1. INTRODUCTION

Simulation in Defense has played a key role in research and development work for newer technologies and techniques. The project Human Simulation intends to justify and graphically depict practically all major actions / stances a live soldier can perform.

After some consideration regarding a meaningful way of putting System, Simulation in an appropriate perspective we arrived at the following distinction.

System: a system exists and operates in time and space.

Model: a model is a simplified representation of a system at some particular point in time or space intended to promote understanding of the real system.

Simulation: a simulation is the manipulation of a model in such a way that it operates on time or space to compress it, thus enabling one to perceive the interactions that would not otherwise be apparent because of their separation in time or space.

Simulation is a discipline for developing a level of understanding of the interaction of the parts of a system, and of the system as a whole. The level of understanding which may be developed via this discipline is seldom achievable via any other discipline.

A simulation generally refers to a computerized version of the model which is run over time to study the implications of the defined interactions. Simulations are generally iterative in there development. One develops a model, simulates it, learns from the simulation, revises the model, and continues the iterations until an adequate level of understanding is developed.

Modeling and Simulation is a discipline, it is also very much an art form. One can learn about riding a bicycle from reading a book. To really learn to ride a bicycle one must become actively engaged with a bicycle. Modeling and Simulation follows much the same reality. You can learn much about modeling and simulation from reading books and talking with other people. Skill and talent in developing models and performing simulations is only developed through the building of models and simulating them. It’s very much learning as you go, process. From the interaction of the developer and the models emerges an understanding of what makes sense and what doesn't.

One of the real benefits of modeling and simulation is its ability to accomplish a time and space compression between the interrelationships within a system. This brings into view the results of interactions that would normally escape us because they are not closely related in time and space. Modeling and simulation can provide a way of understanding dynamic complexity!

Simulation is the process of representing, controlling, manipulating and implementing real world applications virtually.

2. BASIC DEFINITIONS

Simulation is generally referred to in the context of the acronym ``M&S,'' which stands for *modeling and simulation* or *models and simulations* depending on the context of its use.

Model is a physical, mathematical, or otherwise logical representation of a system, entity, phenomenon, or process.

Simulation is a method for implementing a model over time.

Modeling and simulation refers to the use of models, including emulators, prototypes, simulators, stimulators, either statically or over time, to develop data as a basis for making managerial or technical decisions. The terms ``modeling'' and ``simulation'' are often used interchangeably.

Simulator is (a) a device, computer program, or system that performs simulation; (b) for training, a device which duplicates the essential features of a task situation and provides for direct human operation.

3. WORKING TOOLS

3.1 VEGA PRIME

Vega prime is an application programmer interface (api). It is a cross-platform real-time tool kit for visual simulation and a high-performance software environment and toolkit for realtime simulation and virtual reality applications.

It consists of a graphical user interface called LynX Prime, Vega Prime libraries and header files of C++ callable functions.

3.1.1 LYNX PRIME

LynX Prime simplifies the development process. It is the graphical user interface for defining, editing, and previewing Vega Prime applications. LynX Prime allows the user to configure an application without writing code. It is an editor for adding instances of classes and defining the parameters for the instances. These parameters, such as the position of an *observer*, the *objects* and their placement within a *scene*, movement within a scene, lighting, environmental effects, and target hardware platform, are stored in an instance framework in an Application Configuration File (ACF). An ACF contains information that a Vega Prime application needs at initialization, as well as information to be used during runtime.

3.1.2 APPLICATION CONFIGURATION FILE

The Application Configuration File (ACF) contains all the information that a Vega Prime application needs at initialization, as well as some information to be used during runtime. A single Vega Prime executable can produce a variety of applications by interpreting different ACFs. ACF files are in Extensible Mark-up Language (XML) format.

The Lynx prime user interface contains four different sections:

- GUI View
- Instance Tree API View
- API View
- Toolbar and menus

3.2 MULTI-GEN CREATOR

MultiGen Creator produces realistic three-dimensional models and terrain for use in real time applications. Creator includes an integrated set of powerful tools for building hierarchical visual databases in a “what you see is what you get” (WYSIWYG) environment. These databases conform to the *Open Flight* format for general purpose modeling. Creator’s comprehensive set of tools is organized to focus on both creating and editing models and databases. Once a database is constructed, its hierarchy can be quickly rearranged and whole groups of elements can be modified. Individual elements can be isolated for precise editing. Elements such as lights and textures are grouped and saved in palettes for easy reuse. External conversion utilities are included or available through third-party vendors so you can import models saved in other file formats, such as those used by popular animation and CAD applications.

3.3 MICROSOFT VC++

It is used as a platform to combine the C++ code and the Vega Prime Functionalities. VC++ is essential not only for incorporating the many functional dependencies associated with each module of Vega but is also critical to set the environment variables and MPI path for the system on which the package is run.

4. SYSTEM REQUIREMENTS

4.1 ABSOLUTE MINIMUM CONFIGURATION

- Windows workstation
- 512 MB RAM
- 2 GB hard disk space
- CD ROM drive
- OpenGL 1.2 compliant graphics card
- Windows XP Professional or Windows 2000 Professional
- Visual C++ 6.0 Service Pack 4

4.2 RECOMMENDED MINIMUM CONFIGURATION

- Windows workstation, 1.0 GHz
- 1 GB RAM
- 4 GB hard disk space
- CD ROM drive
- OpenGL 1.2 compliant graphics card
- Windows XP Professional or Windows 2000 Professional Service Pack 2
- Visual C++ 6.0 Service Pack 5

5. PHASE I : TORNADO APPLICATION & DOF

During the initial stages of the project, i.e., before the submission of feasibility analysis, the following tasks were assigned to us:

1. Studying and getting familiar with Lynx Prime and VEGA Prime by studying the accompanying manuals.
2. Gaining the knowledge of **linking** in VC++.
3. To design a small object on top of a larger object such that it has a path and observe collision detection.

During the completion of the above tasks, with the help of the provided manuals, we created a **tornado** application which consisted of a terrain, a farm house, a grain silo, a cow, a car and a tornado. In this application, the tornado moved on a pre-defined path and destroyed the farm house. It also lifted the cow and exploded the grain store. The car was made to move on the terrain using mouse controls and debris emerged from the house once the car collided with it. This completed the task of a small object moving on top of the other.

Then, to achieve collision detection, we coded the movement of the tornado using the **moveTornado** flag and made it to stop as soon as it came near the farm house. With the tornado coming to a halt, the lights of the car were also programmed to blink.

This was achieved using the **tripod isector** (inter sector) placed in front of the tornado.

With these tasks completed we were assigned the following new tasks regarding the movement of the tornado:

1. To make the tornado take an alternative path once it detects the farm house. This task prepared us for assigning any stance to a soldier.
2. To replace the farmhouse with another object after the tornado destroys it.

