Emerson Renan Garcia Vida
Felipe Russini Zanatto
Igor Henrique Teixeira
Lucas Rizzi Gomes
Orientador: Prof. Dr. Sc. Paulo Sérgio Silva Rodrigues

# Virtual Snooker

São Bernardo do Campo
2010

Emerson Renan Garcia Vida
Felipe Russini Zanatto
Igor Henrique Teixeira
Lucas Rizzi Gomes
Orientador: Prof. Dr. Sc. Paulo Sérgio Silva Rodrigues

# Virtual Snooker

Monografia apresentada ao Curso de Graduação em Ciência da Computação do Centro Universitário da FEI, como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

São Bernardo do Campo
2010

Emerson Renan Garcia Vida
Felipe Russini Zanatto
Igor Henrique Teixeira
Lucas Rizzi Gomes

**Virtual Snooker**

**Trabalho de Conclusão de Curso - Centro Universitário da FEI**

Comissão Julgadora

_____

Prof. Dr. Sc. Paulo Sérgio Silva Rodrigues
Orientador e Presidente

_____

Examinador (1)

_____

Examinador (2)

São Bernardo do Campo 2010

# Agradecimentos

Ao Prof. D.Sc. Paulo Sérgio Rodrigues pela orientção e esclarecimento fornecido durante o período de pesquisa e desenvolvimento deste trabalho.

Aos nossos familiares que sempre nos apoiaram de forma incondicional e deram a base para que pudéssemos trilhar nossos caminhos.

Ao Centro Universitário da FEI pela estrutura fornecida.

E a todos que contribuíram direta ou indiretamente com a conclusão deste trabalho.

# Resumo

Este trabalho apresenta uma proposta de um jogo de sinuca virtual. O projeto visa a criação de um jogo de sinuca virtual que proporciona a interação entre o mundo real e o mundo virtual, onde mesa, caçapas e bolas são virtuais e apenas o jogador e o taco são reais. O projeto consiste em dois módulos principais: Módulo de Visão Computacional, com o auxílio de uma câmera montada sobre a mesa, deve identificar a posição e direção do taco, e o Módulo de Simulação Física, que com o auxílio de um projetor irá projetar o cenário de um jogo de sinuca em uma superfície plana de cor branca que não produza reflexos, e também que deve simular o mais próximo da realidade possível às relações físicas existentes em um jogo de sinuca como, velocidade e ponto de atrito do taco com a bola e da bola com outras bolas. O projeto tentará minimizar um dos problemas no âmbito da Visão Computacional que é a variação da luminosidade dos sistemas projetados.

**Palavras-chave**: Sinuca Virtual, Visão Computacional, Simulação Física.

# Abstract

This paper presents a proposal for a virtual game of pool. The project aims to create a virtual pool game that provides the interaction between the real and virtual world, where table pockets and balls are virtual and only the player and club are real. The project consists of two main modules: Module Overview Computing, with the aid of a camera mounted on the table, must identify the position and direction of the club and Module Physical Simulation, which with the aid of a projector will project the setting for a game of pool on a flat white that does not produce reflections, and also that it should simulate as closely possible reality of the relations existing in a physical game pool as speed and point of friction with the cue ball and ball with other balls. The project will attempt to minimize one of the problems within the Computer Vision which is the variation of luminosity systems designed.

**Keywords**: Virtual Pool, Computer Vision, Physical Simulation.

# Conteúdo

# Lista de Figuras

# Lista de Tabelas

# Capítulo 1

# Introdução

Há alguns anos, as pessoas vêm se interessando cada vez mais em realizar interações com as máquinas. Com o passar dos anos, pesquisadores vêm focando cada vez mais a atenção para descoberta de formas de interação entre o mundo real e o mundo virtual, tornando assim mais dinâmica a interação entre o homem e a máquina.

A área de jogos caracteriza-se por grandes evoluções nesse segmento, quanto mais interativo e dinâmico for o jogo, mais atrativo é ao público. Os jogos antigamente dispunham de pouca interação, apenas por dispositivos e entrada de dados conectados por fio, como por exemplo, teclados ou mouse para computadores e controles para consoles.

Com o passar dos anos, essas interações se tornaram cada vez mais complexas, exigindo dessa forma cada vez mais do processamento das máquinas. Como um exemplo pode-se citar o console WII, da Nintendo, onde seu controle sem fio é composto de um acelerômetro que capta os movimentos da pessoa e os transmite para o console via bluetooth, simulando-os dessa forma no jogo.

Atualmente, a área tecnológica está visando um crescimento na interação entre o homem e a máquina. A área dos jogos não é diferente, jogos que proporcionam essa relação estão cada vez mais presentes no cotidiano das pessoas, criando um novo conceito de lazer. Pode-se citar nessa área tanto projetos de menor escala, como o projeto de formatura Virtual Hockey [CABRAL et al., 2008], quanto projetos de maior escala, como o console

WII, da Nintendo, já citado, que vem gerando grande interesse dos consumidores, pois os introduzem em um mundo imaginário que simula a realidade de maneira inovadora.

Esse projeto visa desenvolver um jogo que proporciona esse tipo de interação. Unindo visão computacional e simulações físicas, pretende-se criar um jogo de sinuca virtual, onde mesa, caçapas e bolas são virtuais, deixando para o mundo real apenas o jogador e o taco. Para isso, ter-se-á um projetor e uma câmera montados sobre uma superfície, onde o primeiro será responsável por projetor o cenário de um jogo de sinuca e o segundo responsável por capturar os movimentos reais tanto do jogador quanto do taco, introduzindo o jogador em um ambiente simulado.

A sensação de diversão que os jogos de interação entre o real e o virtual proporcionam, combinado com a popularidade do jogo de sinuca entre as pessoas, principalmente entre universitários, inspirou a curiosidade da criação de um jogo de sinuca diferente, um jogo de sinuca virtual, utilizando técnicas tanto de visão computacional e quanto de simulação física.

## 1.1 Objetivo

Este projeto propõe a modelagem e implementação de um jogo de sinuca virtual com taco real, mas com bolas, caçapas e mesa projetadas em uma superfície plana simulando um ambiente de jogo virtual.

# Capítulo 2

# Trabalhos Relacionados

Em [CABRAL et al., 2008], foi desenvolvido um jogo interativo baseado no jogo Air Hockey utilizando de técnicas de visão computacional, processamento de imagem e simulações físicas. A captura da imagem é feita através de uma webcam. Para reconhecimento dos padrões, foi implementado um sistema de subtração de fundo e um sistema de detecção de cor de pele, a fim de comparar seus desempenhos.

O método de detecção de cor de pele apresentou melhores resultados sendo que, foram processados 24 frames em 2,5 segundos com média de 110 ms para cada frame na subtração de fundo e 1,5 segundos com média de 64 ms por frame para a detecção de cor de pele.

Uma dificuldade encontrada foi o atraso médio de 68 ms na transmissão da imagem da webcam para computador, gerando um impacto na dinâmica do jogo.

Em [COSTA, 2007], foi utilizado o jogo de snooker para o estudo dos movimentos de um corpo rígido.

Conceitos de mecânica foram aplicados para calcular o movimento da bola quanto a interações com outras bolas ou com o taco. O movimento é dividido em dois regimes, o transiente e o permanente. O regime transiente trata-se do movimento com deslizamento que ocorre por um instante dependente da região em que a bola foi golpeada até atingir o regime permanente, onde a bola rola sem deslizar.

O artigo trata detalhadamente e cuidadosamente os conceitos envolvidos no movimento

da bola demonstrando assim, a dinâmica de um corpo rígido. Esses conceitos serão tratados neste trabalho.

Em [SMITH, 2007], foi desenvolvido um sistema de inteligência artificial para controlar um robô jogador de snooker.

Chamado de PickPocket, além de preciso assim como os demais, o sistema também utiliza algoritmos de inteligência artificial baseados em busca, adaptados para o domínio do snooker, para decidir qual a jogada que lhe traz o maior beneficio considerando a maior diferença entre o quanto ele e seu adversário estão próximos do objetivo.

Na implementação, dois algoritmos de busca foram testados onde o Monte-Carlo search apresentou melhores resultados que o Probabilistic search. Adicionalmente, o sistema foi campeão do torneio simulado de Pool - 8-ball na 10ª e 11ª competição olímpica de computadores.

Em [GURZONI et al., 2009] foi apresentado um sistema que utiliza redes neurais para a segmentação de cores em tempo real aplicada em um sistema de controle de robôs em uma partida de futebol.

Devido o alto custo computacional de se classificar todos os pixels de uma imagem a cada frame, foi proposto a localização de formas geométricas, e aplicada a detecção de cores somente nas áreas próximas a essas formas previamente encontradas. Para detecção dos padrões de cor, foi implementada uma rede neural que foi treinada para o reconhecimento dos padrões, utilizados no topo dos robôs para identificação dos mesmos.

O sistema mostrou ser robusto em mudanças de ambiente e variações de luminosidade e muito eficiente para classificação em tempo real e com tempo computacional satisfatório.

Em [SHIH, 2010], foi desenvolvido um sistema para treinamento de jogadores de snooker, que combina uma interface visual com algumas instruções que são informadas ao jogador para realizar uma boa tacada.

O sistema utiliza um algoritmo de subtração do fundo utilizando a probabilidade dos valores HSV correspondentes a cada pixel, um filtro de mediana e um algoritmo de iden-

tificação de bordas para encontrar a bola tacadeira, a bola alvo e a caçapa. Com isso, é calculada a região onde o jogador deve acertar a bola com o taco para realizar a jogada calculada pelo sistema. O sistema possui também o tracking do taco, possibilitando ao jogador, ao olhar para o monitor do computador, alinhar sua tacada com a calculada pelo sistema.

Para relacionar as coordenadas capturadas pela câmera com as da mesa de forma precisa, a câmera passou por um sistema de calibração para diminuição de erro utilizando o método dos mínimos quadrados.

O método de calibração se mostrou muito eficiente assim como os filtros para segmentação da imagem, porém o melhor resultado obtido foi a precisão encontrada nos cálculos das melhores jogadas.

Sendo assim, todos esses trabalhos relacionados irão nos ajudar com conceitos necessários para a modelagem e implementação desse projeto.

# Capítulo 3

# Conceitos Fundamentais

## 3.1 Teoria do Jogo da Sinuca

Essa seção irá apresentar os conceitos básicos sobre o jogo de sinuca que irão auxiliar no melhor entendimento do mesmo.

### 3.1.1 Jogo

Sinuca, nome popularmente adotado para abordar regras populares do Snooker. É um esporte praticado por dois jogadores (admitindo conjuntos com maior número) com diversas variações, oficiais e populares, de suas formas de disputa.

O Snooker foi reconhecido internacionalmente como a primeira variação da regra Francesa que originou o esporte, sendo introduzidas características popularmente conhecidas hoje como a Caçapa, inexistente na versão inicial do jogo.

Devido à popularização do esporte, diversas regras foram adotadas, sendo a maior parte delas não oficiais. Uma das formas de jogo mais conhecidas e praticadas é o Pool, oficializado de uma variação popular das regras do Snooker nos Estados Unidos, sendo este dividido em três modalidades: Bola 9 (Nine Ball), Bola 8 (Eight Ball), 14x1 (14.1 Continuous ou Straight Pool).

Neste trabalho, serão consideradas as regras do Pool, modalidade Bola 8 conforme

[SINUCA, 2009] que são resumidas na subseção 3.1.4.

### 3.1.2 Equipamentos

Realizado em uma mesa com medidas oficiais da área de jogo 2,54 m x 1,27 m, sendo limitada por tabelas e dividida em áreas conforme a Figura 3.1, onde se observa também a existência de seis buracos (áreas numeradas na Figura) que são chamados de caçapas. Na superfície da mesa são colocadas 16 bolas, sendo 1 branca chamada de tacadeira, posicionada na marca superior da Figura 3.1 e as demais numeradas de 1 à 15, sendo de 1 à 7 as bolas de lisas ou solidas e de 9 à 15 as bolas chamadas de listradas, posicionadas conforme Figura 3.1 na marca inferior.

Para realizar uma jogada, cada jogador possui um taco, sendo esta, a única forma lícita de interação entre o jogador e as bolas. O taco deve ser utilizado para movimentar a tacadeira e essa por sua vez interagir com as demais bolas.

Nesse trabalho, a mesa será simulada em uma superfície plana.

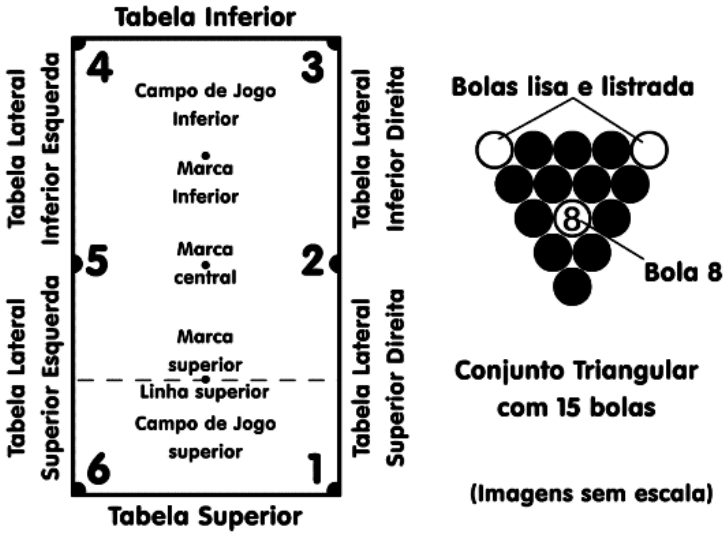


Figura 3.1: Exemplo de uma Mesa de Sinuca

Fonte: [SINUCA, 2009]

### 3.1.3 Como Jogar

É chamado de domínio, os dois grupos de bolas subdivididos pela bola 8, sendo um domínio, as bolas lisas ou solidas (de numeração entre 1 e 7 incluindo) e as bolas listradas (de numeração entre 9 e 15 incluindo). Cada domínio é de responsabilidade de um jogador, sendo definido a partir do momento em que o primeiro jogador encaçapar a primeira bola.

Com as bolas em posição inicial, o jogo se inicia com a primeira tacada. Cada jogador tem direito a uma tacada desde que não encaçape nenhuma bola.

Todas interações devem ser feitas com o taco na tacadeira e esta interagindo com outra bola. É considerado falta, utilizar o taco em uma bola que não a tacadeira, a tacadeira não movimentar outra bola qualquer (antes dos domínios definidos) ou uma bola de domínio do jogador responsável pelo movimento (após domínios definidos).

Caso sem domínio definido, um jogador encaçapar 2 ou mais bolas de domínios distintos, não é definido um domínio e a vez passa ao outro jogador.

Um jogador, ao encaçapar uma bola de seu domínio ou de um domínio qualquer caso não definido, joga novamente e assim sucessivamente.

O objetivo em uma partida de Pool, na modalidade bola 8 é encaçapar a bola 8, após encaçapadas todas as bolas do grupo de mesmo domínio.

