**Name & ID:**

**Name & ID:**

**Name & ID:**

**Hide and Seek Game: Technical Analysis Report**

* **Game Concept Overview:** 
  + - **The implementation represents a strategic "Hide and Seek" game using game theory principles. In this two-player zero-sum game:**
      * **Game World**: A grid of cells (rows × columns) or Linear with the size only.
      * **Players**: Hider and Seeker
      * **Difficulty Levels**: Each cell has a random difficulty level (easy, medium, hard).
      * **Payoffs**: Values determined by difficulty levels that represent the scoring outcomes
      * **Modes**:
        + Human vs. Computer: Player chooses to be either the Hider or Seeker.
        + Computer Simulation: Both roles played by computer using optimal strategies

**The core objective is to apply game theory to determine optimal strategies for both players, creating an AI opponent that makes rational decisions based on mathematical principles.**

* **Game Theory Fundamentals:**
  + - **Zero-Sum Game Structure:**

**This implementation models a classic zero-sum game where one player's gain exactly equals the other player's loss. Key game theory concepts applied include:**

1. **Payoff Matrix**:

* **A mathematical representation of all possible outcomes.**
* **Each cell (i,j) represents the outcome when Hider chooses position i and Seeker chooses position j.**
* **Positive values benefit the Hider; negative values benefit the Seeker.**

1. **Mixed Strategy**:

* **The game uses probabilistic strategies rather than pure strategies.**
* **Each player chooses locations according to probability distributions.**

1. **Minimax Principle**:

* **Hider aims to maximize their minimum payoff.**
* **Seeker aims to minimize Hider's maximum payoff.**
* **The value of the game represents the expected outcome with optimal play.** 
  + - **Linear Programming Solution:**

**The game’s probabilities is calculated using linear programming (LP), a mathematical optimization technique:**

1. **For the Hider (Maximizing Player)**:

* **Maximize v subject to:**
* **For each Seeker position j: -p·A[:,j] + v ≤ 0**
* **Sum of probabilities equals 1**

1. **For the Seeker (Minimizing Player)**:

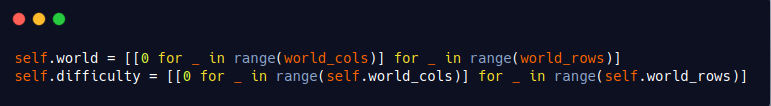
* **Minimize v subject to:**
* **For each Hider position i: p·A[i,:] - v ≤ 0**
* **Sum of probabilities equals 1**

**Where:**

* **p represents the probability distribution over positions**
* **v represents the value of the game**
* **A is the payoff matrix**
* **Data structures Used:**

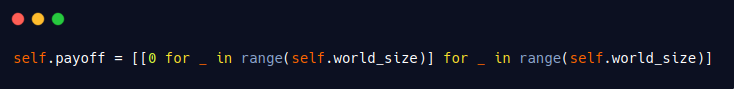
**The implementation leverages several key data structures to represent and process the game state:**

1. **2D Grid Arrays :**



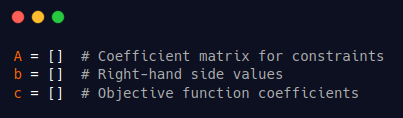
* **Purpose**: Represent the game board and difficulty levels.
* **Benefit**: Direct mapping to the visual grid for intuitive understanding.

1. **Payoff Matrix :**



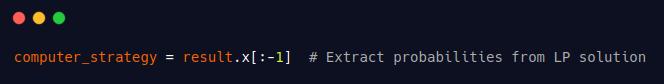
* **Purpose**: Core mathematical representation of game outcomes
* **Structure**: Row i represents Hider position, column j represents Seeker position
* **Values**: Positive values favor Hider, negative values favor Seeker

1. **Linear Programming Arrays :**



* **Purpose**: Represent the minimax optimization problem
* **Usage**: Fed into scipy's linprog solver to find optimal strategies

