|  |  |
| --- | --- |
|  | **Wydział Finansów i Zarządzania** **Kierunek: Informatyka** |

Dawid Miśkiewicz

(numer albumu: 81794)

Temat pracy

The Comparison of Evolutionary Algorithms, Genetic Algorithms, and AI in Flappy Bird Development

Opiekun merytoryczny:

Mgr inż., Sebastian Bień

Wrocław 2024

Table of content

Introduction

1. Flappy Bird Game  
1.1 Flappy Bird Game – basic information

1.2 Tools and technologies used for development

1.3 Structuring the game logic – Creating sprites and animations

1.3.1 Player (Bird):

1.3.2 Obstacles (Pipes):

1.4 Implementing Game Physics

1.4.1 Simulating gravity and player movement

1.4.2 Collision detection between the bird and pipes

1.4.2.1 Mask Creation

1.4.2.2 Mask Offsets

1.4.2.3 Overlap Detection

2. Evolutionary Algorithms

2.1 What it is

2.1.1 Definition and inspiration from natural selection.

2.1.2 Key components: population, fitness, selection, crossover, mutation.

2.2. How does it work

2.2.1 Initialization of the population.

2.2.2 Evaluation of individuals using a fitness function.

2.2.3 Selection of parents for reproduction.

2.2.4 Generating offspring through crossover and mutation.

2.3 Example of use

2.3.1 Solving combinatorial problems (e.g., traveling salesman problem).

2.3.2 Evolving game-playing agents.

2.4 Advantages and Limitations

3. Genetic Algorithms

3.1 What it is

3.2. How does it work

3.2.1 Encoding solutions into a genetic format.

3.2.2 Fitness evaluation and ranking of individuals.

3.2.3 Genetic operators: selection (e.g., roulette wheel, tournament), crossover, mutation.

3.3 Example of use

3.3.1 Feature selection in machine learning.

3.3.2 Evolving strategies for board games.

3.3.3 Parameter optimization in engineering problems.

4. Comparison of Algorithms  
4.1 Evolutionary Algorithms in Flappy Bird  
4.2 Genetic Algorithms in Flappy Bird  
4.3 AI Approaches in Flappy Bird  
4.4 Strengths and Weaknesses of Each Approach

5. Results and Analysis  
5.1 Performance Comparisons  
5.2 Optimization Efficiency  
5.3 Game Performance Evaluation

CONCLUSIONS

BIBLIOGRAPHY

TABLE OF FIGURES

SUMMARY

Introduction

1. Flappy Bird Game

This chapter provides an overview of the development of the Flappy Bird game, highlighting the tools and technologies implemented, the design and implementation of the game architecture, and the core mechanics that govern its functionality.

1.1 Flappy Bird Game – basic information

Flappy Bird is a game developed in 2013 by a Vietnamese programmer Dong Nguyen under his company DOTGEARS COMPANY LIMITED. The game was released in May 2013 on the IOS platform and in January 2014 on the Android platform. At the start of 2014, the game gained an unexpected spike in popularity, leading to mixed reviews caused of its difficulty. The objective of the game is to pass through pipes with a **sprite** using only one button to jump. The player’s score is determined by the amount of pipes passed.

1.2 Tools and technologies used for development

The app was developed in the programming language Python using the game engine Pygame in an integrated development environment Visual Studio Code. The following technologies were used to achieve the goals set in the project:

* Python 3.11.9
* Pygame 2.4.0
* neat-python 0.92

1.3 Structuring the game logic – Creating sprites and animations

The main game loop is managed by main() and manual\_play() functions. Inside the loop, two steps occur. The first one – Event Handling is where the loop processes events such as window events for exiting the game and key presses for jumping the bird.

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

The next step is physics which is handled by the Bird class in the update\_position() method, where gravity and jump mechanics are applied to the bird. The bird’s position is updated for each frame based on velocity that is adjusted with gravity. Pipes move with constant speed and their height is random, as defined in the Pipe class.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

Opis wygenerowany automatycznie

1.3.1 Player (Bird):

The bird is represented using three images (in flight) that have been loaded using pygame.image.load() and scaled to fit inside the window. Animation is controlled in the animation() method inside the Bird class, which shows different bird images according to a counter.



Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

Opis wygenerowany automatycznie

The bird is then drawn with a rotation to simulate tilting during movement. The rotation is handled by pygame.transform.rotate().

1.3.2 Obstacles (Pipes):

Pipes are represented by an image (pipe.png) and have a top and a bottom section. The Pipe class creates, moves, and draws these pipes. Pipes are portrayed through the animation() method of the Pipe class.

Obraz zawierający tekst, Czcionka, zrzut ekranu, numer

Opis wygenerowany automatycznie

Pipes move across the screen in the update\_position() method by decreasing their x position and new pipes are periodically created.

1.3.3 Background and Base:

The background (BG.png) and the base (base.png) are drawn in the draw\_window() function. The base’s horizontal movement creates a scrolling effect, and when it moves out of the screen, it reappears on the opposite side. This is done by the Base class:

Obraz zawierający zrzut ekranu, woda

Opis wygenerowany automatycznie

Obraz zawierający zrzut ekranu, żółty, Prostokąt, zieleń

Opis wygenerowany automatycznieObraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

1.4 Implementing Game Physics

1.4.1 Simulating gravity and player movement   
 In the game, the motion of the bird is governed by a simplified kinematic equation. Said bird’s motion is handled in the PlayerBird class, where gravity and upward movement are defined by the equation presented below. The gravitational effect on the bird’s vertical position (y) is calculated as:

Where:

is the bird’s initial upward velocity of the bird. It is set to -10.5 units when the bird jumps.

is a constant downward gravitational acceleration. The acceleration is approximated to 3 units /

it the time step, tracked as tick\_count.

The position change is calculated as:

Obraz zawierający tekst, Czcionka, zrzut ekranu

Opis wygenerowany automatycznie

To guarantee smooth motion and prevent excessive falling speed, the velocity is capped at a maximum of 16 units:



The bird’s vertical position (self.y) is then updated by adding to its current value:  


When the bird jumps, its velocity is set to an upward value, resisting gravity temporarily. The jump() function resets the parameters to simulate the jump effect:

Obraz zawierający tekst, Czcionka, zrzut ekranu, numer

Opis wygenerowany automatycznie

The bird tilt is also adjusted to reflect its vertical movement visually. When the bird is rising, the tilt is positive, simulating a nose-up effect. Thus, the tilt takes on a positive value during ascension, creating a nose-up impression, and negative during descent, tilting gently downwards until reaching :

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

This guarantees that it appears realistic concerning the effect of gravity in the game.



1.4.2 Collision detection between the bird and pipes

The collision detection mechanism provides an accurate detection between the player (bird) and the game obstacles (pipes). Pixel-perfect collision is achieved by using masks to give a binary representation of the visible pixels of game objects.

Obraz zawierający zrzut ekranu, Prostokąt, sztuka

Opis wygenerowany automatycznie

1.4.2.1 Mask Creation

Masks are generated for the bird as well as both the bottom and top pipes.

Obraz zawierający tekst, Czcionka, zrzut ekranu, linia

Opis wygenerowany automatycznie

1.4.2.2 Mask Offsets

Offsets are calculated to cancel offsets from the bird’s mask to the pipe's masks. The offsets are taking into account the relative positions of the bird and pipes:

Obraz zawierający zrzut ekranu, Czcionka, tekst, linia

Opis wygenerowany automatycznie

1.4.2.3 Overlap Detection

The overlap() method is used to determine whether any pixels in the bird’s mask overlap with the pipe masks:

Obraz zawierający tekst, zrzut ekranu, Czcionka, linia

Opis wygenerowany automatycznie

If any overlap is detected, a collision has occurred:

Obraz zawierający tekst, Czcionka, zrzut ekranu, numer

Opis wygenerowany automatycznie

This method ensures that collisions are accurately detected, even with tricky shapes such as a tilted bird.

