Name & ID: Ayman Ibrahim Mohamed Kotb	22010656
Name & ID: Abdelrahman saeed ramadan gomaa	22010879
Name & ID: Farouk Ashraf Farouk	22011012

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.
2. Mixed Strategy:
   - The game uses probabilistic strategies rather than pure strategies.
   - Each player chooses locations according to probability distributions.
3. 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

2. 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 :

```
self.world = [[0 for _ in range(world_cols)] for _ in range(world_rows)]
self.difficulty = [[0 for _ in range(self.world_cols)] for _ in range(self.world_rows)]
```

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

2. Payoff Matrix :

```
self.payoff = [[0 for _ in range(self.world_size)] for _ in range(self.world_size)]
```

- **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

3. **Linear Programming Arrays :**

```
A = []  # Coefficient matrix for constraints
b = []  # Right-hand side values
c = []  # Objective function coefficients
```

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

4. **Probability Vectors :**

```
computer_strategy = result.x[:-1]  # Extract probabilities from LP solution
```

- **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:

```python
@app.route('/api/start-game', methods=['POST'])
def start_game():
    # Initialize a new game with specified parameters

@app.route('/api/play-round', methods=['POST'])
def play_round():
    # Process a single round of gameplay

@app.route('/api/simulate', methods=['POST'])
def simulate_game():
    # Run a computer vs. computer simulation
```

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

2. Game Logic Helper (`helperMethods.py`) :

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

```python
def initialize_game(rows, cols, game_mode_str, player_role_str):
    # Convert parameters and initialize game state

def get_computer_move():
    # Calculate optimal computer move using game theory

def process_round(player_role, player_move):
    # Process player and computer moves and return results

def run_simulation(total_rounds):
    # Run multiple rounds of computer vs. computer simulation
```

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

3. Core Game Engine (`startUp.py`) :

   The `GameState` class implements the fundamental game mechanics:

```python
def start(self, world_rows, world_cols, game_mode, human_player_mode):
    # Initialize the game world and parameters

def generate_random_world(self):
    # Create a random game world with difficulty levels

def hider_plays(self, i, j):
    # Process the Hider's move

def seeker_plays(self, i, j):
    # Process the Seeker's move and calculate scores

def formulate_game(self):
    # Set up the game matrix for solving

def solve_game_as_LP(self):
    # Find the optimal strategy using linear programming
```

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:

```python
def generate_random_world(self):
    level = [0] * self.world_size
    world_matrix = [[0 for _ in range(self.world_size)] for _ in range(self.world_size)]

    # Generate difficulty level for each position
    for i in range(self.world_size):
        level[i] = random.randint(0, 2)  # 0: easy, 1: medium, 2: hard

        # Set payoff values based on difficulty level
        # ...
```

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

2. Linear Programming Solution :

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

```python
def solve_game_as_LP(self):
    # Set up bounds for all variables
    bounds = [(0, 1)] * self.world_size + [(None, None)]

    A = []  # Coefficient matrix for inequality constraints
    b = []  # Right-hand side of inequality constraints
    c = []  # Objective function coefficients

    if self.human_player_mode == 0:  # Human is hider (maximize minimum)
        # ...
    else:  # Human is seeker (minimize maximum)
        # ...

    # Solve the linear program using scipy
    result = linprog(
        c=c, A_ub=A, b_ub=b, A_eq=eq_constraint, b_eq=eq_value,
        bounds=bounds, method='highs'
    )
```

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

3. Simulation Procedure :

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

```python
def run_simulation(total_rounds):
    # Initialize tracking variables

    # Run simulation for specified number of rounds
    for _ in range(total_rounds):
        # Get optimal strategies for both players
        game.formulate_game()
        result = game.solve_game_as_LP()
        computer_strategy = result.x[:-1]

        # Choose moves for both players based on optimal strategy
        player_move = np.random.choice(len(computer_strategy), p=computer_strategy)
        # ...
        computer_move = np.random.choice(len(computer_strategy), p=computer_strategy)
        # ...
```

This process:

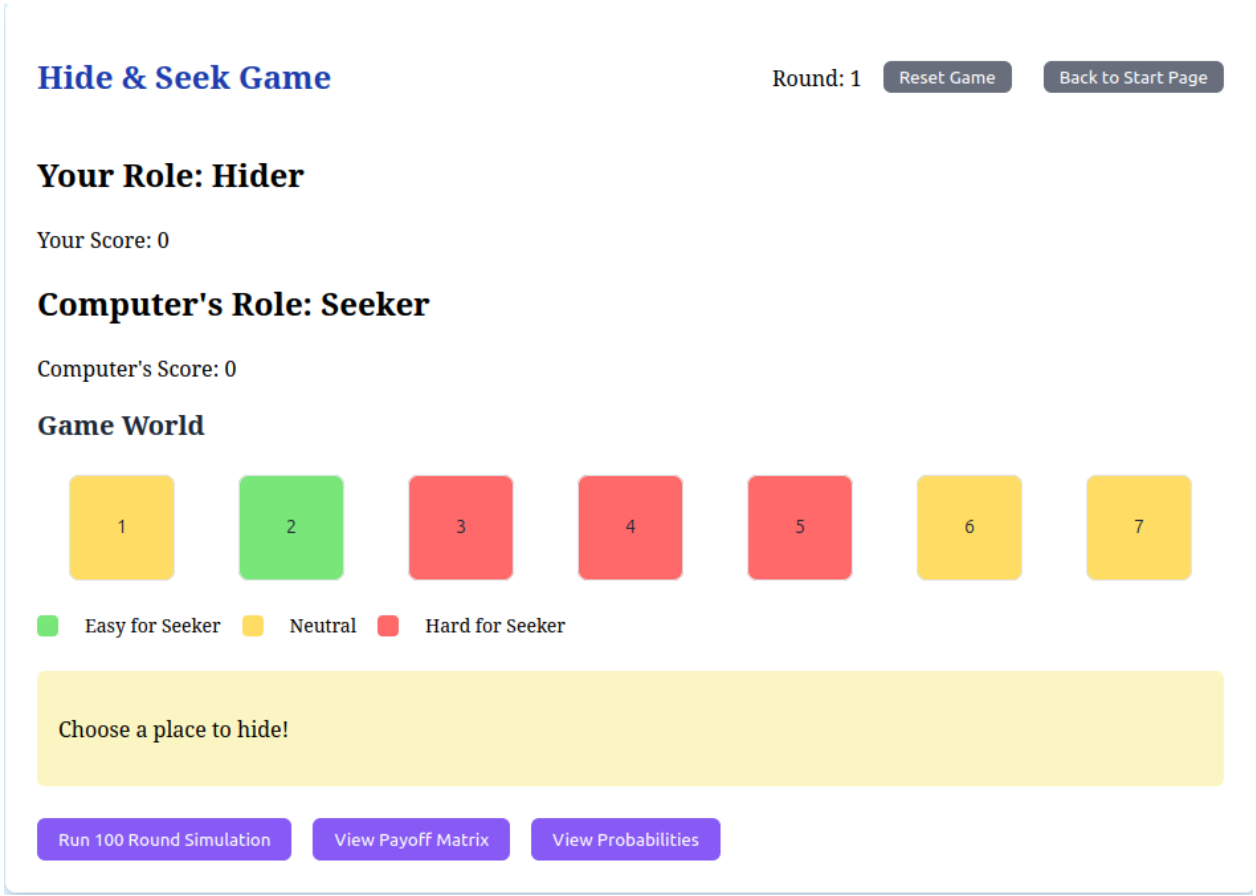
1. Calculates optimal mixed strategies using LP
2. Samples move from these probability distributions
3. Processes moves and updates scores
4. Tracks statistics across all rounds

♦ Test Case Scenarios:

- ### Test 1 :

This test for linear grid (7 cells) and Hider player , Seeker computer

➔ **Grid shape and difficulties :**



➔ Payoff matrix :

