**CS 370 Summer 2023 Research Journal**

**Project Overview:**

The NEAT (NeuroEvolution Augmenting Topologies) algorithm is used to evolve a neural network to play a simple video/board game (somewhat similar to chess or checkers) in which two opponents face off and attempt to eliminate all of their opponents’ pieces. The implementation of the algorithm itself is provided by the [Python-NEAT](https://neat-python.readthedocs.io/en/latest/) library, but the game and network evaluation is developed by myself. Note that the game itself evolved over the course of the project (In two stages: simple, then expanded), as complexity was added later in order to evaluate the network’s ability to learn strategies that are emergent from more complex game rules. Link to the [Github](https://github.com/kevinriek/kriek-cs370) page.

**Simple Game overview**:

Two players play against each other, each taking turns. There is a YxY board of tiles (ex: 5x5, 7x7), and each player is given the same number of pieces. Each turn a player moves each of their pieces. Each piece can be moved a maximum number of tiles (i.e. max move=3 : the piece can be moved 3 tiles). If an opponent’s piece is within the player’s unit’s movement range, the player’s unit can attack, reducing the opponent unit’s hp (health point) value. When that value reaches 0, the piece is removed from the board and can no longer be moved. Only 1 piece from either side can occupy a tile at a time.

**Simple Map overview**:

8x8 board: Player 0’s pieces are represented by 0’s, Player 1’s by 1’s. The possible move positions of the piece highlighted in blue are shown as highlighted blue tiles.

A number of numbers in a row

Description automatically generated

**Simple Scenario Rules**:

* Attacks deal 50 dmg (out of 100 hp, two attacks to eliminate a piece)
* 8x8 board
* 5 units per side
* Semi-random positions on either side (spaced evenly, 0 or 1 tiles from width-wise edge of board)
* Unit max movement = 3
* 15-50 games played per evaluated genome (changes over time)
* Max of 8 turns per game to avoid games that never terminate
* 25 games played vs. Testing script for evaluation
* Initial Move-picker testing:
* ‘Easier’ set of rules to facilitate easier training
  + 5x5 board
  + 3 units per side
  + Attacks deal 100 dmg (1 attack to eliminate a piece)

**Expanded Game overview**:

The map is grown in size in order to make the distribution of units across the board more impactful. Attacks that ‘flank’ the opponent will deal triple damage, but the attacker now also suffers some damage for every non-ranged attack. The damage dealt to the attacker and the defender is proportional to the ratio between the attacker’s effective attack value, and the defender’s effective defense value (accounting for terrain modifiers). Each piece has a unit type (infantry, archer, or cavalry) that determine their attack, defense, movement, and range values. For pieces with a range of >0 (archers), attacks can be made from a distance, rather than requiring movement adjacent to the opponent’s piece. Each tile has a terrain (plains, hills, or forest) that impacts the range and defense values of the unit occupying the tile.

**Expanded Map Overview:**

15x15 board: Player 0’s pieces are represented by lower-case letters, while Player 1’s pieces are upper-case. Cavalry are represented by ‘c’, infantry by ‘i’, and archers by ‘a’. Plains tiles are black, Hills are yellow, and Forests are green. The distribution of tiles’ terrains is symmetrical across both sides.

A pattern of black and green lines

Description automatically generated with medium confidence

**Expanded Game Concepts**:

* Idea: To add more complex rules in order to make the simulated game closer to an actual video or board game, and to investigate the network’s ability to learn complicated and emergent behaviors (i.e., emergent from the game mechanics)
  + Terrain
    - Each tile has a terrain type: Plains (Black), Hills (Yellow), or Forest (Green)
    - Hills provide a defense modifiers (x2) and additional range to archers, but also have a higher movement cost (2)
    - Forests provide a negative defense modifier (x0.5) and a negative range bonus to archers (-2), as well as a higher movement cost (3)
  + Concentration
    - The map is significantly larger (8x8 -> 15x15), with more units (5 -> 12).
    - In theory, the side that groups their units closer together will have an advantage.
  + Unit Types
    - Archers: Weaker defense, but can perform ranged attacks, meaning they do not have to be adjacent to an enemy piece to attack, only within the archer’s current range, which is buy default 2.
    - Infantry: Standard defense, attack, and movement
    - Cavalry: Weaker defense, but stronger attack and double movement.

