UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

DESENVOLVIMENTO DE UMA INTERFACE MULTI-TOQUE PARA UMA MESA DE SIMULAÇÃO TÁTICA USANDO A ENGINE UNITY

TRABALHO DE GRADUAÇÃO

Gabriel Costa Backes

Santa Maria, RS, Brasil

2016

DESENVOLVIMENTO DE UMA INTERFACE MULTI-TOQUE PARA UMA MESA DE SIMULAÇÃO TÁTICA USANDO A ENGINE UNITY

Gabriel Costa Backes

Trabalho de Graduação apresentado ao Curso de Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de

Bacharel em Ciência da Computação

Orientador: Prof. Dr. Cesar Tadeu Pozzer

Trabalho de Graduação N. 415
Santa Maria, RS, Brasil

2016

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

DESENVOLVIMENTO DE UMA INTERFACE MULTI-TOQUE PARA UMA MESA DE SIMULAÇÃO TÁTICA USANDO A ENGINE UNITY

elaborado por
Gabriel Costa Backes

como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação

COMISSÃO EXAMINADORA:

Cesar Tadeu Pozzer, Dr.
(Presidente/Orientador)

Lisandra Manzoni Fontoura, Profª. Drª. (UFSM)

Mateus Beck Rutzig, Prof. Dr. (UFSM)

Santa Maria, 12 de Dezembro de 2016.

RESUMO

Trabalho de Graduação
Curso de Ciência da Computação
Universidade Federal de Santa Maria

DESENVOLVIMENTO DE UMA INTERFACE MULTI-TOQUE PARA UMA MESA DE SIMULAÇÃO TÁTICA USANDO A ENGINE UNITY

AUTOR: GABRIEL COSTA BACKES
ORIENTADOR: CESAR TADEU POZZER
Local da Defesa e Data: Santa Maria, 12 de Dezembro de 2016.

Interfaces multi-toque trouxeram um novo paradigma para a interação humano-computador, dando ao usuário a sensação de imersão ao ambiente virtual por meio de interações mais próximas ao comportamento humano. O presente trabalho faz parte do projeto Sistema Integrado de Simulação Astros (SIS-ASTROS) e apresenta o desenvolvimento de uma interface sensível ao toque, modular e intuitiva, para a mesa de simulador virtual tático utilizando o motor Unity. Esta mesa é um módulo que permite a interação do usuário com o sistema, por meio de diversas ações, para controlar uma simulação militar. Entre outras características, essa interface disponibiliza menus ao usuário, tanto para a realização de ações, quanto para a visualização das informações sobre a simulação. Estes menus podem ser acessados pelo usuário em qualquer posição da extensão do dispositivo. Para tanto, foram estudados métodos e *layouts* de menus que possibilitassem essa prática. A adaptação do paradigma *Model-View-Controller* mostrou-se eficiente para inserção de novos menus para a interface e a conexão destes com outras unidades do sistema.

Palavras-chave: Interface Multi-Toque. Interface Natural de Usuário. Computação em Superfície. Layout Radial.

ABSTRACT

Undergraduate Final Work
Undergraduate Program in Computer Science
Federal University of Santa Maria

DEVELOPMENT OF A MULTITOUCH INTERFACE FOR A TACTICAL SIMULATION TABLE USING THE ENGINE UNITY

AUTHOR: GABRIEL COSTA BACKES
ADVISOR: CESAR TADEU POZZER
Defense Place and Date: Santa Maria, December 12th, 2016.

Multi-touch interfaces brought a new paradigm for human-computer interaction, giving the user a sense of immersion to the virtual environment through closer human behavior interactions. This work is part of the project Sistema Integrado de Simulação Astros(SisAstros), and presents the development of a touch interface, modular and intuitive, for a tabletop virtual tactical simulator using the Unity engine. This tabletop is a module that enables interaction between the student and the system, through many commands type, to control a military simulation. Among other characteristics, this interface provides menus to the user, both for performing actions and for visualizing information about the simulation. These menus can be accessed by the user at any position in the device extension. For this, methods and menu layouts that would allow this practice were studied. The adaptation of the Model-View-Controller paradigm proved to be efficient for inserting new menus into the interface and connecting them to other system units.

Keywords: Multi-Touch. Natural User Interface. Surface Computing. Radial Layout.

LISTA DE FIGURAS

Figura 2.1 – Interfaces de Usuário 11
Figura 2.2 – Minority Report - A Nova Lei(2002) 12
Figura 2.3 – Exemplo de Pie Menu 16
Figura 2.4 – Exemplo de Marking Menu 17
Figura 2.5 – Compact Radial Layout 20
Figura 2.6 – Ciclo de execução do paradigma *Model-View-Controller* 21
Figura 3.1 – Estrutura física do simulador 23
Figura 3.2 – Conjunto de gestos para interação do usuário 24
Figura 3.3 – Evento *OnClick* de um botão na Unity 25
Figura 3.4 – Exemplo de Menu Radial 27
Figura 3.5 – Exemplo de Menu de *Swipe* 28
Figura 4.1 – Diagrama de classes da interface 30
Figura 4.2 – Divisão da tela 34
Figura 4.3 – IconsManager 35
Figura 4.4 – IconListEditor 36
Figura 4.5 – Botão radial no *Inspector* da Unity 37
Figura 4.6 – Diagrama de atividade para criação do Menu Radial 38
Figura 4.7 – Diagrama de processo da interação com o Menu Radial 39
Figura 5.1 – *SwipeInput* no *Inspector* 41
Figura 5.2 – Exemplo do *layout* do Menu Radial desenvolvido 42

LISTA DE ABREVIATURAS E SIGLAS

CLI	*Interface de Linha de Comando*
GUI	*Interface Gráfica de Usuário*
NUI	*Interface Natural de Usuário*
JSON	*JavaScript Object Notation*
HRM	*Menu Radial Hierárquico*
PM	*Pie Menu*
OPM	*Overlapping Pie Menus*
NPM	*Non-Overlapping Pie Menus*
WM	*Wave Menu*
IWM	*Inverse Wave Menu*
CRL	*Compact Radial Layout*
MVC	*Model-View-Controller*

SUMÁRIO

1 INTRODUÇÃO 9
1.1 Objetivos 10
1.1.1 Objetivo Geral 10
1.1.2 Objetivos Específicos 10
1.1.3 Estrutura do Trabalho 10
2 REFERENCIAL TEÓRICO 11
2.1 Interface Natural de Usuário 11
2.1.1 Tela Sensível ao Toque 13
2.1.2 Multi-Toque 14
2.1.3 Gestos 14
2.2 *Layouts* Radiais 15
2.2.1 *Layout* Mono Level 16
2.2.2 Hierarquia com Mono Level 16
2.2.3 Menus Radiais Hierárquicos 17
2.2.4 *Layout* Radial Compacto 18
2.3 Paradigma *Model View Controller* 20
3 ARQUITETURA 22
3.1 Simulador Virtual Tático 22
3.2 Proposta de Interface 23
3.2.1 Gerenciador de eventos da interface 25
3.2.2 Menu Radial Hierárquico 26
3.2.3 Menu de *Swipe* 27
4 IMPLEMENTAÇÃO 29
4.1 Gestos e funcionalidades 30
4.1.1 Toque duplo 31
4.1.2 Toque longo 31
4.1.3 Mover 32
4.1.4 Rotacionar 32
4.1.5 Escalar 33
4.1.6 Sentido de visualização 33
4.2 Gerenciador de Eventos da Interface 34
4.3 *IconsManager* 35
4.4 Menu Radial 36
4.4.1 Tempo de editor 37
4.4.2 Tempo de execução 38
4.5 Menu de *Swipe* 39
5 RESULTADOS 40
6 CONCLUSÃO 43
REFERÊNCIAS 44

1 INTRODUÇÃO

A tela sensível ao toque (*touchscreen* em inglês) é uma tecnologia que revolucionou a forma de como são realizadas as interações em diversos aparelhos eletrônicos. Por meio dela é possível executar tarefas, antes mais complexas, com um simples arrastar de dedos. Com o surgimento da tecnologia multi-toque, a qual possibilitou detectar mais de um ponto em contato com uma superfície, surgiram novas funcionalidades para o tela sensível ao toque.

Plataformas de mídia e dispositivos que permitem uma entrada de toque de dedo do usuário são cada vez mais onipresentes. Assim, o *touchscreen* está no centro das pesquisas de muitas empresas (SAFFER, 2008). Porém, a definição dos estilos de toque que estão disponíveis em cada dispositivo é específico de cada uma. A capacidade do hardware de detectar não somente o toque, mas a ação realizada, como a movimentação de dedos, está presente em quase todos os dispositivos atualmente, porém a ação que será realizada pelo software difere de um dispositivo para outro.

Este trabalho faz parte do projeto SIS-ASTROS, por meio do qual está sendo desenvolvido um simulador virtual tático para a execução do exercício de Reconhecimento, Escolha e Ocupação de Posição (REOP). Neste simulador, há uma mesa tática, no tamanho de 84 polegadas, que apresenta uma carta 2d, onde são inseridos comandos e ações para realizar a simulação. Estes comandos devem ser inseridos facilmente através de menus, que podem ser acessados, de forma intuitiva, através do multi-toque. Por ser um dispositivo diferente dos demais que utilizam uma interface sensível ao toque, criou-se uma implementação de uma solução que atenda ao desenvolvimento da mesa tática, proporcionando realizar a interação com o simulador.

Neste projeto, podemos citar alguns diferenciais da mesa tática para os demais dispositivos sensíveis ao toque. O primeiro deles é o tamanho da tela onde é utilizada a interface multi-toque. Diferente de um smartphone ou tablet, a tela tem a dimensão de 84 polegadas, o que necessitou um estudo sobre como ocorre a disposição dos menus na tela. Outro ponto é a diversidade de ações que podem ser realizadas na mesa, tanto antes da simulação ser iniciada, momento em que é realizada a configuração do simulador, quanto durante a própria simulação. E, por fim, a complexidade das ações que podem ser realizadas pelos usuários. Com isso foi necessário um estudo sobre as possíveis maneiras de acessar essas ações de forma rápida e prática.

1.1 Objetivos

1.1.1 Objetivo Geral

Este trabalho tem por objetivo o desenvolvimento de um sistema de interação multitoque para uma mesa de simulação tática fazendo o uso do motor Unity. Esta mesa poderá ser utilizada por vários usuários, estes poderão estar dispersos ao redor mesa. Assim, todas as funcionalidades devem ser acessíveis independentemente da posição do usuário.

1.1.2 Objetivos Específicos

Esta pesquisa tem os seguintes objetivos específicos:

- Implementar um menu genérico;
- Implementar um gerenciador das interações com a interface;
- Implementar uma interface que possibilite ao usuário interagir independente de sua posição e que se adapte a diferentes resoluções;
- Implementar uma interface intuitiva para novos usuários e eficiente para usuários experientes, possibilitando a evolução do usuário na sua utilização do simulador.

1.1.3 Estrutura do Trabalho

Este trabalho estrutura-se em seis capítulos, apresentando-se no capítulo dois o referencial teórico, abordando as Interfaces Naturais de Usuário, *Layouts* Radiais e o paradigma *Model-View-Controller*. O capítulo três caracteriza o simulador virtual tático, os conceitos e adaptações dos menus e gestos apresentados no referencial teórico. O capítulo quatro descreve a implementação das unidades citadas no capítulo três. No capítulo cinco são avaliados os resultados dos testes, problemas e suas soluções. E por fim, o capítulo seis apresenta a conclusão.

