UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
PRÓ-REITORIA DE GRADUAÇÃO
CENTRO DE CIÊNCIAS EXATAS E NATURAIS
INTERDISCIPLINAR EM CIÊNCIA E TECNOLOGIA

RAFAELA DE OLIVEIRA DIAS

DESENVOLVIMENTO DE CUSTOMIZAÇÃO EM AUTOLISP APLICADA A MODELAGEM DE DESENHOS ARQUITETÔNICOS NO AUTOCAD

MOSSORÓ
2021

RAFAELA DE OLIVEIRA DIAS

DESENVOLVIMENTO DE CUSTOMIZAÇÃO EM AUTOLISP APLICADA A MODELAGEM DE DESENHOS ARQUITETÔNICOS NO AUTOCAD

Monografia apresentada a Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Bacharel Interdisciplinar em Ciência e Tecnologia.

Orientador: Manoel Januário da Silva Junior, Prof. Dr.

Co-orientador: Bruno Rodrigo Simão, Prof. Dr.

MOSSORÓ

2021

D541d Dias, Rafaela de Oliveira.
Desenvolvimento de customização em AutoLISP aplicada a modelagem de desenhos arquitetônicos no AutoCAD. / Rafaela de Oliveira Dias. - 2021. 58 f. : il.

Orientador: Manoel Januário da Silva Junior.
Coorientador: Bruno Rodrigo Simão.
Monografia (graduação) - Universidade Federal Rural do Semi-árido, Curso de Ciência e Tecnologia, 2021.

1. Computer Aided Design. 2. Automação. 3. Processos repetitivos. 4. Esquadrias. 5. Etiquetas de cômodos. I. Silva Junior, Manoel Januário da , orient. II. Simão, Bruno Rodrigo, co-orient. III. Título.

O serviço de Geração Automática de Ficha Catalográfica para Trabalhos de Conclusão de Curso (TCC´s) foi desenvolvido pelo Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (USP) e gentilmente cedido para o Sistema de Bibliotecas da Universidade Federal Rural do Semi-Árido (SISBI-UFERSA), sendo customizado pela Superintendência de Tecnologia da Informação e Comunicação (SUTIC) sob orientação dos bibliotecários da instituição para ser adaptado às necessidades dos alunos dos Cursos de Graduação e Programas de Pós-Graduação da Universidade.

RAFAELA DE OLIVEIRA DIAS

DESENVOLVIMENTO DE CUSTOMIZAÇÃO EM AUTOLISP APLICADA A MODELAGEM DE DESENHOS ARQUITETÔNICOS NO AUTOCAD

Monografia apresentada a Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Bacharel Interdisciplinar em Ciência e Tecnologia.

Defendida em: 01 / 06 / 2021.

BANCA EXAMINADORA

MANOEL JANUARIO DA SILVA JUNIOR:03444846452

Assinado de forma digital por MANOEL JANUARIO DA SILVA JUNIOR:03444846452
Dados: 2021.06.11 18:26:18 -03'00'

Manoel Januário da Silva Junior, Prof. Dr. (UFERSA)
Presidente

Danielle Simone da Silva Casillo

Assinado de forma digital por Danielle Simone da Silva Casillo
Dados: 2021.06.11 16:38:34 -03'00'

Danielle Simone da Silva Casillo, Profa. Dr. (UFERSA)
Membro Examinador

Luciana Klein da S. de Morais

Luciana Klein da Silva de Morais, Profa. (Cursos Construir/Autodesk)
Membro Examinador

AGRADECIMENTOS

Agradeço ao meu orientador, Manoel Januário, e co-orientador, Bruno Simão, pelo apoio e os conhecimentos compartilhados nessa pesquisa, e ao longo do projeto de monitoria da disciplina de Projeto Auxiliado por Computador.

Também agradeço à minha família, em especial à minha mãe, Carla Rosana, e à minha tia, Cleide Regina, por serem as minhas maiores incentivadoras e apoiadoras. Ao meu pai, João Dias, que é o meu exemplo de cientista, e quem despertou em mim o interesse pela engenharia.

Por último agradeço aos meus melhores amigos, Joana Costa, Kadu Brito e Hugo Portela, por terem sido meus companheiros mais próximos, e a minha segunda família, ao longo desses últimos dez anos.

RESUMO

O desenvolvimento de projetos arquitetônicos com o uso do software AutoCAD exige a execução de vários comandos e procedimentos que, para a completa elaboração do desenho, se tornam exaustivamente repetitivos. Com a expectativa de criar rotinas de programação que possibilitem uma personalização do AutoCAD e ajudem a reduzir os processos repetitivos, desenvolveu-se essa pesquisa. Neste trabalho, são apresentados alguns conceitos básicos de programação através da linguagem AutoLISP, e a sua aplicação para a criação de comandos customizados para a execução do desenho de uma planta baixa no software AutoCAD. As funções desenvolvidas visaram agilizar processos repetitivos, tais como a inserção de blocos de esquadrias, e a execução do cálculo de áreas com inserção de textos de representação das etiquetas de cômodo. O uso da linguagem de programação para customização de softwares tem como aplicação a adaptação de programas para usabilidades especificas. Para a aplicação das rotinas criadas nessa pesquisa, foi utilizada uma planta baixa simples, em que a inserção de blocos de esquadrias e de textos de etiquetas de cômodo foram feitas usando os comandos customizados. Após o desenvolvimento do algoritmo e verificação da funcionalidade, julga-se que o resultado da aplicação do algoritmo se mostrou satisfatório, contribuindo para a simplificação de dois processos repetitivos do desenvolvimento da planta baixa.

Palavras-chaves: Computer Aided Design; Planta Baixa; Processos repetitivos; Automação; Esquadrias; Etiquetas de cômodos.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ABNT	Associação Brasileira de Normas Técnicas
CAD	Computer Aided Design
ICG	Interactive Computer Graphics
LISP	List Processing
UCS	Sistema de Coordenadas do Usuário
u.d.	Unidade de desenho
UFERSA	Universidade Federal Rural do Semi-Árido
VBA	Visual Basic for Application

SUMÁRIO

1 INTRODUÇÃO

O **LISP** é uma linguagem de programação que foi criada em 1958 por John McCarthy e Marvin Minsky. Na época os pesquisadores de Stanford tinham como objetivo desenvolver uma linguagem algébrica para processamento de listas, e que pudesse representar informações através da linguagem escrita. O processamento de listas é uma das principais características dessa linguagem, que tem o seu nome derivado de "*List Processing"*. Assim, o LISP foi uma das linguagens pioneiras a seguir o paradigma funcional, ou seja, a ter maior grau de independência do paradigma de Von Neumann, que era característico por determinar o conceito de estado nas variáveis dos programas. A linguagem de McCarthy e Minsky teve como principais vantagens, na época, a clareza e o maior poder expressivo, o que tornou o LISP uma linguagem fundamental para a Inteligência Artificial, permitindo a construção de programas onde o computador simula aspectos do comportamento da inteligência humana, como jogar xadrez (Fonseca Filho, 2007).

Ao longo dos anos, com a evolução do computador pessoal e o desenvolvimento de aplicações, surgiram novas derivações do LISP e outras linguagens de programação. Uma delas foi o **CommonLISP**, um dialeto que teve como objetivo compilar aspectos de todos os outros dialetos LISP já existentes até os anos 80 (Bergin; Gibson, 1996). Segundo Jacoski e Breda (2004), o **AutoLISP** é uma versão reduzida do CommomLISP, derivação essa que foi desenvolvida pela própria Autodesk com o intuito de possibilitar a programação de customizações dentro do **AutoCAD**, através da utilização dos próprios comandos do software.

Para Azevedo (2000), as aplicações computacionais se toraram primordiais para a execução de projetos de engenharia. Possuir habilidade com programas de parametrização, como o **FTool** para cálculo estruturais, **MATLAB** para desenvolvimento de gráficos, ou até o Excel para planilhas, é um requisito básico para que engenheiros atuem adequadamente em suas áreas. Todos esses softwares citados são customizáveis e funcionam através da formulação das necessidades de cada usuário. Apesar de pouco explorados, o AutoCAD também oferece recursos de customização do software, e isso o torna um otimizador de processos na elaboração de projetos (Jacoski; Breda, 2004).

De acordo com Jacoski e Breda (2004), o conhecimento em AutoLISP possibilita tornar o programa um software sob medida para as necessidades dos usuários, tornando o desenho auxiliado por computador, além de um visualizador geométrico, uma ferramenta de parametrização de projetos de engenharia, possibilitando agilizar processos específicos, como obtenção de dados para levantamento topográfico, funções que verificam a compatibilidade

entre projetos de diferentes áreas, entre outros. Para Oliveira, Costa e Baldam (2016), a adoção do LISP na criação de rotinas dentro do CAD apresenta vantagens por possuir uma escrita mais expressiva, o que torna a linguagem mais fácil de ser entendida e aplicada. Assim o AutoLISP se apresenta como um dialeto completo e poderoso para a criação de algoritmos complexos, ao mesmo tempo que é uma linguagem mais didática e acessível para os mais leigos em lógica de programação.

Projetar através do desenho assistido por computador costuma ser um processo repetitivo, sendo necessário desenhar linha por linha, e inserir elementos e dados de forma mais analógica, ou seja, é necessário que o usuário forneça informações por conta própria. Através da experiência com o desenvolvimento de plantas baixas ao longo do curso de Ciência e Tecnologia, foi possível observar alguns processos repetitivos durante o desenho de projetos arquitetônicos. Por exemplo, para inserir identificações de cômodo/área numa planta baixa é necessário, primeiro, inserir os blocos de etiqueta contendo os textos com nome e área; com outro comando medir a área que será identificada; e por último, digitar manualmente o valor medido no bloco de etiqueta que foi inserido. O AutoCAD não possui um comando que faça esse, e outros processos repetitivos do desenho arquitetônico, de forma direta e automatizada, tornando o procedimento de desenho demorado e enfadonho para os projetistas.

Com o AutoLISP é possível desenvolver códigos de automação desde comandos simples até programas complexos. Assim, a partir dos conceitos introdutórios de programação e da linguagem LISP podem ser desenvolvidos no AutoCAD algoritmos de automação que possibilitem evitar repetição de processos durante o desenho de uma planta baixa. Essas customizações serão constituídas pela criação de novos comandos, que quando aplicados servirão para agilizar algumas etapas do processo de desenho do projeto arquitetônico.

Dado o exposto, o objetivo dessa pesquisa foi criar algoritmos de automação, que evitem a repetição de processos durante a inserção de blocos de esquadrias, e a criação de etiquetas de cômodos. Essas customizações foram constituídas da criação de novos comandos utilizando AutoLISP, que servirão para agilizar os processos repetitivos escolhidos para serem customizados de forma que facilite a usabilidade do programa AutoCAD para os desenhistas de projetos arquitetônicos.

2 REFERENCIAL TEÓRICO

Alguns conceitos básicos são necessários para entender o desenvolvimento dessa pesquisa. Com isso, ao longo desse capítulo serão apresentados os principais conceitos envolvidos na criação de um desenho arquitetônico e no desenvolvimento de um algoritmo em AutoLISP. Entre eles uma introdução ao ambiente do software AutoCAD, envolvendo a descrição da sua interface e seus comandos básicos; a definição de desenho arquitetônico, planta baixa e apresentação da norma NBR 6492 - Representação de projetos de arquitetura; e os principais conceitos introdutórios ao AutoLISP, abordando o seu dialeto e a caracterização do seu console, o Visual LISP.

2.1 O Ambiente AutoCAD

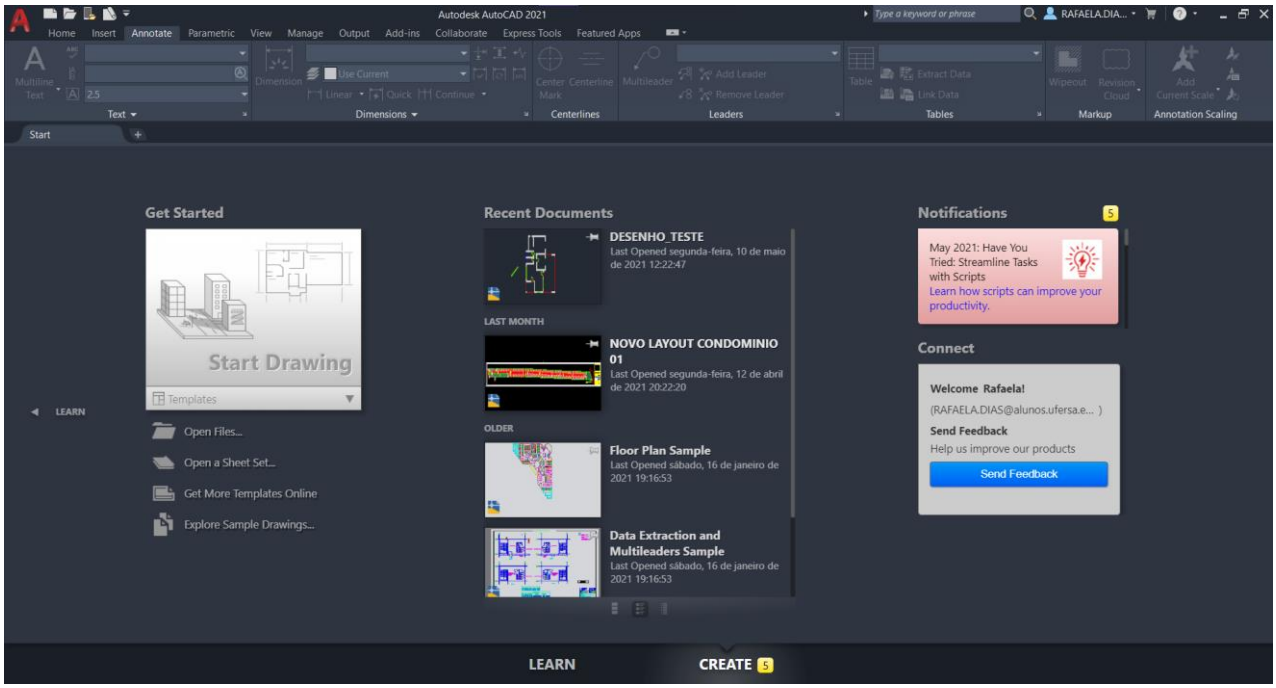
O AutoCAD é definido como um software do tipo *Computer Aided Design* (CAD), desenho auxiliado por computador em tradução livre, e foi criado em 1982 pela Autodesk como o primeiro programa de desenho baseado em vetores. Os gráficos vetoriais utilizam elementos geométricos, como pontos, segmentos de reta, curvas e polígonos, para representar imagens, que são definidas matematicamente e armazenadas em bases de dados (Leake; Borgerson, 2015).

Jacoski e Breda (2004) o descrevem como uma espécie de prancheta virtual, que possibilita a criação de linhas em diferentes espessuras, elementos textuais, entre outros objetos geométricos e entidades que compõem um projeto. Os sistemas de CAD modernos são baseados na computação gráfica interativa (ICG), que consiste em transformar e apresentar dados computacionais em imagens e símbolos, assim o usuário se comunica com o programa através da entrada de dados e comandos fornecidos ao computador, que serão utilizados para gerar os desenhos (Sacar; Rao; Narayan, 2008).

2.1.1 A interface

Desde o primeiro ano de criação do AutoCAD, a Autodesk e outras empresas do ramo investiram no desenvolvimento desses *softwares* apostando em melhorias na usabilidade e interface dos seus programas, de forma que a utilização dessas ferramentas se tornasse cada vez mais intuitiva, o que popularizou o sistema CAD (Amaral; Pina Filho, 2010). A interface, atualmente, é constituída majoritariamente por menus e botões de atalho para os comandos, e ao abrir o programa, a tela inicial apresenta uma área, onde é possível visualizar os projetos anteriormente abertos, um botão de iniciar um novo desenho (*Start Drawing*) e alguns botões de gerenciamento de arquivos, como mostra a Figura 1.

Figura 1 – Tela inicial do AutoCAD 2021.



Fonte: Autoria Própria.

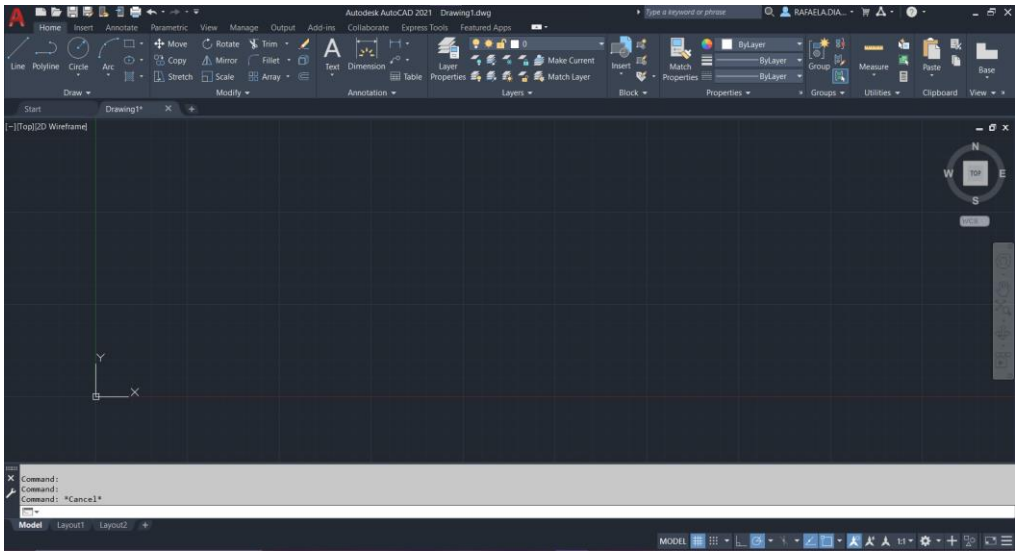
Após clicar no botão *Start Drawing* o usuário é direcionado para a área de desenho (Figura 2), que é a área principal de trabalho do CAD, onde é possível executar os comandos, modificar elementos e compor um projeto. Na parte inferior da área é possível observar que existem duas abas, *Model* e *Layout,* a primeira guia é direcionada para a criação do desenho e seus elementos, enquanto a segunda é voltada para a parte de configuração de escala e impressão (Oliveira; Costa; Baldam, 2016).

