A Project report on

Game Agent using Genetic Algorithm

A Dissertation submitted to JNTU Hyderabad in partial fulfillment of the academic requirements for the award of the degree.

Bachelor of Technology
In
Computer Science and Engineering
(Artificial Intelligence and Machine Learning)

Submitted by

Akshit Rao
(20H51A6666)

Sai Kiran
(20H51A6662)

Prateet Mandhata
(20H51A66B5)

Under the esteemed guidance of

Mr. K. Sudhakar Reddy
(Assistant Professor)

Department of Computer Science and Engineering (AI & ML)

CMR COLLEGE OF ENGINEERING & TECHNOLOGY
(UGC Autonomous)
(Approved by AICTE, Permanently Affiliated to JNTUH, NAAC Accredited with 'A+' Grade)
Kandlakoya, Medchal Road, Hyderabad-501401

2020- 2024

CMR COLLEGE OF ENGINEERING & TECHNOLOGY

KANDLAKOYA, MEDCHAL ROAD, HYDERABAD – 501401

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (AI & ML)

CERTIFICATE

This is to certify that the Major Project Phase-I report entitled "**Game Agent using Genetic Algorithm**" being submitted by **P. Sai Kiran (20H51A6662), Akshit Rao (20H51A6666), Prateet Mandhata (20H51A66B5)** in partial fulfillment for the award of Bachelor of Technology in Computer Science and Engineering (AI&ML) is a record of bonafide work carried out his/her under my guidance and supervision. The results embodied in this project report have not been submitted to any other University or Institute for theaward of any Degree.

Mr. K. Sudhakar Reddy	**Dr. P. Sruthi**	**External**
Assistant Professor	**Associate Professor and HOD**	
Dept of CSE (AI & ML)	**Dept of CSE(AI & ML)**	

ACKNOWLEDGEMENT

With great pleasure we want to take this opportunity to express my heartfelt gratitude to all the people who helped in making this project a grand success.

We are grateful to **Mr. K. Sudhakar Reddy**, Assistant Professor in Computer Science and Engineering (AI&ML) for her valuable technical suggestions and guidance during the execution of this project work.

We would like to thank **Dr. P. Sruthi**, Head of the Department of Computer Science and Engineering (AI&ML), CMR College of Engineering and Technology, who are the major driving forces to complete our project work successfully.

We are very grateful to **Dr. G. Devadas**, Dean-Academics, CMR College of Engineering and Technology, for his constant support and motivation in carrying out the project work successfully.

We are highly indebted to **Dr. V A Narayana**, Professor in CSE and Principal, CMR College of Engineering and Technology, for giving permission to carry out this project in a successful and fruitful way.

We would like to thank the **Teaching & Non- teaching** staff of Department of Computer Science and Engineering (AI&ML) for their co-operation.

We express our sincere thanks to **Mr. Ch. Gopal Reddy**, Secretary, CMR Group of Institutions, for his continuous care.

Finally, we extend thanks to our parents who stood behind us at different stages of this Project. We sincerely acknowledge and thank all those who gave support directly and indirectly in the completion of this project work.

Sai Kiran	20H51A6662
Akshit Rao	20H51A6666
Prateet Mandhata	20H51A66B5

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

The Mario AI project is a groundbreaking endeavor that utilizes genetic algorithms to train an AI agent to excel in the Super Mario game. Inspired by the mechanisms of natural selection, these algorithms work iteratively across multiple generations to refine the AI agent's gameplay strategies. Each generation sees the agent's decision-making processes, such as jumping and navigating obstacles, governed by a unique genetic encoding meticulously crafted for optimal performance.

Through the simulation of natural selection, crossover, and mutation, the AI agent's strategies undergo a continuous process of enhancement. This iterative refinement leads to increasingly adept gameplay as the agent learns to adapt to the dynamic challenges presented by the game environment. Evaluation takes place across a variety of Mario levels, with rewards linked to the agent's ability to progress through levels, survive encounters with enemies and hazards, and achieve high scores.

As successive generations unfold, the genetic algorithm pushes the AI agent to develop ever more effective strategies, honing its skills to mastery. Over time, the agent gains a profound understanding of the intricacies of the game, demonstrating unparalleled proficiency in navigating its obstacles and achieving objectives. The Mario AI project not only showcases the power of genetic algorithms in training AI agents but also highlights their potential for solving complex problems in real-world applications.

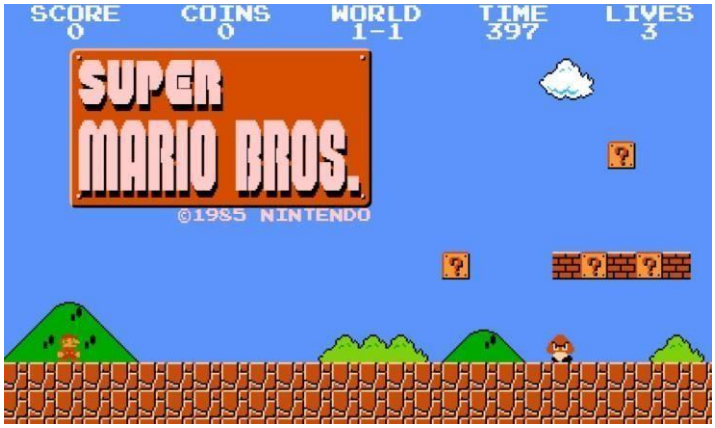


Fig-1 Super Marios

CHAPTER 1

INTRODUCTION

CHAPTER 1
INTRODUCTION

1.1 Problem Statement

The challenge addressed by Project Mario AI lies in developing an artificial intelligence (AI) system capable of autonomously playing the classic Super Mario Bros video game. Traditional gaming relies on human interaction, but this project aims to overcome the limitations of manual gameplay by creating an intelligent agent that can navigate through complex game levels, interact strategically with the environment, and make informed decisions. The problem is to design an AI solution using Python, Genetic algorithm, and advanced machine learning techniques to train an agent that can progressively learn, adapt, and excel at playing Super Mario Bros., showcasing the untapped potential of AI in gaming applications.

1.2 Research Objective

The primary objective of this research is as follows.

- Create an AI agent proficient in playing Super Mario Bros. from scratch, showcasing the potential of AI to autonomously interact with and master complex video game environments.
- Leverage genetic algorithm as a fundamental methodology, enabling the agent to learn and enhance its actions iteratively based on feedback from the gaming environment.
- Document the development process, challenges faced, and insights gained throughout the project to contribute valuable knowledge for others interested in AI-based game development projects.
- Ensure the project's reproducibility by documenting progress and methodologies, enabling others to comprehend and replicate similar AI-based game development initiatives.

In essence, the research objective is to showcase the fusion of Python programming, genetic algorithm , and game development, providing insights into the capabilities and potential applications of AI in gaming and interactive environments.

1.3 Project Scope and Limitations

Project Scope:

The scope of Project Mario AI is to develop an artificial intelligence (AI) system using Python programming language that excels in playing Super Mario Bros. This encompasses understanding the game's mechanics, implementing genetic algorithm techniques like reinforcement learning, and designing custom reward systems to optimize the AI agent's gameplay. The project involves continuous evolution , experimentation with advanced neural network architectures, and documentation of insights and challenges faced during development. The scope extends to showcasing Python's versatility in AI and game development, with the goal of delivering a proficient AI agent capable of navigating and succeeding in the iconic Super Mario Bros. game.

Limitations:

While the project's scope is confined to the classic Super Mario Bros. game, potentially limiting the generalizability of the developed artificial intelligence (AI) agent to more intricate gaming environments. The reliance on in-game rewards and penalties for the agent's learning may be constrained by the lack of a diverse dataset, potentially affecting its adaptability to unforeseen situations. While genetic algorithm is utilized, the algorithmic complexity and potential challenges in fine-tuning could impact the agent's effectiveness. Additionally, the resource-intensive nature of training, particularly when exploring advanced techniques like convolutional neural networks or recurrent neural networks, may pose accessibility issues. The project's documentation efforts aim for reproducibility, but variations in hardware and software configurations may present challenges in this regard. Furthermore, the AI agent's proficiency is tailored to Super Mario Bros., limiting its applicability to other games, and the mutation to next generation introduces potential biases or decision-making limitations. These considerations highlight the project's boundaries and suggest areas for genetic algorithm and future exploration.

