Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Evoluindo Representações de Mapas para o jogo Cube 2: Sauerbraten

Davi Silva Valério Diniz

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientador
Prof. Dr. José Carlos Loureiro Ralha

Brasília
2015

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Evoluindo Representações de Mapas para o jogo Cube 2: Sauerbraten

Davi Silva Valério Diniz

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof. Dr. José Carlos Loureiro Ralha (Orientador)
CIC/UnB

Prof.ª Dr.ª Carla Denise Castanho
CIC/UnB

Prof. Dr. Guilherme Novaes Ramos
CIC/UnB

Prof. Dr. Homero Luiz Piccolo
Coordenador do Bacharelado em Ciência da Computação

Brasília, 16 de novembro de 2015

Dedicatória

Dedico este trabalho à minha família e meus amigos.

Agradecimentos

Agradeço à Universidade de Brasília, Luigi Cardamone, o Departamento de Ciência da Computação e seus docentes.

Resumo

Este trabalho apresenta alternativas às representações de mapas introduzidos no artigo *"Evolving Interesting Maps for a First Person Shooter"*, de Luigi Cardamone et al.. Representação de mapas é utilizado para facilitar ou diminuir os custos de desenvolvimento para Jogos de Tiro em Primeira Pessoa. Para esta monografia, desenvolvemos quatro representações de mapas que aderem as técnicas de projeto de níveis do jogo Cube 2: Sauerbraten (C2) descritas por Marc Saltzman em *Secrets of the sages: Level design*. Visando possibilitar a comparação das representações, implementamos três das quatro representações de mapas descritas em *"Evolving Interesting Maps for a First Person Shooter"* bem como as representações propostas nesta monografia. Dessa forma, tornou-se possível realizar experimentos de comparação baseadas nas mesmas métricas e testes usados por Luigi Cardamone et al.. Os resultados das comparações demonstram que as representações de mapas desenvolvidas para esta monografia podem ser utilizadas como alternativas as descritas em *"Evolving Interesting Maps for a First Person Shooter"*.

Palavras-chave: Tiro em primeira pessoa, Geração procedural de conteúdo, Jogos, Algoritmos evolucionários, Experiência de jogador, Otimização, Cube 2: Sauerbraten

Abstract

This work attempts to create alternative map representations to the ones presented in the research *"Evolving Interesting Maps for a First Person Shooter"* [5] utilized to facilitate or lessen the development costs of First Person Shooter (FPS)es. We propose four representations that follow the design techniques for the game Cube 2: Sauerbraten (C2) as described by [36], these will be compared to three of the four map representations described in [5] with the same metrics and tests. These comparisons gave us results that show that our map representations can be used as viable alternatives to the ones described in [5].

Keywords: First-Person Shooters, Procedural content generation, Games, Evolutionary algorithms, Player experience, Search-based, Cube 2: Sauerbraten

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

AAA triplo-A, um acrônimo utilizado pela indústria de jogos para classificar um jogo de alta qualidade.

ASGGM Algoritmo de Seleção Genética e Geração de Mapas.

C2 Cube 2: Sauerbraten.

FPS First Person Shooter.

GA Algoritmos Genéticos.

GESDT Generalized Extreme Studentized Deviate Test.

NPC Non-Player Characters.

PC Pontos Cíclicos.

PGC Procedurally Generated Content.

PTG Procedural Terrain Generation.

SBPCG Search-Based Procedural Content Generation.

SC1 Salas Cíclicas 1.

SC2 Salas Cíclicas 2.

SC3 Salas Cíclicas 3.

Capítulo 1

Introdução

Com o crescimento da indústria de jogos digitais nos tempos recentes, há uma incessante demanda por conteúdo digital para apoiar o desenvolvimento destes jogos [37]. Este conteúdo é, em sua grande maioria, criado manualmente por equipes de artistas, *designers* e programadores. Estas equipes se expandem a cada nova geração de *hardware* para suprir as demandas dos consumidores [31]. *Assassin's Creed 4 (2013)* da *Ubisoft*, um jogo considerado triplo-A, um acrônimo utilizado pela indústria de jogos para classificar um jogo de alta qualidade (AAA), envolveu cerca de 1.000 pessoas em seu desenvolvimento [40]. *Doom (1993)*, um jogo AAA na época, contou com somente 10 desenvolvedores [23].

Grande parte do tempo de desenvolvimento é dedicado à criação de cenários — níveis que serão usados para entreter o jogador [19]. Estes cenários são feitos à mão, testados e revisados exaustivamente pela equipe de desenvolvimento e testadores antes de serem inseridos no produto final [36]. Uma possível solução para diminuir o tempo deste processo é a automatização da criação de conteúdo, na qual técnicas de geração procedural de conteúdo são de grande eficiência [24]. Estas técnicas eram utilizadas mais popularmente em jogos antigos como *The Elder Scrolls II: Daggerfall(1996)*, que utilizava geração de terreno aleatório para criar um mundo de aproximadamente 62.000 milhas quadradas para contornar as limitações de consumo de memória da época [22].

Atualmente, grande parte destas limitações não existe mais já que as placas gráficas modernas são projetadas: (i) baseadas no conceito de fluxo de dados — altamente paralelizável; (ii) possuindo memória própria de alto desempenho. Contudo, as técnicas de geração procedural de conteúdo ainda podem ser úteis, não somente para grandes desenvolvedores, como também para desenvolvedores independentes [2]. Jogos com modos multijogador, que dependem de uma comunidade ativa de jogadores para se manterem lucrativos no mercado [13], poderiam igualmente desfrutar de grandes vantagens utilizando técnicas de geração procedural de conteúdo, criando novo conteúdo com facilidade objetivando manter seus jogadores interessados.

Um dos gêneros mais populares de jogos com modo multijogador é o jogo de tiro em primeira pessoa, ou First Person Shooter (FPS), cujo estilo depende principalmente do cenário em que os jogadores se encontram. Esta característica é ainda mais evidente em FPSes do subgênero *Arena Shooter*, cuja característica central em sua jogabilidade é o controle de áreas. Por esta razão, os mapas utilizados por estes jogos passam por inúmeras revisões, pois precisam ser balanceados para que os jogadores não estejam em situações onde sempre perderão ou ganharão [29].

Em *"Evolving Interesting Maps for a First Person Shooter"* [5], os autores discutem estes assuntos e propõem não só uma métrica para distinguir mapas desejáveis dos indesejáveis, mas também apresentam quatro representações de mapas em conjunção com um algoritmo de seleção. Não existem outras representações de mapas disponíveis para o jogo C2 além destas propostas em [5].

Neste trabalho propomos quatro novas representações de mapas que utilizam técnicas de projeto e construção tradicionais para FPSes descritas em [36]. Sendo uma alternativa ao trabalho apresentado em [5], torna-se imperativo realizar uma comparação de nossas representações à três das quatro representações de mapas propostos em [5]. Para tanto, foi utilizado a métrica e o código fonte modificado do jogo Cube 2: Sauerbraten (C2) implementados em [5].

1.1 Problema

Há poucas representações de mapas disponíveis para o jogo C2. Em [5], os autores introduzem representações de mapas para C2; contudo, essas representações *não seguem as técnicas tradicionais* de projeto para jogos do tipo FPS.

Esta monografia introduz novas representações de mapas para C2 que *seguem as técnicas tradicionais de projeto e construção* para FPSes descritas em [36].

Dessa forma, se torna possível comparar as duas abordagens técnicas para a geração de mapas para o jogo C2.

1.2 Motivação

Este trabalho tem como motivo principal criar novas alternativas de representações de mapas que diferem das descritas em [5], para projetistas de níveis que desejam utilizar representações de mapas para facilitar o desenvolvimento dos mesmos para o jogo C2.

1.3 Objetivo

Este trabalho tem por objetivo propor quatro representações de mapas 2D, ou seja, mapas de altura, para o jogo C2, que consigam alcançar resultados tão bons quanto ou melhores que os apresentados em [5].

1.3.1 Objetivos específicos

- Reimplementar três das quatro representações de mapas descritas em [5];
- Desenvolver quatro *novas* representações de mapas que sigam as técnicas tradicionais de construção de mapas para FPSes;
- Implementar um algoritmo de seleção genética que consiga evoluir os mapas gerados a partir das representações de mapas;
- Comparar resultados entre as nossas representações e as representações descritas em [5].

1.4 Estrutura do Trabalho

Esta monografia está organizada da seguinte maneira. O Capítulo 2 apresenta uma breve descrição de jogos em primeira pessoa e suas características. O Capítulo 3 apresenta o desenvolvimento realizado neste trabalho no algoritmo de seleção genetica e nas representações de mapas. O Capítulo 4 apresenta os resultados obtidos nos experimentos realizados comparando-os aos apresentados em [5]. Discute-se também os prós e contras de cada abordagem. O Capítulo 5 apresenta as conclusões e propõe alternativas para trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Neste capítulo, descreveremos toda fundamentação teórica necessária para auxiliar na leitura e compreensão dos temas trabalhados. Primeiramente iremos apresentar e descrever os gêneros de jogos. Em seguida, uma categorização do conteúdo que compõe jogos. Após isso descreveremos conteúdo gerado proceduralmente, seguido da descrição de algoritmos genéticos e os testes estatísticos utilizados neste trabalho. Finalizamos descrevendo o jogo *Cube 2: Sauerbraten* e as regras de geração de mapas para o mesmo.

2.1 Jogos

De acordo com [33], os jogos eletrônicos atuais podem ser divididos em múltiplos gêneros e subgêneros. Gêneros são utilizados para descrever o estilo de jogabilidade e são descritos da seguinte maneira:

- *Ação*: jogos de ação que requerem coordenação óculo-manual para jogar. Seus subgêneros são: Ação-aventura, Ação-arcade, Plataforma, Furtivos, Luta, *Beat 'em up/hack 'n' slash.*[1]
- *Tiro*: jogos deste estilo tem um foco em lançar projéteis em inimigos. Seus subgêneros se distinguem principalmente pela visão da câmera de jogo: Tiro em Primeira Pessoa; Tiro em Terceira Pessoa.
- *Aventura*: jogos de aventura tem um foco na resolução de quebra-cabeças, coleta de itens e gerência de inventário. Seus subgêneros são: Aventura Gráfica, *Role-playing game, Massively multiplayer online role-playing game*, Sobrevivência/Horror.
- *Construção/Administração*: este gênero foca na construção e expansão de locais com recursos limitados.

[1]Na ausência de terminologia consolidada em língua portuguesa, usaremos em diversas ocasiões a nomenclatura original em inglês.

- *Simulação da Vida*: similar ao gênero de Construção/Administração, mas com um foco na construção de relacionamentos com formas de vida artificiais. Seus subgêneros são: Simulação de animais de estimação.
- *Música/Ritmo*: o jogador tenta se manter no ritmo de uma música.
- *Festa*: jogos de festa desenvolvidos especificamente para múltiplos jogadores com jogabilidade competitiva.
- *Quebra-cabeças*: são jogos baseados em lógica e completar padrões. Podem ser lentos, metódicos ou usam coordenação óculo-manual.
- *Esportes*: baseados em competições atléticas, sendo essas tradicionais ou extremas. Seus subgêneros são: Administração de Esportes.
- *Estratégia*: pensar e planejar são a marca de jogos de estratégia. Eles podem ocorrer em cenários fictícios ou históricos. Seus subgêneros são: Estratégia em Tempo Real, Baseado em Turnos, Defesa de Torres.
- *Simulação de veículos*: jogadores simulam o ato de pilotar ou dirigir um veículo, de carros esportivos a naves espaciais. Há uma ênfase em fazer a experiência ser o mais realista possível. Seus subgêneros são: Dirigir, Voar.

Neste trabalho, focamos com jogos do gênero Tiro, mais especificamente, do subgênero Tiro em Primeira Pessoa.

2.2 FPS

FPS é um gênero popular de jogos com uma história de quase três décadas repleta de títulos comerciais. Segundo [17], *Wolfenstein 3D* (id software, 1992) e *Catacomb 3-D* (id software, 1991) foram os títulos que popularizaram o gênero e muitas das características dos mesmos são encontradas até hoje na grande maioria dos FPSes. Estas características, de acordo com [17], são:

- utiliza uma perspectiva em primeira pessoa; a câmera do jogo tenta mostrar a visão a partir dos olhos do *avatar*[2].
- o jogador tem um *avatar* antropomórfico que interage diretamente com o mundo do jogo.

[2]Avatar é a representação virtual do jogador dentro do jogo

- no modo primário de jogo, o *avatar* do jogador tenta causar dano a outras entidades do jogo e estas entidades do jogo tentam causar dano ao *avatar* do jogador. Entidades são qualquer tipo de objeto ativo dentro do jogo, podendo ser outros jogadores humanos ou objetos controlados pelas mecânicas do jogo.
- o movimento do *avatar* do jogador é controlado principalmente pelo jogador.

Jogos recentes deste gênero costumam apresentar um modo de jogo *online*, onde jogadores podem participar de partidas onde os únicos adversários disponíveis no mundo de jogo são outros jogadores. Esse tipo de partida foi apelidada de *deathmatch* por um dos desenvolvedores do jogo *Doom (id software, 1994)*, e foi uma das principais razões da popularidade do mesmo ao ponto de grandes companhia da época como *Intel*, *Lotus Development* e *Carnegie Mellon University* criarem políticas internas que proibiam jogar *Doom* nas redes internas dos mesmos em horas de trabalho [23]. O modo de jogo *deathmatch* e outros modos populares de jogo em FPSes serão apresentados na Seção 2.8 onde explicaremos em detalhes o jogo escolhido para este trabalho.

FPSes, por terem uma perspectiva em primeira pessoa, são jogos cujos ambientes virtuais tentam ser realistas e extremamente detalhados. Além disso, estes ambientes precisam ser desenvolvidos para prover uma experiência de jogo gratificante aos jogadores, e alcançar esse objetivo não é uma tarefa trivial.

