Khulna University of Engineering & Technology
Department of Computer Science and Engineering

CSE 4110: Artificial Intelligence Laboratory
Project Title: **Flappy Bird Game**
Date of Submission: 03/09/24

Instructed by:
Md. Shahidul Salim
Lecturer,
Department of CSE, KUET

Most. Kaniz Fatema Isha
Lecturer,
Department of CSE, KUET

Submitted by:
Nushrat Tarmin Meem
Roll: **1907083**
Anisa Walida
Roll: **1907087**
Group: B_2
Year: 4th
Semester: 1st

Flappy Bird Game

Objectives:

- Creating an user-friendly interface
- To create levels for overall players
- Implementation of the A* algorithm to dynamically adjust obstacle placements
- Utilizing alpha-beta pruning to optimize the decision-making process for the calculation of optimal moves
- Integrating fuzzy logic to fine-tune game parameters such as jump height and controlling speed levels
- Employing genetic algorithms to evolve obstacle patterns over time also for the boost up or higher score strategies

Introduction:

In the game Flappy Bird, sophisticated algorithms are combined with a beloved classic. To improve playability and challenge, we have included state-of-the-art AI techniques such as A*, alpha-beta pruning, fuzzy logic, and genetic algorithms into this special edition of Flappy Bird. traversed a number of stages with differing degrees of difficulty, each intended to highlight these algorithms' capabilities. Get ready to explore a new Flappy Bird dimension where strategy and intelligence combine to completely transform the game experience, whether you're an experienced player or an AI aficionado. Every algorithm has a distinct function that improves several parts of the game, such as AI opponent behavior and level design, offering players of all skill levels a difficult and varied experience.

Theory:

A* Search Algorithm:

A* is a best-first search algorithm, classified as an informed search, designed for weighted graphs. Starting from a designated initial node, the algorithm seeks to discover the path to a target node with the lowest possible cost, whether that cost is measured by distance, time, or another factor. A* accomplishes this by constructing and expanding a tree of potential paths, beginning from the start node and incrementally extending these paths one edge at a time until it successfully reaches the goal node.

Alpha Beta Pruning Algorithm:

Alpha–beta pruning is an optimization technique used with the minimax algorithm to reduce the number of nodes that need to be evaluated in a search tree. Commonly employed in adversarial search scenarios, such as two-player combinatorial games like Tic-tac-toe, Chess, or Connect 4, this algorithm improves efficiency by stopping the evaluation of a move as soon as it determines that the move is inferior to a previously considered option.
Since these inferior moves cannot influence the final decision, they are not explored further. When applied to a minimax tree, alpha–beta pruning yields the

same result as the standard minimax algorithm but does so more efficiently by eliminating unnecessary branches.

Fuzzy Logic:

The term "fuzzy" refers to concepts that are unclear or ambiguous. In real-life scenarios, we often encounter situations where it's difficult to determine whether something is strictly true or false. Fuzzy logic offers a valuable approach to reasoning in such cases by accommodating the imprecision and uncertainty inherent in many situations. Unlike traditional logic, which is binary (true or false), fuzzy logic allows for a range of truth values between 0 and 1. This flexibility makes it a powerful mathematical tool for handling vague or uncertain information, particularly in decision-making processes.

Genetic Algorithm:

Genetic Algorithms (GAs) are adaptive heuristic search techniques that fall under the broader category of evolutionary algorithms. These algorithms draw inspiration from the principles of natural selection and genetics. By utilizing historical data to guide random searches towards regions of better performance in the solution space, GAs intelligently exploit randomness. They are particularly effective in generating high-quality solutions for optimization and search problems.

The core concept of genetic algorithms is to mimic the process of natural selection, where species that adapt well to their environment are more likely to survive, reproduce, and pass on their traits to the next generation. In essence, GAs simulate the idea of "survival of the fittest" among individuals across successive generations to solve a problem. Each generation comprises a population of individuals, where each individual represents a potential solution and a specific point in the search space. These individuals are typically encoded as strings of characters, integers, floats, or bits, which are analogous to chromosomes in biological systems.

Manual Flowchart:

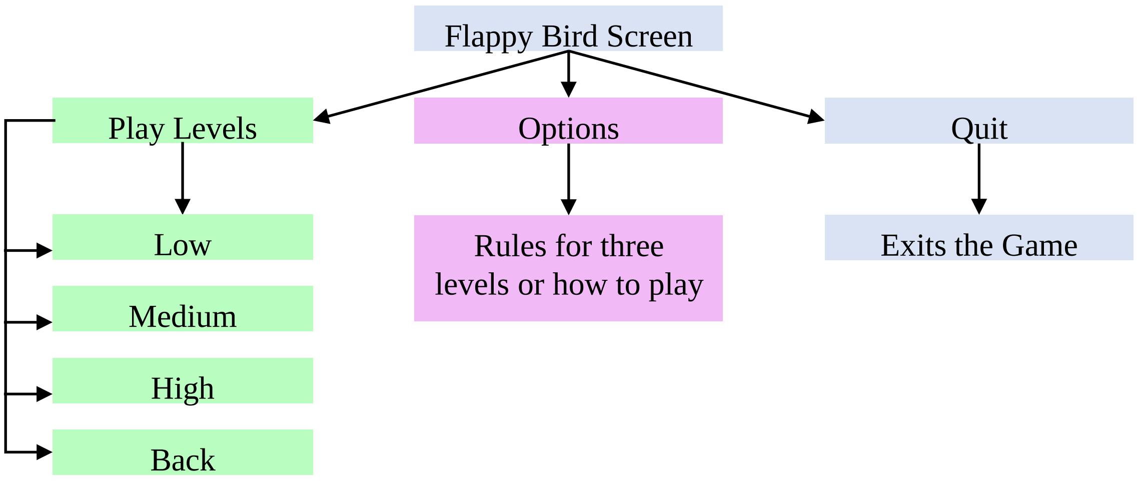


Figure 1: Flowchart of the basic flappy bird game UI

Methodology:

Level 1 Pseudocode:

The A* algorithm in this level has been used for pathfinding to navigate the bird towards the nearest coin while avoiding obstacles. Here is a pseudocode representation of the A* algorithm:

```
function A* (start, goal, obstacles):
    open_list = priority queue containing (0, start)
    came_from = empty map
    g_score = map with default value of infinity
    g_score[start] = 0
    f_score = map with default value of infinity
    f_score[start] = heuristic(start, goal)
    while open_list is not empty:
        current = node in open_list with lowest f_score
        if current == goal:
            return reconstruct_path(came_from, current)
        remove current from open_list
        for each neighbor of current:
            if neighbor is out of bounds or an obstacle:
                continue
            tentative_g_score = g_score[current] + distance(current, neighbor)
            if tentative_g_score < g_score[neighbor]:
                came_from[neighbor] = current
                g_score[neighbor] = tentative_g_score
                f_score[neighbor] = g_score[neighbor] + heuristic(neighbor, goal)
                if neighbor not in open_list:
                    add neighbor to open_list
    return failure
```

Here, Heuristic guides the search by estimating the cost to reach the goal.g_score tracks the actual shortest path from start to any node.f_score combines g_score and heuristic to determine which node to expand next.The open_list always expands the most promising node (lowest f_score). This algorithm has ensured that the shortest path to the goal is found while efficiently pruning unnecessary paths. In each iteration of the game loop:
The bird's current position is the start node.
The position of the nearest coin is the goal node.
Obstacles (like screen boundaries) are avoided during the path calculation.
The algorithm calculates the shortest path to this nearest coin, and the bird moves along this path until it reaches the coin (the goal), at which point a new goal is set (the next closest coin).

