Faculdade de Tecnologia de Americana – Ministro Ralph Biasi

Curso Superior de Tecnologia em Jogos Digitais

Vitor Teixeira de Souza

BacBattle

Simulação de movimento de organismos bacterianos aplicado em jogos digitais

Americana, SP

2019

Faculdade de Tecnologia de Americana – Ministro Ralph Biasi

Curso Superior de Tecnologia em Jogos Digitais

Vitor Teixeira de Souza

BacBattle

Simulação de movimento de organismos bacterianos aplicado em jogos digitais

Relatório técnico desenvolvido em cumprimento à exigência curricular do Curso Superior de Tecnologia em Jogos Digitais sob a orientação do(a) Prof.(a) esp. Antonio Alfredo Lacerda

Americana, SP.

2019

FICHA CATALOGRÁFICA – Biblioteca Fatec Americana - CEETEPS
Dados Internacionais de Catalogação-na-fonte

S719b SOUZA, Vitor Teixeira de

BacBattle: simulação de movimento de organismos bacterianos aplicado em jogos digitais. / Vitor Teixeira de Souza. – Americana, 2019.

38f.

Monografia (Curso Superior de Tecnologia em Jogos Digitais) - - Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza

Orientador: Prof. Esp. Antonio Alfredo Lacerda

1 Jogos eletrônicos 2. Sistemas especialistas I. LACERDA, Antonio Alfredo II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana

CDU: 681.6

Vitor Teixeira de Souza

BacBattle

Relatório técnico apresentado como exigência parcial para obtenção do título de Tecnólogo em Jogos Digitais pelo CEETEPS/Faculdade de Tecnologia – FATEC/ Americana.

Americana, 09 de dezembro de 2019.

Banca Examinadora:

Antonio Alfredo Lacerda
Especialista
FATEC Americana

Jose Mario Frasson Scafi
Mestre
FATEC Americana

Wagner Siqueira Cavalcante
Mestre
FATEC Americana

RESUMO

Essa monografia define um programa para similar os movimentos de uma colônia de bactérias buscando por alimento. Ele segue uma linha de pesquisa em evolução artificial iniciada em 1990 e se baseia em avanços importantes no entendimento das dinamicas de movimentação e cooperação de bactérias alcançados nos ultimos anos. O programa foi feito usando a linguagem de programação Lua, com o *framework* Love2D e algoritmos de movimentação baseados em *Steering Behaviours*.

Palavras Chave: bactéria*, Steering Behaviour*, jogo, simulador.

ABSTRACT

This paper defines a program to simulate the movements of a bacteria colony searching for food. It follows a line of research on artificial evolution that started in 1990 and is based in important advancements on the dynamics of movement and cooperation of bacteria achieved recently. It was made using the Lua Programming language, the Love2D framework and Steering Behavior algorithms.

Keywords: bacteria, Steering Behavior, game, simulation.

SUMÁRIO

1 INTRODUÇÃO 9
2 PROJETO DO JOGO 10
2.1 Evolução artificial 10
2.2 Comportamento do programa 12
2.3 Fluxo do programa 15
2.4 Movimento Browniano 18
2.5 Mapa do Jogo 20
2.6 Cooperação 22
2.7 Criação do menu 25
2.7.2 *Layout* de corte 27
2.7.3 Textos 28
2.7.3 *Sprites* e fontes 29
2.8 Inimigos 30
2.9 Controles 32
3 RESULTADOS OBTIDOS 34
4 CONSIDERAÇÕES FINAIS 36
REFERÊNCIAS BIBLIOGRÁFICAS 37

LISTA DE FIGURAS

Figura 1 - Walter e seu robô autônomo. 9
Figura 2 - Criaturas evoluem através de competição. 10
Figura 3 - Agente escolhe um angulo próximo ao objetivo. 11
Figura 4 - Intervalo de movimentação de uma bactéria. 12
Figura 5 - Algoritmo *Seek*.. 12
Figura 6 - Lógica completa do programa. 14
Figura 7 - Partículas com movimento browniano. 15
Figura 8 - Curva normal padrão. 16
Figura 9 - Labirinto criado por algoritmo de inundação. 18
Figura 10 - Ilustração de veias sanguíneas. 18
Figura 11 - Comportamentos de grupo. 18
Figura 12 - Personagens controlados pelo jogador cooperam para escavar. 19
Figura 13 - Bactérias com maior resistência protegem a colônia. 20
Figura 14 - Tela dividida em uma grade 3x6. 21
Figura 15 - Caixa com a sua altura subdivida por 15%. 23
Figura 16 - Menu do jogo com *Sprite*. 24
Figura 17 - Células B atacam bactérias. 26
Figura 18 - Plaquetas imobilizam bactérias. 27
Figura 19 - Múltiplas bactérias sofrem dano em área. 28
Figura 20 - Menu de atributos lateral. 28
Figura 21 - Caixa de seleção. 29
Figura 22 - Tela final do jogo. 32
Figura 23 - Crescimento exponencial. 32
Figura 24 - A colônia se divide em diferentes direções. 32

1 INTRODUÇÃO

Esta monografia descreve o desenvolvimento de um sistema para simular o movimento e comportamento de um grupo de bactérias usando a linguagem de programação Lua.

O projeto segue uma linha de pesquisa iniciada nos anos 1990 visando simular a evolução de animais usando os recursos da computação gráfica, em especial as pesquisas apresentadas no artigo Evolução artificial para Computação Gráfica (SIMS, 1994). Também há inspiração em jogos como Agar.io, onde jogadores tomam o controle de bactérias com o objetivo de "comer" uns aos outros.

O comportamento do sistema tem como base teórica o curso *Marvels of Bacterial Behavior* ministrado por Howard Berg, professor de biologia da universidade de Harvard. O curso explica em detalhes a maneira como bactérias se movem para procurar nutrientes e os efeitos físicos como o movimento browniano, cuja a implementação no projeto é detalhada na seção 2.4, influenciam no caminho trilhado por elas.

Para animação de particulas, foi-se criado uma implementação de movimento browniano utilizando apenas calculos aritméticos básicos sem operações em vetores.

O projeto contém menus criados a partir de um conceito de grade, dividindo a tela em linhas e colunas. Detalhes deste conceito pode ser visto na seção 2.7.

Visando manter a curva de dificuldade, as bacérias tem que sobreviver há vários inimigos que agem de maneira autonoma para destrui-las. Na seção 2.8 é descrito o padrão de comportamento de cada inimigo.

Espera-se que o sistema seja relevante para pessoas que se interessam em simulações, em particular as que envolvem uma grande quantidade de agentes.

