

Convertendo Imagens para ASCII art e Pixel art utilizando Algoritmos Genéticos¹

Leonardo de Araujo Leitão, Yuri de Almeida Malheiros Barbosa

Departamento de Ciências Exatas - Universidade Federal da Paraíba (UFPB)

{leonardo.leitao,yuri}@dcx.ufpb.br

Resumo. *A inteligência artificial está se tornando cada vez mais presente no dia a dia, diferentes aplicações das mais simples às mais robustas a utiliza para as mais variadas tarefas, por exemplo, avaliação musical, auxiliar na elaboração de escala de trabalho, gerar design inovadores, etc. Este artigo descreve uma aplicação que utiliza algoritmos genéticos para converter uma imagem para formato de ASCII art e Pixel art coloridas de modo automático. Essa implementação é focada na evolução automática de imagens através de algoritmos genéticos. Com essa técnica, usuários que utilizam ferramentas para criação de imagens estilizadas terão um contato mais próximo com a IA que será utilizada como ferramenta para gerar imagens seguindo os estilos propostos pelo artigo. Os resultados mostram que a utilização de algoritmos genéticos para converter imagens são satisfatórios tornando a técnica uma opção para alcançar esse objetivo.*

1. Introdução

Computadores são usados como ferramenta para criação e manipulação de imagens há décadas, se tornando um dos principais meios de produção artística da atualidade, considerando uma demanda cada vez maior por programas destinados a este fim. Os programas utilizados para realização dessas tarefas de manipulação de imagens são na sua grande maioria manuais, onde o usuário deve realizar quase todo trabalho, porém, existem técnicas mais sofisticadas para tornar o processo mais automático.

A inteligência artificial (IA) para aplicações criativas é uma área de pesquisa em pleno desenvolvimento atualmente. Para [Foster 2019] agora é possível ensinar uma máquina a se destacar em habilidades humanas como pintar obras de arte em um determinado estilo, escrever e compor músicas agradáveis de ouvir. Algumas das técnicas existentes para a manipulação de imagens se utilizam da IA para alcançar esses objetivos, assim aproximando as aplicações inteligentes ao meio artístico. Dentre as técnicas existentes, nós temos os algoritmos genéticos (AGs) que podem ser usados para gerar imagens, sons, etc.

Nesse trabalho, os algoritmos genéticos serão utilizados para aplicações criativas visando gerar imagens coloridas utilizando *ASCII art* e *Pixel art*. American Standard

¹ Trabalho de conclusão de curso, sob orientação do professor Yuri de Almeida Malheiros Barbosa submetido ao Curso de Bacharelado em Sistemas de Informação do Centro de Ciências Aplicadas e Educação (CCAEE) da Universidade Federal da Paraíba, como parte dos requisitos necessários para obtenção do grau de BACHAREL EM SISTEMAS DE INFORMAÇÃO.

Code For Information Interchange (ASCII) ou em português Código Padrão Americano para Troca de Informações, foi proposto como uma solução para unificar a forma de como se deve representar os caracteres alfanuméricos nos computadores através de uma tabela composta de 31 caracteres reservados com a finalidade de servirem como funções de controle e outros 95 caracteres que podem ser impressos. Uma consequência da utilização da tabela ASCII acabou dando origem a uma forma de arte conhecida como ASCII art que segundo [Xu et al apud Wikipédia] é uma técnica de composição de figuras com caracteres imprimíveis. De acordo com [Xu et al apud Wikipédia] isso decorreu da incapacidade de apresentação gráfica em computadores antigos. Portanto, os caracteres de textos são utilizados no lugar dos gráficos. A ASCII art utiliza os caracteres presentes na tabela para imitar imagens em forma de texto.

O pixel é o menor ponto de uma imagem digital. Cada pixel é composto por três valores: vermelho, verde e azul, cada valor pode tomar até 256 tonalidades diferentes e combinados podem formar até 16.777.216 (dezesseis milhões setecentos e setenta e sete mil duzentos e dezesseis) cores. Cada pixel pode conter um quarto valor chamado alpha que determina a sua transparência. Uma imagem com resolução de 200 x 200 é composta por 40.000 pixels. A pixel art começou a ser amplamente utilizada em meados dos anos 70 se consagrando nos anos 80. A pixel art utiliza esses pixels no seu desenvolvimento. Ela busca deixar bem definido cada pixel da imagem, se utilizando de baixas resoluções para alcançar esse efeito.

Esse trabalho se propõe a utilizar algoritmos genéticos para criação de imagens estilizadas com o diferencial de gerar imagens coloridas utilizando *ASCII art* e *Pixel art*. Busca-se determinar a viabilidade de se utilizar AGs para essa finalidade, levando em consideração a qualidade da imagem gerada, observando se essa imagem se parece com a original e o tempo levado para se alcançar essa similaridade.

Foi determinado que cada imagem terá sua versão ASCII art e Pixel art geradas para efeito de comparação, porém não é uma obrigatoriedade, podendo ser gerada apenas uma ou outra. Pode-se definir a qualidade da imagem escolhendo a quantidade de blocos e caracteres que serão utilizados para gerá-las, quanto maior esse número, maior a qualidade e consequentemente, maior o tempo a ser demandado para alcançar uma alta similaridade.

2. Algoritmos Genéticos

A IA possui diversas subáreas sendo uma delas a área de algoritmos evolucionários onde se encontram os AGs Holland (1945), baseada no processo de seleção natural identificado por Darwin.

Um AG é composto pelos passos que são mostrados na Figura 1 que representa seu fluxo do funcionamento. Um AG possui operadores genéticos, o de recombinação e mutação que visa transformar a população através de várias gerações. Esses dois operadores tem o maior impacto no desempenho do algoritmo. A seguir será discutido cada passo utilizado.

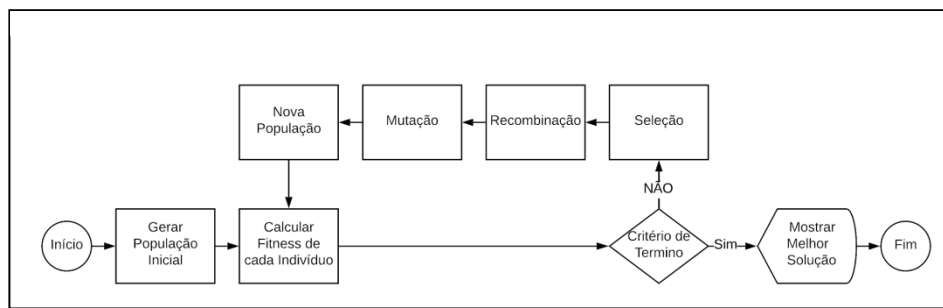


Figura 1. Fluxograma de um Algoritmo Genético

Para [Gen e Lin 2008] no geral, existem dois meios de gerar a população inicial, inicialização heurística e a inicialização aleatorizada, usando um procedimento codificado que satisfaça as restrições do problema e/ou condições de contorno, isto é, a população inicial deve conter soluções, ou seja, indivíduos adequadamente condizentes ao problema abordado. Segundo [Lewis 2008] a população deve ser composta por um valor pré determinado de indivíduos.