2. Evolutionary Algorithm

This chapter describes the technology utilized in the project and the design principles followed by the author while developing and implementing evolutionary algorithms for the thesis.

2.1 What is it?

An evolutionary algorithm is a computer-based approach to problem-solving modeled on natural evolution through the application of computer science and artificial intelligence. It simulates certain biological mechanisms, namely reproduction, mutation, and recombination, in order to solve these problems. Such algorithms are based on Darwin's conception of natural selection whereby weaker solutions are sorted from stronger ones, and the more fit alternatives are preserved and bred, generation after generation. The ultimate goal is the development of solutions that are optimal enough to fulfill the desired purpose.

2.1 Key components: population, fitness, selection, crossover, mutation.

The fundamental components of evolutionary algorithms are:

Population: Set of candidate solutions that evolve with time. In our case, we have the population existing as a list of BirdControl objects.

Obraz zawierający tekst, zrzut ekranu, Czcionka, linia

Opis wygenerowany automatycznie

Fitness Function: A quantitative measure for the assessment of an individual's quality or performance in the population. For example, the fitness of a bird is rated on its performance in the simulation.



Selection Mechanism: A process of identifying and ranking individuals with high fitness for reproduction. In this study, the population is sorted by fitness in descending order.



Crossover: Genetic operation whereby two parents exchange some but not all characteristics in the formation of offspring. The approach employs a simple crossover that weighs features from one of the parents.

Obraz zawierający tekst, zrzut ekranu, Czcionka, linia

Opis wygenerowany automatycznie

**Mutation**: A random or semi-random change applied to individual candidate solutions for creating diversity and thereby avoiding premature convergence. The mutation function is used to alter weights with a certain probability.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

2.2. How does it work

2.2.1 Initialization of the population.

The algorithm begins by initializing a population of candidate solutions. In this study, each solution represents the neural network weights of a simulated bird agent. These weights are initialized randomly to ensure diversity in the initial generation.

Obraz zawierający tekst, zrzut ekranu, Czcionka, linia

Opis wygenerowany automatycznie

2.2.2 Evaluation of individuals using a fitness function.

The evaluation starts from an initial population, depending on the candidate’s solutions. In this case, each solution is just the weight of a neurally driven simulated bird agent. Initially, all the candidate solutions undergo random initialization so as to secure some degree of diversity in the first generation.



2.2.3 Selection of parents for reproduction.

Individuals with higher fitness are more likely to be selected for reproduction. This selection mechanism ensures that good genes are carried over to the next generation.



2.2.4 Generating offspring through crossover and mutation.

Reproduction involves creating offspring by combining traits of selected parents using crossover and mutation. These operations introduce variation and enable exploration of the solution space.



2.3 Example of use

2.3.1 Solving combinatorial problems (e.g., traveling salesman problem).

Evolutionary algorithms are well-suited for solving combinatorial problems, such as the traveling salesman problem (TSP). By evolution on the order of the sequences of cities, EAs are capable of efficiently minimizing the total travel distance in very large-size instances.

2.3.2 Evolving game-playing agents.

In the case of evolving game-playing agents, the agents gradually learn how to overcome obstacles using increasingly efficient methods from one generation to the next, which is reflected in their higher overall performance. The algorithm's ability to optimize the neural network weights controlling the decision-making process of birds offers promising perspectives for the use of intelligent systems.

2.4 Advantages and Limitations

Evolutionary algorithms are characterized by their strengths and numerous applications, making them invaluable in several fields of engineering and optimization problems. However, specific challenges should be taken into consideration. This section provides a brief description of the major benefits and limitations experienced during the implementation and tests on evolutionary algorithms in this study.

2.4.1 Advantages

* Versatility: Capable of addressing a wide range of optimization problems.
* Global Search Capability: Effectively explores complex solution spaces, reducing the likelihood of being trapped in local optima.
* Adaptability: Easily integrates with various problem domains, including game AI and combinatorial optimization.