2 PROJETO DO JOGO

2.1 Evolução artificial

Trabalhos relacionados a criação de entidades independentes precedem a ciência da computação. No final dos anos 40 o neurofisiologista Grey Walter construiu o robô autônomo *turtles* (WALTER, 1950) sendo uma das primeiras máquinas a simular um comportamento similar ao de um ser vivo. Na figura 1 pode-se ver o robô *turtles* no canto inferior esquerdo:

Figura 1 - Walter e seu robô autônomo

Fonte: cyberneticzoo (2009)

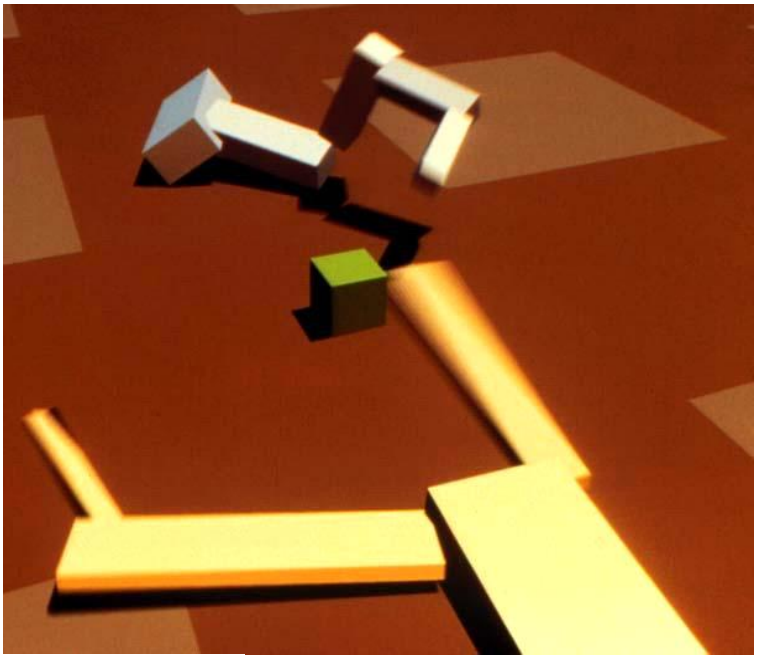
Nos anos 1980 Craig Reynolds criou um Sistema de inteligência de grupo em que várias entidades chamadas *boids* se aglomeram, controlando a densidade e direção do grupo dinamicamente enquanto executam comportamentos de fuga, divergência de obstáculos e busca para aumentar o realismo da simulação. Ao nível individual, a coordenação do grupo é controla por apenas três comportamentos (REYNOLDS, 1999).

Outro exemplo bem conhecido de comportamento emergente é *Conway's Game of Life*. Ele é um *cellular automata* onde um número pequeno de regras simples decide se uma célula vive ou morre, criando movimentos e padrões complexos a cada repetição. O algoritmo tem capacidade de criar entidades

que podem se mover (chamadas de *gliders*) e gerar *gliders* (chamadas de *guns*), isso permitiu Paul Rendell implementar uma *turing machine* utilizando um padrão de *gliders* para criar portões lógicos.

Em 1996, Karl Sims fez um projeto de pesquisa sobre uma simulação de evolução darwiniano com criaturas representadas por blocos virtuais. As criaturas são criadas e avaliadas por uma função de aptidão. Os bem-sucedidos sobrevivem, são combinados e geram uma terceira criatura. A medida que esse processo continua comportamentos complexos emergem após várias interações (SIMS, 1996). A figura 2 mostra duas criaturas competindo por um bloco de alimento:

Figura 2 - Criaturas evoluem através de competição



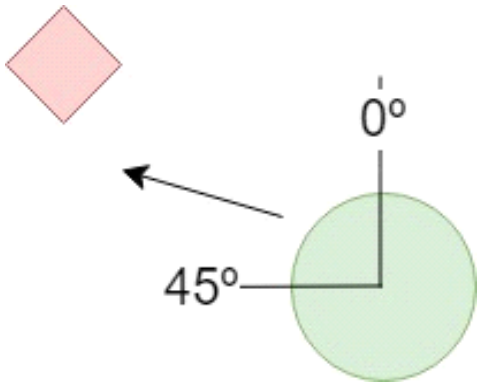
Fonte: Sims (1996)

2.2 Comportamento do programa

O usuário pode interagir com uma única tela, adicionando a ela bactérias e nutrientes. As bactérias se movem em direção aos nutrientes para consumi-los. Cada nutriente aumenta a quantidade de energia da bactéria. Caso a quantidade de energia ultrapasse um certo ponto a bactéria irá se dividir em duas cópias idênticas (reprodução por bipartição). Caso a bactéria fique muito tempo sem absorver nutrientes ela irá desaparecer da tela.

Cada bactéria se move em direção ao alimento através de aproximação. Após detectar um nutriente próximo a ela, é calculado de todos os 360 graus ao redor dela, uma faixa próxima ao seu destino, ela então se move em algum ângulo dentro dessa faixa, escolhido aleatoriamente. Após alguns segundos o processo se repete até que eventualmente ela alcançará o nutriente.

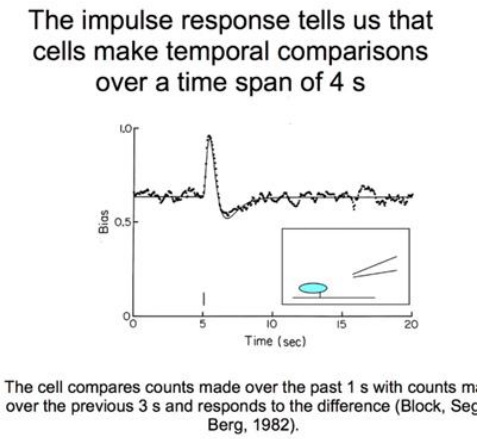
Figura 3 – Agente escolhe um angulo próximo ao objetivo.



Fonte: próprio autor (2019)

O intervalo de tempo entre cada giro da bactéria é de 4 segundos, após rotacional a bactéria continuará nadando em linha reta até esse intervalo estiver se esgotado (BERG, 2013). Na figura 4 os picos no gráfico demonstram variações na trajetória de uma bactéria indicando a rotação:

Figura 4 – Intervalo de movimentação de uma bactéria.



Fonte: Berg (2013)

Esse movimento foi implementado usando um algoritmo do tipo *Steering Behaviour*, uma categoria de algoritmos especializada em simular o movimento de entidades reais como animais, pessoas e veículos através da manipulação de vetores. O algoritmo usado na aplicação é uma variação do *seek*:

> ***Seek*** (ou perseguição de alvo estático) age para guiar o personagem para uma posição específica no espaço global. Esse comportamento ajusta o personagem de modo que sua velocidade está radialmente alinhada com o alvo. Este é diferente de uma força atrativa (como a gravidade) que produziria um movimento orbital em volta do alvo. (REYNOLDS, 1999)

Na figura 5 a linha azul representa o movimento criado pelo *Seek* e o triangulo verde o agente autônomo que executa o movimento:

Figura 5 – Algoritmo *Seek*.

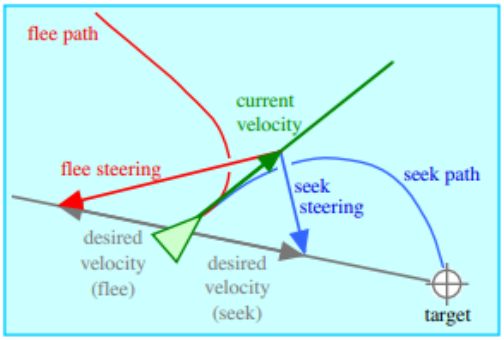


Figure 3: **seek** and **flee**

Fonte: Reynolds (1999)

