4
Desenvolvimento do Trabalho

Este capítulo discorrerá sobre os aspectos referentes ao desenvolvimento deste trabalho bem como sua integração ao Sistema de Jogos Didáticos do Corpo de Fuzileiros Navais a ser apresentado.

4.1. O Sistema de Jogos Didáticos (SJD)

O Sistema de Jogos Didáticos (SJD) é, uma ferramenta de simulação de operações do Corpo de Fuzileiros Navais (CFN), que permite o treinamento de militares para a realização de Operações Anfíbias, Ribeirinhas de Incursão e de Paz.

Apesar de ter sido concebido em 1991, teve seu desenvolvimento a partir de 1993. Porém, foi em 1997, com o convênio junto ao Tecgraf / Puc-Rio que o sistema passou a contribuir efetivamente para o adestramento dos militares do CFN.

Dentre suas características destacam-se a possibilidade de simulação de desembarque, detecção, movimentação, engajamento, lançamentos de obstáculos e engenharia, apoio de fogo, apoio logístico, aliciamento, controle de distúrbios entre outras.

O SJD foi tem sua concepção de acordo com a Figura 7.

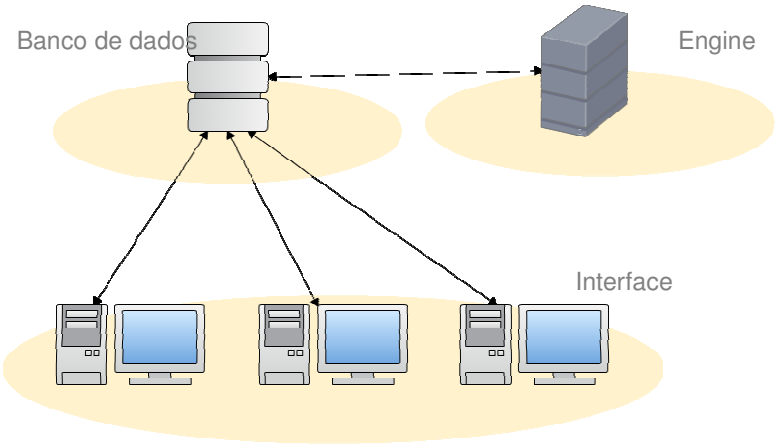

Figura 7. Concepção do SJD

O *engine* do SJD é composto por diversos módulos, cada um dos quais descrevendo uma ação possível, um evento, uma de abstração ou até mesmo um encapsulamento de funções genéricas. Estes módulos são separados em três categorias:

4.1.1. Módulos funcionais

Os módulos funcionais são aqueles responsáveis, de uma forma geral, pelo mapeamento doutrinário. Eles são compostos por funções de validação e execução de apoio de fogo, defesa aérea, desembarque, cálculos de detecção, movimento, detecção, dentre outros.

O módulo *principal.lua* é o único modulo dentre todos funcionais que compõe o *engine* que não é independente e que apenas executa os demais de forma sequencial conforme mostrado na Figura 8.