**Training Configuration**:

* Note: training varied heavily over the course of time and different approaches, but here are mostly standard approaches:
* 150 genomes per population (50, 100 also sometimes used for time purposes, but 150 gave the best training performance)
* Generations: initially 20-100, after interval training adopted, intervals would finish after fitness reached 1.0, or max generations for an interval was reached: this would usually be 30 or 50.
* 1 hidden layer

**Testing Configuration:**

* At the end of each generation, the most fit genome from that generation is evaluated against the evaluation script, and against the ‘hall of fame’, i.e. the best genomes from previous iterations. 100 games are played against the script and each hall of fame member, with the same map layouts (unit distributions) used for each set of 100 games. The win rate against the script or the genome is recorded, as well as expanded metrics, as outlined below.

**Plotted Metrics:**

* Best genome fitness vs. generation
* Average genome fitness vs. generation
* Win rate performance of best genome vs. evaluation script in separate testing game layouts
* Win rate performance of best genome vs. ‘hall of fame’: best genomes from previous intervals, in the same testing game layouts as the script evaluation.

**Expanded Metrics:**

* Note: these extra metrics are intended to measure the network’s ability to learn more complicated / emergent behaviors. They are calculated during the evaluation / testing games, and only for the genome being evaluated.
  + Mean distance: a measure of the concentration of the network’s units, it measures the mean of the mean distance from each unit to every other unit, across all the turns played in a game, and all games played. A lower value means that the network is grouping its units closer together.
  + Proportion Forest: the proportion of all of the network’s units that are on forest tiles, averaged across all turns in all games. Subdivided by archers, infantry, and cavalry.
  + Proportion Hills: Exactly like proportion forest, but for hills.
  + Mean Location: the average location of all of the network’s units as a 2-coordinate tuple, averaged across all turns in all games played. Subdivided by archers, infantry, and cavalry.

**Evaluation (Test) Script**:

* Iterate over all units:
  + Generate unit’s moves
* Move evaluation function: assign weights to move
  + +weight for being close to enemies
  + Extremely high +weight for attacking
  + Smaller +weight for being close to allies
    - Eventually removed for larger board, as script pieces would simply cluster in the middle of the board
* Choose move with highest value
  + Deterministic, no randomness currently
* Semi-Random Test script:
  + Variation of the Test script which uses the weights assigned to each move as a weight for weighted random choice (e.g. higher weight = higher chance of being selected)
  + Developed to provide an opponent that was more difficult than random choice, but not as difficult as the script, in order to provide a bridge into training against the script.

**Fitness Function**:

* Standard: win rate (proportion of wins compared to games played)
  + Elimination method:
    - A Win means that all of the opponents’ pieces are eliminated
    - Concern: No feedback:
      * Not an issue for a small board with a lower number of pieces, but precluded any feedback on the expanded map with more pieces
  + Greater-Than method:
    - Win determined by side with more pieces
    - Quicker feedback, but less potentially need to finish off all of opponents’ pieces
    - Used as the evaluation method for the expanded game, as feedback is much easier to obtain
    - However, incentivizes defensive strategies as hunting down opponents’ pieces is not required
* Alternative: Proportion of opponent units destroyed
  + Idea: fitness is always given for destroying opponents’ units, while the basic goal remains the same: to eliminate the most opponents’ units as possible
  + Issue: Unable to achieve 1.0 winrate vs. semi-random script. Only reached ~0.7 evaluation winrate vs. script when trained directly against script, as opposed to ~0.9 achieved by Greater-Than winrate.

