Development and Analysis of an AI-Driven Pursuit-Evasion Game: Intruder Escape

Deepti Jobi Shri Dharshan Elango Sivapriya Gopi

11 December 2024

Abstract

This project presents *Intruder Escape*, an AI-driven pursuit-evasion game implemented using Python and the pygame library. The game revolves around a grid where an intruder must evade intelligent robots that utilize A* pathfinding algorithms to pursue the player. The system integrates random grid generation, and adaptive robot speeds that escalate with player progress. Key features include grid visualization, user interaction for intruder placement, and real-time gameplay. The project serves as a practical demonstration of AI concepts like heuristic search and adaptive difficulty, fostering engagement and educational exploration in artificial intelligence and game design.

Problem Description

Overview

Pursuit-evasion scenarios represent a class of problems where one or more pursuers attempt to capture or intercept an evader within a defined environment. These problems have widespread applications in robotics, surveillance, gaming, and military strategy.

Context of Intruder Escape:

- **Environment:** A grid consisting of obstacles and open spaces.
- **Agents:** One intruder (controlled by the player) and multiple robots (pursuers controlled by AI).
- **Objective:** The intruder must navigate through the grid, avoiding capture by the robots, while the robots aim to minimize the time and distance to capture the intruder.

The interaction between the agents is governed by a combination of deterministic and dynamic elements. The robots employ the A* pathfinding algorithm, a widely recognized heuristic search method, to compute optimal paths toward the intruder. The player controls the intruder in real-time, creating an engaging dynamic that tests the effectiveness of the robots' strategy against human ingenuity.

Why This Problem is Interesting

1. Relevance to Real-World Applications

- **Robotics:** Autonomous robots often need to navigate dynamic environments to track or avoid other entities.
- **Surveillance and Security:** Developing algorithms for detecting and intercepting intruders in restricted areas.
- **Gaming:** AI-driven enemies or non-player characters (NPCs) in video games are rooted in pursuit-evasion dynamics.
- **Military and Search-and-Rescue Operations:** Coordinated systems of drones or robots need to execute efficient search-and-intercept missions in challenging terrains.

By implementing a simplified version of this problem in *Intruder Escape*, we create a platform for experimenting with strategies that could later inform these complex, real-world applications.

2. Demonstrating AI Concepts

- **Pathfinding Algorithms:** The use of A* showcases heuristic-driven search, balancing computational efficiency with solution quality.
- **Dynamic Decision-Making:** The robot's movements adapt to the intruder's actions in real time, demonstrating reactive AI behaviors.
- **Game Theory:** The interaction between the intruder and pursuers reflects adversarial problem-solving, a cornerstone of AI research.

3. Human-AI Interaction This project uniquely emphasizes the interplay between human intuition and algorithmic precision. While the player relies on quick decision-making and spatial awareness, the robots employ calculated paths to achieve their goal. This juxtaposition creates a challenging yet engaging experience that highlights the strengths and limitations of both human and AI problem-solving.

4. Scalability and Complexity

- **Environment Complexity:** By changing the grid and obstacle generation we can alter the difficulty.
- **Agent Characteristics:** Adjusting the robot's speed or number allows for testing different levels of AI proficiency.
- **Algorithm Enhancements:** Replacing A* with more advanced algorithms (e.g., D* Lite or reinforcement learning) opens avenues for exploring sophisticated AI techniques.

5. Educational and Entertainment Value The simplicity of the grid-based setup makes it accessible for educational purposes, enabling students and enthusiasts to learn AI concepts interactively. At the same time, the strategic challenge and dynamic gameplay ensure it remains entertaining.

Broader Implications

Understanding pursuit-evasion dynamics has implications far beyond this project. As AI systems become increasingly autonomous and integrated into our daily lives, ensuring their ability to navigate complex, interactive environments safely and effectively is paramount. Projects like *Intruder Escape* serve as stepping stones toward achieving these goals, fostering innovation in both academic and practical domains.

Background of the Problem

Pursuit-evasion dynamics, as exemplified in the Intruder Escape project, lie at the intersection of artificial intelligence, robotics, and game theory. This domain investigates scenarios where pursuers attempt to intercept or capture an evader within a defined space. These problems, while grounded in theoretical mathematics and algorithms, also hold significant real-world relevance, spanning applications in robotics, surveillance, gaming, and military strategy. To contextualize the Intruder Escape game, it is essential to delve into the history and foundational principles underpinning pursuit-evasion problems, as well as their evolution into modern applications.

