
# Project Report

# Cat and Mouse Split-Screen Game Report

Artificial Intelligence CS-351

**Authors:**

Afnan Bin Abbas (2022048)
Aqib Shakeel (2022104)

December 27, 2024

# Table of Contents

# Abstract

The *Cat and Mouse Split-Screen Game* is a 1-Player game where the objective of the player and AI is to execute mice by strategically placing traps that deal damage to mice within a grid-based environment. The AI utilizes a Genetic Algorithm (GA) to evolve its trap placement strategy, becoming progressively more developing over multiple generations. This report outlines the problem, the use of GA to solve it, the methodology, key results, and possible future improvements.

The game employs a split-screen design to enhance immersion and interactivity. Visual indicators, sound effects, and a real-time scoring system augment the player experience. This report explores the game's approach to addressing the limitations of static AI, describes the GA implementation, and evaluates the results. Applications for adaptive AI extend to areas like military strategy and surveillance systems. This report also discusses how the game was developed using Python and what functions werse used to achieve the implementation of Genetic Algorithm, and the underlying logic for evolving AI strategies.

You can get the Project on Github:
https://github.com/AfnanBinAbbas/CS351-AI-Semester-Project


# 1 Introduction

The *Cat and Mouse Split-Screen Game* is set in a grid-based environment, where the player controls the left side of the screen and the AI controls the right side. Each player's objective is to catch mice within their own designated area. The player and the AI compete to trap all the mice first, with the winner being the one who eliminates all the mice in their respective section of the screen. The split-screen design allows for real-time comparison of the player's and AI's actions, highlighting the competitive nature of the game.

A key feature of this game is the AI's adaptive learning process, powered by a Genetic Algorithm (GA). Over multiple generations, the AI refines its strategy for catching mice, improving its performance as the game progresses. The game is built on evolutionary computing techniques, with the GA serving as the core mechanism for the AI's evolution. This report explores the methods used for AI evolution, discusses the development tools employed to create the game, and examines the challenges encountered during the development process.

# 2 Problem Statement

Traditional game AI is often static and predictable, which reduces the challenge and replayability for players once they learn the AI's behavior. The problem is to create an AI that can adapt its strategies dynamically to maintain an engaging and challenging experience. This project seeks to address the challenge of developing an AI that evolves over time to improve its performance and keep the player engaged.

# 3 Proposed Solution

To overcome the limitations of static AI, this project proposes using a Genetic Algorithm (GA) to create a dynamic and evolving AI opponent in a split-screen game. The AI will adapt its strategies for placing mousetraps, improving its performance over time as it learns from previous generations. The game is set in a grid where both the player and AI are tasked with killing mice within their respective grids. The AI uses the GA to evolve better trap placement strategies, while the player manually places traps by clicking on the grid.

The Genetic Algorithm works by simulating natural selection. The AI starts with an initial population of random trap placements, each represented as a chromosome. A fitness function evaluates the effectiveness of each trap placement, considering factors like the number of mice trapped and the time taken to eliminate them. Through processes such as selection, crossover, and mutation, the AI continuously evolves and refines its strategies over multiple generations. The result is a dynamic AI that adapts to the player's actions, providing a progressively challenging experience and maintaining high levels of engagement throughout the game.

# 4 Methodology

The GA operates as follows:

- **Initial Population**: The game starts with a random configuration of trap positions for the AI. Each configuration is represented as a chromosome (mousetraps configurations).

```python
class GeneticAlgorithm:
    def __init__(self, population_size, mutation_rate):
        self.population_size = population_size
        self.mutation_rate = mutation_rate
        self.population = []

    def initialize_population(self):
        """Create initial random population of mousetraps."""
        for _ in range(self.population_size):
            mousetraps = [
                (random.randint(SCREEN_WIDTH // 2 // GRID_SIZE, SCREEN_WIDTH // GRID_SIZE - 1) * GRID_SIZE,
                 random.randint(0, SCREEN_HEIGHT // GRID_SIZE - 1) * GRID_SIZE)
                for _ in range(random.randint(5, 10))
            ]
            self.population.append(mousetraps)
```