Logo acima das abas *Layout* e *Model* fica localizada a janela de comandos, essa é a área dedicada para a entrada de funções do AutoCAD e de interação entre o software e o usuário durante a execução de um comando. De acordo com Oliveira (2014), essa parte do software passa instruções através de *prompts*, pedindo informações como valores de medidas dos objetos, coordenada de pontos, entre outros, que são necessários para desenhar os elementos, ela mostra para o usuário o aviso atual e o histórico de comandos previamente utilizados. A parte superior da interface é ocupada pela *Ribbon,* um menu que é caracterizado como uma faixa de opções de comandos do AutoCAD, onde cada guia agrupa uma seleção de comandos que possuem uma finalidade comum, por exemplo a parte *Annotation* reúne os comandos de adição de elementos de anotação, como textos, cotas, linhas de chamada, etc.

Ainda na Figura 2, na parte inferior esquerda, se localiza um ícone com indicações de X e Y. Esse é o símbolo que representa o tipo de sistema de coordenadas em uso no desenho, podendo ser o tipo sistema global de coordenada (WCS), nos desenhos 2D; e o sistema de coordenadas do usuário (UCS), para desenhos 3D. O plano cartesiano que determina a posição de pontos na tela é utilizado como indicativo das direções e dos sentidos positivos dos

eixos X e Y nos desenhos bidimensionais; e X, Y e Z para tridimensionais, além de indicar o ponto de coordenada (0,0) do programa (Netto, 2019).

Figura 2 – Área de desenho do AutoCAD 2021.



Fonte: Autoria Própria.

Por último, o menu que fica na parte inferior à direita da tela (Figura 2) que é a barra de *status* do aplicativo, fornece acesso rápido à algumas configurações de auxílio ao desenho, como o *Snap*, que são configurações de precisão para obtenção de pontos; *Grid*, que serve para ativar e desativar a grade quadriculada ao fundo do plano de desenho; *Polar*, auxilia na obtenção de ângulos durante os comandos; e o *Ortho*, que auxilia o desenho de linhas retas ortogonais. A barra de status também fornece algumas configurações de escala e de customização da interface, sendo possível alterar entre modos de exibição de menus diferentes para desenhos 3D ou 2D, e distribuição da interface de acordo com versões mais antigas do AutoCAD (Oliveira; Costa; Baldam, 2016).

2.1.2 Comandos e conceitos básicos

Os desenhos no AutoCAD são executados através da entrada de dados fornecidos pelos usuários, a comunicação feita entre software e desenhista para obtenção dessas informações é feita através dos comandos. O CAD possui uma variedade de comandos e cada um deles é responsável por executar uma função específica no desenho. Por exemplo, o comando *move* é utilizado para mover objetos; o *Line* é específico para desenhar linhas; *Circle* para círculos, e assim por diante. Para ativar um comando o desenhista deve conhecer o nome, ou abreviatura da função, digitá-lo na janela de comandos e teclar *Enter*, ou buscar o ícone correspondente ao comando que fica localizado na *Ribbon*, no painel específico do grupo de comandos. A partir disso o software interage com o usuário através de perguntas ou instruções na janela de comandos, onde são solicitados os dados necessários para o

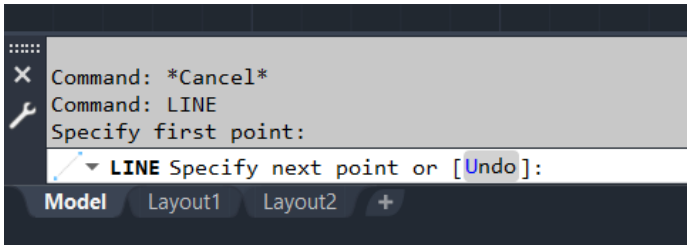
desempenho correto da função, como seleção de objetos, definição de pontos e coordenadas, entre outros, tal como se observa na Figura 3. Para fazer um projeto no AutoCAD é preciso conhecer alguns dos seus comandos básicos.

O comando *Line* é usado para desenhar linhas. Oliveira, Costa e Baldam (2016) definem a linha no CAD como um segmento reto, ou um conjunto deles, desenhado entre dois pontos (Figura 4). O comando exige que o desenhista defina pelo menos dois pontos distintos com cliques na tela, ou informando as suas coordenadas na janela de comandos, sendo possível continuar desenhando novos segmentos a partir do ponto anterior. Outra forma de gerar linhas no CAD é através do comando *Offset*, esse comando cria cópias paralelas de objetos já existentes. Para usá-lo é preciso determinar a entidade base, e especificar uma distância à qual os objetos copiados ficarão. Ao entrar no comando, ele pede que se determine o valor de uma distância fixa entre as linhas, ou a opção *Through*, que permite a inserção de pontos, podendo variar a distância entre uma linha e outra. Nessa opção, após selecionar uma entidade, a linha paralela é gerada em um ponto escolhido pelo usuário, depois de clicar no objeto inicial exibe um *preview* de onde ficará a nova linha e uma caixa em que o usuário pode digitar o valor da distância desejada (Netto, 2019).

O comando *Arc* executa o desenho de arcos, a interação do programa exige três pontos destintos para defini-lo. Assim, como o *Line,* o usuário pode fornecer os pontos através de cliques na tela, ou inserção das coordenadas, sendo possível também defini-lo através do tamanho do raio, ou do ângulo. Por padrão, a ordem de entrada dos pontos é: o ponto inicial, centro do arco e ponto final; porém é possível escolher a ordem e quais tipos de entradas serão fornecidos para este comando, como mostra a Figura 5 (Oliveira, 2014).

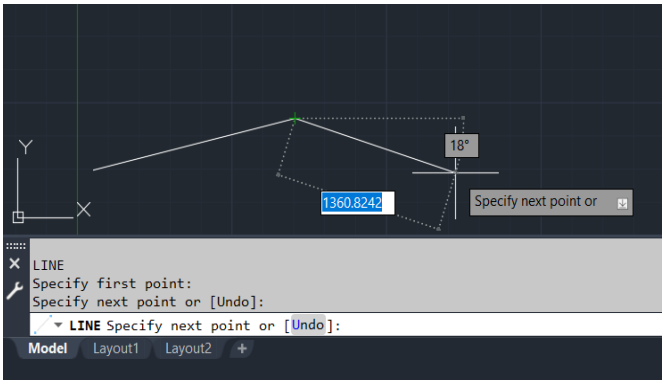
Além de comandos que geram novas entidades, o AutoCAD possui funções que fazem manipulação e modificações em objetos já desenhados, os comandos *Trim* e *Extend* são exemplos desses modificadores. O *Trim* corta uma parte de um objeto até os limites definidos por outro objeto, enquanto o *Extend* funciona de forma contrária estendendo as entidades até que elas se encontrem com outra que defina um limite (Oliveira, 2014). Se dentro do comando *Trim* o usuário desejar estender o objeto ao invés de cortar, é possível ativar o *Extend* segurando a tecla *shift* durante a seleção do objeto, e o mesmo funciona para fazer cortes dentro do *Extend*.

Figura 3 – Execução de um comando na janela de comandos do AutoCAD.



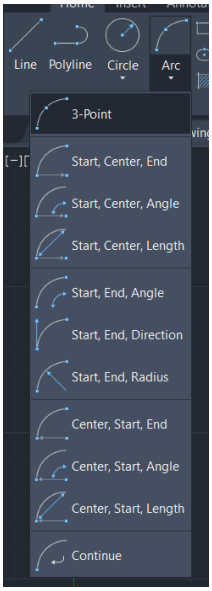
Fonte: Autoria própria.

Figura 4 – Segmento de linha feito com o comando *Line*.



Fonte: Autoria Própria.

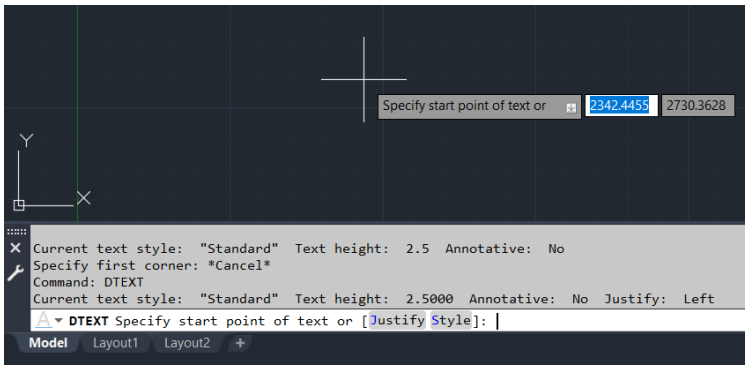
Figura 5 Tipos de arcos de acordo com os dados de entrada.



Fonte: Autoria Própria

Para inserir elementos textuais no desenho, o AutoCAD dispõe de duas funções principais, os comandos *DText* e *MText.* O *DText* cria um texto simples, onde cada linha compõe uma entidade diferente. Esse tipo de texto é comumente utilizado para indicações simples, como etiquetas de cômodo, título de projeto e escala. Já o *MText* cria textos de múltiplas linhas, como um parágrafo. Independentemente da quantidade de linhas escritas, o texto se comportará no desenho como uma entidade única (Oliveira; Costa; Baldam, 2016). Em ambos os comandos será pedido ao desenhista informações de configurações do texto, como a fonte, altura, justificação e rotação; essas entradas podem ser dadas diretamente na janela de comandos durante a execução da função (Figura 6).

Figura 6 – Exemplo do comando *DText* na barra de comando.

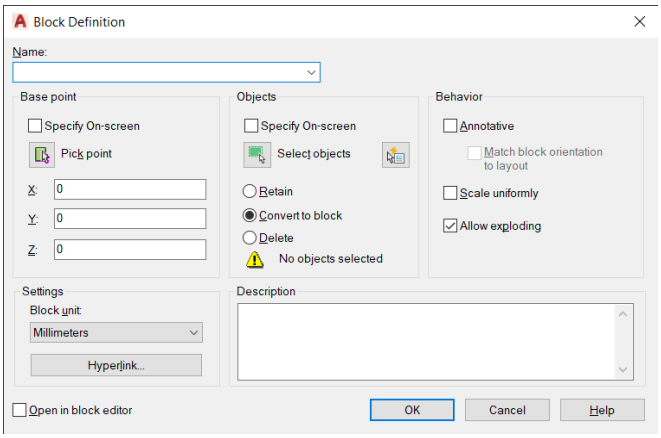


Fonte: Autoria Própria.

Outro conjunto de comandos importante para o desenvolvimento de desenhos técnicos no CAD são os relacionados à criação e inserção de blocos. Os blocos permitem agrupar entidades individuais em um único objeto, sendo geralmente usado quando há a necessidade de inserir elementos repetidas vezes, como portas e janelas. Associar um conjunto de elementos à um bloco garante o padrão do desenho, seja por medidas, cores, ou forma dos elementos (Sousa, 2019). Para criar blocos, usa-se os comandos *Block* ou *WBlock*. O Block cria blocos através de uma caixa interativa e salva esse objeto em uma biblioteca interna do arquivo de desenho, enquanto o WBlock executa a mesma função, porém armazenando o novo bloco na biblioteca externa do computador (Oliveira; Costa; Baldam, 2016). Em ambos os comandos o processo é semelhante, o desenhista precisa selecionar os objetos que serão agrupados, determinar um nome e um ponto de inserção para o objeto final. Na caixa de diálogo do comando também é possível determinar fatores de multiplicação da escala nos eixos X, Y e Z, como mostra a Figura 7.

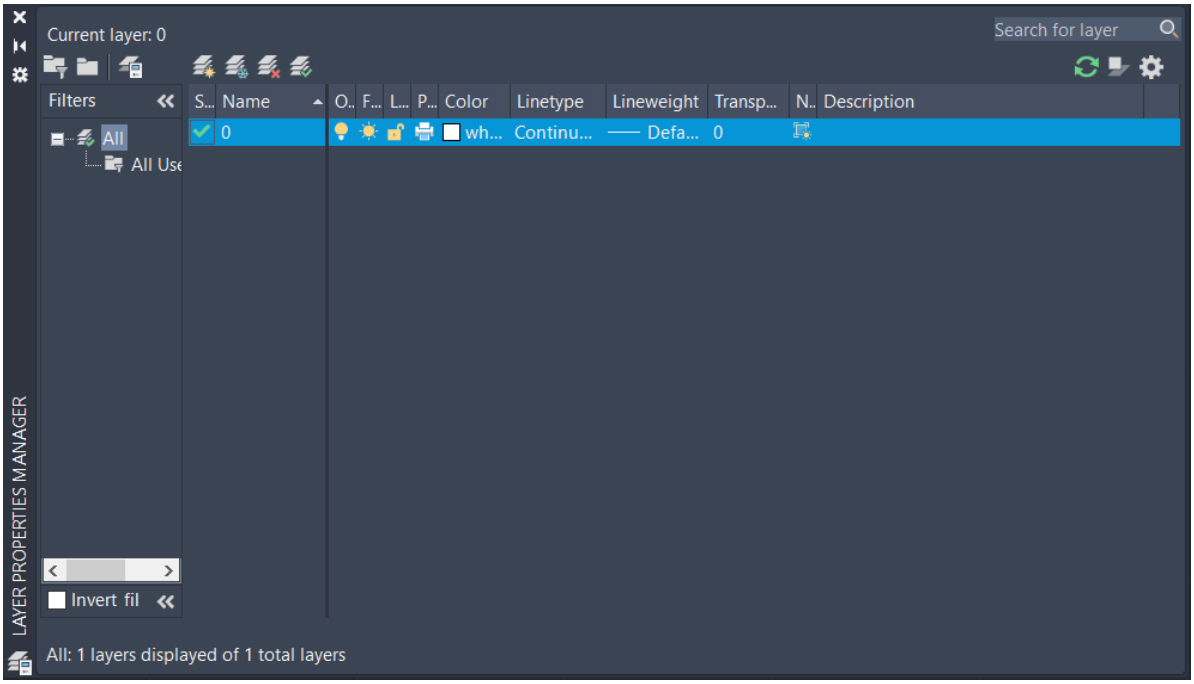
Quando se inicia um novo projeto no AutoCAD é necessário definir e configurar as *Layers* do desenho. Layers são como camadas transparentes e sobrepostas que agrupam um conjunto de informações sobre as entidades de desenho, como cor, tipo de linha e espessura (Oliveira; Costa; Baldam, 2016). É importante que todos os elementos do desenho tenham layers definidas, pois esta funciona como uma espécie de identidade dos objetos, facilitando a distinção de cada um deles no desenho. É possível criar e modificar as camadas através da função *Layer*, na caixa de diálogo do comando ficam listadas todas as camadas existentes no desenho, sendo possível gerenciar, criar, ou apagar as camadas já existentes. O desenhista pode definir configurações como nome, cor, tipo e espessura da linha, essas definições são importantes para a impressão do desenho, pois cada elemento será representado no papel de acordo com as suas configurações de *layer*. A Figura 8 mostra a caixa de diálogo do comando, nesse exemplo existe apenas uma camada com o nome “0”, essa *layer* é padrão do AutoCAD, gerado automaticamente pelo programa e não pode ser apagado.

Figura 7 – Caixa de diálogo do comando *Block*.



Fonte: Autoria Própria.

Figura 8 – Caixa de diálogo do comando *Layer*.



Fonte: Autoria Própria.

2.2 Desenho arquitetônico

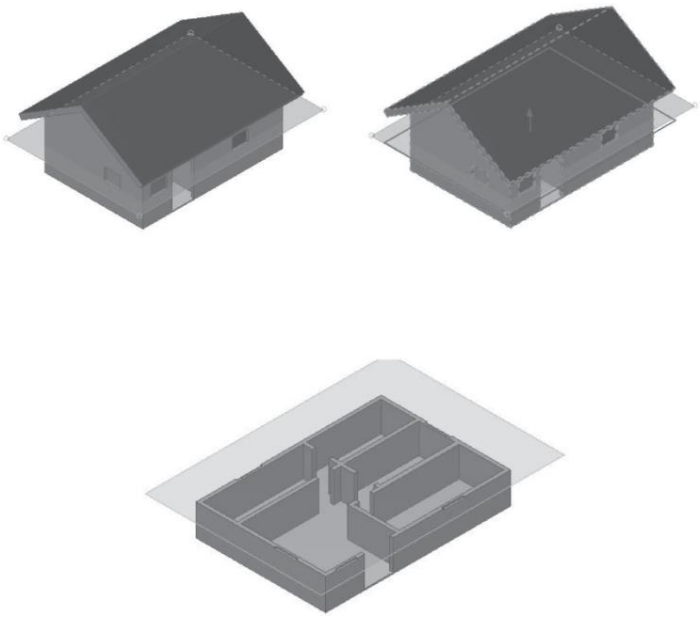
O desenho arquitetônico é a representação técnica de um projeto de edificação, e se refere a um conjunto de desenhos, como planta de fundações, plantas baixas, elevações, cortes, plantas de instalações elétricas, etc., que são reunidos e organizado de modo a transmitir a maior quantidade possível de informação sobre o projeto. O conjunto completo de desenhos representa uma descrição gráfica de um projeto arquitetônico (Kubba, 2014).

2.2.1 A definição de planta baixa

Uma planta baixa representa a vista superior de uma edificação através de um plano secante horizontal, situado à aproximadamente 1,50 m do piso (Associação Brasileira de Normas Técnicas, 1994). A parte acima deste plano é eliminada, sendo representada pela porção que sobra, ou seja, o que um observador imaginário visualizaria se estivesse posicionado no alto da edificação (Figura 9). As paredes externas em geral possuem entre 20 e 25 cm de espessura, e as paredes internas são representadas com 15 cm (Cruz, 2014).