1. **Probability Vectors :**

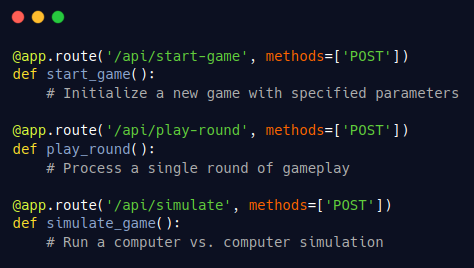


* **Purpose**: Represent mixed strategies from game theory
* **Usage**: Probabilities for selecting each position
* **Implementation**: Used with numpy's random.choice for stochastic decisions
* **Code Architecture and Implementation :**

**The codebase is structured into three main components:**

1. **Flask Web Server (main.py) :**

**This file implements the game's API endpoints using Flask:**

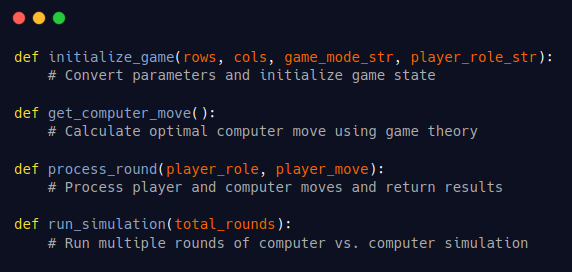


**The server handles three primary operations:**

* **Game initialization with customizable parameters**
* **Individual round processing for human vs. computer mode**
* **Game simulation for computer vs. computer mode**

1. **Game Logic Helper (helperMethods.py) :**

**This intermediate layer coordinates between the API and core game logic:**

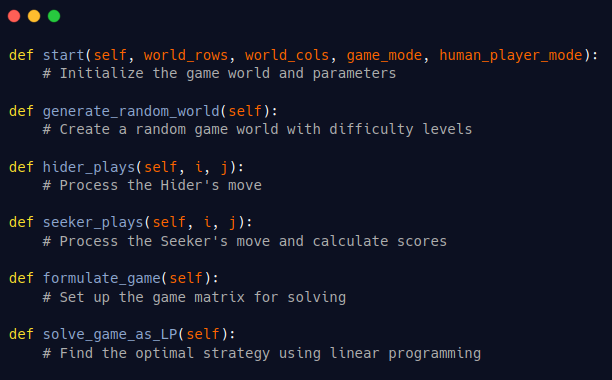


**This module handles:**

* **Parameter conversion between API and game engine**
* **Computer move calculation using the game theory solver**
* **Round processing logic**
* **Simulation execution and results aggregation**

1. **Core Game Engine (startUp.py) :**

The GameState class implements the fundamental game mechanics:



**This class manages:**

* **Game state initialization**
* **Random world generation with difficulty levels**
* **Move processing for both players**
* **Score calculation**
* **Game matrix formulation**
* **Linear programming solution for optimal strategies**
* **Algorithm Analysis :**

1. **World Generation Algorithm :**

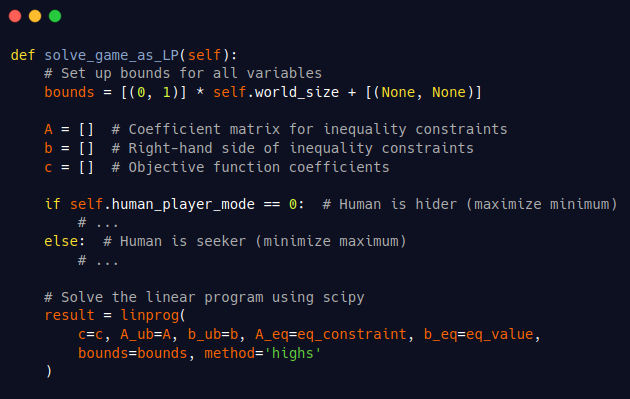
**The game world is generated with random difficulty levels (easy, medium, hard) that affect the payoff values:**

 **Key features:**

* **Each position is assigned a random difficulty (0, 1, or 2)**
* **Payoff values are determined by difficulty level**
* **Nearby positions have adjusted payoffs to create strategic depth**

