

PlantGoshi

Projeto Integrador III - Sistema Autônomo

Anderson J. Silva, Felipe R. de Luca, Nelson J. Dressler

¹Bacharelado em Ciência da Computação – Centro Universitário Senac - Santo Amaro
São Paulo - SP - Brasil
2015

Resumo. *O projeto foi desenvolvido para a disciplina Projeto Integrador III: Sistema Autônomo, com o objetivo de aplicar técnicas e implementar algoritmos de visão computacional em um jogo de tema livre. Para tal, criamos um jogo digital em 2D, desenvolvido em linguagem C, onde o jogador deve cuidar de uma árvore em seu processo de crescimento, com o objetivo principal de colher os melhores frutos. Para isso, o jogador terá como ferramenta de interação uma varinha mágica, que permitirá aplicar poderes que interajam com os elementos dentro do jogo, contribuindo com o crescimento da árvore e impedindo que pragas ataquem os frutos. A interação da varinha com o jogo será por intermédio do reconhecimento dela nas imagens capturadas pela câmera de video instalada no computador, processadas por algoritmos baseados em levantamento bibliográfico.*

1. Introdução

O Projeto desenvolvido para a disciplina Projeto Integrador III: Sistema Autônomo aborda a questão sobre educação ambiental. O jogador tem três minutos para cuidar de uma árvore, desde seu crescimento até o amadurecimento e colhimento dos frutos. Durante a partida, o jogador deverá estar sempre atento à falta de água e pragas que surgirão para comer os frutos.

O desafio do projeto consiste em estudar, desenvolver e implementar algoritmos de visão computacional para processamento de imagens, com a finalidade de promover a interação do jogador com o jogo. Essa interação deverá ocorrer exclusivamente através da câmera de video acoplada ao computador. O jogador terá à sua disposição uma varinha mágica com uma luz de LED na ponta, que será reconhecida pelos algoritmos e seus movimentos traduzidos como coordenadas de posição X e Y dentro do jogo. Basicamente a varinha tem o funcionamento de um mouse.

2. O jogo

Para vencer os desafios, o jogador tem à sua disposição quatro tipos de poderes especiais: Poder de Regar, Poder de Remover Pragas, Poder de Colher Frutos e Poder da Música. Cada um desses poderes tem uma função diferente no jogo e devem ser utilizadas com cautela, pois quando acionado um desses poderes, os restantes ficarão indisponíveis por alguns segundos. Não é possível utilizar mais de um poder por vez.

O jogador tem a opção de mudar manualmente durante a partida a cor reconhecida do objeto usado para interagir com o jogo. Basta selecionar os números de 1-6 e o jogo irá reconhecer a nova cor. As opções possíveis de cor são as seguintes:

1. Amarelo
2. Verde
3. Ciano
4. Azul
5. Magenta
6. Vermelho

2.1. Poderes

2.1.1. Poder de Regar

Com Poder de Regar o jogador garante que a árvore irá crescer mais e consequentemente irá dar mais frutos. É necessário estar sempre atento à barra de nível de água. Nunca regue demais a árvore e também não a deixe sem água.



Figure 1. Poder de Regar

2.1.2. Poder de Remover Pragas

O Poder de Remover Pragas auxilia o jogador a retirar bichos que nascem, evitando assim que os frutos sejam comidos. O jogador tem um tempo curto para remover essas pragas, antes que elas danifiquem os frutos e a árvore.



Figure 2. Poder de Remover Pragas

2.1.3. Poder da Música

O poder da música auxilia no amadurecimento mais rápido dos frutos. O jogador aplica diretamente esse poder em cima do fruto que deseja.



Figure 3. Poder da Música

2.1.4. Poder de Colher

A pontuação final da partida do jogo está diretamente ligada ao Poder de Colher. Existe um momento ideal para colher o fruto, que é quando ele está vermelho e saudável. Caso seja colhido antes da hora ou depois, o jogador perderá pontos. Muita atenção caso o fruto esteja com praga. Se isso ocorrer, será necessário removê-la antes de colher o fruto.