Figure 1: Initial Population Function: Randomn populations of moustraps defined

- **Fitness Function**: The fitness of each configuration is determined by how effectively the traps affect the mice. The number of damage given to the mice by the traps is a key factor in calculating fitness. Other factors, such as the number of traps used and the time it takes for the AI to eliminate all the mice, can be considered as well.

```python
def fitness(self, mousetraps, mice):
    """Calculate fitness as the number of mice in range."""
    fitness = 0
    for trap_x, trap_y in mousetraps:
        for mouse in mice:
            distance = ((trap_x - mouse.x) ** 2 + (trap_y - mouse.y) ** 2) ** 0.5
            if distance <= GRID_SIZE * 3:  # Effective range
                fitness += 1
    return fitness
```

Figure 2: Fitness Function Definition

- **Selection**: After evaluating the fitness of each configuration, the top performers are selected for reproduction. This process uses a tournament selection method, where a set number of configurations are randomly selected, and the one with the highest fitness is chosen.

```python
def select_parents(self, mice):
    """Select parents based on fitness."""
    fitness_scores = [self.fitness(mousetraps, mice) for mousetraps in self.population]
    sorted_population = [x for _, x in sorted(zip(fitness_scores, self.population), reverse=True)]
    return sorted_population[:2]
```

Figure 3: Parent Selection Definition

- **Crossover and Mutation**: The selected configurations undergo crossover, where parts of two configurations are combined to produce a new configuration. Mutation introduces small, random changes to the configurations to introduce variability. This helps the AI explore new strategies that might not have been considered before.

```python
def crossover(self, parent1, parent2):
    """Perform crossover to create offspring."""
    split = len(parent1) // 2
    child = parent1[:split] + parent2[split:]
    return child

def mutate(self, mousetraps):
    """Mutate some traps with random positions."""
    if random.random() < self.mutation_rate:
        index = random.randint(0, len(mousetraps) - 1)
        mousetraps[index] = (
            random.randint(SCREEN_WIDTH // 2 // GRID_SIZE, SCREEN_WIDTH // GRID_SIZE - 1) * GRID_SIZE,
            random.randint(0, SCREEN_HEIGHT // GRID_SIZE - 1) * GRID_SIZE
        )
```

Figure 4: Crossover and Mutation Definitions

- **Replacement**: The old generation is replaced with the newly created configurations, and the process repeats for multiple generations. Over time, the AI evolves more effective trap placement strategies, resulting in better performance against the player.

```python
def create_new_generation(self, mice):
    """Generate a new population."""
    # Select the two best parents based on fitness
    parent1, parent2 = self.select_parents(mice)

    # Clear the current population
    self.population = []

    # Generate a new population
    for _ in range(self.population_size):
        child = self.crossover(parent1, parent2)
        self.mutate(child)
        self.population.append(child)
```

Figure 5: Generation of offspring

The entire process mimics the principles of natural evolution, where only the fittest configurations survive and reproduce to improve the population over time.

# 5 Game Dynamics and Split-Screen Design

The game utilizes a split-screen design, where the left side of the screen displays the player's screen where the player plays the game by placing the mousetraps (visible as a red cross) and the other side (right side) displays where the AI plays by implementing Genetic Algorithm. This division enhances immersion by allowing viewer to observe both the players' movements (of Player and AI) and trap placements in real time. The split-screen design also emphasizes the competitive nature of the game, with the player and AI racing against each other to kill all mice first.

The grid is fixed therefore the mice movements are totally random, meaning both the player and AI must adapt their strategies to the changing environment. As the game progresses, the AI learns and evolves its trap placement strategy based on the fitness function (updates configurations of mousetrap on the basis of damage given to mice) described above. The split-screen design allows the player to observe the AI's evolution, adding an extra layer of engagement and satisfaction as they see the AI become progressively smarter.