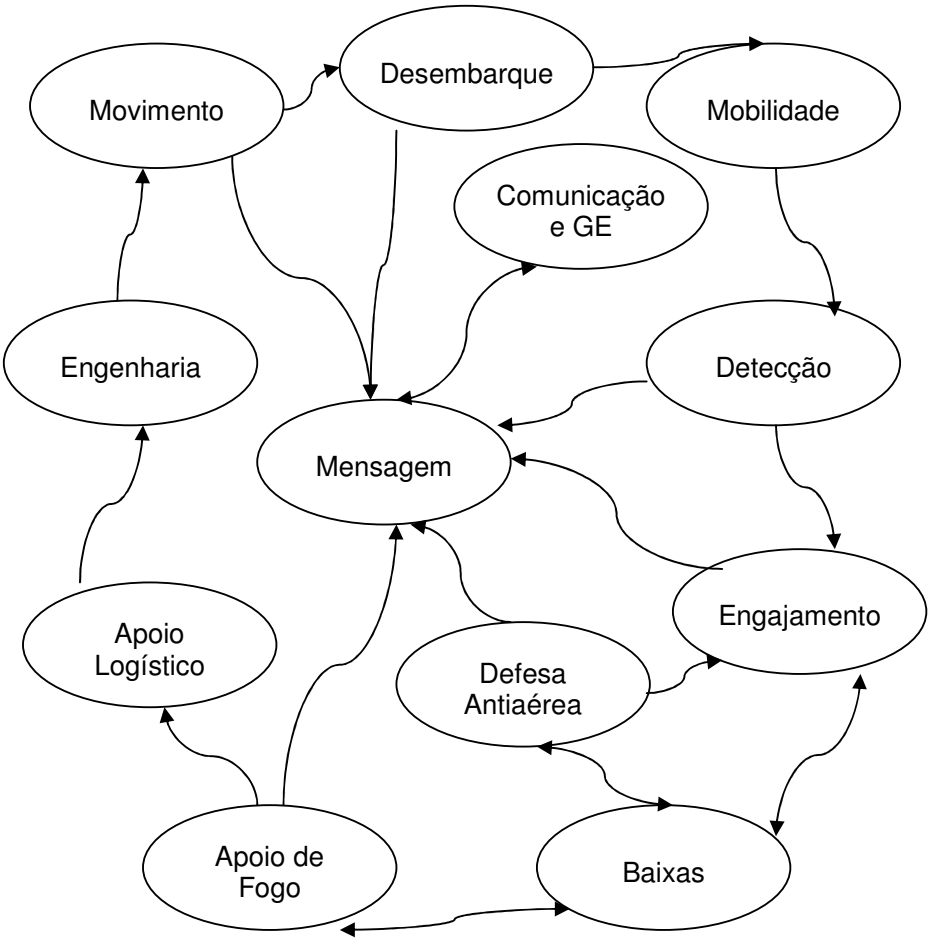

Figura 8. Sequência de execução de módulos do SJD

4.1.2. Módulos Auxiliares

Os módulos auxiliares se traduzem em funções matemáticas, geométricas, camadas intermediárias de API de banco de dados, manipulação de datas dentre outras.

4.1.3.Módulo de configuração

O módulo de configuração é um arquivo de configuração do SJD. Neste são definidos, habilitados ou desabilitados diversos parâmetros.

Dentre as variáveis de configuração, pode-se destacar o Sistema Operacional, o DNS da base de jogo, identificação do usuário, servidor de banco de dados, definição de diretórios de log, exportação, backup além da habilitação dos diversos módulos do SJD.

4.2. Modeling Agents For Real Environment (MARE)

Para o desenvolvimento deste trabalho, foi utilizado com base, um *framework* chamado *Modeling Agents For Real Environment* (MARE), também desenvolvido no Tecgraf, o qual, segundo Gustavo Lyrio (2007) é "*uma biblioteca de funções com o objetivo para construir agentes com comportamento pré-definido e inseri-los em ambientes reais executando em tempo real.* "

Usando a linguagem Lua, este *framework*, originalmente, parte da definição formal de agentes e emprega técnicas de programação de jogos como o algoritmo A*, máquinas de estado finito, terrenos divididos em tiles e outras para criar uma ferramenta altamente customizável para a criação de jogos ou simulações.

Sendo assim, permite também a criação e armazenamento de diferentes modelos de inteligência artificial que poderão ser apenas importados para dentro de uma aplicação e modificados a nível de script, ou seja, sem a necessidade de recompilar tudo.

Por ser escrito em Lua o MARE herda sua portabilidade e está disponível para qualquer plataforma que possua um compilador ANSI C (Windows, Linux, Mac OS, Solaris, etc.).

Feito para simular qualquer ambiente real em tempo real, o MARE oferece mapas genéricos e classes de agentes com métodos para realizar uma

variedade de ações como caminhar, falar e interagir os quais não foram utilizados integralmente neste trabalho.

A ideia por trás do MARE é oferecer um conjunto de ferramentas para permitir a criação de novos terrenos, agentes, ações e interações permitindo a criação de jogos.

Desta forma, o módulo de inteligência artificial, fruto deste trabalho utiliza os conceitos empregados no MARE, resumido apenas na máquina de estado (*state_machine.lua*) proposta por este *framework* uma vez que o terreno, sensores, atuadores e agentes apresentados no conjunto de biblioteca que o compõe não atende o ambiente do SJD por se tratar de uma simulação baseada em uma modelagem de um terreno real com agentes com comportamentos baseados na doutrina militar.

