Instituto Superior de Engenharia do Porto

# SketchyDynamics

## Apoio à Produção de Sistemas Baseados em Interfaces Caligráficas para a Simulação da Dinâmica de Corpos Rígidos

**Abílio Manuel Fernandes da Costa**

**Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Sistemas Gráficos e Multimédia**

**Orientador: Prof. Doutor João Paulo Jorge Pereira**

**Júri**:

Presidente:

Doutora Maria de Fátima Coutinho Rodrigues, ISEP

Vogais:

Doutor António Fernando Vasconcelos Cunha Castro Coelho, FEUP

Doutor João Paulo Jorge Pereira, ISEP

Porto, Outubro 2012

*«À Joaninha e aos meus pais»*

# Resumo

O paradigma de interação proporcionado pelas interfaces caligráficas constitui uma forma natural de interação humano-computador. Esta naturalidade deve-se, sobretudo, à semelhança que este estilo de interação possui com a utilização de um lápis sobre papel, tarefa comum e intuitiva. Apesar disso é ainda pouco frequente o emprego de tais interfaces em aplicações informáticas, sendo o estilo de interação WIMP (*Windows, Icons, Menus and Pointers*) mais utilizado e favorecido. No entanto, antecipa-se um futuro no qual as interfaces caligráficas estarão cada vez mais presentes, pois é notório o surgimento de um número crescente não só de aplicações que adotam este estilo de interação, mas também de equipamentos que incentivam à sua utilização. Com base nesta premissa, é seguro afirmar a necessidade de investir nesta área, de modo a agilizar e acelerar a adoção do estilo de interação caligráfico e, assim, tornar a interação humano-computador num processo cada vez mais natural.

O trabalho descrito neste documento visa um estudo à utilização das interfaces caligráficas orientada para a criação e controlo de um ambiente simulado. Mais concretamente, é apresentado o sistema SketchyDynamics, que integra um módulo de simulação da dinâmica de corpos rígidos em simbiose com uma interface caligráfica munida das ações necessárias para a manipulação da simulação. Recorrendo a este sistema, é facilitada a produção de aplicações que tirem partido destas funcionalidades, sem a necessidade de as reimplementar.

É ainda descrita uma avaliação de técnicas de reconhecimento caligráfico realizada com o objetivo de determinar aquela que melhor se integraria no sistema desenvolvido. No âmbito desta avaliação são ainda apresentados alguns pormenores sobre a implementação dessas técnicas, bem como procedimentos que permitem uma maximização da sua eficácia.

São também discutidos os resultados de uma avaliação de usabilidade conduzida com o propósito de validar o sistema SketchyDynamics do ponto de vista do utilizador. Os resultados desta avaliação mostram que este sistema foi bem-sucedido e que se encontra preparado para o utilizador final, não obstante a existência de margem para futuras melhorias.

**Palavras-chave**: Reconhecimento Caligráfico, Interação Natural, Simulação da Física, Dinâmica de Corpos Rígidos

# Abstract

The interaction paradigm provided by sketch-based interfaces represents a natural method of human-computer interaction. This naturalness is largely due to the similarity that this interaction style has with the use of a pencil on a paper, an intuitive and common task. Despite that, the implementation of these interfaces on computer applications is still unusual, in favor of the WIMP (Windows, Icons, Menus and Points) interaction style. Nevertheless, we can predict a future where sketch-based interfaces will be increasingly more widespread, based on the recent emergence of not only applications that adopt this interaction style, but also equipment that encourage their use. With this premise in mind, it is safe to assert the need for investment in this area, in order to streamline and accelerate the adoption of the sketch-based interaction style and thus make the human-computer interaction a progressively more natural process.

The work described in this document aims the study of the use of sketch-based interfaces in the creation and control of simulated environments. More specifically, we present the SketchyDynamics system, which incorporates a rigid body simulation module in symbiosis with a sketch-based interface provided with the necessary actions for the manipulation of the simulation. Using this system, we hope to ease the production of applications that take advantage of these features, without the need to implement them from scratch.

An evaluation of various sketch recognition techniques, performed in order to find the one that best fits in the developed system, is also described. As part of this evaluation, we also present some details on the implementation of these techniques, as well as procedures that allow us to maximize their efficiency.

Furthermore, we discuss the results of a usability evaluation that was conducted with the purpose of validating the SketchyDynamics system from the user's point of view. The results of this evaluation suggest that, despite the existence of room for further improvements, the system was successful and is ready for final users.

**Keywords**: Sketch-Based Recognition, Natural Interaction, Physics Simulation, Rigid Body Dynamics

# Agradecimentos

Gostaria de começar com um agradecimento muito especial aos meus pais por todo o seu esforço que possibilitou a minha chegada a esta fase da minha vida, pelo apoio com que me brindaram e pela confiança sempre presente, em todos os momentos.

A toda a minha família, em especial aos meus avós, por todo o interesse e preocupação demonstrados.

Ao meu orientador, Professor Doutor João Paulo Pereira, pelo apoio e disponibilidade constantes, sem os quais este trabalho não teria sido possível. Agradeço ainda o interesse, motivação e encorajamento durante todo o processo de realização deste trabalho.

À Professora Doutora Ana Madureira, deixo um agradecimento especial pela sua participação na sessão da avaliação de usabilidade e também pelo consentimento dado à participação dos bolseiros do projeto ADSyS nessa mesma sessão, aos quais também agradeço. Também aos alunos da unidade curricular de Complementos de Sistemas Gráficos do Mestrado em Engenharia Informática do ISEP, agradeço a sua colaboração na sessão de recolha de amostras caligráficas.

Por fim, o meu imenso obrigado à minha namorada, Joana, por todos os seus conselhos, infinita paciência, apoio incansável e pela sua presença constante nos altos e baixos deste percurso.

# Índice

# Lista de Figuras

# Lista de Tabelas

# Acrónimos e Símbolos

## Lista de Acrónimos

| | |
|---|---|
| 1D | Uma dimensão |
| 2D | Duas dimensões |
| 3D | Três dimensões |
| API | *Application Programming Interface* (Interface de Programação de Aplicações) |
| CAD | *Computer-Aided Design* (Desenho Assistido por Computador) |
| DTW | *Dynamic Time Warping* |
| GECAD | Grupo de Investigação em Engenharia do Conhecimento e Apoio à Decisão |
| HMM | *Hidden Markov Models* |
| ISEP | Instituto Superior de Engenharia do Porto |
| WIMP | *Windows, Icons, Menus and Pointers* (Janelas, Ícones, Menus e Apontadores) |

# 1 Introdução

No nosso dia-a-dia é bastante comum recorrer ao papel e lápis para fazer um desenho ou esboço de modo a transmitir uma ideia ou um conceito. A verdade é que esta ação permite uma expressividade muito direta e natural, pois é realizada pelo ser humano desde cedo na sua vida. Apesar disso, são ainda poucas as interfaces humano-computador que tiram partido desse estilo de interação.

Atualmente, a aplicação mais comum do estilo de interação caligráfica é provavelmente a introdução de texto. Com recurso a um estilete ou ecrã tátil, estes sistemas possibilitam ao utilizador introduzir texto escrito à mão, como faria normalmente numa folha de papel, sendo este texto posteriormente reconhecido e interpretado pelo sistema [1]. Existem ainda sistemas CAD (*Computer-Aided Design*) no qual o utilizador desenha diretamente os objetos que pretende criar, em vez de recorrer a menus para o efeito [2]. Uma outra abordagem comum à interação caligráfica, mas que não reflete tão diretamente o paradigma papel-lápis, é a utilização de gestos de ação. Um exemplo popular do uso de gestos de ação encontra-se no navegador de internet Opera, que permite ao utilizador realizar um conjunto de ações como navegar no histórico, abrir e fechar separadores, entre outras, através de simples movimentos do rato na forma de gestos [3]. Com o crescente número de sistemas com ecrãs táteis no mercado, não só em *smartphones* e *tablets*, mas também em computadores portáteis pessoais, é esperado um aumento de aplicações que recorrem a interfaces caligráficas, pela usabilidade que oferecem nesse tipo de sistemas. Isto permite adivinhar um futuro no qual estas interfaces serão cada vez mais populares e a interação humano-computador cada vez mais natural.

A conjugação deste tipo de interface com a simulação do comportamento físico de corpos rígidos poderia ocasionar um conjunto de aplicações úteis no ensino da área da física, possibilitando, por exemplo, uma demonstração instantânea dos conceitos esboçados sobre um quadro interativo. Ou seja, o utilizador seria capaz de desenhar os corpos como faria num quadro tradicional, mas estes seriam simulados automaticamente de acordo com as leis da física e os parâmetros definidos. Isto poderia também fornecer uma forma simples de realizar experiências de física ou até de conceber jogos ou outros sistemas de entretenimento.

O trabalho aqui apresentado incide neste conceito da utilização de interfaces caligráficas para a construção de cenas simuladas. No entanto, não se pretende apenas uma aplicação que implemente um conjunto de funcionalidades em torno desta ideia, mas sim um sistema que auxilie na produção de tais aplicações. Apesar da atual existência de diversas aplicações que combinam uma interface caligráfica com um ambiente fisicamente simulado, a maioria destas possui um âmbito e público específicos. Em oposição, o sistema aqui proposto não se restringe a um determinado âmbito e pretende ser utilizado para desenvolver o mais variado tipo de aplicações, com diferentes objetivos e públicos. Ao disponibilizar um modo facilitado para a criação deste tipo de aplicação, espera-se contribuir para o surgimento de novas e estimulantes aplicações de simulação da dinâmica de corpos rígidos que tirem partido de todas as capacidades providenciadas por uma interface caligráfica.

## 1.1 Objetivos

Este trabalho pretende investigar e analisar as aplicabilidades da analogia do lápis e papel em interfaces, com vista a desenvolver um sistema de simulação da dinâmica de corpos rígidos baseado nesta analogia. Este sistema será, portanto, composto por duas componentes principais: um simulador realista de física, que suportará toda a dinâmica dos corpos rígidos, e uma interface caligráfica a partir da qual o utilizador poderá desenhar os objetos a simular, bem como concretizar a interação necessária com os mesmos. Além disso, deve ser facilitada a sua integração em outros sistemas, viabilizando assim o desenvolvimento rápido de aplicações que usufruam destes componentes.

Antes de prosseguir com o desenvolvimento de tal sistema, será necessário conduzir uma pesquisa bibliográfica referente ao estado da arte de interfaces caligráficas, nomeadamente sobre as diferentes técnicas de reconhecimento caligráfico existentes atualmente. Além disso, para prover uma maior confiança na escolha da técnica de reconhecimento a utilizar, deverá ser realizada uma avaliação de diversas técnicas, com recurso a dados caligráficos obtidos de utilizadores.

No sentido de demonstrar a aplicabilidade do sistema desenvolvido, deverão ser concebidas aplicações baseadas nesse sistema, com fins educacionais mas também de entretenimento.

Finalmente, é importante a realização de uma avaliação do sistema implementado, com o objetivo de garantir a sua funcionalidade. Esta consistirá em avaliar a eficiência na utilização do sistema e a sua usabilidade, com recurso a testes de eficiência e questionários a utilizadores.

Sintetizando, os objetivos fundamentais para este trabalho consistem em:

- Realizar um levantamento do estado da arte no que se refere a interfaces caligráficas e a sistemas que a elas recorrem para simulação da física;
- Avaliar diferentes reconhecedores caligráficos, de acordo com as características definidas para o sistema a desenvolver;

- Desenvolver um sistema que facilite a construção de aplicações de simulação da dinâmica de corpos rígidos com interação caligráfica;
- Desenvolver aplicações de demonstração de aplicabilidade do sistema concebido;
- Avaliar a eficiência e usabilidade do sistema concebido.

## 1.2 Especificação de Requisitos

Durante a fase inicial do trabalho aqui apresentado, foram definidos alguns requisitos essenciais para o sistema a desenvolver. Um dos primeiros requisitos passou pela definição de quais os gestos suportados e quais os objetos originados por cada um desses gestos. É importante salientar que, neste contexto, um gesto trata-se de um desenho esboçado ou de uma forma desenhada pelo utilizador, desejavelmente (mas não obrigatoriamente) com um estilete sobre a superfície de um ecrã táctil. Foi então definido que o sistema deve suportar três gestos para a criação de corpos rígidos: retângulo, triângulo e círculo; podendo estes assumir diversas dimensões e formas de desenho. Deve também ser possível a criação de ligações entre corpos, de modo a possibilitar a restrição dos seus movimentos relativos. Para tal, o utilizador deve ser capaz de desenhar um gesto ziguezague para ligar dois corpos com uma mola, uma cruz para fixar um corpo sobre outro, impedindo qualquer movimento relativo, e um pequeno círculo para fixar um corpo sobre outro através de um eixo de rotação. Dado ser utilizado o mesmo gesto para criar um corpo circular e um eixo de rotação, apresentando variações meramente ao nível das suas dimensões e contexto, apenas existem de facto cinco gestos, apresentados na Figura 1.

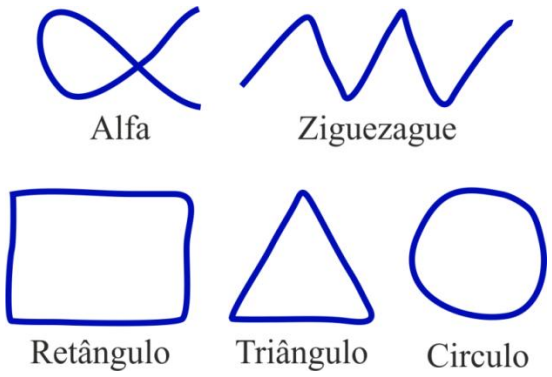



Figura 1 – Conjunto de gestos a reconhecer pela solução proposta

Considerando que a cruz é o único gesto que não pode ser desenhado com apenas um traço, optou-se pela sua substituição por um alfa, pois representa corretamente uma cruz desenhada com uma única linha. A decisão de recorrer apenas a gestos de traço único decorre da intenção de tornar a interação mais simples, para além de ser suficiente para os requisitos do trabalho aqui proposto.

Para que os objetos criados assumam a forma do gesto que lhes deu origem, foi definido que a simulação da física e a representação visual dos objetos seria bidimensional (2D). Além disso,

os objetos devem possuir uma profundidade resultante da ordem com que são desenhados, isto é, os objetos criados mais recentemente devem ser exibidos por cima dos mais antigos. Para além da criação de objetos, o sistema deve permitir ainda a manipulação dos mesmos, disponibilizando meios para a aplicação de transformações geométricas e para a sua remoção.

Um outro requisito incide sobre a existência de dois estados para a simulação. Com o intuito de permitir que o sistema criado possa ser utilizado na conceção de aplicações nos mais variados domínios, este deve dispor da capacidade para interromper e resumir a simulação, possibilitando sempre a interação do utilizador independentemente do estado atual. Quando a simulação se encontrar interrompida, ao adicionar um objeto à cena ele deve permanecer exatamente com a posição e orientação definidas. Este estado poderá ser útil para experiências de demonstração do comportamento de corpos rígidos, onde se pretende estabelecer um cenário e, apenas depois de este estar definido, simular o comportamento desses corpos preestabelecidos. Em contrapartida, um objeto adicionado enquanto a simulação se encontra em execução deve reagir imediatamente de acordo com as forças que sobre ele incidem, caindo em direção ao chão caso exista gravidade, por exemplo. Este estado poderá ser útil para jogos, onde se pretende que o jogador interaja diretamente com os objetos enquanto a ação decorre.

Por fim, é importante referir que a solução aqui proposta, quando integrada numa aplicação, não pretende definir uma interface completa para a aplicação, mas sim disponibilizar as características necessárias para a simulação da dinâmica de corpos rígidos, em conjunto com uma série de ações de interação para o controlo e manipulação dessa simulação. Assim, apesar de disponibilizar uma interface na qual é exibida a simulação e a partir da qual o utilizador interage, deve oferecer os meios necessários para que a aplicação possa completar a interface de acordo com as suas especificidades.

## 1.3 Estrutura da Dissertação

Esta dissertação encontra-se organizada em seis capítulos. Neste primeiro capítulo foi apresentada a motivação para a realização deste trabalho, bem como uma pequena abordagem às vantagens inerentes às interfaces caligráficas. Foram ainda referidos os objetivos e requisitos propostos durante a fase inicial deste projeto.

No segundo capítulo é apresentado um levantamento do estado da arte no que se refere a técnicas de reconhecimento caligráfico, sendo descritas as suas características e funcionamento base. É também realizada uma referência a algumas aplicações que permitem simulações da dinâmica de corpos rígidos com recurso a interfaces caligráficas, encontrando-se assim diretamente relacionadas com o âmbito deste projeto.

O terceiro capítulo inicia-se com uma apresentação da metodologia de desenvolvimento adotada. Segue-se uma descrição detalhada do sistema desenvolvido para implementação e avaliação de algoritmos de reconhecimento caligráfico, bem como do processo de

implementação de algoritmos nesse sistema. É também discutida a avaliação realizada aos algoritmos de reconhecimento, e cujo propósito passa por determinar qual o que melhor se adequa aos requisitos da solução a desenvolver.

No quarto capítulo é apresentada a solução proposta para este trabalho, que se expressa sobre a forma de uma biblioteca de programação e com a qual é possível desenvolver aplicações que incorporam as características acima referidas. Este capítulo faz uma apresentação desta biblioteca primeiramente ao nível das suas funcionalidades e, de seguida, no que respeita à sua arquitetura e implementação.

No quinto capítulo são discutidos os resultados da avaliação de usabilidade realizada à biblioteca desenvolvida, no que se refere ao desempenho possibilitado pela sua utilização e à perceção e opinião dos participantes envolvidos na avaliação.

Por fim, no sexto e último capítulo são apresentados os juízos e conclusões sobre o trabalho realizado, de acordo com os objetivos propostos inicialmente. São também referidas algumas limitações e sugeridas várias possibilidades de melhoria e trabalho futuro.

# 2 Estado da Arte

Este capítulo tem como objetivo abordar o estudo ao trabalho realizado na área das interfaces caligráficas, sendo este de grande importância para a contextualização e fundamentação da solução proposta neste documento.

Inicialmente serão apresentados diversos algoritmos de reconhecimento caligráfico, que visam a identificação de gestos desenhados à mão e que constituem o núcleo de qualquer interface caligráfica. De seguida serão citados alguns sistemas que, com recurso a interfaces caligráficas, permitem a simulação da dinâmica de corpos rígidos.

## 2.1 Algoritmos de Reconhecimento Caligráfico

Dado o potencial do reconhecimento automático de esboços, sejam eles formas geométricas, esquemas ou até letras do alfabeto, muito trabalho vem a ser realizado no sentido de desenvolver reconhecedores caligráficos capazes de lidar com a ambiguidade intrínseca do desenho feito à mão.

Em conjunto com a existência de uma grande variedade de algoritmos de reconhecimento caligráfico, é natural que exista também uma grande diversidade nas suas características. Por exemplo, enquanto alguns reconhecedores funcionam apenas para esboços desenhados com um único traço, outros permitem o reconhecimento de esboços complexos, compostos por múltiplos traços. Para além disso, a capacidade ou incapacidade de um reconhecedor ser capaz de classificar esboços independentemente da orientação, escala, ordem pela qual os sucessivos traços são esboçados e sentido de desenho de cada traço, pode ter um grande impacto na sua utilidade em determinado domínio ou aplicação. Por exemplo, numa aplicação de reconhecimento de texto escrito à mão, os gestos desenhados (carateres) apenas fazem sentido quando realizados com uma orientação pré-definida. Já numa aplicação de desenho livre é importante que o utilizador possa desenhar sem quaisquer restrições.

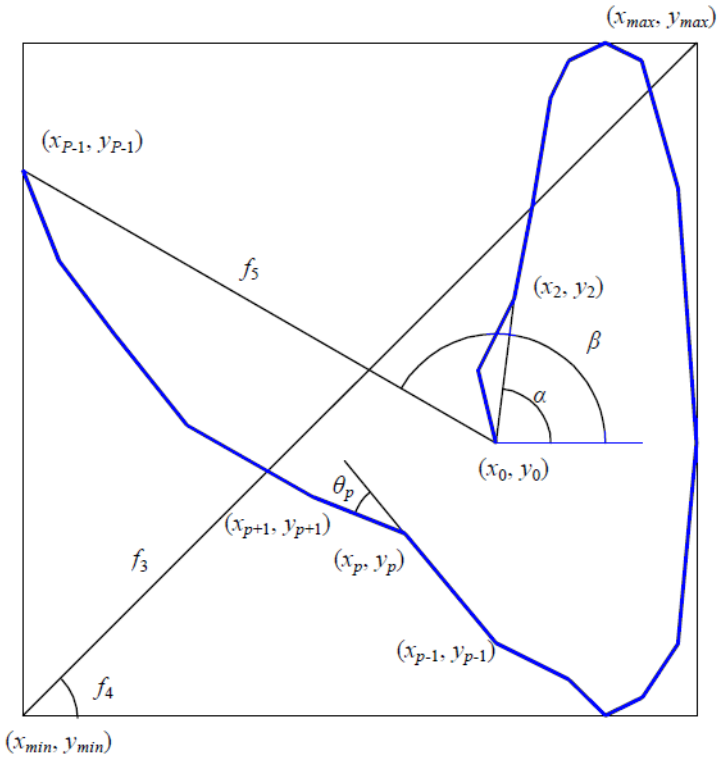
Outra característica importante de alguns reconhecedores é o facto de poderem ser facilmente treinados para identificar novos gestos, a partir de amostras de treino ou *templates* desses gestos, o que significa que podem ser facilmente expandidos e adaptados. Em contrapartida, existem reconhecedores nos quais as suas classes gestuais [1] estão codificadas no próprio reconhecedor. Isto faz com que estes reconhecedores estejam geralmente otimizados para o reconhecimento específico desses gestos, mas torna difícil a sua adaptação a um novo domínio que requeira novos gestos.

### 2.1.1 Reconhecedor de Rubine

Dean Rubine [4] desenvolveu um algoritmo de reconhecimento caligráfico com capacidades de treino, capaz de reconhecer gestos compostos por um único traço. Este algoritmo é sensível à dimensão e orientação dos gestos, bem como ao sentido com que são desenhados.

O reconhecedor de Rubine opera com base num conjunto de características distintivas de identificação de gestos, apresentadas na Figura 2. Rubine definiu onze características geométricas, tais como o seno/cosseno do angulo inicial do gesto, a distância entre o primeiro e último ponto, o comprimento total do gesto, entre outras. Definiu ainda a velocidade máxima e duração do desenho como duas características cinemáticas opcionais. Apesar das treze características definidas por Rubine, o algoritmo permite que sejam adicionadas ou removidas características muito facilmente, podendo assim ser ajustado às necessidades de cada aplicação. Plimmer *et al.* [5] descrevem como melhorar o reconhecimento e torná-lo insensível à dimensão dos gestos, através da remoção de características que envolvam dimensões absolutas, substituindo-as por outras que utilizem rácios. Por exemplo, em vez da altura e largura dos gestos é utilizado o rácio entre essas duas dimensões. Para além disso, estes autores introduziram no reconhecedor de Rubine algumas das características geométricas utilizadas pelo reconhecedor CALI (secção 2.1.2). Esta facilidade na manipulação das características de identificação de gestos, aliada ao facto de o algoritmo poder ser treinado para reconhecer os mais diversos gestos a partir de *templates*, fazem com que o reconhecedor de Rubine seja bastante flexível.

[1] Representação de um gesto que o reconhecedor tem capacidade para reconhecer (Retângulo, por exemplo). Uma classe gestual pode ser representada por um conjunto de regras que identificam um gesto, ou ser constituída por vários *templates*, no caso dos algoritmos com capacidades de treino.




$$f_1 = \cos\alpha = \frac{x_2 - x_0}{\sqrt{(x_2 - x_0)^2 + (y_2 - y_0)^2}}$$

$$f_2 = \sin\alpha = \frac{y_2 - y_0}{\sqrt{(x_2 - x_0)^2 + (y_2 - y_0)^2}}$$

$$f_3 = \sqrt{(x_{max} - x_{min})^2 + (y_{max} - y_{min})^2}$$

$$f_4 = \arctan\frac{y_{max} - y_{min}}{x_{max} - x_{min}}$$

$$f_5 = \sqrt{(x_{P-1} - x_0)^2 + (y_{P-1} - y_0)^2}$$

$$f_6 = \cos\beta = \frac{x_{P-1} - x_0}{f_5}$$

$$f_7 = \sin\beta = \frac{y_{P-1} - y_0}{f_5}$$

Sejam $\Delta x_p = x_{p+1} - x_p$, $\Delta y_p = y_{p+1} - y_p$

$$f_8 = \sum_{p=0}^{P-2}\sqrt{\Delta x_p^2 + \Delta y_p^2}$$

Seja $\theta_p = \arctan\frac{\Delta x_p \Delta y_{p-1} - \Delta x_{p-1}\Delta y_p}{\Delta x_p \Delta x_{p-1} - \Delta y_p \Delta y_{p-1}}$

$$f_9 = \sum_{p=1}^{P-2}\theta_p \qquad f_{10} = \sum_{p=1}^{P-2}\left|\theta_p\right| \qquad f_{11} = \sum_{p=1}^{P-2}\theta_p^{\,2}$$

Seja $\Delta t_p = t_{p+1} - t_p$

$$f_{12} = \max_{p=0}^{P-2}\frac{\Delta x_p^2 + \Delta y_p^2}{\Delta t_p^2}$$

$$f_{13} = t_{P-1} - t_0$$

Figura 2 – Características geométricas utilizadas pelo reconhecedor de Rubine

Durante o processo de treino, quando o algoritmo "aprende" que gestos reconhecer, o reconhecedor de Rubine computa as características de cada *template* de treino associado a cada classe gestual e, com base nessas características, determina o peso que cada classe possui em cada uma das características. Como é referido pelo autor, o reconhecedor necessita de cerca de quinze *templates* de treino por classe gestual para ser eficiente, o que pode tornar a criação desses *templates* uma tarefa morosa. Para identificar um gesto, o algoritmo

realiza uma combinação linear das características do gesto a identificar com os pesos atribuídos às características de cada classe gestual. A classe que maximizar essa combinação é portanto a classe que identifica o gesto.

De forma a resolver algumas limitações do reconhecedor de Rubine, Pereira *et al*. [6] procederam a algumas modificações para que o algoritmo possa aceitar gestos compostos por múltiplos traços. No entanto, tal como referido por Stahovich [7], estes gestos multi-traço necessitam de ser desenhados com um conjunto de traços consistente de forma a serem corretamente reconhecidos. Pereira *et al*. descrevem ainda como tornar o reconhecedor insensível à direção de desenho, através da realização do processo de reconhecimento em duas fases: primeiro com o gesto desenhado e seguidamente com uma versão invertida do mesmo.

### 2.1.2 CALI

CALI [8] é uma biblioteca de programação desenvolvida por Fonseca *et al*. que permite o reconhecimento de gestos multi-traço, independentemente da sua dimensão e orientação. Para a identificação de gestos, o CALI recorre a algoritmos de lógica difusa[2] em conjunto com características geométricas definidas para cada classe gestual.

O conjunto de gestos reconhecidos pelo CALI, apresentado na Figura 3, encontra-se dividido em duas categorias: formas geométricas e comandos. Enquanto as formas geométricas podem ser desenhadas (e consequentemente reconhecidas) com traço normal, tracejado ou espeço[3], os comandos apenas são reconhecidos se desenhados com traço normal. Independentemente de um dado gesto representar uma forma geométrica ou um comando, o utilizador pode desenhá-lo sem quaisquer restrições ao nível da sua orientação ou dimensão.

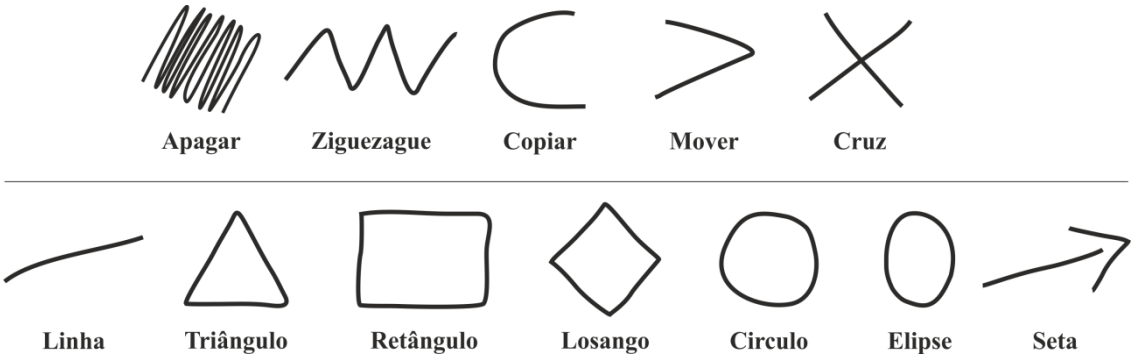



Figura 3 – Gestos de comando (em cima) e formas geométricas (em baixo) reconhecidos pelo CALI

O reconhecimento de gestos do CALI funciona com base num conjunto de características geométricas definidas para identificar cada gesto, tais como a sua espessura, o rácio entre a

[2] Tipo de lógica que admite resultados intermédios entre dois valores, em oposição à lógica tradicional na qual são possíveis apenas dois resultados (Verdadeiro e Falso).

[3] Um traço espeço é desenhado sobrepondo múltiplos traços normais.

altura e largura, número mínimo de traços, entre outros. Todas as características geométricas utilizadas pelo CALI são resultantes de relações entre as áreas e perímetros de três polígonos especiais, computados para cada gesto: o triângulo e o quadrilátero de maiores dimensões inscritos no polígono convexo do gesto; e ainda o retângulo de menor área capaz de envolver o gesto. Para além das características geométricas globais, alguns gestos podem ainda ser descritos por características locais tais como as formas que os constituem e as suas relações, como é o caso do comando "Cruz", identificado por ter duas formas do tipo "Linha" que se intersetam.

Quando um gesto desenhado pelo utilizador entra no processo de reconhecimento, as suas características geométricas são computadas e comparadas com as definidas para cada classe gestual. Esta comparação é feita com recurso a algoritmos de lógica difusa, o que torna possível obter o grau de enquadramento do gesto em cada uma das regras definidas e, consequentemente, em cada classe gestual.

Caso exista ambiguidade no reconhecimento de um gesto, devido à existência de mais do que uma classe gestual a identificar esse gesto, o CALI indica todas as classes que podem classificar o gesto e o respetivo grau de certeza, podendo estes dados ser utilizados pela aplicação para exibir uma lista de expetativas[4] a partir da qual o utilizador tem a possibilidade de indicar a classe gestual correta.

Considerando que o CALI não possui a capacidade de treino a partir de *templates*, a adição de novas classes gestuais revela-se uma tarefa complexa que envolve uma análise rigorosa de quais características definem e distinguem o novo gesto, bem como a respetiva programação dessas características no código fonte do reconhecedor. No sentido de resolver esta limitação, os autores do CALI apresentam também um reconhecedor com capacidades de treino e comparam três algoritmos que podem ser utilizados na fase de treino deste reconhecedor: *K-Nearest Neighbors*, *Inductive Decision Tree* e *Naïve Bayes*, o último apresentando a maior eficiência de treino. Contudo, este reconhecedor possui uma taxa de reconhecimento inferior à do CALI e necessita de inúmeros *templates* de treino para cada classe gestual.

### 2.1.3 Reconhecedor $1

Desenvolvido com a rapidez e facilidade de implementação em mente, o reconhecedor $1 [9] possibilita o reconhecimento de gestos independentemente da sua orientação e dimensão, sendo, no entanto, sensível à direção com que estes são desenhados. Ao contrário da grande maioria dos reconhecedores, este não utiliza modelos estatísticos nem técnicas associadas ao campo da Inteligência Artificial, como modelos de Markov, baseando-se apenas em geometria e trigonometria simples. Desta forma, o reconhecedor $1 revela-se bastante acessível e fácil de compreender. Outra grande vantagem deste reconhecedor é a simplicidade inerente ao suporte de novos gestos, exibindo geralmente bons resultados com apenas um *template* de

[4] Lista com identificações alternativas para um dado gesto, a partir da qual o utilizador pode corrigir a identificação primária realizada pelo reconhecedor.

treino por classe gestual. Porém, um número superior de *templates* de treino permite usualmente obter melhores taxas de reconhecimento.

O algoritmo é composto por quatro etapas principais, sendo as primeiras três utilizadas para processar tanto os *templates* de treino como os gestos a reconhecer; a quarta etapa aplica-se apenas aos gestos a reconhecer. A primeira etapa é responsável por efetuar uma reamostragem dos pontos que constituem o gesto, através da interpolação linear desses pontos, para que todos os gestos (incluindo os *templates* de treino) sejam compostos por um número igual de pontos. Esta reamostragem permite que seja realizada posteriormente uma comparação ponto-a-ponto entre o gesto a reconhecer e os *templates* de treino, independentemente da velocidade e dimensões com que são desenhados, ou da taxa de amostragem do dispositivo de captura de gestos.

Na segunda, etapa o gesto é rodado na tentativa de assumir uma orientação ótima, de forma a reduzir o tempo de reconhecimento numa fase posterior. Esta rotação é feita de acordo com aquilo que os autores referem como "ângulo indicativo" e que se traduz no ângulo entre o centro e o primeiro ponto do gesto, como apresentado na Figura 4.

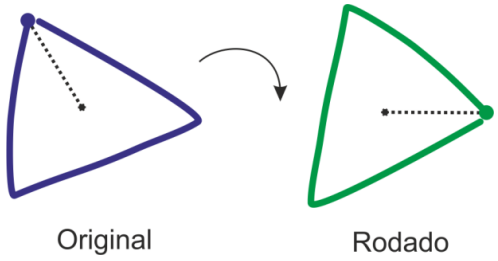



Figura 4 – Rotação do gesto realizada na segunda etapa do reconhecedor $1. Os círculos representam o ponto inicial de desenho do gesto.

De seguida, na terceira etapa, o gesto sofre uma alteração de escala não uniforme passando a assumir as dimensões de um quadrado de referência, e é deslocado de forma a que o seu centro (x,y) passe a situar-se na origem (0,0). Estas transformações, representadas na Figura 5, fazem com que todos os gestos possuam dimensões e posicionamento idênticos, sejam eles *templates* de treino ou gestos a reconhecer. Assim, o reconhecimento revela-se independente da dimensão do gesto e da posição onde é desenhado. No entanto, a alteração não uniforme da escala tem o efeito negativo de impedir o reconhecimento de gestos unidimensionais como segmentos de linhas retas.

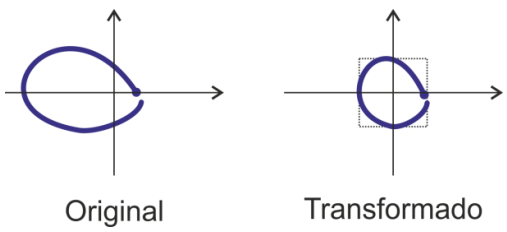



Figura 5 – Escala não uniforme e translação do gesto, realizados na terceira etapa do reconhecedor $1

A quarta e última etapa, aplicada apenas aos gestos a reconhecer, é onde o processo de reconhecimento ocorre efetivamente. Nesta etapa são determinadas as distâncias médias entre o gesto a reconhecer e cada um dos *templates* de treino, com base na distância entre cada ponto do gesto e o ponto correspondente [5] de cada *template*. Durante este procedimento, o gesto a reconhecer é progressivamente rodado, utilizando o algoritmo *Golden Section Search* [10] na tentativa de encontrar o ângulo que minimize a distância média. De seguida, com base na distância média entre o gesto a reconhecer e cada um dos *templates* de treino, é calculada uma pontuação que define o grau de certeza desse *template* na identificação do gesto. Consequentemente, o *template* de treino que obtenha a maior pontuação é aquele que, de acordo com o reconhecedor, representa o gesto a reconhecer. Os restantes *templates* de treino, com menor pontuação, podem ser utilizados para lidar com a ambiguidade no desenho à mão, servindo assim como correspondências alternativas.

Os autores do reconhecedor $1 apresentam ainda formas de tornar o algoritmo sensível à dimensão e/ou orientação dos gestos, podendo esta característica ser aplicada a todos os *templates* de treino ou seletivamente, a *templates* individuais.

### 2.1.4 Reconhecedor $N

Com o objetivo de resolver algumas das limitações do reconhecedor $1, como a inaptidão para reconhecer gestos multi-traço, sensibilidade à direção de desenho, e problemas no reconhecimento de gestos unidimensionais como linhas, Anthony *et al.* otimizaram-no e criaram o reconhecedor $N [11].

Dado tratar-se de uma extensão do reconhecedor $1, o reconhecedor $N utiliza os mesmos princípios de funcionamento, em conjunto com algumas melhorias. Por exemplo, no sentido de reconhecer gestos multi-traço e ser insensível à direção de desenho, para cada *template* de treino multi-traço o algoritmo determina todas as combinações possíveis de traços (com diferentes direções e ordenações), criando um gesto de traço único por cada combinação. Cada um destes gestos é resultante da união das extremidades dos diversos traços do *template*, de acordo com a direção e ordenação definidas pela combinação que o origina. Na Figura 6 são apresentados os gestos de traço único resultantes deste processo quando aplicado ao *template* de um gesto "X" constituído por dois traços. Também os múltiplos traços de cada gesto a reconhecer são unidos mas, neste caso, dando origem a apenas uma representação de traço único formada com base na ordem pela qual os traços foram desenhados. O reconhecedor $N emprega então as mesmas quatro etapas definidas no reconhecedor $1 para o processamento e reconhecimento de gestos, o que se revela possível graças à conversão de todos os gestos multi-traço em representações de traço único.

[5] Dado que o gesto a reconhecer e os *templates* de treino são constituídos pelo mesmo número de pontos, as distâncias são facilmente obtidas entre pontos com o mesmo índice.

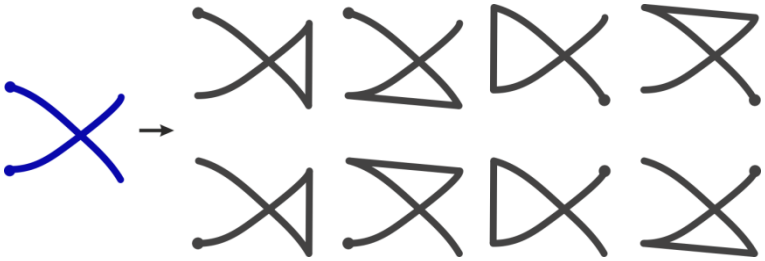


Figura 6 – Gestos de traço único resultantes das oito combinações possíveis para um gesto "X" composto por dois traços. Os círculos indicam o início do traço.

Para a resolução da problemática associada ao reconhecimento de gestos unidimensionais, como linhas horizontais ou verticais, estes são sujeitos a uma alteração uniforme da sua escala, em oposição à alteração não uniforme aplicada aos restantes gestos. Para tal, o reconhecedor $N faz uma diferenciação automática de gestos unidimensionais, com base no rácio entre a altura e largura do seu retângulo envolvente.

Apesar das melhorias em relação ao seu antecessor, o reconhecedor $N possui dificuldades ao nível do reconhecimento de gestos compostos por um número de traços superior aos definidos nos *templates* de treino. Para além disso, podem existir colisões entre gestos multi-traço diferentes que resultem em combinações de traço único iguais. Como pode ser observado na Figura 7, uma das representações de traço único resultantes do gesto multi-traço do sinal de igualdade ("=") pode ser facilmente confundida com a letra "Z". Uma outra limitação prende-se com o reconhecimento de gestos que assumem formas inconsistentes (ver gesto "Apagar" da Figura 3), pois o algoritmo depende da consistência na sequência de pontos para identificar corretamente cada gesto.

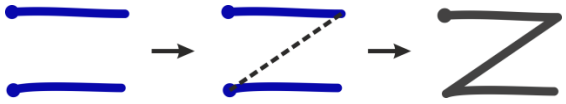


Figura 7 – Exemplo de conflito entre o símbolo "=" e a letra "Z"

### 2.1.5 Reconhecimento Baseado em Grafos

Lee *et al.* [12] desenvolveram um reconhecedor que utiliza grafos relacionais para reconhecer gestos multi-traço, independentemente da orientação, dimensões e direção de desenho dos mesmos. O reconhecedor possui ainda capacidades de treino, utilizando modelos estatísticos para construir cada classe gestual a partir dos *templates* de treino, o que o torna capaz de lidar com as variações inerentes ao desenho à mão de uma forma robusta.

Cada gesto é internamente representado por um grafo relacional, cujos vértices definem o tipo de primitiva (linha ou arco) e o seu comprimento relativo[6]. Já as arestas do grafo

[6] O comprimento relativo da primitiva corresponde ao rácio entre o seu comprimento absoluto e o comprimento total do gesto.

representam as relações geométricas entre cada par de primitivas. Estas relações geométricas são caracterizadas pelo número de intersecções, a localização dessas intersecções e o ângulo de intersecção (no caso de ambas as primitivas serem do tipo "Linha").

A segmentação dos gestos em primitivas individuais é conseguida com recurso a uma técnica que utiliza a velocidade de desenho para dividir um gesto complexo em segmentos de linha ou arcos [13]. A utilização de um reconhecedor de baixo nível ou pré-reconhecedor acarreta a desvantagem de os erros ocorridos no processo de segmentação se propagarem ao processo de reconhecimento.

O processo de treino deste reconhecedor consiste na criação de um grafo por classe gestual, resultante da média dos valores dos grafos dos diversos *templates* de treino dessa classe gestual. Esta abordagem obriga a que todos os *templates* de uma classe sejam desenhados com uma ordem de traços ou com uma orientação constantes.

Para reconhecer um gesto, o algoritmo compara-o às classes gestuais treinadas e calcula os valores de seis métricas de erro resultantes das diferenças entre o gesto a reconhecer e a classe gestual: número de primitivas, tipo de primitivas, tamanho relativo, número de intersecções, ângulo de intersecções e localização das intersecções. Os autores decidiram atribuir pesos diferentes a cada uma das métricas de erro, fazendo com que algumas métricas assumam maior relevância na distinção de gestos. Por fim, com base nas métricas de erro e respetivos pesos é calculado, para cada classe gestual, o grau de certeza dessa classe representar o gesto a reconhecer. A comparação entre os grafos de gestos a reconhecer e aqueles representantes das classes gestuais não é trivial, dado que o mesmo gesto pode ser desenhado com um número variado de primitivas, podendo ainda essas primitivas encontrar-se ordenadas de diversas formas. Para tal, os autores propõem e avaliam cinco técnicas de combinação aproximada de grafos e analisam as suas vantagens e desvantagens.

### 2.1.6 Reconhecimento Baseado em Energias de Deformação Elástica

Vatavu *et al.* [14] apresentam um reconhecedor com capacidades de treino que recorre a uma analogia com energias de deformação elástica para classificar gestos de traço único.

O reconhecedor considera os traços dos gestos como séries de molas elásticas ligadas entre si, cada uma com um determinado comprimento e curvatura, como apresentado na Figura 8. Através da medição da energia necessária para transformar as molas de um gesto nas molas de outro gesto, é possível determinar o grau de similaridade entre eles. Logo, para identificar um gesto, o reconhecedor computa a sua função de curvatura (as suas molas), com base na análise da sua trajetória, e calcula o custo de alinhamento desse gesto a cada uma das classes gestuais, ou seja, a energia necessária para deformar as "molas" do gesto nas "molas" da classe gestual considerada. A classe que minimiza esse custo ou energia é, portanto, aquela que representa o gesto.

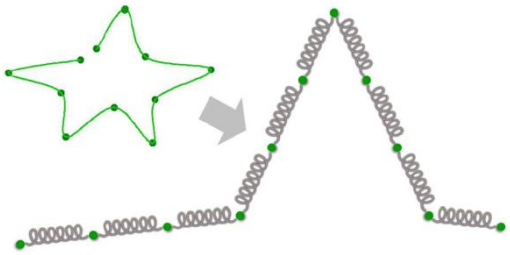


Figura 8 – Representação elástica de um gesto [14]

Relativamente ao processo de treino, a função de curvatura associada a cada classe gestual é resultante da média das funções de curvatura dos diversos *templates* da classe.

O reconhecedor é naturalmente insensível à escala e orientação dos gestos, pois um mesmo gesto possui funções de curvatura similares independentemente da orientação ou dimensão com que é desenhado. No entanto, o reconhecedor é sensível à direção de desenho bem como à posição no gesto do ponto inicial de desenho.

### 2.1.7 Reconhecimento Baseado em *Hidden Markov Models*

A utilização de técnicas baseadas em *Hidden Markov Models* (HMM) para reconhecimento de gestos constitui uma abordagem bastante popular. Por exemplo, Sezgin *et al.* [15] apresentam um reconhecedor multi-traço, baseado em HMM, capaz de reconhecer formas individuais em cenas complexas, mesmo quando a cena ainda não se encontra finalizada, ou seja, enquanto está a ser desenhada. Por outro lado, o reconhecedor apenas consegue identificar gestos quando desenhados com a orientação definida nos *templates* de treino, sendo portanto sensível a esta característica.

Durante o processo de reconhecimento, o algoritmo começa por criar sequências de observações discretas baseadas em treze tipos de características ou símbolos geométricos:

- Quatro para representar linhas: com inclinação positiva; com inclinação negativa; horizontais; verticais;
- Três para representar ovais: círculos; ovais horizontais; ovais verticais;
- Quatro para representar linhas poligonais: com duas, três, quatro, ou mais arestas;
- Um para representar aproximações complexas, como uma combinação de curvas e linhas;
- Um para representar dois traços consecutivos que se intersetam.

A identificação destes símbolos na cena é feita com recurso a uma ferramenta apresentada anteriormente pelo mesmo autor [16]. De seguida, o algoritmo compara várias subsecções[7] da cena com cada um dos HMM treinados, de forma a calcular a probabilidade de cada HMM representar as diversas subsecções. A partir dessas probabilidades, é construído um grafo cujo

[7] Conjuntos de símbolos identificados anteriormente na cena.

caminho mais curto indica a segmentação mais plausível da cena, ou seja, a composição dos gestos ou formas individuais que constituem a cena. Finalmente, para identificar cada um desses gestos apenas é necessário encontrar o HMM que maximiza a probabilidade de o gerar, sendo que cada HMM representa uma classe gestual.

Visto o reconhecimento ser dependente da ordem de traços dos *templates* de treino, este reconhecedor poderá não ser o mais adequado a domínios cuja ordenação de traços não possa ser prevista. Além disso, como os HMMs são apropriados para sequências, o reconhecimento de gestos compostos por um único traço não é possível a não ser que estes sejam pré-segmentados, isto é, repartidos em diversos traços.

### 2.1.8 PaleoSketch

Paulson e Hammond desenvolveram o PaleoSketch [17], um reconhecedor de primitivas de baixo nível compostas por apenas um traço. Para além do reconhecimento de primitivas, o reconhecedor é ainda capaz de identificar combinações dessas primitivas, reconhecendo um total de oito gestos: linha, linha poligonal, círculo, elipse, arco, curva, espiral e linha helicoidal.

Para serem corretamente identificados, os gestos a reconhecer passam por uma fase de pré-processamento, onde são removidos pontos duplicados, bem como ruído do início e fim do gesto, frequentemente resultante das ações realizadas pelo utilizador para iniciar e terminar o desenho, como encostar e levantar o estilete. Para além disso, ainda na fase de pré-processamento, são computados diversos grafos e valores utilizados posteriormente para caracterizar o gesto.

Após o pré-processamento sucede então o processo de reconhecimento. O PaleoSketch divide este processo em múltiplos reconhecedores[8] individuais que, com base nas características computadas na fase de pré-processamento, testam se o gesto corresponde a determinada primitiva. É importante referir que existe uma relação de um-para-um entre as primitivas e os reconhecedores, ou seja, um reconhecedor apenas identifica uma primitiva e vice-versa. Dado que um gesto pode ser identificado positivamente por vários reconhecedores em simultâneo, e dado que cada um deles apenas indica se o gesto foi reconhecido, sem indicar qualquer grau de (in)certeza associado, os resultados são transferidos para uma função de ordenação. Esta função utiliza um algoritmo de deteção de arestas que visa determinar o número mínimo de linhas para descrever corretamente o gesto e compara esse valor ao número mínimo de linhas definido para cada primitiva (e consequentemente para cada reconhecedor). A partir desta comparação, os resultados de reconhecimento são ordenados para que o resultado mais propenso a representar o gesto se encontre em primeiro lugar, podendo os restantes resultados, também ordenados, ser utilizados como identificações alternativas.

O PaleoSketch é insensível à orientação, dimensão e sentido de desenho, mas está limitado à identificação das oito primitivas já referidas. Para gestos mais complexos (como retângulos ou

[8] São designados de "sub-reconhecedores" pelo seu autor.

triângulos) é necessária a utilização de um reconhecedor de mais alto nível em conjunto com o PaleoSketch.

### 2.1.9 Comparação de Características

Apresenta-se de seguida, na Tabela 1, uma comparação das características principais de cada um dos reconhecedores apresentados anteriormente.

Tabela 1 – Comparação das características de reconhecedores

| Reconhecedor | Multi-traço | Treinável | Sensível à escala | Sensível à orientação | Sensível à direção de desenho |
|---|---|---|---|---|---|
| Reconhecedor de Rubine | Não | Sim | Sim | Sim | Sim |
| CALI | Sim | Não | Não | Não | Não |
| Reconhecedor $1 | Não | Sim | Não | Não | Sim |
| Reconhecedor $N | Sim | Sim | Não | Não | Não |
| Reconhecedor baseado em grafos | Sim | Sim | Não | Não | Não |
| Reconhecedor baseado energias de deformação | Não | Sim | Não | Não | Sim |
| Reconhecedor baseado em HMM | Sim | Sim | Não | Sim | Sim |
| PaleoSketch | Não | Não | Não | Não | Não |

### 2.1.10 Avaliação Automática de Reconhecedores Caligráficos

Inerente à existência de diversos reconhecedores caligráficos, surge a necessidade de comparação e avaliação da performance dos mesmos. No sentido de dar resposta a essa necessidade, Schmieder *et al.* [18] desenvolveram uma ferramenta que permite a avaliação automática de reconhecedores caligráficos. Esta ferramenta possibilita a integração de vários reconhecedores, com ou sem capacidades de treino, para que sejam testados e avaliados simultaneamente. Para além disso, a ferramenta possui ainda a capacidade de recolher e

etiquetar esboços, que podem ser utilizados para o processo de treino ou para a avaliação dos reconhecedores.

Dada a variedade de formatos de dados e linguagens de programação utilizadas por diferentes reconhecedores, cada reconhecedor necessita de uma classe de gestão associada. Para além de permitir a implementação de reconhecedores em C#, Java ou C++, esta classe é responsável por converter os dados entre o formato definido pela ferramenta e o utilizado pelo reconhecedor.

Quando é realizado um teste aos reconhecedores, a ferramenta procede à criação de uma folha de cálculo, no formato Microsoft Excel, com os valores e estatísticas resultantes desse teste. Em alternativa, podem ser criados *screenshots* dos gestos que foram classificados erroneamente durante o teste.

Para a demonstração do conceito, os autores integraram, testaram e analisaram os resultados de seis reconhecedores caligráficos: CALI, Microsoft Ink Analyser, o reconhecedor $1, o reconhecedor de Rubine com as características melhoradas presentes no InkKit, o PaleoSketch e ainda um reconhecedor que utiliza técnicas de *Dynamic Time Warping* (DTW). Os autores discutem ainda como o contexto do qual são obtidos os dados de treino dos algoritmos influencia a eficácia de reconhecimento.

## 2.2 Sistemas Caligráficos para a Simulação da Dinâmica dos Corpos Rígidos

A conjugação de uma interface caligráfica com um sistema de simulação da dinâmica dos corpos rígidos origina geralmente um resultado interessante, no qual os esboços ou gestos desenhados "ganham vida", como se possuíssem propriedades físicas.

Para além do âmbito e objetivos destes sistemas, existem algumas características base que os distinguem. Por exemplo, enquanto alguns sistemas apresentam os objetos simulados com uma representação idêntica àquela com que foram desenhados, mantendo as características do desenho à mão, noutros cada gesto reconhecido dá origem a um objeto com uma forma embelezada[9] ou pré-definida. Para além disso, a simulação da física pode ser instantânea e manipulável, na qual os gestos desenhados pelo utilizador dão origem a objetos que são imediatamente simulados, afetando outros objetos na cena, ou realizada apenas após a fase de desenho, sendo neste caso impossível desenhar novos gestos durante a simulação.

São de seguida apresentados alguns exemplos de sistemas que recorrem a interfaces caligráficas para a criação de cenas cuja dinâmica dos objetos é simulada.

[9] Do inglês "beautified". Neste contexto, embelezar um gesto significa converter as suas linhas curvas em linhas retas e direitas, removendo-lhe as características geralmente associadas ao desenho à mão.

### 2.2.1 ASSIST

Desenvolvido por Alvarado e Davis, o ASSIST [19] é um sistema que permite o desenho à mão de esboços de objetos mecânicos e que, após ser realizado o seu reconhecimento, possibilita realização e observação da simulação da dinâmica dos mesmos.

O sistema é capaz de reconhecer círculos e polígonos de linhas retas (simples ou complexos), constituídos por uma ou mais linhas. Permite também fixar dois objetos sobrepostos com um eixo de rotação, através do desenho de um pequeno círculo ou, com o desenho de uma pequena cruz, ancorar objetos ao fundo da aplicação, tornando-os assim estáticos durante a simulação. Após terminar o esboço, o utilizador pode, por intermédio de um botão, transferir os objetos para um simulador mecânico 2D que executa então a simulação da cena desenhada (Figura 9).