The first task involved the knowledge of the co-ordinates of the farm house and the co-ordinate at which the tornado comes in contact with it. Once this was obtained by trial and error, we put a **"for"** loop in the movement of the tornado for a fixed margin of coordinates. In short, we designated a particular co-ordinate path to its movement for a few seconds before it went ahead on its original pre defined path.

This resulted in the tornado detecting the farm house (made evident by the blinking of the car head lights) and then taking a smooth path around the house and leaving it untouched.

Secondly, once the tornado destroyed the farm house, we imported another **".flt"** file at the exact co ordinates of the farm house so that it disappeared and another object appeared in its place; a dead soldier in our case

As a final attempt to complete and random collision detection we also placed detection on the car so that it too destroyed the farm house, lifted the cow and exploded the silo on coming in contact with them.

After the completion of these tasks we concentrated on the next major portion of the project: **Analysis of degrees of freedom.**

This task was aimed at analyzing the degrees of freedom in the various human joints during the course of any stance, for example, **"walking"**, **"running"** etc. Further, it also demanded the study of the various movements of these basic joints during a stance-to-stance conversion like **"walking to running"** etc.

Two algorithms and flow charts (provided in the appendix) were constructed for the same.

An Analysis of the walk subroutine:

Fig. 1: Movements of joints in the Walk sub routine

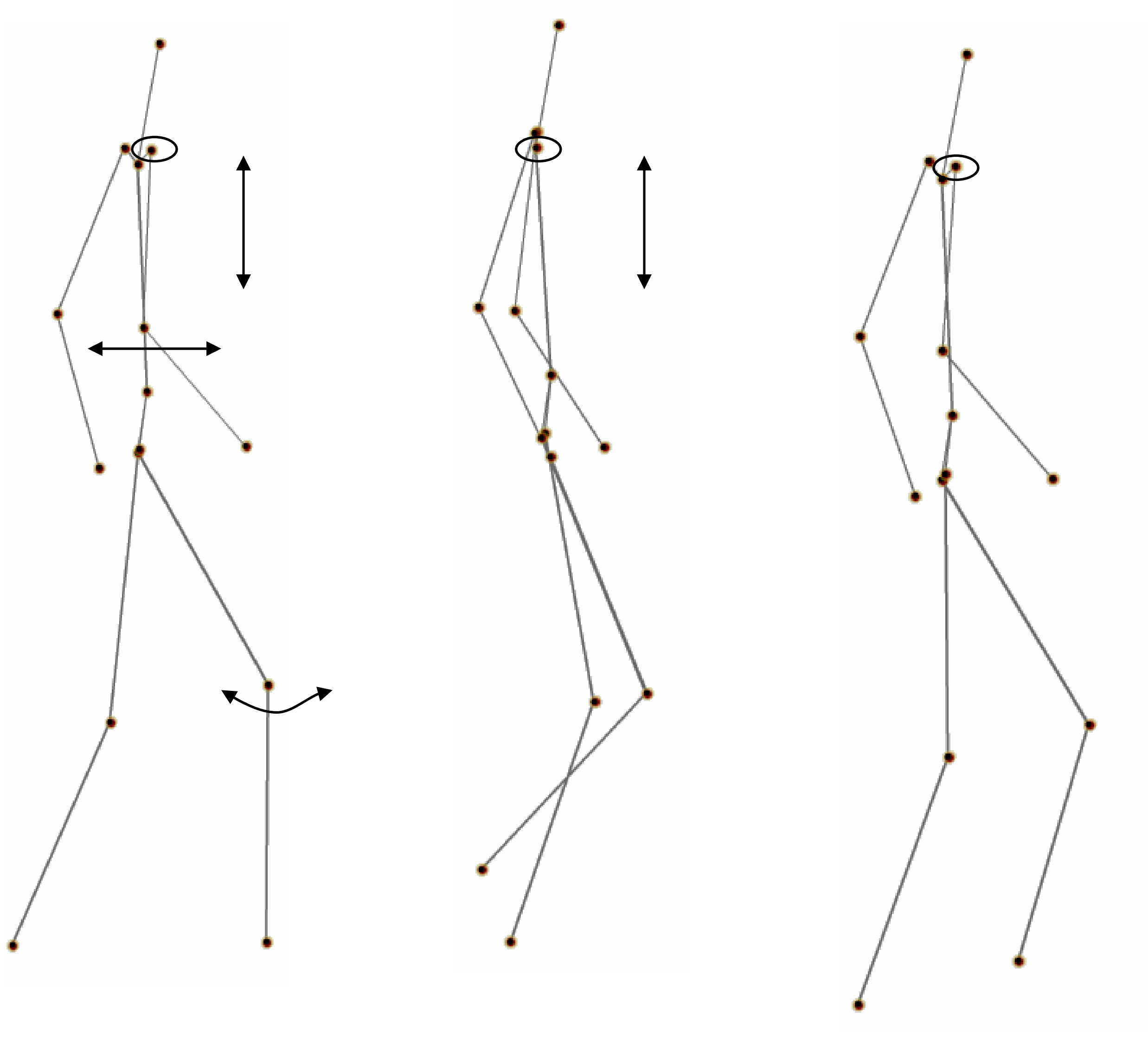
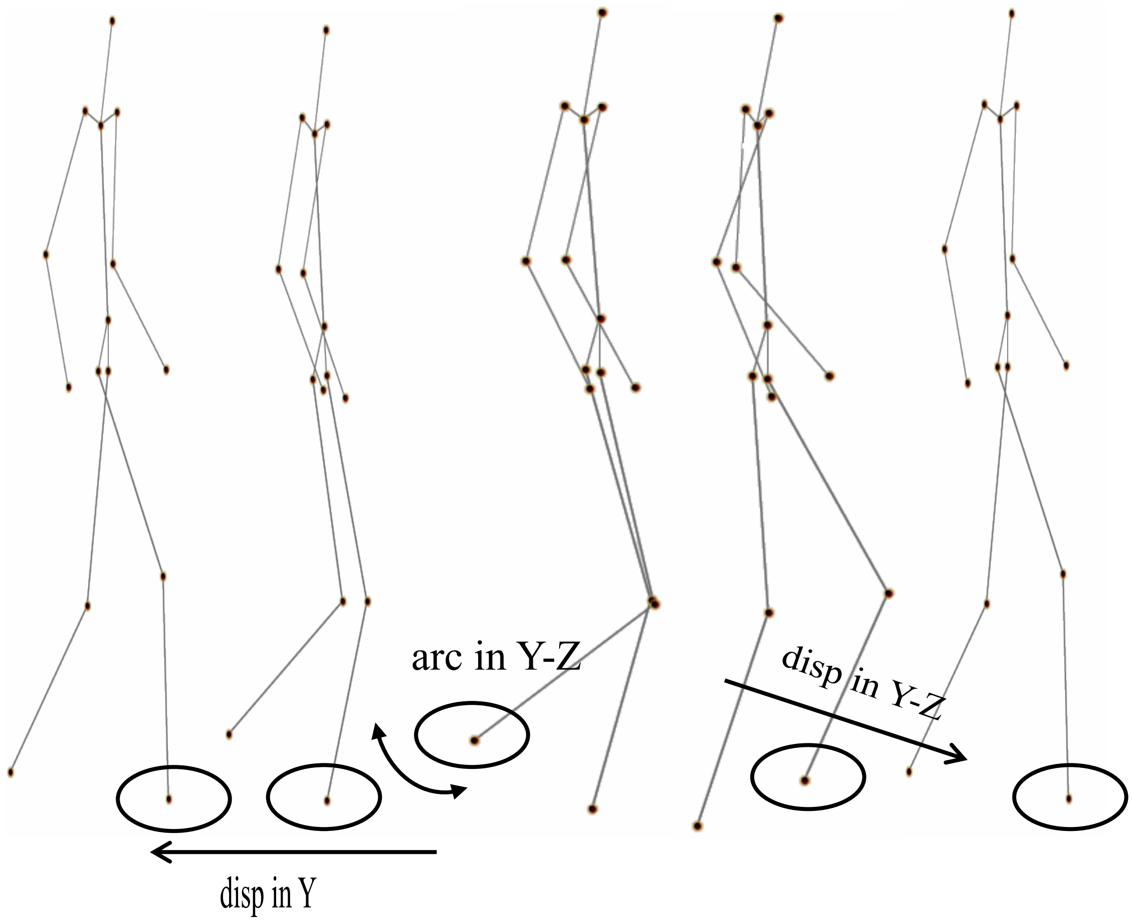