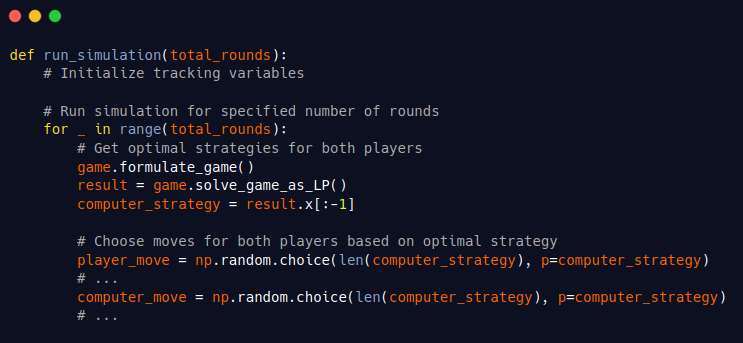
1. **Linear Programming Solution :**

**The core algorithmic component is the linear programming solver that calculates optimal mixed strategies:**

  **The algorithm:**

1. **Sets up the linear programming problem based on player role**
2. **Constructs constraints that enforce game theory principles**
3. **Uses the high-performance 'highs' solver from scipy**
4. **Returns probability distributions representing optimal strategies**
5. **Simulation Procedure :**

**The simulation mode runs multiple rounds with both players using optimal strategies:**

  **This process:**

1. **Calculates optimal mixed strategies using LP**
2. **Samples move from these probability distributions**
3. **Processes moves and updates scores**
4. **Tracks statistics across all rounds**

* **Test Case Scenarios:**
* **Bonus Requirements:**

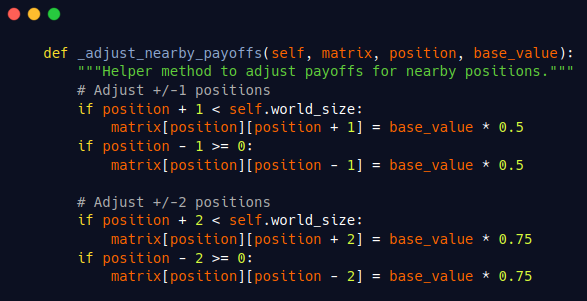
**To further enhance the strategic depth and realism of the game, two optional**  **bonus features were implemented and tested:**

1. **Proximity:**

To make the game more dynamic and fairer, a proximity-based scoring system was introduced. This system penalizes the hider based on how close the seeker’s chosen position is to their hiding spot. Specifically:

* If the seeker’s choice is **1 cell away**, the hider’s score is **multiplied by 0.5**.
* If the seeker is **2 cells away**, the hider’s score is **multiplied by 0.75**.
* If the seeker is farther than 2 cells, the hider’s score remains **unchanged**.

**Our implementation to handle it :**



**This modification encouraged more careful decision-making by the hider and made the seeking process more rewarding and challenging.**

1. **2D array:**

**We add in welcome page when the User want to add the grid size, we add 2 choices:**

1. **Linear grid**
2. **2D grid**

**This allowed for more intricate hide-and-seek dynamics, especially when applying proximity scoring in two dimensions. Movement and payoff calculations were adapted accordingly, and visualization was updated to reflect the new grid-based interface.**

**These bonus features added an extra layer of engagement and analytical depth to the game, demonstrating the versatility of game-theoretic models when combined with algorithmic strategies and user-centered design.**

* **Conclusion :**

**In this lab, we successfully implemented a Hide and Seek game grounded in game theory principles and linear programming strategies. By modeling the interaction between the hider and seeker using a payoff matrix, we analyzed optimal strategies and applied the simplex method to solve for the computer’s decisions. The game dynamically adapts to a linear world of size *N*, with various place types influencing scoring, and includes both interactive and simulation models for comprehensive testing. Additional features such as proximity scoring and a 2D grid layout were explored to enhance realism and strategic complexity. This assignment reinforced our understanding of zero-sum games, optimization techniques, and the practical application of operations research in game-based scenarios.**