Nas sessão 4.3.6 o conceito de máquina de estado proposto neste trabalho será melhor abordado.

4.2.1. O Ambiente

No MARE, o ambiente é dividido em terreno e objeto. Conforme mencionado, ante a complexidade do ambiente de uma simulação e o modelo apresentado neste *framework* , tanto o terreno quanto o objeto foram redefinidos neste trabalho.

O terreno no MARE é definido como sendo um *array* bidimensional (uma matriz) de *tiles*, um célula de um terreno. Por sua vez, objeto MARE, por ser definido como tudo aquilo que não é agente, tal como uma rocha, árvore, móveis, dentre outros.

Como este trabalho está voltado para o SJD, o ambiente será tratado de forma diferenciada, pois o ambiente é composto por 21 camadas, como vegetação, relevo, clima, dentre outras.

4.2.2. O Agente MARE

MARE parte da definição proposta por Russel e Norving para criar seus agentes. Cada agente possui sensores e atuadores. Os sensores foram modelados como perguntas que o agente deve saber responder com verdadeiro ou falso. Cada atuador será uma ação que o agente pode tomar sobre o ambiente, sobre outro agente ou sobre ele mesmo.

Desta forma, para modelar um agente o usuário deve primeiro modelar as perguntas que definirão os sensores. Ou seja, qual porção do ambiente é relevante para o agente.

Assim como o módulo proposto no trabalho, esses sensores devem então ser agrupados em estados. Cada estado definirá uma condição na qual um agente pode estar. Uma vez modelados os estados, é necessário definir ações que o agente vai executar quando em um determinado estado. Essas ações, no MARE, são chamadas de atuadores. É necessário também identificar eventos que podem ocorrer no ambiente fazendo com que um agente altere a sua atual condição fazendo uma transição de um estado para outro. O MARE chama esses eventos de transições, conforme demonstrado na Figura 4.

Entretanto, os agentes MARE possuem atributos fixos os quais não foram empregados e foram completamente redefinidos por ocasião da implementação do módulo. Um agente a ser inserido no SJD é mais complexo e será apresentado nas próximas sessões.

4.3. Descrição do trabalho

Conforme já mencionado, o presente trabalho visa propor a inserção de um módulo de inteligência artificial no SJD tendo como base o MARE apresentado na sessão anterior a partir da modelagem dos agentes baseado na doutrina militar. Sendo assim, o módulo deve conter, não somente uma modelagem que traduza os comportamentos das frações militares, mas também uma arquitetura que permita a integração com o SJD.

Logo, o presente trabalho se integra com o SJD sendo um módulo independente dos demais o qual interage diretamente com o banco de dados do SJD de acordo com a Figura 9.

O usuário, ao inserir um comando através da interface, alimenta o banco de dados do SJD. Por sua vez, o *engine*, através de seus módulos independentes e auxiliares processa essas informações que volta ao usuário via banco de dados.