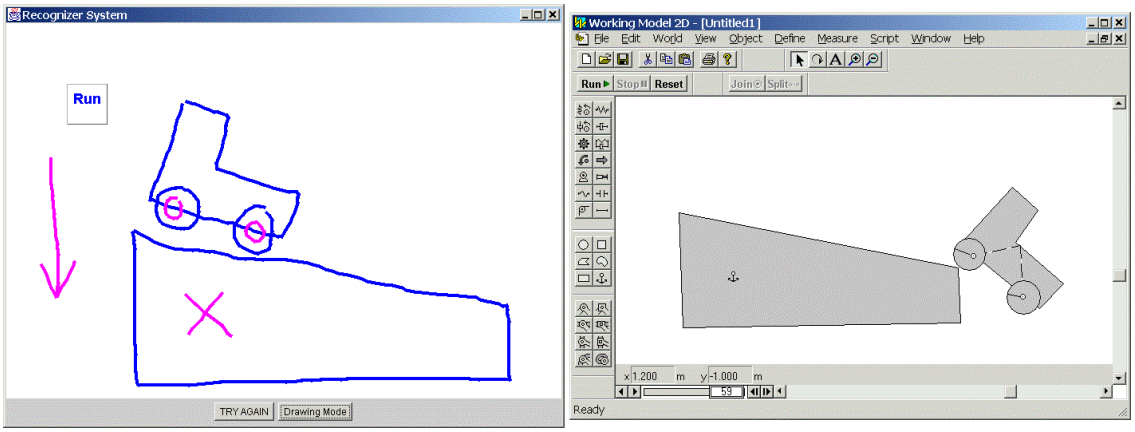



Figura 9 – Esboço realizado com o sistema ASSIST (à esquerda) e respetiva simulação (à direita)

O reconhecimento de esboços é realizado incrementalmente, enquanto o utilizador desenha cada traço, e não apenas quando termina o desenho de um objeto. Desta forma é possível desenhar vários objetos compostos por múltiplos traços, sem qualquer pausa ou tempo de espera, sendo estes identificados instantaneamente. Por exemplo, o utilizador pode desenhar um retângulo composto por quatro traços, seguido de um círculo composto por um único traço, de forma bastante natural e fluída. Para além de dar a perceção de um sistema mais rápido, pois o esforço de interpretação de traços é realizado em simultâneo com o seu desenho, esta abordagem fornece um *feedback* instantâneo ao utilizador, visto que as linhas desenhadas à mão são embelezadas e coloridas de acordo com o tipo de objeto reconhecido.

Durante o desenho da cena, o sistema está constantemente a avaliar os novos traços em conjunto com traços desenhados previamente, dado que um novo traço pode introduzir nova informação referente ao objeto que o utilizador está a construir e do qual traços anteriores poderão também fazer parte. A interpretação de esboços é então dividida em três fases: reconhecimento, raciocínio e resolução. A fase de reconhecimento identifica todas as interpretações possíveis do esboço no estado atual. A fase de raciocínio atribui uma pontuação a cada interpretação realizada na fase anterior, com base num conjunto de

heurísticas como evidência temporal, conhecimento do domínio e *feedback* do utilizador. Por fim, a fase de resolução seleciona as melhores interpretações com base nas suas pontuações, de uma forma coerente e consistente, para reconhecer todos os objetos da cena sem conflitos entre eles. De forma a melhor lidar com a ambiguidade, o utilizador tem sempre a possibilidade de corrigir e indicar ao sistema a interpretação correta através de uma lista de alternativas.

### 2.2.2 Free-Hand Sketch Recognition for Visualizing Interactive Physics

O sistema Free-Hand Sketch Recognition for Visualizing Interactive Physics (Reconhecimento de Esboços à Mão Livre para Visualização Interativa de Física) [20] possibilita o desenho de objetos 2D simples e subsequente simulação do comportamento desses objetos em 3D (Figura 10).

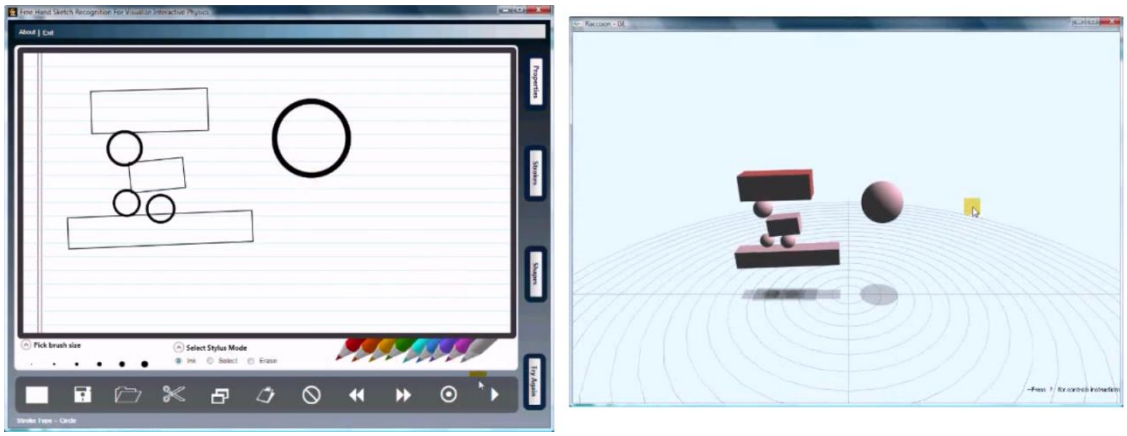


Figura 10 – Sistema Free-Hand Sketch Recognition for Visualizing Interactive Physics: interface de desenho (à esquerda) e janela de simulação (à direita).

O sistema suporta o desenho de quatro tipos de objetos: linhas, círculos, retângulos e triângulos. Sempre que o sistema é incapaz de reconhecer um dado gesto, é apresentada uma caixa de diálogo solicitando ao utilizador que indique qual o objeto pretendido. Após criar um objeto o utilizador pode ainda ancorá-lo, garantindo assim que esse objeto permanecerá estático durante a simulação.

Para permitir a criação mas também edição da cena, o processo de desenho é dividido em três modos: o modo “Ink”, utilizado para desenhar e criar novos objetos; o modo “Select”, no qual o desenho de uma elipse em torno de um objeto seleciona-o, permitindo a alteração das suas propriedades (como a sua massa ou estado de ancoragem); e o modo “Erase”, que possibilita a remoção de objetos. Apesar do processo de desenho ser realizado em 2D, a simulação física é tridimensional e admite que o utilizador manipule manualmente a posição da câmara bem como dos objetos no espaço 3D, não possibilitando, no entanto, a criação de novos objetos.

### 2.2.3 Crayon Physics Deluxe

Um jogo popular que alia a interação caligráfica com simulação de física é o Crayon Physics Deluxe [21], disponível para os sistemas operativos Microsoft Windows, Linux, Mac OS X e iOS, e cujo principal objetivo é conduzir uma bola na tentativa de a fazer atingir outros objetos (estrelas) existentes em cada nível do jogo.

Apesar de poder clicar ou pressionar sobre a representação da bola para a mover com pequenos impulsos, a ideia principal é a de que o utilizador desenhe objetos que colidam com esta, no sentido de influenciar o seu movimento ou até transportá-la, conduzindo-a assim para as estrelas. Os objetos podem ser desenhados livremente, com qualquer forma e sem quaisquer restrições. Para além disso, é possível unir dois objetos através de eixos de rotação e cordas, desenhando um pequeno círculo ou um traço entre eles, respetivamente. Em alguns níveis existem ainda foguetes que disparam quando sofrem um embate de outro objeto e que podem ser utilizados para atingir o objetivo. Por exemplo, o utilizador pode encapsular a bola dentro de uma elipse e, com uma corda, prender a elipse ao foguete, para que este transporte a bola quando disparar. Dado que a simulação se encontra constantemente em execução, os objetos são simulados logo após serem desenhados, resultando numa resposta imediata à interação.

O jogo possui uma temática de "desenho de criança", como pode ser observado na Figura 11, com um fundo que remete para uma folha de papel amarela e traços que aparentam ser desenhados com lápis de cera. Estas características, aliadas ao desenho livre, fazem com que o Crayon Physics Deluxe adote o paradigma papel-lápis com sucesso.

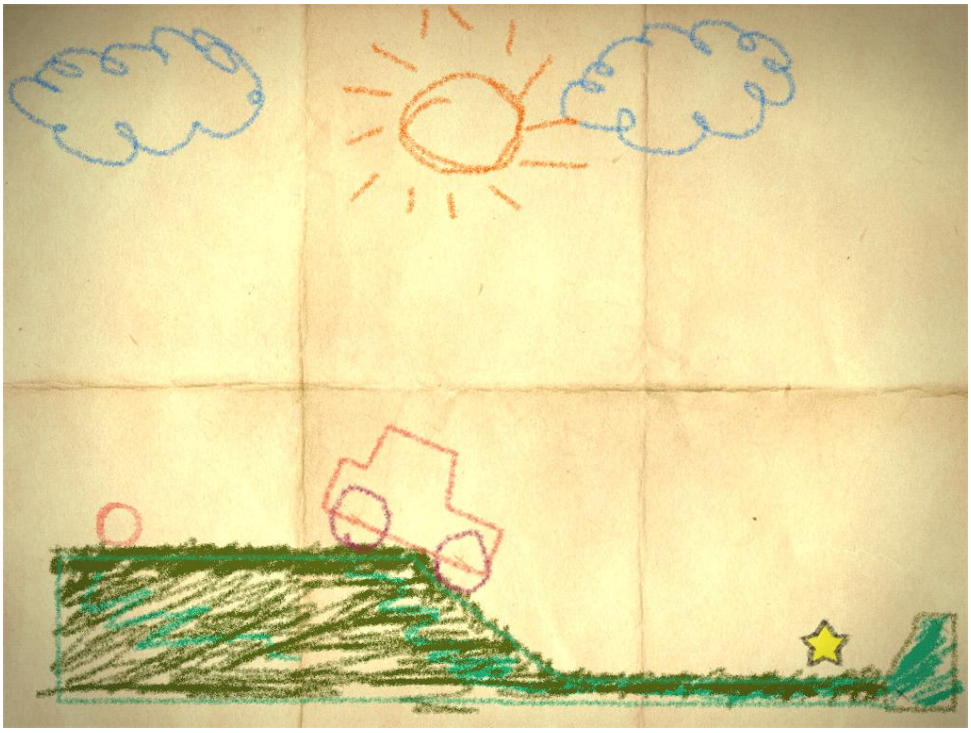


Figura 11 – Crayon Physics Deluxe

A liberdade de criação dada ao jogador faz com que cada nível possa ser ultrapassado das mais diversas formas, através de diferentes composições e combinações de objetos.

O Crayon Physics Deluxe inclui ainda um editor de níveis e um repositório *online*, para que os utilizadores possam criar níveis personalizados e partilhá-los com outros utilizadores. Tal como o jogo, também o editor utiliza o estilo de interação caligráfico, dispondo no entanto de funcionalidades extra, como a possibilidade de deslocar objetos manualmente, criar objetos estáticos e até estrelas (as quais representam os objetivos de cada nível).

### 2.2.4 Space Physics

Desenvolvido para o sistema operativo Android, o Space Physics [22] é um jogo com uma interação, jogabilidade e objetivos similares ao Crayon Physics Deluxe (ver secção 2.2.3), no qual o jogador tem que conduzir uma bola verde com o intuito de alcançar as estrelas existentes em cada nível. Apesar de não existir qualquer restrição temporal para completar cada nível, o jogador é incentivado a fazê-lo no menor intervalo de tempo, no sentido de obter um melhor desempenho. Para além disso, o jogador não está restringido à progressão na sequência de níveis, podendo escolher e jogar qualquer nível em qualquer altura. Na Figura 12 são apresentados quatro níveis distintos, onde é possível observar diversos objetos existentes no jogo, bem como a sua temática de fantasia futurística.

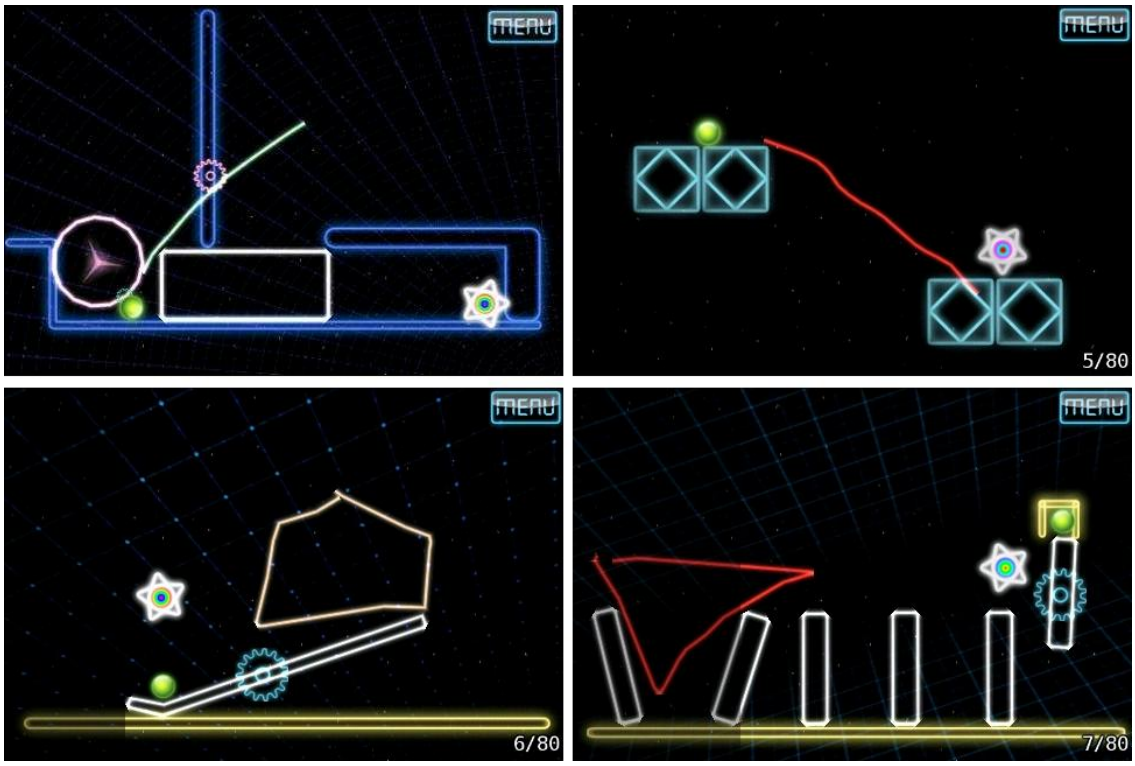



Figura 12 – Space Physics (quatro níveis distintos)

Tal como no Crayon Physics Deluxe, a interação consiste no desenho de objetos que, ao serem animados e simulados, colidam ou transportem a bola em direção ao objetivo. É ainda possível, através de dois toques no ecrã sobre a representação da bola, movê-la com pequenos impulsos. O Space Physics não coloca qualquer restrição no desenho de objetos, podendo estes assumir qualquer forma, incluindo simples linhas. No entanto, quando o jogador desenha um círculo, este é transformado numa roda ou, caso seja desenhado sobre um objeto já existente, numa engrenagem presa ao objeto sobre a qual foi desenhada. Ao desenhar uma linha sobre duas rodas, estas são interligadas e passam a rodar em torno do seu centro, movendo-se no sentido de desenho da linha. Já as engrenagens funcionam como eixos de rotação livre e permitem que um objeto rode sobre outro.

### 2.2.5 Physamajig

Physamajig [23] [24] é uma aplicação desenvolvida por Andy Beaulieu para o sistema operativo Microsoft Windows 8, na qual é possível desenhar objetos 2D de forma livre, simular o seu comportamento físico e até criar pequenos jogos (mini-jogos). O utilizador pode ainda guardar as suas criações em disco, bem como partilhá-las *online*, sendo possível visualizar e experimentar criações partilhadas por outros utilizadores diretamente a partir do menu da aplicação.

Para além de possibilitar o desenho livre de objetos com qualquer formato, a aplicação permite ainda criar círculos, retângulos e polígonos arbitrários, com a dimensão e forma que o utilizador pretender. Para tal, a aplicação disponibiliza um menu de ferramentas no qual é possível escolher qual o tipo de objeto a desenhar. Após desenhar um objeto, o utilizador pode selecioná-lo, de forma a ser exibido um menu de opções de edição do objeto, que possibilita que este seja duplicado, eliminado, ou que se altere algumas das suas características físicas como a fricção e elasticidade. Para além disso, permite colocar o objeto à frente ou atrás de todos os outros, o que é particularmente útil para conferir uma sensação de profundidade nos objetos de cenário. É ainda possível alterar a cor ou textura de cada objeto, bem como a cor e espessura da sua linha.

Para cada objeto criado, o Physamajig permite definir a sua natureza: dinâmica, estática ou de cenário. Os objetos dinâmicos são objetos animados, que colidem com outros objetos e reagem a essas colisões. Já os objetos estáticos, apesar de colidirem com outros objetos, não se movem e, portanto, não reagem a colisões. Por fim, os objetos de cenário, ao contrário dos restantes, não fazem parte da simulação física, sendo utilizados apenas para construir e decorar o cenário.

Uma outra funcionalidade importante do Physamajig é a possibilidade de interligar dois objetos, restringindo os seus movimentos relativos, através de dois tipos de articulações: roda e deslizante. Uma articulação de roda permite fixar um objeto “A” a um objeto “B” com uma linha imaginária (e invisível) que possui um eixo de rotação na extremidade onde se encontra o objeto “A”. Isto permite ao objeto “A” rodar em torno do ponto de fixação, ou ao objeto “B”

rodar em torno do objeto “A”. Este movimento, em torno do eixo de rotação, pode ser livre, estar limitado a um angulo mínimo e máximo ou ser tratado como um motor, através da definição da sua velocidade e força. Já as articulações deslizantes restringem o movimento relativo entre dois objetos a um eixo (horizontal ou vertical). Além disso, permitem que sejam definidas distâncias mínimas e máximas entre os dois objetos, bem como força e velocidade de translação. Desta forma, as articulações deslizantes possibilitam, por exemplo, a criação de plataformas que se movem horizontalmente de forma constante. A Figura 13 apresenta a restrição de movimento aplicada por cada um dos dois tipos de articulações, no qual os objetos se encontram representados a preto, as linhas tracejadas a vermelho representam a linha imaginária de restrição do movimento e as setas representam o movimento relativo possível.

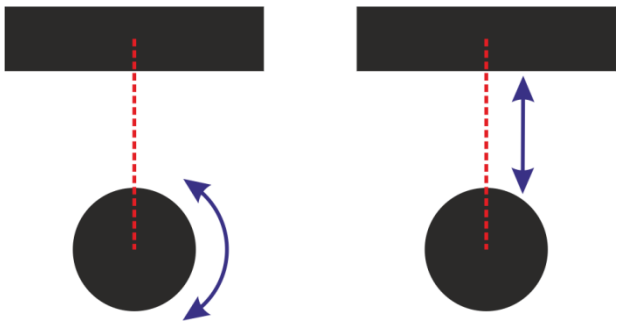


Figura 13 – Restrição de movimento relativo aplicado pelas articulações de roda (esquerda) e deslizante (direita), existentes na aplicação Physamajig

Uma outra funcionalidade interessante do Physamajig é a criação de mini-jogos. Estes são pequenos jogos com objetivo e interação simples, tais como controlar e disparar um canhão para destruir criaturas inimigas (Figura 14a), tentar encestar uma bola de basquete ou até pousar uma nave lunar na plataforma de aterragem (Figura 14b). Deste modo, o Physamajig não consiste apenas numa aplicação de simulação, mas também num sistema para criação de conteúdos interativos. Apesar de não ser possível a definição de sequências de níveis para cada mini-jogo, um efeito similar pode ser obtido através da criação de vários mini-jogos. Visto que um jogo tem geralmente como princípio base o desencadear automático de eventos a partir do *input* do utilizador, bem como a validação da concretização de um objetivo, o Physamajig disponibiliza o sistema *“Trigger - Behaviour”* para esse efeito. Um *Trigger* consiste num evento ou acontecimento, tal como um objeto a ser arrastado pelo utilizador, uma tecla pressionada, uma colisão entre objetos, entre outros. Já um *Behaviour* consiste na reação ou resposta desencadeada por um *Trigger* e que se pode traduzir na aplicação de força ou destruição de um ou mais objetos. Pode também constituir a reprodução de um efeito sonoro, término do jogo, entre outras situações. A grande vantagem deste sistema é que todos os seus parâmetros podem ser configurados de forma simples e intuitiva, sem qualquer necessidade de programação ou utilização de código fonte, sendo acessível a qualquer utilizador.

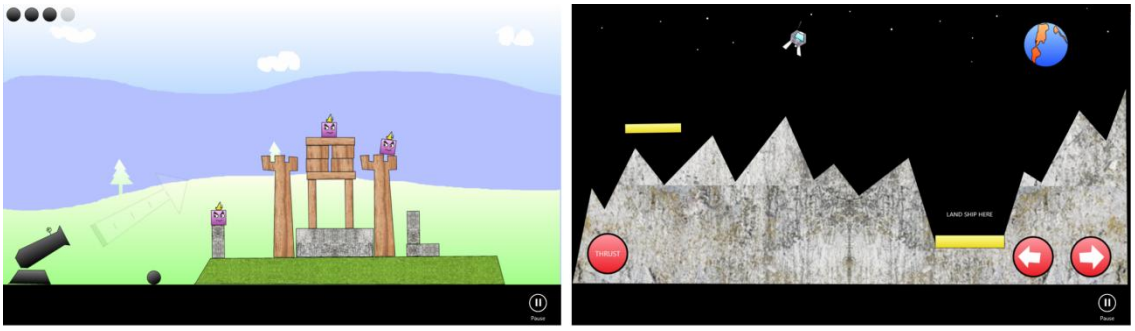



Figura 14 – Dois mini-jogos incluídos no Physamajig: Mad Cannon (à esquerda) e Moon Lander (à direita).

## 2.3 Conclusão

Neste capítulo foram apresentados diversos algoritmos de reconhecimento caligráfico, com o objetivo de conhecer e compreender a diversidade de soluções existentes atualmente. Destes algoritmos, alguns verificam-se orientadas para determinado problema ou domínio, enquanto outros pretendem oferecer soluções genéricas. Para além da variedade de técnicas utilizadas para o reconhecimento, desde simples características geométricas até avançados modelos estatísticos, também as capacidades associadas a cada algoritmo se mostram diversificadas. Apesar de determinadas características destes algoritmos, tais como a insensibilidade à orientação ou direção de desenho, poderem ser consideradas como limitações, estas podem-se revelar importantes e fundamentais num dado contexto ou aplicação. Desta forma, antes da decisão de utilização de determinado algoritmo de reconhecimento caligráfico, torna-se importante avaliar as diversas soluções existentes, analisando o propósito e características de cada uma, pelo que o estudo aqui apresentado constituiu uma fase essencial no processo de desenvolvimento deste trabalho.

Foram também apresentados aqui alguns sistemas que recorrem a interfaces caligráficas para a interação com um ambiente de simulação da dinâmica de corpos rígidos. Estes sistemas, cujo conceito subjacente vai de encontro ao princípio base do trabalho proposto neste documento, mostram o potencial da utilização deste tipo de interfaces para a criação e controlo de simulações da física, pois proporcionam uma forma natural e estimulante de interação com o sistema.

# 3 Implementação e Avaliação de Sistemas de Reconhecimento Caligráfico

A interface caligráfica, de todo o conjunto de componentes que constituem a solução proposta neste trabalho, destaca-se pela sua elevada importância e vitalidade para o sucesso da adoção da solução. Tal como qualquer interface humano-computador, constitui o ponto de comunicação entre o utilizador e o sistema, pelo que, para o utilizador, a interface é o "sistema". Para além disso, como interface caligráfica cujo objetivo é permitir o desenho livre e não restritivo, é necessário que esta proporcione ao utilizador uma forma natural de interagir com o sistema, sendo simultaneamente capaz de interpretar corretamente as intenções subjacentes a essa interação. Neste sentido, o correto reconhecimento caligráfico representa uma peça chave para o sucesso da interface caligráfica.

Com o intuito de determinar qual o reconhecedor caligráfico que melhor se adequa às especificidades do trabalho aqui apresentado, foi conduzida uma avaliação de três reconhecedores. Para tal, foi necessário proceder à implementação destes e desenvolver um sistema que permitisse a sua integração, teste e análise. Este capítulo detalha a implementação desse sistema, a metodologia de desenvolvimento adotada e a consequente integração dos reconhecedores caligráficos. Posteriormente, é também apresentada a avaliação realizada aos reconhecedores e discutidos os seus resultados.

É importante referir que a implementação dos reconhecedores e do sistema no qual foram integrados foi realizada com o objetivo de suportar gestos de traço único, não permitindo gestos multi-traço, dado esta ser uma das características definidas para o trabalho apresentado neste documento.

## 3.1 Metodologia de Desenvolvimento

De modo a que o processo de desenvolvimento de um sistema seja bem-sucedido, é necessária a adoção de uma metodologia adequada. No desenvolvimento de *software* é frequente o recurso a uma abordagem *top-down*, baseada na decomposição de funcionalidades. Esta abordagem está geralmente associada a um ciclo de desenvolvimento que se inicia com a análise do sistema e percorre, de forma sequencial, as etapas de levantamento de requisitos, conceção, desenvolvimento de protótipo, implementação e, por fim, testes e avaliação. Denominado frequentemente de metodologia em cascata, devido à sua estrutura essencialmente sequencial, este tipo de ciclo de vida não se adequa ao desenvolvimento de aplicações complexas, o qual requer um alternar constante (e em ambos os sentidos) entre as diversas etapas de desenvolvimento.

Apresentada por Boehm [25], a metodologia em espiral adota uma abordagem iterativa, e visa colmatar o problema inerente à estrutura sequencial da metodologia em cascata. Na metodologia em espiral, não obstante ser utilizada uma abordagem *top-down* e uma sequência de etapas idêntica à da metodologia em cascata, esta é repetida várias vezes, de forma a incluir cada vez mais informação e detalhe em cada uma das etapas.

Hix e Hartson, após a análise de algumas metodologias de desenvolvimento *top-down*, como as apresentadas anteriormente, e a observação do processo de trabalho de *designers* de *software* e de interação, concluíram que este processo é beneficiado quando realizado sob a forma de ondas alternadas de dois tipos de atividades complementares: atividades *bottom-up* (de síntese, concretas, e criativas, que refletem a aproximação do ponto de vista do utilizador ao do sistema) e atividades *top-down* (de análise, abstratas, e estruturantes, que refletem a aproximação do ponto de vista do sistema ao do utilizador). Logo, é conjugado um modo de síntese, que permite a criação, e um modo de análise, que permite a avaliação do que foi criado. Além disso, este modo de trabalhar ajusta-se bem à ideia de prototipagem rápida, o que tem a vantagem de permitir a avaliação numa fase mais precoce do desenvolvimento [26].

Hix e Hartson verificaram ainda que a abordagem iterativa é preferida em relação à sequencial, pois esta última requer, por exemplo, que o levantamento de requisitos esteja completamente concluído antes da realização da etapa de conceção, o que não é, geralmente, possível. Além disso, nomeadamente em sistemas interativos, a imprevisibilidade do comportamento do utilizador[10] obriga a que exista um processo iterativo de desenvolvimento, orientado para a autocorreção.

Com base nas suas observações, Hix e Hartson propuseram uma metodologia em estrela (Figura 15) de desenvolvimento de sistemas interativos. Ao contrário da abordagem tradicional e estritamente sequencial, a metodologia em estrela adota uma abordagem iterativa. No entanto, em oposição ao que sucede na metodologia em espiral, as etapas não

[10] Um sistema interativo inclui na sua constituição um coprocessador dotado de um comportamento amplamente imprevisível: o utilizador humano [38].

necessitam de ser abordadas por uma sequência específica, pelo que as restrições que decorrem da imposição de uma ordem são minimizadas. Esta característica, que alia a abordagem *top-down* à abordagem *bottom-up*, permite que se inicie a fase de conceção sem que a fase de análise de requisitos esteja concluída, podendo voltar-se a esta em qualquer altura do processo de desenvolvimento. Na metodologia em estrela, a única restrição que é colocada na transição de uma atividade para outra é a existência de uma avaliação de usabilidade dos resultados da atividade anterior. Como consequência, esta metodologia mostra-se centrada na avaliação, promovendo assim sucessivos aperfeiçoamentos do sistema em desenvolvimento.

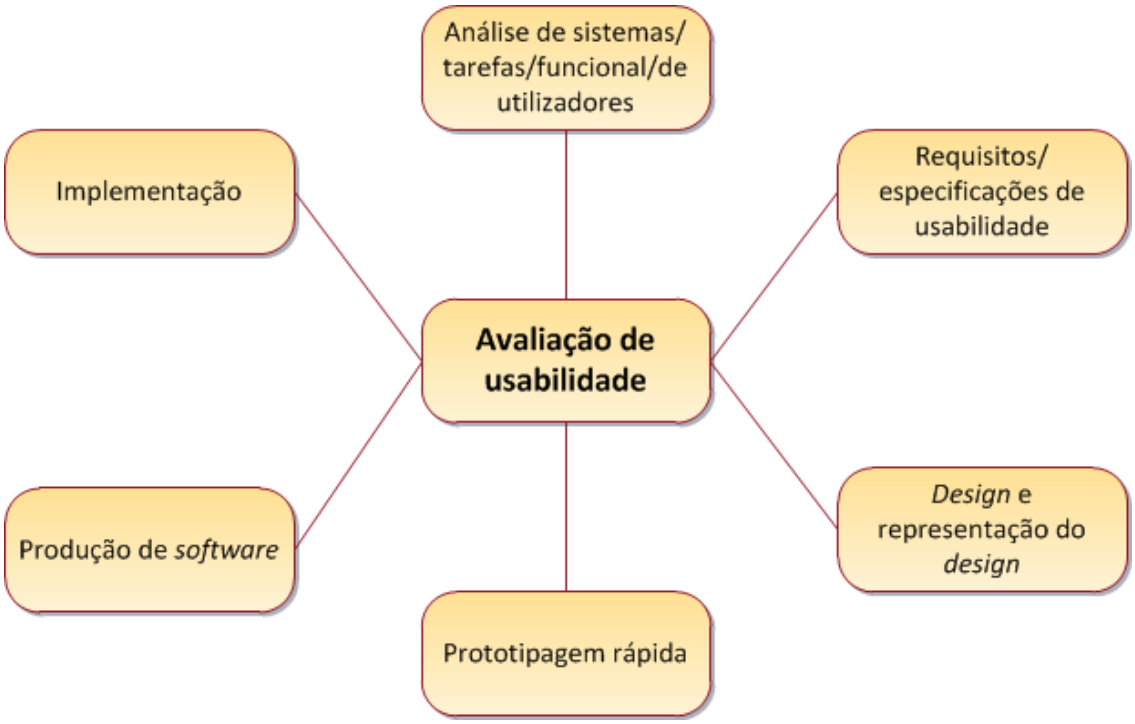



Figura 15 – Metodologia em estrela para o desenvolvimento de *software* interativo

O sistema apresentado neste capítulo foi desenvolvido de acordo com a metodologia em estrela, tendo o seu processo de desenvolvimento sofrido várias iterações. Em cada iteração, o sistema foi testado e avaliado informalmente por diversos utilizadores e, tendo em conta os seus comentários e sugestões, foram concretizadas as correções e melhorias correspondentes.

## 3.2 Sistema SketchTester

O sistema SketchTester teve essencialmente o propósito de ser uma ferramenta de apoio no teste e apreciação de algoritmos de reconhecimento caligráfico, que permitisse a integração de vários reconhecedores caligráficos e a definição dos *templates* de treino respetivos. Com esta ferramenta, seria possibilitada também a recolha de amostras gestuais de utilizadores e respetiva extração de dados de reconhecimento.

### 3.2.1 Arquitetura

A arquitetura do sistema SketchTester encontra-se dividida em três componentes principais: interface, suporte de reconhecedores e suporte de gestos. A componente da interface é composta pelas classes que definem as janelas da aplicação. Na componente de suporte de reconhecedores estão contidas as classes responsáveis pelo reconhecimento de gestos, ou seja, os reconhecedores caligráficos. Por fim, a componente de suporte de gestos possuí classes auxiliares para a representação e processamento de gestos. No diagrama da Figura 16 encontram-se representadas as classes fundamentais do sistema SketchTester, agrupadas pelas componentes que as retratam, bem como os ficheiros de dados geridos pela aplicação.

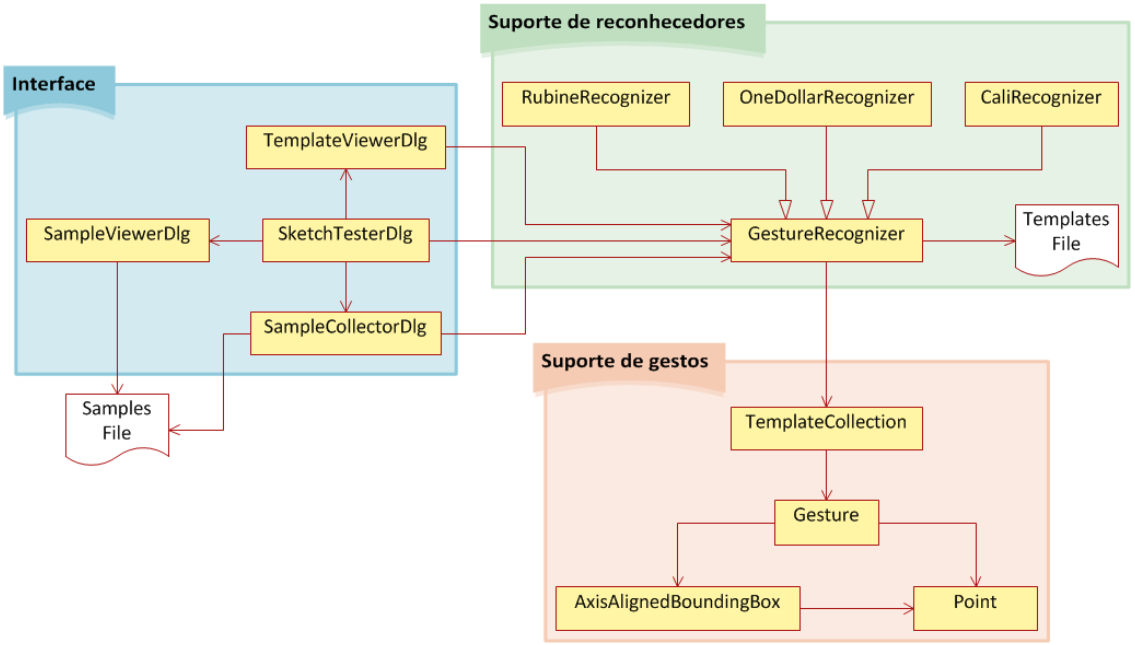



Figura 16 – Diagrama das principais classes do sistema SketchTester

Cada uma das classes existentes na componente da interface representa uma janela da aplicação. Por exemplo, as classes SketchTesterDlg e TemplateViewerDlg representam a janela principal da aplicação e a janela de visualização de *templates*, respetivamente, descritas na secção 3.2.2. Já as classes SampleCollectorDlg e SampleViewerDlg constituem as janelas de recolha e visualização de amostras gestuais, apresentadas nas secções 3.2.3 e 3.2.4, respetivamente.

Para possibilitar a análise dos dados resultantes da recolha de amostras gestuais, estes são armazenados num ficheiro em disco. A criação e escrita deste ficheiro são realizadas pela classe SampleCollectorDlg e a sua leitura é conseguida através classe SampleViewerDlg. Para além de cada um dos pontos que constituem os gestos desenhados pelo utilizador, é também armazenada cada uma das indicações do sistema sobre qual o gesto a desenhar e o respetivo resultado de reconhecimento por parte de cada reconhecedor. Incluir os resultados de reconhecimento de cada gesto no ficheiro possibilita que, posteriormente, estes dados sejam analisados sem a necessidade de voltar a executar os processos de reconhecimento. Na Figura

17, encontra-se exemplificado o formato utilizado por este ficheiro que, como pode ser observado, é construído em XML.

```
<Session>
    <Sample Number="1">
       <Recognition Instruction="Rectangle" CALI="Rectangle"
          OneDollar="Rectangle" Rubine="Rectangle"/>
       <Gesture>
          <Point X="131" Y="108"/>
          <Point X="131" Y="110"/>
          <Point X="131" Y="112"/>
          ...
       </Gesture>
    </Sample>
    <Sample Number="2">
       <Recognition Instruction="Triangle" CALI="Zigzag"
          OneDollar="Triangle" Rubine="Rectangle"/>
       <Gesture>
          ...
       </Gesture>
    </Sample>
    <Sample Number="3">
       ...
    </Sample>
    ...
</Session>
```

Figura 17 – Formato utilizado pelo ficheiro de dados de amostras gestuais

Relativamente à componente de suporte de reconhecedores, esta é constituída pelas classes que implementam os reconhecedores caligráficos. A sua classe central, GestureRecognizer, é uma classe abstrata da qual derivam as classes referentes a cada reconhecedor. Nesta classe estão implementadas diversas funcionalidades genéricas e úteis à maioria dos reconhecedores, tais como a gestão de *templates* e a sua leitura/escrita em ficheiro. Esta define também um conjunto de funções que formam uma interface comum entre as diversas implementações dos reconhecedores. Desta forma, para integrar um novo reconhecedor caligráfico no sistema, apenas é necessário proceder à implementação das suas especificidades numa classe derivada da GestureRecognizer, em conjunto com pequenas alterações à interface da aplicação.

As classes RubineRecognizer, OneDollarRecognizer e CaliRecognizer constituem a implementação dos três reconhecedores integrados no sistema SketchTester: o reconhecedor de Rubine, o reconhecedor $1 e o CALI. Estas classes derivam da classe GestureRecognizer e implementam os algoritmos dos respetivos reconhecedores ou, no caso da CaliRecognizer, fazem a integração da biblioteca do reconhecedor com o restante sistema.

Como referido anteriormente, os *templates* de treino dos reconhecedores são armazenados em disco, em ficheiros geridos pela classe GestureRecognizer. Para cada reconhecedor que utilize *templates* de treino é criado um ficheiro XML individual, no qual são armazenadas as classes gestuais treinadas e os respetivos *templates*. Na Figura 18 é apresentado um excerto

de um ficheiro de *templates*, de modo a ilustrar o seu formato. É importante referir que o termo "Template Collection", exibido neste excerto, se refere a uma coleção de *templates*, ou seja, uma classe gestual.

```
<TemplateCollections>
    <TemplateCollection Name="Alpha">
       <Gesture>
          <P X="264" Y="125"/>
          <P X="262" Y="127"/>
          <P X="261" Y="129"/>
       </Gesture>
       <Gesture>
          ...
       </Gesture>
       ...
    </TemplateCollection>
    <TemplateCollection Name="Circle">
       ...
    </TemplateCollection>
    ...
</TemplateCollections>
```

Figura 18 – Excerto de um ficheiro de armazenamento de *templates* de treino

Finalmente, a componente de suporte de gestos é constituída por classes auxiliares, utilizadas para o armazenamento e processamento de dados relacionados com os gestos. A classe central neste componente é a classe Gesture, que representa todas as características de um gesto e permite a realização sobre este de um conjunto de operações úteis. Por exemplo, a partir da classe Gesture é possível obter o comprimento total do traço do gesto ou o seu centro geométrico, bem como proceder a transformações geométricas como rotação, escala e translação.

Criada com o objetivo inicial de representar os pontos que definem um gesto, a classe Point armazena um par de coordenadas (x,y) inscritas no plano 2D. No entanto, para além de pontos, esta classe é também utilizada para armazenar e processar vetores 2D, o que simplifica a realização de operações que envolvem pontos e vetores. A classe Point possui portanto métodos para realizar comparações booleanas, operações aritméticas, determinar a distância entre dois pontos e ainda computar a magnitude de um vetor.

A classe AxisAlignedBoundingBox representa um retângulo envolvente alinhado com os eixos, utilizado por alguns reconhecedores para caracterizar gestos ou realizar operações sobre eles. Exemplificando, o reconhecedor $1 utiliza as dimensões deste retângulo para proceder à alteração da escala do gesto.

Por fim, a classe TemplateCollection consiste em uma coleção de gestos à qual é atribuído um nome, representando assim uma classe gestual. Esta classe é utilizada pelos reconhecedores caligráficos com capacidades de treino para armazenar as suas classes gestuais e respetivos *templates*.

Grande parte das funcionalidades implementadas pelas classes do sistema SketchTester foram posteriormente reutilizadas na implementação da biblioteca SketchyDynamics (capítulo 4), pelo que a sua utilidade foi estendida para além do âmbito descrito neste capítulo.

De modo a restringir as funcionalidades apresentadas aos participantes da sessão de recolha de amostras, a aplicação SketchTester dispõe de dois perfis de utilização: o administrador e o participante. Enquanto que o perfil de administrador permite o uso de todas as componentes da aplicação, o perfil de participante apenas concede o acesso a uma parte muito restrita da interface, direcionada para a recolha de amostras.

### 3.2.2 Gestão de Reconhecedores Caligráficos

A partir da janela principal da aplicação SketchTester (Figura 19), disponível apenas no perfil de administrador, é possível validar a implementação e as capacidades de reconhecimento dos diversos reconhecedores caligráficos, como também adicionar *templates* de treino aos reconhecedores cujo funcionamento deles dependa. Como o principal objetivo desta janela é possibilitar o teste rápido das capacidades dos reconhecedores implementados, é disponibilizada uma área para o desenho livre de gestos, sendo apresentados os resultados de reconhecimento à direita dessa área.

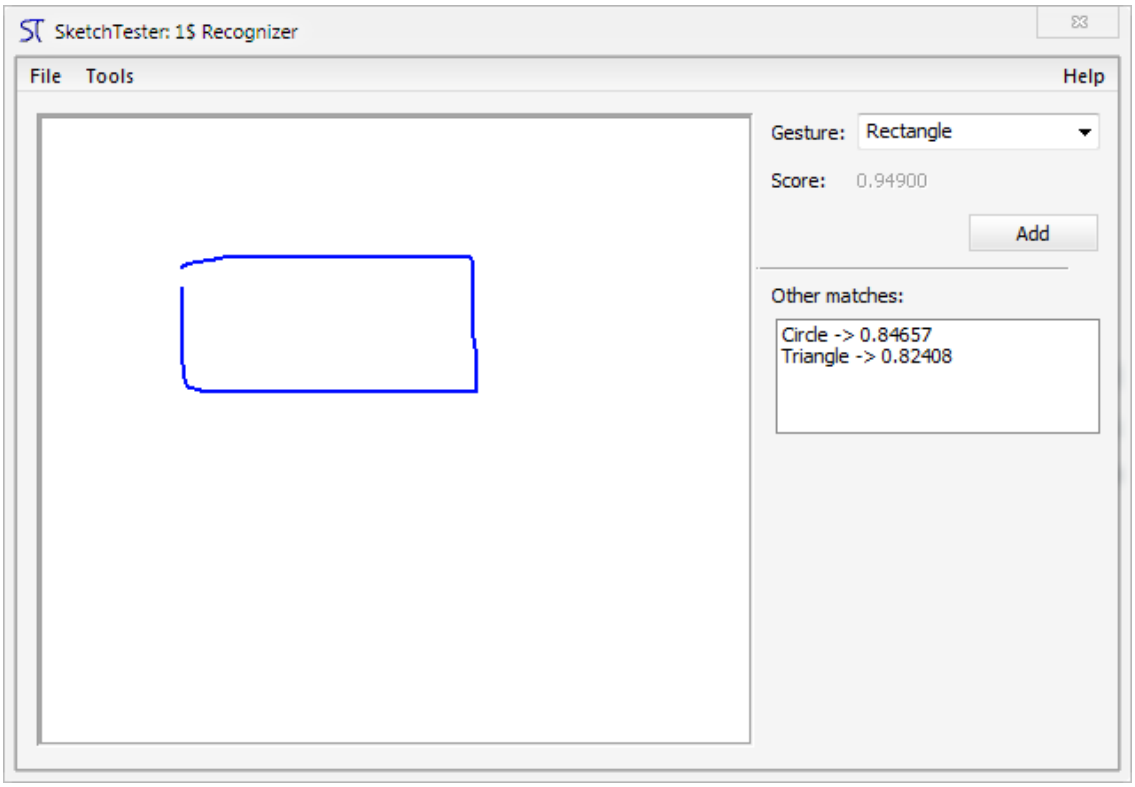



Figura 19 – Janela principal da aplicação SketchTester

Sempre que é realizado um gesto, a aplicação transfere-o para o reconhecedor que se encontra ativo, para que este realize o processo de reconhecimento. Seguidamente, na área lateral direita da aplicação, são exibidos os resultados desse processo, como a classe gestual que identifica o gesto e o respetivo grau de certeza. São ainda referidas outras classes gestuais consideradas pelo reconhecedor como identificações alternativas, bem como o grau de certeza associado a cada uma. A escolha do reconhecedor ativo é feita a partir do menu "Tools", sendo apresentado no título da janela da aplicação a identificação desse mesmo reconhecedor.

No sentido de facilitar a adição de *templates* de treino a cada reconhecedor, a aplicação permite que cada gesto desenhado seja utilizado para esse efeito. Para tal, após desenhar um gesto, basta selecionar a classe gestual à qual o novo *template* deverá ser associado, a partir da lista de classes gestuais (onde é igualmente mostrado o resultado de reconhecimento), e pressionar o botão "Add". É ainda possível associar o *template* a uma nova classe gestual, bastando para isso introduzir a sua designação no lugar de selecionar uma classe já existente. Assim é permitido que um gesto não reconhecido (ou reconhecido erroneamente) seja facilmente adicionado à coleção de *templates* do reconhecedor, possibilitando o seu correto reconhecimento daí em diante. Tal como acontece para o reconhecimento dos gestos desenhados, a adição de *templates* é realizada apenas para o reconhecedor selecionado e ativo. É de realçar que esta funcionalidade só se encontra disponível para reconhecedores com capacidades de treino.

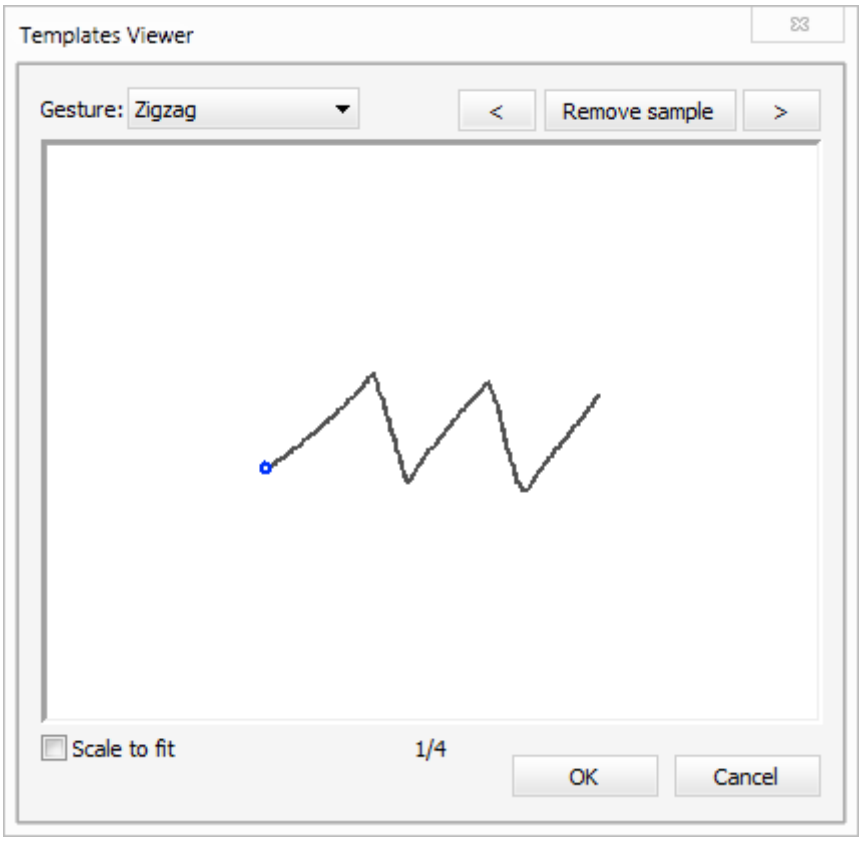



Figura 20 – Janela de visualização de *templates* de treino

A aplicação SketchTester possibilita ainda visualizar e remover *templates* de treino previamente adicionados, através da janela de visualização de *templates* (Figura 20). Esta janela ostenta uma lista das classes gestuais definidas, bem como controlos para navegar no conjunto de *templates* de cada classe, os quais se encontram organizados pela sua ordem de criação. No centro da janela é apresentado o *template* atual, cujo ponto inicial se encontra assinalado através de um pequeno círculo azul. Por baixo da representação do *template* é ainda exibido o seu índice (no conjunto de *templates* da classe gestual atual) e o número total de *templates* associados à classe. Se se pretender efetuar a remoção do *template* em exibição, é também possível fazê-lo através do botão "Remove sample".

### 3.2.3 Recolha de Amostras Gestuais

Para a realização das sessões de recolha de amostras gestuais, descritas em detalhe na secção 3.4, a aplicação SketchTester incorpora uma janela responsável por conduzir o utilizador durante o processo de recolha e armazenar os seus dados gestuais. Para tal, esta janela sugere progressivamente ao utilizador qual o gesto que deve ser desenhado, até serem recolhidas dez amostras de cada um dos cinco gestos a avaliar[11]. De modo a possibilitar a sua utilização pelos participantes das sessões, a janela de recolha de amostras gestuais encontra-se disponível no perfil de participante, ao contrário das restantes janelas da aplicação. Na Figura 21 é apresentada esta janela, após o utilizador desenhar um gesto alfa pedido pela aplicação, e antes de o aceitar e submeter para recolha.

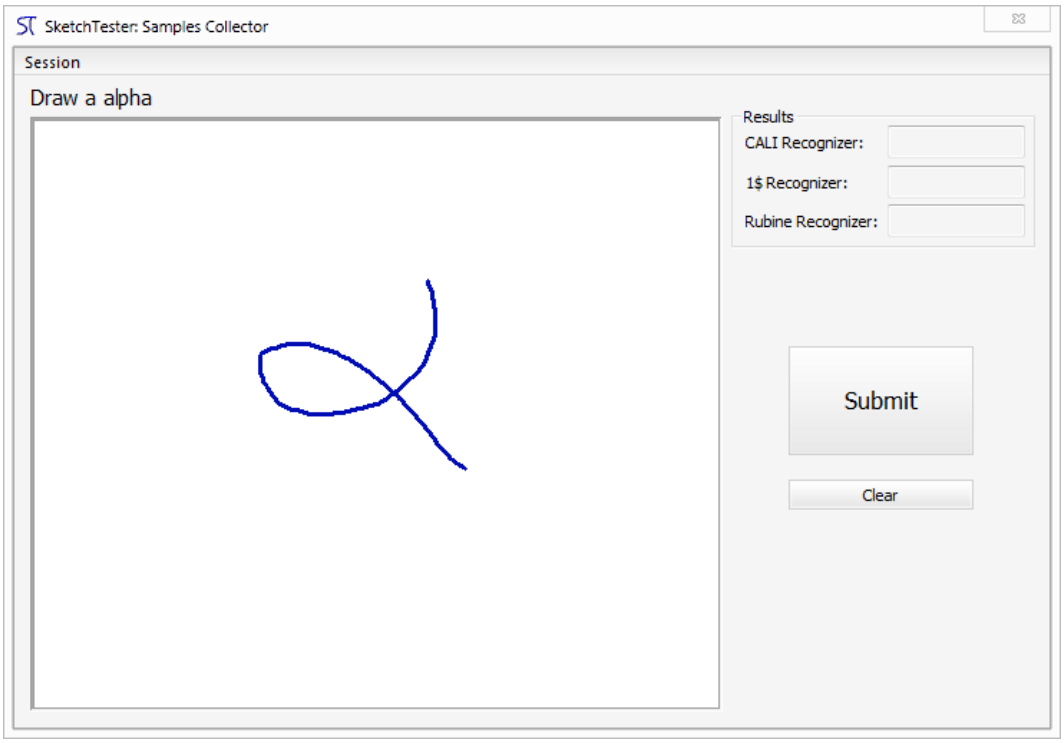



Figura 21 – Janela de recolha de amostras gestuais, após ser desenhado um gesto alfa

[11] Ver secção 3.4.

Após o desenho de cada gesto o utilizador pode aceitá-lo e proceder à sua submissão através do botão "Submit" ou, caso julgue que o gesto foi incorretamente desenhado, pode limpar a área de desenho de forma a realizar um novo gesto, utilizando para isso o botão "Clear". Sempre que um gesto é submetido, a aplicação exibe os resultados de reconhecimento do gesto por parte de cada um dos três reconhecedores implementados. Para além disso é indicado ao utilizador qual o próximo gesto a desenhar. A qualquer momento o utilizador pode reiniciar o processo de recolha, através de uma opção no menu da aplicação, voltando a fornecer todas as amostras necessárias.

A decisão de apresentação dos resultados de reconhecimento após a submissão de cada gesto foi tomada no sentido de dar algum *feedback* aos participantes das sessões de recolha de amostras. No entanto, para evitar que esta informação influencie o participante a desenhar um gesto com determinadas características após verificar que estas são corretamente reconhecidas, ou a evitar determinadas características após perceber que não estão a ser reconhecidas, a aplicação solicita cada gesto de forma aleatória, evitando assim que sejam solicitadas consecutivamente as dez amostras de uma mesma classe gestual.

No final do processo de recolha, após o utilizador ter submetido as cinquenta amostras necessárias, a aplicação exibe alguns resultados de reconhecimento (Figura 22) como o número de gestos corretamente reconhecidos por cada reconhecedor e a respetiva percentagem. Para além disso, todos os dados referentes ao processo de recolha são armazenados num ficheiro em disco, tal como descrito na secção 3.2.1, para possibilitar a sua posterior análise.

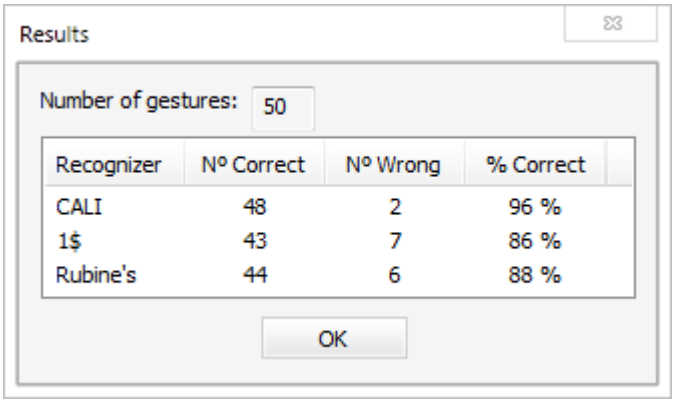



Figura 22 – Resultados de reconhecimento de amostras gestuais

No sentido de melhor ilustrar o processo de recolha de amostras gestuais, é apresentado na Figura 23 um fluxograma que descreve o fluxo de interações entre o utilizador e a janela de recolha de amostras gestuais da aplicação SketchTester.