Em seguida, já com a população inicial criada, o fitness de cada indivíduo gerado é calculado. Segundo [Gen e Lin 2008] a avaliação do fitness checa o valor da solução da função objetivo sujeita a restrições utilizando um procedimento de decodificação. No geral, o fitness é a função objetivo que provê um meio de avaliar a qualidade de cada solução presente em uma população, ou seja, o fitness vai informar o quão boa ou o quão próxima a solução está do objetivo.

Com o fitness calculado, o algoritmo passa para próxima etapa, a de seleção que, ainda segundo [Gen e Lin 2008] é realizada para melhorar a qualidade média da população dando aos indivíduos de melhor qualidade uma chance maior de serem selecionados para gerar soluções que irão compor a próxima geração, ou seja, a operação de seleção busca escolher genitores para a próxima geração dando uma chance maior aos melhores da atual geração. Alguns exemplos dessa técnica que podem ser utilizados para selecionar os indivíduos são: por torneio, por estado estacionário, por roleta, seleção elitista, hierárquica e escalada [Peña e Truyol 2007].

Com os genitores selecionados, inicia-se a etapa do principal operador genético, a recombinação. que opera em cima de dois genitores ao mesmo tempo e gera descendentes pela combinação dos mesmos [Gen e Lin 2008]. Assim como há várias técnicas na seleção, a recombinação segue da mesma forma. Dentre as técnicas existentes para recombinar indivíduos temos uma forma simples de alcançar a recombinação, escolhendo um ponto de corte aleatório e assim compor a descendência pela combinação do segmento de um pai com a de outro. Assim, podemos dizer que a recombinação trata a forma como os novos indivíduos da próxima geração serão compostos. [Gen e Lin 2008] citam outras técnicas recombinatórias além da descrita, que são elas: dois pontos de corte, múltiplos pontos ou um ponto de corte uniforme, [Lima 2018] cita também recombinação baseado em maioria.

Por fim, os novos indivíduos criados a partir da recombinação são submetidos a operação de mutação. A mutação é um operador exploratório que tem por objetivo aumentar a diversidade na população. A mutação ocorre com uma determinada taxa de probabilidade, em geral baixa ($<1\%$) [Pacheco 1999]. A mutação visa modificar algo no

conteúdo da estrutura de dados definida como sendo uma solução, assim variando e criando novas soluções. A mutação evita o problema do ponto local ótimo que é definido por uma solução que no escopo local pode ser ótima, mas no global não, podendo haver melhores soluções que são desconsideradas caso haja uma convergência da solução do AG nesse ponto. [Konak 2006 apud Lima 2018] reitera que a mutação reintroduz a diversidade genética de volta à população, evitando que o algoritmo caia em máximos e mínimos locais.

A mutação conta com um leque de técnicas que podem ser utilizadas e [Lima 2018] cita e explica algumas, por exemplo, a mutação binária simples ou uniforme, mutação por permutação ou *swap* e mutação *creep*. Sobre a técnica de *swap* [Pacheco 1999] explica que a técnica troca de posição dois genes aleatoriamente escolhidos e que a importância deste operador é, no entanto, restrita a problemas baseados em ordem.

Um exemplo simples de situação onde pode-se utilizar um AG é para encontrar uma frase pré-definida. Para esse AG gera-se a população inicial como um conjunto de soluções aleatórias formadas por uma cadeia de caracteres, em seguida é calculado o *fitness* de cada solução verificando a quantidade de caracteres iguais presentes na solução e na frase pré-definida, quanto mais caracteres iguais, maior o *fitness*. A partir daí seleciona-se dois genitores para realizar a recombinação dando origem aos descendentes que irão fazer parte da próxima geração que será avaliada novamente pela função *fitness* e passará por todos os operadores genéticos realizando os passos descritos, gerando novas populações até alcançar a condição de parada, que nesse exemplo seria uma solução conter os mesmos caracteres da frase definida anteriormente.

3. Metodologia

Para alcançar o objetivo deste trabalho foi desenvolvido um programa na linguagem Java utilizando como dependência para renderização das imagens a biblioteca de jogos Slick2D² e para sumarização de cores a biblioteca Color Thief³.

Na aplicação uma imagem é carregada e é realizada a análise para sumarizar as 10 cores predominantes do seu mapa de cores mais a cor branca por padrão para toda e qualquer imagem, totalizando 11 cores. Com essas informações computadas o algoritmo genético tenta converter a imagem para ASCII art e Pixel art tentando aproximar ao máximo da imagem original passando por cada um dos passos descritos nesta seção e seguindo todos os operadores genéticos.

3.1. Codificando e criando indivíduos iniciais

Na Figura 2 observamos que cada indivíduo é composto por uma cadeia de caracteres, cores, fitness e um valor binário indicando se o indivíduo sofreu ou não mutação. Suas cores são representadas pela classe Color da biblioteca Slick2D e há uma associação feita entre cor e caractere através do seu índice, ou seja, a cor no índice n pertence ao caractere no índice n e vice-versa.

² Slick2D: <https://slick.ninjacave.com/>

³ Color Thief: <https://github.com/SvenWoltmann/color-thief-java>

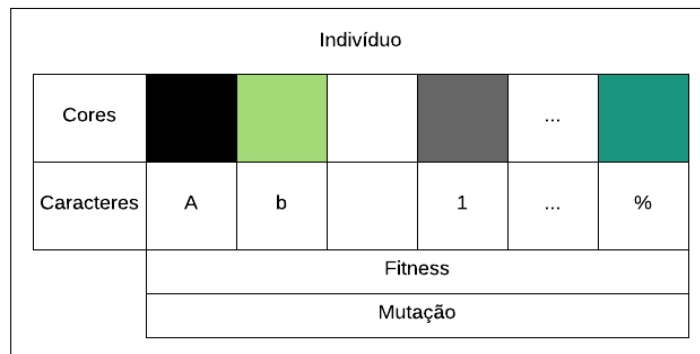


Figura 2. Codifica  o de um Indiv duo

Cada indiv duo da primeira gera  o   montado com as cores transparente e caracteres com o valor \u0000 (vazio). Foi definido que cada par de cor/caractere tem 5% de chance de receber um caract r aleat rio da tabela ASCII e uma cor aleat ria da paleta de cores. Na Figura 3 pode-se observar um exemplo de tela onde mostra um indiv duo da primeira gera  o. O setor A   o local onde cada indiv duo de cada gera  o   renderizado no formato de Pixel art ou ASCII art; o setor B   o local da imagem objetivo; no setor C tem-se o local onde ficam algumas informa  es como FPS (*frames per second*), ou ainda, quadros por segundo que pode ser usado como indicativo de quantos indiv duos s o renderizados em um segundo. Ainda no mesmo setor temos:

- Gera  o: indica a gera  o atual;
- Gera  es por segundo: indica a m dia de gera  es processadas em um segundo;
- Fitness: indica fitness atual;
- % de fitness por segundo: indica a m dia percentual de fitness adquirido por segundo;
- Melhorias: indica a quantidade total de melhorias realizadas;
- Melhorias por segundo: indica a m dia de melhorias que ocorrem por segundo;
- Muta  es: indica a quantidade total de muta  es realizadas;
- Muta  es por segundo: indica a m dia de muta  es que ocorrem por segundo;
- Tempo: indica o tempo decorrido desde o in cio da execu  o do programa.