CHAPTER 2

BACKGROUND WORK

CHAPTER 2

BACKGROUND WORK

2.1 Rule Based Systems:

2.1.1 Introduction

Rule-based AI for Mario involves creating a system where Mario's actions are determined by a set of predefined rules. These rules govern every aspect of Mario's behavior, from movement and jumping to interactions with enemies and obstacles. By analyzing the game state, the AI selects the most appropriate rules to apply in any given situation. Rule priority ensures that critical actions, like avoiding enemies, take precedence over less important tasks, such as collecting coins. The AI's decision-making process relies on evaluating variables such as Mario's position, enemy positions, and the layout of the level. Through iterative refinement and optimization, the AI improves its performance over time. Handling edge cases, such as power-ups and complex level designs, is crucial for the AI's adaptability. While rule-based systems offer simplicity and transparency, they may struggle with the complexity of dynamic game environments. Nevertheless, continuous feedback and refinement enhance the AI's capabilities, making it a formidable Mario player."

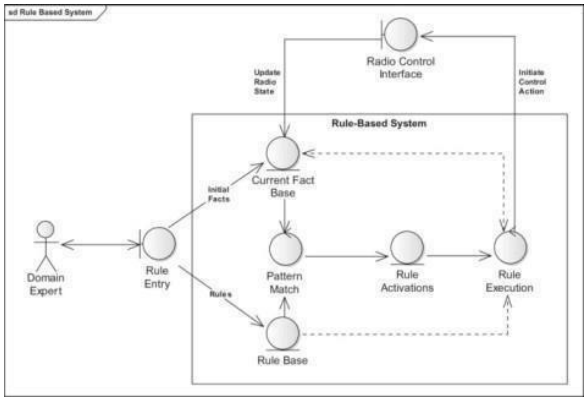


Fig 2.1 Rule-Based

2.1.2 Merits, Demerits and Challenges of Existing Systems

Edge Rule-Based Systems offer interpretability, granting transparency and insight into the AI's decision-making process through explicitly defined rules. This transparency allows developers and players to understand why the AI behaves in certain ways, making it easier to debug and fine-tune its performance. Additionally, since rules are directly mapped to actions, developers have precise control over the AI agent's behavior, which can be advantageous in specific gaming scenarios where strategic planning or fine-tuned adjustments are crucial. For instance, in a game like Mario, where precise platforming and enemy interactions are essential, having control over the AI's actions can lead to more predictable and manageable gameplay experiences. Furthermore, interpretability facilitates collaboration between developers and AI systems, as it enables clear communication and troubleshooting of issues. Overall, the interpretability offered by ``rule-based systems enhances the usability and effectiveness of AI agents in gaming and other domains.

2.1.3. Implementation

Movement Rules: Define rules for basic movement, such as moving left or right, jumping, and crouching. These rules might include conditions like "if the player presses the left arrow key, move Mario left" or "if there is a gap in the ground ahead, jump."

Interaction Rules: Create rules for interacting with objects in the game world, such as collecting coins, stomping on enemies, or picking up power-ups. These rules could specify actions like "if Mario's position overlaps with a coin, collect the coin" or "if Mario lands on top of an enemy, defeat the enemy."

Enemy Avoidance: Implement rules for avoiding enemies to prevent Mario from taking damage. This might involve rules like "if there is an enemy in Mario's path, jump to avoid it" or "if an enemy is approaching from above, move out of the way."

Level Navigation: Develop rules for navigating through the level, such as avoiding pits, navigating platforms, and reaching the end goal. These rules

Power-up Management: Include rules for managing power-ups, such as using them strategically to gain advantages or overcome obstacles. For example, "if Mario collects a mushroom power-up, increase Mario's size" or "if Mario collects a fire flower power-up, allow Mario to shoot fireballs."

Power-up Management: Include rules for managing power-ups, such as using them strategically to gain advantages or overcome obstacles. For example, "if Mario collects a mushroom power-up, increase Mario's size" or "if Mario collects a fire flower power-up, allow Mario to shoot fireballs."
Rule Prioritization: Establish the priority of rules to handle conflicting situations. For instance, prioritizing enemy avoidance over collecting coins ensures Mario's survival takes precedence.

Iterative Refinement: Test the AI within the game environment and refine rules based on performance. This may involve adjusting rule priorities, adding new rules, or tweaking existing ones to improve Mario's behavior.

Edge Case Handling: Account for edge cases and unexpected scenarios in the rule set, such as complex enemy patterns or unique level designs.

Optimization: Optimize the rule-based system for efficiency to ensure it can process the game state and make decisions in real-time.

Feedback Loop: Continuously gather feedback from AI performance and refine rules accordingly to enhance Mario's capabilities

2.2 Reinforcement Learning:

2.2.1 Introduction

In a reinforcement learning-based system for Mario AI, the agent learns from game environment interactions, receiving feedback as rewards or penalties. Over time, it adjusts its decision-making policy to maximize cumulative rewards, using techniques like Q-learning. Initially exploring randomly, it gradually shifts towards exploiting known rewarding actions while occasionally exploring new strategies. Through iterative refinement, the agent adapts its strategy to navigate levels, defeat enemies, and achieve objectives efficiently. It generalizes learned strategies across different scenarios, enhancing robustness and versatility. Continuous improvement occurs as the agent refines its strategy based on new experiences, ultimately achieving high proficiency in playing Mario.

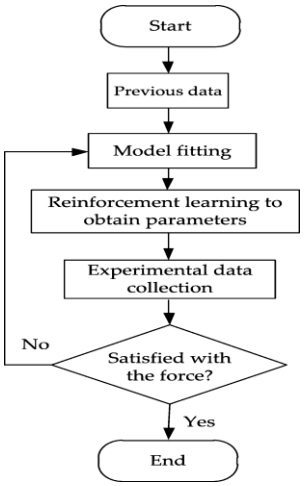


Fig 2.2 Reinforcement learning

2.2.2 Merits, Demerits and Challenges of Existing System

Incorporating reinforcement learning empowers the AI agent to dynamically adjust its actions by learning from the consequences of its decisions within the game environment. Through continuous interaction, the agent receives feedback in the form of rewards or penalties, guiding it to improve its performance iteratively. By leveraging past experiences and learned knowledge, the agent adapts its behavior to effectively navigate various challenges encountered in the game. This adaptive capability allows the AI to evolve its tactics in response to changing circumstances, enhancing its proficiency and versatility. Ultimately, the integration of reinforcement learning enables the AI agent to autonomously refine its decision-making processes, resulting in more intelligent and adaptive gameplay that provides a richer and more immersive experience for players.

2.2.3. Implementation

Environment Setup: Create a simulation environment that accurately represents the Mario game mechanics, including Mario's movement, enemy behavior, level layout, and objectives.

State Representation: Define a state representation that captures relevant information about the game state, such as Mario's position, enemy positions, power-up status, and level layout. This representation should provide sufficient information for the AI agent to make decisions.

Action Space: Define the set of actions that the AI agent can take in the game environment, such as moving left or right, jumping, and using power-ups.

Reward Design: Design a reward structure that provides feedback to the AI agent based on its actions. Rewards should incentivize behaviors that lead to progress in the game, such as collecting coins, defeating enemies, and reaching the end of the level, while penalizing actions that result in loss of health or failure to progress.

Agent Architecture: Choose a reinforcement learning algorithm and architecture for the AI agent, such as Q-learning, Deep Q-Networks (DQN), or Policy Gradient methods. This involves designing the neural network architecture, defining the learning algorithm, and setting hyperparameters.

Training Process: Train the AI agent in the simulation environment using reinforcement learning. During training, the agent interacts with the environment, observes the current state, takes actions, receives rewards, and updates its policy to maximize cumulative rewards over time.

Exploration vs. Exploitation: Balance exploration (trying new actions) and exploitation (selecting actions with the highest expected rewards) during training to ensure that the agent learns robust strategies without getting stuck in local optima.

Evaluation: Evaluate the trained AI agent's performance on unseen levels or environments to assess its generalization capabilities and effectiveness in real gameplay scenarios.

Fine-tuning and Optimization: Fine-tune the AI agent's parameters and training process to improve performance and efficiency. This may involve adjusting hyperparameters, modifying reward structures, or augmenting the state representation.

