Sistema integrado de geração automática de conteúdo para videojogos de plataformas

Fausto Mourato
CITI - Departamento de Informática
Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa
2829-516 Caparica
p21748@campus.fct.unl.pt

Fernando Birra
CITI - Departamento de Informática
Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa
2829-516 Caparica
fpb@di.fct.unl.pt

Manuel Próspero dos Santos
CITI - Departamento de Informática
Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa
2829-516 Caparica
ps@di.fct.unl.pt

Resumo

Neste artigo é apresentado um sistema de suporte à geração automática de níveis para videojogos de plataformas. Este assenta numa framework genérica de representação de níveis, compatível com grande parte dos conceitos existentes neste tipo de jogos. Numa perspectiva de edição de conteúdo, o sistema pode ser utilizado como editor de níveis genérico. A arquitectura desenvolvida é modular e baseada em plugins, permitindo a adaptação para outros formatos de níveis. Com um princípio similar, é possível integrar algoritmos de geração automática e adaptação de conteúdo. É abordada a arquitectura da solução proposta e a forma como é possível estabelecer um pipeline de algoritmos de geração automática. Finalmente, são também descritos alguns destes algoritmos.

Keywords

Videojogos de plataformas, Geração Procedimental de Conteúdo, Ajuste de Dificuldade

1. INTRODUÇÃO

A geração automática de conteúdo no contexto dos videojogos tem suscitado interesse na comunidade académica e em equipas de desenvolvimento de pequena dimensão. Por um lado, trata-se de uma forma de suprimir a escassez de recursos humanos comparativamente às grandes empresas do segmento e, por outro lado, trata-se de um desafio intelectual interessante de algoritmia onde diversos aspectos têm que ser considerados.

Neste artigo abordamos um sistema desenvolvido para o suporte de diversas técnicas de geração automática de níveis, em particular para videojogos de plataformas, onde serão salientados os seguintes aspectos:

- A arquitectura baseada em *plugins*, potenciando a expansão da aplicação para outros jogos e permitindo a inclusão de novas técnicas de geração automática.
- O mecanismo baseado em regras para extracção automática de possíveis caminhos dentro de um nível e a análise dos mesmos.
- Os principais algoritmos desenhados e implementados para a geração automática de níveis.
- Uma técnica de ajuste de dificuldade e adaptação de conteúdo, desenhada para a criação de desafios compostos por um trajecto não linear, baseado na recolha de objectos e activação de *triggers*.

O referido sistema pretende potenciar o desenvolvimento de novos videojogos de plataformas a partir de uma metodologia comum capaz de integrar diversos algoritmos para geração automática. Contudo, é possível utilizar a aplicação enquanto simples editor de níveis sem recurso a geração procedimental.

Os casos de estudo apresentados abordam jogos existentes, considerando o potencial que a aplicação teria se existisse e tivesse sido utilizada como apoio aquando do desenvolvimento desses mesmos jogos.

Relativamente à estrutura deste documento, na Secção 2 são referidos os estudos mais importantes relacionados com a geração automática de conteúdo neste tipo de jogos. Na Secção 3 descreve-se o sistema implementado, com principal ênfase na *framework* para representação de níveis e na arquitectura modular da aplicação. De forma mais superficial, são também abordados alguns dos algoritmos implementados. A Secção 4 complementa a descrição com uma visão geral sobre a utilização da aplicação com recurso a alguns exemplos. Finalmente, as principais conclusões e aspectos para desenvolvimento futuro são apresentados na Secção 5.

2. TRABALHO RELACIONADO

O estudo dos videojogos de plataformas foi iniciado em contexto académico por Compton e Mateas [Compton06]. Os autores analisaram os principais componentes deste tipo de jogos e definiram um modelo conceptual para os caracterizar. Foram também identificados os principais padrões de trajectos.

Posteriormente, uma abordagem mais detalhada foi apresentada por Smith *et al.* [Smith08], com a definição de uma hierarquia para descrição do conteúdo de um nível

com vista à análise do mesmo. De uma perspectiva geral, um nível é apresentado como um conjunto de células interligadas por portais. Uma célula é vista como uma região do nível de tamanho variável, onde a personagem pode realizar um trajecto directo composto por um ou mais grupos de ritmo. Estes grupos representam uma sequência coerente de acções a realizar pelo jogador. Finalmente, a um grupo de ritmo são associados diversos elementos de jogo, nomeadamente: plataformas, obstáculos, itens, *triggers* e auxiliares de movimento. Os princípios apresentados deram seguimento a uma abordagem para a geração automática de conteúdo neste tipo de jogos, designada por *Rhythm Based* [Smith09]. A geração baseada em ritmo consiste na definição inicial de um conjunto coerente de acções para o utilizador realizar e, em seguida, na criação de uma geometria para o nível na qual essas acções sejam possíveis. Esta abordagem concretizou-se num protótipo intitulado de *Launchpad* [Smith11]. Paralelamente a esta linha de trabalho, há ainda a referir alguns estudos relacionados dentro do mesmo grupo de investigação. A capacidade de expressão desta abordagem foi explorada nas vertentes de linearidade do caminho e uma aproximação ao conceito de dificuldade [Smith10a]. Adicionalmente, realizaram-se testes sobre uma adaptação da ideia inicial numa abordagem mista de criação automática com troços fixos definidos por um utilizador humano [Smith10b]. Finalmente, os estudos incidiram ainda sobre a possibilidade de adaptação da geração baseada em ritmo para ter em conta o desempenho do utilizador, adequando o desafio das acções com a perícia demonstrada [Jennings-Teats10].

