Visão geral do projeto VERSUS na disciplina Projeto Integrador III

Helder Doutel e Rodrigo Melo

02 de Junho de 2015

Resumo

O jogo tem a temática resistência. O jogador deve evitar que os inimigos entrem em sua base, e ao mesmo tempo se esquivar dos ataques que causam dano. O jogo é controlado por uma câmera, que por meios de algoritmos abordados neste artigo. Esses algoritmos fazem a análise dos movimentos e repassam para o jogo, para isso existem algoritmos de detecção de cor e de movimentos. Esses códigos são a base do jogo. Com os algoritmos definidos e minuciosamente explicados, descrevemos como os resultados são utilizados e concluímos se são eficientes, pontos positivos e negativos.

Palavras-chave: jogo, 2D, visão computacional, interação, tiro.

1 Introdução

O jogo foi desenvolvido de forma que ele se torne similar ao jogo "Contra", sendo ele de tiro e 2D. No jogo é possível alterar a direção do tiro e pular. É desenvolvido com métodos de visão computacional que maximizam a experiência do usuário, provendo uma interação extremamente agradável ao jogador utilizando de movimentos reais para simular ações similares no jogo e assim optimizar a jogabilidade. O jogo devera detectar os movimentos da "arma" do jogador, sua inclinação e pulos para assim interagir com o jogo.

2 O jogo

Para o desenvolvimento do jogo usando visão computacional, primeiramente é necessário entender a captura da imagem da câmera. Nossa imagem inicial é capturada pela câmera utilizando o sistema de cores RGB.
RGB é um sistema de representação de cores utilizadas em monitores onde todas as cores visíveis são formadas a partir de 3 cores básicas. R (vermelho), G (Verde), B (Azul), A escala individual de cada cor pode variar de 0 à 255, onde 0 seria a ausência da cor em questão e 255 o máximo possível para cor. A combinação de diversos níveis das cores básicas são capazes de gerar todas as outras cores.
O RGB considera apenas a cor do pixel da imagem individualmente, ou seja, as cores em dois pixeis em volta não influenciam. Não existe uma detecção de luminosidade ou qualquer outra medida da imagem.
A imagem capturada fica salva em uma matriz com as dimensões da captura da câmera. Esta matriz além de altura e largura tem uma terceira dimensão de 3 elementos variáveis que representam as cores do RGB, vermelho, verde e azul. Assim podemos trabalhar o algoritmo baseado em uma única matriz atualizada de acordo com o número de capturas.
As interações do jogo são feitas a partir de uma "arma" de duas cores, a partir dessas cores são selecionados dois pontos da arma para interpretar os movimentos do jogador.

3 Seleção de cores

Para uma melhor jogabilidade, decidimos por usar a detecção de cores para selecionar o que nos interessa na imagem. Como visto anteriormente a maioria das cores apresentam variações de todas as cores básicas. Ou seja, para selecionar uma cor temos que considerar os níveis de todas as cores no pixel.
Então para selecionar a cor desejada (no caso vermelho) começamos uma varredura pixel a pixel e aplicamos a ação condicional, onde a cor vermelha (na escala RGB) tem que ser maior que 180 e pelo menos 50 maior que as outras cores básicas (G – verde, B – azul). Caso o pixel se enquadre nessa condição alteramos o valor original do pixel para R-255, G-0, B-0, Tendo assim o nível mais intenso de vermelho que raramente pode ser encontrado em uma captura padrão feita pela câmera. Caso não se enquadre na condição, o pixel passa por uma nova análise para a cor Azul, que em nosso jogo representa a segunda cor de interesse. A condição segue o mesmo princípio que para o vermelho, se azul maior que 160 e pelo menos 50 maior que as outras cores, então alteramos R-0, G-0, B-255. Com esse algoritmo inicial temos as duas cores de interesse ressaltadas na imagem inicial para que seja possível trabalhar alguns aspectos específicos com mais precisão.
A biblioteca utilizada para capturar a câmera foi o OpenCV [1].

4 Detecção de dois pontos

Com a matriz já com seus valores de cores alterados de acordo com a seleção de cores, podemos agora selecionar um ponto para cada cor. Agora precisamos entender a imagem e o comportamento do jogador.
Com a seleção feita anteriormente temos que considerar que possivelmente existem pixeis

com a cor desejada perdidos na matriz que foram convertidos por se enquadrar na regra, mas que são desprezíveis, já que não fazem parte da arma que comanda o jogo. Podemos considerar que nosso instrumento tem como cor predominante as cores desejadas convertidas. Ou seja, se pegarmos um ponto, os pontos em volta devem ser da mesma cor.
Então com esse conceito fixado começamos uma nova varredura da matriz pixel a pixel. Para cada pixel avaliado que se enquadrar na cor selecionada temos que avaliar os pixeis em volta em um determinado raio, no caso, utilizamos quatro variáveis, tendo o pixel central uma coordenada x e y, temos cima(x; y+10), baixo(x; y-10), direita(x+10; y), esquerda(x-10; y). Se todos tiverem a cor exatamente igual então as coordenadas ficam salvas, até que um novo pixel que se enquadre à regra mais ao centro da imagem seja encontrado.
Como estamos trabalhando com duas cores a regra se aplica novamente para a outra cor, mas com uma diferença, essa cor nunca pode estar mais à direita da cor anterior, podemos entender que, o ponto vermelho sempre estará a direita do azul.