Figure 4. Poder de Colher

2.2. Varinha mágica

Com a varinha mágica o jogador poderá interagir com os elementos dentro do jogo como seleção de poderes, regar e remoção de pragas. Ao selecionar um poder a luz na ponta da varinha irá assumir a cor desse poder, indicando que ele foi selecionado e está ativo. Se não houver luz, significa que a seleção de poderes está desabilitada temporariamente. Caso a luz seja a cor branco, então está disponível selecionar qualquer poder.

Para interagir com o jogo, basta mover a varinha em frente á tela do computador e um cursor na tela irá indicar qual a posição atual da varinha dentro do jogo. Para selecionar o poder desejado, é necessário posicionar o cursor por dois segundos em cima desse poder. O poder foi selecionado quando a luz na ponta da varinha mudar de cor.



Figure 5. Varinha mágica com LED na ponta.

2.3. Layout

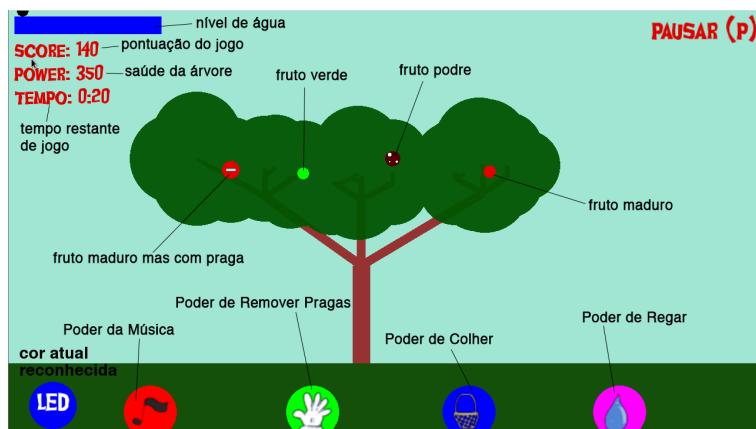


Figure 6. Primeira versão da tela do jogo. Os gráficos serão alterados para a versão final do jogo.

2.4. Etapas

A partida do jogo é dividida em três principais etapas:

1. **Nascimento e Crescimento da Árvore:** o jogador deverá estar atento a regar a árvore sempre que necessário e, ao mesmo tempo, combater ervas daninhas que irão crescer ao pé dela.
2. **Amadurecimento dos Frutos:** os frutos irão crescer mais rápido se o jogador utilizar notas musicais. Também irão crescer pragas nesses frutos, que podem ser combatidas com o poder de remoção de pragas.
3. **Colhimento dos Frutos:** é o momento no qual os frutos cresceram e amadureceram o suficiente para serem colhidos, contabilizando pontos para o jogador.

3. Visão Computacional

Compreendendo a parte de visão computacional, foi realizado um levantamento bibliográfico referente ao processamento digital de imagens, reconhecimento de padrões em imagens, operações aritméticas e um estudo aprofundado sobre os modelos de cores, sua natureza e suas características principais.

3.1. Algoritmos utilizados

3.1.1. HSV e RGB

Escolhemos analisar e processar as imagens capturadas pela câmera de video utilizando o espaço de cor HSV. Essa escolha levou em consideração o fato de que trabalhar puramente com o RGB não seria possível separar a cor da luminância e da saturação. Essa separação é essencial para que haja identificação de diferentes objetos na imagem com a mesma cor, mas que tenham luminância e saturação diferentes. Com isso, é possível isolar o objeto utilizado para interação com o jogo dos demais objetos.

Cada pixel de uma imagem extraída da câmera do computador no modelo de cores RGB (Red, Green, Blue) é convertido em HSV / HSB (Hue, Saturation, Value / Brightness), permitindo descobrir o grau da cor pura (Matiz), as faixas representada por cada cor, a porcentagem de saturação da cor (Pureza) e a porcentagem de brilho (Valor).

