# Military Combat Simulation: A study of Artificial Intelligence.

AiS Challenge
Final Report
April 3, 2002

Team # 044
Lovington High School

**Team Members**
Byron Buxkemper
Clayburn Griffin

**Teachers**
Pam Gray
Jimmy Crawford

**Project Mentor**
Wayne Sikes

# Table of Contents

## Executive Summary

Artificial intelligence (AI) is the concept of creating intelligent machines. Most people are familiar with AI through the use of computer games. AI goes much farther than games. It is a very complicated idea that many people wish to fully understand. Hollywood has realized the fascination of people with the subject of AI. There have been several movies questioning the possibilities of AI. Terminator is likely the most famous of them. The movie stresses the "what if" side to AI research. What if machines make humans obsolete? What if machines learn what humans don't want them to learn? What if machines can learn to kill? What if machines learn to destroy humans? Our project will answer these questions. We will create a simulation of a character trying to find a random square.

AI is known as the creation of intelligent machines. What does this mean? Intelligent machines must be programmed to learn. Several robots have been programmed to carry out a certain task. These robots require human assistance to make decisions. AI robots should be able to not only carry out tasks but also to learn the affects of choices. Computer games have been able to simulate AI in a computerized environment. Intelligent machines should also be able to learn about objects in the real world. In computer games, AI can be simulated by programming an object to "understand" the programming of another object, allowing it to know what the object can be used for. In the real world, however, objects do not have a written program that a machine can read. AI controlled machines will have to be able to understand the use and meaning of objects without the need of written information programmed into themselves.

## Process

AI is known as the creation of intelligent machines. Machines must be programmed to learn. Several robots have been programmed to carry out a certain task. These robots require human assistance to make decisions. AI robots should be able to not only carry out tasks but to learn the affects of choices. Computer games have been able to simulate AI in a computerized environment. Intelligent machines should also be able to learn about objects in the real world. In computer games, AI can be simulated by programming an object to “understand” the programming of another object, allowing it to know what the object can be used for. In the real world, however, objects do not have a written program that a machine can read. AI controlled machines will have to be able to understand the use and meaning of objects without needing written information programmed into themselves.

At the start of the year we knew little about C++. We used Visual C++ because it was easy to understand and use. The Microsoft® Visual C++ editor provided us with an useful tool for programming. At the start, we were planning on using graphics to play the game and have more than one team. The graphics are in appendix B. Since we didn’t get to that point this year we decided that it would be better to do it next year. We made a couple of trips to Albuquerque to see Wayne Sikes who helped us with all of our programming. He taught us what we needed to know about C++.

Our project will demonstrate how AI functions. We will provide a character with a computerized environment. The character will be programmed with the ability to see if a square around it is open, closed, or the winning square. The AI programmed in it will see if the square next to it is the winning square and if it is move there. If it is not the winning square then it will look around at all the  other squares and find out how many are open.  It will then add them up and divide by that number to randomly choose a spot were it will go

## Problem Statement

The purpose of this project is to make a computer game that will simulate a character running around trying to find winning a square.  The character will look around itself to see which squares are open or if it is the winning square.  It then will decide where to move next.

## Data

In the first part of our program we defined all of the variables.  Some of the variables that we had to define were the characters for the player, walls, and winning square, the map height, and width, and the delay between each move.  After defining the variables we set up an enumeration the cells around the ‘s’.  We labeled them BOUNDINGCELL_A, BOUNDINGCELL_B, etc.  Each Bounding Cell is relative to the Player’s (s) position.  For example, Bounding Cell A is located at PlayerX – 1 and PlayerY – 1.  The program checks each of the cells and marks them ‘occupied’ or ‘available’.  The ‘s’ cannot move into an occupied cell.  A cell is occupied if it contains an ‘X’ or is the last cell the player was occupying.  The program also checks for a ‘W’.  If the ‘W’ is adjacent to the ‘s’ the program ends.  The ‘s’ randomly chooses an available cell to move to.  The map is then redrawn with the new data sent to it.  The height and width never change, but the cell the ‘s’ chooses to move to will contain an ‘s’ and the old ‘s’ will be deleted.

