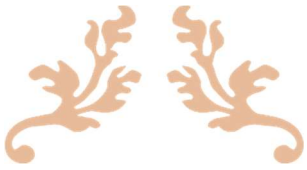


# SNAKE PROGRAM PORTFOLIO

A modern look into the classic Snake arcade game

Dustin Seger

Computer Science Major

# Table of Contents

## Executive Summary

I'm Dustin Seger, a 4th year student at the University of Cincinnati. I'm studying for a B.S. in computer science and a minor in German studies. While my current technical focuses are software development and artificial intelligence, I am also intrigued by the humanitarian side of the engineering world, striving to one day be a software project leader. I have co-oped with Crown Equipment Corporation for three terms and will be at their German plant for my final two co-op semesters.

Co-operative education has allowed me to already gain a solid foothold in the real world. My first two co-ops with Crown were on the C1515 project, an initiative dedicated to creating the next generation of lift trucks with more advanced mechanical features and a touch-screen interface. For this project, I added features to and internally overhauled a utility that downloads software to all modules of the lift trucks. I also pioneered a tool that reshaped how data was managed between engineering teams, moving the work from tedious, archaic Excel scripts to a more modern, user-friendly experience. These programs, used by 50+ engineers and the manufacturing line, were almost solely supported by me for the duration of my first two co-ops. This required great project management and coding skills, for had either of these abilities faltered, the software would have been incomplete or not used throughout the company. These skills then assisted me into my third semester on the ground floor of Crown's C1937 project, a new initiative to bring the touch-screen ideals from the C1515 project into a more portable, Android device. In this new setting with unfamiliar languages, my skills in efficient learning shined. Within my first two weeks, I was creating compilation testing utilities, and by the third, I was programming directly onto the device, adding back-end download utilities and front-end UI elements.

## Contact info


**Name:** Dustin Seger
**Phone:** (419) 790-8007
**Email:** segerde@mail.uc.edu
**Address:** [redacted]


Through my life experiences, professional and otherwise, I have developed a strong set of skills that are useful in the workforce. My brain works heavily in procedural learning, allowing me to promptly and securely learn new information when starting a new job. Especially given my self-teaching capabilities, this means less ramp-up time and less risk for my employer. Furthermore, the University of Cincinnati teaches the complex theory in class and allows for practical applications to come directly from the workforce. This has given me a wide range of computer science experiences and has made me ready for any issue that may come up while on the job. Outside of technical skills, I also enjoy humanitarian ideologies, ultimately improving areas like employee relations and program documentation.

What has brought me here is simply the motivation to do my best every day. From this idea stemmed strong skills in project management, coding, and efficient learning. I push myself even further in the workforce with my procedural mindset, humanitarian focus, and wide range of computer science experiences. I am currently searching for a job after my graduation in May 2021. Feel free to contact me with potential opportunities, and I look forward to speaking with you.

## Mission and Vision

### CONTACT INFO


**Name:** Dustin Seger

**Telephone:** (419) 790-8007

**E-Mail:** segerde@mail.uc.edu

**Address:** [redacted]


### OBJECTIVE

Task-oriented software engineer focused on efficiently producing clean, well-documented programs with thoroughness and the end-user in mind.

### PERSONAL INFORMATION

**Date of Birth:** September 14, 1997

**Place of Birth:** Coldwater, Ohio

**Citizenship:** US – American

**Marital Status:** Single

### REFERENCES

**Dan Walton:** Project Lead, Crown
dan.walton@crown.com

**Dean Winner:** Project Lead, Crown
dean.winner@crown.com

## Curriculum Vitae

### WORK EXPERIENCE

**Crown Equipment Corporation**
New Bremen, Ohio USA
08/2017 – 12/2017
05/2018 – 08/2018

**Software Engineering Co-op** – C1515 Project

- Worked under an Agile process model to develop the newest tech of Crown lift trucks
- Developed and maintained a web application to manage the installation of software onto touch screen devices
- Programmed new unit tests and features onto Linux-based touch screen devices via C++
- Assisted in quality assurance and troubleshooting of touch screen internal processes
- Pioneered the creation of a web application for the enhanced management of lift truck data
- Used HTML, JavaScript, C++, and Bash

