Implementation of Sight and Shooting Systems with Rule-Based Artificial Intelligence in a Military 3-Dimensional Environment

Firas Abdullah Thweny Al-Saedi[1], Fadi Khalid Ibrahim[2]

[1,2] *Computer Engineering Department, Al-Nahrain University, Baghdad, Iraq*

Abstract *— This paper discusses the details of the sight and shooting system used in a 3-Dimensional (3D) military training environment. These systems are used to make the soldier see and detect another soldier and shoot him. An algorithm is developed for the sight and shooting systems that checks the intersection of the sight or shooting rays against the least number of obstacles. Also, the Rule-Based Artificial Intelligence (AI) system used for the computer controlled soldiers is discussed.*

Keywords *— 3D, Sight system, Shooting system, Rule-Based AI, Military squad, Pre-calculated path.*

I. INTRODUCTION

Before moving into the subject the reader must know what is the Virtual Reality (VR), VR is a computer-simulated environment, whether that environment is a simulation of the real world or an imaginary world. Most current VR environments are primarily visual experiences, displayed either on a computer screen or through special or stereoscopic displays, but some simulations include additional sensory information, such as sound through speakers or headphones. Some advanced, haptic systems now include tactile information, generally known as force feedback, in medical and gaming applications. Users can interact with a virtual environment or a Virtual Artifact (VA) either through the use of standard input devices such as a keyboard and mouse, or through multimodal devices such as a wired glove, the Polhemus boom arm, and omni-directional treadmill. The simulated environment can be similar to the real world, for example, simulations for pilot or combat training, or it can differ significantly from reality, as in VR games. In practice, it is currently very difficult to create a high-fidelity virtual reality experience, due largely to technical limitations on processing power, image resolution and communication bandwidth. However, those limitations are expected to eventually be overcome as processor, imaging and data communication technologies become more powerful and cost-effective over time [1].

In this paper and to be cost-effective, Microsoft Visual C# 2008 [2] along with the new XNA 3.0 [3][4][5] graphics technology released by Microsoft were used, actually, the graphics technology used is games-quality, this technology was used to generate a VR environment that is used individually or through network of two computers (this can be expanded easily). Also, the input device used is either the standard keyboard and mouse or using the new Nintendo Wii Remote (Wiimote) [6][7].

In [8], a focus is shown on building a game, AI-live, that is oriented towards the intensive use of AI controlled Bots. The game borrows the idea from the popular game "The sims", but with a strong focus on building characters based on different AI techniques, one of the AI techniques is the Rule-Based AI.

In the next sections, the sight and shooting systems used in the simulation along with the AI system that controls the soldiers are discussed..

II. THE SIGHT SYSTEM

The soldiers in the simulation system need a method to be able to see each other, for example, the enemy soldiers needs a system that makes them able to recognize the soldiers controlled by the trainee and attack them on need. Also, the enemy soldiers must be able to see the other enemy soldiers to recognize if they are running or in alarm mode. For these reasons, the sight system was used. The sight system used is a ray (line of sight) that originates from the soldier's face towards any desired direction. Figure 1 illustrates the sight system used.

For the simulation system, the ray is made in a way that it scans objects in front of the soldier. The scanning range depends on the angle set by the designer in the simulation source code and can be changed by changing the value of the variable that holds that value. Also, the scanning can be stopped so that the ray stays constant and does not move. Figure 2 illustrates the scanning mechanism.

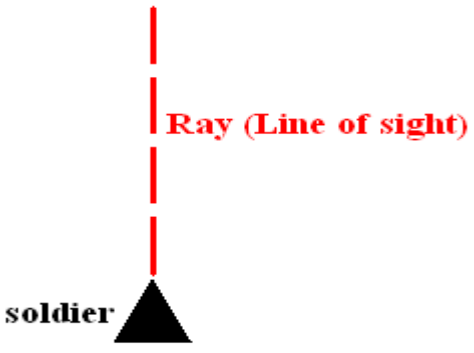

Fig. 1 Illustration of sight system

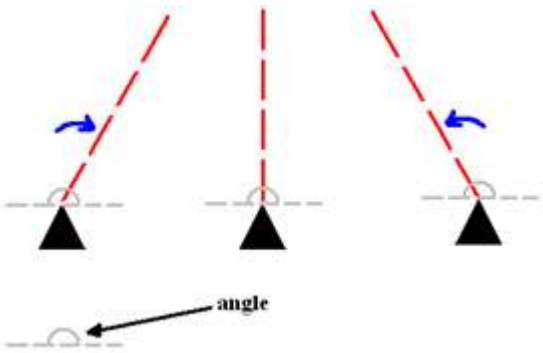

Fig. 2 Illustration of scanning mechanism

The ray is represented by position and direction, the position is a 3D coordinate (X,Y,Z) that represents the ray's origin in space, the direction also is a 3D normalized coordinate that represents the ray's direction. Figure 3 illustrates the ray representation.