# The Program

```
// AI Project.cpp : Defines the entry point for the console application.

#include "stdafx.h"
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <memory.h>
#include <time.h>
// prototypes
void    DrawMap(void);
void    LayoutMap(void);
void    MovePlayer(void);
#defineDISPLAY_DELAY            500
#defineMAP_WIDTH                12
#define MAP_HEIGHT              12
// map characters
#defineMAP_BORDER               'X'
#defineMAP_OPEN                 ' '
#defineMAP_WIN                  'W'
#defineMAP_PLAYER               's'
#defineUSER_EXIT_KEY            'z'
#defineBOUNDING_CELL_AVAILABLE          0
#defineBOUNDING_CELL_OCCUPIED           1
// Map[y][x]
char    Map[MAP_HEIGHT][MAP_WIDTH];
enum
        {
        BOUNDINGCELL_A = 0,         // 0
        BOUNDINGCELL_B,                     // 1
        BOUNDINGCELL_C,                     // 2

        BOUNDINGCELL_D,                     // 3
        BOUNDINGCELL_E,                     // 4
        BOUNDINGCELL_F,                     // 5
        BOUNDINGCELL_G,                     // 6
        BOUNDINGCELL_H                      // 7
        };
// player movement
//      player bounding cells as:
//              A       B       C
//              D       s       E
//              F       G       H
```

```
    //
    struct  PlayerBoundingTag
    {
            int             BoundingCell[8];
    }       PlayerBoundingTest;
    int             PlayerX, PlayerY;
    int             LastPlayerX, LastPlayerY;
    int             WinGame;
    int main(int argc, char* argv[])
    {
            unsigned int    dwCurrentTime = GetTickCount();
            unsigned int    dwNextDisplay = dwCurrentTime + DISPLAY_DELAY;
            int                             count;
            int                             GameLoopCount;
            // seed the random-number generator
            srand( (unsigned)time(NULL) );
            // INITIALIZE all data
            // clear the map
            memset(Map, 0, MAP_HEIGHT * MAP_WIDTH );
            // clear the bounding structure
            for( count = 0; count < 8; ++count )
                    PlayerBoundingTest.BoundingCell[count] =
BOUNDING_CELL_AVAILABLE;
            GameLoopCount = 0;
            WinGame = 0;
            // DESIGN AND SETUP MAP
            // draw border into map
            LayoutMap();
            // BIG GAME LOOP
            // loop until user-requested exit or until the soldier wins
            for(;;)
                    {
                    // wait until delay done - this is our display timer
                    // uncomment these lines to run in normal displayed mode
                    while( dwNextDisplay > GetTickCount() );
                    // next time to display
                    dwNextDisplay = GetTickCount() + DISPLAY_DELAY;
                    // increment number of loops counter
                    ++GameLoopCount;
                    // do the AI
                    // move the player
                    MovePlayer();
                    // draw the map
    // uncomment this to draw the map
                    DrawMap();
                    // did user press a key to exit?
```

```
                        if( kbhit() )
                                {
                                if( _getch() == USER_EXIT_KEY )
                                        break;
                                }
                        // game loop breaks if WinGame is set to 1
                        if( WinGame == 1 )
                                break;
                        }  // end of game loop
                printf("\n\n");
                printf("Took %d times to win...\n", GameLoopCount );
                printf("press any key to exit...");
                while( !kbhit() );
                return 0;
        }
        ///////////////////////////////
        //      MovePlayer()
        //
        //      AI - calculate the new player position and then