A área de *projeto de níveis* tem pouco conhecimento formal. Mesmo existindo vários livros sobre o assunto [4, 7, 8, 14], a literatura explica a história sobre *projeto de níveis* e as práticas na indústria, mas não explora como os mapas influenciam a jogabilidade [20]. Essa é uma área essencial para FPSes pois os mapas impõem sobre o jogador sequências de desafios que precisam ser conquistados (geralmente em mapas de somente um jogador, chamados *singleplayer*) ou áreas onde combate entre jogadores pode ocorrer (geralmente em mapas de modo multijogador, chamados *multiplayer*) [20].

Mapas geralmente passam por inúmeras iterações e modificações até alcançarem uma boa jogabilidade [18]. Nesta monografia vamos utilizar técnicas de geração de conteúdo procedural para tentar aliviar a quantidade de trabalho requerida para criar ambientes virtuais para jogos do estilo FPS. Descreveremos na próxima seção o que pode ser considerado como um conteúdo de jogo e quais desses conteúdos serão utilizados na geração de nossos mapas.

2.3 Conteúdo de Jogos

Conteúdo de jogo é qualquer parte do jogo que afeta a jogabilidade, mas não são parte do comportamento de Non-Player Characters (NPC) (entidades controladas pelo jogo,

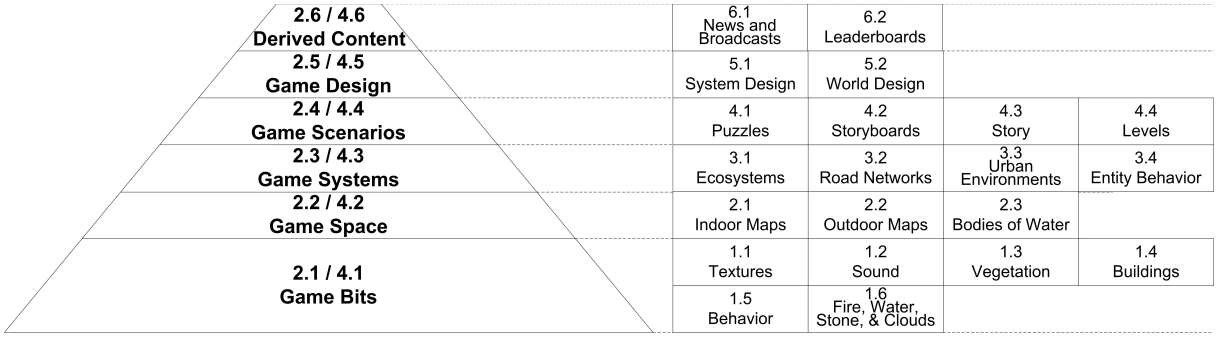

Figura 2.1: A pirâmide das classes de conteúdo.

não pelo jogador) ou do motor do jogo. Mapas, níveis, personagens, regras de jogo, música, sons, itens, estórias e diálogo são exemplos do que é considerado como conteúdo do jogo. Segundo [45], a caracterização apresentada para NPCs exclui o uso de técnicas de aprendizagem e busca, características de IA. Hendrikx, Mark et al. [16] aprofundam a definição de conteúdo de jogo dividindo-a em seis classes em uma pirâmide hierárquica, com as classes no topo da pirâmide podendo ser criadas por elementos das classes da base da pirâmide (Figura 2.1)

Estas 6 classes são: Bits de Jogo, Espaço de Jogo, Sistemas de Jogo, Cenários de Jogo, Projeto de Jogo e Conteúdo Derivado. A seguir serão definidas cada umas delas e seus sub-elementos de acordo com [16].

2.3.1 Bits de Jogo

Bits de Jogo são descritas como unidades elementares de conteúdo do jogo. Elas podem ser divididas em *bits* abstratos e *bits* concretos. *Bits* concretos são objetos que podem sofrer interação dentro do mundo de jogo. Um exemplo de um *bit* concreto seria uma árvore. *Bits* abstratos, como som e texturas, precisam ser combinados para produzir um bit concreto.

Existem 6 tipos de bits de jogo descritos por Hendrikx, Mark et al. [16]: texturas; som; vegetação; prédios; comportamento e fogo, água, pedra e nuvens.

Texturas:

Imagens superimpostas em modelos e geometria para adicionar detalhes, e para dar representação visual de elementos do jogo como menus. Hoje em dia grande parte das texturas de um jogo é gerada proceduralmente. É um *bit* abstrato.

Som:

A música é utilizada para criar uma atmosfera em um jogo. Efeitos sonoros são usados para prover uma resposta para o jogador em ações ou mudanças de ambiente. É um *bit* abstrato.

Vegetação:

Usada em inúmeros jogos para criar um ambiente mais realístico e portanto, mais imersivo. Pode servir como detalhe estético, recurso ou um lugar para se esconder. É um *bit* concreto.

Prédios:

Essenciais para representar ambientes urbanos em jogos. Assim como a vegetação, não são usados somente como um detalhe estético. Prédios podem influenciar na jogabilidade do jogo, e dependendo do tema do jogo, prédios precisam ser diversos e interessantes, mantendo um estilo arquitetural unitário. É um *bit* concreto.

Comportamento:

É a forma como objetos do jogo interagem entre si e com o ambiente. Comportamento procedural é geralmente usado em jogos para criar a ilusão de complexidade. É um *bit* abstrato.

Fogo, água, pedra e nuvens:

São usados em jogos para criar mundos mais realísticos. Em jogos antigos, estes bits de jogo eram puramente decorativos, somente texturas e sons eram necessários para criá-los. Contudo, recentemente, avanços em computação e modelagem dão a possibilidade de usar representações mais detalhadas, realísticas e interativas desses elementos. É um *bit* concreto.

2.3.2 Espaço de Jogo

Espaço de Jogo é o ambiente em que o jogo ocorre. É parcialmente cheio com bits de jogo pelos quais os jogadores se movimentam. Pode ser definido como abstrato ou concreto.

Possíveis exemplos de espaços de jogos concretos são florestas, labirintos, planícies, etc, enquanto espaços de jogos abstratos são representações abstratas de espaço, como o campo de jogo em Xadrez. Existem três tipos principais para esse tipo de conteúdo de jogo:

Mapas Internos:

São uma representação da estrutura e posicionamento relativo de espaços internos, particionados em salas. Salas podem ser conectadas entre si por corredores, sobrepostas em camadas, conectadas por escadas ou agrupadas em calabouços. Podem também ser representações internas de prédios ou cavernas. Pode ser um *bit* abstrato ou concreto.

Mapas Externos:

São representações da elevação e estrutura de um terreno externo. Pode ser um *bit* abstrato ou concreto.

Corpos d'Água:

Rios, lagos, e mares são usados comumente como obstáculos nos mapas ou como ambientes interativos. É um *bit* concreto.

2.3.3 Sistemas de Jogo

Os jogos utilizam comumente teoria e modelagem de sistemas complexos para gerar ou simular partes de seu sistema. Sistemas de jogos podem ser abstratos ou concretos. Um exemplo de sistema abstrato seria um sistema que define as relações entre vegetações e características de um terreno. Um exemplo para um sistema concreto seria um que simule cidades e redes de cidades.

Ecossistemas:

Governam o posicionamento, evolução e interação da fauna e flora com algoritmos e regras. Podem ser abstratos ou concretos.

Redes de Estradas:

Formam a estrutura básica de um mapa externo, podendo servir para vários propósitos como transporte entre pontos de interesse, estruturação e transporte dentro de cidades. Podem ser abstratos ou concretos.

Ambientes Urbanos:

São grandes agrupamentos de prédios onde muitas pessoas vivem em conjunto e interagem com seus arredores. Cidades realísticas demoram séculos para crescer, evoluindo com a influência das pessoas que vivem nela. Algoritmos procedurais usam uma abordagem

diferente, gerando primeiramente redes de estradas, para dividir depois o terreno entre as estradas em lotes de prédios, e, por fim, gerar prédios nos lotes. Podem ser abstratos ou concretos.

Comportamento de Entidades:

Para manter a ilusão de que o mundo de jogo seja um ambiente real, é necessário criar muitos tipos de interações complexas de jogador-ambiente. Entidades como personagens não-jogáveis que interagem com o jogador são ferramentas poderosas para manter a ilusão de um mundo real. Gerar proceduralmente estes comportamentos baseados nas ações do jogador tem o potencial de criar experiências realísticas e imersivas. Podem ser abstratos ou concretos.

2.3.4 Cenários de Jogo

Cenários de jogo descrevem a maneira e a ordem em que os eventos do jogo se passam. Dois tipos de cenários podem ser distinguidos, abstrato e concreto. Cenários abstratos de jogo descrevem como outros objetos relacionam entre si. Cenários concretos são explicitamente mostrados no jogo, como parte da narrativa, por exemplo.

Quebra-cabeças:

Problemas nos quais o usuário pode achar uma solução baseado em conhecimento prévio ou por exploração sistemática do espaço de possíveis soluções inseridas no problema [9]. Para quebra-cabeças, o processo de achar a solução *é* o jogo, e portanto uma experiência recompensável. São *bits* abstratos.

Storyboards:

São usados como um apoio de *design* para o desenvolvedor ou jogador. *Storyboards* são apresentadas na forma de quadrinhos, com painéis sequenciais descrevendo eventos de cenas. Dependendo de como são produzidos e usados, *storyboards* podem ser um exemplo de conteúdo derivado. Podem ser abstratos ou concretos.

Estória:

A estória de um jogo é muitas vezes necessária para criar uma boa experiência de jogo. Ela mantêm o jogador motivado, apresenta uma base lógica para os eventos que decorrem no jogo e provê um objetivo para o jogador cumprir. Podem ser abstratos ou concretos.

Níveis:

O conceito de níveis é usado em quase todo jogo como um separador entre sequências de jogabilidade. Por exemplo, em jogos do gênero plataforma (*Super Mario Bros*, *Sonic*, etc), níveis consistem de espaços de jogo separados, onde em cada um desses espaços o jogador precisa mover da posição inicial até a final, passando por uma série de obstáculos para conseguir chegar ao próximo nível. Podem ser abstratos ou concretos.

2.3.5 Design de Jogos

O *design* de um jogo é composto de conteúdo como regras e metas, onde regras definem o que pode ser feito no jogo e metas são os objetivos que o jogador está tentando completar; um componente estético, como um arco dramático ou tema gráfico, são elementos também importantes em *design* [28].

Design do Sistema:

O *design* do sistema de um jogo implica "a criação de padrões matemáticos descrevendo o jogo e as regras do jogo" [3]. Um dos maiores desafios na geração do *design* de sistemas de jogos é garantir que as regras sejam balanceadas entre todos os jogadores. É um *bit* abstrato.

Design do Mundo:

O *design* do mundo de um jogo é "o *design* de um ambiente, estória ou tema" [35]. É um *bit* geralmente concreto.

2.3.6 Conteúdo Derivado

É o conteúdo que foi criado como um resultado secundário do mundo do jogo. Esse tipo de conteúdo pode aumentar o sentimento de imersão que o jogador tem do mundo do jogo.

Notícias e Transmissões:

Um jogo pode mostrar aos jogadores notícias baseadas nas ações feitas por eles, ou mudanças no mundo do jogo. É um *bit* concreto.

Tabelas de Classificação:

Tabelas de classificação de jogadores. Populares em vários gêneros de jogos. É um *bit* abstrato.

Neste trabalho, utilizaremos somente Bits de Jogo concretos, do tipo Espaço de Jogo, pois tratamos do conteúdo que modifica diretamente a experiência de jogo dos jogadores em mapas multijogador de C2, e diminuir a quantidade de variáveis a serem consideradas na geração de um mapa pelas nossas representações de mapas.

Portanto não iremos tratar de Bits de Jogo dos tipos Cenários de Jogo e Sistema de Jogo pois mapas multijogador do jogo *C2* não apresentam nenhum desses Bits de Jogo. É necessário notar que devemos tomar em conta os Bits de Jogo do tipo Design de Jogo, do jogo escolhido nesta monografia para modelar mapas que funcionem bem com o mesmo. Para isso, iremos descrever o jogo escolhido por completo na Seção 2.8, mas antes iremos descrever o que é conteúdo gerado proceduralmente e algumas das técnicas mais populares utilizadas para gerar tal conteúdo.

2.4 Conteúdo Gerado Proceduralmente

Conteúdo gerado proceduralmente, ou *procedurally generated content* (PGC), se refere à geração de conteúdo automaticamente pelo uso de algoritmos [41]. Este trabalho irá se concentrar em como estas técnicas são utilizadas na área de jogos eletrônicos.

PGC é utilizado em jogos para a criação de vários tipos de conteúdo de jogo para auxiliar na diminuição do tempo de desenvolvimento do jogo, diminuir a quantidade de memória gasta em DVDs ou economizar o tempo ou dinheiro gastos por *designers* humanos [41]. Um dos primeiros jogos que utilizou PGC foi *Rogue(1980)* para gerar aventuras completas [44]. Toda vez que o jogador decidisse criar uma nova partida, o mundo do jogo inteiro seria modificado aleatoriamente.

Algumas abordagens de PGC são mapas controlados por histórias [25], terreno [43] e jogos completos [10]. A seguir, descreveremos as características que um PGC pode ter.

2.4.1 Características de Algoritmos de PGC

Há varias distinções sobre características que um algoritmo de PGC pode ter: necessário ou opcional, *online* ou *offline*, sementes aleatórias ou vetores parametrizados, geração estocástica ou determinística, construtivo ou gerar-e-testar [42]. Estas características são importantes para classificar um algoritmo de PGC, ou para determinar se é vantajoso

ou não implementar PGC para algum jogo. Definiremos estas características a seguir, usando como referência [42].

Necessário versus Opcional

Precisamos distinguir se o conteúdo é necessário ou opcional. Conteúdo necessário é requerido pelos jogadores para progredir no jogo, enquanto conteúdo opcional são elementos que podem ser evitados pelo jogador.