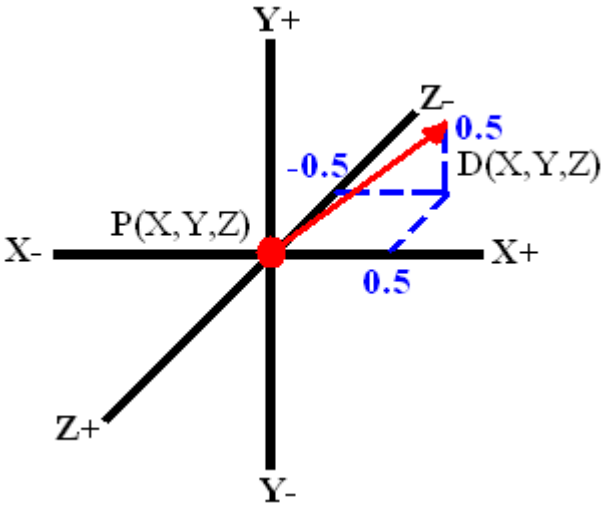

Fig. 3 Illustration of ray representation

The unity 3D axis system shown in Figure 3 shows how to represent the ray's origin and direction. For each soldier, the sight's ray position is the soldier's Y position plus the soldier's height because the ray's origin should be at soldier's head, while the soldier's sight ray direction is the soldier's direction plus minus the scanning angle offset. Figure 4 illustrates the scanning mechanism.

For each soldier there is a sight ray that keeps scanning by the specified viewing angle, the sight ray system represents the eye of the soldier.

1) Detecting Objects Using the Sight Ray

For the soldier to be able to recognize an object, its sight ray must intersect with that object's bounding box. A method called Smit's method is used to know whether a sight ray is intersecting with a bounding box, that is, the soldier is seeing the object that is bounded by that bounding box. Smit's method [9] can be represented by the pseudo code shown below:

Box: minimum extent Bl = (xl, yl, zl)
maximum extent Bh = (xh, yh, zh)
Ray: R0 = (x0, y0, z0) , Rd = (xd, yd, zd)
ray is R0 + Rdt

1. Set tnear = -INFINITY 1. , tfar = +INFINITY
2. For the pair of X planes
   1. if xd = 0, the ray is parallel to the planes so:
      - if x0 < xl or x0 > xh return FALSE (origin not between planes)
   2. else the ray is not parallel to the planes, so calculate intersection distances of planes
      - t1 = (xl - x0) / xd (time at which ray intersects minimum X plane)
      - t2 = (xh - x0) / xd (time at which ray intersects maximum X plane)
      - if t1 > t2 , swap t1 and t2
      - if t1 > tnear , set tnear = t1
      - if t2 < tfar , set tfar = t2
      - if tnear > tfar , box is missed so return FALSE
      - if tfar < 0 , box is behind ray so return FALSE
   3. Repeat step 2 for y, then z
   4. All tests were survived, so return TRUE

2) Recognizing Objects and Obstacles

The environment used has buildings and outdoor obstacles, Smit's method discussed earlier detects if an object is in the view angle range of the soldier's sight ray, it does not detect whether there is an obstacle between the soldier (observer) and the object. In this case, the soldier can not see the object because there is an obstacle ahead although the object is in the soldier's sight. To simulate this, another system was built. This system checks whether there is an obstacle between the soldier and the object. Because there is numerous buildings and obstacles and soldiers that use their sight system, this system was built in a way that it only checks the sight ray intersection against the necessary obstacles only and not against all the obstacles in the simulation environment in order not to affect the performance of the system. Each building is bounded with a bounding box, each area of building is bounded with a bounding box, each area of the building contains obstacles and each one of those obstacles is bounded with a bounding box. Figure 5 illustrates this.