Uma abordagem mais directa foi proposta por Pedersen *et al.* [Pedersen09]. O gerador apresentado parte de uma parametrização inicial onde se define o número de saltos a realizar, a sua amplitude média e a frequência com que estes acontecem, entre outros. Este gerador foi usado para realizar sessões de jogo com utilizadores e extrair percepções como, por exemplo, a de dificuldade.

Finalmente, Mawhorter e Mateas [Mawhorter10] propuseram uma outra técnica alternativa interessante com o intuito de gerar níveis de forma automática. A abordagem parte da criação inicial de um conjunto de amostras de níveis, as quais são replicadas e compostas numa grelha pela aplicação do algoritmo *Occupancy-Regulated Extension* (ORE), o qual é apresentado nesse mesmo artigo.

Desde o ano de 2010, a *IEEE Conference on Computational Intelligence and Games* tem apresentado um *track* específico sobre o tema da geração automática de níveis, utilizando-se para o efeito uma versão adaptada do jogo *Infinite Mario Bros*. Algumas das técnicas previamente desenvolvidas foram testadas também neste contexto. Os detalhes sobre a primeira edição da competição podem ser consultados no artigo *The 2010 Mario AI Championship: Level Generation Track* [Shaker10].

3. O SISTEMA IMPLEMENTADO

No âmbito deste trabalho, concebeu-se e implementou-se uma aplicação na linguagem C# sobre a plataforma .NET funcionando como sistema para integração de diversas técnicas de geração automática. Esta assenta numa *framework* que permite representar genericamente um qualquer nível de plataformas. Em seguida, na Secção 3.1 é apresentada a estrutura desta *framework* e a forma como os níveis podem ser representados. Na Secção 3.2, são apresentados os detalhes da integração com formatos de alguns jogos existentes. Posteriormente, na Secção 3.3, é explicada a arquitectura modular que permite a integração de algoritmos de geração automática. A Secção 3.4 aborda a questão da extracção automática de caminhos a partir de grafos e, finalmente, na Secção 3.5 são apresentados os principais algoritmos de geração incluídos no sistema.

3.1 Modelo de representação de níveis

Conforme referido na Secção 2, foi apresentada uma proposta de *framework* por [Smith08], com o foco principal na representação conceptual de um nível e na identificação de regiões. No caso do trabalho aqui apresentado, pretende-se uma representação mais concreta do conteúdo, capaz de mapear directamente a informação existente e com menor grau de ambiguidade.

Como ponto de partida para a definição dos níveis, assumiu-se uma representação baseada em enumeração espacial discreta sob a forma de uma matriz bidimensional. É comum em videojogos de plataformas o recurso a representações neste formato. Por exemplo, a estrutura dos níveis dos jogos *Infinite Mario Bros.*, *Prince of Persia* e *Rick Dangerous* consiste precisamente numa matriz bidimensional. Adicionalmente, e embora se trate, aparentemente, de uma restrição ao domínio, este tipo de representação potencia um conjunto de técnicas para geração automática, como se irá constatar ao longo desta secção. Assim, um nível é composto por um conjunto de **células**, preenchidas a partir de um conjunto pré-definido de **blocos**. Para aumentar a capacidade de representação desta abordagem, o conjunto de blocos associado a um nível ou jogo é representado numa hierarquia. Em cada célula é ainda possível definir um conjunto de **propriedades** e respectivos **valores** estabelecido para cada tipo de bloco.

Para esclarecer os conceitos anteriores, considere-se como exemplo um jogo minimalista baseado no universo *Super Mario Bros.*, possuindo somente os seguintes tipos principais de blocos:

- *Preenchido*, representando um bloco de existência física onde a personagem não pode passar e que serve também de eventual suporte para a mesma.
- *Vazio*, representando um espaço por onde a personagem pode passar e em que o conteúdo gráfico existente é meramente estético.
- *Plataforma*, representando restrição somente ao movimento vertical descendente da personagem, servindo assim como suporte para a mesma sem impedir o seu movimento, como se estivesse numa camada distinta. A Figura 1 apresenta exemplos de utilização deste tipo de bloco nos jogos *Infinite Mario Bros.* e *XRick*.

Considere-se ainda que o bloco vazio possui variantes para a representação do início e fim do nível e duas reproduções visuais distintas de carácter estético. A sua representação poderia assentar na hierarquia apresentada na Figura 2. Com base neste modelo inicial é possível, por exemplo, descrever o nível apresentado na Figura 3.

Suponha-se também que, na implementação do hipotético jogo correspondente a esta descrição, as flores poderiam ser visualizadas com uma qualquer cor. Neste caso, poder-se-ia definir uma propriedade *Cor* para o bloco *Flor* na qual se poderia indicar o valor RGB que descrevesse essa mesma cor. De momento, do ponto de vista do sistema os valores são todos representados em formato de texto. No entanto, tal não impede que a efectiva implementação do jogo possa processar os dados e efectuar a sua conversão para um qualquer formato.