2 REFERENCIAL TEÓRICO

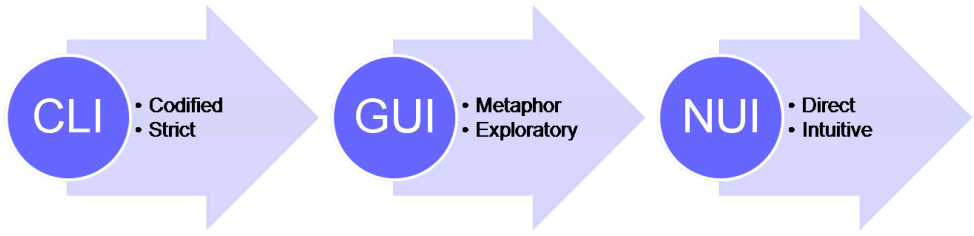
A contextualização do presente trabalho se dá a partir da discussão de pontos fundamentais para a implementação de uma interface sensível ao toque. Dessa forma, serão apresentados alguns conceitos de interfaces naturais de usuário, *layouts* radiais para menus e um paradigma do funcionamento e comunicação da interface com o sistema da aplicação.

2.1 Interface Natural de Usuário

Interface Natural de Usuário (NUI - *Natural User Interface*) é uma linguagem comum usada por designers e desenvolvedores de interfaces de computador para se referir a interface de usuário que é efetivamente invisível, ou torna-se invisível com a evolução do usuário através do sucessivo uso da interface (RAUTERBERG; MAUCH; STEBLER, 1996). Em outras palavras, NUI é um sistema de interação humano-computador que o usuário opera através de ações intuitivas relacionadas a ações naturais (comportamento humano).

As interfaces de usuário (Figura 2.1) surgiram com as linhas de comando (CLIs), onde a interação era realizada através de transações de solicitação-resposta, com as solicitações sendo expressadas por comandos textuais em um vocabulário especializado. Evoluindo para interfaces gráficas (GUIs) onde o usuário tinha uma resposta visual, porém utilizava periféricos para interagir com o sistema. Após, surgiram as interfaces naturais de usuário, tornando mais próximo o usuário do sistema e proporcionando uma capacidade de interação sem precedentes.

Figura 2.1: Interfaces de Usuário



A maioria das interfaces de computador ainda utilizam periféricos, como mouse e te-

clado, para processar a interação do usuário com o sistema. Em NUI, o usuário é capaz de interagir diretamente com a interface e, através do seu *design*, viabiliza a fácil aprendizagem. A medida que o usuário progride e evolui no uso da interface, torna-se possível alcançar interações mais complexas de maneira autodidata.

Interfaces naturais de usuário, como o multi-toque e a computação em superfície (*surface computing* em inglês), foi dita como a próxima principal evolução na área da computação e interfaces de usuário (SEOW et al., 2009). Existem diversas obras fictícias que utilizam NUI, *Minority Report* (Figura 2.2) é uma delas. Assim como nesta obra, o objetivo do uso de NUI é fazer com que o usuário seja capaz de realizar movimentos relativamente naturais ou intuitivos para interagir com o sistema, sentindo-se natural ao ambiente.

Figura 2.2: Minority Report - A Nova Lei(2002)

Assim que as NUIs surgiram, designers e engenheiros de interação definiram um *design* de interações – novas maneiras de interagir com os dispositivos, meio ambiente e até mesmo uns com os outros. Este *design* viria a ser utilizado por todas as aplicações que utilizassem interfaces sensíveis ao toque (SAFFER, 2008). Entretanto, SAFFER (2014) elencou 12 princípios para o desenvolvimento de interfaces naturais de usuário :

- Desenvolver para dedos – Não para cursores, os alvos do toque precisam ser maiores do que para desktops;
- Lembrar-se de fisiologia e cinesiologia – Não fazer com que os usuário façam tarefas repetitivas e cansativas;
- Sem braço de gorila – Os seres humanos não foram feitos para fazer muitas tarefas com as mãos para cima, em frente de seus corpos, por longos períodos de tempo;
- Cobertura da tela – Evitar colocar elementos essenciais abaixo de um controle, de forma que possa ser encoberto pela própria mão;

- Conhecer a tecnologia – O tipo de tela sensível ao toque, sensor ou câmera determina o tipo de gestos que poderão ser projetados para a interação;
- Quanto mais desafiador o gesto, menos pessoas serão capazes de realizá-lo;
- Ativar ações quando o usuário remover o dedo, não enquanto toca a tela;
- Reconhecimento – Utilizar gestos simples e intuitivos para usuário iniciais;
- Evitar ativação de ações não intencionais – A variedade de movimentos realizados pelo usuário pode acidentalmente acionar interações indesejadas;
- Gestos e teclas de comando – Fornecer maneiras fáceis de acessar as funcionalidades, mas também formas mais avançadas de gestos aprendidos como atalhos;
- Variedade de requisitos – Variedade de maneiras de realizar um mesmo gesto;
- Determinar a complexidade do gesto de acordo com a complexidade e frequência da tarefa.

2.1.1 Tela Sensível ao Toque

Uma tela sensível ao toque é capaz de detectar um toque e sua localização dentro da extensão do dispositivo. Um usuário pode interagir dando uma entrada ao sistema de processamento de informação através de um simples toque na tela com determinado gesto (WOBBROCK; MORRIS; WILSON, 2009). A interação do usuário pode ser feita diretamente através do toque com o dedo ou mão, mas também podem ser utilizados objetos, como uma caneta, dispensando o uso de periféricos como mouse ou teclado.

Interfaces sensíveis ao toque permitem aos usuários interagirem diretamente com aplicações de maneira mais intuitiva, se comparadas ao uso de um cursor. Ao invés de mover um cursor para selecionar um arquivo e abri-lo, o usuário pode usar uma representação gráfica do arquivo para realizar sua abertura. Outra aplicação do *touch* pode ser a seleção de vários objetos da aplicação, para isso o usuário pode simplesmente realizar um laço formando uma área que conterá os objetos desejados.

2.1.2 Multi-Toque

Multi-toque é uma tecnologia que permite que telas sensíveis ao toque consigam detectar a presença de mais de um ponto em contato com sua superfície (ENCYCLOPEDIA, 2016). Com a presença de mais de um ponto é possível a criação de novas funcionalidades, gerando um novo paradigma de possibilidades de interação do usuário com os dispositivos.

Um trabalho recente na área do multi-toque que fornece uma visão sobre a natureza deste tema foi apresentado por PELTONEN et al. (2008). O estudo analisou como a disponibilidade pública é conseguida através da aprendizagem social e negociação, porque a interação se torna performativa e, finalmente, como a exibição reestruturou o espaço público. Finalmente, o estudo mostra como o recurso multi-toque, interação baseada em gestos, e o tamanho de exibição física contribuiu diferencialmente a estas utilizações.

Com a possibilidade da interação com duas mãos, o multi-toque provê um escopo para interações que são analogamente mais próximas as interações físicas do que as clássicas interações de janela. O *design* natural e intuitivo dos gestões é um difícil problema já que o programador não tem conhecimento de como os novos usuários irão abordar ao interagir com as interfaces multi-toque e quais gestos tentarão utilizar para realizar tarefas simples.

Uma pesquisa realizada por NORTH et al. (2009) foi inspirada na necessidade de informações para manipulação de objetos. Nela, usuários tiveram de executar tarefas para a classificação de objetos em uma mesa física, em um *tabletop display* e em um computador com um mouse. Foram produzidos um vocabulário de técnicas de manipulação que os usuários aplicaram no mundo físico e comparadas com o conjunto de gestos que os usuários tentaram utilizar na superfície *touch* sem nenhum treinamento. Após o estudo, verificou-se que os usuários que começaram o experimento utilizando o modelo físico, terminaram as tarefas mais rapidamente, quando eles passaram a utilizar a superfície *touch*, do que aqueles que começaram com o mouse.

2.1.3 Gestos

Gestos *multitouch* são a combinação do movimento de mais de um toque em uma superfície *touch* que, após ser processado por um software específico, gera um comando para o movimento realizado. Muitos protótipos de *surface computing* têm empregado gestos definidos por designers de sistema (WOBBROCK; MORRIS; WILSON, 2009). Embora esses gestos sejam apropriados para interações iniciais, não necessariamente irão refletir o comportamento do

usuário.

Para tentar resolver esse problema, WOBBROCK; MORRIS; WILSON (2009) apresentam uma abordagem para o *design* de gesto de uma mesa, que se baseia na elicitação de gestos por usuários não técnicos. No total, foram registrados, avaliados e emparelhados com a técnica de "Pensamento em voz alta", 1080 gestos de 20 participantes para 27 comandos requisitados com uma e duas mãos. Os resultados indicam que: usuários raramente se importam com o número de dedos que utilizam; uma mão é preferível do que duas; a interface GUI influencia diretamente nos modelos mentais dos usuários; alguns comandos provocam pouca concordância gestual, sugestionando o uso de widgets na tela.

2.2 *Layouts* Radiais

Menus são os controles primários de uma interface gráfica de usuário, provendo informações sobre quais comandos estão disponíveis e a maneira de invocá-los. A medida em que o software de uma aplicação cresce em termos de funcionalidades que ele oferece, seus menus crescem em tamanho. Sendo assim, podem ser representados hierarquicamente, entretanto, tornam-se mais difíceis em termos de usabilidade (SAMP; DECKER, 2010).

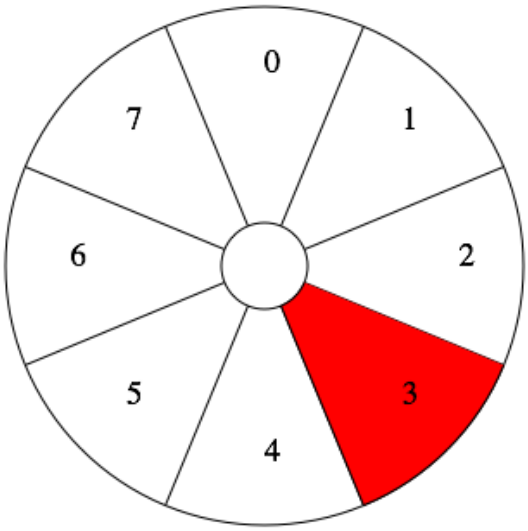
No contexto de exibição de menus, *layouts* radiais foram, inicialmente, utilizados para suportar menus pequenos e que continham um único nível. O suporte para hierarquias foi primeiramente feito através do uso de uma série de menus mono-level e, somente após esses, foi desenvolvido explicitamente um menu hierárquico que utilizava um *layout* radial.

Usuários desfamiliarizados com o conteúdo do menu e a localização dos comandos tendem a procurar-los visualmente. Neste caso, o tempo total de seleção de um comando é composto pelo tempo de busca visual e do tempo de apontar o botão (COCKBURN; GUTWIN; GREENBERG, 2007). Entretanto, usuários desenvolvem rapidamente um conhecimento espacial sobre a localização dos itens do menu, caso o *layout* do menu seja estável (KAPTELININ, 1993), (SOMBERG, 1987). Consequentemente, a busca visual decresce à medida que o usuário se torna experiente e o tempo de seleção geral depende principalmente no tempo de decisão e apontamento (COCKBURN; GUTWIN; GREENBERG, 2007).

2.2.1 *Layout* Mono Level

Layouts radiais foram reconhecidos, primeiramente, como uma técnica para diminuir o tempo de apontamento no *design* de um menu. Conhecidos também como *Pie Menu* (PM) (Figura 2.3), envolvem em disponibilizar itens ao longo de uma circunferência de um círculo com distâncias equivalentes de seu centro (CALLAHAN et al., 1988). O*design* suporta um único nível de itens e é capaz de acomodar um pequeno número de itens efetivamente, mostrando vantagem sobre os tradicionais menus lineares. Além disso, PMs consomem uma grande área da tela e são maiores que menus lineares tanto em largura quanto altura. Em um experimento controlado, realizado por CALLAHAN et al. (1988), PMs foram comparados com menus lineares e os resultados mostraram que menus contendo 8 itens tem um tempo de seleção 15 % mais rápido.