O H é a matiz e é medida em graus compreendendo valores de 0° a 359°. A faixa de cada uma das seis cores principais (primárias e secundárias) é definida numa margem de 60 graus. As faixas são classificadas da seguinte maneira: Vermelho (0° a 59°), Amarelo (60° a 119°), Verde (120° a 179°), Ciano (180° a 239°), Azul (240° a 299°) e Magenta (300° a 359°). O S é a saturação e é medida em porcentagem nos valores de 0 a 100%. Finalmente, o V é o brilho e medido também em porcentagem, (0 a 100%). Como observação importante, é possível notar que as cores branco e preto são definidas de acordo com o valor de V: quando se aproxima de 0, emite a cor preta, e quando se aproxima de 100, branca.

Por exemplo, para reconhecer a luz do LED, é possível por intermédio de uma porcentagem alta do V (Valor de brilho) e sua cor pela faixa de H (Matiz).

Para manipular essa representação, foi necessário a criação de uma estrutura chamada Pixel que deve armazenar todos os dados de cores de apenas um pixel e a

implementação de dois algoritmos que compreendem as conversões nos dois sentidos: RGB para HSV e HSV para RGB.

A fim de padronizar, foram implementadas também algumas funções de formatação: conversão de um valor decimal para porcentagem (S e V), conversão de um grau para um valor de 0 a 5 (H) e conversão de canais de cores RGB em um valor decimal entre 0 e 1.

Além disso, foram desenvolvidas funções complementares de máximo e mínimo dentre três valores (R, G e B).

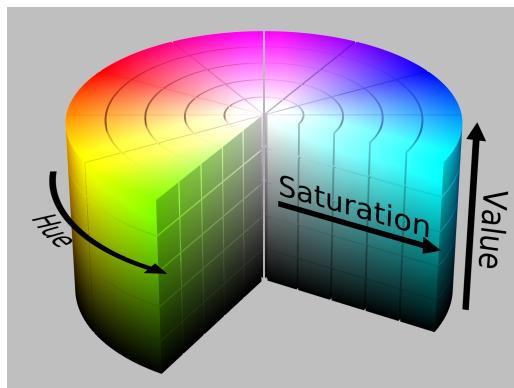


Figure 7. Representação tridimensional do espaço de cor HSV. Créditos: SharkD. http://en.wikipedia.org/wiki/HSL_and_HSV

3.1.2. Escala de cinza / luminância

Os valores RGB de um pixel podem ser convertidos em escala de cinza por uma série de métodos. Com esse fim, foi escolhido o método de extração de luminância Y dos canais RGB. É efetuada a seguinte operação sobre cada pixel:

$$Y = R \times 0.299 + G \times 0.587 + B \times 0.114$$

Dessa maneira, é obtida a luminância da imagem separada das cores.

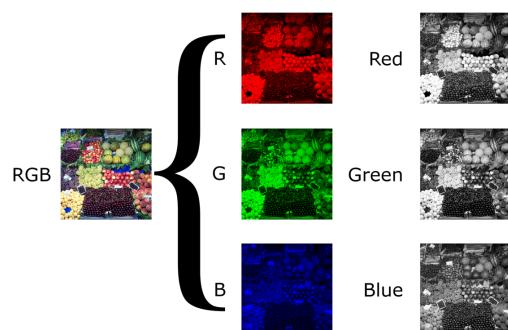


Figure 8. Conversão de cada canal de cor para sua respectiva escala de cinza. Créditos: Nevit Dilmen. <http://en.wikipedia.org/wiki/Grayscale>

3.1.3. Centro de massa

Visando uma definição exata e correta do ponto central que deve ser devolvido pela função da visão computacional, foi necessário a implementação de um mecanismo para obter o centro de massa do maior aglomerado de pixels (clusters) presentes na imagem. Com o valor do centro de massa obtido, é possível retornar as coordenadas x e y desse centro e utilizar como coordenadas de um dispositivo mouse.