### 3.1.4 Regras

Principais regras do Pool oficialmente definidas em [SINUCA, 2009]

- O jogo se inicia a partir da primeira tacada alterando a posição inicial do jogo.
- Defini-se o domínio de cada jogador ao encaçapar a primeira bola, sendo o domínio do jogador que o fizer, aquele que a bola encaçapada pertence. O jogador joga novamente.
- Caso mais de uma bola seja encaçapada, sendo elas de domínios diferentes, nada se define e o próximo jogador tem a vez.

- Uma vez definido o domínio, cada jogador deve acertar, em todas as tacadas, ao menos uma bola de seu domínio com a tacadeira. É considerado falta, não acertar nenhuma bola ou acertar primeiramente uma bola não pertencente ao seu domínio. Como punição, o jogador adversário tem a chamada Bola na mão, que consiste em posicionar a tacadeira utilizando a mão, na posição desejada para facilitar sua tacada.

- Sujeito a mesma punição, também é considerada falta uma jogada onde a tacadeira foi encaçapada ou lançada para fora da área de jogo (mesa).

- Encaçapar a bola 8 antes de encaçapar todas as bolas de seu domínio é considerado falta e a punição é a vitória do adversário.

## 3.2 Redes Neurais

Redes Neurais são sistemas computacionais que tentam simular o modo de funcionamento do cérebro biológico. [MCCULLOCH and PITTS, 1943] criaram um modelo matemático simples para demonstrar como os neurônios da rede funcionam: basicamente, os neurônios são ativados sempre que uma combinação linear das entradas da rede excede um certo limiar. Cada conexão de um neurônio com o próximo nó tem um dado peso, que representa a intensidade do sinal dessa conexão, conforme figura 3.2.

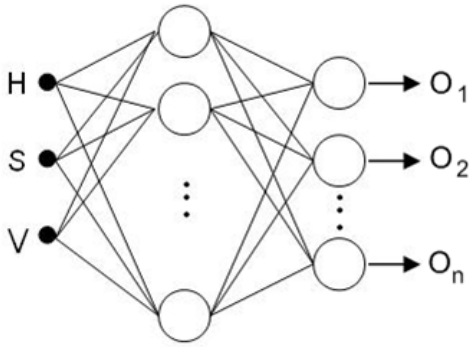


Figura 3.2: Exemplo da Rede Neural

Fonte: [GURZONI et al., 2009]

Exemplo de rede neural, como a implementada por [GURZONI et al., 2009], que tem as entradas ligadas diretamente às saídas são chamadas de redes de perceptrons, e podem facilmente gerar funções booleanas complexas, como por exemplo, a função maioria, que retorna 1 se mais da metade das entradas da rede possuírem valor 1.

Redes que possuem várias camadas funcionam como um agrupamento de redes de perceptrons, uma vez que a camada intermediária recebe os valores de entrada e envia para a próxima camada o que seria a saída do perceptron.

Os algoritmos de treinamento de redes neurais têm como objetivo diminuir uma dada taxa de erro, definida pelo programador, atualizando os valores dos pesos das camadas intermediárias. Assim, dado um conjunto de entradas e as saídas esperadas, a rede faz várias iterações, chamadas de épocas, até que um critério de parada seja atingido.

## 3.3 Espaço de Cores

### 3.3.1 RGB

Conforme [AZEVEDO and CONCI, 2003], o espaço de cores RGB (Red, Green e Blue) é um sistema de cores que utiliza combinações das três cores primárias para gerar as outras cores. Normalmente, esse espaço de cores é descrito como um cubo, onde cada dimensão é dada por valores que variam entre 0 e 255 e representam uma das três cores.

O problema do sistema RGB é que apesar de representar a biologia do olho humano, que possui três tipos de células-cone que respondem a diferentes espaços no espectro visível, ele não representa efetivamente o sistema como um ser humano entende as cores. Por exemplo, o ser humano não consegue definir o quanto de vermelho existe em uma imagem.

### 3.3.2 HSV

Outro sistema de cores é o HSV (Hue, Saturation e Value). Conforme [AZEVEDO and CONCI, 2003], esse sistema representa melhor o sistema de visão humano, pois utiliza a quantidade de branco e brilho em uma certa cor pura.

O sistema HSV é normalmente representado por um cone ou um duplo-cone, onde:

- Hue (Matiz): Valor que varia entre 0 e 359, representa o ângulo de rotação dentro do cone e abrange todos as cores dentro do espectro visível.
- Saturation (Saturacao): Grau de pureza, ou intensidade da cor. Representada por valores entre 0 e 1, quanto maior o valor, mais pura é a cor e, quanto menor, mais perto a cor se encontra da escala de cinza.
- Value (Valor): Representa o brilho da cor. Também entre valores 0 e 1, quanto menor o valor, mais perto do preto a cor se encontra (sem brilho).

## 3.4 Conversão RGB - HSV

O sistema de Rede Neural(Seção 3.2) utiliza como padrão, o sistema de cores HSV, pois possibilita uma separação de cores mais simples. A câmera utilizada envia um sinal RGB para o computador. Após isso, é feita a conversão do RGB para o HSV.

Portanto, deve-se converter os valores recebidos antes de treinar a rede. De acordo com [AZEVEDO and CONCI, 2003], dado um pixel de cor RGB, com R, G e B com valor em 0 e 255, MAX o valor máximo entre R, G e B e MIN o valor mínimo entre R, G e B, temos:

$$r = \frac{R}{R+G+B},\ g = \frac{G}{R+G+B},\ b = \frac{B}{R+G+B} \tag{3.1}$$

Com os valores r, g e b normalizados, descobrimos os valores de HSV:

- H: Se $b \leq g$ sendo $h \in [0, \pi]$ deve ser utilizada a equação 3.2. Se $b > g$ sendo $h \in [0, 2\pi]$ deve ser utilizada a equação 3.3.
- S: Deve ser utilizada a equação 3.4 para $S \in [0, 1]$
- I: Deve ser utilizada a equação 3.5 para $I \in [0, 1]$

$$h = \cos^{-1}\{\frac{0.5 \cdot [(r-g)+(r-b)]}{(r-g)^2+(r-b)(g-b)]^{0.5}}\} \tag{3.2}$$

$$h = 2\pi - \cos^{-1}\{\frac{0.5.[(r-g)+(r-b)]}{(r-g)^2+(r-b)(g-b)]^{0.5}}\} \tag{3.3}$$

$$s = 1 - 3.min(r,g,b) \tag{3.4}$$

$$i = \frac{(R+G+B)}{(3.255)} \tag{3.5}$$

Sendo assim, transformasse os valores normalizados nos seguintes intervalos:

- H = [0,360]; $H = \frac{h \times 180}{\pi}$
- S = [0,100]; $S = s \times 100$
- I = [0,255]; $I = i \times 255$

## 3.5 Conceitos Físicos

Nesse projeto serão aplicados conceitos físicos para que a simulação física envolvida em um jogo de sinuca seja o mais próximo possível.

[HALLIDAY et al., 1992] e [KELLER et al., 1997] foram referências básicas quanto à utilização destes conceitos em todo esse Capítulo.

### 3.5.1 Velocidade

Velocidade ($V$) é uma grandeza vetorial, expressada pela razão entre o deslocamento de um corpo e o intervalo de tempo utilizado para esse deslocamento ser realizado. Representa a rapidez com qual um corpo altera sua posição na trajetoria.

A Equação (3.6) representa a velocidade média

$$\vec{v}_m = \frac{\Delta\vec{s}}{\Delta t} \tag{3.6}$$

Na qual $v_m$ é velocidade média, expressa em metros por segundo (m/s), $\Delta\vec{s}$ é a variação de deslocamento e $\Delta t$ é a variação do tempo deste deslocamento.

### 3.5.2 Aceleração

Aceleração ($A$) é uma grandeza vetorial, expressa pela razão entre a velocidade de um corpo e o intervalo de tempo utilizado para a taxa de variação dessa velocidade. Representa a rapidez com qual a velocidade de um corpo varia.

Existe aceleração em um movimento quando este é variado, ou seja, quando há variação de velocidade ao longo do tempo.

Movimentos acelerados apresentam aumento da velocidade, e os vetores de aceleração $\vec{a}$ e velocidade $\vec{v}$ têm o mesmo sentido, já os movimentos retardados apresentam uma diminuição da velocidade e os vetores aceleração $\vec{a}$ e velocidade $\vec{v}$ têm sentidos opostos.

A Equação (3.7) representa a aceleração média:

$$\vec{a}_m = \frac{\Delta\vec{v}}{\Delta t} \tag{3.7}$$

Onde, $a_m$ é aceleração média, expressa em metros por segundo ao quadrado ($m/s^2$) e $\Delta v$ é a variação de velocidade em um intervalo de tempo $\Delta t$.

### 3.5.3 Atrito

Atrito é a força que atua sobre um objeto quando este está em contato com outro e este se encontra em movimento.

Esta força é causada pelo contato entre dois corpos ou o meio com o qual o corpo em movimento interage.

Para existir a força de atrito deve haver movimentos relativos entre os corpos em contato (atrito cinético), ou pelo menos a tendência de um se mover em relação ao outro (atrito

estático) graças à ação de outras forças, externas a eles aplicadas.

A intensidade do atrito depende da Força Normal entre o objeto e a superfície, no qual quanto maior a Força Normal, maior é a intensidade do atrito.

A energia dissipada pelo atrito é completamente convertida em energia térmica que leva ao aumento da temperatura dos corpos em atrito.

### 3.5.4 Coeficiente de Atrito

É uma grandeza que não apresenta unidade e que representa o grau de rugosidade entre dois corpos.

Pode ser diferenciado de acordo com a situação que se encontra o sistema, em dinâmico ou estático:

- Coeficiente de atrito dinâmico ($\mu d$): presente a partir do momento que os corpos em contato apresentam deslocamento.
- Coeficiente de atrito estático ($\mu e$): presente a partir do momento que os corpos em contato encontram-se em iminência de movimento, ou seja, não em movimento. Relaciona a máxima força de atrito possível (com os corpos ainda estáticos) com as forças normais a eles aplicados.

Realizando a comparação dos módulos de cada um implica que o coeficiente de atrito dinâmico ser menor ou igual ao coeficiente de atrito estático ($\mu d < \mu e$.)

### 3.5.5 Coeficiente de Atrito Estático

A Força de Atrito é aquela que se opõem ao deslizamento entre dois corpos, assim, pode-se concluir que existe uma força que atua contra o movimento durante a interação. Para mover um corpo, deve ser aplicada uma força maior que a do atrito dinâmico. Caso contrário as forças irão se anular e o corpo não irá sair do lugar.

A força de atrito estático é maior que a de atrito dinâmico, mas na maioria dos casos seus valores são tão próximos que podemos considerá-las praticamente iguais e, por isso, sua força pode ser definida pela Equação (3.8), análoga a do atrito dinâmico, dada por:

$$F_{at} = \mu e \cdot N \tag{3.8}$$

Onde $F_{at}$ é a força de atrito expressa em Newton (N), $\mu e$ é o coeficiente de atrito estático do corpo e $N$ a força que é normal à direção do movimento.

### 3.5.6 Coeficiente de Atrito Cinético ou Dinâmico

É a força que surge entre os corpos que apresentam movimento, se opondo a este movimento (deslizamento) entre os corpos. Esta força não é necessariamente oposta ao movimento do corpo.

Por exemplo, quando um corpo está deslizando sobre uma superfície horizontal em uma determinada direção, a força de atrito dinâmico estará aplicada à superfície de contato do objeto e à superfície de apoio paralelamente à superfície e apontando para o sentido contrario ao do movimento.

Caso o objeto esteja deslizando, a força de atrito entre o corpo e à superfície será dinâmica, se opondo ao deslizamento, que nesse caso coincide com a oposição ao movimento do objeto. A Equação (3.9) representa a força de atrito dinâmico:

$$F_{at} = \mu d \cdot N \tag{3.9}$$

Onde $F_{at}$ é a força de atrito expressa em Newton (N), $\mu d$ é o coeficiente de atrito dinâmico do corpo e $N$ a força que é normal à direção do movimento.

No caso do corpo estar em um plano horizontal, tem a mesma intensidade do peso do corpo. Com isso, utiliza a Equação (3.10):

$$N = m \cdot g \tag{3.10}$$

Onde $N$ é a força normal à direção do movimento, expressa em Newton ($N$), $m$ é a massa do corpo e $g$ é a aceleração do campo gravitacional.

### 3.5.7 Energia Cinética

A variação de energia cinética é a quantidade de trabalho que teve que ser realizado sobre um corpo para que sua velocidade seja alterada (seja a partir do repouso, ou seja, a partir de uma velocidade inicial).

A Equação (3.11) representa a energia cinética

$$E_c = \frac{mv^2}{2} \tag{3.11}$$

Onde $E_c$ é energia cinética, expressa em jaules ($J$), $m$ é a massa do corpo, $v$ é a variação da velocidade do corpo em um instante de tempo.

### 3.5.8 Movimentos

Movimento retilíneo é aquele movimento em que o corpo se desloca apenas em trajetórias retas. Para tanto, ou a velocidade se mantém constante ou a variação da velocidade varia somente em módulo, nunca em direção.

A aceleração, se variar, também variará apenas em módulo e nunca em direção, e deverá orientar-se sempre em paralelo com a velocidade.

Como variações dos movimentos podem-se citar o movimento retilíneo uniforme e o movimento retilíneo uniformemente variado.

#### 3.5.8.1 Movimento Retilíneo Uniforme (MRU)

No movimento retilíneo uniforme (MRU), o vetor velocidade é constante no decorrer do tempo (não varia em módulo, sentido ou direção), e, portanto a aceleração é nula. O corpo desloca distâncias iguais em intervalos de tempo iguais, sendo que, não tendo aceleração sobre qualquer corpo em MRU, a resultante das forças aplicadas é nula (primeira lei de

Newton - Lei da Inércia).

A principal característica do MRU é que sua velocidade em qualquer instante é igual à velocidade média.

#### 3.5.8.2 Movimento Retilíneo Uniformemente Variado (MRUV)

No movimento retilíneo uniformemente variado (MRUV) ou movimento uniformemente variado (MUV), é aquele em que o corpo sofre aceleração constante, mudando de velocidade num dado incremento ou decremento conhecido. Para que o movimento ainda seja retilíneo, a aceleração deve ter a mesma direção da velocidade.

Caso a aceleração tenha o mesmo sentido da velocidade, o movimento pode ser chamado de Movimento Retilíneo Uniformemente Acelerado. Caso a aceleração tenha sentido contrário da velocidade, o movimento pode ser chamado de Movimento Retilíneo Uniformemente Retardado.

O movimento retilíneo pode ainda variar sem uma ordem muito clara, quando a aceleração não for constante.

#### 3.5.8.3 Equações dos Movimentos Retilíneos

As equações utilizadas para descrever o movimento retilíneo são:

$$\vec{v}_m = \frac{\Delta \vec{s}}{\Delta t} \tag{3.12}$$

$$\vec{a}_m = \frac{\Delta \vec{v}}{\Delta t} \tag{3.13}$$

Na Equação (3.12), $v_m$ é velocidade média, expressa em metros por segundo ($m/s$), $\Delta s$ é a variação de deslocamento e $\Delta t$ é a variação do tempo deste deslocamento.