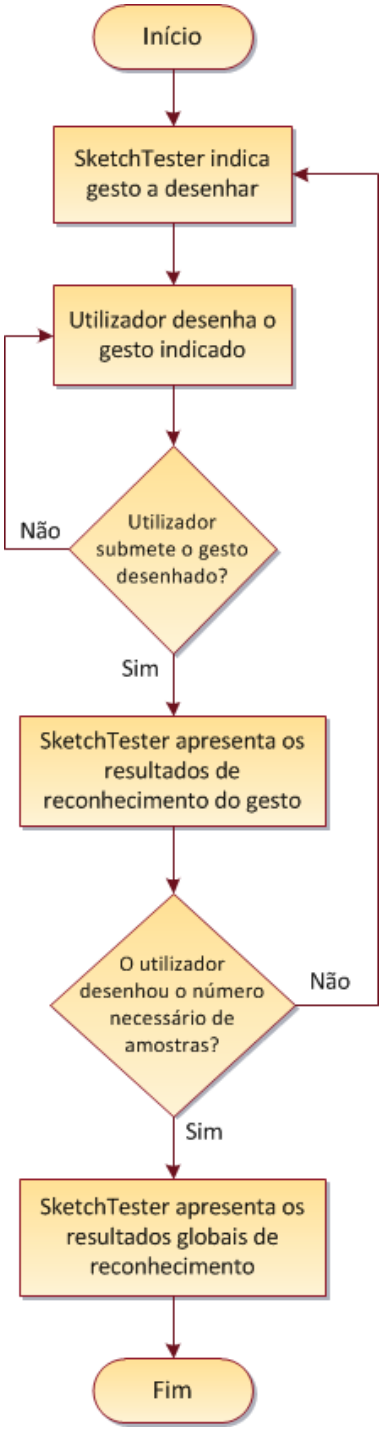



Figura 23 – Fluxograma do processo de recolha de amostras gestuais

Como será referido na secção 3.3, a implementação inicial dos reconhecedores caligráficos foi sujeita a várias alterações, no sentido de ser aperfeiçoada. Após cada alteração à implementação de um reconhecedor, foi necessário voltar a confrontar o reconhecedor (modificado) com as amostras gestuais recolhidas, com o intuito de validar as alterações. Dado que o ficheiro resultante do processo de recolha de amostras gestuais contém, para além do próprio gesto, os dados relativos ao seu reconhecimento, como a classe gestual

indicada ao utilizador e o reconhecimento feito por cada reconhecedor, foi necessário reprocessar cada um destes ficheiros, de forma a que os resultados de reconhecimento contidos fossem atualizados. Neste sentido, a janela de recolha de amostras gestuais possui ainda a capacidade de voltar a executar o processo de reconhecimento de cada reconhecedor sobre cada amostra armazenada. Esta operação pode ainda ser aplicada a vários ficheiros em simultâneo, permitindo atualizar todos os ficheiros recolhidos de uma só vez. De modo a facilitar a utilização da aplicação aos participantes das sessões de recolha de amostras, esta funcionalidade encontra-se disponível apenas no perfil de administrador.

### 3.2.4 Extração de Resultados de Reconhecimento das Amostras Gestuais

Após a recolha de amostras caligráficas foi necessário proceder à análise e estudo das mesmas. Com o objetivo de auxiliar esse processo, a aplicação SketchTester permite a visualização das amostras recolhidas e a extração de informação relativa ao seu reconhecimento, a partir da janela apresentada na Figura 24. Com esta janela é possibilitado o carregamento dos ficheiros de amostras recolhidas e a observação de cada uma individualmente, em conjunto com a informação de qual o gesto indicado ao utilizador no momento em que este desenhou a amostra e os respetivos resultados de reconhecimento. São exibidos ainda os resultados globais de reconhecimento das amostras contidas no ficheiro carregado, como o número total de gestos reconhecidos corretamente por cada reconhecedor e a respetiva percentagem.

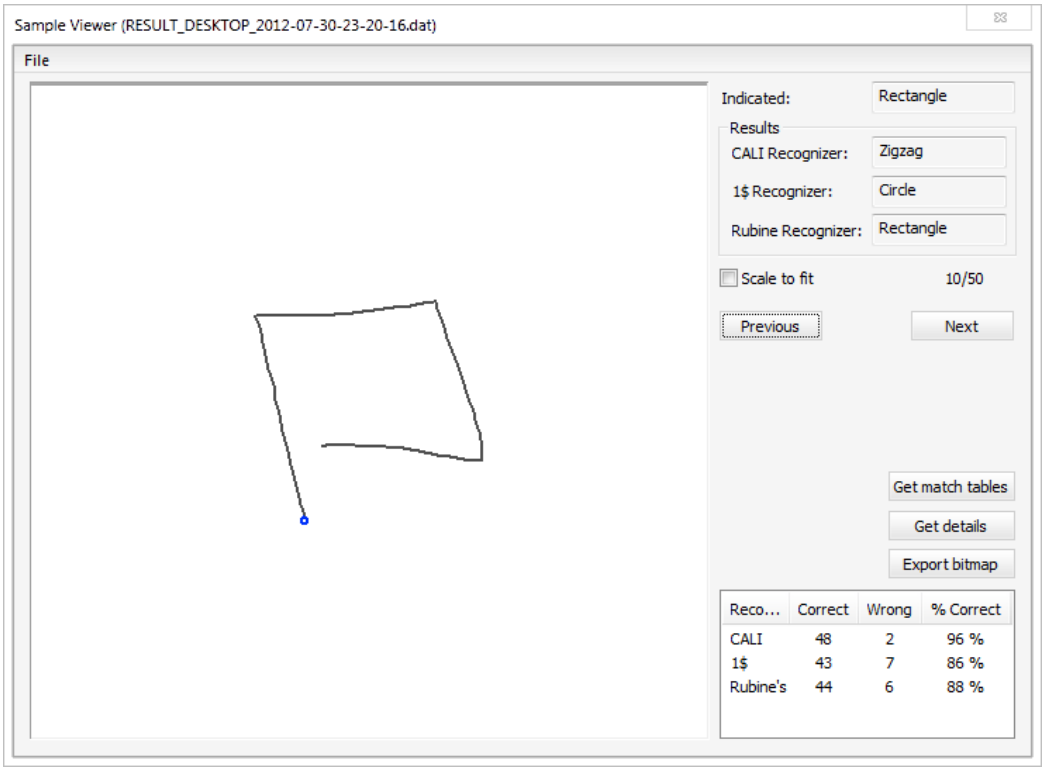



Figura 24 – Janela de visualização de amostras gestuais

Após a abertura de um ficheiro de amostras, a visualização das amostras contidas no mesmo é realizada com recurso a dois botões de navegação (“Previous” e “Next”), sendo exibido constantemente o índice da amostra atual e o número total de amostras contidas no ficheiro. Para além da representação da amostra desenhada pelo utilizador, é ainda assinalado o ponto inicial de desenho, através de um pequeno círculo azul. Existe ainda a opção de escalar a amostra para que esta assuma as dimensões da área disponível para exibição de amostras, através da caixa de marcação “Scale to fit”. Esta opção é especialmente útil para amostras de pequenas dimensões pois permite que, após serem aumentadas, se observem pormenores outrora impercetíveis.

Para além de visualizar as amostras individualmente na janela da aplicação, é também possível fazer a sua extração para um ficheiro *bitmap*. Neste ficheiro é possível observar todas as amostras lado a lado, bem como a identificação do gesto indicado ao utilizador e os resultados de reconhecimento, estando as falhas de reconhecimento assinaladas a vermelho. Isto facilita a identificação de padrões na forma de desenho de um utilizador bem como características adversas ao reconhecimento de um dado gesto. A extração de amostras para *bitmap* pode ainda ser executada para vários ficheiros de amostras em simultâneo, produzindo um *bitmap* por ficheiro. Na Figura 25 está representado um pequeno extrato de um *bitmap* de amostras.

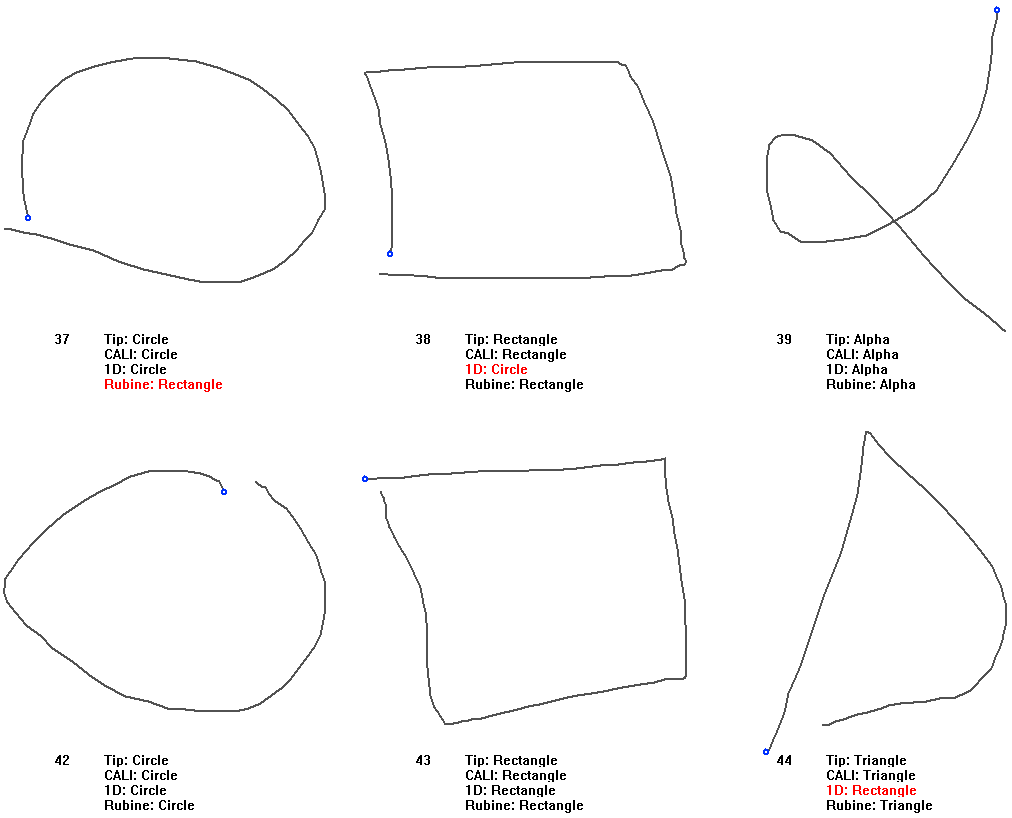



Figura 25 – Extrato de um *bitmap* de amostras caligráficas

Um passo importante na análise dos resultados de reconhecimento das amostras gestuais é determinar o número ou percentagem de gestos reconhecidos corretamente para cada classe gestual e cada reconhecedor. Isto permite averiguar quais as classes gestuais que possam estar com problemas de reconhecimento em cada um dos reconhecedores. Para auxiliar a realização desta tarefa, a janela de visualização de amostras procede à contagem automática desses valores e disponibiliza-os para análise. Na Tabela 2 são apresentados alguns resultados relativos ao reconhecedor CALI, extraídos de cinco ficheiros de amostras gestuais. As colunas "☺", "☹" e "% ☺" contêm o número de gestos reconhecidos corretamente, erradamente e a percentagem de sucesso, respetivamente.

Tabela 2 – Exemplo de resultados de reconhecimento distribuídos por gesto

| **CALI** | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Retângulo** | | | **Triângulo** | | | **Círculo** | | | **Alfa** | | | **Ziguezague** | | | |
| ☺ | ☹ | % ☺ | ☺ | ☹ | % ☺ | ☺ | ☹ | % ☺ | ☺ | ☹ | % ☺ | ☺ | ☹ | % ☺ | |
| 10 | 0 | 100 | 8 | 2 | 80 | 10 | 0 | 100 | 4 | 6 | 40 | 10 | 0 | 100 | |
| 8 | 2 | 80 | 9 | 1 | 90 | 9 | 1 | 90 | 10 | 0 | 100 | 2 | 8 | 20 | |
| 5 | 0 | 100 | 5 | 0 | 100 | 7 | 0 | 100 | 5 | 1 | 83 | 6 | 1 | 86 | |
| 10 | 0 | 100 | 9 | 1 | 90 | 9 | 1 | 90 | 3 | 7 | 30 | 8 | 2 | 80 | |
| 10 | 0 | 100 | 10 | 0 | 100 | 10 | 0 | 100 | 5 | 5 | 50 | 2 | 8 | 20 | |

Outra informação relevante para determinar possíveis causas de erros de reconhecimento entre classes gestuais são as tabelas de colisões. Como é exemplificado na Tabela 3, estas permitem visualizar como cada uma das classes gestuais foi reconhecida. Por exemplo, é possível identificar a percentagem de retângulos reconhecidos corretamente (como retângulos), erradamente (como triângulos, círculos, alfas ou ziguezagues) ou não reconhecidos de todo. No sentido de automatizar e simplificar esta tarefa, a janela de visualização de amostras gestuais permite a criação destas tabelas a partir dos ficheiros de amostras. Desta forma é possível observar rapidamente se o reconhecimento de uma dada classe gestual está a ser afetado por outra.

Tabela 3 – Exemplo de tabela de colisões entre classes gestuais

| **CALI** | **Retângulo** | **Triângulo** | **Círculo** | **Alfa** | **Ziguezague** | **Nenhum** |
|---|---|---|---|---|---|---|
| **Retângulo** | **97,0%** | 0,0% | 0,0% | 0,4% | 1,9% | 0,8% |
| **Triângulo** | 0,0% | **98,1%** | 0,0% | 0,4% | 0,4% | 1,2% |
| **Círculo** | 0,8% | 0,0% | **97,3%** | 0,0% | 0,4% | 1,5% |
| **Alfa** | 6,8% | 0,0% | 0,4% | **84,4%** | 0,0% | 8,4% |
| **Ziguezague** | 3,5% | 0,4% | 0,4% | 0,0% | **91,6%** | 4,0% |

Tal como acontece para a criação de *bitmaps* de amostras, a contagem de gestos reconhecidos corretamente e erradamente por classe gestual (Tabela 2) e a criação de tabelas de colisão (Tabela 3) podem ser realizadas para vários ficheiros numa única operação. Assim, é possível extrair rapidamente dados de todos os ficheiros resultantes das sessões de recolha de amostras, sem a necessidade de abrir e examinar cada ficheiro individualmente.

## 3.3 Características da Implementação de Reconhecedores Caligráficos

Após o desenvolvimento da aplicação SketchTester, procedeu-se à implementação e integração dos três reconhecedores caligráficos escolhidos para avaliação: o reconhecedor de Rubine, o reconhecedor $1 e o CALI. No que respeita à implementação do reconhecedor de Rubine e do reconhecedor $1, estas foram realizadas de acordo com os algoritmos apresentados pelos autores nos artigos respetivos (ver secções 2.1.1 e 2.1.3). No caso do reconhecedor CALI, dado tratar-se de uma biblioteca de programação, apenas foi necessário proceder à sua integração no sistema SketchTester. Em nenhuma das implementações foi considerada qualquer funcionalidade de rejeição de resultados de reconhecimento, dado que se pretende sempre obter um resultado, mesmo que este possua um grau de incerteza elevado.

Em relação aos *templates* de treino, o reconhecedor $1 foi treinado inicialmente com dois *templates* para cada uma das classes gestuais a avaliar[12], enquanto que o reconhecedor de Rubine foi treinado com quinze amostras gestuais por classe, valor apontado como adequado pelo seu autor. Dado que ambos os reconhecedores são sensíveis à direção de desenho dos gestos, sempre que é adicionado um novo *template*, a aplicação SketchTester procede à criação de uma cópia do mesmo, mas com a direção de desenho invertida. No caso do reconhecedor $1, esta cópia é então adicionada à mesma classe gestual do gesto original ou, no caso do reconhecedor de Rubine, a uma classe gestual específica de gestos invertidos[13]. Desta forma, é possível reconhecer um gesto independentemente da direção de desenho, tornando assim os reconhecedores insensíveis a esta característica. A necessidade de criação de novas classes gestuais para conter os *templates* invertidos, no reconhecedor de Rubine, deve-se ao cálculo da média das características geométricas dos *templates* ser pouco útil caso na mesma classe existam *templates* com características demasiado distintas, como direções de desenho inversas.

Dado tratar-se de um reconhecedor sem capacidades de treino, cujas classes gestuais estão codificadas no próprio reconhecedor, o CALI sofreu algumas alterações no seu código fonte para se ajustar às necessidades previstas para o sistema a desenvolver. A primeira alteração realizada foi a desativação do reconhecimento dos gestos do reportório do CALI que não

[12] Ver secção 3.4.

[13] Para cada classe gestual definida é criada automaticamente uma classe gestual "cópia" para conter os gestos com direções de desenho invertidas.

fazem parte do conjunto de gestos utilizados neste trabalho. Para tal, apenas foi necessário remover as classes de código fonte correspondentes, retirando-as da lista de gestos reconhecidos. Desta forma, o reconhecimento de gestos é orientado para as classes gestuais realmente utilizadas.

Após a desativação do reconhecimento das classes gestuais não necessárias existentes no CALI, foi executada a implementação do reconhecimento do gesto alfa. Com esse propósito, foi criada uma classe de código fonte na biblioteca do CALI, na qual foram definidas duas características geométricas para identificar o novo gesto, selecionadas com base em observação empírica: o rácio entre a área do quadrilátero de maiores dimensões[14] e a área do polígono convexo do gesto ($A_{lq}/A_{ch}$); e o rácio entre o perímetro do quadrilátero de maiores dimensões e o perímetro do polígono convexo ($P_{lq}/P_{ch}$). Os conjuntos difusos associados a estas características geométricas, utilizados para determinar o grau de enquadramento de um gesto a reconhecer à classe gestual do alfa, são apresentados na Figura 26.

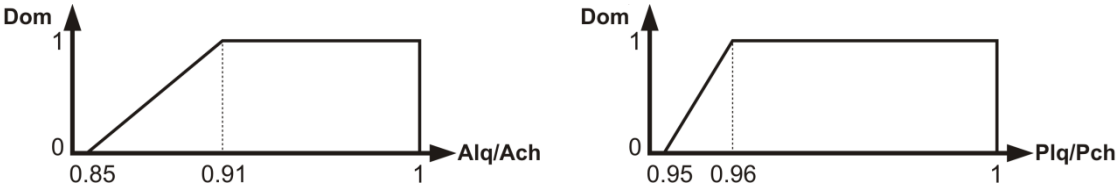



Figura 26 – Conjuntos difusos utilizados pelo reconhecedor CALI para identificar o gesto alfa

Após a definição destas duas características, verificou-se que as mesmas não eram suficientes para a correta identificação do gesto alfa, pois este gesto era frequentemente identificado pelo CALI como sendo um ziguezague e, ao mesmo tempo, um alfa[15]. Para evitar este problema, a definição do gesto ziguezague foi alterada, passando a conter uma característica geométrica local que determina que este não pode conter interseções no traço que o compõe. Para além disso, foi também adicionada na classe do gesto alfa uma característica local na qual é definido que este gesto tem obrigatoriamente que conter uma interseção situada a mais de 10% das suas extremidades, ou seja, cada "cauda" do alfa tem de compor mais de 10% do comprimento total do gesto.

Uma última característica importante na integração do reconhecedor CALI no sistema SketchTester prende-se com o facto de não ser pretendida a diferenciação entre círculos e elipses, pois ambos devem ser reconhecidos como círculos. Deste modo, independentemente do CALI reconhecer um gesto como círculo ou elipse, o sistema SketchTester classifica-o sempre como círculo. A mesma lógica é aplicada para retângulos e losangos, sendo ambos classificados como retângulos. Ao agrupar classes gestuais semelhantes é esperado um aumento das taxas de reconhecimento, dado que o utilizador pode desenhá-los com menos restrições.

[14] Trata-se do quadrilátero de maiores dimensões inscrito no polígono convexo do gesto, como descrito na secção 2.1.2.

[15] Como já foi referido, o CALI pode, em caso de ambiguidade, indicar mais do que um resultado para o reconhecimento de um gesto.

Após a análise das amostras gestuais recolhidas e dos respetivos resultados de reconhecimento, foi possível concluir que a implementação inicial dos reconhecedores caligráficos carecia de algumas melhorias. Por exemplo, de modo a tornar o reconhecimento de gestos pelo reconhecedor de Rubine menos sensível à orientação do gesto desenhado, foram criadas três novas classes gestuais por cada uma das classes existentes. Em cada uma destas novas classes foram adicionadas cópias dos *templates* existentes nas classes originais, mas representados com orientações distintas. A criação de novas classes gestuais é necessária pois, tal como no caso dos *templates* com direções de desenho invertidas, também a inclusão numa mesma classe de vários *templates* com grandes variações na sua orientação iria resultar numa média de características geométricas ineficiente para o reconhecimento. A Figura 27 ilustra como um retângulo se encontraria representado em cada uma das quatro classes gestuais, sendo que antes da inclusão das novas classes apenas a versão apresentada no canto superior esquerdo da figura estaria presente nos *templates* do reconhecedor de Rubine.

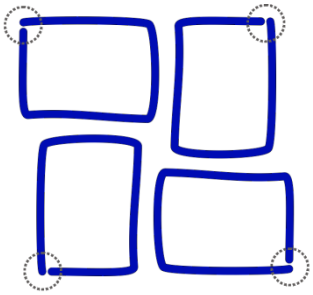


Figura 27 – Representação das quatro classes existentes no algoritmo de Rubine para a representação do gesto retângulo em quatro orientações distintas

Também o reconhecedor $1 sofreu pequenas melhorias ao nível dos seus *templates*. Uma destas melhorias consistiu na inclusão de *templates* nas classes gestuais retângulo e triângulo cujo ponto inicial de desenho se encontra a meio de arestas do gesto, visto que inicialmente se encontrava apenas nos vértices do gesto. Ainda para estas duas classes gestuais, foram também adicionados *templates* cujo ponto inicial se situa em vértices distintos, de forma a contemplar todas as possibilidades de desenho dos mesmos. Como o sistema SketchTester cria automaticamente cópias de cada *template* com uma direção de desenho invertida e o reconhecedor $1 é insensível à orientação dos gestos, não foi necessário proceder à criação de *templates* com pontos iniciais de desenho em cada um dos vértices. Estas melhorias não se aplicam aos restantes gestos, visto que ou não possuem vértices (no caso do círculo), ou têm pontos iniciais de desenho bem definidos (no caso do alfa e do ziguezague).

Por fim, também o CALI foi alvo de melhorias, nomeadamente sobre as modificações realizadas à implementação original do reconhecedor para o suporte do gesto alfa. Ao invés do valor inicial de 10%, foi definido que cada uma das "caudas" do gesto alfa deveria constituir pelo menos 7% do total do gesto. Para além disso, foi introduzida uma nova característica geométrica local no gesto alfa que especifica que a distância (ao longo do gesto) entre os dois pontos que se intersetam necessita de ser superior a 20% do comprimento total do gesto. Por fim, as características locais do gesto ziguezague foram também alteradas no

sentido de permitir que este possua interseções, desde que a distância ao longo do gesto entre os dois pontos que constituem a interseção seja inferior a 13% do comprimento total do gesto.

A análise racional subjacente às melhorias descritas será efetuada na secção seguinte, onde serão também apresentados dados que as validam.

## 3.4 Avaliação de Sistemas de Reconhecimento Caligráfico

Primeiramente, e antes do desenvolvimento da biblioteca SketchyDynamics, descrito no capítulo 4, foram avaliados três reconhecedores caligráficos: o CALI, o reconhecedor de Rubine e o reconhecedor $1; com o propósito de determinar aquele que melhor se adequa às características e especificidades da biblioteca. Para a concretização desta avaliação foram recolhidas 1550 amostras gestuais, realizadas por 32 participantes, a partir das quais foram realizados diversos testes de reconhecimento.

### 3.4.1 Método de Recolha de Dados

Para recolha de amostras gestuais foram conduzidas duas sessões, cada uma com 16 participantes. Durante as sessões os participantes foram instruídos para desenharem 10 amostras de cada um dos gestos ilustrados na Figura 28, resultando num total de 50 amostras por participante. No entanto, dado que dois participantes não completaram o processo e facultaram um número incompleto de amostras, não foram recolhidas as 1600 amostras previstas, mas sim 1550.

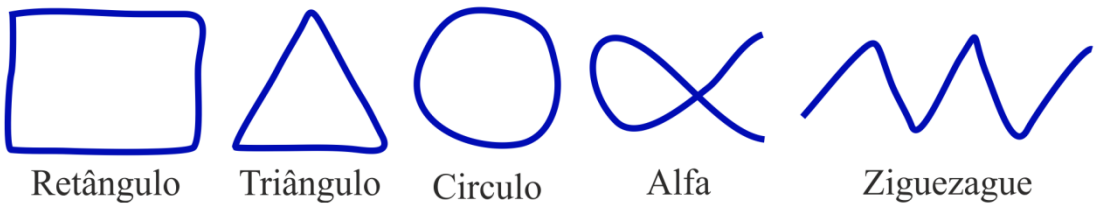



Figura 28 – Gestos a produzir pelos participantes das sessões de recolha de amostras gestuais

O estudo realizado teve como objetivo a avaliação do reconhecimento de gestos de traço único desenhados livremente. Por esta razão foi pedido aos participantes que desenhassem cada gesto sem qualquer restrição, diversificando as suas dimensões, orientações e formas.

Para agilizar o processo de recolha de amostras, este foi feito com recurso à aplicação SketchTester, distribuída pelos participantes juntamente com um guião[16] no qual se disponibilizou informações acerca do âmbito e objetivo da sessão de recolha de amostras e instruções de como proceder para a sua concretização.

[16] Ver Anexo A.

As sessões foram realizadas no Departamento de Engenharia Informática do Instituto Superior de Engenharia do Porto (ISEP), tendo 9 participantes recorrido aos computadores do instituto, que possuem rato tradicional. Os restantes 23 participantes utilizaram os seus portáteis pessoais, dos quais cerca de 25% recorreram ao painel tátil constituinte dos portáteis, tendo os restantes utilizado ratos tradicionais. Todos os participantes envolvidos eram estudantes de Mestrado.

### 3.4.2 Análise de Resultados

Após uma primeira análise aos resultados de reconhecimento das amostras gestuais recolhidas, foram realizadas diversas melhorias aos algoritmos de reconhecimento caligráfico, de modo a ampliar a sua eficácia. De seguida, as amostras gestuais foram novamente processadas pelos reconhecedores e os novos resultados de reconhecimento analisados, com o objetivo de confirmar e validar estas melhorias. No Anexo B encontram-se descriminados, sob a forma de tabelas, os resultados de reconhecimento das amostras de cada um dos participantes, com cada reconhecedor, em cada uma das fazes deste estudo, tendo servido de base para a sua concretização.

O primeiro passo após a obtenção das amostras gestuais consistiu na realização de uma filtragem das mesmas, no sentido de garantir que se adequavam ao domínio no qual os reconhecedores serão utilizados. Esta filtragem traduziu-se na remoção de amostras que não correspondiam ao gesto pedido (e.g., foi pedido ao participante para desenhar um retângulo; contudo este desenhou um círculo) e amostras cuja forma não representava nenhum dos cinco gestos definidos. Como previsto, após a filtragem as taxas de reconhecimentos bem-sucedidos dos três reconhecedores revelaram-se superiores, particularmente no CALI cuja taxa sofreu uma melhoria de 11%. Na Figura 29 é apresentada uma comparação lado-a-lado das taxas de reconhecimento de cada um dos reconhecedores, antes e depois da filtragem, na qual é possível verificar as melhorias obtidas.

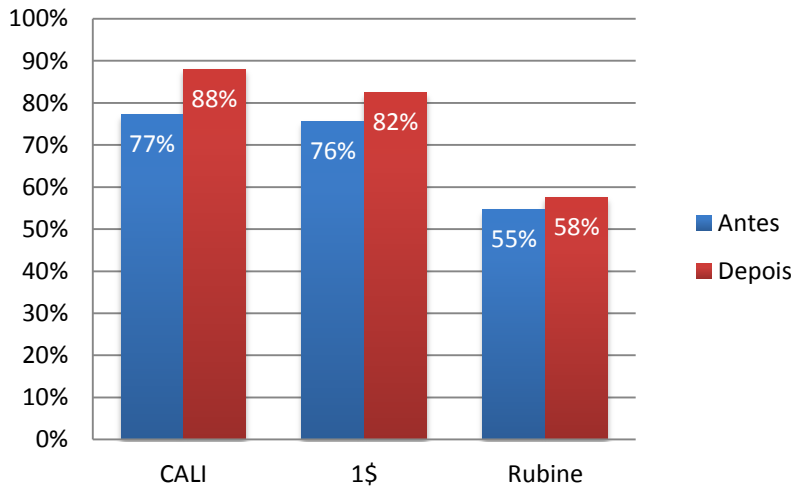



Figura 29 – Taxas de reconhecimento antes e depois da filtragem de amostras gestuais

Pode observar-se que o reconhecedor de Rubine obteve uma taxa de reconhecimento substancialmente reduzida, na ordem dos 58%, mesmo após a filtragem de amostras. Neste sentido, é necessário examinar os dados individuais de reconhecimento de cada uma das classes gestuais definidas, para descobrir se a taxa se encontra a ser afetada apenas por algumas classes ou se existe alguma dificuldade generalizada no processo de reconhecimento. No gráfico da Figura 30, que apresenta as taxas de reconhecimento obtidas por cada reconhecedor em cada um dos gestos, é possível observar que o reconhecedor de Rubine obteve taxas de reconhecimento de 26% e 27% para os gestos triângulo e círculo, respetivamente. Estas taxas, para além de representarem valores bastante baixos, situam-se distantes das taxas de reconhecimento dos restantes gestos.

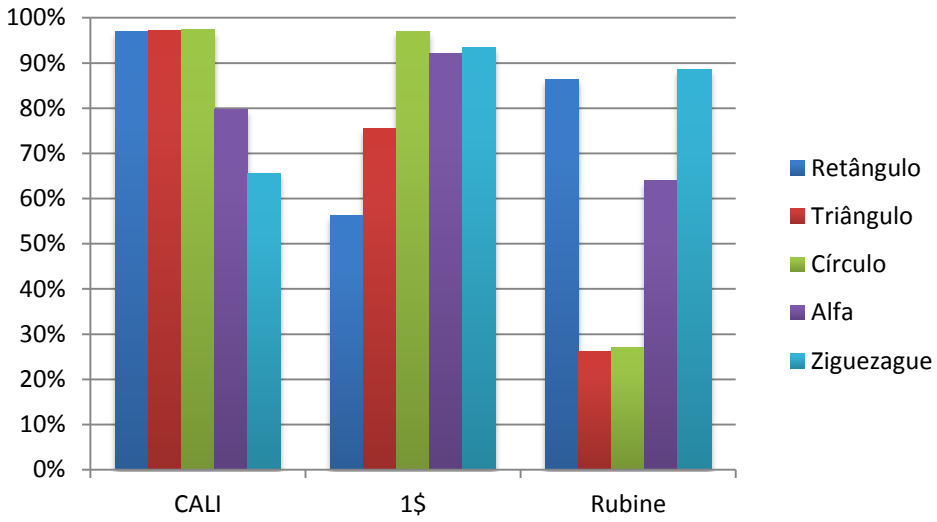



Figura 30 – Taxas individuais de reconhecimento de cada gesto

Posteriormente a uma análise pormenorizada às amostras gestuais, nomeadamente àquelas referentes a triângulos e círculos que não se verificavam corretamente reconhecidas pelo reconhecedor de Rubine, foi notório que na origem dos problemas de reconhecimento desses dois gestos estava a diversidade de orientações de desenho dos mesmos. Como o reconhecedor de Rubine é sensível à orientação com a qual os gestos são desenhados, estes são corretamente reconhecidos unicamente quando a sua orientação corresponde àquela definida pelos *templates*. É importante notar que, apesar de um círculo manter a mesma apresentação independentemente de se confirmar ou não rodado, o seu desenho pode ser iniciado de diferentes pontos, pelo que a variação de orientação deste gesto se refere a esta característica.

A análise individual das várias amostras gestuais permitiu ainda identificar a existência de uma tendência dos participantes para iniciar o desenho de retângulos pelo vértice superior esquerdo, característica também presente nos *templates* de treino do reconhecedor de Rubine e que explica o porquê do reconhecimento desse gesto não ser significativamente afetado pela sensibilidade à orientação do reconhecedor.

Uma outra descoberta interessante foi que em relação ao gesto ziguezague, a orientação de desenho causa um impacto pouco expressivo no seu reconhecimento, sendo bem-sucedido nas mais diversas orientações. Isto deve-se ao facto deste gesto possuir características bastante distintivas em relação aos restantes, e também pela não implementação de qualquer mecanismo de rejeição de gestos, o que faz com que o gesto seja reconhecido mesmo que com um grau de incerteza elevado.

De forma a alcançar um aumento da taxa de reconhecimento do algoritmo de Rubine, foi necessário ultrapassar o problema inerente à sua sensibilidade à rotação. Como descrito anteriormente, a solução passou pela adição de novos *templates* de treino ao reconhecedor que contemplem orientações diversificadas dos gestos. Com esta alteração à coleção de *templates* do reconhecedor de Rubine foi possível obter um aumento bastante significativo das taxas de reconhecimento dos gestos, como é possível observar na Figura 31. Desta forma, apesar de uma ligeira descida na taxa do gesto retângulo, a taxa de reconhecimento global do reconhecedor de Rubine subiu para 79%, em oposição aos 58% obtidos anteriormente.

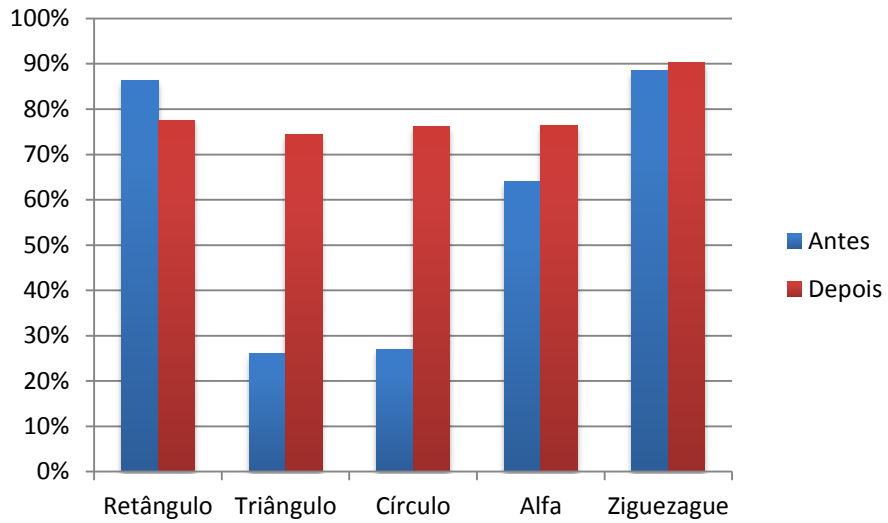



Figura 31 – Taxas de reconhecimento de cada gesto com o reconhecedor de Rubine, antes e depois da inclusão de *templates* de treino com orientações diversificadas

No que respeita ao reconhecedor de $1, como é possível observar pela Figura 30, os gestos mais problemáticos foram o retângulo e o triângulo, com taxas de 56% e 75%, respetivamente. Após análise às amostras gestuais obtidas foi evidente que estes dois gestos necessitavam de *templates* de treino cujo ponto inicial se situasse em diferentes vértices, bem como no centro de arestas, dado que os *templates* de treino existentes inicialmente não usufruíam destas variações. Além disso, também o reconhecimento de triângulos retângulos se verificou dificultado, pelo que foi necessária a criação de *templates* deste tipo de triângulo. Estas alterações possibilitaram um aumento das taxas de reconhecimento de retângulos e triângulos, como é visível na Figura 32, causando um acréscimo da taxa global de reconhecimento do reconhecedor $1 de 82% para 87%. No entanto, o reconhecimento de círculos e alfas revelou-se prejudicado, refletindo-se na descida das taxas referentes a esses

gestos. Isto resulta de um aumento de 1,5% para 12,5% no número de círculos reconhecidos incorretamente como retângulos, e de 6,8% para 12,2% no número de alfas reconhecidos como triângulos. Apesar de terem sido realizadas várias tentativas para contrariar a descida na taxa de reconhecimento destes dois gestos, não foi possível encontrar até ao momento qualquer solução.

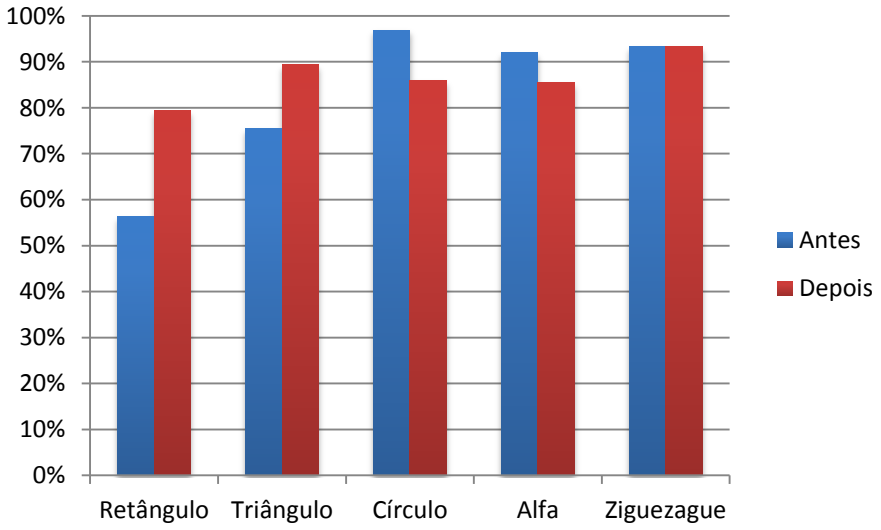



Figura 32 – Taxas de reconhecimento de cada gesto com o reconhecedor $1, antes e depois da melhoria dos seus *templates* de treino

Quanto à taxa de reconhecimento relativamente elevada obtida pelo reconhecedor CALI, que se situa na ordem dos 88%, uma análise posterior aos resultados individuais de cada gesto sugeria a possibilidade de melhorias. Como é referenciado na Figura 30, apesar de as taxas de reconhecimento de retângulos, triângulos e círculos se situarem nos 97% para os três gestos, o mesmo não acontece para alfas e ziguezagues, que possuem taxas de 80% e 66%, respetivamente. Estes dados sugerem que o processo de reconhecimento do CALI possa estar a ser negativamente afetado pelas alterações introduzidas neste trabalho com o objetivo de possibilitar o reconhecimento do gesto alfa (secção 3.3), pois estas foram realizadas precisamente nas classes gestuais que apresentam problemas de reconhecimento. Após uma investigação às amostras gestuais recolhidas foi possível identificar que os alfas cujas "caudas" (os segmentos de linha entre o ponto de interseção e as extremidades do gesto) fossem curtas não eram corretamente reconhecidos. Este problema resulta de uma característica geométrica local definida para o gesto alfa, na qual é especificado que o comprimento de cada uma das "caudas" do gesto deve ser superior a 10% do comprimento total do gesto. Após diversos testes foi possível concluir que este valor deveria ser reduzido para 7%, de modo a possibilitar obter uma melhor taxa de reconhecimento deste gesto sem afetar os restantes.

Também a análise das amostras gestuais referentes ao gesto ziguezague permitiu uma melhoria no seu reconhecimento. Esta análise revelou que, ao contrário do que foi previsto inicialmente, existiam vários utilizadores cujos ziguezagues desenhados continham pequenas interseções, resultantes de imprecisões naturais decorrentes do desenho rápido à mão. Dado

que foi inicialmente definido que este gesto não poderia conter interseções, um grande número de ziguezagues não estava a ser corretamente reconhecido. A solução passou por permitir a existência de interseções no gesto ziguezague, mas apenas se a distância entre os dois pontos que se intersetam for inferior a 20% do comprimento total do gesto, valor obtido a partir do estudo das amostras gestuais recolhidas. Consequentemente, para evitar que gestos ziguezague fossem reconhecidos como alfas, também a classe gestual do alfa foi atualizada para garantir que a distância entre os dois pontos intersetados é superior a 20% do tamanho total do gesto. Na Figura 33 é apresentada uma comparação lado-a-lado dos resultados obtidos pelo CALI antes e depois destas melhorias, e a partir dos quais é percetível o impacto positivo das mesmas no reconhecimento de alfas e ziguezagues, sem qualquer dano apresentado no reconhecimento dos restantes gestos.

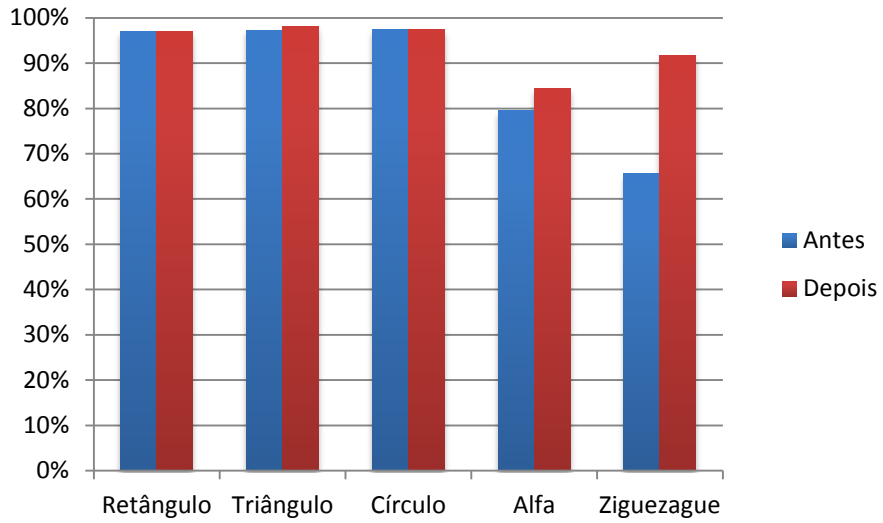



Figura 33 – Taxas de reconhecimento de cada gesto com o reconhecedor CALI, antes e depois das melhorias das características geométricas referentes aos gestos alfa e ziguezague

Tal como foi referido, com base no estudo dos resultados de reconhecimento das amostras gestuais recolhidas, foi realizado um conjunto de melhorias à implementação inicial dos reconhecedores caligráficos. Para além dos dados apresentados anteriormente, o sucesso destas melhorias pode ainda ser validado através da comparação lado-a-lado das taxas de reconhecimento globais obtidas por cada um dos reconhecedores (Figura 34). É possível observar que o desvio-padrão associado às taxas de reconhecimento de cada reconhecedor foi substancialmente reduzido, em resultado de um equilíbrio das taxas referentes a cada um dos cinco gestos. Contudo, os valores de desvio-padrão apresentados são expressivos, pois continua a existir alguma disparidade nas taxas de reconhecimento de diferentes gestos. Isto sugere que as implementações dos reconhecedores poderiam ser sujeitas a mais aprimoramentos, sobretudo naquelas classes gestuais cujas taxas se mostram inferiores.

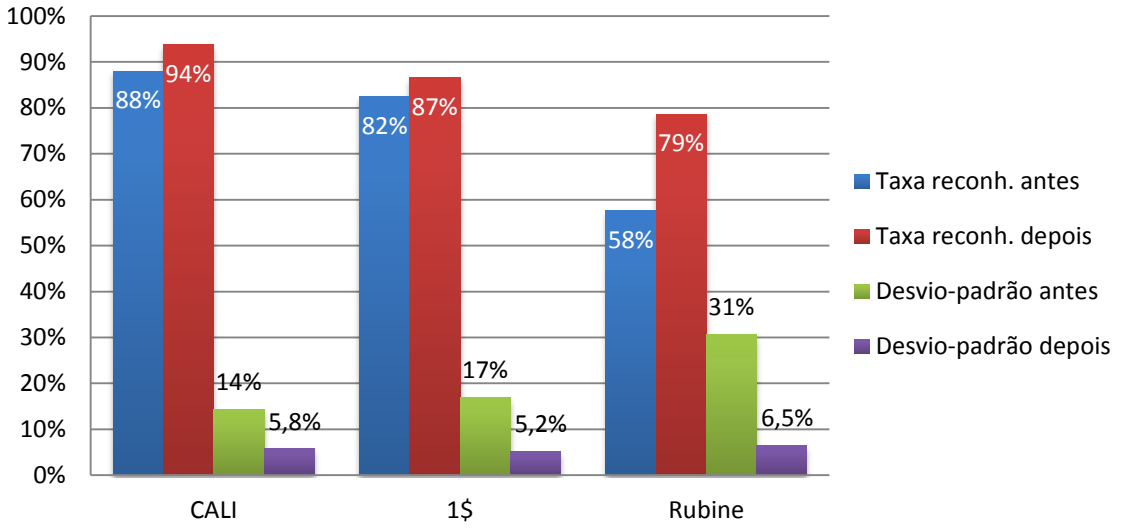



Figura 34 – Taxas de reconhecimento globais e respetivos desvios-padrão, antes e depois das melhorias realizadas aos reconhecedores

### 3.4.3 Comparação de Resultados

Com o intuito de demonstrar o potencial da sua ferramenta de avaliação automática de reconhecedores caligráficos (secção 2.1.10), Schmieder *et al.* [18] conduziram duas experiências na qual avaliaram seis reconhecedores caligráficos, e nos quais se incluem aqueles cuja avaliação foi aqui apresentada. A primeira experiência consistiu no reconhecimento de três gestos de traço único (círculo, retângulo e linha), sendo a segunda orientada ao reconhecimento de diagramas Entidade-Relação. Apesar do âmbito da segunda experiência ser bastante diferente daquele relatado neste documento, a primeira constitui um bom exemplar para confrontação de resultados.

Na primeira experiência realizada por Schmieder *et al.*, o reconhecedor de Rubine obteve a taxa de reconhecimento mais alta, na ordem dos 96%, seguido pelo reconhecedor $1 com 89% e, por fim, pelo CALI com 84%. Apesar de estes resultados aparentarem contradizer os apresentados neste capítulo, com o reconhecedor de Rubine e o CALI em posições invertidas, este facto pode ser facilmente explicado. No que se refere ao reconhecedor de Rubine, enquanto que a implementação utilizada para o estudo apresentado neste capítulo recorreu às características geométricas originais propostas por Rubine, Schmieder *et al.* implementaram um conjunto de características geométricas melhoradas, propostas por Plimmer *et al.* [5] e descritas na secção 2.1.1. A utilização de características geométricas melhoradas para a identificação de gestos permite, evidentemente, a obtenção de melhores resultados.

No que respeita ao reconhecedor CALI, é importante referir que Schmieder *et al.* consideraram o reconhecimento de círculos independente do de elipses, ao contrário da avaliação apresentada neste capítulo, na qual ambos os gestos são considerados como um só, pois não existiu a necessidade de os diferenciar. Contudo, e apesar da experiência realizada

não contemplar elipses, os autores apresentam os seus resultados de reconhecimento com o CALI. Esses resultados mostram que, de um total de 730 gestos avaliados, 94 círculos foram classificados como elipses. Se o reconhecimento de círculos e elipses fosse unificado, essas 94 amostras seriam consideradas como identificações bem-sucedidas, e a taxa de reconhecimento global do CALI ascenderia para os 96%. Se esta mesma lógica fosse também aplicada a retângulos e losangos, a taxa de reconhecimento do CALI seria de 98%. Desta forma é possível concluir que, se o CALI tivesse usufruído de uma configuração semelhante à apresentada neste documento, os resultados de reconhecimento constituir-se-iam também semelhantes.

É ainda relevante referir que quanto menor for o conjunto de gestos a reconhecer, melhores são geralmente as taxas de identificação obtidas, pois existe menos propensão para a ocorrência de conflitos ou colisões entre os vários gestos.

## 3.5 Conclusão

Através da análise dos resultados de reconhecimento das amostras gestuais recolhidas foi possível identificar falhas na concretização da implementação dos reconhecedores e proceder à sua retificação, o que permitiu deste modo obter resultados significativamente melhores. Porém, os resultados obtidos expuseram a possibilidade para posteriores melhorias, mas que por limitações temporais, não puderam ser exploradas no âmbito deste trabalho.

Um dos problemas verificados durante a realização desta avaliação foi a ocorrência de um decréscimo na eficiência de reconhecimento de círculos e alfas por parte do reconhecedor $1, após as melhorias direcionadas ao reconhecimento de retângulos e triângulos. Dado não ter sido possível encontrar uma solução para este problema, poderia ser conduzido posteriormente um novo estudo, que visasse não só resolver este problema, mas também encontrar novas formas de melhorar o reconhecimento.

Futuramente seria interessante realizar uma nova recolha de amostras com recurso a ecrãs táteis ou ecrãs interativos com estilete, pois este tipo de equipamento permite maximizar as vantagens da interação caligráfica. Como as melhorias aos reconhecedores foram realizadas e avaliadas com base no mesmo conjunto de amostras caligráficas, a recolha de novas amostras seria importante para garantir que essas melhorias são válidas não só para as amostras já recolhidas, mas para quaisquer outras.

Um estudo futuro poderia, para além do resultado principal de reconhecimento (aquele com o maior grau de certeza), considerar os primeiros dois ou até três resultados obtidos. Este estudo seria relevante sobretudo para aplicações que recorrem a esses resultados secundários para dispor uma lista de identificações alternativas, com o intuito de ajudar na resolução da ambiguidade.

Dado que o propósito da avaliação de reconhecedores caligráficos com recurso a amostras gestuais era determinar qual o que melhor se enquadraria na componente de interação da biblioteca a desenvolver, é finalmente possível afirmar que, de acordo com os resultados obtidos, tanto o CALI como o reconhecedor $1 são ótimos candidatos. Relativamente ao reconhecedor de Rubine, apesar de este ter apresentado um aumento na sua taxa de reconhecimento (após melhorias) bastante superior aos restantes, não foi o suficiente para se destacar. No entanto, caso fosse introduzido o conjunto de características geométricas melhoradas, apresentadas por Plimmer *et al.* [5], o reconhecedor de Rubine deveria ser capaz de obter taxas similares aos restantes reconhecedores. Para concluir, com base na avaliação realizada, foi escolhido o reconhecedor CALI para integrar a biblioteca a desenvolver, pois foi o que apresentou os melhores resultados entre os três reconhecedores considerados.

# 4 A Biblioteca SketchyDynamics

O propósito principal do trabalho apresentado neste documento é o desenvolvimento de um sistema que facilite a criação de aplicações de simulação de física, e em que a interação do utilizador com a simulação seja conseguida através de uma interface caligráfica. O resultado desse sistema é a biblioteca SketchyDynamics. Esta consiste numa biblioteca de programação C++, que permite a integração rápida e facilitada de um motor de simulação de física em perfeita sinergia com uma interface caligráfica que dispõe de todas as ações necessárias para controlar a simulação.

A decisão de recorrer a uma simulação e representação visual 2D deve-se à sua maior semelhança com o paradigma papel-lápis, pois o desenho do utilizador (que é conseguido em 2D) é interpretado e representado como um objeto também 2D. Isto viabiliza que o utilizador desenhe enquanto a simulação decorre, aparentando o desenho estar inserido no mesmo espaço da simulação.

Sem qualquer esforço por parte de um programador que recorra à biblioteca SketchyDynamics, esta disponibiliza uma janela para a apresentação dos elementos visuais, e uma interface caligráfica a partir da qual o utilizador pode interagir com a aplicação. Para além disso, encarrega-se do reconhecimento e interpretação das ações do utilizador como a criação, transformação e remoção de corpos rígidos e ligações entre eles. Também a gestão e controlo da simulação física desses objetos é efetuada nativamente pela biblioteca, em conjunto com a representação gráfica dessa simulação. Deste modo, a inclusão destas funcionalidades numa aplicação é agilizada.

Neste capítulo é realizada uma abordagem a todas as características da biblioteca SketchyDynamics, inicialmente no que respeita às suas funcionalidades perante um utilizador e posteriormente em termos das suas características de implementação e utilização por parte de um programador. Note-se que, neste contexto, existe uma distinção necessária entre utilizador e programador. Um utilizador é aquele que usufrui de uma aplicação desenvolvida com recurso à biblioteca SketchyDynamics, em oposição a um programador que a utiliza para desenvolver uma aplicação.

Dado que a biblioteca SketchyDynamics permite à aplicação que a integra definir o aspeto da maioria dos seus elementos gráficos, é importante referir desde já que a representação destes elementos apresentada em figuras desta secção, é meramente ilustrativa. Por exemplo, as cores e texturas utilizadas pelos corpos rígidos ou pelo fundo da janela da aplicação não estão limitadas às apresentadas aqui, podendo ser definidas livremente pelo programador de acordo com a temática da sua aplicação. Na Figura 35 é apresentada como exemplo uma aplicação que recorre a uma temática "caneta sobre papel", desenvolvida com recurso à biblioteca SketchyDynamics, na qual o utilizador desenhou uma cena composta por diversos objetos.

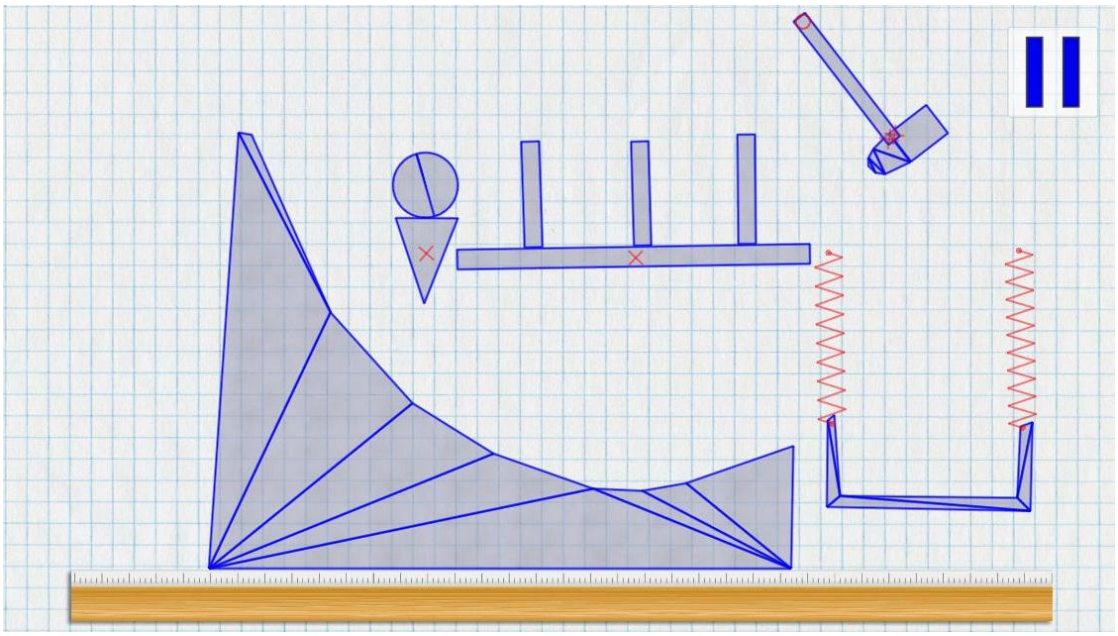


Figura 35 – Exemplo de aplicação desenvolvida com recurso à biblioteca SketchyDynamics

## 4.1 Metodologia de Desenvolvimento

Tal como sucedeu para o sistema SketchTester, também o desenvolvimento da biblioteca SketchyDynamics adotou a metodologia em estrela proposta por Hix e Hartson [26] (secção 3.1). Este caracterizou-se por constantes avaliações informais – a última das quais bastante extensa e rigorosa – e subsequentes melhorias de funcionalidades e dos requisitos definidos, com base no *feedback* obtido.