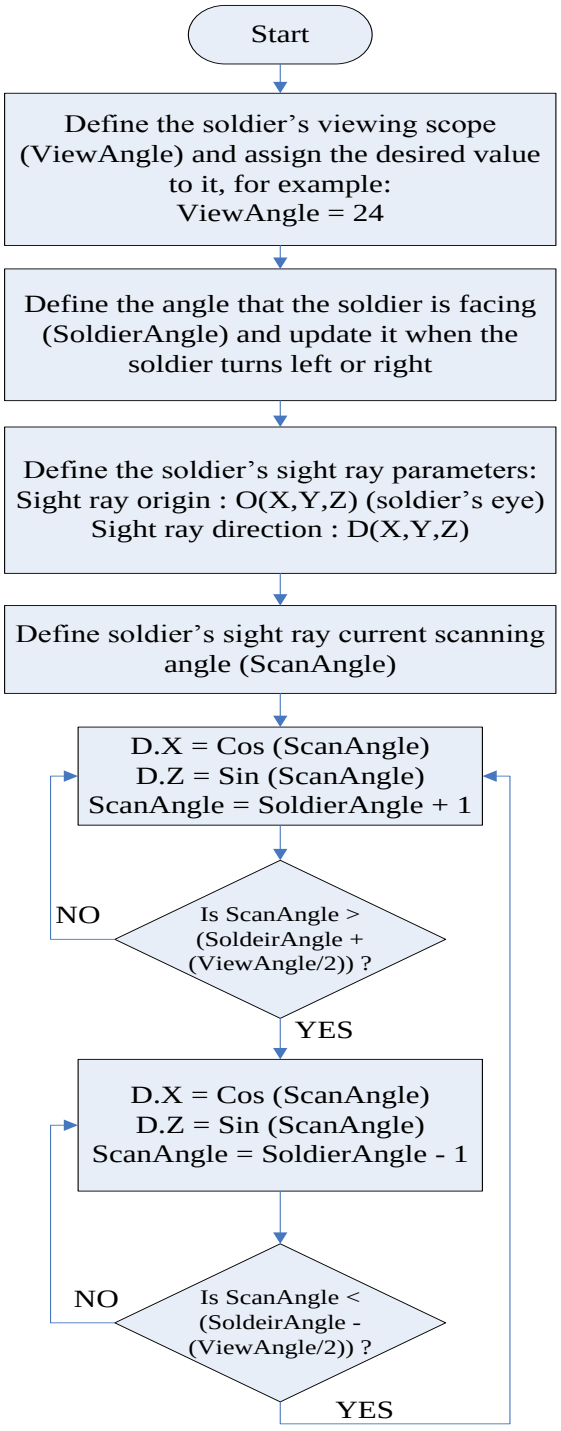

Fig. 4 Soldier's sight ray scanning mechanism

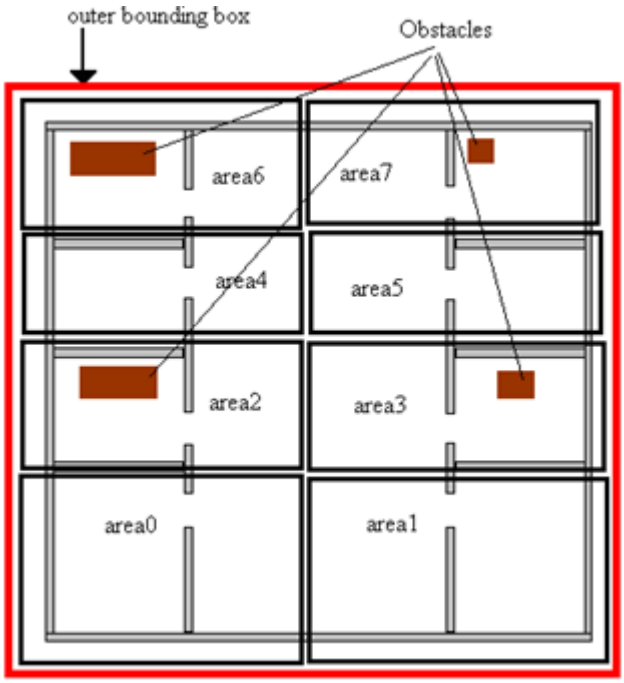

Fig. 5 Division of a building into areas

The reason of using this kind of division is that to check the sight ray intersection against a small number of obstacles each time, so for example, if the soldier is looking at the building, the sight ray intersection will be checked only against the obstacles that are in the areas of the building that the sight ray intersects with. The other part of the system is how to know if there is an obstacle between the soldier and the object. Figure 6 illustrates the problem.