Adicionalmente, a *framework* contempla o conceito de **grupo**, o qual permite a definição de um conjunto de **blocos** que só fazem sentido se interpretados em conjunto. A definição de grupos é opcional mas, no entanto, é útil tanto para a edição manual de um nível como para geração automática, uma vez que providencia informação adicional sobre a possível organização das células. A existência deste tipo de estruturas é comum em abordagens baseadas em células. Por exemplo, no jogo *Prince of Persia* a representação das portas de início e fim do nível recorre ao uso de duas células, tal como é possível observar na Figura 4. Do mesmo modo, um cano no jogo *Infinite Mario Bros*. pode ser visto como um exemplo de um **grupo**. Contudo, este caso em particular possui uma característica complementar que também é tida em conta na *framework*. Apesar de possuírem uma largura fixa de duas células, os canos neste jogo têm altura variável. Este tipo de situações é mapeado sob o conceito de **grupo de dimensão variável**.

Com o intuito de potenciar a possível ligação entre a representação e os processos de geração automática, implementou-se no sistema um mecanismo para definição de **regras** baseadas em padrões, permitindo ao utilizador definir agrupamentos especiais de células que podem originar um determinado processamento. Actualmente, esta funcionalidade tem sido utilizada para a representação aproximada dos principais movimentos da personagem, facultando a posterior extracção automática de grafos de trajectos para os níveis. Deste modo, é possível definir conjuntos de blocos e indicar um sub-grafo associado a cada conjunto, representando os possíveis movimentos da personagem do jogo.

Durante o estudo da capacidade de representação dos níveis de diversos jogos, verificou-se que, por vezes, a hierarquia não é suficiente para designar directamente todos os conjuntos lógicos de blocos. No exemplo simples definido anteriormente, esta questão encontra-se presente. Os blocos *Plataforma* podem ser interpretados da mesma forma que *Vazio*, para algumas situações, ou *Preenchido*, para outras. No movimento horizontal descrito acima, o deslocamento indicado para a personagem

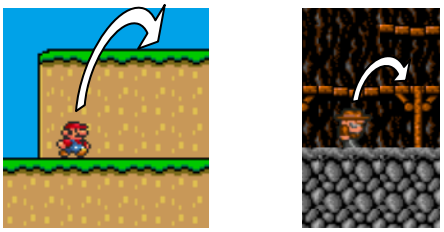

Figura 1 – Exemplo de simulação de múltiplas camadas em jogos de plataformas bidimensional (*Infinite Mario Bros*. à esquerda e *XRick* à direita)

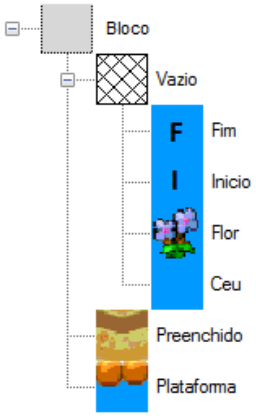

Figura 2 – Exemplo de uma hierarquia de blocos

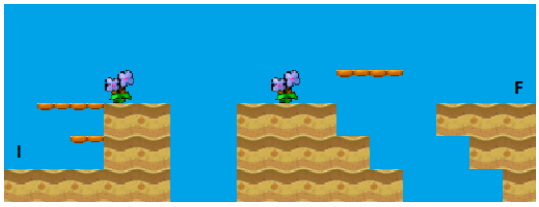

Figura 3 – Nível de exemplo utilizando uma hierarquia de blocos simples

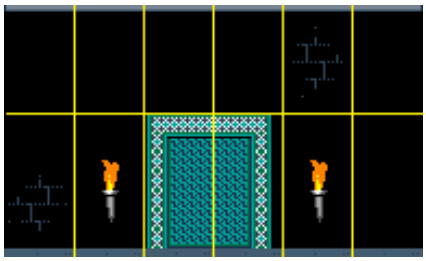

Figura 4 – Exemplo da representação em grelha no jogo *Prince of Persia*

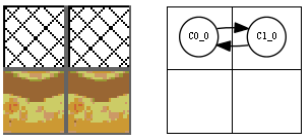

Figura 5 – Exemplo de uma regra para construção do grafo de trajectos. À esquerda é definido o padrão a procurar no nível e à direita é indicado o sub-grafo correspondente.

seria também válido com blocos do tipo *Plataforma* nas células de baixo servindo como suporte, bem como para células de cima, pois não impedem a passagem da personagem.

No entanto, não é possível definir o elemento *Plataforma* como nó filho dos nós *Vazio* e *Preenchido* simultaneamente numa hierarquia simples. Deste modo, estabeleceu-se também a possibilidade de definir **categorias** de blocos, construídas com referências à hierarquia e com recurso a operadores lógicos. Neste caso, poderia ser construída uma categoria *Suporte* através de uma disjunção, usando a expressão *Bloco/Preenchido* ∨ *Bloco/Plataforma*. Do mesmo modo, poder-se-ia definir uma categoria *Movimentável* novamente com recurso a uma disjunção, neste caso com a expressão *Bloco/Vazio* ∨ *Bloco/Plataforma*. Assim, a forma mais completa de descrever o movimento horizontal da personagem seria a regra apresentada na Figura 6.

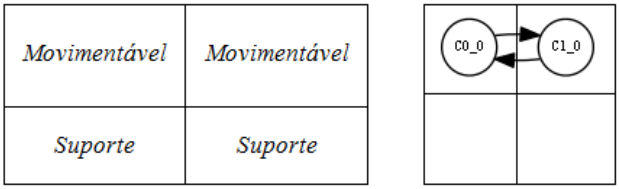

Figura 6 – Exemplo de uma regra para construção de trajectos, utilizando *Categorias*.

