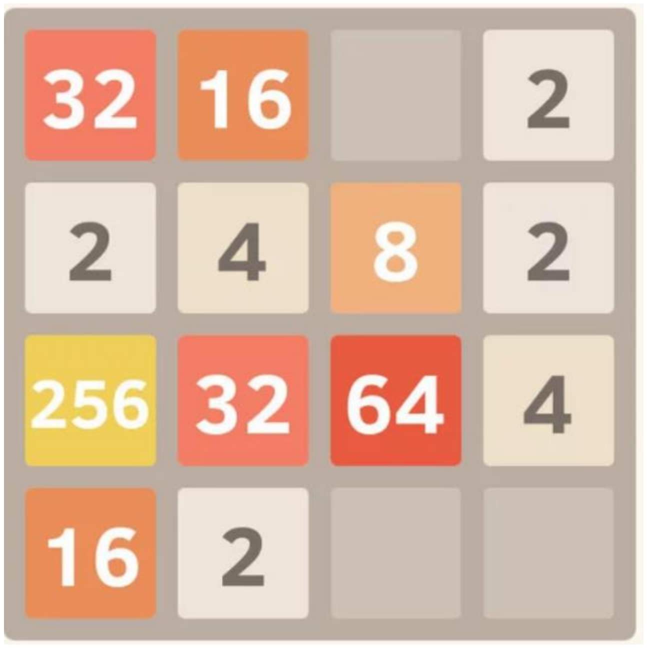
UNI: tmd2142
Name: Tri Minh Do

Project proposal: Implement an AI to solve 2048-puzzle game.