Integration and Deployment: Integrate the trained AI agent into the actual Mario game environment, either as a standalone AI player or as part of a larger game AI system. Monitor its performance and behavior in real-time gameplay and make adjustments as needed

CHAPTER 3

PROPOSED SYSTEM

CHAPTER 3

PROPOSED SYSTEM

3.1 Objective of Proposed Model

The proposed method aims to harness the capabilities of genetic algorithm within the context of Super Mario Bros. Using Python as the foundational programming language, the objective is to implement a robust AI system capable of autonomously navigating the game's challenges. Through the iterative process of genetic algorithm, the system will train the AI agent to enhance its decision-making based on real-time feedback. The key objectives include designing and implementing effective reward systems that motivate the agent to complete levels, collect coins, defeat enemies, and optimize overall performance.

Additionally, the project will explore advanced techniques to continually improve the agent's gameplay. The overarching goal is to create an intelligent agent that not only conquers the complexities of Super Mario Bros. but also serves as a testament to the potential of AI in gaming and interactive applications.

3.2 Algorithms used for Proposed Model

The proposed method integrates cutting-edge algorithms in the field of artificial intelligence, specifically tailored for the challenges posed by Super Mario Bros. The primary algorithm employed in this project include:

Representation: Define a way to represent potential solutions (individuals) in the genetic algorithm. This could involve encoding Mario's actions or behaviors as a sequence of genes, where each gene represents a specific action or decision.

Initial Population: Generate an initial population of individuals with random or predefined genomes. These individuals represent potential solutions to the problem of playing Mario.

Fitness Function: Design a fitness function to evaluate the performance of each individual in the population. The fitness function should measure how well each individual plays Mario, considering factors such as level completion time, coins collected, enemies defeated, and overall survival.

Selection: Use selection techniques such as tournament selection or roulette wheel selection to choose individuals from the population for reproduction based on their fitness scores.

Individuals with higher fitness scores are more likely to be selected for reproduction.

Crossover: Apply crossover (recombination) to pairs of selected individuals to create offspring. Crossover involves exchanging genetic information between parents to generate new candidate solutions.

Mutation: Introduce random changes (mutations) to the genomes of offspring to promote diversity in the population and prevent premature convergence to suboptimal solutions. Mutation rates should be low enough to preserve good solutions but high enough to explore new regions of the search space.

Replacement: Replace the least fit individuals in the current population with the offspring generated through crossover and mutation, creating the next generation of individuals.

Iteration: Repeat the selection, crossover, mutation, and replacement steps for a fixed number of generations or until a termination condition is met (e.g., a solution of sufficient quality is found).

Convergence: Monitor the convergence of the genetic algorithm by tracking the fitness scores of the best individuals in each generation. Convergence indicates that the algorithm has found a good solution or reached a plateau in performance improvement.

Evaluation and Refinement: Evaluate the performance of the best individual(s) produced by the genetic algorithm on unseen Mario levels or environments. Refine the parameters of the genetic algorithm and the representation of individuals based on performance feedback to improve results.

empowering the AI agent to navigate the Super Mario Bros. game intelligently and autonomously. The integration of these advanced techniques showcases the project's commitment to pushing the boundaries of AI in the gaming domain.

3.3 Designing

3.3.1 Block Diagram

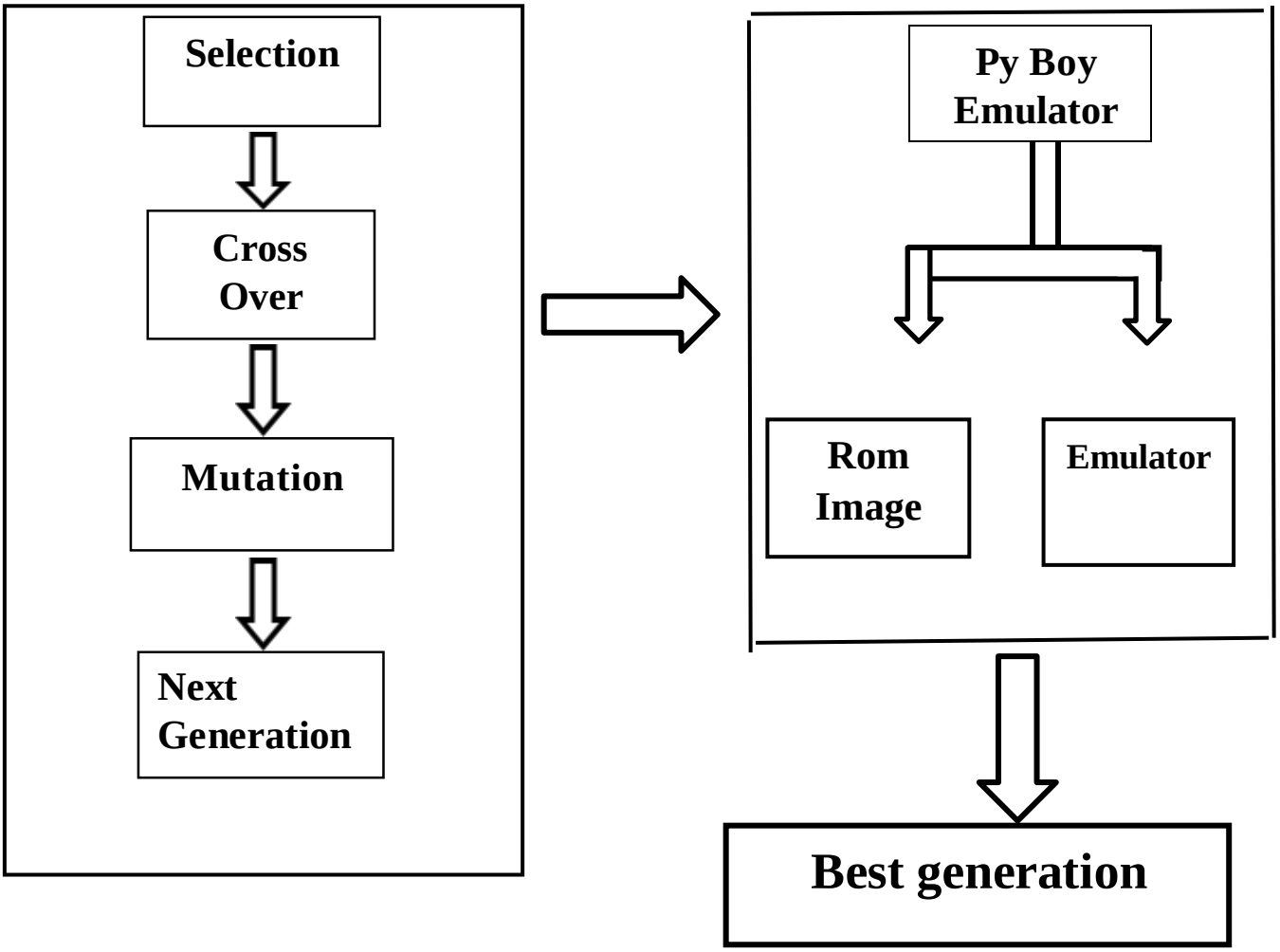


Fig -3.3 Block diagram

3.4 Stepwise Implementation and Code

Implementing a genetic algorithm for a Mario AI involves several steps. Below is a simplified, step-by-step guide. Note that the actual implementation details can vary based on the specific tools and frameworks you are using. Additionally, this example assumes the use of neural networks to control Mario's actions.

1. Initialize Population:

- Generate an initial population of neural networks. Each neural network represents a potential AI controller for Mario.

2. Game Simulation:

- Simulate each individual (neural network) in the population to play the Mario game. Evaluate their performance based on predefined criteria, such as the distance traveled, coins collected, or survival time.

3. Fitness Evaluation:

- Assign a fitness score to each individual based on their performance in the game. Higher scores should indicate better performance.

4. Selection:

- Select individuals from the current population to be parents for the next generation. The probability of selection should be proportional to their fitness scores.

5. Crossover (Recombination):

- Create offspring by combining the genetic material (neural network weights) of selected parent individuals. This emulates the crossover process in genetic algorithms.

6. Mutation:

- Introduce random changes (mutations) to the offspring's neural network weights. This introduces diversity into the population.

7. Replace Weaker Individuals:

- Replace some individuals in the current population with the newly created offspring. The selection of individuals for replacement can be based on their fitness scores.