E por  ltimo o setor D onde encontra-se a paleta de cores que o AG tem a sua disposi  o. Na Figura 4 observa-se duas telas, uma utilizando Pixel art e outra utilizando ASCII art.

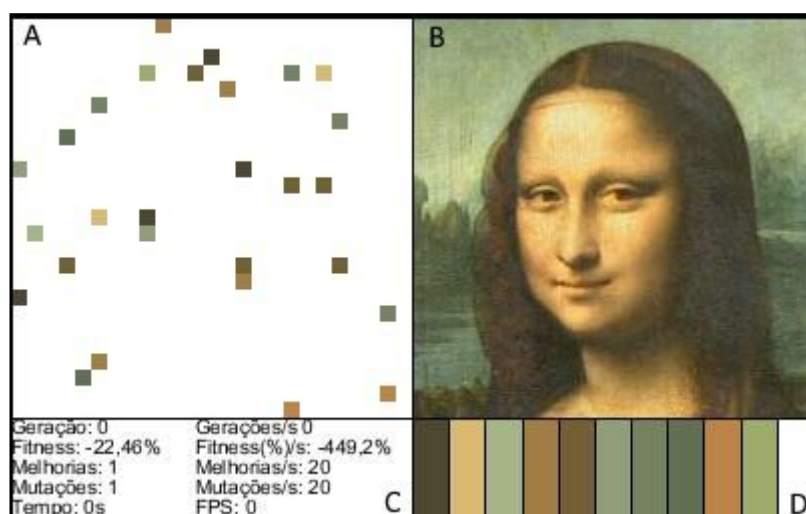


Figura 3. Exemplo de tela com um Indivíduo Inicial utilizando Pixel art

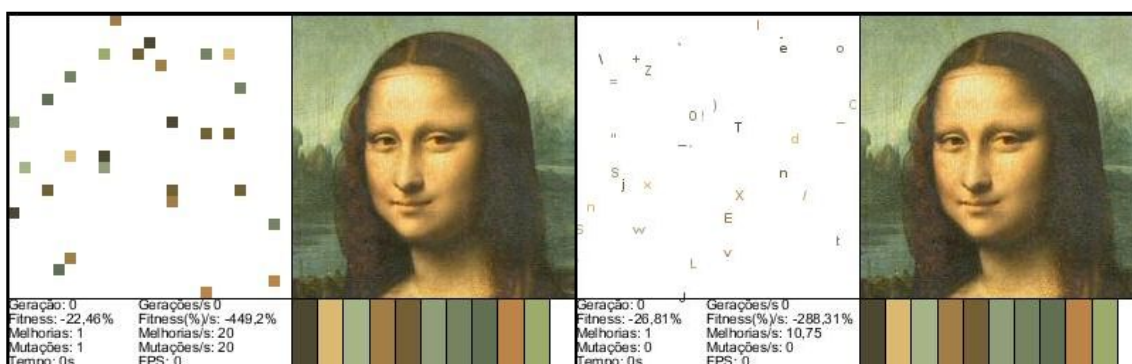


Figura 4. Duas telas mostrando Pixel art (esquerda) e ASCII art (direita) de um indivíduo inicial

3.2. Produzir População Inicial

Deve-se produzir um indivíduo por vez até alcançar a quantidade máxima definida para população. Para esse trabalho foi definida uma população de 50 indivíduos, com o intuito de aliviar o impacto que uma população maior geraria no tempo execução. Uma população maior gera uma maior variabilidade genética, porém aumenta o tempo necessário para processar cada geração devido ao fato de ter mais indivíduos para serem processados. A Figura 5 mostra como é feita a representação da população com um array de indivíduos.

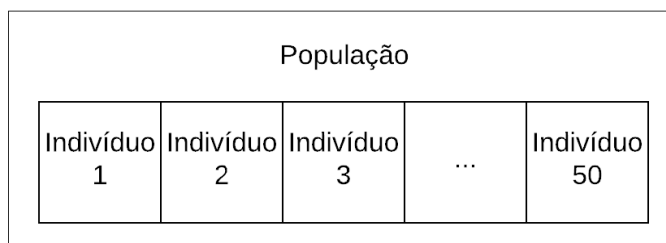


Figura 5. Composição de uma População

3.3. Função Fitness

Para o cálculo do fitness foi utilizada a abordagem de Bergen (2009) que afirma que para imagens o fitness de um indivíduo pode ser calculado baseado na diferença de cor entre todos os pixels da imagem objetivo e a solução gerada pelo algoritmo genético. Levando em consideração essa premissa o fitness é calculado para cada indivíduo todas as vezes que o mesmo é renderizado na tela. Na Figura 6 é mostrado um exemplo de como a função trabalha para calcular a aproximação partir do somatório da distância euclidiana de 4 dimensões utilizando-se o (vermelho, verde, azul, alfa) de cada pixel da imagem gerada com a imagem objetivo dividido pela quantidade total de pixels de uma das imagens para indicar a média entre as distâncias como também é mostrado abaixo na equação da função fitness. A média servirá como indicativo do quão próxima a imagem gerada esta da imagem original. Essa média sempre tenderá a zero enquanto a imagem gerada estiver se aproximando da imagem original, ou seja, quanto mais próximo a média estiver de zero, menor é a distância entre as duas imagens e mais parecidas estão uma da outra.

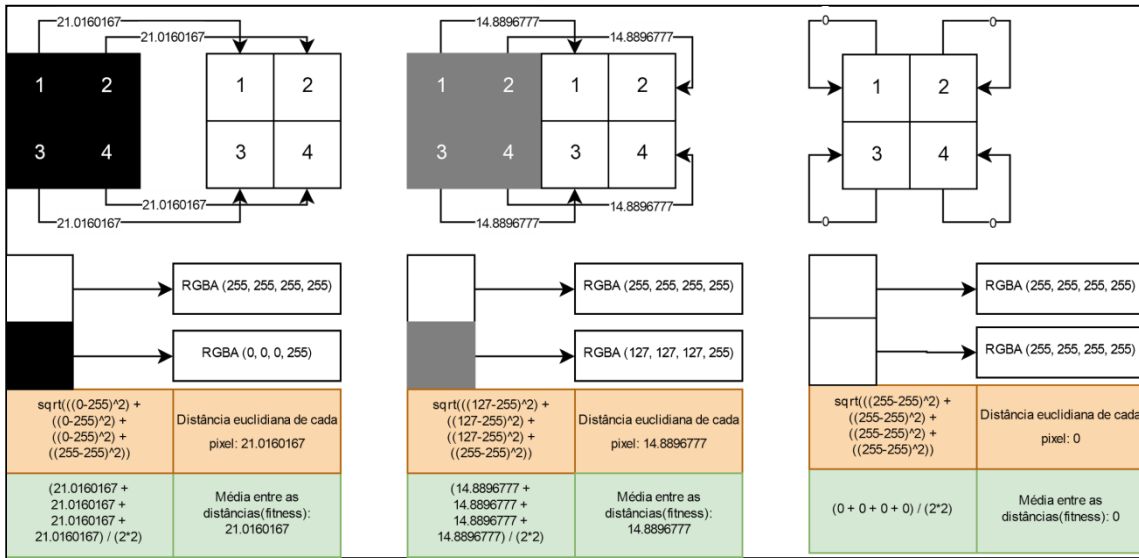


Figura 6. Exemplo de funcionamento da função fitness

Resumindo, podemos calcular o fitness com a seguinte equação:

$$d_{(s, t)} = \frac{\sum_{i=0}^{n_1-1} \left(\sqrt{\sum_{k=0}^{n_2-1} (s_k - t_k)^2} \right)_i}{n_1}$$

onde n_1 é a quantidade total de pixels, n_2 é a quantidade de dimensões a ser calculada, w é a largura e h é a altura.