Historical Context

The study of pursuit-evasion problems dates back to early explorations in differential game theory during the mid-20th century. Mathematicians like Isaacs introduced foundational principles in "Differential Games," focusing on optimal strategies for two adversarial agents—a pursuer and an evader. These problems were initially studied in continuous spaces, where the objective was to determine conditions for capture or escape based on the agents' motion dynamics and constraints.

In the subsequent decades, discrete models began to gain prominence, particularly with advancements in computer science and algorithmic research. Grid-based representations of environments, like those employed in Intruder Escape, became popular for their simplicity and computational tractability. They allowed researchers to model complex environments using manageable abstractions. Early algorithms, such as Dijkstra's algorithm for shortest-path computation, laid the groundwork for modern pathfinding techniques [CS10].

Evolution of Pathfinding Algorithms

A pivotal advancement in the pursuit-evasion domain was the development of heuristic-based pathfinding algorithms. A* (A-star), introduced by Hart, Nilsson, and Raphael in 1968, revolutionized search algorithms by incorporating heuristics to estimate the cost of reaching a goal [HNR68]. This innovation significantly enhanced the efficiency of solving pathfinding problems, striking a balance between computational overhead and solution optimality. In the context of pursuit-evasion, A* enables pursuers to dynamically adapt their paths to intercept the evader while navigating complex terrains. Other notable algorithms, such as D* (Dynamic A*), D* Lite, and Jump Point Search, expanded upon A* by addressing limitations in dynamic or large-scale environments [MF09]. These enhancements underscore the adaptability of pursuit-evasion frameworks to varying levels of complexity and environmental constraints.

Real-World Applications

The principles governing pursuit-evasion scenarios extend beyond academic exercises, influencing diverse fields:

Applications in Robotics

In robotics, pursuit-evasion algorithms enable autonomous agents to navigate dynamic environments. Applications range from robotic vacuum cleaners avoiding obstacles to drones tracking moving targets. These scenarios often incorporate real-time decision-making, requiring adaptive strategies to handle unforeseen changes in the environment.

Surveillance and Security

Pursuit-evasion problems are critical in designing surveillance systems capable of detecting and intercepting intruders. For instance, security robots patrolling restricted areas rely on pathfinding algorithms to optimize their pursuit strategies while navigating around obstacles. Similarly, in cybersecurity, analogous principles are applied to track and neutralize malicious activities within network systems.

Gaming

Video games frequently draw inspiration from pursuit-evasion dynamics to enhance player engagement. AI-controlled non-player characters (NPCs) often employ pathfinding algorithms to chase or evade players, creating interactive and challenging gameplay. The accessibility and versatility of grid-based mazes make them a staple in game design, fostering creativity and experimentation [BBSL11].

Military and Search-and-Rescue Operations

In military contexts, coordinated pursuit-evasion strategies are deployed in scenarios involving autonomous vehicles or drones. These systems aim to efficiently locate and intercept targets in complex terrains. Similarly, search-and-rescue missions leverage pursuit-evasion principles to navigate through disaster-stricken areas, optimizing the search for survivors while avoiding hazards.

Challenges and Opportunities

Despite the progress in pursuit-evasion research, several challenges remain. Dynamic environments, where obstacles and agent behaviors evolve over time, introduce layers of complexity that require advanced algorithms. Furthermore, the unpredictability of human-controlled agents, as seen in Intruder Escape, adds a dimension of uncertainty, challenging the AI to anticipate and counteract human ingenuity.

The development of Intruder Escape underscores the educational and experimental potential of pursuit-evasion problems. By providing a controlled yet engaging platform, it encourages learners and practitioners to explore the intricacies of AI-driven decision-making, heuristic search, and adversarial problem-solving. Through iterative experimentation, such projects pave the way for advancements in both theoretical understanding and practical implementations of pursuit-evasion dynamics.

Approach Description

Representation

To develop the AI-driven pursuit-evasion game *Intruder Escape*, we employed a grid-based representation for both the environment and the movement of agents. This decision stemmed from the need for simplicity and clarity in visualizing the interaction between the intruder (player-controlled) and the robots (AI-controlled pursuers). The game environment consists of:

- **Grid:** A two-dimensional array where each cell represents either an obstacle, open space, or an agent. Obstacles are dynamically generated, ensuring varied layouts in each game session.
- **Agents:** The intruder is placed on the grid by the player, while the robots start at predetermined locations. Both types of agents adhere to grid constraints, moving only up, down, left, or right.
- **Shared Tracking System:** A shared structure records the positions of all robots and the intruder in real time, enabling coordinated interactions and collision avoidance.