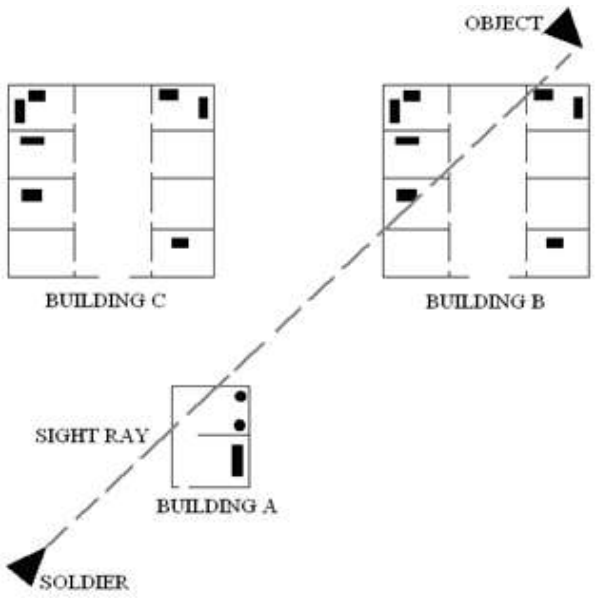

Fig. 6 Illustration of the proposed sight detection algorithm

It is assumed that each room in building B is a collision area (see Figure 5). Building A is small, so, it is only one collision area. Now, the algorithm checks to see with what buildings is the sight ray intersecting with. The sequence of checked buildings depends on the written simulation program. So may be it will check for building A firstly or building B or any building. It is assumed that it checks building A firstly, by this assumption the shortest distance returned from sight ray collision with building A obstacles will be smaller than the distance between the soldier and the object. So, the object is not seen. There is no benefit of checking another building's obstacles. If building B is checked first. Only the obstacles within the areas intersected by the sight ray are checked and by this a lot of CPU cycles are saved. And as long as the algorithm finds some obstacle that is closer to the soldier, it will not check the other buildings in the sight ray track. Figure 7 illustrates the proposed technique used to recognize objects and obstacles.

III. SHOOTING SYSTEM

The soldiers in the simulation environment use the shooting system to represent bullets, the shooting system is actually the sight system itself. This was done in order to use a ray for each soldier. When the enemy is seen by the soldier this means the soldier can shoot the enemy, so, a routine is invoked to determine the probability of shooting success, this is discussed in section V in detail. The shooting system uses the same technique used for the sight system which is shown in Figure 7. When the soldier uses the shooting system, a fire is drawn at the rifle's position and a gun-shot sound is heard

IV. FOLLOW SYSTEM

As stated earlier, the user will lead a team of two soldiers to attack a military base with soldiers. The user will control one of the soldiers in the team, the other soldier will be controlled by the computer AI. In this section, the follow system is discussed, that is, when the user gives the order to the other soldier to follow the soldier he is controlling. The user can control anyone of the two soldiers and give orders to the other soldier. The soldier controlled by the user is called "leader", the other one is called "companion". The companion has another sight ray called "FollowSightRay", the origin for this ray is the companion's head and the direction is always towards the leader's position. The purpose for using this ray is to check if the leader is in the companion's sight (no obstacles ahead), then the companion will move toward the leader directly without using path finding algorithms to find its way to the leader's position. When the leader moves, its position every five meters is recorded. When the companion is ordered to follow the leader and there is a direct line of sight between them, the companion will go toward the recorded position but stops when the distance between itself and the leader is two meters. The reason beyond using the value five meters is that to make the companion in a close distance from the leader. In this case, for every five meters the leader moves, its position is recorded and the companion will run toward that position and this keeps the companion close to the leader always. If twenty meters is used instead, then the leader will move twenty meters before the companion senses the leader's new position and runs toward it. Also, the distance between the leader and the companion is kept to be two meters and this is used as in real life when the distance between a group of soldiers should be not so close and not so far just for safety. Figure 8 shows the mechanism used to record the leader position.

Figure 9 illustrates the steps followed by the companion to follow the leader. The whole simulation environment is covered with nodes that are used by the path finding algorithms [10].

In Figure 9, the (leader position – companion position) will result a 3D vector. By normalizing this vector, the direction is found. The normalizing [11] is done by dividing each component of the 3D vector by the magnitude of the vector.

V. SOLDIER AI

The AI is required for the soldiers in the simulation so that they can interact with each other and with the user. The type of AI used is the Rule-Based AI [12] [13]. The behavior of each soldier is determined using a set of rules and for this reason this type of AI is called the Rule-Based AI. In the next two subsections, the companion soldier AI along with the enemy soldiers AI is discussed.

