Tetris AI Project Report

1. Introduction

The goal of this project was to take an existing JavaScript version of Tetris, fix its bugs, polish the gameplay, and then build two AI bots: a Greedy Heuristic Agent and a Beam Search Agent to play it automatically.

The process went through a few main steps:

1. Cleaning up and tweaking the base code
2. Adding a few quality-of-life gameplay features
3. Implementing the heuristic-based “greedy” AI
4. Implementing a more advanced beam search AI
5. Testing everything, breaking it a few times, and slowly improving it

---

2. Tweaking the Base Code

Before doing anything AI-related, I had to make sure the game itself worked properly. Two main issues made the game feel clunky and inconsistent and VERY annoying: the hovering glitch and the rotation glitch.

Hovering Glitch

When a piece landed and you were still moving or rotating it, it would sometimes just hover for a few frames instead of locking into place. It made the game feel buggy and unpredictable.

To fix it, I changed the `move()` and `rotate()` functions so that if a piece is right above the stack and can’t go any lower, it immediately locks into position. This completely removed the “hover” effect.

Rotation Glitch

Pieces that touched the side walls couldn’t rotate at all — the game would just reject the move.

To solve that, I added a simple wall-kick system. Now, when a rotation would normally be invalid, the piece automatically shifts one or two spaces to the left or right until it finds a valid position. This made the gameplay feel much smoother.

---

3. Adding Quality of Life Features

Once the core mechanics were fixed, I wanted to make the game feel more polished and enjoyable — both for human players and for the AI.

Hard Drop

Pressing **Spacebar** now instantly drops the piece to the bottom. It's a small change, but it makes the game feel way faster and less tedious, since you don't have to hold the down arrow anymore.

Ghost Piece

I added a faint "ghost" projection of the current piece, showing exactly where it would land if dropped. It helps a lot when planning moves, especially for the AI's visualization and debugging.

Level and Speed Curve

The game now tracks how many lines you've cleared. Every 10 lines, the level goes up and the pieces start falling faster. This keeps the gameplay interesting and gradually increases difficulty, just like in traditional Tetris.

---

4. The Heuristic (Greedy) Agent

After the game was stable, I started working on the first AI bot, the simpler Greedy Heuristic Agent.

This agent looks at every possible move for the current piece, simulates what the board would look like after placing it, and gives each option a score. The move with the best (lowest) score is chosen.

How It Evaluates Moves

The scoring system (or "heuristic") measures how good or bad a board looks using a few factors:

- **Total Height:** lower is better.

- **Holes:** empty spaces under blocks are heavily penalized.
- **Height irregularities:** big height differences between columns are bad.
- **Wells:** Deep gaps between tall stacks are also bad.
- **Lines Cleared:** rewarded heavily.

The final score is a weighted combination of all these metrics.

Behavior

The greedy agent is fast and makes pretty reasonable decisions, but it's also short-sighted. It can't see that a move might cause trouble a few turns later. For example, it might create a tall tower just to avoid a small hole right now, which is a bit dumb, to be honest. But that's why we introduce the way cooler and complicated Beam Search!

5. The Beam Search Agent

Once the greedy agent was working, I started on the beam search agent — an AI that could actually *plan ahead*.

The idea behind beam search is simple: instead of only looking at the current piece like the greedy agent, it looks a couple of moves into the future. It tries to figure out how the **next piece** will fit into the board after the current move, and picks the option that leads to the best *overall* situation.

How It Works

Here's what the beam search agent does each turn:

1. It simulates all possible moves for the current piece.
2. It keeps only the best few outcomes (the "beam").
3. For each of those boards, it then simulates all possible moves for the next piece.
4. After evaluating all those outcomes, it picks the path that leads to the lowest final score.

So while the greedy agent is making choices in the moment, the beam search agent is playing *strategically*. It chooses not-so-perfect moves this turn, so it has a better board to play with in the next steps.

Intelligent Hold Feature

I also made the beam agent use the "Hold" feature intelligently.
It basically runs two tests each turn:

- One where it plays normally with the current piece.
- One where it swaps the current piece with the held one and plays that instead.

If the held piece leads to a better outcome, it swaps. Otherwise, it continues normally.
This gives the AI an extra layer of decision-making, something that even skilled human players use constantly.

Behavior

The difference between the greedy and beam agents is immediately noticeable. The beam search bot builds cleaner, flatter stacks and can actually *recover* from awkward boards instead of digging itself into a hole. It's slower to compute, but it plays much smarter.

---

6. Testing, Tuning, and Fixing Things (Again)

Once both AIs were working, I ran a lot of tests. Of course, basically everything broke.

The "Mountain Problem"

At first, both agents tended to build tall towers on one side of the board. They did this because the heuristic punished *holes* so harshly that the AI would rather stack high than risk leaving a gap.

To fix that, I rebalanced the weights in the `evaluateBoard()` function. I reduced the penalty for holes and added a stronger penalty for deep gaps that usually form beside those towers. This made the AI much more stable and less likely to "mountain" itself to death.

AI Freezing / Crashing

There were also a few weird bugs where the AI would stop moving completely:

- In one case, my simulation function wasn't properly clearing the top row, which corrupted the AI's internal board state.
- Another time, when the beam search was looking at too many states at once, it became too slow for real-time gameplay and just stopped working.

The final fix was to **pause the game's natural gravity** while the AI is thinking, then let it execute its chosen move instantly. That way, it never "freezes" mid-game; it just makes its decision, drops the piece, and moves on.

---

7. Results and Conclusions

After a lot of testing and tweaking, the project turned into a surprisingly fun and educational experiment.

The Greedy Agent is simple but effective: it plays decently and reacts fast, but it can't think ahead. The Beam Search Agent, on the other hand, plays like an actual Tetris strategist. It plans, holds pieces intelligently, and survives much longer.

I added a small toggle system too:

- Press **A** to enable or disable the AI.
- Press **B** to switch between the Greedy and Beam Search agents.

That made it easy to compare both bots side by side in real time, and the difference is clear. The beam search agent consistently builds flatter, safer boards and can recover from mistakes that would completely trap the greedy bot.

Overall, this project started as a debugging and cleanup exercise but ended up becoming a full AI showcase. It showed how much difference smarter heuristics and multi-step planning can make in a concept as simple as a very simplistic game of Tetris. =)