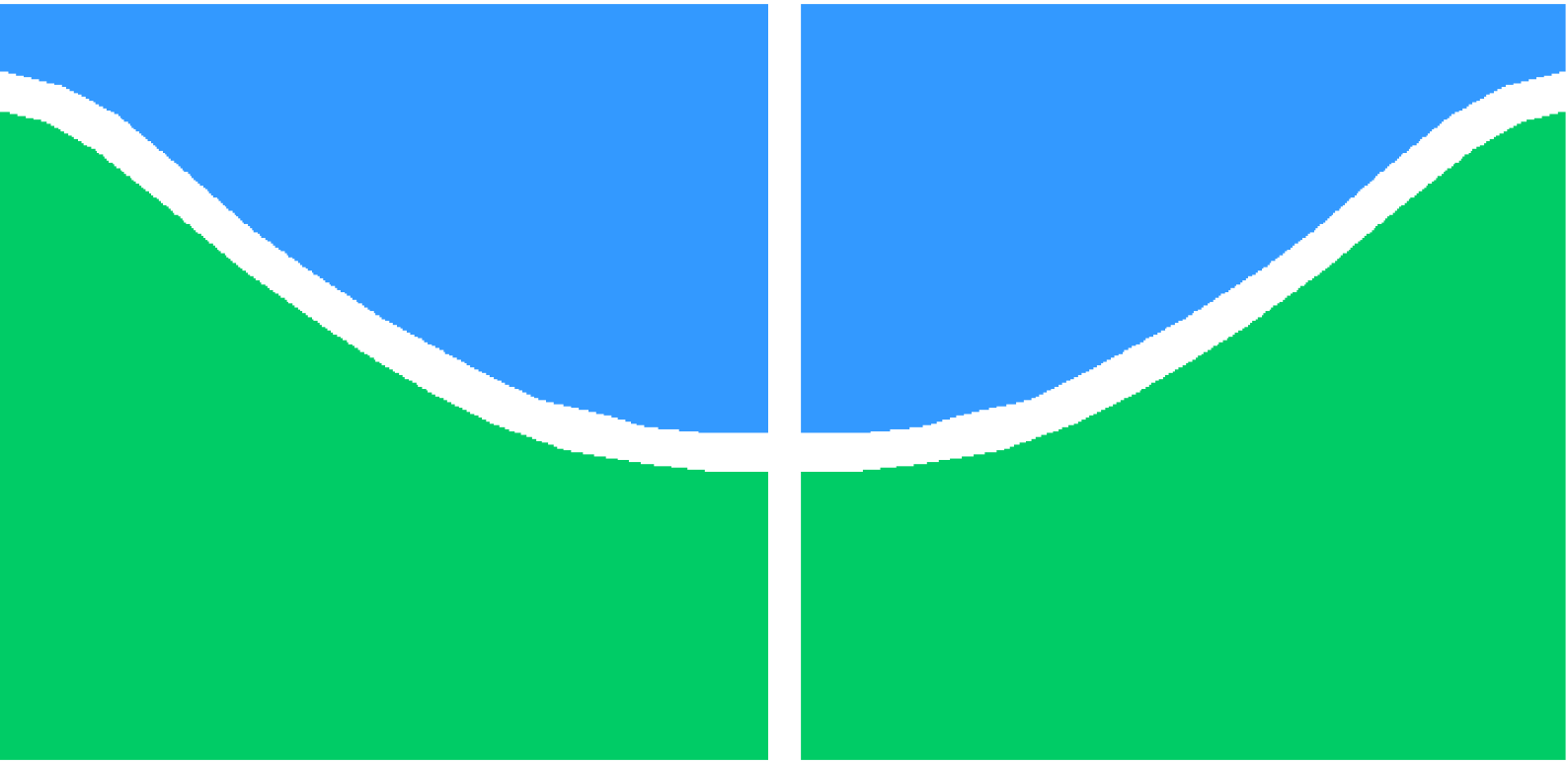

Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Engenharia de Software

Analise de mapas procedurais para jogos *roguelike* através de simulações

Autor: Gustavo Jaruga Cruz
Orientador: Prof. Dr. Edson Alves da Costa Júnior

Brasília, DF
2013

Gustavo Jaruga Cruz

Analise de mapas procedurais para jogos *roguelike* através de simulações

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Dr. Edson Alves da Costa Júnior

Brasília, DF

2013

Gustavo Jaruga Cruz

Analise de mapas procedurais para jogos *roguelike* através de simulações

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 01 de junho de 2013:

Prof. Dr. Edson Alves da Costa Júnior
Orientador

Profa. Dra. Carla Denise Castanho
Convidado 1

Profa. Dra. Carla Silva Rocha Aguiar
Convidado 2

Brasília, DF
2013

Resumo

Neste trabalho serão analisadas as principais características da jogabilidade do gênero *roguelike* a fim de descobrir métricas relevantes para um bom mapa. Será feito uma forma de juntar as diversas métricas estabelecidas e utilizá-las para assertir quanto a qualidade de um mapa. Será produzido uma ferramenta capaz de realizar testes através de jogos simulados automáticos para um dado mapa de entrada, extrair as métricas de jogo e computar a qualidade total do mapa para assegurar que os valores esperados para o mapa estão correspondentes as expectativas.

Palavras-chaves: mapas. jogos. algoritmos. qualidade.

Abstract

It will be analysed in this work the main characteristics of the roguelike genre gameplay and developed a tool capable of using a map data as input to perform an automated simulation of games to assure the quality based in several metrics and their expected values for the map.

Key-words: procedural. games. algorithm. statistics.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	Application Programming Interface
ASCII	*American Standard Code for Information Interchange*
IA	Inteligência Artificial
SDL	Simple Direct Layer
SFML	Simple Fast Media Layer
Tiles	Blocos
RPG	Role-Playing Game
IDE	Integrated Development Environment

Sumário

Introdução

Introdução

É cada vez mais visível em jogos a utilização de conteúdos proceduralmente gerados. A demanda por jogos cada vez mais rejogáveis aumentou muito nos últimos anos, o que é observado através da alta utilização de conteúdos procedurais em jogos modernos, com gráficos tanto de perspectiva tanto 2D quanto 3D.

Não se pode esquecer da influência dos primeiros jogos que utilizaram este tipo de conteúdos automáticos. O gênero *roguelike* surgiu em meados dos anos 80 pelo jogo *Rogue*, que popularizou-se na época devido a seu aparente infinito número de possibilidades de jogabilidade, consequência do fato de que grande parte do seu conteúdo como mapas e ítens, serem randômicamente gerados, assim como seus gráficos representativos em ASCII(*American Standard Code for Information Interchange*).

Com a evolução dos computadores e do gênero em si, jogos *roguelike* tiveram que evoluir também, apesar de ainda haver jogos recentes que preservam o aspecto clássico do gênero, com gráficos ainda em ASCII ou formados *tiles* 2D. Existem também jogos do gênero com gráficos mais modernos e animações, em plataformas móveis, como *Pokemon Mystery Dungeon*[1] e *Izuna*[2], de Nintendo DS e em plataformas recentes como *Guided Fate Paradox*[3], para PS3. O que mostra que o gênero ainda tem público nos dias atuais. Apesar de usarem tão fortemente conteúdos gerados proceduralmente, ainda hoje não é possível obter uma boa métrica de forma precisa quanto qualidade de seus mapas. Normalmente tendo a adequação dos seus mapas gerados através de muitos testes e apenas do sentimento obtido após muitas partidas.

Não há nada de errado com esta forma, porém se houvesse uma ferramenta capaz de produzir métricas de análise para os mapas randômicamente gerados de forma relativamente rápida, isto poderia acelerar o processo de aperfeiçoamento dos algoritmos devido a redução em parte do tempo de testes.

Neste projeto, será desenvolvido um sistema capaz de obter um mapa e testa-lo através de uma série de métricas para assegurar a qualidade do mapa em relação a qualidade desejada. Na segunda parte do projeto será realizado uma analise comparativa para indicar se mapas proceduralmente gerados podem possuir métricas similares a mapas construídos manualmente e como usuários reais se sentem em relação as métricas obtidas,

---

1 Pokemon - (**??**)
2 Izuna - (**??**)
3 Guided Fate to Paradox - http://nisamerica.com/games/guided_fate_paradox

assegurando a confiança do sistema em seus resultados.

Organização do documento

O próximo capítulo, Objetivos, irá apresentar os objetivos gerais, representando o que o projeto se propõe a alcançar, e os objetivos específicos, mostrando o que será alcançado para resolver os objetivos gerais propostos.

O Referencial Teórico irá mostrar alguns conceitos sobre a concepção e a história por traz do gênero de jogo *roguelike*, assim como uma base teórica sobre conteúdos procedurais e taxonomias baseadas em artigos científicos e notícias.

A Justificativa irá explicar a importância do trabalho e o que o motivou a ser criado, demonstrando também seus benefícios para a sociedade no âmbito do desenvolvimento de jogos do gênero.

O quarto capítulo, Delimitação do Assunto, irá tratar de explicitar o que será desenvolvido e representado no trabalho e o que não entrará em seu escopo para que não haja dúvidas quanto suas expectativas.

No quinto capítulo, Procedimentos Metodológicos e Técnicas, inicia-se o desenvolvimento, aonde serão demonstradas técnicas e procedimentos utilizados para a concepção e pesquisa do projeto, assim como a metodologia aplicada para a realização deles.

O sexto capítulo, Cronograma, retratará como foi executado o cronograma do projeto durante esta sua primeira etapa e como será executada a segunda parte do projeto.

O ultimo capítulo, Resultados Alcançados, irá descrever que foi desenvolvido e obtido através da realização do trabalho durante está primeira etapa.

1 Objetivos

Estão especificados aqui os objetivos gerais que mostram as metas de longo alcance que o trabalho procura atingir, de forma mais ampla. E os objetivos específicos que detalham e orientam a forma com que será voltado para completar os objetivos gerais.

1.1 Objetivos Gerais

Este trabalho procura desenvolver uma forma de análise de algoritmos procedurais voltados especificamente para jogos do gênero *roguelike*. O objetivo principal é definir uma métrica para a qualidade de um determinado mapa, a ser obtida através de uma série de testes automatizados de jogabilidade deste mapa. Estes testes gerarão uma série de métricas e estatísticas que serão combinadas de modo a formar a métrica de qualidade desejada, permitindo a avaliação do mapa auto-gerado, em tempo real, de acordo com as características impostas pelo desenvolvedor do jogo.

1.2 Objetivos Especifícos

- Explorar opções de API's gráficas para o sistema.
- Construir um sistema básico de simulação de jogos *roguelike*.
- Criar uma inteligencia artificial para jogar automaticamente através de *scripts*.
- Identificar e estabelecer perfis para possíveis AI(Inteligência Artificial) e melhor simular as métricas extraídas.
- Extrair métricas a partir de partidas jogadas automaticamente pela inteligência artificial.
- Normalizar e classificar as métricas extraídas de uma forma qualitativa através de acordo com parâmetros esperados.

2 Referencial Teórico

2.1 Jogos Roguelike

2.1.1 Caracterização do termo roguelike

Existem atualmente diversos gêneros e sub-gêneros de categorias de jogos, que constituem uma tentativa de classificá-los de acordo com suas principais características e objetivos. Dentre estes muitos tipos, o gênero *Roguelike* é uma derivação de RPG(*Role-Playing Game*) e *Dungeon Crawl*.

