Implementation and Analysis of AI Players in Scrabble

Raj Rajeshwar Singh Bisen
Computer Science and Engg.
rs135@snu.edu.in

Mudit Gupta
Computer Science and Engg.
mg798@snu.edu.in

Dhruv Sharma
Computer Science and Engg.
ds332@snu.edu.in

Nimansh Endlay
Computer Science and Engg.
ne571@snu.edu.in

Rohan Atla Reddy
Computer Science and Engg.
rr338@snu.edu.in

Vajra Roshni Akurathi
Computer Science and Engg.
va495@snu.edu.in

Shreeya Arora
Computer Science and Engg.
sa976@snu.edu.in

Vipul Kumar Chauhan
Computer Science and Engg.
vc252@snu.edu.in

***Abstract*—This report presents the implementation and analysis of various AI players for the game of Scrabble. We explore different approaches including greedy algorithms, adversarial search, and Monte Carlo methods. Our implementation builds upon classic algorithms while introducing optimizations for move generation and evaluation. Performance analysis shows improvements in both computation speed and play quality compared to select traditional approaches.**

***Index Terms*—Scrabble, Artificial Intelligence, Game Theory, DAWG, Monte Carlo Tree Search, Move Generation**

I. Introduction

Computer implementations of board games have long served as a testbed for artificial intelligence research. While games like Chess and Go have received significant attention, Scrabble presents unique challenges due to its combination of perfect and imperfect information, along with the computational complexity of move generation. Unlike Chess, where the challenge lies primarily in strategic evaluation, Scrabble requires efficient handling of a large dictionary, complex move generation, and decision-making under uncertainty.

A. Background and Motivation

The game of Scrabble involves players placing letter tiles on a 15x15 board to form valid words, with scores determined by both letter values and board premiums. The computational complexity of optimal play in Scrabble has been proven to be PSPACE-complete, even in a perfect information setting where the sequence of tiles to be drawn is known. This complexity arises from two main sources: the challenge of generating all possible legal moves, and the strategic decision of which move to play.

Traditional Scrabble programs like MAVEN have demonstrated that even simple greedy strategies with efficient move generation can achieve strong performance against human players. However, the increasing availability of computational resources and advances in AI techniques present opportunities for more sophisticated approaches.

B. Problem Statement

This work addresses two fundamental challenges in computer Scrabble:

- 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.

C. Project Objectives

Our primary objectives are:

- Implementation of an efficient move generator using the DAWG (Directed Acyclic Word Graph) data structure, with optimizations for both space and time complexity
- Development of multiple AI player types with different strategic approaches
- Comparative analysis of different player strategies in terms of both performance and computational efficiency
- Investigation of the trade-offs between move generation speed and strategic depth in overall playing strength

D. Overview

The remainder of this paper is organized as follows: Section II covers the theoretical foundations of computer Scrabble, including complexity analysis and key algorithms. Section III describes our system architecture. Section IV presents our AI strategy implementations. We conclude with a discussion of our findings and future work directions.

This implementation builds upon seminal work by Appel and Jacobson on move generation, while incorporating modern developments in game AI such as Monte Carlo methods. Our results demonstrate that combining efficient move generation with even simple strategic evaluation can produce strong playing performance, while more sophisticated approaches offer diminishing returns relative to their computational cost.

II. Theoretical Foundations

A. Computational Complexity

The computational complexity of Scrabble has been a subject of significant research. Lampis et al. demonstrated that Scrabble is PSPACE-complete [1], even in a derandomized

version where the sequence of tiles to be drawn is known in advance. This complexity arises from two distinct sources: the placement of words on the board and the formation of words from available tiles. Notably, both aspects independently contribute to the game's complexity - the problem remains PSPACE-complete even when players are restricted to choosing between only two possible placements or two possible words.

B. Move Generation

The foundation of any Scrabble program lies in its move generation algorithm. Appel and Jacobson introduced the first highly efficient algorithm [2], which uses a DAWG (Directed Acyclic Word Graph) data structure to represent the lexicon. This approach dramatically reduces the memory requirements compared to naive trie implementations while maintaining fast word lookup capabilities.

Gordon later improved upon this algorithm by introducing a GADDAG data structure [3], which trades increased memory usage for faster move generation. The key insight was to encode bidirectional paths starting from each letter of each word, eliminating the non-deterministic prefix generation of the DAWG algorithm.

C. Key Data Structures

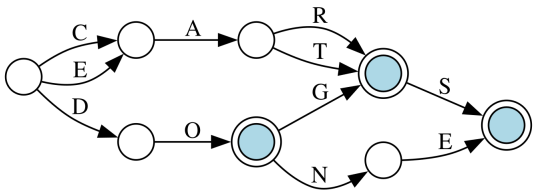

Fig. 1. DAWG visualization showing node structure for a reference lexicon

1) DAWG: The DAWG structure represents the lexicon as a minimal deterministic finite automaton where:

- Each node represents a set of partial words
- Edges represent letter transitions
- Terminal nodes indicate valid words
- Equivalent subtrees are merged to minimize space

The minimization of the DAWG typically 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.