Na Equação (3.13), $a_m$ é a aceleração média, expressa em metros por segundo ao quadrado ($m/s^2$), $\Delta v$ é a variação de velocidade em um intervalo de tempo $\Delta t$.

#### 3.5.8.4 Equação horária de posição para o MRU

As equações utilizadas para descrever o movimento retilíneo uniforme são:

$$v_m = \frac{s - s_0}{\Delta t} \tag{3.14}$$

$$v_m \Delta t = s - s_0 \tag{3.15}$$

$$s = s_0 + v\Delta t \tag{3.16}$$

Onde $s$ é o deslocamento do corpo expresso em metros $(m)$, $s_0$ é posição inicial de um corpo, $v$ é a velocidade média de um corpo em movimento e a $\Delta t$ é a variação do tempo $t$.

#### 3.5.8.5 Equação horária de posição para o MRUV

Permite determinar a posição do corpo após um intervalo de tempo $\Delta t$ , temos a Equação (3.17)

$$s = s_0 + v_0 \Delta t + \frac{a \cdot \Delta t^2}{2} \tag{3.17}$$

Onde $s$ é o deslocamento do corpo a determinar, expresso em metros $(m)$, $s_0$ e $v_0$ são a posição e a velocidade do corpo no instante inicial, respectivamente. $\Delta t$ é a variação do tempo $t$ e $a$ é aceleração média do corpo no instante do tempo $t$.

#### 3.5.8.6 Velocidade no MRUV após um intervalo de tempo $\Delta t$

A Equação (3.18) é utilizada para descrever velocidade no movimento retilíneo uniformemente variado após um intervalo de tempo $\Delta t$.

$$v = v_0 + a \cdot \Delta t \tag{3.18}$$

Onde $v$ é a velocidade do corpo a determinar, expresso em metros por segundos $(m/s)$, e a velocidade do corpo no instante inicial. $\Delta t$ é a variação do tempo $t$ e $a$ é aceleração média do corpo no instante do tempo $t$.

### 3.5.9 Colisão Elástica

É uma colisão entre dois ou mais corpos onde estes não sofrem deformações permanentes durante o impacto.

Em uma colisão elástica se conservam tanto o momento linear como a energia cinética do sistema, e não há troca de massa entre os corpos, que se separam depois da colisão.

### 3.5.10 Intersecção de Reta e Circunferência

Para encontrar os pontos onde uma reta intercepta uma circunferência, utiliza-se a Equação Geral da Reta (Equação 3.19) e a Equação Reduzida da Circunferência(Equação 3.20).

$$ax + by + c = 0 \tag{3.19}$$

$$(x - a)^2 + (y - b)^2 = R^2 \tag{3.20}$$

Considerando essas equações, assumimos o Sistema de Equações da Figura 3.3:

$$\begin{cases} ax+by+c = 0 \\ (x - a)^2 + (y - b)^2 = R^2 \end{cases}$$

Figura 3.3: Sistema de Equações

O Sistema resulta em uma equação de segundo grau onde, analisando o seu discriminante DELTA, é possível determinar a posição da reta em relação à circunferência.

- $\Delta > 0$ reta secante à circunferência (r1).
- $\Delta = 0$ reta secante à circunferência (r2).
- $\Delta < 0$ reta não possui intersecção com a circunferência (r3).

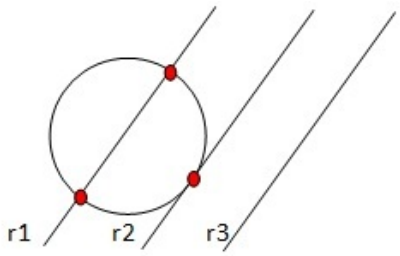


Figura 3.4: Intersecção da Reta com a Circunferência

Fonte: Os autores

Com o uso da Figura 3.4, podemos mostrar que as soluções da equação de segundo grau serão os pontos de intersecção da reta com a circunferência.

## 3.6 Open CV e Open GL

Essa subseção irá apresentar os conceitos básicos as bibliotecas Open CV e Open GL que irão auxiliar no melhor entendimento do mesmo.Todos esses conceitos foram baseados no [AZEVEDO and CONCI, 2003]

### 3.6.1 Open GL

Open GL® (Open Graphics Library), assim como Direct3d® ou Glide®, é uma API (Application Programming Interface) para desenvolvimento e processamento gráfico de ambientes em 2d e 3d. Entre as bibliotecas gráficas, o Open GL se destaca por sua versatilidade, possibilitando o desenvolvimento para diversas plataformas (diferente do Glide®) e diversos sistemas operacionais (diferente do Direct3d®, que possui restrição ao sistema operacional da Microsoft®). O Open GL® possui diversas aplicações como jogos, aplicações de modelagem 3d e processamento de superfícies matemáticas.

### 3.6.2 Open CV

Open CV (Open Source Computer Vision) é uma biblioteca para desenvolvimento de aplicações que utilizam visão computacional. A biblioteca dispõe de algoritmos de alto-desempenho como filtros de imagem, reconhecimento de objetos, calibração de câmeras, análise estrutural e outros. Originalmente desenvolvida pela Intel® em 2000, é uma biblioteca multiplataforma totalmente livre para uso acadêmico e comercial.

# Capítulo 4

# Metodologia

A idéia geral do trabalho é utilizar uma rede neural para reconhecimento de padrões de cores, que serão utilizados como marcador colorido posicionado em um taco (real), possibilitando assim a identificação da posição e dos movimentos do taco, informações essas que serão transmitidas para um outro computador, em rede, que irá calcular as interações físicas dos elementos (virtuais) e projetará uma simulação gráfica em uma mesa (real).

Sendo assim, o sistema é composto de três grandes módulos:

No primeiro módulo, a rede neural será calibrada utilizando exemplos de treinamento e o tamanho do campo de jogo será definido utilizando o cursor do mouse, clicando-se em dois pontos distintos, formando assim uma figura retangular. Esse módulo será executado apenas quando o software é iniciado (offline).

O segundo é o módulo responsável pela leitura dos frames capturados pela câmera e classificação dos pixels, conforme matriz determinada pela rede neural, identificando assim a posição dos marcadores e determinando a posição do taco. Esse módulo é executado durante o jogo (online).

O terceiro é o módulo responsável pelo cálculo das interações físicas, pela validação da jogada, mediante as regras, e pela projeção dos elementos gráficos (bolas, caçapas, acabamentos da mesa e etc.). Esse módulo é executado durante o jogo (online).

A arquitetura do projeto é apresentada na Figura 4.1.

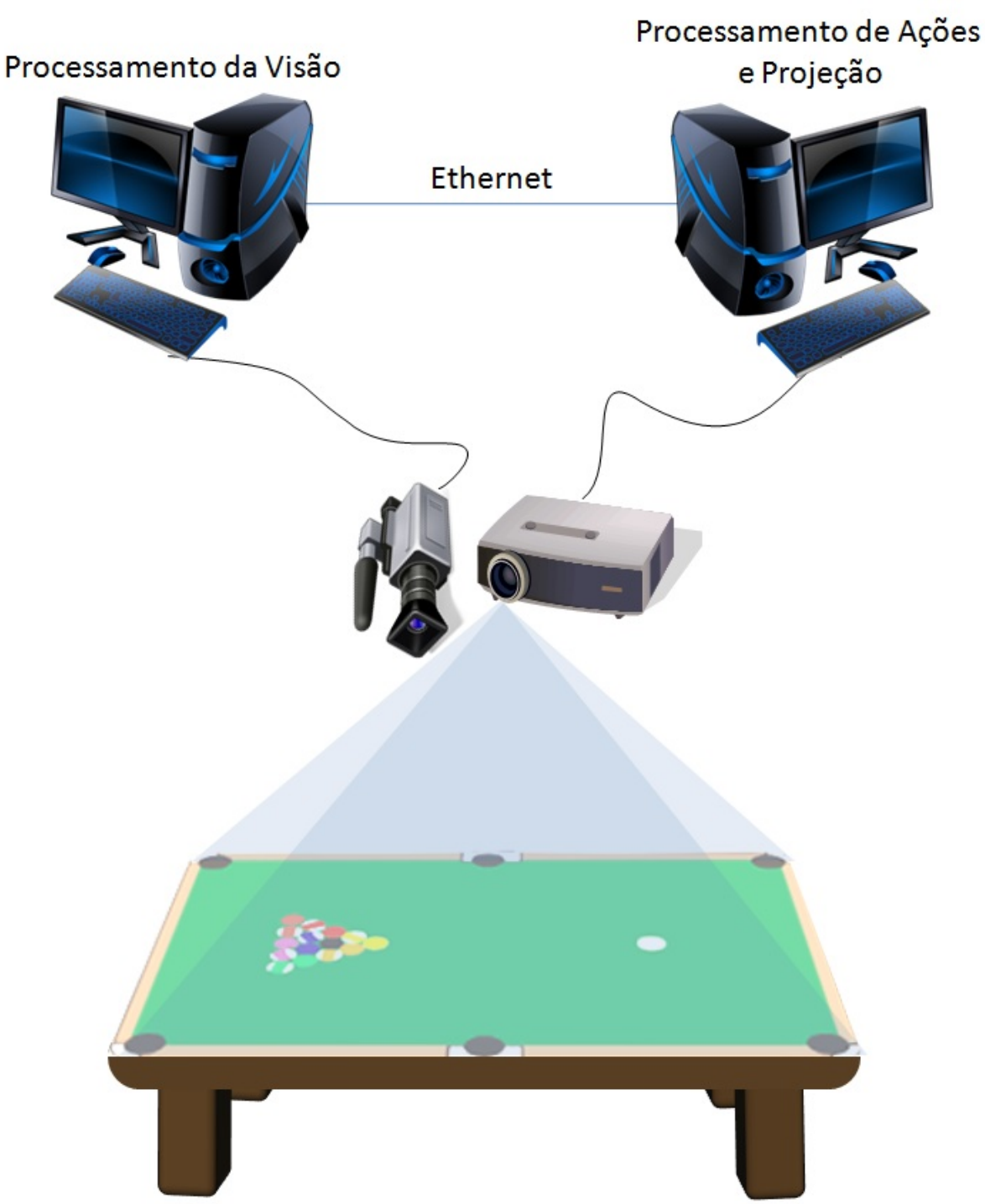


Figura 4.1: Arquitetura do Projeto

Fonte: Os autores

Assim, a metodologia do projeto é baseada na Figura 4.2.

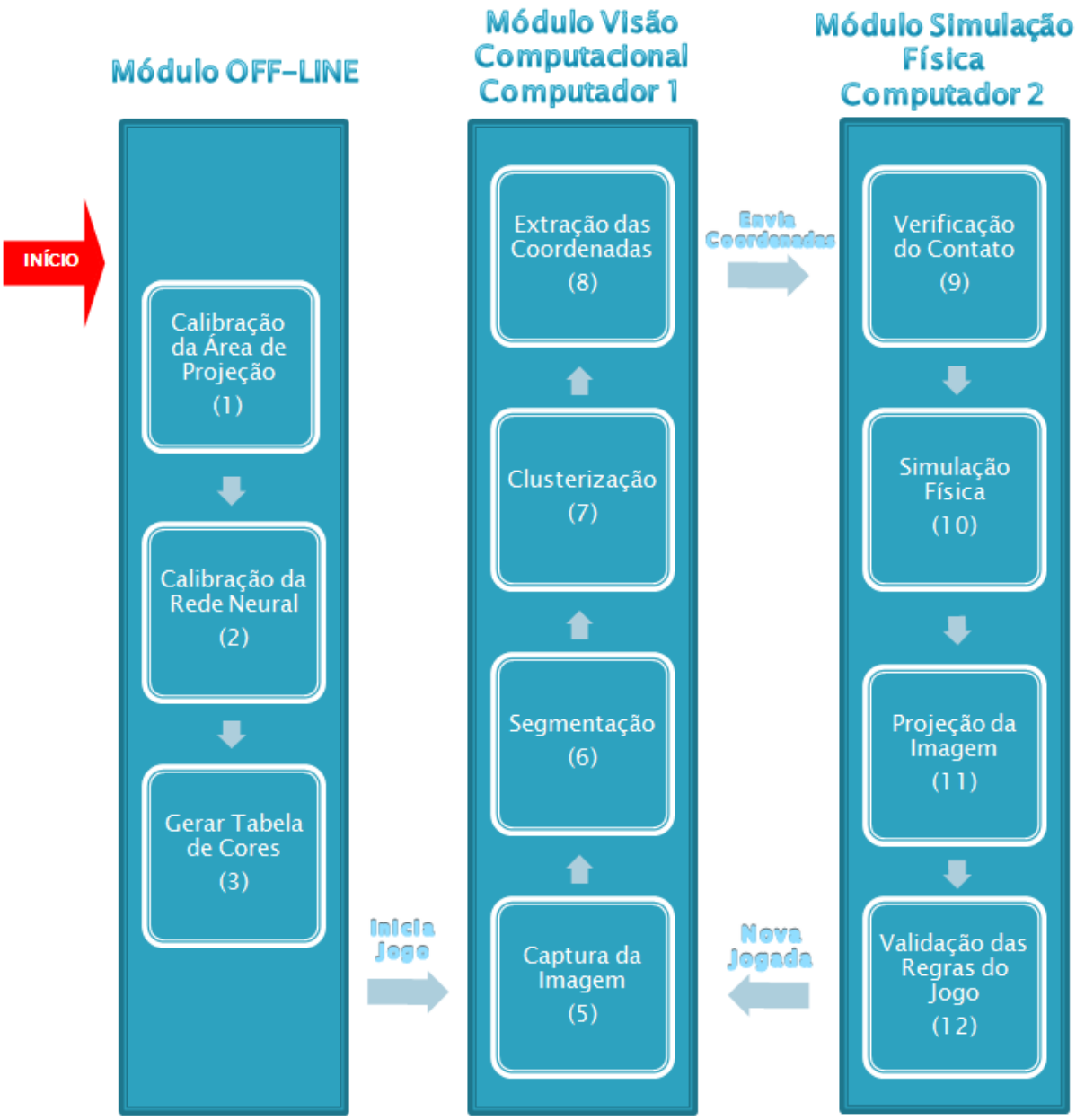


Figura 4.2: Metodologia

Fonte: Os autores

## 4.1 Módulo OFF-LINE

### 4.1.1 Calibração de Área de Projeção (Figura 4.2-1)

A primeira etapa do módulo off-line é a calibração dá área de projeção da mesa de jogo.