        //      move them
        //
        void    MovePlayer(void)
        {
                int             TestX, TestY, count, TotalAvailableCells, RandomCell,
MoveToCell;
                // determine which cells are available
                // A
                TestX = PlayerX - 1;
                TestY = PlayerY - 1;
                if( TestX == LastPlayerX && TestY == LastPlayerY )
                        PlayerBoundingTest.BoundingCell[BOUNDINGCELL_A] =
BOUNDING_CELL_OCCUPIED;
        else if( Map[TestY][TestX] == MAP_OPEN )
                        PlayerBoundingTest.BoundingCell[BOUNDINGCELL_A] =
BOUNDING_CELL_AVAILABLE;
                else if( Map[TestY][TestX] == MAP_WIN )
                        {
                        WinGame = 1;
                        return;
                        }
                else
                        PlayerBoundingTest.BoundingCell[BOUNDINGCELL_A] =
BOUNDING_CELL_OCCUPIED;
                // B
                TestX = PlayerX;
                TestY = PlayerY - 1;
```

```
			if( TestX == LastPlayerX && TestY == LastPlayerY )
				PlayerBoundingTest.BoundingCell[BOUNDINGCELL_B] = 
BOUNDING_CELL_OCCUPIED;
			else if( Map[TestY][TestX] == MAP_OPEN )
				PlayerBoundingTest.BoundingCell[BOUNDINGCELL_B] = 
BOUNDING_CELL_AVAILABLE;
			else if( Map[TestY][TestX] == MAP_WIN )
				{
				WinGame = 1;
				return;
				}
			else
				PlayerBoundingTest.BoundingCell[BOUNDINGCELL_B] = 
BOUNDING_CELL_OCCUPIED;
			// C
			TestX = PlayerX + 1;
			TestY = PlayerY - 1;
			if( TestX == LastPlayerX && TestY == LastPlayerY )
				PlayerBoundingTest.BoundingCell[BOUNDINGCELL_C] = 
BOUNDING_CELL_OCCUPIED;
			else if( Map[TestY][TestX] == MAP_OPEN )
				PlayerBoundingTest.BoundingCell[BOUNDINGCELL_C] = 
BOUNDING_CELL_AVAILABLE;
			else if( Map[TestY][TestX] == MAP_WIN )
				{
				WinGame = 1;
				return;
				}
			else
				PlayerBoundingTest.BoundingCell[BOUNDINGCELL_C] = 
BOUNDING_CELL_OCCUPIED;

			// D
			TestX = PlayerX - 1;
			TestY = PlayerY;
			if( TestX == LastPlayerX && TestY == LastPlayerY )
				PlayerBoundingTest.BoundingCell[BOUNDINGCELL_D] = 
BOUNDING_CELL_OCCUPIED;
			else if( Map[TestY][TestX] == MAP_OPEN )
				PlayerBoundingTest.BoundingCell[BOUNDINGCELL_D] = 
BOUNDING_CELL_AVAILABLE;
			else if( Map[TestY][TestX] == MAP_WIN )
				{
				WinGame = 1;
				return;
				}
```

```
		else
			PlayerBoundingTest.BoundingCell[BOUNDINGCELL_D] =
BOUNDING_CELL_OCCUPIED;
		// E
		TestX = PlayerX + 1;
		TestY = PlayerY;
		if( TestX == LastPlayerX && TestY == LastPlayerY )
			PlayerBoundingTest.BoundingCell[BOUNDINGCELL_E] =
BOUNDING_CELL_OCCUPIED;
		else if( Map[TestY][TestX] == MAP_OPEN )
			PlayerBoundingTest.BoundingCell[BOUNDINGCELL_E] =
BOUNDING_CELL_AVAILABLE;
		else if( Map[TestY][TestX] == MAP_WIN )
			{
			WinGame = 1;
			return;
			}
		else
			PlayerBoundingTest.BoundingCell[BOUNDINGCELL_E] =
BOUNDING_CELL_OCCUPIED;
		// F
		TestX = PlayerX - 1;
		TestY = PlayerY + 1;
		if( TestX == LastPlayerX && TestY == LastPlayerY )
			PlayerBoundingTest.BoundingCell[BOUNDINGCELL_F] =
BOUNDING_CELL_OCCUPIED;
		else if( Map[TestY][TestX] == MAP_OPEN )
			PlayerBoundingTest.BoundingCell[BOUNDINGCELL_F] =
BOUNDING_CELL_AVAILABLE;
		else if( Map[TestY][TestX] == MAP_WIN )
			{
			WinGame = 1;
			return;
			}
		else
			PlayerBoundingTest.BoundingCell[BOUNDINGCELL_F] =
BOUNDING_CELL_OCCUPIED;
		// G
		TestX = PlayerX;
		TestY = PlayerY + 1;
		if( TestX == LastPlayerX && TestY == LastPlayerY )
			PlayerBoundingTest.BoundingCell[BOUNDINGCELL_G] =
BOUNDING_CELL_OCCUPIED;
		else if( Map[TestY][TestX] == MAP_OPEN )
			PlayerBoundingTest.BoundingCell[BOUNDINGCELL_G] =
BOUNDING_CELL_AVAILABLE;
```

```
		else if( Map[TestY][TestX] == MAP_WIN )
			{
			WinGame = 1;
			return;
			}
		else
			PlayerBoundingTest.BoundingCell[BOUNDINGCELL_G] =
BOUNDING_CELL_OCCUPIED;
		// H
		TestX = PlayerX + 1;
		TestY = PlayerY + 1;
		if( TestX == LastPlayerX && TestY == LastPlayerY )
			PlayerBoundingTest.BoundingCell[BOUNDINGCELL_H] =
