Estudo de Caso no Desenvolvimento da Inteligência Artificial do Jogo Bonefighters

Ana C. R. S. Alves, Tiago V. Ficagna, Adriana G. Alves

Universidade do Vale do Itajaí, Design de Jogos e Entretenimento Digital, Brasil

acrsaw@gmail.com, tiago@univali.br, adriana.alves@univali.br

Abstract. *The purpose of this study is to present and evaluate the performance of the free artificial intelligence (AI) framework RAIN AI for the game engine Unity 3D, designed with the intent of streamlining the development of digital games. Aimed at independent developers, who almost always rely on very little time and financial resources, it is capable of abstracting much of the technical aspects of non-playable characters' (NPCs) AI creation, allowing the developer to focus on designing the behavior itself. Its efficiency when compared to other free available options will be demonstrated with an example gameplay situation utilizing the game Information Omitted, whose AI was entirely created using the RAIN AI framework.*

Resumo. *O objetivo desse estudo é apresentar e avaliar a performance da ferramenta gratuita de inteligência artificial (IA) RAIN AI para a engine Unity 3D, cuja finalidade é facilitar o desenvolvimento de jogos digitais. Voltada para desenvolvedores independentes, que quase sempre contam com pouquíssimo tempo e recursos financeiros, ela é capaz de abstrair muito da parte técnica da criação de IA de personagens não jogáveis (NPCs), permitindo que o desenvolvedor foque na elaboração do comportamento em si. A eficácia da ferramenta frente às outras opções gratuitas disponíveis será demonstrada com um exemplo de situação de gameplay, utilizando como base o jogo Informação Omitida, cuja IA foi feita inteiramente com a RAIN AI.*

1. Introdução

A criação de uma inteligência artificial em jogos digitais modernos é um processo quase sempre obrigatório; praticamente todos os jogos são planejados considerando a existência dela de alguma forma, podendo ela agir contra ou cooperativamente com os jogadores [Guevara-Villalobos, 2011][Buckland. 2005].

Foi com isso em mente que a empresa Rival Theory criou a RAIN AI, uma ferramenta feita para controlar a lógica por trás de comportamentos de inteligências artificiais, enquanto o desenvolvedor se preocupa apenas na elaboração do comportamento em si, utilizando os métodos de movimentação, árvore de comportamento, percepção, e capacidade de integração com a ferramenta de animação do próprio Unity 3D. Além de permitir fácil customização, ela procura fornecer uma boa performance em qualquer plataforma.

A RAIN AI é uma ferramenta dentre várias outras que podem ajudar o desenvolvedor. Muitas vezes, uma solução mais simples é o suficiente para um projeto menor, enquanto outros jogos pedem por uma inteligência artificial mais avançada.

Desse modo, o texto procura avaliar a performance da RAIN AI utilizando uma situação comum que utiliza IA em jogos, verificando em que aspectos ela é uma escolha superior à outras ferramentas. Para tanto, é utilizado o jogo *beat 'em up / dungeon crawler* Informação Omitida (Figura 1), criado como projeto de conclusão do curso de Design de Jogos e Entretenimento Digital da Universidade Informação Omitida. Ele é

co-op local para até quatro jogadores, possui uma *dungeon* semiprocedural, e uma variedade de inimigos e armas para serem utilizadas no combate. Utilizando de exemplo o comportamento de um inimigo desse jogo, serão mostradas as ferramentas e suas capacidades, analisadas o quanto ela ajuda e abstrai do desenvolvimento de comportamentos, e suas vantagens e desvantagens perante outros algoritmos e ferramentas. Com essas informações sendo apresentadas, espera-se que o leitor consiga decidir se a ferramenta é adequada para as suas necessidades no processo de criação de jogos. As ferramentas apresentadas são, além da RAIN AI, a Behavior Bricks, Panda BT, A* Pathfinding Project, e o NavMesh e NavMeshAgent do Unity 3D.