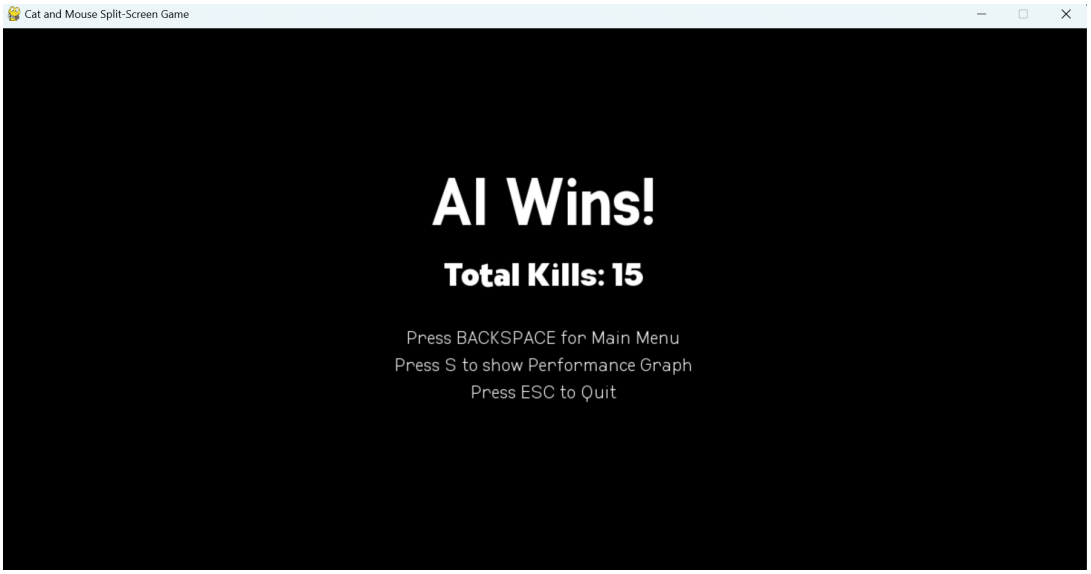
Additionally, the game includes visual and audio feedback, such as trap sound effects, mouse health indicators, and score updates. These features make the game more interactive and fun, enhancing the overall user experience


Figure 6: Main Menu Screen

**Main Menu Screen:** The main menu is the starting point of the game, where the player can begin the game or press ESC to Quit. The player presses Enter to start the game. This screen sets the tone for the game with its minimalist interface.


Figure 7: In Game Screen

**In-Game Screen:** This screen shows the live gameplay, where the player is actively placing mousetraps in their grid (shown as red crosses) while the AI is evolving and placing its traps based on its genetic algorithm. The split-screen design allows for a real-time comparison between the player's actions and the AI's evolving strategies. The grid, which is fixed, shows the random movement of the mice, and both the player and AI must adapt their strategies accordingly to capture and eliminate all mice in their respective areas.




**End Game Screen:** The end game screen appears once either the player or the AI successfully eliminates all the mice in their respective grids. This screen displays the winner, either the AI or the player, and offers several options for the player. The player can press the backspace key to return to the main menu, press 'S' to view the performance graph, or press ESC to quit the game. These options give the player flexibility in navigating the end of the game and provide a way to reflect on the performance through the graph or return to the main menu for a new session.

Figure 8: End menu

# 6 Results

The Genetic Algorithm successfully improved the AI's performance over multiple generations. As expected, the AI's trap placement strategy became more effective, leading to faster elimination of all the mice in its respective grid.

Key findings include:

- The AI's trap placement strategy became more efficient, reducing the time to trap all mice in its grid.
- The AI adapted to the player's actions, making the game more challenging by evolving its strategies over time.
- Performance data (e.g., to eliminate all mice, number of traps used, number of AI generations) was tracked and visualized through graphs. The results showed a consistent improvement in the AI's ability to clear its grid of mice, particularly after 10-15 generations.