Figura 2.3: Exemplo de Pie Menu



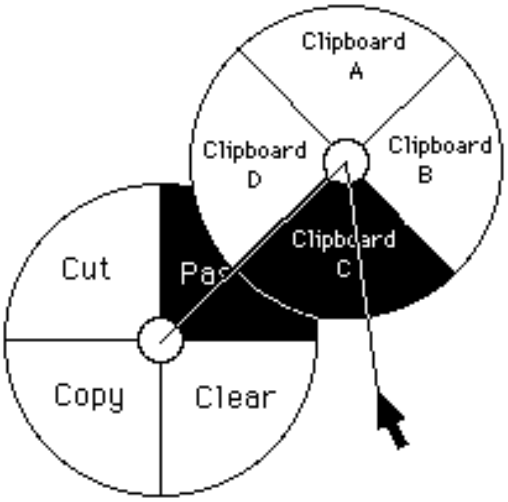
2.2.2 Hierarquia com Mono Level

O suporte a menus hierárquicos utilizando *layouts* radiais são apresentados no contexto de menus de marcação (Figura 2.4) [(KURTENBACH; BUXTON, 1993), (KURTENBACH; BUXTON, 1994), (ZHAO; BALAKRISHNAN, 2004)]. Menus de marcação são uma variante de PMs. Esta abordagem permite ao usuário de realizar a seleção do menu ou selecionar um item através do *PopUp* de um outro PM, ou ainda, realizando um movimento reto na direção do item desejado. Assim como em menus lineares, menus de marcação podem suportar menus

hierárquicos. Para sua interação, o usuário realiza um "Zig-Zag" para selecionar múltiplos níveis de submenus (ZHAO; BALAKRISHNAN, 2004). Cada nível é representado como um PM separado. Existem duas abordagens para o *layout* dos PM dos demais níveis:

- *Overlapping Pie Menus* (OPM): Se o subnível sobrepõem o menu anterior, pelo fato da distância entre eles ser menor do que seus diâmetros, fazendo com que níveis anteriores fiquem inacessíveis.
- *NonOverlapping Pie Menus* (NPM): O inverso do OPM, onde os subMenus são posicionados de maneira a evitar a sobreposição com níveis anteriores. Esta abordagem permite a visibilidade e acessibilidade de todos menus.

Figura 2.4: Exemplo de Marking Menu



2.2.3 Menus Radiais Hierárquicos

O *design* de um *layout* radial para a representação de um menu hierárquico é apresentado por BAILLY; LECOLINET; NIGAY (2007). O *design* inicial é conhecido como *wave menu* (WM), uma variante do menu de marcação com multi níveis. Seu objetivo é melhorar o desempenho de usuários iniciais ao mesmo tempo que preserva sua eficiência para com usuários experientes. Sua principal desvantagem é o espaço limitado de itens em seu primeiro nível e a falta de constância de posição para itens já selecionados e como eles se movem em níveis externos. Um *design* alternativo é a representação inversa do WM (IWM), o posicionamento de níveis subsequentes no centro do menu e a realocação de níveis anteriores em posições externas.

O experimento realizado por BAILLY; LECOLINET; NIGAY (2007) comparou quatro *designs* radiais: WM, IWM, OPM, NPM. Os menus continham 2 níveis com 4, 6 ou 8 itens cada, estes contendo 16 a 64 itens no total. Os resultados mostraram que NPM e WM são significativamente mais rápidos (18 %) e mais precisos que os OPM. WMs tem a mesma performance, com usuários iniciais que NPM. Por sempre mostrar os menus pais e permitir a interação com todos os itens mostrados, WM também oferece a pré visualização de itens de submenus para a busca contínua e o caminho percorrido. Essas duas características aumentam a aprendizagem sobre os menus por facilitar a exploração e a aquisição do conhecimento sobre o menu. Mesmo que os menus pais sejam mostrados, WM necessitam de menos espaço de tela do que NPM e podem ser utilizados de modo degradado, com espaços restritos de tela (BAILLY; LECOLINET; NIGAY, 2007).

2.2.4 *Layout* Radial Compacto

Um modelo de *layout*, chamado de *Compact Radial Layout Design* (CRL), é proposto por SAMP; DECKER (2010). Levando em consideração seu objetivo, que é a criação de um menu que seja conveniente para usuários iniciais e rápido para experientes, foram levantados os princípios para seu desenvolvimento:

- Quantidade e consumo de espaço – Aplicações atuais contém centenas ou até milhares de comandos, tipicamente organizados em menus de cascatas com profundidade de 2, 3 ou mais níveis. Um requisito é acomodar um grande número de comandos através do uso de pelo menos 3 níveis;

- Rápido acesso a usuários experientes – Possibilitar que usuários experientes naveguem rapidamente pelo menu, não somente no primeiro nível, mas em todos eles e, também, entre eles;

- Facilidade de exploração para usuários iniciais – Mostrar o contexto completo da sequência de seleção, e um acesso fácil ao conteúdo de níveis anteriores do menu. Também é necessário possibilitar a rápida visualização dos itens em cada nível do menu;

- Transição suave entre iniciante para experiente – O *design* precisa facilitar a aprendizagem de maneira a acelerar a transição do usuário.

Os níveis da hierarquia do menu são representados por anéis concêntricos. O primeiro

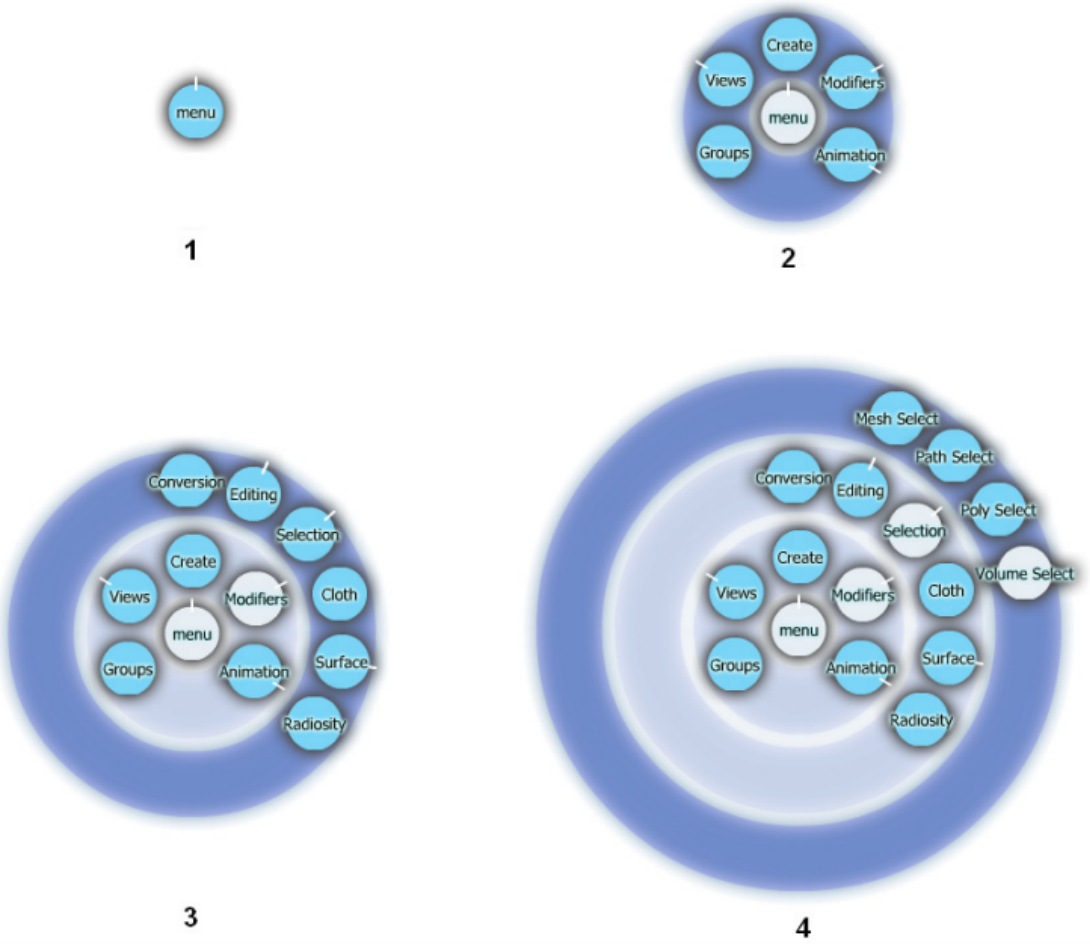
anel corresponde ao primeiro nível da hierarquia do menu. Anéis subsequentes correspondem aos próximos níveis da hierarquia do menu e representam um submenu do item selecionado no anel anterior. Cada anel subsequente é representado por uma circunferência maior que o anel anterior, portanto é capaz de acomodar mais itens. Todos os itens anteriormente selecionados são marcados com um contorno.

A interação com o CRL menu funciona da seguinte maneira: o usuário clica no item central que representa o CRL Menu (Figura 2.5.1). O primeiro anel aparece em torno do item central (Figura 2.5.2). O usuário então aponta o item desejado e o seleciona com um clique do mouse. Se o item não tiver um submenu, o comando é invocado e todos os aneis desaparecem. Se o item contém um submenu, o próximo anel mais externo é mostrado (Figura 2.5.3). Similarmente, qualquer anel subsequente aparecerá caso o usuário selecione um item localizado no anel mais externo. O usuário pode selecionar qualquer item visível. Se um item for selecionado em um anel mais interno, todos os outros anéis mais externos desaparecem e um novo anel mais externo aparece. Independente da interação do usuário, a localização dos itens dos menus e os anéis nunca mudam. Esta abordagem facilita a aprendizagem de usuário novatos [(COCKBURN; GUTWIN; GREENBERG, 2007), (KAPTELININ, 1993), (SOMBERG, 1987)].

Por meio da implementação do CRL de SAMP; DECKER (2010) foi visto que o espaço limitado no primeiro anel, a localização do menu e o tamanho dos textos ocasionaram problemas na exibição e utilização do menu. Sendo recomendável que, pelo fato de que o CRL possa se expandir em todas as direções, ele seja utilizado no centro da tela. Assim como, pela proximidade entre os itens, há a possibilidade de sobreposição de textos. Como alternativa, SAMP; DECKER (2010) sugere que a distância entre os itens ou o tamanho dos itens sejam aumentados, aumentando assim a capacidade espacial.

Um experimento realizado por SAMP; DECKER (2010) para testar a usabilidade do CRL utilizou dois tamanhos de menus, três níveis, um linear e três variações de menus radiais. Seu objetivo foi a investigação de como *layouts* radiais se comparam a *layouts* lineares em termos de busca visual e performance de apontamento do mouse. Os resultados obtidos mostraram que a busca visual é mais rápida em *layouts* lineares. A vantagem é maior para menus pequenos (31 %) e menor para menus maiores (14 %). O apontamento é mais rápido em *layout* radiais. A vantagem é similar tanto para menus pequenos ou grandes. Resultados também demonstraram que *layout* radiais são benéficos para o *design* de disposição de menus hierárquicos.

