UNIVERSIDADE DO EXTREMO SUL CATARINENSE - UNESC

CURSO DE CIÊNCIA DA COMPUTAÇÃO

BRIAN CASTELAN ANDRADE

AVALIAÇÃO DO ALGORITMO A* NA HEURÍSTICA DE PATH-PLANNING EM AMBIENTES DE JOGOS UTILIZANDO O MOTOR UNITY3D

CRICIÚMA

2015

BRIAN CASTELAN ANDRADE

AVALIAÇÃO DO ALGORITMO A* NA HEURÍSTICA DE PATH-PLANNING EM AMBIENTES DE JOGOS UTILIZANDO O MOTOR UNITY3D

Trabalho de Conclusão de Curso, apresentado para obtenção do grau de Bacharel no curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, UNESC.

Orientadora: Profa. Dra. Merisandra Côrtes de Mattos Garcia

CRICIÚMA

2015

BRIAN CASTELAN ANDRADE

AVALIAÇÃO DO ALGORITMO A* NA HEURÍSTICA DE PATH-PLANNING EM AMBIENTES DE JOGOS UTILIZANDO O MOTOR UNITY3D

Trabalho de Conclusão de Curso aprovado pela Banca Examinadora para obtenção do Grau de Bacharel, no Curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, UNESC, com Linha de Pesquisa em Inteligência Artificial.

Criciúma, 24 de junho de 2015.

BANCA EXAMINADORA

Profa. Merisandra Côrtes de Mattos Garcia – Dra. – (UNESC) - Orientadora

Prof. Paulo João Martins – MSc. - (UNESC)

Profa. Leila Laís Gonçalves- MSc. - (UNESC)

À minha família que tornou possível a realização do trabalho.

AGRADECIMENTOS

Agradeço aos familiares que me apoiaram durante minha vida e auxiliaram para que eu pudesse concluir o curso.

As pessoas que participaram deste período de estudos na universidade e me apoiaram nos momentos de dificuldade.

A minha orientadora Merisandra por me direcionar aos objetivos e incentivar a pesquisa.

“Nós podemos caminhar juntos, se nossos objetivos são os mesmos.”

Neil Peart

RESUMO

A indústria dos Jogos Digitais lança anualmente produções que contam com avançados recursos tecnológicos. A aplicabilidade destes recursos depende não somente da criatividade, mas da capacidade de aproveitamento de hardware, que provê o desenvolvimento destes recursos de maneira eficiente. Em um jogo, a performance do desenvolvimento de um recurso pode estar diretamente ligada à complexidade do ambiente em que o mesmo é implementado. Em determinados gêneros de jogo, o desenvolvimento destes recursos ocorre pela Inteligência Artificial, que por vezes, utiliza-se de técnicas que auxiliam na resolução de problemas implícitos. Uma destas técnicas é a busca pelo menor caminho entre dois pontos de um cenário, que envolve a atuação de um agente inteligente. Na resolução de problemas desta natureza, utiliza-se o algoritmo A*, que lida com a busca de menor caminho e popularmente presente nos jogos na forma de agente inteligente. A metodologia que implica na prática deste recurso, se dá por meio da liberação de *engines* gratuitas ao desenvolvedor independente, possibilitando a exploração dos mesmos. Utilizando uma *engine* é possível construir ambientes de jogo e manipular os recursos implementados. Nesta pesquisa, fez-se o uso da *engine* Unity3D na construção de cenários de jogo para implementação de um agente a fim de se avaliar o algoritmo A* codificado no agente por meio do monitoramento e da coleta de dados do seu desempenho referentes à navegação do A*, viabilizando comparações em relação aos recursos do ambiente. Dentre os ambientes testados, os resultados confirmaram as afirmações dos autores de que o desempenho de um agente é mensurada por meio da qualidade do ambiente navegado, estabelecendo que ambientes complexos tendem a afetar o desempenho de um agente.

Palavras-chave: Jogos Digitais. Inteligência Artificial. Algoritmo A*. Agentes Inteligentes. *Path-planning*. Unity3D.

ABSTRACT

The Digital Games industry yearly releases productions that have advanced technological resources. The applicability of these resources depends not only on creativity, but also on the hardware utilization capacity, which provides the development of these resources efficiently. In a game, the developing performance of a resource may be directly linked to the complexity of the environment in which it is implemented. In certain game genres, the development of these resources occurs by the Artificial Intelligence, which sometimes makes use of techniques that help in solving implicit problems. One of these techniques is the search for the shortest path between two points of a scenario, which involves the work of an intelligent agent. To solve problems of this nature, the A * algorithm is used, which deals with the search for the shortest path and popularly present in games in the form of intelligent agent. The methodology involved in the practice of this feature is through the release of free engines to independent developers, enabling their exploitation. Using an engine makes possible to build gaming environments and manipulate implemented features. In this research, the Unity3D engine was used in the construction of game backgrounds for implementation of an agent in order to evaluate the A * algorithm encoded in the agent by monitoring and gathering data on their performance related to navigation A *, enabling comparisons in relation to environmental resources. Among the tested environments, the results confirmed the statements of the authors of the performance of an agent is measured by the quality of the navigated environment, stating that complex environments tend to affect the performance of an agent.

Keywords: Digital Games. Artificial Intelligence. A* Algorithm. Intelligent Agents. Path-Planning. Unity3D

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

3D	3-Dimensional
FPS	Frames-per-second
GUI	Graphical User Interface
IA	Inteligência Artificial
NPC	Non-player character
RPG	Role Playing Game
UI	User Interface

SUMÁRIO

1 INTRODUÇÃO

O jogo é definido como uma atividade lúdica. Esta, é a prática de fantasiar a realidade do mundo dentro de determinados limites pré-estabelecidos. Desta forma, os limites respeitam as regras do jogo e transcendem fenômenos físicos ou reflexos psicológicos, gerando uma realidade virtual que serve de abrigo à vida real (LUCCHESE; RIBEIRO, 2014).

Tendo a realidade virtual proporcionada pela experiência do jogo como a atividade que é transportada ao aspecto psicológico do jogador, é que então se identificam todos os elementos essenciais para que esta experiência seja o mais satisfatória possível, pois os jogos digitais possuem uma abrangência somente limitada pela imaginação humana, uma vez que, em sua essência, um jogo serve como uma fuga da realidade. Encontram-se hoje jogos dos mais diversos modelos, desde simples interfaces de texto até os mais sofisticados gráficos, cada um com seu design particular,e com suas peculiaridades; um ramo onde os desenvolvedores possuem uma diversificação tão mista quanto os modelos de jogos presentes (MENDONÇA, 2012).

Atualmente vive-se em um meio digital que estimula o desenvolvimento de novas ideias, liberando-se *engines*[1] livres para o público. Este fato acarreta em uma evolução do tema *Indie* dos jogos, sendo este, um jogo que é desenvolvido de forma independente, ampliando as possibilidades de construção de jogos para fins pessoais e acadêmicos. Recentemente tornou-se possível o acesso a versões gratuitas de *engines* robustas como o Unreal Engine e o Unity3D, facilitando o uso de recursos como a Inteligência Artificial (IA), que dado o cenário atual, agrega funções vitais para o funcionamento de um jogo.

O acelerado desenvolvimento de novas ideias nos jogos digitais, é um exemplo de como a exigência em explorar novos recursos nos jogos força as empresas a procurarem um meio de aumentar a imersão e produzir

[1] Engine ou motor de jogo é um software que incorpora bibliotecas contendo elementos para o desenvolvimento de um jogo digital em tempo real (FUNGE, 2004, tradução nossa).

elementos gráficos em alto nível de realismo, aliados a uma IA funcional e inovadora (MENDONÇA, 2012).

A presença da IA nos jogos é estritamente conhecida pelas suas técnicas que abordam desde a aprendizagem de um Non-Player-Character (NPC), termo este atribuído a personagens não controlados pelo jogador, até funcionalidades intrínsecas do jogo, ou seja, que não estão visíveis ao jogador, mas atuam para o correto funcionamento do jogo (LANKOSKI; BJÖRK, 2006, tradução nossa).

A IA é reconhecida como instrumento essencial para proporcionar resultados evolutivos nos jogos, no entanto, para cada gênero a IA pode estar presente em diferentes aplicações. Como exemplo, têm-se a técnica de Máquinas de Estados Finitos presente em quase todos os gêneros existentes, porém, técnicas como a lógica fuzzy se fazem presentes apenas em jogos do gênero Aventura, enquanto jogos Role-Playing-Game (RPG) implementam o algoritmo A*, técnica esta que, é explorada e implementada intrínsecamente e estão diretamente ligadas à navegação de objetos no cenário (RABIN, 2015, tradução nossa).

É fato que, o surgimento da IA nos jogos veio a solucionar problemas conhecidos o suficiente para que se adotassem soluções padrão, porém, ao lidar com ambientes de vasta complexidade, a obsolescência das soluções se torna evidente, prejudicando o desempenho do jogo, e da Inteligência Artificial (RABIN, 2015, tradução nossa).

Desta forma, esta pesquisa busca implementar cenários dotados da perspectiva de visão e das características que remetem ao RPG, buscando vincular o algoritmo A* à um objeto que represente um personagem no dado ambiente, realizando um levantamento de resultados relacionados ao seu desempenho.

1.1 OBJETIVO GERAL

Analisar o algoritmo A* na heurística de *path-planning* em ambientes de jogo 3D.

1.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos são:

a) compreender a aplicação de técnicas A* em jogos;

b) modelar ambientes de jogo no motor Unity3D;

c) aplicar critérios para análise do desempenho e viabilidade do algoritmo A*;

d) determinar a padronização da heurística de *path-planning* no contexto;

e) analisar o desempenho e a viabilidade do algoritmo A* no ambiente simulado.

1.3 JUSTIFICATIVA

O uso da técnica A* para implementação dos sistemas de jogo parte do princípio de que historicamente, o A* é tido como o primeiro algoritmo planejador, e a popularidade de sua aplicação na heurística[2] de *path-planning* é devido a simplicidade e a habilidade de encontrar o melhor caminho ainda que em um ambiente composto por inúmeros obstáculos (GERAERTS, 2010).

Conforme a evolução dos jogos, novas tendências de aplicação de algoritmos de IA como o A* surgiram juntamente com a necessidade de se expandir a aplicabilidade de agentes inteligentes em diferentes cenários (MACHADO, PAPPA, CHAIMOWICZ, 2012, tradução nossa). Uma das consequências desta evolução, foi o surgimento de ambientes de jogo robustos, dotados de recursos gráficos e físicos complexos, aumentando significativamente a necessidade de se implementar um agente que ao mesmo tempo possa atender à demanda de recursos, e obter um rendimento aceitável (MACHADO; PAPPA; CHAIMOWICZ, 2012, tradução nossa).

[2] No dado contexto, a heurística é o método explorado, ou os recursos utilizados para a resolução do path-planning. Neste caso, fez-se o uso de um agente inteligente.

Logo, ressalta-se que uma das estratégias mercadológicas utilizadas pelas empresas de desenvolvimento de jogos, é liberar versões gratuitas de suas *engines* para desenvolvimento independente, elevando o potencial de descoberta de novos conceitos e soluções para seus jogos. Utilizando-se deste meio, torna-se viável a utilização de uma engine gratuita como o Unity3D para o desenvolvimento de cenários complexos, tendo ao alcance a integração com ferramentas que facilitam o desenvolvimento de um jogo completo.

Motores como o Unity3D são capazes de dar suporte a criação de objetos 3D, e scripts que o vinculam, garantindo a presença da heurística de path-planning no ambiente.

Desenvolver ambientes de jogo juntamente com um agente inteligente capaz de navegar no cenário, auxilia na identificação e distinção do uso de recursos referentes a estrutura do ambiente, e do agente A*.

1.4 ESTRUTURA DO TRABALHO

O trabalho é dividido em seis capítulos principais. O capítulo inicial faz uma introdução identificando o surgimento dos jogos eletrônicos e faz um breve levantamento sobre a IA nos jogos.

No segundo capítulo, são apresentados conceitos gerais sobre o desenvolvimento de jogos, os recursos que o implementam, bem como as divergências teóricas entre os gêneros e as variáveis do gênero RPG no contexto do desenvolvimento de ambientes.

No terceiro capítulo, a abordagem é sobre o *path-planning,* este que, é um recurso implementado nos jogos sob determinadas heurísticas que fundamentam a sua aplicação para a resolução de problemas de busca de caminho. Além disto, é feita uma abordagem acerca do algoritmo A*, popular por ser o responsável no planejamento de rota de um agente.

No quarto capítulo, discute-se sobre os motores de jogo, estes que são os responsáveis na manipulação e organização dos recursos de jogo, bem como auxiliam no desenvolvimento de ambientes 3D. Em adicional, faz-se um breve resumo sobre o panorâma geral do motor Unity3D e suas características

próprias.

No quinto capítulo têm-se os trabalhos correlatos, nestes, são feitas abordagens de interesse e relativos aos assuntos aqui tratados. As pesquisas apresentadas neste capítulo, abordam a presença da IA em outros gêneros de jogo, além de outras formas de implementação que acrescentam conhecimento na área.

O sexto capítulo está o trabalho proposto, neste, é explicado o que será explorado e estudado de acordo com a metodologia. Nela, é apresentado todas as etapas de desenvolvimento do trabalho que consiste no desenvolvimento de ambientes 3D para a aplicação de um agente A*. Em seguida, é feita a coleta de dados referentes ao desempenho do agente nos ambientes, e a comparação dos resultados na presença e na ausência do agente.

2 JOGOS DIGITAIS

De uma atividade lúdica, à um contexto que engloba produções milionárias pela indústria, os jogos digitais atingiram um patamar que abrange inúmeros campos de estudo, impulsionados pela sua própria evolução e contribuição em diferentes áreas da tecnologia de informação (SCHUYTEMA, 2008). Este potencial tecnológico, provém da composição de um jogo eletrônico como um sistema que é caracterizado pela interface, os motores do jogo, e o enredo (BATTAIOLA, 2000). O enredo retrata o tema do jogo, o desenvolvimento de sua trama juntamente com seus objetivos, estes que, são apresentados sequêncialmente ao longo do progresso do jogo. O motor do jogo, envolve mecanismos de controle das respostas do ambiente às ações e decisões tomadas pelo jogador, e de acordo, efetuar as modificações de estado no ambiente. Por fim, a interface interativa possibilita a interação do jogador com os motores de jogo, atuando como mediador, e facilitador, através de respostas audiovisuais referente às mudanças de estado no ambiente (BATTAIOLA, 2000).

O conceito de um jogo leva em conta diversos aspectos, que identificam e definem o gênero, assim como a obra no geral. O desenvolvimento de jogos segue uma metodologia de processos que é subdividida em equipes organizadas com enfoque em cada processo que compõe o jogo, ao final resultando em uma produção que integra diferentes recursos tecnológicos avançados em um único ambiente pronto para ser explorado pelo jogador (NOVAK, 2011).

2.1 DESENVOLVIMENTO DE JOGOS

A área de desenvolvimento de Jogos Digitais sofreu alterações significativas ao longo da última década. Tais alterações, são provenientes da acelerada evolução de hardwares, que envolvem desde processadores e interfaces gráficas, até os motores de jogos que foram incrementados e disponibilizados aos desenvolvedores independentes, acarretando no alto

índice de crescimento da comunidade que desenvolve jogos (RABIN, 2012).

O panorama brasileiro já não difere do internacional. Empresas brasileiras passaram a ter destaque internacional no desenvolvimento em vários níveis. Conhecidos como consumidores secundários do produto final, os brasileiros emergiram do consumo para a produção. Atualmente, existem diversos estúdios de criação de jogos independentes, além do surgimento de cursos de especialização voltados inteiramente para a área de desenvolvimento de jogos (RABIN, 2012).

2.1.1 Pré-produção

Assim como em um filme, aa pré-produção é a etapa que inicia o desenvolvimento da ideia conceitual do jogo, também nomeado de roteiro. Considerado o principal ponto de enfoque para a construção do jogo, o roteiro é o responsável pela aproximação ou identificação do jogador com as características que irão definir a experiência imersiva (NOVAK, 2011).

Por se tratar de um meio que exige interação direta do jogador com os controles de jogo, o primeiro contato do jogador se da por meio da apresentação dos controles principais do jogo, antecedendo a apresentação do problema principal (NOVAK, 2011).

O diferencial na criação do roteiro do jogo está principalmente na liberdade do desenvolvedor de, seguindo um tema específico, criar inúmeras possibilidades de ações em paralelo à história principal (NOVAK, 2011).

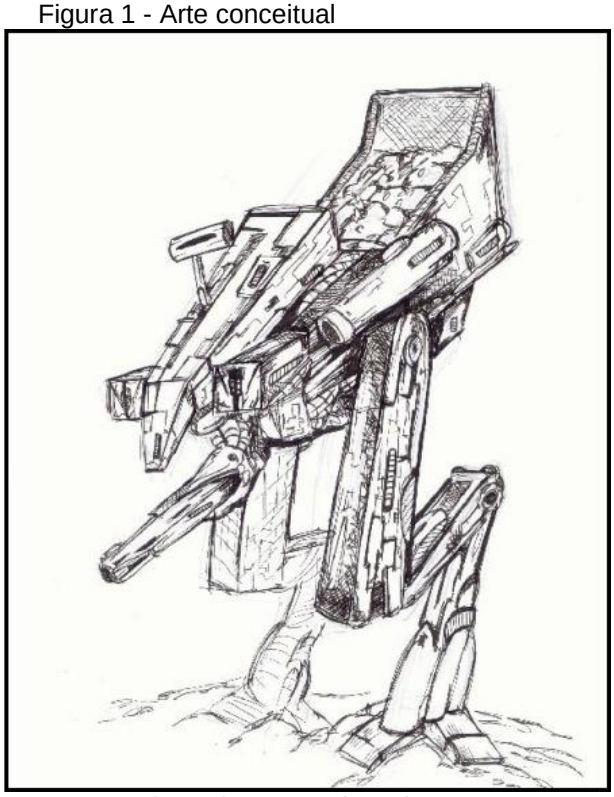
2.1.2 Game Design

Este que, é um termo frequentemente confundido com conceitos artísticos, o Game Design de um jogo engloba diversos aspectos relacionados à história, jogabilidade, personagens e ambientes. No entanto, o design de um jogo envolve desde a criação das narrativas, diálogos, até os processos de criação de mapas, níveis e por fim, a *User Interface* (UI) que determinará o método de interação do jogador com os controles de jogo (DUARTE, 2012). A

User Interface, determina não somente os controles físicos ao qual o jogador se comunicará com o jogo, mas também o método de representação da informação, podendo ela ser sonora, ou através de informações em forma de texto exibidas na tela. Com base nisso, o jogador estará à parte de informações relevantes ao jogo no dado instante (NOVAK, 2011).

O design se estende somente até os níveis de criação conceitual, envolvendo também a parte gráfica (DUARTE, 2012), como exemplo a figura 1 ilustra a criação da arte conceitual.

Figura 1 - Arte conceitual



Fonte: Clua, Bittencourt (2005).

2.1.3 Interface Gráfica

Intrinsecamente, tornou-se comum nos jogos, a prática da interface gráfica ser projetada a partir de um hardware personalizado. Como efeito, têm-se a renderização de objetos 3D de alta complexidade (RABIN, 2012). No entanto, a criação de uma interface gráfica de qualidade, para o jogador significa a identificação com o jogo, e consequentemente, estar a par da

situação que lhe é apresentada (NOVAK, 2011). A interface gráfica envolve a manipulação de elementos como o ambiente gráfico, os objetos que o compõe, e as texturas, que permitem à personalização estética das superfícies (RABIN, 2012).

2.1.3.1 Ambientes 3D

O ambiente é o espaço que dispõe o ambiente físico do jogo. O ambiente físico é definido pelas características da perspectiva de visão, escala de tamanho, limites, estruturas, terrenos, objetos e texturas (NOVAK, 2011).

Na criação destes ambientes, os responsáveis criativos iniciam o processo de criação com um esboço relacionado a cena de jogo, para posteriormente utilizar um editor de ambiente 3D específico para a modelagem (NOVAK, 2011). A figura 2, ilustra a criação de um ambiente 3D em perspectiva de visão isométrica, juntamente com a aplicação de texturas.

Figura 2 - Ambiente 3D

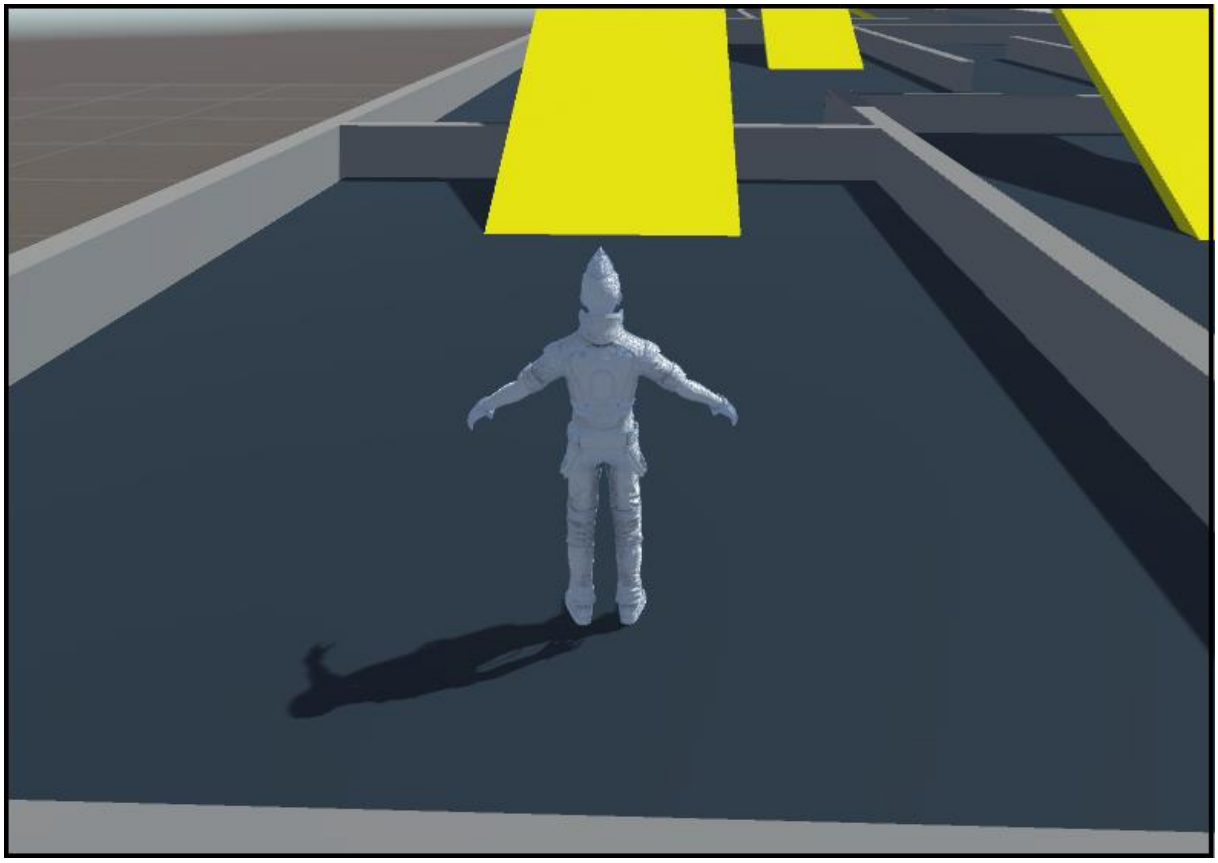
Fonte: Do autor.

2.1.3.2 Objetos 3D

Os objetos presentes em um ambiente de jogo, são dispostos a partir de um modelo geométrico, este modelo, é implementado com diferentes finalidades. Para definir sua finalidade, dois tipos de modelagem são definidos, os elementos dinâmicos e os elementos estruturais (RABIN, 2012). Os elementos estruturais definem qualquer objeto estático no ambiente, como exemplo, a figura 2 apresenta a criação de montanhas, objetos estes gerados a partir da criação de terrenos por uma ferramenta que permite a criação de objetos estáticos. Já os elementos dinâmicos representam objetos que possuem algum tipo de comportamento (PASSOS et al, 2009).

Os motores de jogo atuais, distinguem estas duas classificações automaticamente, proporcionando ao desenvolvedor criar objetos com funcionalidades específicas através do uso de um componente que é vinculado ao objeto (PASSOS et al, 2009). Como exemplo, a figura 3 demonstra um objeto dinâmico. Este objeto, foi criado baseado no modelo de um personagem, sendo este, vinculado a um componente que implementou um comportamento inteligente de busca de menor caminho no ambiente de jogo. No entanto, os objetos no qual o personagem navega como o plano e as rampas são objetos estáticos.

Figura 3 - Objeto dinâmico



Fonte: Do autor.

2.1.4 Física

A modelagem de objetos 3D em um cenário de jogo, sejam eles de quaisquer natureza, são inúteis sem a definição de leis da física. As leis da física em um jogo podem ser exploradas de diferentes maneiras, seguindo diferentes regras (FERNANDES et al, 2009). Em um jogo que contenha a presença de personagens por exemplo, estes devem caminhar sobre um plano sólido, para que não se percam nas coordenadas do espaço, seguindo esta linha, não somente os personagens, mas todos os objetos criados devem respeitar o espaço ocupado. Um objeto jamais poderá atravessar uma barreira sem que haja colisão (ADAMS, 2014, tradução nossa).

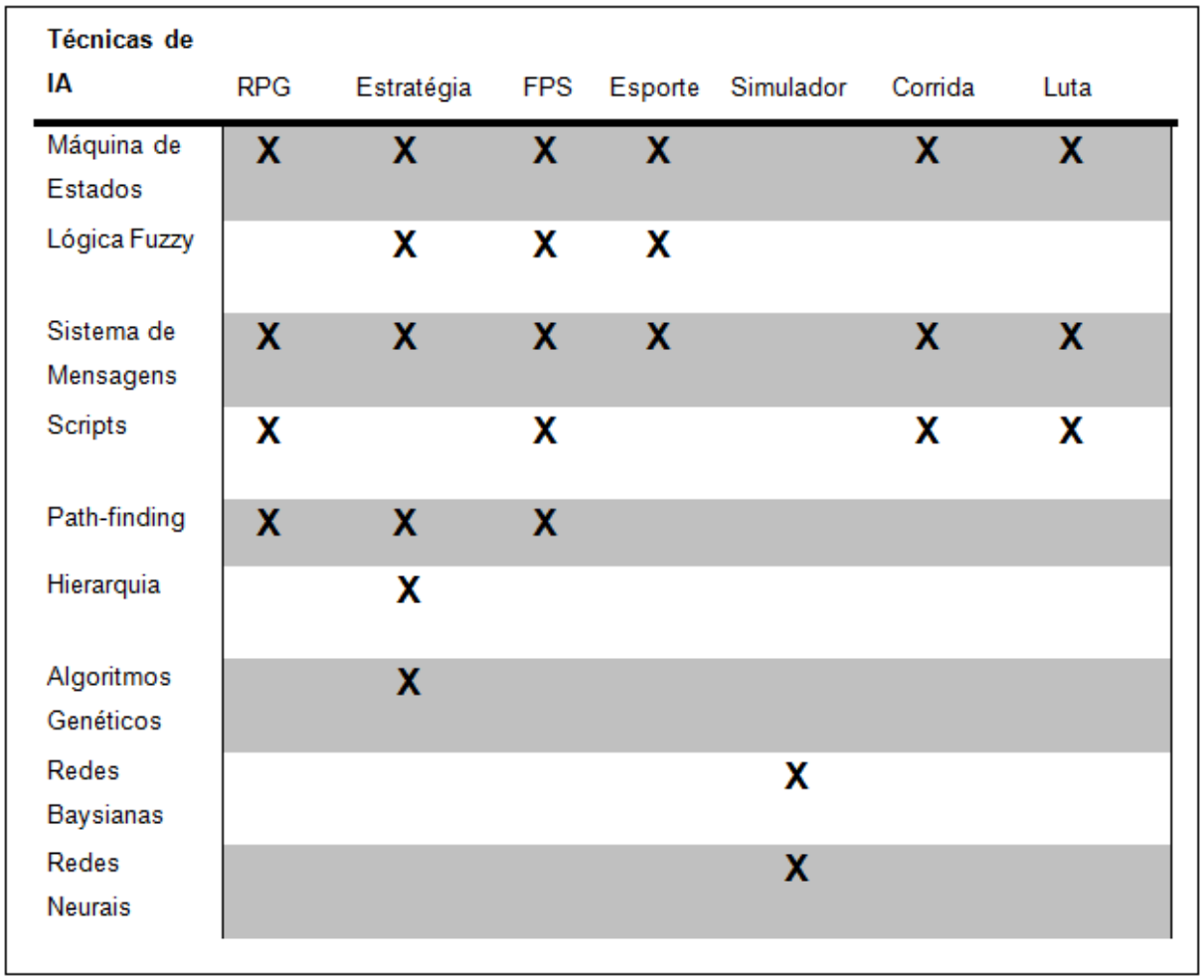
Os sistemas de física podem ser tratados pelos motores de jogo de maneira dinâmica e com certa diversidade de comportamentos. Um exemplo para o dado recurso, é a atribuição de diferentes comportamentos para um mesmo objeto, por exemplo um personagem, pois sua movimentação para cada membro deve ocorrer de maneira independente (PASSOS et al, 2009).

O conceito primitivo que trata da física nos jogos são as colisões. As colisões são implementadas em forma de um componente que é anexado ao objeto, definindo sua solidez. Geralmente, as colisões tomam o mesmo formato do objeto geometricamente, entretanto, as ferramentas responsáveis pela criação das colisões, possuem recursos para alterar o formato de colisão do objeto, facilitando a personalização do comportamento dos objetos (PASSOS et al, 2009). Atualmente, os motores de jogo oferecem suporte a utilização de bibliotecas especifícas para o tratamendo de colisões, bibliotecas famosas como o PhysX trazem consigo diversos recursos altamente personalizáveis para controle de colisões (PASSOS et al, 2009).

2.1.5 Inteligência Artificial

A IA nos jogos é o conjunto de parâmetros e regras que trabalham o design do jogo para que este seja funcional no ambiente programado (RABIN, 2015, tradução nossa). A IA pode fazer parte de diferentes recursos do jogo, como a preparação dos diálogos do personagem em determinado momento, o comportamento dos personagens (sejam eles NPC’s ou controlados pelo jogador), além dos métodos de interação com o jogo em sí (RABIN, 2015, tradução nossa). Estas interações só são possíveis por meio do uso das técnicas de IA, que se fazem presentes massivamente e englobam diversos gênros. A figura 4 apresenta as técnicas de IA e os respectivos gêneros que as implementam.