2.3 Fluxo do programa

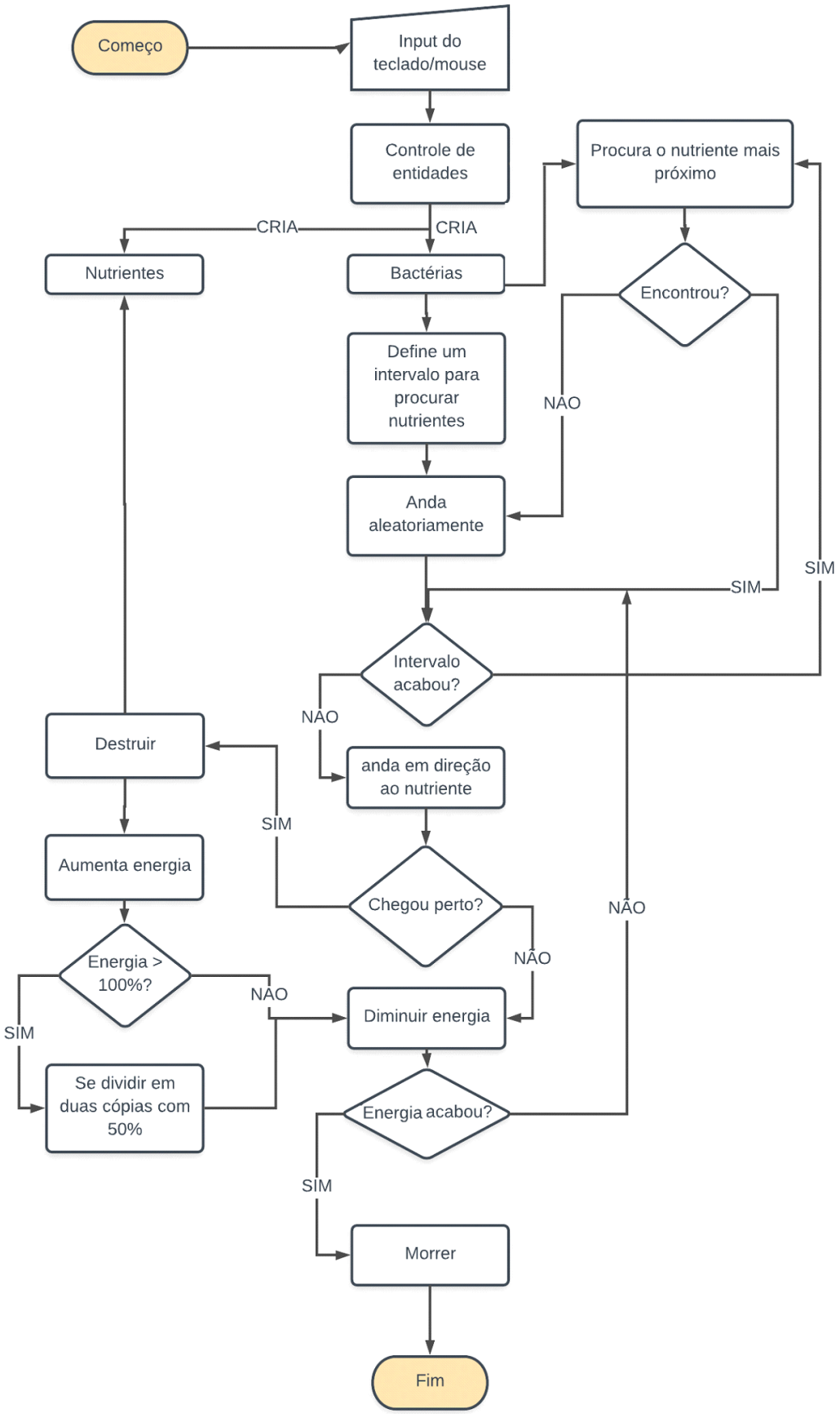
O sistema foi desenvolvido baseado em Löve, um *framework* para a linguagem de programação Lua especializado em jogos 2D.
O *framework* disponibiliza quatro funções principais que beneficiam o projeto:

- load: carregar as imagens, classes e entidades usadas no jogo
- update: atualizar a lógica do jogo
- draw: renderizar imagens e formas geométricas que compõem o gráfico do jogo
- keydown/mousedown: receber *input* do teclado e mouse

Foram criadas duas classes para a Bactéria e o Nutriente. As classes contêm a procedimento necessário para desenhar estes agentes na tela, atualizar a lógica e, no caso da bactéria, gerar movimentação.

Devido ao sistema conter centenas de bactérias, não é prático criar e usa-las diretamente nas funções providas pelo Love. Por isso foi-se criada a classe intermediária EntityManager responsável por criar dinamicamente as bactérias de acordo com os *inputs* recebidos do usuário. Esta classe também media a comunicação entre as bactérias e os nutrientes, dando a cada bactéria a posição do nutriente mais próximo a ela para que ela possa por sua vez executar o algoritmo *Seek* em direção a ele.

Figura 6 – Lógica completa do programa



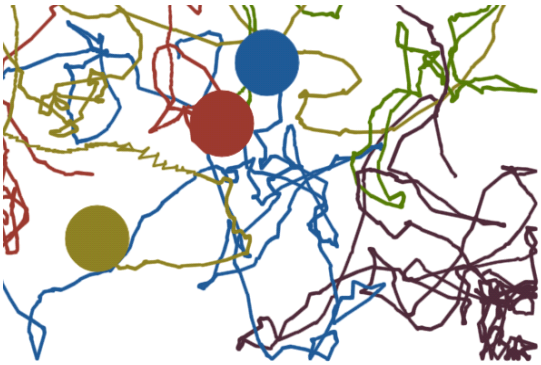
Font

e: Próprio autor (2019)

2.4 Movimento Browniano

Movimento browniano é um distinto comportamento observado em ambientes microscópicos. Partículas suspensas em um fluído se chocam com as partículas que compõem o fluído, gerando um movimento aleatório cujo a trajetória é alterada constantemente. Um exemplo pode ser visto na figura 7:

Figura 7 – Partículas com movimento browniano



Fonte: MinuteLabs (2014)

Pode-se notar que apesar da natureza aleatória de cada movimento, a trajetória final é geralmente contínua.

Essa implementação tem como o objetivo aumentar o número de colisões entre agentes criando uma simulação mais dinâmica com um número maior de interações.

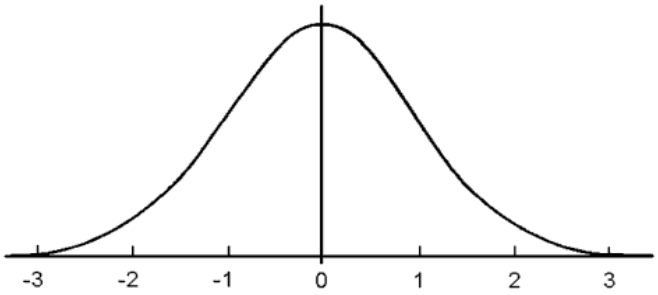
Calcular o choque com centenas de partículas não é eficiente, então, para fins práticos a partícula é considerada como se movendo de maneira independente. Isso pode ser executado usando uma lista de pontos aleatórios que o agente percorre usando um algoritmo simples de interpolação:

```
newX, newY = x + normal_rand(dist), y + normal_rand(dist)

seek(newX, newY, acce)
```

Para a geração de números aleatórios utilizou-se um número escolhido a partir de uma curva normal, também chamada de curva gaussiana, representada na figura 8:

Figura 8 – Curva normal padrão



Fonte: Mathcenter (2015)

Ela é uma maneira de distribuir números onde a maioria dos números tem valores próximo a um ponto central e poucos números tem valores mais distantes desse centro. A curva normal é vista em várias situações na natureza, fazendo com que seu uso crie movimentos mais realistas.

O parâmetro **dist** é a distância percorrida, ele representa a densidade do fluído. Fluídos com poucas partículas permitem ao agente se mover por mais tempo sem mudar de direção. O parâmetro **acce** é a aceleração do agente, representa o tamanho das partículas do fluído, grandes partículas criam um maior impacto e maior aceleração.