Conforme Netto (2014), o objetivo de uma planta é representar a edificação e suas medidas construtivas de forma detalhada, abordando informações de larguras, profundidades de ambientes, espessuras de paredes, janelas, portas e os demais detalhes de medidas e anotações. Esse tipo de representação é uma forma de projeto executivo, assim as suas principais convenções devem ser baseadas nas normas de desenho técnico e arquitetônico, a fim de seguir as padronizações definidas pelas instituições nacionais e internacionais.

Figura 9 – Corte imaginário de uma planta baixa.



Fonte: CRUZ, Michele David da – Desenho Técnico (2014).

2.2.2 A Norma NBR 6492 - Representação de projetos de arquitetura

De acordo com Kubba (2014) a linguagem técnica do desenho arquitetônico é necessária para representar o projeto fielmente no papel, de forma que possa ser compreendido por todos os profissionais envolvidos na sua execução. A Associação Brasileira de Normas Técnicas (ABNT) é o órgão responsável por estabelecer o conjunto de métodos e preceitos destinado à recomendação e fixação das condições para execução de projetos (Blattman, 1994).

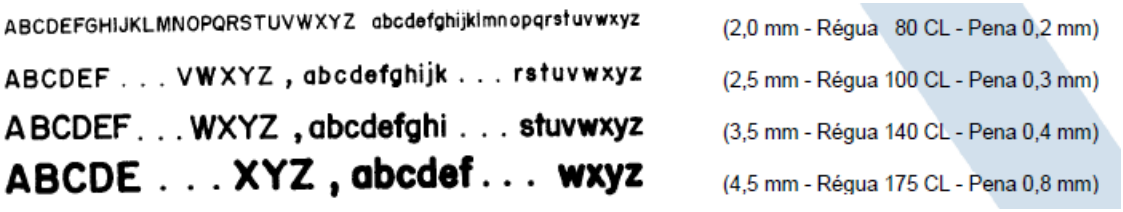
Dentre o conjunto de normas estabelecidos pela ABNT para desenho técnico, existe a "NBR 6492 - Representação de projetos de arquitetura", de 1994, que tem como objetivo fixar as condições exigíveis para representação gráfica de projetos de arquitetura, visando à sua boa compreensão (Associação Brasileira de Normas Técnicas, 1994). A norma apresenta especificações sobre tipos e espessura de linha, representação de quadros e esquadrias, marcação de detalhes, além de documentação dos projetos e conceitos dos elementos de projetos.

Algumas especificações da NBR 6492 são utilizadas como parâmetros para a representação de entidades nas customizações desenvolvidas nesta pesquisa, especificamente os itens *5.3.3.9 Detalhes de esquadrias*, *Anexo A-2 Tipos de letras e números* e *Anexo A-16 Designação das portas e esquadrias*, seguindo as suas respectivas determinações:

> 5.3.3.9 Detalhes de esquadrias: Os detalhes de esquadrias (portas e janelas), devem atender à nomenclatura de porta e janela, respectivamente, P e J (ABNT, 1994).

> Anexo A-2 Tipos de letras e números: Letras e números representados de acordo com a Figura 10, sem inclinação e entrelinhas não inferior a 2 mm (ABNT, 1994).
>
> Anexo A-16 Designação das portas e esquadrias: Utilizar para portas P1, P2, P3 e Pn; e para janelas J1, J2, J3 e Jn (ABNT, 1994).

Figura 10 – Representação de letras por instrumento, item A-2.2.1 da NBR 6492.



Fonte: ABNT – NBR 6294: Representação de projetos de arquitetura (1994).

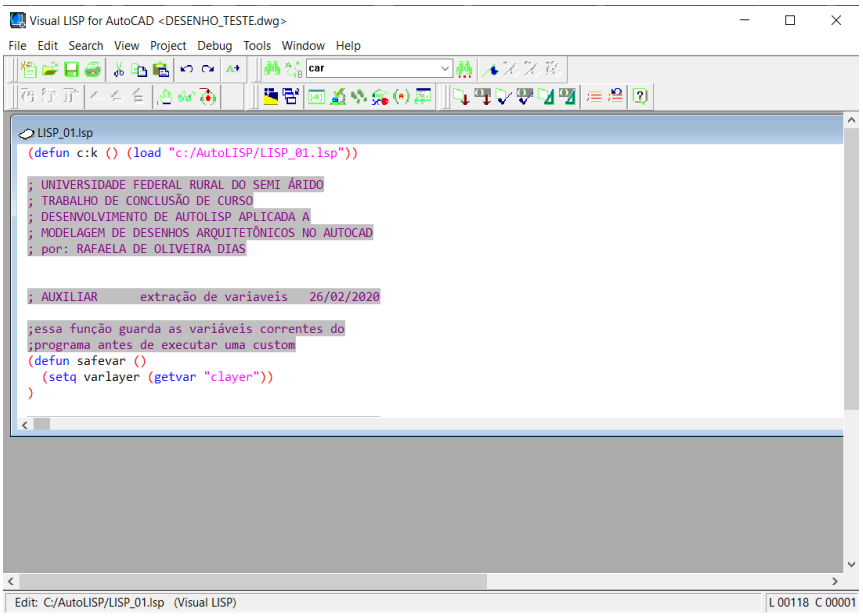
2.3 O Visual LISP e o dialeto AutoLISP

O AutoCAD utiliza o dialeto AutoLISP para possibilitar seus usuários a criar aplicativos dentro do software. Originalmente os algoritmos eram escritos em processadores de texto, como o bloco de notas do *Windows*, atualmente o programa dispõe de um console que funciona como interpretador, e fornece a funcionalidade de edição de algoritmos dentro do ambiente CAD, chamado Visual LISP. Essa ferramenta auxilia a execução e compilação das rotinas utilizando cores, e outros artifícios visuais que facilitam o desenvolvimento pelo programador (Jacoski; Breda, 2004).

A implementação do console pela Autodesk melhorou a edição dos arquivos LISP. Além da utilização das cores para diferenciar os elementos do algoritmo, o visual LISP também possui ferramentas de controle e verificação de bugs, oferecendo um ambiente integrado de desenvolvimento (Oliveira; Costa; Baldam, 2016). Assim, o que era para ser uma linguagem acessível a programadores profissionais, se tornou popular aos usuários comuns impulsionando o aprendizado e desenvolvimento de rotinas dentro do AutoCAD (Jacoski; Breda, 2004).

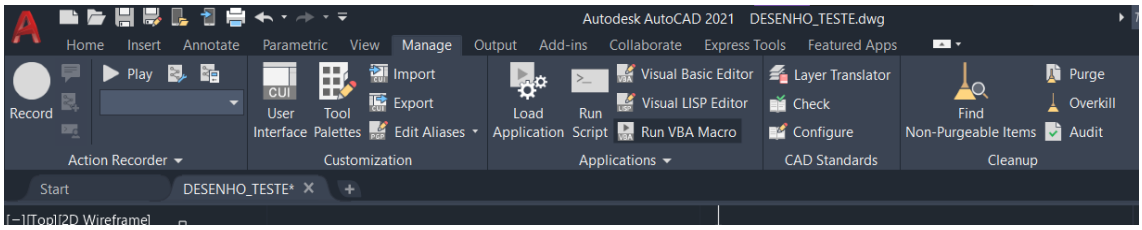
A área de trabalho do console é aberta através do comando *VLISP*, que quando ativado abre a interface mostrada na Figura 11. Os arquivos de customização do AutoLISP são salvos com a extensão *".lsp"* e para executa-los, precisam ser carregados dentro do desenho. Para carregar uma rotina no software o usuário deve acessar a aba *Manage* da Ribbon, e selecionar o botão *Load Application*, Figura 12. Também é possível carregar um arquivo direto na janela de comandos, digitando "*(load "nome_do_arquivo.lsp")*" e pressionando Enter. Outra forma de carregar e descarregar rotinas é associando o arquivo com a inicialização do programa, ou do desenho, salvando o algoritmo com as respectivas extensões: *acad.lsp* e *acaddoc.lsp* (Balda; Costa; Oliveira, 2016).

Figura 11 – Interface do console Visual LISP.



Fonte: Autoria Própria.

Figura 12 – Aba *Manage* da *Ribbon* no AutoCAD 2021.



Fonte: Autoria Própria.

2.3.1 O dialeto AutoLISP

O AutoLISP é um dialeto derivado da linguagem CommonLISP, por isso possui uma sintaxe própria desde os tipos de dados, que são em formato de listas, entre outras características de linguagem, como a formatação da escrita, métodos de alocação de memória, estruturas de decisão lógica, etc. O objetivo da sua implementação no AutoCAD é dar condições aos usuários de programar soluções não existentes no programa (Jacoski; Breda, 2004). Apesar de possuir um ambiente de programação próprio, é possível fazer pequenos algoritmos direto da janela de comandos. Se, por exemplo, escrevermos o código: “(command “line” “0,0” “@8<0” “@5<90” “@8<180” “c”)”, ao apertar a tecla Enter, um retângulo de dimensão 8 u.d. (unidades de desenho) por 5 u.d. será desenhado a partir do ponto (0,0). Nesse código o programa interpreta que deve executar o comando *Line*, com ponto inicial na coordenada cartesiana (0,0), em seguida determina o segundo ponto na coordenada polar (@8<0) e assim por diante.

O *Lisp*, diferente de outras linguagens de programação, trata os dados e os comandos da mesma forma, como listas. Uma lista é iniciada com parêntese aberto "(" e finalizada com parêntese fechado ")", e dentro deles contém os átomos, ou argumentos, que podem ser dos tipos: números inteiros e reais; uma cadeia de caracteres formando palavras ou frases (*string*); símbolos, que podem ser nomes de funções internas ou variáveis; um comando do CAD, ou função do AutoLISP, entre outros (Tavares; Fonseca, 2011). Assim todas as expressões de AutoLISP, tanto comandos quanto dados, seguem a seguinte sintaxe "(*nome da função* [*argumentos*])" sendo necessário que todas as listas e sub listas estejam entre parênteses para o funcionamento correto da função, e alguns argumentos, como *strings*, entre aspas (Oliveira; Costa; Baldam, 2016).

As operações matemáticas em Lisp são escritas com a notação infixa, ou seja, com operador representado antes dos operandos, por exemplo, a operação 1 + 1 é representada na forma (+ 1 1) (Tavares; Fonseca, 2011). As funções aritméticas podem operar dois ou mais argumentos, porém algumas funções matemáticas, como as trigonométricas, suportam um número limitado de fatores, como mostrado nas Figuras 13 e 14 (Oliveira; Costa; Baldam, 2016).

Figura 13 – Operações matemáticas na linguagem LISP.

Função	Descrição
(+ A B)	O resultado será a soma de A e B.
(- A B)	O resultado será a diferença entre A e B.
(* A B)	O resultado será o produto de A e B.
(/ A B)	O resultado será o quociente de A e B.
(max A B)	O resultado será o maior valor entre A e B.
(min A B)	O resultado será o menor valor entre A e B.

Fonte: OLIVEIRA, Adriano de; COSTA, Lourenco; BALDAM, Roquemar de Lima - **AutoCAD 2016:** utilizando totalmente (2016).

Figura 14 – Funções matemáticas na linguagem LISP.

Função	Descrição
(abs x)	O resultado será o módulo de X.
(sqrt y)	O resultado será a raiz quadrada de Y.
(expt A B)	O resultado será A elevado a B.
(exp A)	O resultado será e elevado a A.
(log A)	O resultado será o logaritmo natural de A.
(float X)	Transforma um número inteiro em real.
(fix X)	O resultado será a parte inteira de X.
(sin P)	O resultado será o seno de P, em que P é o ângulo em radianos.
(cos P)	O resultado será o cosseno de P, em que P é o ângulo em radianos.
(atan X)	O resultado será o arco tangente de X.
(1+Y)	O resultado será Y acrescido de 1.
(angle P1 P2)	O resultado será o ângulo, em radianos, entre os pontos P1 e P2.
(polar P1 A d)	O resultado será um ponto a um ângulo A e uma distância d de um ponto P1.
(type A)	O resultado será o tipo da variável A (real, inteiro, lista ou caractere).

Fonte: OLIVEIRA, Adriano de; COSTA, Lourenco; BALDAM, Roquemar de Lima - **AutoCAD 2016:** utilizando totalmente (2016).

As listas que desempenham funções dentro do algoritmo são chamadas de *listas de programas*, e se caracterizam por ter o nome de uma função no seu *car*, ou seja, no primeiro elemento da lista. Assim, se o *car* for o nome de uma subrotina ou função, ela será executada dentro do algoritmo, e os outros elementos da lista, chamados de *cdr*, serão os argumentos dessa função (Tavares; Fonseca, 2011). A Tabela 1 lista e exemplifica as principais funções usadas nessa pesquisa.

Tabela 1 – Tabela de funções do AutoLISP.

FUNÇÕES		
SINTAXE	**FUNÇÃO**	**EXEMPLO**
alert	Cria uma caixa de mensagem na interface	(alert "mensagem")
command	Executar um comando do AutoCAD	(command "comando")
defun	Definir uma rotina como função de um atalho	(defun c:atalho ()
load	Carregar um arquivo	(load "endereço")
prompt	Emitir mensagem na barra de comandos	(prompt "mensagem")
princ	Elimina o *nil*!	(princ)
setq	Definir uma variável	(setq variável valor)
get[type]	Pede para o usuário atribuir um valor para uma variável	(setq var (get[tipo_de_var] "mensagem"))
setvar	Pesquisar ou definir uma variável de ambiente	(setvar "variável de ambiente" "valor")
if	Função de condição	(if (= var "valor"))
initget	Limita a entrada de uma variável para a função *getkword*	(initget "ent1 ent2 ... entn") a função get nesse caso deve ser **getkword**
progn	Utilizado quando necessário usar múltiplas funções dentro da condição do *if*	(progn (sequência de funções))
and / or	Usados dentro do *if* para combinar duas condições	(if (and / or (cond1) (cond2)))
exit	Sair do programa	(exit)

repeat	Repete uma customização durante um determinado número de vezes	(repeat numero_de_repet (sequência de funções))
polar	Posicionar um ponto em relação à outro	(polar ponto_inicial (direção em rad) valor)
angle	Guarda o valor de ângulo entre dois pontos	(angle p1 p2)
ssget	Ativar a seleção de objetos dentro de uma rotina	(ssget)
rtos	Converte um número real ou inteiro em *string*	(rtos argumento)
strcat	Junta em um mesmo texto duas ou mais *strings*	(strcat "texto_1" "texto_2" ... "texto_n")
list	Cria uma lista dos argumentos determinados	(list argumento1 argumento2 ... argumenton)
strcat	Junta em um mesmo texto duas ou mais *strings*	(strcat "texto_1" "texto_2" ... "texto_n")
redraw	Redesenha um objeto modificando manipulando as suas linhas	(redraw <nome da entidade> <argumento de ação>)
entlast	Obtém o par ordenado <-1> (nome de entidade) do último objeto desenhado	(entlast)
entget	Obtém a lista completa de pares ordenados de uma entidade	(enteget <nome da entidade>)
assoc	Localiza um tipo de par ordenado de acordo com seu número	(assoc <número> <nome da entidade>)
cons	Cria um par ordenado	(cons <número> "argumento")
subst	Substitui um par ordenado por outro, dentro de uma determinada lista	(subst <novo par> <par antigo> <nome da lista>)
entmod	Atualiza os valores substituídos em uma lista	(entmod "nome da lista")

Fonte: Autoria Própria.

As variáveis utilizadas na função podem ser variáveis comuns determinadas pelo programador, ou as variáveis do tipo ambiente. As variáveis definidas pelo programador são usadas dentro das funções para guardar informações fornecidas pelo usuário, como nomes, distancias, pontos, entre outros. Para definir esse tipo de variável deve-se usar no algoritmo a função *setq* (Tabela 1), sendo importante que o usuário saiba qual tipo de variável deverá ser fornecida pelo desenhista (Tabela 2), pois essa informação é necessária para utilizar essa informação em outras partes do algoritmo. Conforme Oliveira, Costa e Baldam (2016), as variáveis de ambiente são internas do AutoCAD e guardam informações de configuração do software, como o tipo de *layer*, elas podem ser acessadas e modificadas através da função

setvar (Tabela 1). A Tabela 3 contém alguns dos tipos de variáveis de ambientes existentes no CAD que podem ser manipuladas pelo programador, de acordo com AutoCAD (2021).

Tabela 2 – Tabela de tipos de variáveis do AutoLISP.

TIPOS DE VARIÁVEIS	
SINTAXE	**TIPO DE VARIÁVEL**
getint	Valor numérico inteiro
getreal	Valor numérico real
getstring	Textos e caracteres
getpoint	Ponto
getkword	Palavras chaves para a função *initget*
entsel	Seleção de objetos
getcorner	Ponto com caixa de seleção

Fonte: Autoria Própria.

Tabela 3 – Tabela de variáveis de ambiente do AutoLISP.

VARIÁVEIS DE AMBIENTE		
SINTAXE	**FUNÇÃO**	**OBSERVAÇÕES**
clayer	Variável de *layer*	
ATTDIA	Ligar e desligar caixa de diálogo dos blocos com atributo	valores <0/1>
MIRRTEXT	Ligar e desligar espelhamento de textos	valores <0/1>
osmode	Variável de *Osnap*	valores <0/1>
cmdactive	Variável que indica se um comando está ativo	

Fonte: Autoria Própria.

Outros elementos de sintaxe do dialeto AutoLISP que o programador deve conhecer são os símbolos, eles servem como artifício para desempenhar pequenas funções quando colocados em meio ao código e até nos próprios comandos, Tabela 4.