Figura 4 – Técnicas de IA por gênero



Técnicas de IA	RPG	Estratégia	FPS	Esporte	Simulador	Corrida	Luta
Máquina de Estados	X	X	X	X		X	X
Lógica Fuzzy		X	X	X			
Sistema de Mensagens	X	X	X	X		X	X
Scripts	X		X			X	X
Path-finding	X	X	X				
Hierarquia		X					
Algoritmos Genéticos		X					
Redes Baysianas					X		
Redes Neurais					X		

Fonte: Adaptado de Vieira (2005).

Visto que um dos aspectos de suma relevância no desenvolvimento do jogo é a técnica de IA utilizada, é que define-se a necessidade do seu uso de acordo com a tecnologia empregada. Neste caso, após a definição dos recursos que o jogo implementará, fica por conta do desenvolvedor definir a técnica de IA que será empregada para cada caso (RABIN, 2015, tradução nossa).

As várias técnicas de IA se fazem presentes nos jogos geralmente com um objetivo em comum, a resolução de problemas. Então, define-se o método que deverá ser utilizado para a resolução em questão (HORSWILL, 2014, tradução nossa).

No entanto, deve-se ressaltar que a identidade de cada jogo é única, possibilitando a prática da mistura de técnicas diversas afim da resolução em

um dado contexto de jogo (RABIN, 2012).

2.2 GÊNEROS

Discenir os gêneros dos jogos digitais, é uma tarefa que deve-se considerar alguns critérios que os identificam, citando o objetivo do jogo, o contexto do jogo ou o seu tema, além do método de condução do personagem (CRAWFORD, 2005).

De acordo com estes critérios de identificação do jogo, os gêneros são divididos da seguinte forma (BATTAIOLA, 2000):

a) estratégia: jogos voltados para as habilidades cognitivas, ou seja, são baseados em raciocínio e decisão;
b) simuladores: jogos que dotam de uma representação física complexa, para que a imersão lembre a vida real;
c) aventura: gênero este que combina habilidades de raciocínio e psicomotoras simultaneamente, os enigmas são implícitos visando uma progressão contínua;
d) infantil: jogos voltados para crianças. São baseados em contos de estórias e resolução de quebra-cabeças simples que geralmente tragam alguma recompensa estimulante à criança;
e) passatempo: gênero mais recente que envolve a resolução de quebra-cabeças simples e de rápida solução. São desprovidos de um enredo elaborado e considerados como casuais;
f) RPG: versões digitais de contexto complexo dos jogos tradicionais de representação de papéis. A trama geralmente é longa e leva o jogador a desafios que exigem o planejamento da evolução das habilidades do seu personagem;
g) esporte: é uma representação dos esportes da vida real;
h) educacionais: jogos que podem estar contidos em quaisquer dos grupos acima, porém objetivando o ensino, com fortes critérios didáticos e pedagógicos.

É inevitável que jogos digitais sejam classificados em mais de uma categoria, uma vez que a classificação é feita por meio da sua característica mais explícita. Nos tempos atuais, o surgimento de jogos com múltiplos gêneros é comum, uma vez que a exigência da comunidade por inovações tende a crescer, e assim, força os produtores a tentarem novas formas de representação de um jogo (BATTAIOLA, 2000).

Categorizar os gêneros de jogos é uma tarefa que leva em conta os aspectos práticos do jogo. Estes aspectos consideram questões como a narrativa, o estilo de progressão do jogo, as perspectivas de visão e de controle das entidades, além do objetivo principal. A partir disto, é possível discenir os gêneros conforme estes aspectos.

Os jogos de ação por exemplo, tem enfoque na jogabilidade, habilidade e reflexos do jogador. Logo, é comum explorar este estilo em uma perspectiva de visão em primeira pessoa, uma vez que esta, melhor representa a visão de uma pessoa no mundo real. Jogos como o Call Of Duty exploram um ambiente de guerra, apresentando ao jogador desafios relacionados à precisão e movimentação, já no Grand Theft Auto, a ação se dá por meio de um mundo livre e uma perspectiva em terceira pessoa (CRAWFORD, 2005).

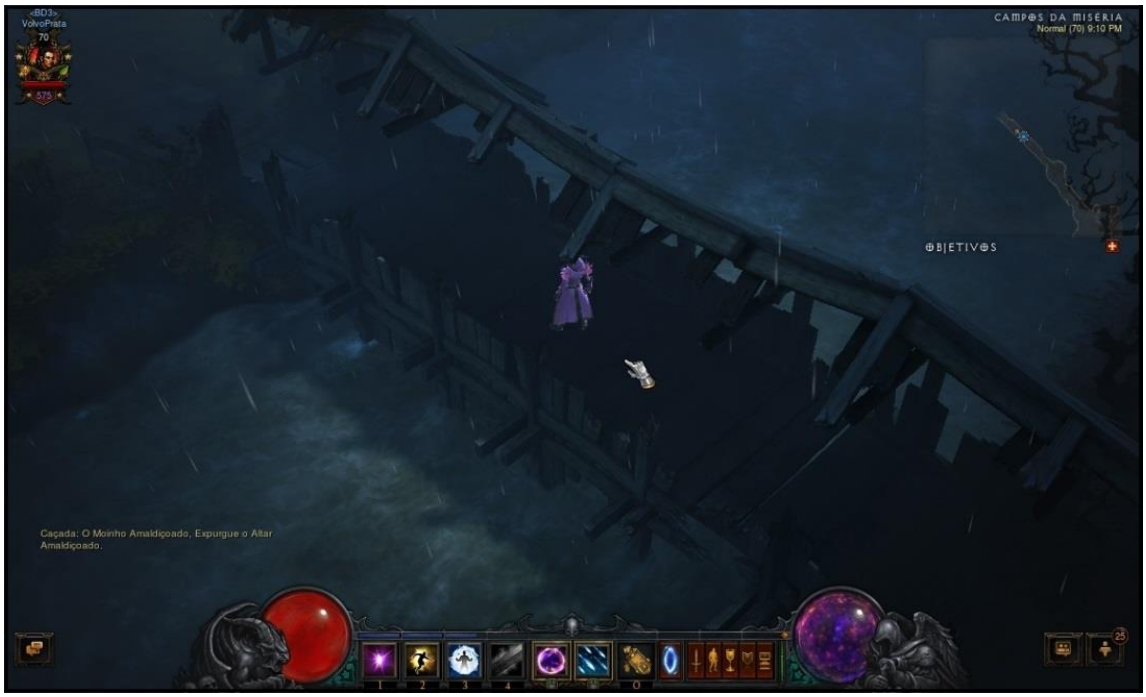
Um dos gêneros que melhor demonstra a mistura entre características de diferentes gêneros, é o RPG, pois em sua essência, o RPG é uma representação da crônica e da narrativa (PEREIRA; BETTOCCHI, 2013). Logo, a diversidade de cenários, personagens e progressão paralela é um dos fatores que contribuem para tornar o gênero complexo, valorizando a otimização dos recursos e das tecnologias empregadas (CARR; BURN, 2006).

2.2.1 As variáveis do gênero Role-Playing-Game no contexto prático

A abordagem no contexto prático diz respeito a criação propriamente dita. Assim como na identificação do gênero, também há algumas características que ditam o método de implementação do ambiente de jogo, no caso do RPG, uma das variáveis do ambiente é a pespectiva de visão. A Figura

5 demonstra a perspectiva de visão isométrica característica de jogos do gênero RPG.

Figura 4 - Perspectiva isométrica



Fonte: Blizzard Entertainment (2012).

A partir da mudança de perspectiva de visão, novos elementos surgem como variáveis do ambiente, estas que, podem se revelar no formato de objetos estruturais. Os objetos estruturais geralmente fazem parte do design gráfico do ambiente, logo, estes podem ser relacionados à obstáculos que devem ser trespassados no caso da atuação de um personagem (RABIN, 2012).

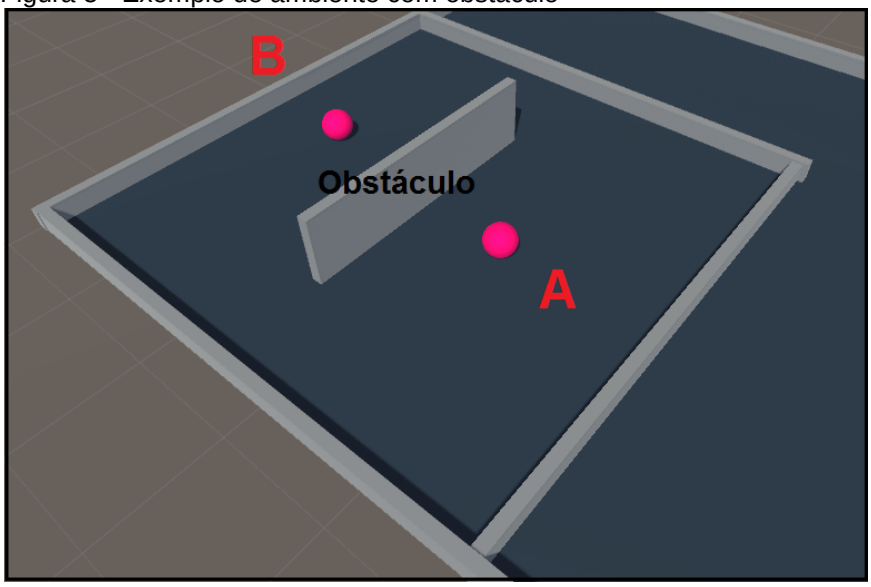
Tendo em vista que uma das técnicas de IA exploradas no gênero é o *path-planning*, este que, visa a resolução de problemas de busca de menor caminho, a referida técnica torna-se vital para a solução em ambientes do gênero em destaque.

3 PATH-FINDING

A resolução de um problema de planejamento de caminho, inicia com a formulação do objetivo final da rota. A identificação do objetivo, se dá por meio do processo da formulação dos problemas, este que, é baseado nas ações, regras e estados considerados relevantes para a solução. (RUSSELL, NORVIG, 2013; WELD, 1999, tradução nossa). O path-finding representa a solução de um caminho no ambiente de jogo. Entretanto, sua composição envolve uma estrutura complexa de tarefas, pois além da busca de um caminho navegável, suas aplicações inferem em considerar as variáveis do ambiente que se distinguem pela análise do ambiente, obstáculos, cálculos de trajetória, identificação de quaisquer entidades que influenciam na busca, além da detecção de colisão (RABIN, 2012).

No contexto dos jogos digitais, os algoritmos de busca de menor caminho, atuam para resolução para ambientes perturbados, ou seja, a presença de elementos físicos nos ambientes faz com que a função dos algoritmos seja de evitar a colisão e atingir o seu destino com sucesso. Para exemplificar, a figura 6 mostra dois pontos distintos, no qual a esfera A é o ponto de origem, e a esfera B o destino final. Para simular a presença de um elemento físico que reflita as situações típicas de cenários dos jogos, o Obstáculo sugere um determinado espaço que não é transitável, trazendo a impossibilidade de navegação por quaisquer pontos que o obstáculo esteja incluso (RABIN, 2012).

Figura 5 - Exemplo de ambiente com obstáculo



Fonte: Do autor.

3.1 HEURÍSTICAS DE PATH-FINDING

Dar início às heurísticas, implica em definir o termo de maneira compreensiva. No entanto, no ramo dos Jogos Digitais seu significado é explícito, pois as heurísticas lidam com a busca de uma resolução que tem maior probabilidade de sucesso (LUGER, 2014). Para isto, existem duas considerações acerca da definição das heurísticas de path-planning:

a) um problema nem sempre possui uma solução exata. Esta ocorrência se dá pelo fato do problema possuir ambiguidades relacionadas a formulação do mesmo, ou nos dados disponíveis;
b) no caso do problema possuir uma solução exata, tipicamente suas soluções podem dotar de um custo computacional inviável. Geralmente esta ocorrência está ligada ao crescimento exponencial do número de espaço de estados para alcançar seu objetivo (LUGER, 2014).

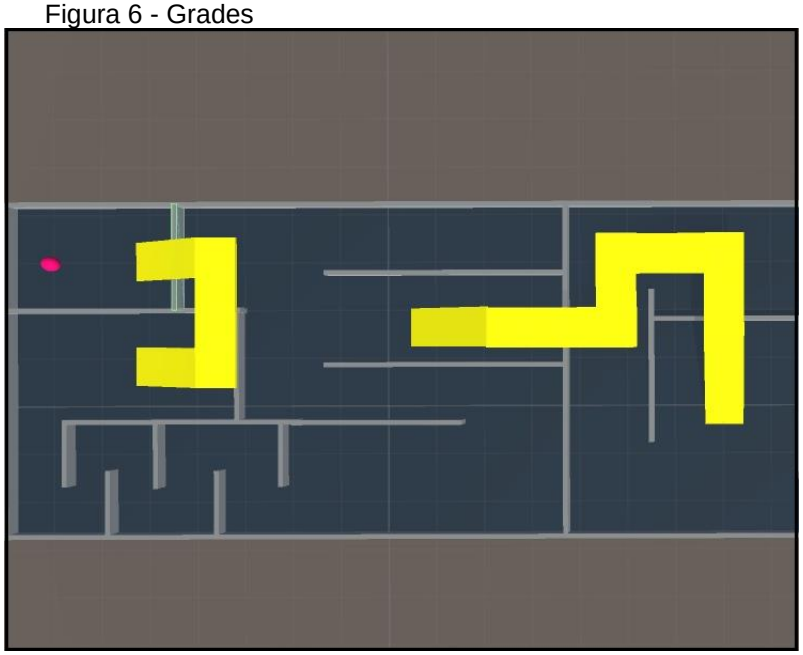
Algumas técnicas que minimizam a complexidade estrutural do *path-planning* levam em conta desconsiderar informações irrelevantes para a navegação. Reforçando sua definição, as heurísticas de *path-planning* são assim definidas pelo uso de técnicas que levam a resolução de um problema

desconhecido (RABIN, 2015, tradução nossa). O nível de complexidade e informações do ambiente afeta diretamente no desempenho do *path-planning*. Para que haja a otimização da navegação de quaisquer entidades, estas técnicas auxiliam no desenvolvimento do próprio ambiente, encorajando a aplicação do *path-finding* (RABIN, 2012).

3.1.1 Grades

Esta técnica representa a projeção do ambiente de forma que todo o plano de navegação é dividido em grades de alinhamento. A grade incorpora toda a estrutura do ambiente, bem como todos os objetos. Uma de suas vantagens práticas, é a composição de células alinhadas permitindo com que sua posição seja facilmente referenciada. A figura 7 ilustra um plano de jogo com as células baseadas no conceito de grades (RABIN, 2012).

Figura 6 - Grades



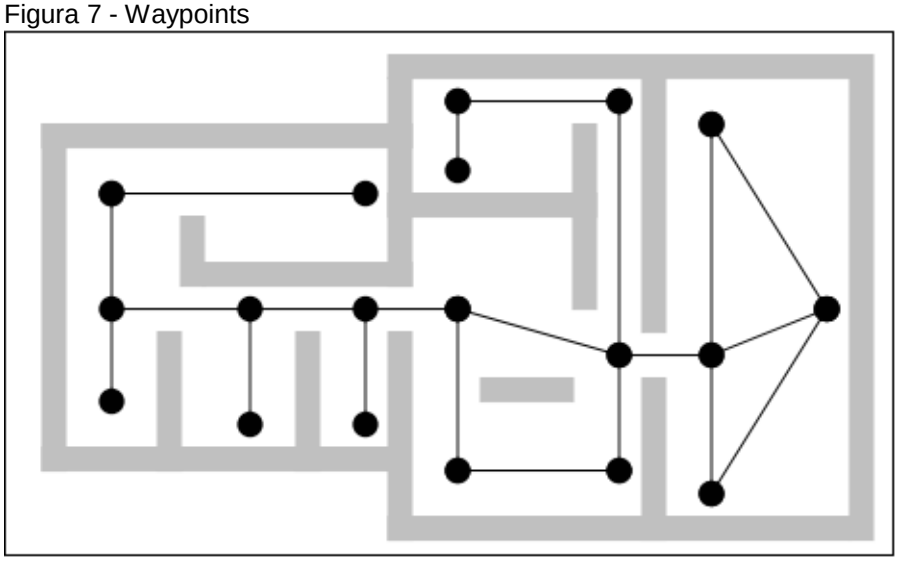
Fonte: Do autor.

3.1.2 Waypoints

O sistema de representação do ambiente através de *waypoints*, define por meio de vértices, os locais navegáveis do mapa. A obstrução do

caminho percorrido por um obstáculo ou a presença de objetos estáticos não afetará o desempenho da navegação, uma vez que os vértices são definidos apenas em áreas de total segurança de navegação (RABIN, 2012). A figura 8 ilustra como esta técnica de path-finding pode ser representada no contexto prático.

Figura 7 - Waypoints



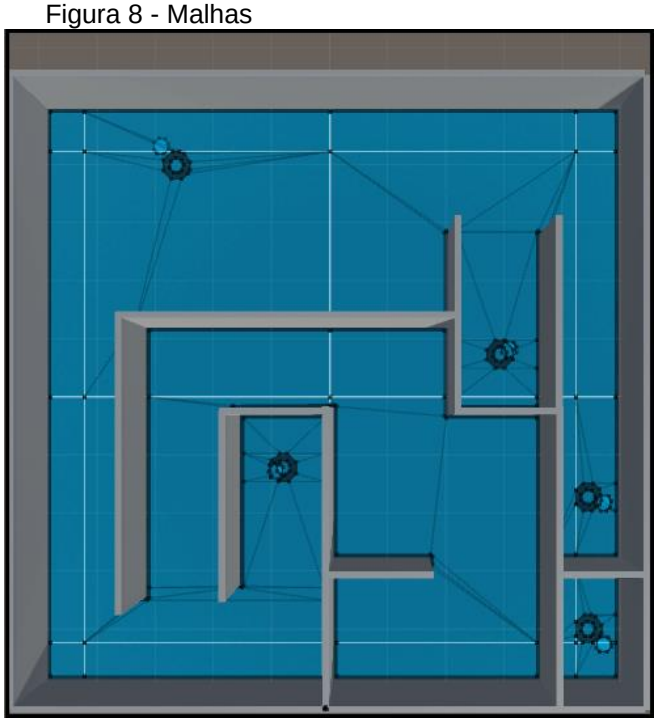
Fonte: Do autor.

3.1.3 Malhas

A mais popular e eficiente técnica de representação do ambiente de jogo, no que diz respeito as áreas de navegação, são as malhas. No âmbito teórico, as malhas são representações de polígonos que tem suas arestas compartilhadas, ou no caso de estarem dispostos nos limites de uma área navegável, não possuem compartilhamento de aresta (RABIN, 2012).

Esta técnica, é considerada a mais eficiente na representação de um plano navegável, pois abrange todos os limites navegáveis do mapa, ao contrário das representações de grades e waypoints, que se limitam a definir locais de acesso seguro, por vezes se mostrando incompletos em determinadas estruturas de ambiente (RABIN, 2012). A figura 9 demonstra a resolução do plano navegável com base na projeção de malhas.

Figura 8 - Malhas



Fonte: Do autor.

Visto as técnicas que definem as áreas navegáveis de um ambiente, o *path-finding* demonstra o acuidade e a sensibilidade no tratamento dos ambientes para o sucesso do seu objetivo, a busca de um caminho. No entanto, este objetivo só é alcançado quando entidades podem identificar e lidar com as técnicas que implementam os ambientes de jogo. Tais entidades desenvolvem o papel de atuadoras, em outras palavras, são os agentes inteligentes que reconhecem o plano, definem suas ações, e atuam (FUNGE, 2004, tradução nossa).

3.2 AGENTES INTELIGENTES

Na IA, um agente inteligente é uma entidade capaz de observar e agir em determinado ambiente e direcionar suas ações para alcançar um objetivo (RUSSELL; NORVIG, 2013). Normalmente, agentes controlados por IA também são chamados de *bot* (termo mais comum em jogos de ação em primeira pessoa) ou Personagem Não-Jogável, do inglês *Non-Playable Character* (NPC).

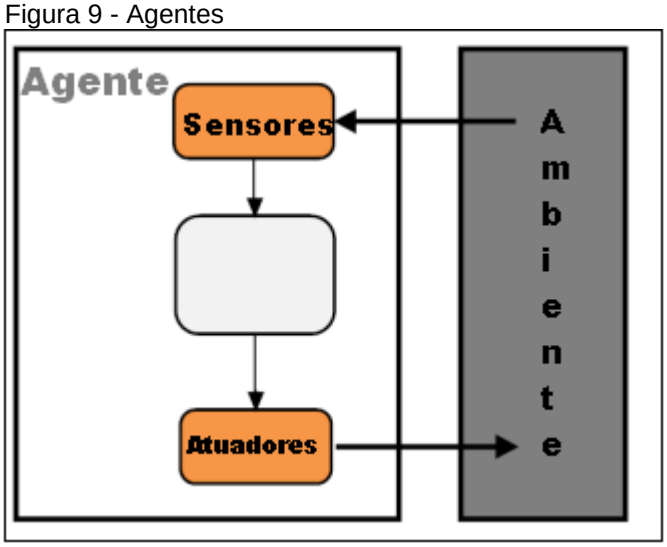
A organização dos agentes segue um ciclo teórico que envolve as etapas de reconhecimento, pensamento e ação. Tais etapas, são tidas como base de conhecimento para que o programador crie o seu comportamento (RABIN, 2012). Agentes inteligentes são considerados elementos evolutivos, então define-se estes como sistemas inteligentes capazes de tomar decisões com certa autonomia e flexibilidade visando um objetivo (WOOLDRIDGE, 1998). Já Russell e Norvig (2013) definem como agentes, tudo aquilo que atua num ambiente de maneira intermediada, a partir de seus sensores. No contexto dos Jogos Digitais, os sensores são identificados pelas ações ou comandos que o jogador executa dentro do jogo.

As principais abordagens para definir um agente inteligente, são:

a) a resposta do agente de acordo com as informações coletadas do ambiente;
b) o conceito de reatividade, na qual os agentes devem alterar seu comportamento de acordo com a mudança de estado do ambiente;
c) a sociabilidade, na qual envolve os esquadrões, pois os agentes devem sempre que possível atuar em conjunto com outros agentes ou jogadores humanos afim de completarem seus objetivos.

Agentes inteligentes podem ser estruturados de diferentes maneiras de acordo com a solução. A maioria das soluções se da na presença de agentes de rotina, responsáveis pela captura e comunicação da informação coletada, agentes de busca que tratam de informações importantes de busca, agentes aprendizes que determinam as regras de acordo com a informação obtida, e os agentes de decisão que executam as tarefas com base nas regras, ou formularam uma nova (LUGER, 2014). Uma das percepções que devem ser consideradas com relação ao agente inteligente, é que sua funcionalidade em um jogo por completo, é invisível e irrelevante para o jogador com uma única exceção caso o resultado trazido ao jogador seja demonstrado em forma de ações (RABIN, 2012). Para melhor ilustrar este conceito, a figura 10 demonstra o esquema organizacional dos agentes perante os ambientes de jogo.

Figura 9 - Agentes



Fonte: Adaptado de Kyaw, Peters e Swe (2013).

Mencionadas as formas de solução de um agente inteligente, ressalta-se que no contexto dos Jogos Digitais, sua popularidade de aplicação se dá a partir de resoluções baseadas na busca pelo menor caminho, logo, unindo-se as técnicas de IA, o comportamento dos agentes tem sua base de criação, no algoritmo A*, técnica mais popularizada e difundida no âmbito dos Jogos Digitais por apresentar soluções eficientes de busca dado a complexidade das áreas de navegáveis em um plano de jogo (RABIN, 2012).

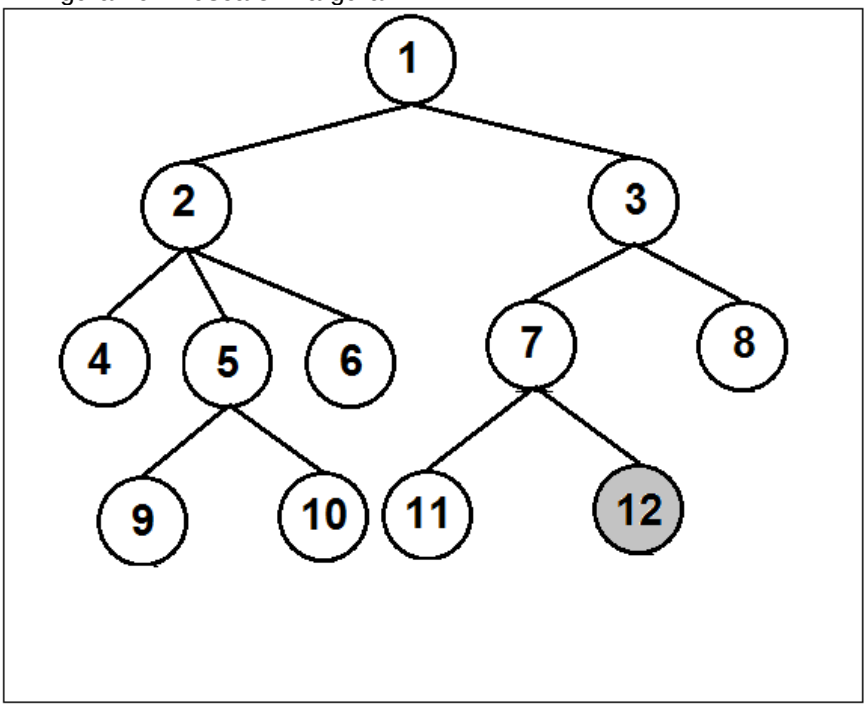
3.2.1 Técnicas de Busca

As técnicas de busca, servem de alicerce para identificar a melhor heurística em determinado contexto. Pois a partir da lógica de busca, obtem-se o retorno da aplicação prática da busca por meio dos resultados computacionais (LUGER, 2014). Nos ambientes de jogo, a busca deve suprir uma margem de eficiência na navegação, para tanto, existem diferentes técnicas de busca que devem ser analisadas.

3.2.1.1 Busca em Largura

A busca em largura, representada pela figura 11, mostra cada nó, contendo o número da ordem de busca. Fica visível que, a busca em largura é feita baseando-se no conceito de níveis, ou seja, somente após o algoritmo percorrer um nível por completo ele irá progredir para o próximo nível e assim continuar a varredura até que encontre o seu destino (LUGER, 2014).

Figura 10 - Busca em largura

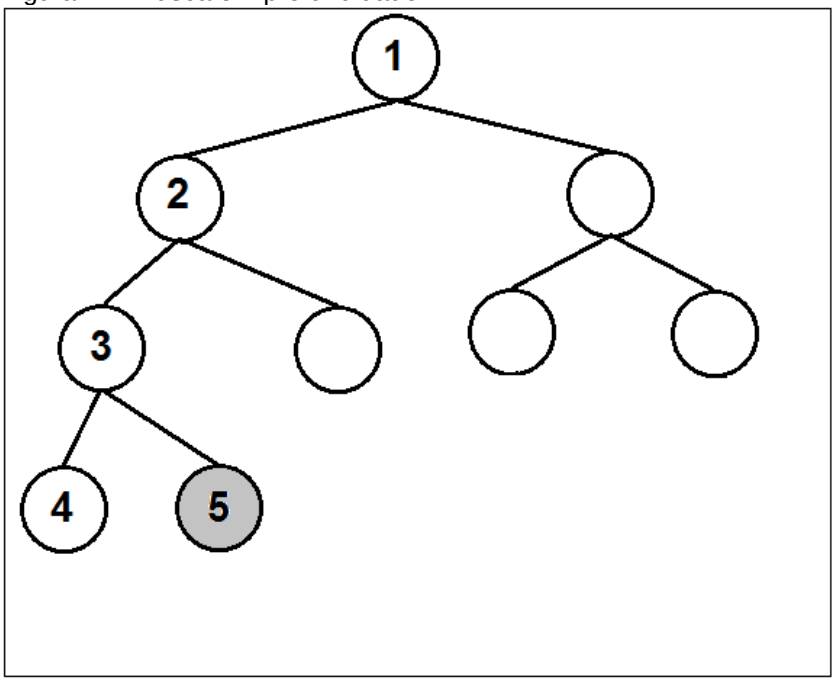


Fonte: Adaptado de Luger (2014).

3.2.1.2 Busca em Profundidade

Já a busca em profundidade considera outra abordagem heurística, esta pode se mostrar mais eficiente que a busca em largura para casos de poucos níveis, tendo garantia no processo de busca, estes algoritmos são exemplos não dotados de heurísticas de planejamento, em outras palavras, não há implementação de código inteligente provido de análise por nó. A figura 12 ilustra a busca em profundidade, no qual tende a se mostrar mais eficiente em comparação ao algoritmo de busca em largura para alguns casos, portanto, sua aplicação deve ser analisada, uma vez que a busca em profundidade se torna ineficiente em casos de busca em vários níveis (LUGER, 2014).

Figura 11 - Busca em profundidade



Fonte: Adaptado de Luger (2014).

3.2.1.3 Adaptações dos Métodos de Busca

Além dos métodos de busca apresentados, alguns outros métodos foram adaptados com o intuito de estabelecer soluções para problemas de natureza progressiva e regressiva. Outra variação interessante, é um método de busca que divide suas metas em sub-metas, possibilitando a adaptação da meta inicial à uma nova meta que melhor satisfaça a busca em dado momento da execução.

Estas adaptações dos métodos de busca são assim descritas:

a) busca Progressiva: esta busca, é uma variação baseada em uma sequência finita de ações visando que o estado inicial atinja a meta. Em outras palavras, o estado inicial passa a ser a meta. Este que, é considerado um dos métodos de busca mais antigos e ainda populares devido principalmente a sua efiência, historicamente dando origem a heurísticas admissíveis;

b) busca Regressiva: basicamente o contrário da busca progresiva. Porém nesta, o algoritmo inicia do estado da meta, visando atingir o estado inicial. Este que, é considerado o método de busca mais delicado de se lidar, pois a cada iteração, há possibilidade de desfazer uma meta já completa do ponto em que se encontra;
c) busca de Ordem Parcial: outro conceito de busca, pois não segue o mesmo método de busca linear. Este método decompõe uma meta em n metas, sub-divididas em grau de importância e complexidade. Em seguida, as metas são executas numa sequência lógica que satisfaça a melhor busca no dado momento de execução. Outra característica deste método, é que a meta inicial não apresenta pré-requisitos para que seja executada, porém as metas sub-divididas interferem no planejamento da meta inicial (RUSSELL; NORVIG, 2013).