**Crown Equipment Corporation**
New Bremen, Ohio USA
05/2019 – 07/2019

**Software Engineering Co-op** – C1937 Project

- Developed code for the ground floor of a new touch screen device
- Added back-end truck download and project build utilities
- Created new front-end Android widgets
- Used C#, Python, and C++

### TRAINING

**University of Cincinnati**
Cincinnati, Ohio USA

**Major:** Computer Science
**Minor:** German Studies
**Completion:** Bachelors of Science, May 2021
**Average Rating:** 4,00/4,00 (max)

**Coldwater High School**
Coldwater, Ohio USA

**Completion:** May 2016
**Average Rating:** 4,10/4,00 (max)

# Curriculum Vitae

## SKILLS

| | |
|---|---|
| **Programming languages** | C++, Java, Processing, JavaScript, HTML, SQL, Python |
| **Programming concepts** | OOP, FP, AI, design patterns (factory, adapter, etc.) |
| **Other skills** | Microsoft Office, GIMP, Windows, Linux, 110 WPM typing |

## AWARDS

| | | |
|---|---|---|
| **Cincinnatus Century Scholarship** | **Supplier**:<br>**Reason:** | University of Cincinnati<br>Academic excellence in high school |
| **CAP Scholarship** | **Supplier**:<br>**Reason:** | Coldwater Academic Promoters (CAP)<br>Academic excellence in high school |
| **Mantei/Mae Award** | **Supplier**:<br>**Reason:** | University of Cincinnati<br>Academic excellence in university |

## OTHER ACCOMPLISHMENTS

- 4.0 GPA into my fourth year of studies
- Designed a neural network to play the classic “Snake” arcade game
- Acquired an 8-month software engineering co-op in Germany
- Wrote an academic paper that was a contestant for a university award

# Description of “The Snake Program”

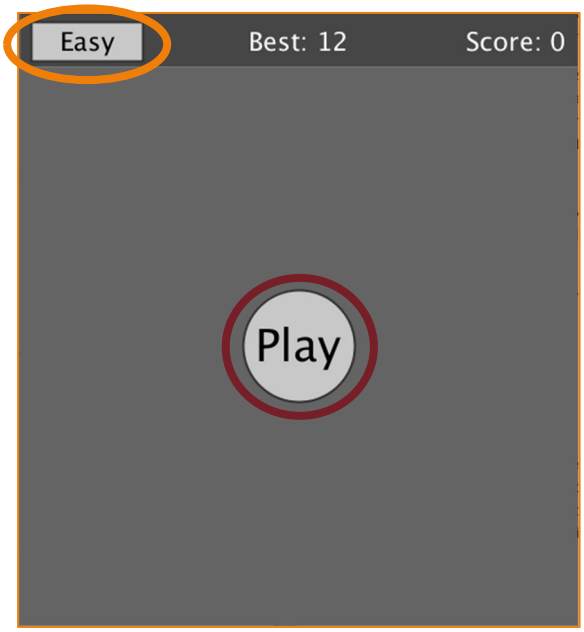
## Conceptual Overview

The Snake Program is a clone of the classic Snake arcade game, initially created in the sole interest of gaining better programming skills. It began as an intro into basic game design but has since evolved into a study of artificial intelligence methods.


Seger, Dustin. "Snake Program mid-game screenshot." 2019. PNG.

The Snake Program follows the same base principles as the classic versions. There is one snake in the game *(in white).* The snake can move only in right turns, and its body segments follow its head exactly. The snake must chase after the food *(in red).* Upon eating the food, the snake’s body grows, and one point is earned. The food then randomly finds a new location. If the snake runs into its own body or the wall, the game is over. The goal of the Snake game is to gain as many points as possible before inevitably dying.

## Specific Usage




Seger, Dustin. "Snake Program start screen." 2019. PNG.

Upon running The Snake Program, the player is greeted with a starting screen. Clicking the play button, circled in red, will begin the game. However, the difficulty can also be configured before starting. Each difficulty has its own associated best score and slightly modified game elements. The button in the top left, circled in orange, indicates the current difficulty.

Clicking this top-left button changes the difficulty, initially cycling from Easy to Medium, Hard, and Sanic. These are the **normal** modes of play, allowing the player to control the snake via the arrow keys. Each difficulty increment in this order results in an increase of snake speed. The other core mechanics remain the same.