## 4.2 Funcionalidades e Interação

No que respeita às suas funcionalidades perante um utilizador, a biblioteca SketchyDynamics visa o desenho e criação de uma cena fisicamente simulada, composta por corpos rígidos que podem estar interligados por conectores. Para além da criação destes objetos, o utilizador deve ser ainda capaz de os transformar até certo ponto, podendo alterar a sua posição,

orientação e escala, bem como concretizar a sua remoção. Na Figura 36 encontram-se representados diversos tipos de corpos rígidos (a azul) suportados pela SketchyDynamics e diversos conectores entre eles (a vermelho).

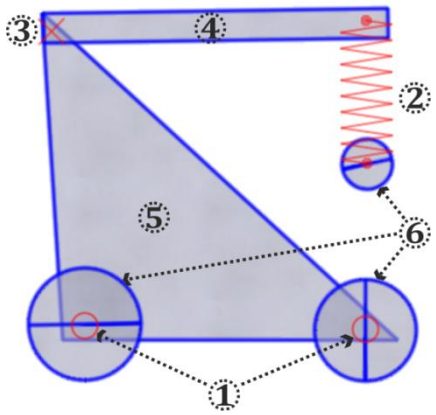



Figura 36 – Diversos tipos de corpos e conectores: 1) conectores de rotação; 2) conector de mola; 3) conector de âncora; 4) corpo retangular; 5) corpo triangular; 6) corpos circulares.

Dado que um dos princípios fundamentais da biblioteca SketchyDynamics é proporcionar uma interação natural, adotando o paradigma papel-lápis, esta foi desenhada de forma a tirar partido de sistemas munidos de ecrãs interativos com estiletes. Deste modo, para concretizar a interação, o utilizador apenas precisa de pressionar e mover o estilete sobre o ecrã, sem serem necessários quaisquer botões. Para além disso, não são utilizados menus na sua interface, e a maioria das ações de interação é realizada deslizando o estilete sobre o ecrã. Apesar de ser projetada com a utilização de estilete em mente, a biblioteca SketchyDynamics também funciona corretamente com sistemas que utilizam rato tradicional, nos quais é utilizado o clique esquerdo para substituir o pressionar do estilete.

### 4.2.1 Criação de Objetos

Como foi já referido, a biblioteca SketchyDynamics disponibiliza uma interface caligráfica, pelo que a criação de objetos, sejam eles corpos ou conectores, é conseguida através do seu desenho. Por exemplo, se o utilizador desejar criar um corpo retangular apenas tem que desenhar um retângulo no ecrã, com as dimensões desejadas e na posição e orientação pretendidas. A SketchyDynamics reconhece então o retângulo desenhado, em conjunto com as suas características geométricas como tamanho, orientação e posição, e procede à criação das representações física e gráfica[17] do mesmo.

Para a criação de corpos rígidos são suportados quatro tipos de gestos: retângulo, triângulo, círculo e forma livre. Após ser desenhado um gesto, o corpo resultante assume sempre a forma do gesto que o originou, com algumas dissemelhanças. Quando um gesto é

[17] A representação física de um objeto é aquela com a qual a sua simulação é conseguida, enquanto que a sua representação gráfica é a tradução da sua representação física num objeto visual no ecrã.

reconhecido como um retângulo, triângulo ou círculo, o corpo resultante é representado de forma embelezada, como exemplificado na Figura 37. Caso contrário, quando o gesto não é reconhecido, este é interpretado como uma forma livre e o corpo originado é representado como uma simplificação do gesto, com menos vértices e com as linhas curvas substituídas por linhas poligonais, por questões de performance. A interpretação como forma livre de um gesto não reconhecido concede ao utilizador a capacidade para criar diversos polígonos, mesmo que não façam parte dos três reconhecidos pela biblioteca SketchyDynamics.

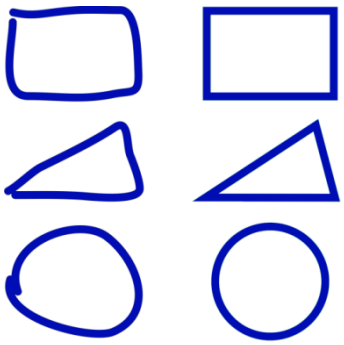


Figura 37 – Exemplos de formas desenhadas (à esquerda) e respetivas representações embelezadas (à direita)

Para além da criação de corpos é ainda possível interligá-los e construir objetos mais complexos, com recurso a três tipos de conectores: âncora, rotação e mola. Os conectores de âncora permitem fixar dois objetos através de um ponto comum, impedindo qualquer movimento relativo entre eles. Tal como os conectores de âncora, os de rotação unem dois corpos através de um ponto comum mas possibilitam que estes rodem livremente em torno desse ponto. Os conectores de mola utilizam um ponto em cada um dos objetos para os interligar e tentam manter uma distância constante entre eles. Com base na distância calculada no momento de criação do conector, este tipo de conector estica e encolhe conforme as forças externas como gravidade, colisões ou peso de outros objetos, são aplicadas aos objetos em causa, como se de uma mola real se tratasse. Para além disso, o conector de mola admite rotação em torno dos dois pontos que unem os objetos, permitindo desta forma a simulação de pêndulos. Em adição à criação de ligações entre corpos criados manualmente pelo utilizador, este pode também conectar corpos integrantes da aplicação, os quais podem ser estáticos ou dinâmicos. Isto permite que, por exemplo, o utilizador prenda objetos ao fundo da aplicação e restrinja assim os seus movimentos.

Tal como a criação de corpos, a criação de conectores é conseguida através do desenho de um gesto correspondente. Desenhar um gesto alfa sobre dois corpos cria um conector de âncora no ponto de interceção do gesto, enquanto que desenhar um círculo pequeno cria um conector de rotação cujo ponto de referência é o centro do círculo desenhado. Para criar conectores de mola, o utilizador deve desenhar um gesto ziguezague com início em um dos corpos e a findar no outro, definindo assim os pontos de ligação como o ponto inicial e final do gesto.

No que respeita à representação gráfica, os conectores de âncora e rotação são representadas por uma pequena cruz e um pequeno círculo, respetivamente, centrados no ponto que define a conexão. Já o conector de mola é representado por uma linha ziguezague cujo início e fim se situam nos dois pontos de ligação localizados nos corpos que une. Ao contrário dos conectores de âncora e rotação, que possuem formas estáticas e constantes, o conector de mola possui uma representação dinâmica, que estica e encolhe de acordo com a distância entre os seus objetos. Para além disso, a representação gráfica do conector de mola pode ser constituída por um número variável de vértices, dependente da distância entre os corpos que liga no momento da sua criação. O objeto apresentado na Figura 36 foi construído com recurso aos três tipos de conectores, pelo que é possível observar as suas representações gráficas.

Considerando o objetivo de melhor lidar com a ambiguidade inerente ao desenho à mão, sempre que o utilizador desenha um gesto é apresentada uma lista de expetativas, que é ocultada automaticamente quando selecionada uma das suas opções, ou quando o utilizador inicia uma nova ação ou desenho. Nesta lista são apresentadas todas as interpretações possíveis de acordo com o contexto do gesto desenhado, com o intuito de permitir ao utilizador indicar um objeto diferente daquele que foi reconhecido pela aplicação. As identificações do gesto sugeridas pelo reconhecedor caligráfico, ou seja, os seus resultados de reconhecimento, surgem no início da lista, ordenados pelo seu grau de certeza e com uma dimensão superior aos restantes, pois possuem uma maior probabilidade de representarem a intenção do utilizador. Existem ainda outras regras de contexto que causam o aparecimento de uma hipótese no início da lista mesmo que não tenha sido indicada pelo reconhecedor. Por exemplo, se o utilizador desenhar um retângulo de pequenas dimensões sobre dois corpos, para além de ser sugerida a criação de um corpo retangular, é sugerida como segunda hipótese a colocação de um conector de rotação. Esta regra foi criada após ser verificado que a tentativa de criação de conectores de rotação por parte do utilizador era frequentemente interpretada como um retângulo pequeno. Este problema ocorre principalmente em sistemas cuja taxa de amostragem é reduzida ou quando o gesto é desenhado rapidamente, pois resultam no desenho de um retângulo em vez de um círculo pequeno. Enquanto que as possibilidades sugeridas pelo reconhecedor caligráfico ou resultantes de regras de contexto surgem ordenados pelo seu grau de certeza de reconhecimento, as restantes são exibidas com uma ordenação fixa e pré-definida. Desta forma, é possível ao utilizador memorizar essa ordem e facilmente encontrar o objeto desejado. Como última opção da lista de expetativas é apresentada sempre a opção para anular o gesto desenhado, o que remove qualquer corpo ou conector criado em resultado deste. Na Figura 38 é apresentada uma lista de expetativas, no qual o sistema interpretou o gesto do utilizador como um triângulo. Apesar de a lista apresentada nesta figura ser constituída por um conjunto de etiquetas *post-it*, qualquer aplicação que recorra à biblioteca SketchyDynamics pode definir livremente o seu aspeto.

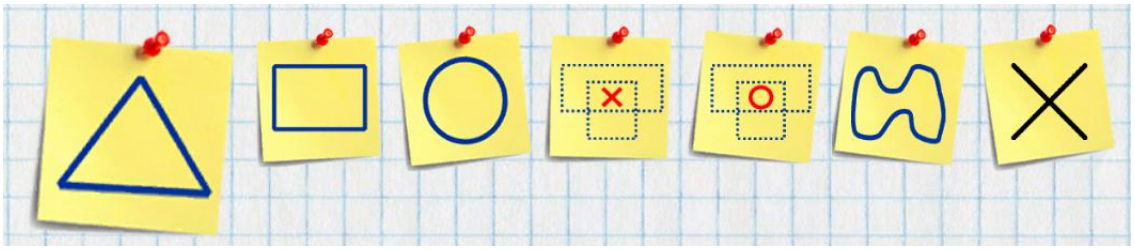


Figura 38 – Exemplo de lista de expetativas

No sentido de tornar a biblioteca SketchyDynamics mais versátil, a lista de expetativas não é considerada uma componente obrigatória, pelo que apenas é exibida se a aplicação requisitar a sua ativação. No caso de a aplicação não ativar a lista de expetativas, é sempre criado o objeto que apareceria em primeiro lugar na lista, ou seja, o primeiro resultado de reconhecimento obtido pelo reconhecedor caligráfico.

Dependendo do contexto no qual o gesto foi desenhado, a possibilidade de criação de um objeto poderá ser restringida. Por exemplo, se o utilizador desenhar um alfa, este só poderá dar origem a um conector de âncora caso seja realizado sobre dois corpos já existentes. Este tipo de restrição é também aplicado na lista de expetativas, fazendo com que apenas sejam exibidas as hipóteses válidas no contexto do gesto desenhado.

De acordo com os parâmetros passados à SketchyDynamics pela aplicação, os objetos podem ser criados enquanto a simulação se encontra em pausa ou em execução, sendo que neste último caso um objeto criado é imediatamente simulado em relação à física aplicada. De qualquer forma, sempre que é exibida a lista de expetativas, a simulação é automaticamente interrompida, para permitir ao utilizador selecionar uma das hipóteses apresentadas, sem se preocupar com o decorrer da simulação, que é automaticamente retomada quando a lista de expetativas for ocultada. Em aplicações cuja lista de expetativas seja desativada, a simulação não é interrompida após o desenho de um gesto.

### 4.2.2 Seleção de Objetos

Previamente à manipulação manual de um objeto, seja este um corpo ou um conector, o utilizador necessita primeiro de o selecionar. Esta ação interrompe automaticamente a simulação da física, possibilitando ao utilizador alterar o objeto sem ser perturbado. Caso a simulação se encontrasse em execução antes de ocorrer a seleção, esta é automaticamente retomada logo que se verifique que nenhum objeto se encontra selecionado.

A seleção de objetos é conseguida pressionando sobre eles com o estilete, ou realizando um clique esquerdo com o rato, funcionando a mesma ação também para desselecionar um objeto previamente selecionado. Deste modo, a seleção de múltiplos objetos revela-se um processo intuitivo, pois apenas é necessário pressionar continuamente sobre cada objeto a selecionar. Para além disso, mesmo quando se encontrem mais do que um objeto selecionado, é possível desselecionar objetos individualmente através do mesmo método. Existe ainda a

possibilidade de desselecionar todos os objetos de uma só vez, pressionando ou clicando sobre uma zona do ecrã onde não exista qualquer objeto ou sobre um objeto configurado como “não-selecionável”. Os objetos “não-selecionáveis” são objetos geralmente criados pela aplicação e úteis para a definição do cenário. Apesar de o utilizador não poder selecionar e manipular corpos “não-selecionáveis”, é possível utilizar conectores para os ligar a outros corpos.

Quando um objeto se encontra selecionado, as suas linhas assumem uma cor distintiva, retornando à cor original no momento em que este é desseleccionado. Esta alteração da linha do objeto dá ao utilizador um *feedback* instantâneo sobre o seu estado, como é possível verificar na Figura 39.

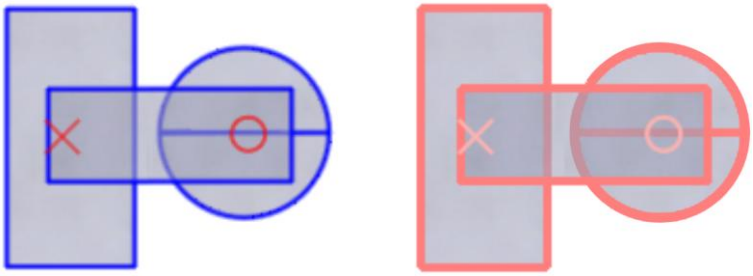


Figura 39 – Conjunto de objetos antes (à esquerda) e depois (à direita) de serem selecionados

No caso de existirem vários corpos interligados por conectores e o utilizador selecionar um desses corpos, todos os que se encontram ligados direta ou indiretamente são selecionados automaticamente, incluindo os seus conectores, desde que sejam selecionáveis (ou seja, não estejam definidos como objetos “não-selecionáveis”). Esta funcionalidade foi introduzida no sentido de melhorar a usabilidade do sistema, pois, após uma avaliação preliminar da interação, foi possível observar que quando existem corpos interligados, há uma grande probabilidade de o utilizador os querer transformar em conjunto. No entanto, caso apenas se pretenda manipular um corpo individual, os conectores existentes no mesmo devem ser removidos antes de este ser selecionado.

### 4.2.3 Translação de Objetos

Um corpo ou conector selecionado pode ser movido manualmente pelo utilizador, necessitando para isso de pressionar o estilete sobre o objeto e arrastá-lo ao longo do ecrã para a posição pretendida. O objeto deslocar-se-á em sincronia com o estilete enquanto este se mantiver em contacto com o ecrã. Quando o estilete for levantado, o objeto para de se mover e assume a sua nova posição.

No caso de existirem diversos objetos selecionados, todos eles se movem como um todo, independentemente de qual o objeto pressionado pelo estilete.

### 4.2.4 Rotação e Escala de Objetos

As transformações de escala e rotação, apenas aplicáveis a corpos, são realizadas de uma forma integrada, isto é, com uma única ação. Tal como a ação utilizada para a translação de objetos, a escala e rotação são também conseguidas pressionando e deslizando o estilete sobre o ecrã. No entanto, ao contrário do que sucede para concretizar a translação, o estilete deve ser pressionado fora das representações dos corpos selecionados. Conforme o estilete é deslizado sobre o ecrã, os corpos selecionados são rodados e escalados simultaneamente, de acordo com a posição inicial e atual do estilete.

Apesar de estas operações não poderem ser aplicadas diretamente a conectores, isto é, não é possível rodar ou escalar um conector, quando aplicadas a corpos que possuem conectores estes movem-se de forma a manterem a sua posição relativa nos corpos. Além disso, os conectores de mola alteram também a sua dimensão quando os corpos a que estão ligados são escalados.

No sentido de aplicar as transformações corretamente, o fator de escala e o ângulo de rotação são determinados com base na posição inicial e atual do estilete. O fator de escala é calculado com base no rácio da distância da posição atual do estilete ao centro do corpo e a distância inicial (antes do estilete ser movido). Já a rotação é calculada com base no ângulo existente entre duas linhas imaginárias: a linha entre a posição atual do estilete e o centro do corpo e a linha inicial, existente antes do estilete ser movido. Desta forma, aproximar ou afastar o estilete do corpo (movimento radial) causa a alteração da sua escala, enquanto que mover o estilete em torno deste (movimento tangencial) causa a sua rotação.

Caso existam diversos objetos selecionados, todos eles sofrem a mesma rotação e escala. No entanto, não é utilizado o centro do objeto como ponto de referência para o cálculo do ângulo e fator de escala, mas sim o ponto resultante da média dos centros de todos os objetos selecionados.

Como é visível na Figura 40, no sentido de auxiliar o utilizador durante uma operação de rotação e escala, é exibido um retângulo envolvente em torno dos objetos, que é também afetado pelas transformações aplicadas. É ainda apresentado um pequeno círculo no ponto de referência central em conjunto com uma linha a unir esse ponto ao ponto correspondente à posição do cursor.

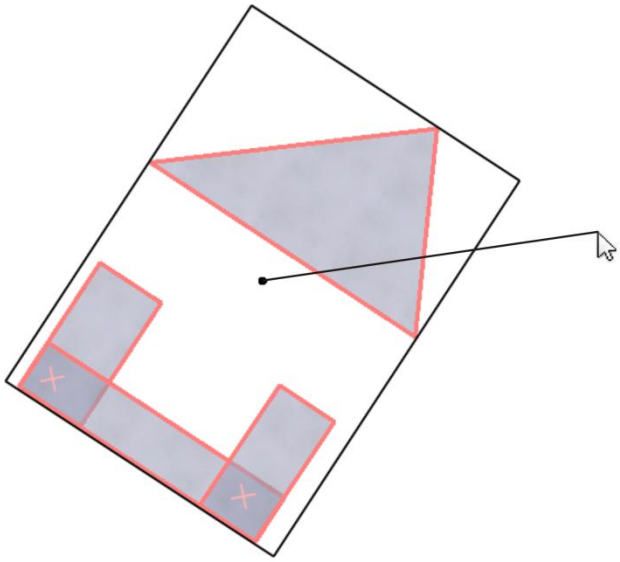


Figura 40 – Conjunto de objetos sujeitos a transformações de rotação e escala em simultâneo

### 4.2.5 Remoção de Objetos

Possibilitar ao utilizador remover objetos previamente criados contribui significativamente para a sua liberdade criativa e vontade de experimentação, revelando-se uma operação de elevada importância. Neste sentido, esta operação foi concebida de forma a ser simples, intuitiva e minimizar a carga cognitiva imposta ao utilizador. De facto, a remoção de objetos constitui apenas um caso particular do procedimento utilizado para a translação.

Quando o utilizador inicia a deslocação de um objeto, é exibido no topo da janela uma grande área retangular com um ícone de um cesto de reciclagem, que desaparece novamente quando o objeto para de ser deslocado. Para que o aparecimento desta área seja um acontecimento suave mas ao mesmo tempo evidente, esta encontra-se inicialmente numa posição acima do limite superior da janela (e portanto invisível para o utilizador) e desliza para baixo para uma posição visível. Também o seu desaparecimento segue este mecanismo, mas no sentido inverso. Se, durante o arrastamento de um objeto, o estilete for posicionado sobre esta área e de seguida levantado, todos os objetos selecionados e que se encontravam a ser deslocados são removidos da cena. Sempre que o estilete é deslizado para a área de remoção de objetos, o ícone do cesto de reciclagem assume um tom vermelho forte, para indicar ao utilizador que os objetos serão removidos caso o estilete seja levantado nesse instante. A Figura 41 apresenta a área de remoção de objetos no contexto de uma aplicação simples, quase sem conteúdos, e também o ícone de cesto de reciclagem antes e depois do estilete ser arrastado para esta área. Para além de ser possível remover conectores através do processo descrito, estes podem ainda ser arrastados para fora de um dos dois corpos que ligam, deixando de os sobrepor e sendo desta forma automaticamente removidos.

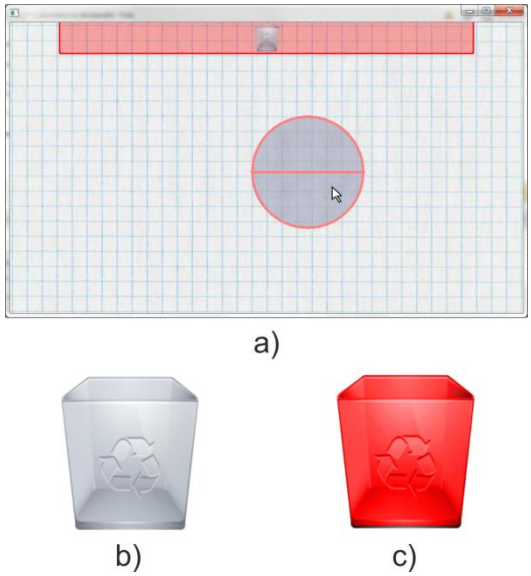



Figura 41 – Representação do mecanismo de remoção de objetos: a) aplicação simples que mostra um objeto a ser deslocado fora da área de remoção de objetos; b) ícone de remoção de objetos no seu estado normal; c) ícone de remoção de objetos quando o estilete se encontra sobre a área de remoção.

A decisão de manter a área de remoção oculta exceto quando o utilizador inicia a translação de objetos foi tomada com o intuito de tornar mais eficiente o uso de espaço de ecrã, dado apenas ser possível remover um objeto quando este está a ser manualmente deslocado. Uma outra razão para esta decisão foi a de que, ao exibir a área de remoção quando o utilizador desloca algum objeto, evidencia que este pode ser removido se arrastado para essa área.

### 4.2.6 Estados da Simulação

A biblioteca SketchyDynamics permite que uma aplicação capacite o utilizador para o controlo do estado da interação, isto é, a habilidade de a interromper e retomar. Como já mencionado, o sistema pode também interromper a simulação de forma automática quando é exibida a lista de expetativas ou é selecionado um objeto, retomando-a quando a interação correspondente termina. Isto acontece com o objetivo de não distrair o utilizador durante essas operações de interação e também de manter a simulação num estado coerente.

Sempre que a simulação é interrompida, é exibido um ícone indicativo no ecrã, o qual desaparece quando esta é retomada (Figura 42a). Dado que a simulação pode ser pausada explicitamente pelo utilizador ou automaticamente pelo sistema, são exibidos ícones distintos em cada situação. Quando a simulação é interrompida explicitamente pelo utilizador, é apresentado o ícone da Figura 42b; se esta for interrompida automaticamente, é apresentado o ícone da Figura 42c. Caso a simulação se encontre interrompida por ordem do utilizador, mesmo que ocorra uma interação que causasse uma interrupção automática por parte do sistema, o ícone exibido mantém-se. Assim, o utilizador é informado não só do estado atual da simulação, mas também se este se irá alterar quando realizar determinada operação como a escolha de um objeto da lista de expetativas. Tal como acontece para outros elementos

gráficos, a biblioteca SketchyDynamics permite à aplicação que a integra o controlo do aspeto visual destes indicadores.

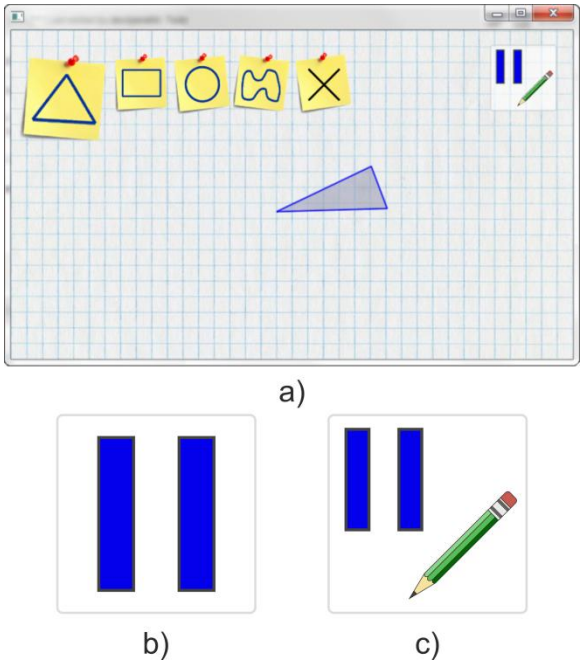



Figura 42 – Ícones de indicação do estado da simulação: a) no contexto de uma aplicação simples, durante a exibição da lista de expetativas; b) quando a simulação foi interrompida explicitamente pelo utilizador; c) quando interrompida em consequência de uma ação de interação.

Uma aplicação pode ainda manter a simulação interrompida durante todo o processo de criação/edição de objetos, executando-a apenas no final deste processo, podendo impedir a edição de objetos enquanto a simulação decorre, se o programador assim o pretender. Esta capacidade permite que uma aplicação possua uma clara distinção entre o modo de edição e o modo de simulação. É ainda possível à aplicação desativar a lista de expetativas e até a seleção de objetos, deixando assim de existir qualquer operação que interrompa a simulação, o que se revela útil para casos em que é pretendido que o utilizador interaja com um ambiente continuamente ativo, como é geralmente o caso dos jogos.

## 4.3 Arquitetura e Implementação

Uma das principais preocupações na conceção da biblioteca SketchyDynamics foi garantir a sua versatilidade, para possibilitar a criação de todo o tipo de aplicações, mas ao mesmo tempo mantê-la simples e acessível, de forma a permitir o desenvolvimento rápido de protótipos. Exemplificando esta simplicidade, com apenas três linhas de código fonte, utilizadas para inicializar a biblioteca e a janela da aplicação, um programador é capaz desenvolver uma aplicação de teste simples, que admite ao utilizador desenhar objetos e observar a sua simulação, de acordo com a força gravítica definida. Com apenas mais algumas linhas de código é possível acrescentar um fundo que permita ao utilizador fixar objetos, e até

um corpo rígido estático como suporte para que exista uma superfície que sustenha os objetos criados pelo utilizador (ver Figura 35 na página nº 54). Como a biblioteca SketchyDynamics é responsável por todas as funcionalidades descritas na secção 4.2, como a gestão da simulação da física, desenho de elementos gráficos base e interação (incluindo reconhecimento caligráfico, listas de expetativas e edição de objetos), o programador pode abstrair-se da implementação destas características e focar-se no desenvolvimento das funcionalidades de mais alto nível da aplicação.

Na Figura 43 encontra-se representado o diagrama das principais classes que definem a biblioteca SketchyDynamics, e que serão referidas ao longo desta secção.

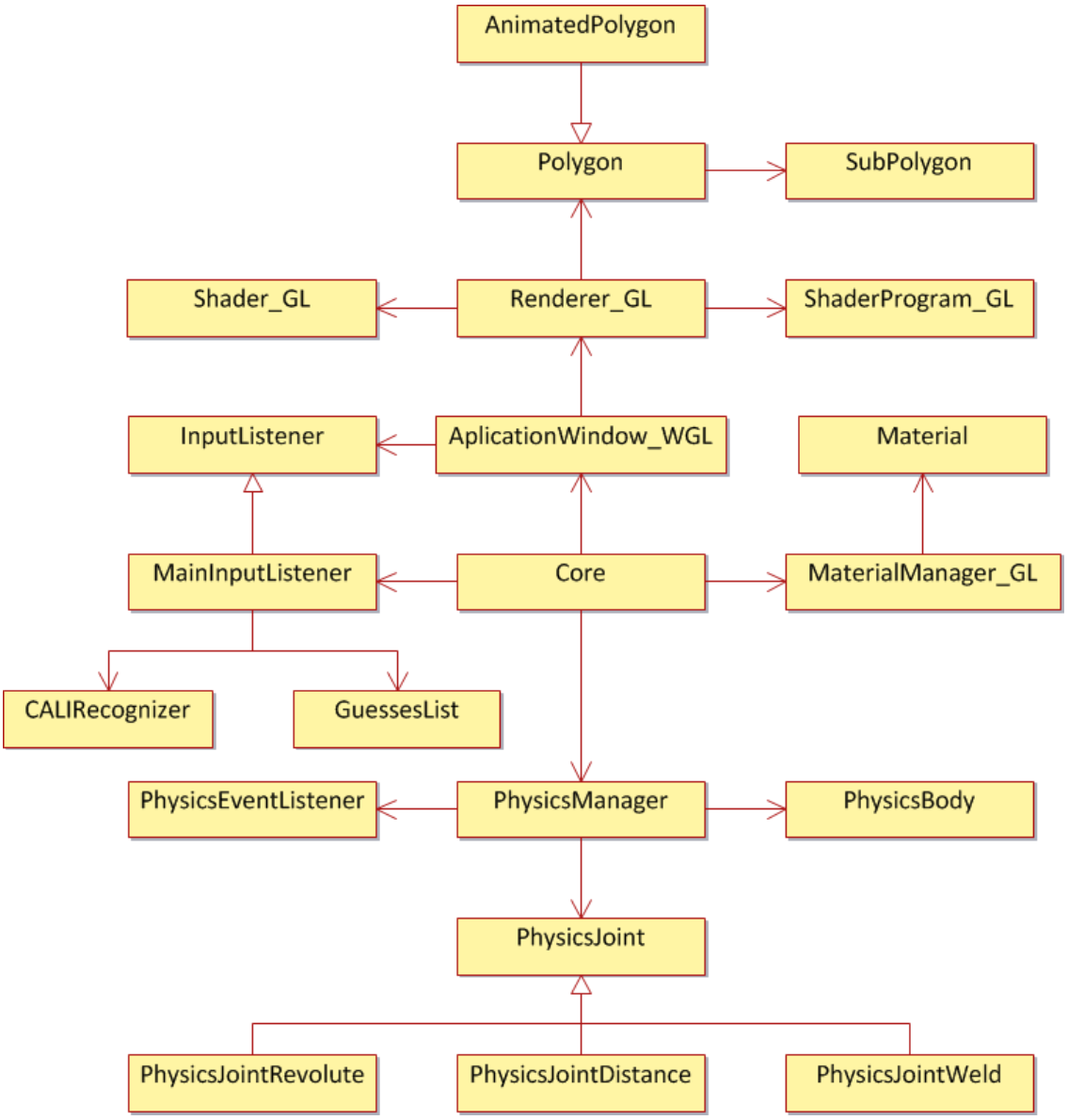



Figura 43 – Diagrama das principais classes da biblioteca SketchyDynamics

### 4.3.1 Criação da Janela da Aplicação e Controlo de *Input*

Uma característica comum a qualquer aplicação desenvolvida com recurso à biblioteca SketchyDynamics será a existência de uma janela primária de aplicação, a partir da qual são representados os seus elementos gráficos e com a qual o utilizador interage. Neste sentido, foi tomada a decisão de incluir na biblioteca o processamento necessário à criação desta janela. Esta funcionalidade é implementada através da classe ApplicationWindow_WGL, na qual se encontra o código fonte necessário para a criação de uma janela com suporte OpenGL em sistemas operativos Microsoft Windows.

Para além da criação de janelas normais, são também suportadas janelas secundárias ou subjanelas, ou seja, janelas que se encontram incorporadas dentro de outras janelas (criadas pelo programador). Isto permite que a interface e funcionalidade da SketchyDynamics sejam integradas numa janela já existente. Por exemplo, uma aplicação pode possuir uma janela com um conjunto de controlos implementados com recurso a uma qualquer ferramenta de criação de interfaces de utilizador (como Qt [27] ou Windows Forms [28]), e nessa mesma interface estar integrada a subjanela criada pela SketchyDynamics. Enquanto que na subjanela o utilizador poderia desenhar os objetos a simular e observar a simulação, os controlos existentes na janela principal poderiam permitir a alteração de parâmetros da simulação como a sua velocidade, estado ou até forças aplicadas.

Tal como acontece para a criação da janela, também a aquisição de eventos de *input* realizados pelo utilizador, como o movimento do cursor ou o pressionar de uma tecla, é encapsulada pela biblioteca. No entanto, do mesmo modo que a biblioteca requer acesso a estes eventos para lidar com as ações do utilizador, também a aplicação que a integra necessita de ser notificada, para que possa desencadear as respostas adequadas aos mesmos. Neste sentido, foi implementado o padrão *Observer* [29] que permite que múltiplas entidades sejam notificadas destes eventos em simultâneo. Para tal, apenas é necessário ao programador a criação de uma classe derivada de InputListener, implementando as funções definidas por esta, e fazer o registo de uma instância dessa classe como Observador na classe ApplicationWindow_WGL. A partir deste instante, sempre que exista um evento de *input* na janela da aplicação, será invocado no Observador o método responsável por lidar com esse evento. Ao recorrer ao padrão *Observer*, é permitido que sejam registados vários Observadores, mesmo que de diferentes classes (desde que derivadas de InputListener), sendo todos notificados em simultâneo de qualquer evento de *input*.

Com a inclusão destas funcionalidades na biblioteca SketchyDynamics, o programador não necessita de lidar com as especificidades da criação da janela e aquisição de eventos de *input* do utilizador. Ademais, dado que todo o código específico para o sistema operativo Windows se encontra na classe ApplicationWindow_WGL, o suporte a outro sistema operativo apenas requer uma diferente implementação desta classe, o que permite que o código fonte das aplicações que integrem a SketchyDynamics seja multiplataforma.

### 4.3.2 Interação

No que respeita ao processamento do *input* e da interação do utilizador, este é realizado pela classe MainInputListener. Esta é uma classe derivada de InputListener e, como tal, é notificada automaticamente dos eventos de *intput* decorrentes da interação do utilizador. No entanto, caso o programador necessite de um maior controlo sobre o processo de interação, pode retirá-la da lista de Observadores existente na classe ApplicationWindow_WGL e fazer a notificação dos métodos de tratamento de *input* através de uma outra classe por si criada. Assim o programador está apto para limitar a divulgação de eventos de *input* para a classe MainInputListener, conforme seja necessário. Considere-se uma aplicação na qual o utilizador apenas deve desenhar objetos numa determinada zona do ecrã. Ao controlar a divulgação de eventos para a classe MainInputListener, o programador pode suprimir os eventos do rato realizados fora dessa área, limitando efetivamente a interação do utilizador a essa área. O mesmo princípio pode ser aplicado para limitações de cariz temporal ou até baseadas no tipo de interação, como impedir que o utilizador crie objetos ou os selecione.

Como referido anteriormente, para a concretização do reconhecimento caligráfico a biblioteca SketchyDynamics faz uso do reconhecedor CALI. Em adição à sua utilidade no reconhecimento, o CALI é bastante útil para a obtenção de diversas características geométricas do gesto desenhado. Como descrito na secção 2.1.2, dois dos polígonos utilizados pelo CALI na identificação de gestos são o maior retângulo e o maior triângulo inscritos no polígono convexo do gesto. A partir destes polígonos é possível obter a informação necessária para a criação dos objetos (corpos ou conectores) correspondentes ao gesto desenhado pelo utilizador. Por exemplo, quando o utilizador desenha um gesto triângulo, o maior triângulo inscrito no polígono convexo do gesto possui características muito semelhantes às do gesto desenhado, mas de forma embelezada. No caso especial do objeto de forma livre, são utilizados para a sua construção os pontos que definem o gesto, em vez dos polígonos referidos anteriormente. Por questões de otimização de desempenho, antes da criação do corpo resultante do gesto de forma livre, é utilizada uma implementação do algoritmo de Douglas-Peucker [30] para a simplificação do gesto e redução do seu número de pontos, como exemplificado na Figura 44. Após a identificação do gesto desenhado e, consequentemente, do objeto a criar, bem como o levantamento das suas características, é utilizada a classe PhysicsManager para a criação dos objetos, como será detalhado posteriormente.

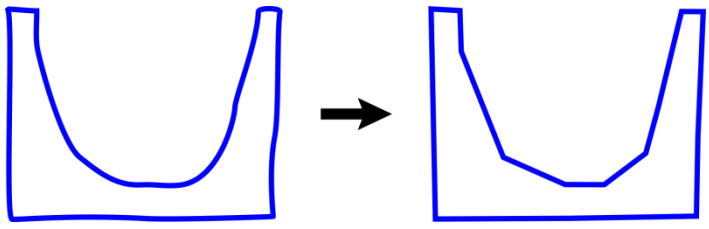


Figura 44 – Demonstração da simplificação da geometria de um gesto de uma forma livre

Para além de processar o *input* do utilizador, a classe MainInputManager é também responsável pela gestão de diversos elementos da interface, cuja presença resulta

diretamente da situação de interação atual. De entre estes elementos destacam-se a área retangular que possibilita a remoção de objetos e os elementos auxiliares exibidos durante a rotação e escala de um objeto. Também a construção da lista de expetativas é controlada por esta classe, apesar de a sua implementação se encontrar na classe GuessesList. Exemplificando, de acordo com o contexto do gesto desenhado, as identificações realizadas pelo CALI e respetivos graus de certeza, a classe MainInputManager define a ordem das opções da lista de expetativas, bem como a visibilidade de cada uma.

Considerando que, de acordo com o seu contexto, uma mesma ação realizada pelo utilizador poderá desencadear respostas diferentes por parte da aplicação, foi utilizado um mecanismo de estados de interação para melhor gerir este processo. Demonstrando esta situação, a ação de pressionar e arrastar o estilete é responsável pelo deslocamento de objetos quando realizada sobre um objeto selecionado. No entanto, caso não exista nenhum objeto selecionado, esta mesma ação irá desencadear o desenho de um gesto. O diagrama da Figura 45 dispõe todos os estados de interação presentes na biblioteca SketchyDynamics, em conjunto com as ações que desencadeiam transições entre estes.

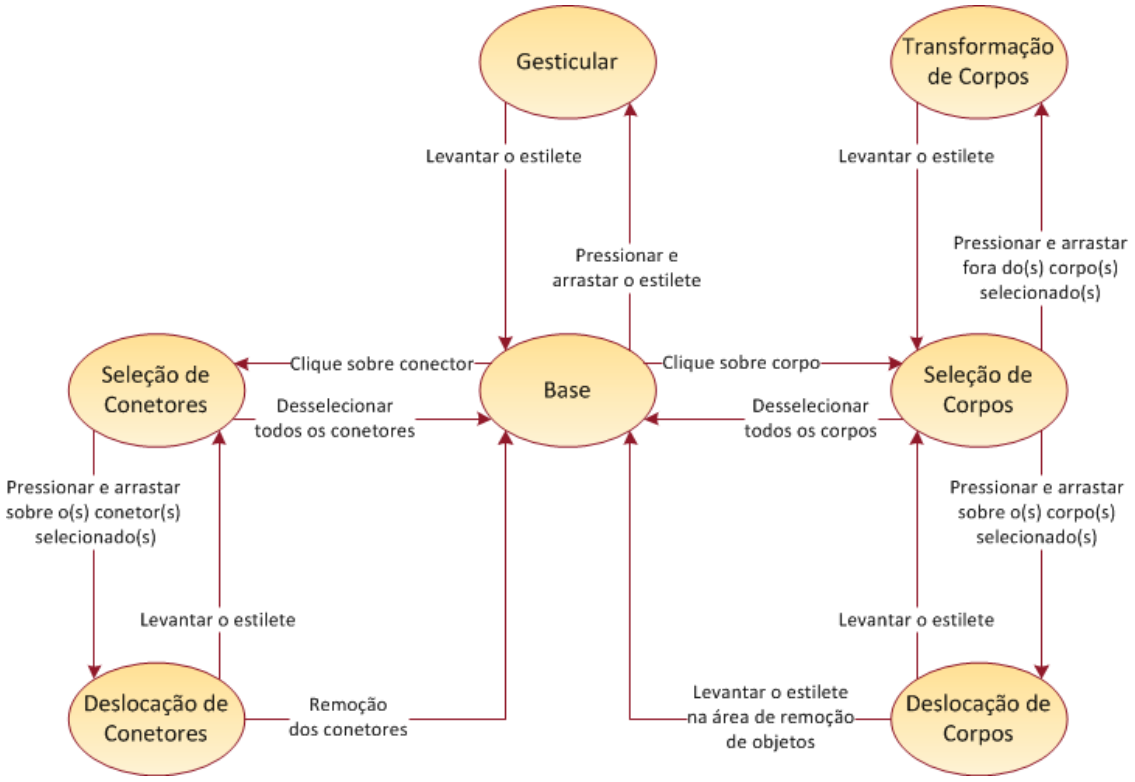



Figura 45 – Diagrama de estados de interação

Após uma avaliação preliminar à utilização das funcionalidades da biblioteca, foi possível constatar que era detetado um pequeno movimento involuntário, geralmente inferior a três píxeis, entre o pressionar e o levantar do estilete. Esta situação tornava complicada a execução de ações como a seleção de objetos, conseguida com o toque e levantar do estilete (o equivalente a um clique). A solução para este problema passou pela definição de um limite

mínimo de três píxeis para que a translação do cursor seja considerada intencional. Deste modo, caso a distância percorrida pelo movimento entre o pressionar e o levantar seja inferior a três píxeis, o sistema ignora esse movimento e considera apenas as ações de pressionar e levantar o estilete.

### 4.3.3 Gestão de Corpos Rígidos e Conectores

Para a realização da simulação da dinâmica dos corpos rígidos foi utilizado o Box2D [31], um motor de física de código aberto capaz de realizar simulações bidimensionais. O Box2D é responsável por simular o movimento dos corpos, processar colisões entre estes e lidar com a existência de conectores entre os corpos. Apesar de recorrer ao Box2D, a biblioteca SketchyDynamics não o encapsula nem oculta. Assim, o programador tem acesso a todos os objetos e funcionalidades existentes no Box2D, para que os possa parametrizar de acordo com as necessidades da sua aplicação. Para exemplificar a utilidade desta decisão, destaca-se o caso em que se pretenda especificar a elasticidade de cada corpo com um valor diferente daquele que é atribuído por omissão pela SketchyDynamics. Ao ter acesso ao Box2D, o programador pode simplesmente alterar o parâmetro referente à elasticidade diretamente para cada corpo. Uma funcionalidade presente no Box2D e que merece referência é a possibilidade de definir um corpo como estático, para que não possua qualquer dinâmica, o que se revela de extrema utilidade para a criação de objetos de cenário como o fundo ou a régua apresentados na Figura 35 (página nº 54).

No centro da gestão de todas as características relacionadas com a simulação da física, incluindo a criação de corpos e conectores, encontra-se a classe PhysicsManager. Sempre que é necessário definir um novo corpo, esta classe instancia um objeto da classe PhysicsBody, cuja tarefa é a de aliar a representação do corpo no Box2D à sua representação visual no ecrã. Para tal, a classe PhysicsBody deduz a representação gráfica do novo corpo a partir da geometria que se encontra definida perante o Box2D. No caso particular de corpos resultantes de gestos de forma livre, dado que estes podem constituir polígonos côncavos, não suportados pelo Box2D, estes são decompostos em polígonos triangulares com recurso a uma biblioteca de triangulação de polígonos [32], para que possam assim originar corpos cuja geometria seja suportada pelo Box2D. Em resultado deste processo, os triângulos que compõem um corpo deste tipo são visíveis na sua representação gráfica, como é observável na Figura 46. A fim de reduzir o número de triângulos resultante do processo de triangulação, este não é aplicado ao gesto original mas sim à sua simplificação, já apresentada anteriormente (Figura 44).

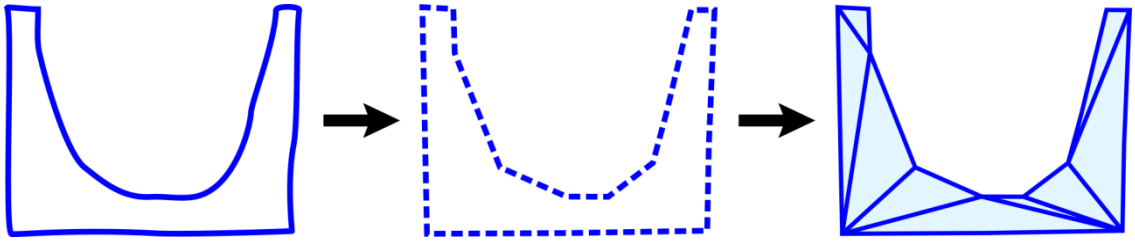


Figura 46 – Geometria e representação de um corpo resultante de um gesto de forma livre

Ainda no que se refere à representação visual de corpos, apesar de a simulação e visualização serem realizadas em 2D, podem ocorrer sobreposições de corpos, nomeadamente quando se encontram ligados por conectores, ou quando o utilizador desenha um novo corpo sobre outro já existente. Nessas situações, os corpos são representados de acordo com sua ordem de desenho, pelo que os corpos criados mais recentemente sobrepõem-se aos mais antigos, encobrindo-os caso possuam texturas opacas.

Relativamente à criação de conectores, esta é também concretizada pela classe PhysicsManager, que procede à instanciação de objetos da classe PhysicsJointWeld, PhysicsJointRevolute ou PhysicsJointDistance, conforme seja necessário criar um conector de âncora, de rotação ou de mola, respetivamente. Estas classes possuem o conhecimento necessário para a correta representação do conector que caracterizam, isto é, para a associação do conector existente no Box2D a uma representação gráfica correta do mesmo.

A classe PhysicsManager disponibiliza também as funções necessárias para a seleção de objetos, para a aplicação de transformações e até para a sua destruição, o que desencadeia a destruição da entidade no Box2D e da sua representação gráfica no ecrã. Esta classe permite ainda definir que determinado objeto não pode ser selecionado, revelando-se útil para impedir que o utilizador o altere. Esta situação é frequente em objetos constituintes do cenário da aplicação, pois geralmente não se pretende que o utilizador os modifique.

Apesar de concretizadas automaticamente pela biblioteca quando o utilizador realiza a ação correspondente, a criação, seleção e manipulação de objetos pode ser feita diretamente pela aplicação que integra a biblioteca. Ao disponibilizar o acesso e controlo de todas estas funcionalidades, é oferecida ao programador a capacidade para a criação automática de objetos, ou a manipulação de parâmetros dos objetos criados pelo utilizador. Um caso exemplificativo seria o de uma aplicação na qual são criados automaticamente corpos que, quando em contacto com um corpo criado pelo utilizador, explodem e alteram a dimensão do corpo com o qual colidiram.

A classe PhysicsManager proporciona também ao programador a oportunidade para desencadear ações no decorrer de determinados eventos relacionados com a simulação. Isto é alcançado com recurso ao padrão *Observer*, tendo o programador apenas que implementar uma classe derivada de PhysicsEventListener e registar uma instância dessa na classe PhysicsManager. Assim, essa instância verá a suas funções invocadas sempre que ocorrer um evento como a criação de um objeto ou uma alteração do estado da simulação.

### 4.3.4 Capacidades Gráficas

Para a representação gráfica no ecrã dos diversos elementos visuais existentes é utilizada a API gráfica OpenGL. Apesar disso, a biblioteca SketchyDynamics foi desenhada de modo a permitir a implementação e integração de outras APIs gráficas, o que se mostra vantajoso para dispositivos que utilizam o OpenGL ES[18] ou para aplicações que dependam de funcionalidades da API Direct3D. Para tal seria apenas necessário criar versões das classes ApplicationWindow_WGL, Renderer_GL e MaterialManager_GL, que acomodassem as especificidades da API gráfica desejada. Ao encapsular todo o código referente à API gráfica num pequeno conjunto de classes, é efetivamente criada uma camada de abstração que permite que esta seja alterada sem a necessidade de proceder a modificações à restante funcionalidade da biblioteca.

Como parte do processo de abstração da API gráfica, foram implementadas diversas funcionalidades relacionadas com a criação e manipulação de elementos gráficos, geralmente presentes em bibliotecas ou motores gráficos. Para além da sua utilização internamente pela biblioteca SketchyDynamics, estas funcionalidades visam reduzir o esforço do programador na implementação da componente visual da aplicação. De entre estas funcionalidades destaca-se a capacidade para a criação de objetos poligonais através da definição dos seus vértices, recorrendo à classe Polygon. Esta classe permite ainda a manipulação de cada polígono, através da aplicação de transformações geométricas, comutação da sua visibilidade e também da sua cor e textura. A classe AnimatedPolygon possibilita, para além das capacidades expostas pela classe Polygon da qual é derivada, que as transformações aplicadas sejam animadas e faseadas ao longo de um período de tempo definido. Com esta classe é possível a criação de efeitos de objetos deslizantes como o utilizado para a exibir a área de remoção de objetos, sendo necessária apenas a definição da nova posição desejada para o objeto e a velocidade a que este se deve mover durante a transição.

Também a utilização e aplicação de texturas é facilitada, pois a biblioteca permite ao programador carregar uma textura de um ficheiro indicando apenas o nome e localização do mesmo. Esta funcionalidade é exposta pela classe de gestão de materiais MaterialManager_GL, cuja é responsável por carregar e inicializar a textura perante a API gráfica OpenGL. Independentemente de serem constituídos por texturas ou apenas cores simples, os materiais, representados pela classe Material, podem ser aplicados a qualquer objeto Polygon para que este passe a exibir as características visuais definidas pelo material.

A apresentação dos objetos no ecrã é conseguida através da classe Renderer_GL, com recurso a três filas de desenho ou *rendering queues* para a adição de objetos Polygon. Cada fila de desenho está associada a uma camada visual para que cada objeto possa ser desenhado no fundo, na frente (como um elemento de interface do utilizador) ou entre estas duas posições. Dentro de cada fila de desenho, o objeto pode ainda ver definida a sua profundidade ou ordem de apresentação, pelo que objetos cuja profundidade é inferior são exibidos à frente

[18] Subsecção do OpenGL utilizada em sistemas embutidos.

dos de maior profundidade que se encontram dentro da mesma fila. A classe Renderer_GL proporciona ainda ao programador funções para consulta de cena ou *scene query*, úteis para determinar quais objetos se situam num determinado ponto da cena, processo muito utilizado para determinar se o utilizador pressionou sobre algum objeto e qual.

Também a inclusão do processo de criação da janela da aplicação na biblioteca SketchyDynamics, descrita na secção 4.3.1, foi resultado do processo de abstração da API de desenho.

## 4.4 Conclusão

Neste capítulo foi apresentada a biblioteca SketchyDynamics, que conjuga um motor de simulação da dinâmica de corpos rígidos com uma interface caligráfica, munida das ações necessárias para o controlo da simulação, com o objetivo de facilitar a criação de aplicações que necessitem destas funcionalidades.

Dado tratar-se de uma biblioteca de programação, esta deverá ser utilizada por programadores, para a conceção de aplicações, e por utilizadores, na utilização dessas mesmas aplicações. Neste sentido, a apresentação das suas funcionalidades foi dividida em duas grandes secções. Na primeira, orientada ao utilizador, foi descrito o estilo de interação adotado, as ações e operações disponíveis, bem como as suas funcionalidades e funcionamento geral.

Na segunda secção, direcionada ao programador, foi descrita a arquitetura implementada, bem como os seus principais componentes. Foram também referidas diversas características que permitem ao programador o controlo total de todo o processo de simulação e interação, bem como do aspeto gráfico dos componentes incluídos na biblioteca. Além disso, foram também apresentados mecanismos que permitem uma abstração de um conjunto de tarefas relacionadas com a criação e manipulação da representação gráfica de objetos, e que se encontram ao dispor do programador.

# 5 Avaliação de Usabilidade da Biblioteca SketchyDynamics

Todo o desenvolvimento da biblioteca SketchyDynamics foi acompanhado por diversos testes preliminares e informais, nos quais utilizadores externos a este trabalho experimentaram, avaliaram e opinaram sobre as diferentes funcionalidades que iam sendo implementadas. Este procedimento vai de encontro ao definido pela metodologia em estrela (secções 3.1 e 4.1), adotada no desenvolvimento da biblioteca, e permitiu receber um *feedback* instantâneo sobre a perceção do utilizador e aperfeiçoar alguns aspetos em simultâneo com o seu desenvolvimento.

Apesar de os testes e avaliações informais terem desempenhado um papel de elevado interesse na evolução da biblioteca, após a sua conclusão foi realizada uma avaliação mais rigorosa da usabilidade da mesma, de modo a verificar a validade das decisões tomadas. Esta avaliação, realizada com um grupo de participantes, consistiu na execução de um teste de eficiência cuja duração foi cronometrada, e consequente preenchimento de um questionário sobre as funcionalidades oferecidas pela biblioteca SketchyDynamics. Este capítulo incide sobre a avaliação de usabilidade realizada, através da exposição e discussão dos resultados obtidos.

## 5.1 Protótipo de Demonstração da Biblioteca SketchyDynamics

Para permitir a experimentação e avaliação da usabilidade da biblioteca SketchyDynamics por parte dos participantes da sessão de avaliação, foi desenvolvida uma aplicação protótipo simples que expõe todas as funcionalidades presentes na biblioteca sem acrescentar qualquer funcionalidade adicional, pois apenas se pretende avaliar o que é oferecido pela biblioteca. Deste modo, o protótipo expõe todas as funcionalidades facultadas pela biblioteca, apresentadas no capítulo 4.

Na Figura 47 é apresentada uma captura de ecrã do protótipo, no qual o utilizador criou alguns corpos rígidos (a azul) e conectores (a vermelho). A "régua", visível também na figura, não constitui um objeto criado pelo utilizador, pois é parte integrante do protótipo e cumpre o propósito de estabelecer um suporte para os objetos assentarem. Se isto não se verificasse, os objetos cairiam para fora da área de desenho, em resultado do efeito da gravidade.