O algoritmo acima funciona de maneira satisfatória se a pressão no ambiente simulado for constante, porém falha ao simular agentes se movendo em direções particulares devido a diferenças de pressão entre uma área e outra. Uma solução é criar um vetor adicional apontando para a área de baixa pressão, comprometendo a natureza aleatória do movimento.

A solução proposta consiste em aumentar o parâmetro **dist**, permitindo o agente se mover por mais tempo quanto menor for a pressão o que ao longo de várias iterações cria uma trajetória que atraí o agente para áreas de baixa densidade.

2.5 Mapa do Jogo

O mapa foi criado de utilizando uma abordagem procedural:

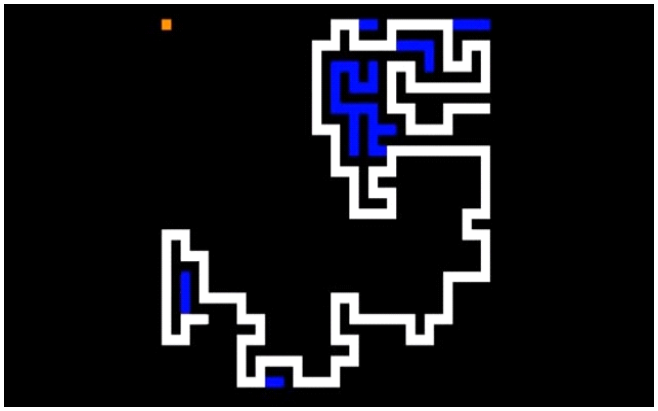
> Em termos simples, geração procedural significa que os dados ou conteúdo são gerados por uma lógica / algoritmo, em vez de um humano criando o conteúdo manualmente.
> [..]. Alguns jogos só são possíveis via geração procedural. O jogo cria o ambiente dinamicamente a medida que o jogador avança pelo jogo, criando uma experiência imersiva.
> [..]A geração procedural tem um elemento de aleatoriedade. O algoritmo leva em conta uma série de regras e irá gerar o conteúdo automaticamente para você (SHIVANG, 2018).

Foi usado como inspiração o algoritmo de inundação. Ele é um dos muitos algoritmos usados para geração procedural de labirintos, sendo particularmente popular em comparação com outros algoritmos disponíveis devido a sua simplicidade. Os passos do algoritmo são descritos a seguir:

- Imagina-se o plano onde vai ser criado o labirinto como um quadriculado de células.
- Começa o programa por uma célula aleatória dentro desse quadriculado.
- Escolhe-se uma célula adjacente que ainda não foi visitada.
- Remove a parede entre essa célula e atual e marca ela como visitada.
- Caso a célula atual seja uma em que todos seus vizinhos já foram visitados ele volta o caminho já percorrido até encontrar uma célula com vizinhos não visitados.
- Repetem-se os passos anteriores até não haver nenhuma célula não visitada.

Na figura 9 a área em preto representa as paredes do labirinto, as linhas em branco o labirinto em si e as linhas azul são o momento que o algoritmo voltou por ter chegado a uma célula sem vizinhos:

Figura 9 - Labirinto criado por algoritmo de inundação



Fonte: Wikimedia Commons (2014)

O algoritmo implementado no projeto difere do algoritmo de inundação por ser linear, isto é, a célula nunca volta para trás. Ao invés disso a célula cria outra célula independente que segue seu próprio caminho formando um padrão fractal. Um fractal é uma forma geométrica repetida infinitamente. Cada contida no todo tem a mesma forma que o todo e qualquer outra parte. Fractais criam padrões de repetição e ramificação que se assemelham a formas vistas na natureza (MARION, 2018).

O propósito de unir esses dois conceitos foi criar uma rede ramificada similar a capilares ou artérias para servir como ambiente para os microrganismos do jogo. Contendo tanto o padrão de repetição dos fractais em conjunto com a necessidade de preenchimento de todo espaço disponível característica de labirintos, sem que duas linhas ocupem o mesmo espaço. A figura 10 mostra um exemplo do resultado esperado para tal algoritmo:

Figura 10 - Ilustração de veias sanguíneas

Fonte: Medical News Today (2018)

2.6 Cooperação

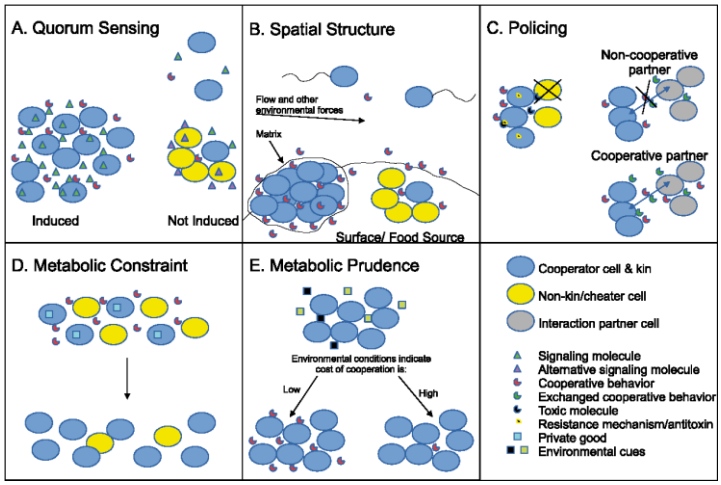
Pesquisas recentes apontam que bactérias possuem comportamentos de coordenação em grupo e divisão de tarefas superiores ao que se previamente pensava, de acordo com Sørensen:

> Bactérias levam nosso entendimento de sinergia de grupo e inspiração para um nível completamente diferente. Elas introduzem atributos nos seus vizinhos que normalmente estariam dormentes. Dessa maneira grupos de bactérias podem expressar propriedades que não são visíveis quando elas estão sozinhas.
>
> Quando elas estão juntas propriedades totalmente novas podem emergir. (SØRENSEN, 2019)

Os pesquisadores estudaram bactérias vivendo dentro de um biofilme, uma massa gelatinosa que protege as bactérias de predadores. Bactérias presentes nesse ambiente não competem por recursos, mas trabalham em conjunto delegando tarefas umas para as outras e desligando mecanismos do seu organismo para poupar energia.

Na figura 11 pode ver ilustrações de diferentes comportamentos em grupo de uma colônia de bactérias. Os círculos em azul representa uma única colônia da mesma espécie. Em B temos o comportamento de aglomeração e em E de troca de nutrientes:

Figura 11 - Comportamentos de grupo



Fonte: Sørensen (2019)

Pode-se fazer um paralelo desse comportamento natural com mecânicas características de um gênero de jogos conhecido como administração de

colônia, que por sua vez é um subgênero de Simulação de Construção e Administração. Onde o objetivo do jogo é delegar tarefas para vários agentes com o objetivo de completar uma tarefa (ROLLINGS, 2003).