O gênero *Dungeon Crawl* é constituído de um cenário de fantasia aonde heróis necessitam navegar por ambientes labirínticos e enfrentar diversos monstros e desafios, encontrando tesouros e recompensas. RPG é o termo que se dá em jogos onde o jogador interpreta um personagem em um mundo fantasia. Em jogos este termo está intrinsicamente ligado a progressão de nível e habilidades e ataques afetadas por parâmetros que podem ser treinados através de pontos de experiência ou outro sistema similar.

O termo *roguelike* é inspirado pelo jogo *Rogue*(Figura 1 - 2.1.1), construído para sistemas baseados em Unix em 1980. *Rogue* se destacou na época pela utilização de gráficos ASCII para representar a contextualização de seu jogo, e pela disposição das salas e ítens, que eram gerados aleatóriamente cada vez que o jogo era jogado, um conceito que poucos jogos utilizavam na época.

O jogo se baseava em turnos e tinha o jogador que se mover por labirintos constituídos de 3 por 3 salas dispostas aleatóriamente sobre o nível.

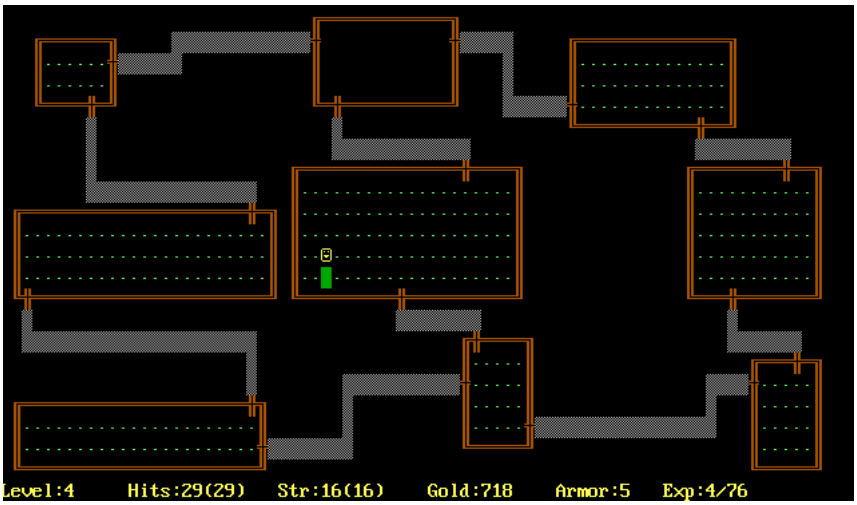

Figura 1 – Rogue em um IBM PC

Rogue sofreu críticas devido ao seu alto custo de CPU para jogos da época, porém a idéia não se desfez e em 1984 o jogo recebeu suas versões para IBM PC e Macintosh.

Apenas 2 anos depois do lançamento de *Rogue*, o termo *roguelike* já estava inspirando jogos como *Hack* em 1982 e *Moria* em 1983.

Moria foi talvez um dos mais influentes jogos que ajudaram a desenvolver o termo *roguelike* e torná-lo mais próximo do que é hoje. *Moria* teve sua ambientação inspirada nas histórias de Tolkien, Senhor dos Anéis (**??**), e tinha como objetivo derrotar Balrog nas profundezas das Minas de Mória. *Moria*(Figura 2 - 2.1.1) já apresentava mapas aleatórios mais bem elaborados e maiores, com um sistema mais bem definido de combate e lojas (**??**).

Figura 2 – Moria em um terminal - '@' representa o jogador, 'r' representa ratos.

O termo *roguelike* então é caracterizado pela forte herança de conteúdos procedurais aleatórios que afetam o jogabilidade(*gameplay*), sejam eles mapas, ítens ou a disposição dos inimigos. Apesar de não sempre verdade, jogos deste gênero possuem o conceito de morte permanente, isto é, uma vez que o seu jogador morra deverá ser necessário recomeçar outro. O gênero também estabelece conceitos como a visibilidade do mapa e movimento baseado em blocos. Uma lista de fatores que característicos do gênero, estabelecendo a sua definição, foram discutidos e compilados durante conferencia de desenvolvimento de jogos *roguelike* (*Roguelike Development Conference*), em 2008, e atualizados recentemente. (**??**).

2.1.2 História

A popularização de jogos *roguelike* foi obscurecida no mercado de jogos ocidental em seu primeiro momento, constituindo-se principalmente de jogos não-comerciais, tendendo a apresentarem gráficos ASCII.

Porém, nos anos recentes, é possível notar um crescimento de jogos *roguelike* no mercado de jogos. Os novos jogos do gênero constituem-se principalmente de graficos animados e bem desenhados, alguns possuindo até mesmo uma ambientação 3D (**??**).

Em um primeiro momento após a fixação do termo, jogos comerciais do gênero foram produzidos por marcas orientais como a Chunsoft[1], uma marca japonesa especializada em jogos *roguelike*, com títulos como *Shiren The Wanderer*[2] (que mais tarde foi publicado com a Sega Nintendo DS), assim como *Pokemon Mystery Dungeon*[3] (Figura 3 - 2.1.2).

Figura 3 – Pokemon Mystery Dungeon - GBA (esquerda) e Izuna - Nintendo DS (direita)

Estes novos jogos são, em sua maioria, formados por salas com visibilidade total e corredores obscurecidos, e os jogos usualmente são divididos em duas partes: uma na cidade onde o jogador poderá se organizar e comprar novos ítens e pequenas áreas de 5 a 10 andares. O conceito de morte permanente tornou-se menos recorrente: a morte no jogo porém faz com que o jogador perca todos os ítens, experiência e dinheiro adquiridos durante aquela área.

Esta nova forma de jogo tornou o gênero *roguelike* mais familiar aos jogadores, trazendo diversas traduções para o ocidente e divulgando mais o estilo de jogo.

Apesar de ainda não ser muito vistos jogos comerciais deste gênero desenvolvidos no ocidente, é notado nos últimos anos um grande aumento de jogos independentes sendo criados. Alguns jogos dentre estes que inclusive encontram-se na Steam são *Dungeons of Dreadmor*[4], *Sword of the Stars: The Pit*[5], *Steam Marines*[6], dentre outros (Figura 4 - 2.1.2).

Vale ainda ressaltar os jogos recentes produzidos pela *Nippon Ichi Software*, que utilizam-se de belos gráficos em um ambiente 3D cheio de animações, o que pode muito

1 Chunsoft - www.spike-chunsoft.co.jp
2 Shiren The Wanderer - http://atlus.com/shiren
3 Pokemon - www.spike-chunsoft.co.jp/games/pokedun/
4 Dreadmor - http://store.steampowered.com/sub/15933
5 Sword of the Stars - http://store.steampowered.com/app/238450
6 Steam Marines - http://store.steampowered.com/app/253630

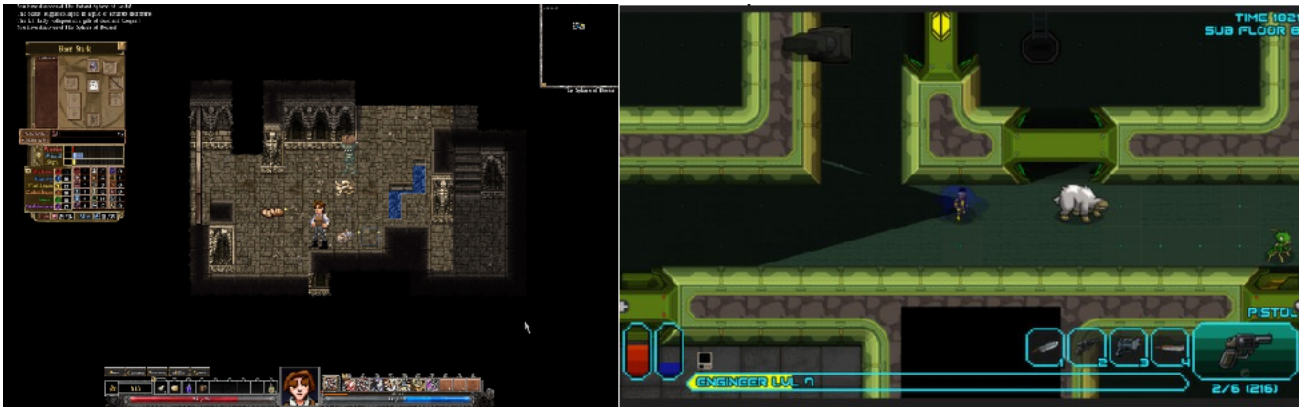

Figura 4 – *Dungeons of Dreadmor* (a esquerda) e *Sword of the Start: The Pit* (direita)

bem indicar o futuro de jogos *roguelikes* comerciais. Seus dois jogos: *Zettai Hero Project*[7] em 2010 e *Guided Fate to Paradox*[8] em 2013 utilizam da mecânica de movimentação de blocos em turnos em uma perspectiva isométrica, e adicona também o conceito níveis-Z, isto é, os ataques e movimentos são influenciados pelas alturas dos terrenos.

Figura 5 – *Guided Fate to Paradox* em sua perspectiva isométrica 3D

2.2 Geração Automática Procedural

A utilização de algoritmos e técnicas para a criação de conteúdos, é chamada de Geração Procedural. A definição do termo, embora não tenha uma versão definitiva e consagrada na literatura, diz respeito a geração procedural, aleatória ou pseudo-aleatória de coisas, sejam elas objetos, figuras, sons ou imagens.

7 ZHP - http://nisamerica.com/games/zhp
8 Guided Fate to Paradox - http://nisamerica.com/games/guided_fate_paradox

2.2.1 Geração Procedural vs Geração Procedural de Conteúdo

Apesar de não possuir uma literatura oficial, a **Geração Procedural** é dividida por diversas fontes e comunidades (**??**) em dois termos de acordo o seu objetivo.

- **Geração Procedural**
  A definição original do termo. Representa coisas geradas de forma aleatória ou pseudo-aleatórias. Este termo ainda é utilizado de forma genérica para referenciar qualquer forma de geração, porém, tenta-se utilizá-lo para se referir a produtos visuais, sonoros ou coisas que não afetam a jogabilidade.

- **Geração Procedural de Conteúdo**
  O termo refere-se a geração de produtos que afetam a jogabilidade de alguma forma. Aqui encontram-se a geração de termos e mapas, ítens, objetivos e outros fatores que podem alterar a forma de se jogar um jogo.

A distinção dos dois termos surgiu com a diferença óbvia de objetivos e impactos que teriam sobre um jogo. No mercado de jogos atuais, pode-se dizer que quase todos os jogos 3D utilizam geração procedural para a criação de seus mapas, através da disposição randômica de artefatos como árvores e gramas ou texturas ligadas entre si. Porém, isto é um fator que não altera efetivamente a jogabilidade e a forma de se jogar o jogo: sua funcionalidade é de representar mais naturalmente e variadamente atrativos visuais ao usuário, e portanto não é uma geração procedural de conteúdo.

2.2.2 Taxonomia e Termos

Como dito, ainda não há uma literatura bem definida com a definição de termos e formas existentes de geração procedural, porém é visível que algumas técnicas utilizadas baseiam-se dos mesmo princípios. Os termos e taxonomias encontrados aqui são referenciados por (**??**) em seu artigo científico e portanto pode-se haver variações entre os termos citados com relações a outras fontes.

As gerações procedurais podem ser então:

Online versus Offline

Indica o modo como é gerado. A forma **online** representa que conteúdo é gerado em tempo de execução enquanto a forma **offline** é gerado durante o desenvolvimento e colocado no jogo estaticamente.