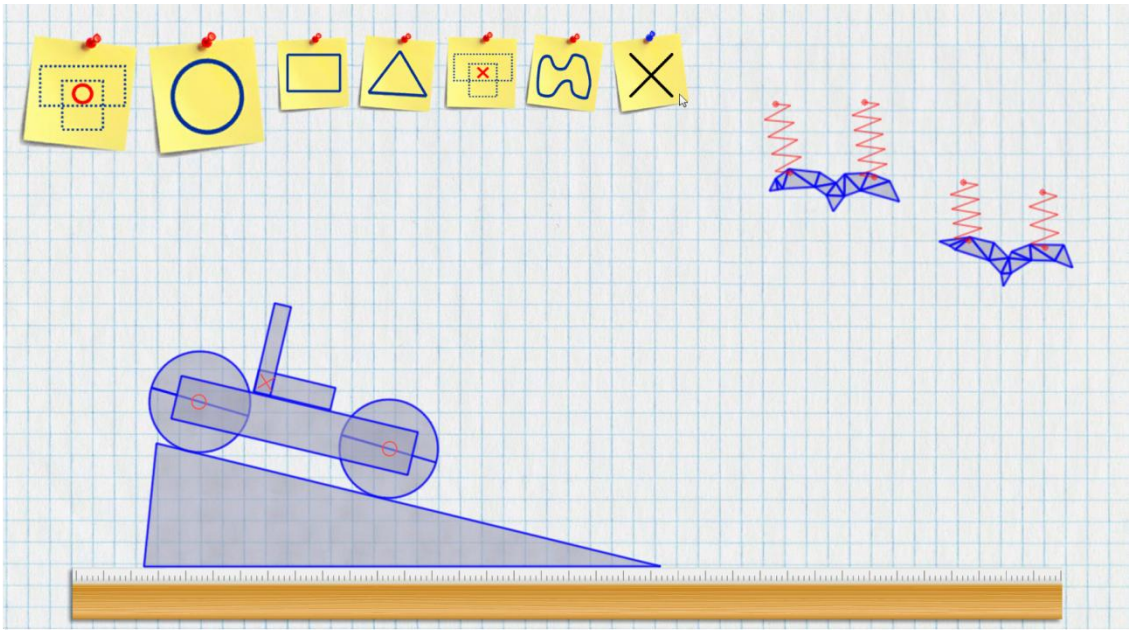


Figura 47 – Protótipo de demonstração da biblioteca SketchyDynamics, com alguns objetos criados pelo utilizador

Na construção do protótipo foram considerados alguns cuidados, de modo a aumentar o realismo da interação e reforçar a metáfora organizacional do esboço feito com uma caneta ou lápis sobre uma folha de papel. A colocação de uma textura de folha de papel quadriculado como fundo da aplicação, em conjunto com a utilização de linha azul e linha vermelha para delinear os corpos rígidos e conectores, respetivamente, são dois aspetos que visam uma maior aproximação a essa metáfora. Também a utilização de uma textura de régua para o corpo rígido da base, bem como a configuração de *post-its* para a lista de expectativas, com a sua colocação deliberadamente desalinhada, visam proporcionar um ambiente mais realista.

## 5.2 Procedimento

A sessão de avaliação de usabilidade das funcionalidades da biblioteca SketchyDynamics realizou-se no dia 25 de Setembro de 2012, nas instalações do GECAD (Grupo de Investigação em Engenharia do Conhecimento e Apoio à Decisão), no ISEP. A sessão contou com a colaboração de 8 participantes, grupo este constituído por docentes e alunos do Mestrado em Engenharia Informática do ISEP e investigadores do GECAD.

Na definição do número de elementos a participar na sessão de avaliação, foi considerado o estudo realizado por Virzi [33], que estabelece uma relação entre o número de participantes numa sessão de avaliação e a percentagem de problemas de usabilidade detetados. De acordo com Virzi, com apenas 3 participantes é possível revelar aproximadamente 65% dos problemas de usabilidade (Figura 48). Este valor sobe para 80% caso na sessão participem 5 elementos. Com 9 elementos, Virzi refere que são detetados cerca de 95% dos problemas. Este estudo revela ainda que apesar de possibilitar a descoberta de um maior número de problemas, o incremento do número de participantes não ocasiona resultados proporcionais, podendo conduzir a um desaproveitamento de recursos. Também Nielson [34] apresenta resultados de um estudo semelhante que mostra que, com 5 participantes, é possível detetar aproximadamente 75% dos problemas, valor que aumenta para 85% com 9 participantes. Apesar de inferiores, estes valores não diferem significativamente daqueles apresentados por Virzi. Com base nestes estudos, é esperado que, ao recorrer a oito participantes, seja possível identificar a grande maioria dos problemas de usabilidade presentes na biblioteca SketchyDynamics.

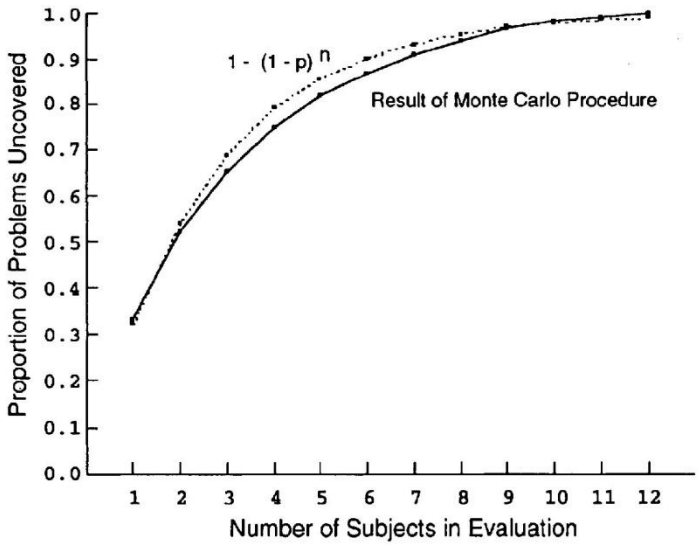



Figura 48 – Relação entre o número de problemas de usabilidade detetados e o número de participantes de uma avaliação [33]

Para a realização da experiência e teste de eficiência, os participantes serviram-se de um posto de trabalho no qual se encontrava instalado o protótipo para avaliação, munido de um ecrã táctil Wacom Cintiq 15X com estilete. Ao utilizar um ecrã com estilete é esperado que os participantes possam tirar o maior partido dos conceitos de interação definidos na biblioteca SketchyDynamics. Contudo, foram também utilizados postos de trabalho equipados com rato tradicional.

A realização da sessão de avaliação de usabilidade foi conseguida com recurso a um guião de acompanhamento[19], distribuído por todos os participantes. Seguindo a metodologia de

[19] Ver Anexo C.

Pereira [35], este guião contextualiza e expõe o propósito da sessão e o seu roteiro de tarefas. Além disso, incorpora a informação necessária para a realização do teste de eficiência, que consiste na criação de uma cena composta por diversos corpos rígidos e conectores. É importante referir que, apesar de o guião indicar uma divisão do teste de eficiência em duas partes, apenas serão aqui apresentados os tempos totais, resultantes da execução consecutiva das duas partes. Esta divisão foi efetuada para que na situação de o participante não conseguir finalizar o teste, fosse possível, ainda assim, recolher o tempo decorrido na execução da primeira parte, situação que não se verificou.

O guião da sessão constitui ainda um questionário a preencher pelos participantes após a realização do teste de eficiência. A partir das respostas a este questionário, espera-se conhecer a opinião dos participantes sobre as diversas componentes e funcionalidades existentes na biblioteca SketchyDynamics, a fim de entender as dificuldades encontradas por estes. Adicionalmente é encorajado o espírito crítico por parte dos utilizadores, sendo solicitado que exponham os seus comentários e sugestões de aperfeiçoamento.

A sessão de avaliação iniciou-se com a receção aos participantes, altura em que foi exposto o objetivo da sessão no contexto do trabalho apresentado neste documento, e descritas as tarefas a concretizar. Foi ainda pedido aos participantes que não se retraíssem a tecer comentários e expor as suas dúvidas.

Após a receção aos participantes, foi realizada uma apresentação do protótipo de avaliação da biblioteca SketchyDynamics. Esta teve como propósito demonstrar todas as características constituintes do protótipo, bem como o estilo de interação adotado, no sentido de munir os participantes com o conhecimento necessário para a concretização do teste de eficiência. No final desta apresentação existiu ainda um período reservado para a colocação de questões por parte dos participantes, e respetivo esclarecimento por parte do autor.

Finda a apresentação do protótipo, seguiu-se a realização do teste de eficiência. Este teste consistiu no registo do tempo demorado por cada participante na criação de uma cena apresentada no guião da sessão. Como é possível observar na Figura 49, a cena proposta incorpora um conjunto de 17 corpos rígidos e 11 conectores a criar pelo participante, estando presente pelo menos um corpo e conector de cada tipo. Destacam-se alguns objetos mais complexos, como o “martelo” fixado no fundo da aplicação e composto por um corpo de forma livre e outro retangular, em conjunto com dois conectores diferentes; o “carro”, formado por dois corpos circulares e um retangular, ligados por dois conectores de rotação; ou até o “baloiço”, que é constituído por dois corpos triangulares fixados num corpo retangular através de conectores de âncora e fixados ao fundo por conectores de mola. Esta cena obriga ainda à alteração do estado da simulação pelo participante, pois é impossibilitada a sua criação sem a interrupção prévia da simulação.

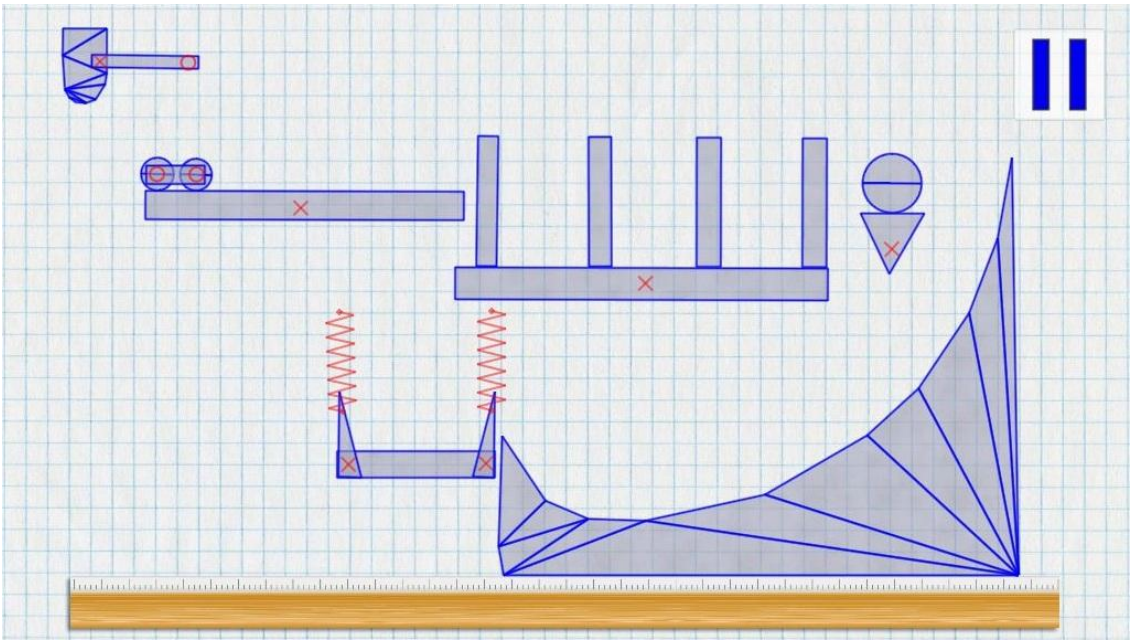


Figura 49 – Cena a produzir pelos participantes durante o teste de eficiência da avaliação de usabilidade

Para a realização do teste de eficiência, os participantes foram divididos em dois grupos. O primeiro grupo, constituído por 5 elementos, realizou o teste em sequência, no posto de trabalho equipado com ecrã tátil e estilete. Este grupo dispôs de um período não cronometrado de familiarização com o dispositivo e com o protótipo. Em alguns casos, o ecrã tátil sofreu afinações da sua sensibilidade para melhor se ajustar ao participante. Além disso, o autor esteve presente durante toda a realização do teste, no sentido de prestar assistência ao utilizador. Este grupo de participantes teve também a oportunidade para experimentar o protótipo em postos de trabalho com ratos tradicionais, de modo a melhor responder ao questionário.

O segundo grupo, constituído pelos restantes 3 participantes, realizou o teste em conjunto num único terminal equipado com rato tradicional. A este grupo não foi dada qualquer assistência durante a realização do teste e a familiarização com o protótipo decorreu em simultâneo com a execução do teste. O objetivo do teste realizado por este grupo consistiu em avaliar o desempenho em condições adversas, na qual os participantes apenas dispunham da informação veiculada pela apresentação, não usufruindo de qualquer dispositivo de interação especializado, sendo sujeitos à exploração individual de soluções para as dificuldades que fossem ocorrendo, sem qualquer assistência da parte dos orientadores da sessão.

Após o término do teste de eficiência, cada um dos oito participantes procedeu ao preenchimento do questionário disponibilizado. Além da resposta a diversas questões sobre a utilização do protótipo desenvolvido com a biblioteca SketchyDynamics, os participantes foram ainda incentivados a expor as suas críticas e sugestões com base na sua experiência.

## 5.3 Análise dos Resultados

Após a análise do tempo despendido por cada participante na execução do teste de eficiência e a sua confrontação com o tempo inicialmente previsto de 15 minutos, é possível afirmar que os resultados são bastante satisfatórios, considerando a complexidade da cena a criar e a inexperiência dos participantes com a biblioteca SketchyDynamics. Os participantes do primeiro grupo (do número 1 ao número 5), que usufruíram do equipamento com estilete para a realização do teste, necessitaram em média de 9 minutos e 12 segundos para a produção da cena proposta, com um desvio-padrão de 3 minutos e 34 segundos. Deste grupo, apenas um dos participantes necessitou de um tempo superior a 10 minutos, tendo o participante mais rápido realizado o teste em apenas 6 minutos. Como foi anteriormente referido, a este primeiro grupo de participantes foi facultado um pequeno período para habituação ao equipamento e ao protótipo, não superior a 10 minutos. O segundo grupo de participantes, composto pelos elementos 6, 7 e 8, não dispôs deste período de habituação, pelo que o contacto com o sistema foi cronometrado logo desde o início. Além disso, não dispôs de qualquer ajuda do autor, contando apenas com a cooperação entre os três elementos. Ainda assim, este grupo conseguiu completar o teste de eficiência em 24 minutos, o que é um resultado bastante positivo, dadas as condições em que o teste foi realizado. No gráfico da Figura 50 encontra-se individualizada a duração do teste de eficiência por cada participante. Note-se que, pelo facto dos participantes 6, 7 e 8 terem realizado o teste em conjunto, os seus resultados encontram-se unificados.

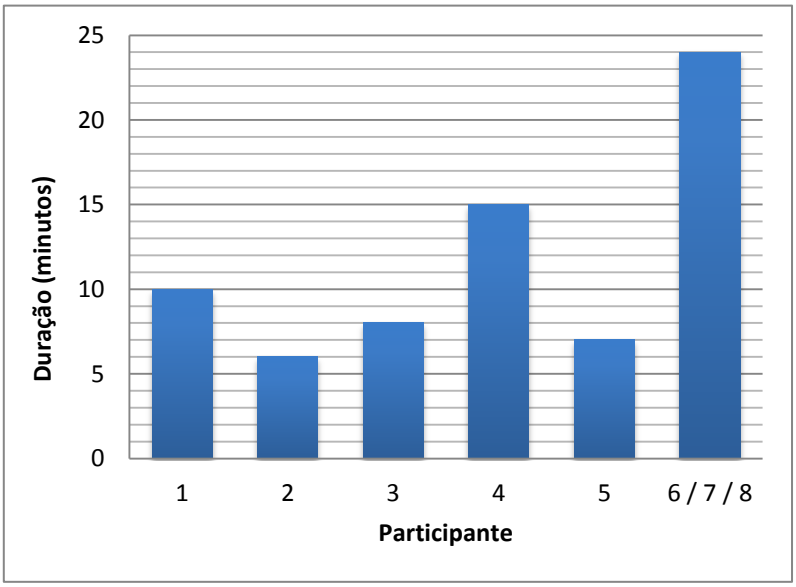



Figura 50 – Duração da realização do teste de eficiência por cada participante

Após a realização do teste de eficiência, cada um dos participantes procedeu ao preenchimento de um questionário sobre a utilização do protótipo construído com a biblioteca SketchyDynamics. Este questionário, incluído no guião da sessão, é constituído por 40 questões, a que os participantes responderam de acordo com uma escala entre dois atributos opostos como “péssimo” e “excelente”, associados ao valor mínimo de -3 e máximo

de +3, respetivamente. Serão apresentadas nesta secção as classificações médias obtidas em cada questão, dentro da escala referida. Estas classificações serão dispostas sob a forma de gráficos, nos quais cada questão se encontra numerada de acordo com a enumeração presente no guião da sessão de avaliação.

No que se refere à adequação do protótipo ao dispositivo de interação, a média das respostas dos participantes, apresentada no gráfico da Figura 51, mostra que não existe uma clara preferência entre o rato tradicional e o ecrã interativo com estilete, tendo a utilização do rato recebido uma classificação ligeiramente superior. Este resultado contraria em parte o que seria esperado, pois o ecrã com estilete constitui um modo de interação mais aproximado ao paradigma papel-lápis, pelo que a utilização de uma interface caligráfica com este equipamento deveria ser mais natural para os participantes. No entanto, o elevado conforto dos participantes com o rato tradicional, em conjunto com alguns problemas existentes na utilização do equipamento com estilete, descritos posteriormente, poderá ter tido um impacto significativo nestes resultados.

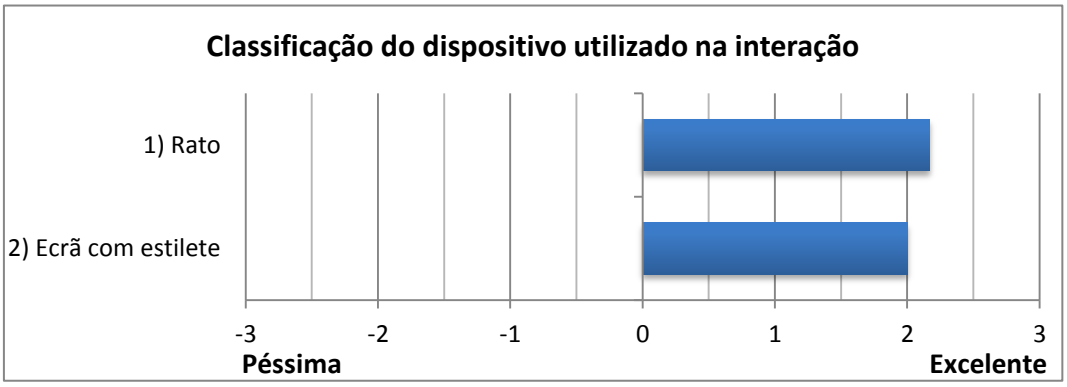



Figura 51 – Classificação média da adequação do dispositivo de interação utilizado na avaliação

Relativamente à adequação dos gestos de criação de corpos e conectores, os resultados obtidos são bastante positivos, pois, como é possível observar no gráfico da Figura 52, a maioria dos gestos obteve classificações médias iguais ou superiores a 2 valores, excetuando-se o gesto de criação de corpos de forma livre e o de criação de conectores de âncora. No que respeita ao gesto de criação de forma livre, a sua classificação inferior deve-se maioritariamente à existência de uma restrição na biblioteca SketchyDynamics que não permite a ocorrência de interseções no gesto de criação de um corpo de forma livre. Esta limitação resultou na impossibilidade de criação do corpo livre pretendido sempre que sucedia uma interseção na linha do gesto, o que poderá ter originado alguma frustração no participante. No que se refere ao gesto de criação de conectores de âncora, foi notada alguma dificuldade no desenho deste gesto com rato tradicional, advindo daí a sua classificação inferior aos restantes.

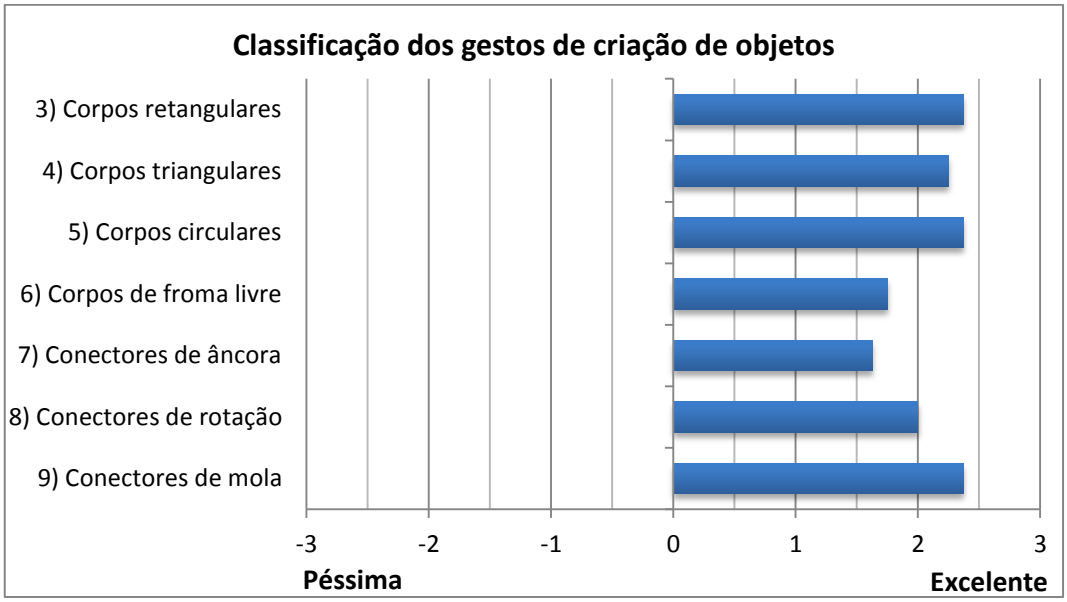



Figura 52 – Classificação média da adequação dos gestos de criação de objetos

As questões seguintes pretendiam determinar a opinião dos participantes em relação à estrutura e disposição dos elementos constituintes da interface gráfica. Como se pode observar na Figura 53, a classificação obtida mostra que o indicador de estados da simulação carece de melhorias.

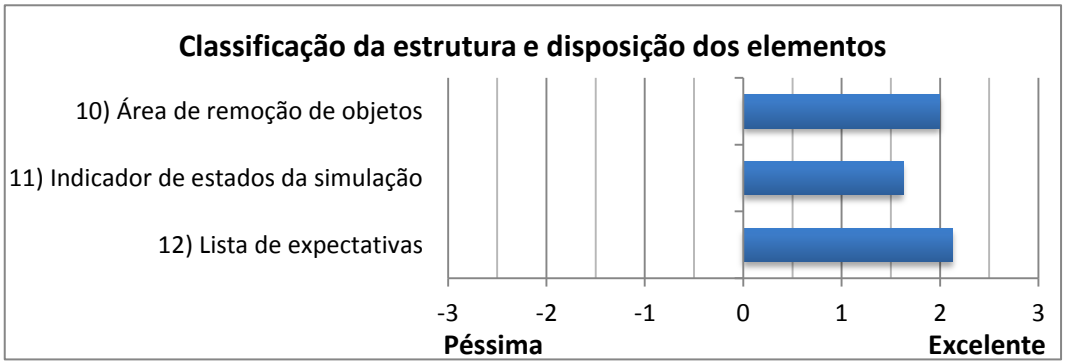



Figura 53 – Classificação média da estrutura e disposição dos elementos da interface gráfica

No que se refere ao processo de criação de objetos, os participantes mostraram-se generalizadamente bastante satisfeitos. Como se pode concluir pelo gráfico da Figura 54, dos elementos envolvidos na criação de objetos, apenas a criação de conectores e o reconhecimento de gestos obtiveram classificações inferiores a 2 valores. Para além da dificuldade na criação de conectores de âncora, uma possível razão para a classificação referente à criação de conectores ser inferior prende-se com a sua complexidade relativamente aos corpos. Isto é, enquanto que um corpo pode ser criado livremente, um conector apenas pode ser criado se o seu contexto o permitir. No entanto, a opinião global média dos participantes relativa ao processo de criação de objetos é bastante satisfatória, situando-se acima dos 2 valores.

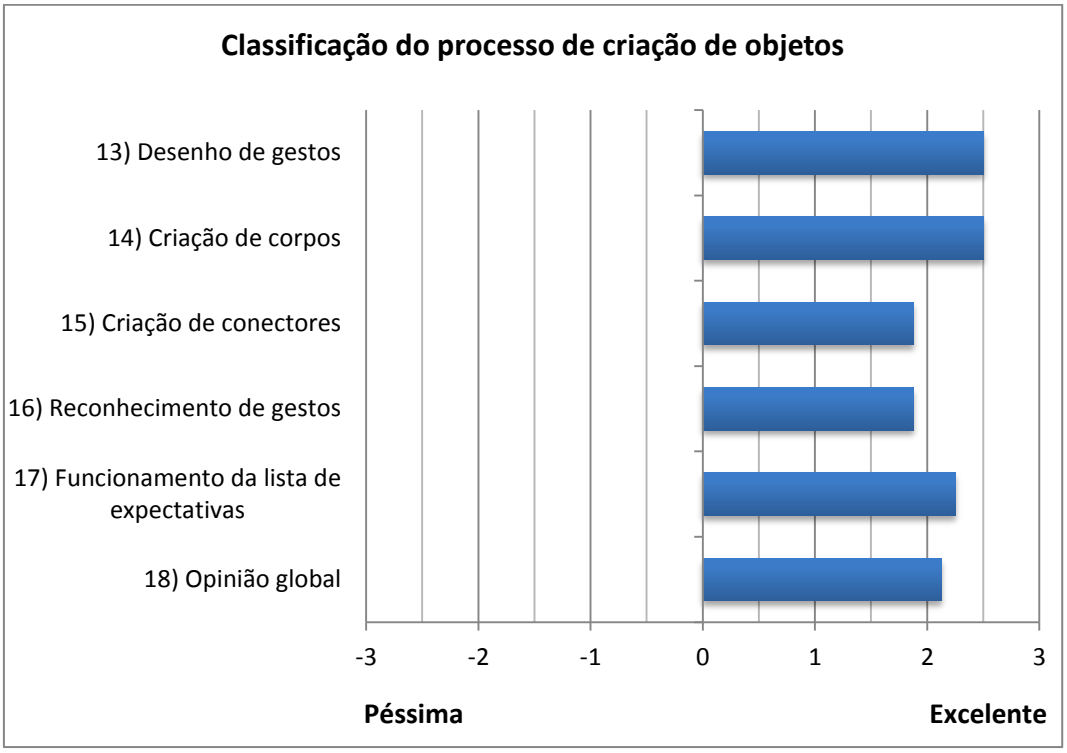



Figura 54 – Classificação média do processo de criação de objetos

As questões subsequentes, cujas classificações médias se encontram no gráfico da Figura 55, pretendem avaliar as diversas ações que constituem o processo de edição de objetos. Destas, apenas a ação de seleção contínua de múltiplos objetos obteve uma classificação inferior a 2 valores. Ao observar os participantes durante a utilização do protótipo, foi evidente que o método definido para a seleção de múltiplos objetos causou alguma dificuldade, dado que entra em conflito com a experiência comum com aplicações informáticas. Enquanto que na generalidade das aplicações um clique sobre um objeto seleciona-o e desseleciona quaisquer outro objeto que se encontre selecionado, na biblioteca SketchyDynamics ao clicar sobre um objeto seleciona-o, mas não desseleciona os restantes objetos. Em resultado deste conflito, os participantes deparavam-se a fazer transformações em objetos que julgavam já estar desselecionados. Em todo o caso, a opinião global dos participantes em relação ao processo de edição de objetos foi boa e apresenta um valor médio superior a 2 valores.

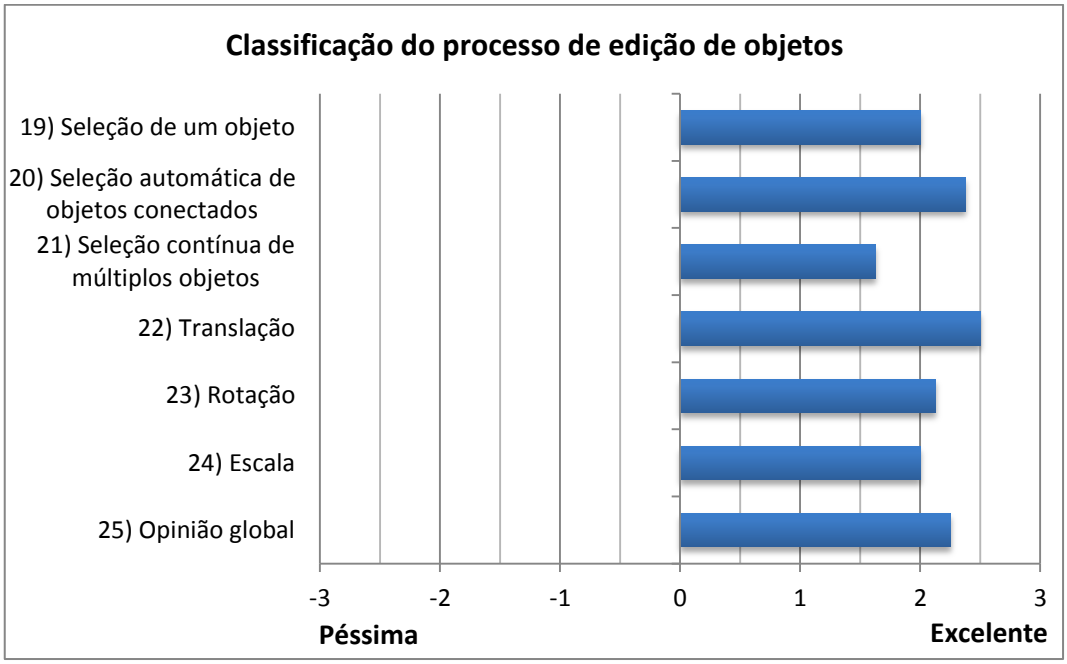



Figura 55 – Classificação média do processo de edição de objetos

Apesar de os participantes acharem útil a possibilidade de remoção de conectores deslocando-os para fora dos corpos que interligam, os resultados, visíveis no gráfico da Figura 56, mostram a necessidade de melhorias no que se refere ao processo de remoção de objetos. Uma das críticas mencionadas por grande parte dos participantes foi a impossibilidade de remoção de objetos através da tecla "Delete". De facto, esta é uma característica presente na grande maioria das aplicações informáticas para a realização da operação de apagar ou remover um objeto.

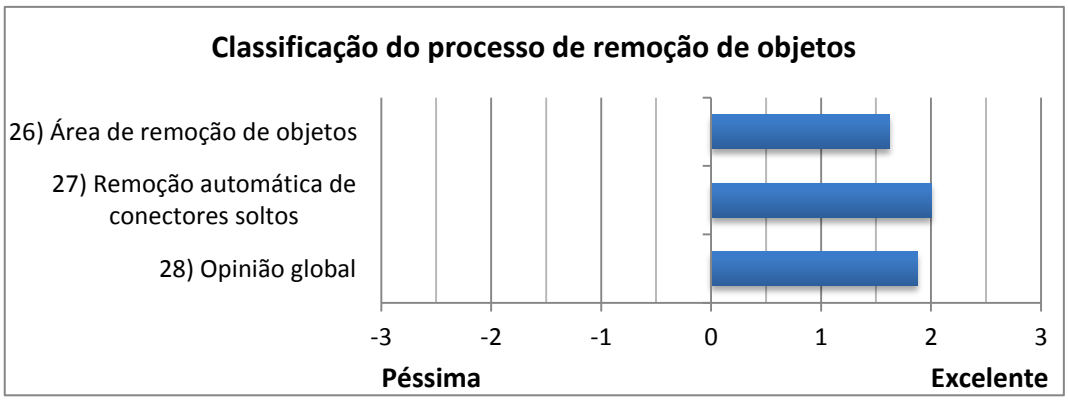



Figura 56 – Classificação média do processo de remoção de objetos

Na generalidade, os participantes consideraram o alternar de estados da simulação um aspeto útil. No entanto, alguns participantes consideraram que não deveria ocorrer a interrupção automática da simulação após o desenho de um objeto (durante a exibição da lista de expectativas), o que justifica a classificação inferior obtida na questão relacionada com essa característica, apresentada no gráfico da Figura 57.

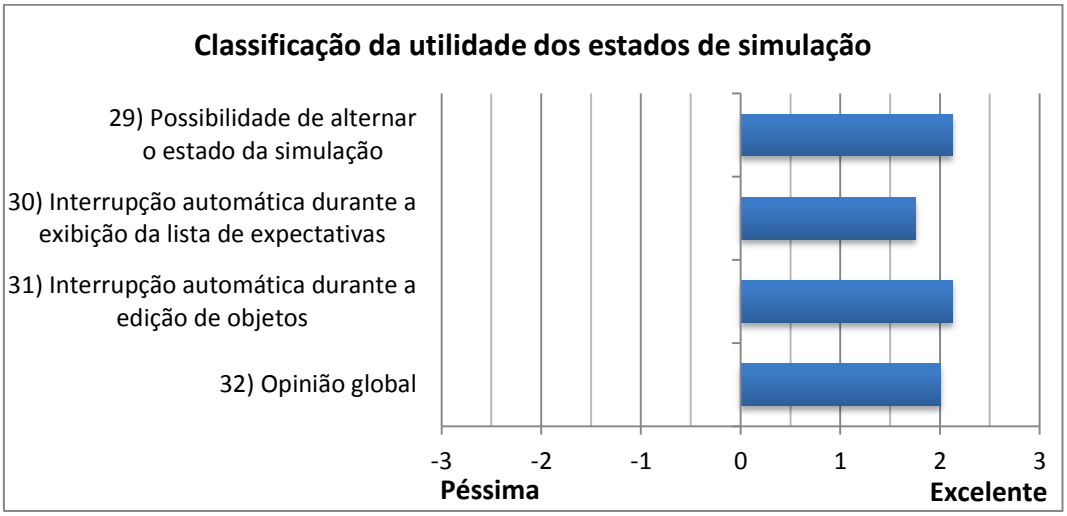



Figura 57 – Classificação média da utilidade dos estados de simulação

No gráfico da Figura 58 encontram-se ilustradas as opiniões globais dos participantes em relação às funcionalidades da biblioteca SketchyDynamics. Das classificações obtidas destaca-se a facilidade de utilização, bem como a adequação da biblioteca para a criação de ambientes fisicamente simulados, que obtiveram uma classificação igual ou superior a 2 valores. Relativamente ao estímulo, alguns dos participantes mostraram alguma frustração, em parte resultante de problemas relacionados com a utilização do equipamento com estilete e monitor interativo, que serão descritos posteriormente. No que respeita à flexibilidade, os participantes sugeriram que a biblioteca deveria suportar um número maior de tipos de objetos.

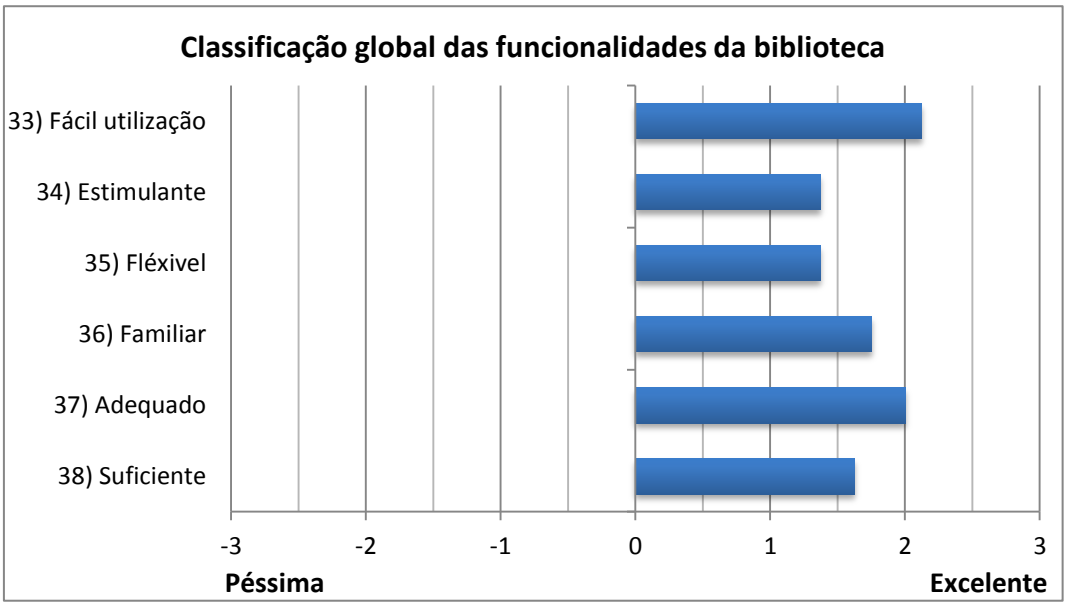



Figura 58 – Média da classificação global das funcionalidades da biblioteca

As últimas duas questões tinham como propósito conhecer a opinião dos participantes em relação à utilização das funcionalidades presentes na biblioteca SketchyDynamics como base para o desenvolvimento de aplicações. Os resultados, apesar de positivos, estão em concordância com os das questões anteriores e mostram que a biblioteca SketchyDynamics requer algumas melhorias de forma a satisfazer completamente os seus utilizadores (Figura 59).

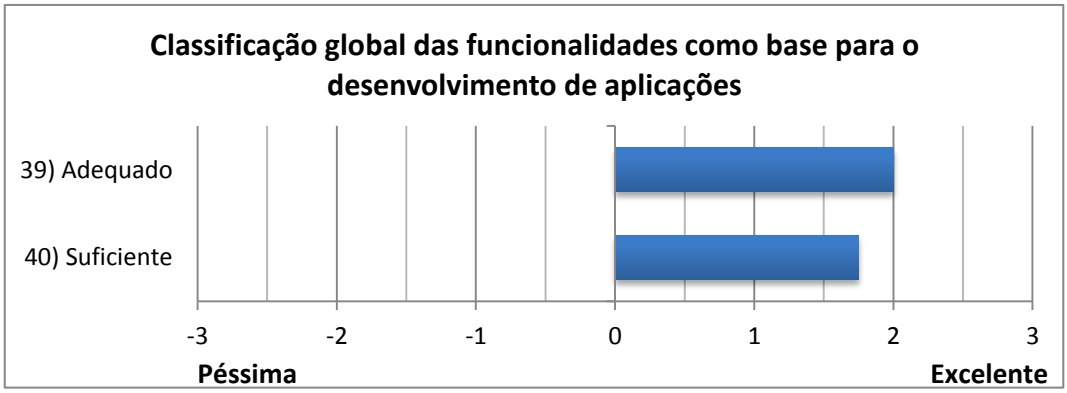



Figura 59 – Média da classificação global das funcionalidades da biblioteca SketchyDynamics como base para o desenvolvimento de aplicações

Após o preenchimento do questionário, foi solicitado aos participantes que expusessem as suas críticas e sugestões de melhoria da biblioteca SketchyDynamics. São de seguida apresentados os comentários mais pertinentes, tendo alguns destes sido partilhados por mais do que um participante, enquanto outros representam opiniões individuais de um único participante:

- Utilização de um ícone diferente e mais distintivo para a opção “anular” da lista de expectativas;
- Manter o indicador de estado de simulação visível durante a execução da simulação, exibindo um ícone de *play* (►);
- Permitir alternar o estado da simulação através do clique sobre o indicador de estado;
- Assinalar explicitamente qual o objeto que será afetado pela lista de expectativas atualmente em exibição, ou seja, aquele que foi recentemente criado;
- Implementar um mecanismo de precisão na criação de corpos rígidos, que permita o alinhamento rigoroso dos mesmos com as principais direções de desenho: horizontal, vertical e, eventualmente, diagonais de ±45º;
- Não utilizar uma seleção contínua de objetos;
- Possibilitar a realização de transformações geométricas individualmente num objeto mesmo que este se encontre ligado a outros por conectores;
- Possibilitar a deformação de um corpo rígido, depois de criado;
- Permitir a realização de cópia ou duplicação de corpos rígidos existentes (*clipboard*);
- Implementar a tecla “Delete” como método de remoção de objetos;
- Incluir um maior número de tipos de objetos;

- Permitir a realização de operações de *zoom* e *pan*, isto é, admitir ampliar determinada parte da cena, aumentando o nível de detalhe dos objetos, possibilitando a deslocação da vista ampliada ao longo da cena.

## 5.4 Conclusão

A importância da realização da avaliação de usabilidade não deve ser de forma alguma minorada, pois permitiu conhecer a opinião de potenciais utilizadores em relação ao trabalho desenvolvido. Com esta, foram identificados fatores a melhorar, funcionalidades bem-sucedidas e ainda recolhidas ideias para desenvolvimentos futuros.

É possível afirmar que os participantes se verificaram satisfeitos com a generalidade das funcionalidades apresentadas, pois todas as questões obtiveram uma classificação média superior a 1 valor, não havendo ocorrências de classificações médias negativas ou nulas.

Apesar da naturalidade associada ao paradigma de interação caligráfica, os utilizadores de aplicações informáticas estão acostumados a um formato de interação muito diferente daquele que é oferecido por uma interface caligráfica. Por esta razão, a utilização deste tipo de interface requer alguma habituação e prática para que seja possível tirar todo o partido da mesma. Além disso, os utilizadores estão a contar com a existência de um conjunto de funcionalidades presentes na maioria das aplicações, como a área de transferência ou *clipboard* para duplicação de objetos. Ao não se encontrarem disponíveis estas funcionalidades, os utilizadores podem manifestar um sentimento de rejeição na utilização do sistema.

Também o equipamento utilizado desencadeou algumas situações que dificultaram a interação. Durante a utilização por parte de alguns participantes, o monitor interativo com estilete apresentou alguns problemas na deteção do pressionar do estilete sobre o ecrã, apesar das tentativas de afinação da sensibilidade. Como resultado, os participantes sentiram alguma dificuldade e frustração na concretização de determinadas operações, sobretudo na seleção de objetos. Um ajuste mais minucioso da sensibilidade do ecrã ou a utilização de um equipamento mais recente talvez pudesse proporcionar uma melhor experiência para o participante e, consequentemente, resultados superiores.

O próprio estilete causou algum embaraço, pois este possui um botão lateral, frequentemente pressionado involuntariamente pelo participante e que inviabilizou a correta interpretação da ação do utilizador. Uma possível solução para esta situação passaria pela desativação desse botão ou pela utilização de um estilete desprovido de botão lateral.

Por fim, a realização da sessão de avaliação gerou uma satisfação adicional ao autor pois permitiu verificar que a grande maioria dos participantes considerou interessantes as funcionalidades apresentadas, tendo tecido diversos comentários sobre o desenvolvimento de aplicações com a biblioteca SketchyDynamics. Este facto, em conjunto com as críticas e

sugestões de melhoria, constituem um incentivo e estímulo para a continuação do desenvolvimento deste trabalho.

# 6 Conclusão e Perspetivas de Trabalho Futuro

O objetivo elementar deste trabalho consistiu no desenvolvimento de um sistema que facilitasse a produção de aplicações de simulação da dinâmica de corpos rígidos, cuja interação fosse realizada com recurso a uma interface caligráfica. Este sistema manifesta-se sob a forma de uma biblioteca de programação que interliga um motor de simulação de física a uma interface caligráfica, a qual permite a realização de diversas ações para alteração da realidade da simulação. Uma das preocupações fundamentais no desenho desta biblioteca foi a de agilizar a sua integração numa aplicação, direcionando o esforço do programador para outras questões mais relevantes. Deste modo, é esperado que este trabalho atue como um estímulo ao desenvolvimento de aplicações que beneficiem do paradigma papel-lápis para a concretização da interação com um ambiente simulado.

No núcleo do funcionamento de uma interface caligráfica encontra-se o seu reconhecedor caligráfico, agindo como que o seu motor. Em boa verdade, o sucesso de uma qualquer interface caligráfica está dependente do bom desempenho do seu reconhecedor. Com esta premissa em mente foi realizado um estudo extensivo aos trabalhos existentes na área de reconhecimento caligráfico e uma avaliação de três dos reconhecedores mais populares, com recurso a dados obtidos de um conjunto amplo de participantes. Esta avaliação não só permitiu definir e executar diversos aprimoramentos na implementação dos reconhecedores, como também validar e garantir a confiança na escolha do reconhecedor posteriormente integrado na biblioteca. O trabalho de análise e avaliação de reconhecedores caligráficos aqui descrito foi ainda exposto num artigo submetido e aceite para apresentação no 20º Encontro Português de Computação Gráfica (20ºEPCG) [36][20]. Esta publicação manifesta a relevância que a avaliação realizada e as melhorias sugeridas possuem perante a comunidade científica.

Relativamente à biblioteca, solução principal do trabalho aqui apresentado, foi também apresentada num artigo submetido à 8ª Conferência Internacional sobre Teoria e Aplicações

[20] Ver Anexo D.

da Computação Gráfica (GRAPP 2013) [37][21], que aguarda notificação sobre a sua possível aceitação para apresentação.

## 6.1 Objetivos Alcançados

O ponto de partida para este trabalho foi o levantamento do estado da arte no que se refere à àrea das interfaces caligráficas, com principal foco sobre reconhecedores caligráficos. Este levantamento permitiu sobretudo obter um maior conhecimento sobre essa área e sobre as várias possibilidades existentes para integração no trabalho a desenvolver.

Para além de um estudo sobre os diversos reconhecedores caligráficos existentes, pretendia-se também uma avaliação de alguns desses reconhecedores, com dados reais e focada nas características do trabalho a desenvolver. Para a sua concretização, foi produzida uma ferramenta de implementação, teste e avaliação, na qual foram implementados três reconhecedores selecionados. Com recurso a esta ferramenta, foram então conduzidas sessões para a recolha de amostras gestuais e, finalmente, realizada a avaliação dos reconhecedores. Para além de permitir encontrar qual o reconhecedor que melhor se adequava às especificidades do trabalho a desenvolver, esta avaliação permitiu ainda descobrir e potenciar melhorias ao nível da implementação dos reconhecedores avaliados.

Como tarefa central do trabalho apresentado neste documento esteve o desenvolvimento de um sistema que, aliando uma interface caligráfica a um motor de simulação física, permitisse o rápido desenvolvimento de aplicações que tirassem partido da conjugação destes dois componentes. A solução proposta consistiu numa biblioteca de programação, com a qual um programador pode incorporar na sua aplicação capacidades de simulação da física, e também uma interface caligráfica através da qual é possível manipular a simulação de uma forma natural. Esta interface caligráfica abrange por omissão um conjunto de ações e processos que possibilitam a criação de objetos simulados e a sua respetiva edição. A biblioteca disponibiliza ainda diversos componentes suplementares com o propósito de auxiliar o programador na criação das componentes gráficas da aplicação.

No final do percurso de desenvolvimento deste trabalho foi realizada uma avaliação de usabilidade da biblioteca SketchyDynamics, de modo a conhecer a opinião e perceção de possíveis utilizadores quanto às suas funcionalidades. Esta avaliação revelou resultados deveras gratificantes e proporcionou a recolha de ideias a implementar em futuras iterações do desenvolvimento da biblioteca.

[21] Ver Anexo E.

## 6.2 Limitações e Trabalho Futuro

Apesar do desígnio principal deste trabalho ter sido satisfeito com sucesso, existem ainda algumas limitações e lugar para futuras melhorias. Como referido na secção 3.5, ao nível da implementação e avaliação de sistemas de reconhecimento caligráfico, esta poderia beneficiar da utilização de novos tipos de dispositivos para aquisição de dados, do emprego de novos dados para validação das alterações realizadas aos reconhecedores, entre outras possibilidades de melhoria.

Relativamente à biblioteca implementada, apesar de conter um conjunto de funcionalidades que permite garantir o seu préstimo, estão ainda presentes algumas limitações que poderiam ser colmatadas e diversas características úteis que poderiam ser implementadas, de forma a permitir a sua adequação a um maior número de aplicações.

### 6.2.1 Maior Suporte na Criação de Objetos

Um dos pontos de partida para o aumento da funcionalidade da biblioteca SketchyDynamics é a inclusão de um maior número de objetos reconhecidos. Não obstante a flexibilidade de criação proporcionada pelo gesto de forma livre, seria útil o reconhecimento e criação de outras primitivas como elipses ou losangos. Além disso, a possibilidade de criação de formas livres que respeitem a composição do gesto que as origina sem proceder à sua simplificação, seria também uma habilidade interessante. Ainda sobre a concretização de formas livres, seria importante que as arestas dos triângulos que as constituem não fossem exibidas, apresentando apenas os limites exteriores como acontece para os restantes objetos. Uma possível solução passaria pela utilização dos métodos de tesselação do OpenGL para a triangulação dos polígonos, em vez da biblioteca de triangulação utilizada. Estes métodos, para além de realizarem a triangulação de polígonos, descriminam, de entre todo o conjunto de vértices resultante, aqueles que são internos ao polígono original. Desta forma, seria apenas necessário não exibir as arestas formadas por esses vértices.

Existem também algumas alterações que poderiam ser conduzidas ao nível dos conectores, como a inclusão de novos tipos ou o acrescento de funcionalidades aos já existentes. Uma adição interessante seria um conector do tipo corda que atuasse como uma corda real, tanto ao nível do seu aspeto como no seu comportamento. Exemplificando a sua utilidade, com este tipo de conector seria possível a criação de sistemas como o apresentado na Figura 60, na qual o objeto triangular se encontra ligado ao quadrado através de um conector do tipo corda e de uma roldana. Uma outra funcionalidade útil, que aumentaria a versatilidade da biblioteca, seria a definição da elasticidade do conector de mola, com base no número de ziguezagues do gesto que o instancia. Desta forma, um conector de mola seria mais elástico quanto maior fosse o número de ziguezagues existentes no gesto que o originou.

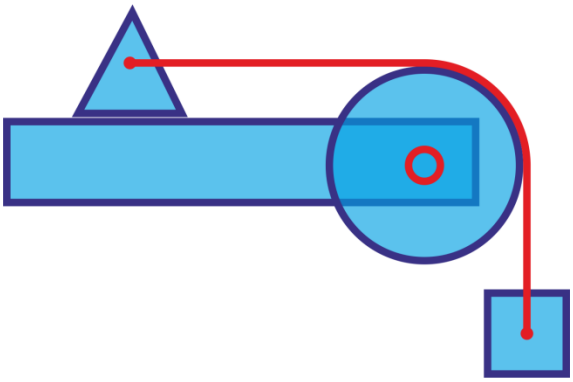


Figura 60 – Exemplo de sistema de roldana com um conector do tipo corda

Uma ideia que surgiu durante o processo de planeamento da biblioteca SketchyDynamics foi a criação de um mecanismo que permitisse a alteração de propriedades dos objetos, como a massa ou elasticidade dos corpos, o grau de elasticidade de um conector de mola ou até a rigidez de um conector de rotação. Este mecanismo permitiria ainda a definição de um conector de rotação como sendo um motor, com uma força de rotação própria. A concretização deste mecanismo passaria pela exibição de um pequeno "balão" informativo perto do objeto selecionado, onde estariam disponíveis as propriedades referentes a esse objeto. A Figura 61 mostra um esboço da idealização deste mecanismo, não tendo no entanto sido implementado.

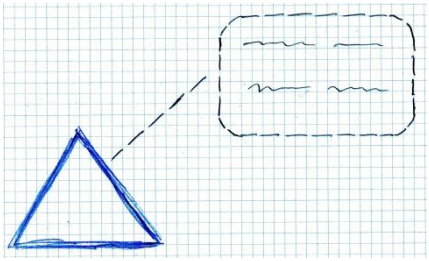


Figura 61 – Esboço do mecanismo de definição de propriedades de objetos

### 6.2.2 Aumento da Eficiência das Operações de Transformação de Objetos

No que se refere às operações de transformação de objetos, a concretização simultânea de escala e rotação permite agilizar o processo de transformação e reduzir o número de comandos existentes no sistema e, consequentemente, a carga cognitiva imposta ao utilizador. No entanto, esta característica introduz algumas dificuldades ao impossibilitar a realização de uma operação sem que a outra seja espoletada. Uma possível solução para este problema seria a definição de uma tecla modificadora ou a utilização de um segundo botão do rato para restringir a ação a uma única operação. Sempre que o utilizador pressionasse esta tecla, o sistema determinaria se o movimento do cursor era predominantemente radial ou tangencial, executando apenas a operação de escala ou a de rotação, respetivamente. Esta tecla modificadora poderia ainda ser aplicada para restringir a translação de objetos a linhas horizontais, verticais ou diagonais de 45 graus.

Em relação ao mecanismo implementado para a seleção de múltiplos objetos, apesar de funcional, este entra em conflito com os padrões de interação utilizados pela maioria das aplicações informáticas. Ao contrário do que sucede nestas aplicações, na biblioteca SketchyDynamics a seleção de um novo objeto é um processo aditivo, pois a seleção de um novo objeto não desseleciona os objetos previamente selecionados. De forma a resolver esta questão, o pressionar do estilete sobre um objeto deveria selecionar esse objeto mas desselecionar qualquer objeto que se encontre selecionado. No entanto, esta situação levanta um novo problema: como selecionar múltiplos objetos para que possam ser transformados em simultâneo? Uma possível solução passaria pelo recurso a uma tecla modificadora que, quando pressionada, tornaria a operação de seleção num processo aditivo. Uma outra alternativa, que não envolve a utilização do teclado, passaria por definir o duplo toque para a adição à seleção. Desta forma, caso o utilizador realizasse um toque sobre um objeto, este seria selecionado e os restantes desselecionados, mas caso fossem realizados dois toques consecutivos, os restantes objetos manter-se-iam selecionados, em conjunto com o objeto que foi pressionado.

Durante a experimentação das funcionalidades da biblioteca, e também em resultado da avaliação de usabilidade, foi evidente que a possibilidade de seleção de apenas um corpo, de um conjunto de corpos ligados, e consequente realização de transformações tendo como ponto de referência o conector do corpo, em oposição ao seu centro geométrico, seria de extrema utilidade. Apesar de se revelar uma funcionalidade com potencial, nomeadamente para a rotação de um objeto em torno do seu conector, esta apresenta alguns problemas de conceção, pois um único corpo pode possuir diversos conectores, sendo difícil definir qual deve ser utilizado como ponto de referência. Este problema intensifica-se no caso de existirem vários objetos selecionados em simultâneo: deverá cada objeto ser transformado com base no seu conector, ou de acordo com o centro geométrico médio dos vários objetos selecionados? Estas questões exigem um estudo cuidado e minucioso de como as resolver, antes de ser possível proceder à implementação definitiva desta funcionalidade.