Para calcular o centro de massa, é necessário fornecer uma matriz onde os valores dos pixels são 0 ou maior. Quando o pixel for maior do que 0, é somado +1 ao valor presente em cada uma das variáveis que contabilizam a soma total de pixels nos eixos x e y . Paralelamente existe outra variável que contabiliza o total de pixels encontrados com valor maior do que 0:

```
REPETIR ENQUANTO i < imagem.altura; i = i + 1
    REPETIR ENQUANTO j < imagem.largura; j = j + 1
        SE O VALOR matriz[ i ][ j ] > 0 ENTÃO
            centroMassa.y = centroMassa.x + i
            centroMassa.x = centroMassa.y + j
            centroMassa.total = centroMassa.total + 1
```

O cálculo final das coordenadas x e y do centro de massa é realizado da seguinte forma:

```
SE O VALOR centroMassa.total ENTÃO
    y = centroMassa.y / centroMassa.total
    x = centroMassa.x / centroMassa.total
SENÃO
    y = -1
    x = -1
```



Figure 9. Centro de massa representado pelo círculo

3.1.4. Redução de cores

A redução de cores é uma técnica que tem por objetivo facilitar o reconhecimento de uma região específica, diminuindo as cores visíveis e tornando uma região mais destacada e isolada que as demais. Para tal, é usada a seguinte operação sobre cada canal RGB:

```

canal.vermelho = (vermelho / Fator ) * Fator
canal.verde = (verde / Fator ) * Fator
canal.azul = (azul / Fator ) * Fator

```

Considerando que o fator é um valor inteiro que define a proporção da quantidade cores que se quer diminuir na imagem. Exemplos de Fatores: 2, 16, 64, 128. Quanto maior o fator, menor a quantidade de cores visíveis.



Figure 10. Redução da quantidade de cores de acordo com valor do parâmetro K. Fonte: <http://opencvpython.blogspot.com.br/2013/01/k-means-clustering-3-working-with-opencv.html>

3.1.5. Erosão e dilatação

A Erosão e a Dilatação são técnicas de filtragem com o objetivo de reduzir ruídos presentes na imagem, que possam dificultar a detecção de pontos principais na tela.

Os ruídos podem ser causados por uma série de fatores, dentre eles limitações da câmera presente no computador e presença de pontos isolados na tela que tenham características similares aos procurados.

A Erosão trata de eliminar os pontos isolados na imagem, os quais não possuem outros em destaque em sua vizinhança.

A Dilatação trata de aumentar os pontos isolados na imagem, consistindo no processo contrário da Erosão.

Com uma sequência razoável de operações de Erosão e Dilatação, a quantidade de ruídos é descartada, ao mesmo passo que a região que deve ser destacada é mantida em seu tamanho original.

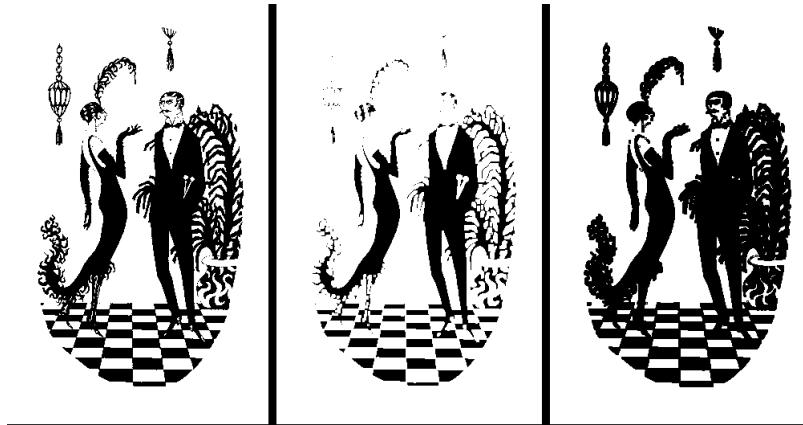


Figure 11. Esquerda: Imagem original. Meio: erosão. Direita: dilatação. Fonte: <http://visiblevisible.org/teaching/setpixel/students/katherine/images/erodeDilate.png>

3.1.6. Filtro de média de pixels

O filtro que processa a média entre os pixels é conhecido como filtro do tipo espacial, ou seja, uma região de pixels é avaliada e processada.