Level 2 Pseudocode:

The Alpha-Beta Pruning algorithm is used to find the optimal path for the bird to collect the nearest coin. The function alpha_beta_pruning() follows this pseudocode structure and aims to find the optimal coin for the bird to collect based on the game's current state. The implementation ensures efficient decision-making by pruning unnecessary paths, improving the game's performance.

```
function AlphaBetaPruning(coins, bird_position, depth, alpha, beta, maximizingPlayer):
    if depth == 0 or coins is empty:
        return bird_position, 0  # Base case: no coins left or maximum depth reached
    if maximizingPlayer:  # Bird's turn to move
        maxEval = -∞
        best_move = None
        for each coin in coins:
            # Generating new positions excluding the current coin
            new_positions = [pos for pos in coins if pos != coin]
            # Recursively calling with the next depth, minimizingPlayer's turn
            eval, _ = AlphaBetaPruning(new_positions, coin, depth - 1, alpha, beta, False)
            if eval[1] > maxEval:  # Updating maximum evaluation and best move
                maxEval = eval[1]
                best_move = coin
            alpha = max(alpha, eval[1])  # Updating alpha value
            if beta <= alpha:  # Beta cut-off
                break
        return best_move, maxEval  # Returning the best move and its evaluation score
    else:
       # Opponent's turn to minimize (obstacles or potential threats)
        minEval = ∞
        best_move = None
        for each coin in coins:
            # Generating new positions excluding the current coin
            new_positions = [pos for pos in coins if pos != coin]
            # Recursively calling with the next depth, maximizingPlayer's turn
            eval, _ = AlphaBetaPruning(new_positions, coin, depth - 1, alpha, beta, True)
            if eval[1] < minEval:  # Updating minimum evaluation and best move
                minEval = eval[1]
                best_move = coin
            beta = min(beta, eval[1])  # Updating beta value
            if beta <= alpha:  # Alpha cut-off
                break
        return best_move, minEval  # Returning the best move and its evaluation score
```

Inputs:

- ❑ coins: A list of available coin positions on the screen.
- ❑ bird_position: The current position of the bird.
- ❑ depth: The current depth level of the recursive search.
- ❑ alpha and beta: The current alpha and beta values for pruning.
- ❑ maximizingPlayer: A boolean indicating if it's the bird's turn (True) or the opponent's turn (False).

Base Case:

If depth == 0 or there are no more coins left (coins is empty), the function returns the current bird_position and a heuristic value of 0.

Maximizing Player (Bird's Turn):

This section tries to maximize the score by finding the closest coin to collect.
The bird checks each coin and recursively evaluates the possible positions using Alpha-Beta Pruning. The maxEval variable stores the best score found so far, and best_move tracks the corresponding coin position. If a branch leads to a beta cut-off (i.e., beta <= alpha), it stops further exploration to save computation time.

Minimizing Player (Opponent's Turn):

The minimizing player represents a simulated opponent or potential threats (e.g., moving pipes or hazards). The goal is to minimize the score by finding the worst possible move for the bird (e.g., avoiding collisions with pipes). The minEval variable stores the lowest score, and best_move tracks the move corresponding to that score. Alpha cut-offs (beta <= alpha) help prune unnecessary paths.

Return Values:

Each function call returns the best_move (next position to move to) and the evaluation score (maxEval or minEval).
Fuzzy Logic has been used to control the speed of the bird.

Level 3 Pseudocode :

Components of the Genetic Algorithm in the Game:

1.Population:

The Population class represents a group of individual AI agents, called Player objects, each with its own neural network (Brain).
The population size is fixed, and the goal is to evolve the players over generations to perform better in the game.

2.Chromosomes (Genotype):

In the context of this game, the chromosome is represented by the neural network (Brain) of each player. The Brain consists of nodes (neurons) and connections (synapses) between them, which determine how the bird makes decisions based on its environment.

3.Fitness Function:

The fitness of each Player is determined by how long they survive in the game. The longer a player survives without hitting a pipe or the ground, the higher its fitness score.

The calculate_fitness method in the Player class is responsible for computing the fitness score based on the player's lifespan.

4.Selection:

After evaluating the fitness of all players, the selection process identifies the fittest individuals to pass their genes (neural network configurations) to the next generation.

The Population class handles selection through species categorization (speciate method) and survival of the fittest within each species. Only the best-performing individuals (champions) and their offspring survive to the next generation.

5.Crossover and Mutation:

Crossover, or genetic recombination, is not explicitly detailed in the code but typically involves combining parts of two or more parent networks to produce offspring.

Mutation introduces random changes in the offspring's neural network to maintain diversity in the population. In this case, mutation occurs in the weights of the connections between nodes in the Brain class through the mutate method in the Connection class. This method slightly alters the weights, with a small chance of completely randomizing them.

6.Speciation:

Speciation is a mechanism to group similar individuals into species, ensuring diversity by protecting weaker individuals that belong to unique species. This helps prevent premature convergence to suboptimal solutions.

The Species class is responsible for grouping players based on the similarity of their neural networks and maintaining species-specific fitness benchmarks. The similarity is measured by the difference in connection weights between neural networks.

7.Next Generation:

After selection, crossover, and mutation, the next generation of players is created. The next_gen method in the Population class generates new players (children) from the best-performing individuals of the previous generation.

A champion from each species is cloned directly into the next generation, ensuring that the best solutions are preserved.

8.Extinction:

The extinction process removes species that are no longer viable, either because all their members have died (kill_extinct_species) or because the species has stagnated in terms of fitness improvements (kill_stale_species).

