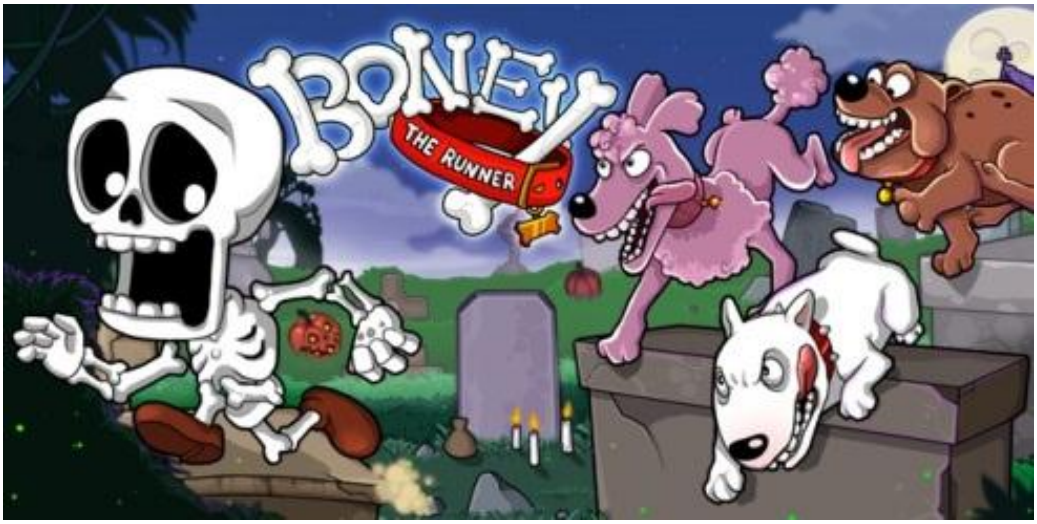


Figura 4.1. Imagem de divulgação do jogo Boney the Runner (BIGHUT GAMES 2012).

O jogo segue o estilo de corrida infinita no qual o jogador controla o Boney, uma caveira que acordou em um cemitério e que esta sendo perseguida por uma matilha de cães famintos. Tanto o Boney quanto os cães sempre estão correndo no sentido da esquerda pra direita, porem a velocidade normal do Boney é maior que a dos cães, deste modo o personagem esta sempre se afastando da matilha enquanto corre.

Durante o caminho outros elementos de jogo são apresentados ao jogador como tumbas, lamas, musgos, fantasmas e moedas.

As tumbas são obstáculos que devem ser evitados pelo jogador, através do único movimento que este pode executar, o salto. Se o jogador não conseguir evitar este obstáculo, isto acontece quando o jogador deixa o avatar (Boney) colidir lateralmente com tumba, como consequência o avatar deverá escalar o obstáculo, o que o manterá parado em uma mesma posição por algum tempo. Como mostra a seguir a figura 4.2.

Figura 4.2. Uma partida do jogo Boney The Runner, avatar escalando a tumba (BIGHUT GAMES 2012).

As lamas são trechos no mesmo nível do solo, no qual é possível visualizar mãos saindo deste elemento. Se o avatar entrar em contato com a lama, o mesmo passa correr a uma velocidade abaixo do normal até que consiga se desvencilhar deste elemento. Como mostra a seguir a figura 4.3.

Figura 4.3. Uma partida do jogo Boney The Runner, avatar mais lento por causa da lama (BIGHUT GAMES 2012).

Os musgos são trechos no mesmo nível do solo, que tem ação oposta à da lama, quando o avatar entra em contato com este elemento, ele desliza, fazendo com que sua velocidade fique acima do normal até que este perca contato com o solo. Como mostra a seguir figura 4.4.