Clicking the button while on Sanic difficulty will cycle the game to AI (artificial intelligence) mode. This is an **automatic** mode and uses a neural network to play the game without player input. Through this portfolio, I hope to research further AI mechanisms to enhance and add more automatic modes. Clicking the difficulty button again will cycle back to Easy.

# Instructions/Procedures

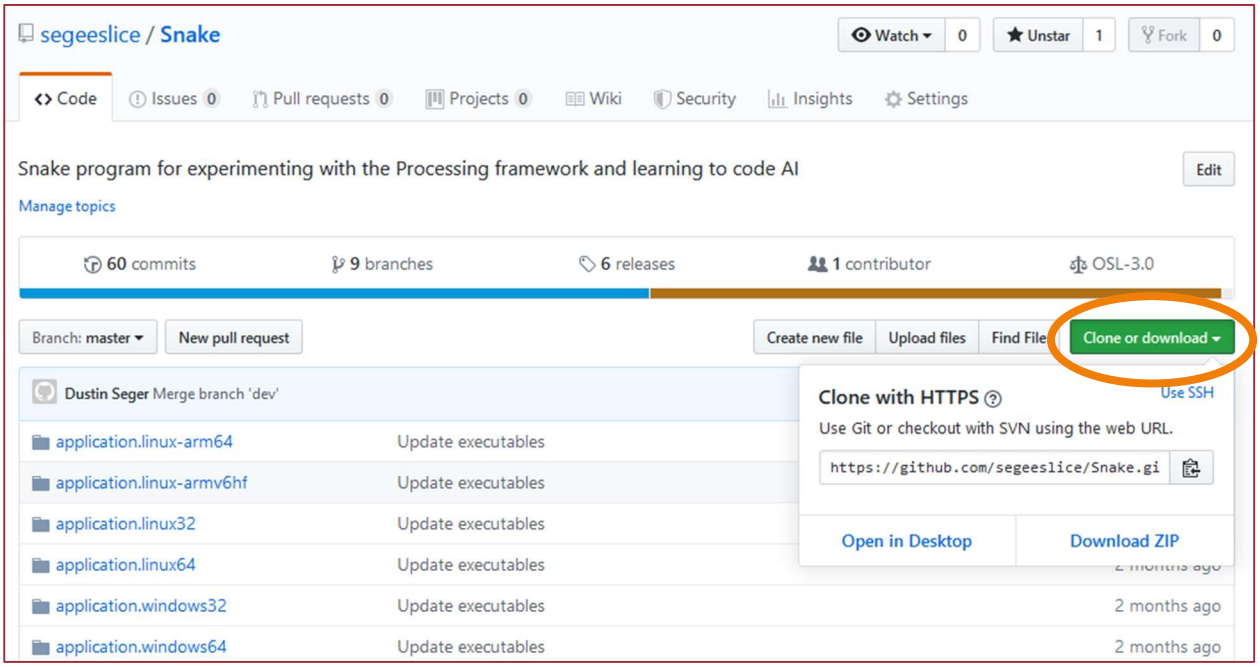
## Snake Program Development Environment Setup

The Snake program is a simple experiment in artificial intelligence (AI) written in Java through the Processing framework. If you are interested in AI or coding with Java, then perhaps you would also like to see the source code to the Snake program or make changes to see how it affects the AI. If so, then follow these steps.

> ***NOTE:*** *These instructions generally assume that you have prior experience in handling files and using GIT. Using GIT is not required, but if interested, go to https://try.github.io/ for multiple GIT installation and learning resources.*

### *Step 1: Retrieving the source code,*

First, go to the Snake program's GIT repository, found at https://github.com/segeeslice/Snake. On this site, click on the green "Clone or download" button *(pictured below)*. Upon doing this, a dropdown will appear that is titled "Clone with HTTPS".




Seger, Dustin. "Snake Program GitHub page screenshot." 2019. PNG.

In this dropdown, you have two options:

- **Click "Download ZIP"** to access just the source code.
- **Copy the URL** in the text box and **clone via GIT** to access both the source code and repository.