3.3 ALGORITMO A*

Navegar em um plano de jogo requer não somente um ambiente dotado de técnicas de *path-planning*, mas também pelas variáveis de atuação, conhecidas como agentes inteligentes. Estes devem usar um algoritmo de busca como o A* para possibilitar a implementação das técnicas de busca.

A abordagem heurística característica dos algoritmos A*, visa o aumento do desempenho computacional, levando em conta que as soluções são específicas de determinado problema (RABIN, 2015, tradução nossa). Como estes algoritmos são geralmente utilizados para resolução de problemas referentes à busca de menor caminho, são comumente baseados em grafos (LUGER, 2014).

A organização do planejamento é baseado nos métodos de busca, logo, para cada situação, um método de busca alternativo tem possibilidade de ser executado (RUSSELL; NORVIG, 2013).

Sendo assim, será evidente que fundamentos dos grafos estarão presentes nas explanações sobre algoritmos de busca, visto que estes implementam técnicas consideravelmente comuns e de básica execução para dar origem a implementações mais complexas (RUSSELL; NORVIG, 2013).

Outros conceitos de grafos são relacionados à realidade de um ambiente de jogo, considerando que obstáculos e áreas não-navegáveis aparecem com frequência.

O A* atua com base na identificação de vértices em um grafo. Inicialmente o processo considera os vértices sendo atribuídas duas variáveis principais que são relativas à distância do vértice inicial até o vértice destino (Sgoal), e a estimativa da distância do vértice atual até o vértice destino (h) (RABIN, 2015, tradução nossa). Esta disposição, é mantida para cada vértice navegado pelo A*. O A* ainda contém duas estruturas adicionais, nomeadas de *Open List* e *Closed List*. A *Open List*, é uma fila de prioridade que armazena os vértices que devem ser considerados para a progressão da busca. A *Closed List*, armazena o conjunto de vértices que já foram expandidos na progressão da busca (RABIN, 2015, tradução nossa). A continuação se dá pela atualização das variáveis g e parent (s) que armazenam estes valores baseados na distância até um vértice adjacente alcançável considerando o caminho do vértice inicial s = g(s) , e o caminho total armazenado de (s) até (s'). O algoritmo atualiza a variável g e parent (s) caso o valor deste caminho seja menor que a distância do vértice inicial (s) até o vértice encontrado até a última iteração (s'). Na figura 13, é apresentado um pseudo-código do algoritmo A* demonstrando que o vértice de origem é declarado na *Open List* com distância nula. Logo, a primeira iteração inicia-se do vértice de origem. Analisando o código, nota-se que para cada vértice (g), adota-se o vértice que armazena a distância percorrida até o momento, representada pela variável (s'). Para cada iteração, atualiza-se o novo custo armazenado na variável iniciada sem valor, para que então seja feita a comparação entre custos. O vértice que apresenta menor distância de navegação é alocado na *Closed List*, para que seja comparado somente com relação ao custo dos vértices adjacentes à ele. A

análise se encerra quando não há mais custos de distância dos vértices na *Open List* a serem armazenados, ou quando o vértice de destino é encontrado.

Ressalta-se ainda, que este pseudo-código é desprovido de quaisquer técnicas de pré-processamento, estas que, devem ser analisadas com cautela de acordo com o ambiente específico para que tornem a busca de caminho do algoritmo A* mais eficiente, e de menor custo computacional.

Figura 12 - Pseudo-código do algoritmo A*

```
1  Main()
2      open := closed := ∅;
3      g(S_start) := 0;
4      parent(S_start) := S_start;
5      open.Insert(S_start, S_start) + h(S_start));
6      While open ≠ ∅ do
7          s := open.Pop();
8          if s = s_goal then
9              return "path found";
10         closed := closed ∪ {s};
11         foreach s' ∈ neighbor_vis(s) do
12             if s' ∉ closed then
13                 if s' ∉ open then
14                     g(s') := ∞;
15                     parent(s') := NULL;
16                 UpdateVertex(s, s');
17     return "no path found";
18 end
19 Update Vertex(s,s')
20     g_old := g(s');
21     ComputeCost(s, s');
22     if g(s') < g_old then
23         if s' ∈ open then
24             open.Remove(s');
25         open.Insert(s', g(s') + h(s'));
26 end
27 ComputeCost(s,s')
28     /* Path 1 */
29     if g(s) + c(s,s') < g(s') then
30         parent(s') := s;
31         g(s') := g(s) + c(s,s');
32 end
```

Fonte: Rabin (2015).

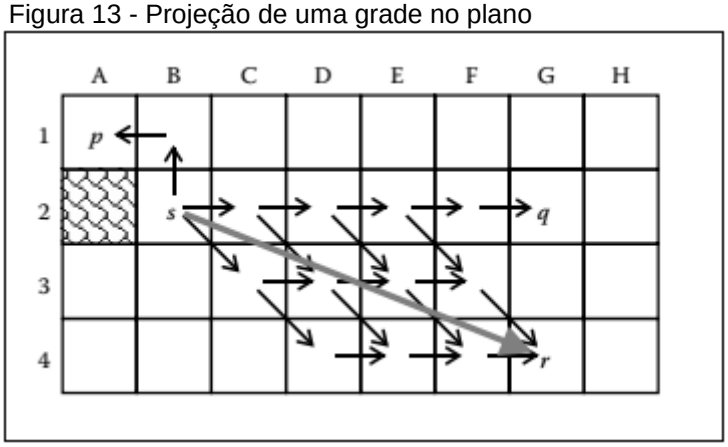
3.3.1 Pré-processamento

Nos jogos, o agente que implementa o algoritmo A* atua nos ambientes com base nas projeções de vértices sobre um plano horizontal. Para tanto, os vértices não se dispõe de maneira aleatória, e sim são criados seguindo conceitos de grafos (RABIN, 2015, tradução nossa).

O Pré-processamento do ambiente utiliza este conceito de grafos com a projeção de uma grade de células afim de identificar as células que contém quaisquer objetos que possam obstruir o caminho entre o ponto de origem e o destino. Desta forma, o pré-processamento otimiza o desempenho de busca do algoritmo A* (RABIN, 2015, tradução nossa). Esta técnica é de suma importância para viabilizar a prática da busca em ambientes que incorporam agentes e objetos 3D.

Considerando o plano navegação na dada perspectiva de projeção de células formando uma grade, o agente A* pode navegar entre as células do plano nos sentidos vertical e horizontal, além do sentido diagonal. No entanto, para que a busca de caminho no sentido diagonal seja possível, o agente deve identificar se a próxima célula é acessível nos dois sentidos (RABIN, 2015, tradução nossa). Afim de facilitar a visualização destas representações do plano, a figura 14 ilustra as células em forma de grade. Considerando o agente s, seu objetivo é alcançar o destino p através da busca do menor caminho no sentido diagonal, porém a célula A2 está acessível apenas no limite horizontal, mas não no sentido vertical, impossibilitando a busca diagonal. Já os destino r e q são alcançáveis por ambos os sentidos.

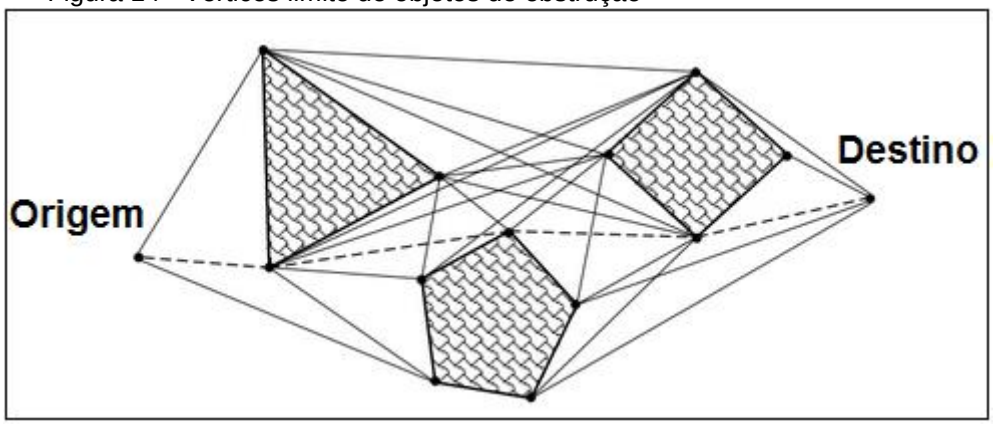
Figura 13 - Projeção de uma grade no plano



Fonte: Rabin (2015)

A resolução para problemas de navegação desta natureza são obtidos por meio do pré-processamento dos objetos identificados como obstáculos não-transitáveis. Tendo os objetos como representações poligonais, esta identificação se dá atribuindo o último vértice alcançável entre os limites de dois sentidos do objeto. Resumidamente, os limites de dois sentidos de um objeto são quaisquer ângulos convexos do objeto (RABIN, 2015, tradução nossa). Para melhor representar esta explicação, as figuras 15 e 16 ilustram objetos que obstruem a navegação com seus ângulos convexos, caracterizando os últimos vértices alcançáveis do objeto.

Figura 14 - Vértices limite de objetos de obstrução

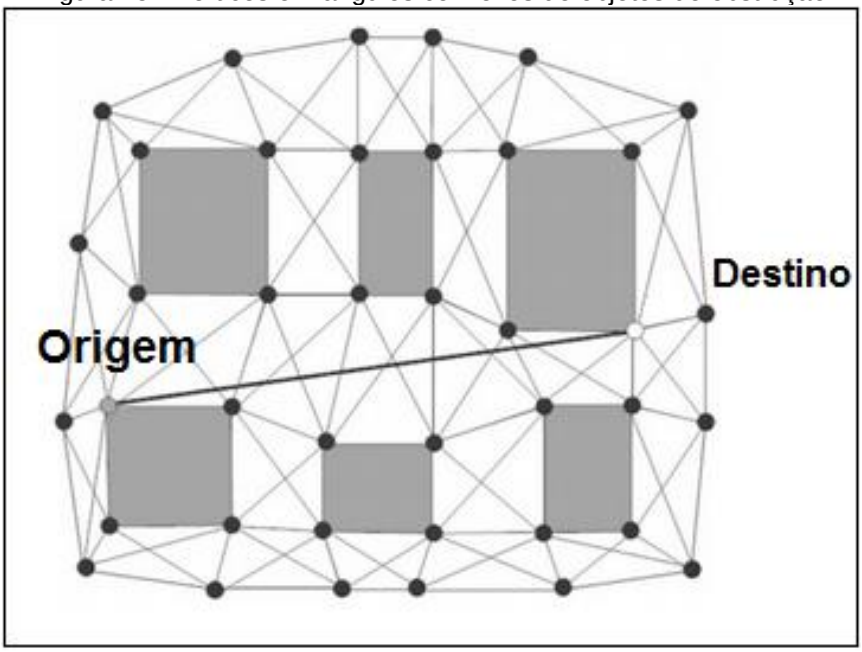


Fonte: Adaptado de Rabin (2015).

Cada ponto localizado nos ângulos convexos dos objetos que obstruem a busca do menor caminho representa o último vértice alcançável

antes do bloqueio. Com esta técnica de pré-processamento do ambiente que o agente A* irá navegar, que então é caracterizado uma malha de navegação.

Figura 15 - Vértices em ângulos convexos de objetos de obstrução



Fonte: Adaptado de Rabin (2015).

4 MOTORES DE JOGO

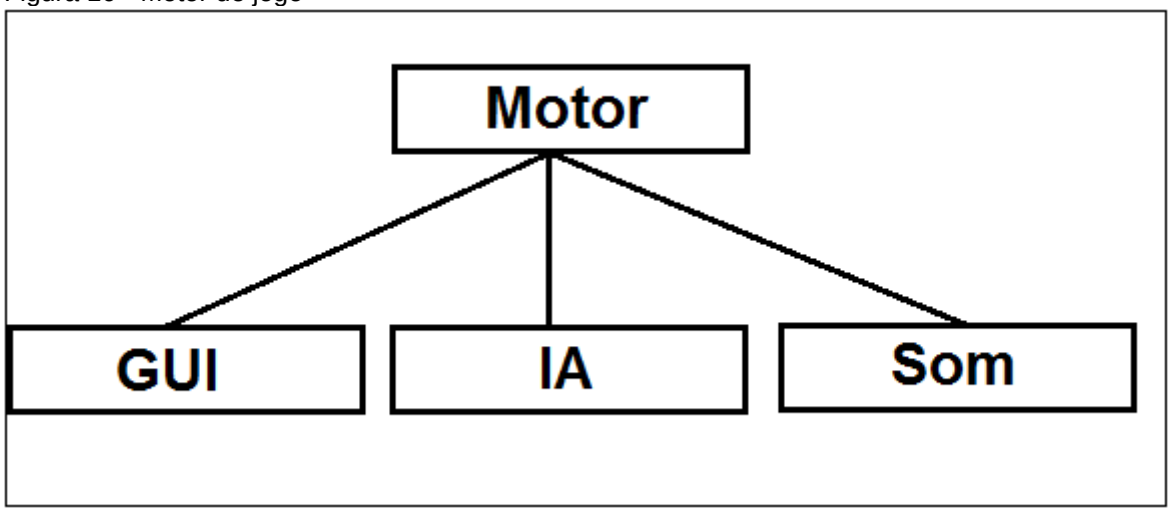
Os motores de jogo envolvem vários tipos de integrações que se fazem indispensáveis para o correto funcionamento da aplicação. Um motor de jogo deve possibilitar a programação de múltiplos objetos renderizáveis, e a simulação física por meio de uma linguagem de scripts eficiente. Outra funcionalidade que se faz indispensável são os recursos de importação de modelos 3D, imagens e efeitos de áudio produzidos a partir de outros meios externos. Além destes recursos que devem fazer parte de um motor de jogo, nos tempos atuais de tecnologia, devido a disseminação do software em múltiplas plataformas, como por exemplo os dispositivos móveis, se faz indispensável que o jogo desenvolvido possa atender a estas exigências para rodar nas diferentes plataformas (PASSOS et al, 2009).

Um dos advento que contribui para a expansão dos motores de jogo, é que antigamente os jogos iam além de programações de eventos repetitivos em uma máquina de estados, e rotinas gráficas relativamente simples (LEWIS; JACOBSON, 2002, tradução nossa).

Atualmente os jogos são construídos de forma modular, utilizando-se exaustivamente dos recursos de reaproveitamento de código, e coleções prontas não relativas ao funcionamento intrínseco do jogo (KYAW; PETERS; SWE, 2013, tradução nossa).

Um motor de jogo manipula somente funções relacionadas à fisica, renderização gráfica, dinâmicas comportamentais dos objetos, além de outras convenções que poupam tempo de desenvolvimento. Tais funções não influenciam na construção artística do jogo em sí. O esquema hierárquico de níveis de compromentimento do motor é baseado no conceito de que sua função é meramente de dar suporte a cada funcionalidade do jogo, mas o desenvolvimento e a criação dos níveis inferiores é manipulada independentemente (KYAW; PETERS; SWE, 2013, tradução nossa). A seguir, a figura 17 demonstra o esquema de hierarquia e suporte a criação de cada funcionalidade do jogo.

Figura 16 - Motor de jogo



Fonte: Do autor.

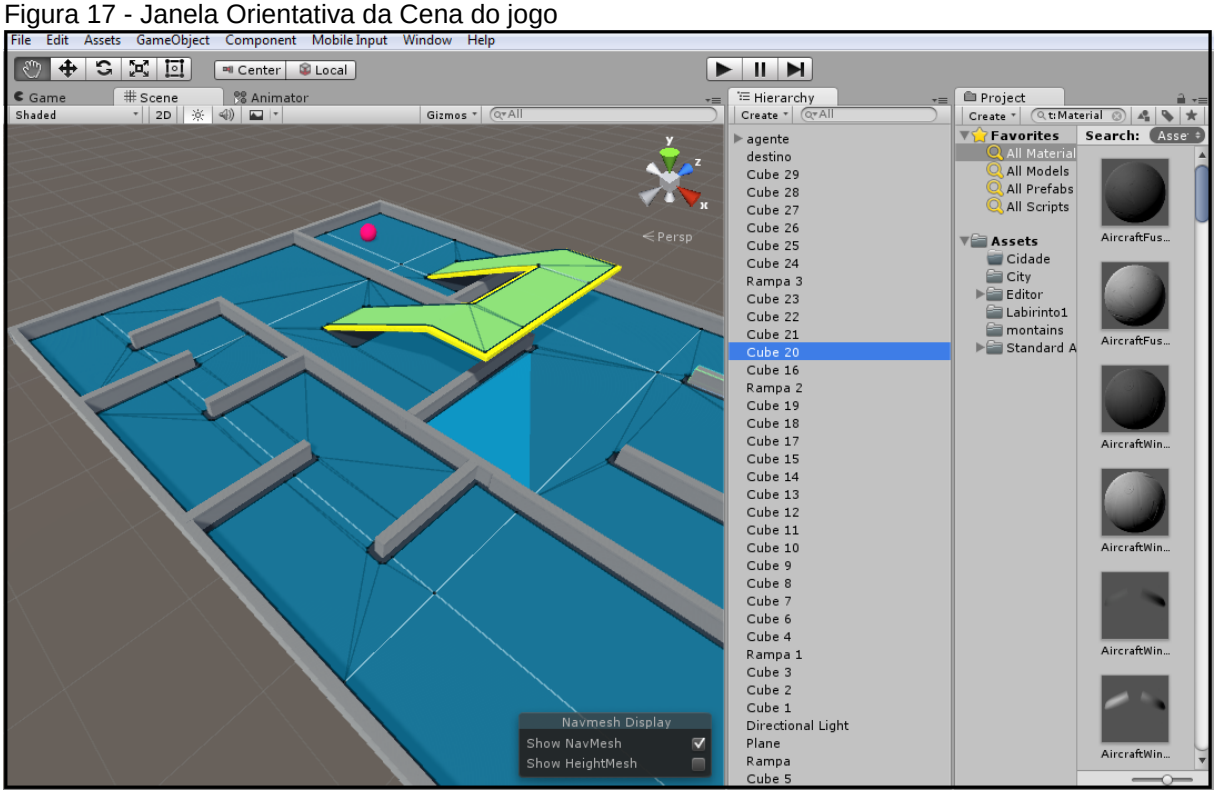
Nos motores de jogo, as funcionalidades são apresentadas ao desenvolvedor de forma relativamente intuitiva. Isto porque, a manipulação dos elementos de jogo deve ser de fácil acesso e alteração (LEWIS; JACOBSON, 2002, tradução nossa). Para isto, quase todos os motores de jogo disponíveis no mercado, desde os mais robustos até os gratuitos, são dispostos de duas janelas principais na qual em uma delas é possivel observar a execução do jogo em tempo real, e na outra os todos os elementos que compõe a cena de jogo individualmente. Nesta interface, o objetivo é que o desenvolvedor tenha total controle sobre os elementos que estão sendo manipulados, estes que, podem ser os ângulos, ambientes ou personagens por exemplo. Dessa forma, o desenvolvedor obtém o retorno orientativo das alterações realizadas (PASSOS et al, 2009).

Ressalta-se que atualmente os jogos contam com um parâmetro de criação que beira o realismo. Entretanto, muitas vezes os recursos fornecidos pelo motor não são suficientes. Porém, os motores oferecem suporte a diversas ferramentas de terceiros para a solução de problemas desta natureza. Um dos exemplos, é o popular uso de programas de modelagem 3D, pois estes dão suporte à criação de objetos artísticos complexos e profissionais. A partir de uma ferramenta externa, é possível importar diretamente estes arquivos para o motor (LEWIS; JACOBSON, 2002, tradução nossa). Outro exemplo, é a

possibilidade de manipulação de motores específicos como o PhysX, popular motor físico que se faz presente em jogos de grandes desenvolvedores, ele permite a criação de ambientes de simulação realistas, possuindo internamente as tecnologias necessárias para uma implementação realista (PASSOS et al, 2009)

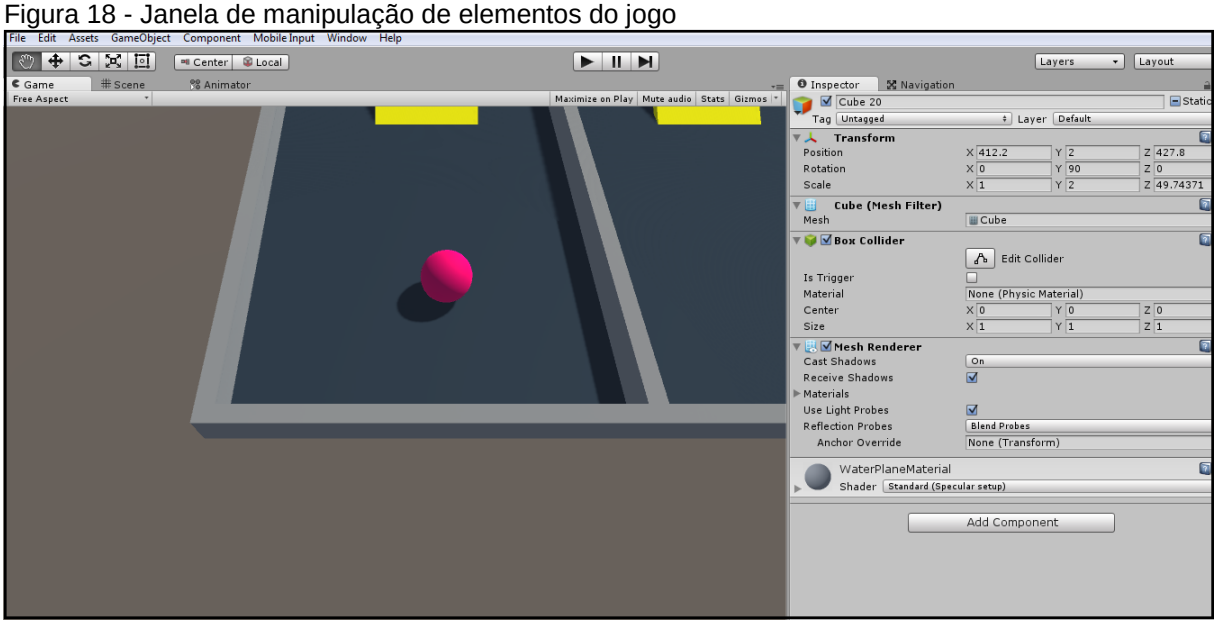
As figuras 18 e 19 esclarecem estes recursos orientativos, apresentando como é feita a manipulação dos elementos de jogo, e o acompanhamento do decorrer da execução da cena.

Figura 17 - Janela Orientativa da Cena do jogo



Fonte: Do autor.

Figura 18 - Janela de manipulação de elementos do jogo



Fonte: Do autor.

A escolha de um motor de jogo é um tanto relativa. Geralmente, os desenvolvedores já iniciam seus estudos com base em alguma recomendação alheia. Visto que a diferença entre funcionalidades é ínfima, e para grandes produções as empresas desenvolvem sua própria engine, a escolha da *Game Engine* se torna basicamente um critério pessoal (LEWIS; JACOBSON, 2002, tradução nossa).

No entanto, o âmbito comercial provê algumas diferenças a se considerar. Para um jogo que pretende-se publicar, a análise acerca da escolha do motor leva em conta critérios como o preço da aquisição definitiva da engine, além dos preços impostos sobre cada produto vendido (LEWIS; JACOBSON, 2002, tradução nossa). Outro critério a se considerar, é a quantidade de desenvolvedores que deverão obter acesso à engine. Algumas empresas que vendem engines fornecem desde pacotes exclusivos para desenvolvedores iniciantes no mercado, até pacotes profissionais para grandes equipes (LEWIS; JACOBSON, 2002, tradução nossa).

4.1 UNITY3D

A popularidade do Unity3D, advém de dois fatores principais que são supervisionados pelos desenvolvedores no momento da escolha da ferramenta: o suporte as ferramentas mais populares que aumentam o dinamismo do desenvolvimento, e o custo financeiro. Uma das considerações a serem feitas,.Considerando que no Unity3D os módulos mais populares de desenvolvimento refletem os usados por grandes empresas, e o seu custo relativamente baixo, se fazendo acessível até para desenvolvedores independentes, atualmente o Unity3D é referência desde pequenas até as maiores empresas (PASSOS et al, 2009).

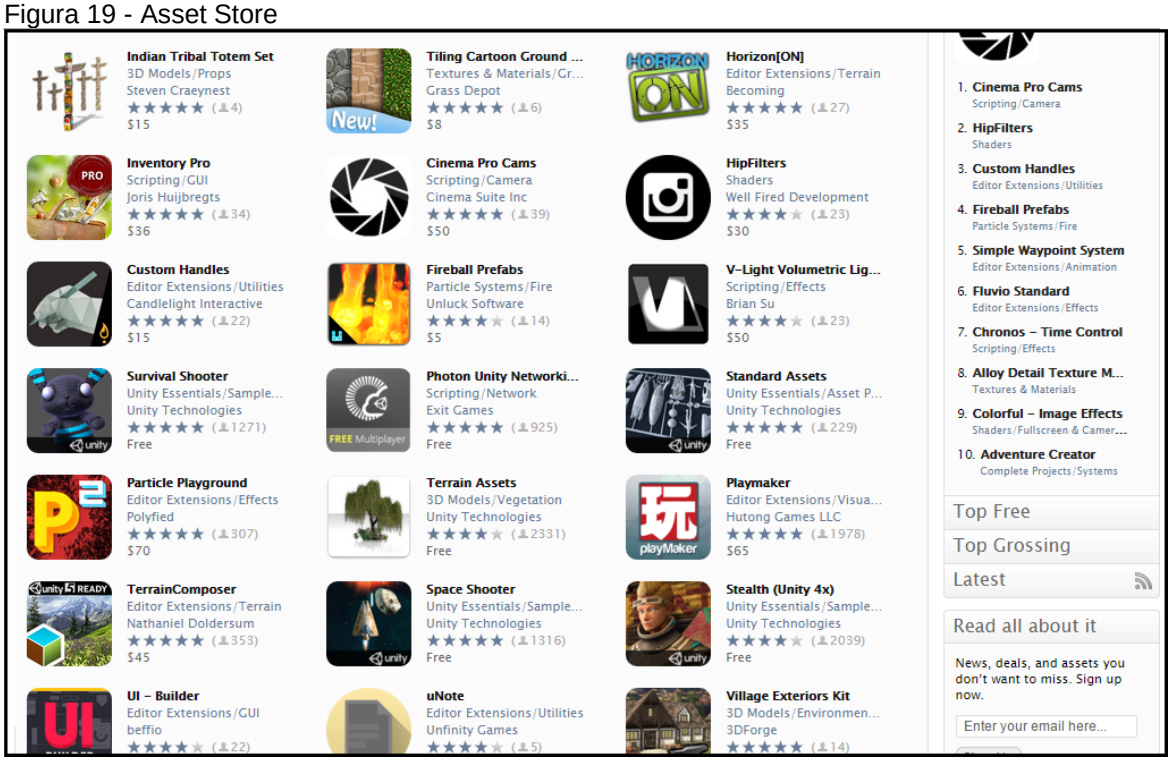
Grande parte dos usuários do Unity3D, fazem análises positivas sobre a intuitividade da interface do Unity3D, sendo esta uma de suas características mais relevantes (PASSOS et al, 2009). A aprovação do motor se da considerando que para os desenvolvedores iniciantes a interface do Unity3D oferece alta simplicidade e organização, dispondo na tela principal apenas os elementos pertinentes ao uso no dado momento. Esta disposição ainda oferece total apoio à personalização, deixando a critério do desenvolvedor escolher o tamanho e a localização de cada janela. Além disto, um grande avanço para o desenvolvimento de jogos no Unity3D é a compatibilidade do compartilhamento dos arquivos com a sua loja virtual, o Asset Store. A partir do motor é possível buscar novos pacotes na loja virtual com a opção de abri-los diretamente na instância da cena.

4.1.1 Asset Store

Como mencionado, o Asset Store permite alto compartilhamento de pacotes e recursos ao desenvolvedor. Cada pacote é apresentado com sua descrição e preço, sendo também sub-divididos de acordo com suas categorias, sejam scripts, terrenos, texturas, efeitos de áudio, modelos 3D entre outras classificações. O principal recurso da utilização de Assets, é a modificação do componente em tempo-real. Uma vez importado ao Unity3D, as

modificações feitas em uma imagem ou animação em outro software, são atualizads automaticamente no Unity3D, dispensando assim a necessidade de uma nova importação, bastando apenas que o arquivo modificado seja salvo pelo software que o desenvolveu (PASSOS et al, 2009).

Figura 19 - Asset Store



Fonte: Unity3D (2015).

5 TRABALHOS CORRELATOS

A Inteligência Artificial é explorada em diversas áreas no ramo dos Jogos Digitais, o que permite uma pesquisa ampla em diversas aplicações. Serão apresentados alguns trabalhos que envolvem algoritmos planejadores em diferentes gêneros de jogos.

5.1 APLICAÇÃO DE PLANEJAMENTO EM JOGOS DE ESTRATÉGIA

A dissertação de mestrado apresentada na Universidade Federal de Uberlândia no ano de 2008 por Bruno Nepomuceno Luiz objetivou a criação de algoritmos planejadores em jogos Real-Time-Strategy aplicáveis aos ambientes característicos do gênero para geração de planos de esquadrões.

Curiosamente, foi selecionado um ambiente de testes com agentes pré-programados para realização dos testes de viabilidade. O Project Hoshimi se constitue em um ambiente de ação de multiagentes, sendo que, segundo Luiz (2008), cada agente é representado por um nanorobô dotado de diversos componentes de controle e busca.

Para a aplicação, foi utilizado o algoritmo A* em um subsistema entitulado de AStarPlanner, cuja arquitetura é definida por nós de um grafo. Nele, dois testes foram realizados com o intuito de comparar a geração de planos de esquadrões e então avaliar a qualidade dos planos.

Nos testes realizados, foram impostas limitações para que fatores externos não influenciassem nos testes, logo, a limitação consistiu em definir uma única meta de preencher completamente um ponto HP, o objetivo dos nanorobôs era coletar enzimas, transporta-las e deposita-las no ponto HP.

Concluidos sete diferentes mapas de testes para preenchimento de pontos HP, os resultados foram idênticos em todos, comprovando que o algoritmo AstarPlanner teve sucesso em gerar planos sem perda de desempenho (LUIZ, 2008).