Sintetizando a *framework* acima descrita, o modelo de representação de níveis é composto pelos seguintes elementos: Hierarquia de blocos, Grupos, Grupos de dimensão dinâmica e Regras.

O sistema implementado permite a serialização da informação anterior em formato XML para possível partilha e reutilização.

Numa primeira abordagem à aplicação desenvolvida, esta pode ser vista como um editor de níveis para o domínio definido. Os níveis são representados para um determinado modelo descrito na *framework* anterior, contendo uma matriz bidimensional de células com referência aos blocos que as descrevem e eventuais pares de parâmetros e respectivos valores. Os níveis podem também ser guardados e lidos em formato XML.

3.2 Integração com diversos videojogos

A aplicação possui uma estrutura modular para potenciar a representação de níveis de quaisquer jogos de plataformas. É possível implementar *plugins* específicos para jogos em particular desde que os níveis estejam de alguma forma acessíveis. Um *plugin* para um determinado jogo deverá conter, quando aplicável:

- O modelo de representação dos níveis desse jogo, descrito na *framework* apresentada anteriormente.
- A implementação de um método capaz de ler os dados a partir do jogo original e traduzi-los no modelo definido.
- A implementação de um método para exportar os dados de um nível criado na aplicação para substituir os dados originais desse jogo.
- Uma implementação de um método para lançar o jogo com o nível em visualização na aplicação.

À data, foram realizados testes de integração com os videojogos *Prince of Persia*, *Rick Dangerous, Infinite Mario Bros* e *Open Sonic*. A escolha destes títulos deveu-se principalmente ao facto de serem versões livres, de código aberto ou com existência de um formato conhecido e passível de edição. De facto, o maior entrave à integração com outros jogos está no facto de estes possuírem níveis tipicamente em formato proprietário e cujas especificações não são do domínio público.

O principal intuito do estudo desta capacidade de integração é, acima de tudo, o reconhecimento da abrangência da *framework* e a identificação de aspectos que possam ser melhorados.

3.3 Arquitectura para Geração Automática

A aplicação permite a definição de um *pipeline* de técnicas sucessivas para geração automática de níveis. Para o efeito, considera-se que uma técnica de geração automática é definida pelas seguintes características:

- Possui um conjunto predefinido de parâmetros de configuração.
- Indica o número de níveis que devem ser facultados ao algoritmo para uma determinada parametrização (dimensão do *input*).
- Indica o número de níveis que o algoritmo produz para uma determinada parametrização (dimensão do *output*).

Por exemplo, considere-se um gerador simples para o jogo *Infinite Mario Bros.* que gere sequências de blocos de terreno com a mesma altura, separados entre si. Os valores de espaçamento médio, mínimo e máximo são exemplos de possíveis parâmetros de configuração. Neste caso particular, independentemente da configuração, o número de níveis que o algoritmo recebe será 0 e o número de níveis gerados será, tipicamente, 1. Eventualmente, poderá ser considerada uma variante que permita a geração de múltiplos níveis em simultâneo, sendo o número de níveis definido como parâmetro.

É possível encadear um qualquer número de algoritmos de geração, preferencialmente se as dimensões de *output* e *input* forem equivalentes em algoritmos sucessivos. Nos casos em que esta condição não se verifica, é possível descartar resultados de um fase anterior e/ou forçar múltiplas invocações dessa mesma fase, impondo um determinado número de níveis.

Considere-se um segundo exemplo de um algoritmo de decoração que complemente um nível com elementos estéticos, ou seja, o algoritmo recebe 1 nível e produz como output também 1 nível. Assim, estes dois algoritmos podem funcionar de forma sucessiva, sendo que o segundo processa o *output* do primeiro, produzindo o

resultado final. Na Figura 7 é apresentado um exemplo da utilização destes dois algoritmos em sucessão. Na Figura 8 é apresentada a interface para o utilizador definir a sequência de algoritmos que pretende utilizar.

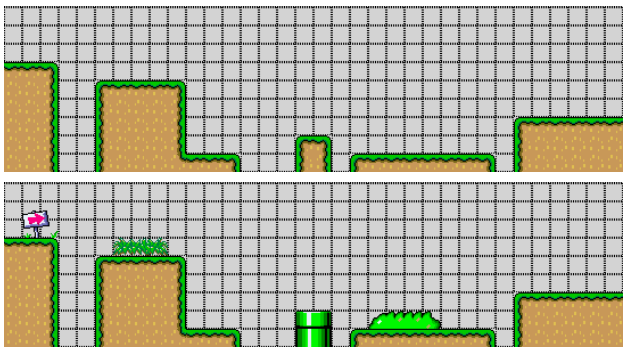

Figura 7 – Exemplo de construção automática de níveis com sequência de algoritmos. Em cima, o resultado de um primeiro algoritmo que cria a estrutura base do nível. Em baixo, o resultado de um algoritmo de decoração aplicado ao primeiro nível.

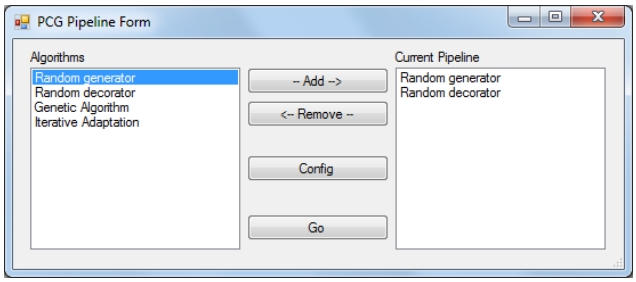

Figura 8 – Janela de configuração do *pipeline* de geração automática de conteúdo

3.4 Extracção automática de grafo de trajecto

Como foi explicado na Secção 3.1, é possível definir regras para associar um padrão de blocos a um sub-grafo. Uma das ferramentas incluídas no sistema permite o processamento de todas as regras para criação de um grafo geral dos trajectos de um nível. Na Figura 9 é possível observar o resultado da aplicação desta ferramenta num nível do jogo *Prince of Persia*.