The representation leverages the Manhattan distance as a heuristic for pathfinding, given the grid's restriction to orthogonal movement. This choice ensures computational efficiency while aligning with the problem's dynamics.

Algorithms

Pathfinding: A* Algorithm

To enable intelligent movement for the robots, we implemented the A* algorithm. A* is a heuristic-driven search method that balances path optimality and computational efficiency. It evaluates potential paths based on two criteria:

1. **Cost to Reach the Current Node (g):** Represents the actual distance traveled from the robot's starting position to a given grid cell.
2. **Estimated Cost to the Goal (h):** The Manhattan distance from the current cell to the intruder's position.

The algorithm computes the total cost (f) as the sum of these two values ($f = g + h$) and selects the path with the lowest total cost. This approach allows the robots to dynamically adapt their routes in real-time as the intruder's position changes.

Dynamic Path Recalculation

One challenge encountered during development was determining the frequency of path recalculations. Frequent updates ensured high responsiveness but caused performance bottlenecks, while infrequent updates reduced the robot's effectiveness. We addressed this by implementing a recalibration interval based on the intruder's movement. Robots only recalculated paths when the intruder moved a specified number of steps, balancing responsiveness and efficiency.

Multi-Agent Coordination

Coordinating multiple robots required additional considerations to avoid collisions and redundant behaviors [Rey87]. Each robot accesses the shared position tracking system to:

- **Avoid Collisions:** Robots dynamically adjust their paths if another robot's planned movement conflicts with their own.
- **Distribute Pursuit:** Each robot independently calculates its path to the intruder, resulting in diverse pursuit trajectories and emergent tactical behaviors without explicit programming for formations.

Difficulty Progression

The difficulty progression was essential for maintaining engagement. Through playtesting, we observed that skilled players could evade the robots indefinitely at a fixed speed. To address this, we:

- **Increased Robot Speed:** Robots gain a 10% speed boost every 100 steps taken by the player, capped at a maximum speed to ensure the game remains playable.
- **Adaptive Pursuit Behavior:** By incrementally escalating the challenge, players experience a natural gameplay arc, transitioning from an introductory phase to progressively intense encounters.

Environment Management

The game environment balances randomness with strategic design:

- **Obstacle Generation:** Obstacles are placed randomly but validated to ensure navigable paths exist between key areas. This prevents scenarios where the intruder or robots are trapped.
- **Collision Detection:** Movement validation checks ensure agents cannot occupy the same cell or pass through walls. These rules simplify the grid mechanics and enhance predictability for both players and robots.
- **Game State Management:** The overall system is divided into distinct phases—setup, gameplay, and game over—to structure the game's logic and ensure smooth transitions.

Gameplay Dynamics

The core movement mechanics are intentionally restricted to four directions, creating tactical depth where positioning and timing are critical. This limitation fosters engaging decision-making, as players must anticipate robot paths while planning their own evasive maneuvers. Each action is validated for legality, ensuring consistent gameplay.

Scoring and Feedback

The scoring system tracks survival time based on steps taken by the intruder, offering players a clear and intuitive metric for success. This approach encourages replayability, as players strive to improve their performance against increasingly challenging AI opponents.

Experiments and Results

We experimented the project using both A* and Uniform Cost Search (UCS) algorithms to evaluate their performance and decide on the most suitable approach for our needs. After extensive testing, we arrived at the following results:

Performance Comparison

Speed

A* significantly outperformed UCS, with an average runtime of 0.08 milliseconds compared to UCS's 0.17 milliseconds. This demonstrates that A* is more computationally efficient in solving pathfinding problems.

The time ratio (UCS/A*) was measured at 2.15x, meaning UCS took over twice as long as A* to compute results.

Efficiency

A* explored only 22.6 nodes on average, whereas UCS explored 100.3 nodes, indicating that A* was 4.43 times more efficient in narrowing down its search space due to its heuristic-driven approach.