Para essa calibração, o usuário deverá usar o mouse do computador para selecionar os pontos de uma diagonal da mesa. Como a mesa é um objeto simétrico, os outros dois pontos são encontrados da seguinte forma:

Pontos de Controle: P1($x1$,$y1$) P2($x2$,$y2$)

Pontos Adicionais: P3($x1$,$y2$) P4($x2$,$y1$)

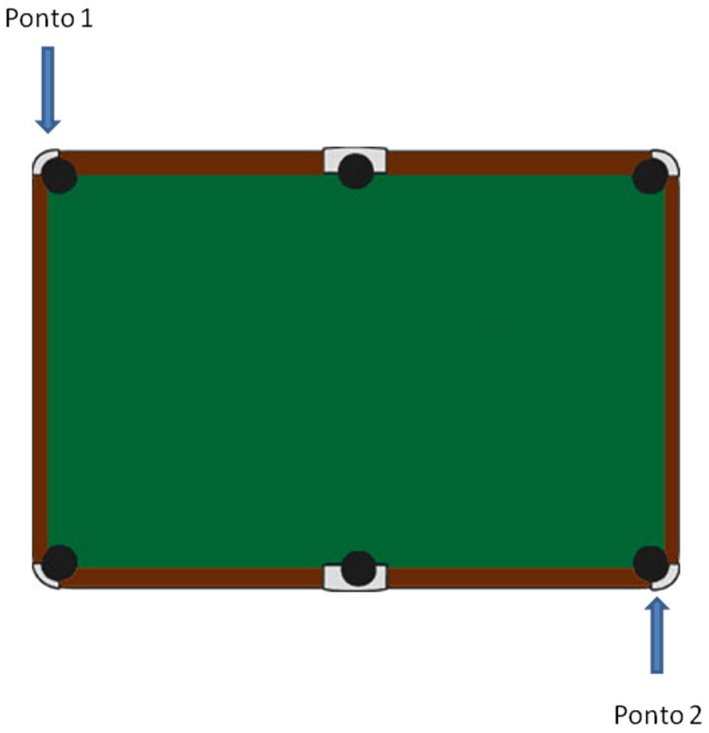


Figura 4.3: Configurando o Espaço da Mesa

Fonte: Os autores

### 4.1.2 Calibração da Rede Neural (Figura 4.2-2)

Para calibrar a rede neural, serão colocadas marcações coloridas espalhadas por todo o campo de visão da câmera. Com isso a rede será calibrada e tenderá a ser robusta em relação aos diferentes níveis de iluminação encontrados no ambiente de jogo.

O usuário deverá selecionar, na tela do sistema, a cor que se quer treinar e, em seguida, clicar nos pontos relativos a essa cor na imagem.

Depois de selecionados todos os pontos de treinamento, o usuário deverá iniciar o treinamento da rede, que irá utilizar as funções de aprendizado da biblioteca OpenCV.

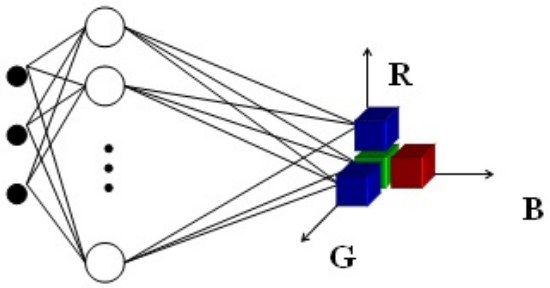


Figura 4.4: Treinamento da Imagem utilizando Redes Neurais

Fonte: Os autores

### 4.1.3 Gerar Tabela De Cores (Figura 4.2-3)

Com a rede treinada, será gerada uma matriz de 256x256x256 posições. Com a resposta da rede para cada valor de RGB existente.

Apesar de consumir uma quantidade considerável de memória, essa matriz é importante pois, de outra maneira, seria necessário executar a rede com os valores RGB de todos os pixels de todos os frames, o que pode consumir um tempo grande de processamento.

## 4.2 Módulo ON-LINE

### 4.2.1 Captura da Imagem (Figura 4.2-5)

A câmera escolhida envia os frames para o computador através de uma interface FireWire. Os frames devem ser capturados e armazenados na memória para o tratamento da imagem e localização do taco.

### 4.2.2 Segmentação (Figura 4.2-6)

Com o frame capturado, a imagem será segmentada utilizando a matriz pré-populada pela rede neural, gerando assim uma imagem onde os pixels não possuem valores RGB, mas a cor definida pela rede neural (Ex: Amarelo, Rosa, Preto).

### 4.2.3 Clusterização (Figura 4.2-7)

Após os pixels serem classificados, o algoritmo de Run Length Encoded (RLE) será executado para agrupar pixels adjacentes em blocos e, posteriormente, agrupar os blocos em regiões.

O primeiro passo do algoritmo é agrupar os pixels vizinhos horizontalmente. Isso é util pois as próximas etapas do sistema podem operar sobre os grupos como um todo, e não sobre pixels individuais. Portanto, as regiões de cada cor precisam apenas procurar por conexões verticais, pois os vizinhos horizontais já estão conectados pelo RLE. Cada bloco gerado pelo RLE tem como parâmetro o seu nó-pai.

Na primeira iteração do algoritmo, todos os blocos têm a si próprios como nó pai, gerando uma floresta totalmente desconexa (nenhum bloco tem um filho além de si mesmo). O processo de união varre as linhas adjacentes a procura de blocos que sejam da mesma cor e que satisfaçam o critério de four-connectedness, que determina que sejam classificados como vizinhos os pixels que sejam adjacentes vertical ou horizontalmente, desconsiderando as diagonais.

Após as conexões serem encontradas, os blocos que formam uma região recebem como nó-pai o bloco mais ao alto e a esquerda que faça parte da região. Após essas duas iterações, o algoritmo é executado novamente para garantir que todos os blocos pertencentes a uma região tenham o mesmo nó-pai.

A Figura 4.5 representa um Algorítimo RLE.

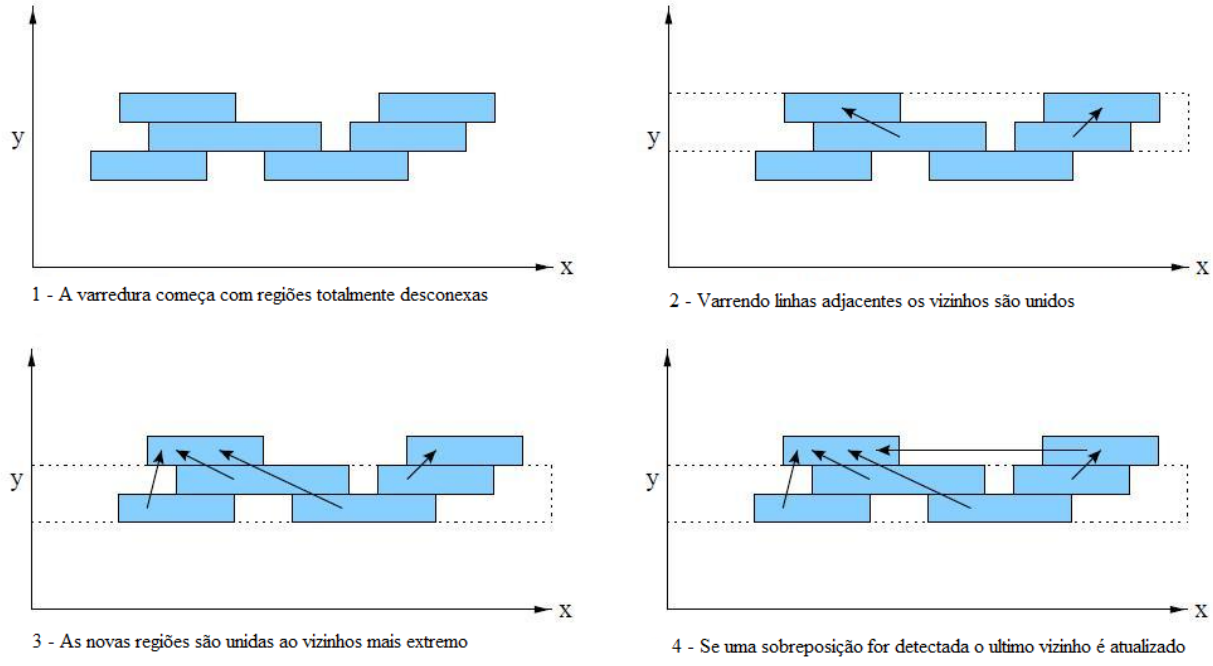


Figura 4.5: Exemplo do Algorítimo RLE

Fonte: [BRUCE et al., 2000]

### 4.2.4 Extração das Coordenadas (Figura 4.2-8)

Para o taco ser detectado, ele irá receber duas marcações coloridas. Essas marcações não apenas servirão para detectar a posição do taco, mas também para detectar sua direção.

Com isso, depois de definidas as regiões da cor das marcações do taco, será aplicado um filtro confiança gaussiana na posição, para que se escolha as regiões mais próximas do taco localizado no frame anterior.

De posse dessas informações, o módulo de visão irá enviar um pacote através da rede para o módulo de simulação física contendo o vetor de posição do taco.

### 4.2.5 Verificação de Contato (Figura 4.2-9)

Com as coordenadas do taco, o sistema irá verificar se a ponta do taco interceptou a tacadeira. Essa verificação ocorrerá de frame em frame, até a colisão ser detectada.

A partir desse momento, o sistema deixa de considerar as interações do taco e inicia a simulação de movimento das bolas dentro da mesa de jogo.

Após todos os movimentos serem completamente simulados, o sistema irá aguardar um novo contato entre o taco e a tacadeira.

### 4.2.6 Simulação Física (Figura 4.2-10)

Nessa seção são explicados os conceitos de física aplicados na simulação do movimentos das bolas do jogo. Após receber o pacote do módulo de visão, o módulo de simulação física inicia os cálculos e atualização das posições das bolas.

Para facilitar o entendimento, as interações de uma bola com o ambiente foram divididas em etapas:

#### 4.2.6.1 Simulação Bola - Taco

Nesse trabalho, sera modelado a movimentação da bola passando a energia cinética do taco, no momento da primeria colisão, para a bola. Sendo assim, utiliza-se a energia cinética do taco da seguinte forma:

$$E_{taco} = \frac{1}{2} \cdot m_t V^2 \tag{4.1}$$

Na qual, $m_t$ é a massa do taco e $V$ é a velocidade que ele atinge a bola, conforme a Figura 4.6

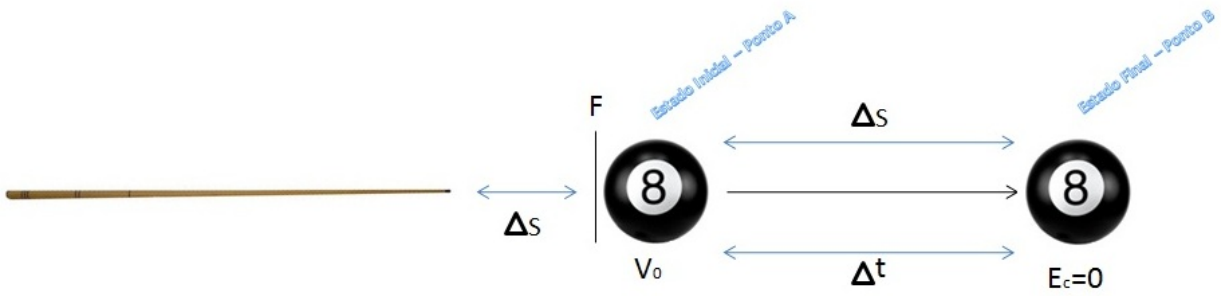


Figura 4.6: Transformação da Energia Cinética do Taco para a Bola

Fonte: Os Autores

Sendo assim, para calcular a distância percorrida do ponto A até B, resolve a Equação (4.1) para $V$. O espaço $\Delta S$ percorrido pela bola é então calculado segundo a Equação (4.3).

$$\Delta S = S_0 + V_0 t + \frac{1}{2} a t^2 \tag{4.2}$$

Para isso, utilizaremos o Algorítimo 1:

**while** $V_0 \geq 0$ **do**

$\Delta S = S_0 + V_0 t + \frac{1}{2} a t^2$;

$S_0 = \Delta S$;

$V = V_0 + a \Delta t$;

**end**

**Algorithm 1**: Algorítimo de movimentação das Bolas

#### 4.2.6.2 Simulação Bola - Bola

Durante a colisão, os componentes da velocidade tangencial não mudam. Usando os componentes da normal, tratamos a colisão como uni-dimensional. Isto é descrito na Figura 4.7. As setas vermelhas representam os vetores velocidade imediatamente antes da colisão. As setas azuis representam as compontentes normais dos vetores e as setas verdes representam o componente vetorial tangencial. A linha pontilhada é o plano da colisão.

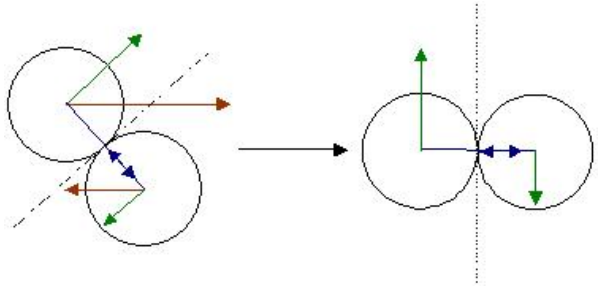

Figura 4.7: Momento da Colisão Bola - Bola

Fonte: Os autores

Nesse trabalho, considera-se a colisão entre as bolas da seguinte maneira (desprezando os efeitos na bola):

- A bola A possui energia cinética dada pela Equação (4.3)

$$E_{bola} = \frac{1}{2} \cdot m_b V^2 \tag{4.3}$$

Em um sistema perfeito elástico, de mesma massa, a colisão entre os corpos é feita da seguinte maneira:

- A energia cinética da Bola A é passada para a Bola B
- O total da energia cinética de saída deve ser igual a energia cinética de entrada.

Sendo assim, utilizando os conceitos de Conservação de Energia, será feito o seguinte processo para identificar as novas velocidades das bolas:

- Encontra o diferença das posições em X e Y para poder calcular a arcotangente dos pontos.
- Com o ângulo encontrado, é rotacionado o vetor velocidade das duas bolas em relação ao ângulo.

- Após a rotação, é feita a troca em a posição X das bolas. A posição X da bola 1 é a posição X da bola 2 e vice versa.
- Com essa troca efetuada, os vetores são rotacionados negativamente com o ângulo encontrado, fazendo com que as bolas sigam a direção e velocidade certa.

#### 4.2.6.3 Bola - Tabela

Seja o esquema de colisão mostrado na Figura 4.8

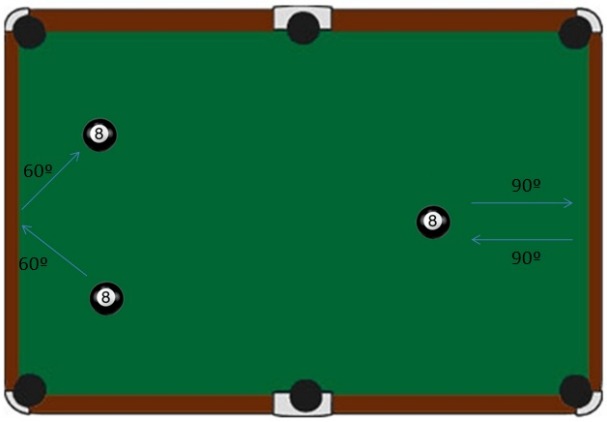


Figura 4.8: Momento da Colisão Bola - Tabela

Fonte: Os Autores

No momento da colisão entre a bola e a tabela, foi considerado um modelo de uma perda de velocidade utlizando a Equação (4.4)

$$V_f = V_i(1 - \mu \sin(\theta)) \tag{4.4}$$

Sendo que $\mu$ é uma constante entre 0 e 1 que será calculado empiricamente, $\theta$ é ângulo entre o centro da bola e a tabela. Com isso, a perda é variada de acordo com o ângulo da colisão.