**Initial Self-Play**:

* Each genome’s fitness is evaluated by playing against the best genome(s) from the past X generations.
* Currently: 1 (only the best genome from the last generation is evaluated)
* Decent results from this method, but “spiking” effect: rather than continuous improvement, the population seems to spike between better and worse performance against the script, potentially even ‘going in circles’
  + Attempted solution: best genome from past generation is always given 0.5 fitness, with the idea that if a new genome in this generation failed to outperform it, then we should try again: however, this simply caused stagnation.
* Problem:
  + Say we have a high-performing generation, followed by an underperforming generation. The next generation will be evolved from the underperforming generation, not the better generation.

**Improved Self-Play (Iterative Self-play)**

* The network is trained against different targets (with all genome fitness evaluated by win rate against the target over the fixed set of board positions) over several different intervals, the idea being that the fitness evaluation becomes more difficult over time, forcing the network to improve over each iteration.
* During an interval, the target that is trained against is held constant. The interval finishes when the maximum number of generations per interval is reached, or the trained network reaches a win rate threshold (currently 1.0)
* After an iteration, the best genome in the iteration is used as the population for the next generation, and said best genome is evaluated against the best genomes of all previous iterations.
* The first interval is against random AI, no AI, or the random-script AI. The second is against the script-AI, and afterwards succeeding generations are trained against the best genome of the preceding interval.

**Hall-of-Fame / Ensemble Self-Play (Iterative Self-play)**

* The network is first trained against the initial random interval, and then the script interval.
* After the first two initial intervals, the network is trained against the script and the Hall of Fame
* The “Hall of Fame” or ‘ensemble’ is made up of the best genome from each previous interval, excluding the initial random phase interval.
* Each genome plays an equal amount of games against each network in the ‘hall of fame’ plus the evaluation script. The genome’s fitness is computed as its total winrate against all of the opponents in the hall of fame.
  + Note: the number of training games is kept constant over all intervals, meaning that as the size of the hall of fame increases, the number of games played against any 1 network (or the script) decreases. (For a low number of intervals, this should not be a major issue, as the total number of training games is 50)

**Initial Random Phase**

* A phase at the start of training, in which the first generations are evaluated against a script that randomly chooses moves. This phase is to allow the network to learn basic behaviors such as attacking, as simply starting the training against the script often prevents the network from getting any feedback at all, as the most successful feedback methods involve winning, and a completely new network will almost never win against the evaluation script.
* Two variations:
  + Fully random phase: used initially
  + Semi-random phase: uses the semi-random script as described above. Used for the expanded game rules, as the fully random phase was too easy (almost always over in a single generation), and did not actually force the network to learn anything. The semi-random phase thus provides a short, but winnable training phase before the standard script phase is started.

**Script Phase**:

* The best genome from the initial random phase is trained against the evaluation script directly, with win rate against the script used as fitness.
* The best genome from this phase will be used as the first network in the hall of fame.

**Multiprocessing**:

* Using python’s multiprocessing (not multithreading) module, genome fitness calculations (i.e game simulations) are divided via multiprocessing amongst the specified number of child processes, as this is the main bottleneck in training time.
* Each process simulates all of the games and calculates the win rate of a genome, and then returns the win rate to the main process.
* Note that the testing evaluation games are also multi-threaded, but this time individual games are split amongst the child processes, rather than genomes (as there is only 1 genome being evaluated.)

**Different Network Approaches**:

* **Move evaluation function**: For each unit, the list of all possible moves the unit can make is generated, and then the network is fed in information about the state of the game after the move is taken. For instance, if the move is an attack, the network is fed information about the game after the attack, including the new positions of the attacking piece and the new health of the attacked piece. The output of the network is a single value, representing the evaluation of the board position. The move with the highest evaluation is taken.
  + **Global positions**: For each piece on both sides, network is given the input (pos y, pos x, hp). The pos x of the unit is = (the unit’s x position / the x dimension of the board - 1)
    - Example input:

A number of numbers in a row

Description automatically generated

* + - The input for the piece in blue would be:

(y: 5/7, x: 1/7, 100/100), i.e.