Pseudocode of level 3:

```
Initialize the game environment
Initialize a population of players (Population object)

WHILE game is running:
    IF all players in the population are dead:
        Perform natural selection to create the next generation:
            Speciate the population
            Calculate fitness for each player
            Kill extinct species (species with no members)
            Kill stale species (species with no improvement in fitness)
            Sort species by fitness
            Generate the next generation of players
        Reset the game environment for the new generation
    ELSE:
        FOR each player in the population:
            IF player is alive:
                Player looks at the environment and gathers input data (look)
                Player makes a decision based on the input (think)
                Update the player's position and state (update)
                Draw the player and environment to the screen (draw)
        Move pipes and check for collisions
        Spawn new pipes at regular intervals
        Increment the game clock and update the display
    ENDIF
ENDWHILE

FUNCTION Population.natural_selection:
    Speciate population by grouping similar players into species
    Calculate fitness for each player based on how long they survived
    Remove extinct species (species with no players)
    Remove stale species (species with no fitness improvement for several generations)
    Sort species based on the fitness of their best player
    Generate the next generation of players:
        Clone champions (best players) from each species
        Create offspring by crossover and mutation from the top players
        Replace old population with new players
    Increment the generation counter
FUNCTION Player.look:
    Identify the closest pipe
    Calculate distances to the top, middle, and bottom of the closest pipe
    Store these distances as input data (vision)
```

```
FUNCTION Player.think:
    Pass the input data (vision) to the player's brain (neural network)
    The brain processes the input and outputs a decision value
    IF decision value exceeds a threshold:
        Flap wings (bird_flap)
    ENDIF

FUNCTION Player.update:
    Apply gravity to the player's vertical position
    Check for collisions with pipes or ground
    IF collision occurs:
        Mark player as dead
    ELSE:
        Increment lifespan
    ENDIF

FUNCTION Brain.feed_forward(input):
    FOR each input node:
        Set output value to the input value
    Activate bias node with output value 1
    FOR each node in the network, in order of layers:
        Activate node by applying the sigmoid function
        Pass the output value to connected nodes
    RETURN the output value of the final output node

FUNCTION Brain.mutate:
    FOR each connection in the neural network:
        IF random chance of mutation occurs:
            Change the connection weight slightly OR randomize it
FUNCTION Species.similarity(brain):
    Compare the connection weights of the given brain with the benchmark brain
    RETURN TRUE if the weight difference is below a certain threshold
    RETURN FALSE otherwise

FUNCTION Species.offspring:
    Clone a player from the species
    Mutate the player's brain
    RETURN the new player

FUNCTION Species.sort_players_by_fitness:
    Sort players in the species based on their fitness in descending order
    Update the benchmark fitness and champion player if a better player is found
```

Fuzzy Logic :

```
Initialize the game environment
Initialize a population of players (Population object)

WHILE game is running:
   IF all players in the population are dead:
      Perform natural selection to create the next generation:
         Speciate the population
         Calculate fitness for each player
         Kill extinct species (species with no members)
         Kill stale species (species with no improvement in fitness)
         Sort species by fitness
         Generate the next generation of players
      Reset the game environment for the new generation
   ELSE:
      FOR each player in the population:
         IF player is alive:
            Player looks at the environment and gathers input data (look)
            Player makes a decision based on the input (think)
            Update the player's position and state (update)
            Draw the player and environment to the screen (draw)
      Move pipes and check for collisions
      Spawn new pipes at regular intervals
      Increment the game clock and update the display
   ENDIF
ENDWHILE

FUNCTION Population.natural_selection:
   Speciate population by grouping similar players into species
   Calculate fitness for each player based on how long they survived
   Remove extinct species (species with no players)
   Remove stale species (species with no fitness improvement for several generations)
   Sort species based on the fitness of their best player
   Generate the next generation of players:
      Clone champions (best players) from each species
      Create offspring by crossover and mutation from the top players
      Replace old population with new players
   Increment the generation counter
FUNCTION Player.look:
   Identify the closest pipe
   Calculate distances to the top, middle, and bottom of the closest pipe
   Store these distances as input data (vision)
```

```
FUNCTION Player.think:
    Pass the input data (vision) to the player's brain (neural network)
    The brain processes the input and outputs a decision value
    IF decision value exceeds a threshold:
        Flap wings (bird_flap)
    ENDIF

FUNCTION Player.update:
    Apply gravity to the player's vertical position
    Check for collisions with pipes or ground
    IF collision occurs:
        Mark player as dead
    ELSE:
        Increment lifespan
    ENDIF

FUNCTION Brain.feed_forward(input):
    FOR each input node:
        Set output value to the input value
    Activate bias node with output value 1
    FOR each node in the network, in order of layers:
        Activate node by applying the sigmoid function
        Pass the output value to connected nodes
    RETURN the output value of the final output node

FUNCTION Brain.mutate:
    FOR each connection in the neural network:
        IF random chance of mutation occurs:
            Change the connection weight slightly OR randomize it

FUNCTION Species.similarity(brain):
    Compare the connection weights of the given brain with the benchmark brain
    RETURN TRUE if the weight difference is below a certain threshold
    RETURN FALSE otherwise

FUNCTION Species.offspring:
    Clone a player from the species
    Mutate the player's brain
    RETURN the new player

FUNCTION Species.sort_players_by_fitness:
    Sort players in the species based on their fitness in descending order
    Update the benchmark fitness and champion player if a better player is found
```
Explanation

Result Analysis:

Main Menu:

First of all, created an Ui with three buttons play, options and quit. Here, options section defines rules and quit section exits the whole gaming environment.

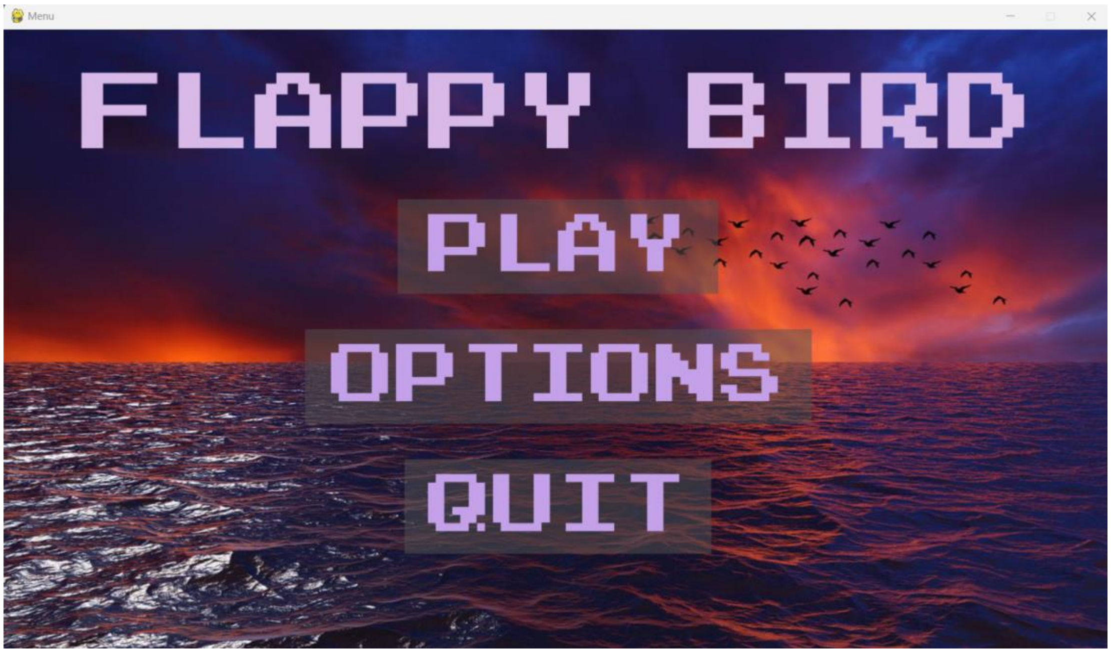


Figure 2: Main Menu Screen

Next comes the play section including three levels: low, medium and high. Back options takes the user back to the main menu.

