**Neuroevolution for Automated Game Playing**

Neuroevolution is a machine learning technique that combines artificial neural networks with evolutionary algorithms to enable automated learning in complex environments like computer games. Here's a detailed explanation of how it works specifically for game playing:

Core Concept

Neuroevolution evolves neural network architectures and/or their weights through processes inspired by biological evolution:

1. A population of neural networks (agents) is created
2. Each network plays the game and receives a fitness score
3. The best performers are selected to "reproduce"
4. Their offspring undergo mutations/crossovers
5. The process repeats across generations

Key Components for Game Playing

1. State Representation

* The game state is encoded as inputs to the neural network (e.g., pixel values, game variables, or object positions)
* Common approaches: raw pixels, feature extraction, or game memory reading

2. Action Representation

* Network outputs correspond to possible game actions (button presses, mouse movements)
* Can be discrete (press A/B) or continuous (analog stick position)

3. Fitness Function

* Measures how well the agent performs (score, survival time, progress)
* Carefully designed to encourage desired behaviors

Popular Neuroevolution Methods for Games

1. NEAT (NeuroEvolution of Augmenting Topologies)

* Evolves both weights and network structure
* Starts with simple networks that grow more complex
* Particularly effective for games with complex state spaces

2. HyperNEAT

* Extends NEAT to exploit geometric patterns in the game state
* Good for games with spatial relationships (like many 2D games)

3. CMA-ES (Covariance Matrix Adaptation Evolution Strategy)

* Optimizes network weights for fixed architectures
* Efficient for high-dimensional parameter spaces

Advantages for Game Playing

1. **No need for labeled data** - Learns directly from interaction
2. **Reward shaping flexibility** - Can design complex fitness functions
3. **Discover novel strategies** - May find human-overlooked tactics
4. **Adaptability** - Can adjust to game changes or new levels

Challenges

1. **Computational cost** - Requires many game simulations
2. **Credit assignment** - Hard to determine which actions led to success
3. **Local optima** - May get stuck on suboptimal strategies

Real-World Examples

* OpenAI used evolution strategies to train agents for Dota 2
* DeepMind's early work used neuroevolution for simple games
* Various Atari game playing agents have been developed with neuroevolution

Neuroevolution remains a powerful approach for game AI, especially when reinforcement learning might be too data-intensive or when novel solutions are desired beyond human-like play styles.

**PART 1**

**"Space Miner"**

A game where the player controls a spaceship that must collect minerals while avoiding asteroids. The ship has limited fuel and must balance mining with survival. We will use **NEAT (NeuroEvolution of Augmenting Topologies)** to train an AI to play the game automatically.

**How It Works**

1. **Game Mechanics**
   * The spaceship must collect minerals while avoiding asteroids.
   * Mining refuels the ship, but fuel depletes over time.
2. **Neuroevolution AI**
   * Uses NEAT to evolve neural networks.
   * Inputs: Distance/angle to minerals, distance to asteroids, fuel level. This is the preliminary suggestion, you should **change** this.
   * Outputs: Steering, thrust, and mining actions.
   * Fitness function rewards both survival time and minerals collected.
3. **Training Process**
   * Runs for 50 or more generations.
   * The best-performing networks will learn to navigate, mine efficiently, and avoid obstacles.

**Your TASK**

Use NEAT to train the Game agent. The basic code is provided to you including the NEAT training code. The first code miner.ph is the game code that has the player as a user. The second code miner\_neat.py is with NEAT agent learning to play the game. However, you need to modify miner\_neat.py as it is not competently train. You need to change the Inputs to the neural network so that it can learn to play the game well.

You can save the agent with this:

A white background with black and blue text

AI-generated content may be incorrect.You can re-load the agent with this:

A computer screen shot of a computer code

AI-generated content may be incorrect.

We will run this as a competition by ranking all the agents. We will hardcode the environment and you run your agent in it. Your score will be given by your ranking based on how many minerals and how long the agent stays alive. The best agent will get 100% and the rest will be appropriately assigned.