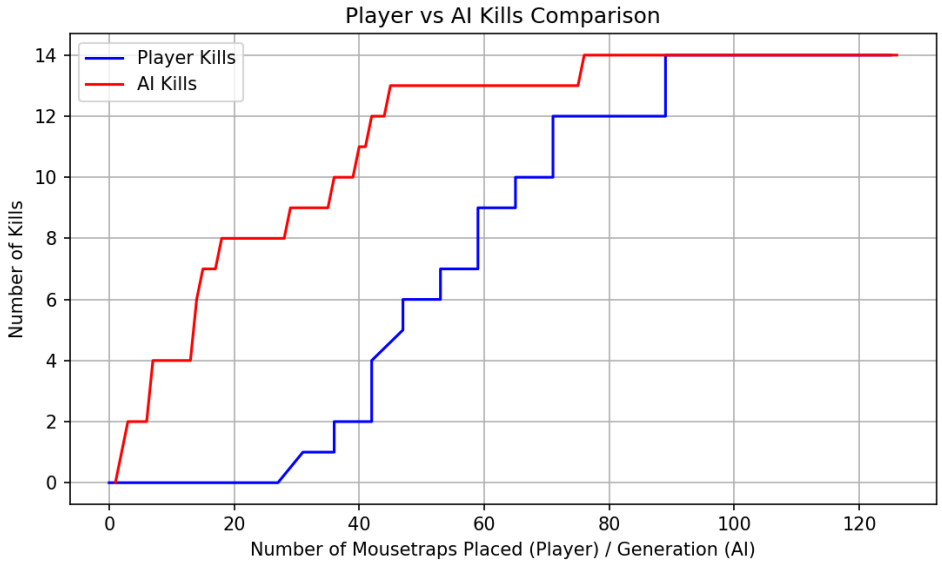



Figure 9: Player's Performance vs AI's Performance Graph; Mice Killed vs Number of Mousetraps/Generations used, Avg Mice Health