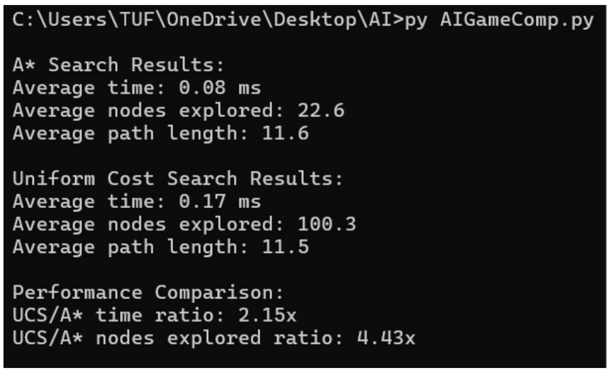
Despite this, both algorithms produced near-identical optimal paths (A*: 11.6, UCS: 11.5) in terms of path length, demonstrating the reliability of both methods in finding optimal solutions.

Heuristic Advantage

The heuristic function used by A* provided a significant advantage, enabling it to focus on promising paths and avoid unnecessary exploration. In contrast, UCS exhaustively explored nodes in a cost-agnostic manner, which increased computational overhead.

Decision to Use A*

Based on the above analysis, we chose to proceed with A* for the project. Its faster runtime and higher efficiency made it the optimal choice for our application, particularly



```
C:\Users\TUF\OneDrive\Desktop\AI>py AIGameComp.py

A* Search Results:
Average time: 0.08 ms
Average nodes explored: 22.6
Average path length: 11.6

Uniform Cost Search Results:
Average time: 0.17 ms
Average nodes explored: 100.3
Average path length: 11.5

Performance Comparison:
UCS/A* time ratio: 2.15x
UCS/A* nodes explored ratio: 4.43x
```

Figure 1: A* vs UCS

in scenarios where computational resources or real-time responsiveness are critical. While UCS remains a reliable algorithm, its lack of heuristic guidance makes it less suitable for large or dynamic environments where efficiency is paramount.

Thus, A* has become the cornerstone of our pathfinding solution, aligning with our goals for performance and scalability.

Visualization

We initially considered two options either Tkinter or Pygame. While Tkinter is suitable for basic GUI applications, we opted for Pygame due to its focus on game development. Pygame offers a simpler approach to creating games, with built-in functions for precise action reading and realtime updation. Additionally, Pygame's active community provides ample resources and tutorials, making it easier to learn and implement.

Analysis of the Results

In this project, we developed a grid-based AI-driven pursuit-evasion game, **Intruder Escape**. The game employs various dynamic states that represent the interaction between the intruder (controlled by the user) and the robots (AI-controlled pursuers). The game offers several key features, including open spaces (white cells), obstacles (grey cells), a clear grid, and options to place the intruder, start the game, and switch to a new random layout. Below is a description of each state depicted in the game's images.

- **Start of the Game - Random Grid Generation (Refer Figure 2: Start):**

  In this initial state, a random grid is generated. The grid consists of **white cells** representing open spaces and **grey cells** indicating obstacles that prevent movement. The game environment is created dynamically, where the obstacles are randomly placed in the grid. Four robots are also placed at random positions on the grid, preparing for their pursuit of the intruder. The intruder has not yet been placed, and the user can begin interacting with the game by using the available control options.

  - **White cells** represent open spaces where the intruder and robots can move.
  - **Grey cells** are obstacles that block movement, adding complexity to the navigation.

