Comparative Analysis of Various Genetic Algorithm Approaches in Video Game Environments

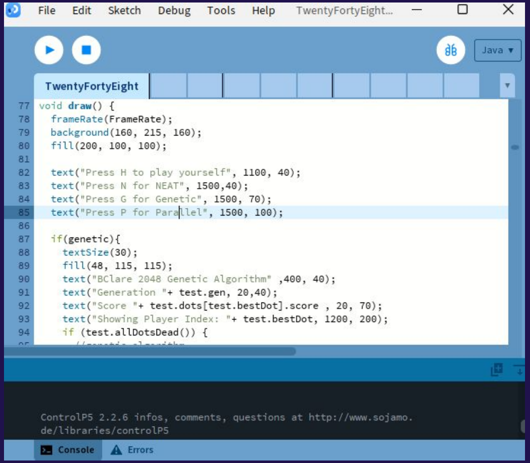
Brendan Clare

Advisor: Professor Douglass

```
Gen 113 Species 2 Genome 18 (16%)
Fitness: 1905 Max Fitness: 3066
```

Introduction

- Genetic algorithms (GAs) are efficient search-based methods inspired by biological evolution using principles of natural selection, crossover, and mutation.[4]
- Three different GAs are used to train autonomous agents to play video games, maximizing their score.
- The motivation behind this research is to compare how effective these algorithms are in a video game environment. Video games provide a controlled yet complex environment where the algorithms can be tested in real-time.
- The games I am using for testing are my adaptations of popular browser/mobile games are as follows:
  1. World’s Hardest Game
  2. 2048
  3. My Own Game I am Calling Flappy Tiles
- I used Processing which is a Java based free graphics library that provides an integrated development environment (IDE), a graphical user interface (GUI), additional classes and mathematical operations.
- I adapted these games to use GA templates I found on GitHub[7][8]

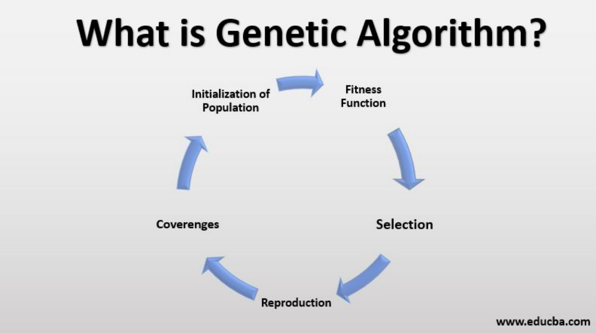


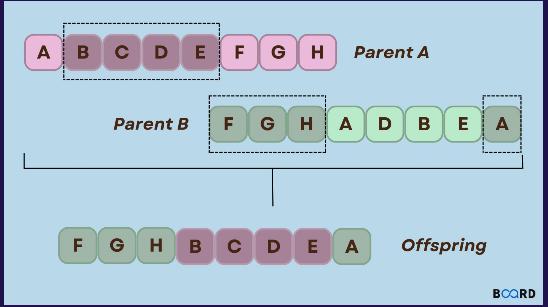
```
void draw() {
  frameRate(FrameRate);
  background(160, 215, 160);
  fill(200, 100, 100);

  text("Press H to play yourself", 1100, 40);
  text("Press N for NEAT", 1500,40);
  text("Press G for Genetic", 1500, 70);
  text("Press P for Parallel", 1500, 100);

  if(genetic){
    textSize(30);
    fill(48, 115, 115);
    text("BClare 2048 Genetic Algorithm" ,400, 40);
    text("Generation "+ test.gen, 20,40);
    text("Score "+ test.dots[test.bestDot].score , 20, 70);
    text("Showing Player Index: "+ test.bestDot, 1200, 200);
    if (test.allDotsDead()) {
```

```
ControlP5 2.2.6 infos, comments, questions at http://www.sojamo.
de/libraries/controlP5
```

Genetic

- GAs define finite-length strings which are possible solutions to a search problem referred to as *chromosomes,* the alphabets are referred to as *genes,* and the values of genes are referred to as *alleles.*
- The algorithm uses the following steps: 1. Initialization, 2. Evaluation 3. Selection 4. Recombination 5. Mutation 6. Replacement 7. Repeat 2-6 until a set stopping condition is met.
- Initialization involves defining the population size as well as the chromosomes. [6] The agents are evaluated based on the performance of the game they are playing using a fitness function. The fitness values are used to select the best players from the population.
- The best players are used as parents to produce offspring that recombine their solutions to potentially produce better solutions.
- There is a chance of mutation occurring by changing an individual trait or traits.
- Based on this selection, recombination, and mutation a new population replaces the previous generation.
- These steps are repeated until generation 1000 where a brand new population is initialized and the generation is set back to 1