Seria também interessante o suporte a hierarquias de objetos, nas quais as transformações aplicadas a objetos de um nível hierárquico superior seriam herdadas pelos objetos de níveis inferiores, mas não o oposto. A construção da hierarquia poderia ser definida a partir da profundidade no ecrã de cada objeto, que resulta da ordem com que estes são desenhados.

### 6.2.3 Melhoria da Interação Durante a Execução da Simulação

Apesar de a criação de objetos, sejam estes corpos ou conectores, ser exequível enquanto a simulação decorre, há um conjunto de operações que veem a sua funcionalidade limitada. É o caso da criação de conectores de rotação ou âncora entre dois corpos, pois não há forma de o utilizador sobrepor os dois corpos para de seguida criar o conector entre eles enquanto a simulação decorre, a não ser que um dos corpos se trate do fundo da aplicação. Uma possível solução passaria pela possibilidade de criar conectores em avanço, ou seja, antes do segundo corpo estar definido. O utilizador criaria então o primeiro corpo e de seguida colocaria o

conector nesse corpo, mesmo sem existir ainda um segundo corpo. Quando um segundo corpo fosse finalmente desenhado sobre o conector, esse corpo passaria a estar ligado ao corpo que possuía esse conector. Esta abordagem acarreta no entanto a impossibilidade de existência de um corpo de fundo da aplicação, pois seria difícil determinar se o utilizador pretende criar um conector em avanço, ou ligar o corpo ao fundo.

Como foi referido anteriormente, existem ações de interação que causam a interrupção automática da simulação, como a edição de objetos ou a exibição da lista de expetativas. Apesar de este ser um atributo desejado na generalidade das situações, a execução dessas operações sem interrupção da simulação poderia ser um requisito para algumas aplicações, pelo que a sua implementação seria importante.

### 6.2.4 Suporte a Operações Comuns de Edição

Existem algumas operações comuns na grande maioria das aplicações informáticas que realizam algum tipo de edição, seja ela de texto, de imagens ou de som. De entre estas operações encontra-se a possibilidade de Cortar/Copiar/Colar (ou duplicar) objetos e de Anular/Refazer ações.

Um trabalho futuro poderia constituir-se na integração das operações de Cortar/Copiar/Colar e Anular/Refazer na biblioteca SketchyDynamics, pois enquanto a primeira agiliza bastante o processo de criação, a segunda dá ao utilizador uma liberdade de criação superior, pois permite facilmente a correção de possíveis erros. Ao ter conhecimento de que é possível desfazer uma ação indesejada, o utilizador sentir-se-á mais à-vontade para explorar o sistema e arriscar durante o processo criativo.

### 6.2.5 Aplicações de Demonstração de Aplicabilidade

De entre os diversos objetivos iniciais para este trabalho encontrava-se a intenção de desenvolver um conjunto de diferentes aplicações que recorressem à biblioteca SketchyDynamics, para mostrar a sua versatilidade e capacidades. No entanto, por questões temporais, apenas foi possível o desenvolvimento de um pequeno protótipo que expõe as funcionalidades da biblioteca, utilizada para o seu teste durante o desenvolvimento e também para a realização da avaliação de usabilidade, mas que não possui qualquer objetivo ou utilização que não aquele para que foi criado.

Futuramente seria importante e prioritário o desenvolvimento de tais aplicações, no sentido de expor o potencial da biblioteca e, por conseguinte, incentivar outros programadores a utilizá-la.

### 6.2.6 Manuais de utilizador e de referências de programação

Um elemento importante na adoção de qualquer biblioteca de programação é a existência de uma boa documentação. Neste sentido, a criação de um manual de referência de programação da biblioteca, no qual sejam detalhados o objetivo e utilização de todas as suas estruturas, classes e funções disponíveis ao programador, revela-se uma tarefa de elevada importância e prioridade.

Além do manual de referências, é também importante a criação de um manual de utilizador. Esta necessidade deve-se à existência de diversas características de interação, utilizadas pelo utilizador para a manipulação da simulação, que se encontram incluídas na biblioteca.

## 6.3 Apreciação Final

A nível pessoal, o trabalho aqui apresentado representou uma experiência muito gratificante pois permitiu um elevado adquirir de conhecimentos. Grande parte destes conhecimentos fora conseguida durante o levantamento do estado da arte em relação a sistemas de reconhecimento caligráfico, pois permitiu-me conhecer os conceitos subjacentes ao funcionamento desse tipo de sistemas, com a qual a minha experiência anterior era nula.

Também o desenvolvimento da biblioteca fomentou a indagação e consequente aprendizagem de melhores práticas no que se refere ao desenvolvimento de componentes de *software* para utilização por terceiros, como é o caso de uma biblioteca de programação.

Confesso ainda que me traria extrema satisfação continuar a desenvolver este trabalho, e prevejo fazê-lo, principiando pela implementação das funcionalidades apresentadas na secção de trabalho futuro. Espero brevemente desenvolver algumas aplicações com a biblioteca, quer para entretenimento quer para o ensino, e também realizar a sua publicação *online*, num website dedicado à mesma, para que possa ser empregada por outros programadores no desenvolvimento de aplicações.

Resta-me apenas manifestar o desejo de, um dia, observar o surgimento de outros trabalhos concebidos com recurso à biblioteca SketchyDynamics, cumprindo assim o propósito com que foi criada.

# Referências


[1] Microsoft Corporation. (n.a.) Write it out: Using Tablet PC Input Panel instead of your keyboard. [Online]. http://windows.microsoft.com/en-CA/windows-vista/Write-it-out-Using-Tablet-PC-Input-Panel-instead-of-your-keyboard [Acedido em 20 de Setembro, 2012]

[2] João P. Pereira, Joaquim A. Jorge, Vasco Branco, and F. Nunes Ferreira, "GIDeS: Uma Abordagem Caligráfica à Edição 3D", *9º Encontro Português de Computação Gráfica*, pp. 101-108, 2000.

[3] Opera Software. (n.a.) Opera Tutorials - Mouse gestures in Opera. [Online]. http://www.opera.com/browser/tutorials/gestures/ [Acedido em 20 de Setembro, 2012]

[4] Dean Rubine, "Specifying Gestures by Example", *SIGGRAPH Computer Graphics, Volume 25 Issue 4*, pp. 329 -337, Julho 1991.

[5] Beryl Plimmer and Isaac Freeman, "A toolkit approach to sketched diagram recognition", *Proceedings of the 21st British HCI Group Annual Conference on People and Computers: HCI.but not as we know it (BCS-HCI '07)*, vol. 1, pp. 205-213, 2007.

[6] J. P. Pereira et al., "Cascading recognizers for ambiguous calligraphic interaction", *Eurographics Workshop on Sketch-Based Interfaces and Modeling*, 2004.

[7] Thomas F. Stahovich, "Pen-based Interfaces for Engineering and Education", *Sketch-based Interfaces and Modeling*, pp. 119-152, 2011.

[8] M. J. Fonseca, C. Pimentel, and J. A. Jorge, "CALI: An online scribble recognizer for calligraphic interfaces", *AAAI Spring Symposium on Sketch Understanding*, pp. 51-58, 2002.

[9] Jacob O. Wobbrock, Andrew D. Wilson, and Yang Li, "Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes", *Proceedings of the 20th annual ACM symposium on User interface software and technology (UIST '07)*, pp. 159-168, 2007.

[10] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery, *Numerical Recipes in C: The Art of Scientific Computing*, 2nd ed. New York: Cambridge University Press, 1992.

[11] Lisa Anthony and Jacob O. Wobbrock, "A lightweight multistroke recognizer for user interface prototypes", *Proceedings of Graphics Interface 2010 (GI '10)*, pp. 245-252, 2010.

[12] WeeSan Lee, Levent Burak Kara, and Thomas F. Stahovich, "An efficient graph-based recognizer for hand-drawn symbols", *Computers & Graphics 31*, pp. 554-567, 2007.

[13] Thomas F. Stahovich, "Segmentation of pen strokes using pen speed", *Proceedings of 2004 AAAI fall symposium on making pen-based in-teraction intelligent and natural*, pp. 152-158, 2004.

[14] Radu-Daniel Vatavu, Laurent Grisoni, and Stefan-Gheorghe Pentiuc, "Gesture Recognition Based on Elastic Deformation Energies", *Gesture-Based Human-Computer Interaction and Simulation*, vol. 5085, pp. 1-12., 2009.

[15] Tevfik Metin Sezgin and Randall Davis, "HMM-based efficient sketch recognition", *Proceedings of the 10th international conference on Intelligent user interfaces (IUI '05)*, pp. 281-283, 2005.

[16] Tevfik Metin Sezgin, Thomas Stahovich, and Randall Davis, "Sketch based interfaces: early processing for sketch understanding", *Proceedings of the 2001 workshop on Perceptive user interfaces (PUI '01)*, 2001.

[17] Brandon Paulson and Tracy Hammond, "PaleoSketch: accurate primitive sketch recognition and beautification", *Proceedings of the 13th international conference on Intelligent user interfaces (IUI '08)*, pp. 1-10, 2008.

[18] Paul Schmieder, Beryl Plimmer, and Rachel Blagojevic, "Automatic evaluation of sketch recognizers", *Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling (SBIM '09)*, pp. 85-92, 2009.

[19] Christine Alvarado and Randall Davis, "Resolving Ambiguities to Create a Natural Computer-Based Sketching", *Proceedings of IJCAI-2001*, pp. 1365-1371, 2001.

[20] Haidi Kamel, Madgy W. Shonoda, Mariam Refeet, and Ramy Nabil. (n.d.) Free-Hand Sketch Recognition For Visualizing Interactive Physics. [Online]. http://code.google.com/p/sketch-recognition-simulation-tool [Acedido em 26 de Julho, 2012]

[21] Petri Purho. (2009) Crayon Physics Deluxe. [Online]. http://crayonphysics.com [Acedido em 27 de Julho, 2012]

[22] Camel Games. (9 de Fevereiro, 2011) Space Physics - Google Play. [Online]. https://play.google.com/store/apps/details?id=com.doodle.main [Acedido em 24 de Agosto, 2012]

[23] Andy Beaulieu. (5 de Março, 2012) Create Games for Windows 8 using Physamajig! [Online]. http://www.andybeaulieu.com/Home/tabid/67/EntryID/225/Default.aspx

[Acedido em 24 de Agosto, 2012]

[24] SpriteHand. (n.d.) Physamajig Help. [Online]. http://www.spritehand.com/PhysamajigHelp/tabid/246/Default.aspx [Acedido em 24 de Agosto, 2012]

[25] B. W. Boehm, "A Spiral Model of Software Development and Enhancement", *IEEE Computer*, vol. 21, no. 5, pp. 61-72, Março 1988.

[26] Deborah Hix and H. Rex Hartson, *Developing User Interfaces: Ensuring Usability Through Product & Process*.: John Wiley & Sons, Inc., 1993.

[27] Digia. (n.d.) Qt. [Online]. http://qt.digia.com/ [Acedido em 4 de Outubro, 2012]

[28] Microsoft Corporation. (n.d.) The Official Microsoft WPF and Windows Forms Site. [Online]. http://windowsclient.net/ [Acedido em 4 de Outubro, 2012]

[29] Erich Gamma, Richard Helm, Ralph Johnson, and John Vissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. USA: Addison-Wesley, 1994.

[30] Craig Selbert. (Maio, 2007) A C# Implementation of Douglas-Peucker Line Approximation Algorithm. [Online]. http://www.codeproject.com/Articles/18936/A-C-Implementation-of-Douglas-Peucker-Line-Approxi [Acedido em 20 de Agosto, 2012]

[31] Erin Catto. (n.d.) Box2D. [Online]. http://box2d.org [Acedido em 25 de Julho, 2012]

[32] Mason Green and Thomas Åhlén. (n.d.) poly2tri - A 2D constrained Delaunay triangulation library. [Online]. http://code.google.com/p/poly2tri/ [Acedido em 10 de Setembro, 2012]

[33] Robert A. Virzi, "Refining the test phase of usability evaluation: How many subjects is enough?", *Human Factors*, vol. 34, no. 4, pp. 457-468, 1992.

[34] Jakob Nielsen, *Usability Engineering*.: Morgan Kaufmann, 1993.

[35] João P. Pereira, "Interacção caligráfica ambígua em sistemas computacionais de modelação", Faculdade de Engenharia da Universidade do Porto, Tese de Doutoramento em Engenharia Electrotécnica e de Computadores, 2004.

[36] Abílio Costa and João P. Pereira, "SketchTester: Analysis and Evaluation of Calligraphic Gesture Recognizers ", *20º Encontro Português de Computação Gráfica (20ºEPCG)*, 2012 (por publicar).

[37] Abílio Costa and João P. Pereira, "SketchyDynamics - A Sketch-Based Library for the Development of Physics Simulation Applications", *GRAPP 2013*, 2012 (submetido para

publicação).

[38] J. D. Gould and C. Lewis, "Designing for Usability: Key Principles and What Designers Think", *Communicotions of the ACM*, vol. 28, no. 3, pp. 300-311, 1985.

# Anexo A

## Guião da sessão de recolha de amostras caligráficas

Instituto Superior de Engenharia do Porto
DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

**Mestrado em Engenharia Informática**

# Sessão de recolha de amostras para análises de reconhecimento caligráfico

Abílio Costa

Orientador: Prof. João Paulo Pereira

Março de 2012

# Introdução

A presente sessão está enquadrada na dissertação “Interface Caligráfica de Apoio à Produção de Sistemas Baseados na Simulação da Dinâmica dos Corpos Rígidos”, que se encontra a ser realizada no âmbito do curso de Mestrado em Engenharia Informática do Instituto Superior de Engenharia do Porto. O foco desta dissertação é o estudo e desenvolvimento de uma biblioteca que disponibilize, para além de um motor de simulação de física, uma interface caligráfica para manipular a simulação. Assim, é facilitado o desenvolvimento de aplicações de simulação de física que tirem partido do paradigma do desenho e da metáfora do “papel – lápis”.

Dado o importante papel que a componente de reconhecimento caligráfico desempenha no projeto, ao constituir as funcionalidades de interação com o utilizador, é necessário realizar um estudo, testes e avaliação intensivos de vários métodos de reconhecimento existentes, a fim de identificar qual o que melhor se enquadra nas características previstas para o projeto. Neste processo foram implementados três algoritmos de reconhecimento caligráfico: o reconhecedor de 1$, o algoritmo de Rubine e o CALI. Posteriormente, um destes três reconhecedores será incorporado na biblioteca a desenvolver.

Esta sessão tem como objetivo a recolha de símbolos desenhados por um grupo de participantes, com recurso a uma aplicação desenvolvida para esse fim. Estas amostras serão posteriormente utilizadas para realizar uma análise das taxas de reconhecimento de cada um dos algoritmos mencionados, a fim de contribuir para uma melhor seleção do algoritmo a utilizar.

Muito obrigado pela colaboração.

# Planeamento

Na Tabela 1 é apresentado o planeamento da sessão de recolha de amostras, contendo a descrição de cada fase e respetiva duração prevista.

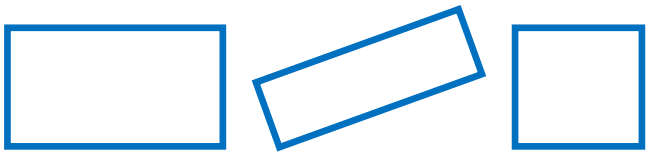
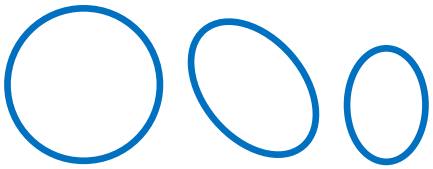
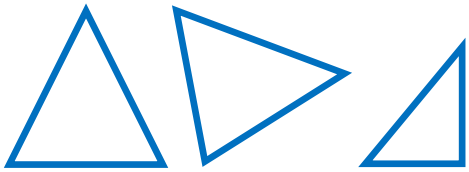
***Tabela 1 - Planeamento da sessão***

| Fase | Duração |
|---|---|
| **Apresentação da sessão** | 10 Minutos |
| **Período de familiarização com a aplicação** | 10 Minutos |
| **Período de recolha de amostras** | 10 Minutos |

# Símbolos a amostrar

Os cinco símbolos a serem recolhidos, apresentados na Tabela 2, foram selecionados de acordo com as características previstas para a biblioteca a desenvolver. É importante que as amostras desenhadas para cada símbolo variem no que respeita à sua orientação, dimensão e proporção.

***Tabela 2 - Símbolos a amostrar***

| Forma | Ilustração |
|---|---|
| **Retângulo** |  |
| **Circulo** |  |
| **Triângulo** |  |
| **Alfa** |  |
| **Ziguezague** |  |

# Descarga da aplicação de recolha de amostras

Como já foi referido, a recolha das amostras será realizada com recurso a uma aplicação desenvolvida para esse efeito e que se encontra disponível *online* para descarga. De forma a obter e executar a aplicação, cada participante deve realizar as seguintes tarefas:

1. Fazer o download do arquivo "zip" que contém a aplicação a partir do endereço http://db.tt/Nd7nY2Xc ;
2. Extrair o arquivo "zip", que deverá dar origem à pasta "SketchTester";
3. Dentro da pasta "SketchTester", executar o ficheiro "SketchTester.exe".

# Descrição da aplicação

São descritos a seguir os vários componentes da aplicação de recolha de amostras:

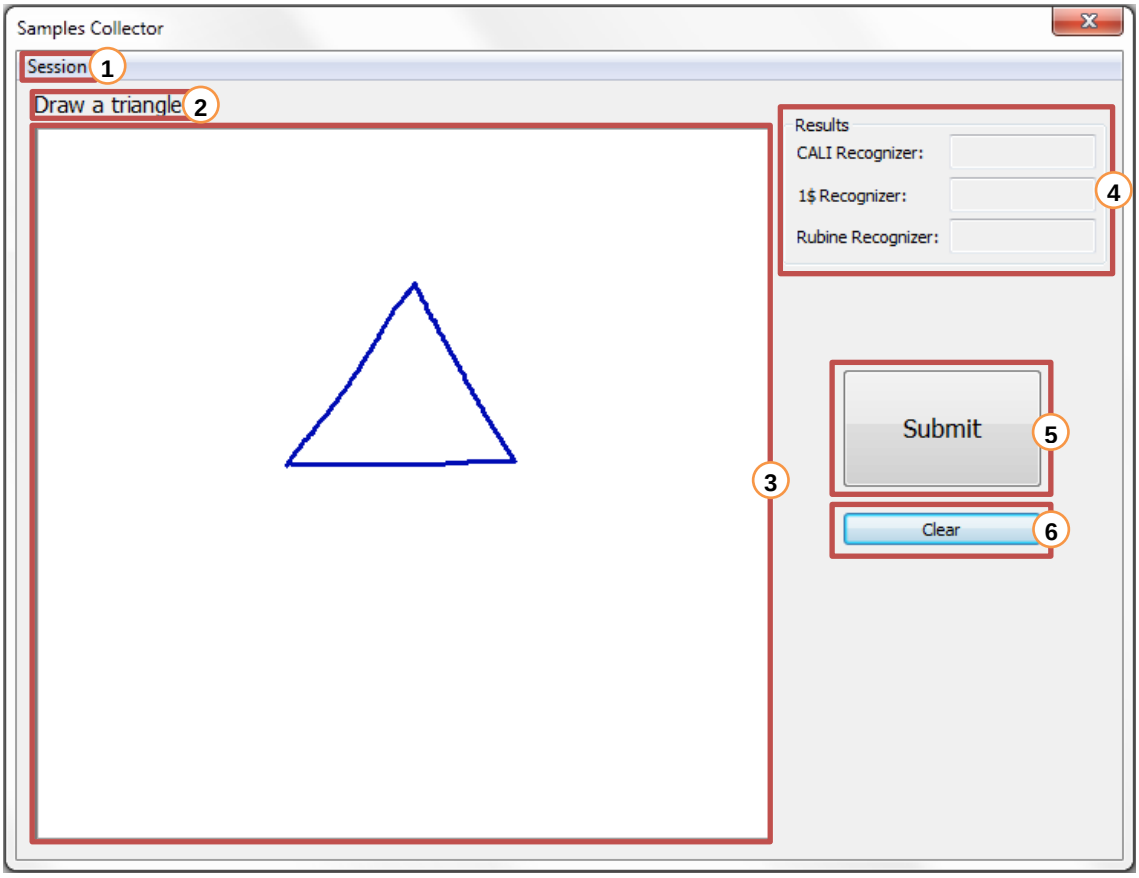



***Figura 1 - Descrição da aplicação de recolha de amostras***

1. Menu da aplicação, onde se encontra o botão “Restart”, que permite reiniciar a sessão de recolha;
2. Indicador do símbolo a desenhar;
3. Zona de desenho;
4. Resultados de reconhecimento do símbolo submetido anteriormente;
5. Botão de submissão do símbolo desenhado;
6. Botão de limpeza da área de desenho (não submete o símbolo desenhado).

# Processo de recolha de amostras

No início da sessão haverá um pequeno período de tempo no qual os participantes poderão testar a aplicação de recolha, a fim de se familiarizarem com a mesma e terem a oportunidade de esclarecer possíveis dúvidas. De seguida cada participante deverá desenhar dez amostras de cada um dos cinco símbolos, realizando assim um total de cinquenta amostras. A cada instante, o símbolo a desenhar pelo participante será indicado de forma aleatória pela aplicação.

Após cada símbolo ser desenhado é dada ao participante a possibilidade de redesenhar o mesmo símbolo ou de submetê-lo como amostra. Ao submeter um símbolo desenhado é apresentada qual a interpretação computada por cada um dos três algoritmos de reconhecimento e o símbolo é armazenado para posterior análise.

No final do processo de recolha, a aplicação exibe as taxas de reconhecimento para cada um dos três algoritmos, relativamente à sessão decorrida.

Resumidamente, após a fase de familiarização com a aplicação, cada participante deverá realizar os seguintes passos:

1. No menu “Session”, selecionar a opção “Restart”;
2. Desenhar uma amostra do símbolo requerido pela aplicação;
3. Carregar no botão “Submit” para submeter o símbolo desenhado ou, no caso de o símbolo não estar de acordo com o desejado, carregar no botão “Clear” para limpar e voltar a desenhar o mesmo símbolo;
4. Repetir os passos 2 e 3 enquanto forem solicitadas amostras pela aplicação.

# Submissão do ficheiro de resultados

No final da sessão a aplicação irá criar, dentro da pasta onde se encontra, um ficheiro que contém os dados relativos às amostras recolhidas. Este ficheiro terá o nome "RESULT_XXXX_YYYY.dat", em que "XXXX" corresponde ao nome da máquina onde foi criado e "YYYY" representa a data e hora na qual a aplicação foi iniciada. Cada participante deve submeter o ficheiro de resultados criado, realizando os seguintes passos:

1. Aceder ao endereço http://bit.ly/whdp6b ;
2. Carregar no botão "Escolher ficheiro" ou "Procurar…" (dependendo do *browser*);
3. Na janela de seleção de ficheiro, localizar e selecionar o ficheiro de resultado criado pela aplicação de recolha de amostras;
4. Carregar no botão "Submit";
5. Após submeter o ficheiro, deverá ser exibida a mensagem "File successfully uploaded!".

# Anexo B

## Tabelas de resultados de reconhecimento das amostras caligráficas

Tabela Anexo B - 1 – Resultados de reconhecimento das amostras caligráficas de cada participante, com o reconhecedor CALI, antes do processo de filtragem

| | **CALI – Resultados iniciais (antes da filtragem)** | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Retângulo** | | | **Triângulo** | | | **Círculo** | | | **Alfa** | | | **Ziguezague** | | |
| **Partic.** | ☺ | ☹ | **☺%** | ☺ | ☹ | **☺%** | ☺ | ☹ | **☺%** | ☺ | ☹ | **☺%** | ☺ | ☹ | **☺%** |
| 1 | 10 | 0 | 100 | 8 | 2 | 80 | 10 | 0 | 100 | 4 | 6 | 40 | 10 | 0 | 100 |
| 2 | 8 | 2 | 80 | 9 | 1 | 90 | 9 | 1 | 90 | 10 | 0 | 100 | 2 | 8 | 20 |
| 3 | 5 | 0 | 100 | 5 | 0 | 100 | 7 | 0 | 100 | 5 | 1 | 83 | 6 | 1 | 86 |
| 4 | 10 | 0 | 100 | 9 | 1 | 90 | 9 | 1 | 90 | 3 | 7 | 30 | 8 | 2 | 80 |
| 5 | 10 | 0 | 100 | 10 | 0 | 100 | 10 | 0 | 100 | 5 | 5 | 50 | 2 | 8 | 20 |
| 6 | 10 | 0 | 100 | 10 | 0 | 100 | 10 | 0 | 100 | 8 | 2 | 80 | 10 | 0 | 100 |
| 7 | 9 | 1 | 90 | 10 | 0 | 100 | 10 | 0 | 100 | 9 | 1 | 90 | 5 | 5 | 50 |
| 8 | 9 | 1 | 90 | 10 | 0 | 100 | 9 | 1 | 90 | 3 | 7 | 30 | 5 | 5 | 50 |
| 9 | 10 | 0 | 100 | 8 | 2 | 80 | 8 | 2 | 80 | 10 | 0 | 100 | 9 | 1 | 90 |
| 10 | 10 | 0 | 100 | 6 | 4 | 60 | 9 | 1 | 90 | 7 | 3 | 70 | 5 | 5 | 50 |
| 11 | 10 | 0 | 100 | 9 | 1 | 90 | 10 | 0 | 100 | 8 | 2 | 80 | 5 | 5 | 50 |
| 12 | 10 | 0 | 100 | 10 | 0 | 100 | 10 | 0 | 100 | 8 | 2 | 80 | 6 | 4 | 60 |
| 13 | 9 | 1 | 90 | 7 | 3 | 70 | 6 | 4 | 60 | 10 | 0 | 100 | 0 | 10 | 0 |
| 14 | 10 | 0 | 100 | 10 | 0 | 100 | 9 | 1 | 90 | 6 | 4 | 60 | 4 | 6 | 40 |
| 15 | 10 | 0 | 100 | 10 | 0 | 100 | 10 | 0 | 100 | 10 | 0 | 100 | 5 | 5 | 50 |
| 16 | 8 | 2 | 80 | 7 | 3 | 70 | 9 | 1 | 90 | 8 | 2 | 80 | 2 | 8 | 20 |
| 17 | 9 | 1 | 90 | 8 | 2 | 80 | 10 | 0 | 100 | 9 | 1 | 90 | 3 | 7 | 30 |
| 18 | 10 | 0 | 100 | 10 | 0 | 100 | 9 | 1 | 90 | 4 | 6 | 40 | 0 | 10 | 0 |
| 19 | 2 | 8 | 20 | 1 | 9 | 10 | 1 | 9 | 10 | 1 | 9 | 10 | 0 | 10 | 0 |
| 20 | 10 | 0 | 100 | 10 | 0 | 100 | 9 | 1 | 90 | 8 | 2 | 80 | 7 | 3 | 70 |
| 21 | 10 | 0 | 100 | 9 | 1 | 90 | 10 | 0 | 100 | 9 | 1 | 90 | 8 | 2 | 80 |
| 22 | 9 | 1 | 90 | 10 | 0 | 100 | 10 | 0 | 100 | 5 | 5 | 50 | 5 | 5 | 50 |
| 23 | 6 | 4 | 60 | 6 | 4 | 60 | 9 | 1 | 90 | 9 | 1 | 90 | 7 | 3 | 70 |
| 24 | 7 | 3 | 70 | 9 | 1 | 90 | 8 | 2 | 80 | 9 | 1 | 90 | 5 | 5 | 50 |
| 25 | 9 | 1 | 90 | 10 | 0 | 100 | 9 | 1 | 90 | 7 | 3 | 70 | 8 | 2 | 80 |
| 26 | 4 | 6 | 40 | 6 | 4 | 60 | 6 | 4 | 60 | 6 | 4 | 60 | 6 | 4 | 60 |
| 27 | 10 | 0 | 100 | 10 | 0 | 100 | 10 | 0 | 100 | 8 | 2 | 80 | 9 | 1 | 90 |
| 28 | 9 | 1 | 90 | 10 | 0 | 100 | 10 | 0 | 100 | 9 | 1 | 90 | 6 | 4 | 60 |
| 29 | 8 | 2 | 80 | 8 | 2 | 80 | 5 | 5 | 50 | 7 | 3 | 70 | 7 | 3 | 70 |
| 30 | 5 | 1 | 83 | 3 | 0 | 100 | 3 | 0 | 100 | 1 | 3 | 25 | 3 | 1 | 75 |
| 31 | 7 | 3 | 70 | 10 | 0 | 100 | 8 | 2 | 80 | 4 | 6 | 40 | 9 | 1 | 90 |
| 32 | 10 | 0 | 100 | 10 | 0 | 100 | 8 | 2 | 80 | 9 | 1 | 90 | 1 | 9 | 10 |
| Total | 273 | 38 | **88** | 268 | 40 | **87** | 270 | 40 | **87** | 219 | 91 | **71** | 168 | 143 | **54** |

Tabela Anexo B - 2 – Resultados de reconhecimento das amostras caligráficas de cada participante, com o reconhecedor $1, antes do processo de filtragem

| | **$1 – Resultados iniciais (antes da filtragem)** | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Retângulo** | | | **Triângulo** | | | **Círculo** | | | **Alfa** | | | **Ziguezague** | | |
| **Partic.** | ☺ | ☹ | **☺%** | ☺ | ☹ | **☺%** | ☺ | ☹ | **☺%** | ☺ | ☹ | **☺%** | ☺ | ☹ | **☺%** |
| 1 | 3 | 7 | 30 | 10 | 0 | 100 | 10 | 0 | 100 | 7 | 3 | 70 | 10 | 0 | 100 |
| 2 | 5 | 5 | 50 | 6 | 4 | 60 | 10 | 0 | 100 | 9 | 1 | 90 | 10 | 0 | 100 |
| 3 | 3 | 2 | 60 | 2 | 3 | 40 | 7 | 0 | 100 | 5 | 1 | 83 | 7 | 0 | 100 |
| 4 | 3 | 7 | 30 | 6 | 4 | 60 | 9 | 1 | 90 | 7 | 3 | 70 | 10 | 0 | 100 |
| 5 | 7 | 3 | 70 | 6 | 4 | 60 | 10 | 0 | 100 | 7 | 3 | 70 | 10 | 0 | 100 |
| 6 | 3 | 7 | 30 | 9 | 1 | 90 | 10 | 0 | 100 | 9 | 1 | 90 | 9 | 1 | 90 |
| 7 | 5 | 5 | 50 | 2 | 8 | 20 | 10 | 0 | 100 | 9 | 1 | 90 | 10 | 0 | 100 |
| 8 | 6 | 4 | 60 | 10 | 0 | 100 | 9 | 1 | 90 | 9 | 1 | 90 | 10 | 0 | 100 |
| 9 | 7 | 3 | 70 | 4 | 6 | 40 | 9 | 1 | 90 | 9 | 1 | 90 | 9 | 1 | 90 |
| 10 | 9 | 1 | 90 | 6 | 4 | 60 | 8 | 2 | 80 | 8 | 2 | 80 | 8 | 2 | 80 |
| 11 | 5 | 5 | 50 | 8 | 2 | 80 | 10 | 0 | 100 | 10 | 0 | 100 | 7 | 3 | 70 |
| 12 | 6 | 4 | 60 | 10 | 0 | 100 | 10 | 0 | 100 | 9 | 1 | 90 | 10 | 0 | 100 |
| 13 | 9 | 1 | 90 | 8 | 2 | 80 | 7 | 3 | 70 | 10 | 0 | 100 | 10 | 0 | 100 |
| 14 | 10 | 0 | 100 | 5 | 5 | 50 | 10 | 0 | 100 | 10 | 0 | 100 | 10 | 0 | 100 |
| 15 | 8 | 2 | 80 | 7 | 3 | 70 | 10 | 0 | 100 | 10 | 0 | 100 | 8 | 2 | 80 |
| 16 | 6 | 4 | 60 | 7 | 3 | 70 | 9 | 1 | 90 | 10 | 0 | 100 | 6 | 4 | 60 |
| 17 | 6 | 4 | 60 | 9 | 1 | 90 | 9 | 1 | 90 | 10 | 0 | 100 | 8 | 2 | 80 |
| 18 | 4 | 6 | 40 | 8 | 2 | 80 | 10 | 0 | 100 | 6 | 4 | 60 | 8 | 2 | 80 |
| 19 | 2 | 8 | 20 | 0 | 10 | 0 | 7 | 3 | 70 | 0 | 10 | 0 | 0 | 10 | 0 |
| 20 | 1 | 9 | 10 | 6 | 4 | 60 | 10 | 0 | 100 | 9 | 1 | 90 | 8 | 2 | 80 |
| 21 | 8 | 2 | 80 | 9 | 1 | 90 | 9 | 1 | 90 | 9 | 1 | 90 | 10 | 0 | 100 |
| 22 | 7 | 3 | 70 | 5 | 5 | 50 | 10 | 0 | 100 | 9 | 1 | 90 | 9 | 1 | 90 |
| 23 | 3 | 7 | 30 | 6 | 4 | 60 | 7 | 3 | 70 | 8 | 2 | 80 | 7 | 3 | 70 |
| 24 | 8 | 2 | 80 | 9 | 1 | 90 | 9 | 1 | 90 | 8 | 2 | 80 | 9 | 1 | 90 |
| 25 | 4 | 6 | 40 | 5 | 5 | 50 | 10 | 0 | 100 | 8 | 2 | 80 | 10 | 0 | 100 |
| 26 | 3 | 7 | 30 | 8 | 2 | 80 | 5 | 5 | 50 | 6 | 4 | 60 | 8 | 2 | 80 |
| 27 | 7 | 3 | 70 | 4 | 6 | 40 | 10 | 0 | 100 | 9 | 1 | 90 | 8 | 2 | 80 |
| 28 | 1 | 9 | 10 | 9 | 1 | 90 | 9 | 1 | 90 | 10 | 0 | 100 | 6 | 4 | 60 |
| 29 | 9 | 1 | 90 | 10 | 0 | 100 | 5 | 5 | 50 | 9 | 1 | 90 | 8 | 2 | 80 |
| 30 | 2 | 4 | 33 | 3 | 0 | 100 | 3 | 0 | 100 | 3 | 1 | 75 | 4 | 0 | 100 |
| 31 | 1 | 9 | 10 | 8 | 2 | 80 | 7 | 3 | 70 | 5 | 5 | 50 | 6 | 4 | 60 |
| 32 | 1 | 9 | 10 | 10 | 0 | 100 | 8 | 2 | 80 | 10 | 0 | 100 | 10 | 0 | 100 |
| Total | 162 | 149 | **52** | 215 | 93 | **70** | 276 | 34 | **89** | 257 | 53 | **83** | 263 | 48 | **85** |

Tabela Anexo B - 3 – Resultados de reconhecimento das amostras caligráficas de cada participante, com o reconhecedor de Rubine, antes do processo de filtragem

| | **Rubine – Resultados iniciais (antes da filtragem)** | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Retângulo** | | | **Triângulo** | | | **Círculo** | | | **Alfa** | | | **Ziguezague** | | |
| **Partic.** | ☺ | ☹ | **☺%** | ☺ | ☹ | **☺%** | ☺ | ☹ | **☺%** | ☺ | ☹ | **☺%** | ☺ | ☹ | **☺%** |
| 1 | 10 | 0 | 100 | 0 | 10 | 0 | 2 | 8 | 20 | 5 | 5 | 50 | 10 | 0 | 100 |
| 2 | 9 | 1 | 90 | 4 | 6 | 40 | 2 | 8 | 20 | 4 | 6 | 40 | 9 | 1 | 90 |
| 3 | 5 | 0 | 100 | 2 | 3 | 40 | 4 | 3 | 57 | 4 | 2 | 67 | 7 | 0 | 100 |
| 4 | 10 | 0 | 100 | 5 | 5 | 50 | 6 | 4 | 60 | 4 | 6 | 40 | 10 | 0 | 100 |
| 5 | 8 | 2 | 80 | 6 | 4 | 60 | 0 | 10 | 0 | 3 | 7 | 30 | 10 | 0 | 100 |
| 6 | 6 | 4 | 60 | 2 | 8 | 20 | 4 | 6 | 40 | 3 | 7 | 30 | 10 | 0 | 100 |
| 7 | 10 | 0 | 100 | 7 | 3 | 70 | 5 | 5 | 50 | 6 | 4 | 60 | 10 | 0 | 100 |
| 8 | 10 | 0 | 100 | 0 | 10 | 0 | 1 | 9 | 10 | 5 | 5 | 50 | 7 | 3 | 70 |
| 9 | 10 | 0 | 100 | 0 | 10 | 0 | 0 | 10 | 0 | 10 | 0 | 100 | 3 | 7 | 30 |
| 10 | 7 | 3 | 70 | 2 | 8 | 20 | 3 | 7 | 30 | 7 | 3 | 70 | 6 | 4 | 60 |
| 11 | 8 | 2 | 80 | 10 | 0 | 100 | 1 | 9 | 10 | 9 | 1 | 90 | 7 | 3 | 70 |
| 12 | 10 | 0 | 100 | 0 | 10 | 0 | 1 | 9 | 10 | 7 | 3 | 70 | 10 | 0 | 100 |
| 13 | 10 | 0 | 100 | 8 | 2 | 80 | 1 | 9 | 10 | 6 | 4 | 60 | 7 | 3 | 70 |
| 14 | 10 | 0 | 100 | 1 | 9 | 10 | 6 | 4 | 60 | 9 | 1 | 90 | 10 | 0 | 100 |
| 15 | 10 | 0 | 100 | 2 | 8 | 20 | 8 | 2 | 80 | 8 | 2 | 80 | 8 | 2 | 80 |
| 16 | 5 | 5 | 50 | 7 | 3 | 70 | 3 | 7 | 30 | 8 | 2 | 80 | 5 | 5 | 50 |
| 17 | 10 | 0 | 100 | 2 | 8 | 20 | 2 | 8 | 20 | 8 | 2 | 80 | 8 | 2 | 80 |
| 18 | 9 | 1 | 90 | 0 | 10 | 0 | 6 | 4 | 60 | 7 | 3 | 70 | 1 | 9 | 10 |
| 19 | 6 | 4 | 60 | 0 | 10 | 0 | 0 | 10 | 0 | 0 | 10 | 0 | 0 | 10 | 0 |
| 20 | 5 | 5 | 50 | 0 | 10 | 0 | 5 | 5 | 50 | 9 | 1 | 90 | 8 | 2 | 80 |
| 21 | 8 | 2 | 80 | 0 | 10 | 0 | 1 | 9 | 10 | 10 | 0 | 100 | 9 | 1 | 90 |
| 22 | 10 | 0 | 100 | 0 | 10 | 0 | 0 | 10 | 0 | 1 | 9 | 10 | 9 | 1 | 90 |
| 23 | 9 | 1 | 90 | 0 | 10 | 0 | 1 | 9 | 10 | 7 | 3 | 70 | 8 | 2 | 80 |
| 24 | 6 | 4 | 60 | 9 | 1 | 90 | 4 | 6 | 40 | 6 | 4 | 60 | 10 | 0 | 100 |
| 25 | 7 | 3 | 70 | 0 | 10 | 0 | 3 | 7 | 30 | 3 | 7 | 30 | 10 | 0 | 100 |
| 26 | 4 | 6 | 40 | 8 | 2 | 80 | 4 | 6 | 40 | 8 | 2 | 80 | 8 | 2 | 80 |
| 27 | 7 | 3 | 70 | 5 | 5 | 50 | 0 | 10 | 0 | 1 | 9 | 10 | 9 | 1 | 90 |
| 28 | 9 | 1 | 90 | 0 | 10 | 0 | 3 | 7 | 30 | 3 | 7 | 30 | 6 | 4 | 60 |
| 29 | 10 | 0 | 100 | 0 | 10 | 0 | 0 | 10 | 0 | 10 | 0 | 100 | 8 | 2 | 80 |
| 30 | 5 | 1 | 83 | 1 | 2 | 33 | 0 | 3 | 0 | 3 | 1 | 75 | 4 | 0 | 100 |
| 31 | 10 | 0 | 100 | 0 | 10 | 0 | 1 | 9 | 10 | 6 | 4 | 60 | 9 | 1 | 90 |
| 32 | 4 | 6 | 40 | 0 | 10 | 0 | 0 | 10 | 0 | 10 | 0 | 100 | 7 | 3 | 70 |
| Total | 257 | 54 | **83** | 81 | 227 | **26** | 77 | 233 | **25** | 190 | 120 | **61** | 243 | 68 | **78** |

Tabela Anexo B - 4 – Resultados de reconhecimento das amostras caligráficas de cada participante, com o reconhecedor CALI, após processo de filtragem

| | **CALI – Resultados após filtragem** | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Retângulo** | | | **Triângulo** | | | **Círculo** | | | **Alfa** | | | **Ziguezague** | | |
| **Partic.** | ☺ | ☹ | **☺%** | ☺ | ☹ | **☺%** | ☺ | ☹ | **☺%** | ☺ | ☹ | **☺%** | ☺ | ☹ | **☺%** |
| 1 | 10 | 0 | 100 | 8 | 2 | 80 | 10 | 0 | 100 | 4 | 5 | 44 | 10 | 0 | 100 |
| 2 | 8 | 1 | 89 | 9 | 0 | 100 | 9 | 1 | 90 | 10 | 0 | 100 | 2 | 7 | 22 |
| 3 | 5 | 0 | 100 | 5 | 0 | 100 | 7 | 0 | 100 | 5 | 0 | 100 | 6 | 1 | 86 |
| 4 | 10 | 0 | 100 | 9 | 0 | 100 | 9 | 1 | 90 | 3 | 5 | 38 | 8 | 2 | 80 |
| 5 | 9 | 0 | 100 | 10 | 0 | 100 | 10 | 0 | 100 | 5 | 5 | 50 | 2 | 8 | 20 |
| 6 | 10 | 0 | 100 | 10 | 0 | 100 | 10 | 0 | 100 | 8 | 2 | 80 | 10 | 0 | 100 |
| 7 | 9 | 0 | 100 | 10 | 0 | 100 | 10 | 0 | 100 | 9 | 1 | 90 | 5 | 5 | 50 |
| 8 | 9 | 0 | 100 | 10 | 0 | 100 | 9 | 1 | 90 | 3 | 0 | 100 | 5 | 5 | 50 |
| 9 | 10 | 0 | 100 | 8 | 0 | 100 | 8 | 0 | 100 | 10 | 0 | 100 | 9 | 1 | 90 |
| 10 | 10 | 0 | 100 | 5 | 0 | 100 | 7 | 0 | 100 | 7 | 2 | 78 | 5 | 4 | 56 |
| 11 | 10 | 0 | 100 | 8 | 0 | 100 | 10 | 0 | 100 | 7 | 2 | 78 | 4 | 1 | 80 |
| 12 | 10 | 0 | 100 | 10 | 0 | 100 | 10 | 0 | 100 | 8 | 1 | 89 | 6 | 4 | 60 |
| 13 | 9 | 0 | 100 | 7 | 1 | 88 | 5 | 1 | 83 | 10 | 0 | 100 | 0 | 0 | 0 |
| 14 | 10 | 0 | 100 | 9 | 0 | 100 | 9 | 1 | 90 | 6 | 4 | 60 | 4 | 6 | 40 |
| 15 | 10 | 0 | 100 | 8 | 0 | 100 | 10 | 0 | 100 | 10 | 0 | 100 | 5 | 3 | 63 |
| 16 | 6 | 1 | 86 | 7 | 1 | 88 | 9 | 0 | 100 | 8 | 2 | 80 | 2 | 2 | 50 |
| 17 | 9 | 1 | 90 | 8 | 1 | 89 | 9 | 0 | 100 | 9 | 1 | 90 | 3 | 4 | 43 |
| 18 | 10 | 0 | 100 | 10 | 0 | 100 | 9 | 1 | 90 | 1 | 0 | 100 | 0 | 2 | 0 |
| 19 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 20 | 8 | 0 | 100 | 10 | 0 | 100 | 9 | 0 | 100 | 8 | 2 | 80 | 6 | 1 | 86 |
| 21 | 10 | 0 | 100 | 9 | 0 | 100 | 9 | 0 | 100 | 9 | 1 | 90 | 8 | 2 | 80 |
| 22 | 9 | 0 | 100 | 10 | 0 | 100 | 10 | 0 | 100 | 3 | 5 | 38 | 5 | 3 | 63 |
| 23 | 5 | 1 | 83 | 5 | 0 | 100 | 9 | 0 | 100 | 9 | 1 | 90 | 6 | 1 | 86 |
| 24 | 7 | 1 | 88 | 9 | 0 | 100 | 8 | 0 | 100 | 8 | 1 | 89 | 4 | 5 | 44 |
| 25 | 9 | 0 | 100 | 5 | 0 | 100 | 9 | 0 | 100 | 6 | 1 | 86 | 7 | 2 | 78 |
| 26 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 27 | 7 | 0 | 100 | 10 | 0 | 100 | 10 | 0 | 100 | 8 | 1 | 89 | 9 | 0 | 100 |
| 28 | 9 | 0 | 100 | 10 | 0 | 100 | 10 | 0 | 100 | 6 | 0 | 100 | 0 | 0 | 0 |
| 29 | 8 | 1 | 89 | 8 | 2 | 80 | 4 | 1 | 80 | 7 | 2 | 78 | 5 | 1 | 83 |
| 30 | 5 | 1 | 83 | 3 | 0 | 100 | 3 | 0 | 100 | 1 | 3 | 25 | 3 | 1 | 75 |
| 31 | 4 | 1 | 80 | 10 | 0 | 100 | 8 | 0 | 100 | 4 | 3 | 57 | 9 | 1 | 90 |
| 32 | 10 | 0 | 100 | 10 | 0 | 100 | 7 | 0 | 100 | 7 | 1 | 88 | 1 | 6 | 14 |
| Total | 255 | 8 | **97** | 250 | 7 | **97** | 256 | 7 | **97** | 199 | 51 | **80** | 149 | 78 | **66** |

Tabela Anexo B - 5 – Resultados de reconhecimento das amostras caligráficas de cada participante, com o reconhecedor $1, após processo de filtragem

| | **$1 – Resultados após filtragem** | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Retângulo** | | | **Triângulo** | | | **Círculo** | | | **Alfa** | | | **Ziguezague** | | |
| **Partic.** | ☺ | ☹ | **☺%** | ☺ | ☹ | **☺%** | ☺ | ☹ | **☺%** | ☺ | ☹ | **☺%** | ☺ | ☹ | **☺%** |
| 1 | 3 | 7 | 30 | 10 | 0 | 100 | 10 | 0 | 100 | 7 | 2 | 78 | 10 | 0 | 100 |
| 2 | 5 | 4 | 56 | 6 | 3 | 67 | 10 | 0 | 100 | 9 | 1 | 90 | 9 | 0 | 100 |
| 3 | 3 | 2 | 60 | 2 | 3 | 40 | 7 | 0 | 100 | 5 | 0 | 100 | 7 | 0 | 100 |
| 4 | 3 | 7 | 30 | 5 | 4 | 56 | 9 | 1 | 90 | 7 | 1 | 88 | 10 | 0 | 100 |
| 5 | 7 | 2 | 78 | 6 | 4 | 60 | 10 | 0 | 100 | 7 | 3 | 70 | 10 | 0 | 100 |
| 6 | 3 | 7 | 30 | 9 | 1 | 90 | 10 | 0 | 100 | 9 | 1 | 90 | 9 | 1 | 90 |
| 7 | 5 | 4 | 56 | 2 | 8 | 20 | 10 | 0 | 100 | 9 | 1 | 90 | 10 | 0 | 100 |
| 8 | 6 | 3 | 67 | 10 | 0 | 100 | 9 | 1 | 90 | 3 | 0 | 100 | 10 | 0 | 100 |
| 9 | 7 | 3 | 70 | 3 | 5 | 38 | 8 | 0 | 100 | 9 | 1 | 90 | 9 | 1 | 90 |
| 10 | 9 | 1 | 90 | 3 | 2 | 60 | 7 | 0 | 100 | 8 | 1 | 89 | 8 | 1 | 89 |
| 11 | 5 | 5 | 50 | 7 | 1 | 88 | 10 | 0 | 100 | 9 | 0 | 100 | 5 | 0 | 100 |
| 12 | 6 | 4 | 60 | 10 | 0 | 100 | 10 | 0 | 100 | 9 | 0 | 100 | 10 | 0 | 100 |
| 13 | 9 | 0 | 100 | 7 | 1 | 88 | 6 | 0 | 100 | 10 | 0 | 100 | 0 | 0 | 0 |
| 14 | 10 | 0 | 100 | 4 | 5 | 44 | 10 | 0 | 100 | 10 | 0 | 100 | 10 | 0 | 100 |
| 15 | 8 | 2 | 80 | 7 | 1 | 88 | 10 | 0 | 100 | 10 | 0 | 100 | 8 | 0 | 100 |
| 16 | 3 | 4 | 43 | 5 | 3 | 63 | 8 | 1 | 89 | 10 | 0 | 100 | 2 | 2 | 50 |
| 17 | 6 | 4 | 60 | 9 | 0 | 100 | 9 | 0 | 100 | 10 | 0 | 100 | 7 | 0 | 100 |
| 18 | 4 | 6 | 40 | 8 | 2 | 80 | 10 | 0 | 100 | 1 | 0 | 100 | 2 | 0 | 100 |
| 19 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 20 | 1 | 7 | 13 | 6 | 4 | 60 | 9 | 0 | 100 | 9 | 1 | 90 | 5 | 2 | 71 |
| 21 | 8 | 2 | 80 | 9 | 0 | 100 | 9 | 0 | 100 | 9 | 1 | 90 | 10 | 0 | 100 |
| 22 | 6 | 3 | 67 | 5 | 5 | 50 | 10 | 0 | 100 | 7 | 1 | 88 | 8 | 0 | 100 |
| 23 | 2 | 4 | 33 | 4 | 1 | 80 | 7 | 2 | 78 | 8 | 2 | 80 | 6 | 1 | 86 |
| 24 | 6 | 2 | 75 | 9 | 0 | 100 | 8 | 0 | 100 | 8 | 1 | 89 | 8 | 1 | 89 |
| 25 | 4 | 5 | 44 | 4 | 1 | 80 | 9 | 0 | 100 | 7 | 0 | 100 | 9 | 0 | 100 |
| 26 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 27 | 7 | 0 | 100 | 4 | 6 | 40 | 10 | 0 | 100 | 9 | 0 | 100 | 8 | 1 | 89 |
| 28 | 1 | 8 | 11 | 9 | 1 | 90 | 9 | 1 | 90 | 6 | 0 | 100 | 0 | 0 | 0 |
| 29 | 8 | 1 | 89 | 10 | 0 | 100 | 4 | 1 | 80 | 9 | 0 | 100 | 5 | 1 | 83 |
| 30 | 2 | 4 | 33 | 3 | 0 | 100 | 3 | 0 | 100 | 3 | 1 | 75 | 4 | 0 | 100 |
| 31 | 0 | 5 | 0 | 8 | 2 | 80 | 7 | 1 | 88 | 5 | 2 | 71 | 6 | 4 | 60 |
| 32 | 1 | 9 | 10 | 10 | 0 | 100 | 7 | 0 | 100 | 8 | 0 | 100 | 7 | 0 | 100 |
| Total | 148 | 115 | **56** | 194 | 63 | **75** | 255 | 8 | **97** | 230 | 20 | **92** | 212 | 15 | **93** |

Tabela Anexo B - 6 – Resultados de reconhecimento das amostras caligráficas de cada participante, com o reconhecedor de Rubine, após processo de filtragem

| | **Rubine – Resultados após filtragem** | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Retângulo** | | | **Triângulo** | | | **Círculo** | | | **Alfa** | | | **Ziguezague** | | |
| **Partic.** | ☺ | ☹ | **☺%** | ☺ | ☹ | **☺%** | ☺ | ☹ | **☺%** | ☺ | ☹ | **☺%** | ☺ | ☹ | **☺%** |
| 1 | 10 | 0 | 100 | 0 | 10 | 0 | 2 | 8 | 20 | 5 | 4 | 56 | 10 | 0 | 100 |
| 2 | 9 | 0 | 100 | 4 | 5 | 44 | 2 | 8 | 20 | 4 | 6 | 40 | 8 | 1 | 89 |
| 3 | 5 | 0 | 100 | 2 | 3 | 40 | 4 | 3 | 57 | 4 | 1 | 80 | 7 | 0 | 100 |
| 4 | 10 | 0 | 100 | 5 | 4 | 56 | 6 | 4 | 60 | 4 | 4 | 50 | 10 | 0 | 100 |
| 5 | 7 | 2 | 78 | 6 | 4 | 60 | 0 | 10 | 0 | 3 | 7 | 30 | 10 | 0 | 100 |
| 6 | 6 | 4 | 60 | 2 | 8 | 20 | 4 | 6 | 40 | 3 | 7 | 30 | 10 | 0 | 100 |
| 7 | 9 | 0 | 100 | 7 | 3 | 70 | 5 | 5 | 50 | 6 | 4 | 60 | 10 | 0 | 100 |
| 8 | 9 | 0 | 100 | 0 | 10 | 0 | 1 | 9 | 10 | 0 | 3 | 0 | 7 | 3 | 70 |
| 9 | 10 | 0 | 100 | 0 | 8 | 0 | 0 | 8 | 0 | 10 | 0 | 100 | 3 | 7 | 30 |
| 10 | 7 | 3 | 70 | 1 | 4 | 20 | 2 | 5 | 29 | 6 | 3 | 67 | 6 | 3 | 67 |
| 11 | 8 | 2 | 80 | 8 | 0 | 100 | 1 | 9 | 10 | 8 | 1 | 89 | 4 | 1 | 80 |
| 12 | 10 | 0 | 100 | 0 | 10 | 0 | 1 | 9 | 10 | 6 | 3 | 67 | 10 | 0 | 100 |
| 13 | 9 | 0 | 100 | 6 | 2 | 75 | 1 | 5 | 17 | 6 | 4 | 60 | 0 | 0 | 0 |
| 14 | 10 | 0 | 100 | 1 | 8 | 11 | 6 | 4 | 60 | 9 | 1 | 90 | 10 | 0 | 100 |
| 15 | 10 | 0 | 100 | 2 | 6 | 25 | 8 | 2 | 80 | 8 | 2 | 80 | 8 | 0 | 100 |
| 16 | 3 | 4 | 43 | 7 | 1 | 88 | 2 | 7 | 22 | 8 | 2 | 80 | 2 | 2 | 50 |
| 17 | 10 | 0 | 100 | 1 | 8 | 11 | 2 | 7 | 22 | 8 | 2 | 80 | 7 | 0 | 100 |
| 18 | 9 | 1 | 90 | 0 | 10 | 0 | 6 | 4 | 60 | 1 | 0 | 100 | 0 | 2 | 0 |
| 19 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 20 | 3 | 5 | 38 | 0 | 10 | 0 | 5 | 4 | 56 | 9 | 1 | 90 | 6 | 1 | 86 |
| 21 | 8 | 2 | 80 | 0 | 9 | 0 | 1 | 8 | 11 | 10 | 0 | 100 | 9 | 1 | 90 |
| 22 | 9 | 0 | 100 | 0 | 10 | 0 | 0 | 10 | 0 | 0 | 8 | 0 | 7 | 1 | 88 |
| 23 | 6 | 0 | 100 | 0 | 5 | 0 | 1 | 8 | 11 | 7 | 3 | 70 | 6 | 1 | 86 |
| 24 | 5 | 3 | 63 | 9 | 0 | 100 | 4 | 4 | 50 | 6 | 3 | 67 | 9 | 0 | 100 |
| 25 | 7 | 2 | 78 | 0 | 5 | 0 | 3 | 6 | 33 | 1 | 6 | 14 | 9 | 0 | 100 |
| 26 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 27 | 7 | 0 | 100 | 5 | 5 | 50 | 0 | 10 | 0 | 0 | 9 | 0 | 9 | 0 | 100 |
| 28 | 8 | 1 | 89 | 0 | 10 | 0 | 3 | 7 | 30 | 3 | 3 | 50 | 0 | 0 | 0 |
| 29 | 9 | 0 | 100 | 0 | 10 | 0 | 0 | 5 | 0 | 9 | 0 | 100 | 5 | 1 | 83 |
| 30 | 5 | 1 | 83 | 1 | 2 | 33 | 0 | 3 | 0 | 3 | 1 | 75 | 4 | 0 | 100 |
| 31 | 5 | 0 | 100 | 0 | 10 | 0 | 1 | 7 | 13 | 5 | 2 | 71 | 9 | 1 | 90 |
| 32 | 4 | 6 | 40 | 0 | 10 | 0 | 0 | 7 | 0 | 8 | 0 | 100 | 6 | 1 | 86 |
| Total | 227 | 36 | **86** | 67 | 190 | **26** | 71 | 192 | **27** | 160 | 90 | **64** | 201 | 26 | **89** |