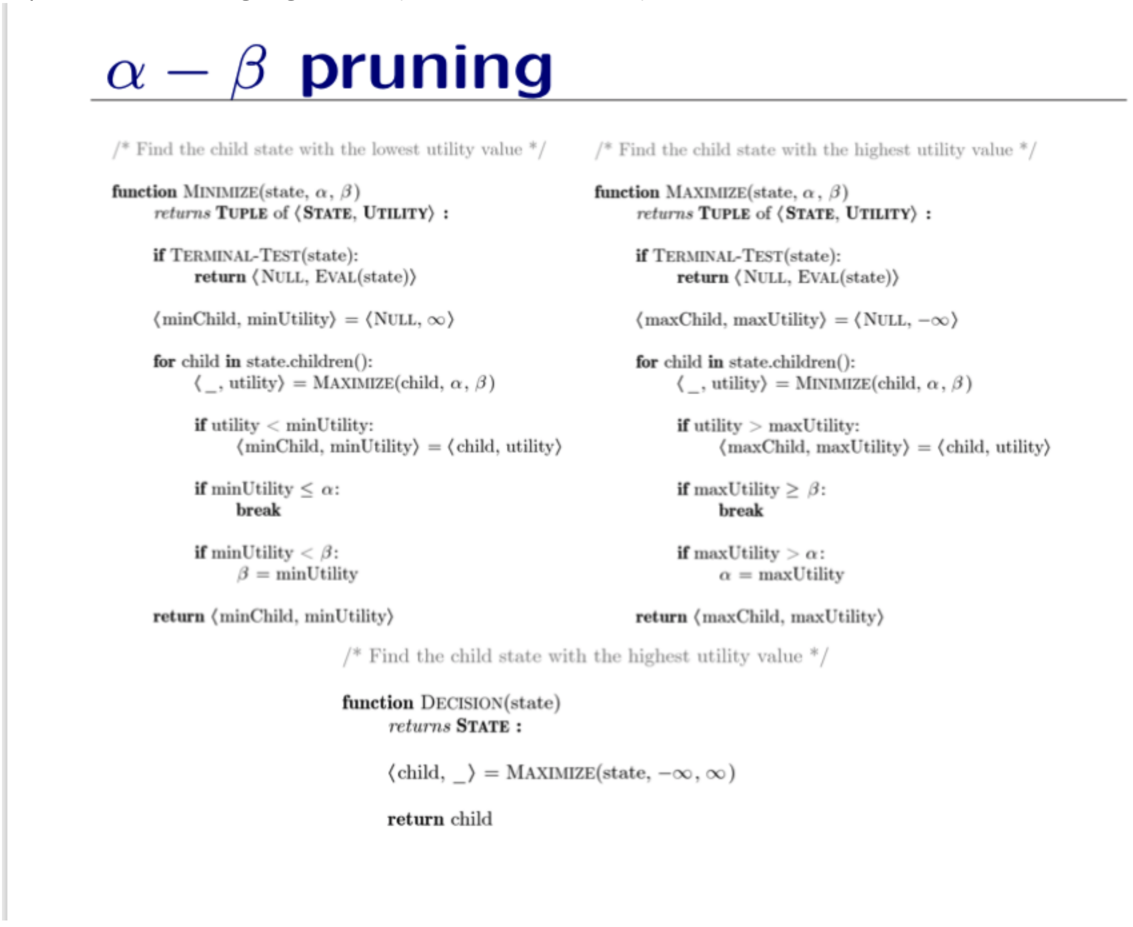


1. Rule of the game
- "2048 is a single-player sliding block puzzle game designed by Italian web developer Gabriele Cirulli. The game's objective is to slide numbered tiles on a grid to combine them to create a tile with the number 2048. However, one can continue to play the game after reaching the goal, creating tiles with larger number" – Wikipedia

- Game can be played online here: https://play2048.co/

2. Implement Adversarial Search Algorithm to play the game intelligently.
- We will implement a data structure to simulate the 4x4 board
- Implement alpha-beta pruning algorithm with a heuristic function at each state to decide which move to take next.
- We will limit the time it takes to calculate each next move. With 1 core, the algorithm can only search up to n-depth without violating the time limit. If we have multiple cores, we can search many branches at the same time which means we can go deeper in the search tree and achieve greater score for the game.
- Testing: We will run the game 10 times on 1 core then calculate the average score. After that, we will run the game 10 times on multiple cores and compare the average score to see how multicore improves the result.

Alpha-Beta Pruning algorithm (Slide from AI class):



$\alpha - \beta$ pruning

```
/* Find the child state with the lowest utility value */

function MINIMIZE(state, $\alpha$, $\beta$)
    returns TUPLE of ⟨STATE, UTILITY⟩ :

    if TERMINAL-TEST(state):
        return ⟨NULL, EVAL(state)⟩

    ⟨minChild, minUtility⟩ = ⟨NULL, $\infty$⟩

    for child in state.children():
        ⟨_, utility⟩ = MAXIMIZE(child, $\alpha$, $\beta$)

        if utility < minUtility:
            ⟨minChild, minUtility⟩ = ⟨child, utility⟩

        if minUtility $\leq \alpha$:
            break

        if minUtility $< \beta$:
            $\beta$ = minUtility

    return ⟨minChild, minUtility⟩
```

```
/* Find the child state with the highest utility value */

function MAXIMIZE(state, $\alpha$, $\beta$)
    returns TUPLE of ⟨STATE, UTILITY⟩ :

    if TERMINAL-TEST(state):
        return ⟨NULL, EVAL(state)⟩

    ⟨maxChild, maxUtility⟩ = ⟨NULL, $-\infty$⟩

    for child in state.children():
        ⟨_, utility⟩ = MINIMIZE(child, $\alpha$, $\beta$)

        if utility > maxUtility:
            ⟨maxChild, maxUtility⟩ = ⟨child, utility⟩

        if maxUtility $\geq \beta$:
            break

        if maxUtility $> \alpha$:
            $\alpha$ = maxUtility

    return ⟨maxChild, maxUtility⟩
```

```
/* Find the child state with the highest utility value */

function DECISION(state)
    returns STATE :

    ⟨child, _⟩ = MAXIMIZE(state, $-\infty$, $\infty$)

    return child
```