**Detailed Comparison of Old Code vs. Refined Code**

**Agent Training Code**

**Old Code**

* **Device Handling**: Explicitly uses .cuda() for GPU without fallback to CPU.
* **Persistence**:
  + High score not persistently saved.
  + Model files are saved without descriptive filenames.
* **Training Logic**:
  + Trains indefinitely without a stopping condition.
  + Does not track recent performance trends (e.g., scores over the last few games).
* **Action Selection**:
  + Utilizes random moves for exploration but does not dynamically adapt randomness.
* **Memory Management**:
  + Memory is sampled but not optimized for limited resources like MAX\_MEMORY.
* **Feedback and Logging**:
  + Limited feedback to users; lacks detailed progress information.

**Refined Code**

**Key Features:**

1. **Device Handling**:
   * Uses torch.device for dynamic handling of GPU or CPU.
   * All tensors and models are explicitly moved to the appropriate device.
2. **Persistence**:
   * High scores are saved to a record.txt file.
   * Introduces methods save\_record and load\_record for high score management.
   * Models are saved only when a new record is achieved, with filenames containing epoch and score details.
3. **Training Logic**:
   * Stops training after 1000 games, avoiding indefinite execution.
   * Tracks the best score of the last 20 games to analyze recent performance trends.
4. **Action Selection**:
   * Balances exploration and exploitation using a dynamic epsilon value based on the number of games played.
5. **Memory Management**:
   * Implements deque with MAX\_MEMORY for efficient memory management.
   * Uses random sampling for mini-batches, ensuring better training diversity.
6. **Feedback and Logging**:
   * Logs progress after every game and provides detailed updates every 20 games.
   * Displays whether a new record is achieved or not, with detailed messages for model saving.

**Model Code**

**Old Code (Simplified Overview)**

* **Save and Load Operations**:
  + Only save method exists.
  + Models are saved without versioning or directory management.
  + No dedicated load function; models need to be manually loaded in scripts.
* **Training**:
  + Limited abstraction for training steps.
  + Tensors are not consistently moved to GPU or CPU.

**Refined Code**

**Key Features:**

1. **Save and Load Operations**:
   * Introduces a load method for seamless reusability in testing or further training.
   * Saves models in a dedicated folder (./saved\_models).
   * Includes versioning in filenames (epoch and score).
2. **Training**:
   * All tensor operations are dynamically adjusted for GPU/CPU.
   * Loss function (MSELoss) and optimizer (Adam) are well-integrated into the training logic.
3. **Error Handling**:
   * Checks if the specified model file exists before attempting to load, preventing runtime errors.

**Test Script**

**Old Code**

* No standalone testing script exists.
* Any testing requires manual modifications to the training code, making the workflow cumbersome.

**Refined Code**

**Key Features:**

1. **Dedicated Script**:
   * A separate load\_test.py is introduced for testing models.
   * Demonstrates how to load a model and interact with the game environment.
   * Allows testing with dummy states or real game states.
2. **Dynamic State Management**:
   * Uses torch.tensor to preprocess states for model prediction.
   * Ensures compatibility with both CPU and GPU for testing.

**Comparison Table**

| **Feature** | **Old Code** | **Refined Code** |
| --- | --- | --- |
| **Device Handling** | .cuda() hardcoded, lacks fallback. | torch.device dynamically selects GPU or CPU. All tensors/models explicitly moved to the correct device. |
| **Persistence** | No high score persistence. | High score saved to record.txt. Models saved only when breaking records, with versioned filenames. |
| **Training Termination** | No stopping condition. | Stops training after 1000 games, making experiments reproducible. |
| **Score Analysis** | Tracks only the best overall score. | Tracks scores of the last 20 games for performance trend analysis. |
| **Action Selection** | Fixed exploration logic. | Dynamically adjusts exploration (epsilon) based on the number of games played. |
| **Memory Management** | Fixed-size memory; lacks sampling diversity. | Uses deque for efficient memory management. Randomly samples mini-batches for improved training. |
| **Feedback & Logging** | Minimal logging and progress updates. | Detailed progress updates after every game. Tracks new records and provides clear messages for model saving and evaluation. |
| **Save/Load Operations** | Save only; no structured file handling. | Save and load methods in Linear\_QNet. Models stored in a dedicated directory with structured filenames. |
| **Testing** | No dedicated testing script. | Introduces load\_test.py for standalone testing. |
| **Error Handling** | Assumes files exist without checks. | Checks for file existence before loading scores or models. |

**Specific Benefits of Refined Code**

1. **Efficiency**:
   * Dynamically handles device selection, ensuring efficient use of available resources.
   * Limits training to 1000 games, saving computational time and energy.
2. **Robustness**:
   * Introduces error handling for file operations.
   * Tracks recent performance trends (last 20 games), avoiding reliance on outdated metrics.
3. **Reusability**:
   * Includes dedicated save/load functions for models and high scores.
   * Simplifies testing with a standalone load\_test.py script.
4. **User Experience**:
   * Enhanced feedback and logging keep users informed of progress and performance milestones.
5. **Maintainability**:
   * Clear separation of concerns (training, saving, testing).
   * Introduced modular components (e.g., save\_record, load\_record) for reuse in other projects.