(0.714, 0.143, 1.0)

This is repeated, in the same order, for each piece on the board

* + - Note: Opponent net has inverted inputs = (1 - x), so inputs look the same from either side
* A graph of a graph

  Description automatically generatedA graph of a graph

  Description automatically generatedA graph of a graph

  Description automatically generated
  + **Relative positions**: For each evaluated move, the network is instead fed information about the positions of the other pieces *in relation* to the new position of the moved piece, but not fed the new position of the unit itself.
    - Example input:

A number in a box

Description automatically generated

* + - Evaluating the highlighted move of the piece in blue (i.e, 2, 1),

the position of the opponent piece (1 at 2, 2) would be fed as:

(y: 2/3 – 1/3, x: 2/3 – 2/3, 100/100), i.e.

(0.33, 0.0, 1.0)

This is repeated, in the same order, for each piece on the board that is not the moved unit.

* + - Note: Opponent net has inverted inputs = (1 - x), so inputs look the same from either side
    - NOTE: the positions of other units are currently ordered only by the original order of the pieces (i.e. initialization order), but should be ordered, perhaps by something like distance to the new move.
      * This updated version has not been tested (or implemented)
* **Move picker**: For each tile on the board, the network is fed information of the form (is Friendly unit, is Opponent piece, piece’s HP). So the input is dimension-x \* dimension-y \* 3. The output is of the size dimension-x \* dimension-y, where each output is the network’s rating of that move, with output x representing tile (x/dimension-y, x % dimension-y,). The *legal* move with the highest rating is chosen.
  + Global
    - Example Input:

A number in a box

Description automatically generated

* + - The network is given 3 input points for each [ ]

(0, 0) has input (1, 0, 1.0)

(1, 0) has input (0, 0, 0)

(2, 2) has input (0, 1, 0)

* + - Issue with input
  + Performance issues:
    - Is tested with training against script directly, but with a modified board scenario
      * When playing against (7x7, 5 units), no feedback was obtained using the elimination (destroy all opponent units) method , i.e. 0.0 fitness for 10ish generations
        + More testing is needed with a variety of scenarios and feedback methods
      * When playing with a smaller board, was able to achieve (>0.7 win rate) performance against script, albeit not stably.
* A graph of a graph

  Description automatically generatedA graph of a graph

  Description automatically generatedA graph of a graph

  Description automatically generated
  + Relative: not yet implemented, the idea is that the outputs represent moves in relation to the unit’s current position, not the actual positions of the moves. The input would then also be in relation to the unit’s current position.

**Comments on Network Approaches:**

* While relative positions move evaluation was initially favored, I eventually chose global positions move evaluation for the expanded game testing.
* This is mostly due to ease of implementation, but also because global positions made more sense to me as a ‘global’ look at the board (similar to how a human would play) as opposed to evaluating moves only in relation to other individual pieces.
* While the ‘Move picker’ was interesting and showed potential in limited testing, its main issue was its scalability. I believe the size of the network, and more importantly, the time to learn the mappings to outputs, would have easily ballooned out of control after the size of the board increased past 8x8. It still could have been used in a modified form, but I did not spend the implementation time.

**Final Network inputs:**

* The final network has 168 inputs, 1 hidden layer, and one output.
* For each unit on the board, there are 7 inputs:
  + Global x position
  + Global y position
  + HP (after the simulated combat is over if this move is an attack)
  + Unit’s attack
  + Unit’s defense, modified by terrain
  + Unit’s movement
  + Unit’s range, modified by terrain
* All friendly units’ inputs are added in order, then the opponents’ units. All inputs are added in the order that the units are created. So the first unit in the list of Team 0’s units has its input entered into the network first, then the second unit’s, and so on.
* When a network is controlled player 1, rather than player 0, (i.e. the pieces on the right side), the position inputs are reverted, team 1’s units are input first into the network, then team 0’s, and lastly the order in which the list of units is input is reversed. This is to achieve total symmetry of the inputs into the network across the side controlled, so that regardless of the side of the network operates, its moves will be the same.