BOUNDING_CELL_OCCUPIED;
		else if( Map[TestY][TestX] == MAP_OPEN )
			PlayerBoundingTest.BoundingCell[BOUNDINGCELL_H] =
BOUNDING_CELL_AVAILABLE;
		else if( Map[TestY][TestX] == MAP_WIN )
			{
			WinGame = 1;
			return;
			}
		else
			PlayerBoundingTest.BoundingCell[BOUNDINGCELL_H] =
BOUNDING_CELL_OCCUPIED;
		// now, add up total number of available cells
		TotalAvailableCells = 0;
		for( count = 0; count < 8; ++count )
			{
		if( PlayerBoundingTest.BoundingCell[count] ==
	BOUNDING_CELL_AVAILABLE )
				++TotalAvailableCells;
			}
		// pick random number from available cells
		RandomCell = rand() % TotalAvailableCells;
		// decide which cell to move to
		TotalAvailableCells = 0;
		for( count = 0; count < 8; ++count )
			{
			if( PlayerBoundingTest.BoundingCell[count] ==
BOUNDING_CELL_AVAILABLE )
				{
				if( TotalAvailableCells == RandomCell )
					MoveToCell = count;
				++TotalAvailableCells;
				}
```

```
        }
// calculate which cell to move to
switch( MoveToCell )
        {
        case    BOUNDINGCELL_A:
                TestX = PlayerX - 1;
                TestY = PlayerY - 1;
                break;
        case    BOUNDINGCELL_B:
                TestX = PlayerX;
                TestY = PlayerY - 1;
                break;
        case    BOUNDINGCELL_C:
                TestX = PlayerX + 1;
                TestY = PlayerY - 1;
                break;
        case    BOUNDINGCELL_D:
                TestX = PlayerX - 1;
                TestY = PlayerY;
                break;
        case    BOUNDINGCELL_E:
                TestX = PlayerX + 1;
                TestY = PlayerY;
                break;
        case    BOUNDINGCELL_F:
                TestX = PlayerX - 1;
                TestY = PlayerY + 1;
                break;
        case    BOUNDINGCELL_G:
                TestX = PlayerX;
                TestY = PlayerY + 1;
                break;
        case    BOUNDINGCELL_H:
                TestX = PlayerX + 1;
                TestY = PlayerY + 1;
                break;
        } // End SWITCH

// move the player!
Map[TestY][TestX] = MAP_PLAYER;
Map[PlayerY][PlayerX] = MAP_OPEN;
// update the last player position
LastPlayerY = PlayerY;
LastPlayerX = PlayerX;
// move player
PlayerY = TestY;
```

```
		PlayerX = TestX;
	}
	//////////////////////////////
	//	DrawMap()
	//
	//	draws the game map on the console
	void	DrawMap(void)
	{
		int	WidthCount, HeightCount;
		for( HeightCount = 0; HeightCount < MAP_HEIGHT; ++HeightCount )
			{
			for( WidthCount = 0; WidthCount < MAP_WIDTH; ++WidthCount )
				{
				printf("%c", Map[HeightCount][WidthCount] );
				}
			printf("\n");
			}
	}
	//////////////////////////////
	//	LayoutMap()
	//
	//	draws the game map on the console
	void	LayoutMap(void)
	{
		int	WidthCount, HeightCount;
		// setup map borders
		for( HeightCount = 0; HeightCount < MAP_HEIGHT; ++HeightCount )
	{
			// draw top and bottom rows of the map
			if( HeightCount == 0 || HeightCount == (MAP_HEIGHT-1) )
				{
				for( WidthCount = 0; WidthCount < MAP_WIDTH;
++WidthCount )
					Map[HeightCount][WidthCount] = MAP_BORDER;
				}
			// get here and we're only drawing the left and right sides of the map
			else
				{
				for( WidthCount = 0; WidthCount < MAP_WIDTH;
++WidthCount )
					{
					if( WidthCount == 0 || WidthCount == (MAP_WIDTH-1) )
						Map[HeightCount][WidthCount] =
MAP_BORDER;
					else
						Map[HeightCount][WidthCount] = MAP_OPEN;
```

```
                        }
                    }
                }

            // add the AI-win cell
            int RandomX = rand() % (MAP_WIDTH-2);
            int MapBottomHalf = (MAP_HEIGHT / 2) - 1;
            int RandomY = rand() % ((MAP_HEIGHT-2)-MapBottomHalf);
            Map[RandomY+MapBottomHalf][RandomX+1] = MAP_WIN;
            // add soldier to the map
            PlayerX = LastPlayerX = 1;
            PlayerY = LastPlayerY = 1;
            Map[PlayerY][PlayerX] = MAP_PLAYER;
                              }
```

