Baby Tetris First Part Report

ERGIN Seçkin Yağmur and GROUSSON Dylan

November 28, 2025

1 Question 1: Discounted MDP For Player

1.1 Implementation

1.1.1 Main Architecture

Based on the our design choices from Section 1, we decided to use an object oriented architecture (C++) for our implementation.

For our structure we have declared 7 classes:

1. *`Point`*: classical point (x,y) class used for grid coordinates
2. *`Field`*: the class that contain a boolean grid s.t. grid[i][j] = 1 if the corresponding cell is fill in the field
3. *`Tromino`*: the class that correspond to the baby Tetris pieces (I Piece or L Piece)
4. *`State`*: the class that contain both a field grid and an incoming Tromino
5. *`Action`*: that represent an action choosable on a given state (combination for placement and rotation of an incoming Tromino)
6. *`Game`*: the class that contain the configuration (completed lines scores), the current state and score of a baby Tetris game (the configuration is loaded from a file : *`config.txt`*)
7. *`MDP`*: the class that all the necessary methods in order to compute the value iteration over a game

Now that we overviewed the global implementation architecture, we will describe our implementation choices.

1.1.2 Implementation Choices

- **Piece teleported to their final position**: in order to make the number of intermediate states when choosing an action, we compute on a state all available actions on this state, then according to the policy we make a choice over these actions and we teleport the incoming piece in the current state directly to its final position (with the right rotation)
- **Reward Function**: The reward function as defined in Section 1 take both the current state s and an available action a_s. Here since we can calculate the available actions, we can know in which state s' we're going to arrive and we can compute directly in s' the number of completed lines before remove them right to the game. Then we abstract the reward function by counting the number of complete lines in the resulting state

- **Game Ending**: According to the subject, the game should end whenever a piece exceed the height limit of the grid. Again there, we can compute all available actions on a given state, since we cannot place a Tromino outside the grid, such an action will not appear in the set of available actions. Then the game end when there is no available action for the current game state

- **Max Iteration**: Since the value iteration isn't an exact method, if we fix an ϵ which is too small, we may iterate for a very long moment. For this reason we fix a maximum number of iteration for the expected value improvement, then we always a value iteration which compute in a reasonable time even if the expected value vector is less qualitative than the ϵ margin we fixed

1.1.3 Incoming Improvement

For now we compute naively the value iteration algorithm on all state combinations, which represents

$$= 2^{\text{gridHeight}*\text{gridWidth}} * \text{nbOfTrominoTypes} = 2^{4*4} * 2 = 2^{17} = 131072$$

states.