**Project Timeline:**

**6/1:**

* First meeting, wrote project outline

**6/5:**

* Downloaded NEAT and ran simple example, revised project outline

**6/12:**

* Developed Game Rules / Scenario, testing for simple simulated games

**6/19:**

* Initial basic testing with NEAT

**6/26:**

* Initial testing with NEAT and simulated game rules

**7/3:**

* Started relative-position move-eval development and testing

**7/10:**

* Continue relative-position move-eval testing

**7/18:**

* Developed basic evaluation opponent script.
* Developed new reporters to track and plot genome fitness over generations, and each generation’s best performance against script.
* Trained directly against script, using single board (starting position) layout
  + Achieved high performance using relative-position move eval
* Developed Global-position move eval
* Saved genome checkpoints to file
* Standard interface for AI modules, and for unit starting layouts generation

**7/19:**

* Self-play Testing with Global-position move eval
* A graph of a graph

  Description automatically generatedA graph with a line

  Description automatically generated
* A graph of a graph

  Description automatically generated
* Inverted inputs for opponent during self-play (so inputs are the same for both sides)
* Generates random board layouts for each game played, rather than a single position
* Developed Move-Choice algorithm, basic testing
* A graph of a line

  Description automatically generatedA graph with blue lines

  Description automatically generatedA graph with blue lines

  Description automatically generated
* Experimented with Fitness feedback methods:
  + elimination, greater number of pieces, difference

**7/20:**

* Save performance plots to file**, s**ave best genome and config to file
* More self-play testing with Global-position move eval
* Fixed self-play: now the highest fitness genome from the last generation, not all-time, is used as the opponent network.
  + Experimented with opponent networks from several past generations.
* Switched to fixed-position games, in which a fixed number of starting board positions are generated at the start (along with the random seed), and then the same positions are used throughout all played games
  + For each position, an equal number of games are played with the ‘player’ network moving first, and with the opponent network moving first, to negative first-move advantage / disadvantage.

**7/21:**

* Switched back to self-play against only the previous generation best
  + 250-population testing for global move eval
    - Significantly longer training times
    - Run only ~10 generations, no marked improvement.
  + Self-play testing for move-picker:
    - 8x8, 5 pieces
    - Very long training times, but getting some fitness score.
    - However, 0.0 performance against script

**7/25:**

* Implemented Interval Self-Play
* A graph of a graph of a number of people

  Description automatically generatedA graph of blue lines

  Description automatically generated
* A graph of a line

  Description automatically generated
* Started Training with Interval self-play,
  + Gap between training (fitness) and testing (script evaluation) win rates (say 0.45 vs. 0.65), determined this was due to limited number of board scenarios played (5x2)
  + realized that training was taking too long (25x3 generation, 100 population, 20 games)
* Running Global move eval using the Most-Pieces-Wins fitness function:
* Random seed 1648, 150 generations, 40 total eval games, 40 total training games, max generations 500, generation threshold 1.0
* A graph of a line

  Description automatically generatedA line graph with numbers and a blue line

  Description automatically generated

**7/26:**

* Started implementation of multithreading to speed up training
  + Using python threading library -> dead end

**7/27:**

* Implemented multiprocessing to speed up training
  + Significant results, able to increase generations, population size (x2 or x3), and games played (x4)
* Used multiprocessing to evaluate interval self-play
  + Results: limited
  + Major stagnation, using stuck ~0.7 win rate vs. script, which decreases after generations
  + Only tested global eval func
* Implementation of basic flanking
  + Realistically, would never apply because only 1 movement’s path to an attack is considered. Only used if the attacker is directly behind the defender when the turn starts.

**7/28:**

* Started evaluation of move-picker with interval self-play
* A graph of a number of people

  Description automatically generatedA graph of lines with different colored lines

  Description automatically generated

**7/31:**

* Fixed script bug (using list of units, not list of live units)
* Testing:
  + Config: survival threshold (0.2 vs. 0.05)
    - Temporarily 0.05, was using 0.025 (can try lower than 0.05)
  + Fitness functions (elimination vs. more units)
    - Both about equivalent vs. script (shorter training for more units vs. random)
    - More units significant when doing self-play, as the network adopts a defensive strategy vs. the script
  + After config changes, able to hit >0.9 fitness vs. script, testing vs. script always seems to hit about 0.8 win rate
* Multithreaded post-generation evaluation, fixing evaluation
* Note: each child process gets copy of input variable (1 copy per process, not per function call)