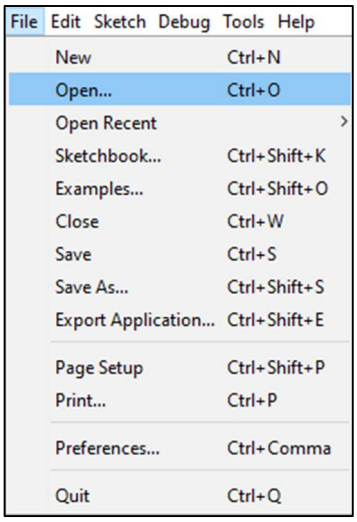
Unzip the files if necessary and put them into a meaningful location. **Make a note of where you saved the files,** as you will be needing this in the next step.

### *Step 2: Running and editing the code*

First, go to the Processing download site at https://processing.org/download/ and click on the link most relevant to your operating system *(pictured below)*. This will download a ZIP file containing the Processing program.


Seger, Dustin. "Processing.org download page screenshot." 2019. PNG.

Unzip the downloaded file. You can place the unzipped folder anywhere, but it is recommended to keep it in the same directory as your other computer programs. Within the unzipped folder, run the file titled "processing" to start the Processing program. Within this software, find the top left toolbar and go to **File -> Open** *(pictured right)*. Click this, and a file dialog should appear. Travel to the folder of your Snake program download from Step 1. In this folder, select the file **"snake.pde"**. This will load the Snake program and all its necessary files into Processing.




Seger, Dustin. "Windows 10 Processing file dialog screenshot." 2019. PNG.

You can now edit the files as you wish and run the program via the play button in the top left corner. The code is in Java, but it utilizes Processing's various utility functions for graphics. If you'd like more information on the Processing framework, go https://processing.org/tutorials/. This contains numerous video tutorials on the framework.

### *Step 3: Enjoy!*

You should be ready to toy with the code for the Snake program. The comments within should outline the code enough to get a grasp on the concepts at play. If you have any contributions or suggestions, feel free to contact me or send a pull request with your changes via GitHub.

# Research Proposal – Search-Driven AI for the Snake Program

***Research Question:*** *Can a perfect snake AI be created via an A* search algorithm?*

## Literature Survey

To answer if an A* search algorithm is feasible for a Snake game AI, various resources needed to be employed. A* uses a heuristic estimate to the goal and a running distance tracker to always find the shortest path to the goal if it exists (Russel & Norvig, 93). Furthermore, search-tree methods like A* are often used in game theory (Peters, 51), showing the potential significance of A* in the Snake program. However, A* is also "not practical for many large-scale problems" (Russel & Norvig, 99). Thus, it was not immediately clear if this would be suitable for the Snake game. Since AI as a field is moving towards more in-depth, "self-teaching" algorithms, these were examined as well. Various forms of these learning algorithms are used in game theory (Peters, 139). Dynamic learning methods are also becoming much more powerful than basic searches, even being related to and inspired by the human subconscious mind (Clark). While these facts do support the worthiness of a smarter algorithm, this results in dramatically increased complexity. Even decision trees that learn and can be searched via algorithms like A* require a significant set of previous theorems and equations to operate similarly (Burkov, 29). It is recommended to treat AI as an "experiment", ensuring to "choose the minimum set of technologies needed to tell the … story" (Overton). Ultimately, A* is an easier task to tackle, and machine learning and similar concepts do not appear necessary for a Snake AI to function. Thus, A* has been deemed a suitable candidate for the AI's advancement.

## Problem Statement

The Snake Program's current neural network AI is limited in its design, resulting in lower scores than expected. A new AI following a different technique could allow for consistently higher scores.

## Current Implementation

It is first important to see how the current AI understands the game. The snake cannot be directly controlled in any part but the head of the snake. The rest of the body simply follows behind the head's movements. As such, the current AI only looks at the head, taking in various input values in relation to the head's direction. This includes X and Y distance to food, X and Y distance to the next obstacle (be it a body part or the wall), and "trap" detection. These inputs are put through a **neural network**, allowing the AI to **prefer** moving towards the food, moving away from obstacles, and staying away from areas in which reaching the food may leave the head in an inescapable position.

The result of the current AI implementation is ultimately non-perfect. That is, the AI does not hit the max score possible, acquired by filling every tile on the screen with body parts. A primary issue in this implementation is that it only works turn-by-turn, resulting in a **lack of foresight**. The other issue is **lack of certainty** resulting from the preferential implementation, meaning the AI is never completely sure if it will reach the food or not.

## Proposed Solution

While various other AI implementation methods were examined for feasibility, it was ultimately decided that a search-driven AI would be the best. The **Literature Survey** should be referenced for an analysis on the other considered options.