Um exemplo recente de uma obra nesse gênero é o jogo *Oxygen not included*, visto na figura 12, onde o jogador controla um grupo de pessoas transportados para um planeta distante que tem como objetivo obter recursos para sua sobrevivência e construir uma base espacial:

Figura 12 - Personagens controlados pelo jogador cooperam para escavar

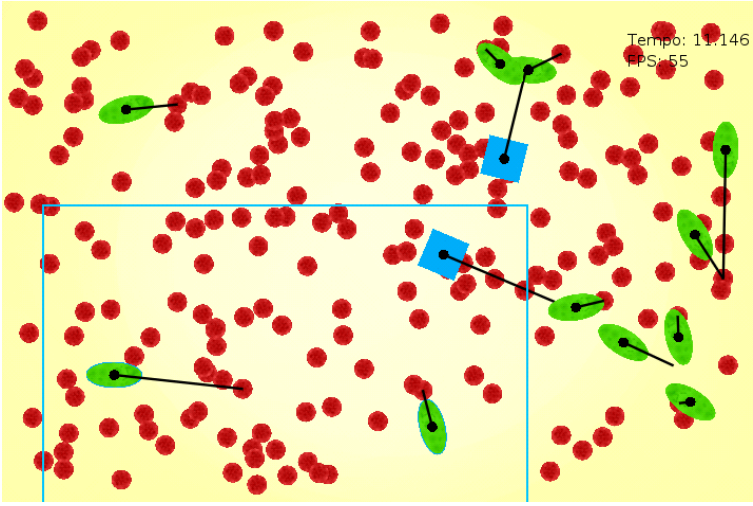
Fonte: Jogorama (2016)

O projeto implementou o conceito de cooperação visto nos jogos de administração visando simular o comportamento de organização bacteriano. Uma característica em comum de todos esses jogos é que os papeis desempenhadas pelos personagens bem como suas características são pré-determinados pelo desenvolvedor com o jogador meramente tendo o papel de decidir quando assignar um personagem para uma determinada tarefa.

No projeto, tais papeis são determinados pelo próprio jogador. Pode-se distribuir os atributos da bactéria em várias categorias diferentes fortalecendo uma em troca de outra. Esse grupo de atributos é chamado de DNA. Um grupo de bactérias pode ser atribuído um DNA com o intuito de desempenhar uma tarefa especifica. O jogador escolhe um genoma clicando na lista no canto direito inferior e arrastando o mouse para criar uma caixa de seleção em um grupo de bactérias.

Dois objetivos principais são a proteção da colônia e a coleta de nutrientes. Bactérias pode-se melhorar a habilidade de absorver nutrientes se movendo rapidamente, porém tornando-se vulneráveis a ataques, ou melhorar a resistência a toxinas e danos porém movendo lentamente. Na figura 13 os quadrados azuis são os inimigos, células do sistema imunológica que querem destruir a colônia. Duas bactérias são escolhidas para terem maior resistência e se posicionar na frente delas, agindo como "guardas":

Figura 13 - Bactérias com maior resistência protegem a colônia



Fonte: próprio autor (2019)

2.7 Criação do menu

Essa seção relata a criação do *layout* do menu. Por *layout* entende-se a maneira em que diferentes elementos são posicionados na tela.

O *layout* do projeto utiliza o conceito de grade. Primeiro é escolhido uma área retangular da tela, essa área é um objeto com 4 valores: uma posição x e y no sistema de coordenadas cartesiano, uma altura e um comprimento. No exemplo escolhido usamos as medidas da própria tela como área:

```
windowBox = {x= 0, y= 0, w= window.getWidth(), h=
window.getHeight()}
```

Essa área subdivida em linhas e colunas criando uma grade, podemos posicionar elementos dentro das células dessa grade.

Para que os elementos não fiquem rentes uns aos outros pode-se opcionalmente definir um espaçamento entre a célula da grade e o elemento. A sintaxe é a seguinte:

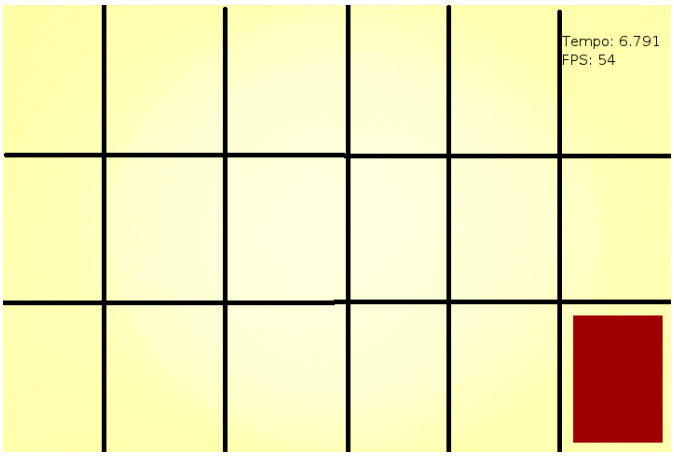
```
grid(<area_retangular>, <colunas>, <linhas>, <coluna>,
<linha>, <espaçamento>)
```

No trecho de código a seguir dividimos a janela em uma grade 3x6 e posicionamos o elemento na última célula com um espaçamento de 10% da largura do elemento:

```
Menu1 = grid(windowBox, 6, 3, 6, 3, 10)
```

Na figura 14 as linhas e colunas da grade estão demarcadas com uma linha preta:

Figura 14 - Tela dividida em uma grade 3x6



Fonte: próprio autor (2019)

A vantagem desse sistema é que elementos estão sempre relativos à área em que eles ocupam. Quando as dimensões dessa área são alteradas as dimensões dos seus elementos também mudam de maneira proporcional.

2.7.2 *Layout* de corte

Para criar elementos relativos que não se encaixam dentro de linhas e colunas usamos o conceito de cortes, uma caixa tem uma de suas medidas alterada por uma porcentagem criando uma nova caixa relativa a ela.

Nesse exemplo temos um menu que é posicionado no centro da tela dentro de uma grade 1x1 com um espaçamento de 15%:

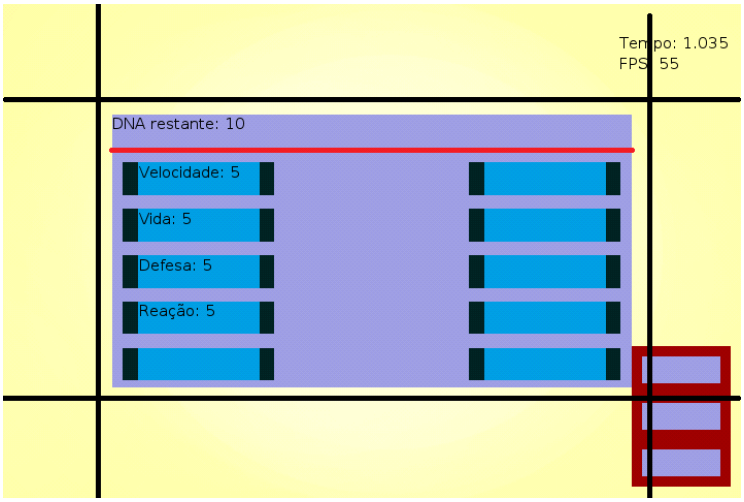
```
menu2 = grid(windowBox, 1, 1, 1, 1, 15)
```

Esse menu tem seu 15% da sua altura cortada para criar um cabeçalho e os outros 85% são usados parar criar outro grid onde são encaixados botões. Usa-se a seguinte função:

```
header = fromBox(menu2, {h = {'%', 15}})
```

Na figura 15 a linha vermelha representa o espaço de 15% removido do menu e reservado para o cabeçalho:

Figura 15 - Caixa com a sua altura subdivida por 15%



Fonte: próprio autor (2019)

Usando porcentagens as divisões do menu estarão sempre proporcionais independe da medida real do mesmo.

2.7.3 Textos

Quando os valores do jogo são alterados todos os textos contendo esse valor são alterados para refletir o novo valor atualizado. Isso foi implementado usando o padrão de projeto *observer*.

