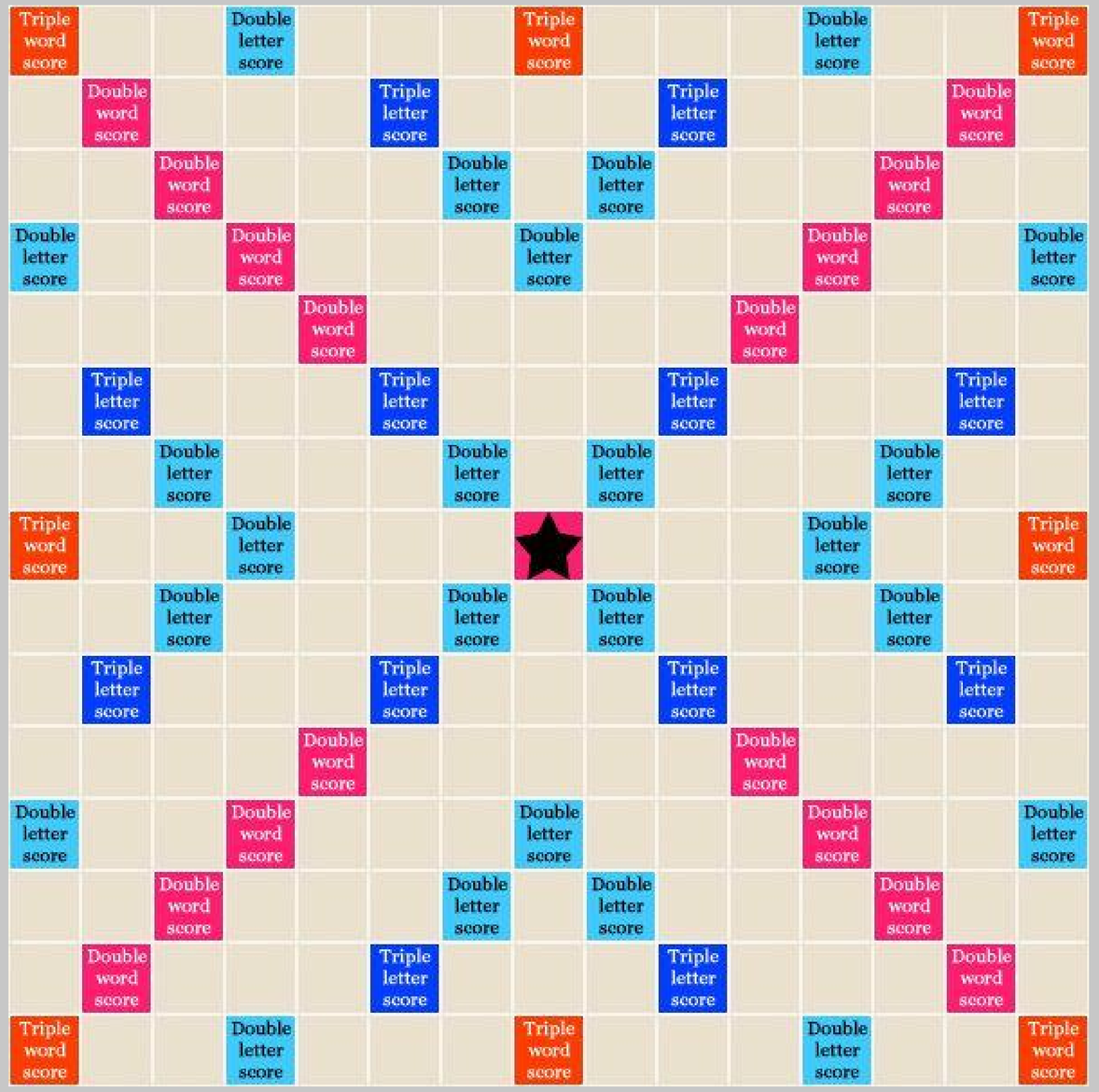
Scrabble AI

FINAL GROUP PROJECT PRESENTATION

CSD311 - ARTIFICIAL INTELLIGENCE

A Note on Scrabble

- Originated during the Great Depression, it was created by **Alfred Mosher Butts**
- Played by 2-4 players on a square grid of size 15 x 15 made up of square shaped cells.
  We implemented our project assuming **two players** were going playing against each other.
- Each letter in the game has a score assigned to it, on the basis of its frequency in the target language, which is English in our case.
- The board also contains “premium” squares, that provide benefits such as “double word” score, or “triple letter” score.
- The game ends when no more moves are possible by any player.