A search-based AI would view the Snake game differently from the current neural network. In general, search-based AI "can decide what to do first by examining future actions that eventually lead to states of known value" (Russel & Norvig, 65). This searching prior to moving would solve the neural network's problem with **lack of foresight**. One reason A* was deemed the primary subject is its completeness (Russel & Norvig, 93), meaning it will always find a solution if one exists. This would solve the neural network's **lack of certainty** issue. Given that a search-driven AI using a complete algorithm would address the current issues, A* can be further concluded as a viable new AI method for the Snake Program. However, it should be noted again that A* can be too cumbersome for some large-scale applications (Russel & Norvig, 99). This will need to be watched moving forward.

## Functional Overview

Searches effectively sort through all possible states of some given system in search of some goal state. When the goal is found, it returns with the steps it took to get there. To accomplish this, A* keeps track of two metrics: the running distance traveled and a heuristic estimated distance to the goal state. A* then adds these metrics together for a combined estimated cost. As the search runs, it always selects the item with the smallest combined cost (Russel & Norvig, 93). This continues until the selected item is a goal state.

The new AI will use these basic A* principles to find a path to the food. The states will be snapshots of the board, including snake and food positions. The goal is any state in which the head of the snake is in the same location as the food, as this would result in the food being eaten and a point being earned. The distance metric will be the number of moves taken, and the heuristic estimation will be based on Manhattan distance from the head to the food. These will both derive from the Snake Program's coordinate system. Upon finding a goal state, the path to that state will be parsed and used by the AI to travel to the food. If no goal state is found, then no path to it exists, and the AI will take any arbitrary path towards its death.

## Implementation Plan

### *High-Level Decisions*

Although acting as a better version of the current AI, the new A*-based AI will be added as a new feature, not a replacement. It will be an option selectable via the upper-left mode button alongside the current AI selection. To hide the gritty details of the implementation, the alias of "**Brute**" will be given to this AI. This is to reflect its "brutish" method of simply finding a path and taking it.

### *Brute Development Plan*

1. Extend the Snake class to allow for equality checks
2. Create a queue item data structure to store the following information:
    a. Current state of the snake
    b. Number of turns to reach this state
    c. Estimated distance to the goal state
    d. Combined cost value of this state (turn number + estimated distance)
    e. Previous move taken to reach this state
    f. Pointer to this state's "parent", or previous state
3. Create an efficient priority queue structure via a Min Heap
4. Create Brute class with basic snake movement capabilities
5. Add selectable Brute mode to the UI
6. Extend Brute class to be able to find all "neighbors" of the current snake
    a. i.e. Check where the snake would be if it turned left, turned right, or continued straight
7. Implement a search algorithm to do the following:
    a. Find the "goal" state via A* (detailed in Functional Overview)
    b. Use parent pointers to generate a path backwards from the goal
    c. Parse the generated path for the list of moves taken
    d. Return the list of moves
8. Update the Brute class to keep a "move list" for turn-by-turn processing:
    a. If the move list is empty, generate a path via A* and update the move list
    b. If the move list is not empty:
        i. Move the snake according to the next move in the list
        ii. Delete this move from the list
9. Test and optimize as necessary

### *Brute Stretch Goals*

- Colorize snake to better show which direction it is going
- Investigate other, similar search algorithms
- Add statistic-collecting features for tests of performance

## Conclusion

Using a search-based AI may have promise for the Snake Program. It effectively addresses the pitfalls of the current neural network AI, but searching may also hurt the performance of the program given the large amount of body position possibilities. Regardless, it is a simpler solution to investigate than most modern approaches, making it a viable option to quickly look into, especially given its established usefulness in the world of video games (Peters, 51).