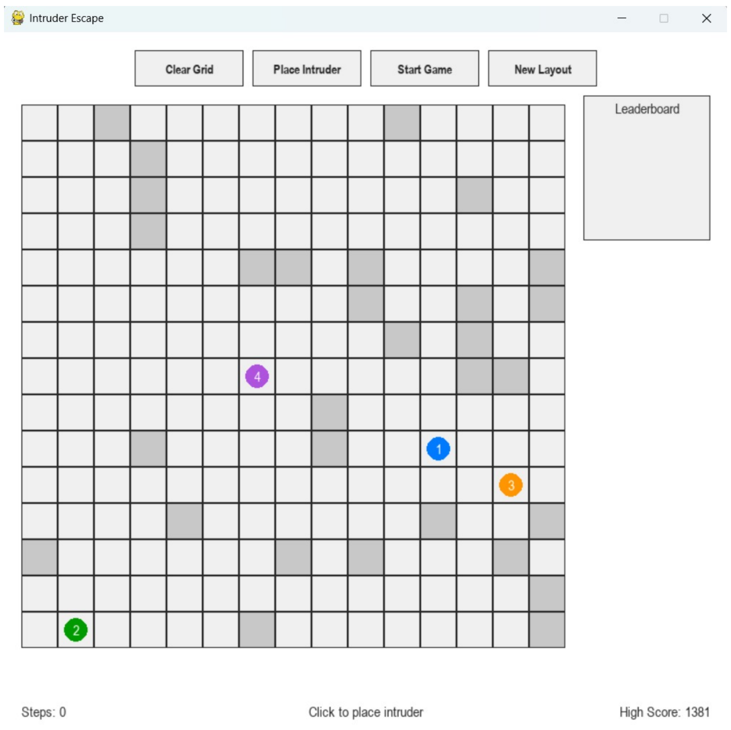


Figure 2: Start

- Robots are randomly placed in different positions, but their starting locations are not yet interacting with the intruder.

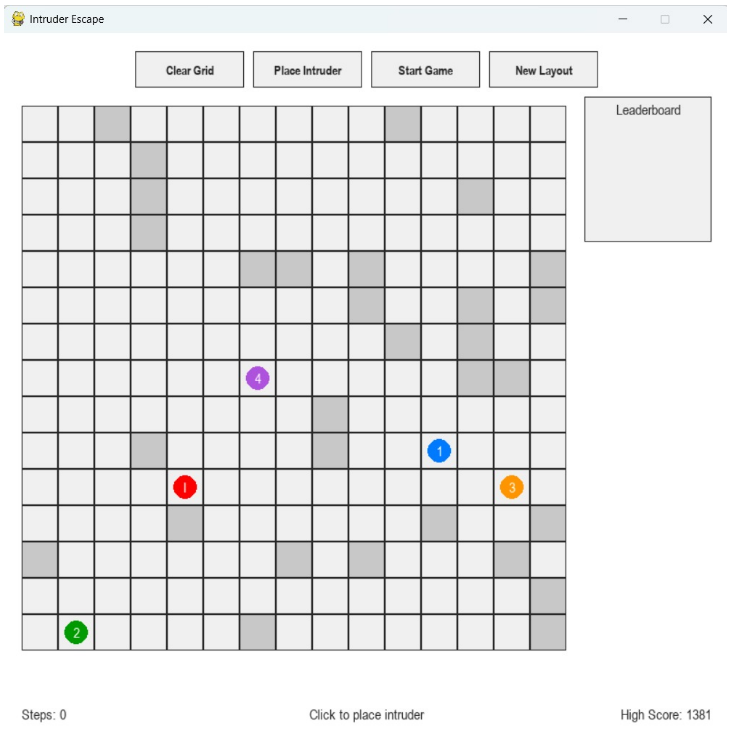
- **User Placing the Intruder at a Desired Position (Refer Figure 3: Intruder Placed):**

  In this image, the user is given the option to place the intruder at any desired position on the grid. The **Place Intruder** feature allows the player to select a spot on the grid where the intruder will begin their escape. Once the intruder is placed, the user can then proceed to start the game by selecting the **Start Game** button. The game begins once the intruder is placed, and the robots will immediately begin pursuing the intruder based on the A* pathfinding algorithm.

  - The user has the flexibility to place the intruder at any open cell (white cell).
  - The game interface includes a **Start Game** button, which activates the game after the intruder is placed.

- **Intruder in Action - Moving to Escape the Robots (Refer Figure 4: Intruder Control):**

  In this state, the user controls the intruder using the arrow keys to move through the grid. The **robots**, controlled by AI, follow the intruder using the A* pathfinding algorithm, continuously recalculating the shortest path towards the intruder's position. The challenge lies in the user's ability to navigate the grid, avoiding obstacles (grey cells) and evading capture by the robots. The robots' speed gradually increases as the game progresses, adding increasing pressure on the user to escape.