3.4. Seleção

A seleção dos genitores que irão gerar novos descendentes se dá por um torneio. A Figura 7 abaixo mostra que o torneio se inicia primeiramente escolhendo dois genitores aleatórios que podem ser iguais ou não, porém enquanto eles forem iguais deve-se

escolher um novo indivíduo buscando sempre que os dois sejam diferentes. Com os genitores selecionados, verifica-se quem tem o fitness que mais se aproxima de zero, ou seja qual tem melhor qualidade e retorna o vencedor, caso tenham o mesmo fitness, então o primeiro é escolhido e retornado.

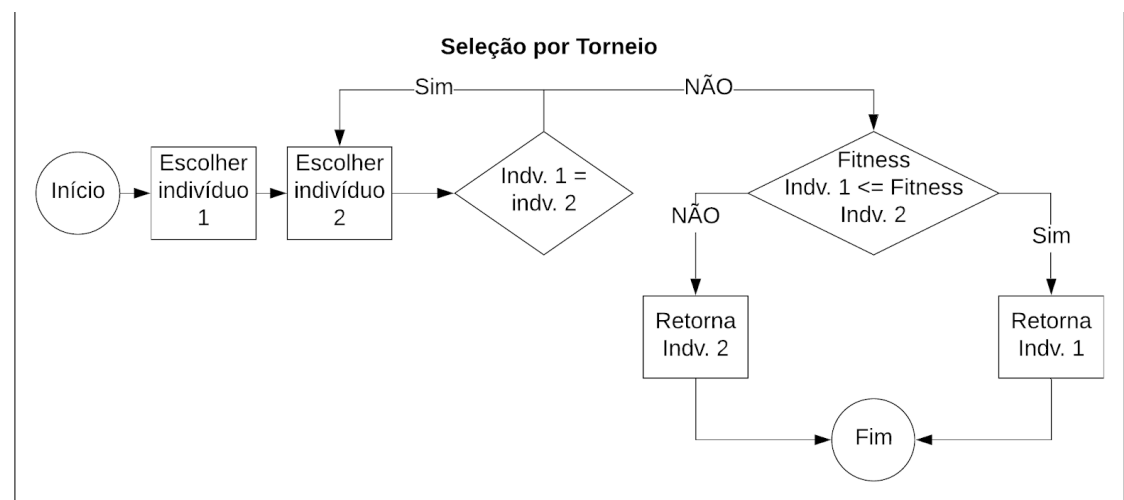


Figura 7. Processo de Seleção por Torneio

3.5. Operadores Genéticos

Nesta seção será abordado e mostrado o passo a passo do desenvolvimento de cada operador genético utilizado no algoritmo.

3.5.1. Recombinação

A figura 8 ilustra o processo de recombinação que é feito utilizando os dois genitores escolhidos nos torneios. Para cada recombinação um novo indivíduo é gerado. Existe um ponto fixo de corte, onde o novo indivíduo recebe a primeira metade do primeiro pai e a segunda do segundo.

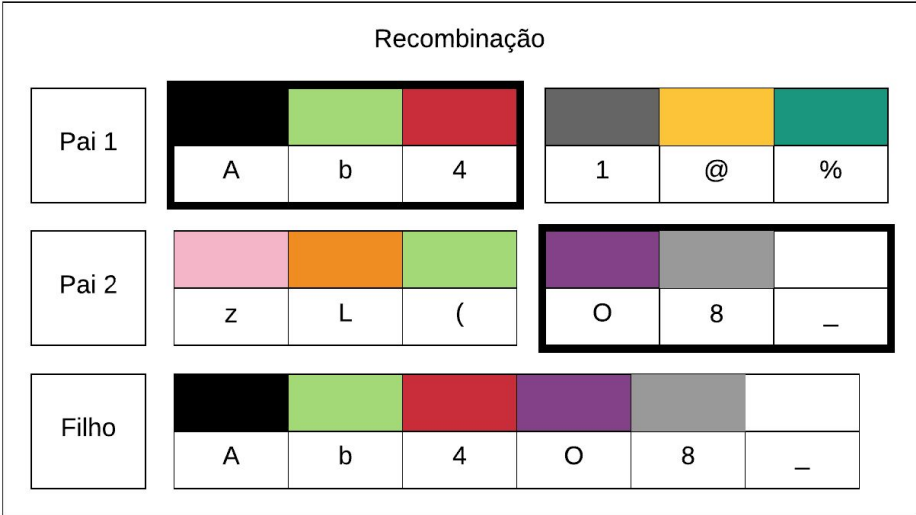


Figure 8. Processo de Recombinação

3.5.2. Mutação

O operador genético responsável pela mutação sempre troca uma cor e seu caractere por novos valores gerados aleatoriamente de forma procedural seguindo alguns critérios para decisão. Essa abordagem da mutação sempre ocorre apesar de normalmente em outros trabalhos serem abordadas como raras, foi devido a necessidade de uma convergência mais rápida. Com uma probabilidade rara de mutação, as soluções poderiam demorar mais para variar, levando assim a um tempo maior de execução para alcançar a convergência. Como podemos observar na Figura 9 a mutação é realizada escolhendo-se aleatoriamente um caractere do indivíduo e trocando seu valor por um novo caractere, após a troca, verifica se a cor associada não é transparente, caso seja, então gera-se um valor entre 1 e 5 para alterar a cor de acordo com os seguintes critérios: caso o valor igual a 1, então a cor torna-se mais escura; igual a 2, torna-se duas vezes mais escura; igual a 3, torna-se mais clara; igual a 4, torna-se duas vezes mais clara e igual a 5 escolhe-se uma cor dentre todas as onze cores presentes no sumário de cores da imagem objetivo. Para tornar uma cor mais clara ou mais escura a classe *Color* da biblioteca *Slick2D* disponibiliza dois métodos para essas finalidades: *Color : brighter(scale : float)* e *Color : darker(scale : float)*, onde em cada um se passa um valor *scale* que pode variar entre 0 e 1, por exemplo se passar 0.03 em um dos dois métodos a cor ficará 3% mais clara ou mais escura.