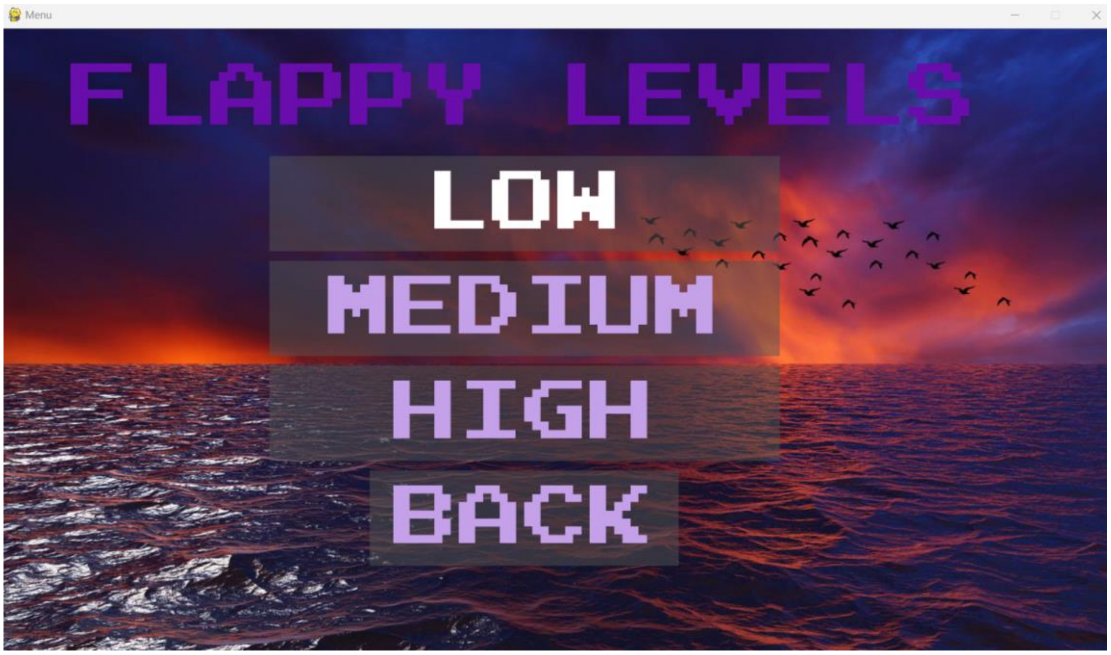


Figure 3: Levels Section

Level 1 (A* Algorithm Applied):

In the level 1 section, there are two rules to play it:

- If bird touches the ground, level 1 fails.
- The bird has to collect all coins
- Here, A* algorithm provides the bird minimum distance or the direction through the arrow to the nearest coin so that it can be collected as soon as possible with the proper direction

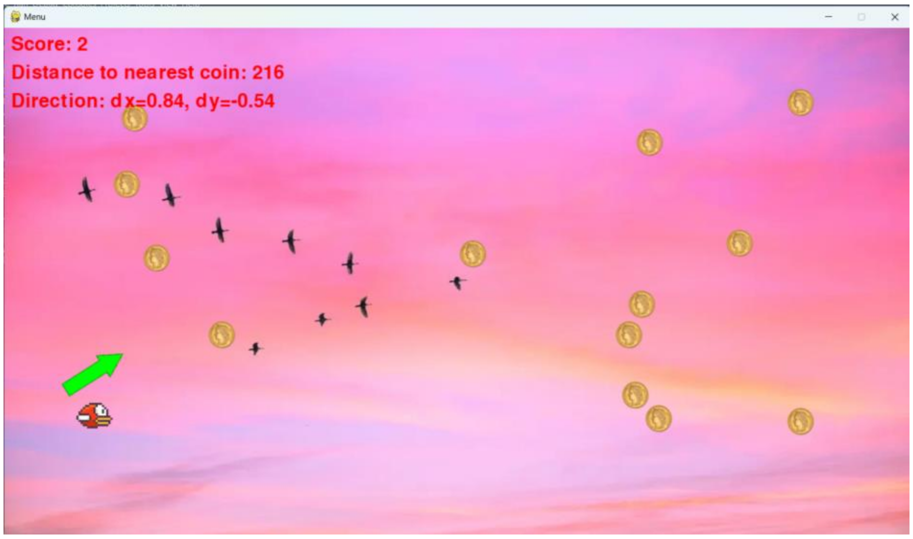


Figure 4: Starting of level 1

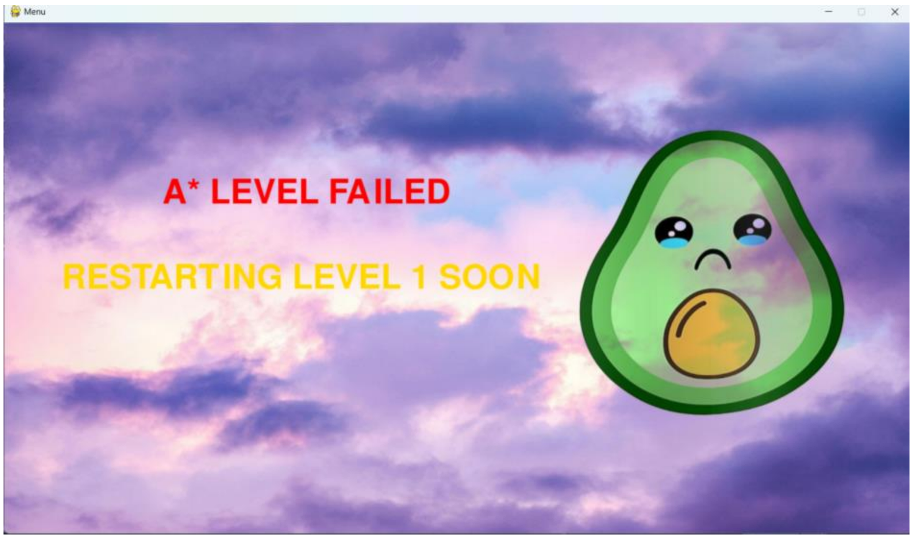
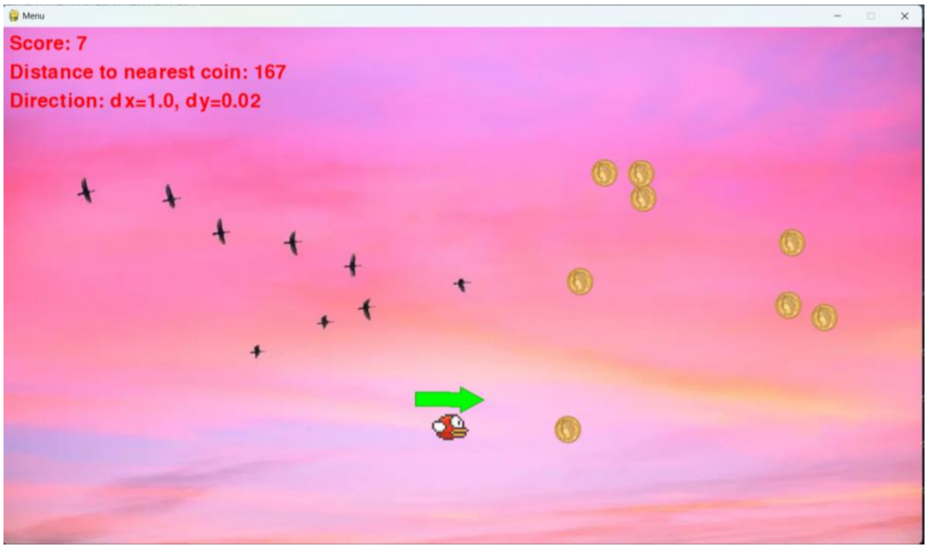