Payoff Matrix

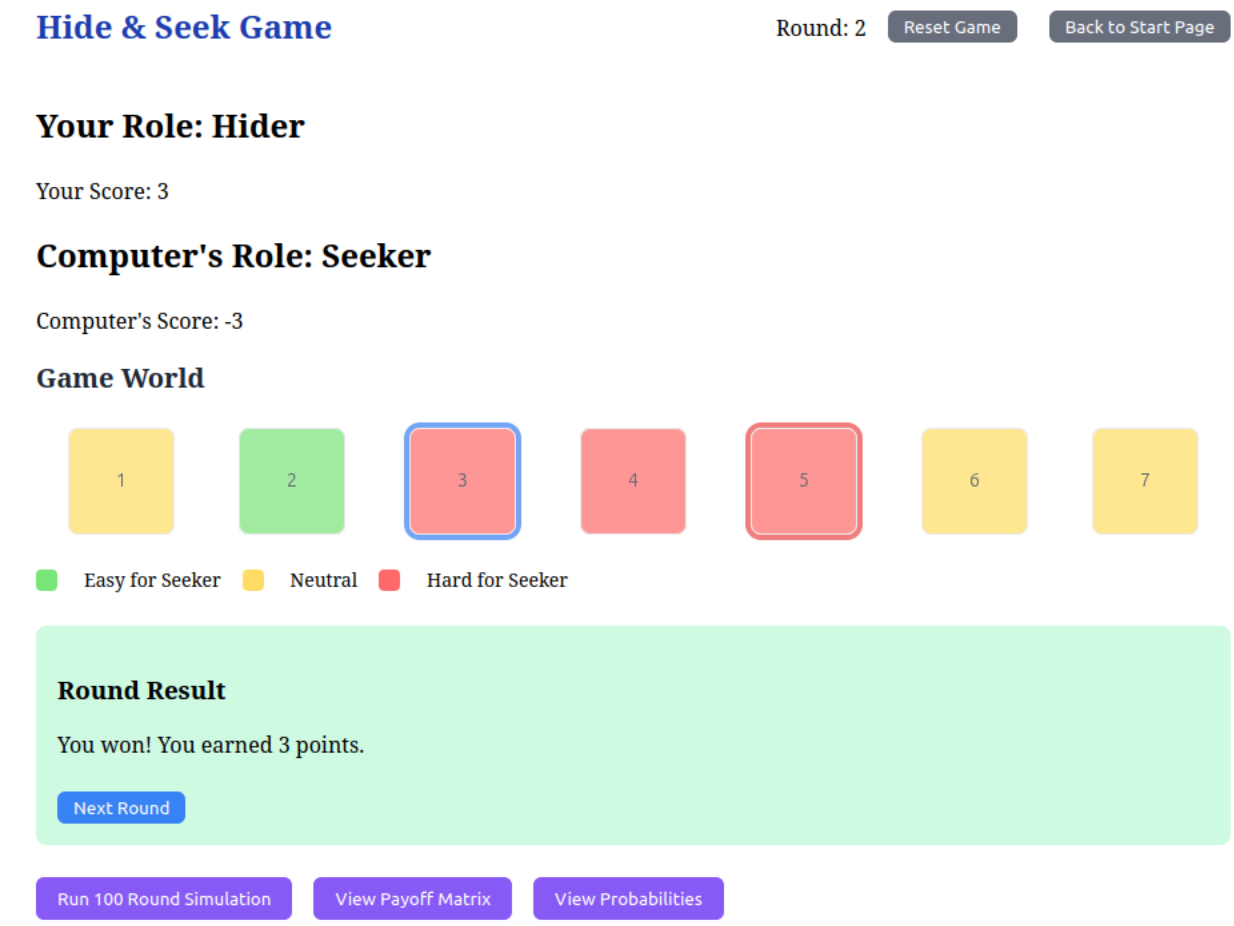
	C1	C2	C3	C4	C5	C6	C7
R1	-1	0.5	0.75	1	1	1	1
R2	1	-1	1	1.5	2	2	2
R3	0.75	0.5	-3	0.5	0.75	1	1
R4	1	0.75	0.5	-3	0.5	0.75	1
R5	1	1	0.75	0.5	-3	0.5	0.75
R6	1	1	1	0.75	0.5	-1	0.5
R7	1	1	1	1	0.75	0.5	-1

➔ Probabilities for every cell :

Computer Probabilities

P1	0.203448
P2	0.131605
P3	0.077139
P4	0.090710
P5	0.085630
P6	0.191039
P7	0.220429

➔ Some choices with scores :



Hide & Seek Game

Round: 3 Reset Game Back to Start Page

Your Role: Hider

Your Score: 11

Computer's Role: Seeker

Computer's Score: -11

Game World

Easy for Seeker Neutral Hard for Seeker

Round Result

You won! You earned 8 points.

Next Round

Run 100 Round Simulation View Payoff Matrix View Probabilities

Hide & Seek Game

Round: 4 Reset Game Back to Start Page

Your Role: Hider

Your Score: 15

Computer's Role: Seeker

Computer's Score: -15

Game World

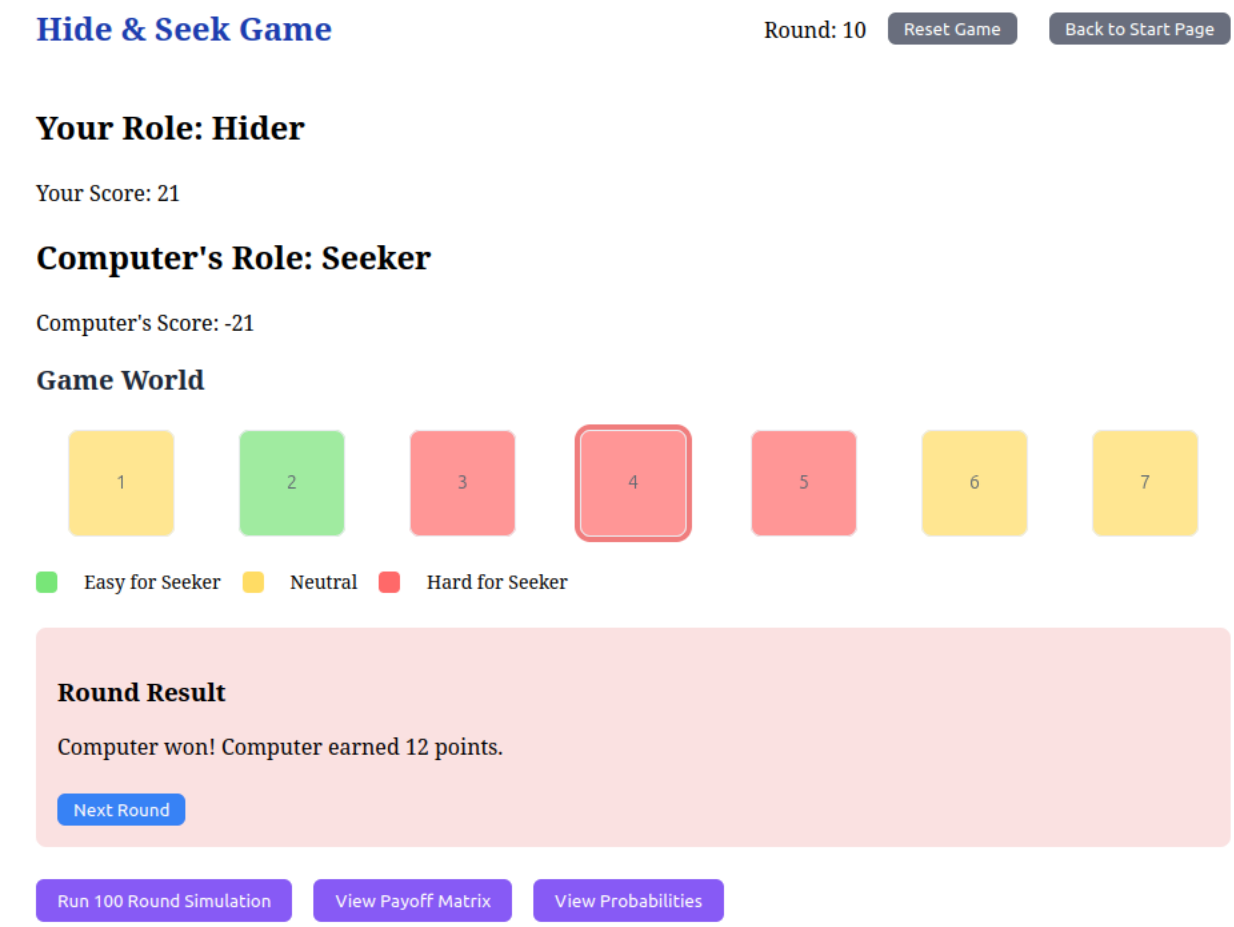
Easy for Seeker Neutral Hard for Seeker

Round Result

You won! You earned 4 points.