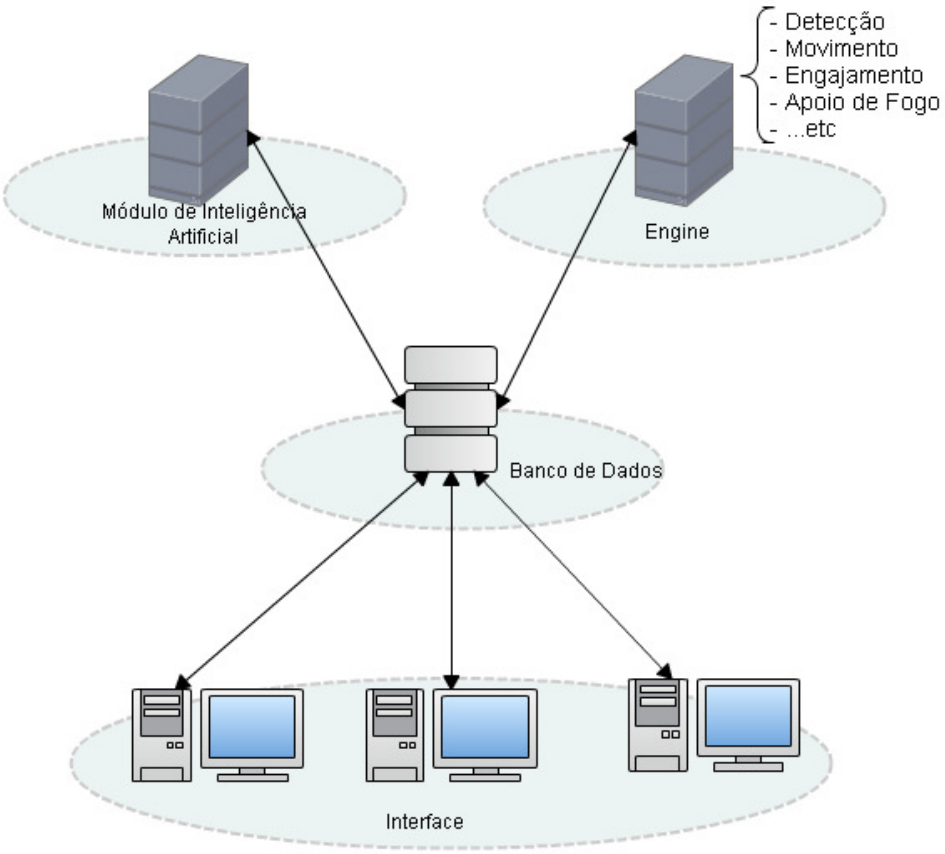

Figura 9. Integração do Módulo de Inteligência Artificial com o SJD

Ao mesmo tempo, o módulo de inteligência artificial, quando habilitado, está em contato direto com o banco de dados e processando as informações de interesse do agente e se comunicando com a interface via banco de dados também.

4.3.1. Conceito Básico

Assim como o MARE, o trabalho constitui-se de uma biblioteca de funções que visam a construção de agentes com comportamentos pré-definidos e inseridos no ambiente de simulação.

Por ser uma biblioteca genérica, o MARE tem todo um suporte a terreno, mas que difere da forma como o SJD trata o terreno. Sendo assim foi preciso desenvolver um módulo do jogo, abandonando o suporte do MARE e seus atributos que nele se apoiavam. Logo, estes foram gerados com base no banco de dados do SJD.

Da mesma forma como o *engine,* o trabalho é também composto por um conjunto de módulos auxiliares e de configuração além dos módulos que modelam o agente e a máquina de estados.

4.3.2. Módulo de Configuração

Alguns parâmetros descrito nos manuais militares tidos como conhecimento doutrinário, tais como distância de uma observação aproximada, observação afastada dentre outras são configuráveis e consolidadas no módulo **config.lua** . *(veja 4.1.3)*

Além destes, conforme já mencionado em sessões anteriores, uma simulação pode ocorrer em diversos níveis hierárquicos e é neste módulo que se define em que escalão a simulação está sendo executada, ou seja, até onde a inteligência artificial será emprega. Se ela será apenas no nível pelotão, ou se será no nível companhia, dependendo do que se espera do tipo de treinamento previsto por ocasião da utilização do SJD.

Algumas destas configurações estão dispostas na Tabela 2.

CFG_ESCALAO	Define a Inteligência Artificial no nível pelotão se for igual a 4 e no nível companhia se for igual a 5
CFG_DIST_SQUADS	Distância entre os Grupos de Combate
CFG_DIST_PEL	Distância entre os Pelotões
CFG_DIST_CIA	Distância entre Companhias
CFG_DIST_INDEF	Inimigo não previsto - Observação afastada
CFG_SGBD	Define o servidor de banco de dados
CFG_SYSTEM	Sistema Operacional

Tabela 2 . Algumas definições do módulo de configuração

4.3.3. Módulos Auxiliares

Da mesma forma que o *engine* o trabalho é composto por módulos auxiliares que são constituídos por funções matemáticas, de data, manipulação do relógio e de camadas de interface com o usuário.

Além destas, diferentemente do *engine*, outro módulo auxiliar foi criado para a inteligência artificial durante a implementação deste trabalho, o **dbcommon.lua**.