Figure 5: Ground touched level failed

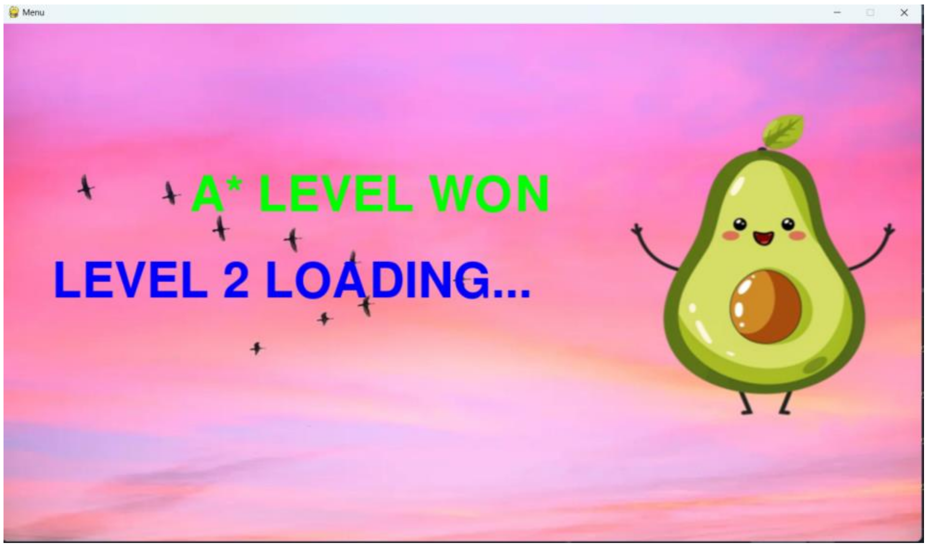
Level 1 (A* Algorithm Applied):

- If any coin is missed, the bird can go to backwards after the last of coin in x-direction has been collected.
- After collecting all coins, level 1 won showing a happy avocado
- Within 5000 millisecond, level 2 starts.



Score: 7
Distance to nearest coin: 167
Direction: dx=1.0, dy=0.02

Figure 6: Level 1 Continued



A* LEVEL WON
LEVEL 2 LOADING...

Figure 7: After Collecting all Coins (Win)

Level 2 (Alpha-Beta Algorithm and Fuzzy Logic Applied):

In the level 2 section, there are three rules to play it:

- If bird touches the ground, level 2 fails.
- If bird touches any pipe, level 2 fails.
- The bird has to collect all leaves.
- Here, Alpha-Beta algorithm provides the bird the direction through the arrow to the nearest leaf so that it can be collected as soon as possible without touching any of the pipes