A principal utilização do filtro de média de pixels é suavizar a imagem e reduzir ruídos provocados pela compressão da imagem e também por outros fatores, entre eles a qualidade da câmera.

O cálculo deste filtro é bem simples e consiste em somar os valores de todos os pixels dentro de uma matriz, que geralmente é de tamanho 3x3, para depois dividir o resultado da soma pelo número de elementos na matriz. O valor obtido é então atribuído ao pixel do centro dessa matriz, conforme exemplo abaixo:

120	10	246
200	16	100
230	83	215

→

120	10	246
200	135	100
230	83	215

Para o desenvolvimento deste projeto foi aplicada uma matriz de tamanho 7x7 individualmente para cada canal de cor RGB, o que resultou em uma imagem bem suavizada e eliminou a maioria dos ruídos provocados pela compressão dos frames de vídeo produzidos pela câmera de captura.



Figure 12. Esquerda: imagem com ruído. Direita: imagem após aplicação do filtro. Fonte: <http://lodev.org/cgtutor/filtering.html>

3.1.7. Redução de escala

Cada câmera de video captura a imagem em resoluções diferentes. Isso é um problema quando não há meios de configurar a resolução de captura da imagem antes de iniciar o jogo, pois o processamento da imagem varia de acordo com o computador utilizado. Para evitar esse tipo de problema e possibilitar que os algoritmos implementados trabalhassem de forma eficiente, empregou-se uma técnica bem simples para reduzir o tamanho da imagem capturada para um tamanho pequeno e fixo.

Adotou-se como padrão a dimensão 320 x 240 que proporciona duas vantagens: menos memória RAM consumida e menor tempo de processamento:

```
REPETIR ENQUANTO y < imagem.altura / 240; y = y + 1
    REPETIR ENQUANTO x < (imagem.largura - 1) / 320; x = x + 1
        escala.x = x * 320
        escala.y = y * 240
        imagem[ escala.y ][ escala.x ] = pixel.valor
```

3.1.8. Calibração de cores

Com o intuito de promover mais opções de cores para interagirem com o jogo, foi implementado um algoritmo para calibrar as cores de acordo com o ambiente em que o jogador está. Ele utiliza as técnicas descritas para ajustar as cores a cada *frame*.

Para calibrar, o jogador deve centralizar dentro do quadrado vermelho o objeto com a cor. O algoritmo detecta a cor e inicia uma comparação entre a quantidade de pixels dentro do quadrado e fora. São realizados ajustes automáticos de luminância, matiz e saturação, até que a quantidade de pixels internos seja pelo menos 300 pixels e externos menos do que 150.

A luminância é ajustada de uma forma particular: calcula os mínimos e máximos do objeto situado dentro do quadrado.