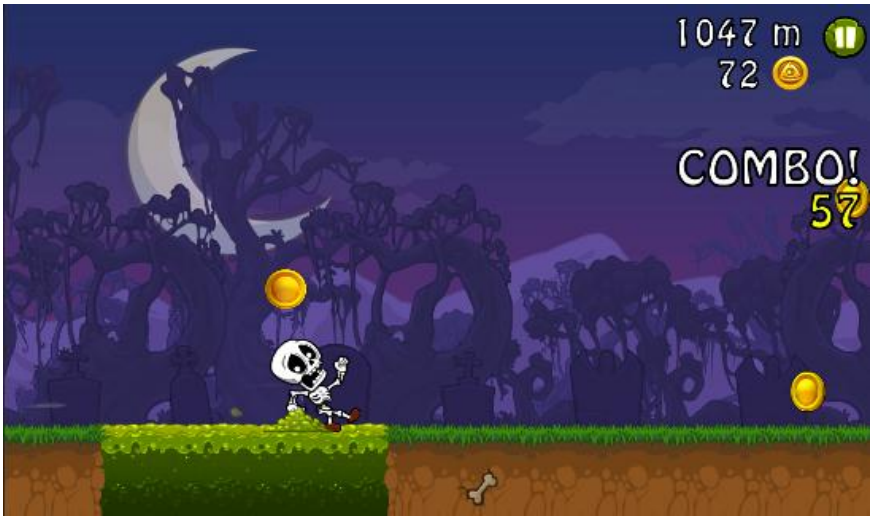


Figura 4.4. Uma partida do jogo Boney The Runner, avatar deslizando no musgo (BIGHUT GAMES 2012).

Os fantasmas são elementos que podem ser encontrados em posições fixas e espalhados pelo cenário. Fantasmas deixam o avatar mais lento por alguns segundos, caso o jogador não consiga evita-los. Assim como as todos os elementos, os fantasmas podem ser evitados se o jogador pular corretamente, isto é no tempo e altura certos. Como mostra a figura 4.5.

Figura 4.5. Uma partida do jogo Boney The Runner, avatar mais lento por causa de um fantasma (BIGHUT GAMES 2012).

As moedas são elementos coletáveis que podem ser encontrados e obtidos ao longo do caminho. Estas moedas podem ser usadas posteriormente para destravar habilidades que tornam o jogo mais fácil.

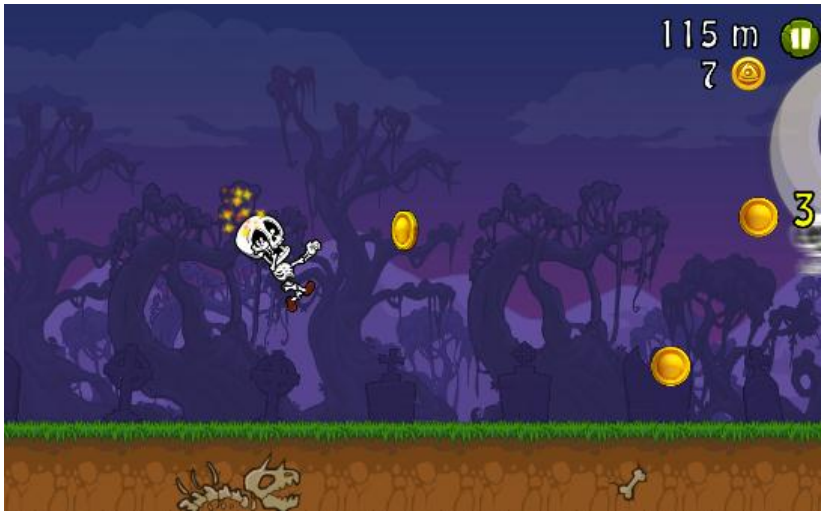


Figura 4.6. Uma partida do jogo Boney The Runner, avatar coletando moedas (BIGHUT GAMES 2012).

Quando jogador comete erros ao tentar desviar de tumbas, fantasmas ou lamas, os cães se aproximam um pouco mais. Se os cães conseguirem alcançar o avatar, a partida é finalizada e a pontuação do jogador é calculada.

Estes são todos os elementos que compõem a versão do jogo utilizada para o propósito deste trabalho. A seguir será explicada a solução proposta para o problema apresentado anteriormente.

4.2.2. Solução Proposta

A solução proposta deste trabalho trata o problema geração de níveis em jogos infinitos como um processo de quatro etapas: Definição da curva de dificuldade, criação dos blocos, classificação dos blocos, e por fim a inserção dos blocos no jogo.

Conforme já explicitado em outras seções, o foco deste trabalho esta na etapa de criação de blocos, entretanto será feita uma descrição das outras etapas mostradas por Carvalho (CARVALHO, 2012).