5.2 UMA ABORDAGEM PARA MAXIMIZAÇÃO DA PRODUÇÃO DE RECURSOS EM JOGOS RTS

Este trabalho desenvolvido em 2012 pela Universidade Federal de Uberlândia, por Thiago França Naves como dissertação de mestrado, visou a implementação de algoritmos verificadores de desempenho em um jogo de RTS. Além disso, um algoritmo de busca estocástica foi utilizado para comparar o desempenho dos jogadores perante o algoritmo na busca por metas visando aprodução de recursos em jogos RTS.

Utilizando como ambiente de testes um próprio jogo do gênero RTS, Naves (2012) realizou testes de performance e comparação baseado em resultados obtidos diretamente de jogadores humanos e fez a comparação dos resultados obtidos da performance dos jogadores, e também do algoritmo aplicado.

Na aplicação dos testes realizados, uma Application Programming Interface (API) foi utilizada para possbilitar a injeção dos algoritmos de testes diretamente no motor do jogo. A API foi desenvolvida para coletar dados internos do jogo viabilizando a manipulação de unidades e recursos do jogo StarCraft.

A execução dos testes se deu pela definição das metas a serem alcançadas por cada um dos jogadores participantes do teste em diferentes limites de tempo estipulados, sendo que os jogadores possuiam níveis de conhecimento distintos sobre o jogo.

Após realizados os testes, os algoritmos se mostraram lentos em seu tempo de execução, uma vez que em alguns casos, o algoritmo iniciava a próxima meta e a terminava antes do término da anterior, ainda assim, os algoritmos superavam os jogadores experientes na maioria dos casos, e de forma unânime em comparação aos jogadores iniciantes e médios. Naves (2010) define isto, como o maior desafio a ser superado em pesquisas futuras.

5.3 UTILIZAÇÃO DE AMBIENTES PARALELOS NO PROCESSO DE APRENDIZAGEM DE ALGORITMOS DE BUSCA DE CAMINHO EM TEMPO REAL

A dissertação de mestrado de Vinicius Marques Terra pelo Instituto de Ciências Exatas da Universidade Federal de Minas Gerais no ano de 2010, considerou a possibilidade da paralelização de algoritmos de busca de caminho em tempo real.

O objetivo da pesquisa foi de impor restrições de tempo aos algoritmos de busca em tempo real diminuindo o tempo de convergência. Para isto, se fez necessária a utilização de múltiplos núcleos para compartilhamento do aprendizado adquirido pela busca principal, dessa forma tornando possível o auxílio proveniente de outros trechos de busca.

Segundo o autor, vários algoritmos de busca em tempo real foram estudados, tendo assim, o entendimento básico do comportamento de cada um deles.

Uma das aplicações testadas, sugeriu a utilização de um algoritmo de busca em tempo real que fosse original, porém, esta se mostrou ineficiente, pois sua arquitetura é baseada na criação de uma tabela hash para armazenamento de valores de aprendizagem advindas das buscas, e com isso, a consequência foi o alto tempo de preenchimento dos *buffers* dos núcleos que processavam as tarefas auxiliares, superando até mesmo o tempo de busca de caminho (TERRA, 2010).

5.4 UM SISTEMA DE APOIO AO JOGADOR PARA JOGOS DE ESTRATÉGIA EM TEMPO REAL

Esta dissertação de mestrado foi desenvolvida por Renato Luiz de Freitas Cunha, pela Universidade Federal de Minas Gerais no ano de 2010. O trabalho é voltado para a experiência do usuário, propondo melhorar o seu desempenho por meio de dicas e tutoriais de interface. Este que, é desenvolvido com base na geração de táticas e estratégias utilizando árvores

de decisão. A observação dos usuários se fez essencial para a coleta de informações relacionadas à experiência durante o jogo.

A metodologia, consistiu em submeter participantes à um jogo RTS, e desafia-los a ganhar. Duas partidas foram realizadas por cada praticante, uma com o sistema de dicas ativado, e outra com o sistema desativado. O total de praticantes era seis. Para o desenvolvimento do trabalho, Cunha (2010) elaborou um questionário pré-teste a fim de coletar infomações básicas dos usuários como por exemplo a intimidade com jogos RTS. Além deste, um segundo questionário pós-teste foi aplicado para coletar informações referentes ao sistema de dicas.

O sistema de dicas implementado tratou de três categorias, sendo elas a gerência de recursos, estratégia básica, e contra-estratégia.

De acordo com o autor, os resultados proporcionaram a observação dos aspectos deficientes de acordo com os questinários aplicados, Cunha (2010) afirma também que, com este experimento, é possível perceber o potencial de aplicação destes sistemas de dicas em jogos reais, ressaltando que os jogadores se mostraram motivados e sugeriram que sistemas deste tipo, se fizessem presentes nos jogos de RTS em geral.

6 AVALIAÇÃO DO ALGORITMO A* NA HEURÍSTICA DE PATH-PLANNING EM AMBIENTES DE JOGOS UTILIZANDO O MOTOR UNITY3D

Na avaliação do algoritmo A* empregam-se heurísticas de *path-planning* a fim de possibilitar a aplicação do algoritmo por meio de um agente. Avaliar o desempenho de um agente implica em definir métricas de qualidade relativas aos ambientes em que este é disposto, pois de acordo com Rabin (2015) “A performance de um agente é definida pela qualidade do caminho percorrido.”

Neste trabalho, foram implementados três cenários distintos com o objetivo de explorar e comparar diferentes resultados. Os resultados foram obtidos com base nas métricas de comparação dos ambientes por meio do uso de uma ferramenta de medição disponibilizada pelo Unity3D.

6.1 METODOLOGIA

Para que os objetivos do trabalho fossem atingidos, realizou-se as seguintes etapas metodológicas: levantamento bibliográfico, modelagem de ambientes de jogo 3D, aplicação da malha de navegação por meio da técnica de *baking*, personalização da ferramenta auxiliar NavMesh Agent, desenvolvimento do *script* de *path-planning* para o agente A*, monitoramento e coleta de dados, e a análise comparativa de desempenho do agente A*.

A pesquisa bibliográfica proporcionou que na fundamentação teórica fossem abordados conceitos sobre as etapas de desenvolvimento de Jogos Digitais fazendo-se uma breve introdução a cada etapa, além do estudo acerca das heurísticas de *path-planning* e sua aplicação envolvendo o algoritmo A*. Por último, é feito uma abordagem sobre os motores de jogo e sua respectiva função, sendo que neste trabalho optou-se pelo uso do motor Unity3D.

6.1.1 Modelagem dos ambientes 3D

A modelagem dos ambientes 3D iniciou-se com a definição de diferentes situações de jogo. Os ambientes permitem que o agente A* navegue por meio das malhas do plano, estas que, são diferenciadas de acordo com a estrutura do ambiente.

A estrutura do ambiente envolve o plano horizontal, os terrenos e os objetos. Estes são responsáveis pelo formato da malha, determinando fisicamente onde o agente A* pode navegar. (RABIN, 2015, tradução nossa).

No desenvolvimento dos ambientes, adotou-se o padrão de criação contendo o plano horizontal, objetos 3D relacionados ao ambiente, além de dois objetos 3D adicionais para representação do agente, e dos respectivos destinos. De acordo com este padrão de desenvolvimento, estabeleceu-se diferentes características físicas com relação ao tamanho, quantidade de vértices de navegação e quantidade de objetos 3D, além de outras diferenças de aspecto visual como texturas e cores.

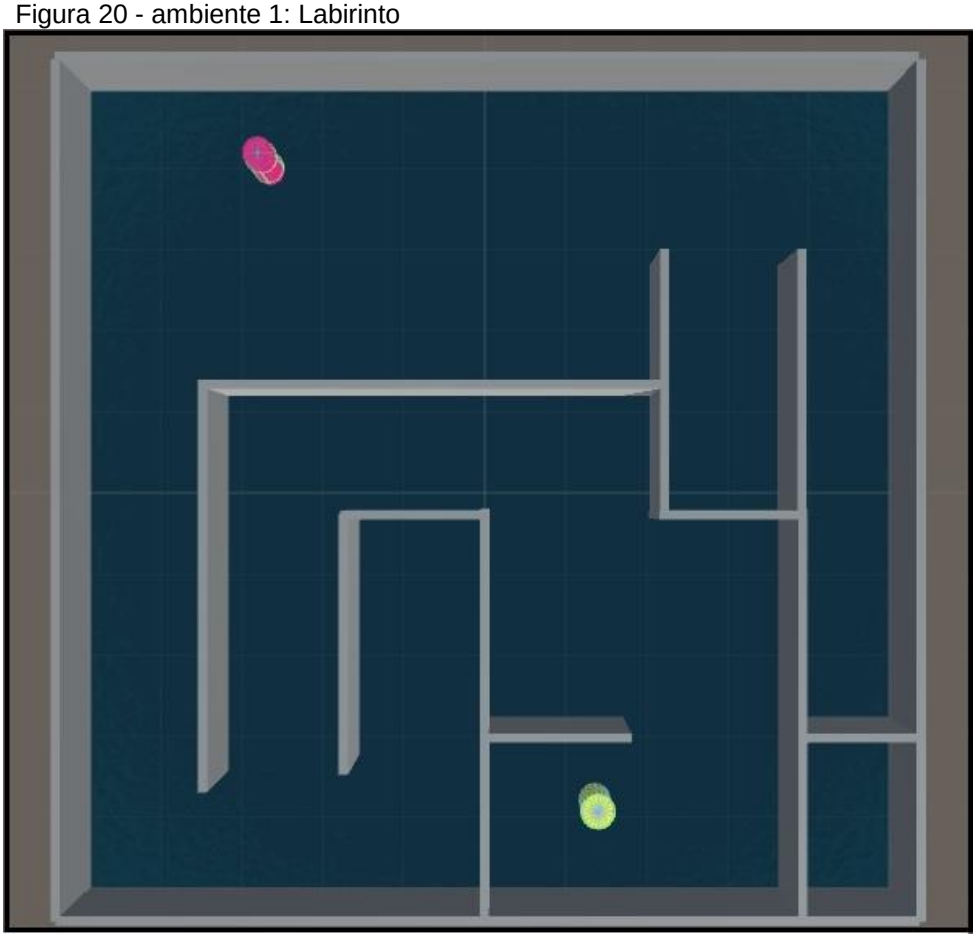
6.1.1.1 Características do ambiente 1: Labirinto

No ambiente 1, criou-se um plano horizontal para a navegação do agente A*, contendo barreiras com o objetivo de impossibilitar a passagem em linha reta do agente em direção aos pontos de destino.

A figura 21 demonstra o ambiente 1 com seu padrão de estrutura. Conforme o desenvolvimento, definiu-se a escala de tamanho do plano em 10x10, sendo esta, suficiente para comportar sua estrutura. O plano horizontal é representado pela cor azul, e o ambiente ainda conta com as barreiras em cinza, e dois objetos 3D representados pelas cores amarelo e rosa, sendo o amarelo o objeto agente A*, e o rosa o seu ponto de destino.

O objetivo consiste em alcançar o ponto de destino percorrendo o menor caminho e evitando as barreiras.

Figura 20 - ambiente 1: Labirinto



Fonte: Do autor.

6.1.1.2 Características do ambiente 2: Montanhas

No ambiente 2 seguiu-se o mesmo padrão de desenvolvimento no que diz respeito aos aspectos físicos. Entretanto, no plano horizontal criaram-se as obstruções no formato de montanhas, a fim de explorar objetos 3D multi poligonais e contribuir para a complexidade do ambiente. Para isto, fez-se necessário o uso de uma escala de 50x50 no plano a fim de comportar as estruturas criadas.

Além disto, foram aplicadas texturas com o intuito de obter um aspecto visual mais detalhado. A figura 22 ilustra o ambiente 2 com uma perspectiva aérea de visão.

Figura 21 - ambiente 2: Montanhas

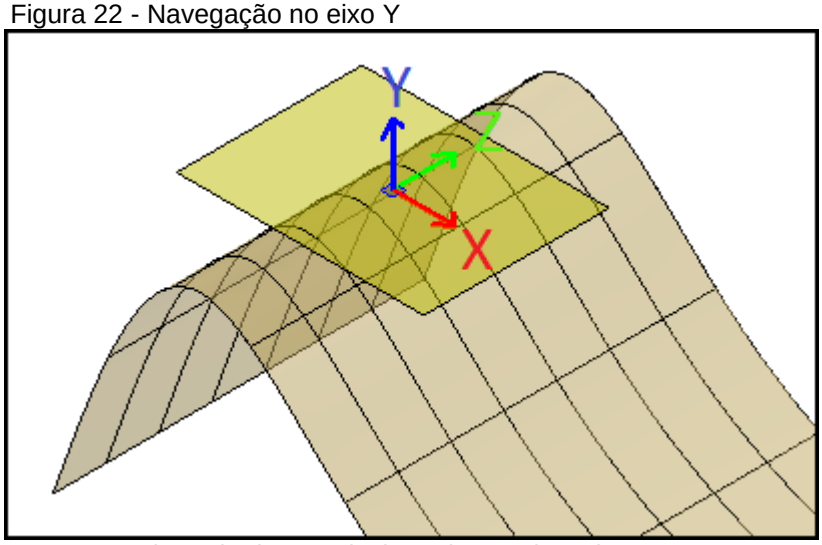
Fonte: Do autor.

6.1.1.3 Características do ambiente 3: corredor com rampas

O terceiro ambiente conta com um diferencial no que diz respeito ao plano de navegação do agente A*. Esta diferença, implica na abrangência da aplicação da malha, visando contribuir com diferentes resultados.

Neste ambiente, além da comum navegação no plano entre vértices dos eixos X e Z, a presença de rampas como objetos 3D possibilitou que o agente A* explorasse vértices do eixo Y, como está representado na figura 23.

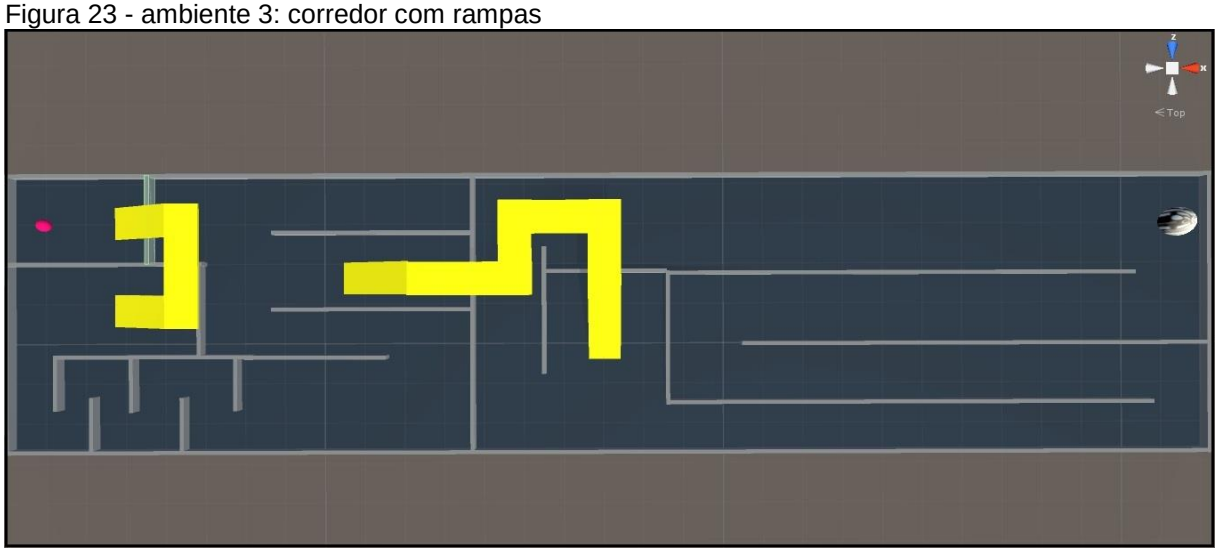
Figura 22 - Navegação no eixo Y



Fonte: Adaptado de Autodesk Exchange (2012).

O ambiente 3 nomeado de corredor com rampas, segue o mesmo padrão estrutural de barreiras do ambiente 1, representadas pela cor cinza. Além disto, estão contidas duas esferas de cores rosa e cinza, que representam respectivamente o agente A* e o seu ponto de destino. Na figura 24, tem-se a visão aérea do ambiente 3.

Figura 23 - ambiente 3: corredor com rampas



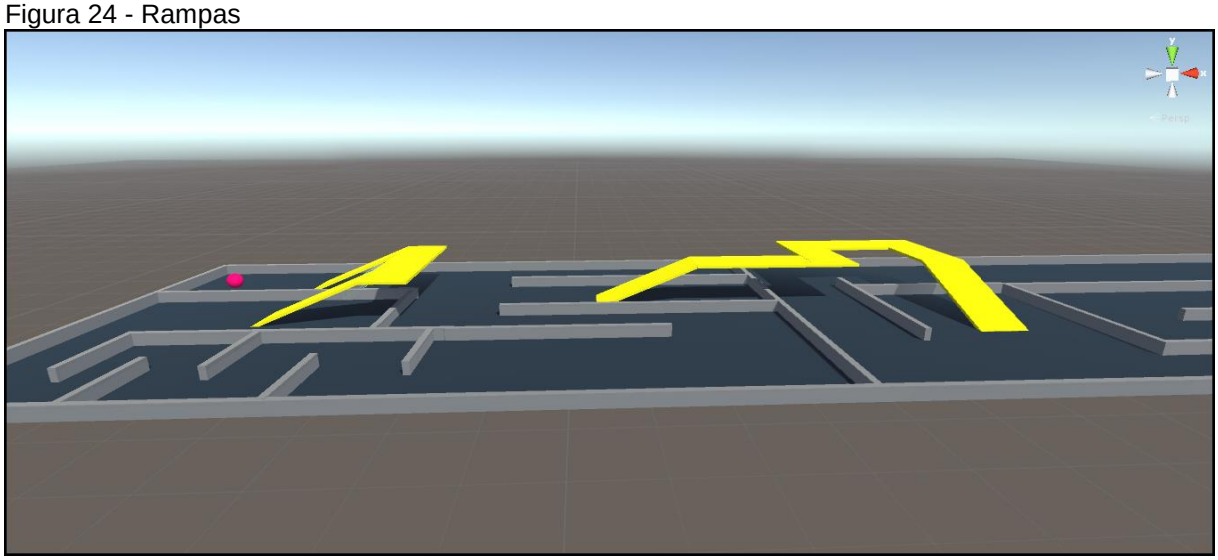
Fonte: Do autor.

Como mencionado, a principal diferença estrutural no ambiente 3 é a presença de objetos 3D que permitem a navegação envolvendo o eixo Y. Esta

diferença, foi implementada por meio do desenvolvimento de rampas, permitindo o agente A* navegar entre os vértices das mesmas.

A figura 25 apresenta o ambiente 3 em uma perspectiva de visão isométrica possibilitando melhor visualização da inclinação vertical das rampas.

Figura 24 - Rampas



Fonte: Do autor.

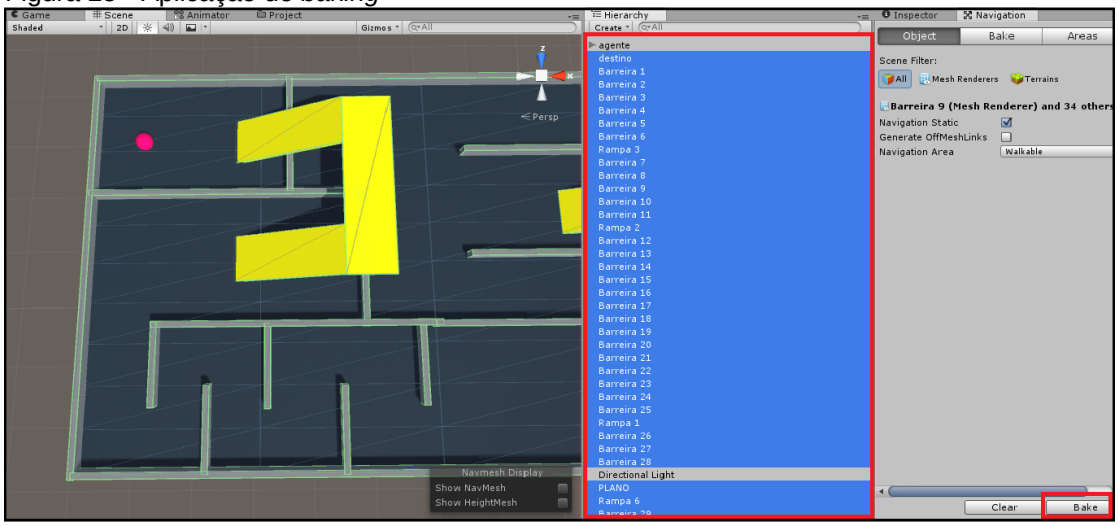
A navegação do agente A*, não somente no eixo Y mas em quaisquer sentidos do espaço nos ambientes, se da por meio dos vértices. No entanto, para determinar cada vértice, deve-se utilizar um recurso que torna os objetos em sólidos. A partir disto, determinam-se os limites navegáveis para o agente, e a disposição dos vértices no ambiente (RABIN, 2015, tradução nossa).

6.1.1.4 Aplicação do recurso *baking*

Nesta etapa, deve-se aplicar o recurso disponível no Unity3D para tornar o plano de navegação e os objetos do ambiente sólidos. Após a identificação das estruturas sólidas do ambiente, o recurso gera automaticamente a disposição dos vértices.

O *baking* é um recurso de uso relativamente simples. Na documentação[3] do Unity3D referente a este recurso, define-se que para tornar a estrutura e os objetos do ambiente sólidos, deve-se selecioná-los a partir do menu de hierarquia e ativar o *bake* como é demonstrado na figura 26.

Figura 25 - Aplicação do *baking*

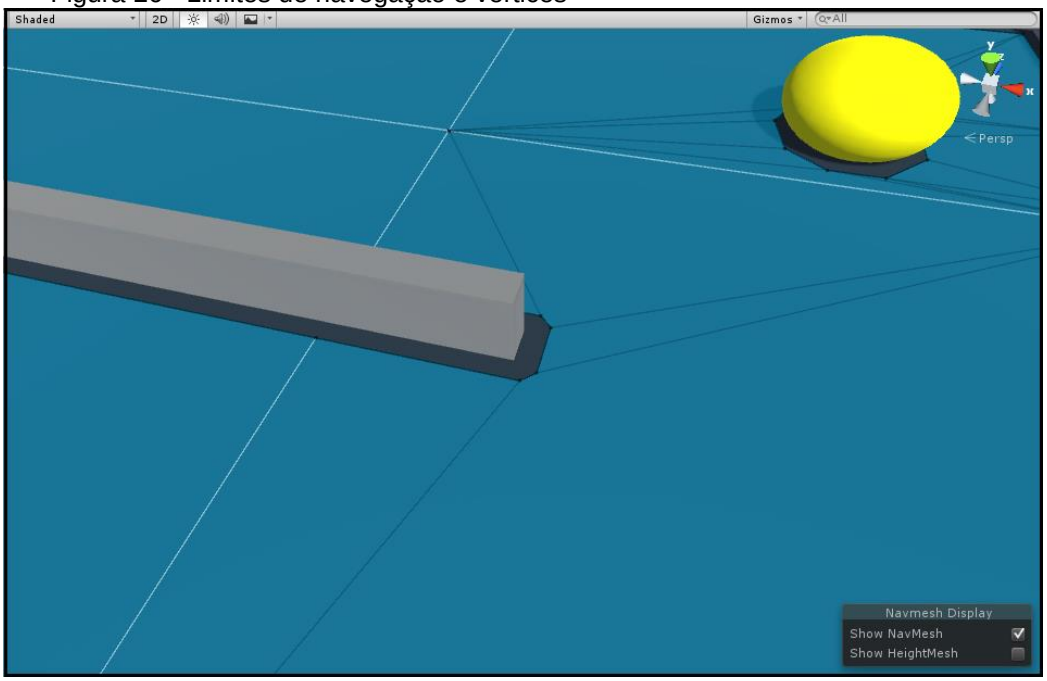


Fonte: Do autor.

Após a aplicação do *baking,* são definidos os limites em que os vértices estarão dispostos no ambiente, pois tornando o plano de navegação e os objetos 3D sólidos, o agente A* navega apenas entre os limites estabelecidos. Na figura 27 são demonstrados os limites de navegação e os vértices. A área escura, caracteriza a área não-navegável, e os vértices representam o limite de navegação do agente.

[3] O tópico que faz a explanação sobre a aplicação do recurso de *baking* está disponível em: http://docs.unity3d.com/Manual/nav-BuildingNavMesh.html

Figura 26 - Limites de navegação e vértices



Fonte: Do autor.

Definidos os ambientes, e utilizado o recurso de *baking* para determinar os limites de navegação e os vértices, a próxima etapa refere-se ao desenvolvimento do agente A* por meio de uma ferramenta de auxílio disponível no Unity3D.

6.1.2 Desenvolvimento do agente A* com a ferramenta NavMesh Agent

Esta etapa refere-se ao desenvolvimento do agente A* utilizando-se a ferramenta NavMesh Agent. O uso desta ferramenta é de suma importância para o sucesso da aplicação do agente A*, pois as dificuldades na implementação de um *script* que defina o A* são expandidas em larga escala quando implementa-se um objeto 3D que o vincule e simultaneamente considere todas as variáveis do ambiente sem comprometer o desempenho (KIAW; SWE; PETERS, tradução nossa).

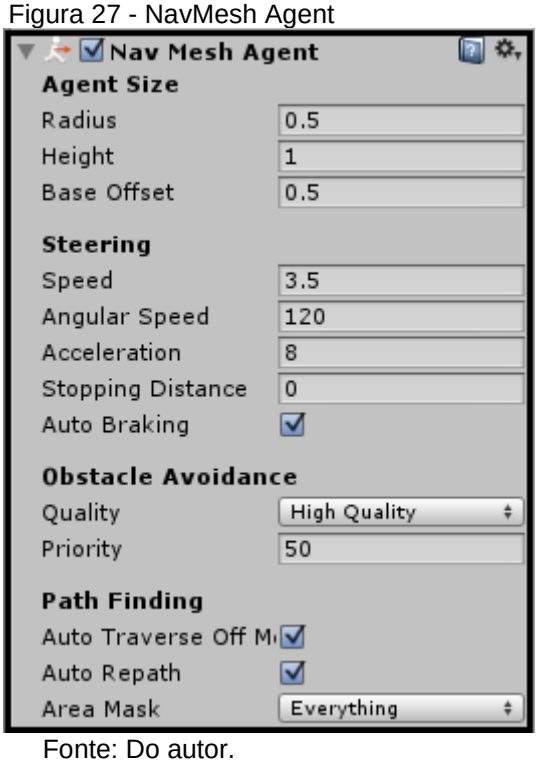
O NavMesh Agent tem o propósito de auxiliar o desenvolvedor na resolução de problemas de navegação nos ambientes criados na própria *engine*, este recurso permite a personalização das funções, métodos, e recursos de navegação.

A navegação do NavMesh Agent tem seu funcionamento baseado na criação de um *script* que define o comportamento do objeto ao qual ele é vinculado. Após esta etapa, a próxima tarefa refere-se à programação do *script* A* que irá vincular o agente.

Ao adicionar o NavMesh Agent como um componente em um objeto, têm-se uma nova aba que exibe funções do agente pré-programadas. Estas funções são apenas convenções básicas de navegação, como por exemplo velocidade, velocidade angular, aceleração, ponto de parada entre outras.

A figura 28 ilustra a aba do recurso NavMesh Agent com suas respectivas funções pré-existentes, já na tabela 1 é apresentado as funções pré-programadas.

Figura 27 - NavMesh Agent



Fonte: Do autor.

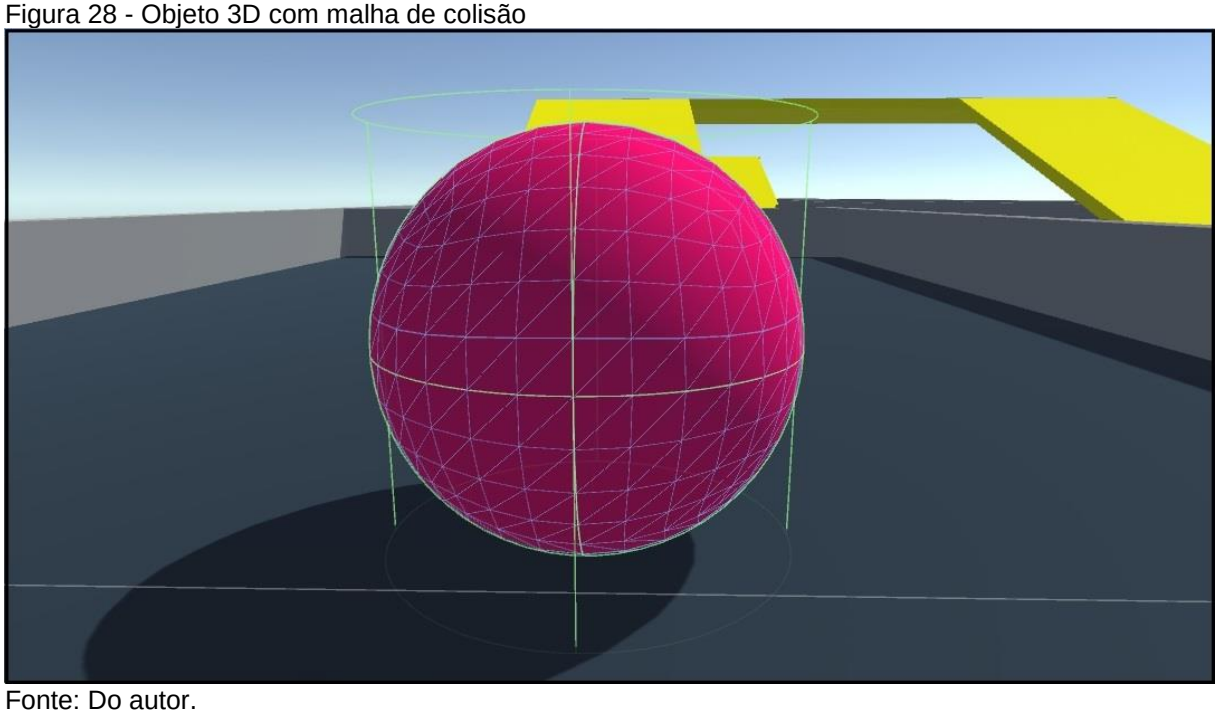
Tabela 1 – Funções pré-programadas

Função	Tipo	Descrição
Radius	float	O raio do agente (graus)
Height	float	Altura do agente
Base Offset	float	Distância da base do agente com relação ao plano
Speed	float	Velocidade de navegação
Angular Speed	float	Rotações (graus/seg)
Acceleration	float	Aceleração
Stopping Distance	float	Distância da parada com relação ao destino
Auto Breaking	bool	Parar automaticamente
Obstacle Quality	NavMeshObstacleQuality	Distância do desvio dos obstáculos
Obstacle Priority	Int	Prioridade de desvio
Auto Traverse Off Mesh	bool	Trespassar entre pontos não-interligados da malha
Auto Repath	bool	Definir uma nova rota caso a atual esteja comprometida
Area Mask	NavMeshAreaMask	Restringir àreas de navegação

Fonte: Do autor.