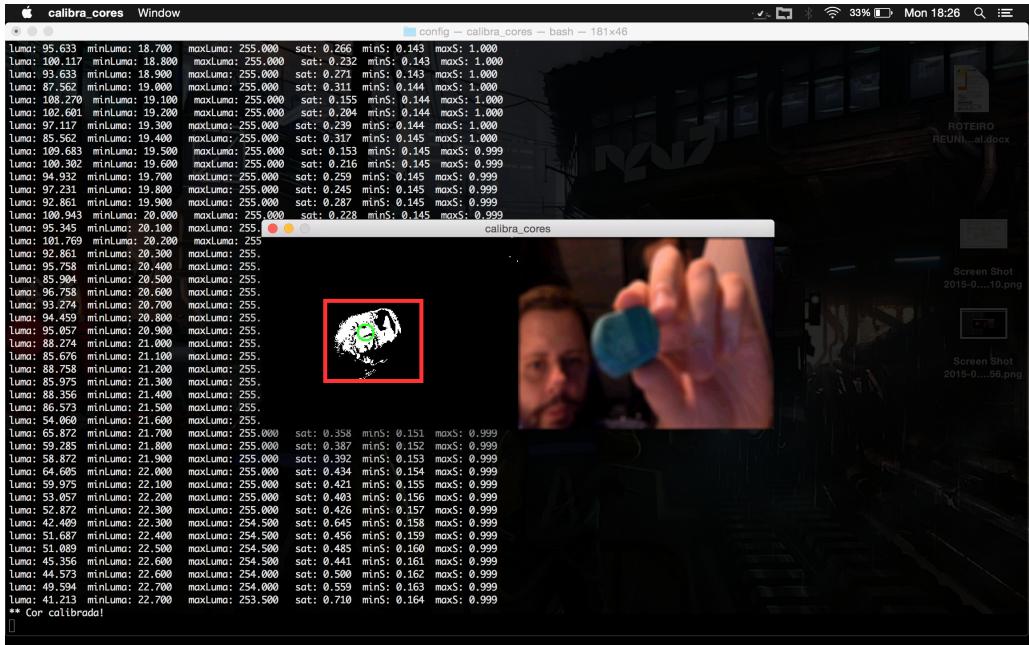


Figure 13. Processo de calibração da cor

A configuração da cor é salva em arquivo no disco rígido após terminado o processo e é utilizada durante a partida do jogo.

3.2. Primeiros experimentos

Para o reconhecimento de uma cor emitida por um LED do Arduino, fizemos uma otimização física com um pedaço de papel para deixar a cor e luz mais difusa e homogênea.

Como experimentos necessários, temos o reconhecimento da luminosidade para verificar a proximidade da luz com a câmera e uma otimização afim de isolar a cor do LED, ignorando as outras cores similares e aparentes na câmera.

3.3. Aplicação dos algoritmos

Utilizando os algoritmos descritos em **3.1**, foi criada uma biblioteca de visão computacional que aplica cada um desses algoritmos para retirar informações sobre os pixels da imagem a cada *frame* de vídeo.

Inicialmente, o jogador tem a opção de calibrar as cores amarelo, verde, ciano, azul, magenta e vermelho antes de iniciar o jogo.

Primeiro, é reduzido o tamanho da imagem capturada pela câmera de vídeo para as dimensões 320 x 240. Em seguida é aplicado o filtro de média, que reduz ruídos na imagem e as informações salvas na matriz *matrizMedia*. Cada canal de cor de cada pixel dessa matriz é armazenado em dois tipos distintos de variáveis sendo uma para cor reduzida e a outra para a cor atual do pixel:

```
pR = redução( matrizMedia[ y ][ x ].vermelho )
pG = redução( matrizMedia[ y ][ x ].verde )
```

```
pB = redução( matrizMedia[ y ][ x ].azul )
```

```
mR = matrizMedia[ y ][ x ].vermelho
```

```
mG = matrizMedia[ y ][ x ].verde
```

```
mB = matrizMedia[ y ][ x ].azul
```

A luminância Y é extraída a partir das variáveis mR , mG e mB .

Ambos os conjuntos mR , mG , mB e pR , pG , pB são submetidos ao processo de conversão HSV que retornará para o conjunto pR , pG , pB os valores pH , pS , pV e para o conjunto pR , pG , pB os valores mH , mS , mV .

Após o processo de conversão HSV, essas variáveis são comparadas com a cor calibrada e, caso haja equivalência, o valor 255 é salvo na matrizProcessada e 0, caso contrário:

```
SE OS VALORES pH >= corCalibrada.hInicio E pH <= corCalibrada.hFim  
OU mS >= corCalibrada.sInicio E mS <= corCalibrada.sFim E Y >=  
corCalibrada.yInicio E Y <= corCalibrada.yFim ENTÃO
```

```
matrizProcessada[ y ][ x ] = 255
```

SENÃO

```
matrizProcessada[ y ][ x ] = 0
```

Em seguida, é calculado o centro de massa na matrizProcessada e são retor-nadas as coordenadas x e y que serão utilizadas para mover o cursor do mouse dentro do jogo.

4. Simulação da árvore

Adotamos a estrutura de dados de árvore ternária para construir a árvore do jogo. Essa estrutura é basicamente constituída em pontos de crescimento que se dividem no máximo em três partes cada um. Essas partes dão origem aos galhos que crescem até determi-nado tamanho, decidido aleatoriamente. As pontas de cada galho são novos pontos de crescimento, que se dividem de novo e dão origem a novos galhos.