A Sample Scrabble Board

This is the problem we want to solve.

- The efficient generation of all legal moves in any given position, building upon classic algorithms while introducing optimizations for modern hardware.

- The development and comparison of different AI player strategies, ranging from simple greedy approaches to more sophisticated methods incorporating adversarial reasoning and Monte Carlo techniques.

& the challenges we'll face

- Scrabble is PSPACE-complete, which means finding the best possible move for a given state might take an exponential amount of time.

- Figuring out all possible moves for a given game state.

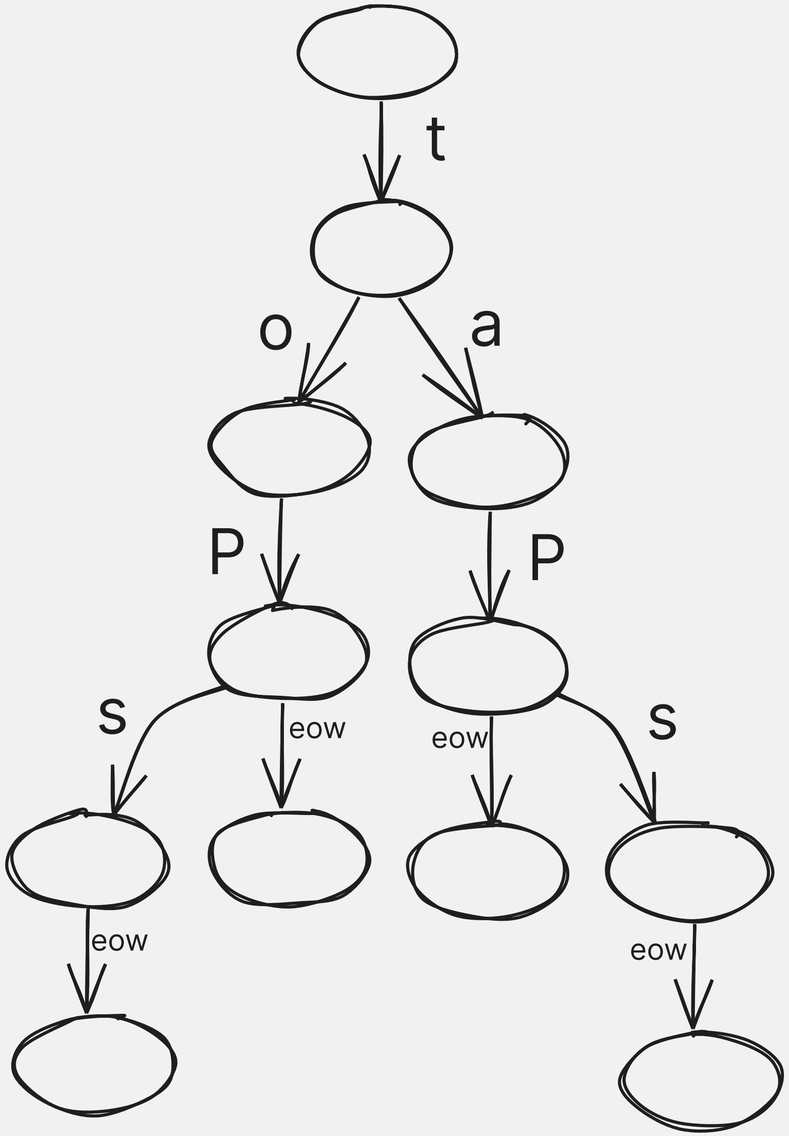
- Choosing an appropriate heuristic for our AI agents.