O método **offline** é muito utilizado para se gerar grandes mapas abertos automaticamente, deixando a cargo do designer de levels o trabalho de modificá-lo manualmente para deixá-los do jeito que deseja. Isto permite gerar detalhes mais naturais em mapas

muito grande que de outra forma consumiram uma quantidade inefetiva de tempo para serem produzidos.

O método **online** cria conteúdos sem a interferência humana e portanto deve se tomar maiores cuidados com a lógica implementada no algoritmo para garantir que o resultado da geração atenda as expectativas da equipe de desenvolvimento e dos jogadores. É comumente usada para partes não vitais do jogo, onde um erro de lógica não crie uma impossibilidade de progresso na historia, apesar de ser utilizado também de outras formas.

Conteúdo Necessário e Opcional

O **conteúdo necessário** é aquele que o jogador irá precisar iterar para que possa progredir no jogo, enquanto que o **conteúdo opcional** é aquele que o jogador pode escolher não fazê-lo e mesmo assim vencer sem problemas.

Sementes aleatórias versus vetores parametrizados

As **sementes aleatórias** e os **vetores parametrizados** representam o modo como os algoritmos procedural efetuam seus cálculos. Um algoritmo utilizando **sementes aleatórias** pode ser representado pelo número dado para o seu gerador de números randômicos. Enquanto no outro extremo o algoritmo pode tomar como entrada um vetor multidimencional de parâmetros reais para definir mais detalhadamente seus propriedades específicas.

Para um algoritmo de geração de calabouços, alguns dos parâmetros poderiam ser o número de quartos, índice de ramificação dos corredores, o tamanho dos corredores, etc. Para um algoritmo de geração de mapas globais alguns parâmetros poderiam ser a porcentagem de água, a quantidade de ilhas, o clima geral do planeta, o índice de ocorrência de montanhas e minérios, etc.

Geração estocáica versus determinística

Uma **geração determinística** é aquela capaz de, dado os mesmos parâmetros de entrada, produzir sempre o mesmo resultado. A **Geração estocáica** é aquela que sempre possuem uma variação aleatória, mesmo que os dados das entradas sejam os mesmos.
Este tipo de geração é muito comum para **algoritimos parametrizados**. Em um gerador de calabouços, dado os mesmo parâmetros de quantidade de salas e tamanho do mapa total, as saídas em geral não resultam em um mapa idêntico, apesar de possuírem entradas iguais.

Construtivo versus Gerar-e-testar

Um **algoritmo construtivo** gera seu conteúdo apesar uma vez e o termina. Ele deve tomar os devidos cuidados durante a geração para garantir que a saída seja suficientemente aceitável. Um **algoritmo de gerar-e-testar** é aquele que cria o conteúdo e, ao finalizá-lo, realiza um ou mais testes para garantir que a saída atende aos critérios especificados. Caso não atenda, partes do conteúdo são geradas e testadas novamente, até que os critérios de aceitação sejam atingidos.

Os mapas de *Dwarf Fortress* são gerados utilizando vetores parametrizados como o tamanho, clima e idade do mapa. Durante a criação do mapa é possível visualizar um informativo mostrando a `quantidade de áreas rejeitadas`.

Figura 6 – Criação de mapas em *Dwarf Fortress* - 93 mapas rejeitados

2.3 Técnicas e Algoritmos

Esta seção irá apresentar algumas técnicas de algoritmos utilizadas para a geração procedural.

2.3.1 Ruído

Na **geração procedural** como um todo, mas principalmente na geração de terrenos 3D, a utilização de algoritmos de ruídos é algo de grande importância, dado o resultado aparentemente mais natural e visivelmente aleatório obtido por eles. (**??**)

Terrenos

Atualmente é a área da geração procedural que possui a maior disponibilidade de informações e técnicas reconhecidas. A geração procedural de terrenos se popularizou muito na última década devido a uma multidão de jogos de mundo aberto com terrenos

cada vez mais bem detalhados e únicos.

Talvez um dos métodos mais populares e utilizados para a geração de terrenos 3D e ou terrenos 2D de plataforma seja a partir da criação de fractóides.
Terrenos são comumente gerados inicialmente a partir da criação de *heightmaps*, mapas de altura que irão definir a altura do terreno. Um exemplo de algoritmo de criação de *heightmaps* utiliza um processo iterativo sobre **ruído fractal Browniano**(*fractal Brownian noise*) (**??**).Outro exemplo visto é na criação de superfícies sintéticas, algoritmos de geração planetares que utilizam o **ruído de Perlin**(*Perlin noise*).Outra técnica utilizada é a simulação de erosões sobre o terreno.

Texturas

A utilização de ruídos é muito comum para a criação de texturas procedurais, como texturas de árvores e terrenos, para mostrar um ambiente mais vivo. Este é um exemplo do termo `Geração Procedural` que não há presença de conteúdo relevante a jogabilidade.

Sons

A utilização de ruídos e filtros através de algoritmos procedurais pode ser utilizada para a geração de sons. Um uso mais comum dessa técnica é a geração de sons de diversos tipos de ventos ou sons ambientes. Uma grande literatura sobre este assunto pode ser observada nos trabalhos de *Farnell* (**??**) (**??**).

2.3.2 Geração de terrenos sob perspectivas 2D

Informações sobre geração de mapas 2D são mais dificilmente encontradas e há atualmente uma falta de formalização das técnicas utilizadas.

Algumas técnicas observadas são:

- **Montagem estática:** Vista principalmente em jogos de plataforma 2D e *dungeon crawlers(2D e 3D)*. Está técnica constitui-se de uma seleção de mapas pré-definidos e armazenados que são "encaixados" uns aos outros durante a criação do mapa, dando ao jogo uma certa diversidade de conteúdo com a segurança de que os mapas gerados serão consistentes com o imaginado.
- **Graph Grammar** O trabalho (**??**) mostra a técnica de criação de calabouços utilizando **Gramática de Grafos**(*Graph Grammar*. Em seu documento, ele divide a criação de mapas em duas etapas: **Topologia** e **Geometria.**

  A **topologia** do nível descreve como a ordem das coisas deve acontecer, sem se

preocupar com os aspéctos físicos ou detalhes dos terrenos. Um exemplo mais simples de um modelo topológico de um level pode ser:
`Inicio->Inimigo->Inimigo->Vida->Inimigo->Item ->Chefe->Fim`

Um exemplo mais complicado, porém menor, poderia ser:
`Inicio->Inimigo->Bifurcação[Bifurcação(Item->Item)->Inimigo]->Chefe->Fim.`

A geração **geométrica** é aquela que utiliza os dados obtidas da topologia e, a partir destas entradas e um certo grau de randomicidade, gera fisicamente as salas e detalhamentos, assim como o posicionamento de inimigos e ítens.

Também é visto a geração de conteúdo procedural através de `Gramática de Grafos` utilizando células *voronoids*[9] para a criação de salas e corredores em (**??**). O autor comenta que sua principal inspiração para a sua criação foi inclusive o trabalho de (**??**).

- **Miners**
  Esta técnica foi apresentada por *Jeffrey Thompson*(**??**) para a criação de cavernas. A idéia é basicamente criar um mundo constituído apenas por blocos, e colocar células randômicas espalhadas, chamadas de **mineradores**(*miners*. O algoritmo irá entrar entrar em um laço onde selecionará os *miners* ativos e chamará sua função *dig* (cavar), que terá uma pequena chance de gerar outro `minerador` adjacente a ele e que irá destruir um bloco adjacente, movendo-se até ele. Caso não haja mais nenhum bloco adjacente ao *miner*, ele será desativado. A inserção de outros limites para a desativação de *miners* também pode ser utilizada, como tempo de vida ou a quantidade de mineradores criados por ele.

  Após a desativação de todos os mineradores o algoritmo entrará na fase de limpeza, onde irá remover defeitos, imperfeições obviamente irregulares como "paredes solitárias", "pequenas ilhas" e outras, definidas a critério do *game design*.

2.4 API Gráfica e C++

Uma API (*Application Programming Interface*), especifica como componentes de software devem interagir entre si. Neste caso, os componentes gráficos do sistema.

Grande parte dos jogos dependem em grande parte da habilidade de ser capaz de representar visualmente algo na tela. Para realizar isto de forma mais natural, são

[9] Célula representando uma região de um diagrama Voronoi

utilizadas API gráficas. Nos jogos em especial, é necessário uma grande liberdade para se desenhar diversas imagens e animações na tela, ao contrário dos restritivos botões e caixas de textos semi-estáticos que são mais comumente vistos.

Desde sua criação, C++ sempre foi uma linguagem muito utilizada para jogos devido a grande liberdade e poder que a linguagem fornece ao programador, além de permitir a utilização da orientação a objetos e comandos em mais alto nível através de bibliotecas e API's.

2.5 A*

O Algoritmo $A*$ (**??**) é utilizado para se achar um caminho entre dois pontos. Ele se baseia no conceito de exploração e descoberta de nós para se guiar e utiliza-se de heurísticas para determinar qual o melhor caminho de se expandir e evitar o desperdício de processamento, garantindo um provável caminho na direção certa. Ele utiliza uma fórmula de custo dado por:

$$F = G + H \tag{2.1}$$

Sendo F o custo total para a realização da abertura do nó, G representa o custo para se mover, em linha reta do ponto inicial ao ponto e H representa o custo estimado para se mover de um determinado quadrado até o destino. Este valor é usualmente chamado de heurística por ser apenas uma estimativa do valor real.

Desta forma, a cada quadrado explorado, o algoritmo irá analisar o custo de todos os seus nós descobertos, que são nós adjacentes aos explorados e explorar o nó cujo o custo de F for o menor possível, repetindo este processo até que se encontre o fim.

2.6 Chi Quadrado

A função χ^2(Chi Quadrado) é um teste de hipótese que destina-se a encontrar o valor da dispersão para duas variáveis nominais, avaliando a associação dentre elas. Ao contrário de vários outros testes, o teste Chi Quadrado não é parametrizado, isto é, não depende de valores populacionais como média e variância.

A sua base se da na comparação entre a as proporções para observar as divergências entre as frequências observadas e esperadas para dado evento. Um dos usos deste teste de hipótese é a realização de um teste de aptidão para descobrir se um dado resultado de um evento se encaixa em um determinada função esperada.

Por exemplo, se um dado de 6 faces for jogado 30 vezes e seus resultados forem anotados em uma tabela. Pode-se utilizar o teste de hipótese Chi Quadrado para verificar

se estes valores condizem com o esperado de um dado de 6 faces "justo", isto é, com chances iguais de obtenção dos valores.

A fórmula para se calcular pode ser dada por:

$$\chi^2 = \sum [\frac{(o-e)^2}{e}] \tag{2.2}$$

sendo 'o': valor obtido; 'e': valor esperado

2.7 Distribuição t de Student

Publicada por William Sealy Gosset, cujo pseudônimo escolhido foi *Student*, a distribuição t representa uma curva probabilística simétrica e companiforme, semelhante a curva normal padrão com caudas mais alargadas.

A fórmula para se calcular a distribuição t de Student pode ser dada por:

$$t = \frac{\mu - M}{\frac{s}{\sqrt{N}}} \tag{2.3}$$

sendo μ a média amostral, M a média populacional, s a variância amostral e N o número de amostras.

O teste de hipótese Student t é outro teste realizado muito na comparação entre dois eventos, podendo-se comparar duas amostras para se determinar se, por exemplo, seus médias populacionais são diferentes. Este processo é utilizado em situações para determinar se um dado novo método é mais efetivo que um antigo.

3 Justificativa

Apesar da grande utilização de conteúdos procedurais em jogos do gênero *roguelike*, ainda não foi estabelecida uma metodologia e um procedimento padrão para teste da qualidade dos algoritmos envolvidos e de suas saídas.