Tabela Anexo B - 7 – Resultados de reconhecimento das amostras caligráficas de cada participante, com o reconhecedor CALI, após as melhorias à implementação do reconhecedor

| | **CALI – Resultados após melhorias** | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Retângulo** | | | **Triângulo** | | | **Círculo** | | | **Alfa** | | | **Ziguezague** | | |
| **Partic.** | ☺ | ☹ | **☺%** | ☺ | ☹ | **☺%** | ☺ | ☹ | **☺%** | ☺ | ☹ | **☺%** | ☺ | ☹ | **☺%** |
| 1 | 10 | 0 | 100 | 8 | 2 | 80 | 10 | 0 | 100 | 6 | 3 | 67 | 10 | 0 | 100 |
| 2 | 8 | 1 | 89 | 9 | 0 | 100 | 9 | 1 | 90 | 10 | 0 | 100 | 9 | 0 | 100 |
| 3 | 5 | 0 | 100 | 5 | 0 | 100 | 7 | 0 | 100 | 5 | 0 | 100 | 7 | 0 | 100 |
| 4 | 10 | 0 | 100 | 9 | 0 | 100 | 9 | 1 | 90 | 5 | 3 | 63 | 10 | 0 | 100 |
| 5 | 9 | 0 | 100 | 10 | 0 | 100 | 10 | 0 | 100 | 5 | 5 | 50 | 9 | 1 | 90 |
| 6 | 9 | 1 | 90 | 10 | 0 | 100 | 10 | 0 | 100 | 8 | 2 | 80 | 10 | 0 | 100 |
| 7 | 9 | 0 | 100 | 10 | 0 | 100 | 10 | 0 | 100 | 10 | 0 | 100 | 7 | 3 | 70 |
| 8 | 9 | 0 | 100 | 10 | 0 | 100 | 9 | 1 | 90 | 3 | 0 | 100 | 9 | 1 | 90 |
| 9 | 10 | 0 | 100 | 8 | 0 | 100 | 8 | 0 | 100 | 10 | 0 | 100 | 9 | 1 | 90 |
| 10 | 10 | 0 | 100 | 5 | 0 | 100 | 7 | 0 | 100 | 7 | 2 | 78 | 7 | 2 | 78 |
| 11 | 10 | 0 | 100 | 8 | 0 | 100 | 10 | 0 | 100 | 7 | 2 | 78 | 4 | 1 | 80 |
| 12 | 10 | 0 | 100 | 10 | 0 | 100 | 10 | 0 | 100 | 8 | 1 | 89 | 10 | 0 | 100 |
| 13 | 9 | 0 | 100 | 7 | 1 | 88 | 5 | 1 | 83 | 10 | 0 | 100 | 0 | 0 | 0 |
| 14 | 10 | 0 | 100 | 9 | 0 | 100 | 9 | 1 | 90 | 6 | 4 | 60 | 10 | 0 | 100 |
| 15 | 10 | 0 | 100 | 8 | 0 | 100 | 10 | 0 | 100 | 10 | 0 | 100 | 7 | 1 | 88 |
| 16 | 6 | 1 | 86 | 8 | 0 | 100 | 9 | 0 | 100 | 8 | 2 | 80 | 3 | 1 | 75 |
| 17 | 10 | 0 | 100 | 9 | 0 | 100 | 9 | 0 | 100 | 10 | 0 | 100 | 6 | 1 | 86 |
| 18 | 10 | 0 | 100 | 10 | 0 | 100 | 9 | 1 | 90 | 1 | 0 | 100 | 2 | 0 | 100 |
| 19 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 20 | 8 | 0 | 100 | 10 | 0 | 100 | 9 | 0 | 100 | 10 | 0 | 100 | 7 | 0 | 100 |
| 21 | 10 | 0 | 100 | 9 | 0 | 100 | 9 | 0 | 100 | 10 | 0 | 100 | 10 | 0 | 100 |
| 22 | 9 | 0 | 100 | 10 | 0 | 100 | 10 | 0 | 100 | 4 | 4 | 50 | 7 | 1 | 88 |
| 23 | 5 | 1 | 83 | 5 | 0 | 100 | 9 | 0 | 100 | 9 | 1 | 90 | 6 | 1 | 86 |
| 24 | 7 | 1 | 88 | 9 | 0 | 100 | 8 | 0 | 100 | 8 | 1 | 89 | 9 | 0 | 100 |
| 25 | 9 | 0 | 100 | 5 | 0 | 100 | 9 | 0 | 100 | 6 | 1 | 86 | 8 | 1 | 89 |
| 26 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 27 | 7 | 0 | 100 | 10 | 0 | 100 | 10 | 0 | 100 | 9 | 0 | 100 | 9 | 0 | 100 |
| 28 | 9 | 0 | 100 | 10 | 0 | 100 | 10 | 0 | 100 | 6 | 0 | 100 | 0 | 0 | 0 |
| 29 | 8 | 1 | 89 | 8 | 2 | 80 | 4 | 1 | 80 | 8 | 1 | 89 | 6 | 0 | 100 |
| 30 | 5 | 1 | 83 | 3 | 0 | 100 | 3 | 0 | 100 | 1 | 3 | 25 | 3 | 1 | 75 |
| 31 | 4 | 1 | 80 | 10 | 0 | 100 | 8 | 0 | 100 | 4 | 3 | 57 | 10 | 0 | 100 |
| 32 | 10 | 0 | 100 | 10 | 0 | 100 | 7 | 0 | 100 | 7 | 1 | 88 | 4 | 3 | 57 |
| Total | 255 | 8 | **97** | 252 | 5 | **98** | 256 | 7 | **97** | 211 | 39 | **84** | 208 | 19 | **92** |

Tabela Anexo B - 8 – Resultados de reconhecimento das amostras caligráficas de cada participante, com o reconhecedor $1, após as melhorias à implementação do reconhecedor

| | **$1 – Resultados após melhorias** | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Retângulo** | | | **Triângulo** | | | **Círculo** | | | **Alfa** | | | **Ziguezague** | | |
| **Partic.** | ☺ | ☹ | **☺%** | ☺ | ☹ | **☺%** | ☺ | ☹ | **☺%** | ☺ | ☹ | **☺%** | ☺ | ☹ | **☺%** |
| 1 | 5 | 5 | 50 | 10 | 0 | 100 | 9 | 1 | 90 | 5 | 4 | 56 | 10 | 0 | 100 |
| 2 | 7 | 2 | 78 | 9 | 0 | 100 | 7 | 3 | 70 | 8 | 2 | 80 | 9 | 0 | 100 |
| 3 | 4 | 1 | 80 | 5 | 0 | 100 | 7 | 0 | 100 | 4 | 1 | 80 | 7 | 0 | 100 |
| 4 | 8 | 2 | 80 | 7 | 2 | 78 | 8 | 2 | 80 | 5 | 3 | 63 | 10 | 0 | 100 |
| 5 | 9 | 0 | 100 | 9 | 1 | 90 | 5 | 5 | 50 | 6 | 4 | 60 | 10 | 0 | 100 |
| 6 | 10 | 0 | 100 | 10 | 0 | 100 | 10 | 0 | 100 | 9 | 1 | 90 | 9 | 1 | 90 |
| 7 | 6 | 3 | 67 | 6 | 4 | 60 | 10 | 0 | 100 | 7 | 3 | 70 | 10 | 0 | 100 |
| 8 | 7 | 2 | 78 | 9 | 1 | 90 | 9 | 1 | 90 | 3 | 0 | 100 | 10 | 0 | 100 |
| 9 | 7 | 3 | 70 | 3 | 5 | 38 | 6 | 2 | 75 | 9 | 1 | 90 | 9 | 1 | 90 |
| 10 | 9 | 1 | 90 | 5 | 0 | 100 | 7 | 0 | 100 | 8 | 1 | 89 | 8 | 1 | 89 |
| 11 | 9 | 1 | 90 | 8 | 0 | 100 | 10 | 0 | 100 | 9 | 0 | 100 | 5 | 0 | 100 |
| 12 | 9 | 1 | 90 | 9 | 1 | 90 | 10 | 0 | 100 | 9 | 0 | 100 | 10 | 0 | 100 |
| 13 | 9 | 0 | 100 | 7 | 1 | 88 | 6 | 0 | 100 | 9 | 1 | 90 | 0 | 0 | 0 |
| 14 | 9 | 1 | 90 | 7 | 2 | 78 | 9 | 1 | 90 | 9 | 1 | 90 | 10 | 0 | 100 |
| 15 | 9 | 1 | 90 | 7 | 1 | 88 | 10 | 0 | 100 | 10 | 0 | 100 | 8 | 0 | 100 |
| 16 | 6 | 1 | 86 | 8 | 0 | 100 | 8 | 1 | 89 | 10 | 0 | 100 | 2 | 2 | 50 |
| 17 | 6 | 4 | 60 | 9 | 0 | 100 | 6 | 3 | 67 | 10 | 0 | 100 | 7 | 0 | 100 |
| 18 | 6 | 4 | 60 | 9 | 1 | 90 | 8 | 2 | 80 | 1 | 0 | 100 | 2 | 0 | 100 |
| 19 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 20 | 7 | 1 | 88 | 10 | 0 | 100 | 5 | 4 | 56 | 8 | 2 | 80 | 5 | 2 | 71 |
| 21 | 8 | 2 | 80 | 9 | 0 | 100 | 9 | 0 | 100 | 9 | 1 | 90 | 10 | 0 | 100 |
| 22 | 9 | 0 | 100 | 8 | 2 | 80 | 10 | 0 | 100 | 6 | 2 | 75 | 8 | 0 | 100 |
| 23 | 2 | 4 | 33 | 4 | 1 | 80 | 5 | 4 | 56 | 8 | 2 | 80 | 6 | 1 | 86 |
| 24 | 7 | 1 | 88 | 9 | 0 | 100 | 6 | 2 | 75 | 8 | 1 | 89 | 8 | 1 | 89 |
| 25 | 6 | 3 | 67 | 3 | 2 | 60 | 9 | 0 | 100 | 6 | 1 | 86 | 9 | 0 | 100 |
| 26 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 27 | 7 | 0 | 100 | 7 | 3 | 70 | 10 | 0 | 100 | 9 | 0 | 100 | 8 | 1 | 89 |
| 28 | 6 | 3 | 67 | 10 | 0 | 100 | 9 | 1 | 90 | 6 | 0 | 100 | 0 | 0 | 0 |
| 29 | 8 | 1 | 89 | 10 | 0 | 100 | 4 | 1 | 80 | 8 | 1 | 89 | 5 | 1 | 83 |
| 30 | 6 | 0 | 100 | 3 | 0 | 100 | 3 | 0 | 100 | 3 | 1 | 75 | 4 | 0 | 100 |
| 31 | 2 | 3 | 40 | 10 | 0 | 100 | 7 | 1 | 88 | 4 | 3 | 57 | 6 | 4 | 60 |
| 32 | 6 | 4 | 60 | 10 | 0 | 100 | 4 | 3 | 57 | 8 | 0 | 100 | 7 | 0 | 100 |
| Total | 209 | 54 | **79** | 230 | 27 | **89** | 226 | 37 | **86** | 214 | 36 | **85,6** | 212 | 15 | **93** |

Tabela Anexo B - 9 – Resultados de reconhecimento das amostras caligráficas de cada participante, com o reconhecedor de Rubine, após as melhorias à implementação do reconhecedor

| | **Rubine – Resultados após melhorias** | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Retângulo** | | | **Triângulo** | | | **Círculo** | | | **Alfa** | | | **Ziguezague** | | |
| **Partic.** | ☺ | ☹ | **☺%** | ☺ | ☹ | **☺%** | ☺ | ☹ | **☺%** | ☺ | ☹ | **☺%** | ☺ | ☹ | **☺%** |
| 1 | 9 | 1 | 90 | 6 | 4 | 60 | 8 | 2 | 80 | 6 | 3 | 67 | 10 | 0 | 100 |
| 2 | 6 | 3 | 67 | 9 | 0 | 100 | 6 | 4 | 60 | 4 | 6 | 40 | 9 | 0 | 100 |
| 3 | 3 | 2 | 60 | 1 | 4 | 20 | 7 | 0 | 100 | 4 | 1 | 80 | 7 | 0 | 100 |
| 4 | 6 | 4 | 60 | 5 | 4 | 56 | 5 | 5 | 50 | 3 | 5 | 38 | 10 | 0 | 100 |
| 5 | 6 | 3 | 67 | 8 | 2 | 80 | 8 | 2 | 80 | 5 | 5 | 50 | 10 | 0 | 100 |
| 6 | 6 | 4 | 60 | 9 | 1 | 90 | 9 | 1 | 90 | 8 | 2 | 80 | 10 | 0 | 100 |
| 7 | 6 | 3 | 67 | 5 | 5 | 50 | 9 | 1 | 90 | 9 | 1 | 90 | 10 | 0 | 100 |
| 8 | 8 | 1 | 89 | 9 | 1 | 90 | 5 | 5 | 50 | 3 | 0 | 100 | 10 | 0 | 100 |
| 9 | 10 | 0 | 100 | 4 | 4 | 50 | 8 | 0 | 100 | 9 | 1 | 90 | 2 | 8 | 20 |
| 10 | 6 | 4 | 60 | 5 | 0 | 100 | 4 | 3 | 57 | 9 | 0 | 100 | 5 | 4 | 56 |
| 11 | 9 | 1 | 90 | 6 | 2 | 75 | 8 | 2 | 80 | 7 | 2 | 78 | 5 | 0 | 100 |
| 12 | 5 | 5 | 50 | 8 | 2 | 80 | 8 | 2 | 80 | 8 | 1 | 89 | 10 | 0 | 100 |
| 13 | 7 | 2 | 78 | 8 | 0 | 100 | 4 | 2 | 67 | 6 | 4 | 60 | 0 | 0 | 0 |
| 14 | 10 | 0 | 100 | 1 | 8 | 11 | 8 | 2 | 80 | 7 | 3 | 70 | 9 | 1 | 90 |
| 15 | 9 | 1 | 90 | 8 | 0 | 100 | 9 | 1 | 90 | 7 | 3 | 70 | 8 | 0 | 100 |
| 16 | 5 | 2 | 71 | 7 | 1 | 88 | 5 | 4 | 56 | 10 | 0 | 100 | 3 | 1 | 75 |
| 17 | 8 | 2 | 80 | 9 | 0 | 100 | 7 | 2 | 78 | 10 | 0 | 100 | 7 | 0 | 100 |
| 18 | 9 | 1 | 90 | 6 | 4 | 60 | 6 | 4 | 60 | 1 | 0 | 100 | 1 | 1 | 50 |
| 19 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 20 | 7 | 1 | 88 | 8 | 2 | 80 | 6 | 3 | 67 | 9 | 1 | 90 | 6 | 1 | 86 |
| 21 | 8 | 2 | 80 | 9 | 0 | 100 | 6 | 3 | 67 | 8 | 2 | 80 | 9 | 1 | 90 |
| 22 | 9 | 0 | 100 | 9 | 1 | 90 | 7 | 3 | 70 | 8 | 0 | 100 | 8 | 0 | 100 |
| 23 | 5 | 1 | 83 | 1 | 4 | 20 | 7 | 2 | 78 | 9 | 1 | 90 | 6 | 1 | 86 |
| 24 | 4 | 4 | 50 | 8 | 1 | 89 | 7 | 1 | 88 | 9 | 0 | 100 | 9 | 0 | 100 |
| 25 | 5 | 4 | 56 | 4 | 1 | 80 | 8 | 1 | 89 | 2 | 5 | 29 | 9 | 0 | 100 |
| 26 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 27 | 7 | 0 | 100 | 8 | 2 | 80 | 10 | 0 | 100 | 9 | 0 | 100 | 8 | 1 | 89 |
| 28 | 4 | 5 | 44 | 7 | 3 | 70 | 9 | 1 | 90 | 5 | 1 | 83 | 0 | 0 | 0 |
| 29 | 9 | 0 | 100 | 6 | 4 | 60 | 2 | 3 | 40 | 6 | 3 | 67 | 5 | 1 | 83 |
| 30 | 5 | 1 | 83 | 2 | 1 | 67 | 2 | 1 | 67 | 2 | 2 | 50 | 3 | 1 | 75 |
| 31 | 4 | 1 | 80 | 5 | 5 | 50 | 6 | 2 | 75 | 3 | 4 | 43 | 10 | 0 | 100 |
| 32 | 9 | 1 | 90 | 10 | 0 | 100 | 6 | 1 | 86 | 5 | 3 | 63 | 6 | 1 | 86 |
| Total | 204 | 59 | **78** | 191 | 66 | **74** | 200 | 63 | **76** | 191 | 59 | **76,4** | 205 | 22 | **90** |

# Anexo C

## Guião da sessão de avaliação de usabilidade da biblioteca SketchyDynamics

Instituto Superior de Engenharia do Porto
DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

**Mestrado em Engenharia Informática**

# Guia da sessão para a avaliação da usabilidade da biblioteca SketchyDynamics

Abílio Costa

Orientação: Prof. João Paulo Pereira

# SketchyDynamics – Apoio à Produção de Sistemas de Simulação da Dinâmica de Corpos Rígidos Baseados em Interfaces Caligráficas

Obrigado por ter aceitado participar nesta experiência.

O objetivo principal desta sessão consiste na avaliação das ideias subjacentes ao desenvolvimento da biblioteca de programação SketchyDynamics. O foco desta biblioteca é disponibilizar não só um motor de simulação de física, mas também uma interface caligráfica para a manipulação direta dos objetos simulados. Assim, com a biblioteca SketchyDynamics, é facilitado e encorajado o desenvolvimento de aplicações de simulação de física que tirem partido do paradigma do desenho à mão e da metáfora do "papel – lápis".

A sessão será conduzida por Abílio Costa e João Paulo Pereira, os quais estarão ao vosso dispor para todo e qualquer esclarecimento. O roteiro da sessão encontra-se descrito na Tabela 1, indicando-se os tempos estimados para a duração de cada uma das tarefas previstas, num total esperado de cerca de 55 minutos.

***Tabela 1 – Roteiro da sessão***

| | | |
|---|---|---|
| 1 | Receção<br>(Descrição dos trabalhos) | 5m |
| 2 | Apresentação do protótipo de demonstração da biblioteca SketchyDynamics, seguida de discussão | 15m |
| 3 | Familiarização com o protótipo de demonstração da biblioteca SketchyDynamics | 10m |
| 4 | Realização do teste de eficiência | 15m |
| 5 | Questionário sobre as funcionalidades da biblioteca SketchyDyncamics<br>(Após a realização do teste de eficiência) | 10m |

Os questionários apresentam uma escala de valores entre dois atributos opostos como, por exemplo, "péssimo" e "excelente". O valor NA (Não se Aplica) corresponde à inexistência de resposta para a questão em causa.

Pretende-se averiguar a utilidade e a facilidade de utilização das funcionalidades da biblioteca SketchyDynamics, pelo que todos os comentários e sugestões serão bem-vindos. Durante a execução das tarefas "pense em voz alta". Não há respostas certas ou erradas. Não se sinta inibido para apontar aspetos negativos ou positivos e para enunciar expetativas frustradas ou recompensadas.

Para finalizar gostaríamos de lhe agradecer o tempo e o esforço despendidos.

# Descarga do protótipo de demonstração da biblioteca SketchyDynamics

A avaliação das funcionalidades presentes na biblioteca SketchyDynamics será realizada com recurso a um protótipo de demonstração, que se encontra disponível *online* para descarga. De forma a obter e executar a aplicação, cada participante deve realizar as seguintes tarefas:

1. Fazer o *download* do arquivo "zip" que contém o protótipo, a partir do endereço http://db.tt/VYxwYtrS;
2. Extrair o arquivo "zip", que deverá dar origem à pasta "SketchyDynamics Demo";
3. Dentro da pasta "SketchTester", executar o ficheiro "Launcher.bat".

# Teste de eficiência do protótipo de demonstração da biblioteca SketchyDynamics

O teste de eficiência passará pela construção da cena apresentada na Figura 1 em duas fases. Não hesite em fazer comentários em voz alta, nem em pedir ajuda sempre que entender necessário. É objetivo desta experiência obter resultados francos, não havendo por isso necessidade de apressar o seu desempenho no decorrer deste teste.

Em cada uma das fases, faça as transformações necessárias para que os objetos correspondam da melhor forma aos apresentados na imagem.

Só inicie a construção da cena depois de receber indicações nesse sentido.

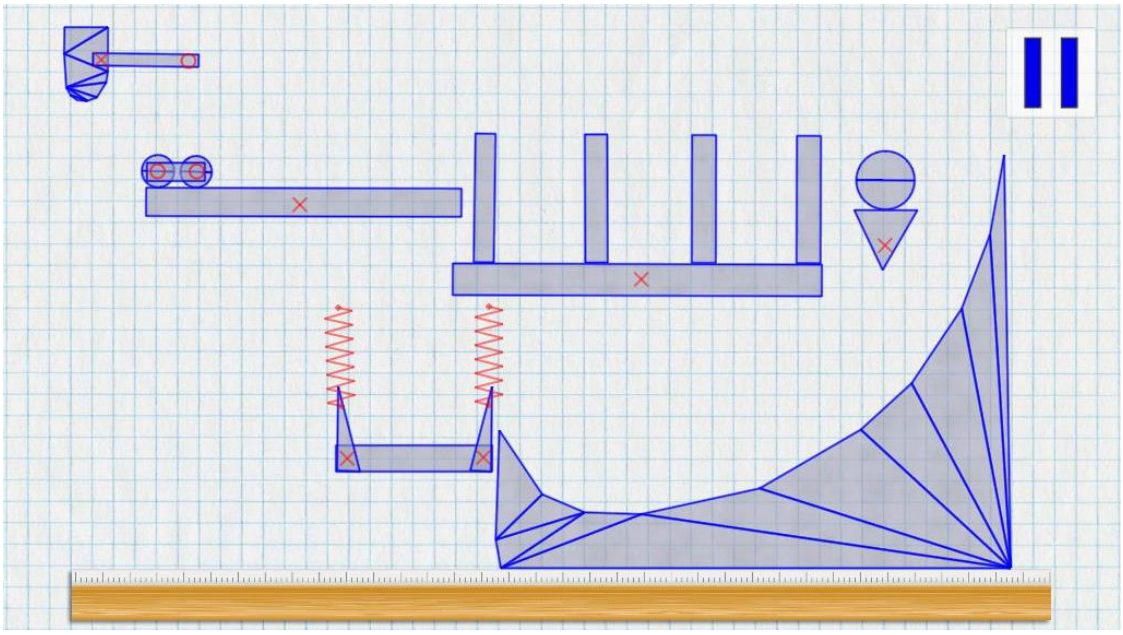


***Figura 1 – Cena para teste de eficiência***

# Teste de eficiência - Parte 1

Por favor proceda à construção da cena apresentada na Figura 2. No final da construção assinale esse facto aos orientadores desta sessão.

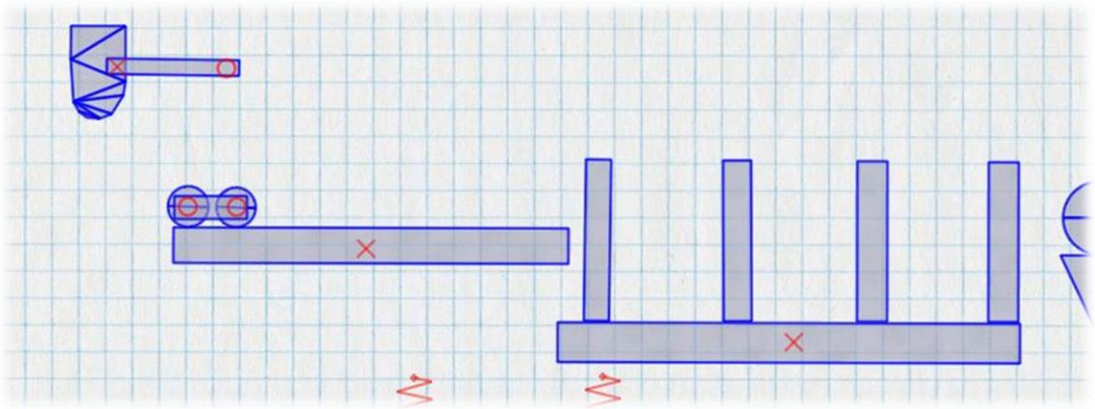


***Figura 2 – Parte 1 da cena para teste de eficiência***

# Teste de eficiência - Parte 2

Por favor proceda à construção da cena apresentada na Figura 3. No final da construção assinale esse facto aos orientadores desta sessão.

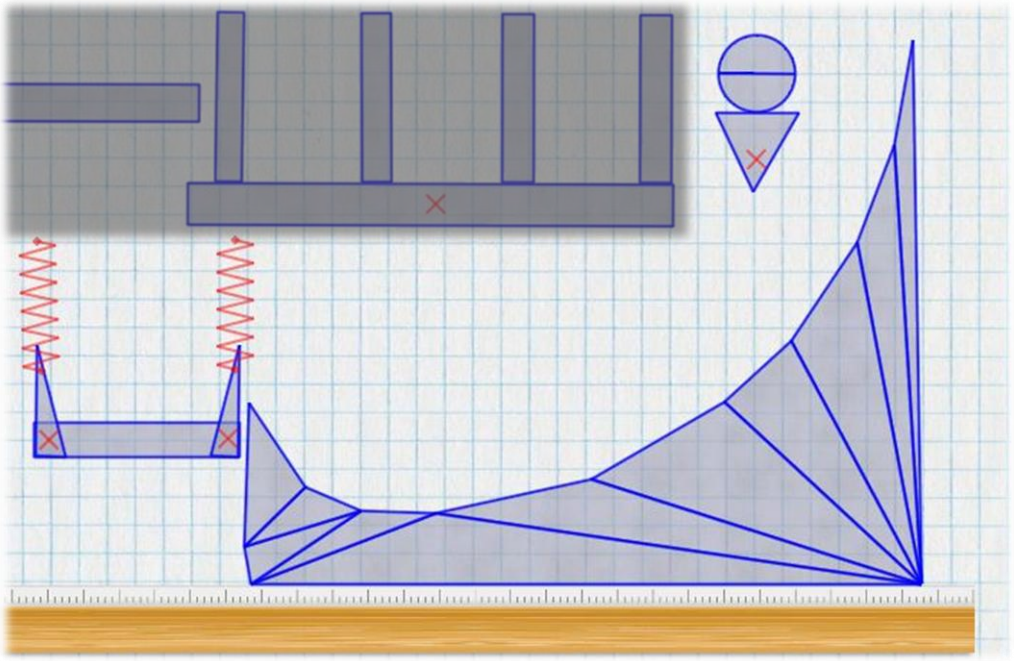


***Figura 3 – Parte 2 da cena para teste de eficiência***

# Questionário após experimentação do protótipo de demonstração da biblioteca SketchyDynamics

| | **Caracterize a adequação do protótipo da biblioteca SketchyDynamics à criação de um ambiente fisicamente simulado no que se refere** | | | | |
|---|---|---|---|---|---|
| | Ao dispositivo utilizado | | | | |
| 1 | • rato | péssima | **0** ☐☐☐☐☐☐☐ | excelente | NA ☐ |
| 2 | • ecrã com estilete | péssima | **0** ☐☐☐☐☐☐☐ | excelente | NA ☐ |
| | Aos gestos caligráficos | | | | |
| 3 | • de criação de corpos retangulares | péssima | **0** ☐☐☐☐☐☐☐ | excelente | NA ☐ |
| 4 | • de criação de corpos triangulares | péssima | **0** ☐☐☐☐☐☐☐ | excelente | NA ☐ |
| 5 | • de criação de corpos circulares | péssima | **0** ☐☐☐☐☐☐☐ | excelente | NA ☐ |
| 6 | • de criação de corpos de forma livre | péssima | **0** ☐☐☐☐☐☐☐ | excelente | NA ☐ |
| 7 | • de criação de conectores de âncora | péssima | **0** ☐☐☐☐☐☐☐ | excelente | NA ☐ |
| 8 | • de criação de conectores de rotação | péssima | **0** ☐☐☐☐☐☐☐ | excelente | NA ☐ |
| 9 | • de criação de conectores de mola | péssima | **0** ☐☐☐☐☐☐☐ | excelente | NA ☐ |
| | À estrutura e disposição na interface | | | | |
| 10 | • da área de remoção de objetos | péssima | **0** ☐☐☐☐☐☐☐ | excelente | NA ☐ |
| 11 | • do indicador de estado da simulação | péssima | **0** ☐☐☐☐☐☐☐ | excelente | NA ☐ |
| 12 | • da lista de expectativas | péssima | **0** ☐☐☐☐☐☐☐ | excelente | NA ☐ |

| | | | | | | |
|---|---|---|---|---|---|---|
| | **Sobre a criação de objetos para a simulação com o protótipo da biblioteca SketchyDynamics** | | | | | |
| 13 | • desenho | difícil | **0** ☐☐☐☐☐☐☐ | fácil | NA ☐ |
| 14 | • criação de corpos | difícil | **0** ☐☐☐☐☐☐☐ | fácil | NA ☐ |
| 15 | • criação de conectores | difícil | **0** ☐☐☐☐☐☐☐ | fácil | NA ☐ |
| 16 | • reconhecimento do gesto | péssimo | **0** ☐☐☐☐☐☐☐ | excelente | NA ☐ |
| 17 | • funcionamento da lista de expectativas | péssimo | **0** ☐☐☐☐☐☐☐ | excelente | NA ☐ |
| 18 | • opinião global | péssima | **0** ☐☐☐☐☐☐☐ | excelente | NA ☐ |
| | **Sobre a edição de objetos com o protótipo da biblioteca SketchyDynamics** | | | | | |
| 19 | • seleção de um objeto | difícil | **0** ☐☐☐☐☐☐☐ | fácil | NA ☐ |
| 20 | • seleção automática de objetos conectados | inútil | **0** ☐☐☐☐☐☐☐ | útil | NA ☐ |
| 21 | • seleção contínua de múltiplos objetos | difícil | **0** ☐☐☐☐☐☐☐ | fácil | NA ☐ |
| 22 | • translação | difícil | **0** ☐☐☐☐☐☐☐ | fácil | NA ☐ |
| 23 | • rotação | difícil | **0** ☐☐☐☐☐☐☐ | fácil | NA ☐ |
| 24 | • escala | difícil | **0** ☐☐☐☐☐☐☐ | fácil | NA ☐ |
| 25 | • opinião global | péssima | **0** ☐☐☐☐☐☐☐ | excelente | NA ☐ |
| | **Sobre a remoção de objetos com o protótipo da biblioteca SketchyDynamics** | | | | | |
| 26 | • utilização da área de remoção de objetos | difícil | **0** ☐☐☐☐☐☐☐ | fácil | NA ☐ |
| 27 | • remoção de conectores através da sua colocação fora do corpo rígido | inútil | **0** ☐☐☐☐☐☐☐ | útil | NA ☐ |
| 28 | • opinião global | péssima | **0** ☐☐☐☐☐☐☐ | excelente | NA ☐ |

| | | | | | |
|---|---|---|---|---|---|
| | **Sobre os estados de simulação no protótipo da biblioteca SketchyDynamics** | | | | |
| 29 | • possibilidade de alternar o estado da simulação | inútil | **0** ☐☐☐☐☐☐☐ | útil | NA ☐ |
| | Relativamente à interrupção automática da simulação | | | | |
| 30 | • durante a exibição da lista de expectativas | inútil | **0** ☐☐☐☐☐☐☐ | útil | NA ☐ |
| 31 | • durante a edição de objetos | inútil | **0** ☐☐☐☐☐☐☐ | útil | NA ☐ |
| 32 | • opinião global | péssima | **0** ☐☐☐☐☐☐☐ | excelente | NA ☐ |
| | **Na sua opinião a funcionalidade da biblioteca SketchyDynamics é** | | | | |
| 33 | | difícil de usar | **0** ☐☐☐☐☐☐☐ | fácil de usar | |
| 34 | | frustrante | **0** ☐☐☐☐☐☐☐ | estimulante | |
| 35 | | rígida | **0** ☐☐☐☐☐☐☐ | flexível | |
| | Para a criação de um ambiente fisicamente simulado | | | | |
| 36 | | nada familiar | **0** ☐☐☐☐☐☐☐ | muito familiar | |
| 37 | | inadequado | **0** ☐☐☐☐☐☐☐ | adequado | |
| 38 | | insuficiente | **0** ☐☐☐☐☐☐☐ | suficiente | |
| | Como base para o desenvolvimento de aplicações | | | | |
| 39 | | inadequado | **0** ☐☐☐☐☐☐☐ | adequado | |
| 40 | | insuficiente | **0** ☐☐☐☐☐☐☐ | suficiente | |
| | **Críticas e sugestões à biblioteca SketchyDynamics** | | | | |

_____

_____

_____

_____

_____

_____

# Anexo D

## Artigo aceite para publicação no 20º Encontro Português de Computação Gráfica

# SketchTester: Analysis and Evaluation of Calligraphic Gesture Recognizers

Abílio Costa
Dep. de Engª Informática, ISEP - IPP
R. Dr. António Bernardino de Almeida, 431, Porto
amfcalt@gmail.com

João P. Pereira
Dep. de Engª Informática / GECAD, ISEP - IPP
R. Dr. António Bernardino de Almeida, 431, Porto
jjp@isep.ipp.pt

**Resumo**

*As interfaces caligráficas apresentam uma forma natural para utilizadores interagirem com aplicações. Dado que o núcleo de uma interface caligráfica é o seu reconhecedor, existe a necessidade de avaliação de diversos reconhecedores antes de optar pela utilização de um. Neste artigo apresentamos uma avaliação de três reconhecedores caligráficos: o reconhecedor de Rubine, o reconhecedor de $1 e o reconhecedor CALI. A avaliação foi realizada com base em amostras caligráficas reais, desenhadas por 32 participantes, com um conjunto de símbolos selecionados para utilização num trabalho futuro. Para além disto, discutimos também alguns aperfeiçoamentos realizados à implementação dos reconhecedores e que ajudaram a obter taxas de reconhecimento superiores. No final, o CALI obteve a melhor taxa de reconhecimento com 94% de sucesso, seguido do reconhecedor de $1 com 87% e finalmente pelo reconhecedor de Rubine com 79%.*

**Abstract**

*Sketch-based interfaces can provide a natural way for users to interact with applications. Since the core of a sketch-based interface is the gesture recognizer, there is a need to correctly evaluate various recognizers before choosing one. In this paper we present an evaluation of three gesture recognizers: Rubine's recognizer, CALI and the $1 Recognizer. The evaluation was done using real gesture samples drawn by 32 subjects, with a gesture set arranged for use in a future work. We also discuss some improvements to the recognizers' implementation that helped achieving higher recognition rates. In the end, CALI had the best recognition rate with 94% accuracy, followed by $1 Recognizer with 87% and finally by Rubine's recognizer with 79%.*

**Keywords**

*Gesture recognition, Calligraphic interfaces, Rubine, CALI, $1 Recognizer.*

## 1. INTRODUCTION

Using pen and paper to draw or sketch something in order to express an idea is very common and also very natural for us. By using this concept in user interfaces one can make the interaction process more natural and spontaneous.

In the future, we aim to develop a programing library to aid in the creation of applications for two-dimensional physics simulations in which the user interacts directly with the scene using a "pen and paper" style interaction. Thus, instead of selecting from a menu which objects compose the scene to be simulated, the user can simply draw the objects directly in the scene. We hope that developing a library that integrates a calligraphic interface and a physics simulation engine will provide a boost for developers to create new applications around this concept, be they for educational purposes, like an application used for teaching physics to students using an interactive whiteboard, or for entertainment purposes, such as a physics-based game where the user draws parts of the scene in order to reach a goal, in the same genre as Crayon Physics Deluxe [Purho09]. These are only two examples of a wide range of possibilities.

The library will support three gestures to draw primitives and other three to define relations between primitives. The first three gestures are used to draw rectangles, triangles and circles, which can be created by drawing these symbols directly. To establish relations between primitives the user can draw a zigzag to connect two primitives with a spring, a cross to pin a primitive over another and a small circle to connect one primitive over another with a rotation axis. Since both the circle primitive and the rotation axis relation use the same gesture[1], we only have in fact five gestures to recognize, presented in Figure 1. Given that the cross is the only gesture that cannot be drawn with only one stroke, we opted to replace it with an alpha, which is an intuitive single-stroke representation of a cross. We chose to use only single-stroke gestures because besides meeting the needs of our library it makes the interaction simpler, since using gestures formed with multi-strokes will force the user to specifically signalize when a gesture is completely drawn or, if

[1] The identification of whether the system should recognize a circle primitive or a rotation axis relation is done by analyzing the size of the gesture and whether or not it is drawn over two existing gestures.

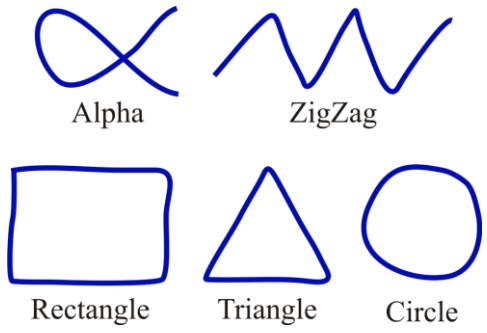



**Figure 1 – Set of gestures used in our work**

using a timer approach, to draw all the gesture's parts within a specific time and wait for the recognition to happen, which may lead to user frustration.

Given the importance of having good gesture recognition, since the user must feel the interaction to be as natural and unrestrictive as drawing with a pen and a paper, we conducted an evaluation of various gesture recognizers in order to select the one that best fits our needs. In this evaluation we have done two sessions to collect samples of the five gestures drawn by various subjects, in order to put the recognizers to test with a wide range of data. This paper describes that evaluation in detail, along with various considerations to achieve higher recognition rates.

In the next section we present an overview of the related work done in the gesture recognition field. This is followed by a description of the application we developed to test gesture recognizers and the implementation of these recognizers. We then present how the evaluation of the gesture recognizers was conducted and discuss its results. Finally we propose potential future developments of this work and present our conclusions.

## 2. RELATED WORK

Given the potential of automatic sketch recognition, a lot of work has been done in order to develop recognizers capable of dealing with the intrinsic ambiguity of hand-drawn sketches. Since there is a great variety of sketch recognition algorithms, it is only natural that there's also diversity in their characteristics. For example, some recognizers only work with single-stroke sketches, while others are oriented towards multi-stroke sketches. Also, whether or not the recognizer can identify sketches independently of their orientation, scale, and drawing order can greatly affect its usefulness in some domains. Another important characteristic is if the recognizer can be trained with new gestures, meaning that it can be easily expanded, or if its gestures are hardcoded, which makes it difficult to change its gesture set to fit a new domain.

Rubine's recognizer [Rubine91], a trainable gesture recognizer, classifies each gesture using a linear classifier algorithm with a set of distinct features. Rubine specifies 11 static geometric features, such as sin/cosine of the initial angle of the gesture, distance between the first and last points, total gesture's length, among others. Rubine also defined two dynamic features: the maximum speed of the gesture and its duration. The recognizer is very flexible since features can be easily added or removed to make the recognizer fit the application needs. For example, [Plimmer07] shows how to improve the recognition and make it independent of gesture's size by removing features that involve absolute sizes and adding new ones that use ratios instead. For the training process, Rubine's recognizer calculates the features of the training templates of each gesture class[2] and computes the weights of those classes based on their features. As pointed by the author, the recognizer requires about 15 training templates per gesture class to be effective, which can make the training process a time consuming task. To recognize a gesture, the algorithm makes a linear combination of the gesture's features and the weights of each class. The class which maximizes that combination is selected as the one that the gesture belongs to. The major limitations of Rubine's recognizer are its sensibility to the drawing direction, scale and orientation and being unable to identify multi-stroke sketches. Pereira et al. [Pereira04] made some modifications to Rubine's recognizer in order to make the algorithm accept multi-stroke sketches, but only when drawn with a constant set of strokes as pointed out by [Stahovich11]. The authors also present a way to make the algorithm insensitive to drawing direction, by doing the recognition twice: first with the original sketch and then with an inverted sketch.

CALI [Fonseca02] is an easy to use multi-stroke recognizer that uses Fuzzy Logic and geometric features to classify gestures independently of their size or orientation. Instead of an individual algorithm, CALI is a complete library that can be easily built into an application. CALI separates gestures into two types: shapes and commands. Shapes can be drawn (and recognized) using solid, dashed and bold[3] lines, while commands are only recognized with solid lines. The recognizer defines a set of geometric rules or features to identify each gesture, like its thinness, aspect ratio, and many others. For example, a Line shape is characterized by being "very thin". In addition to the global geometric features, some gesture classes are also characterized by local features such as the sub-gestures that compose the gesture or whether it has intersections. For example, the Cross command is identified by having two intersecting Line shapes. When an input gesture enters the recognition process its features are computed and checked against each defined gesture rules, using fuzzy sets to find the degree of membership to each rule and therefore to each class. Since CALI is a non-trainable recognizer, adding new gestures is not an easy task, involving hand-coding and analysis of which features characterize and distinguish the gesture. To solve this limitation the authors also present a trainable recognizer and compare three training algorithms: K-Nearest Neighbors, Inductive Decision Tree and Naïve Bayes, the latest being the one with highest training efficiency. Nevertheless, the trainable recognizer has a lowest recognition rate and requires numerous training templates for each gesture class.

[2] A gesture class represents a unique gesture, but can be made from multiple representations of that gesture, i.e. multiple templates.

[3] Bold lines are made from multiple overlapping solid strokes.

In [Wobbrock07], Wobbrock et al. present the $1 Recognizer which aims to be easy to understand and quick to implement. It is insensitive to scale and orientation of sketches, but is sensitive to the drawing direction. One major advantage of $1 Recognizer is the simplicity to add support for new gestures, requiring only one training template per gesture class in most cases. The algorithm has basically four major steps, the first three being applied to both the training templates and the input gesture (the one that is to be recognized), and the fourth step only to the input gesture. The first step is to resample the point path, using simple linear interpolation, so that every gesture (including the training templates) has the same number of points. This enables a direct point-to-point comparison between input gesture and training templates, independently of drawing size and speed. The second step is to rotate the gesture to an orientation that is optimal for matching and thus reduce the recognition time later. This rotation is made based on the angle between the centroid of the gesture and the gesture's first point. The third step is to scale the gesture non-uniformly to a square and translate its centroid to the origin (0,0). The fourth and last step, which is only applied to input gestures, is where the actual recognition happens. The input gesture is compared to each training template to find the average distance between corresponding points and, based on that distance, a score is calculated. The training template with the biggest score is the one that, according to the recognizer, matches the input gesture. Templates with lower score can be used to deal with ambiguity, serving as alternative matches. When the algorithm is computing the average path-distance between an input gesture and a training template, the input gesture is rotated using the Golden Section Search algorithm to find the angle in which that distance is minimized. The authors also explain how to make the recognizer sensitive to scale or orientation, for some or all gesture templates.

In order to solve some of the limitations of the $1 Recognizer, such as not being able to recognizing multi-stroke gestures, sensitiveness to the drawing direction, and problems recognizing uni-dimensional gestures such as lines, Anthony et al. extended it and created the $N Recognizer [Anthony10]. The algorithm starts by computing all the possible combinations of stroke orders and directions for each multi-stroke gesture serving as training template and creates a single-stroke gesture for each combination, by connecting the individual strokes with the order and direction of that combination. These single-stroke gestures are used for comparison with the input gesture, using the same process as the $1 Recognizer, since multi-stroke input gestures are also transformed into single-stroke gestures by connecting their individual strokes by the order they were drawn. The transformations used in $1 Recognizer, such as point resampling, rotation to find the optimal orientation, and translation of the centroid to the origin are also applied by $N to every combined single-stroke gesture. Despite the improvements over the $1 Recognizer, $N has problems recognizing gestures made with more strokes than defined in the training templates. Also, it is not well suited to recognize "messy" gestures like a scratch-out, commonly used for erasing-like actions.

Lee et al. [Lee07] present a trainable graph-based recognizer that is insensitive to orientation, scale and drawing direction and is able to recognize multi-stroke gestures. The recognizer uses statistical models to define symbols, which makes it deal with the small variations associated with hand-drawn gestures naturally. Each gesture is represented by an attributed relational graph, in which nodes depict the type of primitive (line or arc) and its relative length[4]. The edges of the graph represent the geometric relationships between primitives, characterized by the number of intersections, the intersection angle and the intersection location. Gestures are segmented into individual primitives using a technique based on the drawing speed [Stahovich04], meaning that errors in the segmentation process will propagate to the recognition process. When an input gesture arrives, the recognizer compares it to each trained gesture class and computes a dissimilarity score based on six error metrics, each one with a different weight on the resulting score. This dissimilarity score is then converted to a similarity score which is used to identify the gesture class that classifies the input gesture. Since the same gesture can be drawn with varying number of primitives and drawing orders, comparing input gestures and training templates is not straightforward and presents a graph matching problem. To solve this, the authors evaluate and propose five approximate matching techniques. For the training process, an average attributed relational graph is created for each gesture class, by averaging the graphs of multiple training templates. One limitation of this approach is that all training templates of a gesture class must be drawn with a consistent drawing order or consistent orientation.

Vatavu et al. [Vatavu09] present a trainable recognizer that uses elastic deformation energies to classify single-stroke gestures. The recognizer is naturally insensitive to gesture scale and orientation, since the same gesture has similar curvature functions independently of the drawing orientation or size, but is sensitive to drawing direction and starting point within the gesture. To classify a gesture, the recognizer computes its curvature function, based on trajectory analysis, and calculates the alignment cost to each gesture class to find the one that minimizes that cost. Computing the curvature function for each class is done by averaging the functions of multiple training templates for that class.

In [Sezgin05], the authors present a multi-stroke sketch recognizer, based on Hidden Markov Models (HMM), that is capable of recognizing individual sketches in complex scenes even if the scene is not yet completed, i.e. while it is being drawn, and without the need to pre-segment[5] it. On the other hand it can only recognize sketches in their trained orientations, thus being sensitive

[4] The length of the primitive in relation to the total gesture's length.

[5] Pre-segmenting a scene means isolating individual sketches or gestures in the scene.

to orientation. The algorithm first creates discret observation sequences of the scene by identifying various geometric primitives such as sloped lines, horizontal/vertical lines, polylines, among others, with the aid of the Early Sketch Processing Toolkit [Sezgin01]. Then, for each trained HMM, it computes the likelihood for various subsections of the scene given that HMM. Using these likelihoods, it builds a graph in which the shortest path gives the most likely segmentation of the scene, that is, the individual sketches in the scene. Finally, it classifies each segment (the individual sketch) by finding the HMM that maximizes the probability of generating that segment. Since the recognition relies on the stroke order of the trained templates, it is not well suited for domains where the stroke ordering cannot be predicted. Also, because HMMs are suited for sequences, it cannot recognize single-stroke sketches, unless they are pre-segmented.

PaleoSketch [Paulson08] is a low-level non-trainable sketch recognizer for single-stroke primitives. When recognizing a gesture, it starts with a pre-recognition phase, where it removes consecutive duplicate points, cleans drawing noise from the beginning and the end of the gesture, and computes various graphs and values that characterize that gesture. After the pre-recognition process is done, the actual recognition is made. PaleoSketch uses multiple individual sub-recognizers, where each indicates whether or not the input gesture matches a given primitive. Since a gesture can be recognized by more than one sub-recognizer and since each sub-recognizer only returns whether the stroke matches or not, without any "matching score", the results are passed to a hierarchy function for sorting. This function uses a corner finding algorithm to find the minimum number of lines to correctly describe the input gesture and compares that to the minimum number of lines defined for each sub-recognizer's primitive, ordering the results accordingly. PaleoSketch is insensitive to orientation, scale and drawing order, but can only recognize low-level primitives. For more complex gestures (such as rectangles or triangles), one needs to add a higher-level recognizer on top of PaleoSketch.

The need to compare and evaluate the performance of various sketch recognizers is not something new. In [Schmieder09] a toolkit to automatically evaluate recognition algorithms is presented. In addition to sketch data collection and labeling, the toolkit allows the integration of multiple trainable and non-trainable recognizers, which can be tested simultaneously. After testing, the toolkit outputs the results in the form of a Microsoft Excel file or screenshots of the gestures that have been misclassified. As a proof of concept, the authors also present and discuss experimental results of the evaluation of six gesture recognizers: CALI, Microsoft Ink Analyser, $1 Recognizer, Rubine's recognizer with the extended features used in InkKit, PaleoSketch, and a recognizer using Dynamic Time Warping techniques.

## 3. IMPLEMENTATION

In order to test the sketch recognition algorithms, we developed SketchTester, an application that enables us to rapidly prototype and incorporate recognition algorithms. With this application we are able to individually test each algorithm against drawn gestures and immediately see the recognition results (recognized gesture, recognition score, and other alternative matches with lower score). It also provides a graphical interface to add/remove training templates to/from each trainable recognizer, automatically saving them to hard-disk in a recognizer-specific file. As shown in Figure 2, to collect sketch samples from subjects SketchTester offers a window that specifies what gesture should be drawn and, after the gesture is submitted, shows the recognition result of each recognizer. The application keeps asking for random gestures until a predefined number of samples of each gesture is collected, point at which the recognition rates of each recognizer are presented. Each of the submitted sample gesture and corresponding results are saved to a file, so that we can analyze it later.

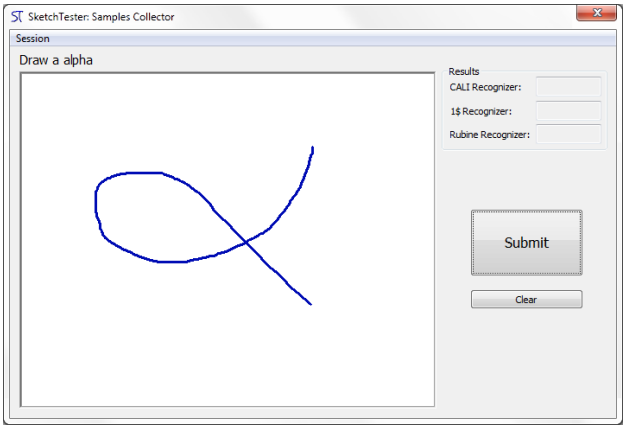



**Figure 2 – Interface used to collect gesture samples from subjects**

SketchTester also incorporates functionality to review collected gesture samples, save them as bitmap files, and extract data from them, such as total number of correctly recognized gestures. Another important feature of SketchTester is the possibility to reprocess multiple files of collected samples. This was particularly useful since we made some improvements to the recognizers after the collection of samples from subjects. By reprocessing those samples, their recognition results were updated according to the improved recognizers.

In SketchTester we implemented three popular recognizers: the $1 Recognizer, Rubine's recognizer, and CALI. The algorithms of the first two recognizers were implemented according to the descriptions given by their authors in [Wobbrock07] and [Rubine91], respectively. In the case of CALI, since it exists in the form of a library, it was only necessary to integrate it with SketchTester. Another important note is that we didn't implement any rejection for gestures with low score, since we always want a result even if it is a low-scored match.

As regards to training templates, $1 Recognizer was first trained with 2 templates for each gesture while Rubine's

recognizer was trained with 15 templates for each gesture. Also, since $1 and Rubine's recognizer are sensitive to the drawing direction, when adding a new training template the application automatically creates a copy of the template but with inverted drawing direction, adding it to the same gesture class as the original template in the case of $1 or to a new class in the case of Rubine's recognizer

CALI was also subject to changes. First we disabled the recognition of gestures we don't need, such as the "copy" gesture, lines, arrows, and some others. Also, since we don't need to discriminate circles from ellipses, we always classify an input gesture as a circle whether it is recognized by CALI as circle or ellipse. The same logic applies to rectangles and diamonds, where they are both classified as rectangles. By removing unneeded gestures from the recognizer and grouping similar gestures we expect to increase the recognition success. Furthermore, we added support for the alpha gesture in CALI by hard-coding a new gesture class in which we defined two features that where selected based on empirical observations: the ratio between the area of the largest quadrilateral and the area of the convex hull ($A_{lq}/A_{ch}$), and the ratio between the perimeter of the largest quadrilateral and perimeter of the convex hull ($P_{lq}/P_{ch}$). The fuzzy sets associated with these features are presented in Figure 3. These two features alone were not enough, because most of the times CALI would classify an input alpha gesture as both alpha and zigzag[6]. To prevent alphas from being misrecognized as zigzags we defined that zigzags can't have any intersections, and that alphas must have an intersection situated at more than 10% away from the limits of the gesture, meaning that each of the alpha's tails must make more than 10% of the gesture.

