Chain Reaction is a deterministic, turn-based, perfect-information game for two players. Each player places colored orbs on an m × n board; when a cell reaches its critical mass (its number of orthogonal neighbors), it “explodes,” distributing orbs to adjacent cells and possibly triggering cascades—and converting opponent’s orbs in the process.
In this assignment, you will:
- ● Model the game state and legal moves.
- ● Implement a minimax agent with alpha-beta pruning.
- ● Design and tune one or more evaluation heuristics.
- ● Experiment and report on agent performance.

The most interesting thing is how unpredictable the game seems to be in the end, at least when you play it with your human friends. The obvious heuristic that tells us you're better off at the moment by having as many orbs as possible turns out to be very wrong. While it seems to everyone that says, red will win, blue suddenly takes over.

Rules of the Game

1. The gameplay takes place on an m×n board. The most commonly used size of the board is 9x6 (9 rows, 6 columns).
2. For each cell in the board, we define a critical mass. The critical mass is equal to the number of orthogonally adjacent cells. That would be 4 for usual cells, 3 for cells in the edge and 2 for cells in the corner.
3. All cells are initially empty. The Red and the Blue player take turns to place "orbs" of their corresponding colors. The Red player can only place an (red) orb in an empty cell or a cell which already contains one or more red orbs. When two or more orbs are placed in the same cell, they stack up.
4. When a cell is loaded with a number of orbs equal to its critical mass, the stack immediately explodes. As a result of the explosion, to each of the orthogonally adjacent cells, an orb is added and the initial cell loses as many orbs as its critical mass. The explosions might result in overloading of an adjacent cell and the chain reaction of explosion continues until every cell is stable.
5. When a red cell explodes and there are blue cells around, the blue cells are converted to red and the other rules of explosions still follow. The same rule is applicable for other colors.
6. The winner is the one who eliminates every other player's orbs.

System Architecture & File Protocol

Your solution must consist of two separate components:
1. Game Engine (Backend):
   - ○ Implements the game logic, minimax search, alpha-beta pruning, and heuristics.
   - ○ Reads the shared game-state file to parse the latest human move.
   - ○ Computes its own move and writes it back to the same file following the protocol.
2. UI (Frontend):
○ Presents the current board to the human player (Web). ○ After each human move, it updates the shared file according to the protocol and waits for the AI's response. ○ When the game ends (win/loss), it displays a message.

File Communication Protocol

All moves and game-state updates travel through a single text file (e.g., gamestate.txt). Both components must agree on the following format:

1. Header line: Indicates who made the current move. One of: ○ Human Move: ○ AI Move:
2. Board state: Nine rows of six cells each.

○ Each cell is represented as either 0 (empty) or where is the orb count and is R or B for Red/Blue.

○ Cells in a row separated by a single space; rows separated by newline.

Example file after human (red) places at (0,0):

```
Human Move:
1R 0 0 0 0 0
0  0 0 0 0 0
... (7 more rows)
```

After reading this, the Game Engine (Blue) updates the board via explosions, chooses its move, and overwrites the file with:

```
Ai Move:
1R 1B 0 0 0 0
0 0 0 0 0 0
... (7 more rows)
```

The UI then reads this update, refreshes the display, and prompts the human for the next move. Continue until one player wins.

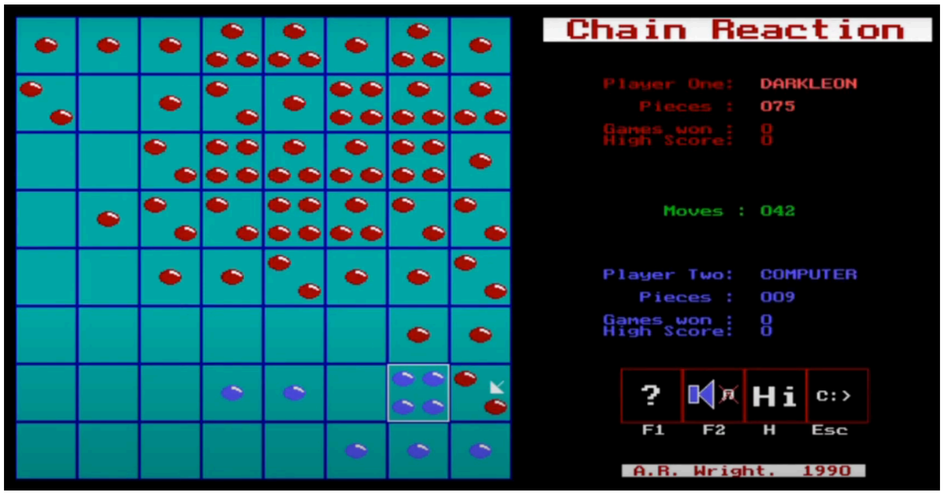
Important:

● Always overwrite the shared file completely on each write.
● Use exact keywords Human Move: and AI Move:
● Synchronize file reads/writes and block until the expected header appears.

Assignment Tasks

Task 1: Game Representation via an UI

● You need to build an nice UI for the game.
● It will be a 2 player variant of the game.
● You can use any coding framework that suits you better to build the UI.
● Orb colour must be red and blue.



Player One: DARKLEON
Pieces : 075
Games won : 0
High Score: 0

Moves : 042

Player Two: COMPUTER
Pieces : 009
Games won : 0
High Score: 0

Task 2: Minimax with Alpha-Beta

● Implement minimax_search (state, depth_limit) with alpha-beta pruning.

● Your agent must return both value and best action.

Task 3: Heuristic Evaluation

● Design at least five domain-inspired evaluation heuristic functions .

● Describe their rationale in the report.

● Integrate into your minimax agent.

Task 4:

● Adapt your UI and game engine so that it can play AI vs AI

● Play matches between:

○ Random-move agent vs. your agent (varying depth/heuristics).

○ Heuristic-A vs. Heuristic-B. (and so on.)

● Include the statistics (win rate, time to complete game etc) in your report