Tabela 4 – Tabela de simbologias usadas nos códigos do AutoLISP.

SIMBOLOGIAS		
SINTAXE	**FUNÇÃO**	**EXEMPLO**
;	Adicionar comentário	
c:	Entrada na barra command	
_	Artifício para executar o comando do CAD em qualquer idioma	"_comando"
pause	Pausa o Lisp e espera o usuário dar continuidade ao comando	
\n	Pular uma linha	
!	Na barra de comando informa o valor de uma varável	!variável
-	Executa um comando sem a caixa de diálogo	"-comando"
_non	Desliga temporariamente o osnap	pode ser usado para definir um snap específico na rotina
/=	Sinal de diferença	
" 'type "	Sintaxe usada para se referir a um tipo de variável (onde "TYPE" é substituído pelo tipo de variável em questão)	pode ser usado para especificar uma condição

Fonte: Autoria Própria.

Portanto, nesse capítulo foram apresentados os conceitos dos comandos básicos, como *Line*, *Offset*, *Trim*, *Extend*, *MText*, *DText*, *Block* e *Layer*, que são utilizados para a representação da planta baixa utilizada como desenho base nessa pesquisa. Esses comandos são necessários para desenvolver as etapas iniciais do desenho arquitetônico, como as paredes e esquadrias. Além da execução do próprio desenho, o conhecimento sobre os comandos também é usado ao longo do Capítulo 4 para a criação das customizações, que inclui os comandos do AutoCAD dentro das rotinas.

Os conceitos e descrição da sintaxe das funções de AutoLISP são importantes para entendimento do algoritmo dos comandos customizados, apresentados ao longo do Capítulo 4. As tabelas apresentadas descrevem a sintaxe de cada uma das funções que foram utilizadas para desenvolver as customizações.

3 MATERIAL E MÉTODOS

Para desenvolver as customizações foi necessário adquirir conhecimento sobre a linguagem de programação e o desenvolvimento de suas rotinas. A capacitação no assunto foi feita através de videoaulas no YouTube em canais dedicados ao desenvolvimento de aulas sobre AutoLISP, leituras no fórum da Autodesk, no blog *Manual de Cad*, e no ambiente de Ajuda do software AutoCAD.

Foi utilizado o software AutoCAD para o desenvolvimento desta pesquisa que é um programa de desenvolvimento gráfico empregado na elaboração de desenhos técnicos com o auxílio de computador, o qual faz parte do conjunto de softwares denominados de CAD. Esse software possibilita a criação de entidades geométricas parametrizadas, utilizado comumente para o desenho de projetos de engenharia, além de outras áreas.

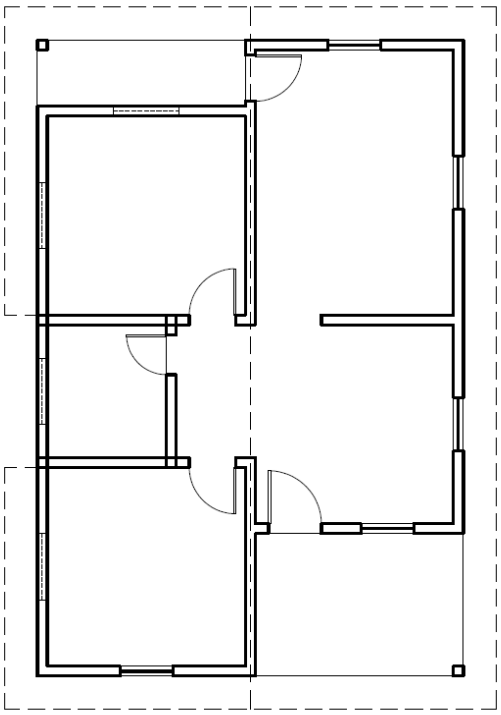
O desenvolvimento das rotinas de customização utilizadas nessa pesquisa foi feito à medida que se obteve conhecimento em AutoLISP que um dialeto de programação próprio do AutoCAD. Derivado da linguagem LISP, o AutoLISP utiliza um console de programação interno, o Visual LISP, que é um visualizador de algoritmo próprio para a programação em AutoLISP que permite a aplicação das rotinas diretamente nos desenhos do CAD.

Foi observado, através da experiência com o uso do AutoCAD na elaboração de desenhos arquitetônicos, os processos mais repetitivos durante a elaboração de uma planta baixa que poderiam ser resumidos através da utilização da programação LISP e que resultasse em economia de tempo na elaboração do desenho. Assim, foram desenvolvidas uma rotina para a inserção de blocos de esquadrias, como portas e janelas, através de um único comando e uma rotina de inserção de etiquetas de cômodos que calcula e insere o valor da área de forma automatizada.

3.1 O desenvolvimento da planta baixa

O projeto arquitetônico utilizado para a aplicação das rotinas em AutoLISP é o modelo de uma residência simples constante em uma apostila de material didático organizado para uso na disciplina Projeto Auxiliado por Computador, ofertada a diversos cursos da Universidade Federal Rural do Semi-Árido (UFERSA). Para o objetivo dessa pesquisa a planta baixa não precisou ser desenvolvida até a etapa de inserção de hachuras, blocos sanitários e cotas, precisando apenas da representação das linhas de paredes e pisos do projeto residencial (Figura 15).

Figura 15 – Planta Baixa modelo utilizada para a execução das rotinas.

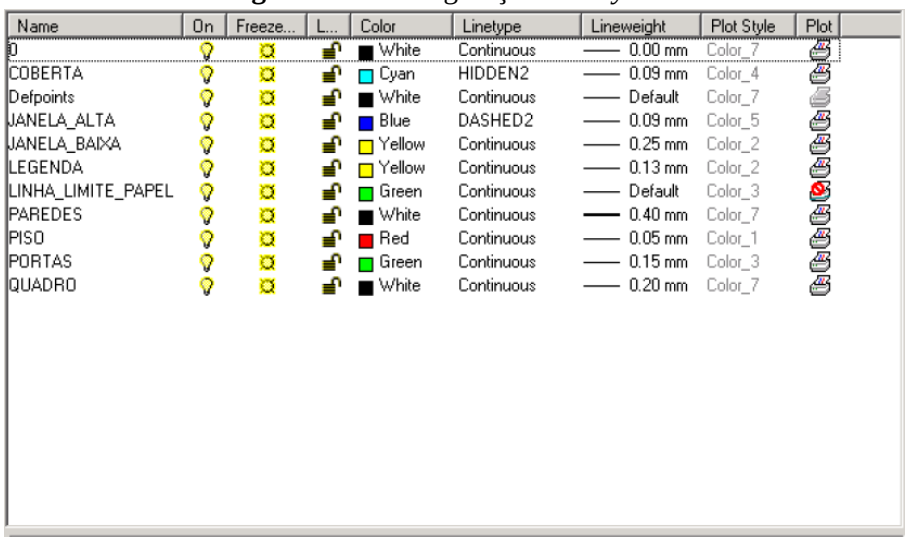


Fonte: SATHLER, Nilson de Sousa - Projeto Auxiliado por Computador (2010).

Para dar início ao desenho arquitetônico no AutoCAD foi necessário começar com a criação das *layers*, pois, como é possível observar na Figura 15, existem diferenças nos tipos e espessuras das linhas que representam paredes, piso, esquadrias e projeção do telhado, além de outros elementos de projetos, como hachuras, textos, cotas e peças sanitárias, que possuem configurações de desenho específicas. As *layers* adotadas no desenho feito nessa pesquisa seguiram os ensinamentos realizados por Sathler (2010) em sua apostila, como está listado na Figura 16.

Para a criação das paredes iniciou-se criando, com o comando *Line*, uma linha horizontal, e outra vertical concorrente e ortogonal à primeira. Essas duas linhas representarão as paredes mais externas do desenho, e a partir delas foi possível criar todas as outras linhas da planta com o comando *Offset*. O desenhista deve colocar as primeiras linhas na *layer* "PAREDES" assim todas as cópias feitas com o *Offset* já estarão na camada correta. O comando deve estar na opção *Through* (T), pois assim foi possível determinar distâncias variáveis entre a linha base do *Offset* e a próxima que foi copiada. Para as espessuras de parede a distância fornecida foi de 0.15 u.d., e, para as demais, a distância variou de acordo com as determinações do projeto. Ao final desse processo obteve-se espécie de croqui da planta baixa final, porém com algumas linhas a mais. Para finalizar essa etapa do desenho o foi necessário identificar as sobras de linhas que não fazem parte do projeto e cortá-las usando o comando *Trim*.

Figura 16 – Configurações de *layers* adotadas.

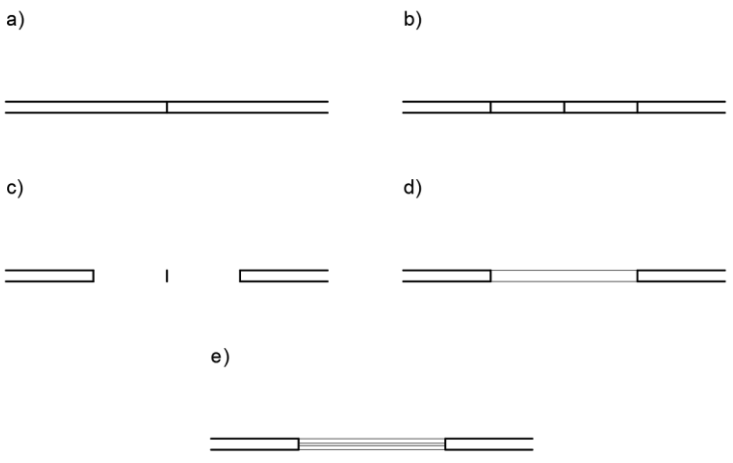


Fonte: SATHLER, Nilson de Sousa - Projeto Auxiliado por Computador (2010).

3.2 Customização para desenho de portas e janelas

Existem algumas formas diferentes de desenhar a representação de esquadrias no AutoCAD. As Janelas podem ser desenhadas através dos comandos *Line* e *Offset*, porém antes de representar a esquadria o projetista deve fazer os cortes de abertura das janelas baixas nas paredes. Para isso, deve-se primeiramente, desenhar uma linha de referência na parede indicando o ponto médio da janela como apresentado na Figura 17(a) e, em seguida, com um *Offset* para esquerda e para a direita estará delimitado o vão da janela (Figura 17(b)). Depois o usuário deve cortar as paredes dentro do vão com o comando *Trim* (Figura 17(c)) e, com a *layer* correta selecionada, desenhar duas linhas entre o vão substituindo as linhas de parede que foram cortadas (Figura 17(d)). Por último deve-se fazer um *Offset* das duas últimas linhas desenhadas com distância de 0.06 u.d. para a parte interna da janela, concluindo a representação da esquadria (Figura 17(e)).

Figura 17 – Etapas do desenho de uma janela no AutoCAD.



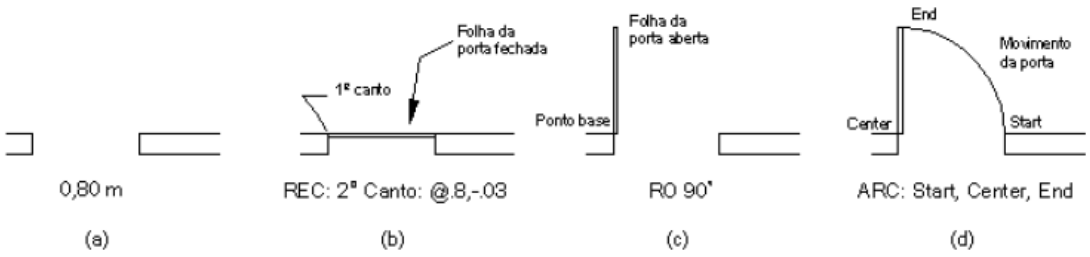
Fonte: Autoria Própria.

Para determinar o vão das portas o procedimento é similar, sendo necessário fazer a marcação e corte dos vãos que receberam as portas. Para isso utiliza-se o comando *Offset*, onde inicialmente é desenhada uma linha no início do vão, demarcando o primeiro canto, e por *Offset* desse ponto com uma distância equivalente à largura da porta foi demarcada a largura. Para finalizar a abertura utiliza o *Trim* para cortar as linhas dentro do vão. A folha ou lâmina da porta é desenhada perpendicularmente à posição do vão, como na Figura 18(c). A folha pode ser feita com *Line* e *Offset*, ou outros comandos como *Rectangle* e *Polyline*, de acordo com a familiaridade do desenhista. Por último, representa-se o movimento da porta, que consiste em uma projeção da rotação da esquadria em torno do seu eixo. Essa representação é feita com o comando *Arc*, desenhando um arco entre as extremidades da lâmina e do vão que se encontravam opostas à boneca da porta, Figura 18(d).

As esquadrias também podem ser feitas em forma de bloco, sendo inseridas como um objeto único no desenho, sem a necessidade de fazer um novo desenho para todas as portas e janelas do projeto. Para poder criar o bloco, é necessário desenhá-las como descrito anteriormente pelo menos uma vez. No caso das portas criou-se um bloco para portas de 60 cm, 70 cm, e 80 cm de largura real, pois depois de criado não é possível mudar a dimensão do objeto. Para criar o bloco para as janelas foram desenhadas pequenas janelas com largura de 0.10 u.d. Assim, quando o vão da esquadria for de 1.50 u.d., basta informar, na inserção do bloco, o fator de multiplicação do eixo da largura de 15, podendo repetir esse processo para as diferentes dimensões de janela existente no projeto.

Dessa forma, para automatizar a inserção das esquadrias, uma vez que diferentes objetos precisam ser inseridas no desenho, foi criado o comando “esq” que permite ao usuário inserir vários blocos de diferentes tipos e dimensões de portas e janelas em um mesmo comando.

Figura 18 – Representação da porta em planta baixa.



Fonte: SATHLER, Nilson de Sousa - Projeto Auxiliado por Computador (2010).

3.3 Customização para a medição de área e inserção das etiquetas de cômodo

As etiquetas de cômodos são compostas por duas linhas de texto e usadas para comunicar informações construtivas, contendo nome de um ambiente e a indicação do valor da sua área. A execução dos textos que representam a etiqueta de cômodo pode ser feita usando o comando *Mtext*. Se o usuário estiver trabalhando com a *Ribbon* na área de trabalho, é possível formatar um texto de múltiplas linhas direto dela, onde são exibidas opções do tipo *Text Formatting.* Para criar a etiqueta o desenhista necessita, primeiro, escolher a *layer* referente aos textos e, em seguida, ativar o comando *Mtext.* Dentro do comando deve escrever o nome e a área do cômodo que está sendo representado (Oliveira, 2014).

De acordo com Oliveira (2014), para medir a área usa-se o comando *Area,* que executa o cálculo através da definição de pontos ou por seleção de objetos. Ao optar pela seleção de ponto, o usuário deve selecionar de 1 a 7 entradas para definir um polígono, e ao pressionar *Enter*, a função informa o valor da sua área e o perímetro. Já a opção *object* calcula esses valores através da entrada de seleção de um objeto. Após calcular o valor da área o desenhista pode inseri-la na etiqueta de cômodo editando o texto que foi inserido anteriormente.

Como esse é um processo que deve ser repetido diversas vezes durante o desenho de uma planta baixa, foi desenvolvido o comando “aarr” que automatiza a etapa de cálculo de área, determinando esse valor no momento da inserção do texto de etiqueta.

4 DESENVOLVIMENTO

Durante essa pesquisa foram desenvolvidas duas customizações voltadas para a agilização de processos de um desenho arquitetônico. Nesse capítulo é mostrado como o algoritmo dos comandos customizados foram escritos dentro do Visual LISP, utilizando funções do AutoCAD e artifícios da linguagem de programação para o seu funcionamento. Para uma melhor compreensão da descrição do programa desenvolvido, é importante entender como se diferencia a escrita de funções próprias do CAD e funções do AutoLISP, e como os elementos da linguagem de programação são diferenciados dentro do console através das cores.

Para fazer um algoritmo executar uma função do AutoCAD usa-se "*command*" seguido do nome do comando e cada um dos seus argumentos separados por aspas. Na execução de um *command* pelo AutoLISP cada par de aspas é entendido como o pressionamento da tecla *Enter*, e os argumentos escritos entre elas são entendidos como entradas digitadas no teclado. No exemplo "(command "erase" "L" "")", o comando ativado é o *erase*, o argumento "L" é uma seleção de objeto pedida pelo comando, e as aspas no final representam um *Enter* que serve para finalizá-lo. Essa sequência de argumentos é característica desse comando em específico, para executar outras funções dentro do LISP é necessário conhecer a sequência de ações de cada uma.

Já os comandos do AutoLISP são ativados quando colocados entre parênteses, assim como os comandos do CAD é preciso conhecer quais tipos de argumentos precisarão ser definidos para o funcionamento da função, pois todos têm um limite mínimo e máximo que deve ser respeitado para que sejam executados corretamente.

Dentro do Visual LISP os tipos de códigos são diferenciados por cores de acordo com a sua função na sintaxe (Figura 19), conforme descrito abaixo:

- Argumentos numéricos: **verde**
- Parênteses que demarcam fim e início de função: **vermelho**
- Argumentos entre parênteses, *strings* e entradas do teclado: **magenta**
- Comando do AutoCAD, AutoLISP e símbolos que exercem funções: **azul**
- Comentários do programador: roxo com realce cinza
- Nomes de macros, variáveis e outros: **preto**

Figura 19 – Exemplo da diferenciação de cores no console Visual LISP.

```
; comentário:
(setq aarr_p2 (list x_p1 (- y_p1 0.3) 0.0))
(setq texto_area (strcat "A: " (rtos area 2 2) "m²"))
(command "text" "j" "mc" aarr_p1 0.15 0 comodo )
(command "text" "j" "mc" aarr_p2 0.15 0 texto_area )
(command "_erase" entity_name "")
(resetvar)
(quest)
```

Fonte: Autoria Própria.