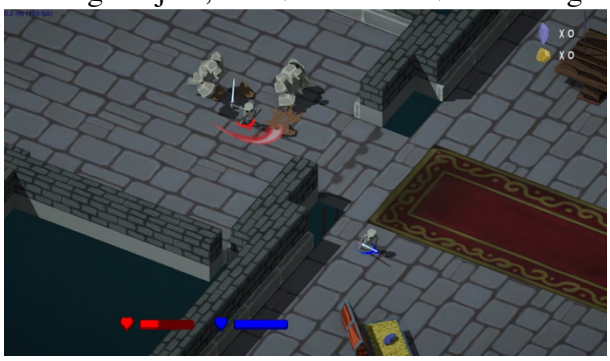

Figura 1. Imagem do jogo Informação Omitida, para Windows e Linux.

2. Metodologia

Para demonstrar as informações e análises desejadas, o jogo Informação Omitida será utilizado, exemplificando com um cenário do jogo que demonstre de maneira eficaz as funcionalidades de cada ferramenta. Para tanto, o seguinte caso de teste foi criado, representado na Figura 2:

Figura 2. Representação do caso de teste.

O caso de teste é uma simplificação do comportamento de um inimigo do jogo Informação Omitida, que fica *idle* enquanto o jogador não está próximo. Caso se aproxime, o inimigo persegue o jogador e o ataca. Se espera que o inimigo seja capaz também de desviar de obstáculos que possam estar no caminho, utilizando *pathfinding*. Se o jogador voltar a se afastar, o inimigo deve desistir e ficar *idle* novamente. Juntamente com a implementação desse cenário utilizando cada uma das ferramentas, serão descritos os passos necessários, aparência e facilidade de uso da interface visual, assim como qualquer outro tipo de fator relevante à suas vantagens ou desvantagens específicas.

3. Trabalhos Relacionados

Livros como Programming Game AI by Example [Buckland, 205] e Artificial Intelligence for Games [Millington. 2014] servem de fontes de algoritmos aprimorados para a criação de inteligência artificial em jogos eletrônicos, independentes do motor

utilizado. Essas técnicas são também utilizadas pela RAIN AI, que as abstrai e simplifica para o desenvolvedor.

O trabalho de Yannakakis (2012) explora métodos modernos de inteligência artificial, e comenta como a indústria ainda utiliza algoritmos antigos. Apresenta técnicas sofisticadas e emergentes, utilizadas por grandes estúdios de jogos, e que deverão ser devidamente aprimoradas e mais utilizadas em geral, criando experiências de inteligência artificial cada vez mais realistas e complexas.

4 Técnicas de Inteligência Artificial

A RAIN AI possui suporte para movimentação com *pathfinding* usando um *nav mesh* e uma árvore de comportamento com um sistema de sensores. Como o objetivo é comparar a eficácia da RAIN AI frente às outras ferramentas gratuitas concorrentes, apenas as técnicas que a RAIN AI possui suporte serão discutidas.

4.1. Árvore de Comportamento

Uma árvore de comportamento, como seu nome diz, é uma hierarquia que controla o fluxo de decisões e comportamentos de um objeto. Esses comportamentos são tarefas ou ações executadas pela inteligência artificial, que usa decisões para verificar qual comportamento executar de acordo com a situação de jogo.

Uma árvore de comportamento é composta por nós, que podem ser de ação, decisão, compostos ou decoradores. Nós de ação executam os comportamentos em si, que podem ser andar, dormir, atirar ou sentar, por exemplo. Já os de decisão são responsáveis por verificar se uma certa condição foi atingida, para então o nó de ação realizar o comportamento correspondente [Simpson, 2014]. Por exemplo, verificar se o jogador está próximo, se o NPC possui mana o suficiente para lançar um feitiço, ou se as luzes estão acesas. A Figura 3 a seguir é um diagrama representando uma árvore de comportamento simples, onde a entidade executa uma ação, verifica uma condição, e executa uma próxima ação de acordo com o resultado.

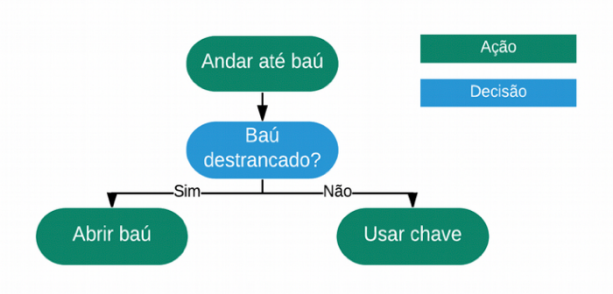

Figura 3. Diagrama de uma árvore de comportamento simples.