Fig. 2: Ankle Movement



6. PHASE II : STANCE IMPLEMENTATION

After the submission of the feasibility report we were provided with the complete code by Lt. Col. P. K. Pal. We were assigned to go through the code and create some new stances and show transition from one stance to other.

The first hassle in our path was a constant error of memory leak and some fatal errors during run time of the code. Thus we came to the conclusion that VEGA Prime is system dependent. These errors were conquered with continuous interaction with Lt. Col. Pal, the MSDN Library and the information gathered from the world-wide-web.

We learnt that the **“.flt”** files of soldiers were given the following joints movement using Multi-Gen creator:

- **Neck**
- **Shoulders**
- **Elbows**
- **Wrists**
- **Hips**
- **Waist**
- **Torso**
- **Knees**
- **Ankles**

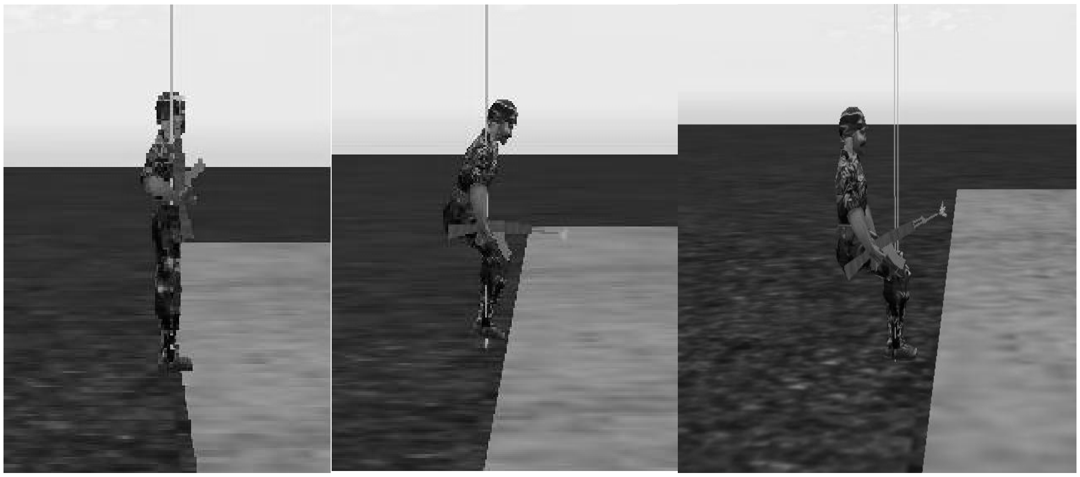
By manipulating the movement of these joints the soldier was made to do various stance conversions. With this knowledge we created a new workspace in VC++ and created our first stance **“Sit on Chair”** wherein the soldier imitates sitting on a chair. Along with this we also used the **“Running”** stance and modified it.

With two stances at our disposal we worked on the transition between these two stances. For this we called these stances as functions at different intervals of time, for example, “running” from 5-10 seconds and “sitting on chair” from 20-30 seconds.

However, there were two problems with this transition. Firstly, to make the soldier to come to a halt after running we had to use too many time frames and secondly, both these stances were totally time dependent.

The soldier continued to execute the same stance if the stance time was increased rather than stopping after executing it once. To overcome this we added an **“if”** condition on the movement of the last joint moved so that after a certain number of degrees moved, the soldier would automatically stop executing the stance.

The snap shots for **“sitting on chair”** are shown below.



The code for **“sit on chair”** is as follows:

```
void myRakshak::SitOnChair(float fSpeed)

{

if(!bSitOnChair)
        {
                bSitOnChair = true;
        }

        if(!bStep1)
        {
                if(fLtShP > -20.0)
                {
                        fLtShP = fRtShP -0.5*fSpeed;
                        fWaistP = fWaistP +0.5*fSpeed;
                        fLtHipP = fLtHipP -0.6*fSpeed;
                        fLtKnP = fRtKnP +0.6*fSpeed;
                }
                else if(fLtShP < -20.0)
                {
                        fWaistP = fWaistP +0.5*fSpeed;
                        fLtShP = fRtShP -0.5*fSpeed;
                        fLtHipP = fLtHipP -0.6*fSpeed;
                         fLtKnP = fRtKnP +0.6*fSpeed;
                }
                if(fRtShP > -20.0)
                {
                        fRtShP = fRtShP -0.5*fSpeed;
                        fWaistP = fWaistP +0.5*fSpeed;
                        fRtHipP = fLtHipP -0.6*fSpeed;
                        fRtKnP = fRtKnP +0.6*fSpeed;
                }

                else if(fRtShP < -20.0)
                {
                         fRtShP = fRtShP -0.5*fSpeed;
                         fWaistP = fWaistP +0.5*fSpeed;
                         fRtHipP = fLtHipP -0.6*fSpeed;
                         fRtKnP = fRtKnP +0.6*fSpeed;
                }
                if(fLtKnP > -45.0)
                {
                        fLtKnP = fRtKnP +0.5*fSpeed;
                }
                else if(fLtKnP < -45.0)
                {
                  fLtKnP = fRtKnP +0.5*fSpeed;
                }
                if(abs(abs(fRtKnP)-45.0) <= 1.2)
                {
                        bStep1 = true;
                }
                if(fRtKnP > -45.0)
                {
                        fRtKnP = fRtKnP +0.5*fSpeed;
                }
                else if(fRtKnP <-45.0)
                {
                        fRtKnP = fRtKnP +0.5*fSpeed;
                }
         }  if((bStep1)&&(!bStep2))
 {


        if(fRakP > -40)
        {
                fRakP = fRakP - 0.35*fSpeed;
```

```
                        fRtKnP = fRtKnP +0.2*fSpeed;
                        fLtKnP = fLtKnP +0.2*fSpeed;
                         if(fRakP > 87.0)
                         {
                                    bDied = false;
                                    bStep2 = true;
                         }
            }
 }
```

As is evident from the code the movement of these basic joints by a few degrees can lead to a completely new stance. That is why modeling is termed as a work of art!