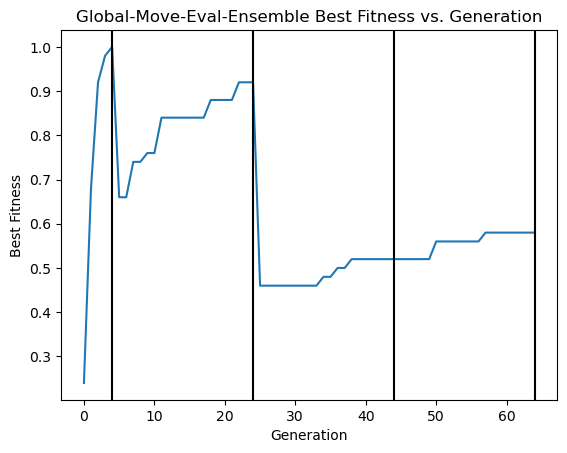
**8/1:**

* Training observation:
  + Using improved self-play, in which old population of each interval is retained, but new fitness function is created using the old interval’s best genome, each interval after the initial vs. random and vs. script genomes stagnate.
    - NOTE: elimination vs. greater than fitness really matters here!!!! Because of the defensive strategy taken vs. the script, subsequent runs using have a couple of pieces on each side left at the end
  + Possible Responses:
    - Note that by submitting a previous run’s best\_genomes list (modified in run), as the best genomes list of the new run will automatically start training against the best genomes of the past run.
  + Runs with elimination fitness:

A graph of a graph

Description automatically generated A graph of a graph

Description automatically generated

A graph of different colored lines

Description automatically generated

* + Runs using the previous game’s networks for training:

A graph of a graph

Description automatically generated A graph of different colored lines

Description automatically generated

* + Runs using greater than fitness:

A graph of a graph of a number of people

Description automatically generated A graph of different colored lines

Description automatically generated

A graph of a graph of a number of people

Description automatically generatedA graph of different colored lines

Description automatically generated

* Expanding game mechanics:
  + Terrain + Expanded Map w/ terrain
  + Flanking
  + Ranged Combat
  + Unit Types
* Adding input to network
  + Option 1: stats: att, defs, range, movement
  + Option 2: discrete categories: infantry, cavalry, archer

**8/3:**

* Training with expanded game rules:

1. Expanded Map: 8x8 -> 15x15, more units, 5 -> 11
   1. Note: removed scripts move towards ally weight, as this resulted in clustering at the center of the map.

A screenshot of a game

Description automatically generated

* A graph of a graph

  Description automatically generated with medium confidenceA graph with a line drawn on it

  Description automatically generated
  + Notice the extremely limited gap between fitness and win rate (around 0.01)
    - 25x2 training games, 50x2 testing games
    - Fitness function: greater than

**8/4:**

* Training with expanded game rules:

1. New fitness function: average proportion of enemies destroyed: (enemies eliminated / total starting number of enemies)
   1. Had to disable the initial random phase, was taking too long (~500 secs per generation), and could not reach 1.0 fitness after ~10 generations
   2. A graph of a graph

      Description automatically generated with medium confidenceA graph with blue lines

      Description automatically generated
   3. Note: fitness is not winrate here
2. Training with Flanking
   * Greater than fitness, with flanking
   * First phase is weighted random.
   * Playing 20 training games:

A graph with a line

Description automatically generatedA graph with a line

Description automatically generated

* + Playing 50 training games:

A graph of a graph with numbers and lines

Description automatically generated with medium confidenceA graph with a line

Description automatically generated

1. Training with +defender damage
   * Greater than fitness
   * First phase is weighted random.
   * Playing 50 training games:
2. Training with terrain
   * Greater than fitness
   * First phase is weighted random.
   * Playing 50 training games:

A graph of a graph showing the number of people in the same direction

Description automatically generated with medium confidence A graph of a graph

Description automatically generated

**8/5:**

1. Training with terrain and unit types
   * Greater than fitness
   * First phase is weighted random.
   * Playing 50 training games:
   * A graph of a graph of a graph