Figura 2.5: Compact Radial Layout



2.3 Paradigma *Model View Controller*

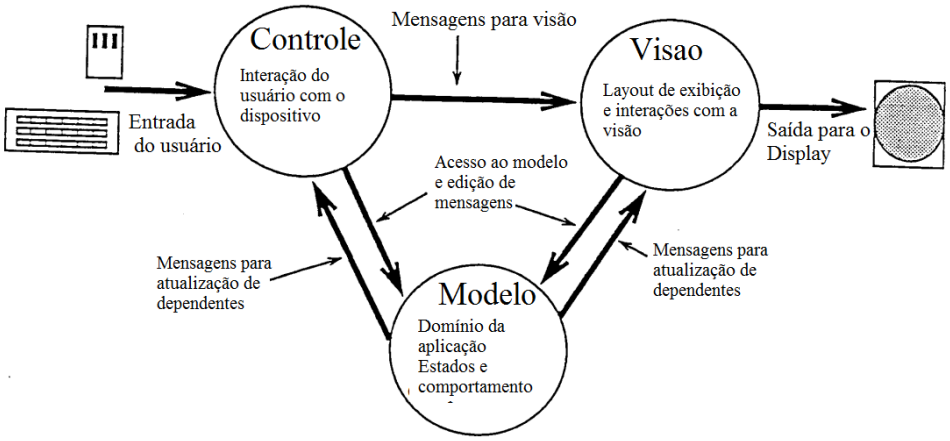
Na construção de aplicações intuitivas, a modularidade de componentes traz um grande benefício. Unidades funcionais isoladas uma das outras garantem um melhor entendimento e modificação de cada unidade por parte dos desenvolvedores da aplicação, que não precisam ter o conhecimento absoluto sobre outras unidade (KRASNER; POPE et al., 1988). Experimentos realizados mostram que uma forma particular de modularidade (separação de três vias), facilita a vida do designer. Esta implica na separação de: 1) as partes que representam o modelo do domínio de aplicação subjacente a partir de 2) a forma como o modelo é apresentado ao utilizador e de 3) a forma como o utilizador interage com ele.

A programação *Model-View-Controller* é a aplicação dessa fatoração de três vias, no qual os objetos de diferentes classes assumem o controle das operações relacionadas ao domínio

da aplicação (o modelo), à exibição do estado da aplicação (a visão), e à interação do usuário com o modelo e a visão (o controle) (BURBECK, 1992). Mais detalhadamente, o modelo de uma aplicação é a implementação de sua estrutura central, os componentes do sistema que realmente executam o trabalho da aplicação. Visão lida com toda parte gráfica, requisita dados de seu modelo e exibe essas informações. Controles contém a interface entre o seu modelo associado e a visão. São usados para enviar mensagens ao modelo e prover a interatividade de dispositivos da interface de usuário.

O ciclo de execução do funcionamento do MVC pode ser descrito como: o usuário realiza uma entrada, ativando assim, o controlador, que notifica o modelo para se atualizar de acordo com a entrada. O modelo executa a operação prescrita, possivelmente, alterando seu estado e comunicando seus dependentes (visões e controladores) que foi alterado, informando-os da natureza de sua alteração. As visões, então, verificam o novo estado do modelo e atualizam suas exibições caso necessário. Controladores podem alterar seus métodos de interação dependendo do novo estado do modelo. O ciclo pode ser visto na Figura 2.6.

Figura 2.6: Ciclo de execução do paradigma *Model-View-Controller*



3 ARQUITETURA

O foco deste trabalho está no desenvolvimento da interface sensível ao toque para a mesa tática, na qual ocorre a interação do aluno com o simulador. A partir de dois pontos principais foram projetados o *design* e a arquitetura da interface de usuário:

- Projeto em desenvolvimento – Havendo alterações e atualizações constantemente, buscou-se desenvolver um gerenciador de menus. Por meio dele é possível criar novos ou atualizar menus já existentes de maneira a facilitar a utilização por parte dos programadores.
- Utilização de uma NUI – Havendo uma maior aproximação entre o usuário e a interface do que em sistemas baseados no uso periféricos, foram desenvolvidas funcionalidades que possibilitam o usuário interagir melhor com o sistema, podendo arrastar, rotacionar ou escalar os menus.

3.1 Simulador Virtual Tático

O Simulador Virtual Tático faz parte do projeto Sistema Integrado de Simulação Astros (Sis-Atros), que é composto por 3 módulos: o posto de instrutor; a mesa tática e o visualizador 3D. Para realizar a comunicação entre esses 3 módulos, existe um servidor central que realiza a troca de mensagens. Por meio deste é possível realizar a simulação de reconhecimento, escolha e ocupação de posição (REOP) (Figura 3.1).

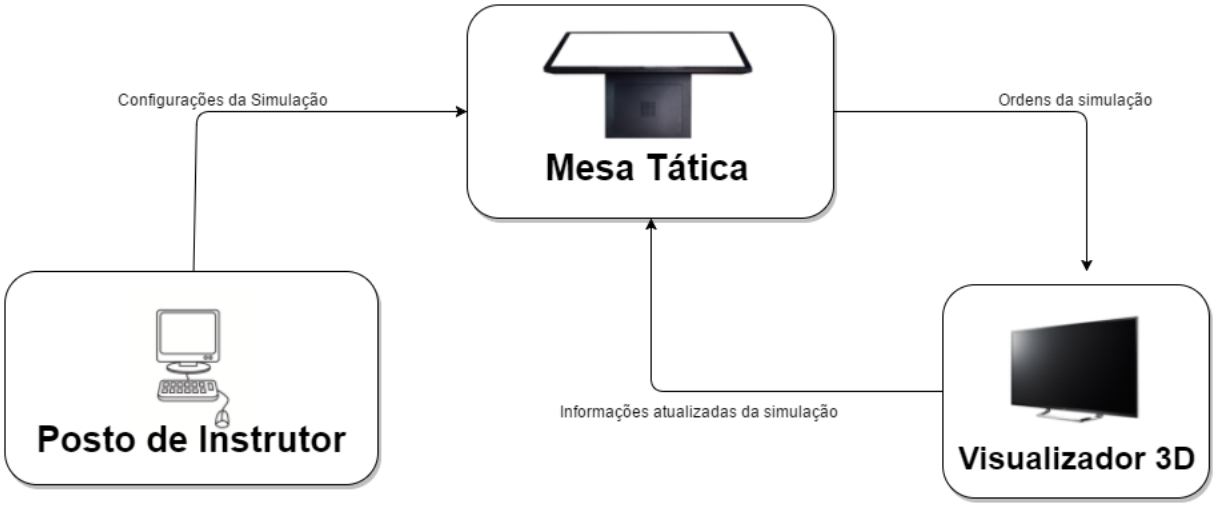
No posto de instrutor são realizadas as configurações para a simulação, é nele que o instrutor da simulação irá inserir as áreas pré definidas, elementos como baterias inimigas e aliadas, além de configurar a hora da simulação, o tipo e a quantidade de munição dos veículos e também poder controlar problemas que irão ocorrer durante a simulação.

Na mesa tática, uma mesa de 84 polegadas sensível ao toque, é onde o aluno irá operar a simulação. Nela poderão ser dadas ordens para as baterias, veículos ou soldados, ordens para ocupação de áreas, levantamento meteorológico, ajustagem de tiro e outras operações que o aluno que está a operar o simulador deverá realizar para concluir a simulação.

No visualizador 3D, com resolução superior a 100 polegadas, ocorre a visualização da simulação. Sendo assim, todas as ordens dadas pelo aluno na mesa tática serão vistas no visualizador. É nela que ocorre o processamento da física dos elementos da simulação, a ativação das animações, movimentações e quaisquer ações executadas pela mesa tática relacionadas a

elementos 3D.

Figura 3.1: Estrutura física do simulador



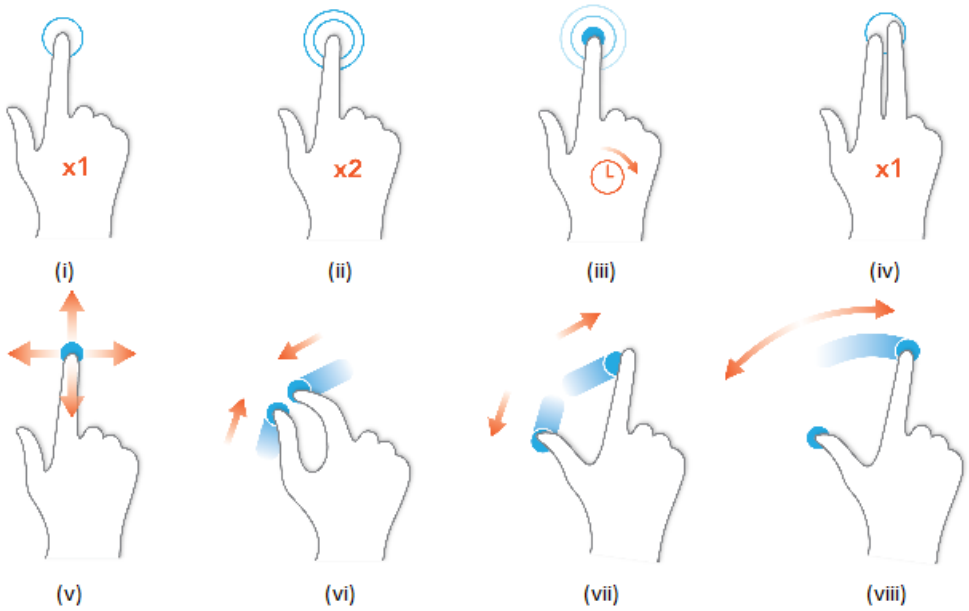
3.2 Proposta de Interface

Embora WOBBROCK; MORRIS; WILSON (2009) apresentem um conjunto de gestos e a maneira que estes seriam realizados com o uso de uma ou duas mãos, em sua pesquisa foi utilizado um dispositivo de 24 polegadas na realização dos testes de usuário. Sendo assim, para o presente trabalho foi necessário verificar a efetividade desse conjunto de gestos em uma tela de 84 polegadas. Apesar de também apresentar a maneira que o usuário realizaria os movimentos, WOBBROCK; MORRIS; WILSON (2009) não informam os algoritmos para o processamento dos gestos.

Com base nessa pesquisa, foram selecionados os gestos (Figura 3.2) para utilização na interface do presente trabalho. Este conjunto é composto por: i) um toque; ii) dois toques rápidos; iii) um toque longo; iv) dois toques simultâneos; v) um toque e arrasto para alguma direção (*Swipe*); vi) dois toques e aproximação (*Pinch*); vii) dois toques e afastamento (Zoom) e viii) rotacionar.

Em sistemas baseados em GUIs, o usuário tem sempre ao seu alcance periféricos para realizar a interação com o sistema. Entretanto, no caso do simulador, que utiliza uma NUI, os periféricos são inexistentes, limitando-se à interação direta do usuário com a tela sensível ao toque. Podendo estar disperso ao redor da mesa tática, o usuário deve estar apto a utilizar o sistema de maneira plena. Em vista disso, a interface deve fornecer ao usuário ferramentas, que

Figura 3.2: Conjunto de gestos para interação do usuário



antes eram proporcionados por meio dos periféricos, de forma a garantir essa interação.

Deste modo, foram conceituadas formas do usuário interagir com todo sistema. Caso o usuário queira acessar menus de entidades que se encontram fora de seu alcance é possível deslocar a carta para que o usuário tenha acesso a eles. Caso queira ter uma visão mais detalhada ou mais geral das entidades presentes na carta, foram desenvolvidas funcionalidades para diminuir ou aumentar suas escalas.

Da mesma maneira que o usuário deve ser capaz de acessar todas as funcionalidades da interface seja qual for sua posição, também deve ser garantido a interpretabilidade do que lhe é apresentado. Desta forma, foi inserida no sistema uma variante que determina, por meio da posição de entrada do usuário, a direção da disposição dos menus, fazendo com que a interface adapte-se à ela. Assim, o sentido de visualização dos elementos da interface sempre estará voltado para o usuário.