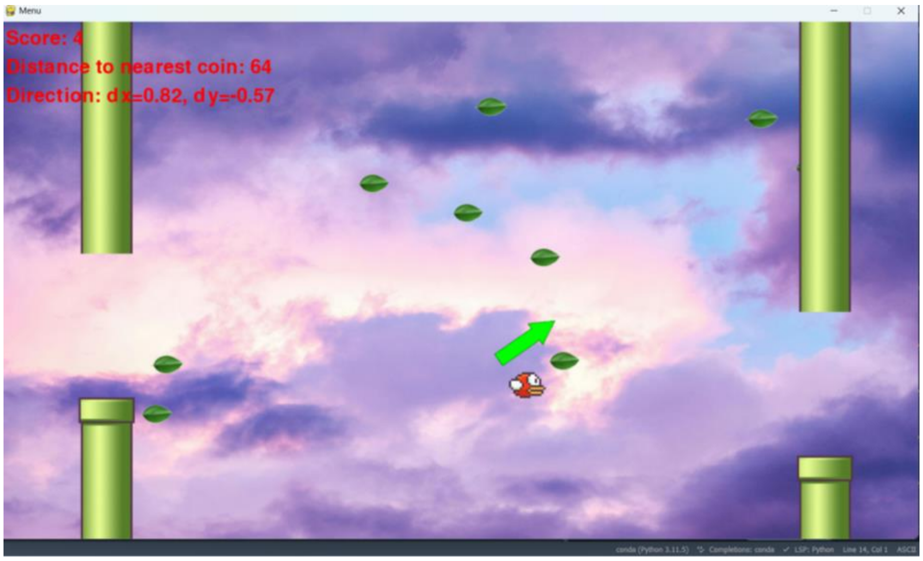

Score: 4
Distance to nearest coin: 64
Direction: dx=0.82, dy=-0.57

Figure 8: Level 2 Starting

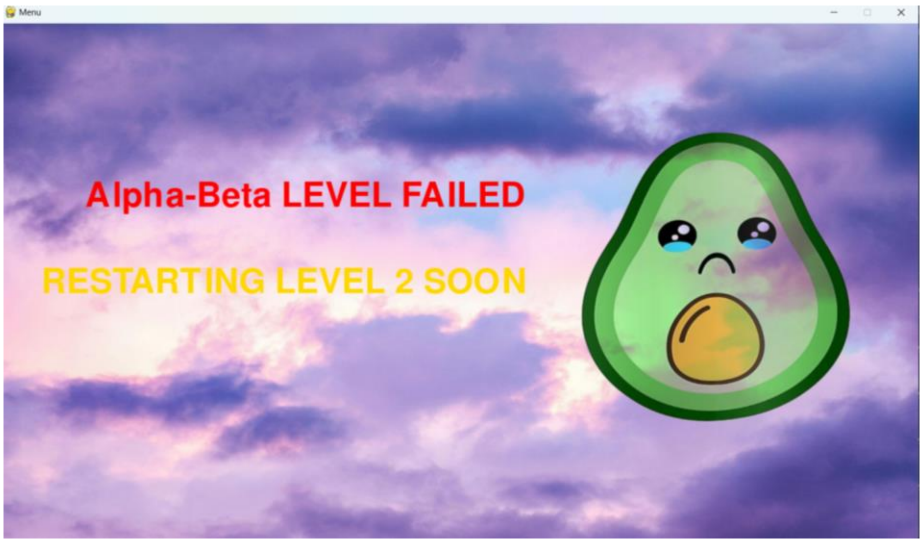
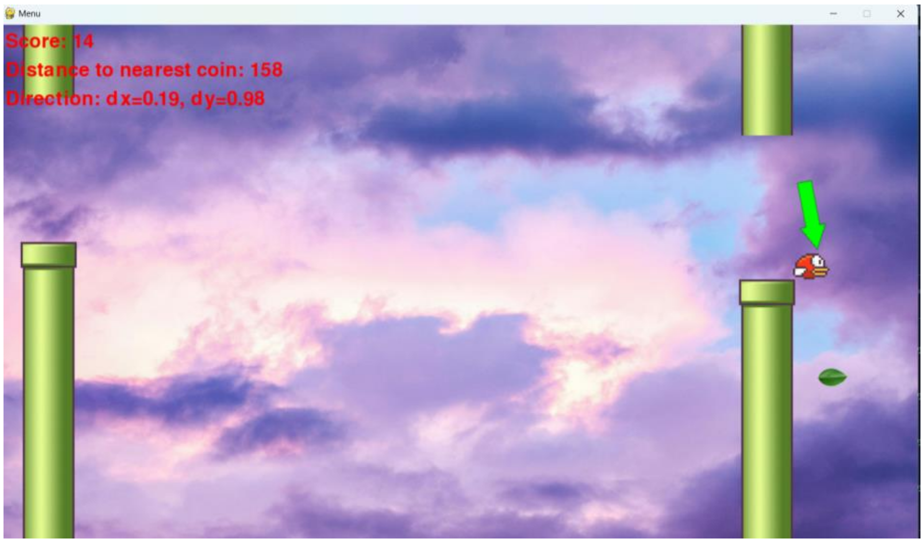

Alpha-Beta LEVEL FAILED
RESTARTING LEVEL 2 SOON

Figure 9: Ground or Pipe touched level failed

Level 2 (Alpha-Beta Algorithm and Fuzzy Logic Applied):

- No leaf can be missed as it's forwarding with the bird also but static positions
- After collecting all leaves, level 2 won showing a happy avocado
- Within 5000 millisecond, level 3 starts.
- Pipes are generated randomly



Score: 14
Distance to nearest coin: 158
Direction: dx=0.19, dy=0.98

Figure 10: Level 2 Ending Positions

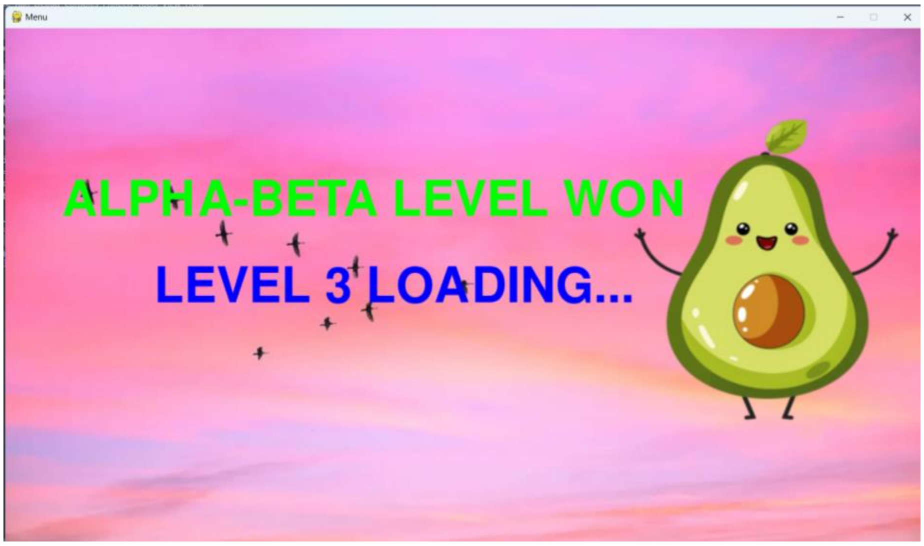


Figure 11: Wining of level 2

Level 3 (Genetic Algorithm and Fuzzy Logic):

For level 3 the rules are –

- If the bird touches the ground ,sky or pipe then it will be game over.
- The pipes will appear randomly and the coins will appear in the middle of some of the top and bottom pipes randomly.
- The speed of the bird will keep increasing according to the score using fuzzy logic.

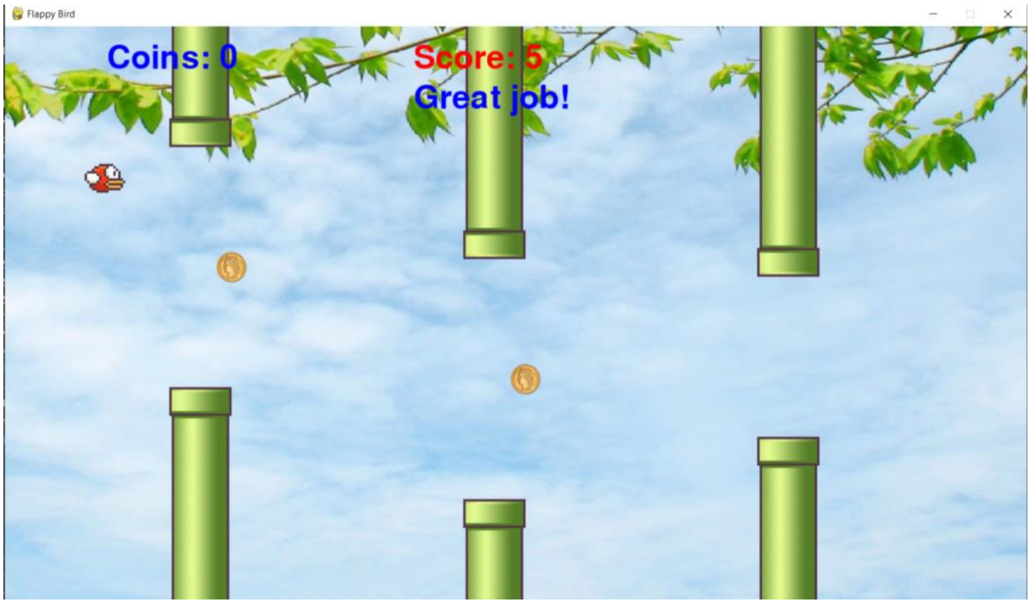


Figure 12: Level 3 appearance of the coins and pipes(random)