Nós compostos são aqueles que podem ter mais de um filho (nó logo abaixo na hierarquia), sendo que seus filhos podem ser avaliados sequencialmente, aleatoriamente, ou paralelamente. Um decorador só possui um filho, já que sua função é alterar algum valor, terminar ou pausar o processo, ou repeti-lo. Para a execução de nós compostos e decorativos, os filhos retornam, em geral, um de três valores: sucesso, fracasso, e executando. Os pais utilizam esses valores para avaliar e realizar uma decisão, alterando o comportamento do objeto.

4.2. Pathfinding e Movimentação

Quando objetos precisam se movimentar por um ambiente, desviando de obstáculos e chegando com sucesso ao seu destino, *pathfinding* é utilizado. Muito comum em jogos, *pathfinding* é o nome de uma solução que pode ser feita utilizando diferentes algoritmos

em jogos digitais. O mais comum é o A*, que se baseia em um *graph* específico do ambiente formado por nós ligados por conexões, por onde o objeto pode se locomover. Ele é simples de implementar, eficiente, e pode ser facilmente otimizado.

O algoritmo A* precisa de pelo menos dois nós: inicial e destino. Ele calcula o caminho utilizando uma heurística, que nada mais é que uma solução aproximada que funciona na grande maioria das situações, porém não em todas. Essa heurística estima o custo (distância percorrida) dos nós do *graph*. Com esses valores calculados, ele passa por nós de acordo com qual é mais provável que será o trajeto mais curto.

Utilizando A* e considerando a existência de um nó de inicio e um de destino, é garantido que um caminho será encontrado. Além disso, desde que bem desenvolvida, a heurística que ele utiliza é altamente eficiente, sendo que nenhum outro algoritmo utiliza essa heurística para achar um caminho ideal que examine menos nós que o A* [Cui, Xiao, and Hao Shi, 2011].

5. Estudo de Caso

As ferramentas que possuem funcionalidades semelhantes às da RAIN AI, que sejam também completamente gratuitas e disponíveis ao público, consideradas concorrentes diretas da RAIN AI são Behaviour Bricks, Panda BT e A* Pathfinding Project. Utilizando o comportamento descrito no item 2, vindo do jogo *beat 'em up / dungeon crawler multipayer* local Informação Omitida, o estudo de cada uma das ferramentas para comparação está a seguir:

5.1. RAIN AI

Utilizando a RAIN AI, há quatro passos gerais que se deve realizar: criar a árvore de comportamento, ajustar o componente de inteligência artificial no inimigo, marcar o jogador como uma entidade com quem o inimigo deve interagir, e criar o *nav mesh*. A árvore de comportamento é criada utilizando uma extensão do editor do Unity 3D, com elementos completamente visuais.

Conforme é mostrado na Figura 4, cada nó possui um indicador do seu tipo, com a indentação indicando seu nível na hierarquia. Ele permite a nomeação dos nós para facilitar a organização, e eles podem ser movidos individualmente ou em grupo dentro da hierarquia. Porém, não há como desfazer mudanças; caso o usuário não fique satisfeito com algo que acabou de mudar ele é obrigado a usar um *software* de versionamento ou refazer manualmente a árvore como ela estava antes. Ao rodar o jogo, a árvore mostra em qual nó ela se encontra, assim como o valor de retorno de todos pelos quais ela passou (sucesso, fracasso, executando).

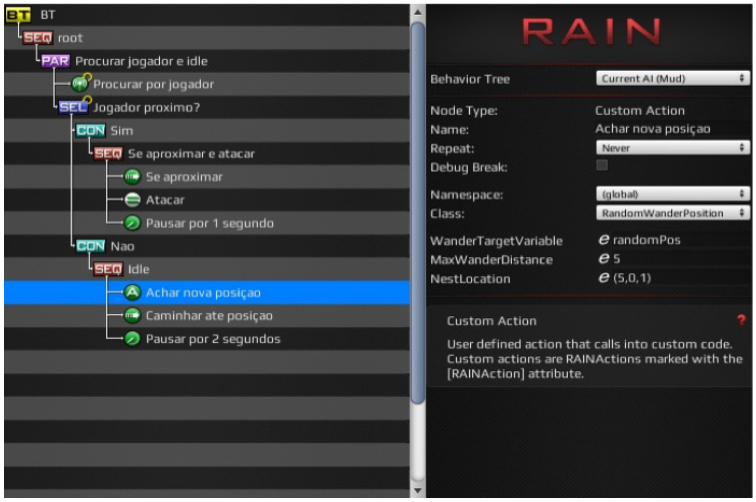

Figura 4. Árvore de comportamento utilizando a RAIN AI.