4.1 Funções Auxiliares

No início do programa foram desenvolvidas quatro funções auxiliares que são utilizadas no carregamento do arquivo e dentro dos novos comandos criados. A primeira delas é a função *load*, escrita na primeira linha do código (Figura 20), que funciona como um comando de carregamento, servindo para carregar rapidamente o arquivo ".lsp" sempre que alguma atualização for feita na customização, facilitando a execução de testes pelo programador. O artifício "c:k" escrito à frente de *defun* determina a letra "k" como atalho de acesso à essa função na janela de comandos do AutoCAD.

Figura 20 – Funções auxiliares do código.

```
(defun c:k () (load "c:/AutoLISP/LISP_01.lsp"))

;
; UNIVERSIDADE FEDERAL RURAL DO SEMI ÁRIDO
; TRABALHO DE CONCLUSÃO DE CURSO
; DESENVOLVIMENTO DE AUTOLISP APLICADA A
; MODELAGEM DE DESENHOS ARQUITETÔNICOS NO AUTOCAD
; por: RAFAELA DE OLIVEIRA DIAS
;


; AUXILIAR      extração de variaveis   26/02/2020

;essa função guarda
;as variáveis correntes do
;programa antes de executar
;uma custom

(defun safevar ()
  (setq varlayer (getvar "clayer"))
  (setq varosmode (getvar "osmode"))
)

; AUXILIAR      devolução de variaveis  26/02/2020

;essa função devolve
;as variáveis
;correntes quando
;a custom é finalizada

(defun resetvar ()
  (setvar "clayer" varlayer)
  (setvar "osmode" varosmode)
)

; AUXILIAR      eco de carregamento     26/02/2020

(prompt "\nSuccessfully Loaded")
(princ)
```

Fonte: Autoria Própria.

Em seguida está escrita a função de extração de variáveis, "*safevar*", que serve para armazenar no algoritmo as variáveis ambientes correntes do programa. Guardar essas variáveis permite que elas sejam modificadas durante a execução do comando e ao final dele sejam devolvidas, evitando atrapalhar a usabilidade do desenhista. Assim, complementando a função "*safevar*", foi criada a função "*resetvar*", para colocar nas variáveis ambientes os valores que elas possuíam antes da execução dos comandos. Na Figura 20 observa-se que no algoritmo desenvolvido as variáveis ambientes que foram modificadas durante os comandos foram as de *layer* corrente, *clayer*, e a variável que determina se o *Osnap* está ligado ou desligado, *osmode*.

A última função auxiliar criada no programa emite uma mensagem na janela de comandos do AutoCAD quando o algoritmo é carregado sem nenhum erro. Para emitir essa mensagem foi usada uma função própria do AutoLISP, *prompt*, que emite a mensagem automaticamente, quando a customização é lida. Na Figura 20, essa função auxiliar foi posicionada no início do algoritmo para fins ilustrativos desse tópico. No arquivo ".lsp" do programa a função *prompt* se encontra na última linha do algoritmo, assim se existir algum erro ao longo do código, a mensagem "*Successfully Loaded*" não será exibida. O *princ*, que está abaixo do *prompt* foi usado para eliminar o eco de nulidade de variáveis do AutoLISP, *nil*, que indica no final das funções que os valores das variáveis se encontram nulas. Esse artifício foi utilizado ao final de todos os comandos desenvolvidos.

4.2 Comando para Inserção de Esquadrias (esq)

O comando de inserção de esquadria foi desenvolvido com o objetivo de ser um comando para inserir janelas e portas de quaisquer dimensões de forma ágil com uma única função, apenas com entradas de seleção de pontos. O atalho para esse comando é "esq" (Figura 21), assim quando o arquivo ".lsp" for carregado no desenho e o usuário digitar o atalho na janela de comandos, a função será ativada. A função *Lisp* que cria uma macro, ou comando, é *defun*, e para atribuir um atalho usa-se o artifício "c:" no segundo argumento da função, seguido das letras de atalho. Ao ser iniciado, o algoritmo executa os comandos auxiliares, primeiro a função "*safevar*" que guarda os valores das variáveis ambientes do programa antes da execução de "esq". Em seguida, a função *command*, que ativa comandos do AutoCAD dentro da customização, foi utilizada para criar uma *layer* específica para as esquadrias que serão inseridas. O símbolo "-" que acompanha a palavra "layer", é um artifício para executar comandos que possuem caixa de diálogo apenas com interações na janela de comandos, isso é necessário para evitar que os algoritmos sejam interrompidos pelas caixas de diálogos, uma vez que as opções exibidas nas caixas interativas só podem ser selecionadas pelo usuário. Quando uma nova *layer* é criada, essa nova camada é colocada como corrente, por isso ao final da primeira parte do comando se executa um "*resetvar*", devolvendo as variáveis ambientes, e consequentemente redefinindo a *layer* corrente para o que estava sendo usado anteriormente pelo desenhista.

Figura 21 – Parte inicial do algoritmo da função “esq”.

```
;
; 1-    inserção de esquadrias em planta baixa com looping              27/05/2020
;

(defun c:esq ()
  (safevar)
  (command "-layer" "m" "ESQUADRIAS" "c" "3" "" "l" "continuous" "" "lw" 0.09 "ESQUADRIAS" "")
  (resetvar)


  ; Primeira quest do programa, pergunta o tipo de esquadria.
  (defun quest ()
    (initget "Janela Porta")
    (setq ans (getkword "\nSelecione o tipo de esquadria: [Janela / Porta] "))
    (cond
      ((or (= ans "J") (= ans "Janela"))
       (quest_j)
      )
      ((or (= ans "P") (= ans "Porta"))
       (quest_p)
      )
    )
  )
```

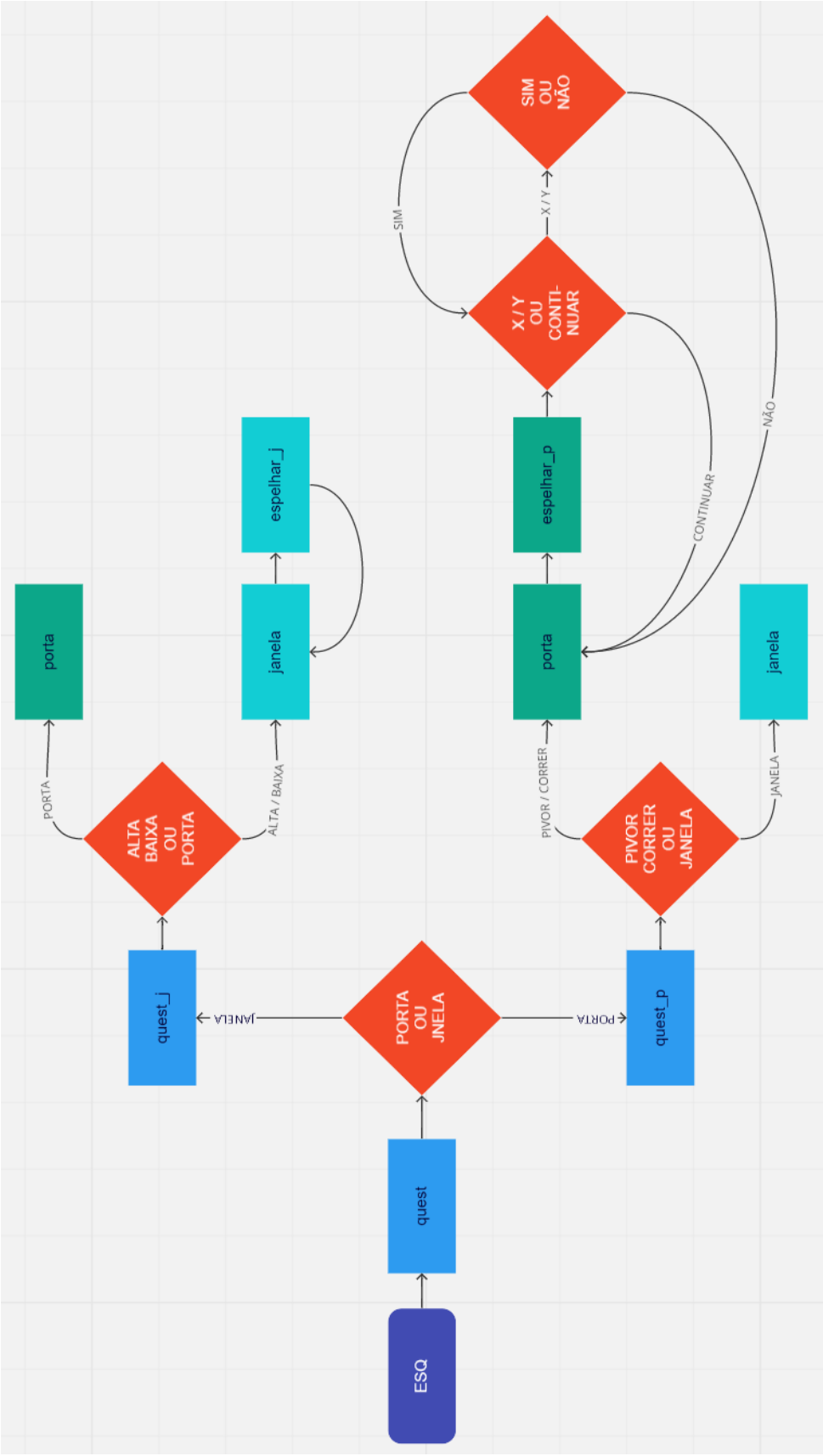
Fonte: Autoria Própria.

Essa primeira parte é composta de funções internas que não possuem interação entre o desenhista e o programa, a parte interativa do comando começa a partir da macro “quest” (Figura 21). A partir dela é definido o escopo de como acontecerá o fluxo da função “esq” de acordo com as decisões lógicas determinada pelo usuário, funcionando da seguinte forma:

1. O comando pergunta qual tipo de esquadria deseja-se inserir entre os tipos “porta” ou “janela”;
2. Se a resposta for “janela” ou “J”, será executada a macro “quest_j”, para inserir as janelas;
3. Se a resposta for “porta” ou “P”, a macro executada é “quest_p”, para inserir as portas.

O fluxograma da Figura 22 demostra a composição de todas as decisões lógicas ao longo do comando “esq”. Para criar essa condição primeiro foi usada a função *initget*, que limita as opções de resposta para uma pergunta, nessa parte do comando as entradas permitidas são as *strings* “Janela” e “Porta” (Figura 21).

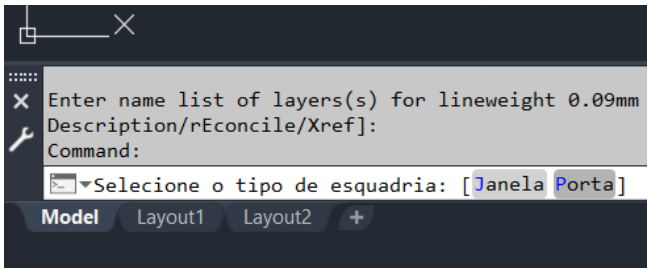
Figura 22 – Fluxograma do algoritmo do comando “esq”.



Fonte: Autoria Própria.

Quando o *initget* recebe argumentos do tipo *string*, a função também reconhece as letras iniciais dessas palavras como entradas válidas, assim “P” e “J” também serão aceitas como resposta. A pergunta de interação que aparece para o usuário é “Selecione o tipo de esquadria [Janela/Porta]”, as palavras chave foram colocadas entre colchetes para que suas iniciais apareçam destacadas na janela de comandos, como na Figura 23. Assim, é possível selecionar essas entradas clicando com o *mouse* nas letras em azul. A resposta dada pelo usuário é extraída pela função *getkword* e armazenada em uma variável denominada *ans* através de um *setq*. Por fim, é executada a condicional *cond* que avalia o valor de *ans* para executar alguma das condições definidas. Se o valor de *ans* for “janela” ou “J” a *cond* executa a macro (quest_j), que insere janelas; e se o valor for “porta” ou “P” a macro executada será (quest_p).

Figura 23 - *Strings* colocadas entre colchetes aparecendo na janela de comandos.



Fonte: Autoria Própria.

4.2.1 Inserção das janelas

Quando é selecionada a opção de janelas na primeira interação, o comando, através da condição, entra na macro de inserção de janela “quest_j”. Essa parte do comando também recebe nome de *quest*, pois é composta por uma condição que avalia uma variável a partir de uma pergunta feita ao usuário. Dessa vez o *prompt* pede ao desenhista que escolha entre os tipos de janela, sendo que as entradas limitadas são “Alta” “Baixa” e “Porta”. A opção de porta foi incluída para ser possível mudar com facilidade para o outro tipo de esquadria. As duas primeiras linhas dessa macro são similares às da função descrita anteriormente, porém aqui a condição utilizada é do tipo *while*, que executa uma função enquanto a resposta para a condição for verdadeira, criando um *looping*. Nesse caso, as entradas verdadeiras são “Alta” ou “A”, “Baixa” ou “B”, se a entrada do usuário foi “P” ou “Porta” a condição irá sair do *looping* e executar a macro “quest_p”.

Ao escolher entre os tipos de janelas a rotina é direcionada para a inserção dessas esquadrias, executando “janela” (Figura 24). Essa parte do comando é escrita com uma *cond* contendo dois argumentos, um para as entradas “B” ou “baixa e a outra para as entradas “A” ou “alta”. Depois de entrar na condição, o programa pede ao usuário que clique em dois pontos da tela, o ponto inicial e final do vão (Figura 25). Esses pontos são guardados respectivamente nas variáveis “j1” e “j2”, em seguida é ativado o comando *insert* que irá

inserir o desenho da esquadria em formato de bloco, que foi previamente salvo no disco local. Para o encaixe correto do bloco no vão, o programa determina um fator em “x” que é determinado pela distância dos pontos j1 e j2, e da mesma forma calcula o ângulo entre os pontos e o utiliza para determinar a rotação do objeto inserido.

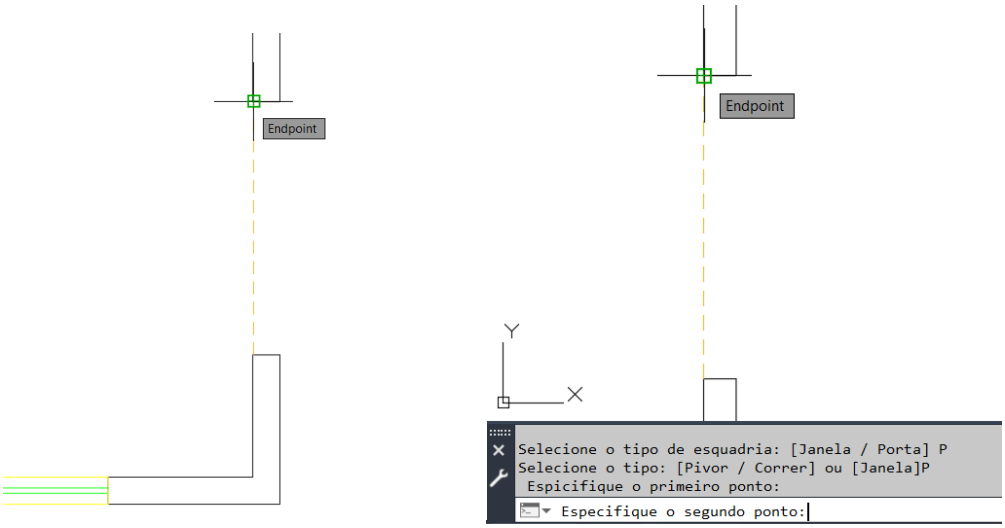
Depois de inserir o bloco, a função irá mudar a *layer corrente* para aquela criada no início da rotina, denominada “ESQUADRIAS”. Para desenvolver a lógica dessa parte é preciso entender que as informações de propriedades dos objetos desenhados no AutoCAD são armazenadas em pares ordenados dentro de uma lista (Figura 26). O par ordenado é composto por um número que representa o tipo de propriedade, e um segundo argumento que mostra o valor atribuído àquela propriedade. O número que representa a *layer* de um objeto é o 8, e seu par ordenado contém como segundo argumento uma *string* com o nome da camada corrente da entidade.