Once this new stance and transition was created Lt. Col. Sabharwal advised us to create key board calls for the stances rather than creating function calls out of them. This eliminated the problem of time frames and increased the user friendliness of the soldier.

With time we created several new stances, modified the previous ones which were not working properly, assigned keyboard calls to all the stances and thus achieved the transition between all the stances.

All the stances are described as children of three parents: **"Standing, Kneeling and Lying Down".** To achieve a smooth transition all the stances are to be brought to these basic stances for stance-to-stance transition.

7. VARIOUS STANCES AND CODES

7.1 BASIC STANCES

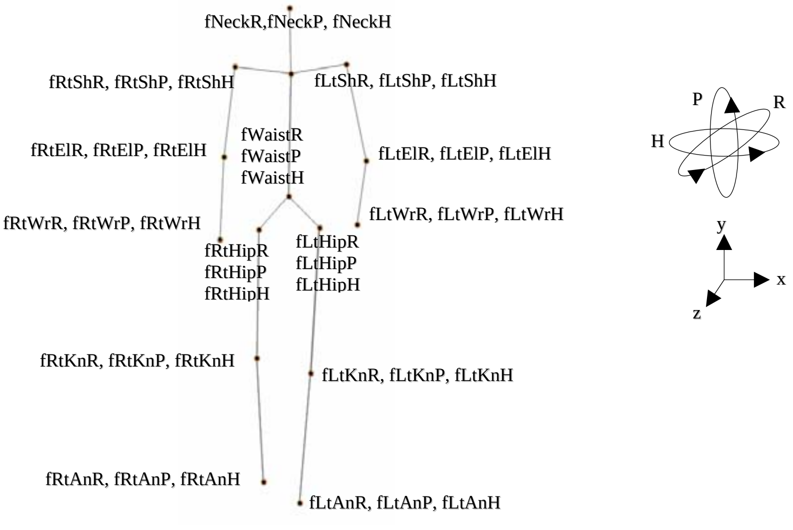
- STANDING
- KNEELING
- LYING

7.2 OTHER STANCES AND JOINT DEFINITION

In any human motion, the following joints are involved:

a) Shoulder (left and right)
b) Hip (left and right)
c) Knee (left and right)
d) Ankle (left and right)
e) Elbow (left and right)
f) Foot (left and right)
g) Waist
h) Neck

These are the 14 joints in the soldier defining the complete motion.



7.2.1 A list of all stances implemented.

- Standing
- Kneeling
- Lying
- Idling
- Pumping Arms
- Lobbing
- Walking
- Search OP
- ROP
- Wading
- Running
- Falling face Down
- Fleeing
- Lying Rifle Firing
- AGL Firing
- Battle Crouching
- Bayoneting
- Back Blasting
- Fwd Blasting
- Crawling
- Para Deployment
- Jump Crouched
- Stand By
- Ready
- Look watch
- Kneeling Fire
- Standing Fire
- Dhava Position
- Swimming
- Rope Up
- Rope Down
- Navigate compass
- Search Binoculars
- Read paper
- Walking RL
- Standing RL
- Fwd Kicking
- Side Kicking
- Sitting on Chair
- Sitting on Ground
- Standing Pistol Fire
- Squatting
- Turning Back
- Wave
- Celebration
- Read map
- Falling backwards
- Stealth
- RL Firing

8 STANCE-TO-STANCE TRANSITION

For analysis of the movement of joints of the soldier's body during his transition from one stance to another we use the two stances Stand and Crawl.

8.1 STAND TO CRAWL

The following joints observe changes in their movement during transition from Stand to Crawl:

1. Shoulder
2. Knee
3. Ankle
4. Foot
5. Neck
6. Hip
7. Waist

The whole transition between these two stances can be broken into the following intermediate stances:

Stand-------> Kneel------> Crawl Position--------> Crawl

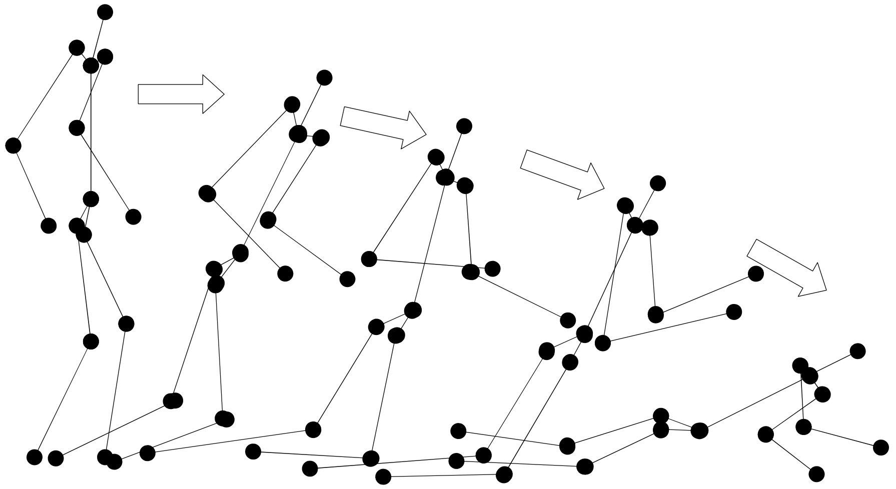


Fig. 4 *Transition from stand to crawl*

8.1.1 STAND TO KNEEL

From stand to kneel there is no apparent movement in the shoulder and the elbow joints. The neck bends in the direction off the “kneel” along with the torso. The knee joints pitch in a direction opposite to the foot pitch and the ankle joint is raised. The waist remains stationary and the hip joints pitch till the torso rests on the ankles. On contact of the knee joints with the ground the torso aligns in the Z axis.

8.1.2 KNEEL TO CRAWL POSITION

The torso is bent forward in the direction of the crawl. The shoulder and the elbow joints are pitched. Simultaneously the knee joint is rolled till the elbows touch the ground.