A detecção do jogador, a movimentação, e as animações são totalmente integradas com a RAIN AI. Ela possui um sistema próprio de detecção de objetos e áudio, assim como movimenta personagens e é capaz de utilizar esses valores com o Mecanim do Unity 3D.

É possível notar que utilizando a API da RAIN AI, é simples de ligar comportamentos padrões com personalizados e criados pelo desenvolvedor. Para auxiliar seus usuários, a RAIN AI conta com uma documentação em seu site, assim como um fórum dedicado e ativo.

O segundo passo é ajustar as configurações no próprio inimigo. Usando um componente já pronto da ferramenta, deve-se selecionar a árvore de comportamento correspondente ao inimigo e criar um sensor visual que detecta o jogador, mostrado na Figura 5. Para que a animação mude de *idle* para correr, foi usado o parâmetro "Speed", que já vem implementado com a RAIN AI.

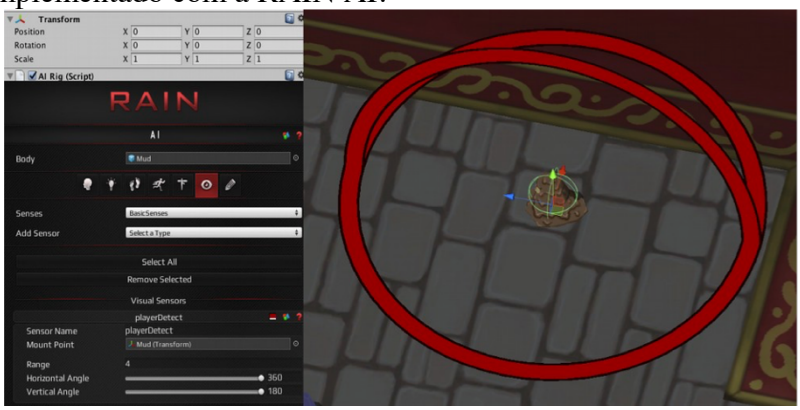

Figura 5. As configurações da inteligência artificial no inimigo e como é visualizado na cena de jogo.

O terceiro e mais simples passo é configurar o jogador como uma entidade com quem o inimigo irá interagir. Deve-se criar um componente de "Entity" e colocá-lo no jogador, de modo similar ao inimigo. As configurações também são realizadas de modo parecido, onde se cria um aspecto visual a ser detectado pelo sensor de IA.

Para o inimigo conseguir desviar de objetos e achar o caminho correto a seguir, a RAIN AI fornece um *nav mesh* próprio para *pathfinding*. Como os outros componentes, deve-se utilizar um objeto já previamente criado dentro da RAIN AI, o

Navigation Mesh, colocá-lo no centro e aumentar seu tamanho até que cubra todo o ambiente desejado, e gerar o *nav mesh*.

Por ser um objeto comum do Unity com um componente da RAIN AI, o Navigation Mesh pode ser aparentado com outro objeto na hierarquia. Desse modo, é possível fazer *prefabs* de *nav meshes,* e se o ambiente que possuir um Navigation Mesh como filho fosse deslocado durante a execução do jogo, o *nav mesh* se deslocaria junto, permitindo a reutilização e movimentação de ambientes, caso o jogo necessite. Além disso, ele permite a geração do *nav mesh* também durante o tempo de execução do jogo, auxiliando na criação de ambientes gerados proceduralmente. Dependendo das configurações, porém, a geração do *nav mesh* é um processo lento, e pode aumentar significativamente o tempo para carregar um ambiente procedural.

