Maxwell Rehm, Frank Xu, Junyu Li, Minghe Fu

Minecraft Farming RL Project

1. Main idea behind your project

Goal: Automate farming

- Farming is a **boring, repetitive task** — repetitive tasks are excellent candidates for machine learning.

Why farming automation matters:

- **Food is a critical resource** — hunger depletes constantly, making food consumption unavoidable.
- **Manual farming is inefficient:**
    - Users often forget to check crops, leading to low output.
    - To compensate, players build **massive farms**, wasting valuable space.

Proposed solution:

- Build an **agent** that **automatically harvests and replants crops** as soon as they are ready.
- **Result:**
    - Small farms can achieve **high crop output** without needing large amounts of space.

2. Input/output your solution/algorithm/program will take/use

Input Setup:

- A **farm level** with:
    - **Fence boundaries** to prevent the agent from walking off.
    - **Dirt blocks** for tilling.
    - **Water nearby** to keep dirt tilled (optionally with **lily pads** to prevent the agent from falling in).
    - **Constant sunlight** for uninterrupted crop growth.
- **Agent starting equipment:**
    - Seeds
    - Hoe

Output Metrics:

- **Primary goal:**
  - **Count total wheat (crop) items collected** within a set time window.
  - In a fixed-size farm, the agent should **learn to harvest mature crops as soon as possible**.
- **Secondary goal:**
  - **Minimize seeds collected** (i.e., minimize premature harvesting).
  - The agent should **learn to replant quickly** after harvesting a mature crop.

Reward weighting:

- **Prioritize** maximizing wheat collection over minimizing seed collection.
- Different output metrics can be assigned **different reward weights** to guide learning.

3. Basic approach, method/algorithm/technique you plan to use

- **Most likely use RL** (found a PyTorch implementation here).

Set up requirements:

- Successful local Malmo installation. (finished by max 4/26 1:41pm)
- PyTorch interfaces well with CUDA toolkit for fast model training on GPU.

Initial agent behavior:

- Agent needs to use a hoe on every dirt block initially.
- Later, agent must recognize and till only *untilled* dirt blocks.

Crop harvesting strategy:

- **Two options:**
  - Traverse the entire area manually.
  - Apply a CNN vision network to check if a crop should be harvested.
- **Details needed:**
  - Confirm if Malmo's world state provides wheat *growth status* at each coordinate.
  - If not, use vision techniques to detect wheat maturity.
  - Wheat changes color from **green to brown** when matured — we can leverage this for identification.

Why RL is a good fit:

- **Simple action space:**
  - Clicking a crop destroys it.
  - If mature → yields wheat + seeds.
  - If immature → only yields seeds.
- **Straightforward reward calculation:**
  - After harvesting, count number of seeds or wheat collected.
  - Reward for collecting wheat; penalty for collecting only seeds (harvesting too early).
- **Reward-maximizing behavior suits the task:**
  - RL tends to maximize *current* rewards.
  - In this case, early harvesting after maturity maximizes output over time.

4. Evaluation plan

Tracking performance:

- We can easily track **how much wheat/crops** are collected.
- But **how do we interpret the numbers?**
  ➔ **We need to establish baselines** for comparison.

Baseline options:

- **Option 1: Random choice**
  - Implement a bot that makes random decisions.
  - Measure how many crops it collects and compare it to our trained agent.
- **Option 2: Wait and harvest**
  - Wait a fixed time (e.g., X minutes) for all crops to mature.
  - Then harvest everything at once, repeating the process.
  - Emulates a human player's harvesting behavior.
- **Option 3: Hardcoded bot**
  - Create a bot that follows a **systematic, hardcoded strategy** (e.g., moving in circles, checking each block).
  - This bot would be **hardest to beat**, but **won't generalize** well to farms of different layouts.

Additional evaluation:

- **Gameplay recordings** could provide **qualitative analysis** — helping us better understand what our bot is doing and where it could improve.

5. Project Milestones

End of Week 5:

- Finish Malmo local installation (ideally on Max's computer for model training).
- Complete project outline.
- Complete Assignment 1 individually.

End of Week 6:

- Finalize project approach details:
    - Can we leverage internal Malmo states?
    - Or will we need computer vision techniques?
- Hardcode one or more baseline bots to get familiar with Malmo Python API.
- Start reading about integrating PyTorch RL into our codebase.
- Research previous projects:
    - What libraries are commonly used with Malmo?
    - What approaches have worked (or failed) for others?
- Complete Assignment 2 individually

End of Week 7:

- Develop a **Minimal Viable Product (MVP)** using PyTorch RL.
    - Focus on getting the agent to act; fine-tuning good behaviors can come later.

End of Week 8:

- Implement detection of **tilled dirt**.
- Build the **farm map** and **farming area** environment.

End of Week 9:

- Apply RL inside the Minecraft farm environment.
    - Use the **reward and penalty system** we designed earlier.
- **Compare** agent results against baseline models.

End of Week 10:

- Prepare **final presentation**.
- Record **demonstration videos**.