**Game**: ***Align3***

A 4x4 board game is played by two players (Machine Bot and Human). The game playing agent is implemented using the Minimax algorithm. To achieve a substantial speedup, the Alpha-Beta node pruning algorithm is also implemented.

The game will be started first by Machine Bot.

**Implementation:**

Game implemented using Python 3.7 and Turtle.

**Running the code:**

1. Run the file Align3-Game.py from IDE (Or open the terminal and type in `python Align3-Game.py ` to play the game.)
2. Wait for Machine Bot to start the game to play as first player.(The Bot plays the first move and uses green coins).
3. Click on the corresponding column to put your coin it (Human plays with blue coins)
4. Whenever one episode of the game ends, click on one of the options given below in the GUI for next games.

**Modules**

Implemented the following logical modules in the code given below and developed other utility modules as well which are required for the development of this game.

1. Successor\_function

2. Terminal\_test

3. Utility\_value

4. MIN\_VALUE

5. MAX\_VALUE

6. Minimax algorithm

7. Alpha Beta pruning

**Analysis Module**

Analysis module in the left frame of the GUI produces the results from R1 to R12, which depicts the analysis based on implemented Algorithms.

**Algorithms:**

Implemented Minimax and Alpha-Beta Pruning Algorithm.

**Pseudocodes of the algorithms:**

**Minimax** **Algorithm**

function MINIMAX-DECISION(state) returns an action

return arg\_max\_a ∈ ACTIONS(s) MIN-VALUE(RESULT(state, a))

function MAX-VALUE(state) returns a utility value

if TERMINAL-TEST(state) then return UTILITY(state)

v ← −∞

for each a in ACTIONS(state) do

v ← MAX(v, MIN-VALUE(RESULT(s, a)))

return v

function MIN-VALUE(state) returns a utility value

if TERMINAL-TEST(state) then return UTILITY(state)

v ← ∞

for each a in ACTIONS(state) do

v ← MIN(v, MAX-VALUE(RESULT(s, a)))

return v

**Alpha-Beta Pruning**

function ALPHA-BETA-SEARCH(state) returns an action

v ← MAX-VALUE(state,−∞,+∞)

return the action in ACTIONS(state) with value v

function MAX-VALUE(state,α, β) returns a utility value

if TERMINAL-TEST(state) then return UTILITY(state)

v ← −∞

for each a in ACTIONS(state) do

v ← MAX(v, MIN-VALUE(RESULT(s,a),α, β))

if v ≥ β then return v

α ← MAX(α, v)

return v

function MIN-VALUE(state,α, β) returns a utility value

if TERMINAL-TEST(state) then return UTILITY(state)

v ← +∞

for each a in ACTIONS(state) do

v ← MIN(v, MAX-VALUE(RESULT(s,a) ,α, β))

if v ≤ α then return v

β ← MIN(β, v)

return v