Dentre as funcionalidades presentes na mesa tática, há diferentes tipos, podendo serem categorizadas em dois grupos gerais. Em um grupo estão os comandos para a simulação, em outro as informações da simulação. O usuário pode controlar a simulação por meio do primeiro grupo, no qual estão contidas todas as ordens para realizar o exercício. No segundo grupo são visualizadas as informações da simulação como: o tempo de simulação; os estados dos

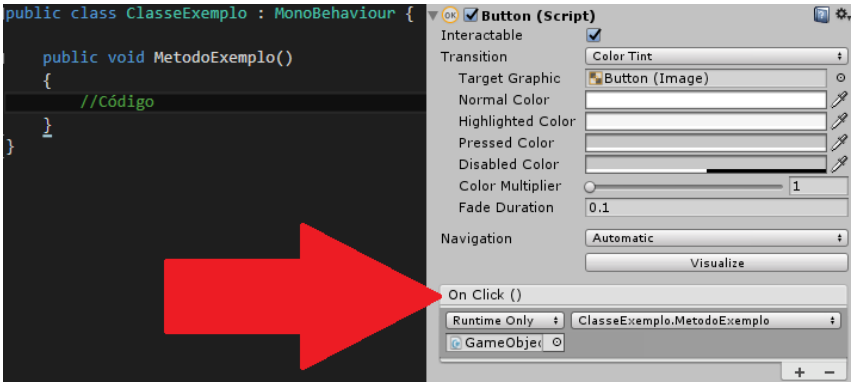
elementos; as configurações utilizadas na simulação; e demais informações relacionadas ao exercício.

Pensando em menus para o acesso a esses dois grupos de funcionalidade, foram escolhidos dois tipo de menus: Menu Radial e Menu de *Swipe*. Para realizar o processamento da interação do usuário com os menus e a conexão com o sistema, foi desenvolvido um gerenciador de eventos específico para a interface.

3.2.1 Gerenciador de eventos da interface

Na unity, todos os elementos gráficos que aceitam a interação do usuário têm, em suas estruturas, *callbacks* que podem ser associadas a funções do sistema. Por exemplo, botões contém o evento *OnClick()*, que é chamado toda vez que há uma interação do usuário. Sendo assim, o programador pode associar métodos do seu sistema a esse evento para processar a interação do usuário (Figura 3.3).

Figura 3.3: Evento *OnClick* de um botão na Unity



Entretando, essa abordagem não é eficiente em sistemas mais complexos, como o presente trabalho. Além de haverem vários programadores envolvidos, também há uma grande quantidade de funcionalidades. Isso faz com que seja necessário a modularização das unidades do sistema. O paradigma proposto por KRASNER; POPE et al. (1988) (MVC) tem como objetivo essa modularização, a fim de permitir aos desenvolvedores um melhor entendimento sobre a aplicação e a facilidade de modificação de cada unidade, não necessitando o conhecimento sobre todo sistema.

Contudo, no paradigma MVC, o controlador tem como uma de suas funcionalidades prover a interatividade de dispositivos da interface de usuário. Entretanto, o motor Unity já dispõe de um sistema de eventos que realiza o processamento das entradas do usuário. Porém,

não há uma unidade que realiza especificamente a comunicação entre a interface e o módulo da aplicação.

Baseado nesse paradigma, trazendo para o contexto da Unity e utilizando a linguagem C-Sharp, criou-se o Gerenciador de Eventos da Interface (IEM). A interface de usuário funciona como a visão, o IEM funciona como o controlador, e os métodos e classes responsáveis por realizar a simulação correspondem ao modelo.

3.2.2 Menu Radial Hierárquico

Layouts Radiais mostram-se eficientes na utilização de aplicações profissionais, na qual o usuário realiza centenas de comandos por sessão (SAMP; DECKER, 2010). Embora tenha um espaço limitado em seu primeiro nível, é possível estruturar uma grande hierarquia de comandos de forma eficiente, com baixo consumo de espaço de tela. Menus Radiais (Figura 3.4) mostram-se melhores que os tradicionais menus lineares pelo fato de que reduzem o tempo de seleção de um comando e diminuem os erros de interação (CALLAHAN et al., 1988).

A proposta do CRL apresentada por SAMP; DECKER (2010) foi desenvolvida em uma interface que tem como meio de interação o mouse. Os testes realizados para avaliar a eficiência dos menus radiais utilizavam como parâmetros o tempo de busca visual e a movimentação do mouse até o botão. Entretanto, no presente trabalho é utilizado uma NUI em uma tela de 84 polegadas, onde a interação do usuário é realizada por meio do toque. Sendo assim, as soluções e problemas encontrados por SAMP; DECKER (2010) tiveram que ser reavaliados, já que o menu se encontra em um dispositivo com diferentes tamanhos de tela, meios de interação e sentidos de visualização.

Neste trabalho é proposto a adaptação do CRL para uma interface NUI, tendo em vista a preservação das vantagens apresentadas por SAMP; DECKER (2010), como o consumo eficiente do espaço, a rápida utilização pelos usuários experientes, a facilidade de exploração por usuários iniciais e uma melhor resposta visual da hierarquia dos comandos da aplicação. Desta forma, foram desenvolvidas funcionalidades que garantem ao usuário uma melhor interação com o menu, permitindo sua movimentação, alteração de tamanho e sentido de visualização através do toque.

Apesar de SAMP; DECKER (2010) sugerir a utilização do CRL no centro da interface, no presente trabalho é necessário que o usuário possa acessá-lo em qualquer extensão da interface. Por se tratar de uma tela de 84 polegadas, o usuário se torna incapaz de acessar o centro

da interface dependendo de sua posição. Logo, o menu não estará situado permanentemente nesta localização. Consequentemente, deve-se garantir que, mesmo que o usuário seja capaz de movimentar o menu pela interface, este esteja visível por completo independente de sua posição.

Figura 3.4: Exemplo de Menu Radial

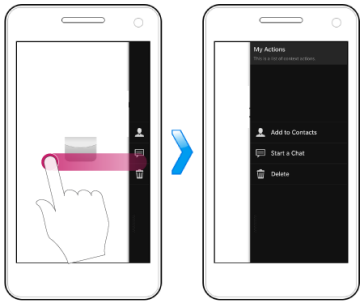
3.2.3 Menu de *Swipe*

Muitas vezes opta-se por utilizar barras de ferramentas em uma interface de usuário pelo fato de que atalhos para a ativação de funcionalidade são complexos para usuários iniciais. Entretanto, ocupam um espaço de tela, que muitas vezes não é disponível (SAMP; DECKER, 2010). Durante a utilização do simulador é necessário que o usuário tenha uma resposta visual de suas ações na simulação. Porém, não é obrigatório que ele veja essas informações o tempo todo, mas devem poder ser acessadas quando achar-se necessário. Por isso, optou-se por desenvolver a funcionalidade de *Swipe*.

O gesto de arrasto (*Swipe* em inglês) pode ser descrito como o toque do usuário em um ponto da tela e a movimentação deste toque para alguma direção. Atributos como a velocidade, direção, posição, distância podem ser considerados na hora de computar um *Swipe*. Este gesto não necessariamente pode ser feito somente com um dedo, podendo haver mais de um toque ou ser computado com o clique e arrastar do mouse.

A partir do *Swipe* é feita a exibição de um menu. Assim como no menu radial, o usuário deve ser ter acesso a ele independente de sua posição. Analisando aplicações que utilizam essa funcionalidade, geralmente vistas em *smartphones*, as extremidades da tela são utilizadas para que o usuário à acesse. Neste trabalho, foi definida que a localização do gesto, ou seja, a extremidade da tela em que o usuário o realiza, irá determinar o sentido de visualização do menu.

Figura 3.5: Exemplo de Menu de *Swipe*



De fácil configuração e grande usabilidade, diferentes tipo de menus de *Swipe* foram desenvolvidos no presente trabalho. O acesso a eles é feito através do mesmo gesto, um arrastar de dedos, porém a quantidade de toques utilizada no deslizar determina qual menu será aberto. Caso o usuário realize o *Swipe* com um toque, por exemplo, é mostrado um menu que contém informações gerais, já para acessar informações detalhadas, o uso de dois toques é necessário.

4 IMPLEMENTAÇÃO

O sistema de eventos na Unity é utilizado no envio de eventos para objetos da aplicação baseados na entrada do usuário, seja essa entrada através do teclado, mouse, toque ou qualquer dispositivo customizado (Unity - Manual: Mobile Device Input, 2016). O sistema de eventos consiste em alguns componentes que trabalham juntos para seu funcionamento. As principais funcionalidades do sistema de eventos são: a) gerenciar quando um *GameObject* é considerado selecionado; b) gerenciar quando um módulo de entrada está em uso; c) gerenciar o *raycasting* e d) atualizar todos os módulos de entradas quando requerido.

- Módulos de entrada. Os módulos de entrada contém a lógica central de como o sistema de eventos irá se comportar. Eles são usados para: a) manipular as entradas; b) gerenciar os estados dos eventos e c) enviar eventos para os objetos da cena. Os módulos de entradas da Unity foram desenvolvidos para serem estendidos ou modificados baseado no sistema de entradas que o programador deseja suportar. Por padrão, os módulos de entrada da Unity foram desenvolvidos para suportar configurações de jogos comuns, que suportam a entrada *Touch*, controles, teclado e mouses. Estes módulos enviam uma variedade de eventos para controlar a aplicação.

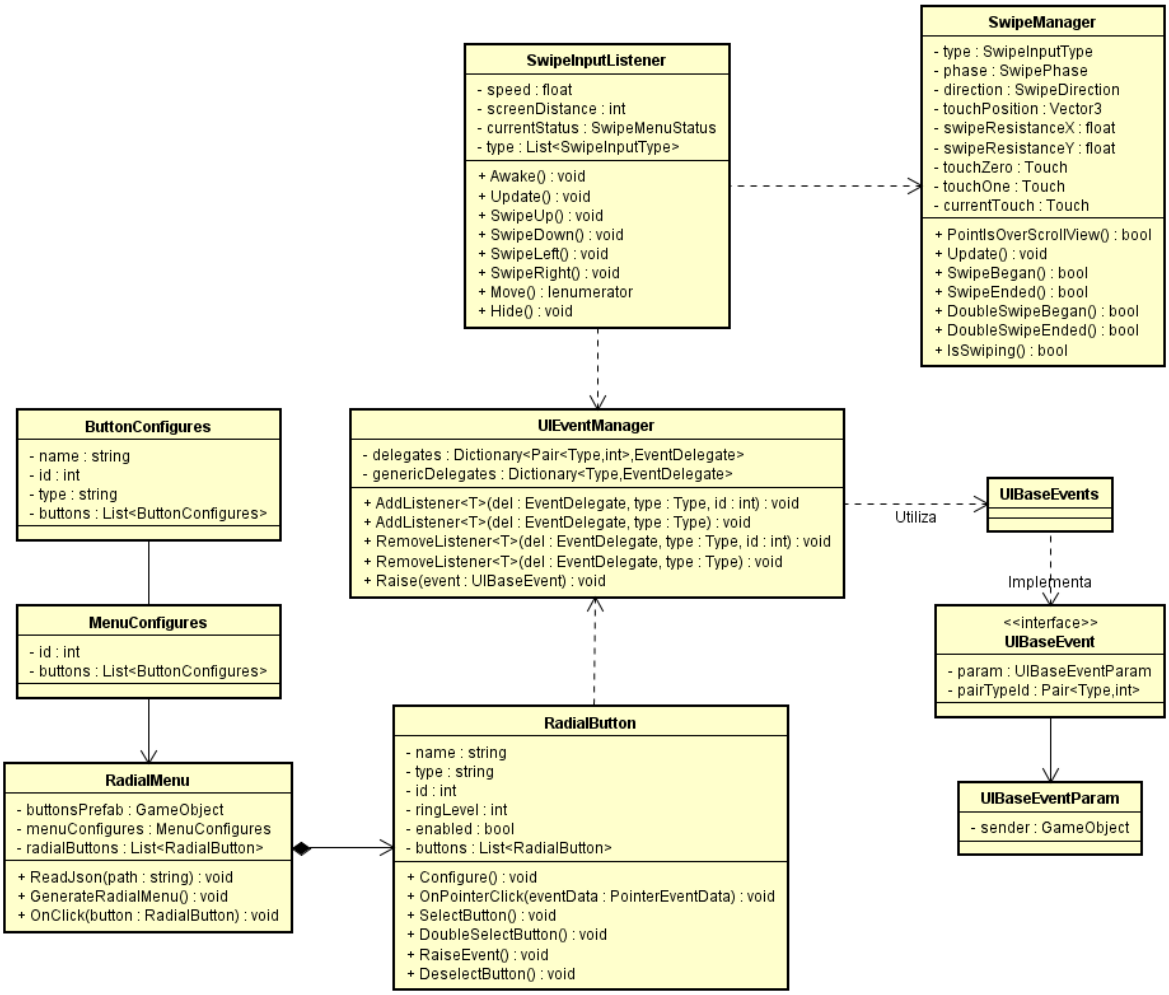
- *Raycasters*. *Raycasters* são usados para descobrir onde o mouse ou toque estão situados na interface. É comum os módulos de entrada utilizarem o *Raycaster* para calcular qual objeto está localizado na posição de entrada do usuário. Existem três tipos padrões de *Raycaster*: *Raycaster* gráfico, usado para elementos gráficos; *Raycaster* físico 2D para elementos físicos 2D e o *Raycaster* físico para os elementos físicos 3D.