Parallel

- Parallel genetic algorithms implement a divide-and-conquer approach, splitting up a task and solving it with multiple processors
- There are different parallelization methods and the one I’m using is the coarse-grained also known as multi-deme, distributed, or island parallel GAs.
- This version of the parallel genetic algorithm resembles the “island model” of Population Genetics where subpopulations of the same species are physically separated so they evolve separately.
- Each population of agents in the algorithm is treated separately, with the possibility of individuals migrating.

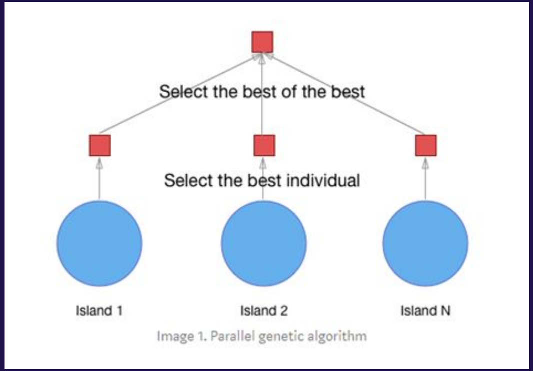
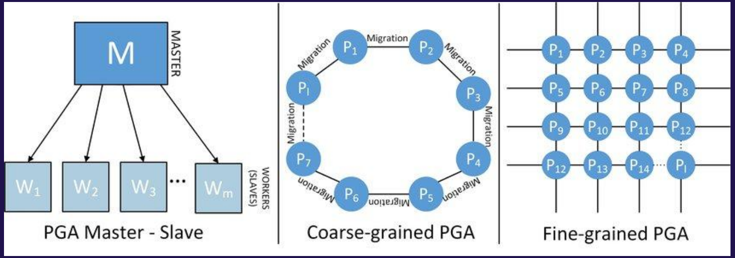
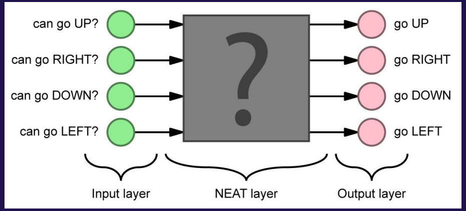


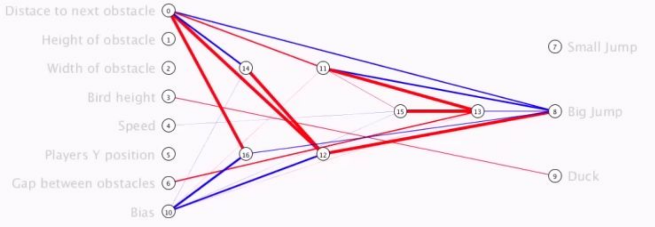
Image 1. Parallel genetic algorithm



NEAT (NeuroEvolution of Augmenting Topologies)

- Proposed by Kenneth O. Stanley and Risto Miikkulainen in June 2002 who found that this method performed better than fixed-topology Neuroevolution methods in tasks such as the pole balancing problem.[3]
- Represents neural networks as genomes, where each genome is a set of genes. genes can represent either a neuron (node) or a connection between neurons (edges).
- Starts with an initial population of simple neural networks
- Networks are grouped into species based on similarity, and individuals within the same species can compete against each other.
- New nodes and connections can be added through mutation, and unnecessary structures can be removed to simplify the network.