2.4.2 Limitations

* Computational Cost: High resource requirements for large populations or complex fitness evaluations.
* Convergence Challenges: Risk of premature convergence, necessitating careful tuning of algorithm parameters.

3. Genetic Algorithms

3.1 What it is

A genetic algorithm (GA) is an evolutionary algorithm inspired by natural selection and genetics; it provides a basis for finding the optimal solution or near-optimal solutions for complex problems. In the context of this study, the problem involves the optimization of an AI agent (the bird) capable of playing the Flappy Bird game with extreme effectiveness. The GA receives training through a process imitative of biological evolution, wherein a population of candidate solutions (genomes) is taken through successive generations of evolution. In our implementation, an evolutionary algorithm called NEAT (NeuroEvolution of Augmenting Topologies) employs an interesting way of evolving neural networks. Derived from biological inspiration, NEAT stretches the boundaries of basic neuron weight evolution by including the evolution of the structure called topology within the neural networks. Because of this, NEAT is able to discover progressively solutions that configure quite adaptable and efficient neural networks. The solutions in NEAT are represented as genomes that encode neural network structures and their associated weights. A genome in NEAT contains a collection of genes representing the interconnections and properties of neurons. This encoding scheme is directly mapped onto the neural networks controlling the movements of the flying objects.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

3.2. How does it work

The process of evolving the bird’s neural network follows stages such as encoding solutions, fitness evaluation, and genetic operators.

3.2.1 Encoding solutions into a genetic format.

Each bird’s neural network is characterized as an aspect of a genome-relating collection of genes for describing the neural network structure and weight parameters. In NEAT, this genome incorporates both the topology (nodes and connections) and the weights (parameters) of the neural network. The network decides a bird's action in the game on the basis of its sensory inputs.

Obraz zawierający tekst, zrzut ekranu, Czcionka, oprogramowanie

Opis wygenerowany automatycznie

3.2.2 Fitness evaluation and ranking of individuals.

Based on how long they live through the game, each bird (individual) is assigned a fitness score. The longer each bird can survive, as measured by passing successfully through pipes, the better its fitness. If a bird collides with a pipe or falls to the ground, its fitness is penalized.

Obraz zawierający zrzut ekranu, tekst, Czcionka

Opis wygenerowany automatycznie

3.2.3 Genetic operators: selection (e.g., roulette wheel, tournament), crossover, mutation.

Neat uses three primary genetic operators: selection, crossover, and mutation. The selection process essentially selects the best individuals to pass their genes. Crossover is the joining of genetic material from two parents to form offspring, while mutation introduces random changes in the genome for the sole purpose of creating diversity in the population.



The population is created with a specific configuration.



Run for 10 generations, where selection and mutation occur.

3.3 Example of use

In this code, the genetic algorithm is used to create an AI capable of navigating the Flappy Bird environment. This application serves as a good example of tasks such as feature selection, strategy evolution, and parameter optimization performed by genetic algorithms.

3.3.1 Feature selection in machine learning.

Even if this can hardly be counted as a typical machine learning problem, the evolution applied to the neural network of the bird can be considered the optimization of inputs (vertical position of the bird and distances from the pipe) to the actions, that is, whether the bird should stay still or jump. This can be related to feature selection, where the system is trying to find the best representation of input features for making decisions.



3.3.2 Evolving strategies for board games.

The evolving of strategies is conceptually similar to the AI systems that play board games, exactly like how AI systems evolve the strategies to play chess or Go. The bird strategies are evolved in the task of traversing pipes, while the AI systems solve it in a practically equal approach. Each bird strategy is governed by a neural network that learns to "decide" when to jump based on varying inputs in the environment.

Obraz zawierający Czcionka, tekst, zrzut ekranu, Grafika

Opis wygenerowany automatycznie

3.3.3 Parameter optimization in engineering problems.

The evolution of the bird controller neural network is like parameter optimization in engineering problems. Neural network weights are varied over generations to get the bird to stay alive longer. One can think of it as optimizing design parameters in an engineering context in order to attain better performance under certain conditions.