8. Termination Check:

- Check if a termination criterion is met. This could be a maximum number of generations, a satisfactory fitness score, or other criteria depending on your goals.

9. Repeat:

- If the termination criterion is not met, go back to step 2 and repeat the process for the next generation.

10. End:

- Once the termination criterion is met, end the genetic algorithm. The individual with the highest fitness score in the final population represents the optimized AI controller for playing the Mario game

Sample Code

```
import numpy as np
import matplotlib.pyplot as plt
import pickle
from matplotlib import style
import time
import pandas as pd
import random
import os
import sys
from pyboy import PyBoy, WindowEvent
import json
style.use("ggplot")
classes = 10
batch_size = 64
population = 5
generations = 5
threshold = 100000000
games = 800
time_h = 0
lucro0 = 0
class environment:
    def__init_(self):
        # start the game
        filename = 'roms/Super Mario Land.gb'
        quiet = "--quiet" in sys.argv
        self.pyboy = PyBoy(filename, window_type="headless" if quiet else "SDL2",
window_scale=2, debug=quiet, game_wrapper=True)
        self.pyboy.set_emulation_speed(1)
        assert self.pyboy.cartridge_title() == "SUPER MARIOLAN"
        self.mario = self.pyboy.game_wrapper()
        self.mario.start_game()
        self.done = False
        self.time = 10

        assert  self.mario.score == 0
        assert self.mario.lives_left == 2
        assert self.mario.time_left == 400
        assert self.mario.world == (1, 1)  # stage
        assert self.mario.fitness == 0  # A built-in fitness score for AI development

        # set state and action size
        self.action_size = 5 # number of possible actions
        state_full = np.asarray(self.mario.game_area())
```

```
        np.append(state_full, self.mario.level_progress)
        self.state_size = state_full.size

        self.start_world = 1
        self.start_level = 1

    def reset(self):
        self.mario.reset_game()  # back to the last state saved
        self.done = False
        self.pyboy.tick()
        assert self.mario.lives_left == 2
        self.position = self.mario.level_progress
        state_full = np.asarray(self.mario.game_area())
        np.append(state_full, self.mario.level_progress)

        return state_full

    def reset_to_saved_state(self):
        self.mario.reset_game()
        self.mario.set_world(self.start_world)
        self.mario.set_level(self.start_level)
        self.done = False
        self.pyboy.tick()
        assert self.mario.lives_left == 2
        self.position = self.mario.level_progress
        state_full = np.asarray(self.mario.game_area())
        np.append(state_full, self.mario.level_progress)
        return state_full
    def step(self, action):
        if action == 0:
            self.pyboy.send_input(WindowEvent.PRESS_BUTTON_A)
            self.time = 50
        elif action == 1:
            self.pyboy.send_input(WindowEvent.PRESS_ARROW_RIGHT)
            self.time = 5
        elif action == 2:
            self.pyboy.send_input(WindowEvent.RELEASE_BUTTON_A)
            self.time = 5
        elif action == 3:
            self.pyboy.send_input(WindowEvent.RELEASE_ARROW_RIGHT)
            self.time = 5
        elif action == 4:
            self.pyboy.send_input(WindowEvent.PRESS_ARROW_LEFT)
            self.time = 5
        return action, self.time
```

```
class Network():
    def __init_(self):
        self.actions = []
        self.generation = 0
        # generate random actions
        for i in range(games):
            self.action = random.randint(0, 5)
            self.actions.append(self.action)
        self.lucro = 0
    def get_actions(self):
        return self.actions

    def set_actions(self, actions, lucro):
        self.actions = actions
        self.lucro = lucro
        return self.lucro
def init_networks(population):
    return [Network() for _ in range(population)]

def fitness(networks):
    for network in networks:
        # init env
        env = environment()
        state_size = env.state_size
        action_size = env.action_size
        fitness = env.mario.fitness
        state = env.reset()
        state = np.reshape(state, [1, state_size])
        actions = network.get_actions()
        strategies = []
        lucro_tot = 0
        time_h = 0
        for act in actions:
            try:
                filteredMario = [x for x in list(state[0]) if (x > 10 and x < 30)]
                index_mario = list(state[0]).index(filteredMario[0])
                feet_val = state[0][index_mario + 20]
            except:
                break
            act, tempo = env.step(act)
            state = np.asarray(env.mario.game_area())
            position = env.mario.level_progress
            state = np.reshape(state, [1, state_size])
```

```
            i = 0
            while feet_val <= 350:
                env.pyboy.tick()
                i += 1
                if i > 60:
                    break

            if feet_val >= 350:
                tempo = 2
                for _ in range(tempo):
                    env.pyboy.tick()

            t = 0.0167 * tempo
            time.sleep(t)

            fitness = env.mario.fitness

            if env.mario.lives_left == 1:
                done = True
                break

            time_h      +=      1
        network.lucro = fitness
        print('Lucro Total: {}'.format(network.lucro))
        env.pyboy.stop()
    return networks
def selection(networks):
    networks = sorted(networks, key=lambda network: network.lucro, reverse=True)
    networks = networks[:int(0.2 * len(networks))]
    return networks
def crossover(networks):
    offspring = []
    for _ in range(int((population - len(networks)) / 2)):
        parent1 = random.choice(networks)
        parent2 = random.choice(networks)
        child1 = Network()
        child2 = Network()
        p1 = int(len(parent1.actions) / 2)
        p2 = int(len(parent2.actions) / 2)
        a = parent1.actions[:p1]
        b = parent2.actions[:p2]
        new_actions = a + b
```

```
        child1.actions = new_actions

        new_actions2 = b + a
        child2.actions = new_actions2

        offspring.append(child1)
        offspring.append(child2)

    networks.extend(offspring)
    return networks

def mutate(networks):
    for network in networks[2:]:
        for _ in range(0, int(games / 2)):
            val = np.random.uniform(0,  1)
            if val <= 0.2:  # mutation chance
                idx = np.random.randint(0, len(network.actions))
                network.actions[idx] = np.random.randint(0, 3)

    return networks

def main():
    lucro_nets = []
    best_lucro_nets = []
    best_networks = []
    lucro_nets_media = []
    networks = init_networks(population)
    try:
        with open('genetic_best_network_mario.json') as json_file:
            data = json.load(json_file)
        lucro = networks[0].set_actions(data.get('actions'), data.get('lucro'))
    except:
        print('Error to load')
        data = None
   for gen in range(generations):
        print('Generation {}'.format(gen + 1))
        networks = fitness(networks)
        network_lucro = []
        for network in networks:
            network.generation = gen
            network_lucro.append(network.lucro)
            lucro_nets.append(network.lucro)
            if network.lucro > threshold:
                print('Threshold met')
                print(network.get_actions())
               print('Best lucro: {}'.format(network.lucro))
```

```
            exit(0)

        networks = selection(networks)
        networks = crossover(networks)
        networks = mutate(networks)

        print('Best Fitness: {}'.format(max(network_lucro)))
        best_lucro_nets.append(max(network_lucro))
        media = sum(network_lucro) / len(network_lucro)
        print('Average Fitness: {}'.format(media))
        lucro_nets_media.append(media)
        print("")

    best_net = {'actions': networks[0].get_actions(), 'lucro': networks[0].lucro, 'generation':
networks[0].generation}
    with open('genetic_best_network_mario.json', 'w') as json_file:
        json.dump(best_net, json_file)
    json_file.close()

    plt.subplot(211)
    plt.plot([i for i in range(len(lucro_nets_media))], lucro_nets_media)
    plt.plot([i for i in range(len(best_lucro_nets))], best_lucro_nets)
    plt.ylabel(f"Average Fitness by generation")
    plt.xlabel("Generation #")
    plt.subplot(212)
    plt.plot([i for i in range(len(lucro_nets))], lucro_nets)
    plt.show()
if__name__== '_main_':
    main()
```

CHAPTER 4
RESULT AND DISCUSSION

CHAPTER 4

RESULTS AND DISCUSSION

To run this project install latest version of python and matplotlib , pandas, numpy and pyboy is the game boy emulator written in python . first implement
Genetic algorithm with the pyboy and then it will start learning the game environment by itself and start to move accordingly.