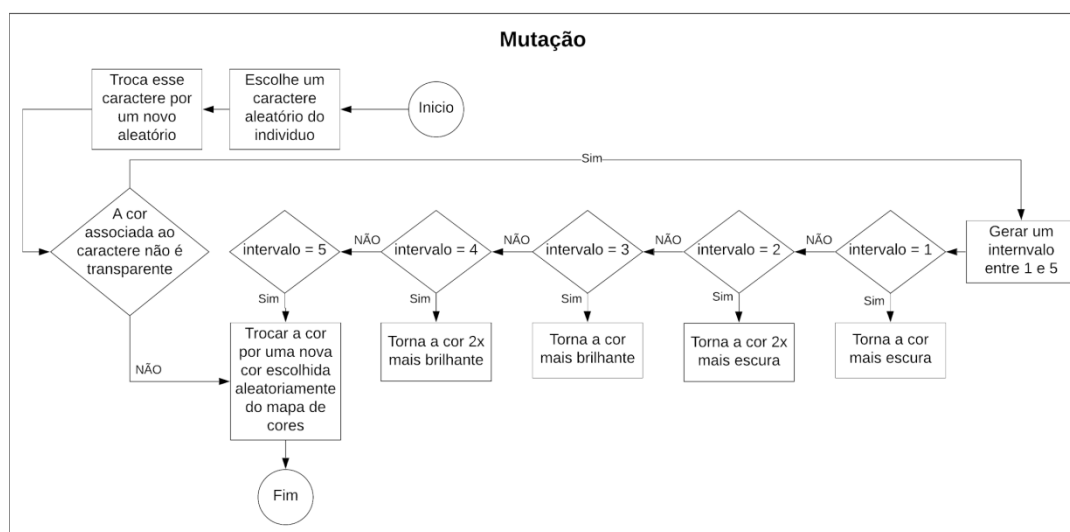


Figura 9. Processo de Mutação

4. Trabalhos Relacionados

[Bergen 2009] apresenta a JNetic, sua ferramenta que utiliza de algoritmos genéticos para gerar imagens utilizando de forma automática e interativa. O usuário pode definir formas geométricas primitivas. O JNetic prover ao usuário um feedback constante e um controle substancial através de uma execução. Podendo editar diretamente um ou mais indivíduos interativamente e apesar do foco do JNetic ser na evolução automática, assim como este presente artigo, ele também permite que o usuário faça interrupções e

interaja guiando o algoritmo à solução desejada. Outra semelhança do trabalho de Bergen com este é o uso de uma paleta de cores, no JNetic o usuário pode criar essa paleta com uma quantidade qualquer de cores e este trabalho tem uma ideia similar, porém a paleta de cores já é pré-definida com 11 cores, cada uma sendo ocupada por uma cor qualquer presente na imagem objetivo. O fitness neste trabalho é calculado da mesma forma que o Bergen utilizou para calcular no seu. O fitness é calculado utilizando a distância euclidiana entre cada pixel da imagem gerada com a imagem original, sendo que neste trabalho esse cálculo é feito em 4-D (vermelho, verde, azul, alpha) e Bergen leva em consideração apenas o vermelho, verde e azul.

[Takeuchi et al 2013] tem um trabalho semelhante a este na forma de gerar ASCII art, Takeuchi et al gera imagens baseadas em tons, diferindo que ele gera em tons de cinza e este trabalho gera imagens coloridas tentando alcançar a melhor fidelidade de cores em relação a imagem objetivo. [Xu et al 2010] apresenta um trabalho sobre ASCII art baseado em estrutura diferindo assim da forma como é abordado por este trabalho e o de Takeuchi. Xu não utiliza tons de cinza para representar seus caracteres, ele utiliza o preto padrão.

5. Resultados

A Figura 10 mostra a grade definida pela distância entre os blocos quadriculares ou caracteres para imagem que será gerada. As distâncias utilizadas foram 10, 8, 5, 4 e 2 pixels de distância. Esses são os valores utilizados no cálculo para definir um conjunto de tamanho proporcional a uma imagem de 200 x 200 levando em consideração a distância entre cada bloco ou caractere. A fórmula utilizada para calcular o tamanho do conjunto de blocos utilizando a distância entre eles é dada pela Fórmula 1 onde w é a largura da imagem, h é a altura e d é a distância. Cada bloco utiliza exatamente uma unidade da grade. A fórmula para calcular o tamanho do conjunto de caracteres levando em consideração a distância é dada pela Fórmula 2. Para evitar muito espaço vazio cada caractere utiliza metade de uma unidade da grade no eixo x, ou seja a distância nesse eixo é dividida por 2 enquanto a distância no eixo y é mantida a original. Todos os resultados podem ser acessados através do link⁴.

$$\text{Fórmula 1: } \frac{(w * h)}{d^2}$$

$$\text{Fórmula 2: } (w * h) * \left(\frac{2}{d^2}\right)$$

⁴ Resultados do TCC: https://drive.google.com/open?id=1mE5adFYj8Lqbm2thcWekBZK1s_EN7x8

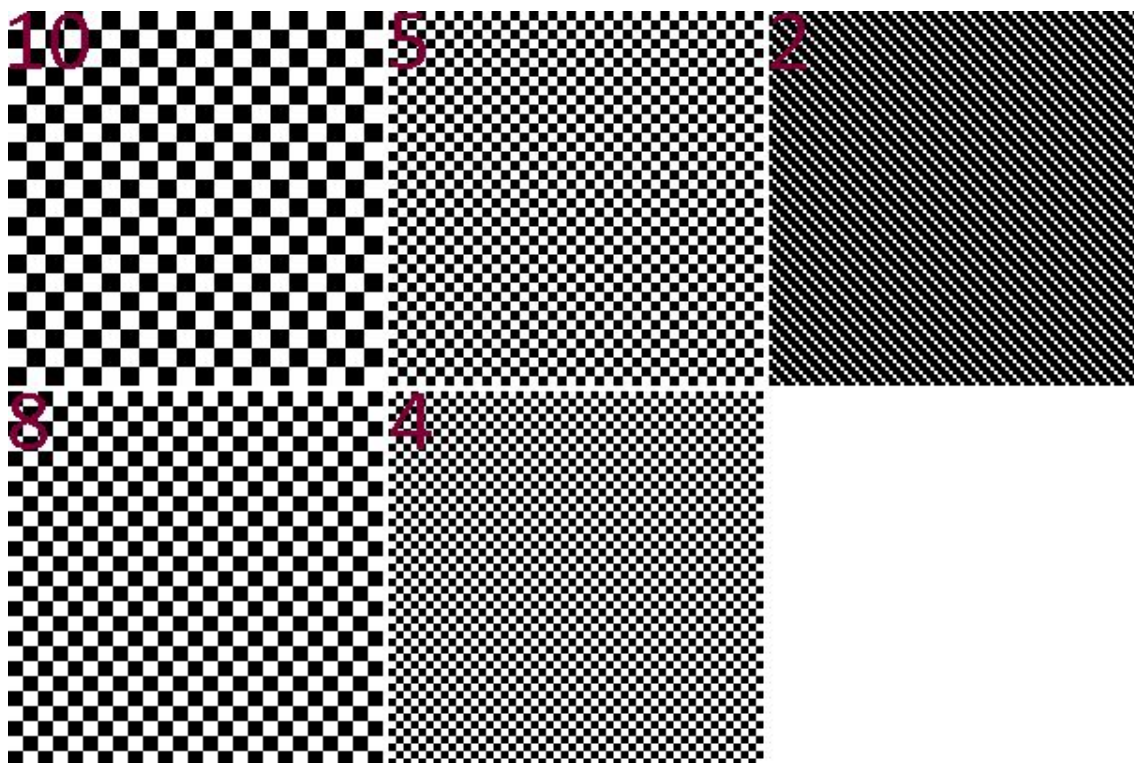


Figura 10. Grades definidas pelas distâncias.