Após a inserção do objeto vinculado ao NavMesh Agent, a ferramenta identifica o formato do objeto de acordo com seus polígonos e define a malha de colisão automaticamente, ilustrado na figura 29.

Figura 28 - Objeto 3D com malha de colisão



Fonte: Do autor.

Com o objeto 3D contendo o componente NavMesh Agent, a próxima etapa consiste em modelar o objeto adicionando as funções desejadas para que o agente A* possa partir do seu ponto de origem, percorrer o caminho entre os vértices do cenário, e atingir quaisquer pontos de destino. No entanto, para efetivar sua funcionalidade no mapa, deve-se elaborar um *script* de *path-planning* baseado nos conceitos do algoritmo A*.

6.1.3 Script de *path-planning*

Nesta etapa, iniciou-se o desenvolvimento de um script de *path-planning* para determinar o comportamento do objeto agente nos cenários.

Visto que ao adicionar o NavMesh Agent são exibidas funções pré-programadas, o restante das funções disponíveis provê o retorno de valores relativos à execução. A tabela 2 apresenta a descrição de algumas destas funções.

Tabela 2 – Biblioteca de funções do NavMesh Agent

Função	Tipo	Descrição
areaMask	bool	Especifica quais áreas são navegáveis
autoRepath	bool	Procura caminho alternativo em bloqueios
avoidancePriority	Int	Nível de desvio em relação à outros objetos
haspath	bool	Há caminho disponível neste momento de execução?
height	float	Custo de navegação ao passar um obstáculo
pathStatus	NavMeshPathStatus	O caminho está (completo, parcial ou inválido)?
remainingDistance	float	Distância restante até o destino
CalculatePath	float	Calcula a distância até o ponto
GetAreaCost	float	Retorna o custo de navegação da área
SetAreaCost	float	Determina um novo custo para a área

Fonte: Unity3D (2015)

As funções apresentadas na tabela 2 permitem a personalização do NavMesh Agent com relação à navegação. Entretanto, não dizem respeito a sua programação funcional. Na tabela 3 têm-se os métodos implementados relativos a sua funcionalidade prática nos ambientes.

Tabela 3 – Métodos de *path-planning* implementados

Método	Tipo	Descrição
Destination objeto	Vector3	Armazena as coordenadas de destino do
Raycast	bool	Traça uma linha entre dois pontos
gameObject vinculado	GameObject	O objeto ao qual o componente é
position aberto	Vector3	A posição das coordenadas no espaço
Transform	Transform	Armazena as coordenadas do objeto
Destination objeto	Vector3	Armazena as coordenadas de destino do
GetComponent vinculado ao objeto	Component	Retorna o tipo da variável do componente

Fonte: Unity3D (2015).

Considerando os métodos em questão, a figura 30 ilustra a implementação do script de *path-planning,* escrito com base na linguagem de programação C#[4].

Figura 29 - Script de path-planning

```
1 using UnityEngine;
2 using System.Collections;
3
4 public class PathFinding : MonoBehaviour {
5 public Transform destino;
6
7
8     void Start () {
9
10    }
11
12
13    void Update () {
14
15        GetComponent <NavMeshAgent> ().destination = destino.position;
16    }
17 }
```

Fonte: Do autor.

O *script* de *path-planning* inicia com a declaração das bibliotecas *UnityEngine* e *System.Collections*. A classe *PathFinding* é do tipo pública, uma

[4] Linguagem de programação de alto nível, orientada à objetos, desenvolvida pela Microsoft e destinada à criação de aplicativos com o Visual Studio e o .NET *framework* (Microsoft, 2015).

vez que esta classe será incorporada à um objeto na engine. Como trata-se de um *script* singular, ou seja, o seu intuito é o *path-planning* e somente ele, declara-se a classe como *MonoBehaviour*, caracterizando uma única ação. Em seguida, tem-se a criação de uma variável do tipo *Transform*, caracterizando assim a variável *destino* por armazenar as coordenadas de quaisquer objetos vinculados à ela. Neste caso, os objetos vinculados a esta variável tem o objetivo de servir como destino para o agente A*. No corpo do código, essencialmente divide-se as tarefas a serem executadas em duas partes. A primeira função, *void Start* determina a tarefa a ser executada no primeiro *frame* de execução do jogo, esta que, permanece sem atividade, uma vez que o objetivo do *script* é fazer com que o objeto agente permaneça em movimento durante todos os frames de execução do jogo até que atinja o seu destino final. Para tornar possível a movimentação constante do agente durante o tempo de execução, deve-se atualizar sua posição a cada *frame* até que atinja uma posição igual a do seu destino. Por último, o método *GetComponent* retorna a referência de posição da variável *destino* atribuída ao objeto de destino vinculado.

A figura 31 demonstra o código responsável pela atualização da posição do agente A* a cada mudança de *frame.*

Figura 30 - Atualização de posição do agente

```
13      void Update () {
14
15          GetComponent <NavMeshAgent> ().destination = destino.position;
16      }
17  }
```

Fonte: Do autor.

Com a implementação do *script* de *path-planning* e devidamente vinculado ao objeto, na próxima etapa é feita a codificação do algoritmo A* com o objetivo de definir a lógica de busca do menor caminho.

6.1.4 Revisão e implementação do A*

Nesta etapa de desenvolvimento, além de uma breve revisão teórica sobre o algoritmo A*, são apresentadas as classes referentes à sua implementação.

Duas estruturas principais nomeadas de *Open List* e *Closed List* são responsáveis por armazenar valores de distância referentes aos vértices já verificados, e os que serão verificados. O desenvolvimento da lógica acerca destas duas estruturas tem-se os seguintes passos:

a) a execução tem início armazenando a distância do vértice de origem *(v = 0)*; este vértice então é colocado na *Open List*;

b) a execução repete os passos de *c* até *i* enquanto a *Open List* possuir vértices a serem analisados;

c) assumindo que o primeiro vértice da *Open List* é o de menor custo, adota-se este como vértice atual;

d) armazenam-se os vértices adjacentes ao vértice atual;

e) para cada vértice adjacente, verifica-se se cada um destes vértices já está contido na *Closed List*, caso não estejam, será calculado o custo total (*f*) utilizando $f = g + h$, sendo *g* o custo do vértice anterior até o vértice atual, e *h* o custo do vértice atual até o vértice destino;

f) armazena-se este valor (f) como custo do vértice adjacente e define-se o vértice atual como principal para definir o caminho do vértice de origem até o vértice atual;

g) o vértice adjacente é armazenado na Open List, e ordena-se os valores por ordem crescente de custo de distância para atingir-se o vértice de destino;

h) verifica-se se existem mais vértices adjacentes ao atual, caso tenham, o processo é repetido a partir do passo *d,* caso não existam mais nós adjacentes, armazena-se o vértice atual na *Closed List* e remove-se da *Open List;*

i) repete-se o processo a partir do passo *c.*

Seguindo-se estes passos, o algoritmo adota o vértice atual como vértice de destino. Com isto, armazena-se os vértices de menor custo entre o

vértice de destino e o vértice de origem gerando o caminho de menor custo. Ao final, inverte-se a lista para que o caminho seja percorrido a partir do vértice de origem.

A implementação do algoritmo A* é dividida entre três classes. A primeira é a classe *Vertice*, responsável por armazenar os valores referentes a *g* e *h*, além de um marcador para os vértices adjacentes que possam caracterizar um obstáculo.

A figura 32 demonstra o código da classe *Vértice*, a variável *custoTotal* é o valor *g*, e a variável *custoEstimado* é o valor *h*. Já a figura 34 ilustra o código referente à segunda classe *Ordenar_Lista*, *que lida com a ordenação dos vértices fazendo a chamada ao método Comparacao*, responsável pela escolha dos vértices de acordo com o *custoEstimado. A terceira classe A_Estrela, lida com as classes* criadas até o momento, e tem seu código dividido entre as figuras 35, 36 e 37.

Ressalta-se que estas três classes, determinam o menor custo de navegação entre os vértices, entretanto, o recurso NavMesh Agent será o responsável pela administração de outros atributos relacionados ao desempenho da navegação. Como mencionado na figura 28 e nas tabelas 1 e 2, este recurso dispõe de métodos e funções que permitem maior controle sobre os aspectos físicos, mecânicos e gráficos, portanto a heurística de *path-planning* é definida pela aplicação do algoritmo A* por meio dos recursos de navegação disponibilizados pelo Unity3D.

Figura 31 - Classe Vertice

```
1  using UnityEngine;
2  using System.Collections;
3  using System;
4
5  public class Vertice : IComparable {
6      public float custoTotal;
7      public float custoEstimado;
8      public bool obstaculo;
9      public Vertice atual;
10     public Vector3 posicao;
11
12     public Vertice() {
13         this.custoEstimado = 0.0f;
14         this.custoTotal = 1.0f;
15         this.atual = null;
16         this.obstaculo = false;
17     }
18
19     public Vertice(Vector3 pos) {
20         this.custoEstimado = 0.0f;
21         this.custoTotal = 1.0f;
22         this.obstaculo = false;
23         this.atual = null;
24         this.posicao = pos;
25     }
26
27     public void CaminhoComObstaculo() {
28         this.obstaculo = true;
29     }
```

Fonte: Do autor.

Além destes métodos, é criado o método *Comparacao* para ordernar os vértices em uma lista que será manipulada pela classe *Ordenar_Lista*. Na figura 33 tem-se o código do referido método.

Figura 32 - Método Comparacao

```
1 public int Comparacao (objeto obj) {
2     Vertice vertice = (Vertice)obj;
3     if (this.custoEstimado < vertice.custoEstimado)
4     return -1;
5     if(this.custoEstimado > vertice.custoEstimado)
6     return 1;
7 }
```

Fonte: Do autor.

A classe responsável pela ordenação dos vértices, tem seu código ilustrado na figura 34. Ressalta-se que após verificados os vértices contidos na lista, faz-se a chamada do método *Comparacao*, nele, o processo de ordenação se dará de acordo com o valor da variável *custoEstimado*.

Figura 33 - Classe Ordenar_Lista

```
1  using UnityEngine;
2  using System.Collections;
3  using System;
4
5  public class Ordenar_Lista {
6      private ArrayList vertices = new ArrayList();
7
8      public int Distancia {
9          get {return this.vertices.Count; }
10     }
11
12     public bool Componente(object vertice) {
13         return this.vertices.Componente(vertice);
14     }
15
16     public primeiroVertice() {
17         if (this.vertices.Count > 0) {
18             return (vertice)this.vertices[0];
19         }
20         return null;
21     }
22
23     public void Adiciona(Vertice vertice) {
24         this.vertices.Add(vertice);
25         this.vertices.Sort();
26     }
27
28     public void Remove(Vertice vertice){
29         this.vertices.Remove(vertice);
30         this.vertices.Sort();
31     }
32 }
```

Fonte: Do autor.

A manipulação destas duas classes apresentadas é feita pela classe *A_Estrela* por meio da construção das duas estruturas principais do A*, a *Open List* e a *Closed List*. Os valores de custo de distância entre dois vértices são calculados pelo método C*ustoEntreVertices*. Este cálculo é feito com o uso do *Vector3*, que armazena as coordenadas do vértice no espaço dos ambientes. A

partir disto, a implementação do código se dá seguindo os passos mencionados no início desta seção.

O cálculo da distância entre os vértices é ilustrado pela figura 35, enquanto os processos lógicos do A* são demonstrados na figura 36.

Figura 34 - Distancia entre vértices

```
using UnityEngine;
using System.Collections;
using System;

public class A_Estrela {
    public static Estrutura closedList, openList;

    private static float CustoEntreVertices (Vertice, verticeAtual,
    Vertice verticeDestino) {
        Vector3 custoVertice = verticeAtual.position = verticeDestino.position;
        return custoVertice.magnitude;
    }
```

Fonte: Do autor

Figura 35 - Código do algoritmo A*

```
public static ArrayList ProcuraCaminho (Vertice inicial, Vertice destino);
    openList = new Estruturas();
    openList.Adiciona(inicial);
    inicial.custoTotal = 0.0f;
    inicial.custoEstimado = CustoEntreVertices (inicial, destino);

    closedList = new Estruturas();
    Vertice vertice = null;

    while (openList.Distancia != 0) {
        Vertice = openList.Primeiro();
        if (Vertice.position == destino.position) {
            return CalculaDistancia(Vertice);
        }

        ArrayList adjacentes = new ArrayList();

        for(int i = 0; i < adjacente.Count; i++) {
            float custo = CustoEntreVertices(vertice,adjacente,destino);
            float custoTotal = vertice.custoTotalvertice + custo;
            float custoEstimadoAdjacente = CustoEntreVertices (adjacente, destino);
            adjacente.custoTotalVertice = custoTotal;
            adjacente.atual = custoEstimadoAdjacente;
            adjacente.custoEstimado = custoTotal + custoEstimado

            if (!openList.Contains(verticeAdjacente)) {
                openList.Adiciona(verticeAdjacente);
            }
        }
```

Fonte: Do autor

Observa-se na figura 37 como fora explanado no início desta seção, que a última etapa do algoritmo A*, adiciona o vértice de menor custo na *ClosedList*, e o remove da *OpenList*, revertendo a ordem dos vértices para que ele seja percorrido da origem até o destino.

Figura 36 - Reversão de ordem da lista

```
40      if (vertice.position != destino.position) {
41      return null;
42      }
43      return CalculaDistancia;
44      }
45
46      private static ArrayList CalculaDistancia(Vertice vertice) {
47      ArrayList ordem = new Arraylist();
48      while (vertice != null) {
49          ordem.Add(vertice);
50          vertice = vertice.atual;
51      }
52      ordem.Reverse();
53      return ordem;
54      }
55  }
```

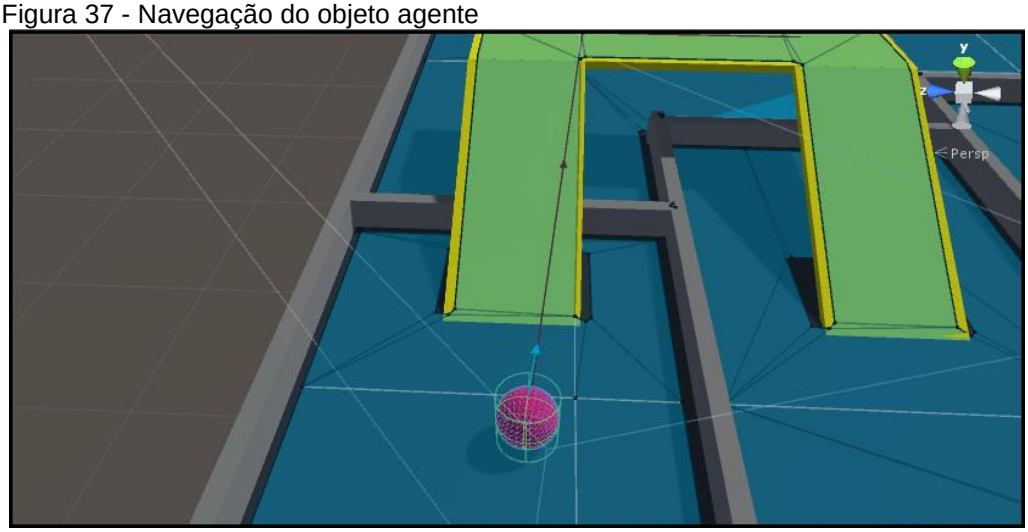
Fonte: Do autor.

Finalizada a etapa de implementação do algoritmo A*, o *script* é vinculado ao agente, e os ambientes estão preparados para a execução da navegação.

6.1.5 Navegação

Na etapa de navegação, consideram-se a execução do jogo e a navegação do agente A* nos ambientes construídos. Na figura 38 observa-se o agente em execução.

Figura 37 - Navegação do objeto agente



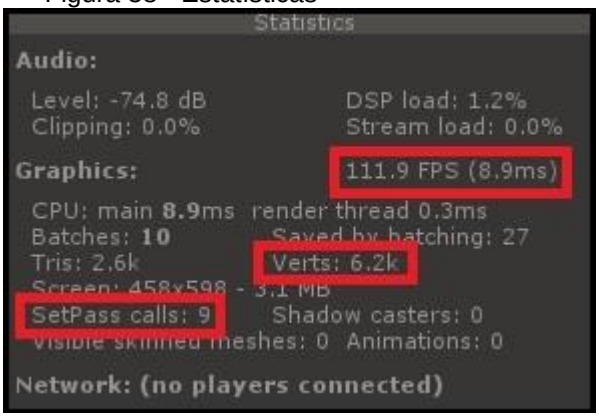
Fonte: Do autor.

Foram definidas dez execuções em cinco destinos diferentes para cada ambiente com o objetivo de não obter-se os resultados com base em uma única execução, visto que para cada uma, os custos computacionais para navegação do agente A* podem divergir.

6.1.6 Monitoramento e coleta de dados

Após as etapas de execução e navegação, fez-se o uso de um monitoramento estatístico básico do Unity3D, a fim de se obter alguns dados com relação aos ambientes. A figura 39 apresenta a janela de dados estatísticos de jogo, dados de áudio não são relevantes ao contexto, no entanto a quantidade de vértices, *frames* por segundo e de renderizações auxiliam no monitoramento das execuções.

Figura 38 - Estatísticas

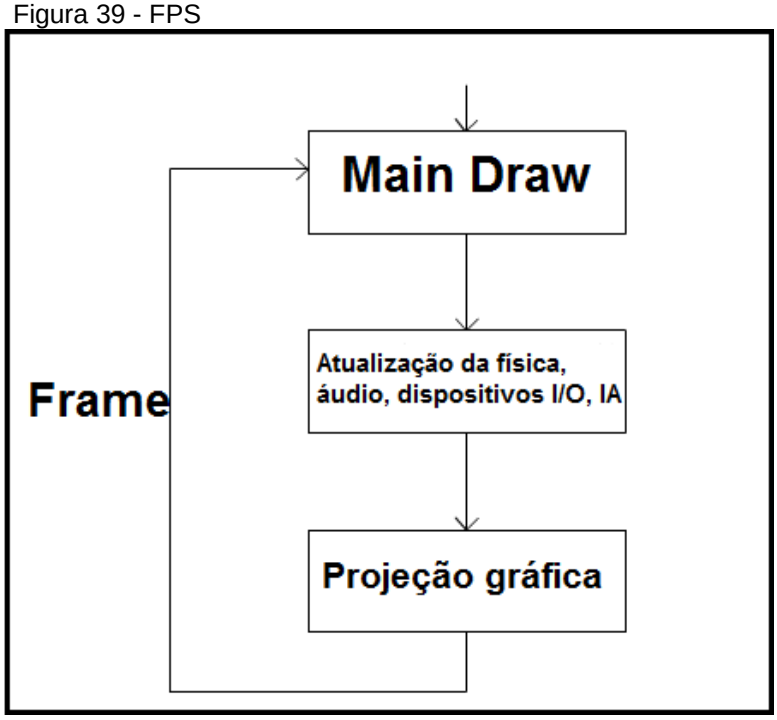


Fonte: Adaptado de Unity3D (2015).

Tendo os primeiros dados referentes à execução ilustrados na figura 39, fez-se uma pesquisa acerca das métricas que determinam a qualidade de navegação do agente A* nos ambientes, pois de acordo com Rabin (2012, p. 536) “A técnica de representação do conhecimento e a quantidade de informações sobre o ambiente, afetam diretamente a eficiência e a qualidade dos caminhos que um agente percorre.” o autor ainda ressalta que “prover um modelo de ambiente excessivamente detalhado pode comprometer os recursos computacionais.” (RABIN, 2012, p. 536). Com isto, para o monitoramento e a coleta de dados, optou-se por utilizar uma ferramenta que provesse dados computacionais referentes às execuções. Esta ferramenta nomeada de Profiler é disponibilizada pelo Unity3D com o objetivo de coletar os dados em tempo de execução, permitindo a observação em tempo-real.

Utilizando-se a ferramenta Profiler, fez-se uma pesquisa com o objetivo de definir os dados de maior relevância para avaliar a qualidade das execuções. A figura 40 refere-se ao processo de atualização dos *frames* em um jogo, e a tabela 4 apresenta os valores qualitativos referentes a este parâmetro de medição.

Figura 39 - FPS



Fonte: Do autor.

Tabela 4 – Performance de um jogo

FPS (Frames por segundo)	Performance
Acima de 200	Excelente
Entre 100 e 200	Ótimo
Entre 60 e 100	Bom
Entre 30 e 60	Regular
Abaixo de 30	Ruim

Fonte: Adaptado de MMCN (2006).

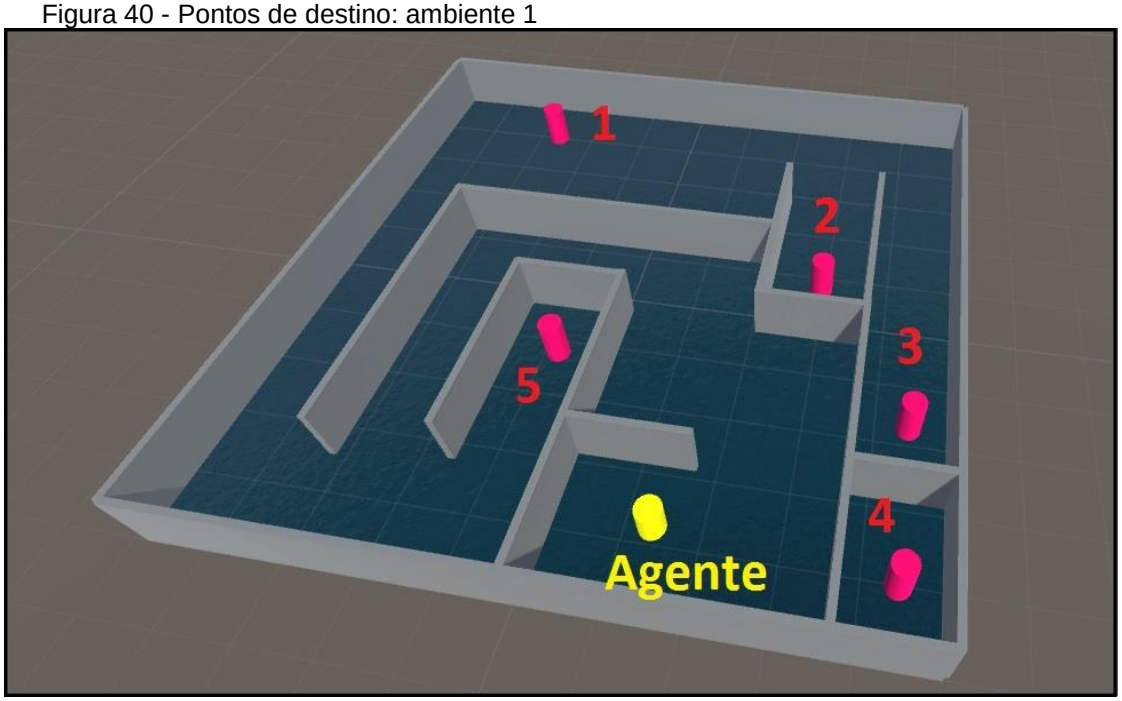
6.1.7 Testes Realizados

Na etapa de realização dos testes foi utilizado o seguinte sistema: Computador desktop com sistema operacional Windows 7, 4GB de memória RAM, processador AMD Phenom II X4 3,4 GHz com 1 TB de armazenamento interno. Utilizando-se deste sistema, para os três ambientes desenvolvidos (figuras 21, 22 e 24), o quadro de testes foi definido da seguinte forma:

a) o objeto agente vinculado ao algoritmo A* será destinado a cinco diferentes pontos de destino em cada ambiente;
b) o agente A* irá percorrer o menor caminho entre os vértices do mapa até o objeto destino em dez execuções;
c) dez execuções extras não envolvendo a busca pelo objeto destino serão realizadas a fim de se fazer a coleta de dados sem a presença do agente;
d) após o fim de todas as execuções e da coleta de dados, serão comparados os resultados na presença e na ausência do agente A*.

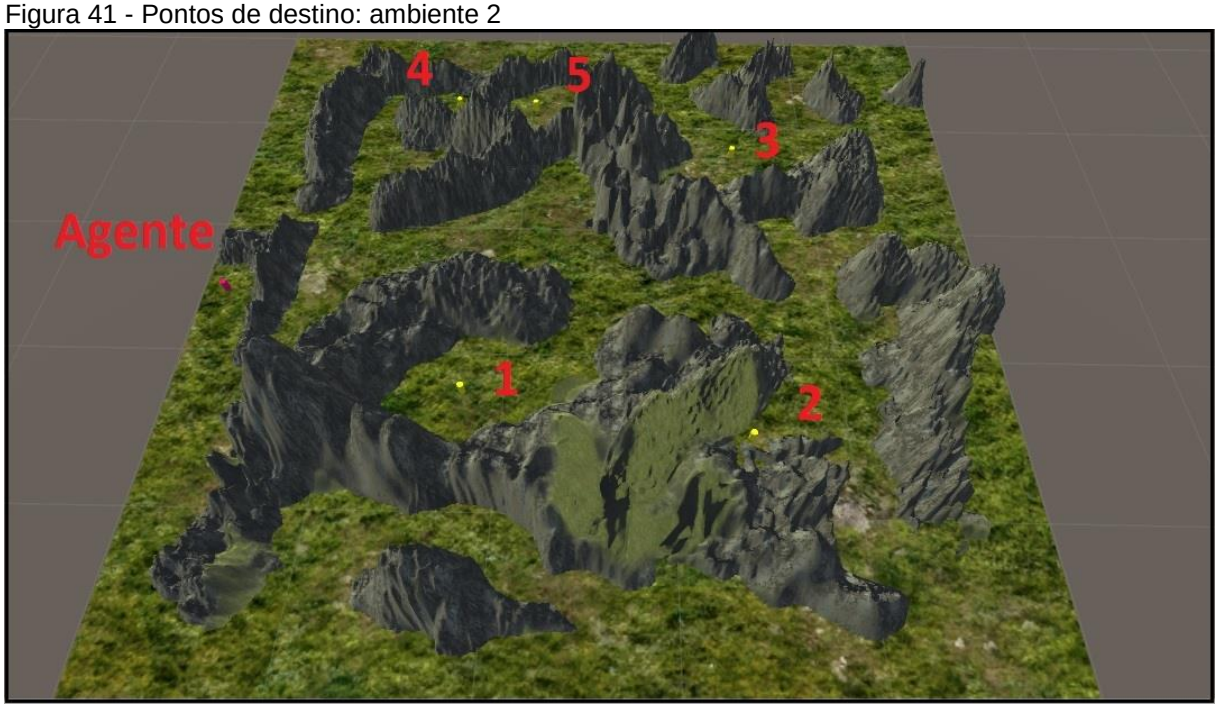
Os cinco pontos distintos em que o objeto destino estará disposto nos ambientes implementados são ilustrados nas figuras 41, 42 e 43.

Figura 40 - Pontos de destino: ambiente 1



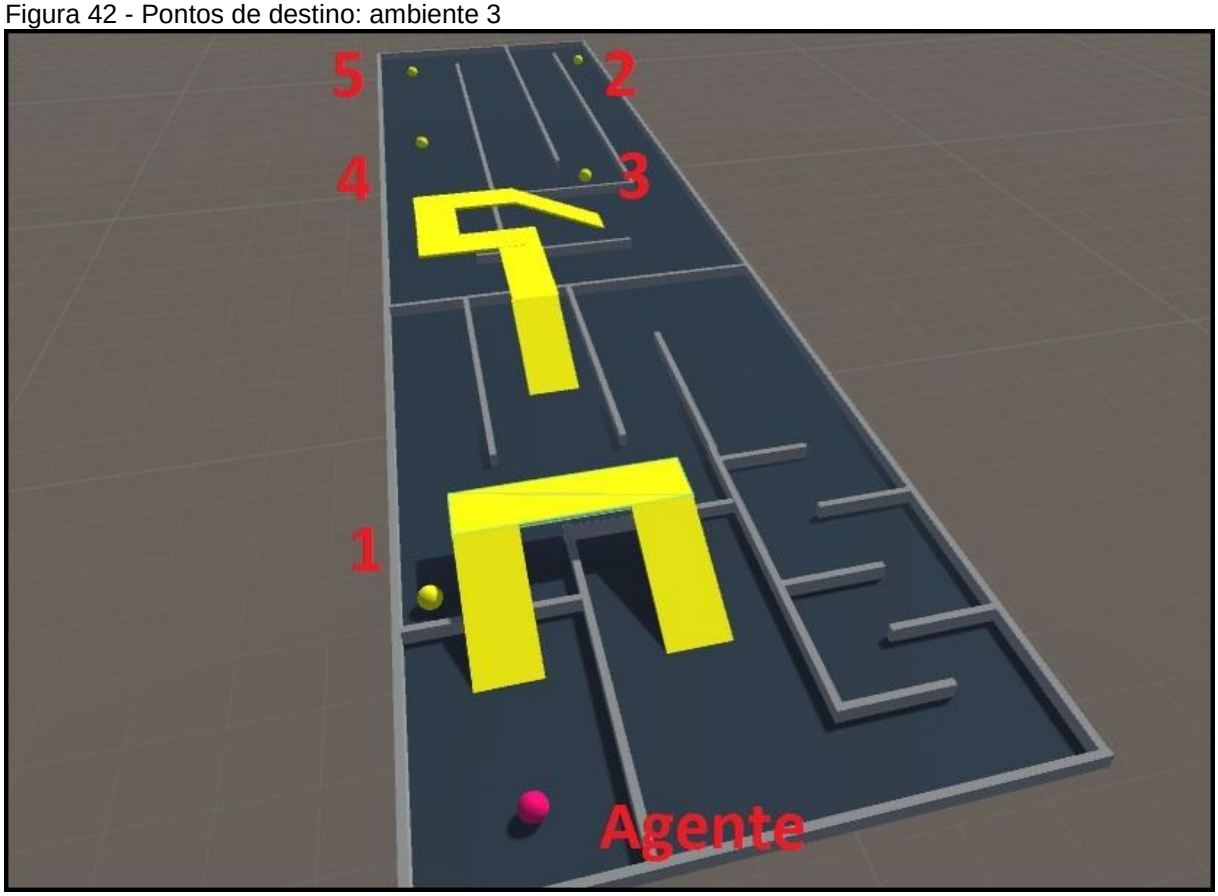
Fonte: Do autor.