3.4 Using NEAT to Train a Flappy Bird AI

3.4.1 Overview of NEAT in Flappy Bird

NeuroEvolution of Augmenting Topologies (NEAT) algorithm to the development of an Adaptive AI for the control of Flappy Bird game by the bird. It uses evolutionary principles to optimize neural-architecture-based complex networks through genetic operations. In this implementation, NEAT optimizes the structure and weights of the neural network of the bird so that there are iterative generations where improvements are measured by performance in overcoming obstacles (pipes) over time. The main function, run\_neat, initializes the NEAT process by loading configuration settings and running the evolutionary process through p.run(). Here, the process encompasses the modeling of multiple generations where genomes compete and adapt.

screen

This code guarantees NEAT to adapt and find an efficient genome to control the bird efficiently.

3.4.2 Inputs and Neural Network Design

The design of the neural networks built in the project has a feed-forward structure through which the environment surrounding the bird is analyzed, and then decisions are made. The inputs of the network include the following three vital inputs:  
The vertical position of the bird is represented by bird.y.  
Vertical distance to the top of the next pipe is given by: abs(bird.y - pipes[pipe\_ind].height).  
Vertical distance to the bottom of the next pipe is given by: abs(bird.y - pipes[pipe\_ind].bottom).

Deep inside the network, these three inputs contribute to outputting a single number. If the output is greater than 0.5, the bird "jumps," which is essentially done by calling the jump() function inside the PlayerBird class. The architecture of the neural network is defined in the config-feedforward.txt file, specifying three input nodes, one output node, and other hyperparameters for mutation and evolution.

Screen

The network architecture allows the mapping of the inputs onto an action using a simple and effective mechanism, allowing quick adaptation via NEAT.

3.4.3 Fitness Evaluation Strategy

Fitness evaluation is a chief part of NEAT; its understanding distinguishes between the genomes that will most likely pass their genes on to the subsequent generation. In this implementation, the fitness function rewarded survival and punished collisions. Each bird would obtain a small fitness increment (0.1) for each frame it lived in and an additional bonus (5 points) for passing a pipe successfully. On the contrary, colliding into a pipe or the ground takes some fitness away from the bird.  
  
Screen  
  
Genomes with higher fitness values are more likely to survive and pass their traits onto subsequent generations, thus paving the way for evolutionary progress.

3.4.4 Generation Lifecycle

NEAT divides itself into initializing genomes, evaluating their fitness, and performing genetic operations to produce the next generation. Each generation begins by creating neural networks for all genomes; then, each is associated with the appropriate bird agent. During the game simulation, these agents interact with the environment, accumulating fitness based on their performance and survival rate. Subsequent to each generation, NEAT cyclically operates crossover and mutation by splicing two genomes together to produce new genomes. Poor performers are culled from the gene pool, letting only the strongest individuals contribute to the next generation. The cycle will keep repeating until the population meets with a supplied fitness threshold or the maximum number of generations is attained.  
  
Screen  
  
This ensures the trial from generation to generation grows, dealing with the bird's decision-making powers.

3.4.5 Real-Time Game Interaction

The administration of the evolved neural network over the bird dynamic in-game is possible via real-time interactions. The loop keeps updating the position of the bird, checking for collision, and rendering the game window. Draw window is the one that handles all sorts of rendering for the game components-bird, pipes, base, and score.   
  
Screen  
  
In the gameplay of this implementation, NEAT's neural network expresses its output. For instance, on the signal of jumping, the associated jump function for the bird gets called:  
  
Screen

Real-time interactions allow the AI agent to seamlessly adapt to the changing setting of Flappy Bird.