Não se deve gerar um conteúdo necessário que não seja correto, ou seja, é inaceitável que o jogador receba um conteúdo que não possa ser completado, como um labirinto sem saída, ou um inimigo que não pode ser derrotado. Por outro lado, não há problema na criação de conteúdo opcional inútil, como armas fracas ou mapas inúteis se estes podem ser ignorados pelo jogador.

Em C2, os conteúdos considerados desnecessários são meramente visuais, todo item ou arma inserido em um mapa multijogador é necessário, e cada área no mapa existe para um propósito específico e útil de alguma maneira para o jogador.

Online versus *Offline*

Quando a geração de conteúdo pode ser realizada durante a execução do jogo ou programa, falamos que ela ocorre *online*. No caso da geração de conteúdo ter sido feita no desenvolvimento do programa, dizemos que ela ocorreu *offline*. Um exemplo de geração de conteúdo online pode ser observado em *Minecraft(2009)*, na geração de um novo mundo, enquanto um exemplo de geração *offline* seria um mapa que foi gerado proceduralmente, e depois aperfeiçoado por um artista antes de ser inserido no jogo. Neste trabalho, o algoritmo de geração de mapas funcionará somente *offline*.

Sementes Aleatórias versus Vetores Parametrizados

Outra distinção sobre o algoritmo de geração em si é o quanto pode ser parametrizado dentro do próprio algoritmo. Em um extremo, algoritmos podem simplesmente receber uma semente aleatória como entrada; em outro extremo, algoritmos podem receber um vetor multidimensional de parâmetros, com valores reais que especificam as propriedades do conteúdo gerado.

Geração Estocástica versus Geração Determinística

A quantidade de variação entre dois resultados do mesmo algoritmo com os mesmos parâmetros é outra característica que deve ser definida. Em um lado os resultados podem

ser idênticos (determinístico), e em outro lado diferentes (estocástico). Algoritmos completamente determinísticos servem principalmente para compressão de dados.

Um exemplo disso seria a *demoscene* [6], um grupo de artistas que tentam criar apresentações com vídeo e música de alta qualidade, mas que ocupam espaços menores que 256KBytes.

Construtivo versus Gerar-e-Testar

Como distinção final, temos algoritmos que geram o conteúdo somente uma vez, pois utilizam técnicas que garantem que o conteúdo gerado é "bom o bastante", e algoritmos que são compostos por duas partes: uma parte prepara o conteúdo e a outra o testa. Caso o conteúdo não passe pela fase de teste, ele é gerado novamente até que seja criado um conteúdo que passe pelos testes.

Para implementar um algoritmo PGC em um jogo, é necessário primeiramente escolher qual será o conteúdo criado; após esta escolha, precisamos determinar quais partes do jogo são afetadas ou afetam na geração deste conteúdo, para finalmente definir as regras de geração do conteúdo escolhido. Portanto, precisamos especificar os aspectos de C2 que afetam na geração de um mapa. Estes aspectos serão descritos nas seções 2.8 e 2.10.

2.5 Geração de Terrenos Procedurais

PTG, ou geração de terrenos procedurais, é uma área dentro de PGC, onde o conteúdo criado pelos algoritmos de geração de conteúdo são mapas ou terrenos. Esta técnica é utilizada com sucesso na indústria e em pesquisas acadêmicas há mais de três décadas [39].

2.6 Search-Based Procedural Content Generation

Search-Based Procedural Content Generation (SBPCG) é considerado um caso especial de algoritmos gerar-e-testar, mas ele difere em certos aspectos. A função de teste não aceita ou rejeita o conteúdo candidato, ela cria um valor ou um vetor de valores reais, que contém a nota ou notas atribuídas aos conteúdos gerados. Esse tipo de função de teste pode ser chamada de função objetivo. Outra característica em que SBPCG difere de um algoritmo de gerar-e-testar é na geração do conteúdo candidato, que é dependente das notas atribuídas às instâncias de conteúdos que foram testados anteriormente. Isto é feito com o objetivo de produzir novo conteúdo que tenha notas com valores maiores [42].

O uso de SBPCG em PTG é recente, poucas pesquisas foram feitas nessa área. A primeira proposta foi feita por [30], e desde então, poucas abordagens foram feitas [32].

Nesta monografia, utilizaremos Algoritmos Genéticos (GA) aplicados aos conceitos acima. Este uso será explicado com maiores detalhes na Seção 3.3.

2.7 Algoritmos Genéticos

Nesta seção, descreveremos as características e a funcionalidade de Algoritmos Genéticos (GA). GAs foram formalizados nos meados de 1960 por John Holland para estudar formalmente o fenômeno de adaptação como ela ocorre na natureza e para descobrir maneiras de implementar estes mecanismos em sistemas computacionais [26].

Este tipo de algoritmo tenta achar candidatos de soluções de um problema dentro de um espaço de busca utilizando métodos e representações que se baseiam nas ideias da evolução natural de seres vivos [26].

Não existe uma definição rigorosa de GA que os diferencia de outros métodos evolucionários de computação, mas a maioria dos GAs contém as seguintes características em comum [26]:

- Populações de cromossomos;
- Seleção de acordo com a função objetivo;
- Recombinação para produzir novos candidatos;
- Mutação aleatória de novos candidatos;

Cromossomos, ou genomas, são indivíduos que serão gerados para serem selecionados pelo GA. As representações destes cromossomos são de grande importância para a solução do problema desejado, pois influencia no desempenho do GA. Uma representação tem um fenótipo e um genótipo; o genótipo é tipicamente composto por cadeias de alelos, estes alelos podem ser números ou cadeias de números que representam características do indivíduo, estas cadeias de números ou números são compostos por *bits*, com possíveis valores iguais a 0 ou 1; um fenótipo é a decodificação do genótipo, utilizado como a solução do problema [26]. Descreveremos as representações, os genótipos e fenótipos dos nosso algoritmo de geração de mapas em detalhes na Seção 3.4.

Populações destes cromossomos são gerados pelo GA, e depois notas são atribuídas à cada um deles. Para decidir o valor destas notas, é necessário ter uma função, chamada função objetivo, que deve atribuir notas aos cromossomos dependendo de quão bem eles resolvem o problema.

Um GA pode aplicar os seguintes operadores sobre seus candidatos:

- Seleção: este operador seleciona cromossomos na população para reprodução. Existem múltiplos tipos de seleção, o mais simples é a seleção roleta russa (indivíduos

recebem uma probabilidade de serem escolhidos proporcional ao valor da função objetivo deles). Quanto melhor a nota da função objetivo do cromossomo, maior a probabilidade dele ser escolhido múltiplas vezes.

- Recombinação: em sua forma mais simples, este operador escolhe um ponto na cadeia de dois cromossomos e troca as subsequências antes e depois do ponto para criar dois novos candidatos. Existem outras versões deste operador onde existem múltiplos pontos na cadeia
- Mutação: este operador altera aleatoriamente o valor de alguns *bits* em um cromossomo de acordo com uma probabilidade escolhida.

O fluxo de um GA simples pode ser visto na Figura 2.2.

O GA criado para este trabalho será descrito em detalhes na Seção 3.3.

A seguir, descreveremos o jogo C2 em detalhes para demonstrar como construímos as representações de mapas e como as mesmas serão escritas no formato de um genoma.

2.7.1 Testes Estatísticos Utilizados

Para analisar nossos dados obtidos com os experimentos, utilizamos os seguintes testes estatísticos: o teste t de Student, o *Generalized Extreme Studentized Deviate Test (GESDT)* e o teste de normalidade Shapiro-Wilk.

Utilizamos o teste t de Student por duas razões, primeiramente este foi o teste utilizado em [5] e para fazer comparações justas utilizaremos o mesmo teste estatístico; em segundo lugar o teste t de Student é um teste simples utilizado para determinar se dois grupos de amostras são estatisticamente diferentes um do outro.

Para podermos utilizar o teste t de Student, nossas amostras precisam estar de acordo com a distribuição normal, portanto para saber se o nossos conjuntos de amostras estão normalizadas usamos o teste de normalidade Shapiro-Wilk, que é recomendado para amostras com valores únicos [38]. Caso nossas amostras não estejam de acordo com a distribuição normal, utilizaremos o GESDT para remover valores aberrantes, ou seja, valores inconsistentes com os demais da amostra [15], para tentar aproximar as amostras da distribuição normal.

A seguir, descrevemos em detalhe os testes utilizados.

Generalized Extreme Studentized Deviate Test (GESDT)

Este teste é uma generalização do teste de Grubbs, que é utilizado para identificar pelo menos um valor aberrante no conjunto de dados. O GESDT trata mais de um valor aberrante.

Testamos o conjunto de dados S com n elementos gerando k estatísticas de teste $G_1, G_2, ..., G_k$ onde cada G_i é um teste bicaudal de Grubbs, definido pelas Equações 2.1 e 2.2

$$S_1 = S \tag{2.1}$$

onde $\bar{x_j}$ é a média de S_j e s_j é o desvio padrão de S_j

$$G_j = \frac{max\left\{|x - \bar{x_j}| : x \,\epsilon\, S_j\right\}}{s_j} \tag{2.2}$$

$S_{j+1} = S_j - \{x_j\}$ onde x_j é igual ao elemento em S_j tal que $|x_j - \bar{x}|$ é maximizado.

Em outras palavras, rodamos k testes de Grubbs separados, testando se $G_j > G_j - crit$ onde $G_j - crit$ é G_{crit} como descrito acima, mas ajustado pelo valor correto do tamanho da amostra, ou seja, n é trocado por $n - j + 1$. Assumindo que r é o maior valor de $j \leq k$ tal que $G_j > G_j - crit$concluímos então que existem r valores aberrantes, $x_1, ..., x_r$. [46, 34]

Teste de normalidade Shapiro-Wilk

O teste de normalidade Shapiro-Wilk é utilizado para saber se um conjunto de amostras está de acordo com a distribuição normal (hipótese nula). Ele é definido pela Equação 2.3.

$$W = \frac{(\sum_{i=1}^{n} a_i x_{(i)})^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{2.3}$$

onde x_i são os valores das amostras ordenados aleatóriamente; $\bar{x}$ é a média da amostra; a_i são as constantes geradas das covariâncias, variâncias e médias das amostras (de tamanho n).

Este teste é recomendado para amostras com valores únicos [38].

Teste t de Student

O teste t de Student é utilizado para determinar se dois grupos de amostras são estatisticamente diferentes uma da outra, com a hipótese nula de que ambos grupos tem médias iguais. A estatística t é calculada pelas Equações 2.4 e 2.5.

$$t = \frac{\bar{X_1} - \bar{X_2}}{s_{X_1X_2} \cdot \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}} \tag{2.4}$$

onde

$$s_{X_1X_2} = \sqrt{\frac{(n_1 - 1)s_{X_1}^2 + (n_2 - 1)s_{X_2}^2}{n_1 + n_2 - 2}} \tag{2.5}$$

$s_{X_1X_2}$ é um estimador do desvio padrão dos dois grupos de amostras [21].

2.8 Cube 2: Sauerbraten

C2 [27] é um jogo do gênero FPS e do subgênero *Arena Shooter*. Os autores de [5] nos disponibilizaram o código fonte das alterações feitas no motor do jogo para injeção de mapas. Estas são as razões descritas pela escolha do jogo C2 em sua pesquisa:

> "Since Cube 2 is open source, the map editor itself can be modified and extended in whatever way necessary. This feature is crucial for our purposes, as we need to inject evolved maps back into the game, and it is one of the main reasons we chose to use Cube 2 rather than a better-known commercial FPS game. The other main reason is that the game engine allows us to run the game in "headless" mode, i.e. without visualization, where it can be speed up to run as fast as the processor permits."[5]

C2 é um jogo voltado principalmente para partidas competitivas entre jogadores, mas também contém um modo de somente um jogador. Ele funciona como a grande maioria dos FPSes, o jogador controla somente um *avatar*, a visão do jogador é de primeira pessoa. O *avatar* pode pular, agachar, andar, correr e atirar. Estas simples características serão de grande importância na geração de regras para construção de mapa mais a frente.

O objetivo mais básico em C2 é matar o *avatar* do seu oponente para ganhar um ponto. O jogador com o maior número de pontos no final de uma partida (que acontece quando um tempo limite ou um número limite de pontos é alcançado) é o vencedor. C2 tem 22 modos de jogo multijogador, cada um desses modos necessita de um mapa especializado e um número recomendado de jogadores para funcionar corretamente. Em [5] decidiu-se usar o tipo de jogo *Free-for-All* com 4 jogadores. Neste modo de jogo, também conhecido como *deathmatch* clássico, um grupo de jogadores lutam entre si sem nenhum tipo de aliança, com um limite de pontos e/ou de tempo.

Em FPSes do subgênero *Arena Shooter*, tradicionalmente é necessário coletar armas no mapa antes de poder usá-las, mas em C2 o jogador começa com todas as armas já disponíveis, porém sem munição. Há vários tipos de munição e itens disponíveis para os jogadores, estes só podem ser coletados e usados se estão disponíveis no mapa escolhido para a partida. Estas munições, itens e armas são descritos em detalhe, de acordo com o manual do jogo C2 [12] nas Tabelas 2.1 e 2.2.

Armas podem ser *hitscan* (termo usado para descrever armas que não tem um projétil real, o tiro acerta o alvo instantaneamente) como a *Chaingun*, ou podem ter projéteis

como a *Rocket Launcher*. Armas que usam projéteis são as mais utilizadas para impedir o movimento de outros *avatares*. Um exemplo seria bloquear duas entradas distintas para uma área com a *Grenade Launcher*, atirando uma granada em cada entrada. O tempo de explosão das granadas é grande o bastante para lançar outras duas granadas nas mesmas posições. Isso faz com que as duas entradas se tornem caminhos perigosos para outros jogadores, que precisariam escolher entre a possibilidade de ser atingidos pelas explosões ou achar uma rota alternativa, que pode não existir. Por esta razão, é preferível que áreas de combate nos mapas tenham pelo menos três passagens de entrada ou saída, para que não seja possível que um jogador consiga bloquear todos os caminhos para uma área de combate.