5 Detecção de movimentos do jogador

Agora com dois pontos selecionados representando dois pontos do objeto que controla o jogo podemos iniciar a interpretação dos movimentos do jogador. Baseados no jogo, onde existem os movimentos de pular e mirar, temos que considerar os mesmos movimentos pro jogador, já que o objetivo é deixar a jogabilidade o mais real possível.
Para detectar o pulo, usaremos como base o ponto selecionado para a cor vermelha, já que é o ponto que menos se mexe verticalmente, este ponto fica na região central da arma e serve como eixo de rotação. Então caso exista algum movimento vertical isso pode ser um pulo. Por se tratar de um jogo de ação, temos que desprezar movimentos naturais do jogador, que podem ocorrer inclusive pelo ato de mirar. Para isso, usaremos duas variáveis de apoio que devem guardar as coordenadas anteriores do ponto em análise. O ponto atual deve ser comparado com o ponto anterior, se a varição for muito grande podemos deduzir que foi a variação resultante de um pulo, mas não podemos fazer essa análise para cada atualização da câmera, pois dependendo da taxa de atualização da câmera a variação será pequena em todos os casos. Então para contornar esse problema utilizaremos um contador de atualização da câmera, que deve ser calibrado de acordo com sua webcam, o contador é incrementado toda vez que a câmera atualizar. Após um número determinado de frames a análise é executada.

O outro movimento que o jogador tem controle é o de mira, que é controlado pela inclinação da arma, definimos a mira em três níveis, 45º graus para cima, para baixo e reto. Para determinar está inclinação, utilizaremos o valor do seno do ângulo formado entre os dois pontos com a horizontal, se a inclinação for maior que o seno determinado (no nosso caso 30º graus) e o ponto azul estiver mais a cima do ponto vermelho temos a mira para cima, e se estiver mais a baixo temos a mira para baixo.
O calculo é feito da seguinte forma, a função recebe as duas variáveis de cada ponto, Xa recebe o maior valor X(horizontal) entre os dois pontos, Ya recebe o maior ponto Y entre os dois pontos, e por consequência Xb e Yb os menores valores.
Utilizando a distância entre dois pontos no plano x, y calculamos a hipotenusa do triângulo retângulo que usaremos para efetuar os cálculos. A distância entre dois pontos pode ser determinada pela fórmula.
$D = \sqrt{(xa - xb)^2 + (ya - yb)^2}$
Tendo a hipotenusa determinada, temos apenas que calcular o seno, que pode ser escrito da seguinte forma.
$Sen = cato/hipo$
Sendo cato o cateto oposto ya-yb e hipo a hipotenusa. Como o seno é sempre o mesmo independente das dimensões do triângulo, o jogador pode ser mover na tela sem afetar o calculo.

6 Os inimigos e o personagem

O jogo em si é bem simples. Com os pontos e as ações funcionando, o jogo se baseia na posição do jogador no momento que os gatilhos são disparados, os inimigos surgem a partir de um ponto aleatório da tela e tentam chegar do outro lado, com fim da trajetória posição é reiniciada, o tiro do personagem, pode sair de três posições de acordo com a inclinação da arma no momento do tiro, os tiros inimigos, surgem de uma posição aleatória, e tendem a ir em direção ao personagem principal. O personagem principal pode tomar cinco golpes no máximo e não pode deixar mais dez inimigos passarem. Se um dos dois acontecer o jogo acaba. O jogo só acaba em uma das duas condições e é um jogo de resistência.
Para a manipulação das imagens no jogo, criação de telas, e fila de eventos foi utilizado a biblioteca Allegro5 [2].

7 Considerações Finais

O jogo se demonstrou bem eficiente quanto a detecção de cor, os algoritmos funcionam bem e são de fácil calibração, tendo em vista que poucos valores são fixos e as mudanças que podem ser necessárias no código na maioria são para calibrar de forma mais fina esses valores. Quanto ao desempenho, o jogo rodou sem dificuldades. A partir das funções descritas a cima, o jogo funciona com colisões, que seriam a análise da posição das imagens na tela, e com simples gatilhos, isso faz com que o jogo não necessite de muito processamento para funcionar. Durante o desenvolvimento o maior desafio foi a detecção dos pontos que muitas vezes mudavam rapidamente de lugar por causa da variação de luminosidade. Isso foi contornado com o ajuste da detecção. Mesmo com o jogo estável, não é recomendado que se jogue com roupas de cores brilhantes pois podem confundir o algoritmo de detecção. Apesar das ideias terem mudado bastante com o desenvolvimento, chegamos a um ponto que o jogo se mostrou bem divertido, todos que jogaram se mostraram bem familiarizados com a jogabilidade e o jogo cumpre com a proposta de passar uma jogabilidade o mais real possivel.

8 Referências

[1] OpenCV: http://www.opencv.org/
[2] Allegro5: http://alleg.sourceforge.net/
[3] Allegro5 Tutorials: http://wiki.allegro.cc/index.php?title=Allegro_5_API_Tutorials
[4] Allegro5 Manual: http://alleg.sourceforge.net/a5docs/5.0.10/