Este irá Estas funções auxiliares bem como suas definições estão dispostas na Tabela 3.

auxfunc.lua	Funções auxiliares independentes
dbcommon.lua	Tabelas Globais que guardam informações que são comuns a diversos módulos
math.lua	Funções matemáticas e geométricas
sqlbase.lua	Camada base da API de banco de dados

Tabela 3. Módulos auxiliares

Particularmente o módulo **dbcommon.lua** é composto por tabelas globais que guardam informações que normalmente não mudam no decorrer do jogo como hora do jogo, natureza do elemento de combate, limites da carta, dentre outras.

4.3.4. Módulos Funcionais

A partir da execução do **imain.lua**, os módulos funcionais serão executados, ativando o módulo do agente, da máquina de estado, dos sensores e dos atuadores além dos que irão realizar consultas e atualizações contínuas no banco de dados.

Dentre os módulos funcionais podemos destacar os dispostos na Tabela 4.

imain.lua	Módulo que executa os demais módulos
unit.lua	Contém os atributos e estados do agente
sensor.lua	Declaração da classe sensor
dbmain.lua	Funções de consulta e atualização do banco de dados
state.lua	Define os atributos da classe State
stateMachine.lua	Define os atributos da classe State _machine
dbmvt.lua	Insere dados na tabela movimento e na tabela tramo

Tabela 4. Módulos Funcionais

4.3.5. Agente

O agente proposto no trabalho, apesar de empregar conceitos comuns ao MARE, possui atributos específicos relacionados ao SJD uma vez que o agente proposto no MARE possui atributos próprios e que não atendem às necessidades do SJD. Logo do MARE, apenas utiliza-se a **state_machine.lua**.

O conceito de agente aqui é empregado tanto para as forças amigas, quanto para as inimigas, podendo ser qualquer tipo de unidade militar presente na simulação.

Os seguintes atributos, dispostos na Figura 10 abaixo foram, então, criados, fazendo com o agente possa ser empregado junto ao SJD.

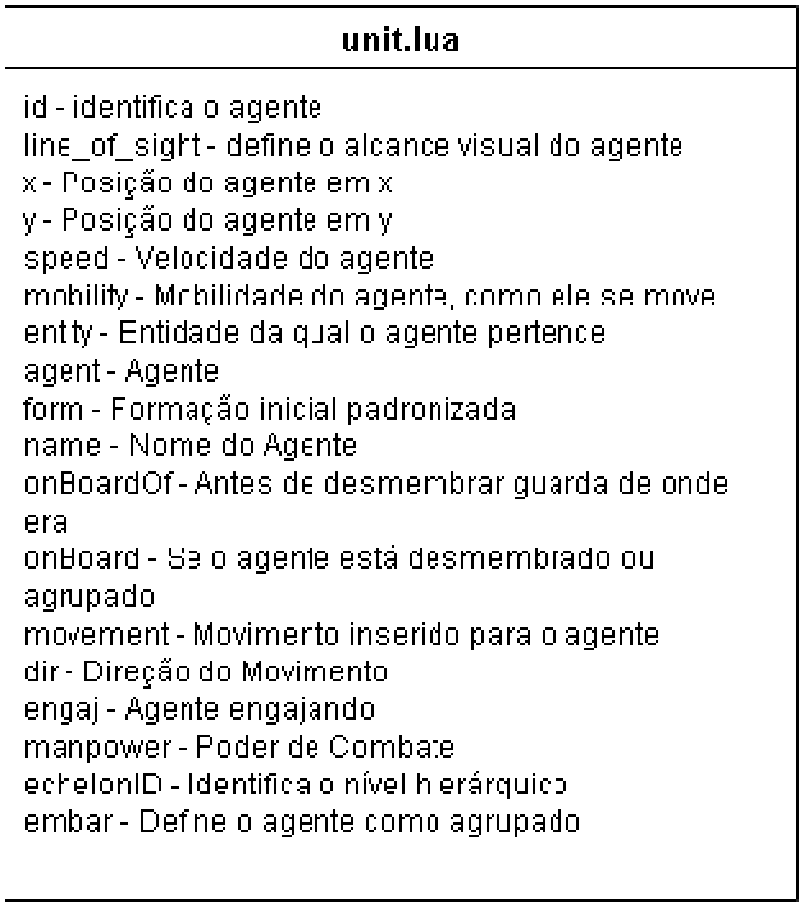

Figura 10. Atributos do agente

Além dos atributos, o módulo **unit.lua** define os estados os quais podem ser assumidos pelo agente através das funções definidas pelos sensores. Para

cada procedimento previsto na doutrina, utilizado para modelar o agente neste trabalho, tem-se um estado correspondente.