O posicionamento de armas e itens é de extrema importância em C2; uma *Rocket Launcher* posicionada em uma área fechada, onde o jogador que a coletou mal consegue se movimentar faz com que a arma seja inútil, pois o jogador tem muito mais chances de se matar com a própria arma. Outro exemplo seria posicionar uma *Shotgun*, em uma área grande e aberta. Em ambos os casos, o mal posicionamento das armas pode fazer com que jogadores as ignorem, ou pior, a próprias áreas que contém a arma sejam ignoradas, tornando-as inúteis. No caso de itens, por exemplo, o *Heavy armour* não pode ser posicionado em um lugar sem risco, pois o jogador que tenta coletá-lo não terá nenhuma razão para fazer o contrário, já que não teme armadilhas de outros jogadores.

Também é importante notar que quando um jogador morre, o jogo escolhe aleatoriamente uma entre várias posições especificadas no mapa para reviver o jogador. Estas posições são chamadas *"spawn points"* ou locais de nascimento e seu posicionamento é crucial no *design* de um mapa.

Existem dois objetos que podem ser utilizados em mapas do jogo chamados de *Jump Pads* e *Teleports*. *Jump pads* são superfícies que lançam o jogador em uma direção predeterminada pelo *designer* do mapa quando o *avatar* do jogador encosta nelas. *Teleports* são objetos que teletransportam o *avatar* do jogador instantaneamente para outra área especificada previamente pelo *designer* do mapa quando o *avatar* encosta no *teleport*.

Estas características da jogabilidade do jogo precisam ser consideradas cuidadosamente para garantir a qualidade do mapa que será gerado. As estratégias de *design* que foram utilizadas neste trabalho para o desenvolvimento das representação de mapas serão descritas na Seção 2.10.

2.9 *Bots* do jogo C2

Bots são NPCs, personagens que são controlados pelo próprio jogo, e não pelos jogadores. Existe pouca documentação sobre os *bots* do jogo C2, e o código fonte dos mesmos

Tabela 2.1: Armas disponíveis em C2 e suas características.

Arma	**Descrição**	**Dano**	**Dano por segundo**	**Recarga**	**Dano em área**	**Projétil**
Fist	Arma de corpo-a-corpo, não utiliza munição. Seu alcance é mínimo, mas seu dano é alto. O jogador começa com esta arma, portanto ela está sempre à disposição.	50	200	0,25s	Não	*Hitscan*
Chaingun	Metralhadora de alcance médio-longo, pouco dano e alta taxa de tiro. O jogador começa com esta arma, portanto ela está sempre à disposição.	30	300	0,1s	Não	*Hitscan*
Shotgun	Espingarda de curto alcance. Atira várias balas que dão pouco dano sozinhas, mas quando todas acertam, causam alto dano.	20x10	143	1,43s	Sim	*Hitscan*
Rifle	Arma de longo alcance, alto dano. Requer alta precisão do jogador para ser usada efetivamente, mas seu alcance é infinito.	100	66	1,5s	Não	*Hitscan*
Pistol	Arma de longo alcance, baixo dano. O jogador começa com esta arma, portanto ela está sempre à disposição.	35	70	0,5s	Não	*Hitscan*
Grenade Launcher	Arma de curto alcance, dano alto. Um lançador de granadas que quicam até explodirem. O próprio jogador que atirou pode ser ferido por suas granadas.	90	150	0,6s	Sim	Projétil
Rocket Launcher	Arma de curto-longo alcance, dano alto. Um lançador de mísseis que explodem em contato. O próprio jogador que atirou pode ser ferido por seus misseis.	120	150	0,8s	Sim	Projétil

Tabela 2.2: Itens disponíveis em C2 e suas características.

Item	**Efeito**	**Tempo de *respawn***
Light Armor	Quando coletado, dá 100 pontos de armadura para o jogador e absorve 1/2 do dano causado à vida do jogador.	20 segundos
Heavy Armor	Quando coletado, dá 200 pontos de armadura para o jogador e absorve 3/4 do dano causado à vida do jogador.	30 segundos
Quad Powerup	Quando coletado, multiplica o dano do jogador por 4 vezes por 20 segundos.	70 segundos
Health	Recupera 25 pontos de vida	20 segundos
Health Boost	Aumenta o limite de vida do jogador por 10 pontos até o fim da partida	60 segundos
Munição *Shotgun*	Munição para *Shotgun*, 10 balas.	12 segundos
Munição *Chaingun*	Munição para *Chaingun*, 20 balas.	12 segundos
Munição *Rocket Launcher*	Munição para *Rocket Launcher*, 5 balas.	12 segundos
Munição *Rifle*	Munição para *Rifle*, 5 balas.	12 segundos
Munição *Grenade Launcher*	Munição para *sGrenade Launcher*, 10 balas.	12 segundos
Munição *Pistol*	Munição para *Pistol*, 30 balas.	12 segundos

não é de fácil compreensão. De acordo com [5], os *bots* do jogo C2 utilizam *waypoints*, pontos específicos no mapa que mostram aonde é possível se locomover. Estes pontos são colocados nos mapas gerados das representações de mapas pelo mesmo algoritmo que faz a injeção dos mesmos em C2. Para nossos testes, utilizaremos quatro *bots* para simular uma partida entre quatro jogadores, pois utilizar humanos para realizar nossos testes consumiria muito tempo e em [5] foi feito o mesmo.

2.10 Regras de Geração de Mapas

Um mapa de um FPS existe para um propósito claro: servir como um limitador dos movimentos que podem ser feito pelo *avatar* controlado pelo jogador [36]. Portanto, um mapa precisa ser gerado baseando-se nas capacidades de movimento e as medidas do *avatar* que o jogador controla. Criar áreas pequenas demais para o *avatar* entrar ou locais que não podem ser acessados são um desperdício de recursos, portanto é necessário saber como o *avatar* se movimenta, até qual altura ele pode pular, qual o menor espaço que ele pode ocupar e como os objetos do jogo interagem com ele. Estas decisões não garantem que o mapa criado terá uma qualidade boa, mas garantem que o mapa condiz com a jogabilidade do jogo [1].

É difícil definir o que faz um mapa ser "bom", e um estudo mais aprofundado seria necessário para conseguir distinguir quais características são importantes para assegurar a qualidade de um mapa. A geração de um mapa jogável é trivial, pois podemos, por exemplo, criar uma área quadrada oca com um local de nascimento. Porém, este mapa não é interessante visualmente, não requer que o jogador memorize sua estrutura ou posicionamento de itens e armas, não contém nenhum tipo de escolha que envolve um fator de risco/recompensa. [5] assume o seguinte sobre isto:

> "We assume that the promise of a map is directly linked to the fighting time of the player, which is defined as the duration from the moment in which the player starts to fight an opponent to the moment in which the player is killed."[5]

Esta suposição é utilizada para criar a métrica, a função objetivo, utilizada pelo algoritmo genético em [5].

> "Since during an entire match the player will die and spawn multiple times, we can compute the average fighting time value for the game per player, T_f . [...] we simulated matches of 10 minutes among 4 bots and we measured the average T_f value across all bots, $\overline{T_f}$, yielding a simulation-based fitness function according to the same classification.
>
> The complete fitness function has another component in addition to the T_f value: the free space of the map, S. We explicitly want our fitness to promote the generation of larger maps since very small maps do not leave enough space for the

placement of weapons and spawn-points, leading to unrealistic values of T_f. [...] Given the above, the complete fitness function of a map is as follows:

$$f = \overline{T_f} + S$$

where $\overline{T_f}$ is measured in milliseconds and it is an integer greater than 0, and S represents the number of free cells in the map and is bounded between 0 and 4096."[5]

Neste trabalho, usamos a mesma métrica descrita acima, que tenta quantificar a qualidade de um mapa a partir do tempo médio de vida de um jogador e do número de espaços livres do mapa. Esta métrica favorece mapas grandes e com rotas ou obstáculos o bastante para que os jogadores consigam ter lutas longas.

É interessante notar que o fator estético dos mapas não é considerado, pois traria uma complexidade maior à este trabalho, precisamos somente que o jogador entenda as possíveis escolhas e riscos contidos no mapa.

De acordo com [36], mapas do estilo *deathmatch* podem ser categorizados em cinco estilos populares, estes são:

- Arena: são mapas que contém uma área central onde a maior parte do combate ocorre, a maioria dos corredores e passagens se ligam a esta área. O mapa tem poucas ou nenhuma outra área que seja tão significante quanto a área central;
- Circular: são mapas com um *design* circular, criado de uma maneira que o jogador não precise parar e voltar no caminho principal do mapa para conseguir completar uma volta completa no mesmo. Deve conter poucos ou nenhum corredor sem saída;
- Linear: são mapas criados com poucos caminhos alternativos, áreas abertas ou corredores largos onde jogadores possam desfrutar combates diretos;
- Baseado em Localizações: este tipo de mapa deve ter como característica principal a facilidade do jogador saber onde ele está. O mapa deve ser repleto de áreas únicas.
- Centralizado em Temas: mapas que são criados com um único tema em mente, um exemplo de um tema poderia ser baixa gravidade ou um mapa embaixo d'água;
- Capturar a Bandeira: estes mapas são feitos com o modo de jogo "capturar a bandeira" em mente, onde dois times tentam roubar a bandeira do outro. O mapa deve conter duas áreas especiais onde a bandeira de cada time está localizada. Este estilo de mapa pode ter duas categorias:
  - Níveis Simétricos: estes níveis contém uma base para cada times. As bases são idênticas e se encontram em uma área aberta no centro do mapa;

- Níveis Assimétricos: estes níveis tentam tem bases assimétricas, e tentam equilibrar a partida dando vantagens (itens ou armas melhores) para o time com a base inferior.

Dentre estes cinco estilos, escolhemos o estilo Circular. No próximo capítulo, iremos descrever as quatro representações de mapas que desenvolvemos e as três das quatro representações descritas em [5] que reimplementamos neste trabalho.

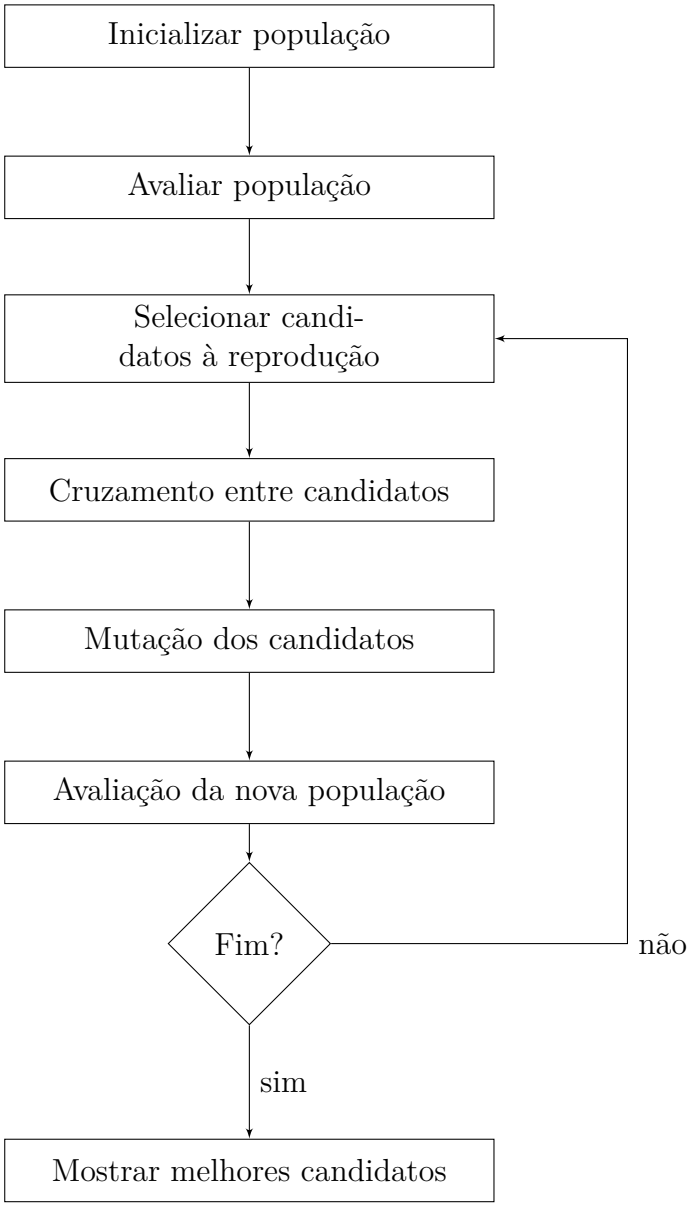

Figura 2.2: Fluxo de um GA simples.

Capítulo 3

Implementação

Neste capítulo, serão descritos os componentes deste trabalho em detalhes, as modificações feitas por [5] no jogo C2, o algoritmo genético, o gerador de mapas e as representações de mapas.

3.1 Visão Geral

Para alcançar os objetivos deste trabalho, foi necessário implementar um algoritmo de seleção genética, reimplementar três das quatro representações de mapas propostas em [5], testar estas representações no algoritmo de seleção genética e comparar os resultados e implementar quatro representações de mapas novas. Apelidamos estas representações de Salas Cíclicas 1 (SC1), Salas Cíclicas 2 (SC2), Salas Cíclicas 3 (SC3) e Pontos Cíclicos (PC). Descreveremos elas e as representações propostas por [5] em detalhes na Seção 3.4.