8.2 STANCE TO STANCE TRANSITION IMPLEMENTATION

In order to achieve Stance to Stance transition, an idea of a Transition tree was implemented.

The idea in implementation goes thus:

Three Stances: Standing, Kneeling and Lying are defined as basic stances.

Each stance is subcategorized as a child of the basic parent stance. Each stance performs a position check at the beginning.

Transition is achieved in 4 steps:

1. Invoking the stance;
2. Calling the parent node;
3. Invoking the appropriate basic Transition routine and finally
4. Invoking the next required stance.

In addition, some stances do not require the position check while a few others can perform transitions without invoking the basic Transition routines.

The Transition tree implemented in Appendix E

9. TIME INDEPENDENT, KEY BASED TRANSITION SEQUENCE

Another major idea implemented was the removal of fLastRenderTime functionality.
This resulted in a completely time independent Keystroke based Transition Sequence.

9.1 ADVANTAGES OF THIS SEQUENCE OVER THE TIME BASED TRANSITION SEQUENCE:

- User friendliness: User is not required to recompile code each time to invoke every routine
- Easy Stance to Stance Transition: Transitions at the push of one keystroke
- Saves unnecessary repeated addition of ToDo( ) function and approximation of time requirement.
- An alphabet is easier to remember than a code.

The complete codes for different stances are available in Appendix D.

10.   SOME CLASSIC SCREENSHOTS

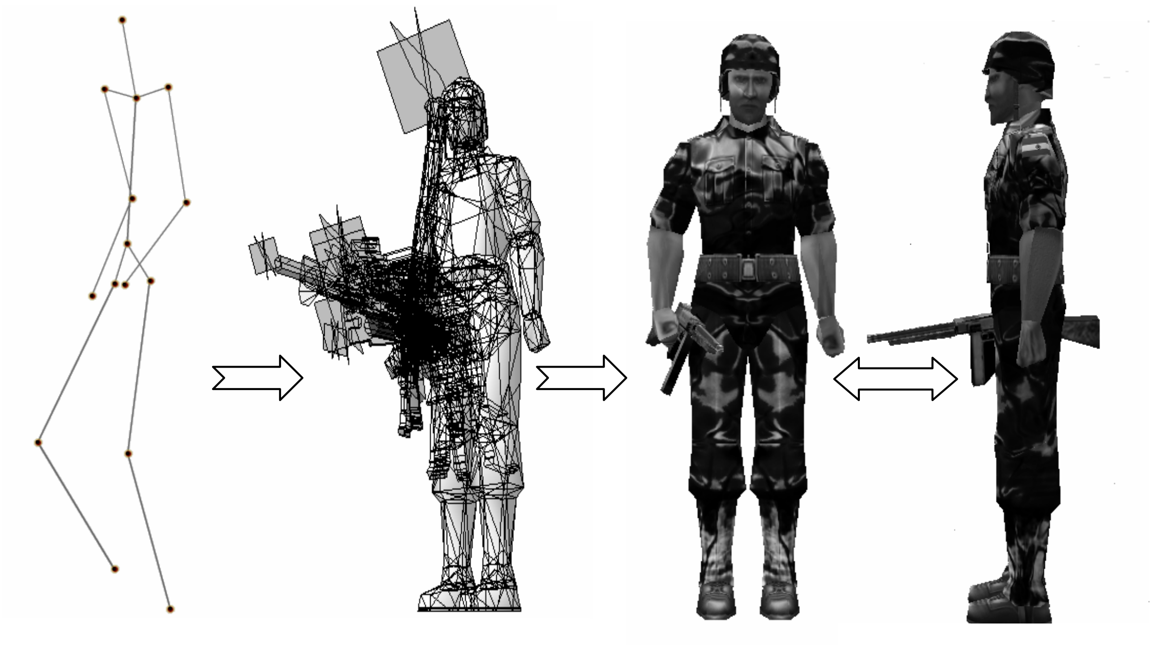


Fig 5 The human model used in the project

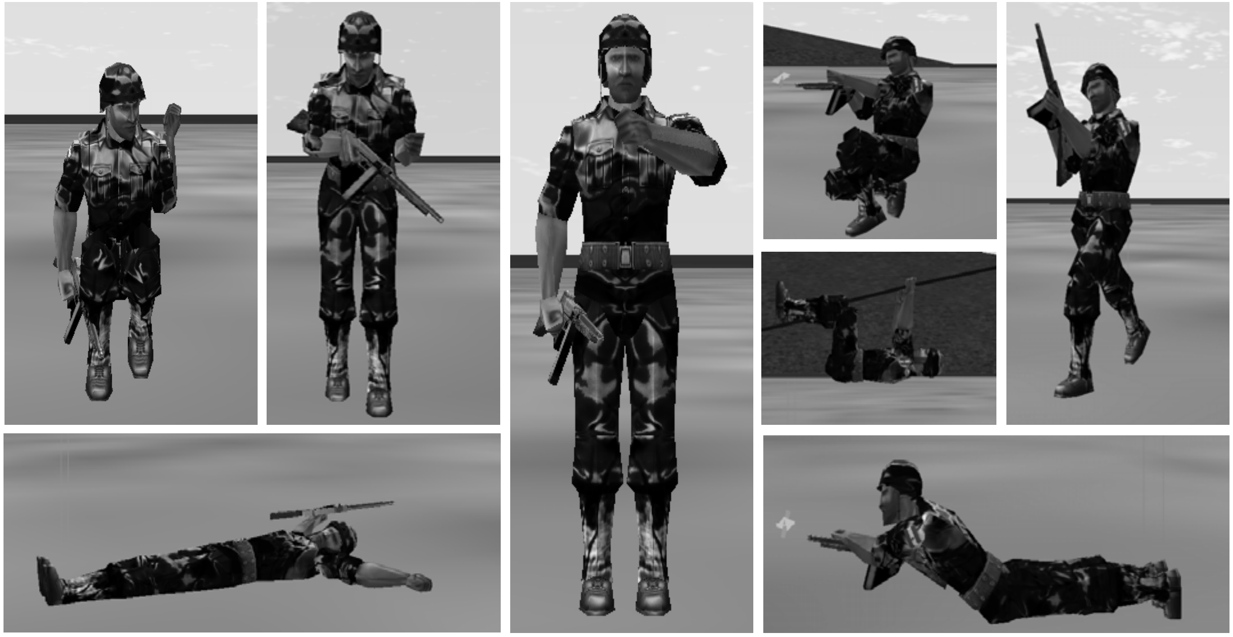


Fig 6 Some Stances in action.

11. APPLICATION

According to the Oxford dictionary:

Simulation is to Pretend to be, have, or feel; counterfeit; Reproduce conditions of (situation, etc.) e.g. for training … simulation noun; simulator noun.