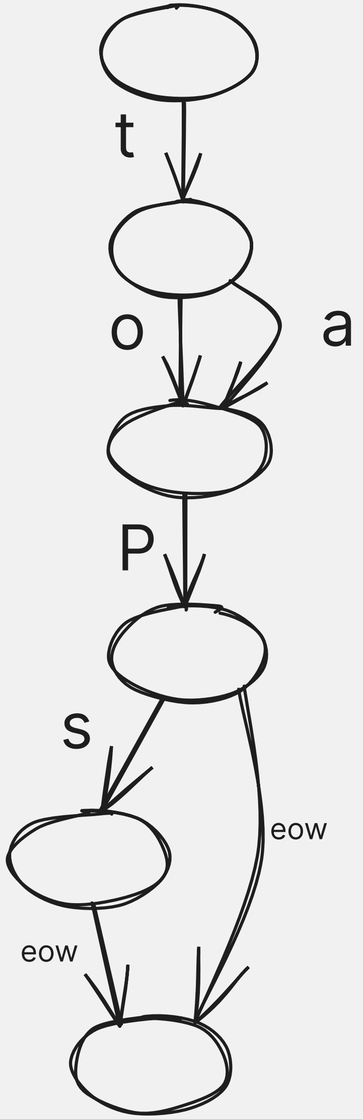
Directed Acyclic Word Graph (DAWG)

THE MOVE GENERATION PROBLEM

- The minimization of the DAWG reduces space requirements by an order of magnitude compared to a trie, while still allowing O(L) word lookup time for words of length L.
- Using the DAWG allows us to perform DFS on all possible positions where we can play a move and check in linear time, if they exist in game dictionary.
- Across all our testing, finding valid moves for a game state is the performance bottleneck. But across all our testing, we found out that increasing computational investment on heuristics yielded diminishing returns.

TRIE VS DAWG





Three types of AI agents that we implemented

The Three AI Agents

WHO DO YOU THINK PERFORMED THE
BEST AMONG THE THREE?

1. The Greedy Player
2. The Adversarial Player
3. The Conservative Player

THE GREEDY PLAYER

- Always selects highest scoring move available to it
- Does not care about rack leave quality ie. does not care about what moves might be available to it in the future.
- Does not care about what the opponent might play.

THE ADVERSARIAL PLAYER

- Attempts to incorporate opponent modelling and defensive play.
- Heuristic:

$$Score_{effective} = MoveScore - \alpha \cdot MaxOpponentScore$$

- MaxOpponentScore is estimated through -
  1. Probabilistic modelling of opponent's rack
  2. Simulation of opponent's potential responses.
  3. Evaluation of board position vulnerability.
- Opponent rack is chosen on weighted probability

$$P(tile_i \in OpponentRack) = \frac{RemainingCount(tile_i)}{TotalRemainingTiles}$$

THE CONSERVATIVE PLAYER

- Evaluates all its move on the following heuristic :

$$Score_{total} = MoveScore + \beta \cdot RackLeaveValue + \gamma \cdot PositionalValue$$

1. β weights the rack leave quality
2. γ weights the positional valuation.
3. RackLeaveValue considers vowel-consonant balance and high-value tile retention
4. PositionalValue evaluates board control and premium square accessibility

- RackLeaveValue is calculated by the following formula :

