# Decision making and Success of Military Actions

**New Mexico**
**Supercomputing Challenge**
**Final Report**
**April 1, 2009**

**Team 90**
**Rio Rancho Mid-high**

**Team members:**
**Sean Taylor**
**Ashwin Chidambaram**
**Adam Garcia**

**Teacher:**
**Mrs. Debra Loftin**

**Mentor:**
**Nick Bennett**

## Executive Summary

Our program's goal is to create tasks that will allow the most efficient movement of individuals in a military situation. This program is important because it will cause the individuals to make successful decisions. The success of this decision making program can be measured according to how many opponents a unit can defeat before he, himself, is defeated.

We are using NetLogo, an agent based modeling program. Initially, we wanted our program to be very complex, simulating a war game. As we progressed, we realized that keeping the program simple was the best way to reach our goal of creating a program where the computer makes decisions. We chose to make a game format so that it was easy to see what the results of each move would be.

After doing much programming, testing, and refining of the Artificial Intelligence Program, we successfully developed a program that can make some decisions by itself. For example, it now has the knowledge of when it can attack the opponent. Our program relates to a military situation because it uses tanks as the objects that the humans can directly interact with. The objects do not move efficiently until it starts to retreat. Then, it only moves units that it knows can get away thereby not wasting any moves.

## Introduction

A basic question that we had to ask before we could do anything was, “What is artificial intelligence?” We got our answer at the Stanford question-and-answer website. The website defined artificial intelligence as, “...the science and engineering of making intelligent machines.” With that in mind, we came up with a purpose and three basic knowledge items we needed. Those items were as follows; how game programs work, how NetLogo allows objects to move on the board and what other decision making programs do to make the computer choose the best or most successful answers. The original purpose of this program was to model military based decision making.

The first important thing to understand is how game programs work. The game we had in mind at the beginning was called “StarCraft”; a strategy RPG (Role Play Game) involving many different types of units. But the most important part of this game was how the computer went about. Meaning it did not have just one way of playing, it had two very different ways. The first was taking a slow defensive stance, and the second was to take a more fast passed, aggressive stance. The way the computer would play was chosen at random.

Second, we absolutely have to know how turtles (or agents/units) move on the screen. We learned that the turtles could only move in a straight line, not diagonal. That proved to be a secondary challenge due to the fact that our first draft program could only move forward and no other way. But we have now overcome these challenges and we have a working program.

Finally, the third important thing to understand was what the other programs do to make the computer choose the best or most successful answers. In the article published by the Dilts Strategy Group on “Success Factor Modeling” they state that, “The SMF process was conceived and based upon a set of principles and characteristics which are uniquely suited to analyze and

identify crucial patterns of values, behavior and interrelationships at the root successful individuals, teams and companies." This lead us to focus on the pattern of behavior that we wanted our tanks to have, rather than simply on each individual move. The program is a guideline for the computer and, once the programs are complete, something for the computer to learn from.

## Problem Definition

The goal of this program is to create tasks that will allow the most efficient movement of individuals in a military situation. The program will be important because it will allow us to write a program that will cause the objects to make successful decisions. The success of this decision making program can be measured according to how many opponents can be defeat before he himself is defeated.

## How the Program Works

This program allows the human player to choose if they want to play against the computer. If the human does choose to play against the computer, they can do many things including passing a turn and moving in four directions. If the player decides to play against the computer, then the computer can do the same types of moves as the person. Some examples of what the computer can ask in order to move include; "Is there an enemy with-in 6 patches of me?", "Can I move the turtle that just attacked the enemy away from danger?", and "How many moves do I have left?". The computer's units answer, or "report", the specified information and the computer does a simple "greater than less than" operation to find what if there are any units close to the enemy which can move in to attack. Next, the computer asks the unit that just attacked, "Are there more

enemies close to you?". If that unit reports “yes”, the computer moves it as far away as possible so the enemy has to use more of their moves to capture the unit that had just attacked. The Program’s design was based on the above questions about unit movement through out the game.