De acordo com Zhart, Padrões de projeto são soluções típicas para problemas comuns encontrados em arquitetura de *software*. Essas soluções são apenas conceitos e a solução real feita em código pode ser diferente dependendo do contexto em que ela é aplicada. (ZHART, 2019)

O padrão de projeto *Observer* é definido da seguinte forma:

> O objetivo do padrão é definir uma relação de um-para-muitos de tal maneira que, quando um objeto muda seu estado, outros objetos são notificados e atualizados automaticamente. Informando-os de eventos que acontecem no sistema. (POYIAS, 2019)

No projeto esse padrão foi implementado com duas funções. A primeira função cria uma expressão contendo uma variável, o nome da variável é inserido no espaço entre chaves '{{ }}':

```
bindText(button2, "Vida: {{maxHealth}}")
```

Dentro da função bindText essa expressão e o objeto que contém ela é atrelada ao nome da variável e salvo em um objeto global chamada LISTENERS:

```
LISTENERS[valor][] = {obj, expr}
```

Quando é necessário atualizar a variável, usamos a função change(<nome do valor>, <valor>):

```
change(maxHealth, 10)
```

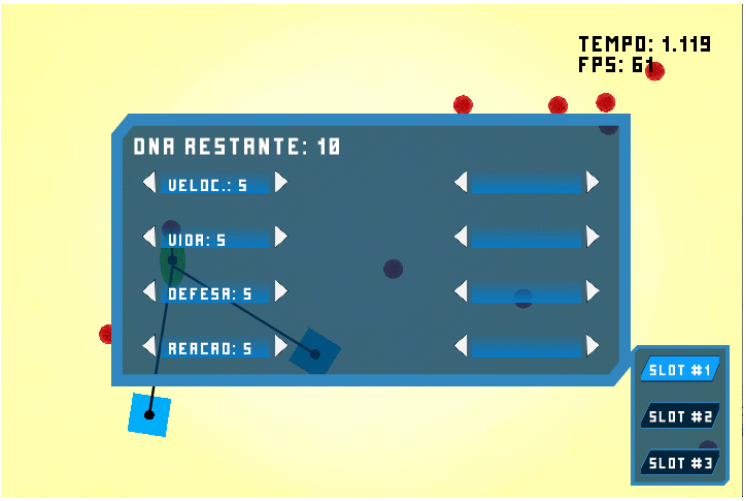
Dentro da função change() é procurado se há algum objeto atrelado a essa variável e se houver texto dele é atualizado para refletir o novo valor:

```
LISTENERS[valor].obj.text = parseExpr(obj.expr, maxHealth)
```

2.7.3 *Sprites* e fontes

Os menus foram desenhados usando a versão 0.92 do software de edição de imagem Inkscape. Os textos foram renderizados com a fonte Future Millenium escolhida por ser monoespaçada e quadrada. Como mostra a imagem da tela na figura 16:

Figura 16 - Menu do jogo com *Sprite*



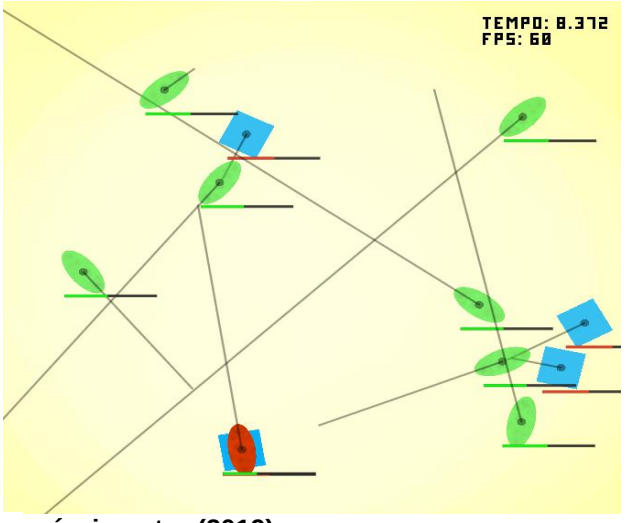
Fonte: próprio autor (2019)

2.8 Inimigos

Durante o decorrer do jogo o jogador deve sobreviver há inimigos vindos fora da tela com o objetivo de destruir as bactérias. O comportamento de cada um deles é descrito a seguir:

O primeiro inimigo é uma célula B. Células B perseguem uma bactéria de cada vez atacando-a diretamente. Ela usa o mesmo algoritmo de comportamento da bactéria: procurando a bactéria mais próxima, se movendo em direção a ela e removendo pontos da vida dela caso ambos entrem em contato.

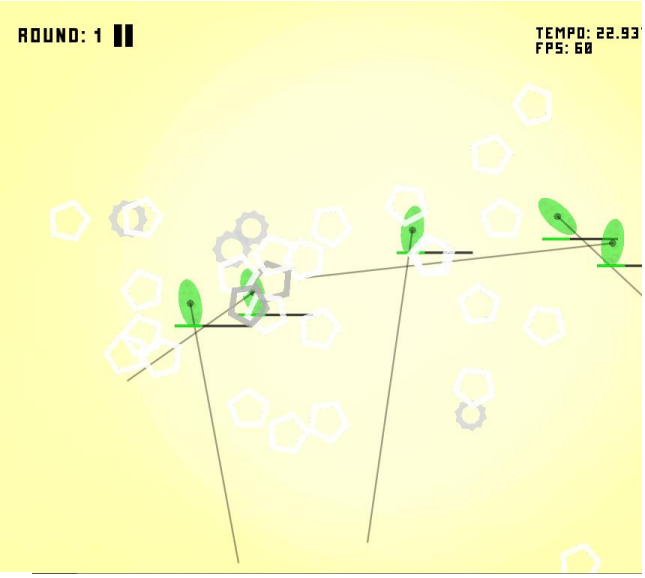
Figura 17 – Células B atacam bactérias



Fonte: próprio autor (2019)

O segundo inimigo é um Magacariócito. Magacariócitos são células responsáveis pela produção de plaquetas. As plaquetas são partes da superfície da célula que de descamam dela. No jogo essa célula cria plaquetas e as lança em uma direção aleatória, ficando menor a cada plaqueta lançada até desaparecer por completo. Quando uma bactéria cruza com uma plaqueta ela se torna incapaz de se mover, consequentemente morrendo por falta de nutrientes ou por um ataque. Na figura 18 pode-se ver as plaquetas presas em uma bactéria destacadas em cinza:

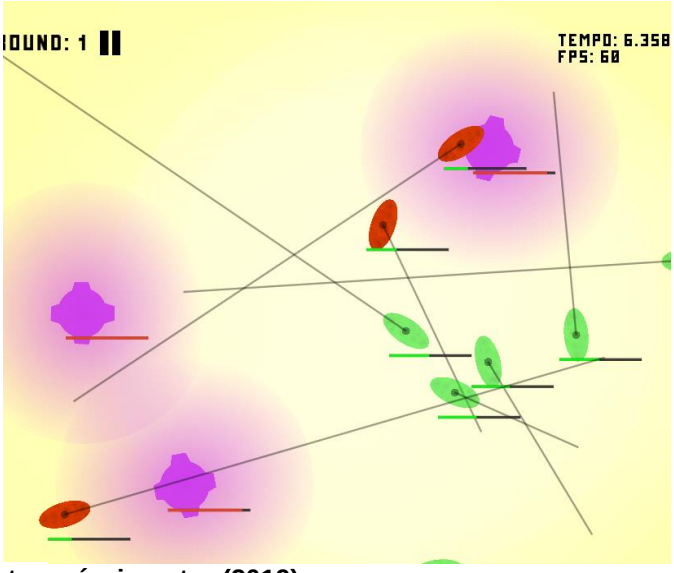
Figura 18 – Plaquetas imobilizam bactérias



Fonte: próprio autor (2019)

O terceiro inimigo foi criado com a intenção de atacar largos grupos de bactérias concentradas em um só lugar. Para tal fim ele foi projetado para causar dano continuo a todos as bactérias dentro do raio indicado pelo círculo roxo. Toda vez que esse inimigo ataca ele perde uma pequena quantidade de pontos de vida, criando assim uma janela de tempo em que o jogador deve sobreviver a ele. Ele pode ser visto em roxo na figura 19:

Figura 19 – Múltiplas bactérias sofrem dano em área



Fonte: próprio autor (2019)

2.9 Controles

Ao lado direito do jogo há um menu onde são alterados os atributos das bactérias. Na figura 17 os botões nomeados *SLOTS* de 1 ao 3 alternam entre os três grupos de configuração disponíveis.