Figura 24 – Algoritmo da parte de inserção de janela do comando “esq”.

```
; Janela:
; Segunda quest do programa, pergunta qual tipo de janela será inserido.
(defun quest_j ()
  (initget "Alta Baixa Porta")
  (setq ans_j (getkword "\nSelecione o tipo de janela: [Alta / Baixa] ou [Porta] "))
  (while (or (= ans_j "Alta") (= ans_j "A") (= ans_j "Baixa") (= ans_j "B"))
    (janela))
  (quest_p)
)

; Condicional que irá inserir a janela de acordo com o tipo selecionado.
(defun janela ()
  (cond
    ((or (= ans_j "A") (= ans_j "Alta"))
     (setq j1 (getpoint "\n Espicifique o primeiro ponto: ")
           j2 (getpoint j1 "\n Especifique o segundo ponto: "))
     (command "-insert" "c:/autolisp/blocos/janelaalta" "x" j1 j2 j1 j2)
     (setq entity_list (entget(entlast)))
     (setq entity_list (subst (cons 8 "ESQUADRIAS") (assoc 8 entity_list) entity_list))
     (entmod entity_list)
     (espelhar_j)
    )
    ((or (= ans_j "B") (= ans_j "Baixa"))
     (setq j1 (getpoint "\n Espicifique o primeiro ponto: ")
           j2 (getpoint j1 "\n Especifique o segundo ponto: "))
     (command "-insert" "c:/autolisp/blocos/janelabaixa2" "x" j1 j2 j1 j2)
     (setq entity_list (entget(entlast)))
     (setq entity_list (subst (cons 8 "ESQUADRIAS") (assoc 8 entity_list) entity_list))
     (entmod entity_list)
     (espelhar_j)
    )
   )
  )
```

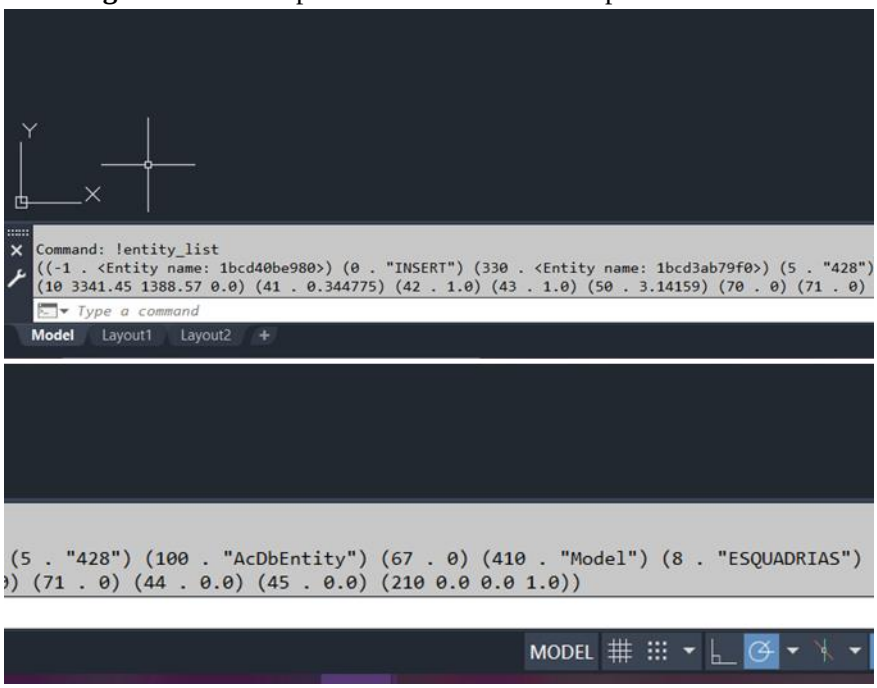
Fonte: Autoria Própria.

Figura 25 – Inserção dos pontos j1 e j2 em um vão.



Fonte: Autoria Própria.

Figura 26 – Exemplo de uma lista contendo pares ordenados.



Fonte: Autoria Própria.

Os pares ordenamos podem ser criados, através da função *cons*, e modificados com a função *subst.* Para mudar o *layer* do bloco inserido foi necessário obter a lista do objeto, através da função *entget*, e salvá-la numa variável de nome "*entity_list*". Em seguida foi feita a manipulação da lista substituindo o valor antigo do par ordenado 8, por um novo par criado com *cons* contendo a nova *layer*. A linha de expressão dessa parte foi escrita da seguinte forma:

(setq entity_list (subst (cons 8 "ESQUADRIAS") (assoc 8 entity_list) entity_list)).

Para finalizar essa *Quest*, executa-se um *entmod* para que a lista seja atualizada e o objeto com a nova *layer* possa ser visualizado na tela gráfica.

A condição para a inserção de janela baixa é exatamente igual, mudando apenas o nome do bloco inserido, como pode ser observado na Figura 24.

No final das condicionais da inserção de janelas, a rotina executa uma macro chamada “espelhar_j” (Figura 27). Nessa parte o programa pergunta se o usuário gostaria de espelhar o objeto recém inserido, essa parte foi criada para evitar que a janela fique representada para fora da parede, devido à ordem de seleção dos pontos. A lógica utilizada também foi uma condicional, onde as respostas possíveis são “sim” ou “não”, caso o usuário responda sim, o objeto será espelhado em torno do eixo definido por j1 e j2, e em seguida executa novamente a macro “janela” criando uma espécie de *looping* dentro do programa, permitindo que o usuário continue a inserir mais objetos sem precisar ativar o comando “esq” novamente. Se a resposta for “não” o comando apenas executa “janela” para que o desenhista prossiga com a criação das janelas no projeto.

Figura 27 – Algoritmo de espelhamento das janelas.

```
; Condicional que irá espelhar ou não a janela inserida pelo usuário.
(defun espelhar_j ()
  (initget "Sim Não Yes")
  (setq ans_j2 (getkword "\n Deseja espelhar o objeto? [Sim/Não] "))
  (cond
    ((or (= ans_j2 "S") (= ans_j2 "Sim") (= ans_j2 "Y"))
     (command "_mirror" "l" "" j1 j2 "y" )
     (janela)
    )
    ((or (= ans_j2 "N") (= ans_j2 "Não"))
     (janela)
    )
  )
)
```

Fonte: Autoria Própria.

4.2.2 Inserção das portas

Quando o desenhista opta pela opção “Porta” ou “P” no início da execução comando “esq”, o programa ativa a macro “quest_p” que é o primeiro passo para a inserção das portas. Assim como na macro “quest_j”, citada anteriormente, a “quest_p” é composta por uma condição que avalia a variável que é definida a partir da resposta do usuário à uma pergunta (Figura 28), essa variável define qual tipo de porta será inserida. As linhas de expressões da função “quest_p” é escrita seguindo a mesma lógica de decisão de “quest_j”, mudando apenas os tipos de entrada para o *initget*, e o *prompt* exibido na janela de comandos.

A função “porta” é formada por uma condicional, do tipo *cond*, que insere o bloco de representação das portas de acordo com o tipo escolhido pelo projetista. A primeira condicional tem como valor verdadeiro as entradas “P” e “Pivor”, que quando ativada pede ao

usuário os dois pontos das extremidades do vão onde será inserida a esquadria, armazenando essas entradas nas variáveis “p1” e “p2” (Figura 29). Quando um ponto é guardado em uma variável, os valores das suas coordenadas são salvos em formato de lista, onde o primeiro argumento da lista é o valor da coordenada “x”, o segundo é o valor de “y”, e o terceiro “z”. É possível extrair os valores x,y das listas usando *car*, função que obtém o primeiro argumento das listas, excluindo os outros argumentos; e *cdr*, que determina uma nova lista excluindo o primeiro argumento.

Figura 28 – Algoritmo da função “quest_p” na parte de inserção de portas do algoritmo.

```
; Porta:
; Terceira quest do programa, pergunta qual tipo de porta será inserido.
(defun quest_p (/ ans_p)
  (initget "Pivor Correr Janela")
  (setq ans_p (getkword "\nSelecione o tipo: [Pivor / Correr] ou [Janela]"))
  (while (or (= ans_p "Pivor") (= ans_p "P") (= ans_p "Correr") (= ans_p "C"))
    (porta))
  (quest_j)
)
```

Fonte: Autoria Própria.

Figura 29 – Algoritmo de inserção de portas (Parte I).

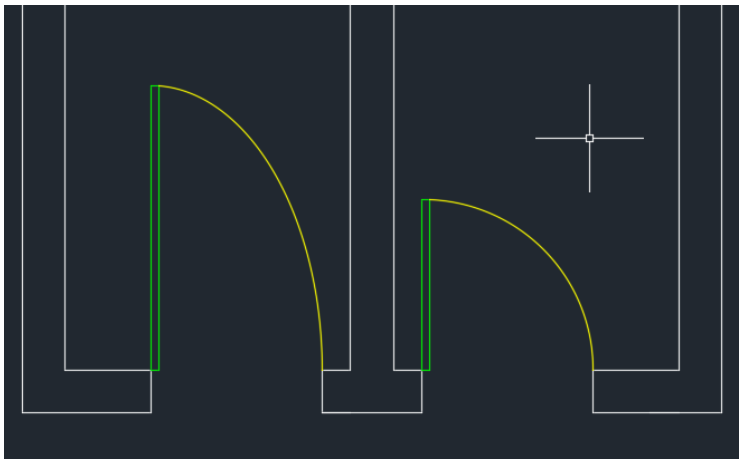
```
; Condicional que irá inserir a porta
; de acordo com o tipo selecionado.
(defun porta (/ p1 x_p1 y_p1)
  (setq p1 0)
  (cond
    ((or (= ans_p "P") (= ans_p "Pivor"))
     (setq p1 (getpoint "\n Espicifique o primeiro ponto:")
           p2 (getpoint p1 "\n Especifique o segundo ponto:")
           x_p1 (car p1)
           y_p1 (car(cdr p1))
           x_p2 (car p2)
           y_p2 (car(cdr p2))
     )

     (command "-insert" "c:/autolisp/blocos/portap" "x" p1 p2 "y" p1 p2 p1 pause)
     (setq entity_list (entget(entlast)))
     (setq entity_list (subst (cons 8 "ESQUADRIAS") (assoc 8 entity_list) entity_list))
     (entmod entity_list)
     (espelhar_p)
    )
```

Fonte: Autoria Própria.

Assim, depois que é obtido os pontos p1 e p2, o algoritmo configura automaticamente quatro novas variáveis, denominadas “x_p1”, “y_p1”, “x_p2” e “y_p2”, que guardam as respectivas coordenadas dos pontos p1 e p2 (Figura 29). Essas coordenadas serão utilizadas no espelhamento da esquadria, pois no caso das portas, a ação poderá ser acionada tanto em torno do eixo X, como em torno do eixo Y, de acordo com a necessidade do desenhista. Depois de todas as variáveis definidas, a macro ativa o comando CAD *insert* para inserir o

bloco de esquadria. Diferente da janela, a macro utiliza os pontos p1 e p2 para determinar o fator de multiplicação nos dois eixos, assim a porta ficará com o tamanho correto entre o vão, e na lâmina, como mostra a Figura 30. Depois de inserir o bloco, a macro repete o passo-a-passo para a mudança de *layer* do objeto exatamente como foi descrito anteriormente no algoritmo de inserção das janelas, e em seguida, é finalizada com uma chamada para a função de espelhamento de portas, denominada “espelhar_p”.

Figura 30 – Bloco de porta com fator de multiplicação aplicado apenas ao eixo X (esquerda) ao lado do bloco de porta com fator de multiplicação aplicado em X e Y (direita).



Fonte: Autoria Própria.

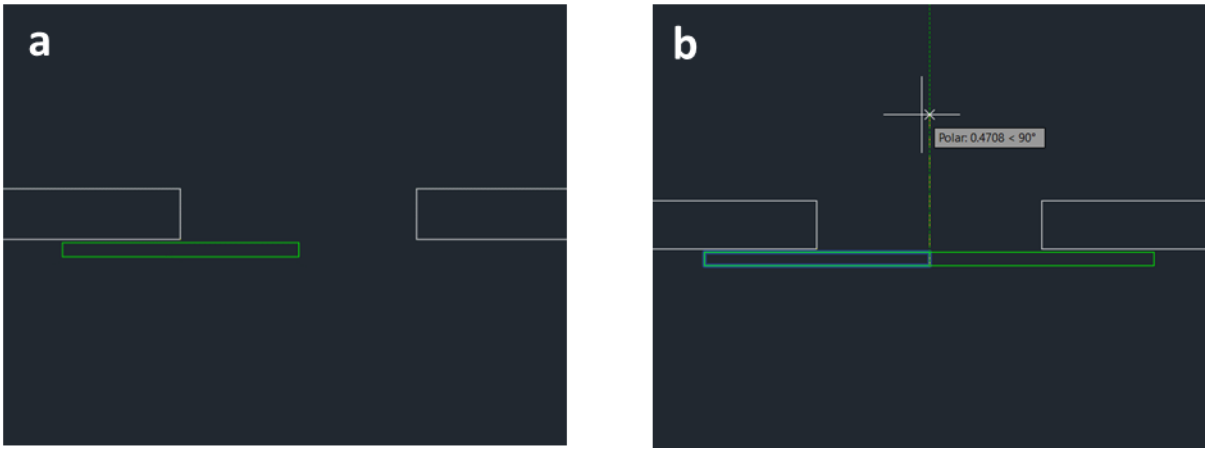
Se durante a execução da função “quest_p” o usuário optar pela opção “Correr” ou “C”, o valor da variável “ans_p” (Figura 28) será verdadeiro para a segunda condicional da função “porta”, a condição de inserção dos blocos de porta de correr (Figura 31). Inicialmente o algoritmo dessa condição se assemelha com a anterior, pedido que seja determinado dois pontos, dessa vez “p3” e “p4”, e armazenando os valores das suas coordenadas em variáveis. Porém para a inserção dessa esquadria serão determinadas mais duas variáveis, “med_x” e “med_y”, que representará as coordenadas de um ponto médio imaginário entre os pontos p3 e p4. A obtenção dessas coordenadas é necessária para espelhar o bloco corretamente, pois diferente dos outros blocos, a porta de correr é representada com um deslocamento do vão (Figura 32(a)), assim se o projetista desejar espelhá-la no eixo perpendicular ao desenho da porta, a linha de espelhamento deverá se localizar bem no meio do vão, como mostra a Figura 32(b).

Figura 31 – Algoritmo de inserção de portas (Parte II).

```
((or (= ans_p "C") (= ans_p "Correr"))
 (setq p3 (getpoint "\n Espicifique o primeiro ponto")
       p4 (getpoint p3 "\n Especifique o segundo ponto")
       x_p3 (car p3)
       y_p3 (car(cdr p3))
       x_p4 (car p4)
       y_p4 (car(cdr p4))
       med_x (/ (- x_p4 x_p3) 2)
       med_y (/ (- y_p4 y_p3) 2)
 )
 (cond
   ((= x_p4 x_p3)
    (setq x_p1 x_p3
          y_p1 (+ y_p3 med_y))
   )
   ((= y_p4 y_p3)
    (setq y_p1 y_p3
          x_p1 (+ x_p3 med_x))
   )
   ((and (/= y_p4 y_p3)(/= x_p4 x_p3))
     (setq y_p1 y_p3
           x_p1 x_p3)
   )
  )
 (setq p1 (list x_p1 y_p1 0))
 (command "-insert" "c:/autolisp/blocos/portac" "x" p3 p4 "r" p3 p4 p3)
 (setq entity_list (entget(entlast)))
 (setq entity_list (subst (cons 8 "ESQUADRIAS") (assoc 8 entity_list) entity_list))
 (entmod entity_list)
 (espelhar_p)
)
)
)
```

Fonte: Autoria Própria.

Figura 32 – a) Representação da porta de correr em planta baixa. b) Espelhamento da porta de correr a partir do meio do vão.



Fonte: Autoria Própria.

Depois da definição das variáveis, o algoritmo entra em outra *cond*, esta irá avaliar se o objeto está sendo inserido ao longo do eixo X ou Y. Essa etapa é necessária para que o eixo de espelhamento, que se originará no ponto de coordenadas médias, seja direcionado

corretamente, ou seja, na direção de Y, se o objeto estiver no eixo X, e na direção de X, se o objeto estiver em Y. Os argumentos da condição funcionam da seguinte forma:

1. Se os valores de “x_p3” e “x_p4” forem iguais, matematicamente entende-se que o objeto foi inserido ao longo do eixo Y, então a coordenada “med_y” é atribuída à variável “y_p1”.
2. Se os valores de “y_p3” e “y_p4” forem iguais, matematicamente entende-se que o objeto foi inserido ao longo do eixo X, então a coordenada “med_x” é atribuída à variável “x_p1”.
3. Se nenhuma das duas condições forem satisfeitas, o objeto foi inserido diagonalmente, nesse caso as coordenadas do ponto médio não serão utilizadas.

Depois dessa etapa o ponto “p1” de espelhamento é definido com *setq*, a partir das coordenadas obtidas com a *cond*, e ficará armazenada para ser utilizado na próxima etapa do programa. Por fim, o bloco é inserido com o comando *insert*, sendo que as portas de correr se assemelham com as janelas, tendo tamanho variável apenas em uma dimensão, por isso, aqui também só é usado o fator de multiplicação em “x”. As últimas quatro linhas da macro também são usadas para definir a *layer* do objeto, e ativar o próximo passo do programa, “espelhar_p”, como na inserção de portas de pivor.

A última etapa da inserção é o espelhamento das portas (Figura 33). A macro “espelhar_p” é mais complexa que a função de espelhamento das janelas, o posicionamento de uma porta na representação da planta baixa pode variar em torno de X e de Y, por isso no algoritmo da função foi criada uma interação que dá ao usuário duas opções de espelhamento, “X” e “Y”, e uma opção “continuar”, caso não seja desejado ativar essa funcionalidade. Dentro da condicional, os argumentos avaliam a resposta do usuário para executar o comando *mirror*, caso o espelhamento seja feito em torno de X, os pontos da linha de espelhamento serão “p1”, e um ponto “p5” determinado pelo programa, em que sua coordenada “x” será a soma de 1 u.d. com a cordenada “x_p1”. Se o espelhamento for em torno de Y, a coordenada “y” de “p5” recebe a soma de 1 u.d. mais o valor de “y_p1”. Após espelhar a esquadria, o comado aciona a função “quest_2”, que irá perguntar ao usuário se ele gostaria de continuar espelhando o objeto. Para isso, utilizando *while* foi criado um argumento para determinar que enquanto a resposta for “sim” o comando volta para a função “espelhar_p”, e em caso de “não”, o comando executa a macro “porta”, criando um *looping* na inserção das portas, para que o projetista continue inserindo mais blocos sem precisar reativar o “esq”. O mesmo *looping* também é ativado se a escolha do usuário for “continuar” para as opções em “espelhar_p”. A Figura 33 mostra o algoritmo das macros “espelhar_p” e “quest_p2”.