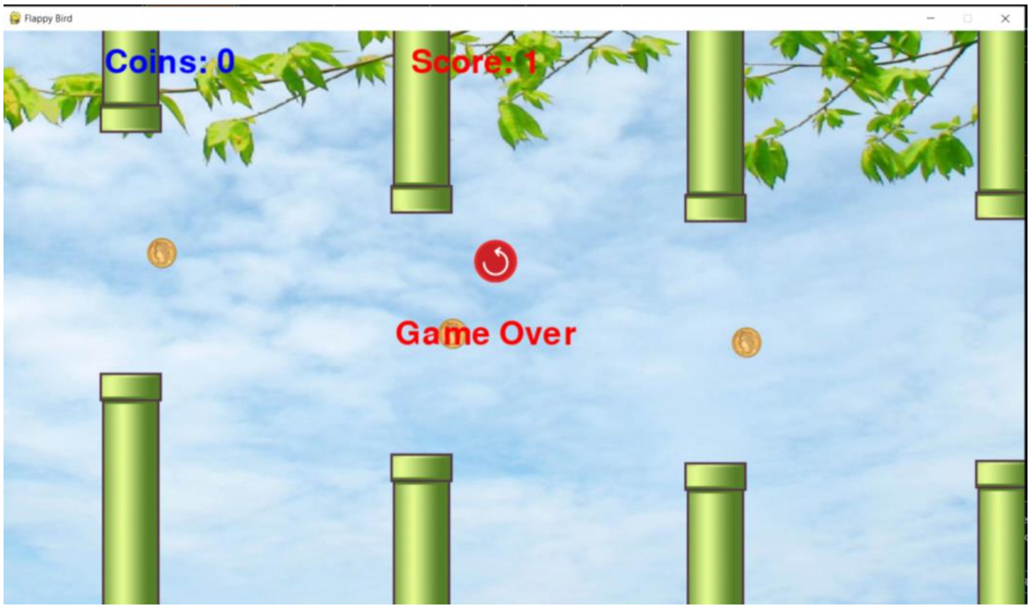


Figure 13: Game Over Level3

Level 3 (Genetic Algorithm and Fuzzy Logic):

- If the player is able to collect 5 coins a boost button will appear. A player can purchase it using 5 coins .
- If the boost button is purchased it will take the bird to the auto play space mode.
- In this mode 2000 individuals of the bird will be created. Only the best individuals will survive and overcome all the pipes.
- Then the points collected in the auto play is added to the score earned by the player. After overcoming some pipes the bird will come back to the normal mode.

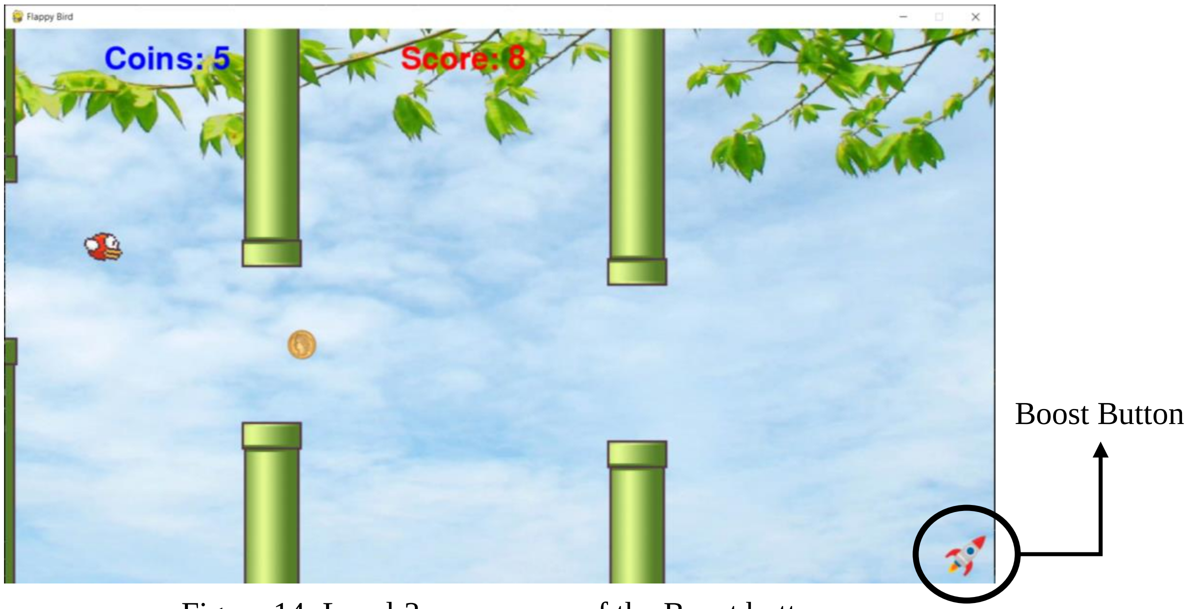


Figure 14: Level 3 appearance of the Boost button

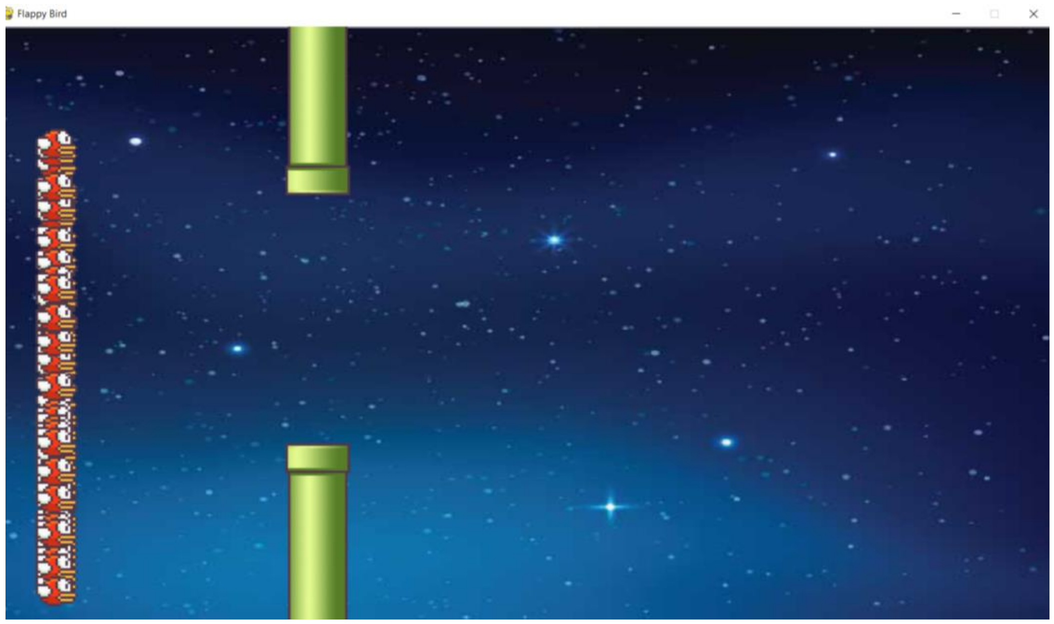


Figure 15: Space auto play mode

Level 3 (Genetic Algorithm and Fuzzy Logic):

- If the player survives a fixed amount of time he/she will win the game.
- Than the restart button will take the player to the main menu of the game