Parte da dificuldade de uma abordagem sistemática do problema é a dificuldade em mensurar certos fatores presentes em um jogo que não são facilmente medidos como, por exemplo, diversão, interesse, apreciação, etc. Porém há outras métricas que podem ser extraídas e compiladas para que juntas permitam a construção de um indicador associado à qualidade das saídas de um algoritmo procedural, segundo critérios pré-definidos pela equipe de desenvolvimento do jogo.

Uma ferramenta capaz de obter um mapa e realizar múltiplos testes sobre ele, garantindo que o maior número de possibilidades possíveis sejam testadas em partidas simuladas automaticamente e capaz de extrair métricas significativas sobre a qualidade desejada pode servir de auxílio para a verificação e validação qualidade do conteúdo gerado sem que haja a longa e desgastante participação de usuários e ajustes nos algoritmos baseados apenas nos seus sentimentos e impressões pessoais.

Esta ferramenta não irá retirar o usuário do processo: ela servirá para selecionar e elencar os mapas já previamente otimizados de acordo com as métricas geradas pela ferramenta. Esta abordagem permitirá alocar melhor o tempo e obter uma contribuição mais significativa dos usuários finais, levando a um refinamento e aperfeiçoamento da geração procedural de conteúdos do jogo.

4 Delimitação do Assunto

Aqui serão representados o que este trabalho propõe-se a desenvolver e quais resultados podem ser esperados através do uso da ferramenta a ser desenvolvida.

4.1 Proposta

Construir uma ferramenta capaz de medir a qualidade esperada de um mapa ou algoritmo procedural. Propõe-se também a longo prazo, mostrar os benefícios que conteúdos procedurais trazem a jogos e uma comparação entre mapas feitos manualmente e automaticamente para demonstrar a diferença de qualidade entre os dois modelos de construção de mapas.

4.2 Ferramentas desenvolvidas

A principal ferramenta desenvolvida auferirá a qualidade de um mapa carregado através de um arquivo externo com o seu devido formato. A partir de entradas sobre os valores esperados pelo usuário da ferramenta, ela irá gerar saídas sobre a qualidade do mapa em relação aqueles quesitos especificados e aos diversos perfis de usuário.

A ferramenta também terá suporte à carga de algoritmos de geração de mapas através da sua codificação de arquivos externos no formato *lua*, evitando o desgaste de ter que formatar cada um dos mapas para o formato da ferramenta e testá-los um a um.

Uma ferramenta auxiliar para a criação de mapas manuais e para criação e testes de algoritmos de criação procedurais em *lua*. Contendo também um editor de *scripts* com uma listagem de comandos e variáveis que podem ser entendidas pela ferramenta para se gerar o mapa através dele.

4.2.1 Especificações

- O sistema deve ser capaz de obter um valor desejado para cada uma das métricas que estará testando a fim de garantir que o mapa estará a qualidade desejada para aquele propósito.
- Deverá permitir que seja dado um peso para cada métrica, aferindo peso 0 para métricas indesejadas.
- Ler um arquivo externo, de acordo com a formatação pré-estabelecida e carregar a partir dele um mapa com o posicionamento de inimigos e itens.

- A ferramenta deverá ser capaz de simular jogos, baseados em regras simples do gênero, para testar os mapas.
- Deverá analisar diversos jogos utilizando uma IA customizada para simular os diversos perfis de usuários encontrados.
- Extrair métricas de diversos jogos simulados e compilá-las em um parâmetro de qualidade, baseado nos valores esperados.
- Deverá ser capaz carregar algoritmos de geração mapas escritos em *lua* para que possam ser testado a qualidade do algoritmo como um todo a partir dos mapas gerados por ele.
- Deverá entender a padronização de ítens e inimigos em arquivos lua separados, para que possa escrever a criação dos mapas de forma mais clara e limpa.

4.2.2 Escopo

Está dentro do escopo a criação de uma ferramenta capaz de simular regras básicas de um jogo e extrair métricas a partir dele. A criação de mapas baseados em arquivos ou algoritmos externos, dado que estejam no devido formato entendido pelo sistema. E a obtenção de uma métrica de qualidade para um devido mapa ou algoritmo testado.

4.2.3 Não Escopo

Não está no escopo deste projeto desenvolver um jogo completo, ou qualquer tipo de jogo. A simulação representado pela ferramenta, apesar de ser jogável, contém apenas os requisitos básicos para se testar uma amostra genérica dos atributos contidos em um jogo do gênero.

Não está no escopo a criação de algoritmos procedurais novos e otimizados, apesar de que serão utilizados algoritmos procedurais para gerar as entradas da ferramenta. Estes não serão completamente inventados ou otimizados e representarão diferentes tipos de geração procedural para dar a diversidade ao se testar a ferramenta e garantir que ela funciona apesar da diferença estrutural entre no que diz respeito à mapas.

Não está no escopo a criação de algoritmos de inteligência artificial completamente inéditos. Algoritmos de caminhos como *A** ou *Depth First Search* serão utilizados com ligeiras alterações para se adaptarem ao sistema.

4.2.4 *Meta-Engine* - Ferramenta principal

A principal ferramenta do sistema é responsável por rodar simulações sobre um dado mapa e aferir a qualidade do mesmo através de uma série de parâmetros definidos

pelo usuário. Ela possui uma interface de comandos shell chamada pela tecla ';' que permite a execução de comandos simples do sistema como para alterar a velocidade de processamento da IA.

Esta ferramenta é permite o carregamento tanto de mapas salvos ou a geração de mapas "instantâneos", criados na hora através de um algoritmo procedural de geração de mapas codificado em *scripts lua*. Pode-se também escolher o número de vezes que um determinado mapa irá ser executado para a coleta das métricas. Os mapas simulados podem ser jogados através de jogos automatizados ou manuais, isto é, por comandos do jogador.

4.2.5 *Map Builder* - Ferramenta auxiliar

Esta ferramenta permite a criação de mapas de forma intuitiva e possui algumas ferramentas básicas de edição para o auxilio da produção de mapas. Ela permite a criação e posicionamento de inimigos e itens, pontos de entrada e saída do mapa, e especificação de passagem ou não por um determinado bloco. Todos os comandos e operações são visuais e facilmente vistas.

Possui também um editor textual capaz de identificar a sintaxe léxica da linguagem *lua* e disponibilizando um dicionário de comandos entendidos pelas ferramentas para a criação dos mapas e a descrições de suas utilizações. Permite também que o algoritmos criado seja executado em tempo real e altere o mapa na área principal do programa, identificando erros de sintaxe ou de execução do *scrip* caso ocorram e indicando sua linha.

5 Procedimentos Metodologia e Técnicas

Estarão citadas aqui as formas e técnicas que foram utilizadas para a realização deste trabalho, e quaisquer outras formas metodológicas utilizadas durante o desenvolvimento do mesmo.

5.1 O Processo de Desenvolvimento

O sistema será desenvolvido baseando-se em uma forma mais simples da metodologia ágil SCRUM. Utilizando uma metodologia ágil de desenvolvimento, o projeto teve pequenas iterações incrementais de uma semana com um valor visível, ou seja, com mudanças aparentes ou novas funcionalidades implementadas.

Ao final de cada semana, houve uma revisão do trabalho com o que foi realizado e o que ainda precisa ser realizado, aonde foi planejado o que se fazer para a próxima iteração.

Não houve programação em pares devido a restrições do trabalho e não houve reuniões diárias pelo fato de não haver um *scrum master* associado ao projeto e nem um cliente em si. As revisões semanais das iterações foram realizadas com o orientador do projeto que serviram de uma forma similar a um cliente e *scrum master*, dando opiniões e comentários sobre o andamento do projeto.

5.2 A API Gráfica do Sistema

Devido a facilidade de programação para jogos e prévio conhecimento, a linguagem de programação utilizada para o desenvolvimento da ferramenta utilizada foi escolhida como C++.

Dentro das API's mais conhecidas para programação de jogos em C++, gratuitas, encontram-se:

- Allegro 5
- SDL 2
- SFML 2.1

Allegro é uma API conhecida por sua simplicidade, porém devido a prévias tentativas e problemas encontrados em sua versão 4 com placas gráficas *onboard* está não foi

realmente considerada como uma opção viável para o momento, dado também o tempo de aprendizado adicional que seria necessário para seu domínio.

A SDL é uma ferramenta que recentemente disponibilizou sua versão 2.0 com grandes mudanças e melhoras de performance, podendo-se dizer com facilidade que é uma das mais conhecidas e utilizadas das três API's (baseado em sua versão 1.0), assim como a mais antiga. É construída C, apesar de garantir suporte nativo a C++.

A SFML é uma API relativamente recente que utiliza-se de diversos benefícios da linguagem C++11 internamente para segurança e integridade. Seus benefícios se dão pela sua construção orientada a objetos, o que facilita na utilização e organização de seus diversos módulos.

Todas as API's descritas são multiplataformas e permitem a compilação de aplicativos para Windows, Linux e MacOS.

Um breve comparativo entre SDL e SFML foi realizado, cujos resultados estão compilados na Tabela 1 (5.2).

Característica	**SDL**	**SFML**
Multiplataforma	Sim	Sim
Orientado a Objetos	Não	Sim
Aceleração de Hardware	Sim	Sim
Integração com Áudio	Sim, através de bibliotecas	Sim
Integração com imagens *.png* e *.jpg*	Sim, através de bibliotecas	Sim
Alto Grau de Maturidade	Sim	Não*
Referências e exemplos	Sim	Não**
Open Source	Sim	Sim

Tabela 1 – Comparativo entre SDL e SFML

*A SFML, por ser uma API relativamente mais nova, ainda não possui o mesmo nível de maturidade que o SDL possui vindo de exaustivos testes e utilizações por usuários ao longo dos anos.

**A SFML, por ser mais recente e só agora começar a ser mais divulgada, ainda não possui uma grande base de tutoriais externos além dos presentes no site oficial e o seu manual de API.

Por fim, foi escolhido a utilização da SFML devido a suas facilidades, orientação a objetos nativa e objetos primitivos pré-construidos como classes para representação de textos e gráficos.

Foi também utilizado a biblioteca Qt 5 para o interfaceamento gráfico, principalmente da ferramenta auxiliar. O Qt disponibiliza diversas características comuns para janelas como menus, botões, e outros padrões gráficos que usuários estão acostumados a

esperar de ferramentas. A utilização desta biblioteca torna ferramenta o mais intuitiva e efetiva possível, enquanto as API's citadas acima, que são muito bem utilizadas para jogos, falham em ter uma padronização de interface, dando a liberdade ao programador de faze-las, o que para o contexto de um jogo é algo desejável. Por isso foi-se utilizado o SFML para a produção dos gráficos e contextos da simulação e o Qt para a produção da ferramenta.

5.3 Simulação

O primeiro passo para o projeto foi a construção de uma ferramenta confiável que permitisse um ambiente de simulação controlado para os testes que devem ser executados sobre o mapa. E o jogo, apesar de simplificada, deve conter os elementos principais de um rogo *roguelike* para poder-se comparar a efetividade de cada mapa para este gênero.

A simulação do jogo presente na ferramenta principal consta de regras básicas para que se possa testar, da forma mais genérica possível, jogos do gênero *roguelike*. O jogador iniciará em um nível carregado a partir de um arquivo externo ou gerado por um algoritmo procedural lido pela ferramenta. O nível carregado deverá ter, mínimamente, um ponto de entrada e um ponto de saída, podendo também haver uma série de inimigos e itens de diferentes atributos espalhados pelo mapa.