Figura 41 - Pontos de destino: ambiente 2



Fonte: Do autor.

Figura 42 - Pontos de destino: ambiente 3



Fonte: Do autor

6.2 RESULTADOS OBTIDOS

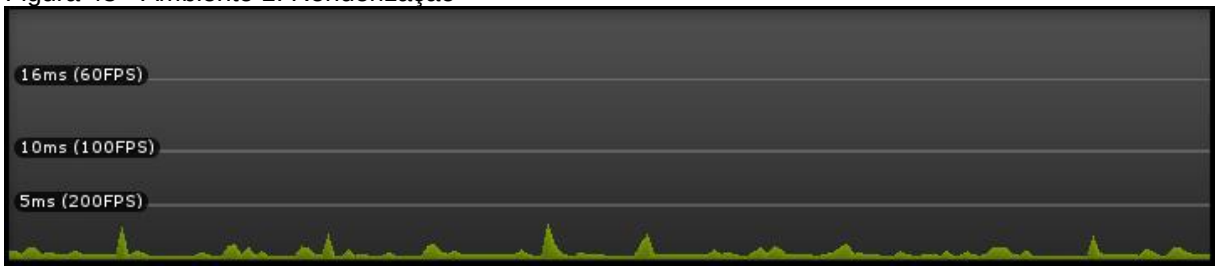
Os resultados foram obtidos somando-se os valores de cada *frame* e dividindo-se pelo número total de *frames* construídos após todas as execuções, obtendo-se a média destes valores. Entretanto, de acordo com os parâmetros de avaliação qualitativa do FPS demonstrados na tabela 2, para valores acima de 200 FPS, a performance é considerada Excelente. Logo, o cálculo é dispensado para médias acima desse valor visto que o resultado não irá alterar a classificação.

A coleta referente à taxa de atualização dos *frames, é* dividida nas métricas de sincronia e renderização. A partir da definição dos gráficos de renderização observou-se em tempo de execução as áreas de maior complexidade em que o agente A* navegou, e com os gráficos de sincronia foi possível analisar se houveram quebras de renderização relativas à estrutura dos ambientes afetando o desempenho da navegação do agente A*. Além disto, gráficos referentes ao consumo de memória RAM permitiram a observação do nível de complexidade dos recursos utilizados. Portanto, para cada gráfico, foram descritas as suas propriedades e considerações.

6.2.1 Ambiente 1: Labirinto

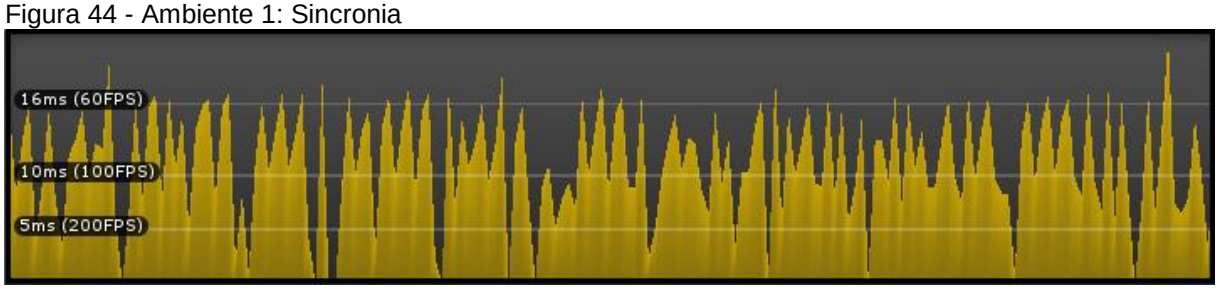
Analisando os dados obtidos no primeiro dos três ambientes, a taxa de renderização estabeleceu-se acima de 200 FPS durante as dez execuções de busca do agente A*. No entanto a sincronia sofreu variações entre 60 e 200 FPS, e a memória alocada foi de 297MB. Estes dados são apresentados nas figuras 44, 45 e 46.

Figura 43 - Ambiente 1: Renderização



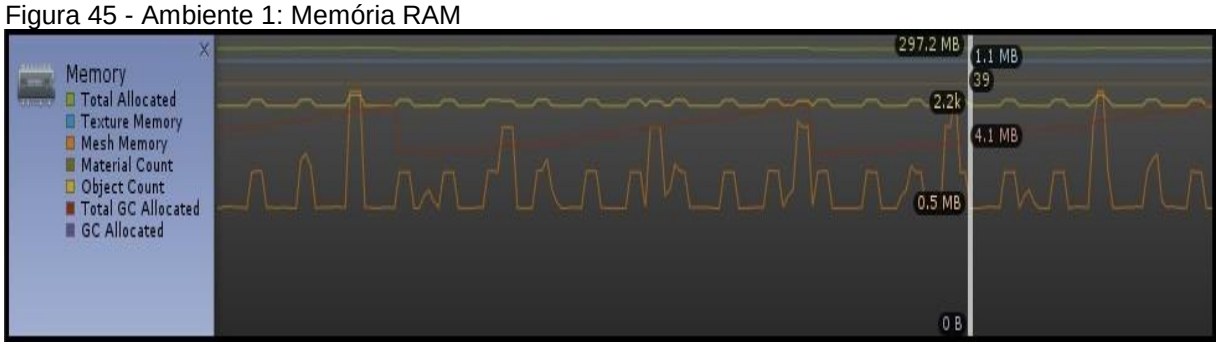
Fonte: Do autor.

Figura 44 - Ambiente 1: Sincronia



Fonte: Do autor.

Figura 45 - Ambiente 1: Memória RAM

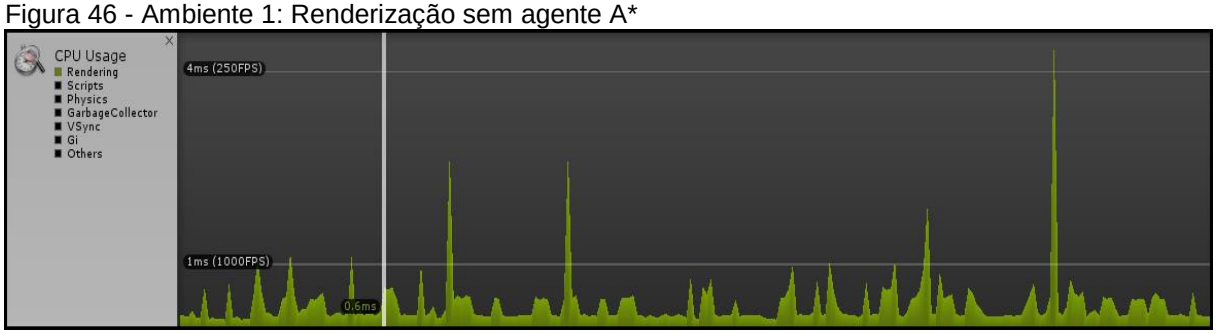


Fonte: Do autor.

Após a obtenção destes resultados, dez execuções extras foram realizadas na ausência do agente A* e de um destino com o objetivo de avaliar a diferença de resultados.

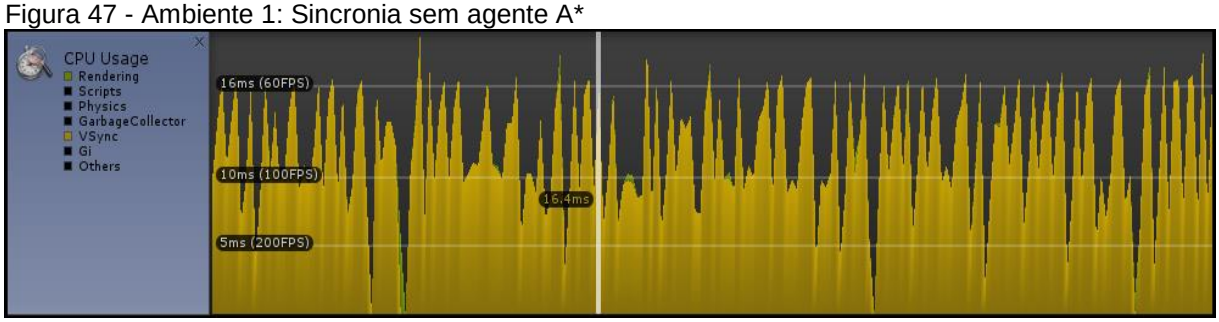
É visto que a taxa de renderização teve melhora significativa em relação às execuções anteriores, estabelecendo-se acima de 1000 FPS. Observou-se que a ausência do agente A* por meio de um objeto 3D controlado pelo NavMesh Agent afetou a renderização, já a sincronia acompanhou os resultados anteriores variando entre 60 e 200 FPS. A alocação de memória sem o agente A* foi de 294MB. Estes resultados são demonstrados nas figuras 47, 48 e 49.

Figura 46 - Ambiente 1: Renderização sem agente A*



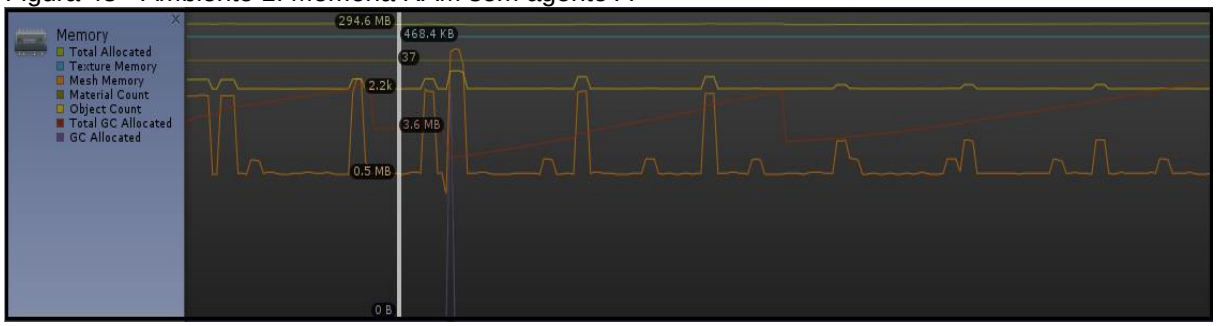
Fonte: Do autor.

Figura 47 - Ambiente 1: Sincronia sem agente A*



Fonte: Do autor.

Figura 48 - Ambiente 1: Memória RAM sem agente A*

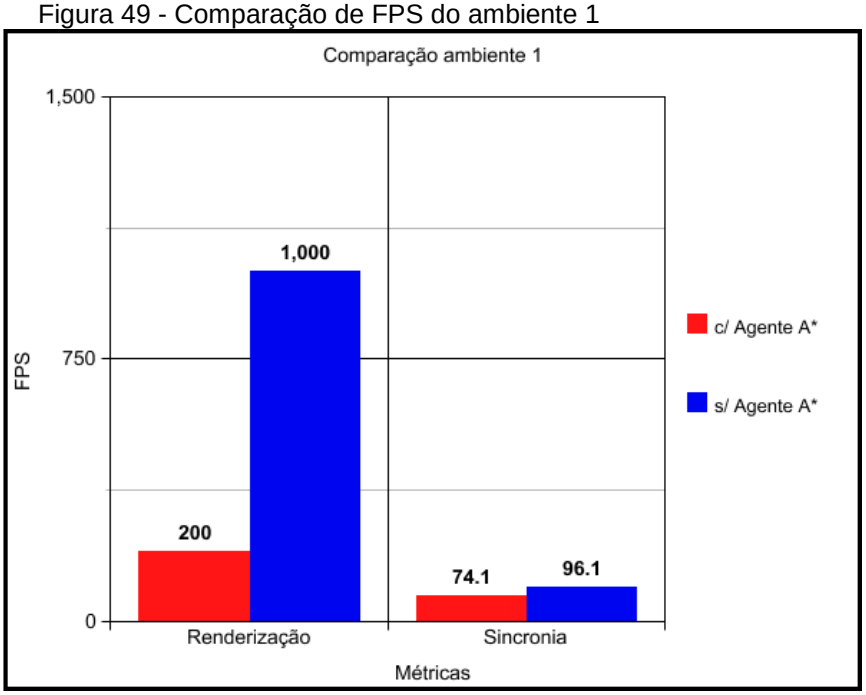


Fonte: Do autor.

A figura 50 apresenta um gráfico comparativo do desempenho do FPS referentes ao ambiente 1. Dentre os limites de 0 e 1500 definidos nos gráficos, a renderização obteve valores de 200 nas execuções com o agente A*, e 1000 nas execuções sem o agente A*. Já na sincronia, as médias foram obtidas por meio do cálculo da média aritmética, somando-se os valores dos *frames* e dividindo-se pela quantidade total de *frames* obtidos em cada ambientes.

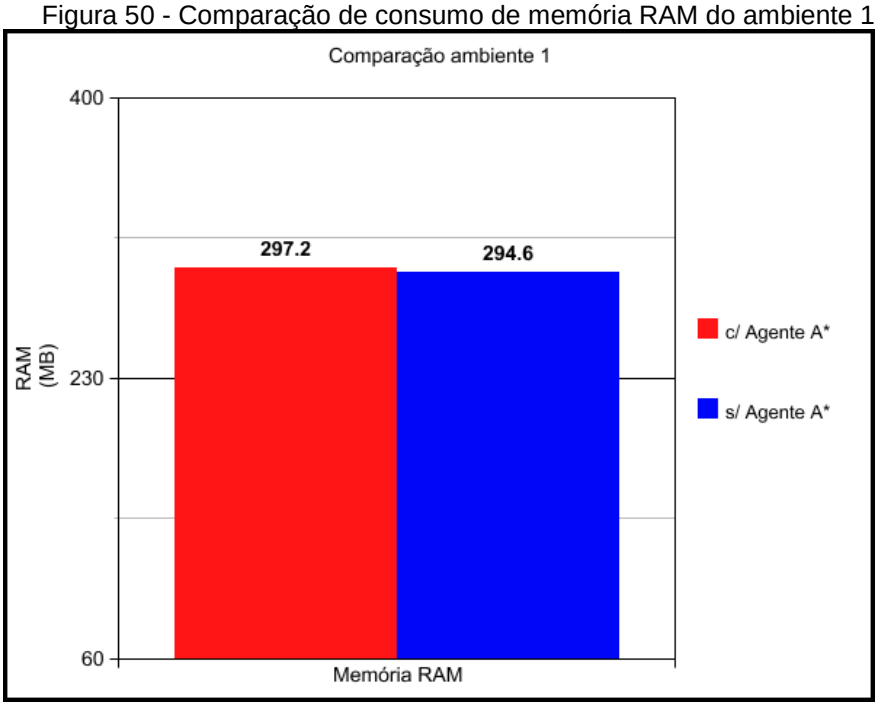
Na figura 51 é demonstrado o consumo de memória RAM comparando-se valores com e sem o agente A*. Desta forma, foi possível observar a demanda do agente A* referente a este recurso.

Figura 49 - Comparação de FPS do ambiente 1



Fonte: Do autor.

Figura 50 - Comparação de consumo de memória RAM do ambiente 1



Fonte: Do autor.

6.2.2 Ambiente 2: Montanhas

Seguindo os modelos de representação gráfica do primeiro ambiente, no segundo ambiente observou-se uma queda na taxa de renderização devido a maior complexidade na estrutura do ambiente. O agente A* percorreu maior quantidade de vértices, e alguns dos pontos de destino foram posicionados em áreas de alta complexidade de objetos renderizáveis. Pois como mencionado, no ambiente 2 os bloqueios foram dispostos em forma de montanhas, caracterizando a presença de objetos 3D multipoligonais. No entanto, os valores de sincronia acompanharam os valores de renderização.

A figura 52 compreende os dados de renderização (verde) e sincronia (amarelo) sobrepostos a fim de demonstrar este acompanhamento. Já a figura 53 é relativa ao uso da memória RAM necessário à estas execuções.

Figura 51 - Ambiente 2: Renderização e sincronia

Fonte: Do autor.

Figura 52 - Ambiente 2: Memória RAM

Fonte: Do autor.

As figuras 54, 55 e 56 apresentam os gráficos referentes às execuções sem o agente A* no ambiente 2. No entanto, a renderização e a sincronia estão em gráficos distintos, visto que os valores não se ajustaram

como visto no gráfico das execuções do agente A*. Nas figuras 57 e 58, são vistos os gráficos comparativos após o tratamento dos dados obtidos.

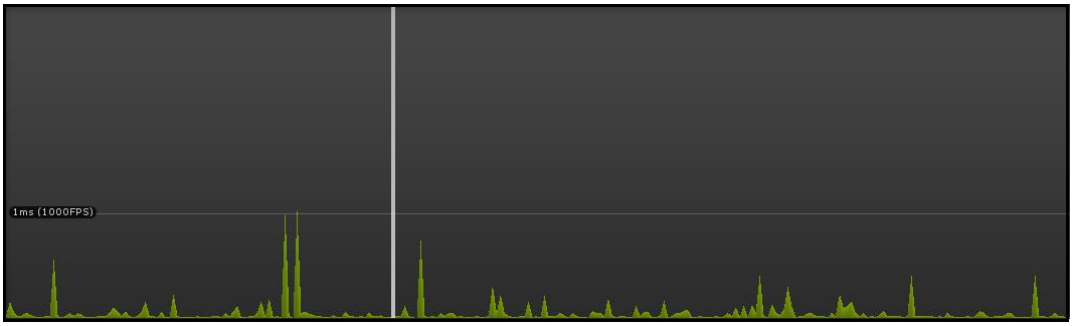
Como ilustrado na figura 54, no gráfico da renderização obteve-se diversas variações ao longo dos *frames*. No entanto os valores se mantiveram abaixo de 200 FPS com quedas a 100 e 60 FPS.

Figura 53 - Ambiente 2: Renderização sem o agente A*

Fonte: Do autor.

Os dados obtidos referentes à sincronia, demonstram valores superiores a 1000 FPS ao longo das execuções sem o agente A*. Como ilustrado na figura 55, a sincronia foi superior comparado as execuções com a presença do agente A*.

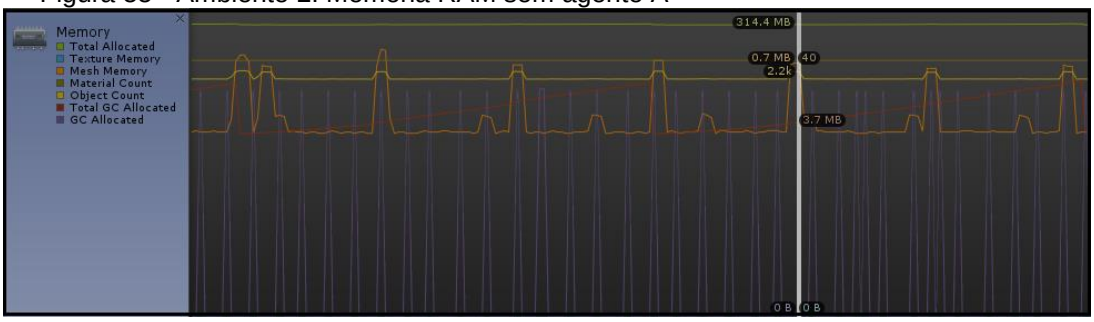
Figura 54 - Ambiente 2: Sincronia sem agente A*



Fonte: Do autor.

Entretando, a demanda do recurso de memória RAM sem o agente A* não sofreu a variação como esperado. Demonstrando que a renderização e os elementos do ambiente exigem maior quantidade de recurso.

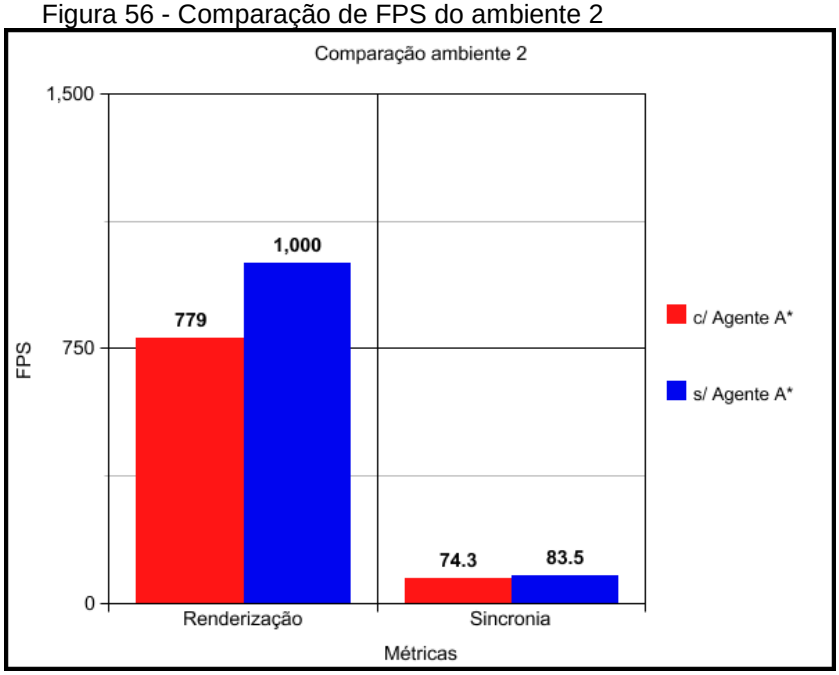
Figura 55 - Ambiente 2: Memória RAM sem agente A*



Fonte: Do autor.

A seguir, são ilustrados os gráficos referentes à comparação da taxa de renderização e sincronia na presença e na ausência do agente A*. Observa-se que no ambiente 2, a renderização estabeleceu-se com uma média superior a 750, tendo maior proximidade com os valores sem o agente A*. As colunas da direita no gráfico comparam as médias de sincronia nas duas situaçoes.

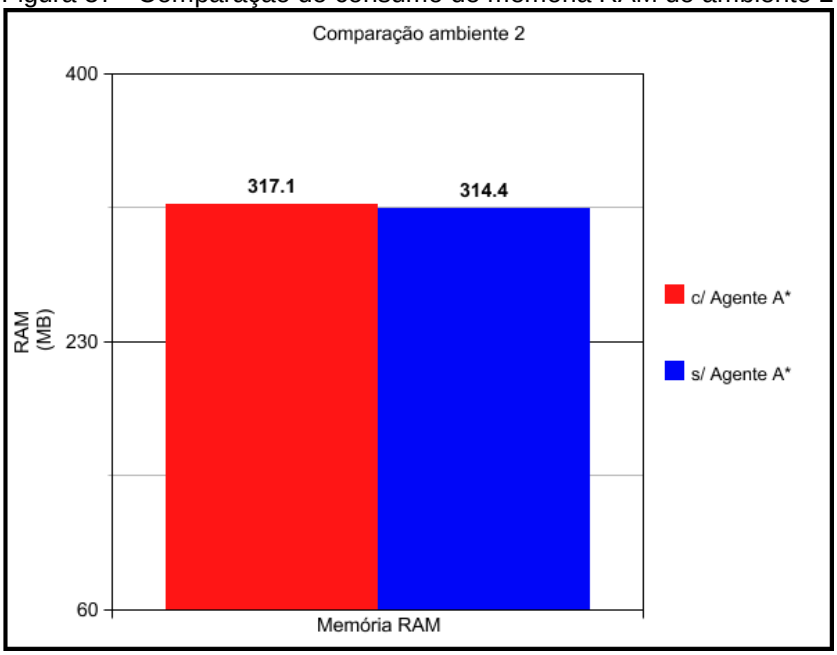
Figura 56 - Comparação de FPS do ambiente 2



Fonte: Do autor.

Na figura 58 é ilustrado a comparação da demanda de memória RAM nas duas situações, confirmando que o ambiente e os objetos 3D demandam maior reserva do recurso que o agente A*.

Figura 57 - Comparação do consumo de memória RAM do ambiente 2

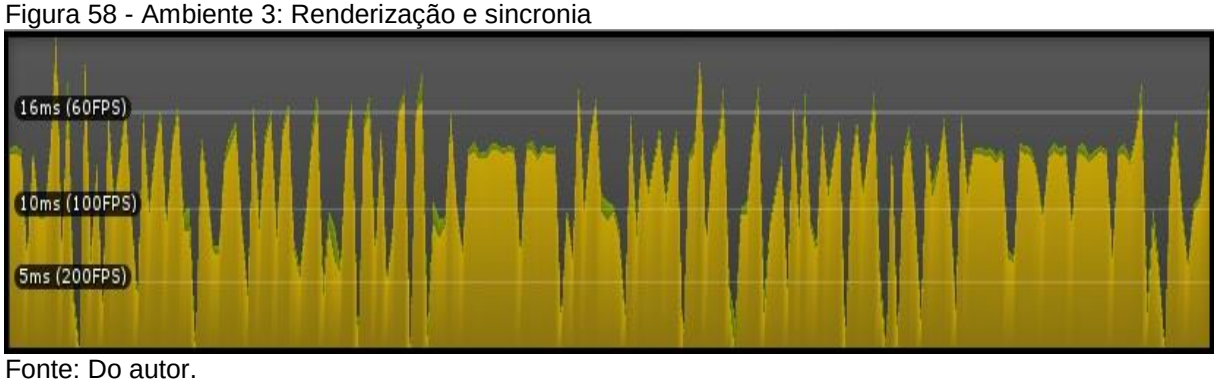


Fonte: Do autor.

6.2.3 Ambiente 3: Corredor com rampas

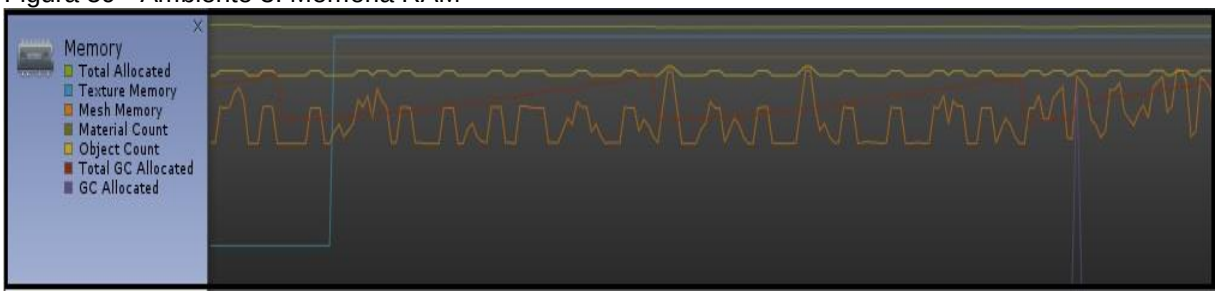
No último dos ambientes, um dos aspectos que contribuiu para a variação da taxas de FPS foi a presença de vértices no eixo vertical (y). As rampas são objetos 3D de múltiplos polígonos, exigindo maior renderização e consequentemente afetando a sincronia. A figura 59 ilustra os valores de renderização e sincronia, sobrepostos no mesmo gráfico, pois os valores se mantiveram proporcionais. Na figura 60 é demonstrado o uso de memória RAM no terceiro ambiente.

Figura 58 - Ambiente 3: Renderização e sincronia



Fonte: Do autor.

Figura 59 - Ambiente 3: Memória RAM

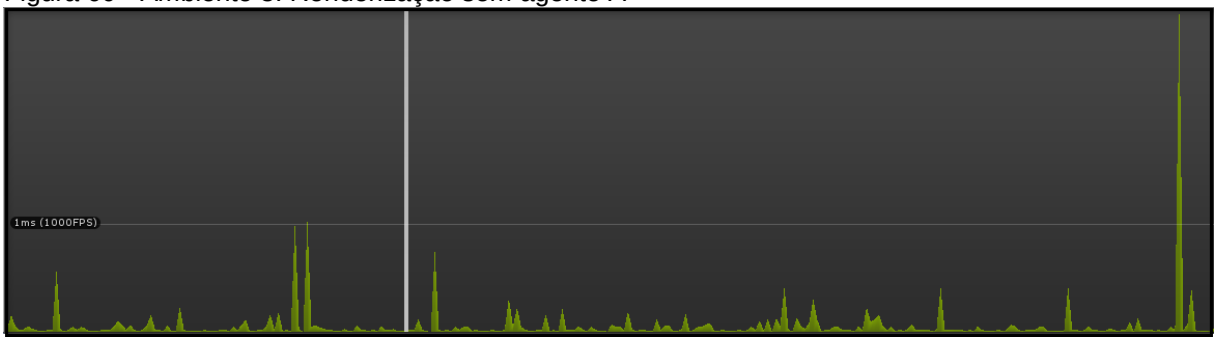


Fonte: Do autor.

No ambiente 3, a taxa de renderização na ausência das execuções do agente A* seguiu as variações dos dois primeiros ambientes, confirmando que a navegação afetou o desempenho dos demais recursos de jogo.

Os resultados do ambiente 3 na ausência do agente A* são demonstrados nas figuras 61, 62 e 63. Já o gráfico comparativo entre os valores obtidos é ilustrado nas figuras 64 e 65.

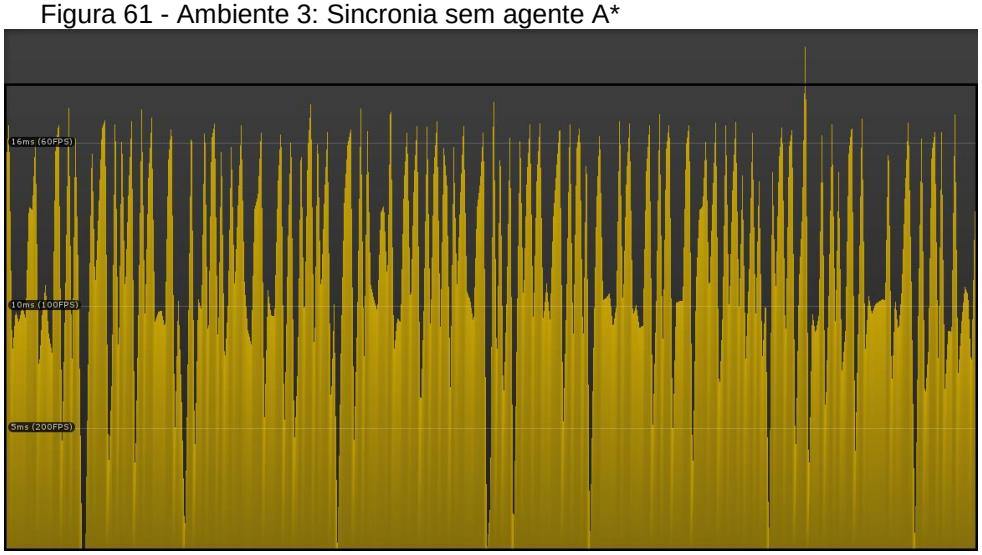
Figura 60 - Ambiente 3: Renderização sem agente A*



Fonte: Do autor

Como observado nos resultados dos ambientes anteriores, a renderização na ausência do agente A* tende a se manter acima de 1000 FPS. Logo, pressume-se que a ausência de objetos dinâmicos como o agente contribuiu para a obtenção destes valores.