1) Companion Soldier AI

The user can control any soldier of the two soldier team, when the user controls a soldier, the other one will automatically be controlled by the computer AI. The leader can give orders to the companion using keyboard keys or the Wiimote. The orders given to the companion are "Fire at Will" or "Hold Fire" and "Follow Me" or "Don't Follow". Also, while the sight ray of the companion is scanning, a routine chooses a random value between 0 and 9 and the corresponding field index content of a ten elements array (probability table [12]) is returned, this array contains a group of "True" and "False" values filled by the designer in design-time. If the returned value is "True", the sight ray calculations are started and thus the routine determines whether the companion is seeing an enemy. If the returned value is "False" the sight ray calculations will not start and no action is taken by the companion. The number of "True" and "False" values in the array is a mean to determine the skill of the companion in shooting the enemies. So there is always a chance that the companion will not shoot the enemy,

this gives a realistic view to the simulation. Also, this system is used by enemy soldiers too, using this system has a benefit, that is, only some of the soldiers in each simulation frame will invoke the sight calculation routine (depending on the random number chosen). By this, the calculations are distributed between frames and the performance is perceived. Also, when anyone of the soldiers is attacked by the enemy, the attacker position will be stored so that it can be used by the soldier to defend itself.

Each soldier has a life gauge that can be changed in the design time, this gauge is used just for evaluation purposes, when the soldier's life gauge is zero, it is dead. Figure 10 shows a dead soldier. Figure 11 illustrates the companion's Rule-Based AI