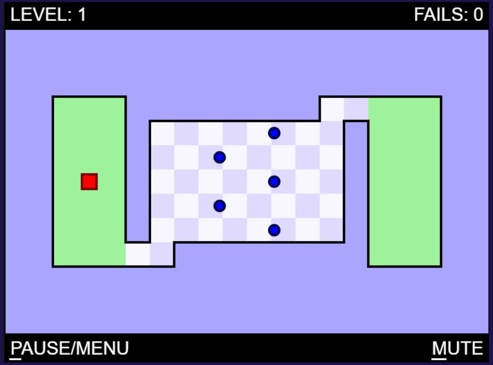




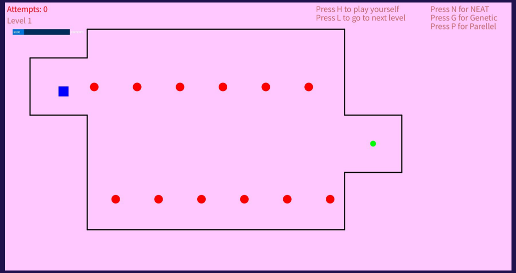
World’s Hardest Game

- Developed by Snubby Land in 2008
- Difficult 2D puzzle game where the player is a red square and they are trying make it to the safe zone while avoiding blue circles
- Player can move up, down, left and right
- My version has a player class to define the player and movement, an enemy class that moves a circle up and down and a level class to set up the levels with the enemies, defined barriers and goal.
- Here are my inputs for NEAT:
    - void look() {
    - vision[0] = pos.x;
    - vision[1]= pos.y;
    - PVector NearestEnemyPos = lvl.FindNearestEnemy(pos);
    - vision[2] = NearestEnemyPos.x;
    - vision[3]= NearestEnemyPos.y;
    - vision[4] = lvl.goal.x;
    - vision[5]= lvl.goal.y;
    - }
- Fitness is based on the distance to the goal:
    - void calculateFitness(PVector goal) {
    - if (reachedGoal) {//if the player reached the goal then the fitness is based on the amount of steps it took to get there
    - fitness = 1.0/16.0 + 10000.0/(float)(step * step);
    - } else {//if the player didn't reach the goal then the fitness is based on how close it is to the goal
    - float distanceToGoal = dist(pos.x, pos.y, goal.x, goal.y);
    - fitness = 1.0/(distanceToGoal * distanceToGoal);
    - }
    - }

Actual Game



My Version



2048

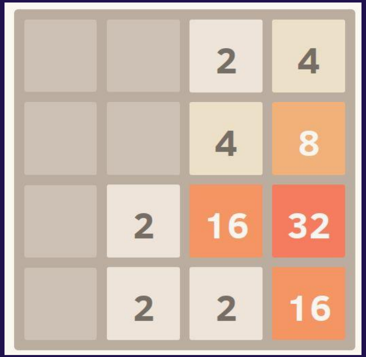
- Developed by Gabriele Cirulli in two days in March 2014 and has since had millions of downloads/plays
- The game consists of a 4X4 grid of tiles
- Using the arrow keys, tiles of the same value can be combined to merge into one tile, double the value of the original tiles
- The new value of the merged tiles is added to the score and this is the same as the fitness
- My version consists of a player class and a tile class:

```
Player() {
    grid = new Tile[4][4];
     for (int i = 0; i < 4; i++) {
      for (int j = 0; j < 4; j++) {
        grid[i][j] = new Tile(i, j, tileSize);
      }
    }
    addRandomTile();
    addRandomTile();
    }
Tile(int x, int y, int size) {
    this.x = x;
    this.y = y;
    this.size = size;
    this.value = 0;  // Start as an empty tile
    setTileColor();
  }
```

- When a player moves a new tile is randomly placed in an empty slot, however in my version I took away the randomness by always putting the new tile in the last empty slot and having the value cycle from 2 nine times and 4 once
- The game ends when the player can't move anymore i.e. there are no empty tiles and no adjacent tiles have the same value
- The inputs for NEAT are the values of all the tiles:

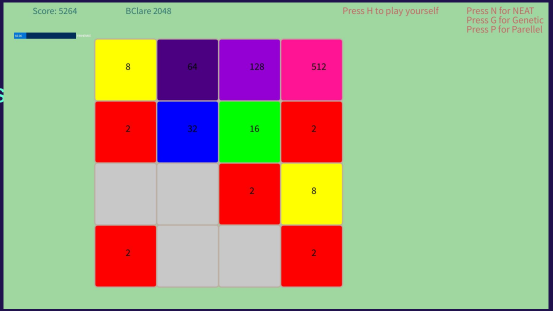
```
void look() {
  int index = 0;
  for (int i = 0; i < 4; i++) {
    for (int j = 0; j < 4; j++) {
      vision[index] = (grid[i][j].value);
      index++;}}}
```