Next Round

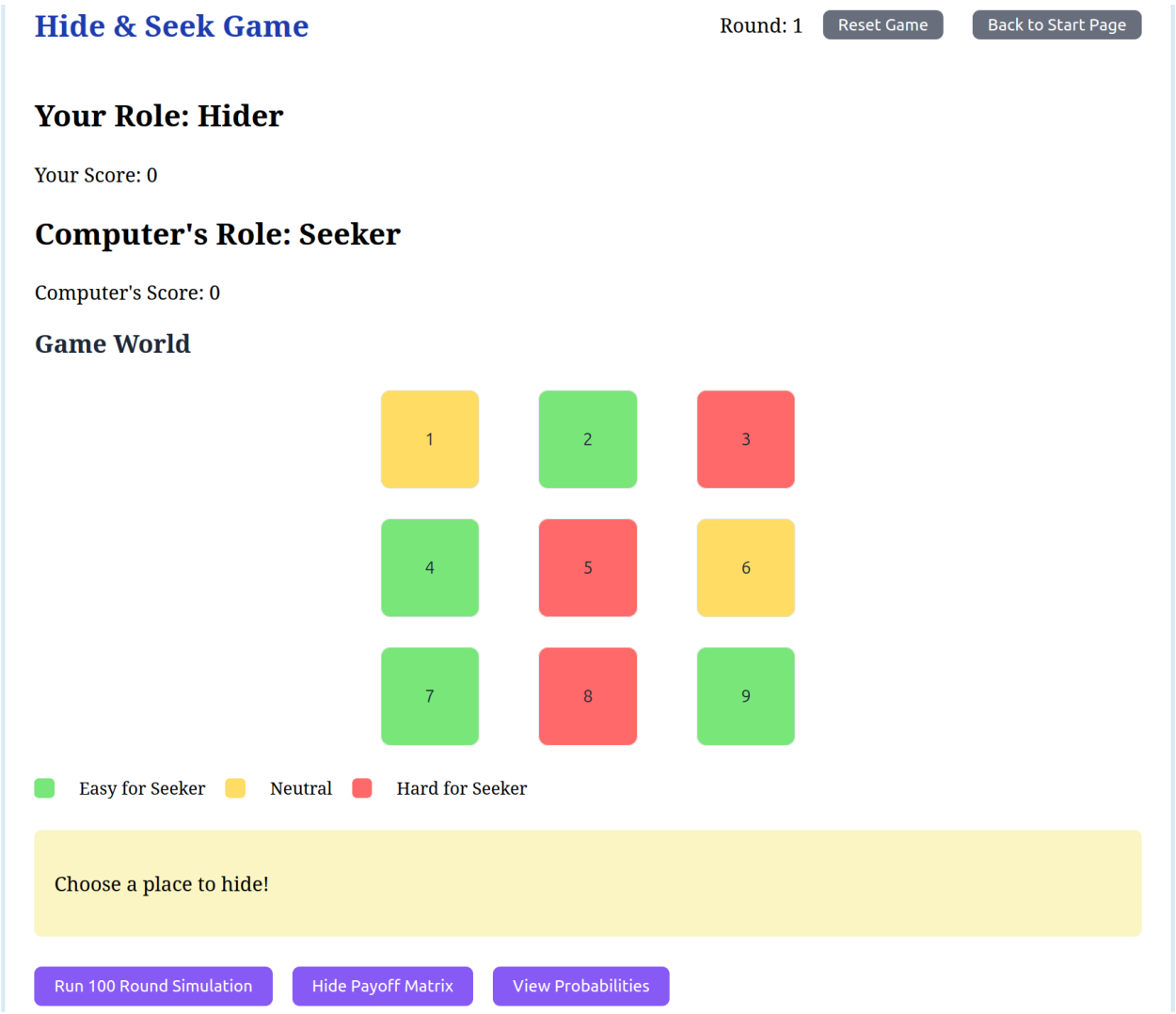
Run 100 Round Simulation View Payoff Matrix View Probabilities



- **Test 2 :**

  This test for 2D grid and Hider player , Seeker computer

  ➔ Grid shape and difficulties :



➔ Payoff matrix :

Payoff Matrix

	C1	C2	C3	C4	C5	C6	C7	C8	C9
R1	-1	0.5	0.75	0.5	0.75	1	0.75	1	1
R2	1	-1	1	1.5	1	1.5	2	1.5	2
R3	0.75	0.5	-3	1	0.75	0.5	1	1	0.75
R4	1	1.5	2	-1	1	1.5	1	1.5	2
R5	0.75	0.5	0.75	0.5	-3	0.5	0.75	0.5	0.75
R6	1	0.75	0.5	0.75	0.5	-1	1	0.75	0.5
R7	1.5	2	2	1	1.5	2	-1	1	1.5
R8	1	0.75	1	0.75	0.5	0.75	0.5	-3	0.5
R9	2	2	1.5	2	1.5	1	1.5	1	-1

➔ **Probabilities for every cell :**

Computer Probabilities

P1	0
P2	0.291667
P3	0
P4	0.208333
P5	0
P6	0
P7	0.208333
P8	0
P9	0.291667

Some choices with scores:

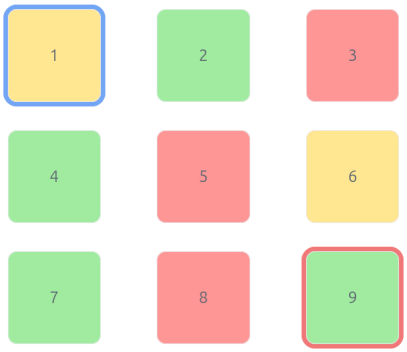
Your Role: Hider

Your Score: 4

Computer's Role: Seeker

Computer's Score: -4

Game World



Easy for Seeker Neutral Hard for Seeker

Round Result

You won! You earned 4 points.

Next Round

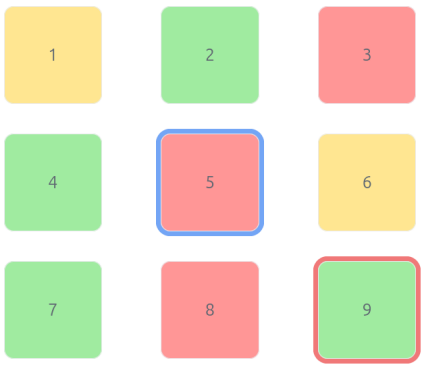
Your Role: Hider

Your Score: 7

Computer's Role: Seeker

Computer's Score: -7

Game World



Easy for Seeker Neutral Hard for Seeker

Round Result

You won! You earned 3 points.