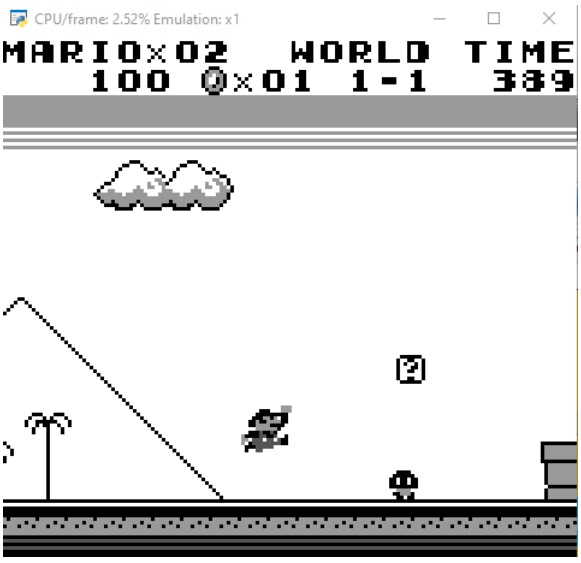


Fig -4.1 Mario Ai Generation -1

The above video shows the first level of the game and the character in the game is not dodging the obstacle due to the exploratory nature and the evolution nature of the genetic algorithm .

As we increase the generations the amount of accuracy will increase and the weights will be stored in the trained data set. This will help in improving the Performance of the next generation

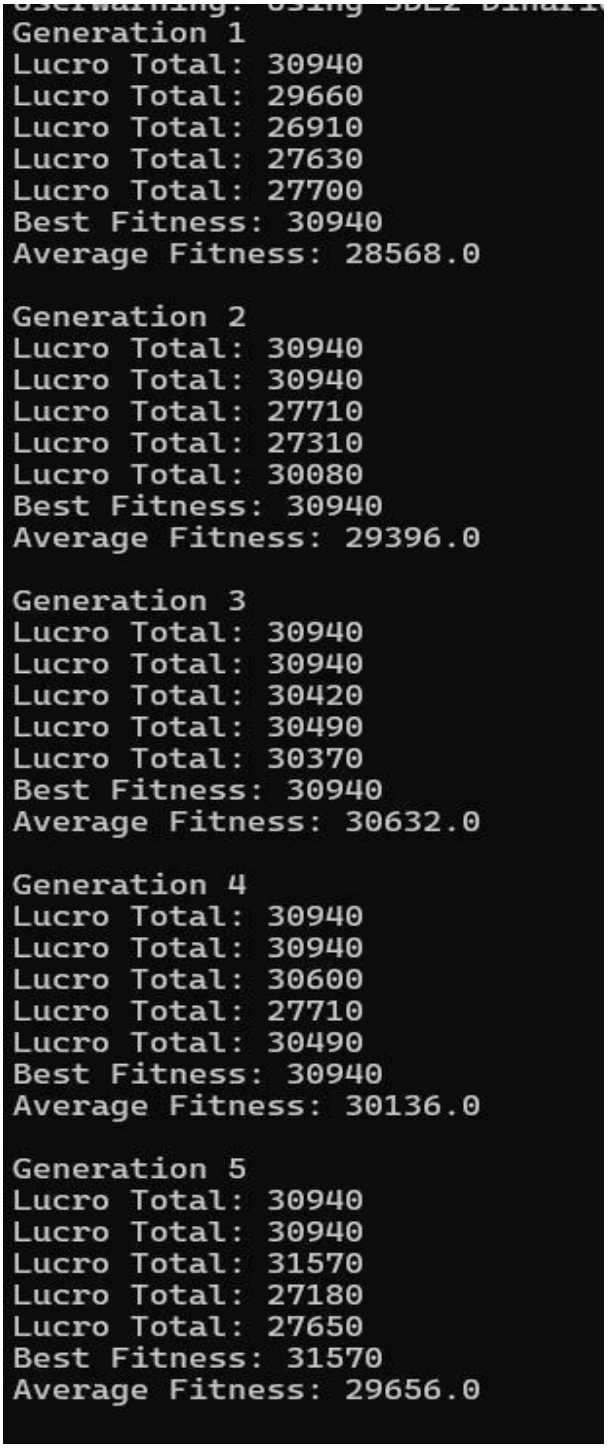


Fig-4.2 Fitness Values

The above png image shows the number of generation and the lucro which is the measure of the fitness /performance of each network in the genetic algorithm and the fitness of the generations

In the above image it show each generations have the lucro network performance and the best fitness and the average fitness of the one particular generation .

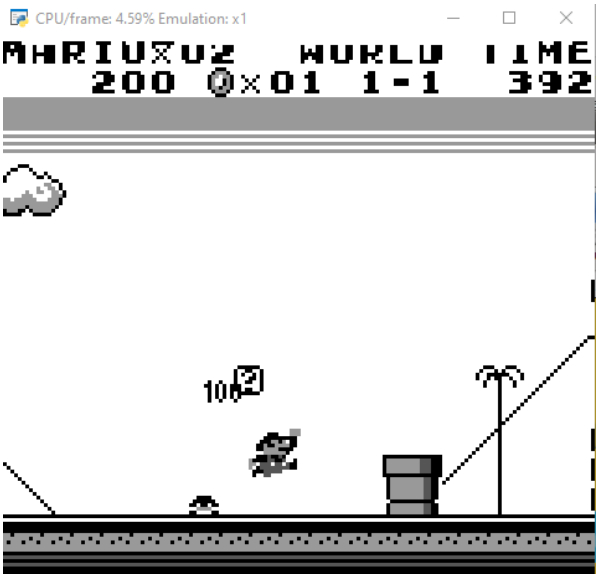


Fig-4.3 Mario Ai Generation -10

The above video shows that the character in the game is moving in the right direction and trying to didge the obstacle and as the environment movies on the character in the game need to learn and analyze the environment and then it movies on to the next generation

4.1 Performance Metrics

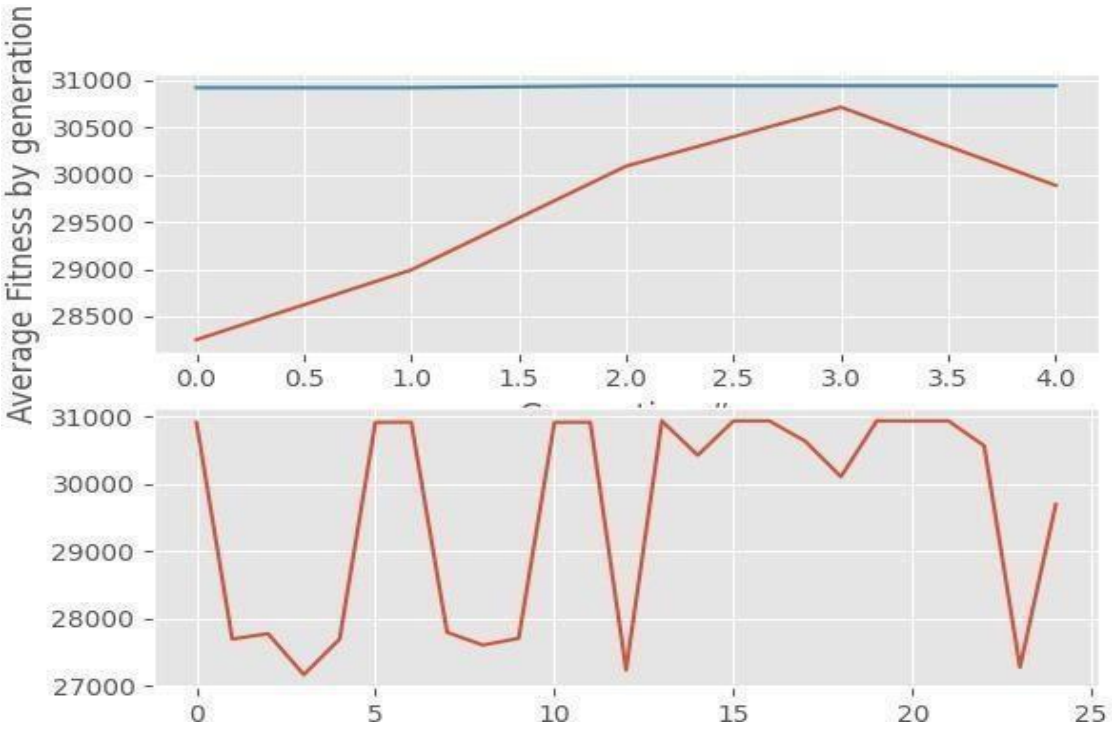


Fig-4.4 plots of Generation -5

In the above image the performance of 5 Generations is being show.