Estes estados foram definidos da seguinte forma:

- local *nome_do_estado* = State()
- *nome_do_estado*.set_id(2)
- *nome_do_estado*.set_value("*status do estado*")
- *nome_do_estado*.set_func(*função_do_atuador*)
- *nome_do_estado*.set_transition_n(0)
- *nome_do_estado*.set_trans_table({})
- *nome_do_estado*.setSensor (*n*, 0,"*questionamento do sensor*", *função_do_sensor*)

Basicamente, primeiro é definido o nome do estado e em seguida o mesmo recebe uma identificação bem como um *status* o qual durante o processo de transição facilitará a identificação do mesmo.

Como já dito, cada estado representa um procedimento previsto nos manuais. Logo, este estado deve possuir um atuador que fará com que nosso agente execute tal ação, através de funções específicas para cada estado e mostrada na Tabela 4.

Estado	Atuador	Ação
s1	*stop()*	Faz o agente parar
s2	*moveInLine()*	Faz o agente andar na formação "em linha"
s3	*moveInV()*	Faz o agente andar na formação "em V"
s4	*moveInCo()*	Faz o agente andar na formação "em coluna"
s5	*moveInTriangle()*	Faz o agente andar na formação "em triângulo"
s6	*ptrMovCont()*	Faz o agente executar uma patrulha com movimento contínuo
s7	*ptrMov()*	Faz o agente executar uma patrulha em dois escalões
s8	*ptrMovLances()*	Faz o agente executar uma patrulha por lances sucessivos
s9	*stop()*	Faz o agente parar quando há uma possibilidade de engajamento
s10	*engaj1()*	Faz o agente engajar quando detecta inimigo menor
s11	*engaj2()*	Faz o agente engajar quando detecta inimigo maior
s12	*engaj3()*	Faz o agente engajar com inimigo menor quando há detecção mútua
s13	*engaj4()*	Faz o agente engajar com inimigo maior quando há detecção mútua
s14	*stop()*	Faz o agente parar na preparação para a marcha para o combate
s15	*colMCmb()*	Faz agente andar na formação em coluna na Marcha para o Combate
s16	*colTat()*	Faz agente andar em Coluna tática na Marcha para o Combate
s17	*marApr()*	Faz agente andar em Marcha de Aproximação na Marcha para o Combate
s18	*cmbtEncDesborda()*	Faz agente engajar no Combate de Encontro desbordando o inimigo
s19	*atqCoord()*	Faz agente realizar um ataque coordenado
s20	*cmbtEncIniMenor()*	Faz agente engajar no Combate de Encontro contra inimigo menor em deslocamento
s21	*cmbtEncIniMaior*	Faz agente engajar no Combate de Encontro contra inimigo maior em deslocamento

Tabela 5. Funções dos atuadores dos estados

Após definido o estado, este é inicializado "setando" o número de transições para zero e a tabela de transições para uma tabela vazia.

Cada estado é composto de um ou mais sensores e definidos, dentro de cada um deles da seguinte forma:

- *nome_do_estado*.setSensor (*id*, *value*, *question*, *evaluate_func*)

Onde, *id* é o identificador do sensor, *value* inicializa o sensor como negativo ("0"), *question* é pergunta realizada pelo sensor que verifica se determinada condição ocorre a qual deve ser respondida positivamente para que o agente permaneça ou entre no estado correspondente. Por sua vez, *evaluate_func* é a função que executa tal verificação.

Conforme já mencionado, cada estado possui um ou mais sensores, os quais estão listados na Tabela 6 . Logo, a interação do usuário, via interface com o *engine* provocará as alterações no banco de dados. Estas por sua vez, são percebidas pelos sensores que, ao serem atendidos positivamente, fazem o agente mudar de estado e executar o atuador deste novo estado.