Simulation is a kind of modeling which is useful for wide range of problems and situations. It has applications to both quantitative and qualitative problems with either very good data, or very little data. It has important implications for disciplines such as social anthropology which are basically non-experimental, providing a means of exploring problems which could never be observed to order. Simulation is not a panacea for all of our problems, but it can be an important tool for the social researcher aware of its limitations.

Simulations are distinguished from other kinds of models more in terms of goal than form. Simulations are typically used for problems which are seen as complex and intractable, where no direct means of evaluation is known or the conventional means of evaluation is extremely difficult to execute, or which requires interactive decisions by the investigator during the course of the model.

- For defense purpose: This can be used in Army, Navy, for different type of testing.
- For industrial setup
- In 3d games: Many 3D games can be developed with the help of simulation.
- Cost effective measure to analyse and test working conditions without involving actual conditions / weapons / machinery.
- Leads to efficient monitoring of working environment.
- Simulation helps acquaint soldiers with adverse conditions.

12. FUTURE SCOPE OF THE PROJECT

Integration of the code with other projects:

- Each new stance is a function which can be assigned any code as per the requirement.

- Each stance function can simply be copied / pasted on the codes involved in other projects.

- Basic function calls are largely the same.

- Alphabetic codes can be implemented and modified at any stage.

- There are more keystrokes available for use in the key based sequence for increased functionality at any later stage.

This code will largely be implemented by assigning a path to the soldier movement thereby made to perform various tasks.

13. CONCLUSION

The military training simulation arena is ripe with challenges. Training applications, arguably, are the primary driving force behind several of the current Department of Defense modeling and simulation initiatives. While many of the applications and concepts employed in the military training simulation domain bear little resemblance to ``traditional'' discrete event simulations and discrete event simulation concepts, a synthesis of these two cultures is likely to benefit each. We are hopeful that this tutorial is a useful contribution toward such a synthesis.

The development done with the code can result into any number of transitions from one stance to another and can be continued further to the study of the behavior of other objects such as vehicles etc. This package and code offers a unique solution to specify live human movements and is soon to be widely used in various Simulation projects. There is indeed however room for innovation and addition. A task surely a much easier one to undertake in future owing to the ground work already done.

APPENDIX - *A*

REFERENCE CO-ORDINATE AXIS

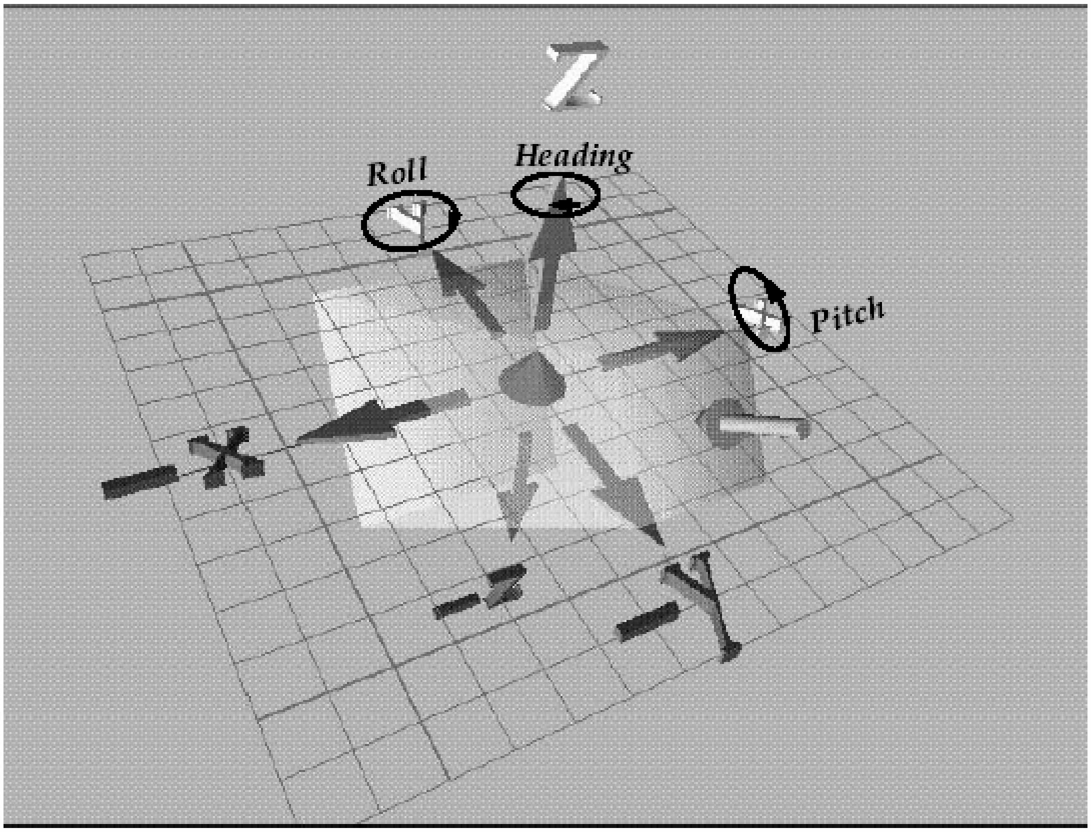


Fig. A *Reference co-ordinate axis*

For this analysis we’ll be using the following three axis or degrees of freedom---X, Y and Z; for a standing posture the nose points in the positive Y, the right shoulder towards positive X and the direction from foot to head denotes positive Z.

APPENDIX - *B*

ALGORITHMS

FOR WALK SUBROUTINE

1. Check the current position (stance) of the soldier.
2. If the soldier is not standing go to next; else to step 4
3. Bring to stand position.
4. Orient the soldier in the required direction of motion.
5. Start “Walk” sub-routine.

    5.1 The left leg & the right arm are thrust forward and left arm thrust backward simultaneously.

    5.1.1 for the left leg: lift ankle, bend knee along pitch. Simultaneously bend elbow joints and arc shoulder joints in the required direction for arms movement.

    5.2 The hip joint is rolled.

    5.3 In a similar movement of the joints the right leg and the left hand is thrust forward. However, the degree and displacement this time is doubled for each joint because all subsequent movements are double in magnitude of the first stride.

    5.4 Steps 5.1.1 and 5.2 are repeated for the same joints only with double magnitude of movement.

6. Check for any new instruction.
7. Execute any new instruction if given.
8. End WALK subroutine.

FOR STANCE TO STANCE TRANSITION

1. Check the current position (stance) of the soldier.
2. If the soldier is not standing go to next; else to step 4
3. Bring to stand position.
4. Orient the soldier in the required direction of crawl.
5. Start “Kneel” sub-routine.
   - 5.1 Bend knee in the direction of pitch and the torso along with the neck in the direction of the kneel. Simultaneously, raise the ankles.
   - 5.2 Keep the waist stationary and pitch the hip joint till the torso rests on the ankles.
6. Check whether the torso has touched the ankles.
7. If yes go to next; else go to 5.2
8. Pitch the knee joints in the direction of kneel.
9. Check whether the knee joints have touched the ground.
10. If yes go to next; else go to step 8.
11. Bend the torso forward and pitch the shoulder and elbow joints. Roll the knee joints till the elbows touch the ground.
12. Check whether the elbows have touched the ground.
13. If yes go to next; else go to step 11.
14. Start crawl sub routine.