$$VCBalance = 1.0 - |V_{ratio} - 0.4|$$

1. Vratio is the ratio of vowels to total tiles, with 0.4 being the empirically determined optimal ratio.

Monte Carlo Simulations

FOR ENDGAME SCENARIOS AND CRITICAL SITUATION, WE IMPLEMENT MONTE CARLO SIMULATIONS

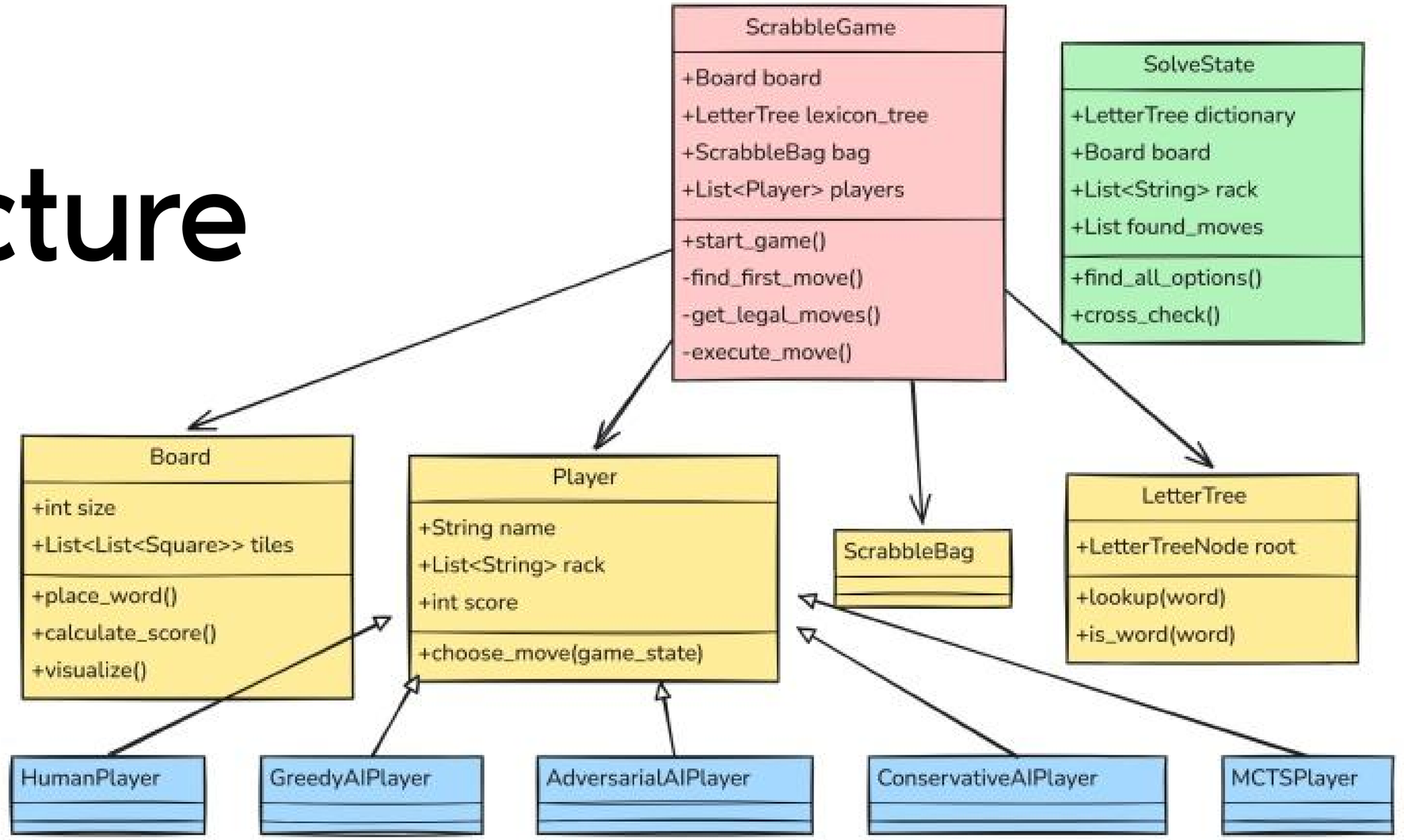
Algorithm 1 Monte Carlo Move Evaluation

```
moves ← GenerateLegalMoves(position)
for move in moves do
    score ← 0
    for i ← 1 to NumSimulations do
        rack ← SimulateOpponentRack()
        response ← SimulateOpponentMove(rack)
        score ← score + (MoveScore − ResponseScore)
    end for
    move.value ← score/NumSimulations
end for
return BestMove(moves) =0
```