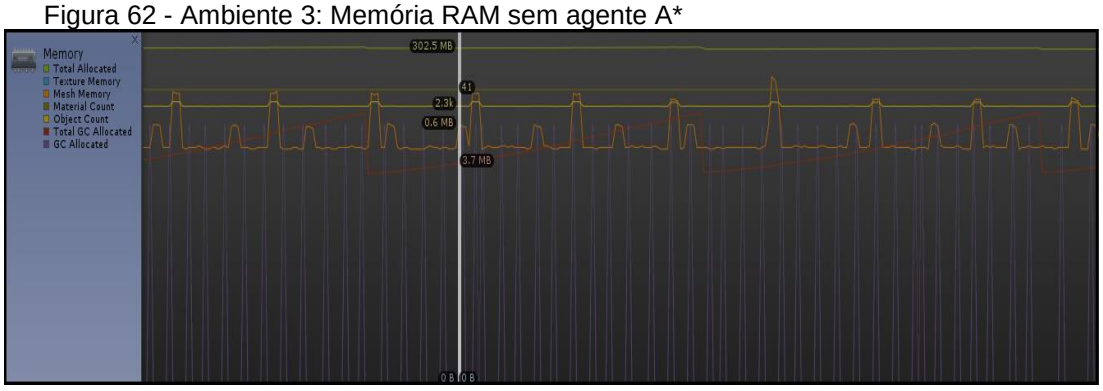
Figura 61 - Ambiente 3: Sincronia sem agente A*



Fonte: Do autor

Na ilustração do gráfico obtido na figura 62, observa-se que a variação da sincronia se deu de forma semelhante aos ambientes anteriores, com quedas a valores inferiores a 60 FPS.

Figura 62 - Ambiente 3: Memória RAM sem agente A*



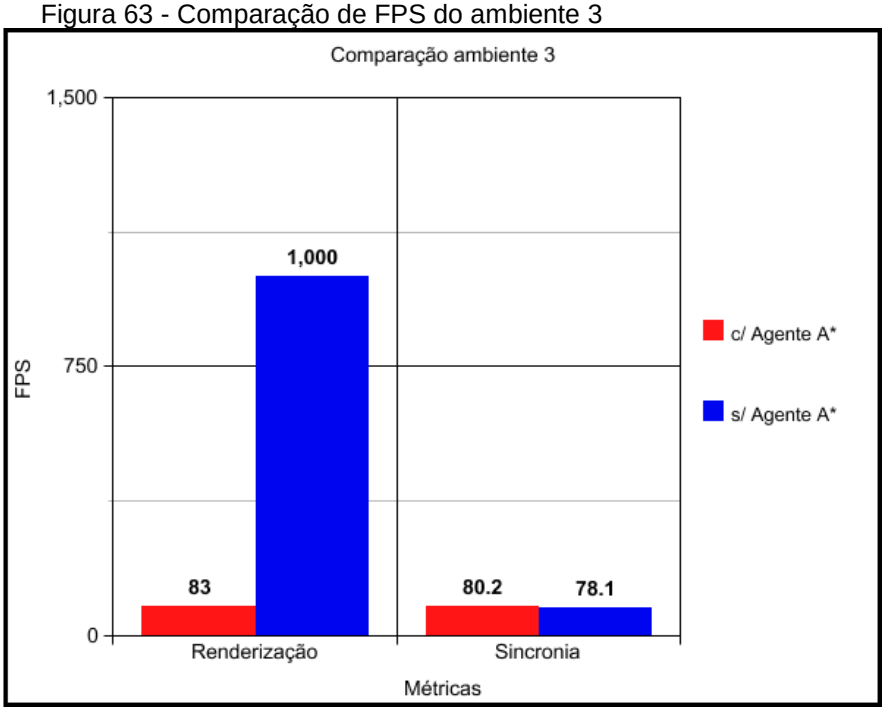
Fonte: Do autor

Assim como nos ambientes 1 e 2, no ambiente 3 o consumo de memória em sua maioria é proveniente principalmente da demanda dos objetos do ambiente, reservando um valor menor ao agente A*.

Já na figura 64, são apresentados os valores referentes a comparação das médias obtidas no ambiente 3. Assim como no ambiente 2, a inclusão de elementos que contribuem para o aumento da complexidade da

navegação afetaram as taxas de renderização nas execuções com o agente A*.

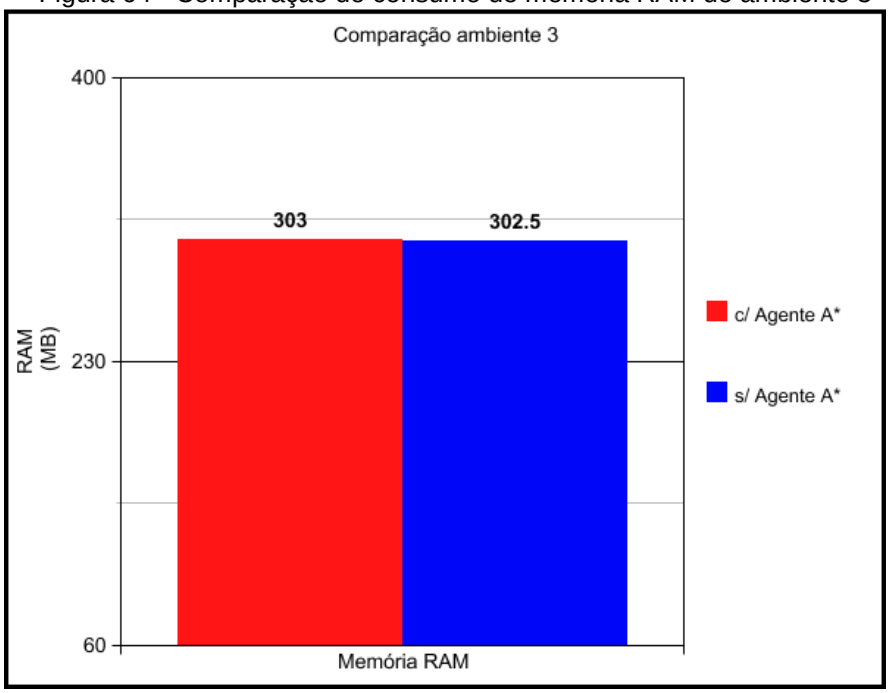
Figura 63 - Comparação de FPS do ambiente 3



Fonte: Do autor.

A comparação do consumo de memória RAM no ambiente 3 considerando as duas situações de execução, obeteve diferença da média de apenas 0.5 MB.

Figura 64 - Comparação do consumo de memória RAM do ambiente 3



Fonte: Do autor

6.2.4 Discussão dos Resultados

Ao analisar os resultados obtidos, tornou-se evidente que a variação do desempenho do agente A* nos ambientes foi ínfima dado as configurações de hardware utilizadas durante o processo. Considerando que os vértices navegados pelo agente A* foram dispostos em ambientes de complexidade gráfica baixa, nem mesmo a navegação envolvendo o eixo Y no caso do ambiente 2 causou maiores variações de desempenho. Outro ponto a se considerar, foi a busca do agente A* pelo Ponto 4 do ambiente 1. Neste caso, o objeto destino foi isolado e totalmente cercado por bloqueios impossibilitando o sucesso da busca. No entanto, o agente obteve sucesso em determinar o caminho até o vértice mais próximo do bloqueio, permanecendo no mesmo vértice até o final da execução. Esta ação, não originou resultados imprevistos, e consequentemente o desempenho se mostrou estável comparando-se a outros destinos.

Um dos aprendizados diante deste cenário de resultados, é que primeiramente os ambientes criados no motor Unity3D devem envolver

estruturas mais complexas. A navegação em um ambiente que explore a física poderia influenciar no desempenho da técnica A* utilizada. Outra possibilidade, é fazer o uso de tecnologias gráficas de terceiros tornando viável a construção de cenários mais elaborados e complexos.

Diante disto, três gráficos demonstrados nas figuras 66, 67 e 68 foram elaborados para facilitar a visualização dos resultados discutidos nesta seção, além das características gerais dos ambientes na tabela 5.

Observa-se que com exceção da taxa de renderização, não se obteve grandes variações diante da ausência do agente A*.

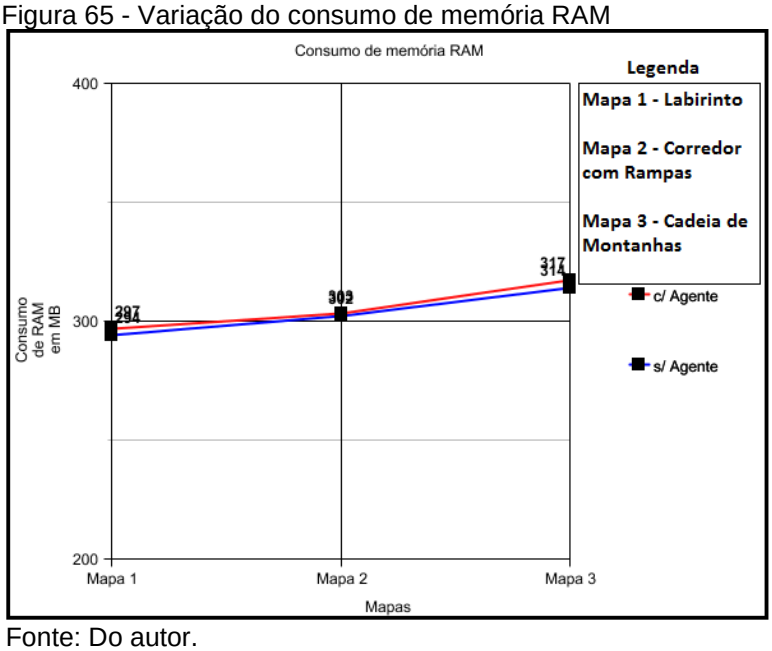
Tabela 5 – Características dos ambientes

	Labirinto	Cadeia de montanhas	Corredor com rampas
Vértices	7100	14700	9700
Objetos	16	38	26
Tamanho	10x10	50x50	30x7

Fonte: Do autor.

A partir dos gráficos de variação do consumo de memória RAM, observou-se que apesar de valores diferentes para cada ambiente, a menor variação de consumo se deu no ambiente 2.

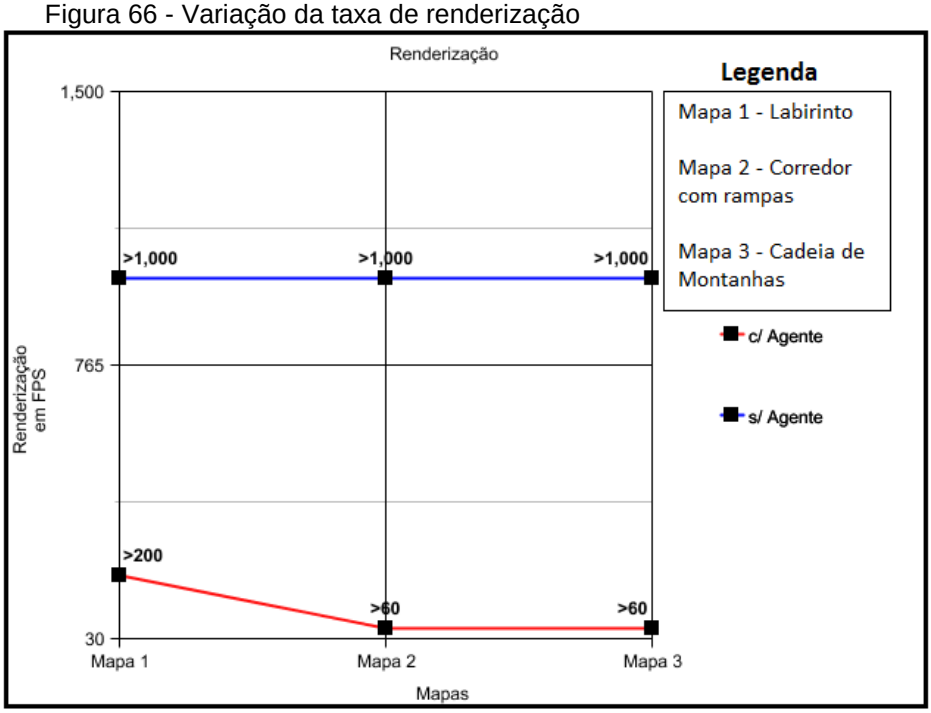
Figura 65 - Variação do consumo de memória RAM



Fonte: Do autor.

O gráfico da variação da taxa de renderização confirmou que as execuções da navegação do agente A* nos ambientes de maior complexidade obtiveram valores menores.

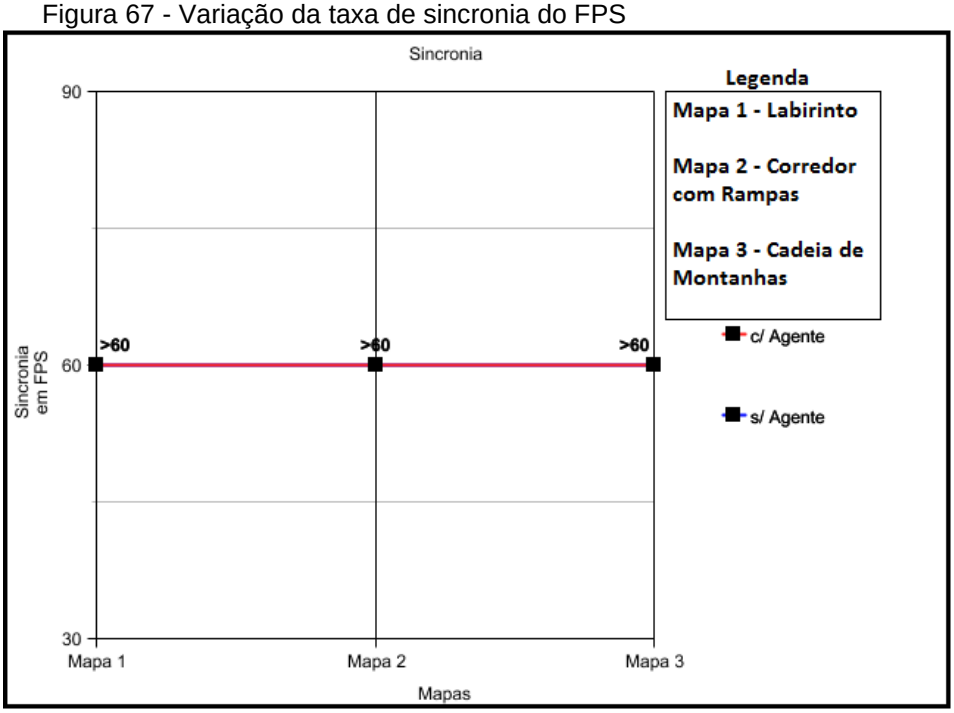
Figura 66 - Variação da taxa de renderização



Fonte: Do autor.

Considerando os limites entre 30 e 90 FPS, para todos os casos em todas as execuções os valores estabeleceram-se superiores a 60 FPS nos três ambientes.

Figura 67 - Variação da taxa de sincronia do FPS



Fonte: Do autor.

Considerando os parâmetros de medição de performance demonstrados na tabela 4, o desempenho das execuções é mensurada a partir da coleta de dados referente à complexidade do ambiente. Logo, a análise das execuções com o agente A* é demonstrada na tabela 6.

Tabela 6 – Avaliação do agente A*

Ambiente	**Renderização**	**Sincronia**
Labirinto	Excelente	Bom
Montanhas	Excelente	Bom
Corredor com rampas	Bom	Bom

Fonte: Do autor.

Com a avaliação qualitativa baseada nestes parâmetros apresentados, observou-se que no ambiente 3 a caracterização da navegação por meio dos vértices do eixo Y afetou o desempenho da renderização, classificada como boa. Entretanto a classificação da avaliação referente aos ambientes Labirinto e Montanhas é justificada pela navegação em ambientes

de baixa complexidade além de menor quantidade de vértices navegáveis. Logo a renderização foi classificada como Excelente para os dois ambientes.

Considerando a sincronia como a taxa de renderização em função do tempo, a presença do agente A* por meio de um objeto dinâmico contribuiu para a limitação da média dos valores obtidos referentes à sincronia. No entanto, ainda assim sua média possibilitou classificar esta métrica como boa nos três ambientes.

7 CONCLUSÃO

A aplicação de uma heurística de *path-planning* nos ambientes desenvolvidos proporcionou a experimentação de diferentes recursos referentes ao motor de jogo, ampliando o campo de pesquisa durante a fundamentação teórica e aplicação prática nas áreas de algoritmos planejadores, desenvolvimento de jogos digitais, motores de jogo e inteligência artificial.

Considerando as abordagens realizadas ao longo deste trabalho, o empenho se manteve com foco principal na avaliação da aplicação do algoritmo A* nos ambientes de jogo, determinando a pesquisa acerca das ferramentas disponibilizadas pelo motor Unity3D a fim de se compreender os resultados obtidos.

A aplicação do NavMesh Agent se mostrou de fundamental importância para a avaliação desta técnica, uma vez que o algoritmo A*, atualmente é utilizado como principal técnica de resolução de problemas relacionados ao planejamento de caminho em determinados gêneros. A integração deste recurso com sua biblioteca de métodos, técnicas e convenções para a navegação do agente A*, facilitaram o sucesso de sua aplicação nos três ambientes desenvolvidos.

Para tanto, algumas métricas de avaliação foram aplicadas a fim de se obter dados para avaliação comparativa entre os elementos que incorporam o jogo, e o agente inteligente, pois a eficiência da busca do menor caminho do agente inteligente em um jogo, é proporcional à qualidade do caminho percorrido (RABIN, 2012). Logo, foram definidas métricas como o uso de memória, a taxa de renderização gráfica e a taxa de sincronia do FPS que remetem à avaliação qualitativa.

Os resultados obtidos foram essenciais para a compreensão da influência de um agente A* em cenários específicos. Concluiu-se que a presença de um objeto agente alterou minunciosamente os valores dos recursos utilizados, sendo indispensável a construção de ambientes de qualidade. Na importância destes testes, é visto que a produção de jogos de

empresas famosas como a Ubisoft, utilizam-se de protótipos em baixa escala na produção de ambientes de testes, a fim de se obter resultados que sirvam de parâmetro para a aplicação da tecnologia em ambientes robustos para disceminação comercial.

Inicialmente, dificuldades relacionadas ao comportamento do agente em relação a alguns objetos 3D em específico, serviram para aprofundamento da pesquisa e estudo das malhas 3D que incorporam um objeto permitindo a colisão entre os mesmos. Entretanto, as dificuldades gerais se mantiveram principalmente na busca e aprendizado sobre um tema que envolve outra metodologia de estudo e desenvolvimento. Além disto, o contato com uma *engine* desconhecida contribuiu para a aquisição de conhecimento sobre a modelagem de ambientes de jogo, organização e utilização de recursos.

A implementação do algoritmo A* vinculado a um agente com o auxílio da ferramenta NavMesh Agent proporcionou que sua aplicação nos ambientes criados obtivessem resultados de seu desempenho. No entanto, a discussão se deu acerca da influência do agente A* nestes resultados tendo em vista os diferentes graus de complexidade dos ambientes.

Na análise dos resultados perante os parâmetros de medição de performance, o agente A* foi classificado como Excelente na maioria dos casos. Entretanto, a exploração de objetos 3D multipoligonais, bem como a navegação por meio do eixo Y como visto no ambiente 2, afetou as taxas de sincronia do FPS das execuções contribuindo na redução da qualidade das mesmas.

Ressalta-se que o algoritmo A* soluciona a busca do menor caminho dentre os vértices de um ambiente de jogo de maneira eficiente, porém a diversidade de recursos disponíveis na criação de ambientes de jogo tende a influenciar no seu desempenho de de maneira que os testes devão ser realizados de acordo com cada situação.

A área de Jogos Digitais e da Inteligência Artificial impõe em diversos aspectos, possíveis correlações e expansões das tecnologias existentes. No entanto, ressalta-se que pesquisas futuras neste âmbito específico são praticáveis de diversas formas:

a) explorar uma das diversas sucessões do algoritmo A*, como por exemplo o IDA*;
b) utilizar-se de diferentes *engines* por meio de comparação da tecnologia NavMesh;
c) implementar ambientes de alta complexidade e aplicar diferentes custos de navegação para determinadas áreas;
d) implementar um jogo por completo, incluindo conjuntos de regras por meio de Máquinas de Estados e testar diferentes heurísticas *de path-planning*;
e) ampliar o recurso NavMesh para uma abordagem de Sistemas Multiagentes e comparar diferentes heurísticas de *path-planning*.

REFERÊNCIAS

ABATH, Daniel. O Desenquadramento Possível: Aspectos do Cotidiano em Gamescapes. Em: CONGRESSO DE CIÊNCIAS DA COMUNICAÇÃO NA REGIÃO NORDESTE, 13., 2011, João Pessoa. **Anais... .** João Pessoa: Congresso de Ciências da Comunicação na Região Nordeste, 2011. p. 1 - 12.

ADAMS, Ernests. **Fundamentals of Game Design.** 3. ed. New York: Pearson Education, 2014. 560 p.

BATTAIOLA, André Luiz - Jogos por Computador – Histórico, Relevância Tecnológica e
Mercadológica, Tendências e Técnicas de Implementação - XIX Jornada de Atualização em Informática/SBC – 2000.

BITTENCOURT, João Ricardo; GIRAFFA, Lucia Maria. **Role-Playing Games, Educação e Jogos Computadorizados na Cibercultura.** 2005. 14 f. TCC (Graduação) - Curso de Ciência da Computação, Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, 2005.

BJORNSSON, Yngvi; HALLDÓRSSON, Kári. Improved Heuristics for Optimal Pathfinding on Game Maps. 2006. Reykjavik University, Reykjavik, Iceland, 2006.

BLUM, Avrim L.; FURST, Merrick L.. **Fast Planning Through Planning Graph Analysis.** Pittsburgh: School Of Computer Science Carnegie Mellon University, 1997. 20 p. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=3509A9A72B48F1E0B8A2006A478CF89B?doi=10.1.1.43.7263&rep=rep1&type=pdf>. Acesso em: 25 out. 2014.

BURN, Andrew. **Game AI Dynamic Path Planning.** 2003. 79 f. TCC (Graduação) - Curso de Artificial Intelligence, University Of Durham, Durham, 2003.

CARR, Diane et al. **Computer Games: Text, Narrative and Play.** Engelska: Wiley, 2014. 224 p.

COUCHOT, E., 2003. A tecnologia na arte: da fotografia à realidade virtual. Trad. Sandra Rey. Porto Alegre: Editora da UFRGS.

CRAWFORD, Chris. Storytelling. In: CRAWFORD, Chris. **Chris Crawford on game design.** S.i.: New Riders Publishing, 2003. Cap. 11. p. 157-168.

DUARTE, Luiz Cláudio S.. **Jogos de Tabuleiro no Design de Jogos Digitais.** Brasília: Sbc, 2012. 6 p. Disponível em:

<http://base.gamux.com.br/events/2012.11.02-SBGames12/proceedings/papers/artedesign/AD_Full17.pdf>. Acesso em: 14 mar. 2015.

FUNGE, John David. **Artificial Intelligence for Computer Games:** an introduction. Massachussetts: A K Peters-wellesley, 2004. 146 p.

HOANG, Hai; LEE-URBAN, Stephen; MUÑOZ-AVILA, Héctor. **Hierarchical Plan Representations for Encoding Strategic Game AI.** Bethlehem: Department Of Computer Science & Engineering, 2005. 6 p. Disponível em: <http://www.aaai.org/Papers/AIIDE/2005/AIIDE05-011.pdf>. Acesso em: 19 out. 2014.

HORSWILL, Ian. **Game Design for Classical AI.** Evanston, Illinois: Aaai, 2014. 7 p. Disponível em: <http://www.exag.org/papers/exag2014_submission_12.pdf>. Acesso em: 13 maio 2015.

HUNICKE, Robin; LEBLANC, Marc; ZUBEK, Robert. MDA: A Formal Approach to Game Design and Game Research. Em: AAAI Workshop on Challenges in Game AI, 2004.

JAKLIN, Norman; VAN TOLL, Wouter; GARAERTS, Roland. **Way to go - A framework for multi-level planning in games.** Utrecht: , Department Of Information And Computing Sciences, 2011. 4 p. Disponível em: <http://icaps13.icaps-conference.org/wp-content/uploads/2013/05/pg13-proceedings.pdf#page=11>. Acesso em: 11 out. 2014.

KIAW, Aung Sithu; PETERS, Clifford; SWE, Thet Naing. **Unity 4.x Game AI Programming:** Learn and implement Game AI in Unity3D with a lot of sample projects and next-generation techniques to use in your Unity3D projects. Mumbai: Packt Publishing, 2013. 213 p.

LANKOSKI, Petri; BJÖRK, Staffan. **Gameplay Design Patterns for Believable Non-Player Characters.** Helsinki: University Of Art And Design Helsinki, 2006. 8 p. Disponível em: <http://xn--h1ai1d.xn--80aqald0c0e.xn--_-8sbfcf2byalst0g.homepage.auditory.ru/2006/Ivan.Ignatyev/DiGRA/Gameplay Design Patterns for Believable Non-Player Character.pdf>. Acesso em: 10 nov. 2014.

LEWIS, Michael; JACOBSON, Jeffrey. **Game Engines in Scientific Research.** Pittsburgh: Communications Of The Acm, 2002. 5 p. Disponível em: <http://publicvr.info/publications/Lewis2002.pdf>. Acesso em: 01 jun. 2015.

LUCCHESE, Fabiano; RIBEIRO, Bruno. **Conceituação de Jogos Digitais.** Campinas: Universidade Estadual de Campinas, 2009. 16 p. Disponível em:

<http://www.dca.fee.unicamp.br/~martino/disciplinas/ia369/trabalhos/t1g3.pdf>. Acesso em: 14 nov. 2014.

LUGER, George F.. **Inteligência Artificial.** 6. ed. São Paulo: Pearson Education do Brasil, 2014. 614 p.

LUIZ, Bruno Nepomuceno. **Aplicação de Planejamento em Jogos de Estratégia.** 2008. 88 f. TCC (Pós-Graduação) - Curso de Ciência da Computação, Universidade Federal de Uberlândia, Uberlândia, 2008.
MARQUES, Jane A.; CEONI, Karina Trajano. **O Histórico e a Importância da Mídia Digital nos Jogos On-Line: dos RPGs tradicionais para os CRPGs e os On-Line.**São Paulo: V Congresso Nacional de História da Mídia, 2007.

MACHADO, Marlos C.; PAPPA, Gisele L.; CHAIMOWICZ, Luiz. **Characterizing and Modeling Agents in Digital Games.**Belo Horizonte: Federal University Of Minas Gerais, 2012. 8 p. Disponível em: <http://homepages.dcc.ufmg.br/~chaimo/public/SBGames12-marlos.pdf>. Acesso em: 13 maio 2015.

MENDONÇA, André Noronha Furtado de; VALLERIUS, Denise Mallmann. O Letramento E O Ensino De Literatura Mediados Por Jogos Digitais Educacionais. In: CONGRESO ARGENTINO DE CIENCIAS DE LA COMPUTACIÓN, 18., 2013, Pelotas. **Anais... .** Pelotas: Congreso Argentino de Ciencias de La Computación, 2013. p. 574 - 583. Disponível em: <http://sedici.unlp.edu.ar/bitstream/handle/10915/32046/Documento_completo.pdf?sequence=1>. Acesso em: 09 set. 2014.

MENDONÇA, Raphael Leal. **Imersão e emoção em jogos digitais: uma abordagem a partir de sistemas especialistas, lógica fuzzy e mapas auto-organizáveis.**2012. 171 f. Dissertação (Mestrado) - Curso de Engenharia Elétrica, Universidade Presbiteriana Mackenzie, São Paulo, 2012. Disponível em: <http://tede.mackenzie.com.br//tde_busca/arquivo.php?codArquivo=2756>. Acesso em: 15 nov. 2014.

NAVES, Thiago França. **Uma Abordagem para Maximização da Produção de Recursos em Jogos RTS.** 2012. 118 f. TCC (Pós-Graduação) - Curso de Ciência da Computação, Universidade Federal de Uberlândia, Uberlândia, 2012.

NOVAK, Jeannie. **Desenvolvimento de games.** 2. ed. São Paulo: Cengage Learning, 2011. 443 p.

ORKIN, Jeff. Agent Architecture Considerations for Real-Time Planning in Games. Em: AIIDE 2004, Proceedings, Marina Del Rey, CA, 2004.

ORKIN, Jeff. Applying Goal-Oriented Action Planning to Games. AI Game Programming Wisdom 2, 217-228. Hingham, Mass.: Charles River Media, 2005.

ORKIN, Jeff. Symbolic Representation of Game World State: Toward Real-Time Planning in Games. Em: AAAI Workshop on Challenges in Game AI, San Jose, CA, 2004.

PASSOS, Erick Baptista et al. Tutorial: Desenvolvimento de Jogos com Unity 3D. In: VIII BRAZILIAN SYMPOSIUM ON GAMES AND DIGITAL ENTERTAINMENT, 8., 2009, Rio de Janeiro. **Anais... .** Rio de Janeiro: Viii Brazilian Symposium On Games And Digital Entertainment, 2009. p. 1 - 30. Disponível em: <http://sbgames.org/papers/sbgames09/computing/tutorialComputing2.pdf>. Acesso em: 14 nov. 2014.

PAULA, Bruno Campagnolo de. Mymyqui: Um Jogo Multijogador para Rotulagem de Animações, Instituto de Tecnologia do Paraná, 2010.

PAULA, Bruno Campagnolo de. **Sistema Inteligente Para Jogos de Estratégia Baseados em Turnos: Uma Abordagem Utilizando Planejamento Baseado em Casos.**2007. 118 f. Dissertação (Mestrado) - Curso de Informática, Pontifícia Universidade Católica do Paraná, Curitiba, 2007.