Ao clicar em um deles os valores desse grupo é carregado na lista a esquerda da figura 17. A função de cada atributo é a seguinte:

- Velocidade: Determina o quão rápido a bactéria se move em direção a um objetivo (ex. comida).
- Vida: Numero representando a saúde máxima que uma bactéria pode atingir.
- Defesa: Valor que é subtraído de todos os ataques recebidos pela bactéria. Aumentando o tempo de sobrevivência dela.
- Absorção: Número de pontos de saúde que a bactéria recebe ao comer um nutriente.

O valor de cada atributo pode ser aumentado clicando no botão '+' à direita e reduzido clicando no botão '–' à esquerda.

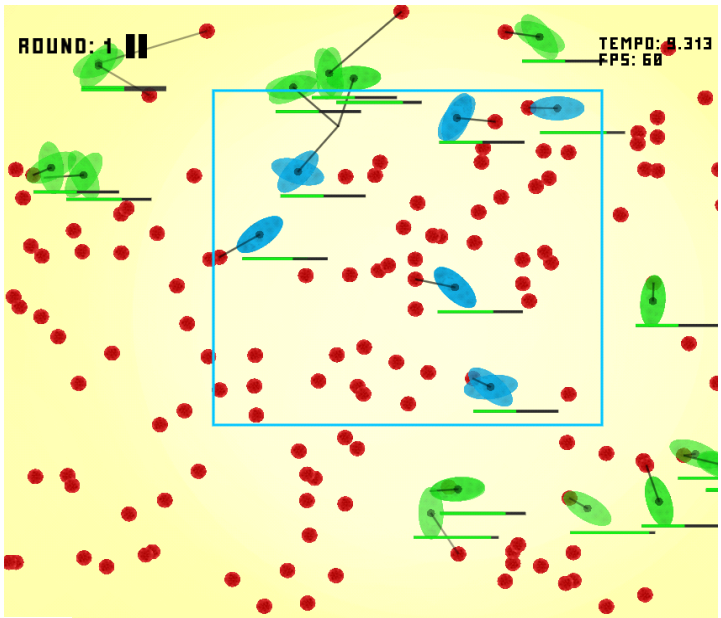
Na figura 20 pode-se ver o menu editado em duas colunas para melhor visualização:

Figura 20 – Menu de atributos lateral

Fonte: próprio autor (2019)

Esses atributos são atribuídos as bactérias clicando e arrastando com o botão direito do mouse para criar uma caixa de seleção. A cor da caixa é correspondente com a cor do botão do *slot* sendo aplicado, como visto na figura 21:

Figura 21 – Caixa de seleção

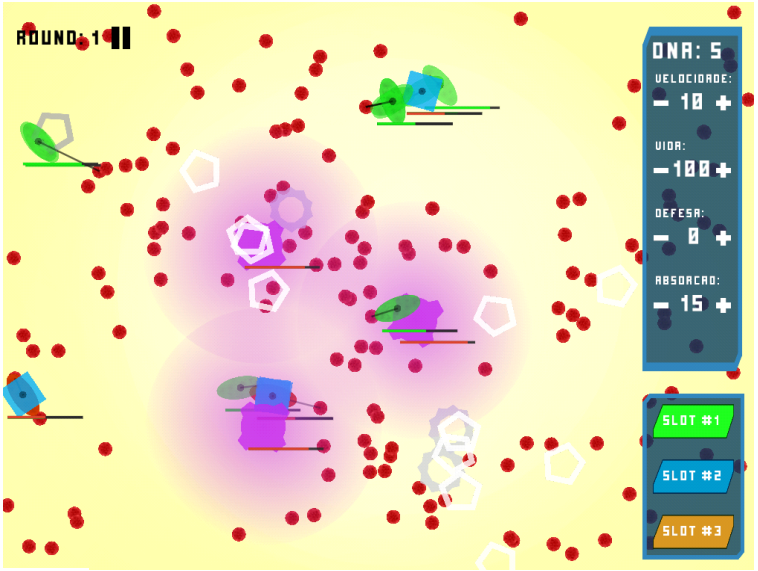


Fonte: próprio autor (2019)

3 RESULTADOS OBTIDOS

Na figura 22 pode-se observar o resultado final da tela do jogo, com o menu lateral à direita e os inimigos descritos na seção 2.8 visíveis no mapa:

Figura 22 – Tela final do jogo



Fonte: próprio autor (2019)

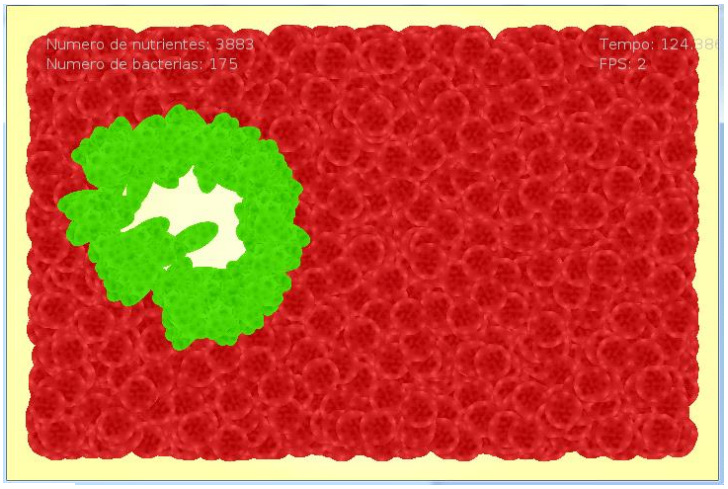
O projeto final atendeu as expectativas inicias, com cada bactéria buscando por comida se o ambiente estiver estéril e quando inserido um nutriente, o detectando após um pequeno intervalo de tempo e alterando sua rotação e movimento na direção dele.

Cada bactéria consome sua energia, desaparecendo do ambiente após o valor da sua energia chegar a zero.

Bactérias aumentam sua energia absorvendo nutrientes e, caso esta chega a um nível máximo, se dividem em dois clones, cada um possuindo metade da energia da bactéria original.