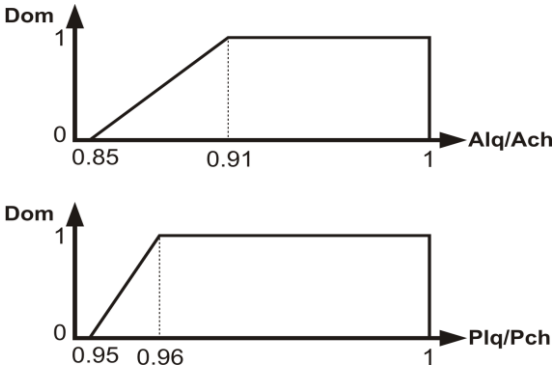



**Figure 3 – Fuzzy sets defined for the alpha gesture**

After analyzing the samples collected from subjects, we found that the implemented recognizers could be improved. In Rubine's recognizer we added three more classes (plus three "duplicates" with inverted drawing direction) per gesture to contemplate different orientations. Figure 4 exemplifies how the rectangle gesture would be represented in each of the four classes. Before the inclusion of new classes to represent various orientations, only the orientation represented in the top-left side of Figure 4 was present.

[6] The zigzag gesture is originally called WavyLine in CALI.

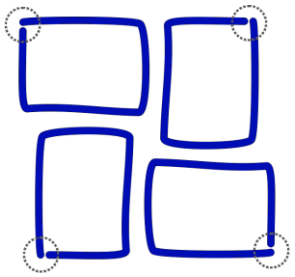


**Figure 4 – Representation of four classes for the rectangle gesture contemplating different orientations; the circles mark start and end points**

For the $1 Recognizer, one possible improvement we found was adding training templates describing rectangles and triangles starting at the middle of an edge, since the first version only had templates starting at vertices. We also added training templates with different starting vertices for these two gestures in order to contemplate every drawing possibility, but without overdoing it, since the algorithm is insensitive to drawing orientation and so we don't need templates of the gesture starting in every vertex.

We have also improved CALI implementation, by setting the minimum size of each of the alpha's tails to 7% instead of 10%. Also, we defined that a zigzag can have intersections as long as the distance along the gesture between the two intersecting points is less than 13% of the total gesture's length. Finally, the distance along the gesture between the two intersecting points in an alpha must be more than 20%.

The rationale behind these improvements will be described in the next section of this paper.

## 4. EVALUATION

In our effort towards finding the recognizer which best fits our purpose and has the greatest recognition rate we collected 1550 gesture samples from 32 subjects.

### 4.1 Method

We conducted two sessions to collect gesture samples, each session with 16 subjects. Using the SketchTester application and following a guide document[7], each subject was asked to draw 10 samples of every of gesture, for a total of 50 samples per subject which provided us with 1550 gesture samples[8]. Both sessions were conducted at our institution where 9 subjects used the institution's desktop computers with traditional mice. The remaining 23 subjects had personal laptops and about 25% of them used the laptop's built-in touchpads and the remaining subjects used conventional mice. All the subjects were MSc students.

Since our evaluation aims to test the recognition of single-stroke gestures with no drawing restrictions, we asked the subjects to freely draw the five gestures presented in Figure 1 with diversified sizes, orientations and shapes.

[7] http://dei.isep.ipp.pt/~i060687/guiao_recolha_caligrafica.pdf

[8] 1550 and not 1600 because two subjects provided less than 50 samples each.

### 4.2 Results

As previously described, we made some improvements to the recognizers and their training templates after analyzing the first recognition results of the collected gesture samples. Then, we reprocessed these samples with the improved recognizers. In this section we will present the recognition results obtained before and after the improvements and discuss how these improvements affect the recognition rates.

To make sure the collected samples fit the domain in which we intend to use the recognizers, we first cleaned them by removing samples that do not match the requested gesture (e.g., the subject was asked to draw a rectangle and drew a circle) and samples drawn incorrectly (e.g., gestures drawn with multiple overlapping lines or gestures that don't represent any of the five required gestures). As expected, cleaning the samples enhanced the recognition rates, especially with CALI whose improvement reached 11%. This enhancement is shown in Figure 5, with a side-by-side comparison of the recognition rates of the three algorithms before and after cleaning the samples, with CALI achieving the highest rates.

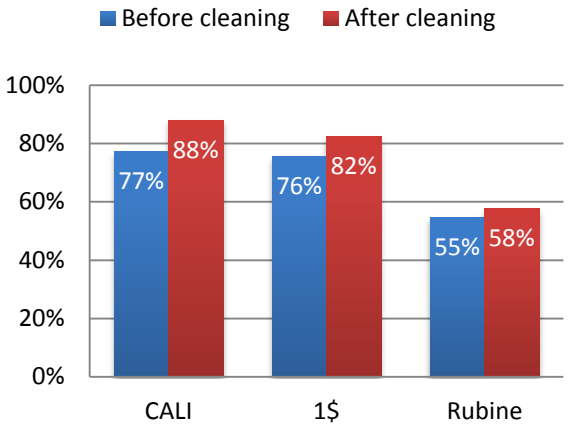



**Figure 5 – Recognitions rates before and after cleaning the collected sample gestures**

In order to understand why Rubine's recognizer obtained such low recognition rates we need to examine the individual recognition rates of each gesture. As presented in Figure 6, the recognition rates of the triangle and circle gestures are 26% and 27% respectively, which is extremely low and far from the remaining gestures' rates. After reviewing individual samples it was clear that what was affecting the recognition of these gestures was the diversification in drawing orientations. We also noticed that users have a tendency to draw rectangles starting with the top-left vertex, which explains why that particular gesture was not suffering much with the orientation sensitivity problem, since the training samples had that same orientation. Interestingly, the drawing orientation had a low impact on the recognition of the zigzag gesture, mainly because it is very distinctive from the remaining gestures in terms of features, and since we did not implement gesture rejection, it is recognized even with a low recognition score.

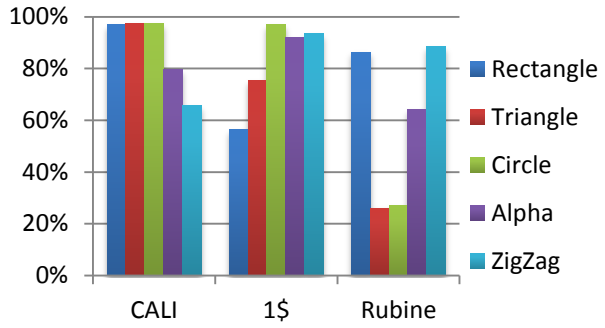



**Figure 6 – Recognition rates of each gesture with each recognizer**

To increase Rubine's recognizer recognition rates we needed to overcome the problem of rotation dependence. As described earlier, the solution was to add new classes representing each gesture in different orientations. As show in Figure 7, this greatly increased the recognition rates and, despite a slight decrease in the recognition of the rectangle gesture, the overall recognition rate with Rubine's recognizer was improved to 79%, against the previous 58%.

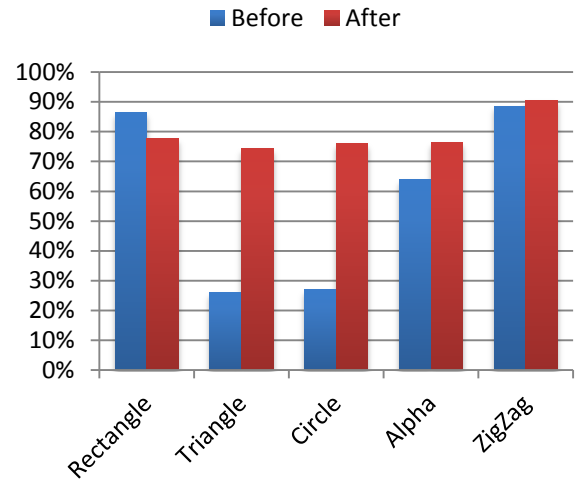



**Figure 7 – Recognition rates for each gesture with Rubine's recognizer, before and after adding gesture classes representing multiple orientations**

Regarding the $1 Recognizer, the most problematic gestures were rectangles and triangles, with recognition rates of 56% and 75% respectively. We found that both gestures needed training templates with starting points at different vertices and also at the middle of edges. We also found that a training template representing right triangles was needed. After these additions to $1 Recognizer's training templates the recognition rates of rectangles and triangles were improved and, despite a decrease in the recognition rates of circles and alphas, the global recognition rate of the algorithm was improved to 87% against the previous 82%. We then tried to improve the recognition of circles and alphas, since they were affected in these changes, but found no success. Figure 8 shows the recognition rates for each gesture before and after the addition of the new training templates.

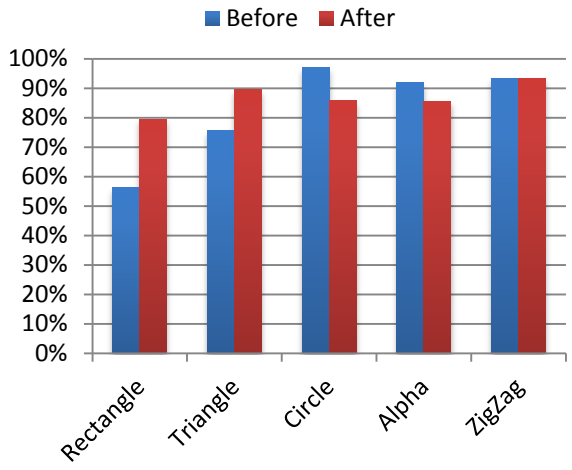



**Figure 8 - Recognition rates for each gesture with $1 Recognizer, before and after improving its training templates**

Despite CALI already reaching a fairly high recognition rate of 88%, analyzing the results for individual gesture rates suggests that it could be further improved. As show in Figure 6, the alpha and zigzag gestures have low recognition rates when compared to the other gestures, which may be due to the changes we've made to the original CALI source code to include the alpha gesture, discussed previously. Further investigation showed that many alphas with short tails were simply not recognized. After detailed examination of the collected gestures we found that 7% was the optimal value as the minimum relative size of the gesture's tails. In relation to the zigzag gesture, we found that many subjects made small intersections when drawing zigzags and since we first specified that zigzags must not have intersections, many were recognized as alphas. After inspecting the collected samples, we specified that zigzags can have intersections but only when the two intersecting points are not more than 20% away from each other. We also updated the alpha gesture so that the distance between the two intersecting points must be more than 20% of the total gesture's length. As presented in Figure 9, these changes to CALI had a positive impact on the recognition of alphas and zigzags, without affecting the other gestures.

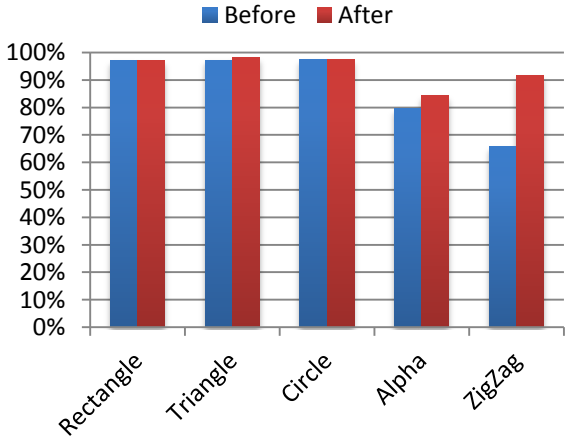



**Figure 9 – Recognition rates for each gesture with CALI, before and after updating the recognition source code of the alpha and zigzag gestures**

By analyzing the first recognition results of the samples collected from subjects, we were able to identify flaws in our implementation of the recognizers. As show in Figure 10, our attempts to correct these flaws where successful and all the recognizers reached higher recognition rates, with CALI achieving the best rates, followed by $1 and then by Rubine's recognizer. Also, Figure 10 shows that even after improving the recognizers, the respective standard deviations are high, which is caused by a significant disparity in the individual recognition rates of each gesture. This suggests the possibility that the recognizers can still be improved. If we were able to improve the recognition of the gestures with lowest rates, the overall recognition rate would increase and the standard deviation would decrease, meaning that the recognizer would be recognizing all the gestures in a balanced way.

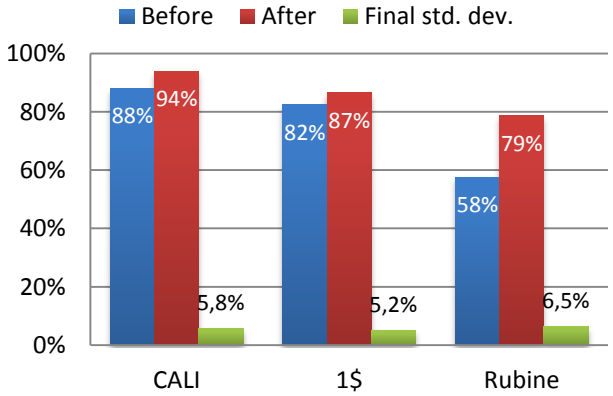



**Figure 10 – Overall recognitions rates before and after improving the recognizers, along with the standard deviation of the final rates**

In [Schmieder09], to show the potential of their automated recognizer evaluation toolkit, the authors conducted an experiment in which they evaluate six recognizers, including the same three we evaluated, with three basic single-stroke gestures[9]: circle, rectangle and line. In their experiment, Rubine's recognizer had the best recognition with a 96% success rate, followed by $1 Recognizer with 89% and CALI with 84%. While at first these results can seem to contradict our evaluation's results, with CALI and Rubine's recognizer inverting positions, they can be easily explained. While we used the original feature set in Rubine's algorithm, they implemented the extended feature set used in InkKit [Plimmer07], which explains why Rubine's recognizer achieved such high recognition rates. In respect to CALI, they consider circles and ellipses as independent gestures, unlike our evaluation where we don't need to differentiate these two gestures and consider both as one. In their results, there have been 94 circles misclassified as ellipses, in a total of 730 evaluated gestures. If we consider these 94 ellipses as being correctly classified, effectively merging circles and ellipses, the recognition rate for CALI increases to 96%. If we also merge rectangles and diamonds it rises to 98%. These

[9] There's also a second experiment which we won't cover on this paper because it is done with Entity Relationship (ER) diagrams instead of basic gesture shapes.

rates are close to those obtained in our evaluation. Regarding $1 Recognizer, both evaluations yield similar results. Finally, it is important to notice that since they used fewer gestures than us it is normal that they obtained higher recognition rates, as the misclassifications tend to increase with the number of gestures.

## 5. FUTURE WORK

In the future we could implement more recognizers in SketchTester, and even improve Rubine's features as described in [Plimmer07], in order to also evaluate them against the collected samples. The inclusion of other gestures could also be subject of study if the library is extended beyond five gestures.

An evaluation of the recognizers with gestures collected using touchscreens or interactive whiteboards would also be an interesting evaluation, since these are the kind of devices that most benefit from calligraphic interfaces.

Since the improvements to the recognizers were made and evaluated using the same set of gesture samples, it would be important to re-evaluate these improvements with new samples, in order to confirm that they are valid not only for our sample set but also to generic samples.

Finally, although the most relevant result of recognition is the gesture with the highest score, a study considering the first two or three high-score results could be relevant in cases where the application presents a list of alternative matches to solve ambiguity.

## 6. CONCLUSION

In this work we've collected sample gestures from various subjects and evaluated three popular gesture recognizers to find the one that best fits in the interaction layer of our physics simulation library. We've also presented some insights on how the implementations of these recognizers can be improved to yield better results. Also, despite the specificity of the tested data, our work can serve as a base to others exploring gesture recognizers.

To conclude, both CALI and $1 are good candidates for our library since both achieved high recognition results. Also, we are confident that if the improved features described in [Plimmer07] where implemented in Rubine's recognizer, it could have results as good as the results of the other two recognizers. Nevertheless, since CALI archived higher rates it shall be selected to integrate our library.

## 7. ACKNOWLEDGMENTS

We wish to thank all the participants in the recognizer evaluation sessions. We are also grateful to Manuel J. Fonseca for providing us the CALI library source code.

## 8. REFERENCIES

[Anthony10] Lisa Anthony, Jacob O. Wobbrock. 2010. A lightweight multistroke recognizer for user interface prototypes. In *Proceedings of Graphics Interface 2010* (GI '10), 245-252.

[Fonseca02] Manuel J. Fonseca, César Pimentel, and Joaquim A. Jorge. 2002. CALI: An online scribble recognizer for calligraphic interfaces. In *AAAI Spring Symposium on Sketch Understanding*, 51-58.

[Lee07] WeeSan Lee, Levent Burak Kara, and Thomas F. Stahovich. 2007. An efficient graph-based recognizer for hand-drawn symbols. In *Computers & Graphics 31*, 554-567.

[Paulson08] Brandon Paulson and Tracy Hammond. 2008. PaleoSketch: accurate primitive sketch recognition and beautification. In *Proceedings of the 13th international conference on Intelligent user interfaces* (IUI '08), 1-10

[Pereira04] Pereira, J. P., Branco, V. A., Jorge, J. A., Silva, N. F., Cardoso, T. D., and Ferreira, F. N. 2004. Cascading recognizers for ambiguous calligraphic interaction. In *Eurographics Workshop on Sketch-Based Interfaces and Modeling*.

[Plimmer07] Beryl Plimmer and Isaac Freeman. 2007. A toolkit approach to sketched diagram recognition. In *Proceedings of the 21st British HCI Group Annual Conference on People and Computers: HCI...but not as we know it* (BCS-HCI '07), vol. 1.

[Purho09] Petri Purho. 2009. Crayon Physics Deluxe. <http://crayonphysics.com/>

[Rubine91] Rubine, Dean. 1991. Specifying gestures by example. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques* (SIGGRAPH '91), 329-337.

[Schmieder09] Paul Schmieder, Beryl Plimmer, and Rachel Blagojevic. 2009. Automatic evaluation of sketch recognizers. In *Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling* (SBIM '09), 85-92.

[Sezgin01] Tevfik Metin Sezgin, Thomas Stahovich, and Randall Davis. 2001. Sketch based interfaces: early processing for sketch understanding. In *Proceedings of the 2001 workshop on Perceptive user interfaces* (PUI '01), 1-8.

[Sezgin05] Tevfik Metin Sezgin and Randall Davis. 2005. HMM-based efficient sketch recognition. In *Proceedings of the 10th international conference on Intelligent user interfaces* (IUI '05), 281-283.

[Stahovich04] Thomas F. Stahovich. 2004. Segmentation of pen strokes using pen speed. In *Proceedings of 2004 AAAI fall symposium on making pen-based interaction intelligent and natural*, 152-158.

[Stahovich11] Thomas F. Stahovich. 2011. Pen-based Interfaces for Engineering and Education. In *Sketch-based Interfaces and Modeling*, 119-152.

[Vatavu09] Radu-Daniel Vatavu, Laurent Grisoni, and Stefan-Gheorghe Pentiuc. 2009. Gesture Recognition Based on Elastic Deformation Energies. In *Gesture-Based Human-Computer Interaction and Simulation*, vol. 5085, 1-12.

[Wobbrock07] Jacob O. Wobbrock, Andrew D. Wilson, and Yang Li. 2007. Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. In *Proceedings of the 20th annual ACM symposium on User interface software and technology* (UIST '07), 159-168.

# Anexo E

**Artigo submetido para publicação na 8ª Conferência Internacional sobre Teoria e Aplicações da Computação Gráfica (GRAPP 2013)**

# SketchyDynamics
## *A Sketch-Based Library for the Development of Physics Simulation Applications*

Abílio Costa[1] and João P. Pereira[1,2]
[1] *Computer Science Department, School of Engineering (ISEP), Polytechnic of Porto, R. Dr. António Bernardino de Almeida 431, Porto, Portugal*
[2] *Knowledge Engineering and Decision Support Group (GECAD), School of Engineering, Polytechnic of Porto, R. Dr. António Bernardino de Almeida 431, Porto, Portugal*
*amfcosta13@gmail.com, jjp@isep.ipp.pt*

Keywords: Sketch-Based Interfaces : Gesture Recognition : Physics Simulation : Rigid Body Dynamics : Education : Entertainment.

Abstract: Sketch-based interfaces provide a powerful, natural and intuitive way for users to interact with an application. By combining a sketch-based interface with a physically simulated environment, an application offers the means for users to rapidly sketch a set of objects, like if they are doing it on piece of paper, and see how these objects behave in a simulation. In this paper we present SketchyDynamics, a library that intends to facilitate the creation of applications by rapidly providing them a sketch-based interface and physics simulation capabilities. SketchyDynamics was designed to be versatile and customizable but also simple. In fact, a simple application where the user draws objects and they are immediately simulated, colliding with each other and reacting to the specified forces, can be created with only 3 lines of code.

## 1 INTRODUCTION

Using pen and paper to draw or sketch something in order to express an idea is very common and also very natural for us. By using this concept in user interfaces one can make the interaction process more natural and spontaneous.

In this paper we propose SketchyDynamics, a programing library to aid in the creation of applications for two-dimensional physics simulations in which the user interacts directly with the scene using a "pen and paper" style interaction. Thus, instead of selecting from a menu which objects compose the scene to be simulated, the user can simply draw the objects directly in the scene. We hope that developing a library that integrates a calligraphic interface and a physics simulation engine will provide a boost for developers to create new applications around this concept, be they for educational purposes, like an application used for teaching physics to students using an interactive whiteboard, or for entertainment purposes, such as a physics-based game where the user draws parts of the scene in order to reach a goal. These are only two examples of a wide range of possibilities.

The library supports three gestures to draw rigid bodies and other three to define connections between them. The first three gestures are used to draw rectangles, triangles and circles, which can be created by drawing these symbols directly. To establish connections between bodies the user can draw a zigzag to connect two bodies with a spring, a cross to pin a body over another and a small circle to connect one body over another with a rotation axis. Since both the circle body and the rotation axis relation use the same gesture, we only have in fact five gestures to recognize, presented in Figure 1.

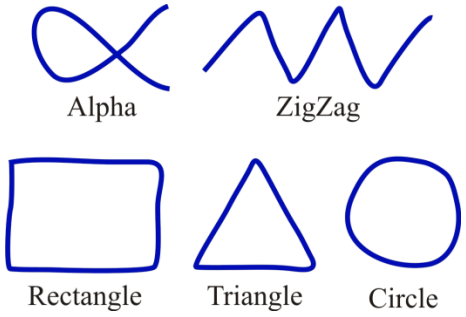



Figure 1 - Set of gestures used in our library

Given that the cross is the only gesture that cannot be drawn with only one stroke, we opted to replace it with an alpha, which is an intuitive single-stroke representation of a cross. We chose to use only single-stroke gestures because besides meeting the needs of our library it makes the interaction simpler, since a single stroke converts into a single gesture.

Due to the high importance of having good gesture recognition, since the user must feel the interaction to be as natural and unrestrictive as drawing with a pen and a paper, the gesture recognizer used in SketchyDynamics was selected based on previous evaluation.

Although there are already several applications that combine physics simulation with a sketch-based interface, most of them have a specific scope and audience. As a library, SketchyDynamics is intended to be used in different types of applications and does not have a definite scope. We hope that our work helps developers create new and exciting applications with little effort in combining the physics simulation with the sketch-based interface.

In the next section we present an overview of various works done in the sketch recognition field and also works that combine sketch-based interfaces with rigid body physics simulation. Section 3 gives a little insight on a previous evaluation whose purpose was to select the sketch recognizer that best integrates with our library. In section 4 we present our library, its technical characteristics, along with its functionality. Section 5 discusses a preliminary informal evaluation and section 6 concludes this paper and presents potential future work.

# 2 RELATED WORK

This section presents some of the related work in the sketch-based interfaces domain and is divided into two subsections. The first subsection will address the work done in the sketch recognition field, while the second presents some examples of applications that result from the combination of sketch-based interfaces with rigid body physics simulation.

## 2.1 Sketch recognizers

Given the potential of automatic sketch recognition, a lot of work has been done in order to develop recognizers capable of dealing with the intrinsic ambiguity of hand-drawn sketches. Since there is a wide variety of sketch recognition algorithms, it is only natural that there's also diversity in their characteristics. For example, some recognizers only work with single-stroke sketches, while others are oriented towards multi-stroke sketches. Also, whether or not the recognizer can identify sketches independently of their orientation, scale, and drawing order can greatly affect its usefulness in some domains. Another important characteristic is if the recognizer can be trained with new gestures, meaning that it can be easily expanded, or if its gestures are hardcoded, which makes it difficult to change its gesture set to fit a new domain.

Rubine's recognizer (Rubine, 1991), a trainable gesture recognizer, classifies each gesture using a linear classifier algorithm with a set of distinct features. Rubine specifies 11 static geometric features, such as the first and last points of the gesture or the total gesture's length, and two dynamic features, the maximum speed of the gesture and its duration. The recognizer is very flexible since features can be easily added or removed to make the recognizer fit the application needs. For example, Plimmer and Freeman (2007) have shown how to improve the recognition and make it independent of gesture's size by removing features that involve absolute sizes and adding new ones that use ratios instead. The major limitations of Rubine's recognizer are its sensibility to the drawing direction, scale and orientation and being unable to identify multi-stroke sketches. Pereira et al. (2004) made some modifications to Rubine's recognizer in order to make the algorithm accept multi-stroke sketches, but only when drawn with a constant set of strokes as pointed out by Stahovich (2011). The authors also present a way to make the algorithm insensitive to drawing direction.

CALI (Fonseca, Pimentel, & Jorge, 2002) is an easy to use multi-stroke recognizer that uses Fuzzy Logic and geometric features to classify gestures independently of their size or orientation. Instead of an individual algorithm, CALI is a complete library that can be easily built into an application. CALI separates gestures into two types: shapes and commands. Shapes can be drawn (and recognized) using solid, dashed and bold[1] lines, while commands are only recognized with solid lines. In CALI, each gesture is identified by a set of geometric rules or features, like its thinness, aspect ratio, and many others. For example, a Line shape is characterized by being "very thin". Additionally, some gestures are also characterized by local features, like the

[1] Bold lines are made from multiple overlapping solid strokes.

Cross command whose local feature is being made of two intersecting Line shapes. When an input gesture enters the recognition process its features are computed and checked against each defined gesture rules, using fuzzy sets to find the degree of membership to each rule and therefore to each class. Since CALI is not trainable, adding new gestures is not an easy task, involving analysis of which features characterize and distinguish the new gesture and hand-coding these features. To solve this limitation the authors also present a trainable recognizer but it has a lower recognition rate and requires numerous training templates for each gesture class[2].

In (Wobbrock, Wilson, & Li, 2007), the authors present the $1 Recognizer which aims to be easy to understand and quick to implement. It is insensitive to scale and orientation of sketches, but sensitive to the drawing direction. One major advantage of $1 Recognizer is the simplicity to add support for new gestures, requiring only one training template per gesture class in most cases. The algorithm has basically four major steps, the first three being applied to both the training templates and the input gesture (the one that is to be recognized), and the fourth step only to the input gesture. The first step is to resample the point path, so that every gesture (including the training templates) has the same number of points. This enables a direct point-to-point comparison between input gesture and training templates. The second step is to rotate the gesture to an orientation that is optimal for matching and thus reduce the recognition time later. The third step is to scale the gesture non-uniformly to a square and translate its centroid to the origin. The fourth and last step, which is only applied to input gestures, is where the actual recognition happens. The input gesture is compared to each training template to find the average distance between corresponding points and, based on that distance, a score is calculated. The training template with the biggest score is the one that, according to the recognizer, matches the input gesture. The authors also explain how to make the recognizer sensitive to scale or orientation, for some or all gesture templates.

In order to solve some of the limitations of the $1 Recognizer, such as not being able to recognizing multi-stroke gestures, sensitiveness to the drawing direction, and problems recognizing uni-dimensional gestures such as lines, Anthony & Wobbrock (2010) extended it and created the $N Recognizer. The algorithm computes all the possible combinations of stroke orders and directions for each multi-stroke gesture serving as training template and creates a single-stroke gesture for each combination. These single-stroke gestures are then used for comparison against the input gesture, using the same process as the $1 Recognizer, since multi-stroke input gestures are also transformed into single-stroke gestures by connecting their individual strokes by the order they were drawn. Despite the improvements over the $1 Recognizer, $N has problems recognizing gestures made with more strokes than those used in the training templates. Also, it is not well suited to recognize "messy" gestures like a scratch-out, commonly used for erasing-like actions.

Lee, Kara, & Stahovich (2007) present a trainable graph-based recognizer that is insensitive to orientation, scale and drawing direction and is able to recognize multi-stroke gestures. The recognizer uses statistical models to define symbols, which makes it deal with the small variations associated with hand-drawn gestures naturally. Each gesture is represented by an attributed relational graph, in which nodes depict the type of primitive (line or arc) and its relative length[3]. The edges of the graph represent the geometric relationships between primitives, characterized by the number of intersections, the intersection angle and the intersection location. In order to classify input gestures, the algorithm compares them to each trained gesture class and computes a dissimilarity score based on six error metrics, each one with a different weight on the resulting score. For the training process, an average attributed relational graph is created for each gesture class, by averaging the graphs of multiple training templates. One limitation of this approach is that all training templates of a gesture class must be drawn with a consistent drawing order or consistent orientation.

Vatavu, Grisoni, & Pentiuc (2009) present a trainable recognizer that uses elastic deformation energies to classify single-stroke gestures. The recognizer is naturally insensitive to gesture scale and orientation, since the same gesture has similar curvature functions independently of the drawing orientation or size, but is sensitive to drawing direction and starting point within the gesture. To classify a gesture, the recognizer computes its curvature function, and calculates the alignment cost

[2] A gesture class represents a unique gesture, but can be made from multiple representations of that gesture, i.e. multiple templates.

[3] The length of the primitive in relation to the total gesture's length.

to each gesture class to find the one that minimizes that cost. Computing the curvature function for each class is done by averaging the functions of multiple training templates for that class.

In (Sezgin & Davis, 2005), the authors present a multi-stroke sketch recognizer, based on Hidden Markov Models (HMM), that is capable of recognizing individual sketches in complex scenes even if the scene is not yet completed, i.e. while it is being drawn, and without the need to pre-segment it[4]. On the other hand it can only recognize sketches in their trained orientations, thus being sensitive to orientation. The algorithm first creates discret observation sequences of the scene by identifying various geometric primitives such as sloped lines, horizontal/vertical lines, and polylines, among others. Then, for each trained HMM, it computes the likelihood for various subsections of the scene given that HMM. Using these likelihoods, it builds a graph in which the shortest path gives the most likely segmentation of the scene, that is, the individual sketches in the scene. Finally, it classifies each segment (the individual sketch) by finding the HMM that maximizes the probability of generating that segment. Since the recognition relies on the stroke order of the trained templates, it is not well suited for domains where the stroke ordering cannot be predicted. Also, because HMMs are suited for sequences, it cannot recognize single-stroke sketches, unless they are pre-segmented.

## 2.2 Physics Simulation with Sketch-Based interfaces

The idea of using a sketch-based interface to create and manipulate a physics simulated scene is not something new. For example, ASSIST (Alvarado & Davis, 2001) is able to recognize sketches and convert them to mechanical objects which can then be simulated. The system recognizes circles and straight-line polygons (simple or complex) made of single or multiple strokes. The recognition is done incrementally, while the user is drawing, instead of being done only when the user finishes the drawing. In addition to make the system feel quicker since the interpretation effort is done while the user is drawing, incremental interpretation also gives an instantaneous feedback to the user, since hand-drawn lines are converted to straight lines and colored according to the type of object recognized. When interpreting user input, ASSIST constantly evaluates new input strokes along with previous strokes, since a new stroke can bring new information about the object that the user is constructing. Sketch interpretation is done based on three stages: recognition, reasoning and resolution. The recognition stage identifies all the possible interpretations of the sketch. The reasoning stage scores each interpretation based on a set of heuristics such as temporal evidence, domain knowledge and user feedback. Finally, the resolution stage selects the best interpretation based on its score. When an improper interpretation is made, the user is able to correct it using a list of alternative interpretations. Users can also pin one object over another with a rotational axis by drawing a small circle, or anchor objects to the background by drawing a small cross, and thus making them static during the simulation. After finishing the sketch, the user can press a "Run" button to transfer his design to a 2D mechanical simulator that runs and displays the simulation of the designed scene.

Another application, "Free-Hand Sketch Recognition for Visualizing Interactive Physics" (Kamel, Shonoda, Refeet, & Nabil, n.d.) enables users to draw simple 2D objects and simulate how these objects behave in 3D. The application is able to recognize four types of objects: lines, circles, rectangles, and triangles. When the gesture cannot be recognized a small dialog is presented, requesting the user to specify the desired gesture. After creating an object, the user is able to anchor it so that it remains static and does not move during the simulation. The design process consists of three modes: the "Ink" mode where the user can draw new objects; the "Select" mode, where a circle selects the enclosed objects; and the "Erase" mode, used to remove objects. Despite the designing being done in 2D, the physics simulation is 3D and the user is able to move the camera and also move objects in 3D space.

There are also games that take advantage of a sketch-based interface and a physics simulated environment to entertain the player. One popular example is Crayon Physics Deluxe (Purho, 2009), a puzzle game where the main objective is to guide a ball so that it touches all the stars in each level. Instead of controlling the ball directly, the user needs to draw objects that influence the ball, leading it to the stars. The user can draw rigid bodies with any shape and connect them with pivot points and ropes. Since the simulation is always running, sketched objects are simulated and interact with other objects right after being drawn. In some levels

[4] Pre-segmenting a scene means isolating individual sketches or gestures in the scene.

there are also rockets that fire after some other object collides with them, and that can be used to achieve the level's objective. The game has a "children's drawing" theme, with a background that resembles a yellow paper sheet and crayon-like sketches, both characteristics that make it successfully adopt the pen-paper paradigm. Crayon Physics Deluxe also includes a level editor and an online playground, so users can create their own levels and submit them online.

# 3 SKETCH-BASED RECOGNITION EVALUATION

Previously to this work, we have conducted an evaluation of three gesture recognizers in order to select the one that best fits our needs (Costa & Pereira, 2012). The evaluation was based on real gesture samples drawn by 32 subjects, with a gesture set specifically arranged for our library.

For the evaluation process we developed an application to collect samples from subjects, process them, and compute the recognition results. With this tool we evaluated Rubine's recognizer, CALI and the 1$ Recognizer, concluding that for our gesture set CALI achieved the highest recognition rates.

With this evaluation we were also able to improve recognition rates by tweaking the templates and the recognizer's implementation to our specific gesture set.

# 4 THE SKETCHYDYNAMICS LIBRARY

SketchyDynamics is a programing library that aims to simplify the implementation of 2D physics simulation applications with sketch-based interfaces. Using 2D graphics and physics simulation means that the user sketch (in 2D) is interpreted and represented as a 2D object instead of being transformed to a 3D representation. This closely resembles the pen-paper paradigm and simplifies user interaction.

Out of the box, SketchyDynamics provides an interface for the user to interact with an application along with recognition and processing of user actions such as drawing, moving, scaling and removing rigid bodies and their joints. SketchyDynamics also deals with the physics simulation of these elements and visually represent them on screen along with other user interface elements. Thus, a developer can integrate these features in an application with almost no effort.

## 4.1 Architecture

A major concern when designing SketchyDynamics was to make it versatile, so that developers can create all kind of applications, but at the same time simple enough to enable rapid prototyping. For example, with only 3 lines of source code a developer can create a simple test application where the user can draw objects and they are simulated, colliding with each other and reacting to the specified "gravity" (which can be positive and/or negative, horizontal and/or vertical and have any force value). With a dozen more lines the developer is able to add a background body where the user is able to attach objects, or a ground body so that drawn bodies have something to fall onto.

As stated previously, we use CALI as the gesture recognizer since it yielded good results in our evaluations.

For the physics simulation SketchyDynamics uses the Box2D physics engine (Catto, n.d.). Box2D is an open-source 2D physics engine and is responsible for simulating the motion of the rigid bodies and dealing with collisions between them. Despite using Box2D, SketchyDynamics does not encapsulate it and hide it from the programmer. Instead programmers have access to all Box2D objects and functionality so they are able to parameterize them according to the applications needs. For example, instead of using the default body density set by SketchyDynamics, the application is able to change the density for each drawn body.

Although bodies and joints are created automatically by the library when the user draws them, the application is also able to programmatically create and remove them (which also creates and removes the visual representation of these objects). Furthermore, SketchyDynamics also gives the application full control over the simulation state, by providing functions to pause and resume it.

To render the bodies simulated by Box2D and any other visual elements we used the OpenGL API. Despite using OpenGL as the default graphics API, SketchyDynamics was designed so that a developer can easily use another API, which is useful for devices that have OpenGL ES, or for applications that depend on Direct3D functionality. This is achieved by ensuring that all OpenGL-specific code is encapsulated in a few classes, thus creating a conceptual abstraction layer. Thereby, a developer

only has to implement a version of these few classes conforming to the desired API, without the need to make deep changes to SketchyDynamics main functionality.

While implementing the OpenGL abstraction we took the opportunity to add some "graphics library"-like functionality. For example, a programmer can easily create polygons by defining their vertices and then apply geometric transformations on them, toggle their visibility on screen, among other operations, all done in an object-oriented manner. Additionally, the library provides scene query functionality, allowing easy retrieval of the objects that lie in a specific point, which is useful to find if a user has pressed over a user interface control, for example. To render these objects SketchyDynamics offers three rendering queue layers so that each individual object can be drawn on the background, on the front (as a user interface element) or in the middle of the two. Furthermore, the depth[5] of each object inside each layer can also be specified. SketchyDynamics also offers a material manager class so that the programmer can easily set textures on all objects by only specifying the texture file, letting the library do all the texture loading and setup work.

Another design decision that resulted from the OpenGL abstraction was the incorporation of the window creation process inside SketchyDynamics. This also reduces the effort on the developer side, since he does not need to worry about how the window is created. Moreover, in addition to normal windows, the library supports the creation of child windows, embedded inside another window or form created by the application. Regardless of the window type, SketchyDynamics delivers events received by the window, like mouse and keyboard inputs, to the application using the observer pattern (Observer pattern, n.d), thus letting the developer take actions based on user input.

## 4.2 User Interaction

In order to best represent the pen-paper paradigm, the user interaction was designed to take advantage of systems with a touchscreen and stylus. Thus, the user only needs to press and move the stylus to interact with the system, without needing extra buttons[6]. Furthermore, no menus are used and most of the interaction is done by sliding the stylus across the screen. Although it was designed with that type of devices in mind, SketchyDynamics also works well with a traditional computer mouse.

There are two types of objects the user is able to create: bodies and joints. Bodies are rigid objects that are simulated according to physics laws and joints are used to connect bodies. Figure 2 shows various bodies and three types of joints.

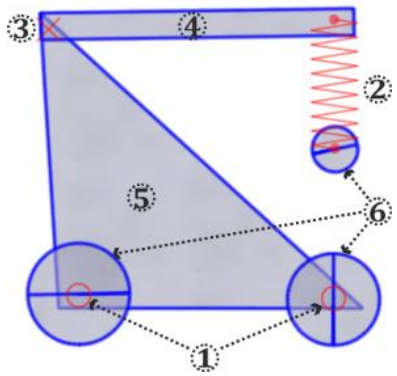



Figure 2 – Various types of joints and bodies: 1) revolute joints; 2) spring joint; 3) weld joint; 4) rectangular body; 5) triangular body; 6) circular bodies.

It is also important for the user to be able to manipulate created objects to a certain point so SketchyDynamics lets the user change an object's position, scale and orientation, or even delete it.

### 4.2.1 Creating

The creation of an object, be it a body or a joint, is done by drawing it. So, for example, if users want to create a rectangle body, they simply draw the rectangle on the screen, with the desired dimensions and on the desired position. SketchyDynamics recognizes the rectangle and its properties like size and orientation, and creates the physical and visual representations of it.

SketchyDynamics supports four types of bodies: rectangles, triangles, circles and the freeform. When the user input is recognized as a rectangle, triangle or circle, it is represented in a beautified manner, as illustrated in Figure 3. Otherwise, when the input is not recognized, it is interpreted as a freeform and represented in a simplified form (with fewer vertices) for performance reasons.

[5] Depth here means the order in which objects are presented on screen, making them appear behind or in front of other objects.

[6] In a traditional mouse system this means that only the left mouse button is needed.

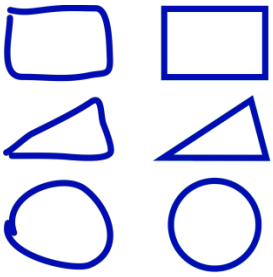


Figure 3 - Example of drawn shapes (left) and respective beautified representations (right).

The user can also connect two bodies with three different joint types: weld, revolute and spring. Weld joints connect two bodies at a specific anchor point, preventing any relative movement between them. Like weld joints, a revolute joint connects two overlapping bodies at a specific point but allows the bodies to rotate freely around that point. Spring joints try to keep a constant distance between two connected bodies, based on the distance at the time the joint was created, stretching and shrinking like a real spring.

Just like creating bodies, the creation of joints is done by drawing them. Drawing an alpha gesture over two bodies connects them with a weld joint with an anchor at the gesture's intersection, while drawing a small circle creates a revolute joint anchored at the circle's center. To create a spring joint, the user draws a zigzag gesture starting in one body and ending in another one, defining the two spring's anchor points as the start and end points of the gesture.

Regarding the visual representation of joints, the weld and revolute joints are represented by a small cross and by a small circle, respectively, on the joint anchor point while the spring joint is displayed as a zigzag line starting in one anchor point and ending on the other, stretching and shrinking subject to the distance between the bodies. The object presented in Figure 2 was constructed using joints of the three types.

In order to better deal with the ambiguity in hand-drawn gestures, a guesses list is presented whenever the user executes a gesture. The guesses list shows all the available objects so that the user can choose an object other than the recognized one. The objects corresponding to gestures identified as matching by CALI recognizer appear bigger and first in the list, since they are the most probable choices, followed by the remaining objects. Except for the objects identified as matches by CALI, which are ordered by their recognition score, the objects always appear ordered in a consistent manner. The guesses list feature can be disabled by the developer, in which case the most probable object is always selected.

Depending on the application-specific setup passed to SketchyDynamics, objects can be created while the physics simulation is in a paused state or while it is running and thus making other objects react instantly to the new object. This instantaneous simulation mode is useful for applications where the user interacts with a live environment as usually happen in games.

### 4.2.2 Selecting

For an object to be manually manipulated by the user, it first needs to be selected. When any object is selected the physics simulation is paused so that the user can easily edit it without being interrupted by other moving bodies. If the simulation was running before the selection of an object, it will resume after all objects are unselected.

Objects are selected by tapping on them with the stylus[7], and can be deselected with the same action. This makes selecting multiple objects an intuitive process since users only need to keep tapping on the other objects they want to select. It is also possible to unselect individual objects when there are multiple objects selected. When an object is selected, its lines assume a distinctive color, returning to the original color after being unselected. As shown in Figure 4, this gives a clear feedback regarding the object's state. Also, tapping on an area of the screen with no objects or on an object configured as non-selectable, deselects all selected objects. Non-selectable objects are useful to create the application's scenery, which the user cannot manipulate but may be able to interact with, for example by connecting a user-made body to a scenery object.

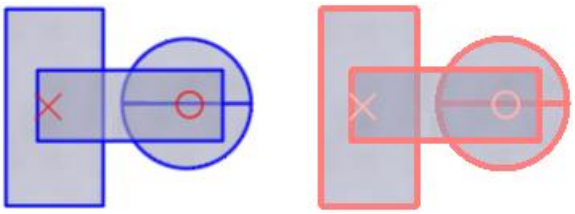


Figure 4 - Set of objects in unselected (left) and selected (right) states

[7] Tapping with a stylus is equivalent to a left click with a traditional computer mouse.

When there are multiple bodies connected by joints and one of them is selected, all the other connected bodies are automatically selected, as long as they are selectable objects. This feature was introduced in order to improve the usability of the system, since we found that when multiple bodies are connected the user typically wants to manipulate them as a whole. If only one body from a set of connected bodies should be selected, then its joints should be removed before selecting it.

### 4.2.3 Moving

A selected body or joint can be moved by pressing over it and dragging the stylus. The object will move in sync with the stylus as long as the user keeps it pressed on the screen. When the user lifts the stylus of the screen, the objects stop moving and assume their new position.

When there are multiple selected objects they all move in a synchronized manner, regardless of which object was pressed by the stylus.

### 4.2.4 Scaling and Rotating

Scaling and rotation of bodies is done simultaneously in a single action. As the action to move an object, scaling and rotation is done by pressing and dragging the stylus, but instead of pressing inside the selected body, the user needs to press outside it. As the user drags the stylus, the selected bodies scale and rotate based on the stylus initial and current positions. Only bodies can be rotated or scaled, so this operation is not applicable to joints[8].

The scale factor is calculated based on the current distance from the stylus position to the body center and the initial distance (before dragging the stylus). Regarding rotation, it is done based on the angle between two imaginary lines: the line from the current stylus position to the body's center, and the initial line (before dragging the stylus). Thus, moving the stylus closer or farther from the body scales it while moving the stylus around the body rotates it.

When multiple bodies are selected, they are all subject to the same rotation and scaling factor, but instead of using the body's center point as the reference point, the geometric average of all individual center points is used.

[8] Although joints cannot be rotated or scaled, they follow their respective bodies, maintaining their position on the body even when the body is transformed.

As presented in Figure 5, to visually aid the user when a scaling or rotation operation is in course, the system shows an axis-aligned bounding box encompassing all the selected bodies, that rotates and scales along with them. Also, a small circle appears at the center reference point, along with a line from that point to the stylus position.

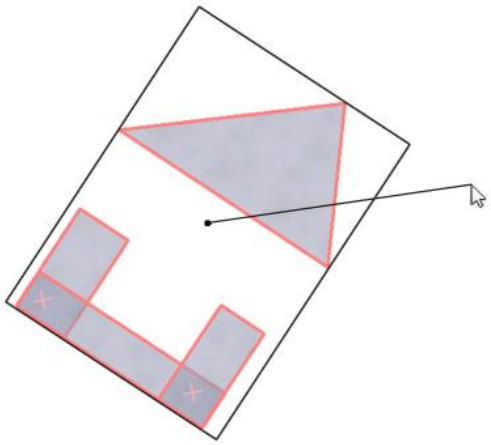


Figure 5 - Multiple objects being scaled and rotated

### 4.2.5 Removing

Since removing objects is an important operation that contributes to user's creative freedom, it was designed to be simple, intuitive, and to have a low impact on the user's cognitive load. In fact, removing an object is a just special case of moving it.

When an object starts being moved by the user, a large rectangle with a trash bin icon slides down from the top of the screen, sliding back up and off-screen when the object cease to be moved. If the stylus enters the trash bin area while moving any object, the trash bin icon turns red. If the user lifts the stylus while on this rectangle all the selected objects are removed. Figure 6 shows the trash bin area in context of a simple, almost empty, application, and also the trash bin icon representations before and after the stylus drags an object onto it. We choose to keep this area hidden unless the user starts moving objects to improve the use of screen real estate, since objects can only be deleted when they are being moved by the user.

Joints can also be removed by simply being moved outside any of the two bodies they connect, without the need to move them to the trash bin rectangular area, although the trash bin works for joints too.

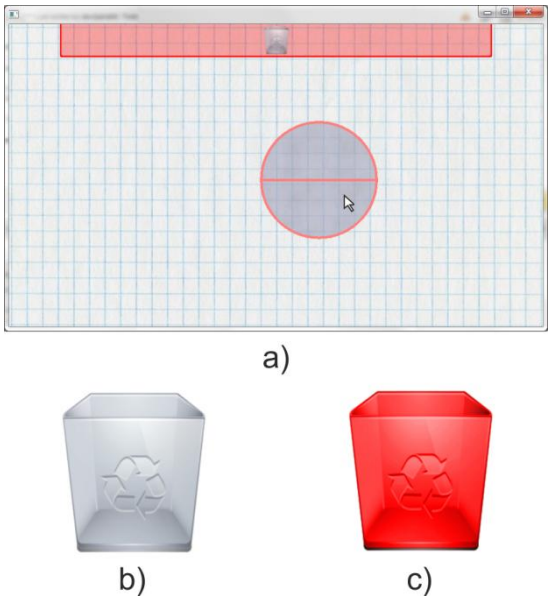



Figure 6 – a) simple application showing the trash bin area in its normal state, when an object is being dragged outside the rectangular trash area; b) trash bin icon in its normal state; c) trash bin icon when an object is dragged inside the trash area.

# 5 PRELIMINARY EVALUATION

On a preliminary and informal evaluation, we must point out how easily and fast new applications can be created with an intuitive sketch-based interface and physics simulation capabilities.

We noticed that SketchyDynamics is easier to use in devices with touchscreen and stylus, in comparison to traditional mice, which comes with no surprise since these devices are more suited for drawing and sketching.

Regarding the interaction with the system, creating objects is an intuitive action because it resembles the use of a pen over a paper. Also, the sliding trash bin area makes it easy to figure how to remove an object.

The automatic selection of connected objects when the user selects an object made the interaction faster and more convenient. However, it would also be useful to be able to select only one object from a set of connected objects, and scale or rotate it around the joint anchor point, instead of the object's center.

Another aspect worth mentioning is that while having the rotation and scale operations done simultaneously reduces the user's cognitive load and can improve the interaction speed in some situations, doing only one of these operations without triggering the other is complicated.

As this preliminary evaluation was done by the authors and a few other users, a formal and complete evaluation of the work presented in this paper is being prepared and will be conducted in the near future.

# 6 CONCLUSIONS

We have presented a library capable of speeding up the development of applications by providing them with a sketch-based interface combined with physics simulation. The library also provides facilities in managing the graphical side of the application and dealing with user input.

In an effort to make the library suitable for the widest range of applications we are working on adding more functionality into it, such as a new rope-like joint that can be created by drawing a line connecting two bodies. Since this introduces a new gesture, a new sketch recognition evaluation could be conducted, in order to refine the recognition of the new gesture.

On a preliminary evaluation we noticed that it would be useful to be able to select an individual body from a set of connected bodies and transform it using the joint anchor point as a reference. This poses some design problems since an object can have multiple joints (which one would be used?). The problem further increases if there is more than one selected object: should each object rotate around the average geometric center or around its joints? Before implementation, further study on how to overcome these problems is needed.

Another interesting feature would be the possibility to create object hierarchies, in which transformations applied to parent objects are passed onto the child objects, but not the opposite. The hierarchy could be constructed based on the depth of the objects.

Regarding the difficulty of doing a scale or rotation operation individually, a possible solution would be the use of a modifier key or secondary mouse button to restrict the action to a single operation. When the user presses this modifier key, the system could resolve what operation should be done based on the mouse movement being mainly radial or tangential, applying a scale or rotation, respectively. This modifier key could also be applied to restrict the movement of objects to horizontal, vertical and 45 degree translations.

The online publishing of SketchyDynamics is also being prepared.

# REFERENCES


Alvarado, C., & Davis, R. (2001). Resolving Ambiguities to Create a Natural Computer-Based Sketching. *Proceedings of IJCAI-2001*, pp. 1365-1371.

Anthony, L., & Wobbrock, J. O. (2010). A lightweight multistroke recognizer for user interface prototypes. *Proceedings of Graphics Inter-face 2010 (GI '10)*, pp. 245-252.

Catto, E. (n.d.). *Box2D*. Retrieved Jully 25, 2012, from http://box2d.org

Costa, A., & Pereira, J. P. (2012). SketchTester: Analysis and Evaluation of Calligraphic Gesture Recognizers (to be published). *20º Encontro Português de Computação Gráfica (20ºEPCG)*.

Fonseca, M. J., Pimentel, C., & Jorge, J. A. (2002). CALI: An online scribble recognizer for calligraphic interfaces. *AAAI Spring Symposium on Sketch Understanding*, pp. 51-58.

Kamel, H., Shonoda, M. W., Refeet, M., & Nabil, R. (n.d.). *Free-Hand Sketch Recognition For Visualizing Interactive Physics*. Retrieved Jully 26, 2012, from http://code.google.com/p/sketch-recognition-simulation-tool

Lee, W., Kara, L. B., & Stahovich, T. F. (2007). An efficient graph-based recognizer for hand-drawn symbols. *Computers & Graphics 31*, pp. 554-567.

Observer pattern. (n.d). *Wikipedia*. Retrieved July 28, 2012, from http://en.wikipedia.org/wiki/Observer_pattern

Pereira, J. P., Branco, V. A., Jorge, J. A., Silva, N. F., Cardoso, T. D., & Ferreira, F. N. (2004). Cascading recognizers for ambiguous calligraphic interaction. *Eurographics Workshop on Sketch-Based Interfaces and Modeling*.

Plimmer, B., & Freeman, I. (2007). A toolkit approach to sketched diagram recognition. *Proceedings of the 21st British HCI Group Annual Conference on People and Computers: HCI...but not as we know it (BCS-HCI '07), vol. 1*, pp. 205-213.

Purho, P. (2009). *Crayon Physics Deluxe*. Retrieved Jully 27, 2012, from http://crayonphysics.com

Rubine, D. (1991, July). Specifying Gestures by Example. *SIGGRAPH Computer Graphics, Volume 25 Issue 4*, pp. 329 -337.

Sezgin, T. M., & Davis, R. (2005). HMM-based efficient sketch recognition. *Proceedings of the 10th international conference on Intelligent user interfaces (IUI '05)*, pp. 281-283.

Stahovich, T. F. (2011). Pen-based Interfaces for Engineering and Education. *Sketch-based Interfaces and Modeling*, pp. 119-152.

Vatavu, R.-D., Grisoni, L., & Pentiuc, S.-G. (2009). Gesture Recognition Based on Elastic Deformation Energies. *Gesture-Based Human-Computer Interaction and Simulation, vol. 5085*, pp 1-12.

Wobbrock, J. O., Wilson, A. D., & Li, Y. (2007). Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. *Proceedings of the 20th annual ACM symposium on User interface software and technology (UIST '07)*, pp. 159-168.