APPENDIX - *C*

FLOW CHARTS

FOR WALK SUB ROUTINE

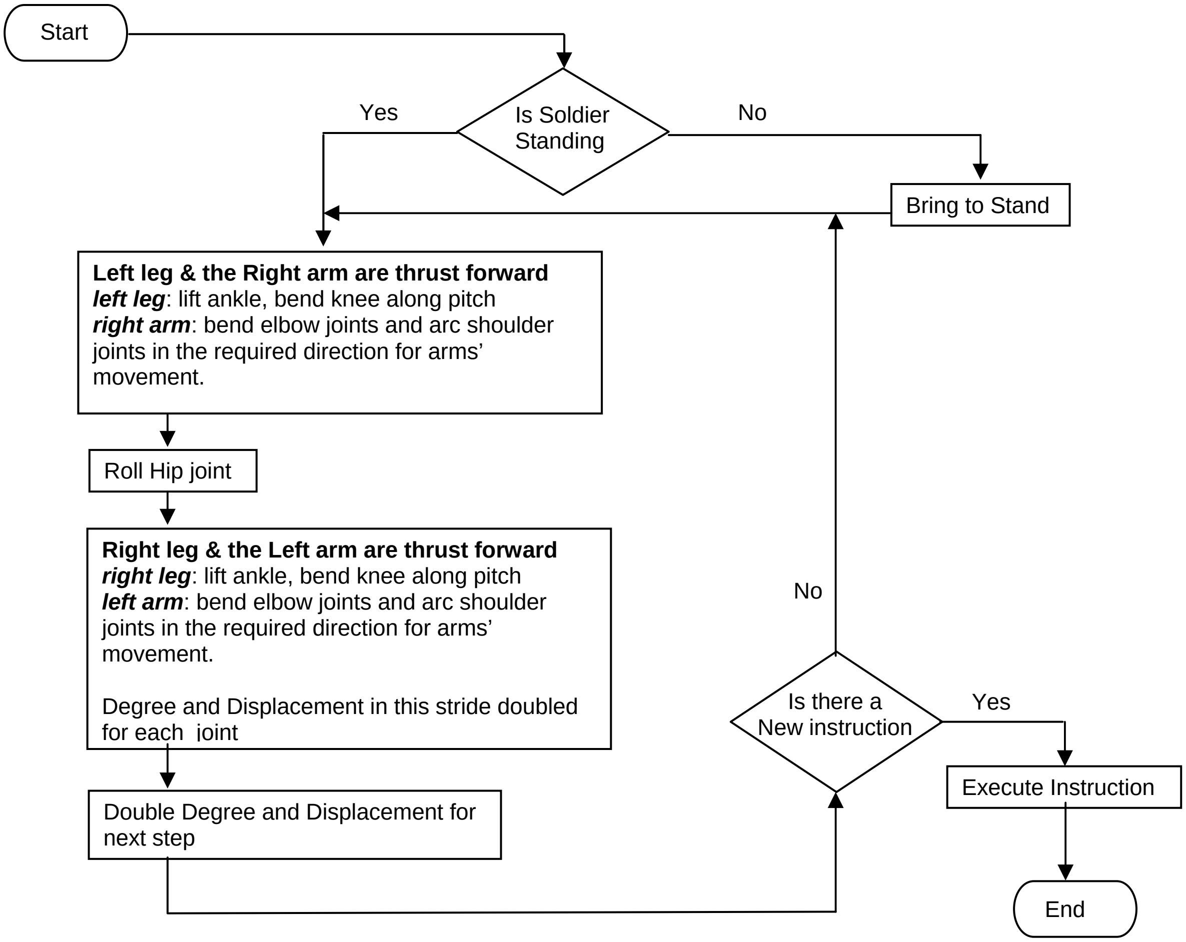

Start
Is Soldier Standing
Yes
No
Bring to Stand
Left leg & the Right arm are thrust forward
left leg: lift ankle, bend knee along pitch
right arm: bend elbow joints and arc shoulder joints in the required direction for arms' movement.
Roll Hip joint
Right leg & the Left arm are thrust forward
right leg: lift ankle, bend knee along pitch
left arm: bend elbow joints and arc shoulder joints in the required direction for arms' movement.
Degree and Displacement in this stride doubled for each joint
Double Degree and Displacement for next step
No
Is there a New instruction
Yes
Execute Instruction
End