De acordo com a Tabela 1 pode-se aferir que para uma distância de 10 pixels, uma imagem de 20 x 20 blocos ou 40 x 20 caracteres será gerada conforme é mostrado na Figura 10 na grade referente a essa distância. Ainda analisando a Tabela 1 para a distância de 8 pixels será gerada uma imagem formada de 25 x 25 blocos ou 50 x 25 caracteres; para a distância de 5 pixels teremos uma imagem formada de 40 x 40 blocos ou 80 x 40 caracteres; para uma distância de 4 pixels a imagem formada será composta por um conjunto de 50 x 50 blocos ou 100 x 50 caracteres e por último para uma distância de 2 pixels a imagem comportará 100 x 100 blocos ou 200 x 100 caracteres. A quantidade de caracteres no eixo X é sempre o dobro da quantidade de blocos devido ao que foi mencionado anteriormente sobre a Fórmula 2, cada caractere ocupa metade de uma unidade da grade, dobrando assim a quantidade de caracteres em relação aos blocos para o eixo X. Essa abordagem foi utilizada para que houvesse um aproveitamento maior do espaço entre um caractere e outro. Com apenas um caractere utilizando o espaço de uma unidade da grade, havia mais espaço vazio.

Tabela 1. Valores referentes ao cálculo dos conjuntos de bloco e caracteres.

w (pixels)	h (pixels)	d (pixels)	Fórmula 1 (blocos)	Fórmula 2 (caracteres)
200	200	10	400	800
200	200	8	625	1250

200	200	5	1600	3200
200	200	4	2500	5000
200	200	2	10000	20000

Para a obtenção dos dados dos resultados foram executados testes utilizando conjuntos de 800, 1250, 3200, 5000 e 20000 caracteres. Executando cada conjunto por 5, 10, 15, 20, 25 e 30 minutos. O gráfico a esquerda da Figura 12 mostra que para 800 (C800) e 1250 (C1250) caracteres executando por até 30 minutos alcançam fitness máximo de 36,10% e 35,82% de semelhança com a Figura 11. Pode-se observar também que o fitness começa a degradar um pouco ao aumentar para 3200 (C3200), 5000 (C5000), chegando a um alto nível de degradação aos 20000 (C20000) caracteres, demonstrando assim a necessidade de um maior tempo de execução para uma imagem de maior qualidade alcançar um melhor fitness. O aumento da dificuldade do problema é diretamente proporcional a quantidade de caracteres, pois existem mais combinações possíveis, dificultando o alcance de um bom fitness



Figura 11. Imagem utilizada para gerar imagens em ASCII e Pixel art.

Na Figura 12 estão presentes dois gráficos com os resultados baseados nas execuções levando em consideração a Figura 11 que foi utilizada para gerar as imagens em ASCII art e Pixel art. O primeiro gráfico representa o fitness calculado para diferentes conjuntos de caracteres (ASCII art) e o segundo representa o fitness calculado para diferentes conjuntos de blocos (Pixel art) durante tempos de execuções divididos de 5 em 5 minutos variando dos 5 aos 30 minutos para ambos os gráficos.

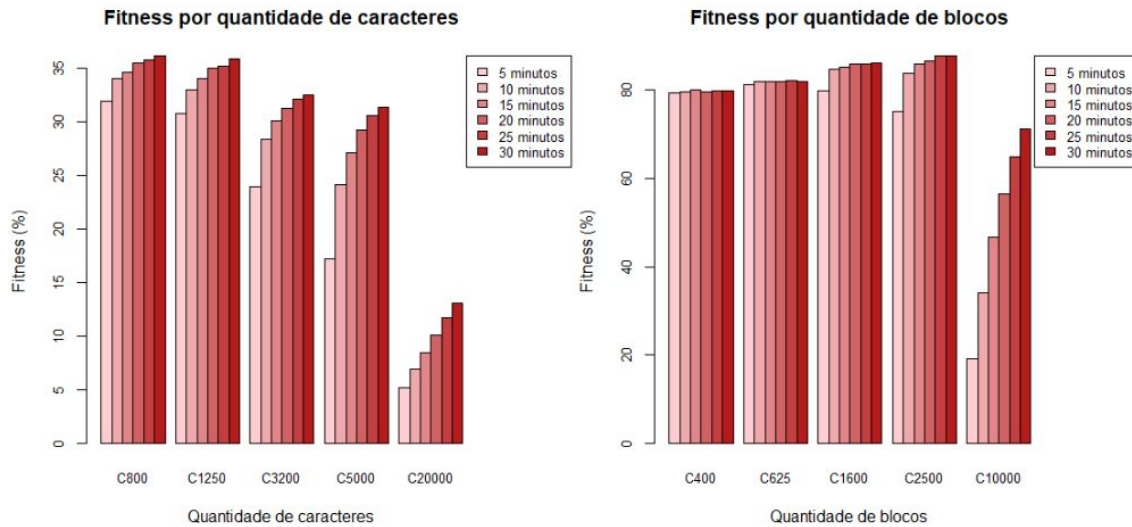


Figura 12. Gráfico de resultados em ASCII art (esquerda) e Pixel art (direita).

A Tabela 2 apresenta o maior fitness alcançado em cada conjunto durante determinado período de tempo no qual o algoritmo ficou em execução. Para os conjunto composto por 800, 1250, 3200 e 5000 caracteres pode-se notar um aumento gradual a partir dos 10 minutos de execução; para o conjunto composto por 20000 caracteres nota-se aumento gradual desde o início da execução. De acordo com a análise estatística da variância de cada conjunto, observa-se que os dois primeiros (800 e 1250) por ter pouca variância, o algoritmo está convergindo, o que pode indicar que não há como melhorar muito mais e que pode ser entregue resultados similares; o conjunto de 3200 caracteres ainda pode variar podendo entregar resultados melhores; o conjunto de 5000 caracteres foi o conjunto que mais variou, indicando assim que o algoritmo pode entregar resultados muito melhores se executado por mais tempo. Por fim, o conjunto de 20000 caracteres teve a 3ª variação mais baixa, o que pode indicar que o fitness ainda vai variar. Diante dessa análise, pode-se concluir que quanto maior a variância, menor a convergência das soluções, sendo assim, significando também que há espaço para melhores soluções.