A forma como os blocos são posicionados ao longo da partida é baseada no desafio que o bloco proporciona ao jogador, no qual duas curvas de dificuldade determinam a

sequencia de tipos de bloco em função do tempo. A primeira curva é usada nos primeiros minutos do jogo, em seguida a segunda curva é usada de maneira circular, ou seja, quando o jogador chega ao final da curva, a curva é reiniciada. A seguir a figura 4.7 mostra um exemplo de curvas.

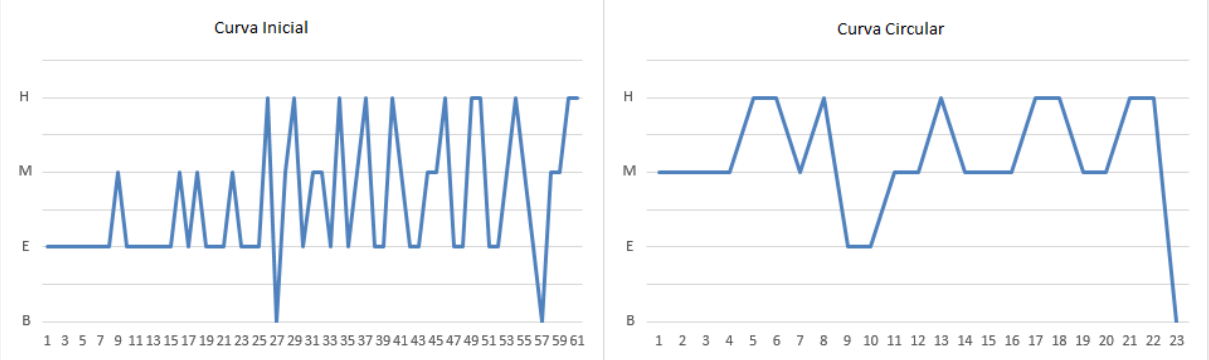


Figura 4.7. Curvas de dificuldade (CARVALHO et al., 2013, adaptado pelo autor).

Em ordem decrescente de dificuldade os blocos podem ser do tipo difícil(H), médio(M), fácil (E) e bônus(B).

A criação dos blocos, mesmo de forma automatizada, ainda necessita que seja especificada uma configuração desejada para os blocos, esta configuração pode ser entendida como uma descrição da quantidade máxima de peças que se espera ter para cada tipo de elemento de jogo, o uso desta configuração é essencial, pois ela impõe um limite máximo de expressividade para o gerador.

Ainda no que diz respeito à estratégia de geração, decidimos que o posicionamento dos elementos no bloco seria feito tipo-a-tipo, ou seja, primeiro todos os elementos de um tipo são posicionados, depois os de outro tipo e assim sucessivamente. Esta abordagem facilita a adição de novos elementos, algo que não seria tão prático se todos os tipos elementos fossem posicionados ao mesmo tempo.

A seguir será exibida a modelagem do problema como um PSR para cada tipo de elemento de jogo, de modo que cada um recebe como entrada a quantidade de elementos desejados para aquele tipo e um vetor com os elementos gerados anteriormente. Primeiro é preciso citar algumas limitações pré-determinadas:

- Todos os blocos tem o mesmo tamanho, as dimensões do bloco serão representadas pelas constantes `larguraBloco` e `alturaBloco`.
- Todos os tipos de elemento do jogo ocupam áreas retangulares no bloco, sendo representados pelas seguintes variáveis `inicioX`, `fimX`, `inicioY`, `fimY`, `largura` e `altura`. E já são iniciadas com as seguintes restrições:
  - `inicioX + largura = fimX`
  - `inicioY + altura = fimY`
- Também é necessário especificar posição no eixo Y que o avatar corre, visto que alguns elementos não podem estar abaixo do solo, esta posição será representada pela constante `solo`.
- `intersecciona` uma função que verifica se existe intersecção entre dois elementos, ou seja, intersecção entre dois retângulos.

Obviamente não precisaríamos ter simultaneamente `largura` e `fimX` ou `altura` e `fimY`, porem achamos proveitoso inclui-los para facilitar a especificação de outras restrições.

1. Tumba

Sendo ‘`n`’ a quantidade de tumbas pretendidas no bloco (serão representadas como `T[1]`, `T[2]`,..., `T[n]`); `VET` um vetor de tamanho ‘`z`‘ com os elementos inseridos anteriormente:

Constantes envolvidas:

- `larguraBloco`.
- `solo`.