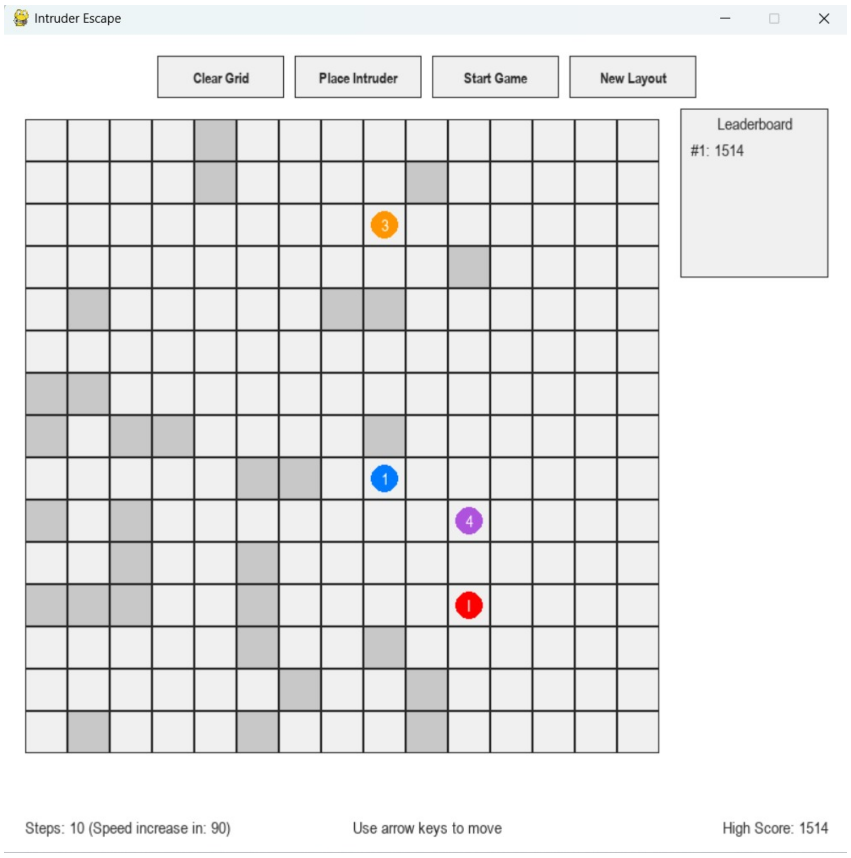


Steps: 0

Click to place intruder

High Score: 1381

Figure 3: Intruder Placed



Leaderboard

#1: 1514

Steps: 10 (Speed increase in: 90)

Use arrow keys to move

High Score: 1514

Figure 4: Intruder Control

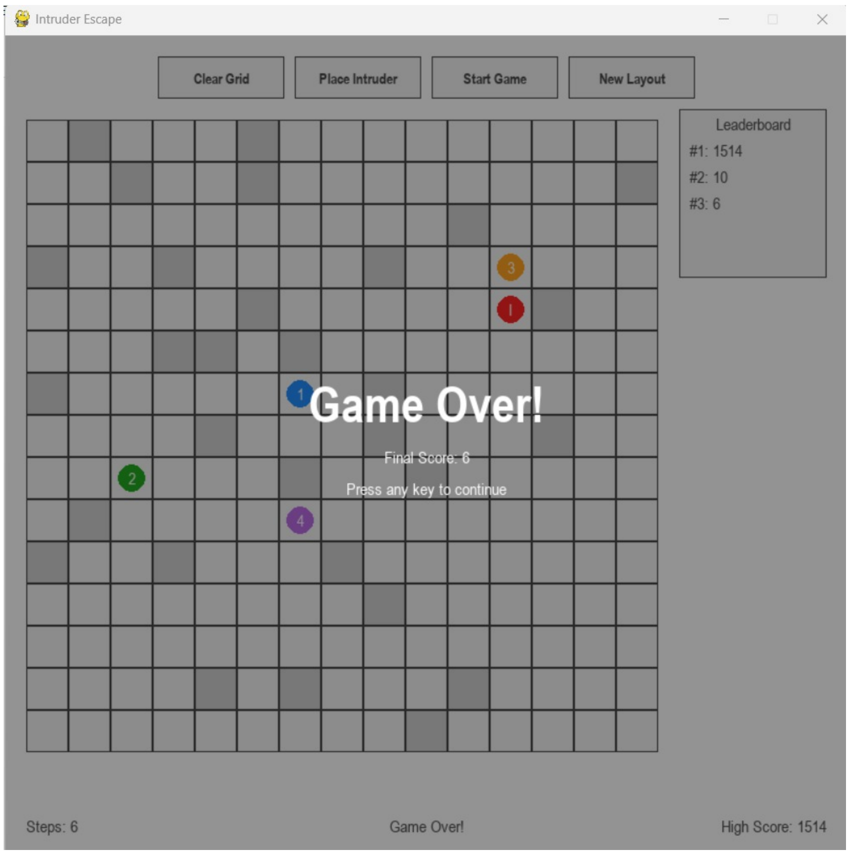


Figure 5: Game Over

- The **arrow keys** are used to control the intruder's movement across the grid.
- Robots pursue the intruder by calculating optimal paths in real-time.
- Obstacles and the grid layout affect both the intruder's and robots' movements.
- The user must strategically navigate to avoid capture.