## **Future Plans**

Next year we plan to add more characters and another team. We plan to make combat simulation using Artificial Intelligence. We eventually plan to create visual graphics (see appendix B) to simulate combat. That may take two more years.

## **Conclusion**

Our project has been intensive in code. We not only learned how to code in C++ we learned that we had to make a change over to Visual C++. Our initial intentions were to program a combat simulation in top-down view. Our goal was to learn video gaming as a skill and take this project into next year doing an isometric view. However, after visiting with our mentor in Albuquerque, we soon discovered that even a 2-D animation would take far more programming knowledge and time to do than we had. Even though we spent many weeks creating graphics and characters for our military simulation video game (see appendix B) we soon discovered that this effort was going to take more coding experience than we had. After paying another visit to our mentor we decided to create a simple modified version of our game in a format that we could both understand and complete. Our program uses the beginnings of AI by having a character (‘s’) look for a clear path until it becomes adjacent to the winning square (W). When it become adjacent with the winning square the game is over and it will tell you how many moves it took to win.

## **Acknowledgements**

We would like to thank Mrs. Gray and Mr. Crawford for letting us use the lab and helping us in any way that we needed. We would also like to thank Wayne Sikes for helping us understand and develop the program.

## **Bibliography**

Bauer, Matthew; Roberge, James; Smith, George K. Engaged Learning For Programming in C++ 2nd. 2001 Jones & Bartlett Publishers.