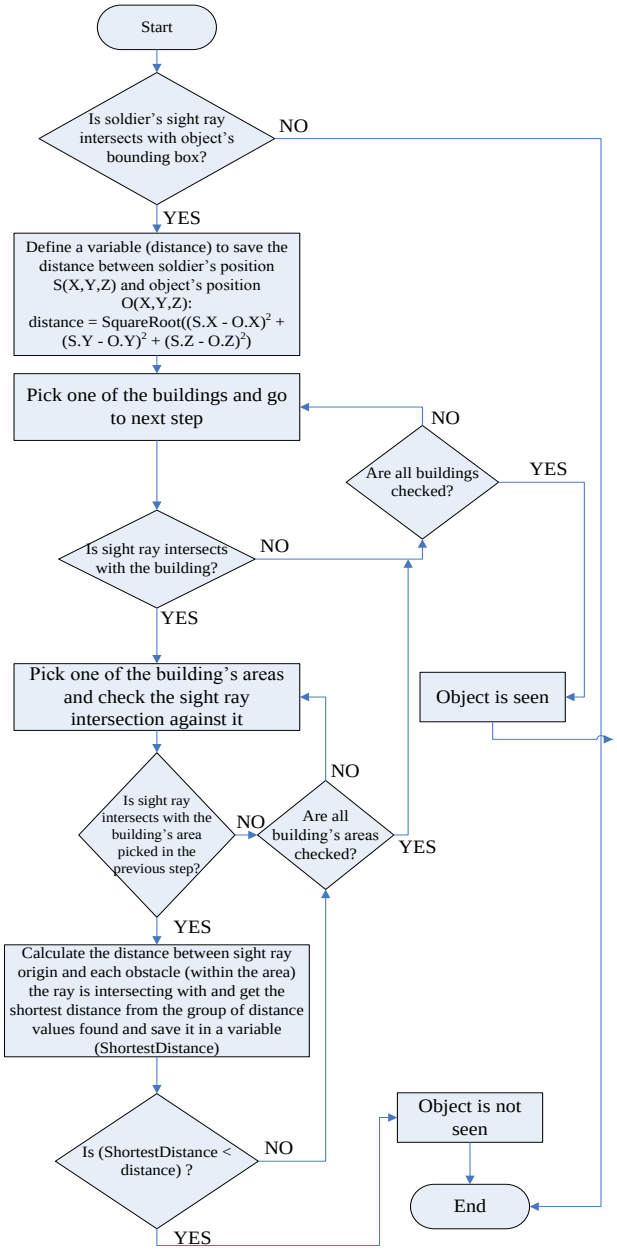

Fig. 7 Proposed technique to recognize objects and obstacles

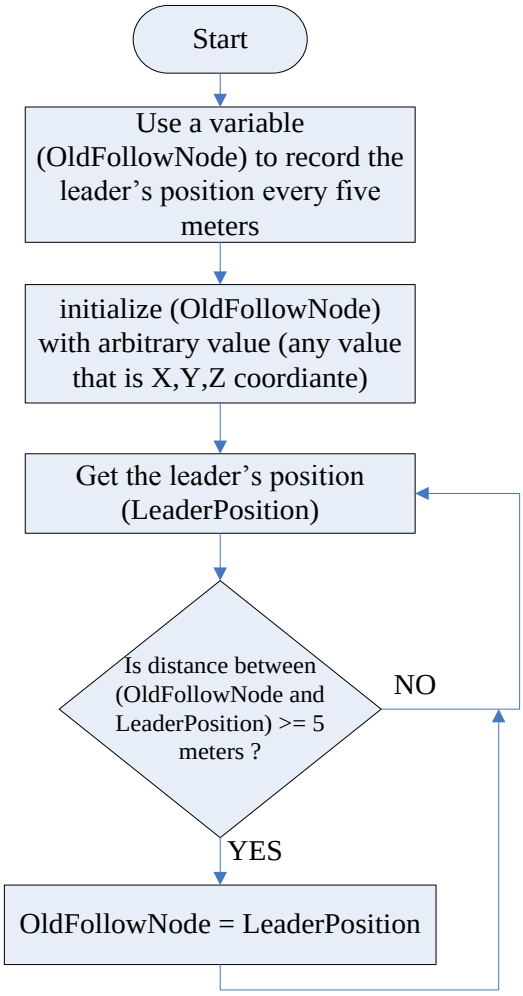

Fig. 8 Leader position recording mechanism

2) Enemy Soldiers AI

Enemy soldiers are the soldiers that guard the military base, all the soldiers have the same type of AI. That is, Rule-Based AI. Enemy soldiers AI is similar to the companion soldier's AI and it differs in only some aspects that will be discussed in the context of this section. Each enemy soldier has two modes, the first mode "Peaceful" and the second mode is "Attack". Initially, the enemy soldier mode is "Peaceful" and it has a constant path to move in, normally, this path is in some area of the simulation environment, and this is to show that the soldier is guarding or watching some part of the environment. Figure 12 shows a view of one of the simulation environments to show soldiers walking in paths to protect the environment.

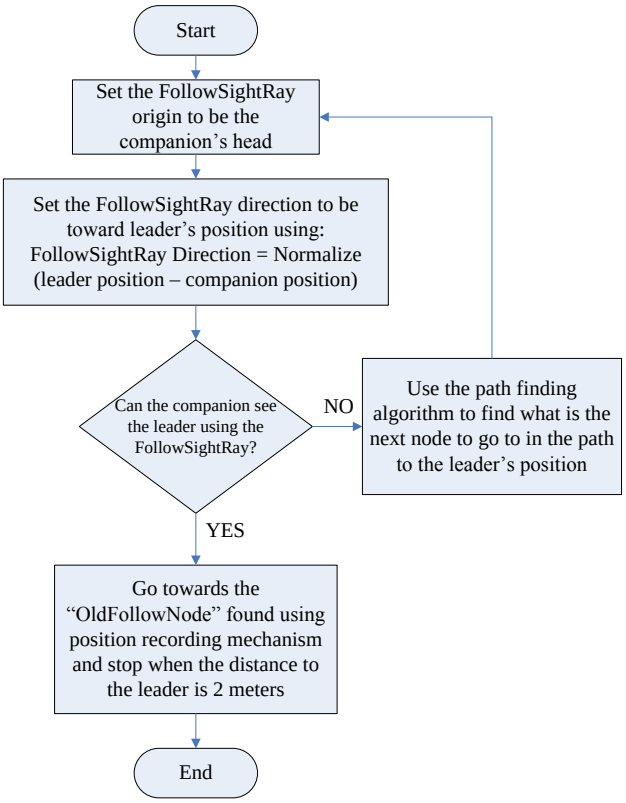

Fig. 9 Steps used by the companion to follow the leader

Fig. 10 Illustrates a dead soldier

when the enemy soldier is in "Peaceful" mode, it continues walking in its path until it sees a user soldier (leader or companion) live or dead or another enemy soldier in "Attack" mode, in this case, the enemy soldier will change its mode to "Attack" mode and begins to shoot and follow the user soldier depending on an algorithm discussed in this section. The path for each soldier is determined in design-time and followed by the soldier when it is in "Peaceful" mode. For a soldier to walk in a path, its initial position is needed, also, the angle its facing and number of steps to walk in the direction its facing is needed too. Figure 13 shows an example of an object walking in a pre-calculated path.