Next Round

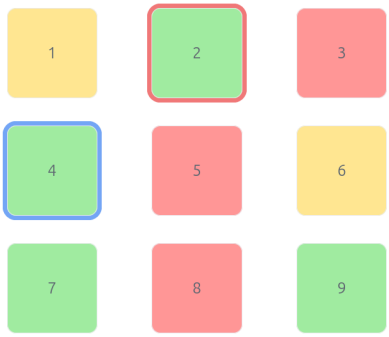
Your Role: Hider

Your Score: 13

Computer's Role: Seeker

Computer's Score: -13

Game World



Easy for Seeker Neutral Hard for Seeker

Round Result

You won! You earned 6 points.

Next Round

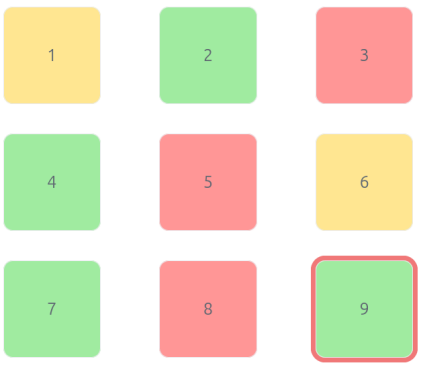
Your Role: Hider

Your Score: 9

Computer's Role: Seeker

Computer's Score: -9

Game World



Easy for Seeker Neutral Hard for Seeker

Round Result

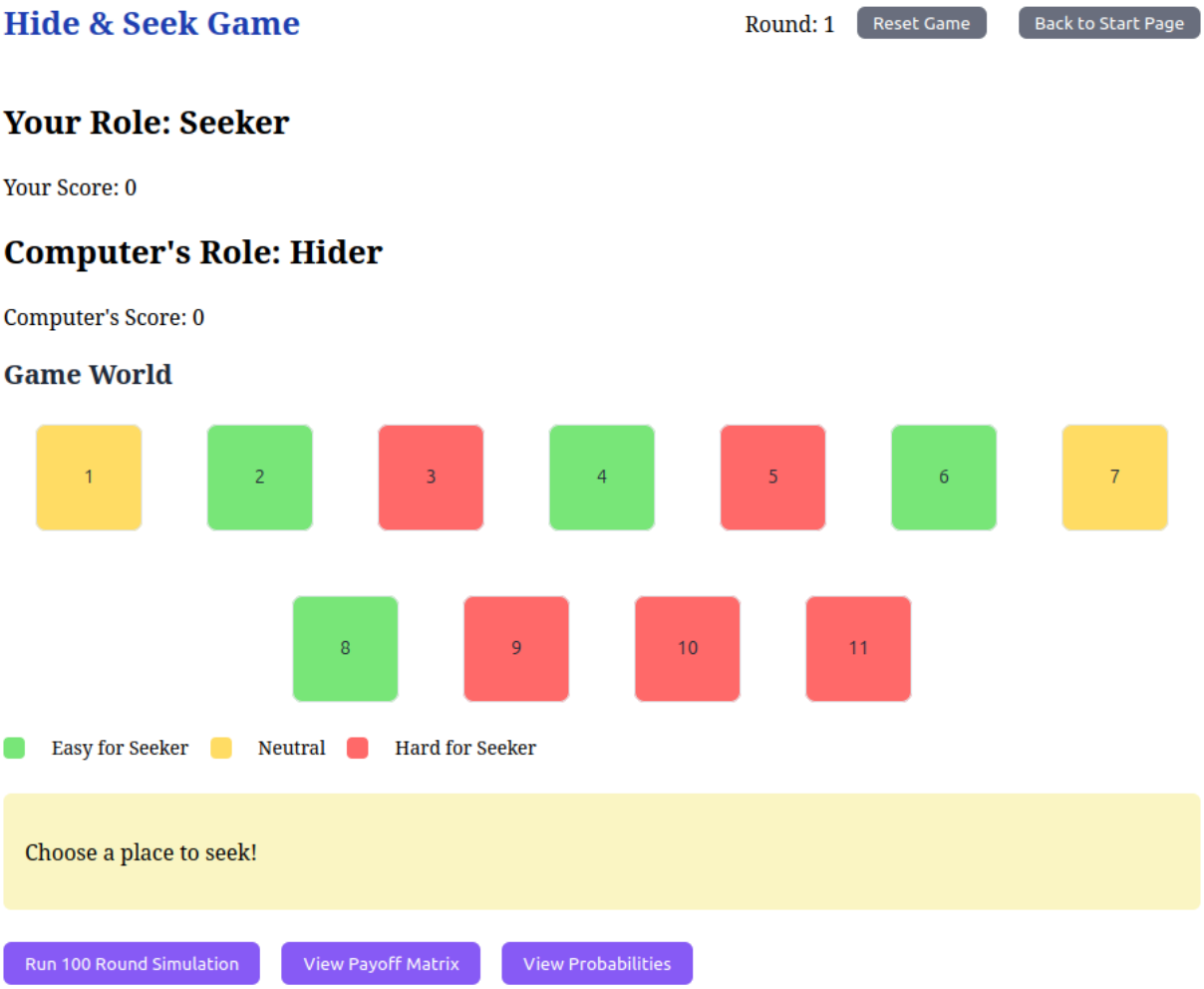
Computer won! Computer earned 4 points.

Next Round

- **Test 3 :**

  This test for linear grid (11 cells) and Seeker player , Hider computer

  ➔ **Grid shape and difficulties :**



  ➔ **Payoff Matrix :**

Payoff Matrix

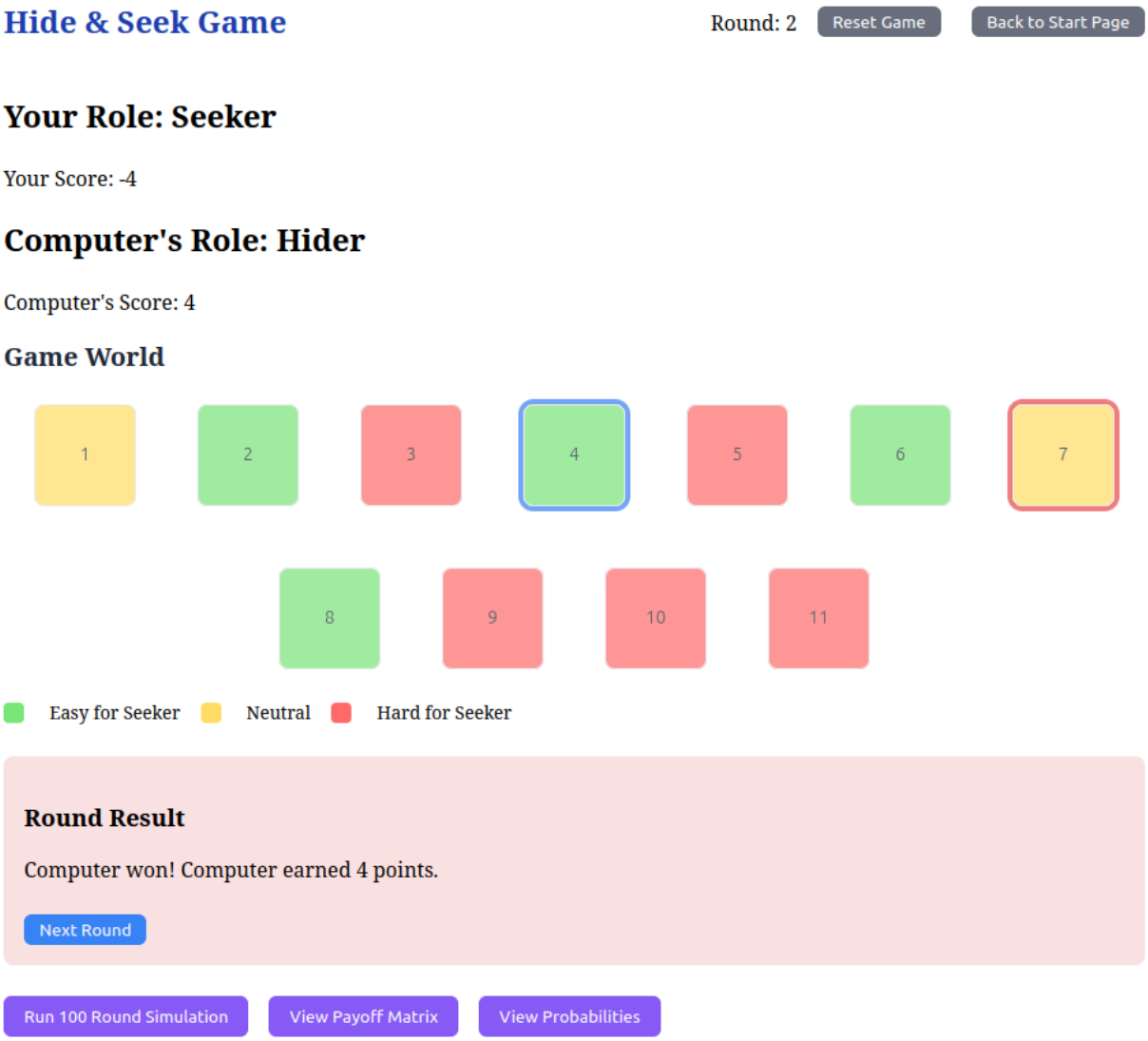
	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11
R1	-1	0.5	0.75	1	1	1	1	1	1	1	1
R2	1	-1	1	1.5	2	2	2	2	2	2	2
R3	0.75	0.5	-3	0.5	0.75	1	1	1	1	1	1
R4	2	1.5	1	-1	1	1.5	2	2	2	2	2
R5	1	1	0.75	0.5	-3	0.5	0.75	1	1	1	1
R6	2	2	2	1.5	1	-1	1	1.5	2	2	2
R7	1	1	1	1	0.75	0.5	-1	0.5	0.75	1	1
R8	2	2	2	2	2	1.5	1	-1	1	1.5	2
R9	1	1	1	1	1	1	0.75	0.5	-3	0.5	0.75
R10	1	1	1	1	1	1	1	0.75	0.5	-3	0.5
R11	1	1	1	1	1	1	1	1	0.75	0.5	-3