Domínio das Variáveis:

- As variáveis `inicioX` e `fimX` de cada tumba podem assumir valores entre zero e `larguraBloco`.
- A variável `fimY` de cada tumba pode assumir valores entre `solo` e `alturaBloco`, entretanto `inicioY` é sempre igual a `solo`.
- A variável `altura` de cada tumba é limitada pelo tipo da tumba, que pode ser pequena, média ou alta, sendo assim o valor desta variável será selecionada a partir

de um vetor com três valores delimitados pelo *game designer*, entretanto é preciso que os valores deste vetor sejam menores que o pulo do avatar.

- A variável `largura` também é selecionada a partir de um vetor de três elementos delimitado pelo *game designer*, entretanto é preciso que os valores deste vetor sejam menores que o tamanho do bloco.

Restrições:

- `Para i de 1 até n`
  `T[i].incioX <= larguraBloco – T[i].largura`
- `Para i de 2 até n`
  `T[i].incioX > T[i-1].incioX + T[i-1].largura`
- `Para i de 1 até n`
  `Para j de 1 ate z`
  `intersecciona(T[i],Vet[j]) = falso`

2. Lama

Sendo ‘`n`’ a quantidade de lamas pretendidas no bloco (serão representadas como `L[1]`, `L[2]`,..., `L[n]`); `VET` um vetor de tamanho ‘`z`‘ com os elementos inseridos anteriormente:

Constantes envolvidas:

- `larguraBloco`.
- `solo`.
- `larguraMaximaLama`.
- `larguraMinimaLama`.

Domínio das Variáveis:

- As variáveis `inicioX` e `fimX` de cada lama podem assumir valores entre zero e `larguraBloco`.
- As variáveis `inicioY` e `fimY` de cada lama são fixas recebendo os valores zero e `solo` respectivamente, consequentemente também deixa a `altura` fixa.

- A variável `largura` de cada lama pode assumir valores entre `larguraMinimaLama e larguraMaximaLama`.

Restrições:

- `Para i de 1 até n`
  `L[i].incioX <= larguraBloco – L[i].largura`
- `Para i de 2 até n`
  `L[i].incioX > L[i-1].incioX + L[i-1].largura`
- `Para i de 1 até n`
  `Para j de 1 ate z`
  `intersecciona(L[i],Vet[j]) = falso`

3. Musgo

Sendo ‘`n`’ a quantidade de musgos desejados no bloco (serão representadas como `M[1], M[2],..., M[n]`); `VET` um vetor de tamanho ‘`z`‘ com os elementos inseridos anteriormente:

Constantes envolvidas:

- `larguraBloco`.
- `solo`.
- `larguraMaximaMusgo`.
- `larguraMinimaMusgo`.

Domínio das Variáveis:

- As variáveis `inicioX` e `fimX` de cada musgo podem assumir valores entre zero e `larguraBloco`.
- As variáveis `inicioY` e `fimY` de cada musgo são fixas recebendo os valores zero e `solo` respectivamente, consequentemente também deixa a `altura` fixa.
- A variável `largura` de cada musgo pode assumir valores entre `larguraMinimaMusgo` e `larguraMaximaMusgo`.

Restrições:

- `Para i de 1 até n`
  `M[i].incioX <= larguraBloco – M[i].largura`
- `Para i de 2 até n`
  `M[i].incioX > M[i-1].incioX + M[i-1].largura`
- `Para i de 1 até n`
  `Para j de 1 ate z`
  `intersecciona(M[i],Vet[j]) = falso`

4. Fantasma

Sendo ‘`n`’ a quantidade de fantamas desejados no bloco (serão representadas como `F[1]`, `F[2]`,..., `F[n]`); `VET` um vetor de tamanho ‘`z`‘ com os elementos inseridos anteriormente:

Constantes envolvidas:

- `larguraBloco`.
- `alturaBloco`.
- `larguraFantasma`.
- `alturaFantasma`.
- `alturaPuloAvatar`.