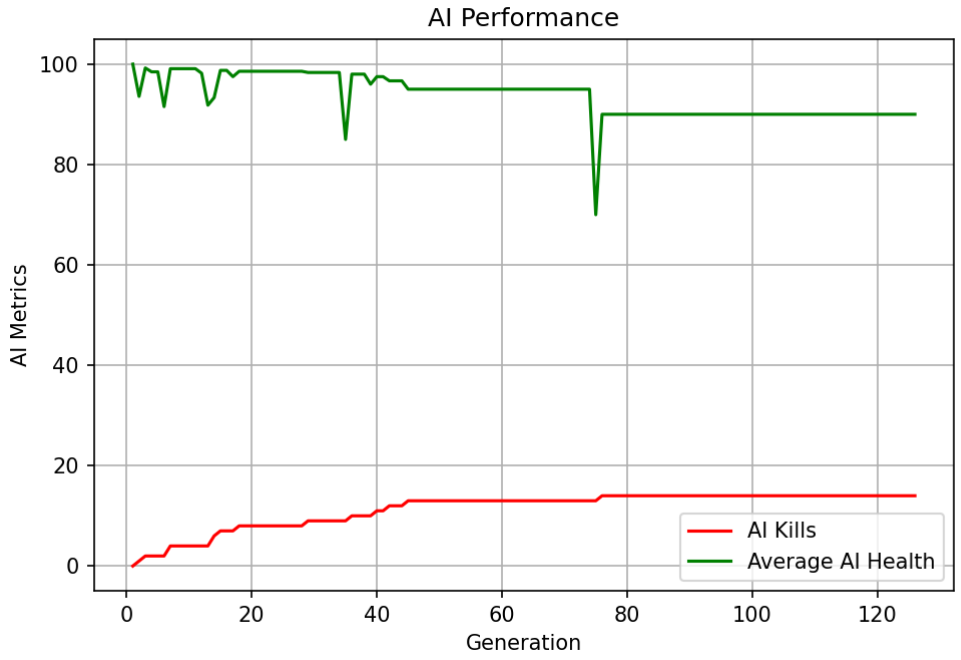



Figure 10: Player's Performance Graph; Mice Killed vs Number of Moustraps used

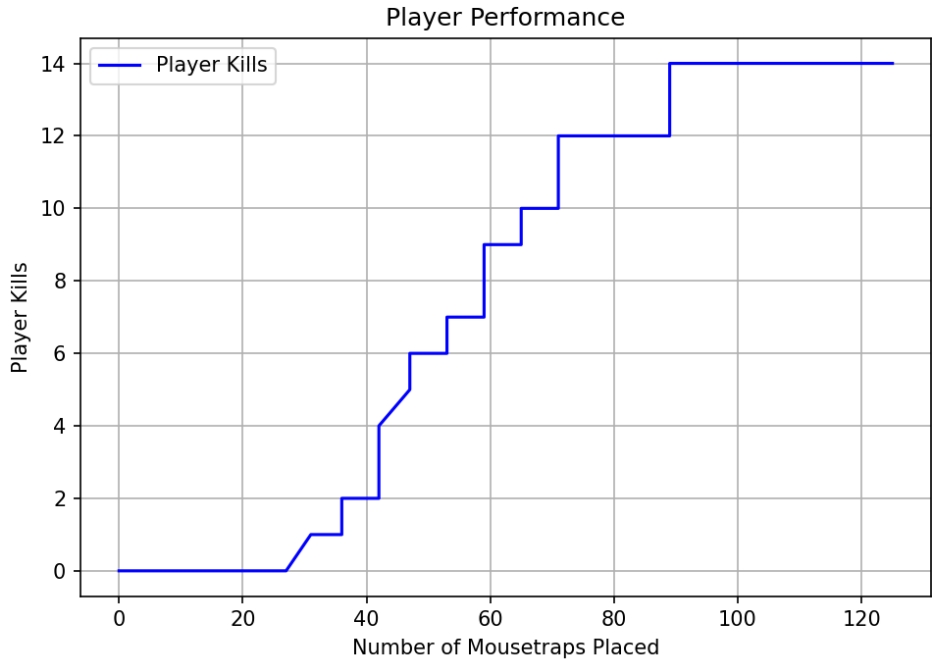



Figure 11: Player's Performance Graph; Mice Killed vs Number of Moustraps used

# 7 Conclusion

The use of the Genetic Algorithm in the *Cat and Mouse Split-Screen Game* successfully created an adaptive AI capable of evolving over time. This made the game more challenging and engaging for the player, as the AI adapted its trap placement strategies to compete against the player. Over multiple generations, the AI's performance improved, leading to faster elimination of mice in its grid, thus enhancing the overall competitive dynamic.

# 8 Future Improvements

- **AI Enhancement**: Incorporate reinforcement learning to further enhance the AI's adaptability, allowing it to learn from more complex interactions and strategies. This could include training the AI to respond to the player's changing tactics in real-time.
- **Multiplayer Mode**: Implement a multiplayer mode where two players can compete against each other by placing traps and controlling their mice. The split-screen design would allow both players to see each other's moves, adding to the competitive nature.
- **User Interface Improvements**: Improve the user interface (UI) by adding interactive menus, difficulty settings, and more visual feedback for the player, such as animations for trap activation and mouse reactions.
- **Sound and Visual Effects**: Enhance sound effects and add visual effects to improve the overall gaming experience. Implementing background music, mouse

squeals when trapped, and more diverse trap effects could improve immersion.

- **Real-Time Analytics**: Introduce real-time analytics to track the performance of both the player and AI. Providing feedback to the player on how their strategies are evolving would add an educational component to the game.