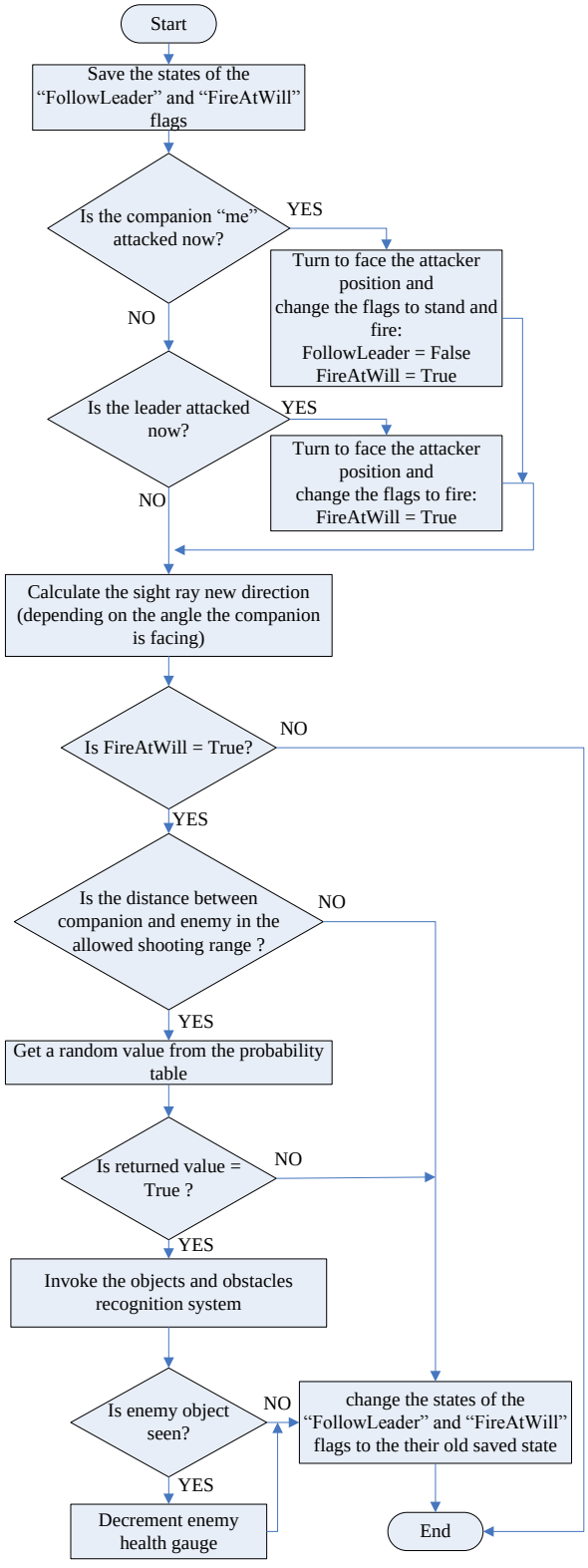

Fig. 11 Companion Rule-Based AI

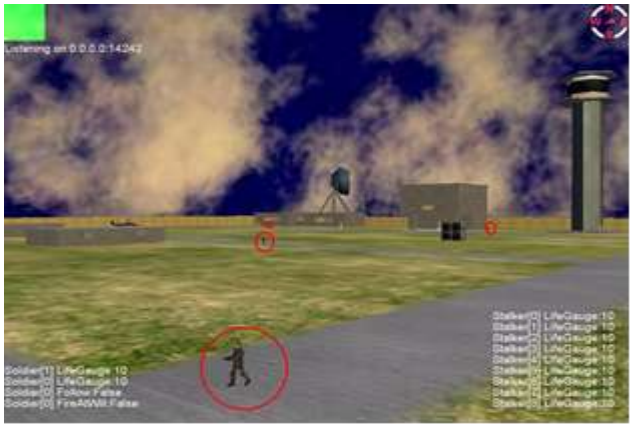

Fig. 12 Enemy soldiers walking in paths

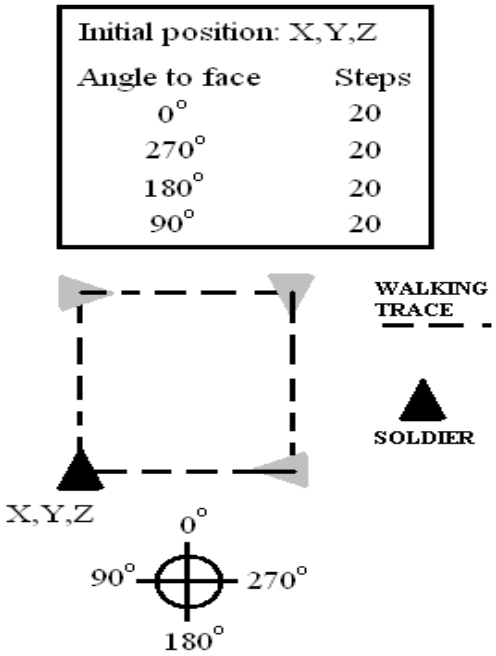

Fig. 13 Pre-calculated path demonstration

The enemy soldier always continues using the steps in Figure 14 until its life gauge is zero. The life gauge value is set in design-time and decreases by one for each shot by the user soldiers. Also, the shooting system used works in the same way as the shooting system of the companion soldier (see Figure 11).

VI. Conclusions

The subjects discussed in this paper are parts of a project that simulates a military environment. The sight and shooting systems along with the AI for the companion and enemy soldiers were discussed here. As seen in the previous section, the AI system makes use of the shooting and sight systems to determine the soldier's behaviour.