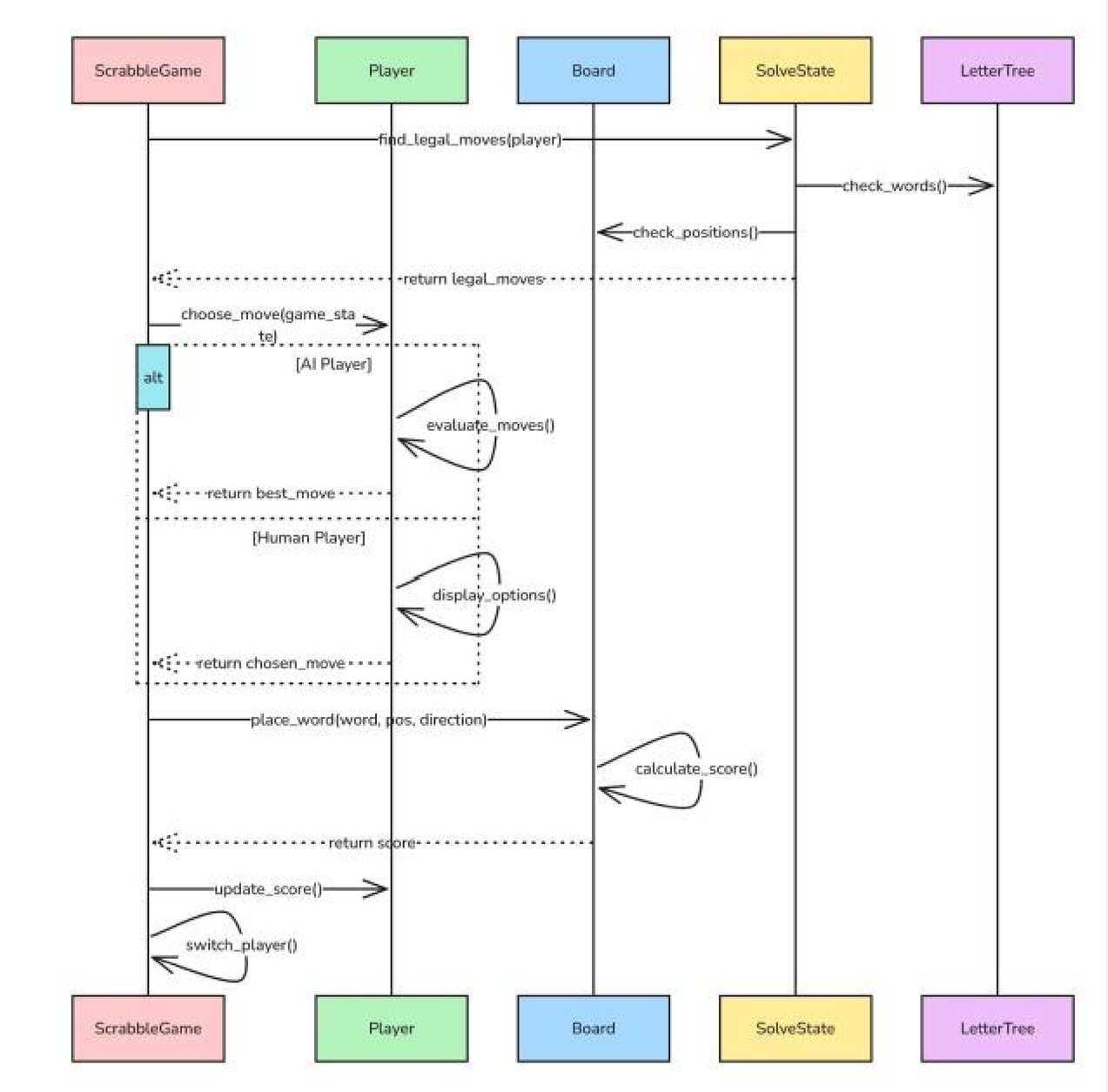
Code Flow

1. All possible moves for a particular board state are generated.
2. The moves are passed into the AI agent, which then evaluates each move on the basis of the heuristic assigned to it.
3. A move is selected and is played. Game state is update and control is passed to the other AI agent.
4. New letters are given to the AI agent to complete it’s rack once again.

System Architecture



How one round of play works



Observations

The Greedy Player's simple strategy offers rapid move selection but misses strategic opportunities, particularly in rack management

The Adversarial Player's opponent modelling provides defensive capabilities at the cost of increased computational overhead

The Conservative Player's focus on rack balance and future potential sometimes sacrifices immediate high-scoring opportunities

Simple scoring maximization proved surprisingly effective; complex strategic planning may be overvalued in computer Scrabble

Thank you

FINAL GROUP PROJECT PRESENTATION

CSD311 - ARTIFICIAL INTELLIGENCE