4. Comparison of Algorithms

The Flappy Bird has been the testbed for various other methods in AI training, preferably due to non-trivial dynamics and rapid decision-making in uncertain environments. In this section, we provide a particularly thorough comparison of different algorithms applied to Flappy Bird, namely Evolutionary Algorithms (EAs), Genetic Algorithms (GAs), and AI-based approaches, including Reinforcement Learning and Neural Networks. Where benefits and drawbacks exist, we aim to highlight and discuss them. Hence, a more rounded impression of their applicability is provided.

4.1 Evolutionary Algorithms in Flappy Bird  
Evolutionary Algorithms (EAs) are optimization techniques based on Darwinian principles of natural selection. Concerning the game "Flappy Bird," EAs are used to develop bird controllers which will improve their survival strategies to gain more points in the game. In contrast to gradient-based approaches, EAs treat the game environment as a black-box function, thus allowing them more flexibility against non-linear, noisy, and unpredictable problems. The algorithm refines the population of candidate solutions (bird controllers) iteratively, evaluating each one's fitness in the gameplay loop, and applying biologically inspired operations of selection, crossover, and mutation to produce new generations. Fundamental Mechanisms in the Code:  
Initialization: It creates a population of bird controllers where every bird is initialized with a random set of weights representing its decision-making capabilities. Do take into account that the resulting weights would determine the bird's response towards game inputs such as height difference between the bird and the next obstacle. This diversity creates a wide space for search at the beginning, increasing the chance of finding successful strategies.  
Fitness Evaluation: Every individual bird gets to play the game separately. Performance is then measured by the fitness function as a function of time survived and score. The surviving strategies are rewarded/disciplined according to collision avoidance and gameplay progression. The fitness function usually will have multiple objectives mixed in: survival time maximization and movement reduction, but not by its arbitrary measures (in this case, needing them only at certain moments during the process).  
Selection: The subset of the fittest birds is selected to reproduce. Therefore, methods such as rank-based selection or tournament selection may also be implemented, and great performers are ensured visibility for each respective generation.  
Crossover: The genetic material from the two parent birds is combined to simulate sexual reproduction to produce offspring. Crossover in the code may employ techniques such as one-point crossover or uniform crossover to merge the weights of the two parents while ensuring that offspring inherit traits from each parent.  
Mutation: Some uniform noise is then introduced randomly into the weights of the offspring to add additional genetic diversity. The mutation prevents the stagnation of the population into some local optimum and promotes search space exploration.   
Termination and Convergence: For a fixed number of generations, the algorithm terminates after the fitness of the best-performing bird converges, which indicates that the algorithm converged to an optimal or near-optimum solution.  
4.1.1 In-Depth Strengths  
Adaptive: Black-box nature allows the EA to perform remarkably well under Flappy Bird without the availability of an explicit mathematical model of the environment.  
Explorative: Population-based approaches ensure a wider search of potential strategies and promote diversity and robustness in the solutions.  
Flexible: They can deal with multiple conflicting objectives, such as maximizing score while minimizing energy expenditure (if modeled).  
4.1.2 In-Depth Weaknesses  
Computational Costs: For large populations and longer simulations, searching the tower through repeated rounds makes it quite resource-intensive to assess the populations all the time.  
Parameter sensitivity: Different settings for mutation rate, crossover probability, and population size largely determine how an EA operates. Values that are not properly set can make the algorithm converge too quickly or steeply, thereby showing excessive computational behavior.   
Convergence Speed: While robust, EAs may converge more slowly than gradient-based optimizations on some problems, particularly in high-dimensional spaces.