➔ **Probabilities :**

Computer Probabilities

P1	0.059889
P2	0
P3	0.390788
P4	0
P5	0.044161
P6	0
P7	0.390788
P8	0
P9	0.021263
P10	0.045796
P11	0.047315

➔ some choices with scores :



Hide & Seek Game

Round: 4 Reset Game Back to Start Page

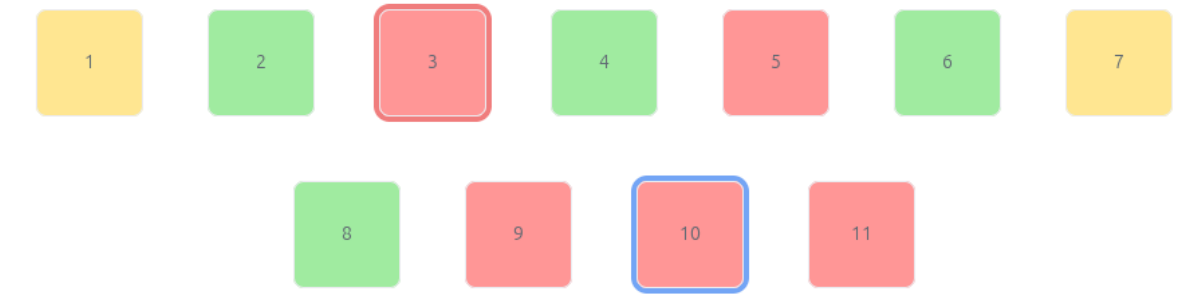
Your Role: Seeker

Your Score: -12

Computer's Role: Hider

Computer's Score: 12

Game World



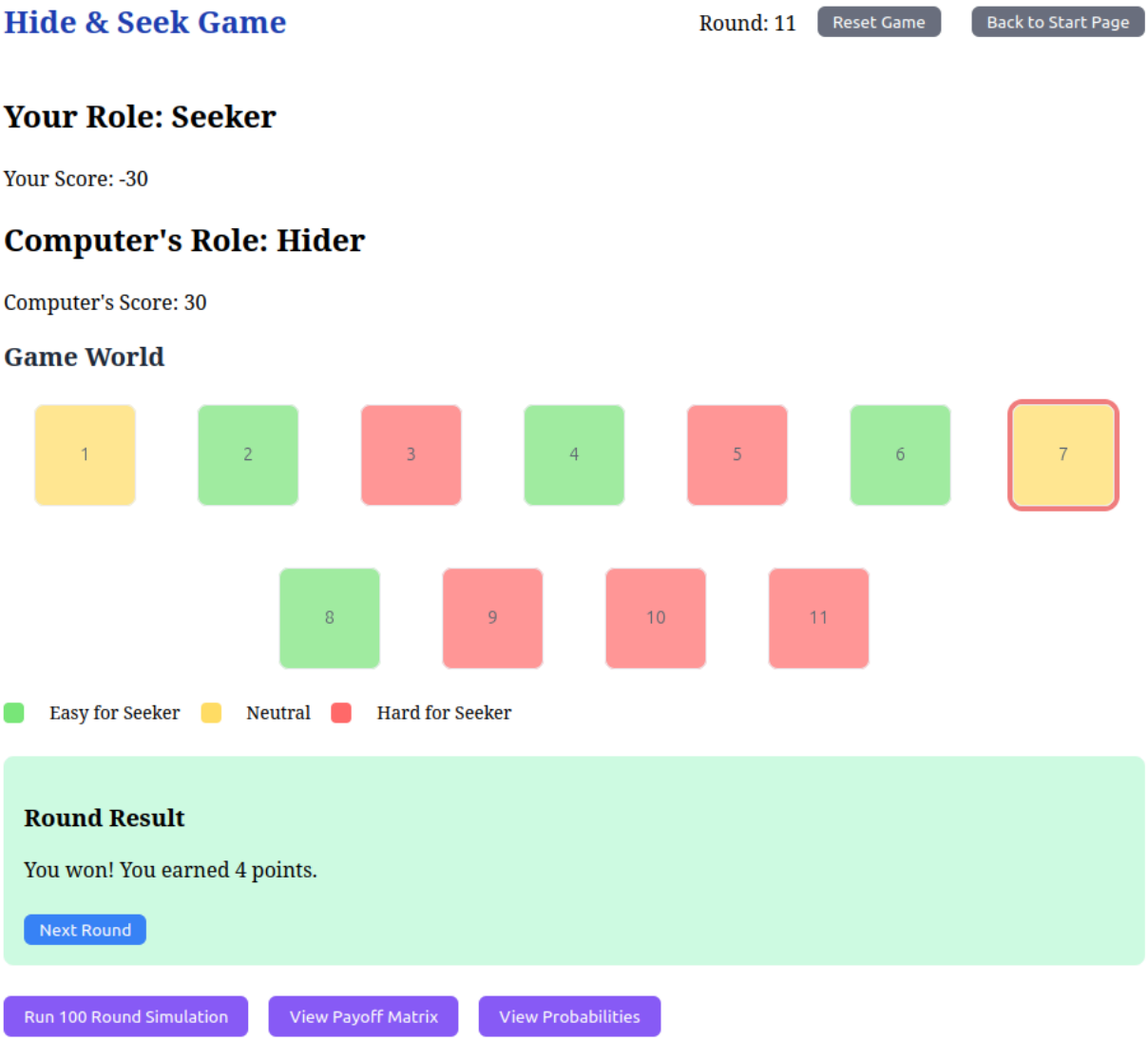
Easy for Seeker Neutral Hard for Seeker

Round Result

Computer won! Computer earned 4 points.