Domínio das Variáveis:

- As variáveis `inicioX` e `fimX` de cada fantasma podem assumir valores entre zero e `larguraBloco`.
- As variáveis `inicioY` e `fimY` de cada fantasma podem assumir valores entre solo e `solo + alturaPuloAvatar`.
- As variáveis `largura` e `altura` de cada fantama são sempre iguais as constantes `larguraFantasma` e `alturaFantasma` respectivamente.

Restrições:

- `Para i de 1 até n`
  `F[i].incioX <= larguraBloco – F[i].largura`
- `Para i de 1 até n`
  `F[i].incioY <= alturaBloco – F[i].altura`
- `Para i de 1 até n`
  `Para j de i+1 ate n`
  `Intersecciona(F[i],F[j]) = falso`
- `Para i de 1 até n`
  `Para j de 1 ate z`
  `intersecciona (F[i],Vet[j]) = falso`

5. Moeda

Sendo ‘`n`’ a quantidade de moedas desejadas no bloco (serão representadas como `M[1]`, `M[2]`,..., `M[n]`); `VET` um vetor de tamanho ‘`z`‘ com os elementos inseridos anteriormente:

Constantes envolvidas:

- `larguraBloco`.
- `alturaBloco`.
- `larguraMoeda`.
- `alturaMoeda`.
- `alturaPuloAvatar`.

Domínio das Variáveis:

- As variáveis `inicioX` e `fimX` de cada moeda podem assumir valores entre zero e `larguraBloco`.
- As variáveis `inicioY` e `fimY` de cada moeda podem assumir valores entre solo e `solo + alturaPuloAvatar`.
- As variáveis `largura` e `altura` de cada moeda são sempre iguais as constantes `larguraMoeda` e `alturaMoeda` respectivamente.

Restrições:

- `Para i de 1 até n`
  `M[i].incioX <= larguraBloco – M[i].largura`
- `Para i de 1 até n`
  `M[i].incioY <= alturaBloco – M[i].altura`
- `Para i de 1 até n`
  `Para j de i+1 ate n`
  `Intersecciona(M[i],M[j]) = falso`
- `Para i de 1 até n`
  `Para j de 1 ate z`
  `intersecciona(M[i],Vet[j]) = falso`

Utilizando uma biblioteca de satisfação de restrições é possível modelar computacionalmente o PSR descrito nesta seção, e assim aplicar a estratégia descrita na seção 4.1, de modo a criar uma aplicação orientada a restrições que criará os blocos de maneira automatizada repeitando as restrições desejadas.

A próxima etapa diz respeito à classificação dos blocos em relação à dificuldade, a proposta mostrada por Carvalho (CARVALHO, 2012) desta fase, visa classificar os blocos de maneira automática, utilizando uma rede neural para avaliar os blocos sem a necessidade de um *game designer.*

A última etapa trata da inserção dos blocos no decorrer da partida, esta utiliza um algoritmo de sorteio simples, no qual se busca na curva de dificuldade qual deve ser a dificuldade do bloco naquele momento, uma vez obtida a dificuldade do bloco, é sorteado um número aleatório entre um e a quantidade total de blocos disponíveis daquele tipo, o número sorteado indica qual será o bloco a ser inserido naquele momento da partida.

4.3 Implementação

A caracterização do problema como um PSR indicou a necessidade de implementação de uma aplicação orientada a restrições, e no sentido de prover um ambiente prático e eficiente para tal tipo de aplicação foi utilizado o GECODE (GECODE, 2005), uma biblioteca *open-source* em C++ que se mostrou capaz de favorecer este ambiente ao prover mais de 70 tipos de restrição, muitos motores de busca além de um sistema avançado de heurísticas de ramificação (branching) no qual é definida uma estratégia para os motores de busca.

O sistema foi projetado de maneira modular, com a intenção de facilitar uma possível manutenção ou extensão. A seguir a figura 4.8 deixa claro como ficou estruturada a arquitetura da ferramenta.