5.2. Behaviour Bricks

A Behaviour Bricks não possui seu próprio sistema de *pathfinding*, sendo que por padrão utiliza o integrado do Unity para sua movimentação. Caso o desenvolvedor deseje utilizar os comportamentos já criados da ferramenta, será necessário criar um Nav Mesh pelo Unity, assim como colocar um componente de Nav Mesh Agent no inimigo. Porém, como a Behavior Bricks permite a criação de comportamentos pelo usuário, é possível utilizar qualquer outra solução de *pathfinding* e movimentação, como a A* Pathfinding Project, caso o usuário deseje.

A construção da árvore de comportamento é através de uma interface específica, como na RAIN AI. Porém, como mostrado na Figura 6, seu visual é mais legível e sua edição é mais eficaz, por seus componentes serem maiores, facilmente modificáveis, e há a possibilidade de desfazer e refazer mudanças. Usando nós já prontos, o comportamento *idle* é simples de criar, pois a ferramenta já vem com nós para escolher uma posição aleatória e caminhar até um destino específico, assim como a detecção do jogador. O ataque, entretanto, é mais complexo, já que os nós que vêm junto com a ferramenta não possuem a funcionalidade de modificar diretamente parâmetros de animação. Também diferentemente da RAIN AI, ela não possui a capacidade de mostrar qual nóestá sendo executado e por quais ela passou durante o jogo.

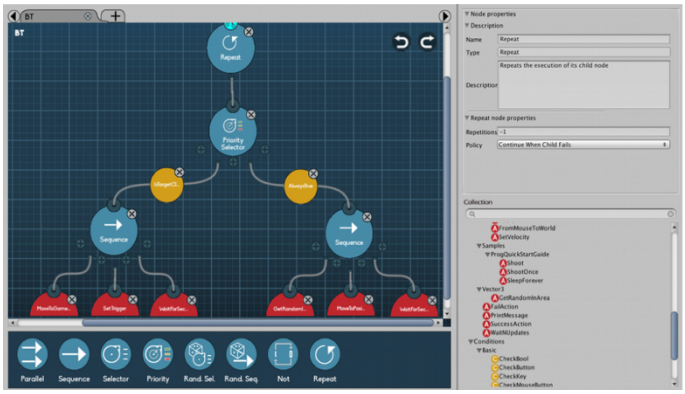

Figura 6. Janela de edição de árvores de comportamento.

Para a integração com o Mecanim do Unity, foi criado um *script* que utiliza APIs e métodos da Behaviour Bricks. Ele é ainda mais simples que o do RAIN AI, precisando de menos linhas de código para utilizar variáveis da árvore de comportamento. Apesar de a ferramenta vir com um guia de inicialização, nenhuma documentação foi encontrada no *site*, assim como fóruns ou qualquer outro tipo de guia ou auxilio.

O último passo é integrar o inimigo à arvore de comportamento. Basta adicionar o *script* Behavior Executor ao inimigo, e configurar os parâmetros necessários; nesse caso, um objeto que seja seu local inicial, com um colisor indicando a distância máxima que ele pode caminhar para seu comportamento *idle*, e o jogador, para ele detectar e perseguir. Diferentemente da RAIN AI, não é necessário fazer alteração alguma no jogador.

5.3. Panda BT Free

A Panda BT Free é a versão gratuita da Panda BT. Ela não possui algumas capacidades que a versão paga, como acesso ao código fonte e *debugging* com *break points*, mas nada que afete a capacidade de criar o comportamento desejado para a comparação. Diferente das alternativas, ela não possui um diagrama visual para a criação da árvore de comportamento; no seu lugar, ela utiliza uma linguagem de *script* própria, onde o desenvolvedor escreve o comportamento em um arquivo ".txt", como o mostrado na Figura 7. Com isso, ela possui uma documentação completa em seu *site*, porém não foram encontrados fóruns ou outra forma de auxilio específico para a Panda BT.

BT.txt

```
tree("Root")
    fallback
        tree("Attack")
        tree("Idle")

tree("Attack")
    sequence
        PlayerSeen
        Follow
        Attack
        RealtimeWait(1.0)

tree("Idle")
    sequence
        MoveToRandomPos
        RealtimeWait(2.0)
```

Figura 7. *Script* para criar a árvore de comportamento dentro da Panda BT.