3.1.1 Componentes

Este projeto é dividido em 5 componentes principais:

- O jogo C2
- O Algoritmo de Seleção Genética e Geração de Mapas (ASGGM)
- Os algoritmos contidos no gerador de mapas
- A representações dos mapas gerados pelo gerador de mapas
- Um *script bash* que coordena os anteriores

Destes componentes, os seguintes foram desenvolvidos e utilizados pelos autores de [5]: o jogo C2 com seu código fonte modificado (Seção 3.2) e 3 descrições de algoritmos de

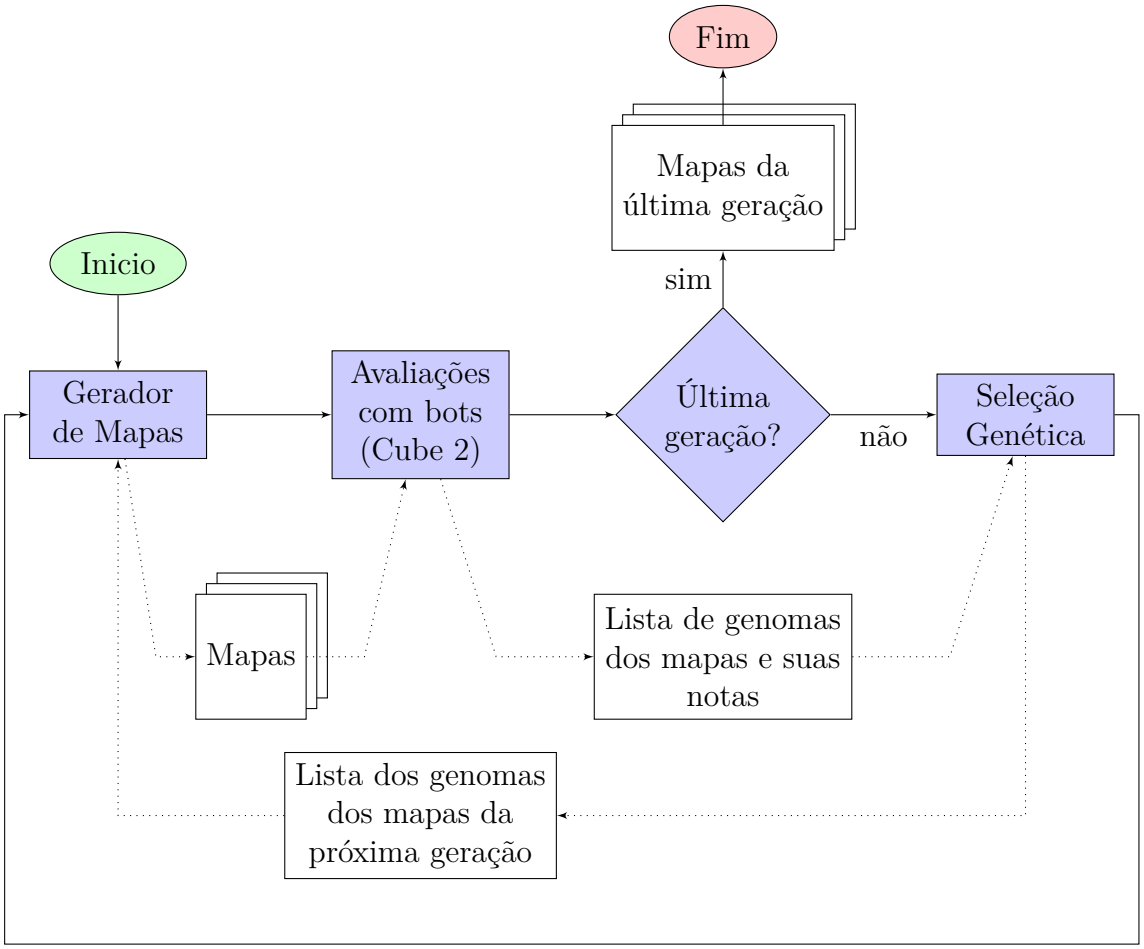

Figura 3.1: O fluxo de dados e dos processos envolvidos na seleção genética e na geração de mapas.

geração de mapas (Seção 3.4). Os autores de [5] nos forneceram o código fonte modificado de C2; reimplementamos os algoritmos de geração de mapas a partir das descrições fornecidas em [5].

A seguir, explicaremos detalhadamente cada um destes componentes e suas funções. Uma visão geral de como o fluxo de dados e dos processos envolvidos na seleção genética e na geração de mapas pode ser visto na Figura 3.1.

3.2 Modificações feitas em C2

Como dito anteriormente, a versão do jogo C2 utilizada neste trabalho teve seu código fonte modificado por [5]. As três principais mudanças foram:

- O jogo pode ser inicializado em dois modos: no primeiro, o jogo funciona normalmente e duas novas opções foram adicionadas ao menu principal do jogo: *"Load Generated Map"*, que carrega a representação de um mapa no formato ".txt" com

nome “map.txt” localizado na pasta principal do jogo e deixa o usuário se movimentar sozinho nele; e *“Play Generated Map”*, que funciona como *“Load Generated Map”* mas o jogador irá jogar contra 4 *bots* com nível de habilidade 50;

- No segundo modo, toda renderização do jogo, sons e *input* do jogador são desabilitados e o limite do tempo *delta* do jogo (que limita a velocidade em que jogo executa) é alterado para que o jogo possa funcionar na maior velocidade possível. Este modo é ativado quando se inicia o jogo com o parâmetro “-T”, e é usado para todos os testes deste trabalho.

  Quando esta segunda forma de inicialização é escolhida, o jogo faz as seguintes tarefas com os dados do arquivo “map.txt” (a grande maioria destas tarefas foram implementadas nos arquivos “generatedmap.h” e “generatedmap3d.h” no código fonte do C2):

  - Um mapa de dimensões 128 por 128 células é criado;
  - Uma área de 64 por 64 células no centro deste mapa é levantada em uma unidade para cima;
  - Todos os espaços que não foram levantados em uma unidade para cima se tornam áreas que matam um personagem em contato;
  - O arquivo “map.txt” é lido e todos os dados contidos nele são copiados para o mapa do jogo;
  - *Waypoints*, pontos de navegação que se conectam como grafos cíclicos utilizados pelos *bots* para andar pelo mapa, são criados a partir do mapa do jogo para possibilitar o movimento dos mesmos pelo mapa;
  - Uma partida entre 4 *bots* com nível de habilidade 50 é iniciada;
  - São impressos pelo jogo em seu andamento: o número de células livres que o mapa contém, o tempo que levou para um *bot* sofrer dano (*TimeToDamage*) e o tempo que levou para um *bot* morrer (*TimeToFight*).

- Por fim, é possível passar como argumento um número que será usado como a semente do gerador de números aleatórios do jogo. Desta maneira, as ações feitas pelos *bots* em um mapa específico sempre serão as mesmas se utilizarmos o mesmo valor da semente, isso é feito para garantir que os mapas simulados não produzam valores da função objetivo diferentes (caso o mesmo mapa seja testado novamente na próxima geração) no decorrer dos testes genéticos. Neste trabalho escolhemos arbitrariamente o valor da semente como 8.

3.3 Algoritmo genético

O algoritmo genético é baseado nos parâmetros e métricas usados por [5] descritos anteriormente na Seção 2.10. Nesta seção iremos descrever em detalhes os parâmetros e fases do algoritmo de seleção usado neste trabalho.

[5] diz o seguinte sobre seu algoritmo genético:

> "To evolve the maps, we applied a simple genetic algorithm with standard operators: tournament selection with tournament size 2, single point crossover, mutation rate of $\frac{1}{n}$ (where n is the length of the genome), mutation range 0.2 (following a uniform distribution) and a crossover probability of 0.2. The parameters of the genetic algorithm were set empirically. We applied the genetic algorithm with a population size of 50 individuals and let it run for 50 generations."[5]

Portanto, utilizamos seleção do tipo torneio. Após gerar a primeira população aleatória de indivíduos de um tipo de representação e testar seu valor da função objetivo, tiramos dois indivíduos aleatórios da população corrente e vemos se o mais forte ou o mais fraco vence (0,2 de probabilidade do mais fraco vencer, para não perdermos a variedade genética da população) e colocamos o vencedor no grupo de cruzamento. Repetimos isso até que tenhamos o mesmo número de indivíduos da primeira população (no caso deste trabalho, 50).

Após isso, escolhemos dois indivíduos aleatórios do grupo de cruzamento e verificamos se ocorre cruzamento (0,2 probabilidade). Se houve cruzamento, escolhemos um ponto entre 0 e n (onde n é o comprimento do genoma), aplicamos o operador de cruzamento e produzimos dois novos indivíduos a partir desse novo cruzamento. Estes dois novos indivíduos têm uma chance de $\frac{1}{n}$ de sofrer mutação. Após isso, os inserimos na população da próxima geração. Se não houve cruzamento, os dois indivíduos são inseridos na população da próxima geração sem sofrer mutações.

Esse procedimento é repetido até que a 50ª geração tenha 50 indivíduos. Escolhemos o indivíduo com o melhor valor da função objetivo como o melhor mapa e terminamos o teste.

3.4 Gerador de mapas

O gerador de mapas tem duas funções: traduzir o genótipo para o fenótipo, ou seja, traduzir o genoma para um mapa (que será lido pelo jogo C2); e gerar a população inicial de mapas que serão utilizados nos testes genéticos. Ele contém a implementação das sete representações mapas que serão descritas na próxima seção. Destas sete representações, três são reimplementações das representações **Grid**, **All-White** e **All-Black** descritos por [5] e as quatro outras restantes são as representações propostas neste trabalho, SC1,

SC2, SC3 e PC. A representação de mapa **Random-Digger**, também descrito em [5], não foi reimplementada pois sua descrição era ambígua.

Como foi descrito na Seção 2.10, as características que precisam ser consideradas para produzir um mapa bom são inúmeras e podem ser subjetivas em grande parte. Isso levou [5] a diminuir o número de variáveis a serem consideradas na geração com as seguintes decisões:

- O mapa tem um tamanho fixo de 64 por 64 células em grade;
- Ele pode ser inciado com todas as células preenchidas por paredes ou não. Quando ele é iniciado com todas as células preenchidas por paredes, chamaremos seu estado inicial de **cheio**, caso contrário, chamaremos seu estado inicial de **vazio**;
- O posicionamento dos itens e locais de nascimento é fixo e igual para todos os mapas gerados;
- Caso o mapa gerado seja dividido em várias áreas desconexas, escolhemos a área que contém o ponto central do mapa e apagamos as restantes.

A Figura 3.2 mostra um mapa sem alterações com estado inicial vazio com itens e locais de nascimento.

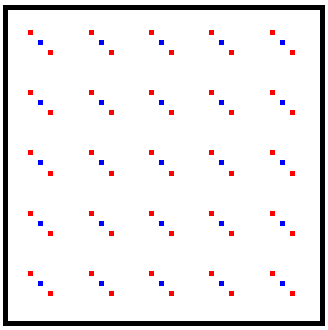

Figura 3.2: O mapa inicial, sem alterações. Pontos azuis são locais de nascimento e pontos vermelhos são munições.

Estas decisões nos ajudam com os seguintes problemas respectivamente: mapas grandes demais que podem aumentar o valor da função objetivo de uma maneira que o tempo médio de luta não seja a variável com maior peso no calculo da mesma; facilidade na tradução de genótipo para fenótipo, pois algumas das representações funcionam inserindo paredes e outras retirando paredes; complexidade da representações para não tratar dos posicionamento dos itens e locais de nascimento; e por fim, limpar áreas que não podem ser alcançadas pelos *bots*. Estas escolhas facilitam o trabalho feito pelas representações, e nos deixam focar somente no posicionamento de paredes no campo de jogo, simplificando a complexidade das representações de mapas substancialmente.

3.4.1 Formato dos mapas gerados

Os mapas gerados seguem um simples padrão de símbolos em um arquivo de texto padrão *UTF-8*, onde cada caractere representa um espaço na matriz do mapa dentro do jogo. Os caracteres neste arquivo que são reconhecidos pelo jogo são os seguintes:

- "1" - parede
- "," - vazio
- "." - chão
- "2" - *Green Armour*
- "3" - *Health*
- "4" - munição *Shotgun*
- "5" - munição *Chaingun*
- "6" - munição *Rocket Launcher*
- "7" - munição *Rifle*
- "8" - munição *Pistol*
- "9" - munição *Grenade Launcher*
- "s" - local de nascimento

3.5 Representações de mapas

Nesta seção, descreveremos em detalhes cada uma das representações utilizadas neste trabalho. Discutiremos os mapas evoluídos destas representações e as características deles na Seção 4.1.2.

3.5.1 Grid

Esta representação começa no estado vazio e divide o espaço do mapa em uma matriz de 9 por 9 espaços, totalizando 81 espaços ou células. Cada célula desta matriz 9 por 9 contém um valor de 0 à 3, que indica onde deve ser colocado uma parede dentro desta célula. O valor 0 indica que não há paredes nesta célula; 1, indica que haverá uma parede ao norte da célula; 2, ao leste; 3 ao norte e leste ao mesmo tempo.

O mapa começa vazio e o genoma desta representação tem tamanho $n = 81$, onde $n = t * N_w$ com t sendo o tamanho do alelo (neste caso $t = 1$) e N_w é a quantidade de alelos

em que o genoma é dividido (neste caso $N_w = 81$). Alguns mapas dessa representação podem ser vistos na Figura 3.4.

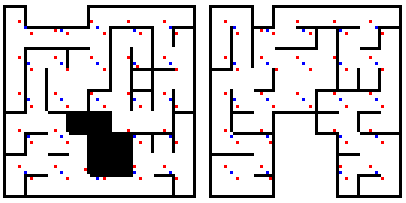

Figura 3.3: Alguns mapas Grid. Pontos azuis são locais de nascimento e pontos vermelhos são munições.

3.5.2 All-White

O mapa nesta representação começa no estado vazio. São adicionados segmentos de paredes horizontais ou verticais em pontos dentro do espaço de 64 por 64. A quantidade de segmentos de paredes utilizada em [5] é de 30, e usaremos este mesmo valor neste trabalho. Os alelos do genoma são da seguinte forma: $< x, y, l >$, onde x e y definem a posição do canto noroeste da parede e l corresponde ao tamanho da parede. Se l for positivo, a parede será horizontal, se negativo, vertical. A grossura das paredes é de 1 unidade, l tem um valor entre -64 e 64, x e y tem um valor entre 0 e 64.

O genoma desta representação tem tamanho $n = 90$, pois $t = 3$ e $N_w = 30$.