Initialization: The fitness function initializes the game environment and sets up the necessary parameters for evaluating the fitness of each network.

Simulation: For each network in the population, the code simulates the gameplay by executing the actions specified by the network within the game environment.

Fitness Calculation: During the simulation, the code tracks various aspects of the agent's performance, such as the agent's progress in the game, score, remaining lives, or any other relevant metrics. These metrics are used to calculate the fitness of each network.

Fitness Assignment: The calculated fitness values are assigned to each network based on its performance in playing the game.

Output: The fitness values are typically printed or logged to provide information about the performance of each network in the population.

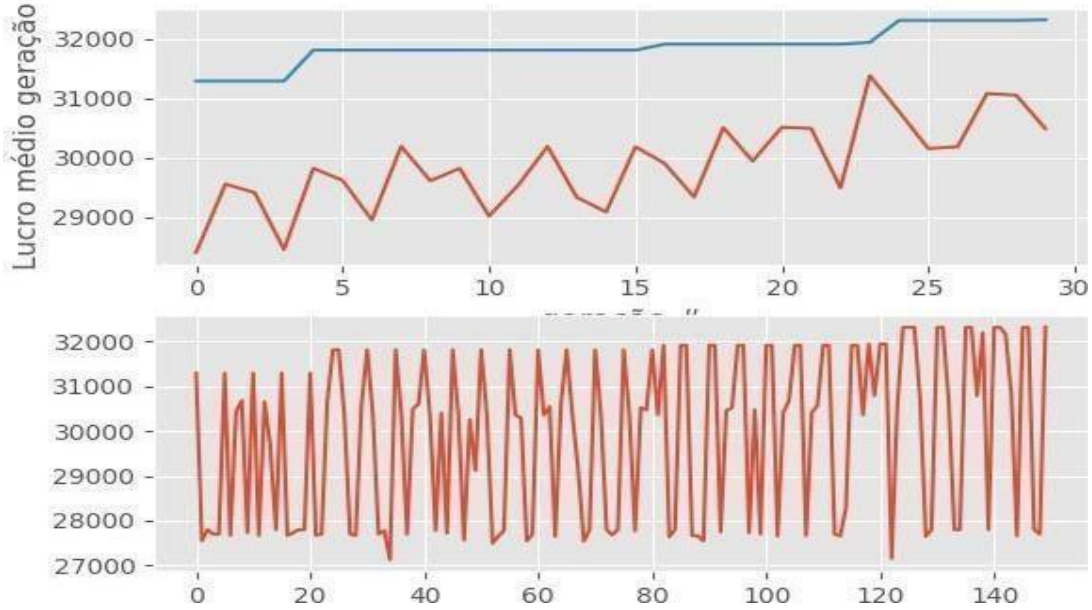


Fig-4.5 Plots of Generation-10

In the above image the performance of 10 generation is being shown

Initialization: The fitness function initializes the game environment and sets up the necessary parameters for evaluating the fitness of each network.

Simulation: For each network in the population, the code simulates the gameplay by executing the actions specified by the network within the game environment.

Fitness Calculation: During the simulation, the code tracks various aspects of the agent's performance, such as the agent's progress in the game, score, remaining lives, or any other relevant metrics. These metrics are used to calculate the fitness of each network.

Fitness Assignment: The calculated fitness values are assigned to each network based on its performance in playing the game.

Output: The fitness values are typically printed or logged to provide information about the performance of each network in the population.

After the performance of each generation will be stored in the json file in dictionary format

CHAPTER 5
CONCLUSION

CHAPTER 5
CONCLUSION

In conclusion, this project, we designed an automatic agent using genetic algorithm to play the Mario game. In conclusion, the incorporation of a genetic algorithm into the Mario AI system has yielded significant advancements. Through successive generations, the AI characters have demonstrated enhanced performance, showcasing the algorithm's capacity to evolve effective strategies. Notably, the adaptability of the algorithm enables the AI to navigate changing environments and level designs adeptly. While trade-offs were made concerning computational resources, the AI exhibits resilience in overcoming diverse challenges. Comparative analysis positions the genetic algorithm as a versatile and robust approach. Looking forward, opportunities for future research involve refining the algorithm, incorporating additional features, and exploring alternative strategies to further optimize the AI's behavior in gaming scenarios. This study marks a substantial step forward in the pursuit of sophisticated and adaptive AI systems for gaming applications. for example, in our genetic algorithm, we did not design the state for Mario to grab mushroom and flowers. In addition, our algorithm focuses on optimizing the successful rate. We believe that our work provides a good introduction to this problem and will benefit the people with interests in using genetic algorithm to play computer games.

CHAPTER 6
REFERENCES

REFERENCES

[1] Mario AI Championship 2010: Results (2010). https://sites.google.com/a/marioai.com/www/results. Accessed 24 May 2015

[2]. Bellemare, M.G., Naddaf, Y., Veness, J., Bowling, M.: The arcade learning environment: an evaluation platform for general agents. JAIR 47, 253–279 (2013)

[3]. Emgallar: Intelligent NPC for Mario AI Championship (2011). https://www.youtube.com/watch?v=u 0pgFQ8HcM. Accessed 22 May 2015

[4]. Fujii, N., Sato, Y., Wakama, H., Katayose, H.: Autonomously acquiring a video game agent's behavior: letting players feel like playing with a human player. In: Nijholt, A., Rom˜ao, T., Reidsma, D. (eds.) ACE 2012. LNCS, vol. 7624, pp. 490–493. Springer, Heidelberg (2012)

[5]. Fujii, N., Sato, Y., Wakama, H., Kazai, K., Katayose, H.: Evaluating human-like behaviors of video-game agents autonomously acquired with biological constraints. In: Reidsma, D., Katayose, H., Nijholt, A. (eds.) ACE 2013. LNCS, vol. 8253, pp. 61–76. Springer, Heidelberg (2013)

[6]. Karakovskiy, S., Togelius, J.: The Mario AI benchmark and competitions. IEEE Trans. Comput. Intell. AI Games 4(1), 55–67 (2012)

[7]. Meffert, K., Rotstan, N.: JGAP: Java Genetic Algorithms Package (2015). http://jgap.sourceforge.com. Accessed 6 July 2015

[8]. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A., Veness, J., Graves, A., Riedmiller, M., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature 518, 529–533 (2015)

[9]. Steadman, I.: This AI 'Solves' Super Mario Bros. and Other Classic NES Games(2013).http://www.wired.co.uk/news/archive/2013-04/12/super-mariosolved. Accessed 22 May 2015

[10]. Togelius, J., Karakovskiy, S., Baumgarten, R.: The 2009 Mario AI competition. In: 2010 IEEE Congress on Evolutionary Computation, pp. 1–8 (2010)

[11]. Togelius, J., Karakovskiy, S., Koutnik, J., Schmidhuber, J.: Super Mario evolution. In: 2009 IEEE Symposium on Computational Intelligence and Games, pp. 156–161

ss

iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 12 **Issue:** IV **Month of publication:** April 2024

DOI: https://doi.org/10.22214/ijraset.2024.59798

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 12 Issue IV Apr 2024- Available at www.ijraset.com

Game Agent using Genetic Algorithm

V. Akshit Rao[1], P. Sai Kiran[2], Prateet M[3], K. Sudhakar Reddy[4]
[1, 2, 3]UG Student, [4]Asst. Professor, Department of CSE (AI&ML), CMR College of Engineering & Technology, Hyderabad, Telangana

Abstract: ***The Mario AI Project is a pioneering project that uses genetic algorithms to teach an AI agent to play Super Mario. Inspired by natural selection, the algorithms work iteratively over generations to improve the AI agent's gameplay strategies. Every generation sees the agent's decision-making processes like jumping and navigating obstacles guided by a unique genetic code carefully crafted for optimum performance. Simulating natural Discrimination, Interchange and Transmutation, the strategy of AI Evolution agent undergoes a constant improvement.***

I. INTRODUCTION

In Mario AI, genetic Algorithmize AI entities that evolve over multiple generations. First, a population of potential fixes is created at random. Every solution depicts an AI agent with distinct characteristics. Fitter solutions are chosen for replication through evaluation according to fitness parameters like distance travelled and challenges conquered. Offspring that undergo crossover and mutation are more diverse and varied. Over several generations, this cyclical process of mutation, reproduction, and selection results in the creation of AI beings that are ever more skilled. In the end, genetic algorithms make it possible to create intelligent agents who can play the challenging platformer game Super Mario Bros. and navigate and complete levels on their own.