Ela possui pouquíssimos comportamentos já criados, como o Wait, que pausa a execução pelo número de segundos ou *ticks* fornecidos. Os outros comportamentos, como MoveToRandomPos e PlayerSeen são implementados em um outro *script* em C# normal do Unity, usando a API da Panda BT. Suas funções devem começar com "[Task]", para identificar que é uma implementação de um comportamento da árvore.

Para a comunicação entre os *scripts* e com o objeto dentro do Unity, os dois arquivos devem ser colocados como componentes no inimigo. No inspetor, é possível ver a árvore de comportamento (Figura 8), que assim como a RAIN AI indica quais nós já foram avaliados e executados, assim como o nó que se encontra atualmente, durante a execução do jogo.

Figura 8. Inspetor mostrando a árvore de comportamento e o *script* com a implementação.

Assim como a Behavior Bricks, ela não possui seu próprio sistema de *pathfinding*, então para o exemplo foi utilizado o mesmo Nav Mesh e configurações de Nav Mesh Agent. Também, por ser fácil de criar comportamentos próprios, pode-se utilizar outra solução de *pathfinding* e movimentação, sendo que não é necessário utilizar o próprio do Unity.

5.4. A* Pathfinding Project

A ferramenta A* Pathfinding Project, diferente das outras apresentadas, não controla o comportamento do inimigo, e sim seu *pathfinding* e movimentação. Como a Behavior Bricks e a Panda BT não possuem seu próprio sistema de *pathfinding*, ele pode ser utilizado em conjunto com qualquer uma das duas ferramentas, substituindo o Nav Mesh e o Nav Mesh Agent do Unity 3D, ou em conjunto com a RAIN AI, substituindo o *nav mesh* que vem incluso com a ferramenta.

Primeiro, deve-se criar um objeto com o componente Astar Path. Nesse exemplo, foi então criado dentro do componente um Grid Graph, que gera, como o nome diz, um mapa com seus nós em formato de grade. Com o Graph ajustado com o tamanho, coordenadas e *layers* corretos, basta clicar em *scan*, que o Graph é gerado, como mostrado na Figura 9 a seguir.

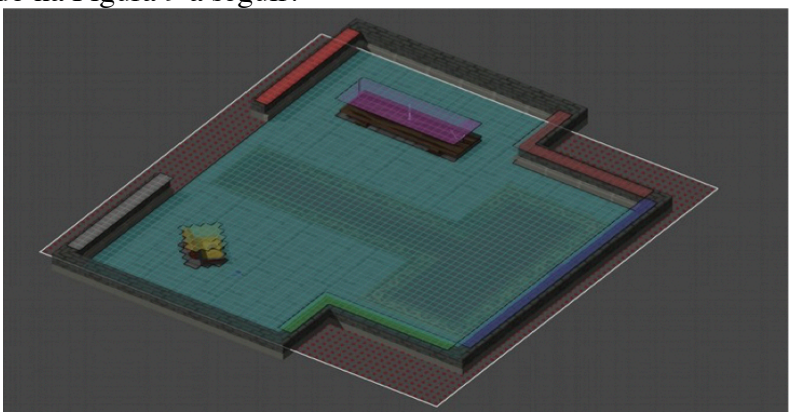

Figura 9. Grid Graph gerado pela A* Pathfinding Project.

Em seguida, um *script* chamado Seeker deve ser adicionado ao inimigo. Ele é responsável por calcular o caminho para se percorrer, e não precisa ser modificado. Para realmente movimentar o inimigo, outro *script* chamado AI Path, que vem junto da ferramenta, foi adicionado. Ele é responsável por apenas movimentar o objeto pelo caminho gerado, sendo que o desenvolvedor pode escrever seu próprio *script* para isso com o apoio do *site* que conta com uma extensa documentação com tutoriais, além de um fórum específico. Ainda, há o *script* Simple Smooth Modifier, que deixa o caminho gerado mais arredondado e natural. A Figura 10 mostra todos esses três *scripts* no inspetor.