4.2 Genetic Algorithms in Flappy Bird. Sprawdz tool !!!

Genetic algorithms (GAs) are a more ordered subset of EAs, especially created for the optimization problem with well-defined genetic representation. In Flappy Bird, GAs aim at evolving bird controllers, treating weights of their neural networks or decision rules as genes in a chromosome. The most striking distinction between GAs and generic EAs is found in applying genetic operators in a systematic way: representing solutions as chromosomes while using specialized selection, crossover, and mutation strategies, among other mechanisms. Key Mechanisms in the Code:  
Solution Encoding: Encoding weights of bird controllers into chromosomes provides a more structured representation for the application of genetic operators. In the code, this chromosome can represent a fixed-length vector of weights, with each gene corresponding to a particular neural network parameter.  
Parent Selection: Using appropriate means of fitness-proportionate selection, like roulette-wheel or tournament selection, to identify high performers for reproduction. The selection of parents allows more favorable genetic contributions for the solution to pass on to the next generation, allowing at least some less favorable individuals to compete against them and avoid premature convergence.  
Crossover methods: This is where the parents exchange genetic material, with overwriting and one-point techniques being two examples of crossover methods often used in evolutionary computation.  
The code may use:   
 One-Point Crossover: a single point on the parent's chromosome is chosen, and the resulting segments are exchanged by the two parents.  
 Uniform Crossover: Every gene is chosen independently from one of the two parents with a fixed probability.   
Mutation Operations: Variability is introduced by low-probability mutations changing genes. In the Flappy Bird GA, for example, it may be done by perturbing the value of weight by a small random amount or replacing the weight with a new randomly generated one. Mutation ensures genetic diversity and keeps the population from getting stuck at local optima.   
Fitness Evaluation:  
As with EAs, it is the fitness function that determines how long each bird will survive and its score. Well-defined and quantifiable metrics ensure that the genetic algorithm has a clear goal of optimization.  
Termination and Optimization:  
The algorithm runs for a fixed number of generations or until stabilization in the fitness of the population has occurred, which means further evolution is unlikely to significantly improve it.

4.2.1In-Depth Strengths  
Efficiency through Structure: GAs exploit the structured representation of solutions and specialized genetic operators to explore and exploit efficiently compared to generic EAs.  
Systematic Genetic Diversity: GAs combine diverse genetic material through crossover and maintain it through mutation to have an effective balance in exploration and exploitation.

Scalability to Discrete Problems: GAs are very effective in discrete or combinatorial optimization problems where the solutions can be naturally encoded as chromosomes.

4.2.2 In-Depth Weaknesses  
Encoding Challenges: Continuous or high-dimensional problems, such as large neural network optimization, require solution encoding into chromosomes with much care, lest inefficiencies or important solution details be lost. Premature Convergence: Without sufficient diversity in the population or robust mutation mechanisms, GAs may easily stagnate at local optima, reducing their effectiveness.   
Sensitivity to Parameter Tuning: Similar to EAs, the success of GAs depends on appropriately setting parameters such as mutation rate, crossover probability, and population size. Improper settings degrade performance or lead to excessive computational demands.  
Comparison to Evolutionary Algorithms: Whereas the common principles of GAs and EAs exist, GAs are more regular and systematic, often performing better than generic EAs for problems with well-defined solution encodings. EAs, however, can provide more flexibility in black-box or poorly defined problem spaces where solutions may not easily fit into the chromosome structure required by GAs.

4.3 AI Approaches in Flappy Bird  
In the design of AI for Flappy Bird, the majority depend on reinforcement learning and some on neural networks. While most reinforcement learning-based models and those that rely on neural networks operate on having agents learn through trial-and-error learning to maximize rewards across multiple game plays, none of them apply these Evolutionary and Genetic Algorithms-like techniques. The process with AI approaches is instead strongly directed in a more algebraic process for quick convergence.

Two of the most famous AI methodologies tested on Flappy Bird include Q-Learning and Neural Networks, including hybrid variants such as Deep Q-Learning.  
4.3.1 Q-Learning  
Mechanics and Implementation:  
Q-Learning is a model-free reinforcement learning algorithm that learns to take actions in given states. It maintains a Q-table, in which each entry represents an expected future reward for an action-state pair.  
In Flappy Bird:  
State Representation: The state can be represented by the height of the bird, its velocity, and the position of the next obstacle. Actions: There are two possible actions for the bird: "flap" or "do nothing." Reward Signal: Rewards condition on survival time and scores; penalties for collisions. The algorithm updates the Q-values using the Bellman equation:

4.4 Strengths and Weaknesses of Each Approach