Development progress can be tracked at the Snake Program GitHub page.
*(See **Snake Program Development Environment Setup** for GitHub URL and installation steps)*

# Formal Report – Search-Driven AI for the Snake Program

***Research Question:*** *Can a perfect snake AI be created via an A* search algorithm?*

## Literature Survey

To answer if an A* search algorithm is feasible for a Snake game AI, various resources needed to be employed. A* uses a heuristic estimate to the goal and a running distance tracker to always find the shortest path to the goal if it exists (Russel & Norvig, 93). Furthermore, search-tree methods like A* are often used in game theory (Peters, 51), showing the potential significance of A* in the Snake program. However, A* is also "not practical for many large-scale problems" (Russel & Norvig, 99). Thus, it was not immediately clear if this would be suitable for the Snake game. Since AI as a field is moving towards more in-depth, "self-teaching" algorithms, these were examined as well. Various forms of these learning algorithms are used in game theory (Peters, 139). Dynamic learning methods are also becoming much more powerful than basic searches, even being related to and inspired by the human subconscious mind (Clark). While these facts do support the worthiness of a smarter algorithm, this results in dramatically increased complexity. Even decision trees that learn and can be searched via algorithms like A* require a significant set of previous theorems and equations to operate similarly (Burkov, 29). It is recommended to treat AI as an "experiment", ensuring to "choose the minimum set of technologies needed to tell the … story" (Overton). Ultimately, A* is an easier task to tackle, and machine learning and similar concepts do not appear necessary for a Snake AI to function. Thus, A* has been deemed a suitable candidate for the AI's advancement.

## Problem Statement

The Snake Program's current neural network AI is limited in its design, resulting in lower scores than expected. A new AI following a different technique could allow for consistently higher scores.

## Neural Implementation

It is first important to see how the previously-developed neural network AI understands the game. The snake cannot be directly controlled in any part but the head of the snake. The rest of the body simply follows behind the head's movements. As such, the current AI only looks at the head, taking in various input values in relation to the head's direction. This includes X and Y distance to food, X and Y distance to the next obstacle (be it a body part or the wall), and "trap" detection. These inputs are put through a **neural network**, allowing the AI to **prefer** moving towards the food, moving away from obstacles, and staying away from areas in which reaching the food may leave the head in an inescapable position.

The result of the current AI implementation is ultimately non-perfect. That is, the AI does not hit the max score possible, acquired by filling every tile on the screen with body parts. A primary issue in this implementation is that it only works turn-by-turn, resulting in a **lack of foresight**. The other issue is **lack of certainty** resulting from the preferential implementation, meaning the AI is never completely sure if it will reach the food or not.

## Search-Driven Solution

### *High-Level Overview*

While various other AI implementation methods were examined for feasibility, it was ultimately decided that a search-driven AI would be the best. The **Literature Survey** should be referenced for an analysis on the other considered options.

The search-based AI views the Snake game differently from the current neural network. In general, search-based AI "can decide what to do first by examining future actions that eventually lead to states of known value" (Russel & Norvig, 65). This searching prior to moving should theoretically solve the neural network's problem with **lack of foresight**. One reason A* was deemed the primary subject is its completeness (Russel & Norvig, 93), meaning it will always find a solution if one exists. This should solve the neural network's **lack of certainty** issue. Given that a search-driven AI using a complete algorithm would address the current issues, A* can be further concluded as a viable new AI method for the Snake Program. However, it should be noted again that A* can be too cumbersome for some large-scale applications (Russel & Norvig, 99). This will need to be watched in the coming analyses.

To hide the gritty details of the implementation, the alias of "**Brute**" was given to this new AI. It is an option selectable via the upper-left mode button alongside the pre-existing "**Neural**" AI.

### *Functional Overview*

Searches effectively sort through all possible states of some given system in search of some goal state. When the goal is found, it returns with the steps it took to get there. To accomplish this, A* keeps track of two metrics: the running distance traveled and a heuristic estimated distance to the goal state. A* then adds these metrics together for a combined estimated cost. As the search runs, it always selects the item with the smallest combined cost (Russel & Norvig, 93). This continues until the selected item is a goal state.

The new Brute AI uses these basic A* principles to find a path to the food. The states are snapshots of the board, including snake and food positions. The goal is any state in which the head of the snake is in the same location as the food, as this would result in the food being eaten and a point being earned. The distance metric is the number of moves taken, and the heuristic estimation is based on Manhattan distance from the head to the food. These both derive from the Snake Program's coordinate system. Upon finding a goal state, the path to that state is parsed and used by the AI to travel to the food. If no goal state is found, then no path to it exists, and the AI takes any arbitrary path towards its death.