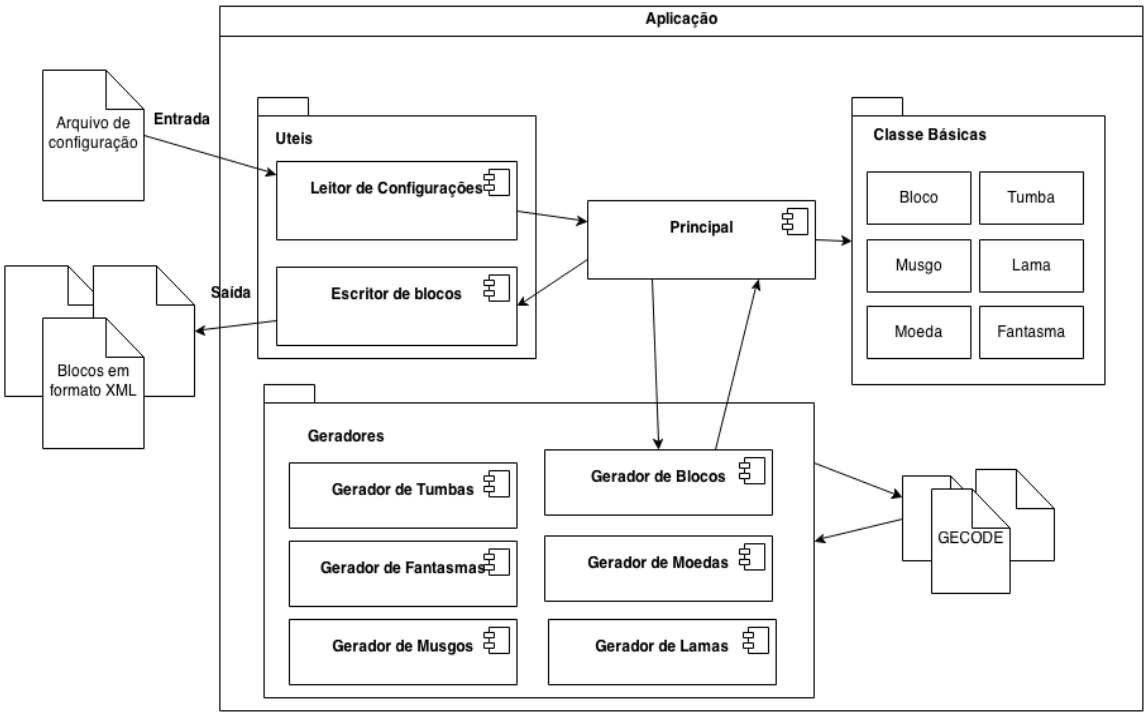


Figura 4.8. Diagrama de componentes da ferramenta de criação de blocos.

A aplicação faz a leitura de um arquivo com as configurações desejadas para os blocos, estas configurações servem de entrada para os geradores que utilizam diretamente os recursos da biblioteca GECODE para prover a soluções de blocos corretos, estes são escritos separadamente em arquivos XML. No anexo 1 pode ser visto um guia de como utilizar a ferramenta.

Capítulo 5 – Avaliação dos resultados

Após a implementação da ferramenta foi feita uma avaliação para validar sua corretude e utilidade. A avaliação consistiu na geração de 35 blocos para que um membro da equipe de desenvolvimento do jogo utilizado no estudo pudesse avaliar a corretude dos blocos com o jogo em execução.

Leonardo Carvalho, membro da equipe de desenvolvimento do jogo utilizado fez a avaliação dos blocos, de um modo geral a opinião dele foi que o posicionamento dos elementos do bloco esta correta, não ocorrendo nenhuma sobreposição de elementos, entretanto ele fez algumas ressalvas sobre alguns objetos ficaram posicionados de maneira que não afetam o *gameplay*, a exemplo moedas que estariam muito altas para o avatar alcançá-las. Outros pontos de melhorias indicados por ele dizem respeito ao posicionamento e a quantidade das moedas no bloco que deveriam ser melhor explorados esteticamente e que alguns blocos teriam uma dificuldade muito elevada.

Depois da avalição um dos problemas descritos acima foi solucionado, a presença de elementos que não afetam o gameplay foi corrigida alterando valores de domínio de alguns elementos, entretanto questões ligadas à estética não tiveram melhora significativa.

Depois das mudanças não houve tempo hábil para uma reavaliação, entretanto ficou bastante perceptível que todos os elementos inseridos no bloco podem interagir com o jogador.

Capítulo 6 – Conclusão

Neste capítulo serão exibidas algumas considerações acerca deste trabalho, de modo a relatar as contribuições oferecidas, as limitações identificadas, e algumas possibilidades de melhoria da ferramenta.