Para calcular o sentido da bola após a colisão, será utilizado o mesmo ângulo de entrada da batida. Como exemplo, se o ângulo de colisão for 60º na entrada, a bola sairá com o mesmo ângulo de 60º, de acordo com a Figura 4.8.

### 4.2.7 Projeção da Imagem (Figura 4.2-11)

Essa seção apresenta a simulação e projeção do sistema.

Após o início da tacada, a bola irá percorrer uma distância $\Delta S$ que é calculada no algorítimo 1 (Seção 4.2.6.1), que calcula a ocorrência de uma colisão entre bolas ou entre bola e a tabela da mesa.

Caso ocorra a colisão, ela será calculada conforme explicado na seção 4.2.6. A saída da colisão são as seguintes informações: Posição da Bola, Velocidade da Bola e Direção da Bola.

Com isso, será projetado, em um tempo $\Delta t$ que será definio mais tarde, para o usuário verificar a simulação.

Em conjunto com todas as verificações, será verificado se a bola atingiu a mesma posição da caçapa. Se ela atingir, a bola será excluída da simulação, e com isso, não irá ser projetada no sistema e também não irá participar da simulação física.

### 4.2.8 Validação das Regras do Jogo e Simulação (Figura 4.2-12)

Neste trabalho serão consideradas variações das regras oficiais, definidas na seção 3, com o objetivo de tornar o jogo mais competitivo, divertido e próximo do praticado popularmente, para isso, a Figura 4.9 representa um fluxograma de eventos.

No início do jogo as bolas se encontram na posição inicial (Figura 4.9-1). Quando realizada a primeira tacada (Figura 4.9-2), é necessária a verificação das interações da tacadeira (Figura 4.9-3). É considerado falta caso não ocorra contato da tacadeira com uma bola qualquer, ou a tacadeira seja encaçapada. Nesse caso, a bola tacadeira retorna a posição inicial e a vez passa ao próximo jogador.

Em qualquer momento do jogo, é permitida a interação da tacadeira com qualquer bola da mesa, desde que não definido o domínio de jogo de cada jogador, caso não ocorra, é considerado falta e o jogador é punido com uma rodada sem jogar.

Um domínio é definido no momento em que uma bola é encaçapada, sendo de responsabilidade da equipe que realizou a tacada, o domínio do qual a primeira bola encaçapada

pertence.

A partir deste momento, todas as interações entre a tacadeira e a primeira bola atingida são verificadas (Figura 4.9-4) e atualizadas suas posições (Figura 4.9-5). Caso a tacadeira colida primeiramente com uma bola do domínio do adversário, não colida com nenhuma bola ou seja encaçapada, é considerado falta, punida com o encaçapamento automático da bola de menor numeração do adversário.

A bola 8 não pertence a nenhum domínio, mas caso atingida primeiramente pela tacadeira e ainda existirem bolas do domínio do jogador responsável pela tacada na mesa, é considerado falta com punição conforme definido acima. Caso a bola 8 seja encaçapada nessas condições, o jogo termina e o jogador é punido com a derrota.

Caso uma jogada seja considerada licita, são computados pontos a cada bola encaçapada.

Quando um jogador encaçapar todas as bolas de seu domínio, tem como regra para vencer o jogo que encaçapar a bola 8, podendo nesse momento realizar a interação entre a tacadeira e a bola 8 diretamente e ser punido com falta caso isso não ocorra (não acertar a tacadeira na bola 8 ou primeiramente acertar uma bola do domínio adversário). O jogo termina quando a bola 8 é encaçapada. Se o jogador que o fizer não cometer falta nessa jogada, é declarado o vencedor. Caso contrário, seu adversário vence o jogo.

Sendo assim, nessa etapa, será analisado toda as regras para poder dar continuidade no jogo (Figura 4.9-6).

### 4.2.9 Fluxograma de Eventos

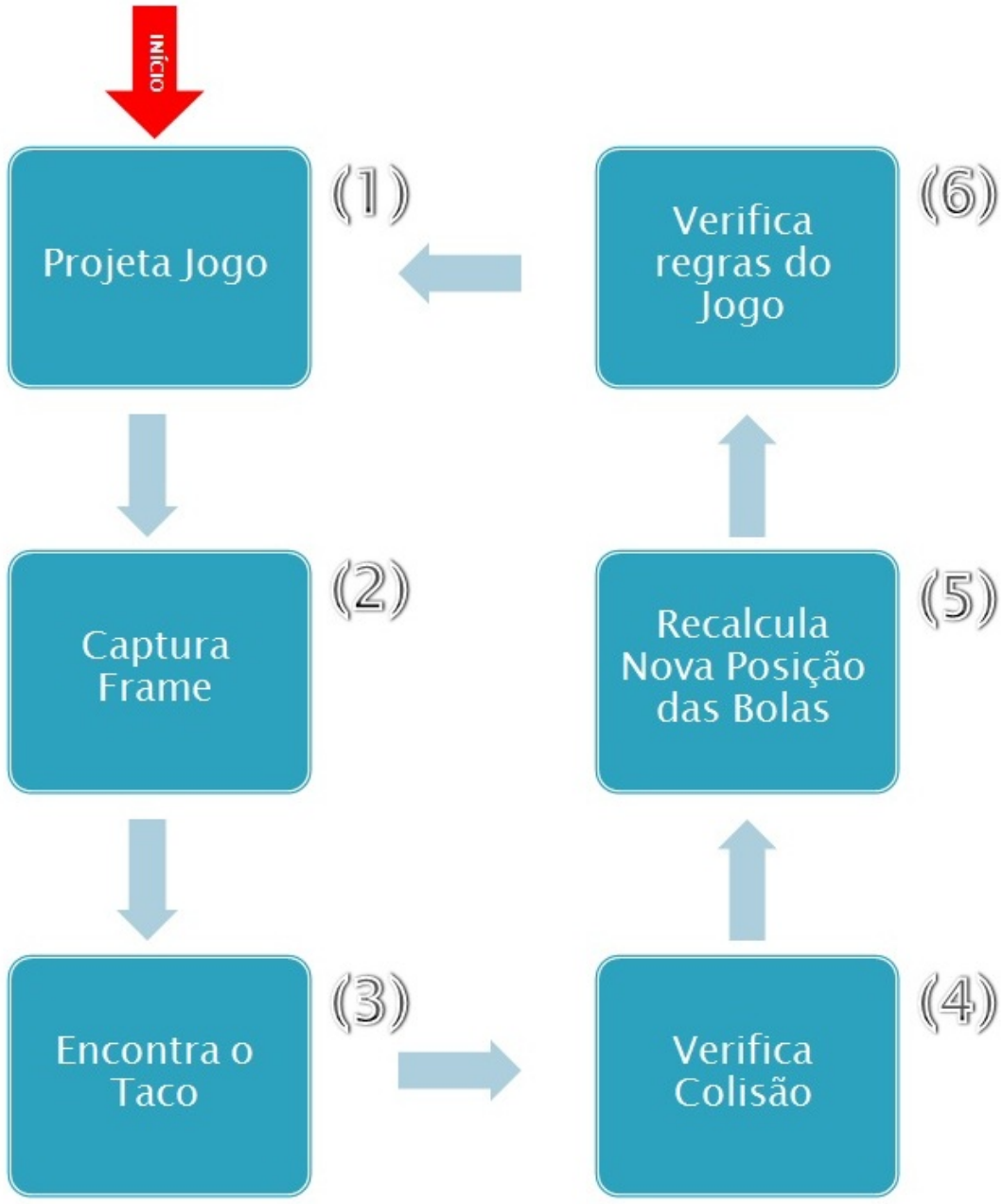


Figura 4.9: Fluxograma de Eventos

Fonte: Os Autores

# Capítulo 5

# Análise e Modelagem

Essa seção irá apresentar os requisitos e a modelagem para o desenvolvimento do projeto. É apresentado o cenário de caso de uso, diagrama de classes e diagrama de sequência do mesmo.

## 5.1 Análise de Requisitos

### 5.1.1 Requisitos

A seguir, na Tabela 5.1 , são apresentados os requisitos de sistema, negócio e usuário.

Tabela 5.1: Requisitos do Projeto Virtual Snooker

| **Código** | **Nome do Requisito** | **Tipo** | **Mudança** |
|---|---|---|---|
| R01 | O sistema pode ser utilizado por diversos jogadores desde que divididos em dois times | Negócio | Estável |
| R02 | A iluminação do ambiente não poderá sofrer alterações bruscas | Sistema | Estável |

| **Código** | **Nome do Requisito** | **Tipo** | **Mudança** |
|---|---|---|---|
| R03 | Após a inicialização do sistema, a estrutura física (mesa, projetor e câmera) não devem ser movidas | Sistema | Estável |
| R04 | O usuário deve evitar movimentos bruscos com taco, movimentos que não sejam de tacada | Sistema/Usuário | Estável |
| R05 | O desempenho do sistema deverá ser otimizado buscando o máximo de realismo | Negócio | Estável |
| R06 | Quando um jogador encaçapar a bola branca, a mesma deverá voltar na posição inicial e o jogador penalizado com falta conforme requisito R07 | Negócio | Estável |
| R07 | Quando um jogador cometer uma falta, a bola de menor numeração do adversário deve ser automaticamente encaçapada | Negócio | Estável |
| R08 | Quando um jogador encaçapar a bola 8 (preta), o jogo deverá acabar. Caso ela seja encaçapada de forma correta, o jogador ou a equipe vencem a partida, caso encaçapada com falta, o jogador ou equipe perdem a partida | Negócio | Estável |
| R09 | Todas as bolas do domínio do jogador devem ser encaçapadas antes da bola 8 (preta) | Negócio | Estável |

| **Código** | **Nome do Requisito** | **Tipo** | **Mudança** |
|---|---|---|---|
| R10 | Deverá ser considerado falta não acertar a tacadeira em nenhuma bola, e em caso de domínios definidos, acertar a tacadeira primeiramente em uma bola do domínio do adversário | Negócio | Estável |
| R11 | É necessário, para a utilização do sistema, um projetor, uma câmera e uma mesa plana, de cor branca que não produza reflexos | Sistema | Estável |
| R12 | O projetor e a câmera devem estar fixados sobre a mesa, na posição vertical, a uma altura máxima de 3 metros | Sistema | Volátil Emergente |
| R13 | Movimentos do taco deverão ser despresados após uma tacada até que a simulação termine | Negocio | Estável |
| R14 | O taco deve possuir duas marcações de cores distintas | Sistema | Estável |

Fonte: Os autores

## 5.2 Diagrama de Casos de Usos

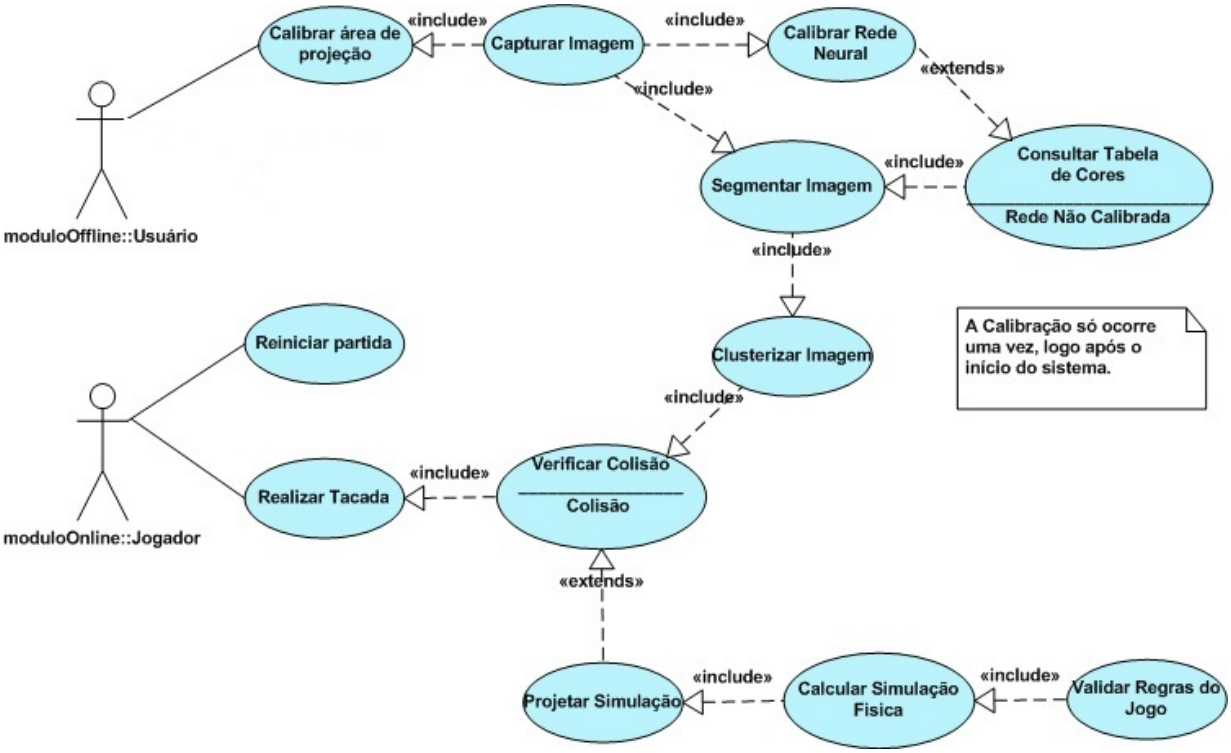


Figura 5.1: Diagrama do Caso de Uso

Fonte: Os autores

### 5.2.1 Modo OFF-LINE

**Caso de Uso Calibrar Área de Projeção.**

**Nome:** Calibrar Área de Projeção.

**Fluxo de Eventos:**

1. Sistema inicia a captura de imagem. (Inclui: Capturar Imagem).
2. Usuário seleciona dois pontos com o mouse formando um retângulo.
3. Sistema define como área de projeção o retângulo selecionado pelo usuário.

**Caso de Uso Clusterizar Imagem.**

**Nome:** Clusterizar Imagem.

**Fluxo de Eventos:**

1. Sistema obtém imagem segmentada.
2. Sistema agrupa um pixel com seus vizinhos de mesma cor determinando uma única forma, definida por tamanho, posição e cor.

**Caso de Uso Segmentar Imagem.**

**Nome:** Segmentar Imagem.

**Fluxo de Eventos:**

1. Sistema inicia a captura de imagem. (Inclui: Capturar Imagem).
2. Sistema varre a imagem pixel a pixel acessando uma tabela que vai determinar a cor que aquele pixel deverá assumir. (Inclui: Consultar Tabela de Cores).

**Caso de Uso Consultar Tabela de Cores.**

**Nome:** Consultar Tabela de Cores.

**Fluxo de Eventos:**

1. Ponto de Extensão: Caso a rede neural não esteja calibrada, sistema calibra a rede neural (Estendido em: Calibrar Rede Neural).
2. Sistema acessa uma tabela utilizando os valores RGB de um pixel como índice.
3. Sistema obtém um valor correspondente a cor que será atribuída para o pixel.