Para facilitar o entendimento do sistema os itens estarão limitados a itens de recuperação, que aumentarão a quantidade de vida do jogador, e itens de atributos, que aumentarão a quantidade de ataque ou defesa do jogador até o termino do nível, se coletados.

5.3.1 Turnos

A simulação se desenvolve em turnos. Um turno só ocorrerá caso o jogador faça um movimento, que pode ser um movimento a um espaço vazio ao seu redor, ou um ataque a um inimigo adjacente. Ao realizar um turno, todos os outros inimigos do mapa irão também realizar seus turnos e mover-se em direção ao jogador para atacá-lo, caso estejam em sua área de observação.

A sucessão dos turnos se baseia em um atributo de custo de velocidade. Cada movimento do jogador irá causar no sistema a passagem de t unidades de tempo, sendo este número igual a quantidade de custo que o jogador necessita para realizar um movimento (turno). Inimigos mais comumente possuirão um atributo de custo para se moverem maior que o jogador, tornando assim possível um jogador lutar contra diversos inimigos mais fracos com chances reais de vitória.

Para o caso do jogador se mover e o inimigo não atingir o seu valor de custo para

movimentação, este valor é armazenado e utilizado para o próximo turno, decrementando somente a quantidade utilizada ao se mover. Por exemplo: Um jogador com custo de movimento 100 se move e é visto por um inimigo de custo 150. O inimigo não se moverá, porém terá 100 de custo armazenado. O jogador realiza outro passo, deixando-o agora com 200 armazenado, o qual realiza o seu movimento de custo 150 e deixa armazenado 50 para o próximo turno, no qual conseguirá novamente se mover, sobrando zero e repetindo este processo (Figura 7 - 5.3.1).

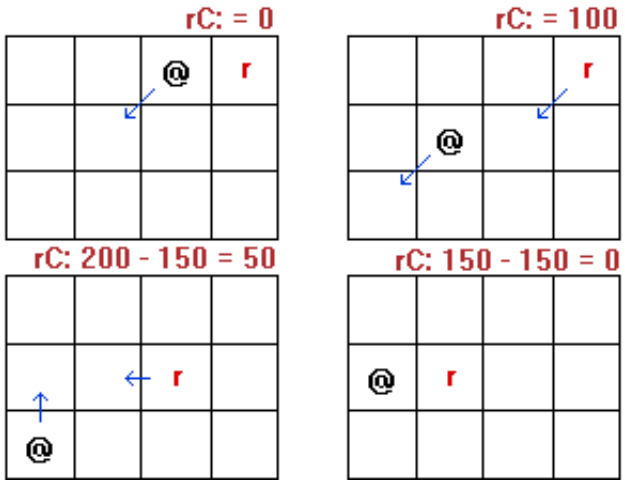

Figura 7 – Desenvolvimento de Turnos

5.3.2 Mapas

O mapa é dividido em *tiles* (blocos). Cada bloco é representado por um quadrado que pode ser passável ou não. Em cada bloco pode haver até um item ou um inimigo ao mesmo tempo. Jogadores ao passarem por cima de itens, irão consumí-los para aumentar seus atributos ou recuperar sua vida. E não poderá haver jogadores e inimigos coexistindo ao mesmo tempo em um bloco, havendo um ataque caso um inimigo ou jogador esteja tentando se mover para um bloco ocupado.

A ferramenta terá a capacidade de leitura de um arquivo externo *.map*, que nada mais é que um arquivo de texto com a devida formatação entendida pelo sistema. O arquivo terá informações sobre a localidade inicial do jogador, os blocos do mapas e possíveis inimigos e itens espalhados por ele.

5.3.2.1 Blocos

Os primeiros valores do arquivo representam a disposição do mapa em relação ao seu tamanho, posição inicial do jogador, tipos e gráficos dos blocos. A formatação segue

o padrão:

```
P_x . P_y
map_X-map_Y
id:tipo  id:tipo ...
```

Os primeiros dois valores P_x e P_y representam a posição inicial do jogador (a qual deve ser um bloco passável),sendo separados por um ponto. Na linha seguinte será indicado o tamanho nas direções do mapa em X e Y, com ambos valores separados por um traço. As map_Y linhas seguintes, representarão os blocos. Cada linha terá map_X identificadores dispostos por um *id* e *tipo*, separados por dois-pontos.

Cada conjunto de *id* : *tipo* representa o bloco na sua posição de acordo com sua linha/coluna. Um identificador de um bloco pode ser:

- 0 - Bloco não passável
- Maior que 1 - Bloco passável

O segundo identificador *tipo* é o representativo visual que o bloco terá dentro do jogo. O *tipo* nada mais é do que um índice de recorte utilizado de um arquivo carregado pela ferramenta encontrado em *"data/img/tileset.png"*. Este arquivo pode ser de qualquer tamanho, e o jogo irá utilizá-lo para extrair o seus blocos. A imagem será recortada em pedaços de 16x16 pixels e será atribuída cada pedaço a um índice. Desta forma um *tileset.png* de 160x32 pixels terá índices entre 0 e 19 para representações. Os índices crescem horizontalmente, e ao chegar ao final de uma linha vão para a linha de baixo.

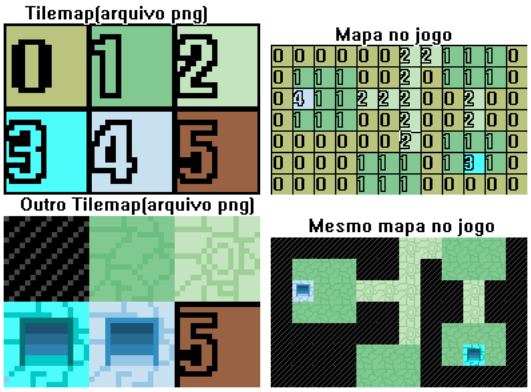

Figura 8 – Mapa, '3' e '4' indicando pontos de saída e entrada do mapa

5.3.2.2 Inimigos e Itens

Uma linha imediatamente após o último bloco ser descrito no arquivo *.map*, haverá então os itens e inimigos presentes no mapa. Ao contrário dos blocos, os inimigos e itens

não precisam ser dispostos seqüencialmente de acordo com suas posições e podem estar intercalados entre si.

Existem 3 variações possíveis de entradas nesta parte do formato. Elas são identificadas por uma linha contendo os seguintes textos:

- "*Item*:" - Itens de recuperação de vida ou de incrementos de atributos.
- "*Gold*:" - Dinheiro, para simulação dos benefícios do nível.
- "*Enemy*:" - Inimigos.

A linha subseqüente a um destes três textos conterá uma seqüencia de valores representando suas características. Em todos os três casos, os valores iniciam-se por dois identificados: x,y, representando as coordenadas do bloco x, y em que serão colocados.

5.3.2.2.1 *Gold* - Dinheiro

O mais simples deles, contém apenas mais uma variável g que representa a quantidade de dinheiro. O *sprite* escolhido para o dinheiro é automaticamente definido pelo sistema. Os primeiros seis *sprites* do mapa de itens representam os estados do dinheiro, que são definido como:

- 1-5g - *Sprite* 0
- 6-15g - *Sprite* 1
- 16-30g - *Sprite* 2
- 31-50g - *Sprite* 3
- 51-100g - *Sprite* 4
- 100g+ - *Sprite* 5

5.3.2.2.2 *Enemy* - Inimigo

Descrito pelos atributos: `hp`,`atk`,`def`,`range`,`cost`,`sprIDx`, `sprIDy`. Os primeiros valores são os atributos básicos do inimigo: sua vida, ataque e defesa.

O valor de *range* indica a visão do inimigo, isto é, a quantidade de blocos de distância que ele conseguirá observar o jogador e tomar a decisão de atacá-lo. O valor *cost* representa o custo de movimentação do inimigo, ou seja, quanto menor o valor, mais rápido será o inimigo. Por fim, os últimos dois valores representam a posição X e Y do arquivo gráfico para visualização do inimigo (*data/img/chars.png*).

5.3.2.2.3 Item

Este é o mais complicado dentre os três tipos de entrada do mapa, uma vez que pode abranger tipos de itens diferentes em uma mesma linha. É determinado pelos valores: `buff`, `hp`, `mp`, `atk`, `def`, `sprIDx`, `sprIDy`

O primeiro e mais importante parâmetro ($buff$) indica se o item é um item de recuperação ou de atributos. Tem valor 0 caso seja item de atributo ou 1 caso seja de recuperação.

O parâmetro hp e mp serão apenas utilizados caso o item possua o valor $buff$ igual a 1. E atk e def representam o quanto de ataque e defesa o jogador irá ganhar caso seja um item de aumento de atributos.

Similarmente aos inimigos, os últimos dois parâmetros indicam a posição no arquivo de imagem que irá representar graficamente o item em questão. Este arquivo é nomeado *data/img/itens.png*.

Um exemplo de um pequeno mapa 3x3 é dado a seguir (Figura 9 - 5.3.2.2.3):

```
1.0
3-3
0:0 1:1 0:0
1:1 1:1 1:1
1:1 1:1 2:4
Gold:
0,1,10
Item:
1,2,1,5,0,0,0,1,2
Item:
0,1,0,0,0,1,0,0,1
Enemy:
0,2,5,2,1,3,200,2,0
```

5.3.3 Visibilidade

É muito comum no gênero *roguelike* o conceito de visibilidade. Um mapa começa escondido e o jogador só pode ver seus itens, inimigos e blocos caso estes estejam em seu campo de visão.

Existem porém dois tipos de visibilidade. A visibilidade de terreno, ou visibilidade parcial, que identifica o tipo de bloco e itens sobre ele no momento de sua última visualização. Estes blocos costumam ser representados visualmente mais escuros para haver uma

Figura 9 – Representação do mapa - Objetos utilizados em destaque

diferenciação, e a visibilidade total, onde pode-se ver tudo que está acontecendo naquele espaço.

A visibilidade é um importante fator de jogabilidade em *roguelikes*, uma vez que muda o modo de pensar e agir de jogadores. Como eles não sabem aonde está a saída, eles devem explorar o mapa às cegas, sem conhecimento do que esta por vir, até que encontrem a saída.

O sistema simulará jogos utilizando os dois esquemas de visibilidade. Contudo, deve-se ressaltar que não há diversão em explorar um mapa em que já se saiba aonde estão seus inimigos e sua saída, o que constitui um dos principais elementos do gênero.

5.3.4 Inimigos e Batalha

A batalha da simulação ocorre através da tentativa de movimento de uma entidade sobre a outra. Por ser uma simulação simples e não um jogo complexo, uma batalha decorre-se apenas pela fórmula:

$$Dano = atkAtacante - defInimigo \tag{5.1}$$

$$VidaDefensor = VidaDefensor - Dano \tag{5.2}$$

Eliminando a entidade que chegar a vida zero primeiro. Um jogador sempre terá prioridade de ataque por ser o ator que executa a ação que gera o movimento dos inimigos, sendo a única exceção caso um inimigo seja mais rápido que o jogador e consiga realizar dois ataques em um **turno**.

Os inimigos terão uma área de observação na qual tomarão a iniciativa de atacar o jogador caso ele adentre esta área. A determinação do caminho a ser percorrido pelo inimigo costuma ser direta, devido a pequenas áreas de observações dos inimigos, porém será utilizado o algoritmo $A*$ dado a sua grande eficiência e rapidez em encontrar o melhor

caminho possível a um destino. Isto cobrirá casos de mapas com caminhos estreitos e confusos, caso seja necessário.

5.3.4.1 A*

O algoritmo $A*$ foi escolhido pela sua grande velocidade de processamento tanto de dados grandes ou pequenos e possuir uma implementação relativamente simples.