6.1. Contribuições

Uma das principais contribuições deste trabalho é a tentativa de complementar um processo para a geração de níveis de fato infinito, o tornado cada vez mais automatizado. O desenvolvimento de uma ferramenta e processo desta natureza é uma contribuição valiosa e oportuna, pois a varias das soluções existentes apresentam a capacidade de gerar uma quantidade infinita de níveis, mas poucas destas abordagens demonstram formas possíveis de se criar um único nível sem fim de maneira computacionalmente viável.

A automatização da etapa de criação de blocos apresentada neste trabalho, pode afetar diretamente na redução dos custos envolvidos no processo, visto que é possível criar de forma mais rápidas os blocos que compõem o nível, e sendo a quantidade e a diversidade do conteúdo apresentado por um jogo fatores que aumentam o potencial replay, é possível concluir que a solução refletira em uma experiência positiva para jogador. O processo também o torna o sistema mais genérico para introdução de novos elementos, facilitando a criação de atualizações para o conteúdo existente.

6.2. Limitações

Uma das limitações encontradas é a dificuldade em modelar o problema de maneira que possam ser gerados blocos esteticamente agradáveis. Outra limitação a se considerar, é que a velocidade em que os blocos são gerados não é suficiente para que esta etapa ocorra de forma Online, mesmo porque a etapa de classificação que é realizada em seguida é outra barreira que impede que a fase de geração de blocos seja feita durante uma partida.

6.3. Trabalhos Futuros

Durante o processo de teste e validação da ferramenta foram descobertas aberturas para melhoria em diversos pontos do processo de geração e da ferramenta apresentada.

Em relação ao processo podemos destacar a estética apresentada pelos blocos, que pode ser aprimorada através de uma melhor especificação das restrições ou por meio de um maior aprofundamento nos recursos da biblioteca utilizada.

No que diz respeita à ferramenta, a substituição do arquivo de configuração por uma interface gráfica tornaria a aplicação mais amigável para equipe de desenvolvimento do jogo, como também o armazenamento de configurações para uso posterior.

A adição de novos parâmetros de configuração tais como porcentagem máxima e mínima de área ocupada por determinado tipo de elemento são importantes para que se tenha mais controle sobre resultados esperados.

Referências

BIGHUT GAMES 2012. Website do produtor do jogo Bonny The Runner, BigHut Games. http://bighutgames.com/. Acesso em 10/12/2013

CARVALHO, Leonardo. Bonnymetrics - Ferramenta para Jogos Infinitos, 2012.

CARVALHO, Leonardo; MOREIRA, Átilia; ALBUQUERQUE, Túlio e Ramalho, Geber. Generic Framework for Procedural Generation of Gameplay Sessions, 2013.

COMPTON, Kate e MATEAS, Michael. Procedural Level Design for Platform Games, em Proc. Artif. Intell. Interactive Digit. Entertain. Int. Conf., 2006, pp. 109–111.

DORAN, Jonathan e PARBERRY, Ian. Controlled Procedural Terrain Generation Using Software Agents, 2010.

FRATTESI, Timothy; GRIESBACH, Douglas; LEITH , Jonathan e SHAFFER, Timothy.Replayability of Video Games, Maio 2011.

GECODE 2005. Website do projeto da biblioteca GECODE. http://www.gecode.org/. Acesso em 10/12/2013

HALFBRICK STUDIOS 2011. Website do produtor do jogo Jetpack Joyride, Halfbrick. http://www.halfbrick.com/our-games/jetpack-joyride/. Acesso em 10/12/2013

HENDRIKS, Mark; MEIJER, Sebastian; VELDEN, Joeri e IOUSUP, Alexandru, Procedural Content Generation for Games: A Survey, Fev. 2011.

HUNICKE, Robin; CHAPMAN, Vernell. AI for Dynamic Difficulty Adjustment in Games, em Challenges in Game Artificial Intelligence AAAI Workshop 91–96, 2004.

JENNINGS-TEATS, Martim; SMITH, Gillian e WARDRIP-FRUIN, Noah. Polymorph: Dynamic Difficulty Adjustment Through Level Generation, 2010.

KOSTER, Ralph. A Theory of Fun for Game Design. Praraglyph press, 2005.

LASKOV, Atanas. Level Generation System for Platform Games Based on a Reinforcement Learning Approach, 2010.