Tabela 2. Valores referentes ao maior fitness de cada execução em ASCII art

Tempo(m)/ Caracteres	800	1250	3200	5000	20000
5	31.86%	30.77%	23.89%	17.17%	5.13%
10	34.06%	32.91%	28.34%	24.09%	6.92%
15	34.63%	34.00%	30.11%	27.11%	8.44%
20	35.45%	35.02%	31.25%	29.25%	10.07%
25	35.70%	35.16%	32.13%	30.55%	11.70%

30	36.10%	35.82%	32.44%	31.31%	13.08%
Variância	2.38	3.46	10.31	28.05	8.87

A Tabela 3 apresenta o maior fitness alcançado em cada conjunto durante determinado período de tempo no qual o algoritmo ficou em execução. Para os conjunto composto por 400 e 625 blocos não há aumento significativo de fitness para cada um desses conjuntos durante todo o período observado, porém um em relação ao outro houve uma diferença entre a média do fitness de cada conjunto de 2,11%, indicando um aumento pequeno entre o conjunto de 400 e o de 625 blocos; para o conjunto de 1600 blocos tivemos um aumento de 4,95% entre a execução de 5 e 10 minutos desse conjunto variando muito pouco a partir daí. Sua variância indica que se rodado por mais algum tempo haverá uma melhor convergência. Houve um aumento médio de 2,77% do conjunto 1600 em relação ao conjunto 625. Para o conjunto de 2500 blocos houve um aumento de 8,72% em relação ao tempo de execução entre 5 e 10 minutos desse conjunto, mantendo um aumento gradual a partir daí. Sua variância indica que pode haver uma melhor convergência das soluções. Por fim, o conjunto de 10000 blocos que teve a maior variância dentre todos os conjuntos da Tabela 3 o que pode-se dar pela dificuldade do problema de alcançar um bom fitness com essa quantidade de blocos. A dificuldade do problema aqui segue o mesmo raciocínio que foi utilizado para descrevê-lo para os caracteres. A dificuldade é diretamente proporcional a quantidade de blocos.

Tabela 3. Valores referentes ao maior fitness de cada execução em Pixel art

Tempo(m)/ Blocos	400	625	1600	2500	10000
5	79.47%	81.28%	79.77%	75.20%	19.06%
10	79.64%	82.05%	84.72%	83.92%	34.08%
15	80.00%	81.92%	85.28%	85.97%	46.61%
20	79.74%	81.95%	86.04%	86.69%	56.45%
25	79.94%	82.16%	85.99%	87.74%	64.98%
30	79.80%	81.93%	86.16%	87.70%	71.13%
Variância	0.03	0.09	6.04	22.90	384.96

Na Figura 13 podemos observar o melhor resultado de cada execução em ASCII art. No setor A temos os resultados para as execuções utilizando 800 caracteres durante 5, 10, 15, 20, 25 e 30 minutos respectivamente. Os demais setores segue o mesmo padrão de execução sendo o setor B utilizado para mostrar os resultados utilizando 1250 caracteres, o setor C utilizando 3200, o setor D utilizando 5000 e por fim o setor E

utilizando 20000 caracteres. A Figura 13 é um reflexo do gráfico a esquerda presente na Figura 12. No setor E podemos ver que a aproximação das imagens geradas para a imagem original ainda é pouca, porém com uma densidade alta de caracteres que levará a uma imagem mais detalhada com maior qualidade. Por outro lado no setor A se tem uma boa aproximação, porém uma qualidade menor devido à densidade baixa de caracteres.

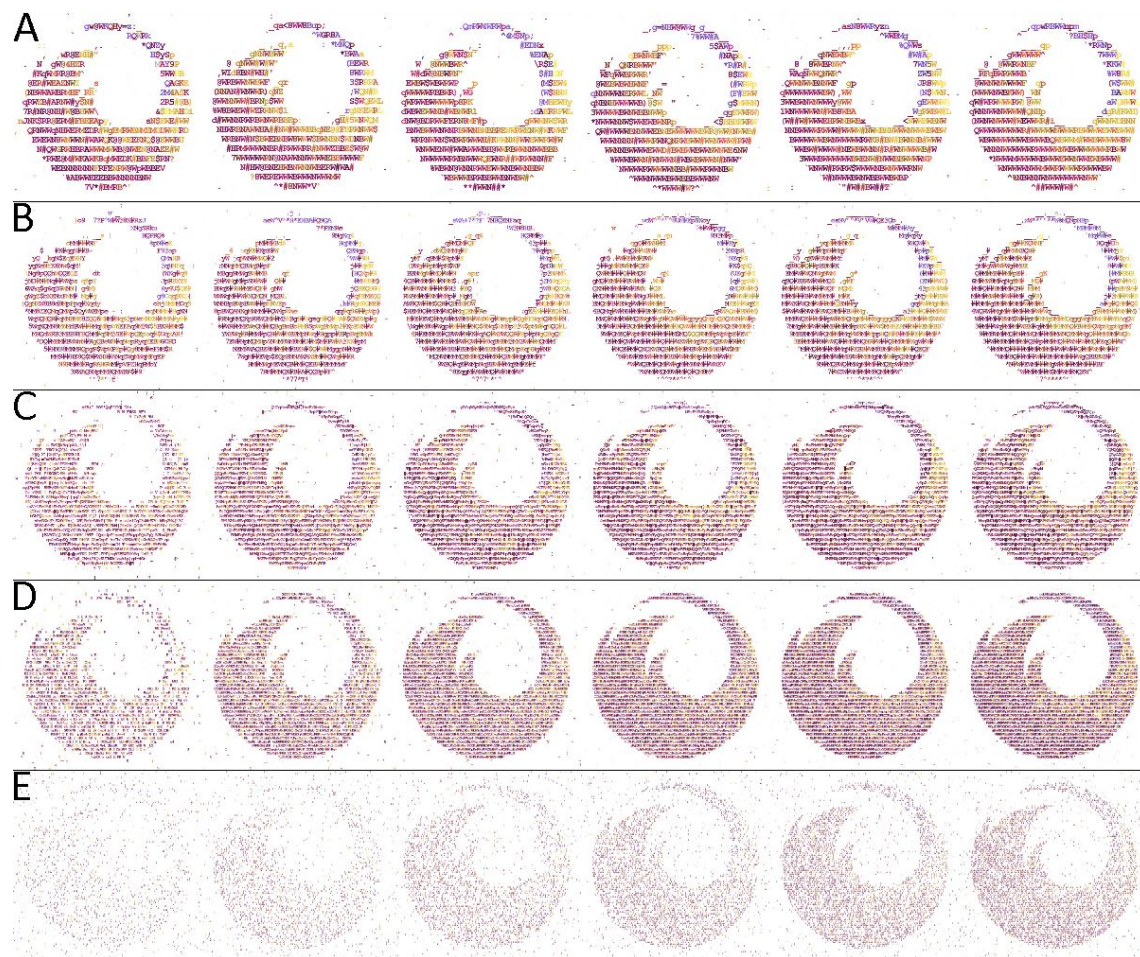


Figura 13. Resultados em ASCII art obtidos durante a execução dos testes

Na Figura 14 podemos observar o melhor resultado de cada execução em Pixel art. No setor A temos os resultados para as execuções utilizando 400 blocos durante 5, 10, 15, 20, 25 e 30 minutos respectivamente. Os demais setores seguem o mesmo padrão de execução sendo o setor B utilizado para mostrar os resultados utilizando 625 caracteres, o setor C utilizando 1600, o setor D utilizando 2500 e por fim o setor E utilizando 10000 blocos. A figura 14 é um reflexo do gráfico à direita na figura 12. Podemos observar que em relação aos resultados em ASCII art os resultados em Pixel art foram muito mais satisfatórios; No setor A com 400 blocos as mudanças foram irrisórias no decorrer do tempo, as cores foram bem definidas, porém devido a baixa densidade de blocos, o nível de detalhamento foi o menor. O setor B com 625 blocos as pontas da imagem começaram a ficar mais definidas, com um pouco mais de blocos. No setor C com 1600 blocos já pode-se observar uma melhora maior no detalhamento, os