- *TouchInput*. Dispositivos conseguem localizar um número de diferentes informações sobre um toque, incluindo sua fase(se ela começou, terminou ou se moveu), sua posição e o momento em que foi realizado. Além disso, a continuidade de um toque entre atualizações de *frames* podem ser detectadas pelos dispositivos. Sendo assim, é dado um número ID para o toque, podendo ser utilizado para determinar como o dedo está se movendo. A estrutura do toque utilizada pela Unity para armazenar dados relacionados a um único toque pode ser acessada através da função “Input.GetTouch”.

Sendo assim, a implementação foi realizada explorando as funcionalidades oferecidas pela Unity e seu sistema de eventos. A estrutura da implementação do presente trabalho pode

ser vista no diagrama de classes da Figura 4.1. O detalhamento da implementação é apresentada nas seções a seguir.

Figura 4.1: Diagrama de classes da interface



4.1 Gestos e funcionalidades

Para realizar a implementação dos gestos e das funcionalidades que são utilizadas no presente trabalho, a Unity fornece ao programador as primitivas sobre a interação do usuário com a interface, a quantidade de toques, a fase de cada toque e suas posições. Entretanto, a partir dessas informações, cabe ao programador implementar os métodos que irão processar os gestos do usuário.

4.1.1 Toque duplo

No método para processar o toque duplo (*ProcessDoubleClick*) são utilizadas duas variáveis: *LastClick* e *CurrentTime*. Estas variáveis representam, respectivamente, o momento em que ocorreu o último toque e o horário atual. Em um primeiro momento é calculada a diferença de tempo entre o toque atual (*CurrentTime*) e o anterior (*LastClick*). Caso essa diferença seja menor que o tempo definido pelo programador (0.1 segundos, neste caso), a função retorna verdadeiro, afirmando que há um toque duplo. Sempre que o método é executado, após o processamento do gesto, a variável *LastClick* é atualizada para o valor do tempo atual.

Algorithm 1 DoubleClick

```
1: função PROCESSDOUBLECLICK(lastClick)
2:     se currentTime − lastClick < 0.1 então
3:         lastClick = currentTime
4:         devolve true
5:     senão
6:         lastClick = currentTime
7:         devolve false
8:     fim se
9: fim função
```

4.1.2 Toque longo

O toque longo é processado a partir de um toque do usuário e a continuidade deste por um determinado tempo sem que haja mudança em sua posição. O método *ProcessLongPress* contém as variáveis *TouchCount*, *PressingTime*. Estas representam, respectivamente, a quantidade de toques na tela, e o tempo total que o toque está sendo realizado. Inicialmente, é verificado a existência de um toque através da variável *TouchCount*, caso haja, a variável *PressingTime* é incrementada com o valor de *DeltaTime*, que corresponde à diferença de tempo entre o *frame* atual e o anterior. Caso não haja nenhum toque, a variável *PressingTime* é setada para o valor 0. Após a atualização da variável *PressingTime*, é verificado se o seu valor é maior que o valor determinado pelo programador (0.3 segundos, neste caso). Caso seja, o método retorna verdadeira, afirmando que há um toque longo.

Algorithm 2 LongPress

```
    função PROCESSLONGPRESS
2:      se TouchCount == 1 então
            pressingTime+ = deltaTime
4:      senão
            pressingTime = 0
6:      fim se
        se pressingTime > 0.3 então
8:          devolve true
        senão
10:         devolve false
        fim se
12: fim função
```

4.1.3 Mover

A partir do toque inicial do usuário é calculado um *offSet*, que corresponde ao vetor entre o centro do objeto e a posição do toque. Este serve para garantir que o objeto seja transladado de acordo com a posição do toque inicial. O método *translate* recebe a posição do toque e o *offSet*, processando a nova posição do objeto.

Algorithm 3 Translation

```
    função TRANSLATE(touchPosition, offSet)
        position = touchPosition − offSet
3:      Move(position)
    fim função
```

4.1.4 Rotacionar

Para processar a rotação de um objeto através da interação do usuário foi implementado um método (*ProcessRotation*) que recebe 4 variáveis por parâmetro. *TouchZero* e *TouchOne* representam a posição dos dois toques do usuário no momento atual. *PrevTouchZero* e *PrevTouchOne*, representam a posição dos toques do usuário no *frame* anterior. A partir desses pontos, são criados dois vetores, *currentTouch* e *prevTouch*, que representam os vetores da diferença entre dois toques. O primeiro utiliza as variáveis *TouchZero* e *TouchOne*, o segundo utiliza as varíaveis *PrevTouchZero* e *PrevTouchOne*. O ângulo de rotação é dado através do arco tangente da diferença entre esses dois vetores, para isso é utilizado método Atan2. Por fim é aplicada uma transformação de radianos para graus e realizada a rotação.

Algorithm 4 Rotation

função PROCESSROTATION(touchZero, touchOne, prevTouchZero, prevTouchOne)
$currentTouch = touchZero - touchOne$
$prevTouch = prevTouchZero - prevTouchOne$
$angle = Atan2(currentTouch.y, currentTouch.x) - Atan2(prevTouch.y, prevTouch.x))$
$angle = Rad2Deg(angle)$
$ApplyRotation(angle)$
fim função

4.1.5 Escalar

Assim como no método *ProcessRotation*, o método que processa a escala utiliza das mesmas variáveis de entrada e o cálculo dos vetores *currentTouch* e *prevTouch*. Entretando, ao invés de calcular o ângulo entre esses dois vetores, é processada a variação do comprimento dos vetores. Como a magnitude de um vetor representa seu comprimento, por meio da diferença entre a magnitude dos vetores é possível saber se o usuário está afastando ou aproximando os dedos. A variável *resizeSpeed* é utilizada para adaptar a velocidade de escala, fazendo com que o objeto aumente ou diminua de tamanho de acordo com a necessidade do programador. Por fim, temos o valor que representará a nova escala do objeto.

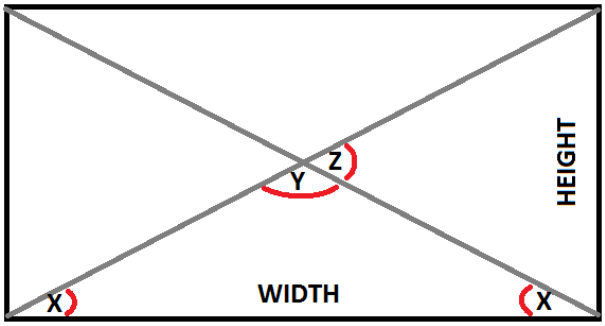
Algorithm 5 Scale

função PROCESSSCALE(touchZero, touchOne, prevTouchZero, prevTouchOne)
$currentTouch = touchZero - touchOne$
$prevTouch = prevTouchZero - prevTouchOne$
$resizeMagnitude = (currentTouch.magnitude - prevTouch.magnitude) * resizeSpeed$
$ApplyResize(resizeMagnitude)$
fim função

4.1.6 Sentido de visualização

Partindo do pressuposto que o usuário pode estar situado em qualquer posição ao redor da mesa, foi necessário implementar uma interface que se adapte a ele. Por meio da posição da entrada do usuário, é processado o sentido em que os menus serão exibidos. Para calcular a rotação dos itens da interface, a tela é dividida em quatro partes. Cada uma representa uma direção de visualização para o usuário. Um pseudocódigo é apresentado no Algoritmo 6. Este recebe a posição do toque, a largura e altura da tela. Primeiramente é calculado o ângulo X (*WidthAngle*), que pode ser visto na Figura 4.2, através do arco tangente da divisão da altura

Figura 4.2: Divisão da tela



pelo comprimento da tela. O ângulo Y (*VerticalAngle*) é calculado pela subtração de 180 por duas vezes o ângulo X. O ângulo Z (*HorizontalAngle*) equivale a duas vezes o ângulo X. Para determinar a rotação do menu é calculado o ângulo do toque do usuário e, a partir disso, em qual parte da tela ele se encontra.

Algorithm 6 InterfaceAdapter

função PROCESSSCALE(touchPosition, screenWidth, screenHeight)
  $widthAngle = RadtoDeg(Atan(screenHeight/screenWidth))$
  $verticalAngle = 180 - (2 * widthAngle)$
  $horizontalAngle = 2 * widhtAngle$
  $positionAngle = RadtoDeg(Atan2(position))$
  se $Abs(positionAngle < horizontalAngle/2)$ **então**
    $AplyRotation(90)$
  senão se $Abs(positionAngle > (horizontalAngle/2) + verticalAngle)$ **então**
    $AplyRotation(-90)$
  senão se $positionAngle > 0$ **então**
    $AplyRotation(180)$
  senão$AplyRotation(0)$
  fim se
fim função

4.2 Gerenciador de Eventos da Interface

Por meio da interação do usuário com os menus da interface do simulador, são executados os comandos para a simulação. Assim, o IEM foi projetado para realizar a conexão entre a interface e as outras unidades do sistema. Deste modo, cabe a ele gerenciar os eventos gerados pela interação do usuário e executar ações que estejam associadas a esses eventos. Esse sistema é composto por duas classes principais:

- *UIEventClasses* – Nesta classe estão definidos todos os tipos de eventos que poderão ser lançados pela interface. Como todos os eventos devem conter atributos pré-determinados,

a classe*UIBaseEvent* –uma classe base para que as demais classes a estendam– foi implementada. Seu principal atributo é um Par<Tipo, ID> que é utilizado no armazenamento e localização pelo *UIEventManager*. A partir dessa implementação é possível gerar eventos customizados. Sendo assim, podem ser estruturados para que executem novas funcionalidades;

- *UIEventManager* – Esta classe é responsável por gerenciar o registro de métodos na tabela de eventos e o levantamento destes eventos. Por ser necessário apenas um gerenciador de eventos da interface, que possibilite o acesso global à sua instância, foi implementado por meio de um *singleton* – um padrão de projeto de software que garante a existência de apenas uma instância de uma classe, mantendo um ponto global de acesso ao seu objeto. Os dois atributos principais da classe *UIEventManager* são um dicionário de *delegates* e um dicionário de *delegates* genéricos. Estes possibilitam o armazenamento e mapeamento dos eventos utilizados pela interface na comunicação com as demais unidades do sistema.

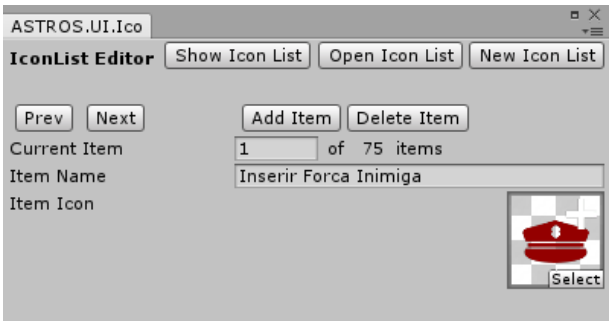
4.3 *IconsManager*

Diversas classes da interface de usuário utilizam ícones para compor seus menus ou painéis. Para facilitar a localização e modularização dos ícones foi implementada a classe *IconsManager*. Assim como a classe *UIEventManager*, ela também é definida por um *singleton*. Os ícones são armazenados por meio de um *Scriptable Object* – um contêiner de dados. A vantagem é que os *Scriptable Objects* podem ser salvos como *prefabs* sem a necessidade de serem associados a um *GameObject*.