Artificial Intelligence (AI) has advanced significantly in the last several years, especially in the gaming industry. The creation of intelligent beings that can play video games on their own is an intriguing use of AI. Because of its rich intricacy and clear rules, the beloved platformer game "Super Mario Bros." Embarked a prominent testbed for AI research among these titles.

Genetic algorithms are a method for developing AI bots that can play Super Mario Bros. The process of natural selection refinement and evolution, in which solutions to problems develop over many generations, serves as an inspiration for genetic algorithms.

II. RELATED WORK

A. Rule-Based Systems

In Mario AI, rule-based systems use pre-established heuristics and rules to control how AI agents behave in the game world. Developers usually create these rules using their knowledge of the game mechanics and winning gameplay techniques. Rules could control things like dodging obstacles, hopping over opponents, and gathering power-ups. Rule-based systems have the advantage of being transparent; because the rules are clearly established, it is easy to comprehend how the AI agent behaves.

Moreover, such systems can be computationally efficient, as they do not require extensive training or learning processes. However, the Advancement of comprehensive rule sets for complex games like Mario Bros. can be challenging and time-consuming. Additionally, rule-based systems may struggle to adapt to novel or the Unanticipated scope of their predefined rules, limiting their flexibility and robustness in dynamic game environments.

B. Reinforcement Learning

Real-life agents engage with the gaming environment by acting and getting feedback in the form of incentives or punishments according to how well they succeed. Through frequent encounters, RL agents eventually pick up efficient techniques for completing objectives like clearing a level, gathering coins, and vanquishing foes. The Pivotal of reinforcement learning is its adaptability; RL agents can effectively navigate complicated and dynamic game contexts because they can adjust to new circumstances and learn from past mistakes. To converge to optimal behavior, RL agent training can be computationally demanding and necessitate Abundance interactions with the environment. Additionally, RL agents have to balance the exploration-exploitation trade-off, which can be difficult in unpredictable or stochastic environments like Super Mario Bros., between exploring novel actions and utilizing tried-and-true high-reward tactics.

III. METHODOLOGY

The first step in using the genetic algorithm with the PyBoy emulator for Mario AI is to initialize a population of possible solutions, each of which represents an AI agent Endowed with particular parameters that determine its behaviour. Via Py Boy, these agents communicate with the gaming environment and offer a simulated gaming platform. After the emulator has been initialized, Aptly they performed by looking at data like distance traveled, coins collected, and obstacles overcome. Then, selection techniques, which imitate natural selection principles, decide which agents advance to the next generation based on their fitness scores. By transferring genetic information across chosen agents to produce children with a combination of features, crossover promotes genetic diversity. Furthermore, mutation procedures add more variation to the population by introducing random modifications to its genetic makeup. As new offspring are introduced, the replacement mechanism makes sure that the population size stays constant. The performance of AI agents gradually increases through repeating cycles of crossover, mutation, and selection, convergent towards ideal solutions for traversing the complex and dynamic obstacles of Super Mario Bros. Through the Fusion PyBoy, researchers can effectively investigate an extensive array of AI agent setups, customizing solutions to thrive in the intricate and constantly changing Super Mario Bros. gaming environment.

IV. RESULT AND DISCUSSION

Fig.1. This is the Py Boy Emulator. Running the super mario game environment .

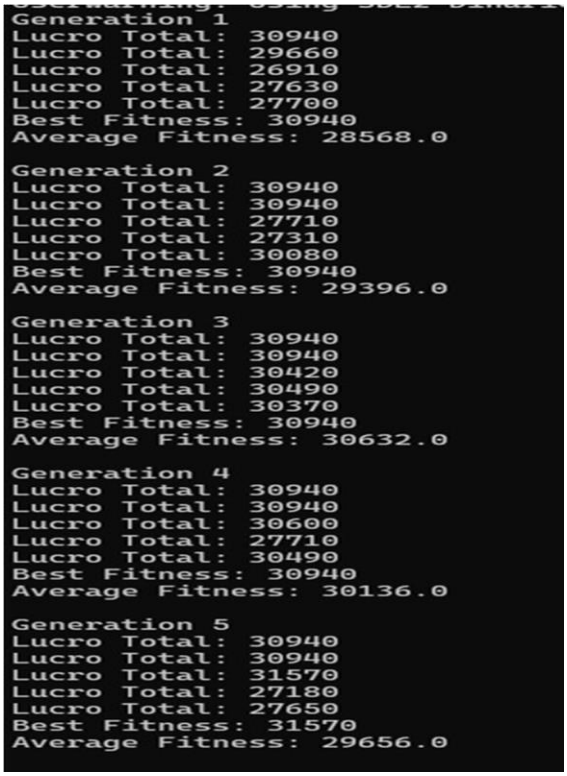


Fig.2. This is the output Generated for the successive 5 Generations .

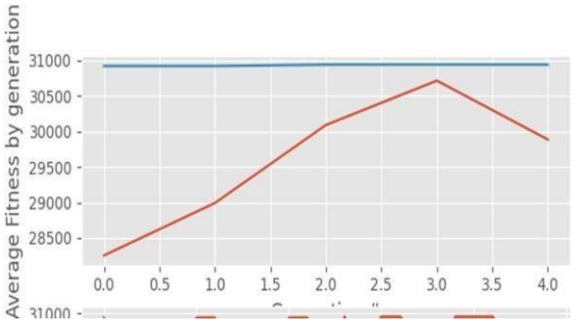


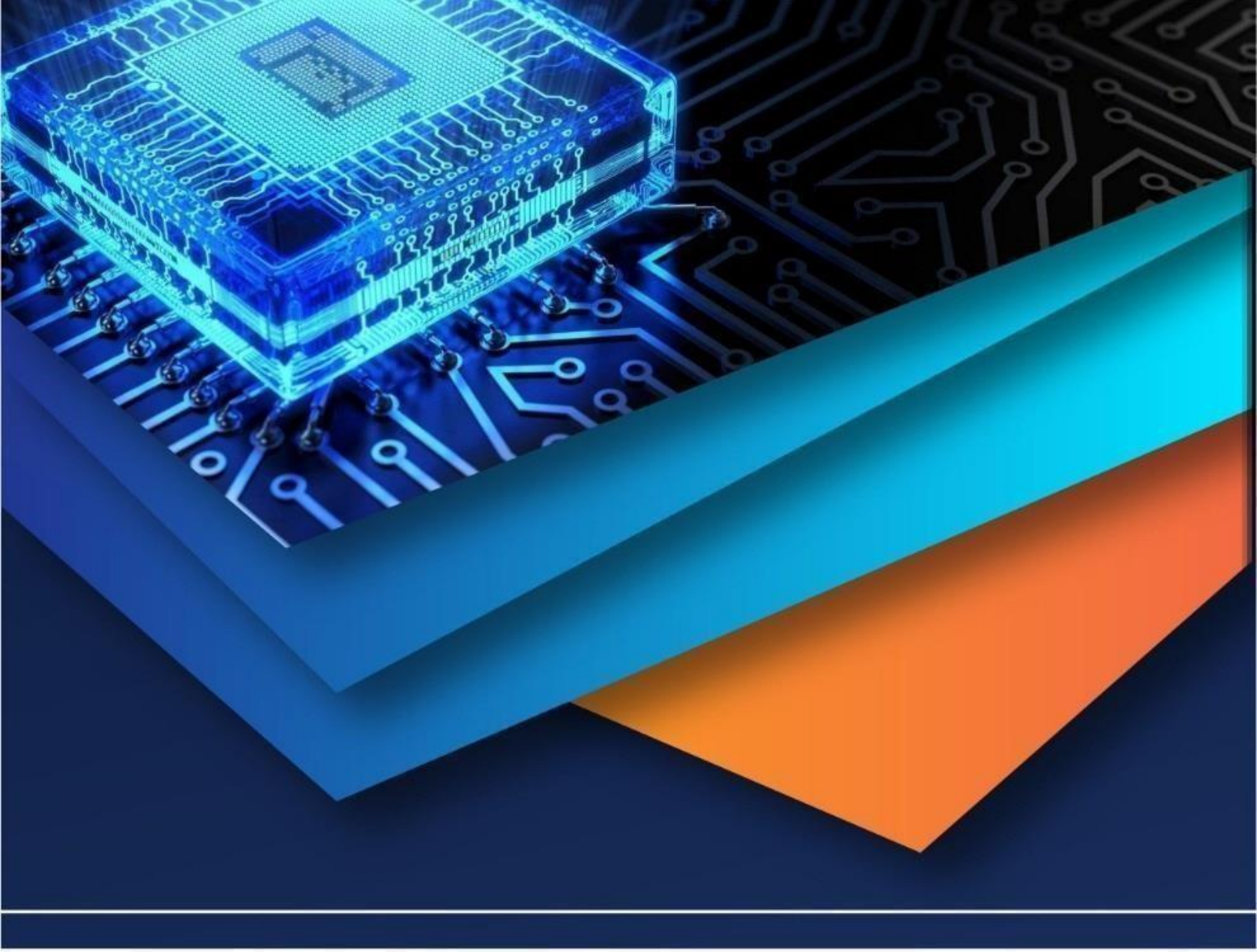
Fig.3. This is the Plotting of the 5 Generations with the average fitness values.