**Caso de Uso Calibrar Rede Neural.**

**Nome:** Calibrar Rede Neural.

**Fluxo de Eventos:**

1. Sistema inicia a captura de imagem. (Inclui: Capturar Imagem).
2. Sistema lista os padrões de cor a serem calibrados.
3. Usuário seleciona padrão de cor a calibrar.
4. Usuário seleciona através de cliques do mouse sobre a imagem capturada da câmera, pixels de exemplo da cor selecionada.

5. Sistema, após todos os padrões exemplificados, treina a rede neural.
6. Sistema gera entradas para todas as combinações RGB possíveis e obtém a saída da rede neural.
7. Sistema armazena a saída da rede neural em uma tabela indexada pelos valores RGB com 256x256x256 posições.

**Caso de Uso Capturar Imagem.**

**Nome:** Capturar Imagem.

**Fluxo de Eventos:**

1. Sistema aciona a câmera de captura de vídeo.
2. Sistema obtém a imagem.

### 5.2.2 Modo ON-LINE

**Caso de Uso Reiniciar Partida.**

**Nome:** Reiniciar Partida.

**Fluxo de Eventos:**

1. Jogador reinicia o jogo.
2. Sistema recoloca as bolas na posição inicial.

**Caso de Uso Realizar Tacada.**

**Nome:** Realizar Tacada.

**Fluxo de Eventos:**

1. Usuário realiza uma tacada.
2. Sistema verifica colisão do taco (real) com a bola (virtual). (Inclui: Verificar Colisão).

**Caso de Uso Verificar Colisão.**

**Nome:** Verificar Colisão.

**Fluxo de Eventos:**

1. Sistema obtém imagem clusterizada.
2. Sistema verifica colisão do taco com uma bola.
3. Ponto de Extensão: Ocorreu colisão (Estendido em: Projetar Simulação).

**Caso de Uso Projetar Simulação.**

**Nome:** Projetar Simulação.

**Fluxo de Eventos:**

1. Sistema atualiza a posição das bolas e verifica interações entre os objetos da mesa (Inclui: Calcular Simulação Física).

**Caso de Uso Calcular Simulação Física.**

**Nome:** Calcular Simulação Física.

**Fluxo de Eventos:**

1. Sistema utiliza as coordenadas do taco para calcular a velocidade e a posição das bolas e as possíveis colisões.
2. Sistema verifica as regras do jogo (Inclui: Validar Regras do Jogo).

**Caso de Uso Validar Regras do Jogo.**

**Nome:** Validar Regras do Jogo.

**Fluxo de Eventos:**

1. Sistema verifica as regras do jogo.
2. Sistema define punições conforme as regras, caso violadas.

## 5.3 Diagramas de Classes

A Figura 5.2 representa a modelagem da aplicação Virtual Snooker. Esta modelagem utiliza os conceitos explicados na seção 4.


Figura 5.2: Diagrama de Classe

Fonte: Os Autores

**Classe Manipulador de Vídeo**

**Atributos:**

- frame: Armazena o frame capturado pela câmera.

**Métodos:**

- capturarImagem(): Captura o frame da câmera.

**Classe ClassificacaoDeCor**

**Atributos:**

- rede: Armazena a rede neural.
- rgb: Armazena as informações de cor em RGB de um dado pixel.
- hsv: Armazena as informações de cor em HSV de um dado pixel.
- tabelaIndice: Tabela utilizada para otimizar as consultas à rede neural.
- conjuntoTreinamento: Vetor que armazena os pontos que irão ser utilizados para treinar rede.

**Métodos:**

- capturarPontos(): Captura os pontos para o treinamento da rede neural usando o callback do mouse.
- treinarRede(): Treina a rede, de acordo com o algoritmo de Back Propagation, usando o o vetor conjuntoTreinamento.
- gerarTabela(): Popula a tabelaIndice com a resposta da rede para todos os valores do espaço RGB.
- classificarPixel(): Classifica um pixel utilizado a rede neural.
- consultarTabela(): Consulta tabela gerada pela rede neural.

**Classe Visão**

**Atributos:**

- imagem: Armazena a imagem capturada pelo manipulador da câmera.
- manipuladorDeVideo: Instância do objeto que captura os frames da câmera.
- classificacaoDeCor: Instância do objeto que irá segmentar a imagem usando a rede neural.
- taco: Armazena as informações do taco como posição X e Y.
- sender: Instância do objeto que faz a conexão com o outro módulo do sistema, módulo de simulação física.

**Métodos:**

- enviarPacote(): Envia o pacote via rede do módulo da visão computacional para o módulo da simulação física.
- clusterizar(): Extrai as informações de regiões de cores.
- identificarTaco(): Utiliza as regiões de cores para tentar localizar o taco.
- escreverSender(): Prepara o pacote para ser enviado via rede.
- receberFrame(): Recebe o frame do manipulador de vídeo.
- segmentar(): Segmenta a imagem ja clusterizada.
- calibrarCampo(): Calibra o campo de jogo.
- calibrarRedeNeural(): Calibra a rede neural.

**Classe Posicao**

**Atributos:**

- x: Armazena a posição X no plano cartesiano.
- y: Armazena a posição Y no plano cartesiano.

**Métodos:**

- Posicao(): Construtor da classe.

- distanciaEntrePontos(): Calcula a distância entre pontos dois pontos do plano cartesiano.

**Classe Taco**

**Atributos:**

- vetorVelocidade: Armazena 50 posições do taco para cálculo da velocidade do mesmo.
- indiceVetor: Armazena o índice atual do vetor. Índice utilizado no vetorVelocidade para identificar em que momento do vetor ocorreu a colisão do taco com a bola branca.
- posicaoTaco: Armazena a posição atual do taco.
- posicaoTacoOld: Armazena a posição anterior do taco.
- velocidadeTaco: Velocidade do taco.
- dist: Armazena a distância entre a posição atual do taco e a posição atual da bola branca.
- distOld: Armazena a distância entre a posição anterior do taco e a posição anterior da bola branca.
- aReta: Armazena o coeficiente angular da reta.
- bReta: Armazena o coeficiente linear da reta
- a: Armazena a variável A da equação da intersecção entre reta e circunferência.
- b: Armazena a variável B da equação da intersecção entre reta e circunferência.
- c: Armazena a variável C da equação da intersecção entre reta e circunferência.
- delta: Armazena a variável responsável por identificar se existe pontos de interseção entre a reta e a circunferência.
- X1: Armazena o limite inferior do segmento de reta. Responsável por identificar se existe pontos de intersecção entre o segmento de reta e circunferência.
- Y1: Armazena o limite inferior do segmento de reta. Responsável por identificar se existe pontos de intersecção entre o segmento de reta e circunferência.
- X2: Armazena o limite superior do segmento de reta. Responsável por identificar se existe pontos de intersecção entre o segmento de reta e circunferência.

- Y2: Armazena o limite superior do segmento de reta. Responsável por identificar se existe pontos de intersecção entre o segmento de reta e circunferência.

**Métodos:**

- Taco(): Contrutor da classe.
- VerificaColisaoTaco(): Verifica se ocorreu colisão entre a o taco e a bola branca, e em caso positivo, chama o método trataColisaoTaco.
- setPosicaoXYSimuladorTaco(): Tranfere as posições do taco capturadas pela classe simulador para a classe taco.
- trataColisaoTaco(): Responsável pelo cálculo referente a velocidade da colisão entre o taco e a bola branca.

**Classe Bola**

**Atributos:**

- posicao: Armazena a posição da bola no plano cartesiano.
- velocidade: Armazena a velocidade da bola.
- inGame: Armazena a situação da bola no jogo. Se a bola está em jogo ou está parada.
- numero: Armazena o número da bola.
- colisaoMesa: Armazena a situação da bola referente a colisao com a mesa. Se a bola colidiu ou não com a mesa.
- raio: Armazena o raio da bola.
- sqrRaio: Armazena o raio da bola ao quadrado.
- idTextura: Armazena a chave para a textura responsável pela bola.
- *imagemTextura: Armazena a imagem da textura da bola.

**Métodos:**

- Bola(): Construtor da classe.
- andaBola(): Anda a bola no plano.

- desenha(): Desenha a bola no plano.
- solidSphere(): Desenha uma esfera sólida no plano.
- verificaColisaoBola(): Verifica a colisão entre duas bolas, e em caso positivo, chama o método trataColisaoBola.
- trataColisaoBola(): Realiza o tratamento da colisão entre duas bolas.
- tradeXSpeedy(): Troca o eixo X do vetor velocidade entre duas bolas colididas.
- rotacionaVetorVelocidade(): Rotaciona o vetor velocidade no eixo X entre as duas bolas.
- isMoving(): Situação de movimento da bola.
- ajuste(): Ajusta colisão de duas bolas, caso as mesmas estejam sobrepostas.
- mataBola(): Defini o estado da bola no jogo. Se a bola está ou não no jogo.

**Classe Arbitro**

**Atributos:**

- bolaPrimeiraColisao: Armazena a primeira colisão entre bolas para retornar a bola branca a sua posição inicial, caso a mesma não tenha colidido.
- endGame: Armazena a situação atual do jogo. Se o jogo está ou não e andamento.
- vencedor: Armazena o vencedor do jogo.
- podeTrocarPlayer: Armazena qual, pela regra do jogo, pode interagir com o jogo.
- mataBola: Verifica se alguma bola deve ser retirada no final da jogada.
- dominioPlayer1: Armazena o domínio de bolas do jogador 1.
- dominioPlayer2: Armazena o domínio de bolas do jogador 2.

**Métodos:**

- Arbitro(): Construtor da classe.
- isEndGame(): Verifica o final do jogo.
- verificaPrimeiraTacada(): Verifica a primeira tacada do jogo.
- setPosicaoInicialBolaBranca(): Inicia a bola branca em sua posição inicial.
- verificaTodasBolasParadas(): Verifica se todas as bolas estão paradas no jogo.

- trataQuedaBolaBranca(): Trata queda da bola branca.
- trataQuedaBola8(): Trata queda da bola 8.
- trocaPlayer(): Troca a vez do jogador.
- mataBolaAdversario(): Retira bola do adversário devido à alguma falta durante a jogada.
- trataQuedaPrimeiraBola(): Trata a queda da primeira bola da jogada.
- mostraVezPlayer(): Defini qual jogador deve realizar a tacada.
- definiDominioJogo(): Defini os domínios dos dos jogadores do jogo.

**Classe Mesa**

**Atributos:**

- objVerticeTabela: Armazena todos os vértices que correspondem a mesa.
- objFaceTabela: Armazena quais os vértices da mesa que formam uma face.
- idTextura: Armazena o índice que corresponde a textura da mesa.
- imagemTexturaFeutro: Armazena a imagem carregada correspondente a textura do feutro da mesa.
- imagemTexturaFeutroTabela: Armazena a imagem carregada correspondente a textura do feutro da tabela da mesa.
- imagemTexturaFeutroMesa: Armazena a imagem carregada correspondente a textura da borda da mesa.

**Métodos:**

- Mesa(): Construtor da classe.
- carregaObj(): Realiza a importação do arquivo .obj para as variáveis vértices e faces.
- verificaColisao(): Verifica se a bola está em alguma posição onde irá colidir com uma tabela e realiza a chamada para o método que tratará a colisão.
- trataColisaoSimples(): Trata as colisões que ocorrem nas tabelas superiores e inferiores. Este método não trata a colisão com a região curva da tabela.

- trataColisaoBorda(): Trata somente as colisões que ocorrem nas regiões curvas das tabelas da mesa.
- trataCaçapa(): Verifica se a bola caiu em alguma caçapa da mesa e, caso positivo, retira a bola de jogo.

**Classe Simulador**

**Atributos:**

- mesa: Instância do objeto que armazena as informações da mesa como tamanho e posição.
- listener: Instância do objeto que armazena as posição X e Y do taco.
- largura: Armazena a largura da mesa.
- altura: Armazena a altura da mesa.
- idTextura: Armazena o índice da textura da bola.
- idTexturaMesa: Armazena o ídice da textura da mesa.
- bolas: Instância do objeto que armazena um vetor com todas as bolas do jogo.
- erroColisao: Responsável pela verificação de duas ou mais colisões no mesmo frame.
- transX: Variável responsável pela translação da bola no eixo X.
- transY: Variável responsável pela translação da bola no eixo Y.
- primeiraTacada: Verificação da primeira tacada do jogo
- player: Armazena o player que irá realizar a jogada.
- taco: Instância do objeto que armazena as informações do taco como posição X e Y.
- arbitro: Instância do objeto que armazena o comando das regras do jogo.

**Métodos:**

- Simulador(): Construtor da classe.
- LoadBMP(): Carrega as imagens.
- iniciarTriangulo(): Inicia o jogo com todas as bolas em suas posições iniciais.
- iniciarSimulacao(): Responsável pela verificação anterior a simulação física.
- simulacao(): Inicia a simulação física do jogo, taco-bola, bola-bola, mesa-bola.

- possuiColisaoBola(): Verifica se possui colisão entre as bolas do jogo.
- andaBola(): Anda as bolas no plano.
- desenha(): Desenha as bolas no plano.
- setPosicoesXYDisplaySimulador(): Tranfere as posições do taco capturadas pela classe display para a classe simulador.
- possuiColisaoTaco(): Verifica se possui colisão entre o taco e a bola branca.

**Classe Display**

**Atributos:**

- angle: Armazena os parâmetros de especificação da câmera.

**Métodos:**

- Display(): Construtor da classe.
- Desenha(): Desenha as imagens no plano.
- InicializaParametrosVisualizacao(): Especifica os parametros de inicialização do sistema.
- GerenciaMouse(): Gerencia o movimento do mouse, utilizando a callback do mouse.
- Teclado(): Gerencia a captura de teclas do teclado, utilizando a callback do teclado.
- Timer(): Responsável por executar um frame da simulação a cada intervalo de tempo definido.

**Classe Listener**

**Atributos:**

- taco: Armazena as informações do taco como posição X e Y.

**Métodos:**

- receberPacote(): Recebe o pacote via rede do módulo da visão computacional, com as posições X e Y do taco.

**Classe Sender**

**Atributos:**

- taco: Armazena as informações do taco como posição X e Y.

**Métodos:**

- enviaPacote(): Envia o pacote via rede para o módulo da simulação física, com as posições X e Y do taco.