Função	***Question***
isStopped	O Agente está parado, teve movimento cancelado?
enemyListed	Existe inimigo listado?
enemyPosition	Posição inimiga conhecida?
enemyReach	Dentro do alcance do armamento inimigo?
platoonOrder	Ordem dada a um pelotão?
notEnemyReach	Não está dentro do alcance do armamento inimigo?
notEnemyPosition	Não há posição inimiga conhecida?
ptrOrder	Há ordem para executar patrulha?
condVisibility	Condição de visibilidade favorável?
enemyPosition2	Inimigo entre 2Km e 4Km?
enemyPosition4	Inimigo entre 2Km e 4Km durante a noite?
psbEngaja	Existe possibilidade de engajamento?
notIniDetPel	Inimigo não detectou o agente?
pelDetIni	Agente detectou o inimigo?
pelMaior	Agente maior que inimigo?
pelMenor	Agente menor que inimigo?
movOrder	Há ordem para o agente mover-se?
escCia	O escalão do agente é Companhia?
remCtt	O contato com o inimigo é remoto?
unlCtt	Contato com inimigo é pouco provável?
immCtt	Contato com inimigo é iminente?
psbEngajaCia	Há possibilidade de engajamento para Companhia?
iniInf	O escalão do inimigo inferior à Pelotão?
iniSup	O inimigo é suficientemente forte?
iniStopped	O inimigo está parado?
iniNotStopped	O inimigo não está parado?

Tabela 6. Funções dos sensores que compõe os estados

Com as funções dos atuadores definidas, e as funções dos sensores definidas, pode-se ter cada estado modelado de acordo com a Figura 11.

```
1316    -------------------------
1317    -- MOVIMENTO
1318    -------------------------
1319    -- PELFUZNAV DESLOCANDO EM LINHA
1320    local s2 = State()
1321    s2.set_id(2)
1322    s2.set_value("Deslocando em Linha")
1323    s2.set_func(moveInLine)
1324    s2.set_transition_n(0)
1325    s2.set_trans_table({})
1326    s2.setSensor(1, 0,"Existe Ordem de Movimento?", movOrder)
1327    s2.setSensor(2, 0,"Inimigo Previsto?", enemyListed)
1328    s2.setSensor(3, 0,"Posição Ini Definida?", enemyPosition)
1329    s2.setSensor(4, 0,"Dentro do Alcance Armamento Inimigo?", enemyReach)
1330    s2.setSensor(5, 0,"Ordem para Pelotão?", platoonOrder)
```

Figura 11. Implementação de um estado do agente

4.3.6. Máquina de Estados

Fica evidenciado nas sessões anteriores que a técnica utilizada para implementar a Inteligência Artificial neste trabalho foi a chamada Máquina de Estados Finita.

A grande justificativa para a escolha desta técnica foi a de que o agente proposto neste tipo de simulação tem seu comportamento pré definido composto de ações já previstas quando motivado por determinados estímulos observados no ambiente no qual ele está inserido.

Uma máquina de estados finitos, por sua vez, pode ser definida como um conjunto de estados com regras de transições entre esses estados. E a utilização deste modelo consiste em representar através de estados as possíveis ações de um agente com regras de transição através de sensores que representam as condições que devem ser verificadas para avaliar se o agente deve mudar de estado.

Pode-se dizer ainda que:

> *"Uma máquina de estados finita consiste em um conjunto de estados (incluindo um estado inicial), uma série de entradas, uma série de saídas e uma função de transição. A função de transição de estados recebe as entradas e o estado corrente e retorna um único novo estado e um conjunto de saídas. Como só há um novo estado possível, Máquinas de Estados Finitas são usadas para codificar comportamento determinístico. (FUNGE)"*

Logo, tem-se que estado inicial da máquina é o estado inicial do agente. As entradas são os sensores e as saídas são estes sensores atualizados. Por

fim, as transições são condições que precisam ser satisfeitas para permitir que a função de transição de estado modifique o estado corrente.

Sendo o agente modelado baseado em um comportamento pré definido, o fator de aleatoriedade não se aplica neste caso. Este é realizado pelo próprio *engine* do SJD através das interações do usuário via banco de dados.

No próximo capítulo, serão estudados os casos de emprego dos estados com seus sensores e atuadores bem como as transições entre eles.