Also, the AI makes use of the path finding system discussed in [10]. When the probability table for the soldier is filled with "True", the soldier will be perfect in shooting. The purpose beyond using "False" values is that to simulate the human's imperfectness in an activity

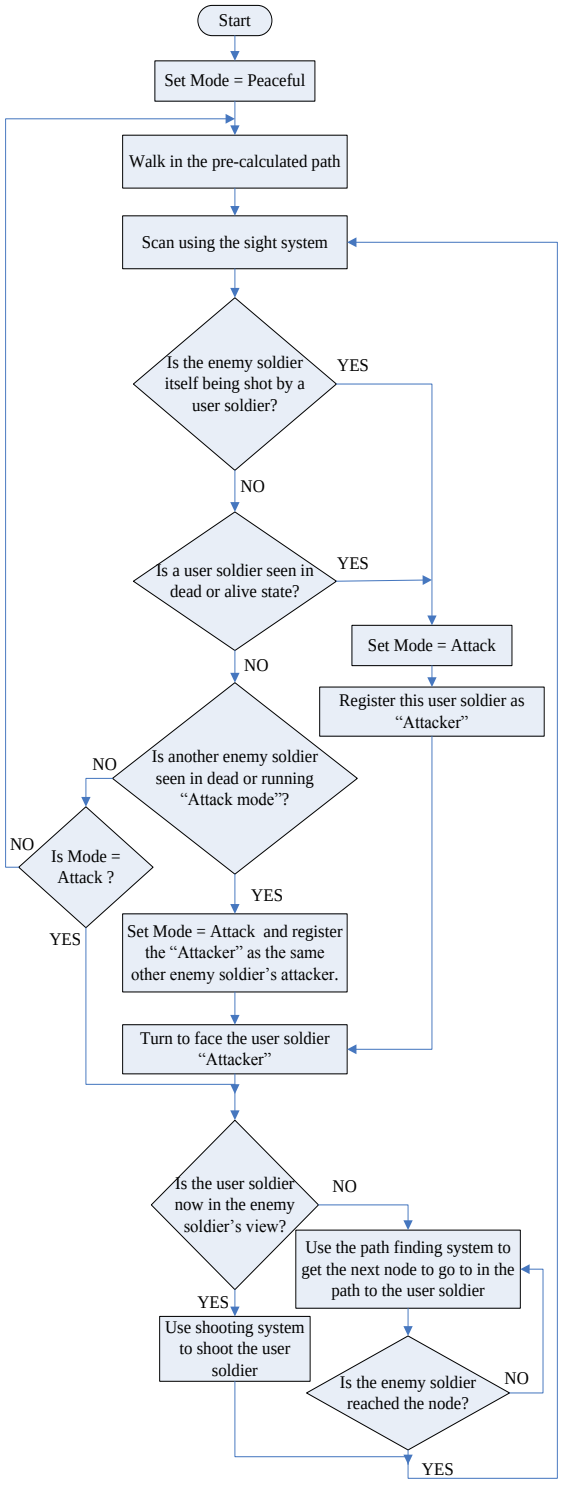

Fig. 14 Enemy soldiers AI

REFERENCES

[1] en.wikipedia.org/wiki/Virtual_reality.

[2] Rob Miles, "C# Development", Department of Computer Sciences, University of HULL, October 2008.

[3] Aaron Reed, "Learning XNA 3.0", O'Reilly Media, 2009.

[4] Chad Carter. "Microsoft XNA Unleashed: Graphics and Game programming for XBOX360 and Windows", SAMS Publishing, 2008.

[5] Reimer Grootjans, "XNA 3.0 Game Programming Recipes: A Problem-Solution Approach", Apress, March 9, 2009.

[6] en.wikipedia.org/wiki/Wii.

[7] http://www.msdn.com.

[8] Susana Fern´andez, Roberto Adarve, Miguel P´erez, Mart´ın Rybarczyk and Daniel Borrajo, "Planning for an AI based virtual agents game", 2006.

[9] Brian Smits. Efficient bounding box intersection. Ray tracing news, 15(1), 2002.

[10] Firas Abdullah Thweny, Fadi K. Ibrahim, "Implementation of Path Finding in 3-Dimensional Environment", IJCTT Journal V. 14, No. 1 August 2014.

[11] Fletcher Dunn and Ian Parberry, "3D Math Primer for Graphics and Game Development", Wrodware Publishing Inc , 2002.

[12] David M Bourg, Glenn Seemann, "AI for Game Developers", O'Reilly Media, July 2004.

[13] Ian Millington, "Artificial Intelligence for Games", Elsevier Inc., 2006.