Também pode-se observar comportamentos de grupo como rápido crescimento exponencial e uma colônia se dividindo em subgrupos para obter nutrientes em regiões diferentes, visto na figura 23:

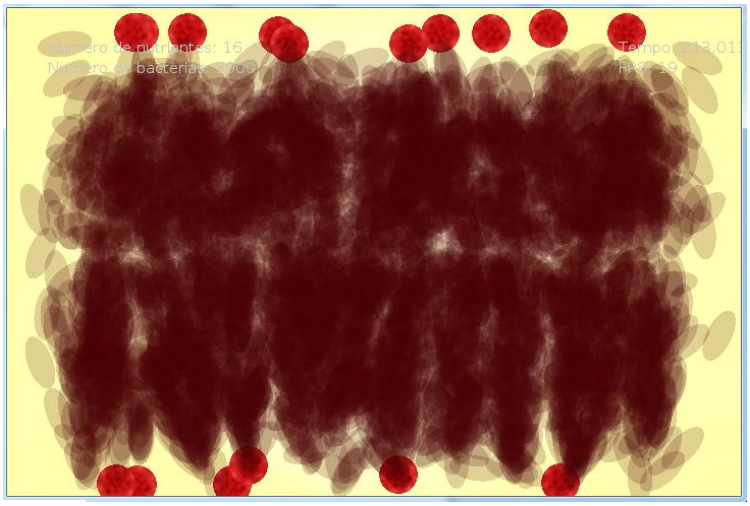
Figura 23 – Crescimento exponencial



Fonte: próprio autor (2019)

Quando os nutrientes estão distribuídos em diferentes extremidades pode-se observar a colônia se repartindo em diversos subgrupos direcionados a região mais próxima que contém nutrientes:

Figura 24 – A colônia se divide em diferentes direções



Fonte: próprio autor (2019)

A performance obtida foi de 60 *fps* e para 5.000 bactérias sendo simuladas simultaneamente 10 *fps* para 34.000 bactérias. Medido em um computador com processador Intel core i5-7400 com 3.00 GHz, 8 Gigabytes de memória RAM e Windows 10 64 bits.

4 CONSIDERAÇÕES FINAIS

O jogo foi criado baseado em uma *engine* própria, utilizando apenas uma biblioteca de renderização gráfica para mostrar elementos na tela. Isso teve a vantagem de criar um projeto mais customizado e removeu a necessidade de lidar com frameworks complexos com longas curvas de aprendizado, mas também significou que várias funcionalidades como detecção de colisão foram implementadas manualmente o que consumiu tempo de desenvolvimento. É incerto se isso foi a melhor decisão ou se o projeto se beneficiaria em ser produzido usando um *framework* de produção de jogos tradicional.

O desenvolvimento do sistema de layout de menu provou ser mais difícil do que o esperado e não foi justificado devido à baixa quantidade de menus no jogo, em retrospectiva seria melhor usar métodos procedurais tradicionais para tal fim.

O escopo do projeto foi reduzido quanto ao mapa. A proposta inicial era a criação de um simulador completo incluindo um ambiente navegável e interativo, porém limitações quanto o tempo necessário para transformar o algoritmo de criação de mapa em um ambiente jogável e a performance necessária para processar as ações de agentes fora da tela do jogo resultaram na mudança para um ambiente estático onde os inimigos vêm em direção ao jogador.

No jogo há apenas uma colônia de bactérias criada pelo jogador. Uma abordagem mais realista contaria com a presença de várias colônias similares, mostrando comportamentos de competição de colaboração entre diferentes espécies de bactérias.

REFERÊNCIAS BIBLIOGRÁFICAS

ALVES, LEANDRO. ***Oxygen Not Included* é o mais novo jogo dos criadores de *Don't Starve***. 2016. Disponível em <https://jogorama.com.br/noticias/10614/oxygen-not-included-e-o-mais-novo-jogo-dos-criadores-de-dont-starve/>. Acesso em 2 de nov. 2019.

BERG, Howard**. *Marvels of Bacterial Behavior*** (Maravilhas do Comportamento Bacteriano). 2013. Disponível em: <https://www.youtube.com/watch?v=ioA1yuIA-t8%22atch?v=ioA1yuIA-t8"A-t8>. Acesso em maio/2018.

CYBERNETIC ZOO. ***CORA the tortoise***. Disponível em: <http://cyberneticzoo.com/cyberneticanimals/cora-cyberneticanimals/cora-the-tortoise-m-docilis/>. Acesso em 21 jun. 2019.

GROMNICA, ZEDENEK**. *Future Millenium***. 2006. Disponível em <https://www.dafont.com/pt/futuremillennium.font>. Acesso em 10 nov. 2019.

MATHCENTER EXFORD. ***Normal Distribution*.** Disponível em: <http://mathcenter.oxford.emory.edu/site/math117/normalDistribution>. Acesso em: 20 de set. 2019.

MARION, Bruno. ***Fractals for Dummies***. 2018. disponível em: <**https://brunomarion.com/fractals-for-dummies/**>. Acesso em: 9 de outubro de 2019.

MINUTELABS. ***Brownian Motion***. 2015. Disponível em: <https://labs.minutelabs.io/Brownian-Motion/>. Acesso em: setembro/2019.

POYIAS, ANDREA**. *A guide to the observer pattern***. 2019. Disponível em <https://medium.com/datadriveninvestor/design-patterns-a-quick-guide-to-observer-pattern-d0622145d6c2>. Acesso em 21 de nov. 2019.

RAY, Thomas. Karl Sims. ***Virtual Creatures***. 1996, p. 10. Disponível em <http://life.ou.edu/pubs/fatm/node10.html>. Acesso em 14 de jun. 2019.

RENDELL, Paul. ***This is a Turing Machine implemented in Conway's Game of Life***. Disponível em <http://rendell-attic.org/gol/tm.htm>. 2015. Acesso em 25 de jun. 2019.

ROLLINGS , ANDREW. ***Andrew Rollings and Ernest Adams on Game Design***. New Riders. 1 de maio de 2003. Disponível em: <https://my.safaribooksonline.com/1592730019/ch14>. Acesso em 10 de nov. 2019.

REYNOLDS, Craig. ***Steering Behaviors For Autonomous Characters*** (Comportamentos Guiados para personagens autônomos). disponível em: < HYPERLINK "%3chttps://www.red3d.com/cwr/steer/gdc99%22https://www.red3d.com/cwr/steer/gdc99" HYPERLINK "%3chttps://www.red3d.com/cwr/steer/gdc99%22https://www.red3d.com/cwr/steer/gdc99" HYPERLINK "%3chttps://www.red3d.com/cwr/steer/gdc99%22https://www.red3d.com/cwr/steer/gdc99"https://www.red3d.com/cwr/steer/gdc99"https://www.red3d.com/cwr/steer/gdc99>. 1997. Acesso de 1de maio 2018

SIMS, Karl. ***Artificial Evolution for Computer Graphics*** (Evolução artificial para computação gráfica). 1991. Disponível em: <http://www.karlsims.com/papers/siggraph91.html>. Acesso em 1 de maio 2018.

SHIVANG. 2018. ***Procedural Generation***. Disponível em: <https://www.8bitmen.com/procedural-generation-a-comprehensive-guide-in-simple-words/>. Acesso em: 9 de outubro de 2019.

TOWNLEY, Chiara. ***Leaky blood vessels may trigger Alzheimer's***. 2019. Disponível em: <https://www.medicalnewstoday.com/articles/324198.php>. Acesso em: 9 de outubro de 2019.

UNIVERSIDADE DE COPEHAGEN. ***Bacteria contradict Darwin: Survival of the friendliest.*** 2019. Disponível em: <https://www.sciencedaily.com/releases/2019/10/191011095910.html>. Acesso em 13 de nov. 2019.

ZHART, DIMITRY. ***What's a design pattern?***. 2014. Disponível em <https://refactoring.guru/design-patterns/what-is-pattern>. Acesso em 15 de nov. 2019.

WIKICOMMONS. ***Depth-First Search in action***. 2014. Disponível em: <https://en.wikipedia.org/wiki/File:Depth-First_Search_Animation.ogv>. Acesso em: 9 de outubro de 2019.