Next Round

Run 100 Round Simulation View Payoff Matrix View Probabilities



- **Test 4 :**

  This test for 2D grid (5x5) and Hider computer , Seeker player

  ➔ Grid shape and difficulties :

Hide & Seek Game

Round: 1 Reset Game Back to Start Page

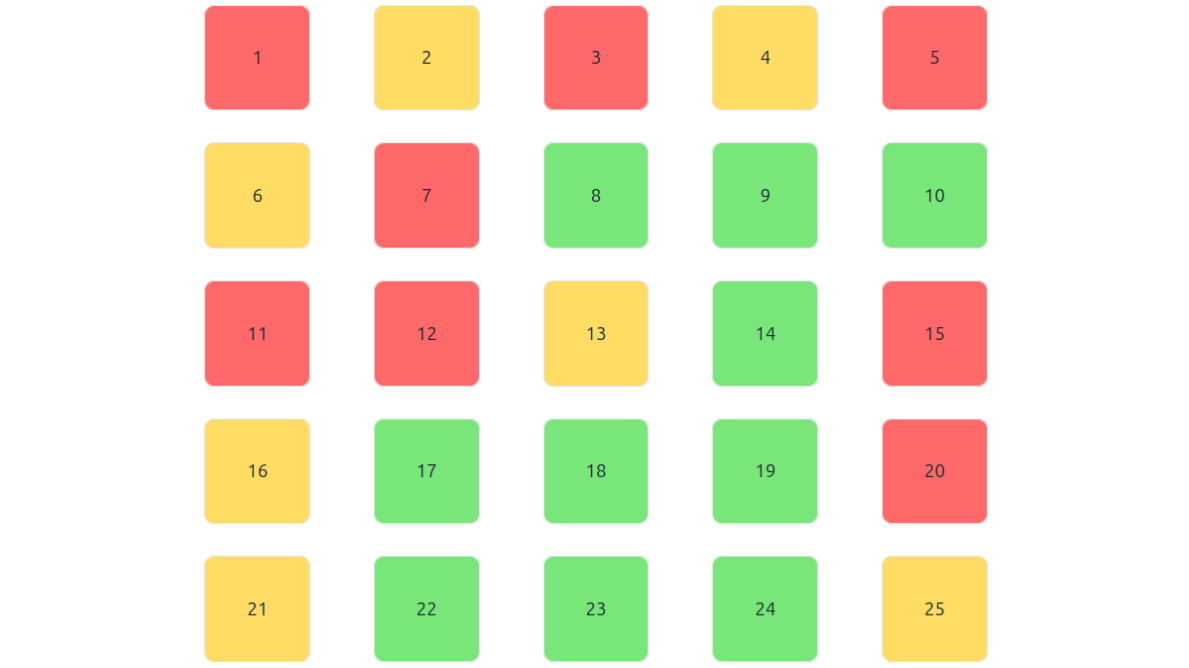
Your Role: Seeker

Your Score: 0

Computer's Role: Hider

Computer's Score: 0

Game World



Choose a place to seek!

Run 100 Round Simulation View Payoff Matrix View Probabilities

➔ **Payoff Matrix :**

C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18
0.75	1	1	0.5	0.75	1	1	1	0.75	1	1	1	1	1	1	1
0.5	0.75	1	0.75	0.5	0.75	1	1	1	0.75	1	1	1	1	1	1
-3	0.5	0.75	1	0.75	0.5	0.75	1	1	1	0.75	1	1	1	1	1
0.5	-1	0.5	1	1	0.75	0.5	0.75	1	1	1	0.75	1	1	1	1
0.75	0.5	-3	1	1	1	0.75	0.5	1	1	1	1	0.75	1	1	1
1	1	1	-1	0.5	0.75	1	1	0.5	0.75	1	1	1	0.75	1	1
0.75	1	1	0.5	-3	0.5	0.75	1	0.75	0.5	0.75	1	1	1	0.75	1
1	1.5	2	1.5	1	-1	1	1.5	2	1.5	1	1.5	2	2	2	1.5
1.5	1	1.5	2	1.5	1	-1	1	2	2	1.5	1	1.5	2	2	2
2	1.5	1	2	2	1.5	1	-1	2	2	2	1.5	1	2	2	2
1	1	1	0.5	0.75	1	1	1	-3	0.5	0.75	1	1	0.5	0.75	1
1	1	1	0.75	0.5	0.75	1	1	0.5	-3	0.5	0.75	1	0.75	0.5	0.75
0.75	1	1	1	0.75	0.5	0.75	1	0.75	0.5	-1	0.5	0.75	1	0.75	0.5
2	1.5	2	2	2	1.5	1	1.5	2	1.5	1	-1	1	2	2	1.5
1	1	0.75	1	1	1	0.75	0.5	1	1	0.75	0.5	-3	1	1	1
1	1	1	0.75	1	1	1	1	0.5	0.75	1	1	1	-1	0.5	0.75
2	2	2	2	1.5	2	2	2	1.5	1	1.5	2	2	1	-1	1
2	2	2	2	2	1.5	2	2	2	1.5	1	1.5	2	1.5	1	-1
2	2	2	2	2	2	1.5	2	2	2	1.5	1	1.5	2	1.5	1
1	1	1	1	1	1	1	0.75	1	1	1	0.75	0.5	1	1	0.75
1	1	1	1	1	1	1	1	0.75	1	1	1	1	0.5	0.75	1
2	2	2	2	2	2	2	2	2	1.5	2	2	2	1.5	1	1.5
2	2	2	2	2	2	2	2	2	2	1.5	2	2	2	1.5	1
2	2	2	2	2	2	2	2	2	2	2	1.5	2	2	2	1.5
1	1	1	1	1	1	1	1	1	1	1	1	0.75	1	1	1

➔ **Probabilities :**

Computer Probabilities

P1	0.021135
P2	0.021614
P3	0
P4	0.127504
P5	0.025565
P6	0.039829
P7	0.003374
P8	0
P9	0
P10	0
P11	0
P12	0.075153
P13	0.116103
P14	0
P15	0.064362
P16	0
P17	0
P18	0
P19	0
P20	0.157053
P21	0.194630
P22	0
P23	0
P24	0
P25	0.153679

➔ some choices :

Hide & Seek Game

Round: 2 Reset Game Back to Start Page

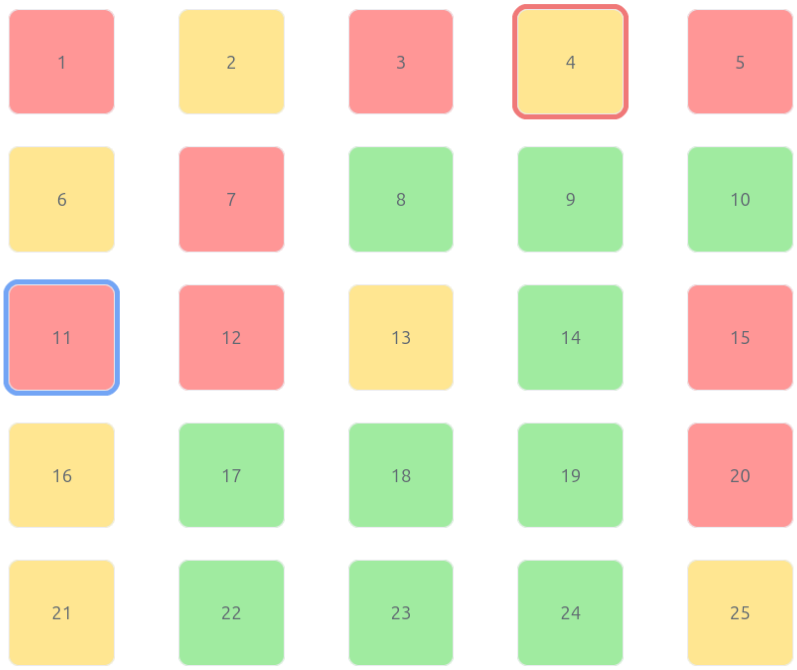
Your Role: Seeker

Your Score: -4

Computer's Role: Hider

Computer's Score: 4

Game World



Easy for Seeker Neutral Hard for Seeker

Round Result

Computer won! Computer earned 4 points.