We also thought that military commanders might use similar principals in their plan of action for battle. Such questions as; how many kilometers until we get to our enemy? If I send a man in can he get out before he is very heavily fired on? How much time do we have before sun up/sun set? All these are relevant questions to the one man’s survival as well as his companies survival.

Here are just some of the “rules” the computer must follow:

```
to computer-rules…
   ... let threatened-turtles (my-turtles with [(any? opponents-in-
range) and (retreat-patch != nobody)])
    ifelse (any? threatened-turtles) [
     let selected-turtle min-one-of threatened-turtles [distance-
to-closest-in-range]
     ask selected-turtle [
      move-to retreat-patch
      set moves-left (moves-left - 1)
     ]
    ] [
     let selected-turtle one-of (turtles with [(color = turn) and
can-move-safely?])
     if (selected-turtle != nobody) [
```

```
      ask selected-turtle [
        let destinations safe-neighbors
        let preferred-destination (patch-at 1 0)
        let target ifelse-value (member? preferred-destination
destinations) [preferred-destination] [one-of destinations]
        move-to target
        set moves-left (moves-left - 1)
      ]
     ]
    ]
   ]
  ]
  set current-player-has-moved? true
end
```

This piece of code relates directly to two of the questions the computer must ask about movement to retreat. After the computer has finished attacking, it needs its unit back, it first asks what units are in danger, and of those, which have the best chance to get away or make the enemy use the most moves to get the retreating computer unit. Then, it asks of those units that fit that profile if they can move more to the side they came from (the left one, generally) via a "safe-neighbor". It executes and deducts one move. After all five moves are gone it changes the turn to the human opponent.

## Program Verification

The way our program relates to the real world is that it shows a demonstration of artificial intelligence. The opponent turtles (a.k.a. the computer) act on their own using AI and try to defeat their opponent. The computer currently attacks its opponent when it is within five spaces of you (if it has five moves left). The computer also knows when it can be attacked and it doesn't have enough moves to attack and tries to get out of range. If the computer is not in range of its opponent, it will choose a unit to move forward randomly. This is like artificial intelligence because it has the computer making decisions on its own.

## Conclusion

In conclusion, our project was supposed to simulate basic military action but it ended up simulating some artificial actions instead. Making artificial decisions was still a main goal in the beginning of the program. However, we also wanted to make a learning program that gets progressively harder to beat, which we hope to do at a later date. We were not able to achieve this goal because NetLogo lacked the types of commands necessary to make the computer learn. Alternatively, we found that we could give the computer the ability to move not only in a straight line, but also in a L-shape that appears diagonal on the computer screen. Our best achievement was that we got the computer to play by it self and make its own decisions.

## Bibliography

1) "Basic Questions." Formal Reasoning Group. Stanford. 31 Mar. 2009 <http://www-

formal.stanford.edu/jmc/whatisai/node1.html>.

2) http://www.acm.org/crossroads/xrds2-2/ethics.html -“Wiley InterScience”

3) http://www3.interscience.wiley.com/journal/119212583/abstract?CRETRY=1&SRETRY=0 - “Computer Based Decisions-Making: Three Maxims”, by Jeff Robbins

4) "Tic-Tac-Toe Strategy." Ostermiller.org. Ed. Stephen Ostermiller. 17 Mar. 2009 <http://ostermiller.org/tictactoeexpert.html>.

5) Harris, Robert. "Introduction to Decision Making." VirtualSalt. 2 July 1998. 17 Mar. 2009 <http://www.virtualsalt.com/crebook5.htm>.