Figura 33 – Parte final do algoritmo da função “esq”.

```
; Condicional que irá espelhar ou não a porta.
(defun espelhar_p ()
  (initget "X Y Cancel")
  (setq ans_p3 (getkword "\n Selecionar eixo de espelhamento ou Continuar: [X/Y/Continuar] <C>"))
  (cond
    ((= ans_p3 "X")
     (setq p5 (list (+ x_p1 1) y_p1 ))
     (command "_mirror" "l" "" p1 p5 "y" )
     (quest_p2)
    )
    ((= ans_p3 "Y")
     (setq p5 (list x_p1 (+ y_p1 1)))
     (command "_mirror" "l" "" p1 p5 "y" )
     (quest_p2)
    )
    ((or (= ans_p3 "C") (= ans_p3 "Continuar"))
     (porta)
    )
  )
)

; Confirmção do espelhamento.
(defun quest_p2 ()
  (initget "Sim Não Yes")
  (setq ans_p2 (getkword "\n Continuar a espelhar? [Sim/Não] "))
  (while (or (= ans_p2 "Sim") (= ans_p2 "S") (= ans_p2 "Y"))
    (espelhar_p))
  (porta)
)

(quest)
(princ)
)
```

Fonte: Autoria Própria.

4.3 Comando para Inserção de Etiquetas de Cômodos (aarr)

O comando de inserção de etiquetas tem como objetivo ser uma função única para inserir identificadores de cômodos e executar o cálculo da sua área de uma única vez. O atalho definido para esse comando foi “aarr”, sendo que assim esta deve ser a entrada que deve ser dada na janela de comandos para a execução dessa customização. O algoritmo começa salvando as variáveis ambientes através do “safevar” (Figura 34), ao longo do comando são modificadas as variáveis de *layer* corrente e de *status* do *Osnap*. Em seguida é usado o comando *Layer*, através de um *command* para a criar uma camada própria para as etiquetas, com as seguintes definições: nome “ETIQUETAS”; cor número “4” (ciano); tipo de linha contínua; espessura 0,03 mm (Figura 34).

Depois da execução das funções auxiliares, o código segue para a primeira parte interativa do comando, chamada “quest” (Figura 34). Essa parte foi criada através da definição de uma função interna denominada *defun*, que pergunta ao usuário qual forma ele desejará desenhar a área do cômodo, tendo como opções de entrada “Polilyne” e “Boundary”. A primeira linha da função “quest” utiliza uma função do AutoCAD, *osmode*, para ativar o *Osnap*, essa parte é importante para garantir que o usuário tenha acesso aos pontos de precisão, caso eles estejam desligados. Na sequência do algoritmo é utilizada a linha de expressão “(setvar “clayer” “ETIQUETAS”)” para definir a camada recém criada como layer

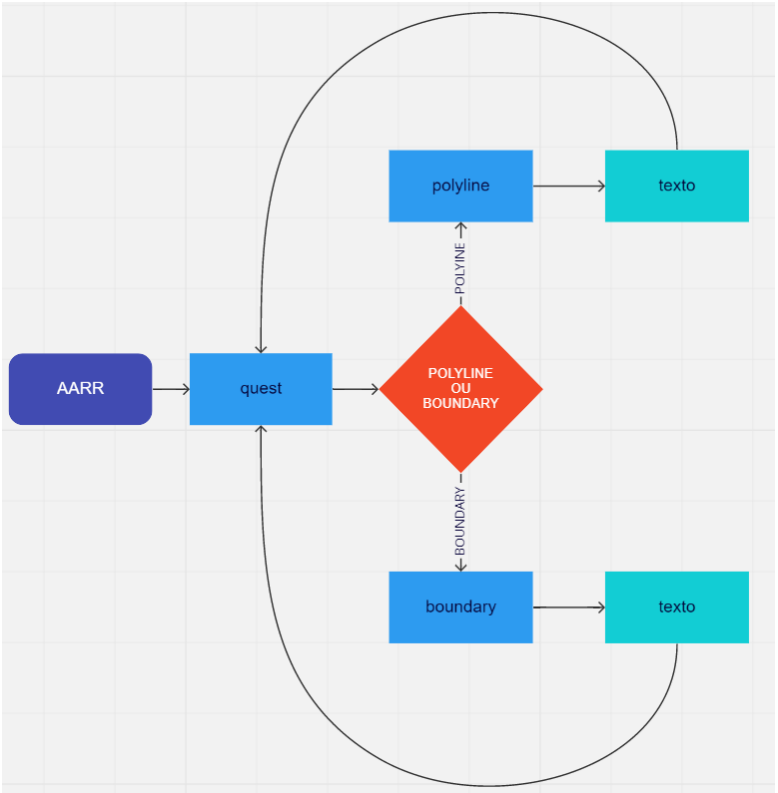
corrente. Para limitar as entradas foi usado o *initget* antes da definição da variável, tornando as *strings* “Polilyne” e “Boundary” e as letras “P” e “B” como entradas válidas para essa parte do comando (Figura 34). Na Figura 35 consta o fluxograma do funcionamento das decisões lógicas ao longo do comando.

Figura 34 – Algoritmo do comando “aarr”.

```
;
; 2-    Inserção de etiquetas de área          13/04/2021
;

(defun c:aarr ()
  (safevar)
  (command "-layer" "m" "ETIQUETAS" "c" "4" "" "l" "continuous" "" "lw" 0.03 "" "")

  ; Pergunta de seleção de área:
  (defun quest ()
    (setvar "clayer" "ETIQUETAS")
    (command "osmode" "1")
    (initget "Polilyne Boundary")
    (setq ans (getkword "\n Método de seleção da área: [Polilyne/Boundary]"))
    (cond
      ((or (= ans "P") (= ans "Polilyne"))
       (polilyne)
      )
      ((or (= ans "B") (= ans "Boundary"))
       (boundary)
      )
    )
  )
```

Fonte: Autoria Própria.

Figura 35 – Fluxograma do algoritmo do comando “aarr”.



Fonte: Autoria Própria.

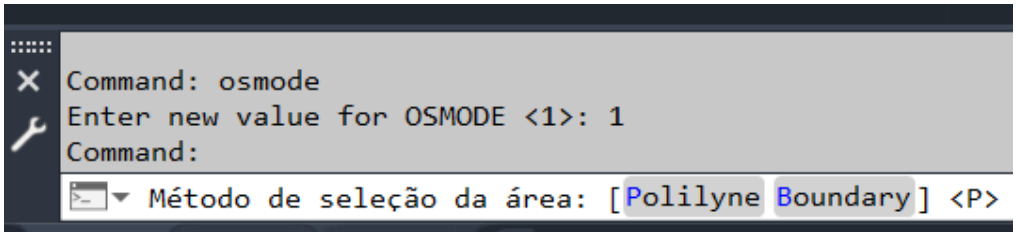
Quando o comando “aarr” é ativado, a primeira mensagem interativa que aparece na janela de comandos é: “Método de seleção da área: [Polilyne/Boundary]” (Figura 36), pedindo ao usuário que escolha entre as duas opções disponíveis, e armazenando a entrada na variável “ans”. De acordo com a opção escolhida pelo projetista, o comando executará a macro “polilyne” ou a macro “boundary”, a condição é avaliada por *cond*, em que o primeiro argumento é ativado para valores “P” ou “Polilyne”, e o segundo, caso o valor seja “B” ou “Boundary”.

Ao executar a função “polilyne” uma nova mensagem de interação será exibida na janela de comandos informando: “Desenhe a área fechada e aperte Enter”. Nesse momento o comando estará pausado para que o usuário desenhe uma poli linha representando área desejada. O desenho do objeto é executado com a função CAD *Pline*, como mostrado na Figura 37, porém para que o algoritmo entenda que o número de pontos necessários para definir uma área é indeterminado, foi preciso criar uma condicional que verifique se o comando *Pline* está ativado ou se foi encerrado pelo desenhista. Para isso usou-se a variável ambiente *cmdactive*, que indica se um comando foi finalizado ou se continua ativo. Assim, enquanto o usuário estiver definindo novos pontos, é necessário que o algoritmo fique pausado para receber as novas entradas. Essa parte do novo comando ficou escrita da seguinte forma:

(while (> (getvar ‘cdmactive) 0) (command pause)),

determinando que enquanto a variável *cmdactive* for maior que “0”, a leitura do código será pausada para novas entradas de pontos. Quando a tecla *Enter* é pressionada o valor de *cmdactive* se iguala a “0”, fazendo com que o programa saia do *looping* e prossiga a função. Depois do *looping*, uma variável “entity_name” é definida. Ela será utilizada para guardar o nome de entidade do último objeto desenhado, no caso a poli linha que define a área trabalhada.

Figura 36 – Interação do comando “aarr” na janela de comandos.



Fonte: Autoria Própria.

Para extrair o valor da área do objeto *Pline* foi preciso utilizar uma função do tipo *Visual Basic for Application* (VBA), que é uma incorporação da linguagem de programação da *Microsoft* à aplicativos de terceiros. Essa linguagem é usada para exportar dados do desenho para programas como o *Excel*, o que permite transformar dados de desenho em informações textuais. Para isso, primeiro foi feita a conversão do objeto em AutoLISP para VBA, utilizando a linha de expressão "(vlax-ename->vla-object entity_name)", após a conversão, a função *vlax-get* extraiu o valor da área e o armazenou na variável de mesmo nome definida através de *setq*, (Figura 37). Essa parte do código termina com "(redraw entity_name 3)", um comando LISP que, quando usado com o valor "3" em seu último argumento, redesenha um objeto destacando suas linhas; assim como com a macro "texto", que irá inserir o valor da área e o nome da etiqueta.

Caso na primeira interação da função "quest" o usuário opte por executar o comando com a função "boundary", a mensagem exibida será "Selecione um ponto dentro da área", conforme se observa na Figura 38. Após a determinação do ponto pedido pelo programa, o algoritmo executa a função *Boundary* do AutoCAD, que desenha automaticamente uma poli linha através do reconhecimento dos limites ao redor de um ponto. Utilizar essa função parece ser mais simples, pois só é necessário informar um ponto para determinar a área, mas nem sempre é possível aplicá-la satisfatoriamente em uma planta baixa, já que alguns elementos de desenho, como as portas, podem atrapalhar essa identificação com precisão (Figura 39). O restante do código foi escrito e funciona exatamente como na função *Polilyne*, como se observa na Figura 38.

Figura 37 – Algoritmo da função *Polilyne* dentro do comando "aarr".

```
; Polilyne:
(defun polilyne ()
  (prompt "\n Desenhe a área fechada e aperte Enter \n")
  (command "_pline")
  (while (> (getvar 'cmdactive) 0)
    (command pause))
  (setq entity_name (entlast))
  (setq area (vlax-get (vlax-ename->vla-object entity_name) 'area))
  (redraw entity_name 3)
  (texto)
)
```

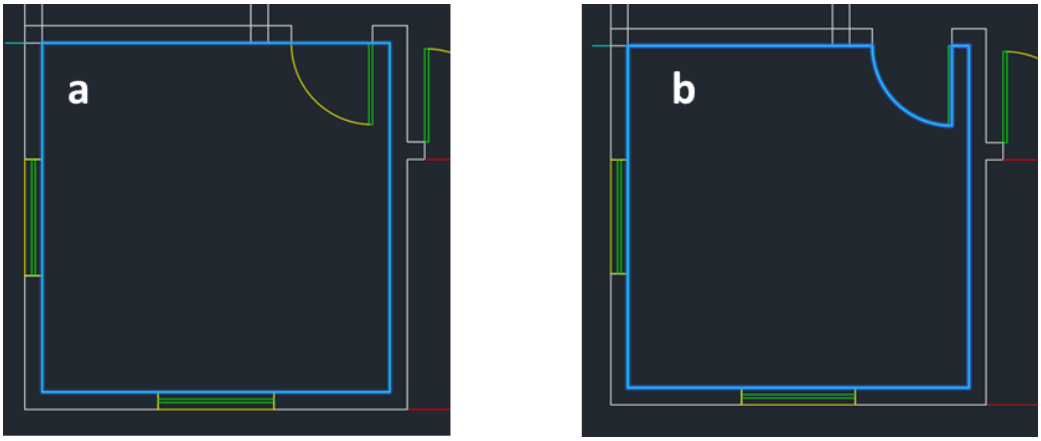
Fonte: Autoria Própria.

Figura 38 – Algoritmo da função *Boundary* dentro do comando "aarr".

```
; Boundary:
(defun boundary ()
  (setq p1 (getpoint "\n Selecione um ponto dentro da área \n"))
  (command "-boundary" p1 "")
  (setq entity_name (entlast))
  (setq area (vlax-get (vlax-ename->vla-object entity_name) 'area))
  (redraw entity_name 3)
  (texto)
)
```

Fonte: Autoria Própria.

Figura 39 – Procedimento de reconhecimento do contorno para determinação da área do cômodo. a) com utilização da função *polilyne* b) com utilização da função *boundary*.



Fonte: Autoria Própria.

A última parte do comando “aarr” é executada através da função “texto”, que insere a etiqueta de cômodo com o texto de identificação do cômodo e o valor da área no local da planta baixa que o usuário determinar. Esse algoritmo inicia com um comando para desligar o *Osnap*, passo necessário para evitar que os textos sejam inseridos em coordenadas erradas, caso estejam próximas à um ponto de precisão (Figura 40). Em seguida o programa interage com o usuário pedindo o nome de identificação do cômodo (Figura 41(a)), guardando essa informação na variável “comodo”. Após esse passo outra interação é feita pedindo o ponto de inserção da etiqueta, que será armazenado na variável “aarr_p1” (Figura 41(b)). Extraindo-se o *car* desse ponto, obtém-se sua coordenada “x”, e extraindo o *car* do *cdr* do mesmo ponto, obtém-se a coordenada “y”. Esses valores são salvos separadamente para que o código possa usá-los nas operações matemáticas na parte seguinte do comando.

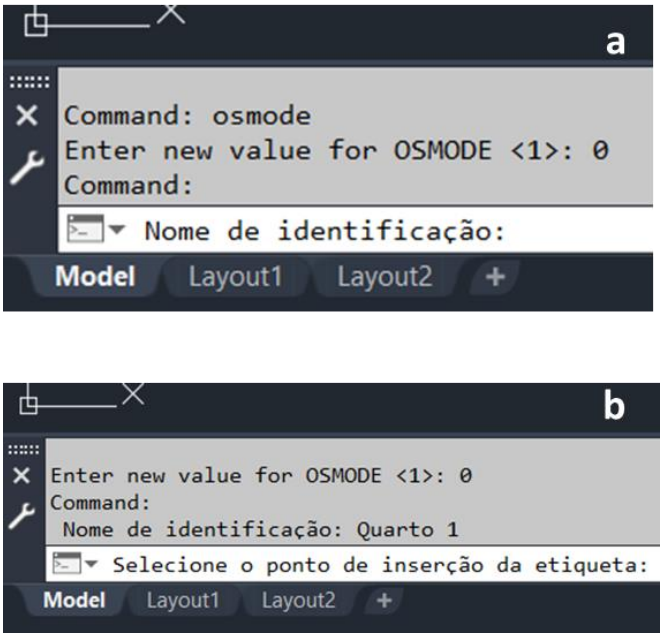
O ponto “aarr_p1” informado pelo desenhista será usado para a inserção do nome da etiqueta, sendo que o comando necessita da definição de outro ponto logo abaixo para inserir o valor da área. Esse segundo ponto é guardado na variável “aarr_p2”, sendo definido por uma lista das coordenadas “x” do primeiro ponto, e a subtração de 0,3 u.d. da coordenada “y”.

Figura 40 – Algoritmo da função texto dentro do comando “aarr”.

```
;inserção dos textos:
(defun texto ()
  (command "osmode" "0")
  (setq comodo (getstring t "\n Nome de identificação: "))
  (setq aarr_p1 (getpoint "\n Selecione o ponto de inserção da etiqueta: "))
  (setq x_p1 (car aarr_p1))
  (setq y_p1 (car (cdr aarr_p1)))
  (setq aarr_p2 (list x_p1 (- y_p1 0.3) 0.0))
  (setq texto_area (strcat (rtos area 2 2) "m²"))
  (command "text" "j" "mc" aarr_p1 0.15 0 comodo )
  (command "text" "j" "mc" aarr_p2 0.15 0 texto_area )
  (command "_erase" entity_name "")
  (resetvar)
  (quest)
)
```

Fonte: Autoria Própria.

Figura 41 – a) Segunda interação do comando “aarr” na barra de comandos. b) Terceira interação do comando “aarr” na barra de comandos.



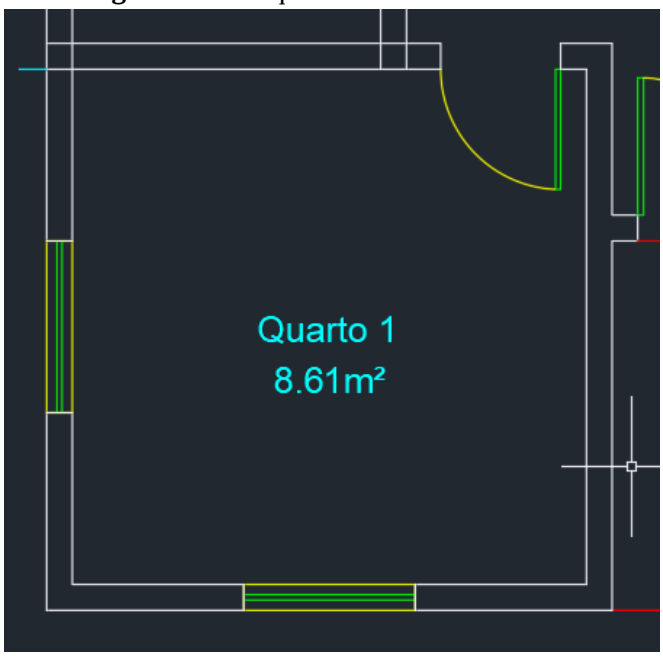
Fonte: Autoria Própria.

Na sequência do algoritmo, mais uma variável é configurada, “texto_area”, que será usada para colocar o valor da área calculado com o VBA dentro de uma *string*. Esse valor poderia ser inserido diretamente com o comando *Dtext* no formato que já se encontra, porém ele seria escrito com várias casas decimais e, para evitar isso, foi usada a função AutoLISP *rtos* para converter o valor da área em uma *string* contendo apenas duas casas decimais. Além do *rtos* a linha de expressão também possui um argumento *strcat*, que foi utilizada nessa parte da macro para juntar a área com mais uma *strings*, o sufixo “m²”, para indicar a unidade da área (Figura 40).