Next Round

Hide & Seek Game

Round: 3 Reset Game Back to Start Page

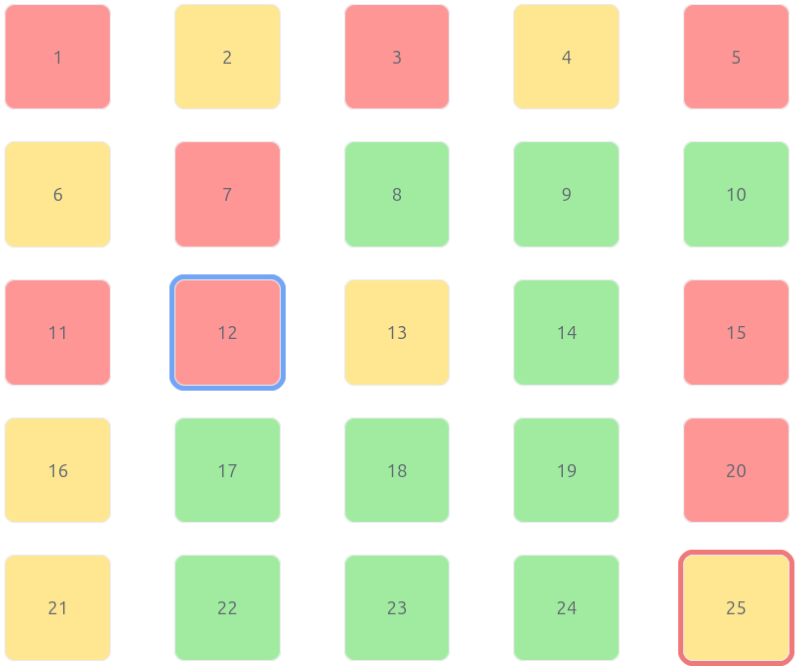
Your Role: Seeker

Your Score: -8

Computer's Role: Hider

Computer's Score: 8

Game World



Easy for Seeker Neutral Hard for Seeker

Round Result

Computer won! Computer earned 4 points.

Next Round

Hide & Seek Game

Round: 5 Reset Game Back to Start Page

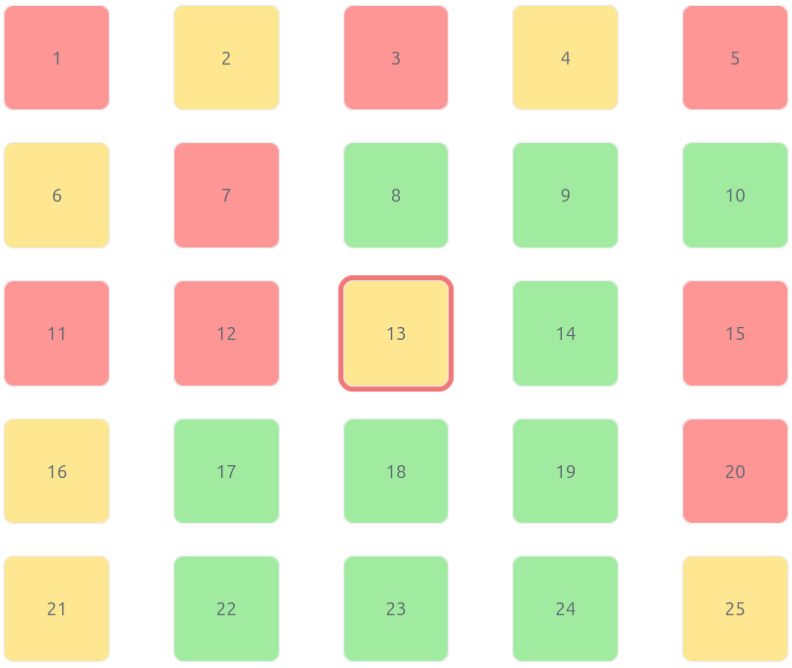
Your Role: Seeker

Your Score: -8

Computer's Role: Hider

Computer's Score: 8

Game World



Easy for Seeker Neutral Hard for Seeker

Round Result

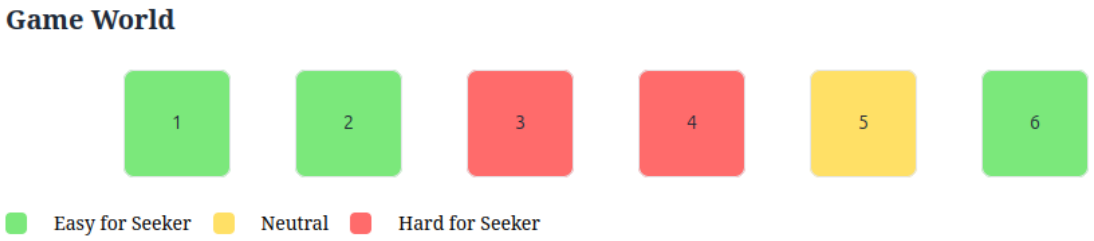
You won! You earned 4 points.

Next Round

- **Test 5 :**

This test for Simulation linear grid (6 cells) and Seeker player , Hider computer

➔ **Grid shape and difficulties :**



➔ **Payoff Matrix :**

Payoff Matrix

	C1	C2	C3	C4	C5	C6
R1	-1	1	1.5	2	2	2
R2	1	-1	1	1.5	2	2
R3	0.75	0.5	-3	0.5	0.75	1
R4	1	0.75	0.5	-3	0.5	0.75
R5	1	1	0.75	0.5	-1	0.5
R6	2	2	2	1.5	1	-1

➔ **Probabilities :**

Hider Probabilities (Simulation)

P1	0.258760
P2	0.235849
P3	0
P4	0.183288
P5	0
P6	0.322102

Seeker Probabilities (Simulation)

P1	0.088710
P2	0
P3	0.709677
P4	0
P5	0.064516
P6	0.137097

➔ results of simulation :

Simulation Results (100 Rounds)

Player Wins: 80

Player Score: 204

Computer Wins: 20

Computer Score: -204

And if you want to see Rounds details , we showed all rounds like :