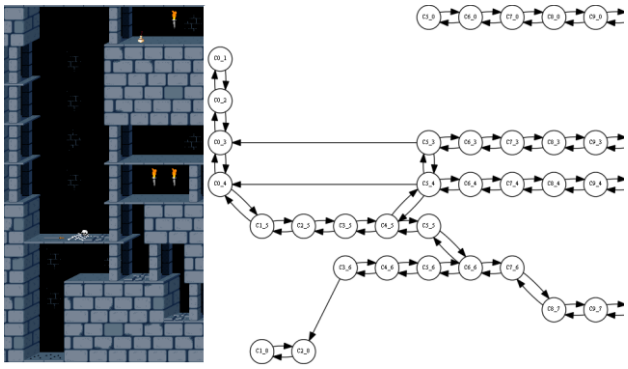

Figura 9 – Exemplo de um grafo de trajectos no primeiro nível do jogo *Prince of Persia*

Para tornar o processamento do grafo mais eficiente, é possível aplicar um passo de compactação ao mesmo, removendo os nós que representem somente transições obrigatórias entre dois outros nós. Na Figura 10 é apresentado o grafo do exemplo anterior após compactação.

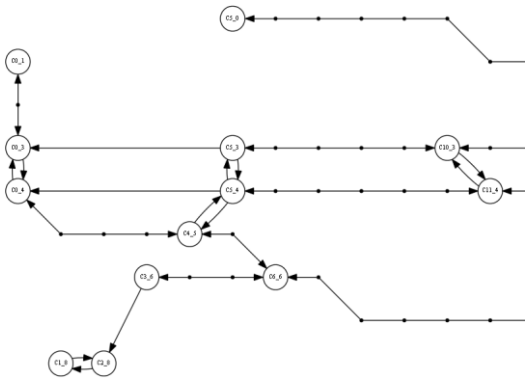

Figura 10 – Exemplo de um grafo compactado

Uma das funcionalidades expectáveis após a extracção do grafo é o cálculo do caminho mais curto entre dois pontos, tendo esta sido implementada recorrendo ao algoritmo de Dijkstra. Contudo, há limitações naturais na análise do grafo. Por um lado, existe a limitação prática associada à complexidade computacional, caso sejam representadas situações menos triviais, como caminhos de uso único, recolha de recursos ou interruptores. Um estudo detalhado sobre este assunto pode ser consultado no artigo intitulado *Gaming is a hard job, but someone has to do it* [Viglietta12]. Por outro lado, o nível a analisar pode ser uma representação incompleta num passo intermédio de uma sequência de algoritmos. Assim, um dos intuitos da criação e análise do grafo pode ser o auxílio à construção de um nível mais completo.

Deste modo, a análise ao grafo contempla, adicionalmente, o cálculo de todos os caminhos possíveis entre os dois pontos definidos como início e final do nível, excluindo todos os trajectos que repetem nós. Posteriormente, os nós do grafo são classificados consoante o seu papel no trajecto do nível, existindo as seguintes classificações:

- **Obrigatório**, para os nós que estão presentes em todos os trajectos entre o ponto inicial e o ponto final.
- **Opcional**, para os nós que fazem parte somente de alguns dos trajectos entre o ponto inicial e o ponto final.
- **Fim de caminho**, para os nós onde a única saída é um nó de entrada.
- **Caminho sem saída**, para os nós que não fazem parte de nenhum dos trajectos mas fazem parte de uma ligação entre um ou mais desses trajectos e um nó classificado como *Fim de Caminho*.
- **Inacessível**, para os nós que correspondem a posições que a personagem não consegue alcançar.
- **Sem retorno**, para os nós que correspondem a posições alcançáveis pela personagem mas que não permitem o retorno a nós dos caminhos válidos para finalizar o nível.

Esta classificação faculta um conhecimento adicional sobre a estrutura do nível. Na Secção 3.5.2 é apresentado um algoritmo para adaptação do conteúdo de um nível que tira proveito desta classificação.

3.5 Algoritmos de geração automática

Neste ponto abordamos duas técnicas implementadas e testadas sob o sistema apresentado.

3.5.1 Geração com Algoritmos Genéticos

Uma das técnicas implementadas permite a geração da estrutura base para níveis do jogo *Prince of Persia*, muito embora a mesma abordagem possa ser aplicada a jogos similares, em cenários tipicamente fechados com salas interligadas por corredores. Esta consiste na definição de um algoritmo genético onde os níveis são abordados como indivíduos de uma população, e passam por um processo de evolução mediante um conjunto de regras de avaliação, mutação e combinação. Estas foram estabelecidas a partir de heurísticas de desenho, tendo em conta os seguintes aspectos:

- Estrutura do caminho principal, nomeadamente em relação à linearidade e ramificação do mesmo.
- Análise individual de cada célula em relação ao seu contexto (vizinhança).
- Posicionamento da célula final do nível e dimensão do caminho principal criado.
- Equilíbrio do uso dos diversos tipos de bloco.
- Uso adequado do espaço disponível.