# Appendix A

```
globals [
  current-turtle
  current-player-has-moved?
  move-limit-per-turn
  moves-this-turn
]


turtles-own [
   health
   opponents-in-range
  ]
```

```
patches-own [
    threats
  ]

to setup
  clear-all
  create-turtles 10 [
    set color red set shape "soldier"
    set heading 90
    setxy -10 (who - 5)
  ]
  create-turtles 10 [
    set color green set shape "tank 1"
    set heading 270
    setxy 10 (who - 15)
  ]
  set current-player-has-moved? true
  set current-turtle nobody
  set move-limit-per-turn 5
  setup-grid
end
to move
```

```
  if ((not any? turtles with [color = red]) or (not any? turtles
with [color = green])) [
    stop
    reset-current-turtle
  ]
  if (current-player-has-moved?) [
    set current-player-has-moved? false
    reset-current-turtle
    set moves-this-turn 0
    ifelse (turn = red) [
      set turn green
    ] [
      set turn red
    ]
  ]
  ifelse ((turn = green) or (not computer-plays?))
  [detect-click]
  [computer-rules]
end
to pass
  set current-player-has-moved? true
end
```

```
to detect-click
  if (mouse-down?) [
    while [mouse-down?] []
    if (mouse-inside?) [
      let test-turtle (one-of turtles-on (patch mouse-xcor
mouse-ycor))
      if (test-turtle != nobody) [
        if (([color] of test-turtle) = turn) [
          reset-current-turtle
          set current-turtle test-turtle
          ask test-turtle [
            set size 2
          ]
        ]
      ]
    ]
  ]
end
to move-turtle [direction]
  if (current-turtle != nobody) [
    ask current-turtle [
```

```
  let current-heading heading
  set heading direction
  if (can-move? 1) [
   let turtle-ahead (one-of turtles-on patch-ahead 1)
   ifelse (turtle-ahead != nobody) [
     ifelse (([color] of turtle-ahead) != color) [
       ask turtle-ahead [
         die
       ]
       forward 1
       set moves-this-turn (moves-this-turn + 1)
     ] [
       set heading current-heading
     ]
   ] [
     forward 1
     set moves-this-turn (moves-this-turn + 1)
   ]
  ]
]
if (moves-this-turn >= move-limit-per-turn) [
```

```
      set current-player-has-moved? true
    ]
  ]
end
to reset-current-turtle
  if (current-turtle != nobody) [
    ask current-turtle [
      set size 1
    ]
    set current-turtle nobody
  ]
end
to setup-grid
 ask patches [
  ifelse (((pxcor + pycor )mod 2 ) = 0)
    [set pcolor white] [
      set pcolor black
    ]
 ]
end
to computer-rules
 let moves-left 5
```

```
 let my-turtles (turtles with [color = turn])
 while [moves-left > 0] [
  ask my-turtles [
   set opponents-in-range in-range (ifelse-value (color = red)
[green] [red]) moves-left
  ]
  ifelse (any? my-turtles with [any? opponents-in-range]) [
   let selected-turtle min-one-of my-turtles [distance-to-
closest-in-range]
   ask selected-turtle [
    let distance-moved distance-to-closest-in-range
    let target min-one-of opponents-in-range
     [manhattan-distance myself]
    move-to target
    ask target [
     die
    ]
    set moves-left (moves-left - distance-moved)
   ]
  ] [
   ask my-turtles [
    set opponents-in-range in-range (ifelse-value (color = red)
```

```
[green] [red]) 5
    ]
    let threatened-turtles (my-turtles with [(any? opponents-in-
range) and (retreat-patch != nobody)])
    ifelse (any? threatened-turtles) [
     let selected-turtle min-one-of threatened-turtles [distance-
to-closest-in-range]
     ask selected-turtle [
      move-to retreat-patch
      set moves-left (moves-left - 1)
     ]
    ] [
     let selected-turtle one-of (turtles with [(color = turn) and
can-move-safely?])
     if (selected-turtle != nobody) [
      ask selected-turtle [
       let destinations safe-neighbors
       let preferred-destination (patch-at 1 0)
       let target ifelse-value (member? preferred-destination
destinations) [preferred-destination] [one-of destinations]
       move-to target
       set moves-left (moves-left - 1)
```