## 5.4 Diagramas de Sequências

### 5.4.1 Modo OFF-LINE

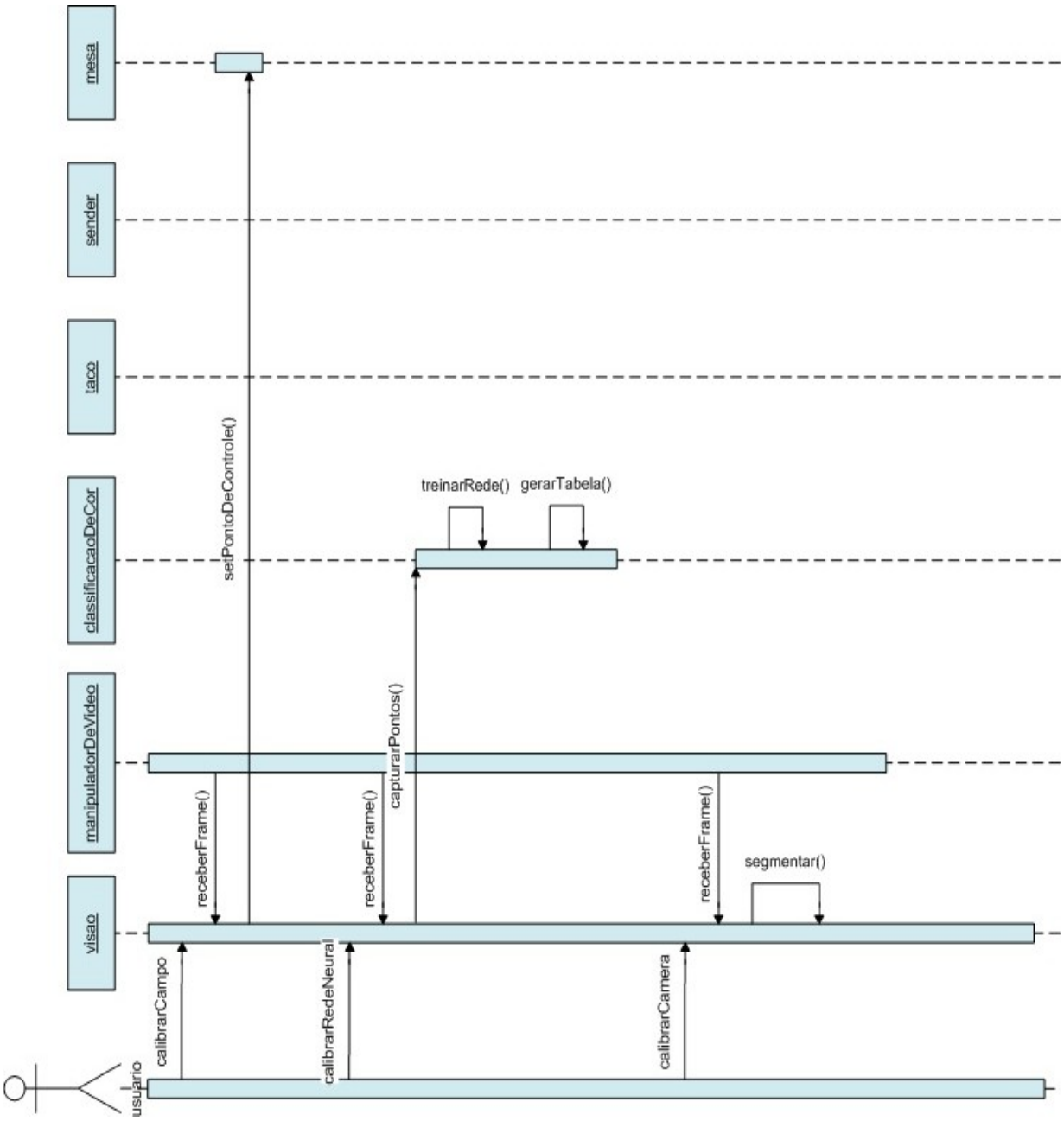


Figura 5.3: Diagrama de Sequência do Modo OFF-LINE

Fonte: Os Autores

### 5.4.2 Modo ON-LINE

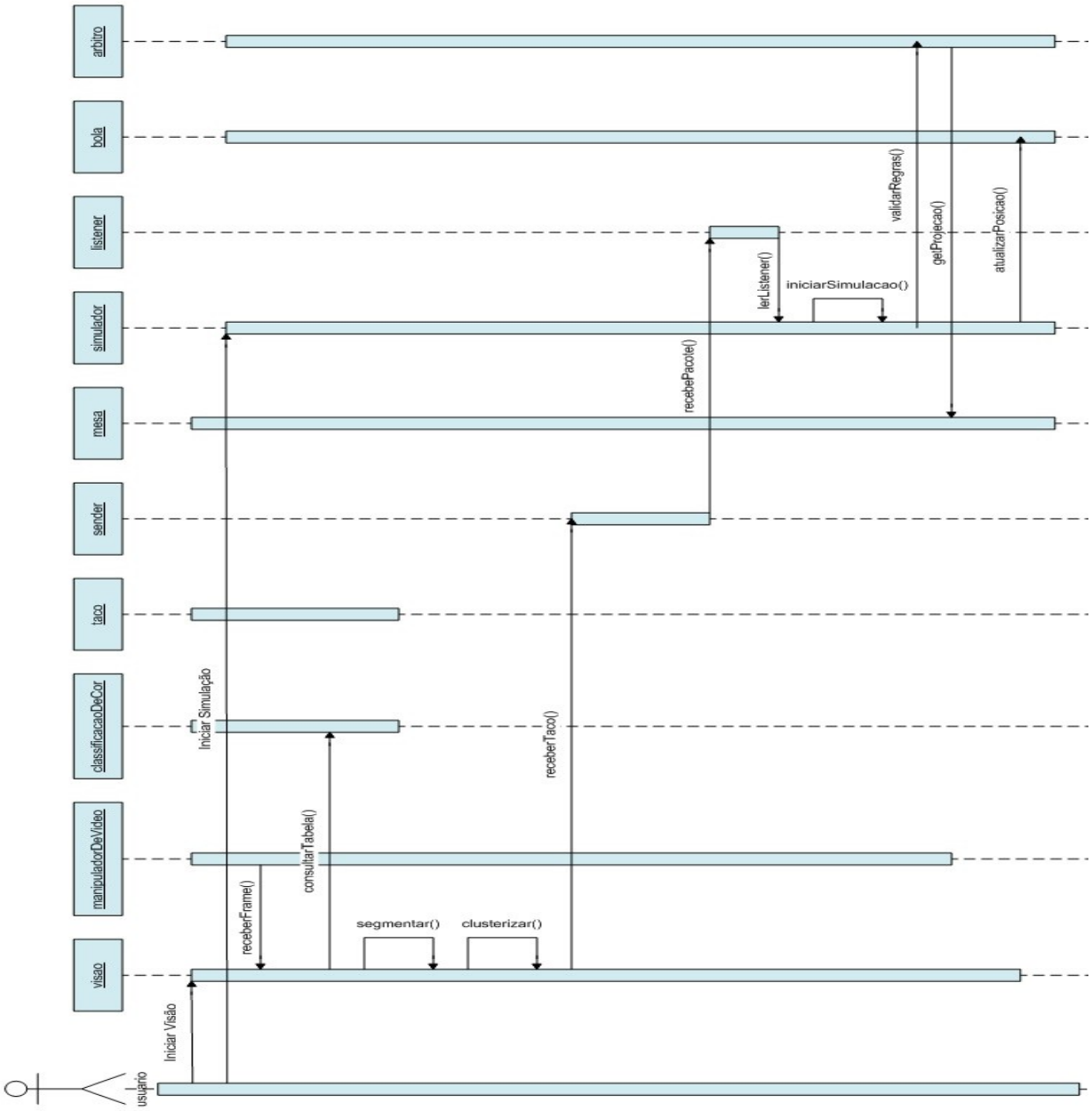


Figura 5.4: Diagrama de Sequência do Modo ON-LINE

Fonte: Os Autores

# Capítulo 6

# Planejamento

O planejamento do projeto deve ser utilizado para que os processos sejam realizados em datas, com início e fim, pre-determinadas pelo cronograma. Desenvolvendo assim um projeto de forma uniforme e organizada com o mínimo de gerenciamento.

Sendo assim, a Figura 6.1 refere-se ao Cronograma desse projeto.

| CRONOGRAMA | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Número da Tarefa | Tarefa | Duração (Dias) | Início | Término | PERÍODO | | | | |
| | | | | | *Agosto* | *Setembro* | *Outubro* | *Novembro* | *Dezembro* |
| **1** | Criação da Simulação Física | 25 | 01/08/2010 | 26/08/2010 | | | | | |
| **2** | Testes da Simulação Física | 5 | 26/08/2010 | 31/08/2010 | | | | | |
| **3** | Verificação das Regras do Jogo | 10 | 31/08/2010 | 10/09/2010 | | | | | |
| **4** | Criação do Módulo OFF-LINE | 15 | 10/09/2010 | 25/09/2010 | | | | | |
| **5** | Projeção da Simulação | 10 | 25/09/2010 | 05/10/2010 | | | | | |
| **6** | Captura do Frame | 5 | 05/10/2010 | 10/10/2010 | | | | | |
| **7** | Verificação do Taco | 5 | 10/10/2010 | 15/10/2010 | | | | | |
| **8** | Implantação da Comunicação dos Servidores | 1 | 15/10/2010 | 16/10/2010 | | | | | |
| **9** | Adaptação para receber Posição do Taco | 5 | 16/10/2010 | 21/10/2010 | | | | | |
| **10** | Testes da Projeção | 5 | 21/10/2010 | 26/10/2010 | | | | | |
| **11** | Testes do Jogo completo | 5 | 26/10/2010 | 31/10/2010 | | | | | |
| **12** | Entrega do Documento Final para a Banca | 8 | 31/10/2010 | 08/11/2010 | | | | | |
| **13** | Apresentação e Avaliação Final da Banca | 4 | 08/11/2010 | 12/11/2010 | | | | | |

Figura 6.1: Cronograma do Projeto Virtual Snooker

Fonte: Os Autores

# Capítulo 7

# Resultados Experimentais

Nessa seção serão apresentados os resultados dos testes realizados durante a implementação do sistema Virtual Snooker, bem como análises comparatívas e descrições dos métodos e equipamentos utilizados.

A Visão Computacional (primeiro módulo desse projeto) foi implementada em C++, utilizando como ferramenta de desenvolvimento o Visual Studio 2008. Os testes foram realizados em um PC Intel Core 2 Duo de 2.0GHz com 4GB de memória RAM, Sistema Operacional Windows XP Professional com Service Pack 3 e uma Câmera Firewire.

A Simulação física (segundo módulo desse projeto) foi implementada em C++, utilizando como ferramenta de desenvolvimento o DEV C++. Os testes foram realizados em um notebook AMD Turion II Dual-Core 3.2GHz com 4GB de memória RAM, Sistema Operacional Windows 7 Home Premium 64 Bits.

## 7.1 Disposição dos Equipamentos

Para a realização dos testes, os equipamentos foram colocados na horizontal. Assim, podemos identificar a distância necessária para que a câmera capture as posições do taco de forma corretamente.

O projetor foi colocado a 3 metros de distância da parede e a uma altura de 80 cm do

chão. A câmera foi colocada ao lado do projetor e ajustada para identificar corretamente o tamanho da projeção.

Apesar da disposição estar na horizontal, foram feitos testes para reconhecimento do taco e envio das informações para o módulo da simulação física projetar a movimentação do taco.

A Figura 4.1 representa a forma correta para o funcionamento do jogo.

## 7.2 Testes Realizados

### 7.2.1 Visão Computacional

O módulo de visão computacional deve ser executado em tempo real, portanto o tempo de processamento de seu algoritmo deve ser observado com muito cuidado. Um algoritmo lento pode fazer com que os frames enviados para o módulo de simulação física tenham um grande atraso em relação ao mundo real. Isso causaria uma grande perda da sensação de realidade e interação.

Realizamos um teste para determinar se o uso da tabela de cores é realmente necessária, ou se a própria rede neural é rápida o bastante para ser utilizada em tempo real. Para esse teste, utilizamos uma amostra de 5000 pixels em cada método, calculando o tempo necessário para, com os valores de R, B e G de um pixel, obter a reposta que define a cor desse dado pixel.

Os resultados, apresentados na Figura 7.1, mostram um ganho de aproximadamente 95% quando utilizamos a tabela de cores.

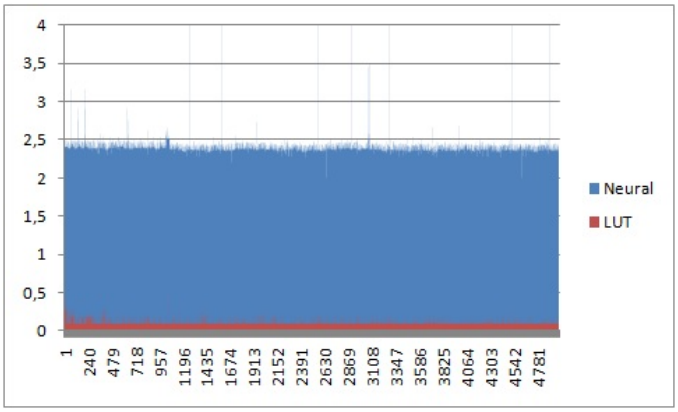


Figura 7.1: Gráfico da Utilização da Tabela de Cores X Rede Neural

Fonte: Os autores

| | Rede Neural | Tabela de Cores |
|---|---|---|
| Média | 2,42112 | 0,111081 |
| Variância | 0,099936 | 0,001114 |
| Desvio Padrão | 0,316127 | 0,033384 |

Figura 7.2: Utilização da Tabela de Cores X Rede Neural, em microsegundos

Fonte: Os autores

Com os pixels já classificados, testamos o tempo de resposta do algoritmo proposto para a obtenção dos blobs, o RLE.

Nesse teste, utilizamos novamente uma amostra de 5000 frames. Foi medido o tempo que o algoritmo leva para, dado uma imagem com os pixels já classificados com os valores da tabela de cores, obter uma lista de blobs para cada cor existente na imagem. Os resultados, apresentados na Figura 7.3, mostram que o algoritmo é eficiente e que pode ser utilizado em tempo real.

| | RLE |
|---|---|
| Média | 1513,772 |
| Variância | 24983,3 |
| Desvio Padrão | 158,0611 |

Figura 7.3: Tempo X Lista de Blobs para cada Cor Existente, em microsegundos

Fonte: Os autores

Outro teste realizado no módulo de visão computacional foi o de tempo total de processamento. Esse tempo inclui todos os passos que o módulo deve realizar para capturar o taco. Portanto, estão inclusos a captura da imagem, a segmentação com a tabela de cores, o algoritmo RLE e as heurísticas de identificação do taco. Os resultados, apresentados na Figura 7.4, estão dentro do aceitável e, mesmo com um certo atraso em relação ao processamento máximo da câmera, o algoritmo é capaz de trabalhar em tempo real.

| | RLE |
|---|---|
| Média | 16130,06 |
| Variância | 7118643 |
| Desvio Padrão | 2668,079 |

Figura 7.4: Tempo Total do Sistema da Visão, em microsegundos

Fonte: Os autores

Por fim, foi realizado um teste de identificação do taco. Neste teste, utilizamos um taco de madeira com 1 metro de comprimeto e 6 centímetros de largura. Duas marcações coloridas, uma amarela e uma rosa, com 6 centímetros de comprimento e 8 centímetros de largura foram coladas na ponta e a 15 centímetros da ponta, respectivamente. A rede neural foi treinada e a tabela de cores foi preenchida. Depois, foram contados, dentro de uma amostra de 5000 frames, o número de frames em que o taco era encontrado. No intervalo de 5000 frames do teste, o taco foi movimentado por toda a área de captura da câmera.