     Description automatically generated with medium confidence A graph of a graph with blue lines

     Description automatically generated
   * A screen shot of a computer screen

     Description automatically generated
   * With terrain and unit types:
   * A graph of blue lines

     Description automatically generatedA graph of a line graph

     Description automatically generated with medium confidenceA graph of a graph

     Description automatically generated with medium confidence

Observations from training results:

* + 1st network:
    - Overall ‘defensive’ strategy
    - Targets lower defense units (units that can be 1-shot)
    - Some usage of central hills, mostly corner-camping
    - Attack enemies when in range, but some forward probing
  + 2nd network:
    - More use of hills
      * Archers avoid woods, concentrate on hills
    - Concentration of cavalry in woods:
      * NOTE: could simply be an offshoot of strategy of camping in top-left corner, but
      * Advantageous because of high cost to move into woods, and high attack of cavalry?
      * Interesting because initially woods was thought of to be a bad place to stay, because of lower defense modifier.
    - Overall higher concentration of army
    - Overall defensive strategy
      * Some forays towards enemy
      * Units from opposing side trickle in (1st network)
      * Vs. script: effective use of archers, including on hills
        + Interesting because the range modifier of hills was actually not included into network inputs by mistake (defense modifier of hills was included in hills, however)
  + 3rd network:
    - Archers: shooting closer targets (2 archers on one target)
    - Significant reliance on 2 archers on hills
  + 4th network:
    - Vs. scripts: archers retreating away from opponent infantry (onto hills) instead of simply attacking
  + Overall:
    - at least initially, training vs. script has significant distortive effects (defensive focus)
    - Interesting: different behavior based on side
      * Weird because map and inputs should be perfectly symmetrical
      * Possible bug here
      * Network on right side (with inverted inputs) has different behavior
        + More infantry on hills
        + Advances more often
    - Note: some ties between older networks

**8/9:**

* Fixed Unit ratios
* Added terrain modifiers for range into inputs
* Can now move through and shoot through allies, correctly
* Remove attacker damage from ranged attacks

**8/10:**

* Try training without script or semi-random script
* Fixed opponent units’ inputs
* Fixed symmetrical input for team 1
* Training results:
* **A graph of a graph

  Description automatically generated with medium confidenceA graph of a graph showing the same number of people

  Description automatically generated with medium confidence**
* **A graph of a graph of a game

  Description automatically generated with medium confidence**
* Note: I unfortunately lost the expanded metrics data for this run, so I cannot plot it here
  + However, many of the behaviors first seen in the previous run were confirmed with statistics
    - Infantry and archers use hills and avoid forests
    - Cavalry use forests disproportionately

**Potential Next Steps**

* Best Move first principle: the network evaluates all moves available to all of its pieces collectively and picks the best move from all moves across all pieces, then moves there and re-evaluates the move set, and picks the next best move, and so on.
* Hidden Information / Fog of War: The player/ network can only see enemy pieces that are close to their own pieces (within some distance). This forces the network to operate without perfect information on all its opponent’s pieces, and better models a real video game.
* Other network modes: re-implement and try the relative-position move evaluation method, as well as a modified version of the move choice method.
* Try other algorithms or variations of the NEAT algorithm. MM-NEAT is especially interesting to me, as the modules-based approach might be very helpful in separating different tasks in gameplay.
* Tree-Search: implement minimax with alpha-beta pruning, using the network as the heuristic that evaluates different moves, in order to allow the network to ‘think’ several turns ahead, rather than simply evaluating the next move.
* Developing a more complex, smarter script-based AI, that takes in more game features into account, for better evaluation of the network’s performance.
* Remove the script from the training process, (i.e. remove the script and semi-random script phases)