Para a implementação no sistema assumiu-se que cada quadrado explorado acarretará em 10 unidades de custo para H e o valor de G seria dado por:

$$G = (|X_i - X_f| + |Y_i - Y_f|) \times 10 \tag{5.3}$$

sendo $_i$ e $_f$ indicações das posições para o nó de origem e destino (final).

A figura (Figura 10 - 5.3.4.1) demonstra uma iteração do algoritmo. A origem se dá pelo quadrado verde escuro e o fim pelo bloco vermelho. Blocos pretos são obstáculos, enquanto azuis representam nós explorados e verdes descobertos. F_N é o custo de exploração daquele nó. Sempre explorando o nó com a menor soma total.

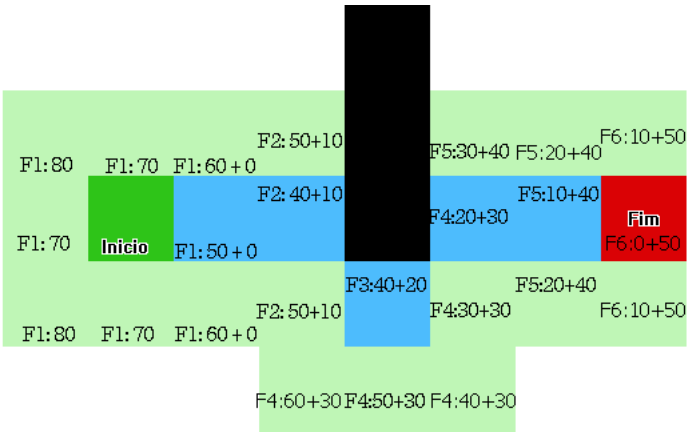

Figura 10 – Algoritmo A*

5.3.5 Informações e estados

A simulação possuirá ainda um sistema de registro (*log*) de mensagens informativas, mostrando danos recebidos e causados para a melhor jogabilidade da simulação por jogadores humanos.

As mensagens serão mostradas no canto superior da tela e escritas sobre o mapa. Estas informações serão apagadas ao toque de uma tecla ou um movimento realizado pelo jogador.

Além das mensagens haverá um painel de estados representando os diversos atributos do jogador como seu Ataque, Defesa, Vida e Dinheiro. Este painel possui fim mera-

mente ilustrativo para jogadores humanos possam realizar a simulação sem uma completa desorientação.

5.4 Automatização de jogos

Para que o sistema se torne uma importante ferramenta para auxiliar no teste da qualidade de mapas, inevitavelmente vários jogos deverão ser realizados sobre um mapa. Trazer jogadores humanos para jogar os mapas e testá-los um a um não traria um dos benefícios esperados do trabalho que é precisamente retirar o usuário do processo inicial de testes e ser capaz de analisar com um certo grau de confiança a qualidade esperada do mapa.

Desta forma, sente-se a necessidade de que as simulações de jogos do sistema sejam automatizadas.

5.4.1 Robot (BOT)

Robot, ou BOT em jogos como é mais comumente chamado, é um termo que costuma ser usado para representar a situação na qual um programa ou algoritmo é utilizado para controlar as ações que normalmente seriam efetuadas por jogadores. BOT's devem ser capazes de receber dados do ambiente em que se encontram e processá-los tomando ações de acordo com alguma regra interna estabelecida. O BOT nada mais é do que uma IA capaz de realizar o controle de ações do jogador.

O sistema utilizará um BOT para simular ações humanas e realizar o jogo de um mapa inúmeras vezes para que ele possa recolher uma grande base de métricas em um pequeno intervalo de tempo, se comparado com o que usuários reais levariam.

Uma IA porém, mesmo que esteja otimizada, não se compara a um ser humano. Humanos muitas vezes comentem erros ou podem realizar o mesmo jogo de formas diferentes dependendo de seu humor ou personalidade. Para melhor simular está discrepância entre o processo cognitivo de cada pessoas, o BOT será construído de acordo com parâmetros que o auxiliarão em sua tomada de decisões.

Desta forma, com apenas alguns ajustes nos parâmetros, pode-se criar uma IA que simularia um usuário com um perfil mais desafiador, ou um usuário com um perfil mais amedrontado.

5.4.1.1 Perfis

Os principais perfis que pode-se observar em jogadores através de diversos vídeos (**??**) (**??**) (**??**) e análises escritas podem ser divididos em:

- **Explorador** - Aquele jogador que não quer deixar nada para traz, e gosta de explorar o máximo possível do mapa.
- **Ganâncioso** - Aquele jogador que fará tudo para conseguir mais dinheiro ou itens em um mapa.
- **Corredor** - Aquele jogador que só entra em combate quando extremamente necessário, evitando-os caso possa.
- **Corajoso** - Não deixa nenhum inimigo para tráz, enfrentando todos os inimigos em seu caminho.
- **Esperto** - Um meio termo entro os outros perfis: analisa os riscos e evita batalhas as quais está em desvantagem.
- **Apostador** - Assim como o perfil anterior, analisa a situação, porém aceita riscos caso entenda que exista boas recompensas pelas ações.

Cada um destes perfis, apesar de similares, podem afetar completamente a jogabilidade de um mapa. Um nível que talvez seja impossível de se completar ao enfrentar todos os inimigos pode ser extremamente fácil para um perfil **Corredor** caso os inimigos do mapa sejam lentos e dispersos.

Desta forma, é vital a implementação da inteligência artificial do BOT para que simule as diversas formas de se jogar em um mapa.

O sistema poderá ainda utilizar um menu para escolher quais perfis deverão ser testados no mapa ou alterar e criar novos perfis pessoais pela alteração dos parâmetros da IA.

5.4.1.2 Algoritmos e parâmetros

Como dito na seção anterior, é vital a utilização de parâmetros para a conformidade com os diversos tipos de perfis identificados. Aqui será discutido um pouco sobre as técnicas e algoritmos utilizados para movimentação e tomada de decisões do BOT.

Para melhor simular a randomicidade do processamento humano, todos os BOTs utilizados não terão conhecimento qualquer sobre *tiles* que não possam ser visualizados.

A inteligência artificial do BOT inicia-se na detecção de possíveis alvos e objetos de interesse em sua visão, que são os mesmo para qualquer perfil utilizado.

A IA do BOT é chamada pela ferramenta através do arquivo *playerExplorer.lua*, que estará na pasta *data/scripts*, podendo ser alterado pelos usuários da ferramenta, embora esta não seja uma opção recomendável. O *script* será chamado a cada iteração(movimento) do BOT e irá dizer-lher qual a ação deverá ser tomada.

Em seguida é executado um algoritmo similar ao *Depth First Search* (**??**). Este algoritmo inicia-se em um ponto e expande sempre o seu nó descoberto menos distante do caminho até que seja encontrado o destino. O algoritmo utilizado segue mesmo processo de exploração do *Depth First Search* de se explorar sempre os nós mais distantes. Porém o ele é utilizado em formas de iterações, sendo que cada iteração do algoritmo é expandido um nível de movimento e criado um mapa de distâncias. Qualquer objeto de interesse ou bloco não descoberto será considerado como um possível destino (Figura 11 - 5.4.1.2).

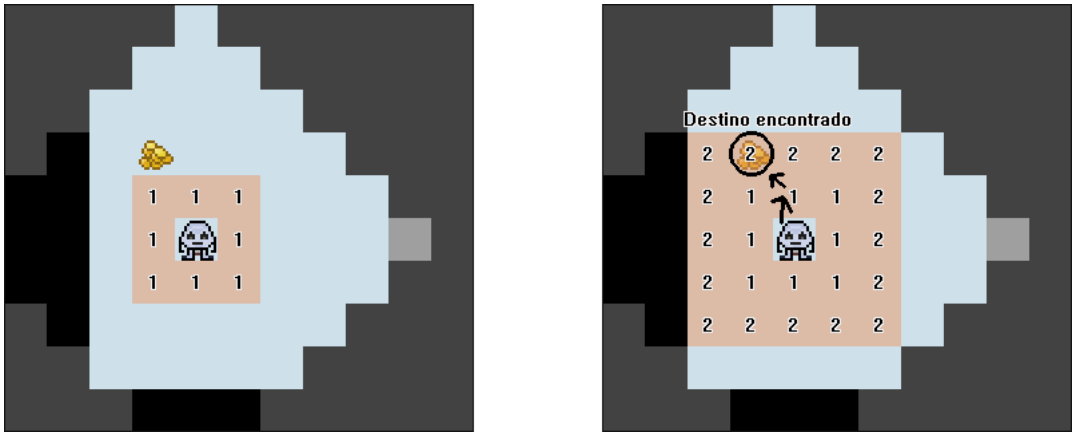

Figura 11 – *Depth First Search* em iterações por distancia

O *script* então procura por vários possíveis destinos. A cada nó descoberto é feita uma análise para identificar que tipo de bloco e que objetos estão sobre ele, a partir das seguintes regras:

- **Bloco já visto e sem itens ou inimigos** - Nada a fazer
- **Bloco não visto** - Adiciona a lista de *tiles* de interesse.
- **Bloco com itens** - Adiciona a lista de *itens* avistados.
- **Bloco com inimigos** - Adiciona a lista de inimigos avistados.
- **Bloco de terminação do nível** - Adiciona a lista de alvos prioritários.

Vale a pena lembrar que, uma vez que o bloco não seja visível, o BOT não terá conhecimento sobre o que há nele. Podendo até mesmo ser uma parede.

Este processo continua seqüencialmente junto ao descobrimento de novos blocos. A cada novo nó aberto, o BOT irá realizar a checagem para confirmar se ele chegou ao seu objetivo.

Para garantir que prioridades de escolha possam ser estabelecidades mais tarde é necessário que sejam avistados mais do que um único destino durante a descoberta, como é o caso dos algortimos de descoberta de caminho. Para se obter tais informações,

ao algoritmo avistar o primeiro objeto de interesse, ele irá marcar-se como concluido e executará um número N de iterações adicionais, guardando os novos destinos em uma lista para garantir que existam diversos possíveis alvos e que o BOT possa realizar uma análise mais abrangente.

Após rodar esta etapa do algoritmo, o *script* possuirá suas devidas listas de *tiles* avistados, itens e inimigos, e iniciará a etapa de processamento destes dados para escolher o seu novo destino (Figura 12 - 5.4.1.2).

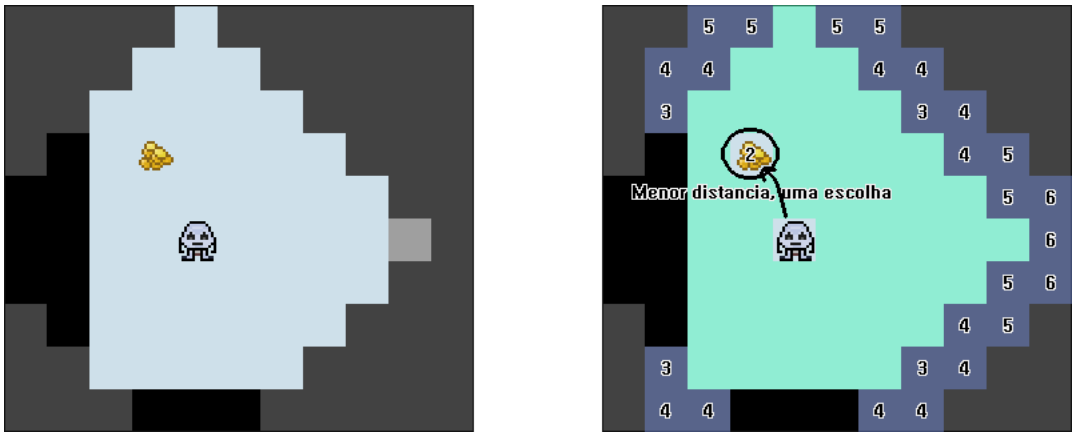

Figura 12 – Passos simplificados demonstrando o encontro dos blocos pela IA

O BOT divide a prioridade de suas escolhas baseado na proximidade, no tipo e nos *parâmetros de ganância* especificados pelo tipo de perfil. A IA também tem o conceito de blocos solitários, *singlets*, e tenta priorizar para que não sejam deixados para traz quando não muito custoso (evitando assim movimentos adicionais). Blocos solitários se identificam por um nó do mapa não visto cercado apenas por blocos visíveis. Muitas das vezes este pedaço do mapa será uma área passável, podendo haver itens. Isto dará um custo muito alto de movimento para voltar e re-explorar um bloco esquecido caso seja um perfil explorador ou, num caso extremo, tal bloco pode ser o bloco de saída do mapa.