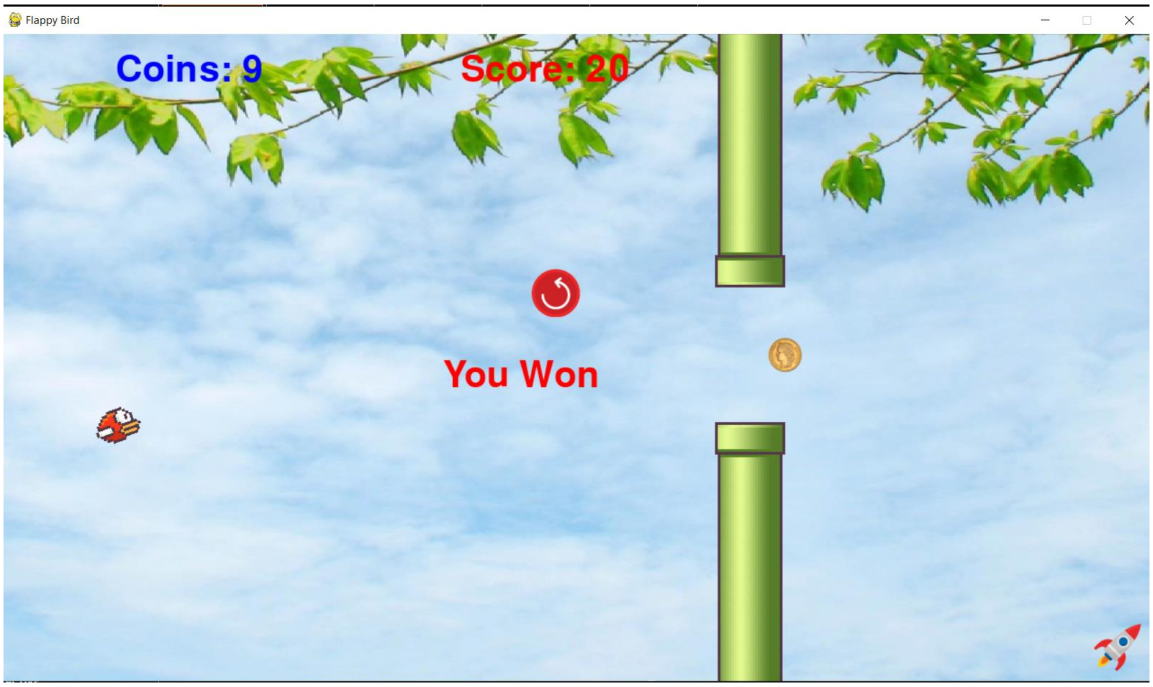
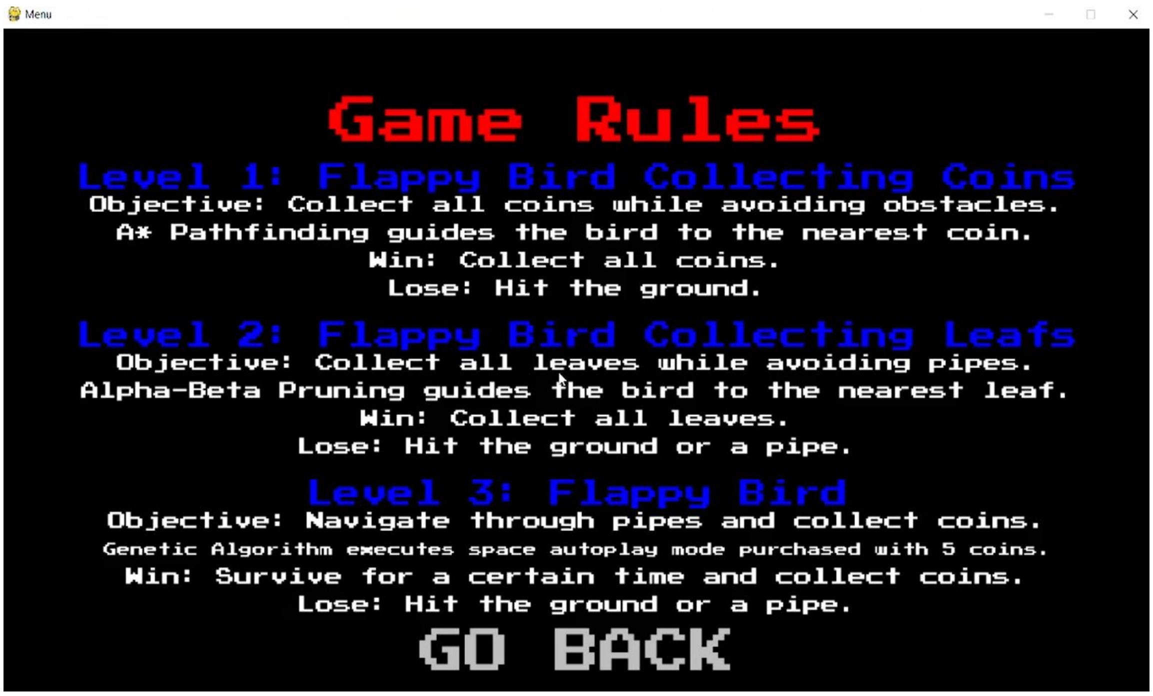


Figure 16: Winning Level3

Gaming Rules in the options section:



Menu

Game Rules

Level 1: Flappy Bird Collecting Coins
Objective: Collect all coins while avoiding obstacles.
A* Pathfinding guides the bird to the nearest coin.
Win: Collect all coins.
Lose: Hit the ground.

Level 2: Flappy Bird Collecting Leafs
Objective: Collect all leaves while avoiding pipes.
Alpha-Beta Pruning guides the bird to the nearest leaf.
Win: Collect all leaves.
Lose: Hit the ground or a pipe.

Level 3: Flappy Bird
Objective: Navigate through pipes and collect coins.
Genetic Algorithm executes space autoplay mode purchased with 5 coins.
Win: Survive for a certain time and collect coins.
Lose: Hit the ground or a pipe.

GO BACK

Figure 17: Options Section

Contribution:

Roll: **1907083**	Roll: **1907087**
UI Design (Button, Images, Font)	UI Design (Images, Audio)
Level 1: A* Algorithm Level 2: Alpha-Beta and Fuzzy Logic	Level 3: Genetic Algorithm and Fuzzy Logic

Conclusion:

This game has successfully integrated advanced artificial intelligence techniques to enhance gameplay and provide a challenging, engaging experience for players. The A* algorithm dynamically adjusts the bird's path in Level 1, ensuring efficient coin collection and obstacle avoidance. The Alpha-Beta Pruning in Level 2 guides the bird while collecting leaves, with fuzzy logic adjusting game parameters for a smoother experience. The genetic algorithms in Level 3 is implemented to evolve obstacle patterns and boost strategies, maintaining player interest and increasing difficulty levels. The successful integration of these techniques in a classic game like Flappy Bird makes the game more innovative , interactive and entertaining. The game not only entertains but also serves as a learning tool for understanding the implementation and impact of AI in interactive environments.