**Project Goals**

* This project was successful in developing a NEAT-based AI to play a mock video game, with some complex aspects, to a reasonable degree of proficiency.
* The network was able to learn to beat the evaluation script in up to 100% of games played.
* The network was able to learn some complex and emergent behaviors, such as:
  + Developing a strategy (in this case, defending its side of the board)
  + Utilizing terrain and unit types effectively
    - Utilizing the defensive and range bonuses of hills for archers and infantry
    - Interestingly, using cavalry in forests
* Most of the proposed game mechanics made it into the final version, with the exception of fog of war.
* However, the network was not integrated into the actual game developed in Unity, and as such was not directly evaluated against human opponents, which limited my perception of its actual competitiveness.
* Different algorithms, such as Deep Q-Learning, or MM-NEAT, were not attempted.
* Finally, even more complex behaviors, such as large-scale maneuvers or formations, were not observed, but this is likely in part due to the limitations of the scenario and training time.

**Conclusions**

First, I would like to acknowledge that the original goal of this project was to begin development of an AI to play a video game that I had started developing around May of 2023. As someone who both plays video games and is a programmer with some limited experience in AI, it always frustrated me that AI in video games, especially games that require a lot of strategy or thinking, are often very poor (dumb) as opponents. Clearly this does not have to be the case, as evidenced by the incredible success of AlphaZero in playing Go, or even StarCraft2. In essence, I wanted to bring good AI and video games together to create an AI in a video game that could actually ‘out-think’ the player.

On one hand, this project is certainly informative on the challenges of developing such AI’s using neural networks. Issues of scale and of developing the correct manner in which information is given to the network, output as an action from the network, and in which feedback is provided can slow development. It is much simpler, and likely more time-efficient to simply create a script-based algorithm to play the game. Additionally, any behaviors that are desired of the AI can be coded directly, as opposed to needing to be learned by the AI.

On the other hand, neural networks, in this case NEAT-based networks, have the ability not only to learn these behaviors when given the correct training setup, but also to learn new behaviors not anticipated by the developers, thus leading to new and interesting strategies. In this project it certainly displayed this ability, not only learning an overall strategy to consistently beat the evaluation script that I developed, which consisted of keeping its units close together on its side of the board and waiting for it opponent to cross the board, but also learning other interesting behaviors in implementing that strategy, such as placing archers and infantry on hills (something that I anticipated and wanted the network to learn), as well as keeping cavalry in forests, which is something that I did not, and still do not quite understand the advantages of.

As I did not attempt any other algorithms or variations on NEAT, I don’t think I can reasonably comment on its capabilities in learning to play such games based on this project. While it was able to learn some behaviors, I often noticed that during training it would get stuck for many consecutive generations on the same genome, unable to progress, sometimes to the point that I would have to restart the training. In a continuation of this project, I would like to try other algorithms in order to assess NEAT’s comparative ability and see if another alternative would be superior for this task, and if so, how.

On the methods used for this project, I believe that the training and testing setup that I used was satisfactory for this project (albeit with some areas of improvement) but would have to be changed as the domain of the game itself expanded. The process of setting up a series of random, but constant starting positions allowed for some diversity in the original arrangement of units, ensuring that games would play out differently, without advantaging any tested genome simply by giving it a random, advantageous starting position. In the future, this system would be phased out in favor of some system of choosing starting positions, rather than being given one. While there often was a gap between the training performance and testing performance, this was successfully limited by increasing the number of games played / starting positions evaluated, as well as by increasing the size of the map and number of units. On the training setup, one very valuable observation was that the network can learn more effectively if it can learn in stages. Attempting a very difficult task often resulted in no feedback, meaning no improvement, while starting with an easier task allowed for some incremental improvement. This observation culminated in the interval training system, which I again believe was satisfactory for this project, but could use refinement in the future. Specifically, the system of self-play, while good in theory, did not really produce significant new behaviors after those learned against the script. In the future, the script-training phase should likely be removed entirely.

Overall, I believe that this project was successful, both in that I achieved most of the (reasonable) goals for the project, and that I learned a lot in regards to the development of neural-networks-based AI’s. I would like to continue to develop and adapt this project in parallel with the development of the game it is supposed to play, hopefully so that the project develops from a proof-of-concept into a real, competitive AI that will provide an enjoyable, challenging experience to a player.