De acordo com [5] os *bots* de C2 ocupam somente uma célula de espaço, então a passagem por "portas" com o tamanho de uma célula seria possível. Nossos testes mostraram que os *bots* na verdade ocupam uma área de 2 por 2 células. Isso provocou uma tendência interessante na seleção de candidatos pelo algoritmo de seleção genética: mapas que continham uma área fechada com uma "porta" com tamanho de uma célula e um local de nascimento dentro da área se tornaram extremamente dominantes, pois o tempo médio de vida dos *bots* que conseguiam nascer dentro destas áreas era enorme, chegando à valores acima de 60.000 na função objetivo. Para resolver este problema, o algoritmo desta representação passa por uma fase extra após a geração do mapa: checamos em todas as células livres do mapa se existem células ocupadas ao norte e ao sul, ou ao leste e oeste da célula livre. Caso haja, transformamos a célula norte ou a célula leste em células livres respectivamente. Alguns mapas dessa representação podem ser vistos na Figura 3.4.

3.5.3 All-Black

O mapa nesta representação começa inicialmente no estado cheio e gradualmente abre espaços em posições definidas pelo genoma. Há dois tipos de espaços livres: arenas e

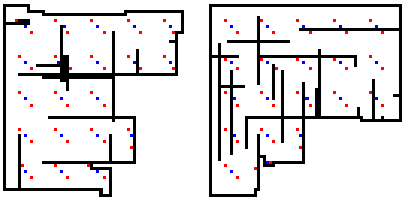

Figura 3.4: Alguns mapas All-White. Pontos azuis são locais de nascimento e pontos vermelhos são munições.

corredores, ambos são codificados da mesma maneira pela tripla $< x, y, s >$, onde x e y representam as coordenadas do centro do corredor/arena e s representa o raio da arena ou a largura do corredor, s tem um valor entre -32 e 32, x e y tem um valor entre 0 e 64. Corredores tem uma altura fixa de valor 3. [5] utilizaram 30 corredores e 5 arenas, ou seja, $(N_w = 35)$, e este trabalho fará o mesmo. O genoma desta representação deste tem tamanho $n = 105$, pois $t = 3$ e $N_w = 35$. Alguns mapas dessa representação podem ser vistos na Figura 3.5.

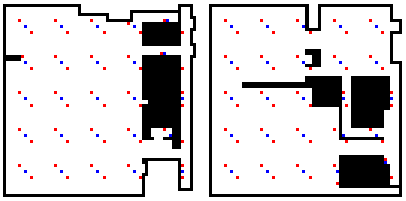

Figura 3.5: Alguns mapas All-Black. Pontos azuis são locais de nascimento e pontos vermelhos são munições.

3.5.4 Salas Cíclicas 1

A representação SC1, criada para este trabalho, tenta gerar mapas de uma maneira bem direta: o mapa começa no estado cheio e inserimos nele 12 salas dentro dos limites do espaço de 64 por 64, e cada uma destas salas são conectadas à três outras salas por meio de corredores. Para representar cada sala no mapa, utilizamos a sêxtupla $< x, y, s, a, b, c >$, onde x e y representam as coordenadas do centro da sala; s o raio da sala; a, b e c o número identificador das salas que estão conectadas a esta sala, s tem um valor entre 0 e 32, x e y tem um valor entre 0 e 64, a, b e c tem um valor entre 0 e 11. O número de salas e de conexões entre salas foi escolhido arbitrariamente. O genoma desta representação deste tem tamanho $n = 72$, pois $t = 6$ e $N_w = 12$. Alguns mapas dessa representação podem ser vistos na Figura 3.6.

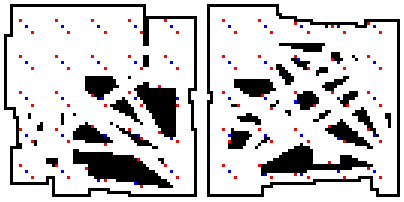

Figura 3.6: Alguns mapas Salas Cíclicas. Pontos azuis são locais de nascimento e pontos vermelhos são munições.

3.5.5 Salas Cíclicas 2 e 3

Estas representações diferem de SC1 somente no possível valor atribuído a s, que foi modificado para estar entre 0 e 16 em SC2, e entre 0 e 8 em SC3. Alguns mapas dessas representações podem ser vistos nas Figuras 3.7 e 3.8, respectivamente.

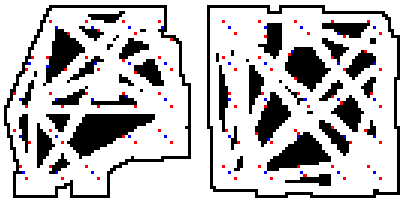

Figura 3.7: Alguns mapas Salas Cíclicas 2. Pontos azuis são locais de nascimento e pontos vermelhos são munições.

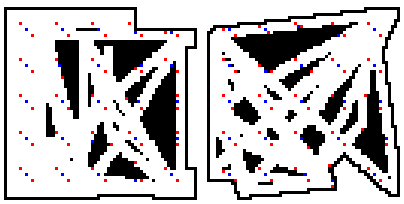

Figura 3.8: Alguns mapas Salas Cíclicas 3. Pontos azuis são locais de nascimento e pontos vermelhos são munições.

3.5.6 Pontos Cíclicos

Por fim, na representação PC nos livramos das salas e utilizamos somente pontos interligados. O genoma é composto por 12 pontos, cada um representado pela quíntupla $< x, y, a, b, c >$, onde x e y representam as coordenadas do centro do ponto e a, b e c representam quais pontos estão ligados a este ponto. O genoma desta representação tem tamanho $n = 60$, pois $t = 5$ e $N_w = 12$.

Alguns mapas dessa representação podem ser vistos na Figura 3.9.

O código fonte do algoritmo genético, do gerador de mapas e das representações de mapas estão disponíveis no repositório [11]

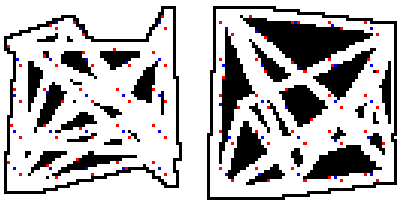

Figura 3.9: Alguns mapas Pontos Cíclicos. Pontos azuis são locais de nascimento e pontos vermelhos são munições.

Capítulo 4

Resultados

Neste capítulo descreveremos os experimentos realizados neste trabalho seguido do tempo de execução do nosso algoritmo genético, das simulações e dos testes de 50 gerações. Apos isso descreveremos em detalhes os resultados obtidos por cada representação de mapa e suas características.

4.1 Experimentos

Os experimentos foram feitos em um computador com um processador *Intel Core i3 M 330 2,13GHz x2* com o sistema operacional *Ubuntu 15.04 Vivid x64*. O jogo C2, o algoritmo de seleção genética, o algoritmo de geração de mapas e os arquivos gerados por ambos foram controlados por um *script Bash*, o tempo de execução foi medido com o comando padrão *time* e utilizamos o comando padrão *taskset* para designar um teste de 50 gerações para cada *core* do computador, ou seja, quatro testes eram executadas ao mesmo tempo. Repetimos cada teste de 50 gerações 50 vezes para cada representação.

4.1.1 Tempo de execução

Apesar do foco deste trabalho não ser na velocidade em que conseguimos produzir os resultados, colhemos em meio aos testes de 50 gerações o tempo de execução do ASGGM, das simulações dos *bots* e dos próprios testes. O tempo total da execução de um teste de 50 gerações foi em torno de 300 a 400 minutos, com cada simulação de jogo durando em torno de 5 a 15 segundos e cada execução da seleção genética e geração de mapas durando em torno de 0,14 segundos. Em [5], as simulações de jogo duravam em torno de 10 segundos e o tempo médio dos testes de 50 gerações foi de 360 minutos.

Tabela 4.1: Teste t de Student comparando os valores da função objetivo dos mapas gerados após 50 gerações de seleção genética.

TTEST	All-black	All-white	Grid	PC	SC1	SC2	SC3
All-black	1	> 0,00001	> 0,00001	> 0,00001	> 0,00001	> 0,00001	> 0,00001
All-white	> 0,00001	1	> 0,00001	0,000072	0,086439	0,401615	0,924154
Grid	> 0,00001	> 0,00001	1	0,468701	> 0,00001	> 0,00001	> 0,00001
PC	> 0,00001	0,000072	0,468701	1	> 0,00001	0,000252	0,000149
SC1	> 0,00001	0,086439	> 0,00001	> 0,00001	1	0,003953	0,069054
SC2	> 0,00001	0,401615	> 0,00001	0,000252	0,003953	1	0,461938
SC3	> 0,00001	0,924154	> 0,00001	0,000149	0,069054	0,461938	1

Tabela 4.2: Média dos valores da função objetivo dos mapas e suas representações.

All-black	All-white	Grid	PC	SC1	SC2	SC3
13509,88	18163,45	20582,8	20243,73	17466,55	18493,68	18205,83

4.1.2 Representações de Mapas

Como dito anteriormente, coletamos 50 mapas para cada tipo de representação de mapa. Vimos com o teste de normalidade Shapiro-Wilk que algumas das nossas amostras não estavam de acordo com a forma normal, mas chegavam perto (nível de significância estatística $p \approx 0{,}15$). Para normalizar estas amostras, utilizamos o GESDT, que é utilizado para retirar valores aberrantes, ou seja, valores inconsistentes com os demais da amostra [15], para tentar aproximar as amostras da distribuição normal.

Dos 50 mapas para cada representação, retiramos os valores aberrantes da função objetivo com o *GESDT* (com o nível de significância em 0,5 e o número máximo de valores aberrantes igual a 10) pois nossos valores estavam perto da forma normal (nível de significância estatística $p \approx 0{,}15$), checamos se estavam na forma normal com o teste de normalidade *Shapiro-Wilk* (com o nível de significância estatística p = 0,10) pois a maioria dos valores encontrados são únicos, e aplicamos o teste t de Student independente de duas amostras com variância igual e tamanho de amostras diferentes entre cada uma das representações. Os resultados podem ser vistos na Tabela 4.1, a média dos valores da função objetivo está na Tabela 4.2.

A seguir, descreveremos as equações estatísticas utilizadas, os resultados, as tendências e os melhores mapas de cada uma das representações de geração de mapas.

All-Black

De acordo com nossos resultados com o teste t de Student, vimos que a representação All-Black é estatisticamente pior (p-valor < 0,0001) do que todos as outras representações deste trabalho.

Os mapas criados por esta representação tendem a ter grandes áreas abertas conectadas por conjuntos de corredores. Estes corredores muitas vezes não levam a lugar algum, são sem saída.

A melhores pontuações da função objetivo alcançadas com esta representação foram 17261, 16765 e 16380 (Figura 4.1).

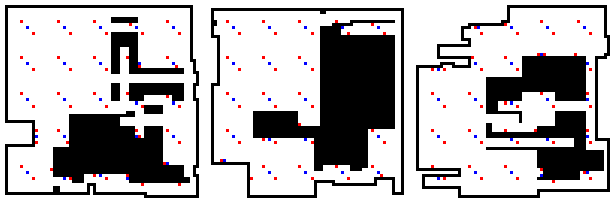

Figura 4.1: Melhores mapas All-Black gerados, com valores da função objetivo de 17261, 16765 e 16380 respectivamente. Pontos azuis são *spawn points* e pontos vermelhos são munições.

All-White

Vimos que a representação All-White é estatisticamente igual à SC1 (p-valor = 0,086), SC2 (p-valor = 0,401) e SC3 (p-valor = 0,924); e estatisticamente pior do que Grid (p-valor $<$ 0,0001) e PC (p-valor $<$ 0,0001). Mapas desse tipo tendem a ter áreas longas horizontais ou verticais, muitas delas terminando em corredores sem saída; há também áreas grandes com obstáculos no meio. As melhores pontuações da função objetivo alcançadas com esta representação foram 23312, 22804 e 22528 (Figura 4.2).

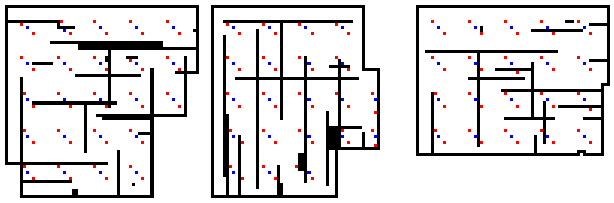

Figura 4.2: Melhores mapas All-White gerados, com valores da função objetivo de 23312, 22804 e 22528 respectivamente. Pontos azuis são *spawn points* e pontos vermelhos são munições.

Grid

Grid é estatisticamente igual à PC (p-valor = 0,468) e melhor do que todos as outras representações (p-valor $<$ 0,0001). Este tipo de representação tende a criar mapas com características que lembram labirintos básicos com *loops*, ou seja, seções de paredes que não estão ligadas às bordas do labirinto. Isso faz com que os mapas tenham muitos corredores sem saída.

As melhores pontuações da função objetivo alcançadas com esta representação foram 24271, 24226 e 24057 (Figura 4.3).

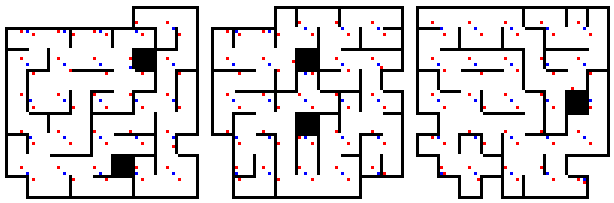

Figura 4.3: Melhores mapas Grid gerados, com valores da função objetivo de 24271, 24226 e 24057 respectivamente. Pontos azuis são *spawn points* e pontos vermelhos são munições.

SC1

SC1 é estatisticamente igual à SC3 (p-valor = 0,069) e All-White (p-valor = 0,086); pior do que SC2 (p-valor = 0,003), PC (p-valor < 0,0001), Grid (p-valor < 0,0001); e melhor do que All-Black (p-valor < 0,0001). Esta representação cria mapas com grandes áreas abertas com múltiplas rotas de saída, mas a grande quantidade de corredores entre as áreas grandes tem um efeito imprevisto: nas regiões onde há muitos cruzamentos entre corredores, estas regiões se tornam áreas grandes com múltiplos obstáculos.