Actual Game



		2	4
		4	8
	2	16	32
	2	2	16

My Version



Score: 5264 BClare 2048 Press H to play yourself

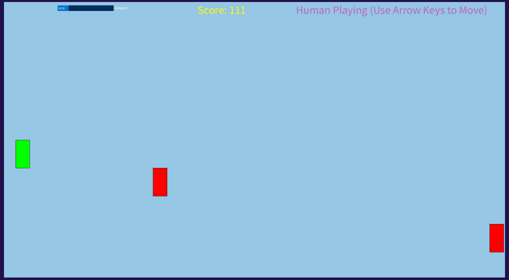
Press N for NEAT
Press G for Genetic
Press P for Parellel

8	64	128	512
2	32	16	2
		2	8
2			2

Flappy Tiles

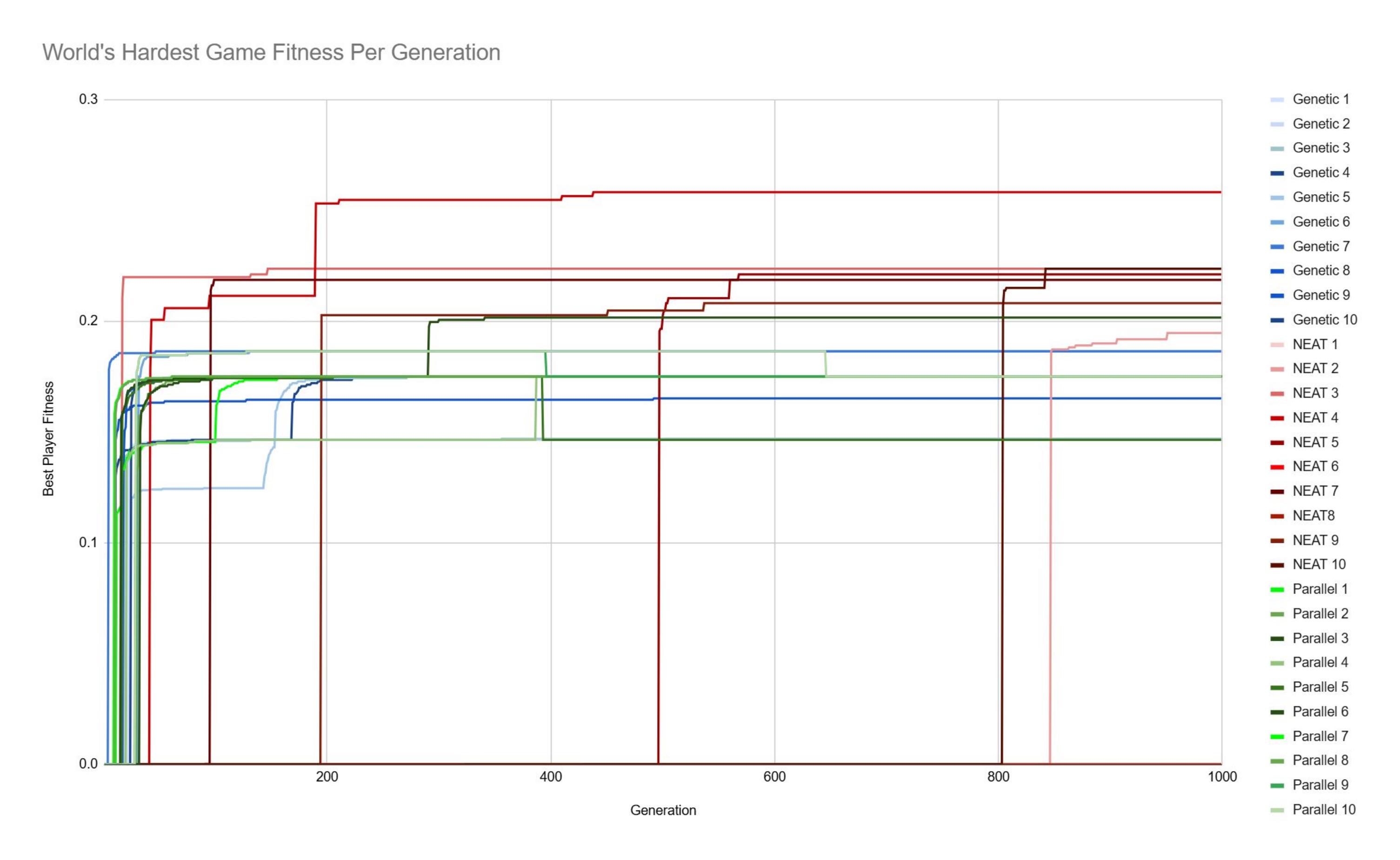
- 2D survival game I made where the player is a green rectangle that can move up and down to avoid red rectangles (enemies).
- Enemies move from left to right spawning on a random y position
- Score increases every frame survived
- Fitness is the score
- The inputs for NEAT are the Y coordinate of the nearest enemy and the player:

```
void look() {
    float minDistance = Float.MAX_VALUE;
    int minIndex = -1;
    for (int i = 0; i < enemies.size(); i++) {
      float distance = enemies.get(i).x + 50 - (x - 25);
      if (distance < minDistance && distance > 0) {
        minDistance = distance;
        minIndex = i;
      }
    }
    if (minIndex == -1) { // If there are no enemies
      vision[0] = 0;
    } else {
      vision[0] = enemies.get(minIndex).y; // Set to Y coordinate of the nearest enemy
    }
    vision[1]= y;
}
```