Rounds Details

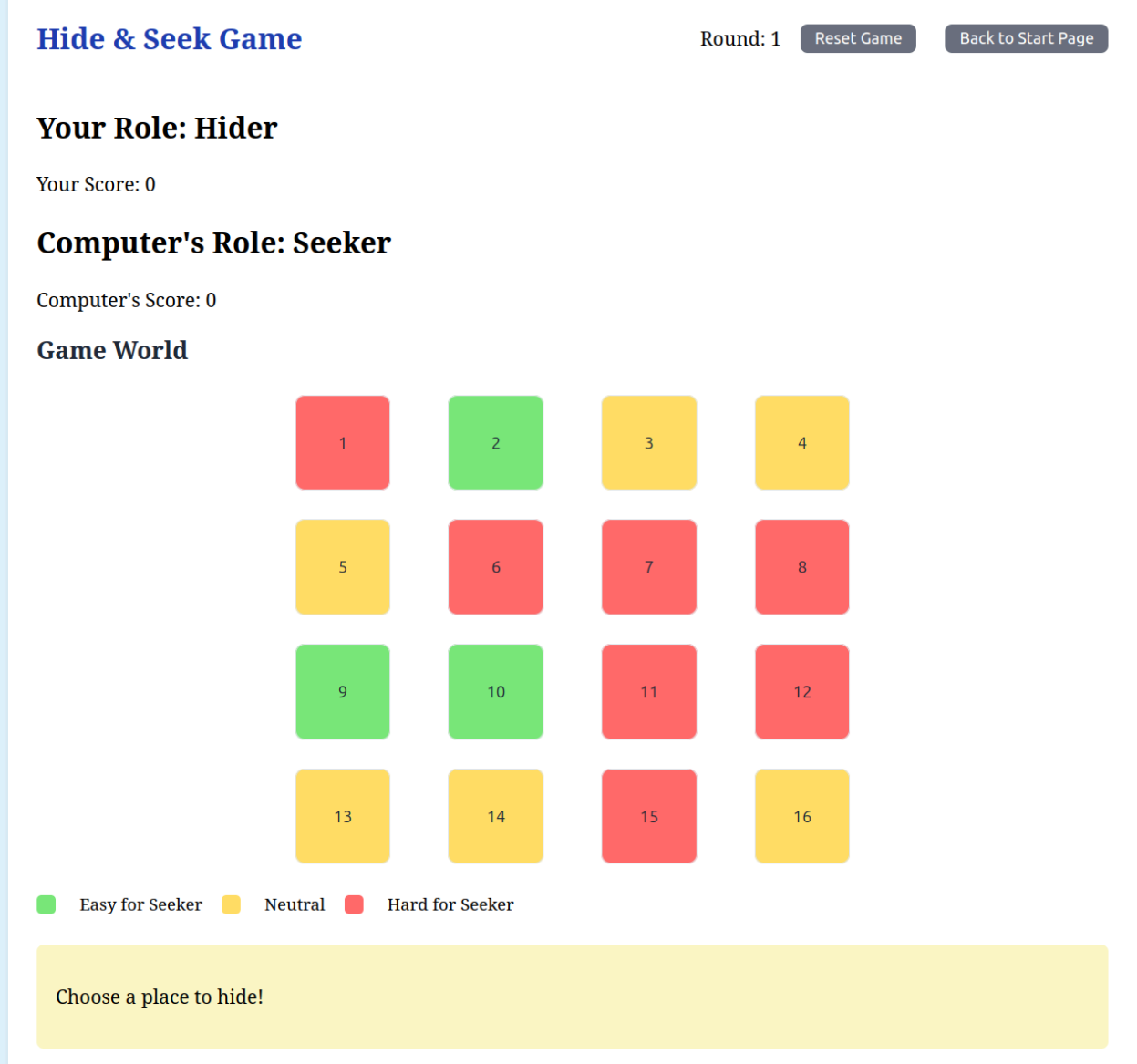
Round	Winner	Hider Move	Seeker Move	Hider Score	Seeker Score
1	player	3	2	2	-2
2	player	0	5	8	-8
3	player	0	2	6	-6
4	player	3	2	2	-2
5	player	1	2	4	-4
6	player	5	2	8	-8
7	player	0	5	8	-8
8	player	0	2	6	-6
9	player	1	2	4	-4
10	player	1	2	4	-4
11	player	1	0	4	-4
12	player	3	2	2	-2
13	player	1	2	4	-4
14	player	1	2	4	-4
15	player	1	2	4	-4
16	player	0	2	6	-6
17	player	5	2	8	-8
18	player	0	5	8	-8
19	player	3	2	2	-2
20	player	1	5	8	-8

And all rounds like this….

- **Test 6 :**

  This test for Simulation 2D grid (4x4) and Seeker computer , Hider player

  ➔ Grid shape and difficulties :



  ➔ Payoff Matrix :

Payoff Matrix

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C1
R1	-3	0.5	0.75	1	0.5	0.75	1	1	0.75	1	1	1	1	1	1
R2	1	-1	1	1.5	1.5	1	1.5	2	2	1.5	2	2	2	2	2
R3	0.75	0.5	-1	0.5	1	0.75	0.5	0.75	1	1	0.75	1	1	1	1
R4	1	0.75	0.5	-1	1	1	0.75	0.5	1	1	1	0.75	1	1	1
R5	0.5	0.75	1	1	-1	0.5	0.75	1	0.5	0.75	1	1	0.75	1	1
R6	0.75	0.5	0.75	1	0.5	-3	0.5	0.75	0.75	0.5	0.75	1	1	0.75	1
R7	1	0.75	0.5	0.75	0.75	0.5	-3	0.5	1	0.75	0.5	0.75	1	1	0.7
R8	1	1	0.75	0.5	1	0.75	0.5	-3	1	1	0.75	0.5	1	1	1
R9	1.5	2	2	2	1	1.5	2	2	-1	1	1.5	2	1	1.5	2
R10	2	1.5	2	2	1.5	1	1.5	2	1	-1	1	1.5	1.5	1	1.
R11	1	1	0.75	1	1	0.75	0.5	0.75	0.75	0.5	-3	0.5	1	0.75	0.
R12	1	1	1	0.75	1	1	0.75	0.5	1	0.75	0.5	-3	1	1	0.7
R13	1	1	1	1	0.75	1	1	1	0.5	0.75	1	1	-1	0.5	0.7
R14	1	1	1	1	1	0.75	1	1	0.75	0.5	0.75	1	0.5	-1	0.
R15	1	1	1	1	1	1	0.75	1	1	0.75	0.5	0.75	0.75	0.5	-3
R16	1	1	1	1	1	1	1	0.75	1	1	0.75	0.5	1	0.75	0.

➔ **Probabilities :**

Hider Probabilities (Simulation)

P1	0
P2	0.139632
P3	0
P4	0.191783
P5	0
P6	0
P7	0
P8	0.082821
P9	0.141047
P10	0.087407
P11	0
P12	0.063012
P13	0
P14	0
P15	0.086840

Seeker Probabilities (Simulation)

P1	0.010818
P2	0
P3	0.061352
P4	0.098745
P5	0.253227
P6	0.244576
P7	0
P8	0.018233
P9	0
P10	0
P11	0.023529
P12	0.036334
P13	0.074617
P14	0.050851
P15	0.033594
P16	0.094124

➔ result of simulation :

Simulation Results (100 Rounds)

Player Wins: 89

Player Score: 324

Computer Wins: 11

Computer Score: -324

Rounds Details

Round	Winner	Hider Move	Seeker Move	Hider Score	Seeker Score
1	player	1	5	4	-4
2	player	8	4	4	-4
3	player	1	3	6	-6
4	player	1	5	4	-4
5	player	1	13	8	-8
6	player	3	4	4	-4
7	player	1	4	6	-6
8	player	3	5	4	-4
9	player	11	4	4	-4
10	player	15	4	4	-4
11	computer	3	3	-4	4
12	player	9	5	4	-4
13	player	3	12	4	-4
14	player	1	4	6	-6
15	computer	15	15	-4	4
16	player	15	12	4	-4
17	computer	3	3	-4	4
18	player	3	13	4	-4
19	player	1	3	6	-6
20	player	15	11	2	-2
21	player	3	5	4	-4
22	computer	11	11	-12	12
23	computer	11	11	-12	12

♦ 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 :

```python
def _adjust_nearby_payoffs(self, matrix, position, base_value):
    """Helper method to adjust payoffs for nearby positions."""
    # Adjust +/-1 positions
    if position + 1 < self.world_size:
        matrix[position][position + 1] = base_value * 0.5
    if position - 1 >= 0:
        matrix[position][position - 1] = base_value * 0.5

    # Adjust +/-2 positions
    if position + 2 < self.world_size:
        matrix[position][position + 2] = base_value * 0.75
    if position - 2 >= 0:
        matrix[position][position - 2] = base_value * 0.75
```

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

2. **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.

♦ Assumptions :

1. Reset Button:

will reset the difficulty level of each cell which will change the payoff matrix causing the probabilities to change but the size of the grid has not changed.

2. Scores calculation :

we assume that the scoring unit is 4 to eliminate the fractions of proximity so:

- in hard place for seeker
  - catching a hider will bring to the seeker 4*3 = 12 points
  - failing in catching the hider will bring to him 4*1 = 4 points
- in neutral place for seeker
  - catching a hider will bring to the seeker 4*1 = 4 points
  - failing in catching the hider will bring to him 4*1 = 4 points
- in easy place for seeker
  - catching a hider will bring to the seeker 4*1 = 4 points
  - failing in catching the hider will bring to him 4*2 = 8 points

3. Back to Start page :

this will make everything null and return you to the welcome page to start a new game with a new grid.

♦ 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.