### *Efficiency Issues and Resolution*

The creation of the Brute AI was not without its hiccups, the most apparent being efficiency issues. These issues were not anticipated prior to development due to the game's logical simplicity. However, it was quickly discovered that the snake body can have a massive number of potential positions. Upon reaching a new position, an increasing number of state comparisons involving each individual body part is also necessary. These issues resulted in a large amount of required work for the CPU, occasionally freezing the program. As warned by various studies (Russel & Norvig, 99), this proved A* to be potentially incompatible with the Snake Program due to its large number of possible outputs.

However, work was done to mediate these efficiency issues. The queue of upcoming states was refactored from a list to a min-heap, changing the add and remove efficiencies from *O(n)* to *O(log(n))*. Snake equality and set membership checks were also refactored to use hashes instead of basic list comparisons. A snake is hashed only once upon snake movement, after which it has near instant equality and set membership tests. While these improvements did help, they were not enough for a full A* search. That is, the full body of the snake could still not be considered without the program freezing for significant amounts of time. With no further avenues for direct improvement, this was remedied by only considering a frontal subsection of the snake instead of the whole body. The decided value that results in good scores but does not drastically reduce performance is *6* parts. That is, only the first *6* body parts of a snake are considered when finding a path to the food. The result is an efficient but impure A* hybrid algorithm that is not complete, meaning it may not always find paths that require the direct manipulation of more than 6 body parts.

## AI Comparison Data

*All data based on 20 random trials of the Brute and Neural AI's…*


**Figure 1 - Seger, Dustin. "AI Scores." 2019. Graph.**

*Analysis*

As far as raw score, the new Brute AI is roughly the same as the old Neural one. As shown by the table to the right, the Brute's and Neural's average scores are roughly the same with the Neural having a slight edge. However, their standard deviations are both very high, indicating that the scores in both AI's are fairly inconsistent.

| | Neural | Brute |
|---|---|---|
| **Max Score** | 59 | 52 |
| **Min Score** | 14 | 21 |
| **Average Score** | 38.80952 | 37.14286 |
| **Score Std. Dev.** | 11.56122 | 10.02141 |
| **Max MPP** | 68.9434 | 25.33333 |
| **Min MPP** | 19.82609 | 16.58621 |
| **Average MPP** | 37.57786 | 21.814 |
| **MPP Std. Dev.** | 10.82058 | 2.728144 |

**Figure 2 - Seger, Dustin. "AI Statistics." 2019. Table.**

Where the Brute AI shines is in it's moves per point (**MPP**). This is the result of dividing a game's total number of moves by its final score, showing an average of how many moves it took to gain each point. Thus, lower values mean that the points were received faster. As shown in the table, the Brute AI has consistently lower MPP's across the board, meaning it reached its food much faster than the Neural did. This can also be seen in the graph above where the Neural's scores have a wide spread across the number of moves it took. On the other hand, the Brute's scores are compacted to the left and linearly increase with the moves it takes. This suggests the Brute AI is generally faster and has better scalability than the Neural AI.

## Timing

By examining the program's GIT history, it was discovered that the Brute AI took roughly **811** lines of code changes, whereas the Neural AI took around **367** lines. Thus, the Brute AI took more to complete in terms of raw size. When looking at it this way, it may seem that the Brute is more logically complex. However, many of the Brute changes were simple data structure instantiations and may not directly contribute to the complexity or timing of the AI itself. For instance, the min-heap used for an efficient queue took 169 lines of code alone, but this was written in a few hours. Same situations could be said for extraneous Neural AI work.

The algorithm files themselves may tell a different story. When considering only the logical implementation of the algorithms rather than including all dependencies, the Brute AI took **207** lines of code, whereas the Neural AI took **212**. This shows their raw output to be roughly the same. However, the Neural AI is much more abstract and required much more time and experimentation to fine tune. On the other hand, the Brute AI had a clear vision and goal, ultimately taking less time to logically lay out. However, it did still take a sizeable amount of extra work in optimization and heuristic experimentation.

In short, it is hard to make a decisive comparison. When considering the amount of time and logical simplicity to be the deciding factor, it could be concluded that the Brute AI is a more timely and simpler solution to the AI problem. However, when considering the lines of code and amount of reworking to accommodate each implementation, the amount of work likely evens out.