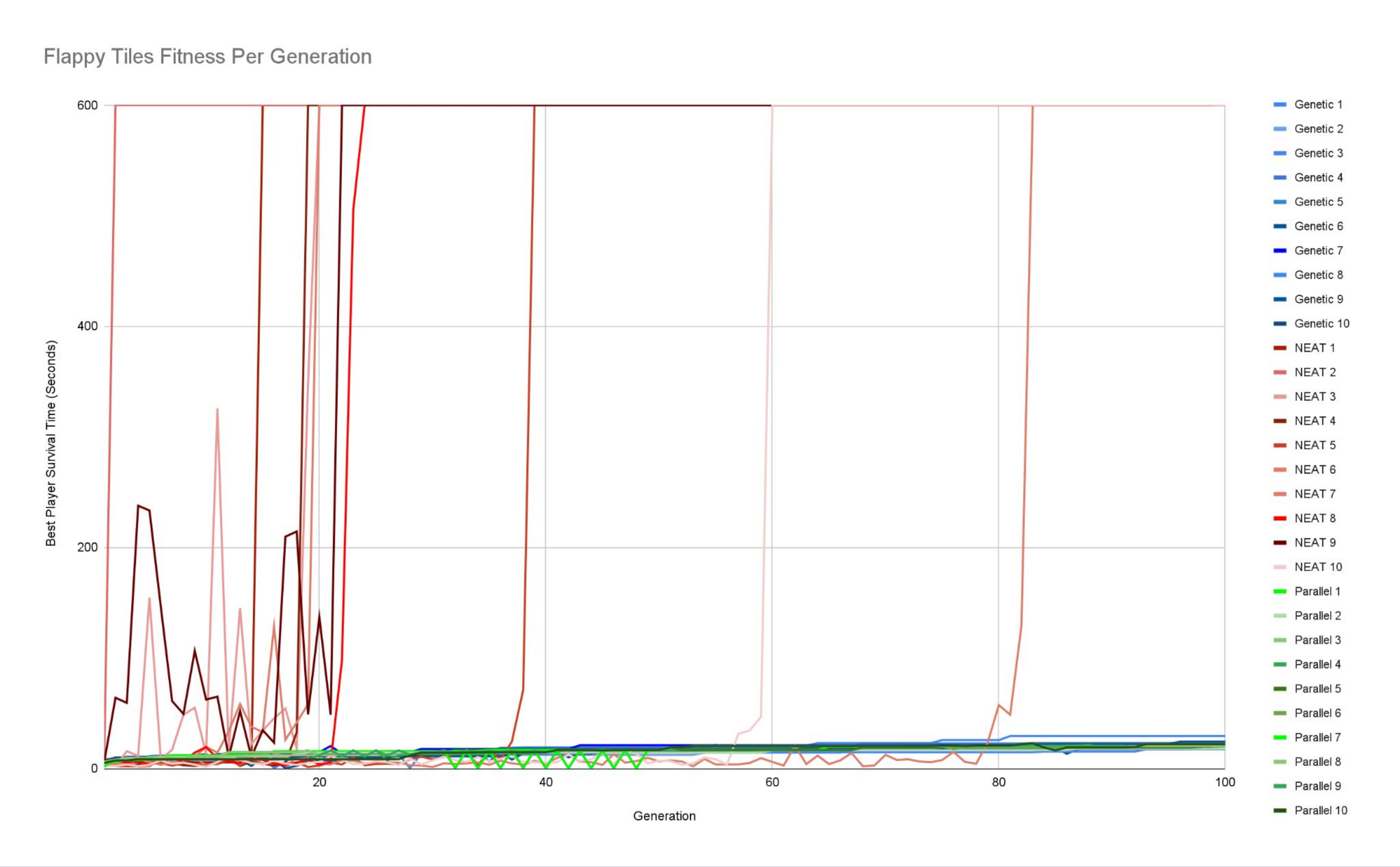


Demo

Results

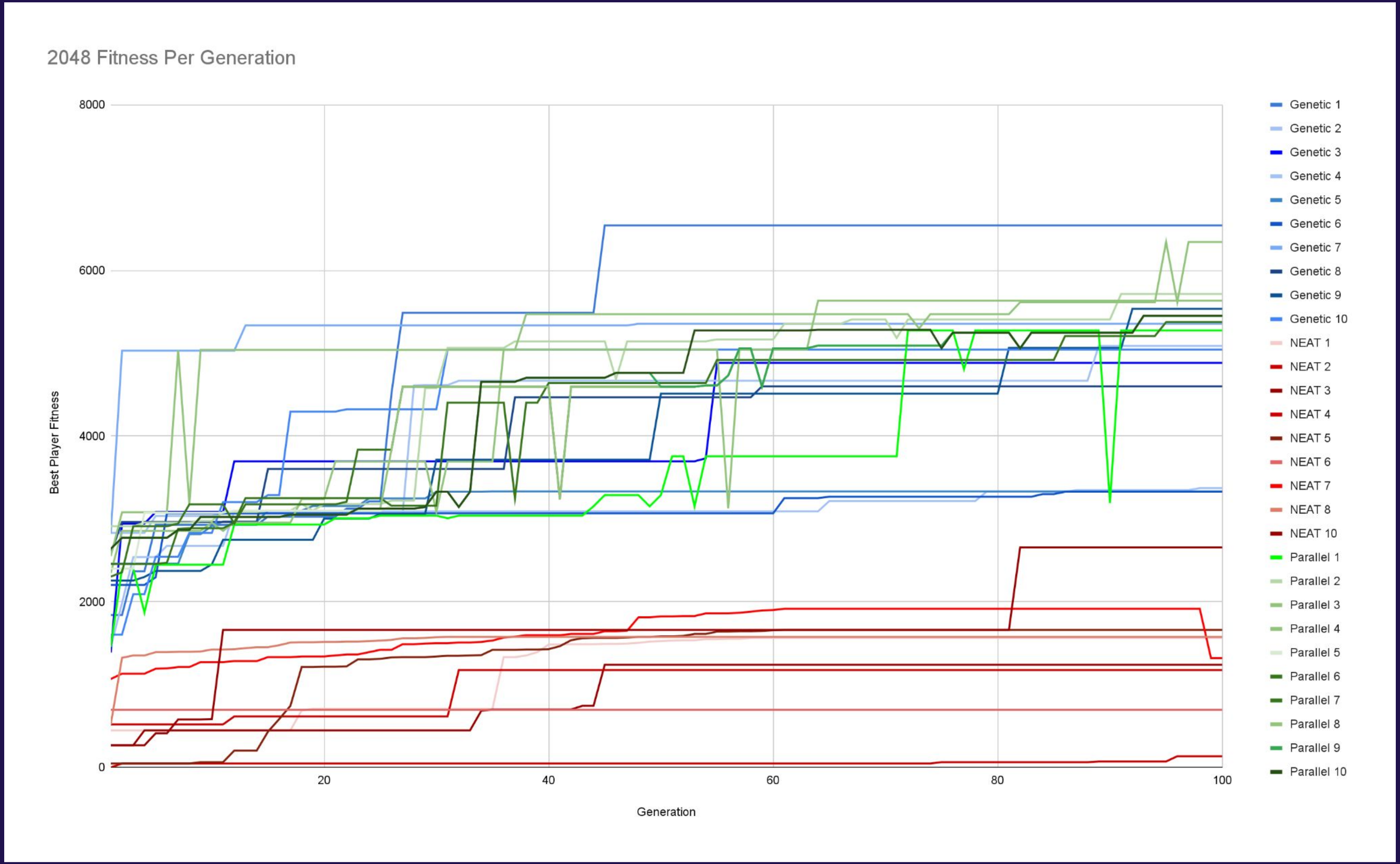

World's Hardest Game Fitness Per Generation
Best Player Fitness
0.3
0.2
0.1
0.0
200
400
600
800
1000
Generation
Genetic 1
Genetic 2
Genetic 3
Genetic 4
Genetic 5
Genetic 6
Genetic 7
Genetic 8
Genetic 9
Genetic 10
NEAT 1
NEAT 2
NEAT 3
NEAT 4
NEAT 5
NEAT 6
NEAT 7
NEAT8
NEAT 9
NEAT 10
Parallel 1
Parallel 2
Parallel 3
Parallel 4
Parallel 5
Parallel 6
Parallel 7
Parallel 8
Parallel 9
Parallel 10

Results



Flappy Tiles Fitness Per Generation
Best Player Survival Time (Seconds)
600
400
200
0
20
40
60
80
100
Generation
Genetic 1
Genetic 2
Genetic 3
Genetic 4
Genetic 5
Genetic 6
Genetic 7
Genetic 8
Genetic 9
Genetic 10
NEAT 1
NEAT 2
NEAT 3
NEAT 4
NEAT 5
NEAT 6
NEAT 7
NEAT 8
NEAT 9
NEAT 10
Parallel 1
Parallel 2
Parallel 3
Parallel 4
Parallel 5
Parallel 6
Parallel 7
Parallel 8
Parallel 9
Parallel 10

Results


2048 Fitness Per Generation
Best Player Fitness
8000
6000
4000
2000
0
20
40
60
80
100
Generation
Genetic 1
Genetic 2
Genetic 3
Genetic 4
Genetic 5
Genetic 6
Genetic 7
Genetic 8
Genetic 9
Genetic 10
NEAT 1
NEAT 2
NEAT 3