Esta técnica permite a geração de estruturas base como a apresentada na Figura 11. Uma descrição mais detalhada do funcionamento do algoritmo e dos testes realizados pode ser consultada em [Mourato11].

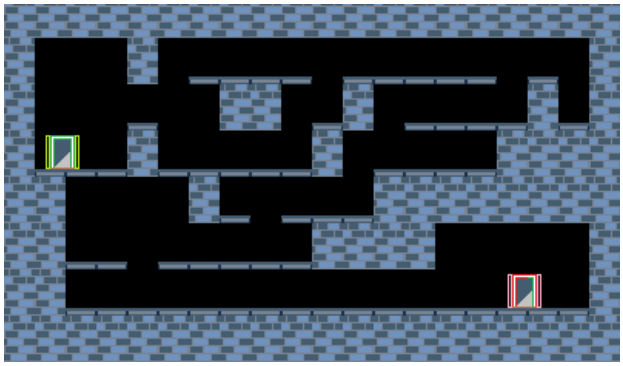

Figura 11 – Exemplo de um nível gerado automaticamente dentro da aplicação a partir de um algoritmo genético

3.5.2 Algoritmo Iterativo para adaptação de conteúdo e ajuste de dificuldade

O algoritmo de adaptação implementado visa a personalização de um nível existente. Este pode ter sido manualmente criado ou gerado automaticamente por um algoritmo como o apresentado no ponto anterior. Tipicamente, deverá recorrer a um conjunto limitado de blocos, descrevendo a geometria base, uma vez que será este algoritmo o responsável pela colocação de outro tipo de blocos que tornam o conteúdo do nível mais completo.

O ponto de partida para este processo é a análise do grafo apresentada anteriormente na Secção 3.4, nomeadamente no que respeita à classificação dos nós desse mesmo grafo. Adicionalmente, o caminho é decomposto em segmentos sendo identificadas as diversas alternativas de caminho para cada segmento. A ideia principal é a de alterar alguns dos blocos existentes para atingir um determinado valor de comprimento no caminho a realizar, dentro de uma determinada estimativa de dificuldade. As alterações são realizadas iterativamente em diversas passagens em cada segmento do nível.

Com base na informação recolhida, o algoritmo procede ao seguinte tipo de alterações:

- Alteração da dificuldade de um segmento, pela inclusão ou remoção de entidades adicionais nesse mesmo segmento, como por exemplo adversários ou armadilhas. As probabilidades de acrescentar ou remover entidades são definidas a partir da estimativa geral de dificuldade.
- Criação de desvios ao trajecto, pela colocação de interruptores em caminhos sem saída e portões ou entidades semelhantes no caminho principal, associados a esses interruptores. Este tipo de alteração permite controlar o comprimento do trajecto a realizar.
- Ajustes para jogabilidade com dois jogadores de perícia distinta, adaptando segmentos paralelos, tornando um deles particularmente mais difícil que o outro e forçando a utilização de caminhos distintos entre os jogadores.
- Inclusão de entidades de bónus em caminhos sem saída para conferir significado a essas mesmas regiões do nível.

Na Figura 12 é apresentado um exemplo de nível criado com este algoritmo, novamente aplicado ao jogo *Prince of Persia*, tendo como base o nível gerado inicialmente com a técnica anterior.

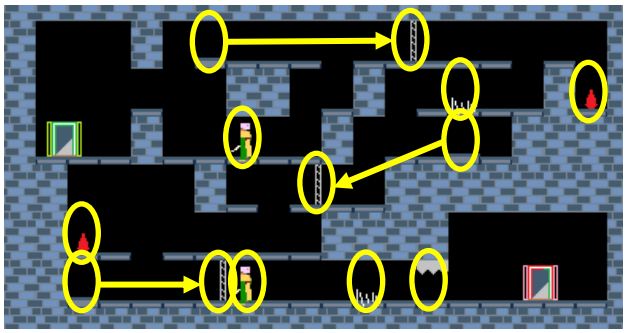

Figura 12 – Exemplo de um nível ajustado pelo algoritmo iterativo de adaptação de conteúdo. As modificações efectuadas estão marcadas com uma elipse. A associação entre botões e portões encontra-se marcada com setas.

Todas as entidades presentes no nível para além da geometria base foram colocadas pelo algoritmo de adaptação. É possível observar a existência de guardas e armadilhas no caminho, sendo o seu número dependente da dificuldade requerida pelo utilizador. É também possível observar a colocação de duas poções em troços sem saída. Finalmente, o sistema colocou também três portões fechados aos quais associou botões para os abrir, tendo estes sido colocados fora do trajecto principal, forçando o utilizador a realizar desvios a esse mesmo trajecto.

Uma descrição mais detalhada desta abordagem pode ser consultada em [Mourato12].

4. EXEMPLOS DE UTILIZAÇÃO

Nesta secção é descrito o funcionamento da aplicação, clarificando-se as suas potencialidades.

Como ponto de partida, na Figura 13 é apresentada uma visão geral da janela principal do programa. Nesta também é possível observar a aplicação de um dos *plugins* desenvolvidos, neste caso para importar níveis do videojogo *Rick Dangerous*.