As melhores pontuações da função objetivo alcançadas com esta representação foram 21136, 20640 e 20548 (Figura 4.4).

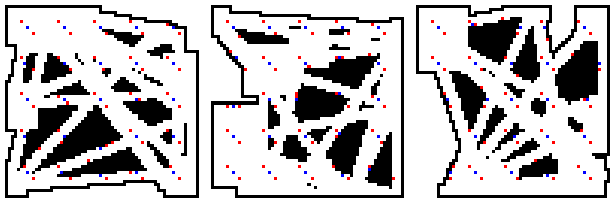

Figura 4.4: Melhores mapas SC1 gerados, com valores da função objetivo de 21136, 20640 e 20548 respectivamente. Pontos azuis são *spawn points* e pontos vermelhos são munições.

SC2

SC2 é estatisticamente igual à SC3 (p-valor = 0,461) e All-White (p-valor = 0,401); melhor do que SC1 (p-valor = 0,003) e All-Black (p-valor < 0,0001); pior do que Grid (p-valor < 0,0001) e PC (p-valor = 0,0002). Os mapas desta representação são semelhantes aos mapas da representação SC1, mas com menos áreas grandes sem obstáculos.

As melhores pontuações da função objetivo alcançadas com esta representação foram 21748, 21579 e 21504 (Figura 4.5).

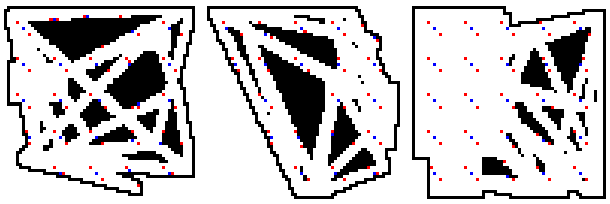

Figura 4.5: Melhores mapas SC2 gerados, com valores da função objetivo de 21748, 21579 e 21504 respectivamente. Pontos azuis são *spawn points* e pontos vermelhos são munições.

SC3

SC3 é estatisticamente igual à SC2 (p-valor = 0,461), SC1 (p-valor = 0,069) e All-White (p-valor = 0,924); melhor do que All-Black (p-valor < 0,0001); e pior do que Grid (p-valor < 0,0001) e PC (p-valor = 0,0001). Os mapas desta representação são semelhantes aos mapas da representação SC1 e SC2, mas com um número ainda menor de áreas grandes sem obstáculos.

As melhores pontuações da função objetivo alcançadas com esta representação foram 22313, 21809 e 21510 (Figura 4.6).

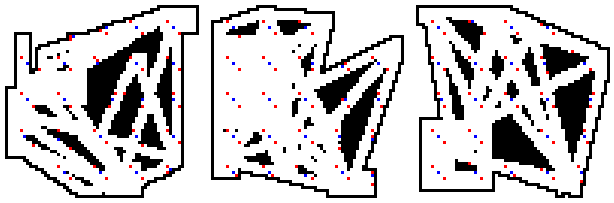

Figura 4.6: Melhores mapas SC3 gerados, com valores da função objetivo de 22313, 21809 e 21510 respectivamente. Pontos azuis são *spawn points* e pontos vermelhos são munições.

PC

PC é estatisticamente igual à Grid (p-valor = 0,468); melhor do que todos as outras representações (p-valor < 0,0002). Os mapas desta representação são semelhantes aos mapas das representações SC1, SC2 e SC3, mas com um número quase inexistente de áreas grandes sem obstáculos.

As melhores pontuações da função objetivo alcançadas com esta representação foram 26308, 26360 e 25077 (Figura 4.7).

4.2 Comparações

De acordo com os resultados obtidos, vemos que a ordem de melhores para piores representações de mapas é: Grid e PC empatados como melhores; All-White, SC2 e SC3

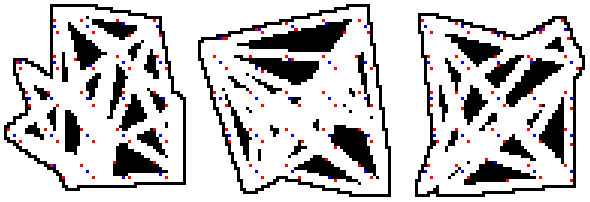

Figura 4.7: Melhores mapas PC gerados, com valores da função objetivo de 26308, 26360 e 25077 respectivamente. Pontos azuis são *spawn points* e pontos vermelhos são munições.

empatados em segundo lugar; SC1 em terceiro e All-Black em último lugar. O mapa com o maior valor da função objetivo entre todas as representações, teve um valor igual à 26360, da representação PC. Nas Figuras 4.10,4.9,4.8 e 4.11 mostramos as notas da função objetivo de todos os mapas gerados das melhores representações (Grid e PC), das piores representações (SC1 e All-Black), das representações medianas (SC2, SC3 e All-White) e todas as representações em conjunto, respectivamente.

Os resultados obtidos em [5] foram os seguintes: All-White e Grid empataram como os melhores, All-black em segundo e Random-Digger em terceiro. All-White gerou o melhor mapa com um nota da função objetivo de 22882.

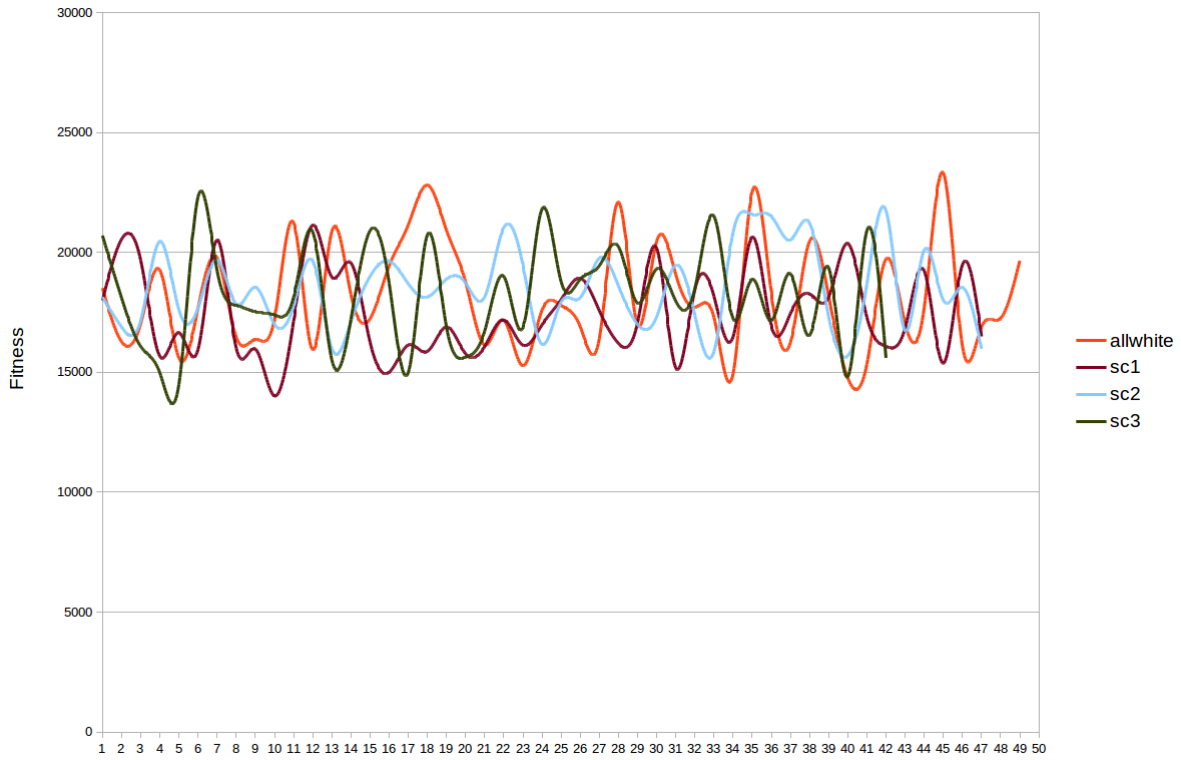

Figura 4.8: Valores da função objetivo das representações medianas: SC1, SC2, SC3 e All-White.

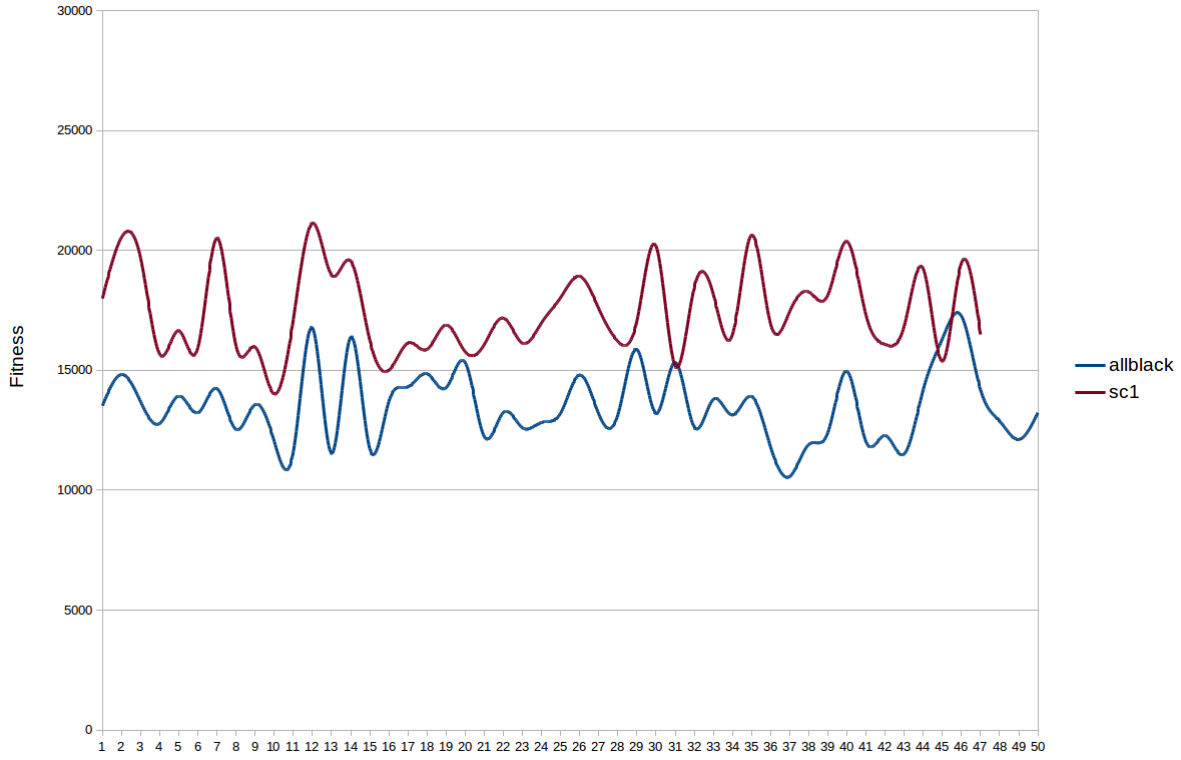

Figura 4.9: Valores da função objetivo das piores representações: SC1 e All-Black

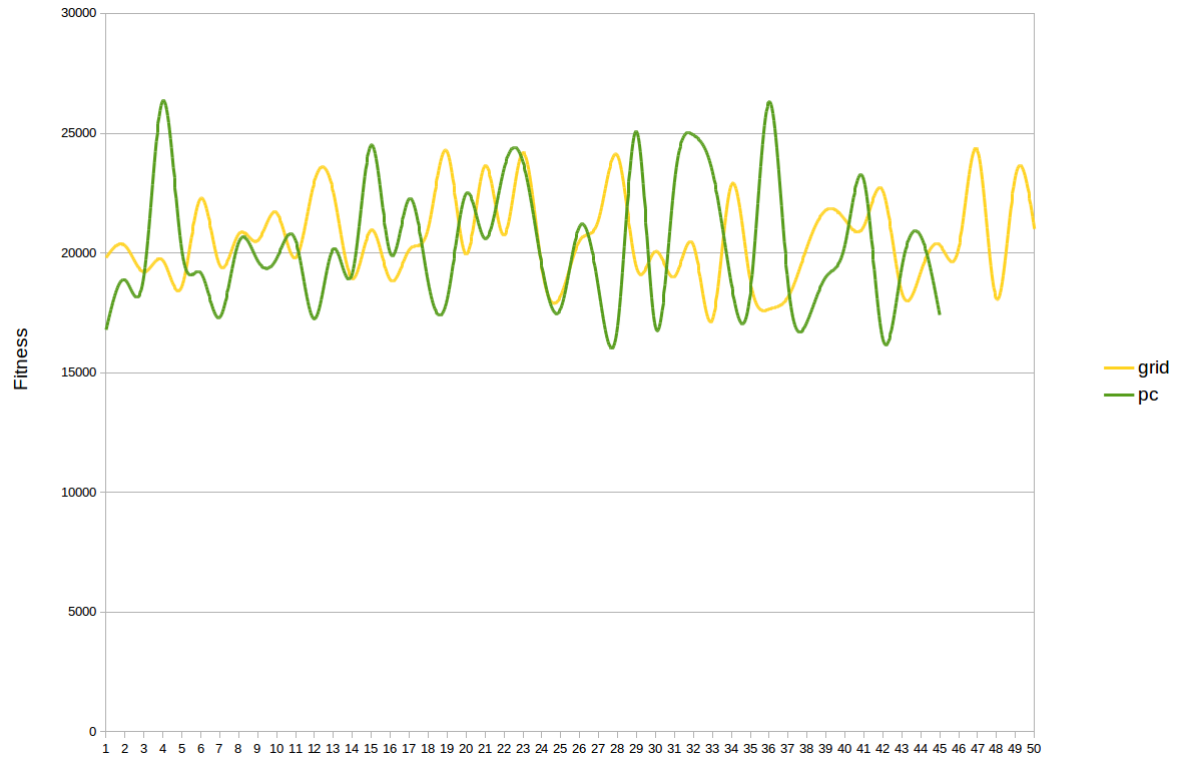

Figura 4.10: Valores da função objetivo das melhores representações: PC e Grid.