Para que a simulação tenha um aspecto e comportamento mais próximo de uma árvore, elaboramos uma solução que inclui a matéria estudada na disciplina de Estrutura de Dados e elaboramos algoritmos para simular o crescimento da árvore, dos galhos e dos frutos. Basicamente a árvore cresce de acordo com um valor de energia de crescimento fornecida a ela logo no início do jogo. Uma parte dessa energia é consumida pelo tronco e o restante é distribuído de maneira aleatória para os próximos galhos que irão nascer, de acordo com o seguinte critério:

```
SE galho.energiaConsumida == ((galho.energiaLimite * 50) / 100) E  
galho.temFilhos == FALSO ENTÃO galho.criarFilhos = SIM
```

Isso se repete até que não haja mais energia suficiente para repassar ao galho seguiente. O jogador tem a oportunidade de fornecer mais energia ao longo da partida, o

que irá proporcionar uma árvore mais desenvolvida e com mais frutos para serem colhidos.

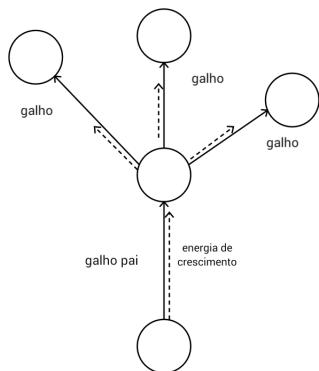


Figure 14. Crescimento dos galhos e energia transferida de pai para filho.

4.0.1. Crescimento

Para que os galhos crescam controladamente mas mantendo certa diferenciação em tamanho e direção entre um e outro, estabelecemos um critério que chamamos de energia de crescimento e energia limite. A energia limite determina o quanto cada galho da árvore vai crescer e é estabelecido através de uma porcentagem, calculada a partir do total de energia de crescimento fornecida para o galho, o nível de altura desse galho e a energia limite da árvore:

```
galho.energiaLimite = (energiaRecebida * (arvore.energiaLimite -
(galho.profundidade * 2)) / 100)
```

O valor da energia limite pode ser alterado de acordo com a interação do usuário ao longo do jogo, o que pode proporcionar uma árvore mais ou menos desenvolvida.

Enquanto os galhos crescem eles podem dar origem a novos galhos. Isso é determinado após ser consumida uma determinada quantidade de energia de crescimento fornecida ao galho, o que permite que a simulação de crescimento da árvore fique mais natural. A única exceção é o tronco da árvore, que tem o crescimento mais controlado:

```
SE galho.energiaConsumida < galho.energiaLimite E
galho.energiaRecebida > 0 ENTÃO galho.crescer
```

4.0.2. Pontos de crescimento de frutos e folhas

Os pontos de crescimento de frutos e folhas são determinados de acordo com o mínimo de energia que o galho tem para crescer. Caso esse valor seja igual ou abaixo de determinado critério, o algoritmo de simulação assume que não irão surgir novos galhos a partir do galho atual, ainda durante a fase de crescimento. Com isso determinado, o galho passa a produzir folhas e frutos ao invés de novas ramificações de galhos.

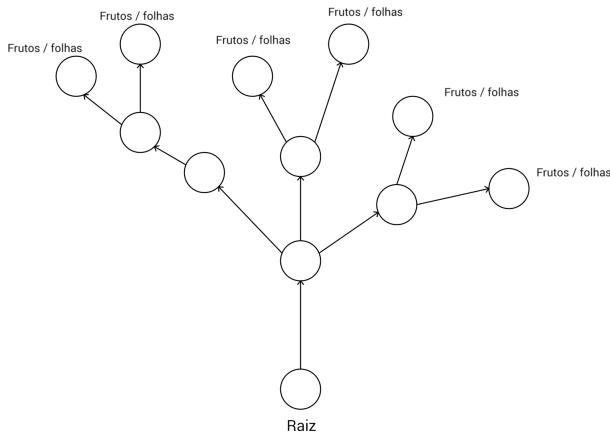


Figure 15. Estrutura da árvore e pontos de crescimento de frutos e folhas

5. Bibliotecas

5.1. OpenCV 2.4.11

OpenCV (Open Source Computer Vision Library) é uma biblioteca de código aberto de visão computacional e aprendizado de máquina, desenvolvida para dar suporte a aplicativos que requerem processamento de imagens.