Figura 4.3: IconsManager

Cada ícone é salvo como uma *Sprite* que, posteriormente, é associada a um componente *Image* na interface. Para que seja possível acessar um ícone específico, foi associada a ele uma *string*. Dada uma *string*, um método da classe IconsManager (Figura 4.3) retorna o ícone correspondente, presente no *Scriptable Object*. Para que seja possível editar o *ScriptbleObject* que contém a lista de ícones, foi desenvolvido um *IconListEditor* (Figura 4.4), onde é possível inserir ou remover ícones, atribuir ou atualizar o nome e a *Sprite* correspondente.

Figura 4.4: IconListEditor



4.4 Menu Radial

O *layout* do menu do presente trabalho seguiu o padrão proposto por SAMP; DECKER. Entretanto, foram realizadas algumas modificações, já que o *layout* desenvolvido pelos autores utiliza uma interface gráfica com periféricos. Dessa forma, seguiram-se os princípios de SAFFER para o desenvolvimento de uma interface natural, com adaptações no menu, a fim de ter uma melhor usabilidade na mesa sensível ao toque.

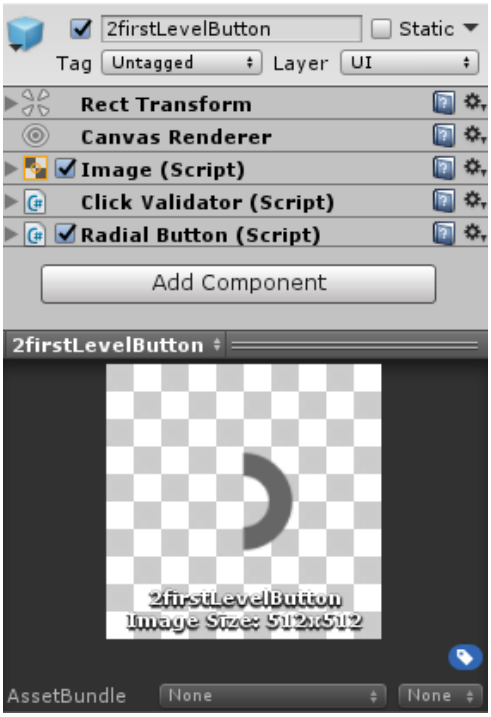
A implementação do menu desenvolvido para a mesa sensível ao toque é genérica e pode processar quantidades variadas de botões e níveis. Para este menu, foi definida a quantidade máxima de 10 botões por nível, podendo conter até três níveis. Após a realização de testes, concluiu-se que a utilização de um maior número de botões e níveis poderia ocasionar problemas de visualização e interação do usuário com a interface.

Cada botão do HRM é composto por um componente *Image* e dois scripts: *RadialMenuButton* e *ClickValidator* (Figura 4.5). O componente *Image* é responsável por armazenar a *Sprite* do botão que será representado por ele na interface gráfica. Já o script *RadialMenuButton* é utilizado para guardar as informações do botão, além de realizar a comunicação com a classe *RadialMenu* e *UIEventManager*.

Os componentes *Image* são representados por um retângulo e, por meio de sua área, a Unity processa a interação do usuário com os elementos gráficos. Entretanto, os botões do HRM não são retangulares, fazendo com que o processamento do toque sobre um botão não seja preciso. Para solucionar este problema, foi desenvolvido o script *ClickValidator* que, a partir da posição do clique do usuário, realiza um cálculo de transformações para determinar a posição exata na *Sprite*. De acordo com essa posição, é verificado o seu valor *Alpha*. Caso seja 0 (transparente), o *Raycast* é invalidado, não processando o clique sobre o objeto.

No total, foram criados 30 botões que podem compor os anéis do HRM, dependendo da

Figura 4.5: Botão radial no *Inspector* da Unity



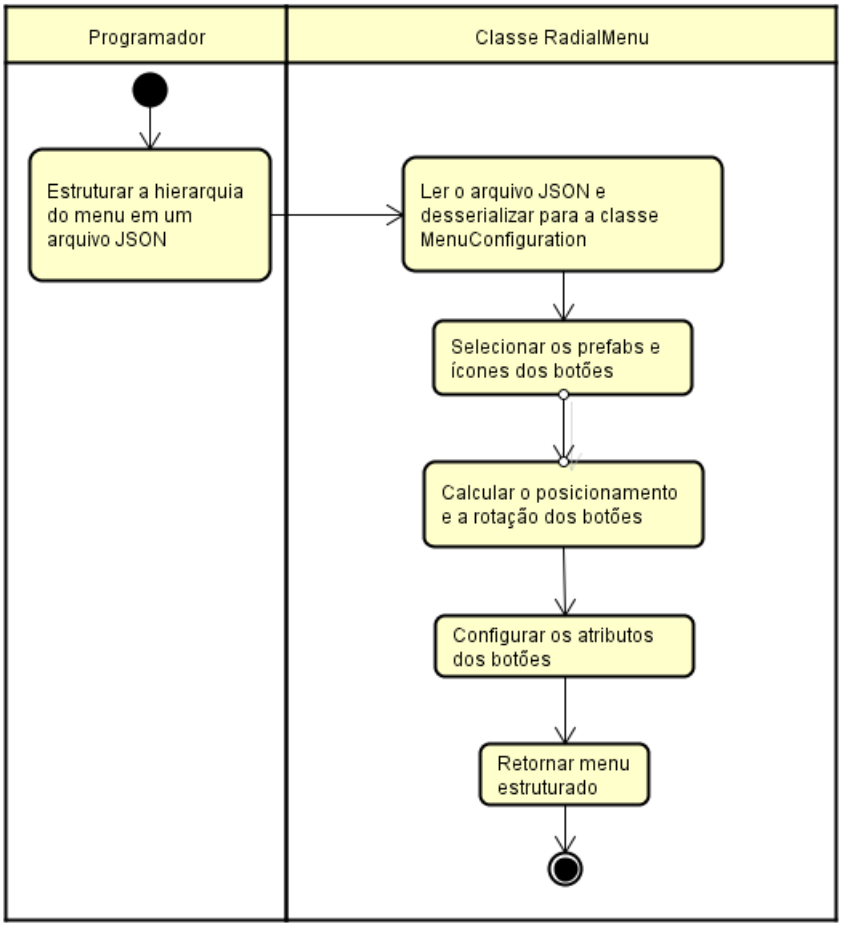
quantidade de botões e seus níveis, e um botão central fixo. Todos esses componentes foram salvos como *prefabs*, possibilitando a utilização por parte da classe *RadialMenu* no momento de criação de um novo menu.

4.4.1 Tempo de editor

Visando a facilidade no carregamento e geração do HRM, foi utilizado *JSON* para armazenar a hierarquia dos menus. Para isso, o arquivo foi estruturado de forma que contenha uma lista de botões. Estes têm como atributos fixos ID e tipo, que serão utilizados para o levantamento de evento quando houver a interação com o usuário, e como atributo opcional, uma lista de botões, possibilitando a criação da hierarquia.

A classe *RadialMenu* tem um atributo do tipo *MenuConfiguration* e, por meio do método *GenerateRadialMenu*, utiliza o objeto carregado para criar o menu. São feitos cálculos para saber quais *prefabs* dos botões serão utilizados para compor o menu, suas posições e rotações em relação ao botão central, além de serem configuradas suas informações na classe *RadialButton* (Figura 4.6).

Figura 4.6: Diagrama de atividade para criação do Menu Radial



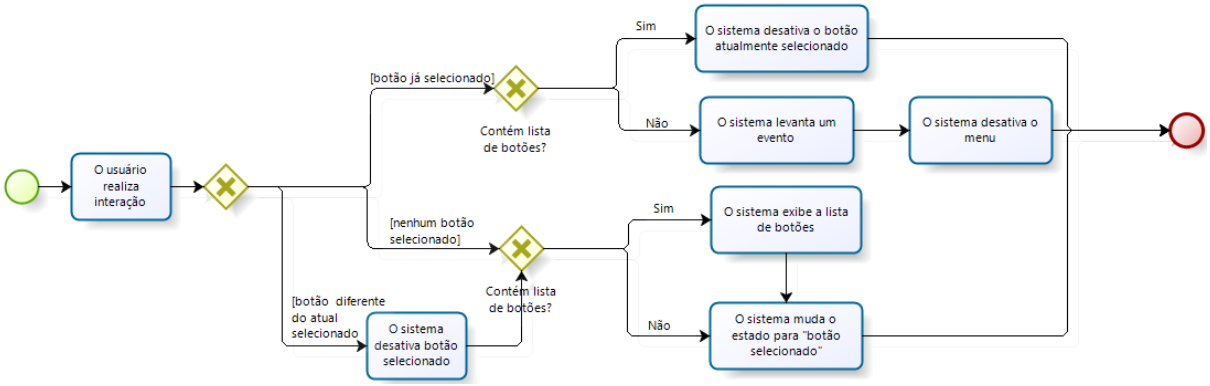
4.4.2 Tempo de execução

A classe *RadialMenu*, além de conter métodos para a criação do menu, também é responsável por gerenciar as informações gráficas, ou seja, como o menu será exibido na tela quando um botão está selecionado ou quando deve executar seu comando. Para isso, foi implementado o método *OnClick*. Assim, toda vez que é realizado o clique em um botão, o objeto do *RadialButton* correspondente informa o *RadialMenu*, onde é feito o processamento do clique. O ciclo de interação do usuário com o HRM implementado neste trabalho é descrito por um diagrama de processo apresentado na Figura 4.7. Quando há um clique do usuário sobre um botão, há três possibilidades:

- Possibilidade 1 – O botão clicado já está selecionado: neste caso, é processado um duplo clique sobre este botão. Caso o botão contenha uma lista de botões, esta lista é escondida e o botão é desselecionado. Caso contrário, é lançado um evento para execução do comando do botão.

- Possibilidade 2 – O botão clicado não é o mesmo que está selecionado: caso haja um botão selecionado, diferente do qual ocorreu o clique, o botão selecionado é desativado e o último botão a ser clicado é ativado.
- Possibilidade 3 – Nenhum botão selecionado: caso não haja nenhum botão selecionado, é ativado o botão clicado e, se ele contiver uma lista de botões, estes botões são exibidos.

Figura 4.7: Diagrama de processo da interação com o Menu Radial



4.5 Menu de *Swipe*

Os menus de *Swipe* foram pensados como atalhos para a simulação, para que o usuário seja capaz de ter uma resposta visual das informações e de suas ações no simulador. Para que fosse possível detectar o gesto de *Swipe* por parte do usuário e para a inserção dessa funcionalidade em menus, foram desenvolvidas as classes *SwipeManager* e *SwipeInputListener*.

- *SwipeManager* – A Unity disponibiliza ao programador informações da quantidade, velocidade e estados do toque do usuário. Seu objetivo é processar, mediante ao acesso a essas informações, um gesto de Swipe realizado pelo usuário, registrar a sua direção e o tipo de entrada. O *SwipeManager* foi implementado, assim como os gerenciadores anteriores, por meio de um *singleton*.
- *SwipeInputListener* – Este tem como objetivo acessar as informações do *SwipeManager* e, a partir delas, exibir ou esconder um menu de acordo com a direção e posição do gesto. Implementado de maneira genérica, é possível associar este script a qualquer *GameObject*, fazendo com que qualquer novo menu desenvolvido tenha a funcionalidade de *Swipe*.