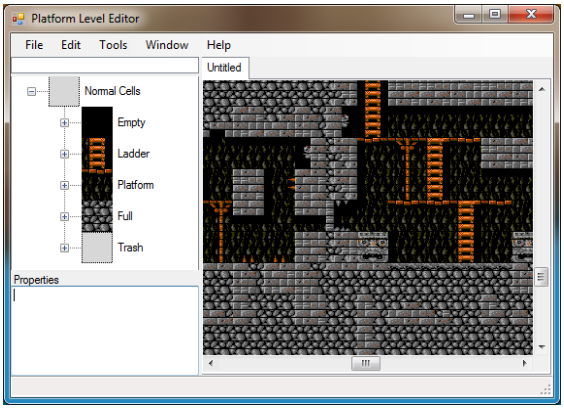

Figura 13 – Janela principal da aplicação desenvolvida, visualizando um nível importado do videojogo *Rick Dangerous*

Na Figura 14 é possível visualizar a capacidade de geração de níveis e utilização dentro de um jogo, aqui concretamente para uma versão adaptada do *Infinite Mario Bros*. Como foi referido anteriormente, este programa tem sido utilizado em contexto académico, no qual têm sido implementados alguns algoritmos de geração automática. Neste caso, o gerador criado acede aos dados do nosso sistema, lendo um nível no formato XML e interpreta-o como se o gerador fosse interno ao próprio jogo.

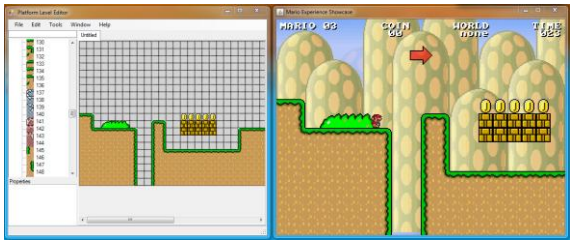

Figura 14 – Visualização lado a lado da janela de criação e edição de um nível e a janela do jogo *Infinite Mario Bros.* executando esse mesmo nível

Conforme indicado, a aplicação permite a definição de todo o modelo de representação dos níveis. Na descrição do sistema apresentou-se um exemplo de hierarquia, tendo sido considerado um caso simples. De uma forma geral, a definição do modelo não é particularmente complexa. Por exemplo, considerando o videojogo *Prince of Persia*, é possível uma adequação ao jogo original recorrendo à definição de 6 categorias para agrupar os diversos tipos de blocos e 16 regras para definir o cálculo dos diversos trajectos possíveis para a personagem. Na Figura 15 é possível observar a janela de configuração das categorias e, na Figura 16, é apresentada a janela de definição das regras para a trajectória da personagem. Nessas figuras são apresentados os dados de teste utilizados no jogo *Prince of Persia*, conforme indicado acima.

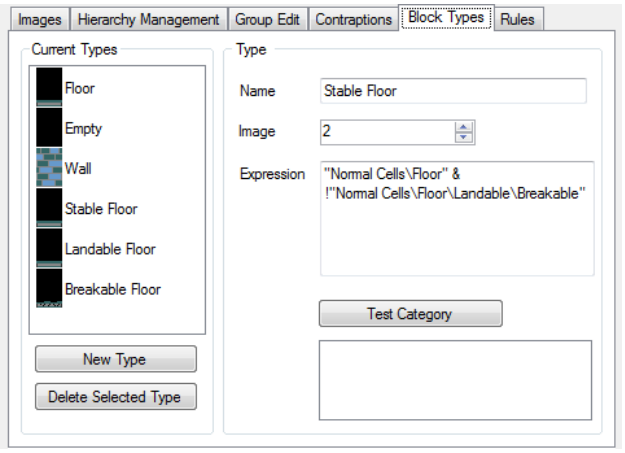

Figura 15 – Janela de edição do modelo de representação de níveis, separador de definição de categorias.

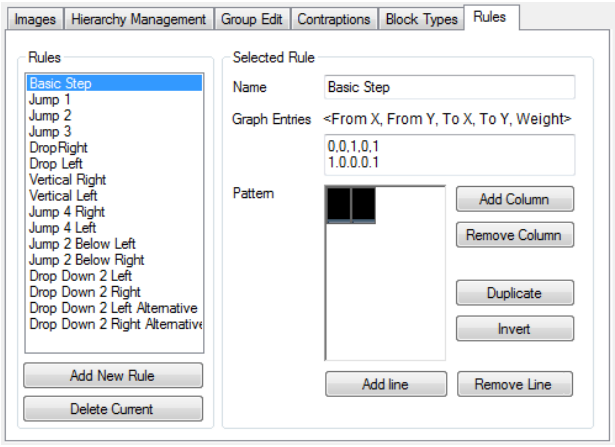

Figura 16 - Janela de edição do modelo de representação de níveis, separador de definição de regras.

5. CONCLUSÕES E TRABALHO FUTURO

No presente artigo descreveu-se um sistema integrado de suporte à criação de níveis para videojogos de plataformas, no qual foram implementadas diversas ferramentas de apoio à geração automática de conteúdo. Este poderá servir de base para a construção de um qualquer videojogo de plataformas, permitindo a definição de um modelo para os níveis e a respectiva edição dos mesmos, de forma manual ou automática.