Fig. B *Flow chart for walk subroutine*

FOR STANCE TO STANCE TRANSITION

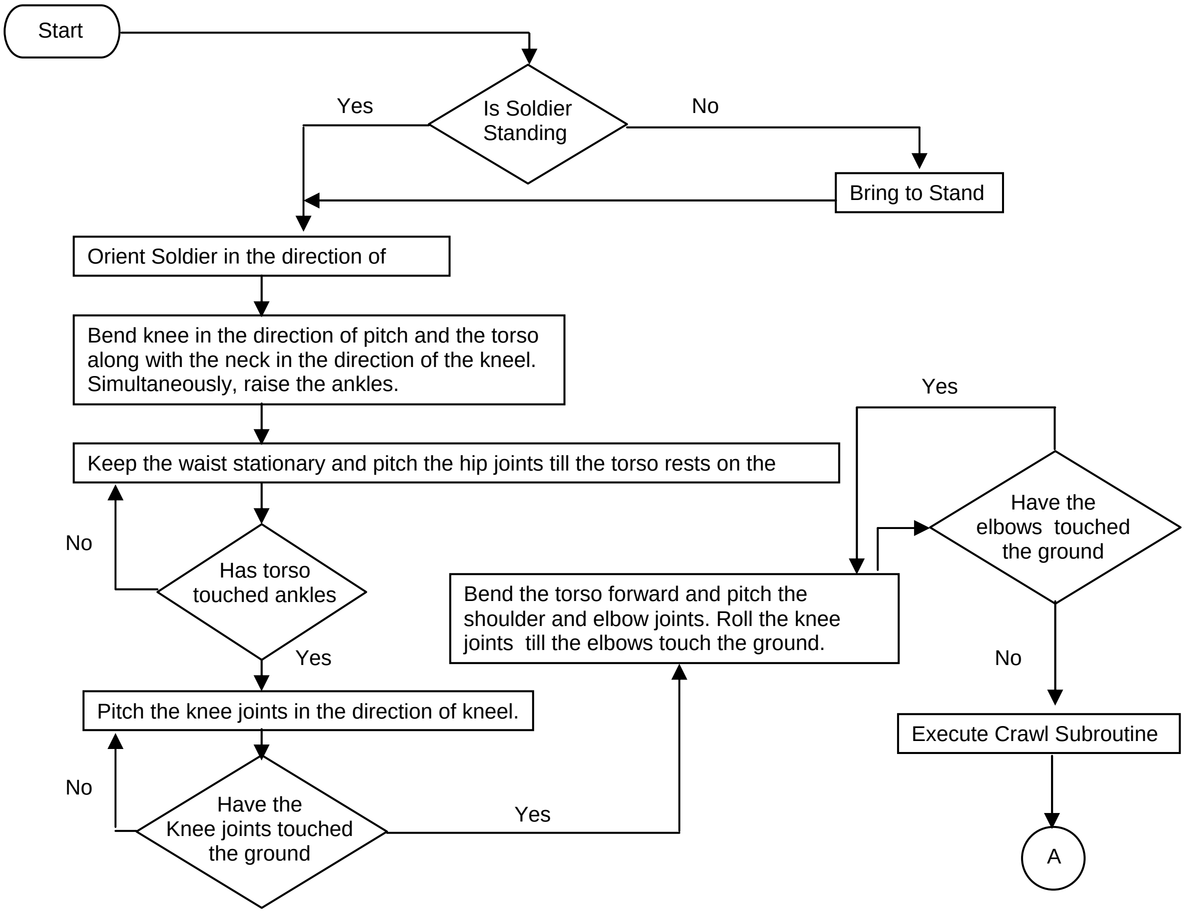

Start
Is Soldier Standing
Yes
No
Bring to Stand
Orient Soldier in the direction of
Bend knee in the direction of pitch and the torso along with the neck in the direction of the kneel. Simultaneously, raise the ankles.
Keep the waist stationary and pitch the hip joints till the torso rests on the
No
Has torso touched ankles
Yes
Pitch the knee joints in the direction of kneel.
No
Have the Knee joints touched the ground
Yes
Bend the torso forward and pitch the shoulder and elbow joints. Roll the knee joints till the elbows touch the ground.
Yes
Have the elbows touched the ground
No
Execute Crawl Subroutine
A

Fig. C *Flow chart for stance to stance transition*

APPENDIX - *D*

Basic Stances			*1)*	*Standing*	*130*
			2)	*Kneeling*	*146*
			3)	*Lying*	*111*
a)	*Idling*	*101*	*A)*	*Look watch*	*126*
b)	*Pumping Arms*	*102*	*B)*	*Walking RL*	*127*
c)	*Lobbing*	*103*	*C)*	*Horizontal Rope Up*	*132*
d)	*Walking*	*104*	*D)*	*Horizontal Rope Down*	*133*
e)	*Search OP*	*105*	*E)*	*Navigate compass*	*134*
f)	*ROP*	*106*	*F)*	*Search Binoculars*	*135*
g)	*Wading*	*107*	*G)*	*Read map*	*136*
h)	*Running*	*108*	*H)*	*Standing RL*	*137*
i)	*Falling face Down*	*109*	*I)*	*Fwd Kicking*	*138*
j)	*Fleeing*	*110*	*J)*	*Side Kicking*	*139*
k)	*Lying Rifle Firing*	*112*	*K)*	*Sitting on Chair*	*140*
l)	*Battle Crouching*	*114*	*L)*	*Sitting on Ground*	*141*
m)	*Bayoneting*	*115*	*M)*	*Standing Pistol Fire*	*142*
n)	*Back Blasting*	*116*	*N)*	*RL Firing*	*143*
o)	*Fwd Blasting*	*117*	*O)*	*Squatting*	*144*
p)	*Crawling*	*118*	*P)*	*Turning Back*	*145*
q)	*Jump Crouched*	*120*	*Q)*	*Wave*	*147*
r)	*Stand By*	*121*	*R)*	*Celebration*	*148*
s)	*Ready*	*122*	*S)*	*Read Paper*	*149*
t)	*Kneeling Fire*	*123*	*T)*	*Falling backwards*	*125*
u)	*Standing Fire*	*124*	*U)*	*Stealth*	*128*
			4)	*Stand to Kneel*	*153*
			5)	*Stand to Lie*	*154*
Transition Routines			*6)*	*Kneel to Stand*	*155*
			7)	*Kneel to Lie*	*156*
			8)	*Lie to Stand*	*157*
			9)	*Lie to Kneel*	*158*

APPENDIX - *E*

STANCE TO STANCE TRANSITION

Basic Stances

Standing

Standing	130
Lobbing	103
Walking	104
Standing RL	137
Search OP	105
Walk ROP	106
Wading	107
Running	108
Fleeing	110
Standing Fire	124
Bayoneting	115
Back Blasting	116
Fwd Blasting	117
Stand By to Jump	121
Ready to Jump	122
Standing Pistol Fire	142
Rope Movt Up	132
Rope Movt Down	133
Fwd Kicking	138
Side Kicking	139
Sit on Ground	141
Sit on Chair	140
RL Firing	143
Walking RL	127

Kneeling

Kneeling	146
Kneeling Fire	123
Battle Crouching	114
Jump Crouched	120
Squatting	144

Lying

Lying	111
Lying Rifle Firing	112
Crawling	118

Basic Transitions:

Stand to Kneel	153
Stand to Lie	154
Kneel to Stand	155
Kneel to Lie	156
Lie to Stand	157
Lie to Kneel	158

Stances where no transition is required:

Pumping Arms	102
Idling	101
Turning Back	145
Falling Face down (dying front)	109
Falling backwards (dying back)	125

Additional Stances which checks for the initial condition: Stand / Kneel / Lie

Navigate compass	134
Search Binoculars	135
Read paper	131
Stealth	128
Look watch	126
Celebration	119
Read map	136
Wave	113

Basic Stances: 3
Basic Transitions: 6
Total No. of Stances: 54

Underlined text indicates stances initiated by this team.
***Italicized text* indicates stances reinvented.**

APPENDIX - *F*

ABBREVIATIONS

DSC	Defense Simulation Center
Isector	Inter-Sector
VC++	Visual C++
.flt	Flight File Extension
GUI	Graphic-User Interface
API	Application-Program Interface

LIST OF REFERENCES

1. *Getting Started, VEGA Prime Documentation.*

2. *VEGA Prime API Documentation.*

3. *VEGA Prime Desktop Tutor.*

4. *VEGA Prime Options Guide.*

5. *VEGA Prime Programmers' Guide.*

6. *VEGA Prime Reference Guide.*

7. *MSDN Library.*

8. *Character Shop.*

10. *World Wide Web*