- **Game Over - Intruder Caught, Leaderboard Displayed (Refer Figure 5: Game Over):**

  In the game over state, the **intruder** has been caught by the robots, marking the end of the game session. This image shows the final scenario where the robots surround the intruder, signaling the end of their escape attempt. On the side of the screen, the **Leaderboard** is displayed, showing the number of steps the intruder took before being caught and the **highest score** achieved in the game. The score is determined based on how long the intruder successfully evades the robots. The leaderboard serves as a motivational tool for the player to aim for a higher score in subsequent games.

  - The **Game Over** message appears when the intruder is caught.
  - The **Leaderboard** on the side tracks the number of steps taken by the intruder and displays the highest score at the top.
  - The leaderboard provides feedback to the player, encouraging them to improve their evasion skills.

- **Additional Game Controls:**
  - **Clear Grid:** Clears the grid and resets the game, allowing the player to start fresh.
  - **New Layout:** Generates a new random grid with a fresh set of obstacles and robot placements, providing variety in gameplay and increasing replayability.

This sequence of images illustrates the dynamic nature of the **Intruder Escape** game, from the random grid generation to the final game over state. The interaction between the intruder and robots is influenced by the layout of obstacles, the user's movement decisions, and the AI's pathfinding behavior. The leaderboard ensures that players can track their progress and challenge themselves to improve their performance in future sessions.

Conclusion

The development and analysis of Intruder Escape, an AI-driven pursuit-evasion game, effectively demonstrates the application of heuristic pathfinding algorithms and dynamic difficulty adjustments within a simple yet engaging gameplay structure. The grid-based environment, complemented by the use of the A* algorithm, enables intelligent robot behavior that provides a challenging experience for the player. Through dynamic path recalculation and multi-agent coordination, the robots adapt to the intruder's movements, fostering a competitive and immersive environment. The difficulty progression, driven by incremental speed boosts and adaptive AI strategies, ensures that players are consistently engaged while also preventing frustration.

Emergent behaviors observed during playtesting—such as robots unintentionally forming trapping patterns or being drawn into single-file pursuits—add an additional layer of complexity and interest to the game. These behaviors highlight the effectiveness of the AI's decision-making process and contribute to the game's replayability, as players experiment with different strategies to evade capture.

Future Work

There are several avenues for enhancing **Intruder Escape** to increase its depth and challenge. Below are the key areas for future development:

- **Diagonal Movement:**
  - Introducing diagonal movement would expand tactical options for both the player and robots.This change would require rebalancing the robots' movement logic to prevent unfairness or unnecessary complexity.
- **Terrain Types:**
  - Adding different terrain types (e.g., areas that slow movement or provide temporary cover) could introduce strategic layers.Players would need to plan evasion tactics while also utilizing the environment to their advantage.

- **Improved Robot Coordination:**
  - Introducing more sophisticated multi-agent behaviors could allow robots to act in coordination to trap the intruder.This would mimic more complex, real-world multi-agent systems and require careful balancing to avoid making the game too difficult or unfair.
- **Enhanced AI Capabilities:**
  - Expanding the game's AI could involve incorporating advanced algorithms like D* Lite or even reinforcement learning.This would enhance the robots' adaptability, allowing them to learn and evolve their strategies in real-time based on the player's behavior.

Bibliography

References

[BBSL11] Vadim Bulitko, Yngvi Björnsson, Nathan R. Sturtevant, and Ramon Lawrence.

[CS10] *Real-Time Heuristic Search for Pathfinding in Video Games*, pages 1–30. Springer New York, New York, NY, 2011.

[HNR68] Xiao Cui and Hao Shi. A*-based pathfinding in modern computer games. *IJCSNS International Journal of Computer Science and Network Security*, 11, 11 2010.

[MF09] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[Rey87] Ian Millington and John Funge. *Artificial Intelligence for Games.* CRC Press, 2009.