PEIKIDIS, Panagiotis. **StarPlanner:** Demonstrating the use of planning in a video game. 2010. 63 f. TCC (Graduação) - Curso de Computer Science, University Of Sheffield, Sheffield, 2010.

PEREIRA, Carlos E. Klimick; BETTOCCHI, Eliane. **Idade Mística: a Matéria da Bretanha no contexto do RPG.** 2013. 16 f. TCC (Graduação) - Curso de Letras, Departamento de Letras, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2013. Disponível em: <http://www.historias.interativas.nom.br/incorporais/pdfs/mistprojeto.pdf>. Acesso em: 18 nov. 2014.

PONSEN, Marc et al. Automatically Generating Game Tactics via Evolutionary Learning, Institute of Knowledge and Agent Technology, 2006.

PONSEN, Marc et al. Effective and Diverse Adaptive Game AI. Em: *IEEE Transactions on Computational Intelligence and AI in Games*, Vol. 1, No. 1, p. 16-27.

RABIN, Steve. **Game AI Pro 2:** Collected Wisdom of Game AI Professionals. London: A K Peters-wellesley, 2015. 533 p.

RABIN, Steve. **Introdução ao Desenvolvimento de Games:** Programação, técnica, linguagem e arquitetura. 2. ed. São Paulo: Cengage Learning, 2012. 637 p. (2).

ROCHA, HeloÍsa Vieira da; BARANAUSKAS, Maria CecÍlia Calani. **DESIGN E AVALIAÇÃO DE INTERFACES HUMANO-COMPUTADOR.** Campinas: Nied, 2003. 244 p.

RUSSELL, Stuart; NORVIG, Peter. **Inteligência Artificial.** 3. ed. Rio de Janeiro: Campus, 2013. 1016 p.

SIMPÓSIO BRASILEIRO DE JOGOS E ENTRETENIMENTO DIGITAL, 11., 2012, Brasília. Characterizing and Modeling Agents in Digital Games. Belo Horizonte: Simpósio Brasileiro de Jogos e Entretenimento Digital, 2012. 8 p. Disponível em: <http://homepages.dcc.ufmg.br/~chaimo/public/SBGames12-marlos.pdf>. Acesso em: 20 mar. 2015.

SCHUYTEMA, Paul. **Design de Games: Uma abordagem prática.** São Paulo: Thomson Learning, 2008. 472 p.

TERRA, Vinicius Marques. **Utilização de Ambientes Paralelos no Processo de Aprendizado de Algoritmos de Busca de Caminho em Tempo Real.** 2010. 53 f. Dissertação (Mestrado) - Curso de Ciência da Computação, Universidade Federal de Minas Gerais, Belo Horizonte, 2010.

ULTIMATEGAMETOOLS. Site Oficial do Ultimate Game Tools. Disponível em: <http://www.ultimategametools.com/products/concave_collider>. Acesso em: 21 nov. 2014.

UNITY3D. Site oficial do Unity3D (A). Manual do Usuário. Disponível em: <www.unity3d.com/support/documentation/Manual>. Acesso em: 14 nov. 2014

UNITY3D. Site oficial do Unity3D (B). Fórum. Disponível em: < forum.unity3d.com>. Acesso em: 14 nov. 2014.

UNITY3D. Site oficial do Unity3D (C). Documentação. Disponível em: < www.unity3d.com/support/documentation/tutorials>. Acesso em: 14 nov. 2014.

UNITY3D. Site oficial do Unity3D (D). Tutoriais. Disponível em: < www.unifycommunity.com/wiki>. Acesso em: 14 nov. 2014.

WELD, Daniel S.. **Recent Advances in AI Planning.** Washington: Ai Magazine, 1998. 49 p. Disponível em: <http://homes.cs.washington.edu/~weld/papers/pi2.pdf>. Acesso em: 29 out. 2014.

WOOLDRIDGE, Michael. **An Introduction to MultiAgent Systems.** West Sussex: Library Of Congress Cataloging-in-publicatio, 2002. 348 p. Disponível em: <http://coltech.vnu.edu.vn/httt/media/courses/AI++/Tai lieu/TLTK.pdf>. Acesso em: 17 nov. 2014.

APÊNDICE(S)

Apêndice A – Artigo

Avaliação do algoritmo A* na heurística de *path-planning* em ambientes de jogos utilizando o motor Unity3D

Brian C. Andrade[1], Merisandra C. De Mattos Garcia[1]

[1]Universidade Do Extremo Sul Catarinense (UNESC) – Criciúma, SC - Brasil

brian.c.a@hotmail.com, mem@unesc.net

Abstract. *With the digital games evolution, new technological features are essentials for the practical application of new features becomes acceptable. The Artificial intelligence area provides techniques that helps the development of techniques around implicit problems in games. As an example, the path planning. This planning, is based upon the use of an shortest-path algorithm, allowing the navigation in a game plan. The navigation occurs by the agent that implements the A* algorithm, known as the most popular in games, and responsible for the search technique. Thus, this article explores the results of the application of a path-planning heuristics in specific environments allowing the qualitative evaluation of the heuristic performance.*

Resumo. *Com a evolução da indústria de jogos digitais novos adventos tecnológicos são essenciais para que a aplicação prática de novos recursos seja aceitável. A área da Inteligência Artificial provê técnicas que auxiliam no desenvolvimento da tecnologia acerca de problemas implícitos nos jogos, como por exemplo o planejamento de caminho. Este planejamento, baseia-se no uso de um algoritmo de busca de menor caminho possibilitando a navegação em um plano de jogo. A navegação se da por meio de um agente que implementa o algoritmo A*, sendo o de maior popularidade nos jogos e responsável pela técnica de busca. Desta forma, este artigo explora os resultados da aplicação de uma heurística de path-planning em ambientes específicos permitindo a classificação qualitativa do desempenho da heurística.*

1. Introdução

O acelerado desenvolvimento dos jogos digitais, força as empresas a produzir ideias que para sua implementação prática, recorrem a uma nova tecnologia. Estas tecnologias visam a contribuição para uma experiência de jogo inovadora. Como exemplo, têm-se o aumento do realismo e a tentativa da reprodução do mundo real no jogo. Para que estas questões sejam satisfeitas, utiliza-se principalmente técnicas de Inteligência Artificial que tratam da resolução de problemas implícitos.

Nos jogos digitais, a presença da IA pode ser classificada de acordo com os gêneros, pois a distinção dos gêneros é feita a partir da identificação das principais características dos jogos. A ambientação, os objetivos e a jogabilidade são critérios que diferenciam os gêneros e consequentemente abordam tecnologias e técnicas de IA

diferentes. Visto que os gêneros abordam ambientes específicos, uma das técnicas de IA pertinentes a este quesito é o *path-planning*. Este trata do planejamento do menor caminho entre dois pontos. No entanto, ao trabalhar com o planejamento de caminho em ambientes específicos, deve-se considerar algumas variáveis de ambientes como os bloqueios e a navegação por meio de objetos dinâmicos. Trabalhar com estas variáveis implica em mensurar o desempenho computacional a fim de obter-se uma qualidade de navegação aceitável.

Neste artigo, é realizada a avaliação da técnica de navegação em ambientes específicos utilizando um motor de jogo.

2. Jogos Digitais

Um jogo digital é um sistema composto a partir de seu motor de jogo. Logo, o motor caracteriza a interface gráfica que agrega os aspectos tecnológicos que estão contidos em um determinado ambiente [BATTAIOLA, 2000].

No panorâma atual de desenvolvimento de jogos, segue-se uma metodologia de desenvolvimento que é composta pela subdivisão das equipes de acordo com cada processo de jogo. Considerando a robustez dos ambientes de jogos atuais, as equipes visam um método organizacional na implementação dos componentes. Ao final, obtém-se uma produção que integra diferentes recursos tecnológicos avançados dispostos a serem explorados pelo jogador [NOVAK, 2011].

2.1 Desenvolvimento de jogos

A realidade da liberação dos mais populares motores de jogo de forma gratuita permitiu que no cenário nacional os desenvolvedores independentes pudessem explorar recursos avançados. Com isso, pequenas empresas sairam do estado de consumidores secundários para a produção, acompanhando o mercado de jogos [RABIN, 2012].

2.1.1 Pré-produção

É na etapa de pré-produçao, que o desenvolvimento é responsável pela criação da ideia conceitual do jogo. Também nomeado de roteiro, este define o enfoque principal para a construção do jogo. Nesta etapa, é criado o método de aproximação do jogador com as características imersiva a partir da apresentação dos controles de jogo [NOVAK, 2011].

2.1.2 Game Design

A etapa de desenvolvimento prático inicia-se no Game Design. Nele, o trabalho se dá acerca da narrativa, diálogos, definição da ambientação e presença dos personagens.

A qualidade de desenvolvimento dos aspectos do jogo depende diretamente do Game Design. A principal relevância desta etapa de desenvolvimento está no tratamento da *User Interface*, pois lida com o método de representação das informações de jogo ao jogador [DUARTE, 2012].

2.1.3 Interface Gráfica

Atualmente, a interface gráfica gera especulação acerca das tecnologias de hardware implementadas. De acordo com a capacidade computacional, a interface gráfica

reproduz elementos de maior complexidade e consequentemente gera um ambiente mais detalhado e rico. Esta etapa lida não somente com a definição da tecnologia de renderização gráfica, mas também com uma interface que auxilia o jogador na identificação do ambiente de jogo [RABIN, 2012].

2.1.4 Inteligência Artificial

A criação de um jogo depende não somente das etapas de construção conceitual e dos ambientes gráficos, mas da capacidade de reproduzir o jogo próximo a realidade do mundo [RABIN, 2015, tradução nossa]. Com isso, a área da inteligência artificial atua na resolução de problemas intrínsecos, contribuindo com o aumento do realismo. Este realismo depende das técnicas de IA, pois cada técnica visa a resolução de um problema por vezes específico de um determinado gênero. Como exemplo, a IA compromete-se em atribuir um comportamento específico à um personagem. Ou como explorado neste artigo, problemas relacionados à busca do menor caminho em um plano de jogo.

Na figura 1 é ilustrado uma tabela relacionando diferentes técnicas de IA e alguns gêneros de jogos. Nota-se que a IA abrange diversos gêneros, ressaltando a sua importância nos jogos.

Técnicas de IA	RPG	Estratégia	FPS	Esporte	Simulador	Corrida	Luta
Máquina de Estados	X	X	X	X		X	X
Lógica Fuzzy		X	X	X			
Sistema de Mensagens	X	X	X	X		X	X
Scripts	X		X			X	X
Path-finding	X	X	X				
Hierarquia		X					
Algoritmos Genéticos		X					
Redes Baysianas					X		
Redes Neurais					X		

Figura 1. Técnicas de IA por gênero

3. Path-finding

O path-finding representa a solução de um caminho em um ambiente. Entretanto no contexto dos jogos digitais, sua estrutura pode compor uma sequência de tarefas de alta complexidade. Em ambientes de jogos, deve-se considerar a inferência das variáveis de ambiente, identificadas pela presença de obstáculos, detecção de colisões, além de

outras entidades que influenciem na busca do caminho [RUSSEL; NORVIG, 2013]. Ressalta-se ainda, que os ambientes de jogos derminam a navegação física, ou seja, a navegação se dá por meio de um objeto dinâmico que explora o plano. Estes objetos são nomeados de Agentes Inteligentes.

3.1 Agentes Inteligentes

Como mencionado, a busca do caminho em um ambiente de jogo se dá por meio de um Agente Inteligente. Os agentes seguem um ciclo teórico a partir das etapas de reconhecimento, pensamento e ação, e assim os programadores definem o seu método de busca [LUGER, 2014]. Essencialmente, o método de busca define a técnica. Uma das técnicas de maior popularidade e reconhecimento por realizar a busca de menor caminho obtendo um resultado aceitável é o algoritmo A*.

3.2 Algoritmo A*

As abordagens acerca do algoritmo A* mencionam que sua heurística é voltada para o desempenho computacional, visto que as soluções são específicas de determinado problema [RABIN, 2015, tradução nossa].

No contexto dos jogos digitais, a busca se dá por meio da representação de um grafo no plano de jogo, ou seja, o algoritmo A* se encaixa perfeitamente para aplicações desta natureza. Sua estrutura é baseada na presença de variáveis que armazenam as distâncias entre os vértices [LUGER, 2014]. Trabalhando com a lógica deste conceito, que define-se a técnica de busca do menor caminho entre dois pontos.

Na figura 2 é ilustrado um esquema representando a lógica de manipulação das variáveis do algoritmo A*.

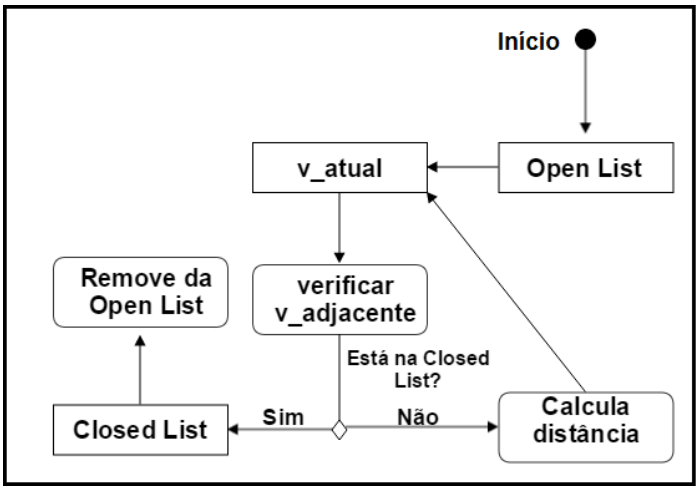


Figura 2. Algoritmo A*

4. Avaliação do algoritmo A* na heurística de path-planning em ambientes de jogos utilizando o motor Unity3D

Utilizando-se de uma heurística de path-planning para jogos, define-se a aplicação do algoritmo por meio de um agente. Na avaliação de desempenho desta heurística, implicam-se métricas computacionais afim da classificação qualitativa para cada ambiente específico.

Serão descritas as etapas metodológicas na criação de ambientes de jogo para navegação do agente A*, a implementação do agente, além da definição da etapa das execuções.

4.1. Modelagem dos ambientes 3D

A modelagem dos ambientes se deu visando explorar diferentes situações de jogo. O padrão de estruturação de cada ambiente, foi definido a partir da criação de um plano horizontal de navegação, objetos 3D que representam os bloqueios, o agente, além dos pontos de destino.

Na figura 3 é demonstrado o primeiro ambiente criado, sendo este dotado dos bloqueios, de um objeto agente, e de cinco objetos adicionais para representar seus respectivos pontos de destino.

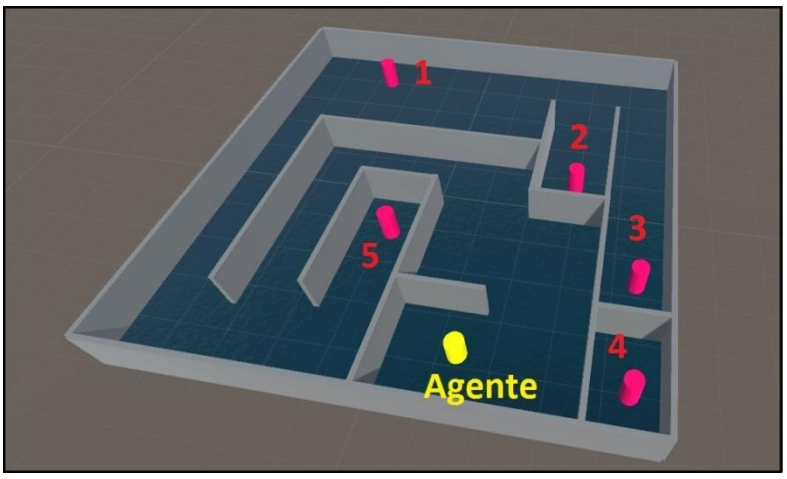


Figura 3. Ambiente 1

Como mencionado, cada ambiente visa explorar uma situação de jogo adversa. Logo, na criação do ambiente 2, optou-se por criar os bloqueios com base em objetos 3D multipoligonais que representassem montanhas, além de aplicar-se texturas ao longo do plano. Esta criação estrutural, ilustrada na figura 4, visa o aumento da complexidade do ambiente, propondo afetar o desempenho da navegação.

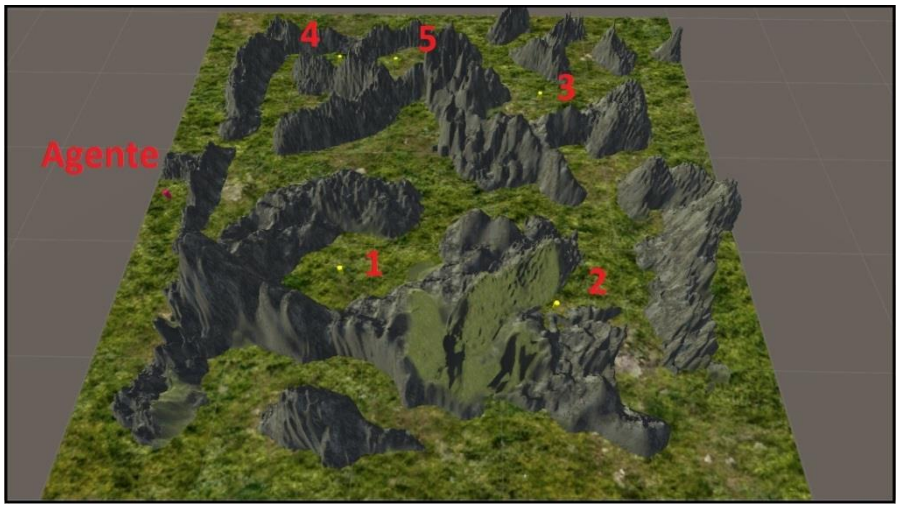


Figura 4. Ambiente 2

O ambiente 3 é dotado de uma estrutura que força a navegação do agente A* por meio de vértices no eixo Y. Esta peculiaridade, é representada na figura 5 pelas rampas em amarelo.

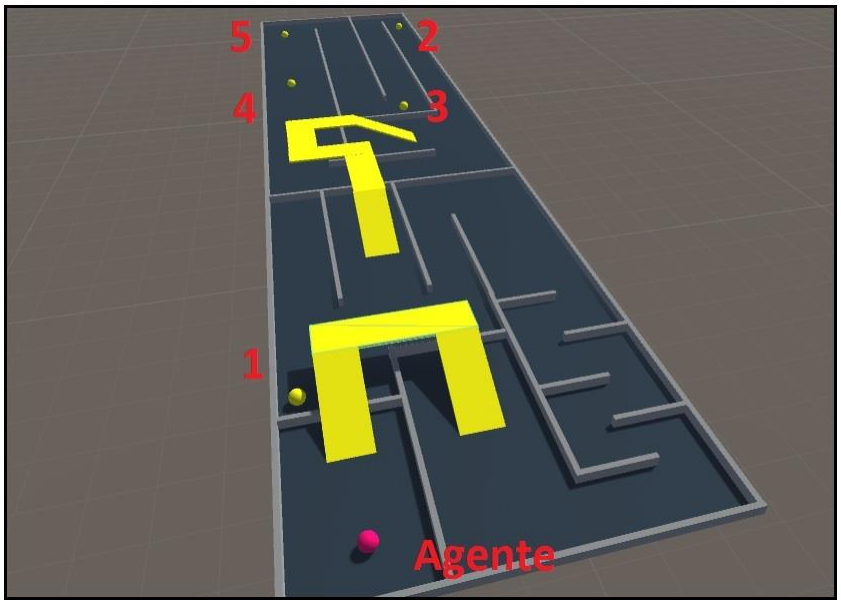


Figura 5. Ambiente 3

4.2 Desenvolvimento do agente A* com a ferramenta NavMesh Agent

Na implementação do agente A*, fez-se o uso de uma ferramenta disponibilizada no Unity3D para auxiliar na navegação de objetos 3D nos ambientes. O NavMesh Agent permitiu vincular um *script* de *path-planning* à um objeto dinâmico. A ferramenta disponibiliza uma biblioteca de métodos e funções para personalizar a navegação do agente de acordo com cada situação e padronizar a heurística de *path-planning*. As funções implementadas no agente são ilustradas na figura 6.

Método	Tipo	Descrição
Destination	Vector3	Armazena as coordenadas de destino do objeto
Raycast	bool	Traça uma linha entre dois pontos
gameObject	GameObject	O objeto ao qual o componente é vinculado
position	Vector3	A posição das coordenadas no espaço aberto
Transform	Transform	Armazena as coordenadas do objeto
Destination	Vector3	Armazena as coordenadas de destino do objeto
GetComponent objeto	Component	Retorna o tipo da variável do componente vinculado ao

Figura 6. Métodos do NavMesh Agent

4.3 Implementação do algoritmo A*

A implementação do algoritmo A* foi feita com base na linguagem de programação C#[5]. O código foi desenvolvido no editor de scripts do próprio motor. Após a criação das classes do algoritmo, o arquivo do script é vinculado ao agente por meio de uma variável pública que é atualizada automaticamente no decorrer da execução.

4.4 Realização das execuções

O quadro de testes que representa as execuções foi definido da seguinte forma:

a) o objeto agente vinculado ao algoritmo A* será destinado a cinco diferentes pontos de destino em cada ambiente;

b) o agente A* irá percorrer o menor caminho entre os vértices do mapa até o objeto destino em dez execuções;

c) dez execuções extras não envolvendo a busca pelo objeto destino serão realizadas a fim de se fazer a coleta de dados sem a presença do agente;

Ao término das execuções e da coleta de dados, foram comparados os resultados na presença e na ausência do agente A*.

4.5 Resultados Obtidos

Na avaliação qualitativa dos resultados obtidos, adotou-se o *frame* como parâmetro de medição, visto que este é um processo que envolve a atualização dos recursos de jogo em função do tempo.

Os resultados foram obtidos somando-se os valores de cada *frame* e dividindo-se pelo número total de *frames* construídos após todas as execuções, obtendo-se a média destes valores. Entretanto, de acordo com os parâmetros de avaliação qualitativa do FPS adotados, para valores acima de 200 FPS, a performance é considerada Excelente. Logo, o cálculo é dispensado para médias acima desse valor visto que o resultado não irá alterar a classificação. Na tabela 1, é demonstrado os valores que determinam a classificação do desempenho da navegação.

FPS (Frames por segundo)	Performance
Acima de 200	**Excelente**
Entre 100 e 200	**Ótimo**
Entre 60 e 100	**Bom**
Entre 30 e 60	**Regular**
Abaixo de 30	**Ruim**

Tabela 1. Desempenho de um jogo

[1] Linguagem de programação de alto nível, orientada à objetos, desenvolvida pela Microsoft e destinada à criação de aplicativos com o Visual Studio e o .NET *framework* (Microsoft, 2015).

Após os cálculos das médias obtidas, foram elaborados três gráficos para facilitar a visualização da variação dos valores nas execuções com a presença e a ausência do agente A*. A figura 7 refere-se a taxa de consumo de memória RAM.

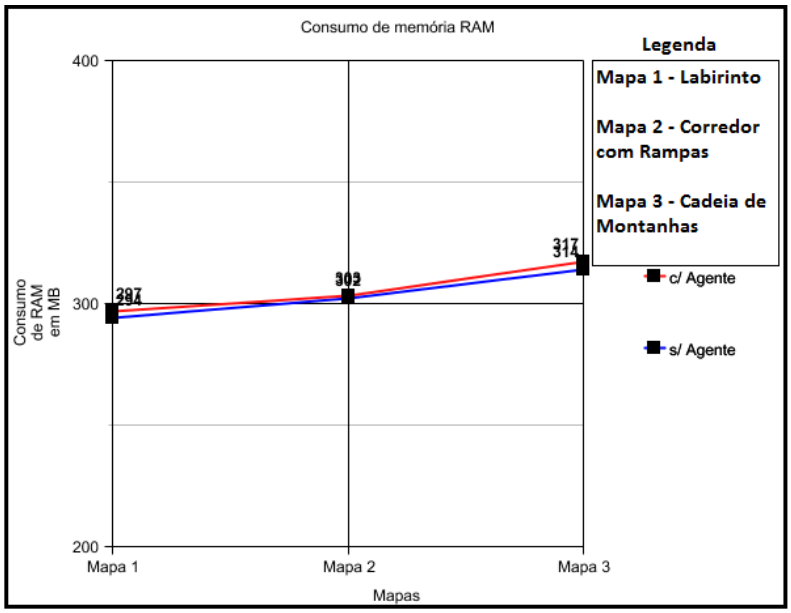


Figura 7. Variação do consumo de memória RAM

Na figura 8 é apresentam-se as taxas de variação da renderização. Observa-se que nos ambientes de maior complexidade a navegação obteve valores menores.

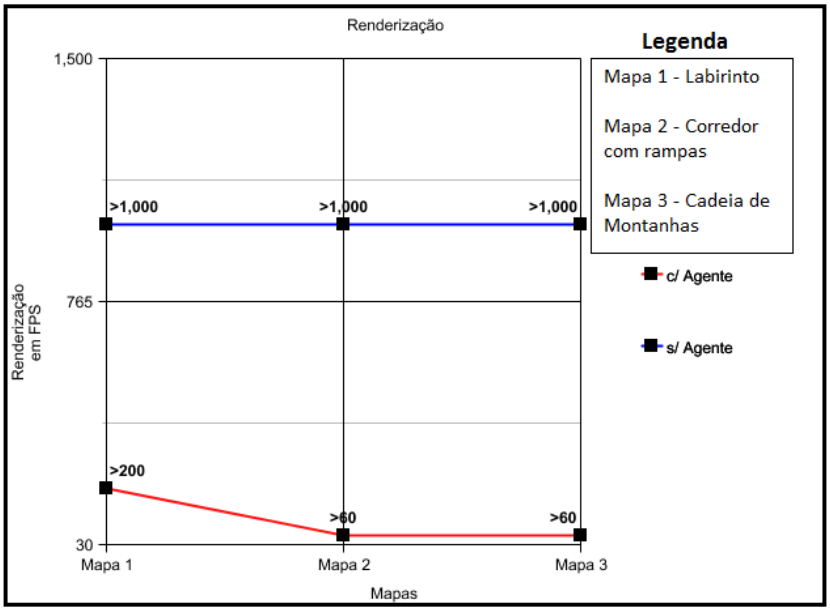


Figura 8. Variação da taxa de renderização

A taxa de sincronia trata do processo de renderização em função do tempo. Para ambos os casos, observa-se na figura 9 que a taxa de sincronia se manteve superior a 60 e inferior a 90 FPS, apresentando poucas variações mesmo que em ambientes de complexidade diferente.

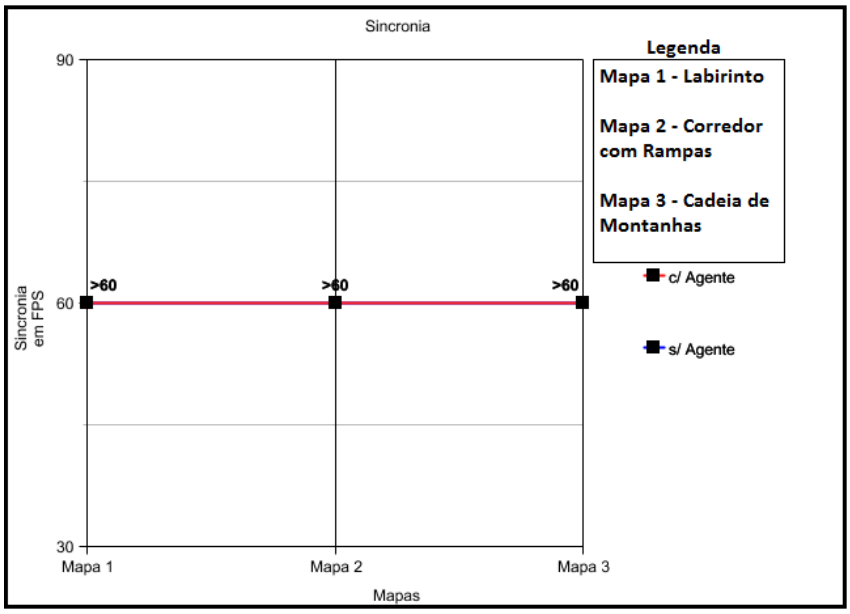


Figura 9. Variação da taxa de sincronia

5. Conclusão

Os resultados obtidos proporcionaram a compreensão da influência de um agente A* em cenários específicos. Concluiu-se que a presença de um objeto agente deve dotar de uma heurística de busca de menor caminho que corresponda à dada situação de jogo e de acordo com a estrutura de cada ambiente. Os valores dos recursos utilizados serviram de parâmetro para comparação das médias obtidas.

Na análise dos resultados perante os parâmetros de medição de performance, o agente A* foi classificado como Excelente na maioria dos casos. Entretanto, a exploração de objetos 3D multipoligonais, bem como a navegação por meio do eixo Y como visto no ambiente 2, afetou as taxas de sincronia do FPS das execuções contribuindo na redução do desempenho. Ressalta-se que o algoritmo A* soluciona a busca do menor caminho dentre os vértices em um plano de jogo de maneira eficiente, porém a diversidade de recursos disponíveis na criação de ambientes de jogo tende a influenciar no seu desempenho de maneira que os testes devão ser realizados de acordo com cada situação.

Referências

BATTAIOLA, A.L. - Jogos por Computador – Histórico, Relevância Tecnológica e Mercadológica, Tendências e Técnicas de Implementação - XIX Jornada de Atualização em Informática/SBC – 2000.

DUARTE, Luiz Cláudio S.. **Jogos de Tabuleiro no Design de Jogos Digitais.** Brasília: Sbc, 2012. 6 p. Disponível em: <http://base.gamux.com.br/events/2012.11.02-SBGames12/proceedings/papers/artedesign/AD_Full17.pdf>. Acesso em: 14 mar. 2015.

LUGER, George F.. **Inteligência Artificial.** 6. ed. São Paulo: Pearson Education do Brasil, 2014. 614 p.

NOVAK, Jeannie. **Desenvolvimento de games.** 2. ed. São Paulo: Cengage Learning, 2011. 443 p.

RABIN, Steve. **Game AI Pro 2:** Collected Wisdom of Game AI Professionals. London: A K Peters-wellesley, 2015. 533 p.

RABIN, Steve. **Introdução ao Desenvolvimento de Games:** Programação, técnica, linguagem e arquitetura. 2. ed. São Paulo: Cengage Learning, 2012. 637 p. (2).

RUSSELL, Stuart; NORVIG, Peter. **Inteligência Artificial.** 3. ed. Rio de Janeiro: Campus, 2013. 1016 p.