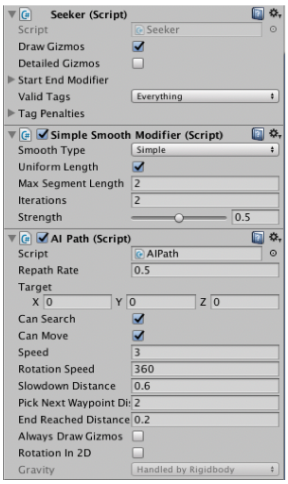

Figura 10. Os três *scrips* utilizados para *pathfinding* e movimentação.

Agora, para configurar a variável Target, ou seja, qual o alvo do inimigo, deve-se apenas editar o *script* que controla a árvore de comportamento para passar a sua variável de destino. Assim como a RAIN AI, o Grid Graph pode ser gerado durante o tempo de execução, permitindo ambientes gerados proceduralmente. Além disso, ele permite sua movimentação e outros tipos de alterações por código, sendo muito mais flexível que o *nav mesh* do próprio Unity. Ele não permite, porém, que se movimente ele através de um parente na hierarquia ou pelo seu próprio *game object*, sendo que é necessário modificar o componente do Grid Graph em si por código.

6. Conclusão

Considerando primeiro a parte de *pathfinding*, é possível notar que a ferramenta inclusa no Unity 3D é a menos eficaz e prática dentre as alternativas, por possuir grandes desvantagens. Por exemplo, se o objeto com Nav Mesh Agent tiver um *rigidbody*, ele deve ser marcado *kinematic* (ou seja, ele não pode ser movido pela física), já que ele contrasta com o Nav Mesh Agent. Além disso, o Nav Mesh não pode ser gerado em tempo de execução, devendo ser obrigatoriamente criado antes de rodar o jogo, assim como não pode ser movido junto com o ambiente. Desse modo, ele não possui as facilidades que as outras alternativas possuem, como a capacidade de aparentar o mapa na hierarquia, gerar em tempo de execução de jogo, e criar prefabs.

Levando em conta o comportamento como um todo, a RAIN AI é a única ferramenta capaz de integrar os dois aspectos da árvore de comportamento com o *pathfinding*. Para o jogo Informação Omitida, a RAIN AI foi escolhida justamente por o projeto necessitar de um comportamento que utilizasse extensamente *pathfinding*, sendo que seu mundo é criado proceduralmente. Além disso, o sistema de *pathfinding* da RAIN AI foi o mais simples de utilizar, necessitando menos passos que o A* Pathfinding Project.

Dependendo das preferências da equipe de desenvolvimento, ou também das necessidades de projeto, as outras ferramentas podem ser mais eficazes; se o projeto não necessitar de pathfinding, supõe-se que as outras ferramentas acabam sendo mais eficientes: a Behavior Bricks possui um diagrama visual mais atraente e rápido de se trabalhar, voltado à designers e artistas, enquanto a Panda BT foca em programadores e na criação de comportamentos próprios para o projeto, utilizando uma linguagem própria de *script* para controlar a árvore de comportamento.

Referências

Guevara-Villalobos, Orlando. "Cultures of independent game production: Examining the relationship between community and labour." *Proceedings of DiGRA 2011 Conference: Think Design Play*. 2011.

Buckland, Mat. *Programming Game AI by Example*. Jones & Bartlett Learning, 2005.

Mark, Dave. “Is this AI tool right for you? A RAIN{indie} review”. *Gamasutra*. Disponível em <http://www.gamasutra.com/view/news/190615/Is_this_AI_tool_right_for_you_A_RAI Nindie_review.php>, Abril 2013. Acesso em: Março 2017.

Millington, Ian. *Artificial Intelligence for Games*. Hoboken: CRC, 2014. Yannakakis, Geogios N. "Game AI revisited." *Proceedings of the 9th conference on*

Computing Frontiers. ACM, 2012.

van Waveren, J. M. P. "The Quake III Arena Bot." *University of Technology Delft* (2001).

Simpson, Chris. “Behavior trees for AI: How they work”. *Gamasutra*. Disponível em <http://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for _AI_How_they_work.php>, Julho 2014. Acesso em: Maio 2017.

Cui, Xiao, and Hao Shi. "A*-based pathfinding in modern computer games." *International Journal of Computer Science and Network Security* 11.1 (2011): 125-130.