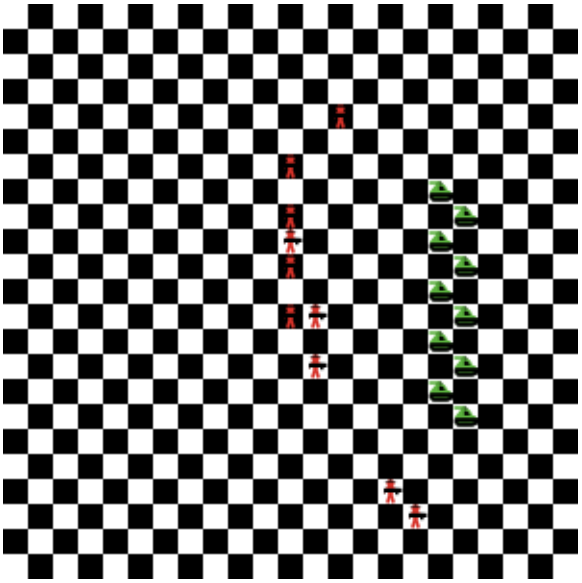
```
      ]
     ]
    ]
   ]
 ]
 set current-player-has-moved? true
end
to-report in-range [target-color steps-left]
 report (turtles with [(color = target-color)
  and ((manhattan-distance myself) <= steps-left)])
end
to-report distance-to-closest-in-range
 let result 999999
 if (any? opponents-in-range) [
  set result min [manhattan-distance myself] of opponents-in-
range
 ]
 report result
end
to-report manhattan-distance [target]
 report (abs (pxcor - [pxcor] of target)) + (abs (pycor -
[pycor] of target))
```

```
end
to-report retreat-patch
 let target nobody
 let avalabul-neighbors neighbors4 with [not any? turtles-here]
 if (any? avalabul-neighbors) [
  ask avalabul-neighbors [
   set threats [opponents-in-range] of myself
  ]
  let destinations avalabul-neighbors with [(min [manhattan-
distance myself] of threats) > [distance-to-closest-in-range] of
myself]
  if (any? destinations) [
   set target one-of destinations
  ]
 ]
 report target
end
to-report can-move-ahead?
 let turtles-ahead (turtles-on patch-ahead 1)
 let turtle-ahead (ifelse-value (any? turtles-ahead) [one-of
turtles-ahead] [nobody])
 report (turtle-ahead = nobody) or (([color] of turtle-ahead) !=
```

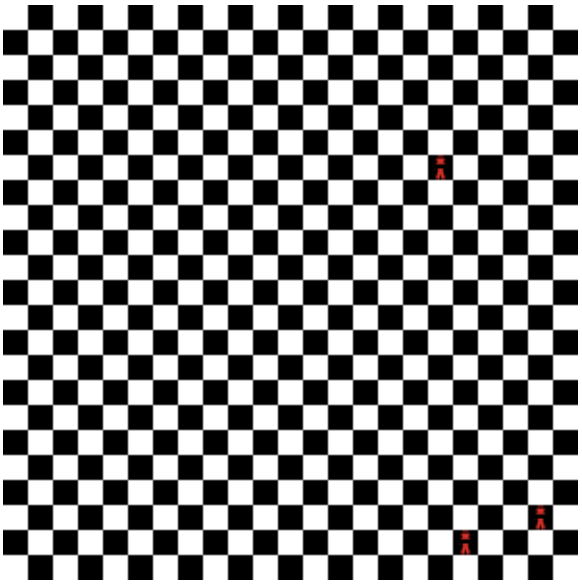
```
color)
end
to-report can-move-safely?
 report (any? safe-neighbors)
end
to-report safe-neighbors
 report (neighbors4 with [(not any? turtles-here) and (not any?
in-range (ifelse-value (([color] of myself) = red) [green]
[red]) 5)])
end
```

# Appendix B

This is what a on going battle between the computer and a human could look like.



Green (the human) as a tight formation, which will be their down fall…



Red (the computer) has beat green showing that the program can make decisions.

# Acknowledgments

We would like to thank Mrs. Debbra Loften for helping us from the beginning to the end. Second we want to thank Nick Bennitt for helping us to learn the NetLogo language. Last but not least we want to than our family and friends for their support.