O principal foco incidiu sobre a *framework* desenvolvida no âmbito deste projecto. Esta permite a representação do conteúdo de diversos jogos, tal como foi possível observar nos exemplos facultados. A maior limitação é, de facto, a acessibilidade ao conteúdo dos diversos jogos, o qual é tipicamente guardado em formatos proprietários. Para compreender melhor a capacidade de representação da referida *framework*, está prevista a representação do conteúdo de outros videojogos, nomeadamente dos títulos *Super Tux* e *Secret Maryo Chronicles*, ambos projectos livres e em que existe acesso à estrutura dos níveis. Um dos aspectos interessante a abordar como trabalho futuro é o mapeamento nesta *framework* do conteúdo de jogos que não possuam qualquer restrição a uma grelha, identificando possíveis perdas de informação e eventuais alterações possíveis para que estas não aconteçam.

A aplicação desenvolvida assenta numa arquitectura fortemente modular, pelo que é possível expandir os resultados aqui apresentados. Como foi referido, pretende-se expandir o conjunto de jogos suportados pela aplicação, sendo este processo facilitado pela referida modularidade da aplicação. Do mesmo modo, está prevista a expansão do conjunto actual de mecanismos de geração automática, nomeadamente com suporte a recolha de dados sobre utilizações de jogadores. Deste modo, pretende-se potenciar o desenvolvimento de algoritmos mais orientados à personalização. Novamente, a arquitectura modular do programa facilita estes potenciais desenvolvimentos.

Um dos *plugins* mais completos actualmente implementados assiste a integração com o videojogo *Prince of Persia*. Um aspecto interessante a constatar é que, para além de se conseguir representar os níveis na sua totalidade, existe uma maior expressividade da própria representação em relação ao jogo original. Por exemplo, a versão original do jogo divide o mapa em ecrãs isolados de 10 por 3 células, com um limite de 24 ecrãs e não permite a existência de mais do que um guarda por ecrã. Este tipo de limitações é comum em jogos antigos que, por escassez de memória, compactavam ao máximo a informação. Este tipo de limitação não existe dentro do sistema, pelo que, por exemplo, seria possível considerar mais do que um guarda por ecrã.

Finalmente, há a referir que se pretende lançar um projecto paralelo de desenvolvimento de um videojogo original de plataformas que tire proveito da *framework* e das ferramentas aqui apresentadas.

6. AGRADECIMENTOS

O trabalho apresentado tem sido parcialmente subsidiado pelo Instituto Politécnico de Setúbal sob a bolsa FCT/MCTES com a designação SFRH/PROTEC/67497/2010 e pelo CITI sob a bolsa FCT/MCTES com a designação PEst-OE/EEI/UI0527/2011.

7. REFERÊNCIAS

[Compton06] Compton, K., Mateas, M. 2006. Procedural Level Design for Platform Games. *Proceedings of the 2nd Artificial Intelligence and Interactive Digital Entertainment Conference*.

[Jennings-Teats10] Jennings-Teats M., Smith G., Wardrip-Fruin, N. 2010. Polymorph: Dynamic Difficulty Adjustment Through Level Generation. *Proceedings of the Workshop on Procedural Content Generation in Games*.
<http://doi.acm.org/10.1145/1814256.1814267>

[Mawhorter10] Mawhorter, P., Mateas, M. 2010. Procedural Level Generation Using Occupancy-Regulated Extension. *CIG-2010 - IEEE Conference on Computational Intelligence and Games*.

[Mourato11] Mourato, F., Próspero dos Santos, M., Birra, F. 2011. Automatic level generation for platform videogames using Genetic Algorithms. *ACE 2011, 8th international conference on advances in computer entertainment technology*.

[Mourato12] Mourato, F., Birra, F., Próspero dos Santos, M. 2012. Enhancing level difficulty and additional content in platform videogames through graph analysis. *ACE 2012, 9th international conference on advances in computer entertainment technology*.

[Pederson09] Pedersen, C., Togelius, J., Yannakakis, G. 2009. Modeling player experience in super mario bros. In *Proceedings of the 5th international conference on Computational Intelligence and Games* (CIG'09). IEEE Press.

[Shaker10] Shaker N., Togelius J., Yannakakis N., Weber B., Shimizu T., Hashiyama T., Soreson N., Pasquier P., Mawhorter P., Takahashi G., Smith G., Baumgarten R. 2010. The 2010 Mario AI Championship: Level Generation Track, *Special Issue of IEEE Transations on Procedural Content Generation*.

[Smith08] Smith, G., Cha, M., Whitehead, J. 2008. A framework for analysis of 2D platformer levels. *Proceedings of the 2008 ACM SIGGRAPH Symposium on Video Games*. Sandbox '08. ACM
<http://doi.acm.org/10.1145/1401843.1401858>

[Smith09] Smith, G., Treanor, M., Whitehead, J., Mateas, M. 2009. Rhythm-based level generation for 2D platformers. *Proceedings of the 4th international Conference on Foundations of Digital Games*
<http://doi.acm.org/10.1145/1536513.1536548>

[Smith10a] Smith, G., Whitehead, J. 2010. Analyzing the Expressive Range of a Level Generator. In *Proceedings of the Workshop on PCG in Games*.

[Smith10b] Smith, G., Whitehead, J., Mateas, M. 2010. Tanagra: A Mixed-Initiative Level Design Tool. *Proceedings of the 2010 International Conference on the Foundations of Digital Games*.

[Smith11] Smith G., Whitehead J., Mateas M., Treanor M., March J., Cha M.. 2011. Launchpad: A Rhythm-Based Level Generator for 2D Platformers. *IEEE Transactions on Computational Intelligence and AI in Games* vol. 3, issue 1.

[Viglietta12] Viglietta G. 2012. Gaming is a hard job, but someone has to do it. *6th International Conference on Fun with Algorithms*