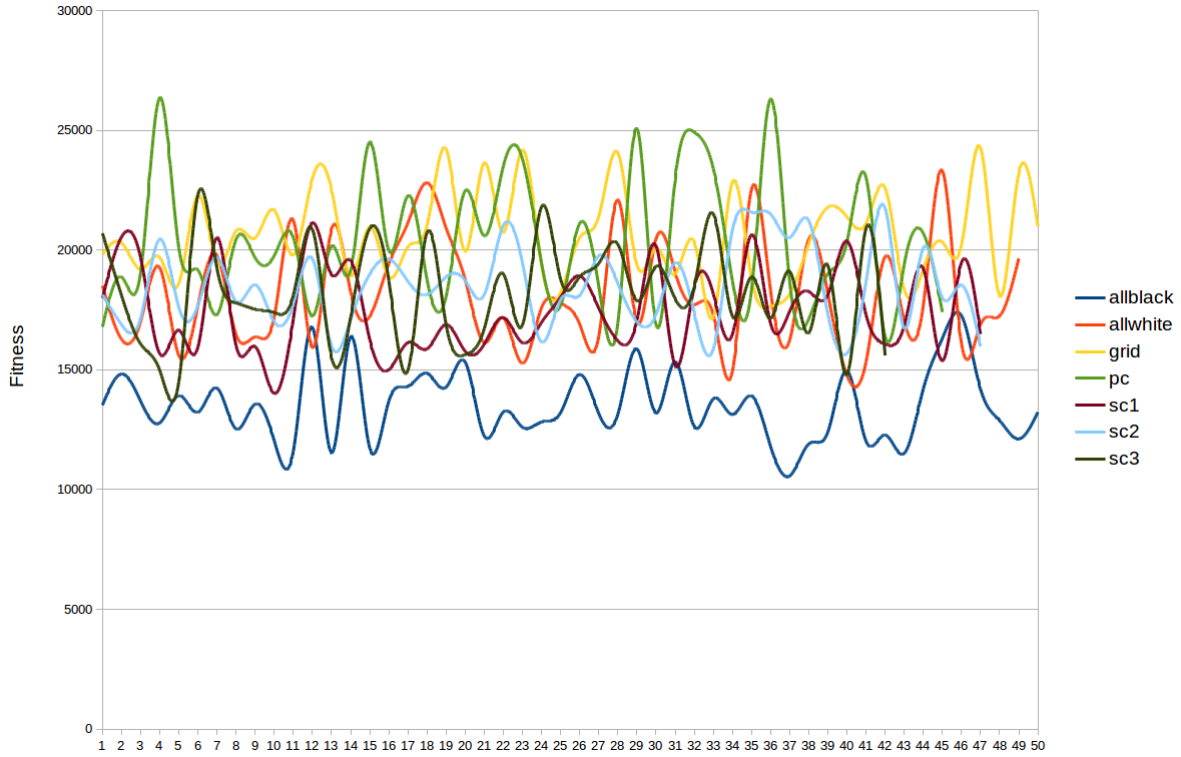

Figura 4.11: Valores da função objetivo de todas as representações (PC, SC1, SC2, SC3, All-White, All-Black e Grid).

Capítulo 5

Conclusões

Neste capítulo apresentaremos nossas conclusões e os trabalhos futuros recomendados.

5.1 Conclusões

Neste trabalho, nosso objetivo principal foi desenvolver representações de mapas que produzissem mapas tão bons quanto ou melhores do que os apresentados por [5]. De acordo com nossos resultados, nossa representação PC se enquadra exatamente no que procurávamos alcançar. Ele não só gerou resultados tão bons quanto a representação Grid e gerou o mapa com o maior valor da função objetiva (26360), como conseguiu estes resultados criando mapas que diferem do formato e das características das reresentações All-White, Grid e All-Black mas ainda condizem com as técnicas de *design* de níveis descrita em [36].

Nossas outras representações — SC1, SC2 e SC3 — produziram resultados bons, comparáveis aos resultados obtidos com a representação All-White, e podem ser usadas como alternativas à Grid e PC quando necessário ou desejado.

É importante notar que as representações criados neste trabalho não foram feitas com o intuito de remover o aspecto humano da criação de mapas, do *design* de níveis. Recomendamos que estas representações sejam usadas como uma ferramenta de auxílio, para diminuir os custos e o tempo gasto na criação de um mapa para um jogo FPS.

5.2 Trabalhos futuros

De acordo com [5], é uma prioridade para estudos futuros pesquisar casos de uso para verificar se a função objetivo realmente corresponde com o julgamento de jogadores humanos. Utilizar *bots* projetados para copiar o comportamento de um jogador humano padrão também poderia ajudar na automatização dos testes e na melhoria da função objetivo.

Há também o problema do posicionamento dos itens e munição dentro do mapa que pode ser resolvido em um estudo futuro, rodando em um mapa previamente evoluído (utilizando as mesmas técnicas apresentadas neste trabalho) um algoritmo genético para descobrir a melhor maneira de posicionar itens e munição pelo mapa.

Outro estudo futuro que poderia ajudar para facilitar testes com jogadores humanos seria desenvolver uma versão deste trabalho (ou um exportador de mapas) para jogos, motores ou editores de mapas mais populares como *GTKRadiant* (utilizado por jogos que usam o engine *id Tech 3* como *Quake 3* ou a série *Call Of Duty*), *Hammer* (utilizado pelos jogos que usam o engine *Source* como *Counter Strike: Global Offensive* e *Left 4 Dead*), *Unity* e *Unreal 4*.

Referências

[1] Ernest Adams. *Fundamentals of Game Design*. New Riders, 1996. 21

[2] Diaz-Furlong Hector Adrian e Solis-Gonzalez Cosio Ana Luisa. An approach to level design using procedural content generation and difficulty curves. In *CIG*, pages 1–8. IEEE, 2013. 1

[3] Brenda Brathwaite e Ian Schreiber. *Challenges for Game Designers*. Charles River Media, Inc., Rockland, MA, USA, 1 edition, 2008. 11

[4] Ed Byrne. *Game Level Design*. Charles River Media, 2005. 6

[5] Luigi Cardamone, Georgios N. Yannakakis, Julian Togelius, e Pier Luca Lanzi. Evolving interesting maps for a first person shooter. In *EvoApplications (1)*, volume 6624 of *Lecture Notes in Computer Science*, pages 63–72. Springer, 2011. vi, 2, 3, 16, 18, 21, 22, 23, 25, 26, 28, 29, 31, 32, 35, 40, 43

[6] Anders Carlsson. A micro history of demoscene music. `http://rhizome.org/editorial/2010/may/18/a-micro-history-of-demoscene-music/`, 5 2010. 14

[7] Andrew Clayton. *Introduction to Level Design for PC Games*. Charles River Media, 2003. 6

[8] Phil Co. *Level Design for Games: Creating Compelling Game Experiences*. New Riders, 2006. 6

[9] Simon Colton. Automated puzzle generation. In *In Proceedings of the AISB'02 Symposium on AI and Creativity in the Arts and Science*, 2002. 10

[10] Michael Cook, Simon Colton, e Jeremy Gow. Initial results from co-operative co-evolution for automated platformer design. In *EvoApplications*, volume 7248 of *Lecture Notes in Computer Science*, pages 194–203. Springer, 2012. 12

[11] Davi Diniz. Repositório geneticalgorithmfps. `https://github.com/DSVDiniz/GeneticAlgorithmFPS`, 12 2015. 33

[12] Kristian Duske. Cube 2: Sauerbraten documentation. `http://sauerbraten.org/docs/game.html`, 5 2010. 18

[13] Sean H. Fahy. Why most multiplayer only games are doomed to fail. `http://sites.psu.edu/ist446/2015/03/01/why-most-multiplayer-only-games-are-doomed-to-fail/`, 3 2015. 1

[14] John Feil e Marc Scattergood. *Beginning Game Level Design*. Cengage Learning PTR, 2005. 6

[15] Frank E. Grubbs. Procedures for detecting outlying observations in samples. *Technometrics*, 11(1):1–21, February 1969. 16, 36

[16] Mark Hendrikx, Sebastiaan Meijer, Joeri Van Der Velden, e Alexandru Iosup. Procedural content generation for games: A survey. *TOMCCAP*, 9(1):1, 2013. 7

[17] Michael Hitchens. A survey of first-person shooters and their avatars. *Game Studies*, 11(3), 2011. 5

[18] James Holloway. Deathmatch map design: The architecture of flow. `http://www.gamasutra.com/view/feature/195069/deathmatch_map_design_the_.php`, 2013. 6

[19] Remco Huijser, Jeroen Dobbe, Willem F. Bronsvoort, e Rafael Bidarra. Procedural natural systems for game level design. In *SBGames*, pages 189–198. IEEE Computer Society, 2010. 1

[20] Kenneth Hullett e Jim Whitehead. Design patterns in fps levels. In Ian Horswill e Yusuf Pisan, editors, *FDG*, pages 78–85. ACM, 2010. 6

[21] Zhang Joy. Confidence interval and the student's t-test, 12 2006. 18

[22] Patrick Klepek. Elder scrolls ii: Daggerfall's map is massive - how's it compare to oblivion? `http://www.g4tv.com/thefeed/blog/post/697303/elder-scrolls-ii-daggerfalls-map-is-massive-hows-it-compare-to-oblivion`, 07 2009. 1

[23] David Kushner. *Masters of Doom: How Two Guys Created an Empire and Transformed Pop Culture*. Random House, 2003. 1, 6

[24] Antonios Liapis, Georgios N. Yannakakis, e Julian Togelius. Designer modeling for personalized game content creation tools. pages 1–6, 2013. 1

[25] Elizabeth A. Matthews e Brian A. Malloy. Procedural generation of story-driven maps. In *CGAMES*, pages 107–112. IEEE, 2011. 12

[26] M. Mitchell. *An Introduction to Genetic Algorithms*. The MIT Press, 1996. 15

[27] N/A. Cube 2: Sauerbraten. `http://sauerbraten.org`, 5 2010. 18

[28] D.A. Norman. *The design of everyday things*. New York: Doubleday, 1990. 11

[29] Peter Thorup Olsted, Benjamin Ma, e Sebastian Risi. Interactive evolution of levels for a competitive multiplayer fps. In *CEC*, pages 1527–1534. IEEE, 2015. 2

[30] TeongJoo Ong, Ryan Saunders, John Keyser, e John J. Leggett. Terrain generation using genetic algorithms. In Hans-Georg Beyer e Una-May O'Reilly, editors, *GECCO*, pages 1463–1470. ACM, 2005. 14

[31] Tony Ponce. Aaa game development teams are too damn big. http://www.destructoid.com/aaa-game-development-teams-are-too-damn-big-247366.phtml, 3 2013. 1

[32] William L. Raffe, Fabio Zambetta, e Xiaodong Li. A survey of procedural terrain generation techniques using evolutionary algorithms. In *IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE, 2012. 14

[33] Scott Rogers. *Level Up!: The Guide to Great Video Game Design*. Wiley Publishing, 1st edition, 2010. 4

[34] Bernard Rosner. Percentage Points for a Generalized ESD Many-Outlier Procedure. *Technometrics*, 25:165–172, 1983. 17

[35] Jim Rossignol. *This Gaming Life: Travels in Three Cities*. Digital Culture Books - Imprint of: The University of Michigan Press, Ann Arbor, MI, 2008. 11

[36] Marc Saltzman. Secrets of the sages: Level design. http://www.gamasutra.com/view/feature/131767/secrets_of_the_sages_level_design.php?print=1, 5 2010. vi, 1, 2, 21, 22, 43

[37] Noor Shaker, Georgios N. Yannakakis, e Julian Togelius. Towards automatic personalized content generation for platform games. In G. Michael Youngblood e Vadim Bulitko, editors, *AIIDE*. The AAAI Press, 2010. 1

[38] S. S. Shapiro e M. B. Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4):591–611, Dec. 1965. 16, 17

[39] Ruben M. Smelik, Klaas Jan De Kraker, Saskia A. Groenewegen, Tim Tutenel, e Rafael Bidarra. A survey of procedural methods for terrain modelling. In *Proc. of the CASA Workshop on 3D Advanced Media In Gaming And Simulation (3AMIGAS*, 2009. 14

[40] Shui Ta. Assassin's creed 4 ps4 and xbox 720 development team size revealed. http://www.examiner.com/article/assassin-s-creed-4-ps4-and-xbox-720-development-team-size-revealed, 03 2013. 1

[41] Julian Togelius, Mike Preuss, Nicola Beume, Simon Wessing, Johan Hagelbäck, e Georgios N. Yannakakis. Multiobjective exploration of the starcraft map space. In Georgios N. Yannakakis e Julian Togelius, editors, *CIG*, pages 265–272. IEEE, 2010. 12

[42] Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley, e Cameron Browne. Search-based procedural content generation: A taxonomy and survey. *IEEE Trans. Comput. Intellig. and AI in Games*, 3(3):172–186, 2011. 12, 13, 14

[43] Paul Walsh e Prasad Gade. Terrain generation using an interactive genetic algorithm. In *IEEE Congress on Evolutionary Computation*, pages 1–7. IEEE, 2010. 12

[44] Glenn R. Wichman. A brief history of "rogue". http://www.wichman.org/roguehistory.html, 1997. 12

[45] Georgios N. Yannakakis e Julian Togelius. Experience-driven procedural content generation. *T. Affective Computing*, 2(3):147–161, 2011. 7

[46] Charles Zaiontz. Generalized extreme studentized deviate test. http://www.real-statistics.com/students-t-distribution/identifying-outliers-using-t-distribution/generalized-extreme-studentized-deviate-test/, 2015. 17