Ambos os textos são inseridos através do comando *Dtext*, com justificação *midle center*, para que os textos fiquem centralizados com os pontos definidos, o valor da altura foi determinado de acordo com a norma NBR 6492, que estabelece uma altura de 3,00 mm para esse tipo de texto. Porém o valor determinado pela norma é a altura do texto na impressão, por isso é necessário compensar o valor da escala de redução aplicada no desenho, para que o texto seja impresso na altura correta. Assim, foi pré-definido um valor de escala de 1:50 para o desenho, resultando em uma altura de 0.15 u.d. para os textos inseridos.

Por ser a última parte do programa, a função “texto” tem na sua penúltima linha a macro “resetvar”, para que as variáveis ambientes que foram modificadas sejam devolvidas para o programa. Por fim, o comando finaliza executando novamente a primeira função “quest”, criando um looping que permita ao usuário continuar utilizando o comando sem precisar reativá-lo (Figura 35). A inserção final da etiqueta aparece no desenho como mostrado na Figura 42.

Figura 42 – Etiqueta de cômodo inserida.



Fonte: Autoria Própria.

Com isso, quando o programa em AutoLISP desenvolvido é carregado dentro do AutoCAD, o usuário terá acesso a dois novos comandos de desenho. O primeiro, denominado “esq”, otimiza o processo de inserção dos blocos de esquadria, permitindo que elas sejam inseridas no tamanho que o desenhista desejar sem precisar determinar fatores de multiplicação de escala, podendo variar o tamanho dos objetos para cada nova inserção, sem a necessidade de modificar nenhum parâmetro na caixa de diálogo do comando *insert*. Além disso, o procedimento de procura de blocos na biblioteca do computador também é melhorado, pois as macros de inserção fazem uma busca automática durante a execução do programa.

A possibilidade de espelhar as esquadrias, principalmente no caso das portas, evita a necessidade de criar blocos duplicados, como portas que abrem para a esquerda e para a direita, o que deixa a biblioteca de blocos extensa e confusa. Portanto, a quantidade de blocos salvos na biblioteca interna do desenho é reduzida usando a rotina do AutoLISP e essa redução é vantajosa para a compactação do tamanho de arquivos “*.dwg*”.

O comando de inserção das etiquetas de cômodos, denominado “aarr”, aprimora a inserção desses elementos anotativos, evitando a necessidade de utilizar dois comandos separados para criar textos e calcular a área, compilando os dois passos em um único comando. Além disso, a opção de definição de área *boundary* permite determinar os limites de área de uma forma mais simples que o comando *Area* do AutoCAD, apesar desse tipo de seleção não ser precisa para todos os ambientes dos projetos arquitetônicos, ele é útil para a aplicação em algumas partes do desenho, como os ambientes “Área” e “Serviço” constante na figura do Anexo A.

As macros de criação e modificação de *layers*, usadas em ambos os comandos, auxilia na padronização dos objetos que são inseridos através das customizações, pois elas evitam que as novas entidades sejam adicionadas ao desenho com a *layer* corrente que está em uso pelo projetista. Sem essa modificação, cada vez que os comandos fossem ativados os objetos poderiam ser inseridos em camadas diferentes, ficando fora de padrão na impressão, e necessitando que o usuário modifique as *layers* de cada objeto manualmente. A padronização de camadas agrupa as novas entidades em um mesmo tipo, assim, se o usuário desejar modificar a *layer* das esquadrias e etiquetas, esse processo pode ser feito de forma simplificada selecionando todos os objetos de uma única vez.

5 CONSIDERAÇÕES FINAIS

Os resultados obtidos com a aplicação dos comandos customizados com o AutoLISP, “esq” e “aarr”, se mostraram satisfatórios, uma vez que as rotinas desenvolvidas alcançaram o objetivo de agilizar e simplificar dois processos repetitivos durante a execução do desenho de uma planta baixa, quais sejam: a inserção de esquadrias e a identificação de etiquetas de cômodo.

A implementação do comando “esq” possibilita ao usuário inserir blocos de esquadria no desenho de forma mais ágil, sem precisar fazer buscas na biblioteca para a adição dos objetos. A inserção das janelas e portas é feita apenas com a indicação de dois pontos na tela, o que é um procedimento mais simples em relação ao método convencional de inserção, através do comando *insert*. Com a obtenção dos dois pontos o comando adiciona a esquadria no tamanho correto do vão, e com a *layer* padronizada para receber as esquadrias.

O comando “aarr” obtém e insere o valor das áreas dos cômodos no momento da inserção dos textos de etiqueta, o que é um aprimoramento em relação à utilização dos comandos básicos do AutoCAD que necessitam que o cálculo de área seja feito separadamente através do comando *Area*. Assim como no comando “esq”, as etiquetas são inseridas com uma *layer* padronizada criada dentro do próprio comando customizado.

O dialeto AutoLISP possui uma sintaxe intuitiva e uma escrita simplificada o que facilita o aprendizado da linguagem. Apesar da facilidade, por ser uma linguagem ampla, o AutoLISP dispõe de vários caminhos para executar uma mesma função, o que dificultou o desenvolvimento de algumas partes do algoritmo. Por exemplo, não foi possível utilizar *while* nem *if* para fazer os comandos continuarem ativos até o usuário encerrá-los por conta própria. Esse problema teve que ser contornado transformando as partes que devem se repetir no programa em comandos internos, que são ativados ao longo da execução do algoritmo, criando *loopings* nas partes desejadas e evitando que o comando encerre antes do tempo.

Em trabalhos futuros podem ser feitos aprimoramentos no código desenvolvido nessa pesquisa. No comando de inserção de esquadrias (esq), por exemplo, pode-se incluir um método mais complexo de busca por blocos, que consiga verificar a existência de blocos de esquadria em todas as bibliotecas vinculadas ao AutoCAD. Também é possível acrescentar nesse comando uma macro que gere e salve automaticamente os blocos de esquadrias, pois nessa pesquisa foram usados objetos pré-existentes, além de melhorar o fluxo do programa permitindo que o usuário mude o tipo de opção de esquadria sem precisar finalizá-lo. Já no comando “aarr”, pode-se adaptar o algoritmo para que os textos sejam inseridos em tamanhos diversos de acordo com a escala que for escolhida pelo projetista, uma vez que nessa pesquisa foi determinada uma escala fixa de 1:50. Outra melhoria que pode ser feita nessa parte do algoritmo é incluir uma função, ou artifício do AutoLISP, que formate o separador de casas decimais no texto de dimensão da área para o padrão brasileiro, mudando o ponto, que é o padrão do AutoCAD, para uma vírgula e, também ajustando o número de casas decimais sempre para dois algarismos após a virgula.

REFERÊNCIAS

AMARAL, Renato Dias Calado do; PINA FILHO, Armando Carlos de. A Evolução do CAD e sua Aplicação em Projetos de Engenharia. In: SIMPÓSIO DE MECÂNICA COMPUTACIONAL, 9., 2010, São João Del-Rei. **Anais [...] .** São João Del-Rei: Associação Brasileira de Métodos Computacionais em Engenharia, 2010. p. 1-9.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 6492: Representação de Projetos de** Arquitetura. Rio de Janeiro. 1994.

AUTOCAD. **Computer Aided Design. Versão estudante.** [s.l.]: Autodesk, 2021. Disponível em < https://www.autodesk.com.br/products>. Acesso em 25 maio 2021.

AZEVEDO, Álvaro F. M. A Utilização de Software Comercial no Ensino Universitário. In: VI CONGRESSO NACIONAL DE MECÂNICA APLICADA E COMPUTACIONAL,6 Edição, 2000, Aveiro. **Painel integrado no VI Congresso Nacional de Mecânica Aplicada e Computacional.** Aveiro: Universidade de Aveiro, 2000. p. 1-6.

BERGIN, Thomas J.; GIBSON, Richard G. **History of Programming Languages**. New York: Association For Computing Machinery, 1996. 880 p.

BLATTMANN, Ursula. **Normas técnicas**: estudo sobre a recuperação e uso. 1994. 128 f. Dissertação (Mestrado) - Curso de Biblioteconomia, Pontifícia Universidade Católica de Campinas, Campinas, 1994.

CRUZ, Michele David da. **Desenho Técnico.** 1. ed. São Paulo: Érica, 2014.

FONSECA FILHO, Cléuzio. **História da computação**: o caminho do pensamento e da tecnologia. Porto Alegre: Edipucrs, 2007. 205 p.

JACOSKI, Claudio Alcides; BREDA, Luciano Rodrigo. CUSTOMIZAÇÃO EM AUTOLISP VISANDO **A COMUNICAÇÃO DE INTERFERÊNCIAS EM PROJETOS DE EDIFICAÇÕES.** In: X ENCONTRO NACIONAL DE TECNOLOGIA DO AMBIENTE CONSTRUÍDO, 10., 2004, São Paulo. Anais do X ENCONTRO NACIONAL DE TECNOLOGIA DO AMBIENTE CONSTRUÍDO. São Paulo: Associação Nacional de Tecnologia no Ambiente Construído, 2004. v. 1, p. 2-10.

KUBBA, Sam A. A. **Desenho Técnico para Construção**. Porto Alegre: Bookman, 2014.

LEAKE, James M. **Manual de Desenho Técnico para Engenharia:** Desenho, Modelagem e Visualização. 2. ed. Rio de Janeiro: LTC, 2015.

LIMA, Cláudia Campos Netto Alves de. Estudo Dirigido de AutoCAD. 1. ed. São Paulo: Érica, 2019

NETTO, Claudia Campos. **Desenho Técnico e Design de Interiores.** 1. ed. São Paulo: Érica, 2014.

OLIVEIRA, Adriano de. **Desenho Computadorizado: Técnicas para Projetos Arquitetônicos.** 1. ed. São Paulo: Érica, 2014.

OLIVEIRA, Adriano de; COSTA, Lourenco; BALDAM, Roquemar de Lima. **AutoCAD 2016**: utilizando totalmente. São Paulo: Érica, 2015. 560 p.

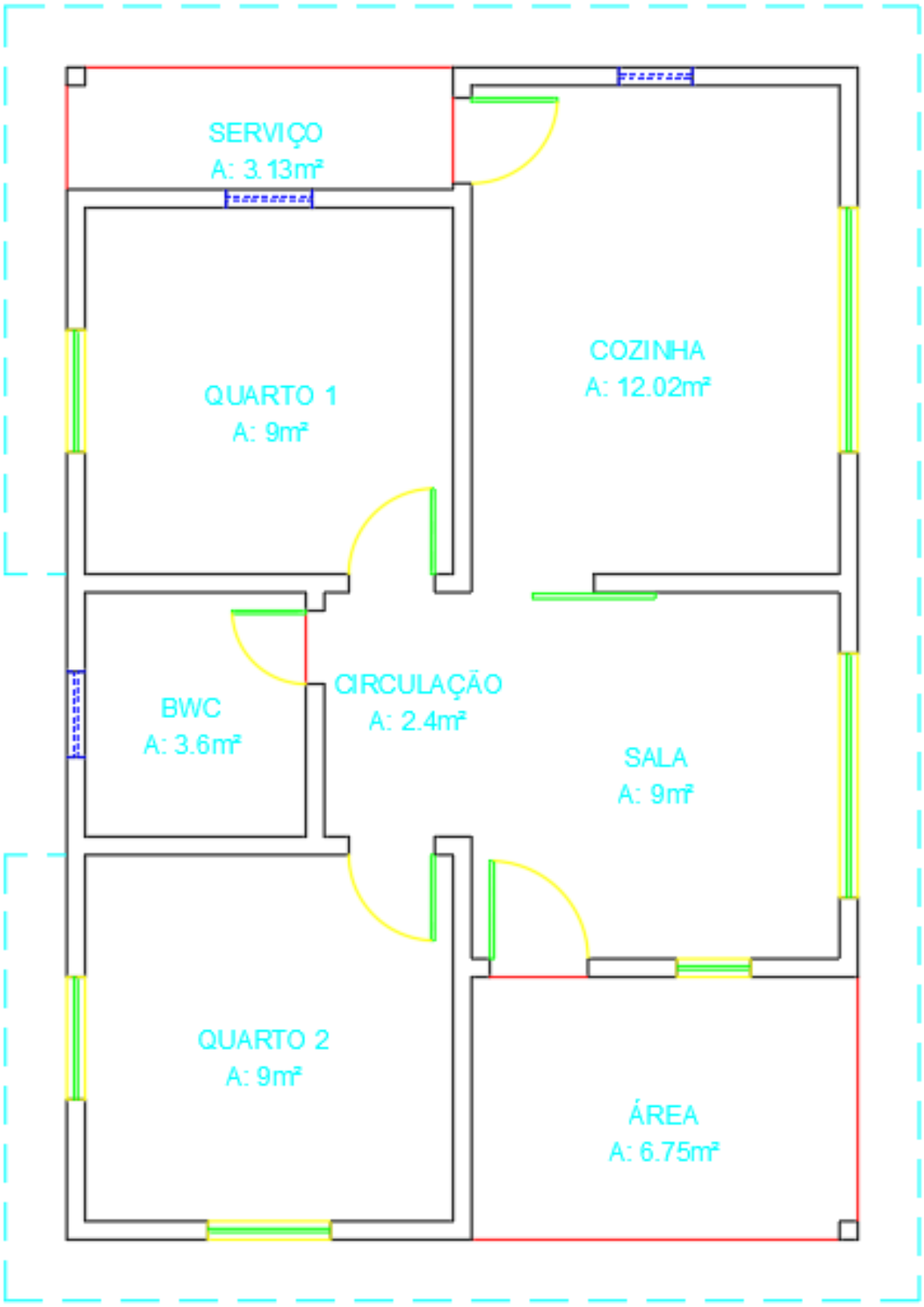
SARCAR, M. M. M.; RAO, K. Mallikarjuna; NARAYAN, K. Lalit. **Computer aided design and manufacturing**. PHI Learning Pvt. Ltd., 2008.

SATHLER, Nilson de Sousa. **Projeto Auxiliado por Computador – PAC**: Desenho Arquitetônico 2D. 1. ed. Mossoró: UFERSA, 2010.

SOUSA, Renato Cleber Mendes de. **OTIMIZAÇÃO DE PROCESSOS E TAREFAS USANDO A LINGUAGEM AUTOLISP EM FERRAMENTA CAD**. 2019. 53 f. TCC (Graduação) - Curso de Ciência da Computação, Centro de Ciências Exatas e Naturais, Universidade Federal Rural do Semi-Árido, Mossoró, 2019.

TAVARES, João Manuel R. S.; FONSECA, Joaquim Oliveira. **AutoLISP - I Introdução**. Porto: Faculdade de Engenharia da Universidade do Porto, 2011. 22 slides, color.

ANEXO A – PLANTA BAIXA COM BLOCOS E TEXTOS INSERIDOS ATRAVÉS DAS CUSTOMIZAÇÕES DESENVOLVIDAS.



Fonte: Autoria Própria.

MINISTÉRIO DA EDUCAÇÃO
Universidade Federal Rural do Semi-Árido – UFERSA
Centro de Engenharias - CE

ATA DE DEFESA DE TRABALHO DE CONCLUSÃO DE CURSO

Às dezenove horas do dia primeiro de junho de dois mil e vinte e um, excepcionalmente na sala de webconferência do google meet - meet.google.com/ocq-fddy-dgq - em razão das medidas de prevenção e controle da disseminação da covid-19, reuniu-se a Banca Examinadora de defesa de trabalho de conclusão de curso de autoria da estudante **RAFAELA DE OLIVEIRA DIAS**, aluna do curso de Bacharelado Interdisciplinar em Ciência e Tecnologia desta universidade, N° de matrícula **2017010745**, com o título ***"DESENVOLVIMENTO DE CUSTOMIZAÇÃO EM AUTOLISP APLICADA A MODELAGEM DE DESENHOS ARQUITETÔNICOS NO AUTOCAD"***. A Banca examinadora ficou constituída por três integrantes, a saber: Prof. Dr. MANOEL JANUÁRIO DA SILVA JÚNIOR, presidente da banca e orientador do Trabalho de Conclusão de Curso; Profa. Dra. DANIELLE SIMONE DA SILVA CASILLO e Profa. LUCIANA KLEIN DA SILVA DE MORAIS, como examinadoras. A defesa transcorreu conforme normas institucionais sem necessidade de registrar ocorrências. Concluída a defesa, procedeu-se o julgamento pelos integrantes da banca examinadora, em reunião fechada, ocasião em que a aluna foi considerada **APROVADA**. E para constar, eu, MANOEL JANUÁRIO DA SILVA JÚNIOR, lavrei a presente ata que, após lida e aprovada pelos integrantes da banca examinadora, será assinada por todos.

Mossoró, 01 de Junho de 2021.

Assinatura dos integrantes da Banca Examinadora.

MANOEL JANUARIO DA SILVA JUNIOR:03444846452
Assinado de forma digital por MANOEL JANUARIO DA SILVA JUNIOR:03444846452
Dados: 2021.06.04 19:37:46 -03'00'

Prof. Dr. MANOEL JANUÁRIO DA SILVA JÚNIOR – UFERSA
Presidente e orientador

Danielle Simone da Silva Casillo
Assinado de forma digital por Danielle Simone da Silva Casillo
Dados: 2021.06.04 15:33:10 -03'00'

Profa. Dra. DANIELLE SIMONE DA SILVA CASILLO – UFERSA
Primeiro Examinador

Luciana Klein da Silva de Morais

Profa. LUCIANA KLEIN DA SILVA DE MORAIS - CURSOS CONSTRUIR/AUTODESK
Segundo Examinador