MAWHORTER, Peter e MATEAS, Michael. Procedural Level Generation Using Occupancy-Regulated Extension, 2010.

PEDERSEN, Chris; TOGELIUS, Julian e YANNAKAKIS, Georgios N.. Modeling player experience in super Mario Bros, em Proc. IEEE Symp. Comput. Intell. Games, pp. 132–139, 2009.

ROSSI, Francesca; BEEK, Peter e WALSH, Toby. Handbook of Constraint Programming, chapter one: Constraint Programming, 2006

SHAKER, Noor; YANNAKAKIS, Georgios e TOGELIUS, Julian. Towards Automatic Personalized Content Generation for Platform Games, em Proc. Artificial Intelligence and Interactive Digital Entertainment, pp. 63-68, Out. 2010.

SILVA, Pedro. Modelação Procedimental para Desenvolvimento de Jogos de Computador, Junho 2010.

SMITH, Gillian; TREANOR, Mike; WHITEHEAD, Jim e MATEAS, Michael. Rhythm-based level generation for 2d platformers, em in Proc. Found. Digit. Games, 2009, pp. 175–182.

SORENSON, Nathan e PASQUIER, Philippe. Towards a Generic Framework for Automated Video Game Level Creation, em Proc. European Conf. Applications of Evolutionary Computation, pp. 130-139, 2010.

SWEETSER , Penelope e WYETH , Peta. GameFlow: a model for evaluating player enjoyment in games. Comput. Entertain, 2005.

TOGELIUS, Julian; YANNAKAKIS, Georgios e STANLEY, Kenneth. Search-based procedural content generation. In: Proceedings of EvoApplications. Springer LNCS, 2010.

TOGELIUS, Julian; CHAMPANDARD, Alex; LANZI, Pier; MATEAS, Michael; PAIVA, Ana; PREUSS, Mike e STANLEY, Kenneth. Procedural Content Generation: Goals, Challenges and Actionable Steps, 2013.

TOGELIUS, Julian; SHAKER, Noor e NELSON, MARK. Procedural Content Generation in Games: A Textbook and an Overview of Current Research, Chapter one: Introduction, 2014.

VENDRIG, Nicky. Automated level generation and difficulty rating for Trainyard, Out. 2013.

Anexos

Anexo 1 – Guia do Usuário

O primeiro passo para utilizar a ferramenta é preencher o arquivo de configuração boney.cfg, no qual deve-se apontar o valores desejados para os parâmetros que delimitam a expressividade do gerador. A seguir um exemplo de como preencher o arquivo de configuração:

```
#Geral

numeroDeBlocos = 3 #número de blocos que se deseja gerar

nomeBase = arcj #Nome base para os arquivos XML gerados (arcj1 até arcj3)

dificuldadeDoBloco = E #dificuldade do bloco(E = fácil, M = médio, H = difícil)


#Tumbas

numeroDeTumbas = 0 #numero de tumbas desejadas nos blocos

maxRandomTumbas = 3 #gera de 1 a N tumbas.Se for diferente 0 numeroDeTumbas é ignorado


#Areia Movediça(lama)

numeroDeLamas = 0 #numero de lamas desejadas nos blocos

maxRandomLamas = 0 #gera de 1 a N lamas.Se for diferente 0 numeroDeLamas é ignorado


#Musgo

numeroDeMusgos = 0 #numero de musgos desejados nos blocos

maxRandomMusgo = 0 #gera de 1 a musgos .Se for diferente 0 numeroDeMusgos é ignorado


#Fantasmas

numeroDeFantasmas = 3

maxRandomFantamas = 0#gera de 1 a N fantamas.Se não for igual a 0 numeroDeFantasmas é ignorado
```

```
#Moedas

numeroDeMoedas = 0

maxRandomMoedas = 5 #gera de 1 a N moedas. Se for iguak a 0 numeroDeMoedas
é ignorado
```

A configuração mostrada gerará três blocos que serão indicados como fáceis, os bloco gerados terão no máximo três tumbas é no mínimo uma, não existirão lamas nem musgos, terão exatamente três fantasma e de uma à cinco moedas.

Depois de configurado abra o arquivo BoneyBlocksGen.exe e espere até este fechar sozinho, ao final será criado para cada bloco o arquivo XML que o representa.