V. CONCLUSION

To sum up, for this project, we created An independent Mario using a genetic algorithm. In conclusion, the Mario system has advanced significantly Consequence genetic algorithm's integration. The AI characters have performed better throughout the course of multiple generations, demonstrating the algorithm's ability to develop techniques that work. Notably, the AI can handle shifting settings and level designs with ease thanks to the algorithm's versatility. Trade-offs were made Concerning to processing resources, yet the AI is resilient Amidst variety of obstacles. The GE is positioned as a flexible and reliable method by comparative analysis. In order to further enhance the AI's behaviour in gaming scenarios, there are potential for future study to refine the algorithm, add new features, and investigate alternate techniques. This work represents a significant advancement in Proliferation intelligent and flexible AI systems for use in gaming. For instance, we did not create the condition in our genetic process where Mario might pick flowers and mushrooms. Furthermore, our system prioritizes optimizing the success rate. Proposition useful overview of this issue and will be helpful to anyone who are interested in playing computer games with genetic algorithms.

REFERENCES

[1] Mario AI Championship 2010: Results (2010).https://sites.google.com/a/marioai.com/www/results. Accessed 24 May 2015
[2] Bellemare, M.G., Naddaf, Y., Veness, J., Bowling, M.: The arcade learning environment: an evaluation platform for general agents. JAIR 47, 253–279 (2013)
[3] Emgallar: Intelligent NPC for Mario AI Championship(2011).https://www.youtube.com/watch?v=u 0pgFQ8HcM. Accessed 22 May 2015
[4] Fujii, N., Sato, Y., Wakama, H., Katayose, H.: Autonomously acquiring a video game agent's behavior: letting players feel like playing with a human player. In:Nijholt, A., Rom˜ao, T., Reidsma, D. (eds.) ACE 2012. LNCS, vol. 7624, pp. 490–493. Springer, Heidelberg (2012)
[5] Karakovskiy, S., Togelius, J.: The Mario AI benchmark and competitions. IEEETrans. Comput. Intell. AI Games 4(1), 55–67 (2012)
[6] Meffert, K., Rotstan, N.: JGAP: Java Genetic Algorithms Package (2015). http://jgap.sourceforge.com. Accessed 6 July 2015
[7] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A., Veness, J., Graves, A., Riedmiller,M., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I.,King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level controlthrough deep reinforcement learning. Nature 518, 529–533 (2015)
[8] Steadman, I.: This AI 'Solves' Super Mario Bros.andOtherClassicNESGames(2013).http://www.wired.co.uk/news/archive/2013-04/12/super-mariosolved. Accessed 22 May 2015
[9] Togelius,J.,Karakovskiy, S., Baumgarten, R.: The 2009 Mario AI competition. In:2010 IEEE Congress on Evolutionary Computation, pp. 1–8 (2010)
[10] Markus Persson. Infinite Mario Bros. http://www.mojang.com/notch/mario/. 2011.
[11] Jarret Raim. Finite State Machine In Game,from:www.cse.lehigh.edu/~munoz/CSE497/classes/FSM_In_Games.ppt. 2011.
[12] K.-T. Chang, K.-O. Chin, and J. Teo, "The Evolution of Gamebots for 3D First Person Shooter (FPS)," in press. The Sixth International Conference on Bio-Inspired Computing: Theories and Applications, 2011.
[13] A.E. Eiben, and J.E. Smith, Introduction to Evolutionary Computing. Springer, 2003



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

ISSN No. : 2321-9653

IJRASET

International Journal for Research in Applied Science & Engineering Technology

IJRASET is indexed with Crossref for DOI-DOI : 10.22214

Website : www.ijraset.com, E-mail : ijraset@gmail.com

ISRA Journal Impact Factor: 7.429

45.98 INDEX COPERNICUS

THOMSON REUTERS Researcher ID: N-9681-2016

10.22214/IJRASET doi cross ref

TOGETHER WE REACH THE GOAL SJIF 7.429

Certificate

It is here by certified that the paper ID : IJRASET59798, entitled

Game Agent Using Genetic Algorithm

by

V Akshit Rao

after review is found suitable and has been published in Volume 12, Issue IV, April 2024

in

International Journal for Research in Applied Science & Engineering Technology

(International Peer Reviewed and Refereed Journal)

Good luck for your future endeavors

Editor in Chief, iJRASET

ISSN No. : 2321-9653

IJRASET

International Journal for Research in Applied Science & Engineering Technology

IJRASET is indexed with Crossref for DOI-DOI : 10.22214

Website : www.ijraset.com, E-mail : ijraset@gmail.com

ISRA Journal Impact Factor: 7.429

45.98 INDEX COPERNICUS

THOMSON REUTERS Researcher ID: N-9681-2016

10.22214/IJRASET

TOGETHER WE REACH THE GOAL SJIF 7.429

Certificate

It is here by certified that the paper ID : IJRASET59798, entitled

Game Agent Using Genetic Algorithm

by

P Sai Kiran

after review is found suitable and has been published in Volume 12, Issue IV, April 2024

in

International Journal for Research in Applied Science & Engineering Technology

(International Peer Reviewed and Refereed Journal)

Good luck for your future endeavors

Editor in Chief, iJRASET

ISSN No. : 2321-9653

IJRASET

International Journal for Research in Applied Science & Engineering Technology

IJRASET is indexed with Crossref for DOI-DOI : 10.22214

Website : www.ijraset.com, E-mail : ijraset@gmail.com

ISRA Journal Impact Factor: 7.429

45.98 INDEX COPERNICUS

THOMSON REUTERS Researcher ID: N-9681-2016

10.22214/IJRASET doi cross ref

TOGETHER WE REACH THE GOAL SJIF 7.429

Certificate

It is here by certified that the paper ID : IJRASET59798, entitled

Game Agent Using Genetic Algorithm

by

Prateet M

after review is found suitable and has been published in Volume 12, Issue IV, April 2024

in

International Journal for Research in Applied Science & Engineering Technology

(International Peer Reviewed and Refereed Journal)

Good luck for your future endeavors

Editor in Chief, iJRASET

ISSN No. : 2321-9653

IJRASET

International Journal for Research in Applied Science & Engineering Technology

IJRASET is indexed with Crossref for DOI-DOI : 10.22214

Website : www.ijraset.com, E-mail : ijraset@gmail.com

ISRA Journal Impact Factor: 7.429

45.98 INDEX COPERNICUS

THOMSON REUTERS Researcher ID: N-9681-2016

10.22214/IJRASET doi cross ref

TOGETHER WE REACH THE GOAL SJIF 7.429

Certificate

It is here by certified that the paper ID : IJRASET59798, entitled

Game Agent Using Genetic Algorithm

by

K Sudhakar Reddy

after review is found suitable and has been published in Volume 12, Issue IV, April 2024

in

International Journal for Research in Applied Science & Engineering Technology

(International Peer Reviewed and Refereed Journal)

Good luck for your future endeavors

Editor in Chief, iJRASET