5 RESULTADOS

O presente trabalho desenvolveu uma interface de usuário para uma mesa de simulação tática, baseada no paradigma MVC, buscando a modularização das unidades do sistema. A partir disso, implementou-se um gerenciador de eventos específico para realizar a comunicação entre a interface e o sistema. A arquitetura desenvolvida permite a inserção de novos menus e funcionalidades na interface, sem que seja necessária a reestruturação do código, garantindo uma rápida programação e fácil entendimento por parte do programador.

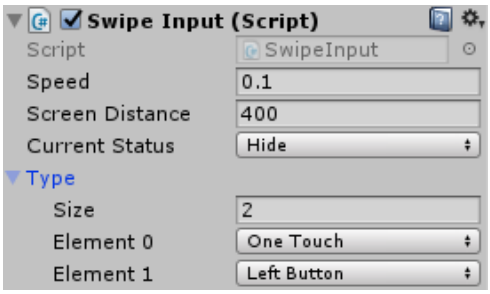
Antes da utilização de um gerenciador específico para os eventos da interface, botões e menus estavam diretamente associados aos métodos do sistema. Em um sistema de pequeno porte, em que não há uma complexidade elevada de sua estrutura, é possível fazer a ligação direta entre a interface e os métodos que serão executados na aplicação. Porém, no contexto do simulador, além de haver uma arquitetura complexa, fatores como a atualização constante e a necessidade de inserção e remoção de componentes impossibilitaram essa prática.

Desde as primeiras versões dos menus e gestos implementados, a avaliação da usabilidade foi uma atividade constantemente realizada. Foram feitos diversos testes com os usuários do simulador e, com isso, muitos problemas foram relatados. A partir das dificuldades apresentadas pelos usuários, foram realizadas alterações nos modos de interação, buscando facilitar o uso do simulador. Os principais problemas encontrados pelos usuários foram a dificuldade na realização de gestos como rotacionar e escalar, devido à posição dos toques e à velocidade de movimentação.

No movimento de *Swipe*, usuários iniciais encontraram dificuldade para realizar a ativação e a desativação dos menus devido à posição inicial do toque e à distância no movimento para validação do gesto. Para facilitar a realização dos testes de usuário, os parâmetros que influênciam na interação com o menu de *swipe* são exibidas e podem ser alteradas diretamente no *Inspector* da Unity (Figura 5.1). A utilização do menu de *Swipe* mostrou-se de fácil adaptação do usuário ao seu funcionamento, sendo um atalho adequado para a utilização no simulador.

A interferência entre os gestos utilizados na interface foi um dos problemas que surgiram à medida em que novas funcionalidades eram inseridas no simulador. A diferenciação entre a escolha do usuário realizar um *Swipe* para ativar um menu ou um *Swipe* para movimentar o mapa da simulação é um exemplo deste problema. A ideia inicial para resolvê-lo foi alterar a quantidade de toques que ativariam o menu e movimentariam o mapa. Porém, isto implicaria,

Figura 5.1: *SwipeInput* no *Inspector*



futuramente, na recorrência do problema quando novos menus fossem inseridos. Assim, a solução encontrada foi a inserção de um *delay* entre o usuário tocar a tela e movimentar os dedos (gesto de toque longo). Este *delay* diferencia entre um gesto e outro.

Desde a implementação do HRM, muitas funcionalidades foram adicionadas ao simulador, necessitando a criação de novos menus. Programadores que utilizaram a implementação do HRM, apresentaram facilidade para estruturação do menu por meio do arquivo Json, suprindo a necessidade do projeto de poder criar ou modificar de maneira simplificada os menus.

Apesar de SAMP; DECKER (2010) apresentarem os problemas encontrados em sua implementação e suas respectivas soluções para o *layout* radial, algumas não puderam ser adaptadas para o presente trabalho. A utilização de uma interface sensível ao toque necessitou a busca por outras soluções. Sendo assim, os problemas encontrados na implementação do HRM foram:

- Um clique para execução de um comando – A interação por meio do toque é muito mais imprecisa, por parte do usuário, do que com a utilização do mouse. Dessa maneira, no simulador, o *missClick* é muito recorrente. A solução encontrada foi fazer com que o usuário realize um clique duplo para executar uma ação.

- Textos grandes – Há inúmeras funcionalidades presentes no simulador que podem ser acessadas por meio do HRM. Porém, algumas apresentam textos muito longos, que nem a diminuição da fonte, nem a quebra de linhas foram capazes de resolver. Aliado à solução do problema anterior, optou-se pela utilização de ícones para representarem comandos. Ao primeiro toque do usuário, o texto do botão é exibido no centro do menu, dando ao usuário uma melhor resposta para sua interação.

- Localização do menu – Como o HRM pode ser acessado em qualquer posição na tela pelo usuário, foi necessário inserir uma *BoundingBox*, fazendo com que o usuário não possa

arrastar o mesmo para fora da tela, impossibilitando sua visualização.

Em testes realizados para a elaboração deste trabalho, o posicionamento do toque para realizar os gestos sobre o HRM apontou-se como um problema. Toques computados sobre os botões, em momentos que o usuário desejaria rotacionar, movimentar ou escalar o menu, poderiam ocasionar a execução de cliques indesejados. A definição de uma área para a realização dos gestos foi a solução encontrada para este problema. Sendo assim, toques que não estejam localizados sobre a região central do menu são invalidados, impedindo o usuário de executar a ação de um botão equivocadamente.

A Figura 5.2 apresenta um exemplo do HRM desenvolvido. A cor laranja representa os botões no estado "Selecionado". Os botões mais externos representam a hierarquia do botão no anel subjacente no estado "Selecionado". No centro do menu é exibido o texto referente ao botão mais externo no estado "Selecionado". A cor vermelha representa os botões no estado "Desativado", no qual é desativada a interação com o usuário. O menu é transparente, possibilitando a visualização da interface sob sua posição.

Figura 5.2: Exemplo do *layout* do Menu Radial desenvolvido

6 CONCLUSÃO

Esta pesquisa buscou implementar uma interface sensível ao toque que seja intuitiva para o usuário e de fácil programação. Para isso, foi realizada uma revisão de literatura sobre NUI, paradigmas de programação de interfaces e *layouts* de menus, que servem como base teórica para este trabalho. Além disso, foram analisados os trabalhos de implementação realizados por SAMP; DECKER (2010) sobre menus radiais, e os trabalhos de WOBBROCK; MORRIS; WILSON (2009) sobre a definição de gesto para uma interface sensível ao toque. Dessa forma, foi proposta uma implementação baseada nos conceitos dos autores citados neste trabalho.

O HRM permitiu a fácil criação de diversos novos menus para a interface do simulador. A possibilidade de mover, rotacionar e escalar o menu, além da adaptação de seu sentido de visualização, proporcionou ao usuário uma maior liberdade na utilização do simulador, podendo se movimentar livremente ao redor da mesa, melhorando, assim, a sua interação. A arquitetura proposta para a organização das unidades do sistema, baseada no paradigma MVC, permitiu a fácil inserção de novas funcionalidades para a interface. Assim, independente de ser o HRM, Menu de *Swipe* ou qualquer outro componente que realiza a interação do usuário, utiliza-se o gerenciador de evento da interface para realizar a comunicação com as unidades que processam a simulação.

Sendo assim, não será necessário realizar alterações na arquitetura para inserção de novos componentes na interface. Por outro lado, temos o revés da usabilidade e, ainda que tenham sido realizados testes de usabilidades das funcionalidades desenvolvidas para a interface sensível ao toque da mesa tática, os usuários que participaram dos testes também fazem parte do projeto, tendo conhecimento prévio do simulador. Sendo assim, como possibilidade de trabalhos futuros, é oportuna a realização de testes com usuários leigos, podendo ser descobertos outros problemas de usabilidade.

REFERÊNCIAS

BAILLY, G.; LECOLINET, E.; NIGAY, L. Wave menus: improving the novice mode of hierarchical marking menus. In: IFIP CONFERENCE ON HUMAN-COMPUTER INTERACTION. **Anais...** [S.l.: s.n.], 2007. p.475–488.

BARRETT, G.; OMOTE, R. Projected-capacitive touch technology. **Information Display**, [S.l.], v.26, n.3, p.16–21, 2010.

BURBECK, S. Applications programming in smalltalk-80 (tm): how to use model-view-controller (mvc). **Smalltalk-80 v2**, [S.l.], v.5, 1992.

CALLAHAN, J. et al. An empirical comparison of pie vs. linear menus. In: SIGCHI CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS. **Proceedings...** [S.l.: s.n.], 1988. p.95–100.

COCKBURN, A.; GUTWIN, C.; GREENBERG, S. A predictive model of menu performance. In: SIGCHI CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS. **Proceedings...** [S.l.: s.n.], 2007. p.627–636.

ENCYCLOPEDIA, C. D. **Multi-Touch**. 2016.

KAPTELININ, V. Item recognition in menu selection: the effect of practice. In: INTERACT'93 AND CHI'93 CONFERENCE COMPANION ON HUMAN FACTORS IN COMPUTING SYSTEMS. **Anais...** [S.l.: s.n.], 1993. p.183–184.

KRASNER, G. E.; POPE, S. T. et al. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. **Journal of object oriented programming**, [S.l.], v.1, n.3, p.26–49, 1988.

KURTENBACH, G.; BUXTON, W. The limits of expert performance using hierarchic marking menus. In: INTERACT'93 AND CHI'93 CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS. **Proceedings...** [S.l.: s.n.], 1993. p.482–487.

KURTENBACH, G.; BUXTON, W. User learning and performance with marking menus. In: SIGCHI CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS. **Proceedings...** [S.l.: s.n.], 1994. p.258–264.

NORTH, C. et al. Understanding multi-touch manipulation for surface computing. In: IFIP CONFERENCE ON HUMAN-COMPUTER INTERACTION. **Anais...** [S.l.: s.n.], 2009. p.236–249.

PELTONEN, P. et al. It's Mine, Don't Touch!: interactions at a large multi-touch display in a city centre. In: SIGCHI CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS. **Proceedings...** [S.l.: s.n.], 2008. p.1285–1294.

RAUTERBERG, M.; MAUCH, T.; STEBLER, R. How to improve the quality of human performance with natural user interfaces as a case study for augmented reality. **Advances in Occupational Ergonomics and Safety I**, [S.l.], p.150–153, 1996.

SAFFER, D. **Designing Gestural Interfaces**: touchscreens and interactive devices. [S.l.]: O'Reilly Media, Inc., 2008.

SAFFER, D. **Basic Principles of NUI Design**. 2014.

SAMP, K.; DECKER, S. Supporting Menu Design with Radial Layouts. In: INTERNATIONAL CONFERENCE ON ADVANCED VISUAL INTERFACES, New York, NY, USA. **Proceedings...** ACM, 2010. p.155–162. (AVI '10).

SEOW, S. C. et al. Multitouch and surface computing. In: CHI'09 EXTENDED ABSTRACTS ON HUMAN FACTORS IN COMPUTING SYSTEMS. **Anais...** [S.l.: s.n.], 2009. p.4767–4770.

SOMBERG, B. L. A comparison of rule-based and positionally constant arrangements of computer menu items. **ACM SIGCHI Bulletin**, [S.l.], v.18, n.4, p.255–260, 1987.

Unity - Manual: mobile device input. 2016.

WALKER, G. A review of technologies for sensing contact location on the surface of a display. **Journal of the Society for Information Display**, [S.l.], v.20, n.8, p.413–440, 2012.

WOBBROCK, J. O.; MORRIS, M. R.; WILSON, A. D. User-defined gestures for surface computing. In: SIGCHI CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS. **Proceedings...** [S.l.: s.n.], 2009. p.1083–1092.

ZHAO, S.; BALAKRISHNAN, R. Simple vs. compound mark hierarchical marking menus. In: ACM SYMPOSIUM ON USER INTERFACE SOFTWARE AND TECHNOLOGY, 17. **Proceedings...** [S.l.: s.n.], 2004. p.33–42.