5.2. Allegro 5.0

O Allegro é uma biblioteca multiplataforma de programação voltada para desenvolvimento de jogos. Ela oferece suporte para programação baixo nível em C e C++, ou seja, fornece ferramentas para que o usuário desenvolva sua própria programação de jogo.

5.3. Arduino-Serial

Biblioteca de código aberto em linguagem de programação C, que oferece suporte a comunicação via porta serial do computador com o Arduino.

5.4. Linguagens de programação

O projeto inteiro foi desenvolvido em linguagem C padrão c99.

6. Equipamentos

- Câmera de captura de vídeo
- Placa controladora Arduino Uno
- LED RGB
- Computador (desktop ou notebook)

7. Resultados e conclusão

O estudo de técnicas em visão computacional proporciona novas alternativas em usabilidade e interface usuário/máquina. Além de contribuir muito ao aprendizado em desenvolver novas soluções tecnológicas e fortalecendo a habilidade em criar algoritmos cada vez mais sofisticados e robustos.

Um dos principais desafios para a realização desse projeto foi a integração da visão computacional com o processamento gráfico do jogo, de modo que, fosse possível obter eficiência e uma jogabilidade razoável para o jogador, ao mesmo tempo que, deve haver o processamento gráfico do jogo e da visão.

Outro desafio que enfrentamos foi a criação de um algoritmo que simulasse o crescimento da árvore de forma aleatória e que recebesse devidamente as interações do usuário.

Como melhorias para o projeto, é possível desenvolver mecanismos de calibração e detecção para outras cores e uma interação maior do arduino com o jogo, atribuindo uma cor luminosa, pré-determinada pelo usuário, para cada ícone ou poder específico.

Analisando os resultados obtidos com o uso de um arquivo de configuração para a calibração das cores, é possível afirmar que o mesmo método pode ser aplicado em outros projetos relacionados à visão computacional, tornando a implementação do programa mais modularizada e organizada.

8. Bibliografia

References

- Cavalcanti, J. Disciplina de computação gráfica. http://www.univasf.edu.br/~jorge.cavalcanti/comput_graf06_Cores.pdf.
- Conci, A., Azevedo, E., and Leta, F. R. Computação gráfica. <http://computacaografica.ic.uff.br/transparenciasvol2cap4.pdf>.
- Cook, J. D. (2009). Three algorithms for converting color to grayscale. <http://www.johndcook.com/blog/2009/08/24/algorithms-convert-color-grayscale/>.
- E, A., A, C., and R, L. F. (2009). *Computação Gráfica: teoria e prática*, volume 2. Campus.
- Itseez. Opencv. <http://opencv.org/>.
- Kurt, T. E. (2006). Arduino-serial. <http://todbot.com/blog/2006/12/06/arduino-serial-c-code-to-talk-to-arduino>.
- Marengoni, M. and Stringhini, D. Tutorial: Introdução à visão computacional usando opencv. http://seer.ufrgs.br/rita/article/viewFile/rita_v16_n1_p125/7289.
- Parker, J. R. (2011). *Algorithms for Image Processing and Computer Vision*. Wiley Publishing, Inc., 10475 Crosspoint Boulevard. Indianapolis, IN 46256, 2nd edition.
- RapidTables. Rgb to hsv color conversion. <http://www.rapidtables.com/convert/color/rgb-to-hsv.htm>.
- Team, A. Allegro 5. <http://alleg.sourceforge.net/readme.html>.
- Wright, S. (2010). *Digital Compositing for Film and Video*. Focal Press, 3rd edition edition.