NEAT 4
NEAT 5
NEAT 6
NEAT 7
NEAT 8
NEAT 10
Parallel 1
Parallel 2
Parallel 3
Parallel 4
Parallel 5
Parallel 6
Parallel 7
Parallel 8
Parallel 9
Parallel 10

Conclusion

- NEAT took longer to find a good solution, but it often outperformed genetic and parallel overall.
- Genetic and parallel found better solutions faster than NEAT, however they didn’t make as much improvement over time.
- Genetic and Parallel performed largely the same.
- The algorithm to choose for a problem depends on what you would prioritize and the specific problem:
    - If you don’t have as much time or processing power you might want to go with genetic and parallel
    - If you want to find a more optimal solution NEAT would be a better option
    - If you can’t find good inputs for NEAT, genetic and parallel could be better
    - If there is random chance involved, go with NEAT

References

[1] Baldominos, A., Saez, Y., Recio, G., & Calle, J. (2015). “Learning levels of Mario AI using genetic Algorithms.” In Lecture notes in computer science (pp. 267–277). https://doi.org/10.1007/978-3-319-24598-0_24

[2] Autin, Russell A., "Super Mario Evolution by the Augmentation of Topology" (2024). University of New Orleans Theses and Dissertations. 3161. https://scholarworks.uno.edu/td/3161

[3] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. Evolutionary Computation, 10(2):99–127, June 2002

[4] Cantú-Paz, Erick. “A Survey of Parallel Genetic Algorithms.” (2000). [PDF] A Survey of Parallel Genetic Algorithms | Semantic Scholar

[5] Katoch, S., Chauhan, S.S. & Kumar, V. A review on genetic algorithm: past, present, and future. Multimed Tools Appl 80, 8091–8126 (2021). https://doi.org/10.1007/s11042-020-10139-6

[6] Sastry, K., Goldberg, D., Kendall, G. (2005). Genetic Algorithms. In: Burke, E.K., Kendall, G. (eds) Search Methodologies. Springer, Boston, MA. https://doi.org/10.1007/0-387-28356-0_4

[7] Code-Bullet. (n.d.). *GitHub - Code-Bullet/NEAT_Template: This is mainly for me, but if anyone wishes to use it then go ahead.* GitHub. https://github.com/Code-Bullet/NEAT_Template

[8] Code-Bullet. (n.d.). *GitHub - Code-Bullet/NEAT_Template: This is mainly for me, but if anyone wishes to use it then go ahead.* GitHub. https://github.com/Code-Bullet/NEAT_Template