O teste resultou em uma média de 91% de identificação correta. O principal motivo de falha foi a oclusão do taco, que ocorre quando o jogador se posiciona entre a câmera e o taco, impedindo sua localização.

### 7.2.2 Simulação Física

#### 7.2.2.1 Bola - Bola

O primeiro teste desse tipo de colisão, foi feito com apenas duas bolas no mesmo eixo X, forçando uma colisão conhecida para verificar se o resultado era o esperado.

Assim, três bolas foram inseridas no sistema e inicializadas, resultando na colisão correta, cada uma realizou o percurso na direção que deveria. Após esse teste, foram inseridas todas as 15 bolas do jogo, para verificar o funcionamento correto de todas as colisões do jogo entre bolas.

A Figura 7.5 identifica os graus em radianos entre as bolas para poder fazer a correção do vetor velocidade após colisão.

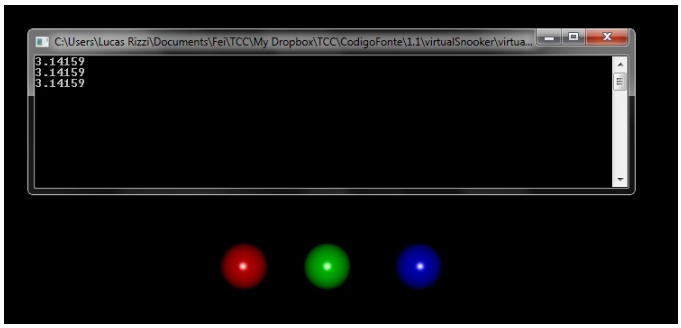


Figura 7.5: Graus em Radianos entre as Bolas

Fonte: Os autores

A Figura 7.6 identifica as posições das bolas e a distância entre elas.

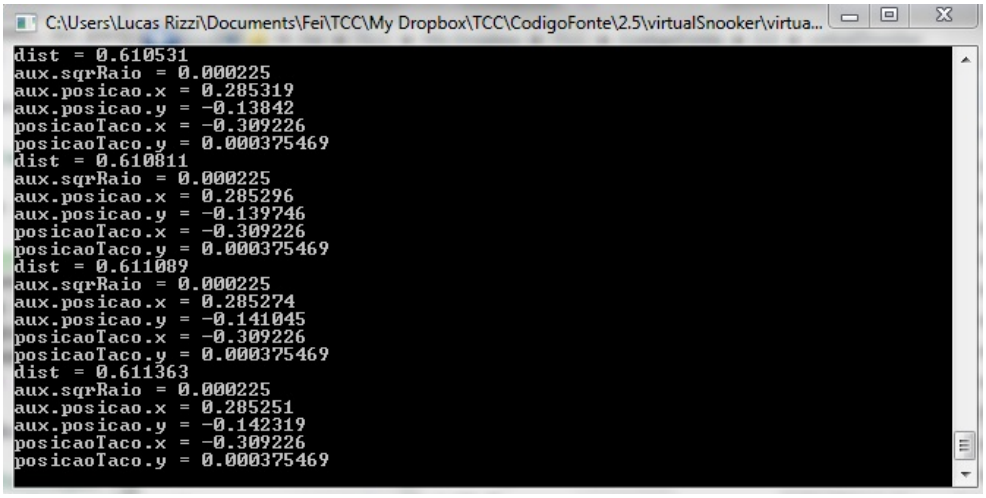


Figura 7.6: Testes da Colisão Bola - Bola

Fonte: Os autores

A Figura 7.7 exemplifica com algumas bolas em jogo.

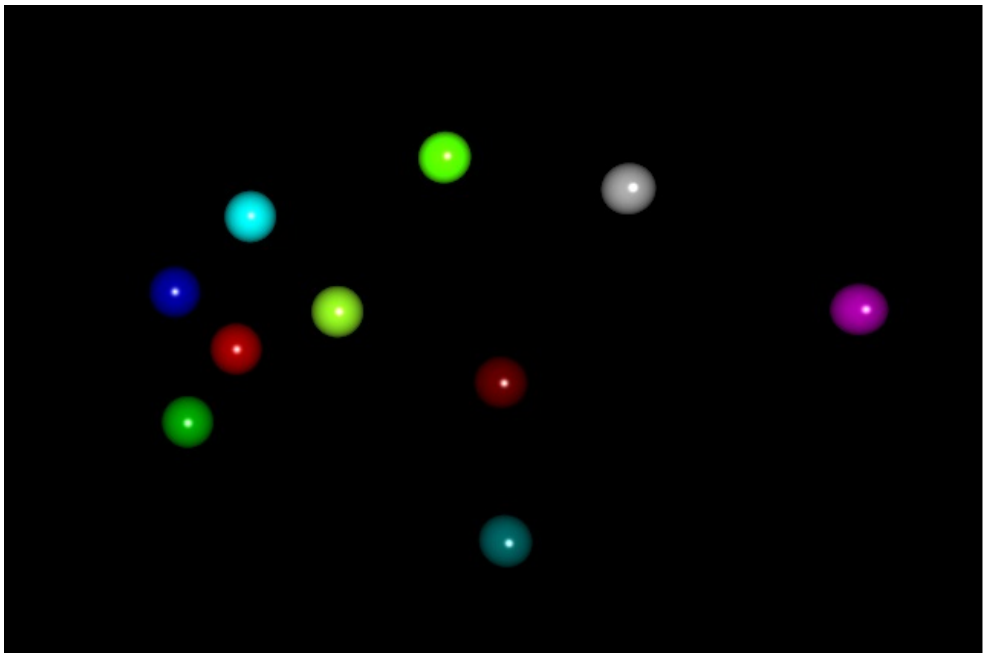

Figura 7.7: Colisão entre várias bolas

Fonte: Os autores

#### 7.2.2.2 Bola - Taco

Inicialmente, para as posições do taco foram utilizadas as posições do mouse, sendo assim, o mouse seria utilizado para verificar o ponto de colisão com a bola branca.

Posteriormente, foram realizadas alteração para a real utilização do sistema, sendo que, as posições X e Y do taco eram recebidas do módulo de visão computacional e utiliadas no módulo da simulação física. Com isso conseguimos detectar falhas nos frames devido ao delay do timer do OpenGL. Algumas alterações foram feitas e chegados aos resultados esperados.

#### 7.2.2.3 Bola - Mesa

A colisão entre a bola e a mesa foi testada com uma bola para identificar a posição máxima da ViewPort para a bola não atravessar a tabela. Esse teste foi realizado com precisão,

pois dependendo da velocidade da bola, ela pode se encontrar fora da mesa no próximo frame. Todas essas verificações foram efetuadas e encontradas soluções para suportar esse tipo de erro no frame.

De acordo com a Figura 4.8, a colisão Bola - Mesa deve se comportar da seguinte forma:

- Ao identificar colisão, deve ser feito a alteração do vetor velocidade de forma que não volte no próximo frame uma colisão com a mesa, pois assim, ela entraria em um laço e não teria fim.

- Após a alteração do vetor velocidade, é identificado se não irá sobrepor nenhuma bola, caso sim, e feito o tratamento de colisão entre as mesmas, conforme explicado na seção 4.2.6.2

Sendo assim, concluimos os testes da colisão Bola - Mesa.

### 7.2.3 Textura do Jogo

Inicialmente, criamos o jogo com uma interface simples para ser testada, essa interface pode ser vista na Figura 7.7.

Sendo assim, foi adicionado textura nas bolas e na mesa, inserindo uma realidade maior ao jogo.

A Figura 7.8 apresenta a textura utilizada.

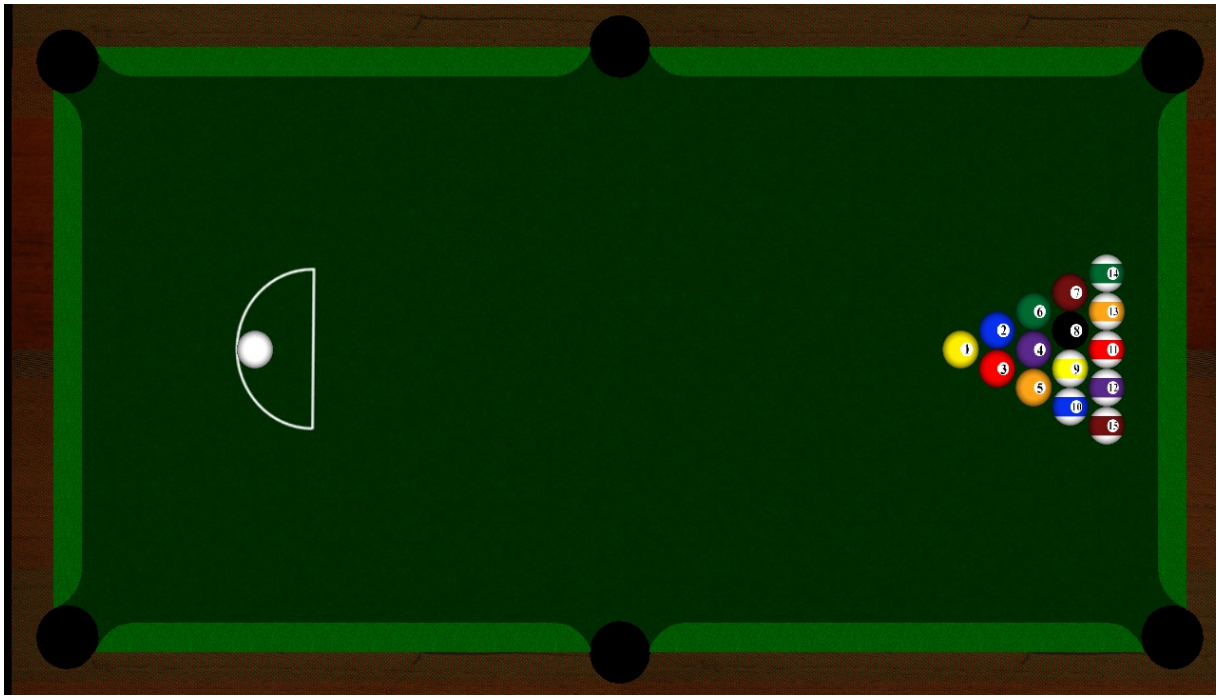

Figura 7.8: Textura do Jogo

Fonte: Os autores

# Capítulo 8

# Trabalhos Futuros

Para trabalhos futuros sugere-se o cálculo do efeito da bola em relação ao taco. Esse efeito ocorre no jogo real devido a posição que o jogador bate na bola, caso tenha sido na quina, a bola sairia rotacionando de uma maneira diferente, podendo ocasionar uma diferença na sua velocidade.

Seria interessante também inserir a rotação das bolas, pois criaria uma sensação de interação com o mundo real maior para o usuário.

Para que os usuários possam personalizar o jogo, poderia ser feito a criação de um menu indicando os tipos de jogos de sinuca disponíveis, pois nesse projeto foi inserido apenas um de acordo com o conceito explicado na seção 3.1.

Caso queiram utilizar esse projeto para treinamento de profissionais na sinuca, poderia ser implementado uma linha vermelha saindo do taco, para que o usuário possa identificar a rota da bola que será percorrida caso ele batesse naquela direção.

Para finalizar, poderia ser criado o modo computador x usuário, criando níveis de dificuldades para o usuário poder treinar suas habilidades no jogo da sinuca.

# Capítulo 9

# Conclusão

Este projeto apresentou um sistema de simulação de jogo de sinuca baseado em um sistema de visão computacional para reconhecimento do taco (real) e interação com um sistema de partículas para bolas virtuais. A arquitetura é composta de um módulo separado para visão, que reconhece as posições do taco, e um módulo independente para o sistema de partículas (simulado). O módulo de visão é sincronizado com o módulo de partículas.

Os resultados experimentais mostram que há um ganho de aproximadamente 95% quando usada a tabela de cores proposta, o que sugere que o sistema proposto é adequado para interação em tempo real.

Considerando os pixels já classificados pela rede neural, o algoritmo RLE proposto para o sistema mostrou-se adequado, de acordo com os experimentos mostrados na Tabela 7.3.

O tempo total de processamento do módulo de visão, que inclui diversos algoritmos, mostra-se dentro do aceitável para interação com em tempo real. Foram propostas heurísticas para identificação do taco, incluídos nesse módulo.

No teste de identificação do taco, os resultados mostram que, mesmo movimentando o taco por toda a área de visualização, o sistema mostrou-se robusto em relação ao tempo de resposta. Neste caso, a performance chegou a 91% de acerto, o que mostra que o sistema de visão é adequado para ser utilizado em um jogo interativo.

A simulação física foi testada de acordo com as especificações e não apresentou delay no

sistema como um todo. Sendo assim, pode-se dizer que o sistema está dentro dos requisitos esperados para a utilização em tempo real.

Não foram feitos testes de usabilidade da interface para verificar a aceitação do jogo em relação a sistemas reais, uma vez que não faz parte do escopo desse projeto.

# Bibliografia


[AZEVEDO and CONCI, 2003] AZEVEDO, E. and CONCI, A. (2003). *Computação Gráfica*, volume 1. Rio de Janeiro, Rio de Janeiro, 1 edition.

[BRUCE et al., 2000] BRUCE, J., BALCH, T., and VELOSO, M. M. (2000). Fast and inexpensive color image segmentation for interactive robots. *Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '00)*, 3:2061 – 2066.

[CABRAL et al., 2008] CABRAL, E. N., RATTO, L. S. Z., GOMES, R. C., and ROSA, R. O. (2008). Virtual hockey: Jogo interativo baseado em técnicas de visão computacional. *TCC - FEI*, 0:1–74.

[COSTA, 2007] COSTA, E. V. (2007). Bola, taco, sinuca e física. *Revista Brasileira de Ensino de Física*, 29:225–229.

[GURZONI et al., 2009] GURZONI, J. A., MARTINS, M. F., TONIDANDEL, F., and BIANCHI, R. A. C. (2009). A neural approach to real time colour recognition in the robot soccer domain. *Departamento de Elétrica e Engenharia Eletrônica - FEI*, 1:1–6.

[HALLIDAY et al., 1992] HALLIDAY, D., RESNICK, R., and WALKER, J. (1992). Física 1. *LTC Livros Técnicos e Científicos S.A.*, 1.

[KELLER et al., 1997] KELLER, F. J., GETTYSA, W. E., and SKOVE, J. M. (1997). Física. *Física*, 1.

[MCCULLOCH and PITTS, 1943] MCCULLOCH, W. S. and PITTS, W. H. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133.

[SHIH, 2010] SHIH, C. (2010). Aiming strategy error analysis and verification of a billiard training system. *Knowledge - Based Systems*, 181:1–11.

[SINUCA, 2009] SINUCA, C. B. B. (2009). Regras gerais do pool - bola 9 - bola 8 - 14x1. Website. `http://www.sinuca.com.br/arquivos/RegrasdoPool2009.doc`.

[SMITH, 2007] SMITH, M. (2007). Pickpocket: A computer billiards shark. *Science Direct*, 171:1–23.