resultados tornam-se menos serrilhados devido ao aumento na densidade de blocos, o detalhamento do degradê está mais evidente e as pontas mais definidas. No setor D com 2500 blocos as curvas começam a ficar mais definidas e a transição de cores mais suavizadas. No setor E com 10000 blocos nota-se a evidente necessidade de se executar o algoritmo por mais tempo para alcançar um melhor resultado, mas ainda assim pode-se perceber que as curvas estão muito mais suavizadas, pontas mais definidas em relação aos outros setores.

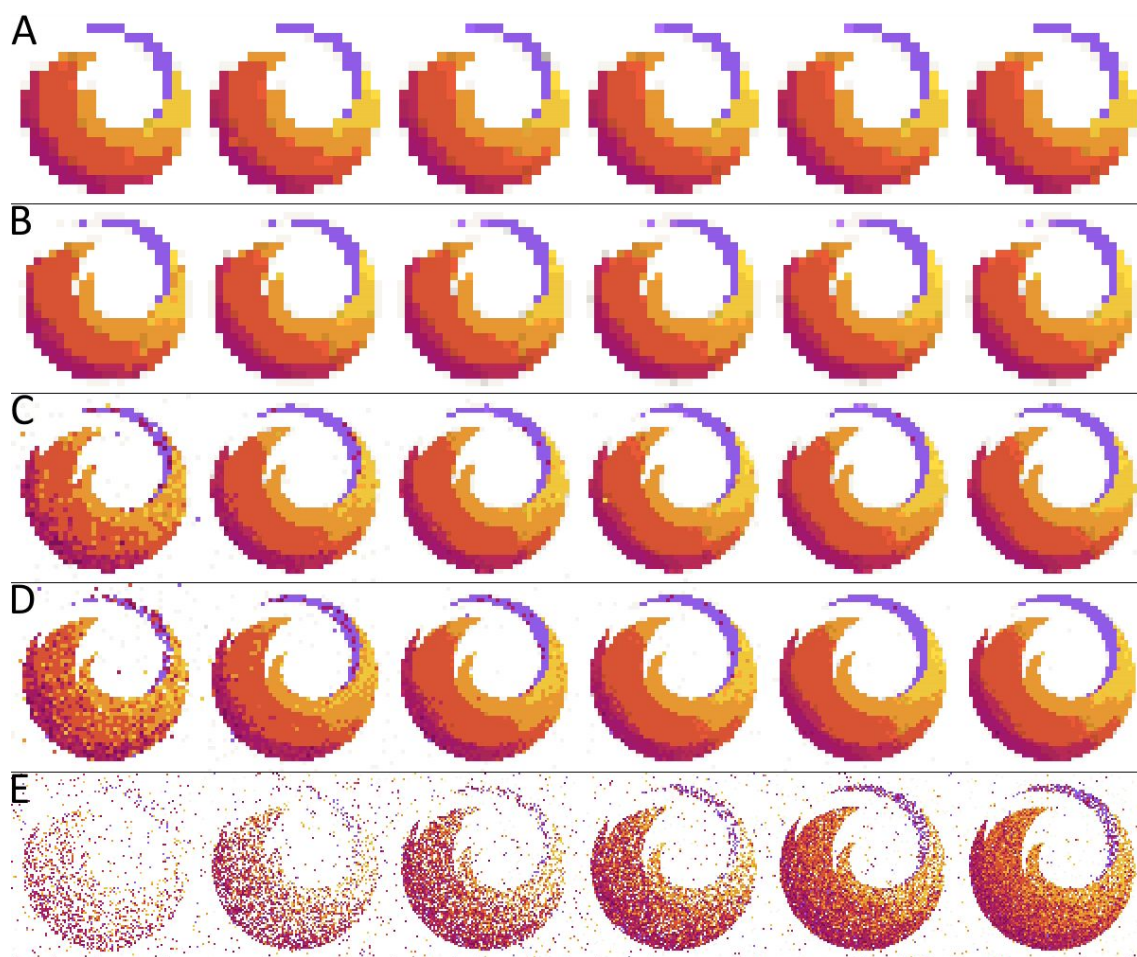


Figura 14. Resultados em Pixel art obtidos durante a execução dos testes

6. Conclusão

Foi realizado um trabalho exploratório para verificar a viabilidade de se utilizar algoritmos genéticos para gerar imagens coloridas em ASCII e Pixel art a partir de uma imagem fonte. Para isso foi implementado um algoritmo genético em Java para produzir esses dois tipos de arte.

Essa implementação demonstrou que em 30 minutos ou menos e utilizando uma densidade de 2500 blocos ou 5000 caracteres pode-se gerar imagens com qualidade satisfatória nos dois formatos propostos neste artigo, para a densidade mais alta de 10000 blocos ou 20000 caracteres, foi identificada a necessidade de um tempo maior devido a dificuldade do problema.

Por fim, para possíveis trabalhos futuros especula-se melhorar a metodologia do cálculo do fitness para se obter um maior desempenho, gerando assim soluções com maior velocidade. Pensa-se também na criação de uma interface usável para usuários externos utilizarem a ferramenta e também a realização de mais testes com diferentes tipos de imagens.

7. Referências

- Bergen, S. "Evolving stylized images using a user-interactive genetic algorithm." *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*. 2009.
- Gen, M, and Lin, L. "Genetic algorithms." *Wiley Encyclopedia of Computer Science and Engineering* (2007): 1-15.
- Lewis, M. "Evolutionary visual art and design." *The art of artificial evolution*. Springer, Berlin, Heidelberg, 2008. 3-37.
- Lima, G. "Uma abordagem para seleção automática de bugs em projetos de software baseados em componentes." (2018).
- Pacheco, M. "Algoritmos genéticos: princípios e aplicações." *ICA: Laboratório de Inteligência Computacional Aplicada*. Departamento de Engenharia Elétrica. Pontifícia Universidade Católica do Rio de Janeiro. Fonte desconhecida (1999): 28.
- Peña, J, and Truyol, A. "Algoritmos genéticos." *Universidad Carlos III* (2007).
- Foster, D. (2019) "Generative Deep Learning: Teaching Machines to Paint, Write, Compose, and Play"
- Xu, X, et al. "Structure-based ASCII art." *ACM SIGGRAPH 2010 papers*. 2010. 1-10.
- Takeuchi, Y, et al. "ASCII Art Generation Using the Local Exhaustive Search on the GPU," *2013 First International Symposium on Computing and Networking*, Matsuyama, 2013, pp. 194-200.