Brown, Beth; Corica, Tim; Presley, Bruce. A guide to programming in C++. 1997 Lawrence Press.

Davis, Stephen R. C++ For Dummies 3rd Edition. 1998 IDG Books Worldwide.

# Appendices

## Appendix A: Game Sample

```
XXXXXXXXXXXX
Xs         X
X          X
X          X
X          X
X          X
X          X
X          X
X          X
X    W     X
X          X
XXXXXXXXXXXX
```

The starting position in the game.

---

```
XXXXXXXXXXXX
X          X
X  s       X
X          X
X          X
X          X
X          X
X          X
X          X
X    W     X
X          X
XXXXXXXXXXXX
```

The character starts looking for the winning square.

---

```
XXXXXXXXXXXX
X          X
X          X
X          X
X          X
X   s      X
X          X
X          X
X          X
X    W     X
X          X
XXXXXXXXXXXX
```

The Character keeps looking for the winning square.

---

```
XXXXXXXXXXXX
X          X
X          X
X          X
X          X
X          X
X          X
X    s     X
X          X
X    W     X
X          X
XXXXXXXXXXXX
```

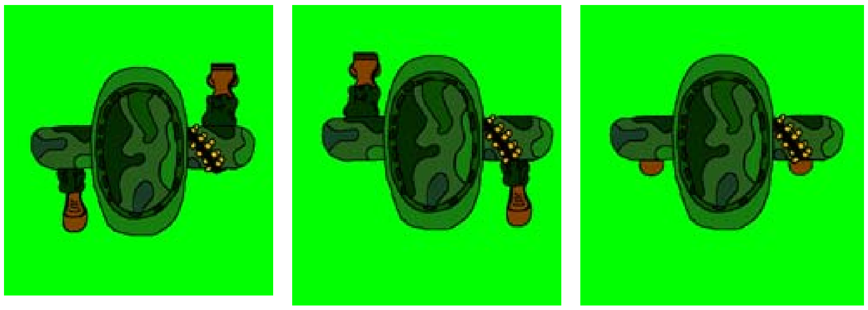
It is almost to the winning square.

---

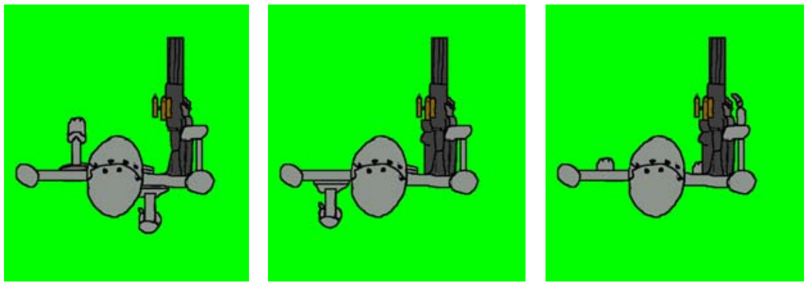
```
XXXXXXXXXXXX
X          X
X          X
X          X
X          X
X          X
X          X
X          X
X    s     X
X    W     X
X          X
XXXXXXXXXXXX

Took 29 times to win…
Press  any key to exit…
```

The character has found the winning square and the game is over.

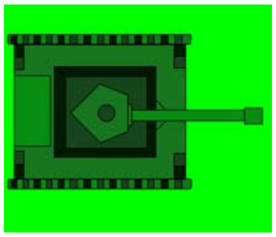
## Appendix B: Graphics



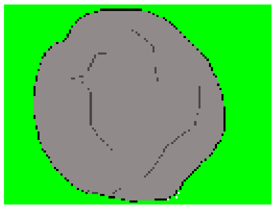
Our Man Walking.



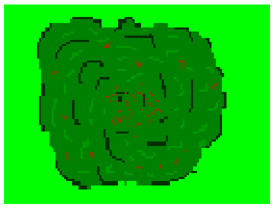
Our robot walking.



Our Tank.



A rock.



A tree.