A ordem de prioridade de escolha de alvos, sem a alteração de quaisquer *parâmetros de ganância*, são:

- 1 - *Singlets* - Blocos solitários
- 2 - Itens
- 3 - Inimigos
- 4 - Blocos inexplorados

O primeiro objeto que entrar na lista de escolhas terá o valor de escolha mínimo igual a sua distância, isto é, todos as outras listas analisadas serão somente adicionadas

a lista de possíveis escolhas caso estejam a uma distância mínima menor que o primeiro objeto prioritário encontrado. Porém, para a criação dos diversos perfis é adicionado um *parâmetro de ganância* para adicionar objetos que estejam menos distantes que sua nova distância, dado por:

$$Dist = dist - greed\Theta \tag{5.4}$$

sendo $Dist$ sua nova distância, $dist$ a distância real do objeto e $greed\Theta$ o parâmetro de ganância do determinado tipo de objeto em questão.

Desta forma é possível alterar a forma com que a IA irá colocar as suas escolhas de acordo com o resultado esperado daquele perfil. Por exemplo, um perfil explorador e ganancioso terá altos parâmetros de ganância para itens e possuirá um maior número de iterações extras para melhor identificar itens nas redondezas, enquanto um perfil Corajoso possuirá altos parâmetros com inimigos (Figura 13 - 5.4.1.2).

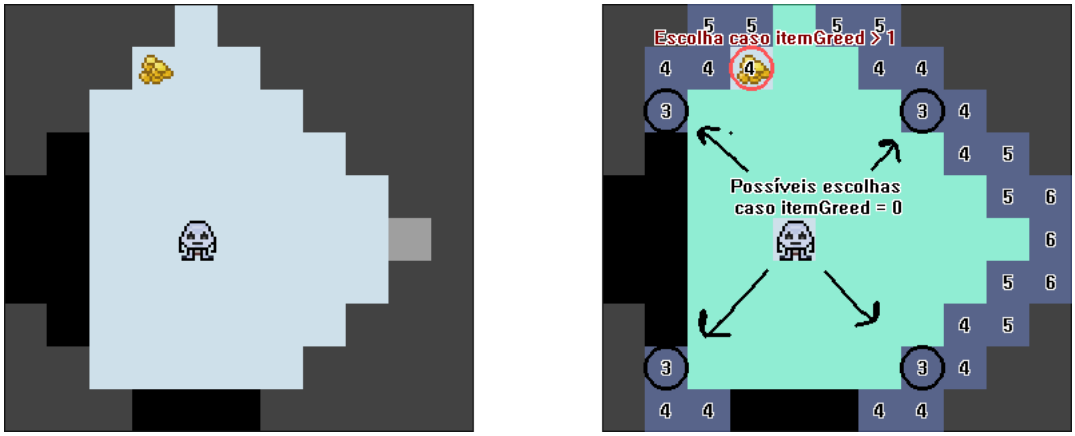

Figura 13 – BOT - Alternativas e parâmetros de ganância

Ao final do processo, caso não hajam mais meios de filtrar a lista de escolhas restantes, é então escolhido um alvo aleatório dentro das possíveis escolhas.

Por fim, o *script* chamará a função da classe do jogador de dentro do sistema e pedirá que seja construído o caminho até o alvo encontrado.

Na próxima ação que o BOT for tentar executar, será antes checado se é necessário que haja uma nova análise dos dados para escolha de um novo alvo ou se a IA deverá prosseguir até o seu alvo antes de iniciar o processamento novamente. Isto é testado pela verificação se existe uma rota existente em progresso: caso não haja, simplesmente chama-se a análise novamente. Porém, caso seja encontrada uma rota, o *script* ainda fará verificações para otimizar a inteligência dele e garantir que não esteja andando para um grupo de inimigos ou um visível beco sem saída, por exemplo.

Desta forma, mesmo havendo rotas presentes para o jogador, caso aviste algum **novo** inimigo em seu caminho ou realize um movimento de tal forma que o bloco destino esteja visível porém não note a presença de quaisquer novos blocos, irá ser realizada uma

nova análise e a escolha de novos destinos (Figura 14 - 5.4.1.2).

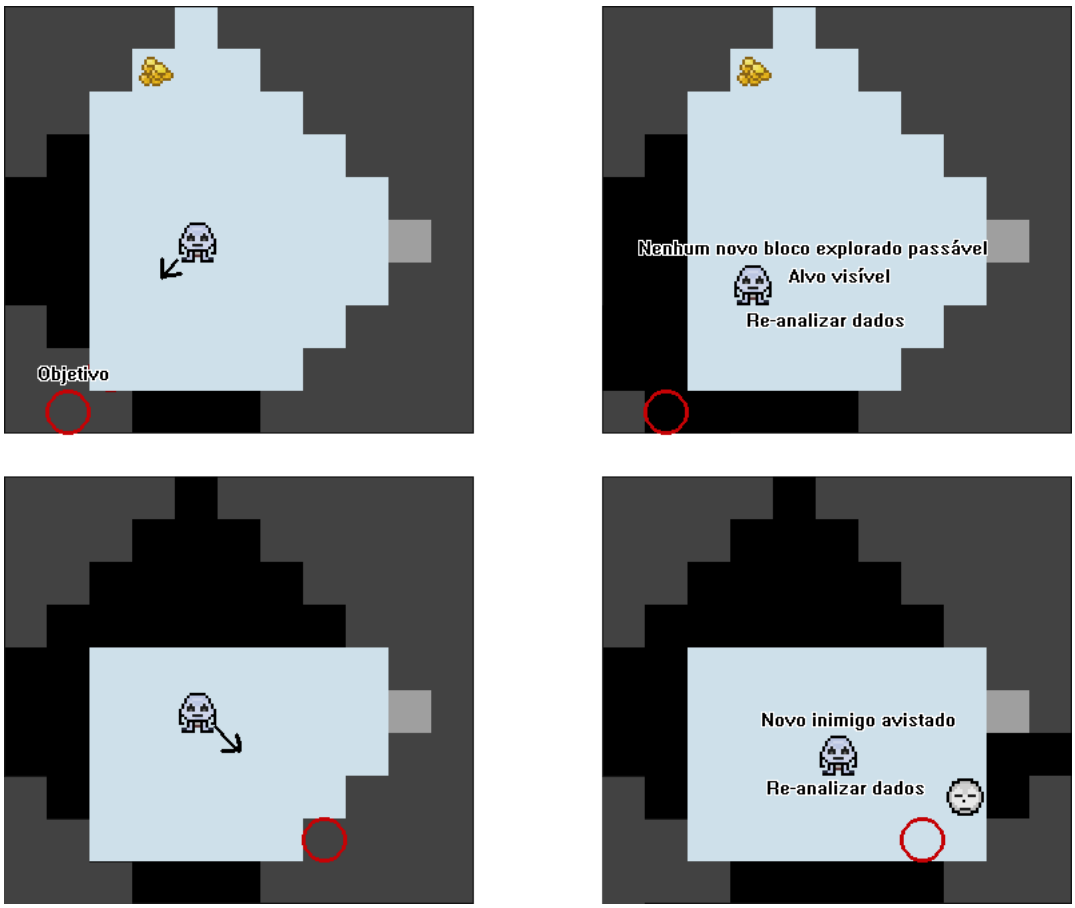

Figura 14 – Situações de nova análise dos dados pela IA

5.5 Métricas

O sistema terá um módulo de métricas que será capaz de extrair uma série de métricas sobre o decorrer de um jogo simulado. As métricas são recolhidas e armazenadas, separando-as para cada mapa e rodada em que foram testadas.

Desta forma, supondo que existam a métricas A, B e C e seja realizado três jogos em um dado mapa. A ferramenta deverá ser capaz de salvar informações sobre as métricas obtidas A, B e C para cada uma daquelas rodadas e exibir também os valores compilados de todas as métricas, disponibilizando para visualização o seu desvio padrão amostral e variância.

O primeiro passo que deve-se fazer é colocar na ferramenta os valores esperados ideais para o mapa de acordo com a expectativa do usuário da ferramenta (provavelmente do *game designer* da equipe). Não há uma qualidade geral de um mapa: um mapa pode

ter diferentes objetivos, fica a cargo do usuário decidir que tipo de mapa ele espera. Se é um mapa para um jogo rápido e casual, provavelmente será um mapa com um pequeno tempo de duração e dificuldade. Ou se for um jogo com mais ação, provavelmente seja desejado que haja uma maior quantidade de inimigos e espaço livre para a movimentação.

5.5.1 Métricas utilizadas

As principais métricas identificas durante esta primeira parte do trabalho foram:

- **Movimentos**:
  Indica a quantidade de passos realizados. Através desta métrica é capaz de obter-se a duração de um jogo. O tempo utilizado para a realização de um turno em um jogo do gênero possui uma grande variação, dependendo do momento em que o jogador se encontra, ele pode demorar vários segundos em um único turno em situações de batalha ou vários completar vários turnos em 1 segundo, para corredores e áreas já vistas. A conversão de turnos-segundos utilizada foi 1:1, 1 turno(movimento) igual a 1 segundo. Futuros testes com usuários reais serão necessários para ajustar esta razão de conversão.

- **Itens coletados**:
  A métrica indica a quantidade de itens coletados. Isto pode indicar se o nível criado é recompensador o suficiente para o jogador. Esta métrica pode ser tão simples quanto a quantidade de itens coletados, ou a utilização de um fator para representação da importância do item coletado para o jogador. Pode também dividir-se em métrica sobre a quantidade de itens de atributos, que aumentam as habilidades do jogador, ou métrica sobre a quantidade de itens de recuperação.

- **Dinheiro coletado**:
  Indica a quantidade de dinheiro coletado através do jogos. É possível a visualização de quão recompensador foi o jogo, e garantir que os mapas tenham um progresso consistente na evolução do jogo. Por exemplo, para manter um jogador motivado em um jogo é comum que se tenha um progresso constante: grandes variações neste fluxo pode obter efeitos negativos, como a desmotivação do jogador, ou deixar o jogo muito fácil, devido a capacidade de compra de melhores itens muito rápido.

- **Inimigos:**
  Esta métrica indica a quantidade de inimigos derrotados. Também pode ser expandida para a quantidade inimigos vistos, porcentagem de inimigos vistos e/ou derrotados. Pode ser utilizada para o balanceamento dos mapas, garantindo que os mapas não possuam uma grande dificuldade.