## Conclusion

The Brute AI turned out to be a worthwhile avenue to explore, although it did not solve all of the issues at hand. While it performs faster than the Neural AI, the end scores did not improve as expected. This is likely in part due to the sacrifices made for efficiency in the Brute AI. Ultimately, further foresight would be required to move towards a truly perfect AI. For instance, the Brute may know how to perfectly reach the goal, but it has no knowledge of information after that, meaning it could be moving into a dead end. Perhaps this could be resolved in the future by redefining a goal state, ensuring that the food is eaten in that position *and* a path to the tail is present, allowing for a way out. Or perhaps the heuristic could be improved to increase efficiency enough that the whole body could be checked, resulting in better scores. In any case, more work needs to be done, and the "dumb" Brute needs to become "smarter" if it wishes to truly trump the Neural AI.

To try the Brute AI and track any future progress, see the Snake Program GitHub page.
*(See **Snake Program Development Environment Setup** for GitHub URL and installation steps)*

## Sources Cited/Referenced


Barron, E. N. *Game Theory: An Introduction*. 2nd ed., John Wiley & Sons, 2013.

Botea, Adi, et al. "Optimal Reconfiguration for Supply Restoration With Informed A* Search." *IEEE Transactions on Smart Grid*, vol. 3, no. 2, 1 Mar. 2012, pp. 583–593. *IEEE Xplore Digital Library*, doi:10.1109/TSG.2012.2184778.

Burkov, Andriy. *The Hundred-Page Machine Learning Book.* Andriy Burkov, 2019.

Clark, Jack. *Artificial Intelligence: Teaching Machines to Think Like People*. O'Reilly Media, Inc., 2017.

Overton, Jerry. *Artificial Intelligence: The Simplest Way.* 2018.

Peters, Hans. *Game Theory: A Multi-Leveled Approach*. Springer Berlin Heidelberg.

"Processing.org." *Processing.org,* Processing Foundation, https://processing.org/

"Resources to Learn Git." *Resources to Learn Git*, GitHub, https://try.github.io/

Russell, Stuart J, and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2015.

Zeng, W., and R. L. Church. "Finding Shortest Paths on Real Road Networks: The Case for A*." *Zenodo*, 1 Apr. 2009, doi:10.1080/13658810801949850.

# Appendixes

## Appendix 1 – Professional Skills

- **Coding Languages – Proficient**
  - C, C++
  - C#
  - Python
  - Java, Processing
  - HTML
  - JavaScript, NodeJS, Vue.js
- **Coding Languages – Familiar**
  - CLISP, Scheme
  - Haskell
  - Visual Basic
  - Bash
- **Software Proficiencies**
  - Microsoft Office (Word, PowerPoint, Excel, Visio)
  - GNU Image Manipulation Program (GIMP)
  - Vim, Neovim
  - Windows, Linux
- **Other Skills**
  - German (intermediate)
  - 110 WPM typing speed

## Appendix 2 – Informal Report

| | |
|---|---|
| **TO:** | Dr. Kristen Race |
| **FROM:** | Dustin Seger |
| **DATE:** | October 23, 2019 |
| **SUBJECT:** | Progress Report |
| | Serves as an update on the progress of my Fall 2019 semester as well as a brief overview of expected future events. |

### Work Completed

| | |
|---|---|
| AI Principles: | Completed one programming and one written assignment surrounding AI topics like state search methods. Took the first mid-term exam. |
| Computer Networks: | Completed two written assignments on networking concepts as we work down the TCP/IP protocol layers. Took the first mid-term exam. |
| German (2nd Year): | Wrote two essays and submitted weekly homework assignments to enhance my German comprehension. Took the first mid-term exam. |
| Operating Systems: | Finished three homework assignments on OS concepts such as Unix processes and CPU scheduling. Took the first mid-term exam. |
| Technical Writing: | Worked on an ongoing portfolio, adding various portions including personal professional info and instructional details surrounding a coding project. |

### Future Work

| | |
|---|---|
| AI Principles: | Will have up to two more homework assignments as well as quizzes and two more exams over further, more sophisticated AI concepts. |
| Computer Networks: | Will continue to learn the details behind the TCP/IP network layers via two more homework assignments and two more exams. |
| German (2nd Year): | Will learn more German constructs and vocabulary through weekly assignments, one more essay, and two more exams. |
| Operating Systems: | Will delve deeper into OS concepts such as file structures via possible future homework assignments as well as two more exams. |
| Technical Writing: | Will update and eventually complete the portfolio project, adding sections like a research proposal and formal report. Will also need to update my personal coding project to reflect my desired research. |