- **Vitórias:**
  Quantidade de vitórias e derrotas que um BOT obteve em N simulações de partidas em um mapa. Estabelece o quão aceitável o mapa pode ser e a progressão de dificuldade. Jogos casuais, por exemplo, possuem pequeno aumento de dificuldade entre mapas e normalmente costumam ter vitórias fáceis, enquanto jogos *hardcore* costumam ser extremamente difíceis. O jogo *Dungeon Crawl Stone Soup*[1], voltado ao *roguelike* clássico por exemplo, realizou uma competição (**??**) utilizando 1749 pessoas, com uma média de vitórias de 1.37% e uma média de duração de jogo de 14.8 horas.

5.5.2 Análise de métricas

Uma das grandes partes deste trabalho é descobrir uma forma quantitativa de qualificar de forma significante uma série de dados obtidos dos mapas como bons valores ou ruins de acordo com um intervalo ideal de entrada e através da compilação destes valores identificar se um dado mapa é qualificado como bom ou ruim para aquele tipo de resultados esperados.

Uma das primeiras alternativas buscadas para se resolver este problema foi a utilização de distribuições normais de probabilidade e qualificar a porcentagem de chance dos resultados obtidos pertencerem a ela. Esta forma de avaliação porém apresenta em sua definição um grande problema para se adequar ao problema.

Uma distribuição normal de probabilidade estima-se que os valores estejam normalmente distribuídos, isto é, quanto mais próximo a média populacional, maior será a concentração de resultados. Isto não é sempre o que acontece em jogos, a alternativa de explorar primeiro um caminho A ou um caminho B pode mudar significativamente o resultado do jogo, tornando os resultados algo mais próximo a uma série clusterizada, semente em alguns casos obtendo algo próximo ao desvio normal, e mesmo assim contendo também variações anormais.

A análise das métricas será realizada de acordo com parâmetros dispostos pelo usuário da ferramenta. Desta forma, é possível testar se um dado mapa tem a qualidade esperada de um mapa casual ou um mapa mais voltado ao *roguelike* clássico.

O usuário da ferramenta deverá então ser capaz de inserir no sistema os valores esperados para as métricas ou então escolher ignorá-las por completo, classificando-as como irrelevantes para ele.

A ferramenta será capaz de salvar perfis de métricas para evitar que o usuário tenha que redigitá-las a cada vez que inicie o programa. Poderão ser definidos alguns perfis de métricas padrões escolhidos para acelerar e facilitar o processo para novos usuários.

[1] http://crawl.develz.org/wordpress

Através das escolhas das métricas esperadas e métricas obtidas a partir de diversos jogos, a qualidade do mapa será dada por um único fator de qualidade geral, obtida através de uma função de distribuição normal.

O sistema irá gerar uma função de distribuição de densidade (**??**) para cada uma das métricas a partir dos desvios padrões e valores esperados da agregação dos valores para aquelas métricas. Desta forma, uma dada métrica possui m medidas de acordo com o número de partidas realizadas:

$$data = \begin{bmatrix} m_1 \\ m_2 \\ ... \\ m_n \end{bmatrix} \tag{5.5}$$

Utilizando estes dados é obtido o valor esperado (média aritmética) e o desvio padrão amostral são obtidos.

$$\mu = \frac{\sum_{i=1}^{N} m_i}{N} \tag{5.6}$$

$$\sigma = \sqrt{\sum_{i=1}^{N} \frac{(\mu - m_i)^2}{N(N-1)}} \tag{5.7}$$

sendo μ o valor esperado (média) e σ o desvio padrão. É construído uma função de densidade de probabilidade utilizando o método da distribuição normal.

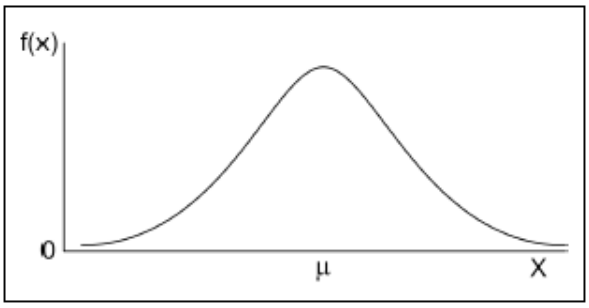

Figura 15 – Distribuição normal dos dados

Por fim, a qualidade para métrica é a probabilidade de que o valor esperado para a métrica, $E(M)$, esteja entre os dados com uma variação ΔX que será dada por 20% do valor de $E(M)$.

Desta forma, mesmo que uma métrica esperada se aproxime do valor médio, ela mesmo assim não terá uma boa qualidade a menos que os valores tenham pouca variação.

O teste da qualidade utilizando a confiança de 20% é apenas um teste, na segunda parte do projeto será avaliado a sua relevância quanto sua assertividade e possivelmente deixando a alteração da porcentagem de confiança pelo usuário, caso assim desejar.

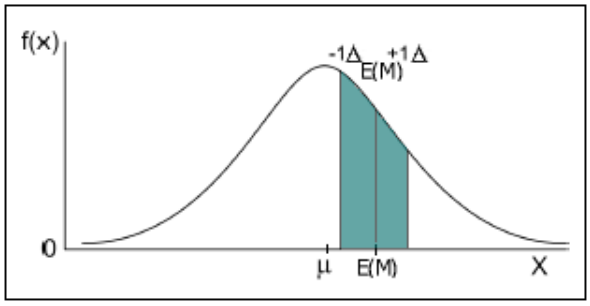

Figura 16 – Distribuição normal dos dados

A obtenção na prática dos valores das probabilidades (qualidade) se dá através dos valores Z obtido pelas tabelas das distribuições normais de P(Z < X), conforme apresentado na Equação 5.8.

$$Z = \frac{E(M) - \mu}{\sigma} \tag{5.8}$$

e qualidade da métrica será então obtida por:

$$P(Z_1 < Z < Z_2) = P(Z < Z_2) - P(Z < Z_1) \tag{5.9}$$

sendo Z_1 e Z_2 os limites de aceitação da métrica:

$$Z_1 = \frac{(E(M) - \Delta X) - \mu}{\sigma} \tag{5.10}$$

$$Z_2 = \frac{(E(M) + \Delta X) - \mu}{\sigma} \tag{5.11}$$

Por fim, a métrica de qualidade geral do mapa Q será equivalente a média aritmética de todas as qualidades medidas q_i.

$$Q_{total} = \frac{\sum_{i=1}^{N} q_i}{N} \tag{5.12}$$

5.6 Ferramenta auxiliar

A ferramenta auxiliar foi desenvolvida no *framework* Qt utilizando a linguagem C++ para ajudar no desenvolvimento e observação de mapas, uma vez que construí-los através das regras definidas pelos arquivos *.map* se torna extremamente ineficaz e cansativo se feitos a mão. A ferramenta permite a criação e alteração de mapas, sejam eles feitos a mão ou mapas gerados proceduralmente em tempo de execução.

A interface é adaptável e permite a utilização de janelas flutuantes ou janelas fixadas nas bordas do programa, dando o usuário a liberdade de criar a sua área de trabalho do jeito que lhe seja mais desejável. A interface de edição possui 3 modos de operação

para a alteração do mapa que servem para melhor orientar o usuário da ferramenta em relação ao que está sendo feito.

O primeiro, é o modo de edição de gráficos, neste modo o usuário pode utilizar utilitários como a ferramenta "pincel"para escrita dos blocos no mapa unitáriamente e a ferramenta "retangulo"para a criação de blocos em área. Isto permite a criação mais efetiva do mapa, dando também ao usuário a opção de e selecionar os blocos que desejam ser pintados um a um, ou em grupos.

O segundo modo é pertinente a edição de entidades, neste modo o usuário poderá criar e alterar entidades para o seu mapa, criar entidades padrões, copiá-las e posicioná-las da forma que desejar. As entidades estão divididas em 3 grandes tipos genéricos: Itens, Inimigos e Dinheiro.

O ultimo modo é o modo de caminhos, com ele é possível visualizar quais blocos são passáveis e quais blocos não serão. Ferramentas para edição rápida como o "retangulo"e a cópia em grupo também se aplicam para esse modo.

Todos os utilitários da aplicação podem ser utilizados através de atalhos de teclado e ela também disponibiliza algumas opções para melhor visualização como a opção de mostrar ou não a malha de quadrados. Por fim, existem 2 botões sinalizando bandeiras que indicam aonde serão as entradas e saídas do mapa, indicando aonde o jogador começará o jogo e aonde poderá chegar para terminá-lo.

Como um outro extra para a ferramenta, foi adotado uma biblioteca *qscintilla* que fornece uma API para a representação de algumas funcionalidades de editores de texto e de código. Com ela foi possível construir uma mini-IDE embutida na ferramenta com a capacidade de interpretar comandos e sintaxes da linguágem *lua* para a criação de *scripts* procedurais e se ter a visualização dos resultados dele na sua frente no mesmo momento, indicando também os erros que ocorreram durante a interpretação dele.

6 Resultados

6.1 Técnicas de coleta de metrica

6.2 Avaliação das métricas colhidas

7 Resultados Alcançados

A realização do trabalho até este ponto resultou em um protótipo de parte da ferramenta. O protótipo é capaz de carregar mapas, ítens e inimigos a partir do formato especificado e realizar uma simulação básica de jogos. A ferramenta é capaz de coletar métricas básicas sobre:

- Quantidade de passos;
- Inimigos derrotados;
- Porcentagem de vitórias;
- Dinheiro coletado;

Ainda não é realizada a concretização das métricas em um valor único de qualidade esperada, porém ela gera as médias, desvios padrões e variâncias para medidas de N jogadas em um mesmo mapa.

O sistema possui um BOT parametrizado com variáveis de ganância com itens, *tiles* solitários e número de iterações extras, podendo ser alterado somente através da modificação dentro do *script* Lua no momento. Não há ainda a diferenciação de perfis por parte do sistema.

Ítens e inimigos são carregados através de *scripts* Lua, atribuindo-os a um identificador de texto para serem chamados pela função de geração procedural do mapa de forma mais consistente.

A ferramenta disponibiliza uma simples interface de botões e menus para ajustes de parâmetros para o algoritmo procedural do mapa que será gerado ao iniciar o jogo. É disponibilizado também um botão para carregar um mapa de um arquivo, sem a geração de um novo mapa.

As devidas condições de vitória e derrota para o jogador estão implementadas, e o sistema possui um console interno que pode ser chamado para alterar propriedades de *debug*, como *setFog* para remoção ou adição de visibilidade, ou *botDelay*, para ajustes na velocidade de processamento do BOT.

Inimigos estão atacando o jogador quando dentro de sua área de visão e as devidas mensagens de informações são mostradas em uma área na parte de cima da tela como: “Você causou 5 de dano ao inimigo”.

O sistema está utilizando gráficos 16x16 pixels para representação visual de seus elementos.

7.1 Conclusão

O trabalho realizado possibilitou a confirmação da viabilidade da ferramenta e constatou que a necessidade que se tem em obter métricas concretas para mapas gerados automaticamente para acelerar o processo de desenvolvimento de jogos do gênero *roguelike*, é um fator real.

A definição de diversos perfis de usuário foram observados e classificados para que uma melhor representação das métricas obtidas dos jogos simulados. A ferramenta para a avaliação dos mapas, assim como a forma com que serão testados e extraídas as métricas foram estabelecidas, confirmando-as através da aplicação prática do sistema através de protótipos obtendo a comprovação de que seus conceitos básicos estão funcionando e sendo aplicados.

O trabalho obteve resultados positivos quanto aos seus benefícios para o desenvolvimento dos jogos, possibilitando a realização de simulações e obtenção de métricas de dezenas de jogos em apenas alguns minutos.