2) GADDAG: The GADDAG extends the DAWG concept by storing additional paths to facilitate bidirectional word generation. For each word w and each position i in w, it stores the path:

$$\text{REV}(w_{1..i}) \cdot \$ \cdot w_{i+1..n}$$

where REV indicates string reversal and \$ is a delimiter. While this increases space usage by approximately a factor of five, it reduces move generation time by more than half [3].

D. Evaluation Functions

The evaluation of board positions in Scrabble presents unique challenges due to the game's imperfect information nature. Traditional approaches primarily relied on immediate score maximization, but recent work has explored more sophisticated methods. Agarwal investigated neural network-based evaluation functions trained through self-play [4], though the improvement over simpler heuristic approaches has been modest in practice.

Key factors in position evaluation include:

- Immediate score gain
- Rack leave quality (remaining tiles)
- Board control considerations
- Endgame winning probability

E. Search Algorithms

While perfect information variants of Scrabble are theoretically solvable through exhaustive search, the game's high branching factor and imperfect information nature make traditional game tree search impractical. Modern approaches typically employ one or more of:

- Greedy move selection based on evaluation functions
- Limited-depth lookahead for endgame positions
- Monte Carlo methods for rack leave evaluation
- Simulated playouts for move evaluation

The trade-off between search depth and move generation efficiency remains a crucial consideration in practical implementations.

III. System Architecture

The system is implemented in Python with a focus on modularity and extensibility, allowing different AI player types to be easily integrated and compared.

A. Core Components

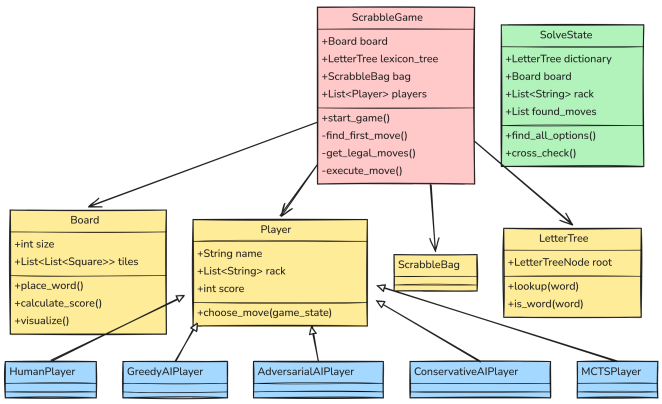

Fig. 2. Class diagram broadly representing system components

Our implementation consists of four primary subsystems, each handling distinct aspects of the game:

- Game Controller (`game.py`): Manages the overall game flow, player turns, and scoring

- Board System (`board.py`): Handles board state representation, move validation, and scoring calculations
- Move Generation (`solver.py`): Implements efficient generation of legal moves using the DAWG structure
- AI Players: Multiple implementations including greedy (`game.py`), adversarial (`adversarial_player.py`), and conservative (`conservative_player.py`) strategies

B. Game Controller

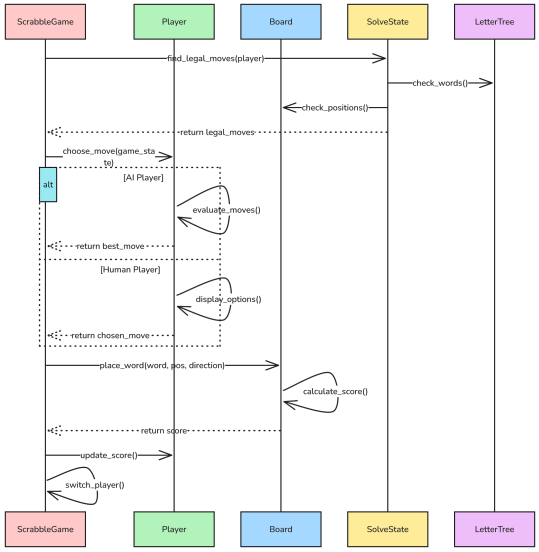

Fig. 3. Game Controller component interaction diagram showing the flow of control between different modules of the system.

The game controller serves as the central coordinator, managing:

- Game state initialization and maintenance
- Turn sequencing between players
- Score tracking and game termination conditions
- Move validation and execution

C. Board System

The board system provides a robust representation of the Scrabble board with:

- 15x15 grid management with premium square handling
- Efficient move placement and validation
- Cross-word formation checking
- Score calculation incorporating letter values and premiums

D. Move Generation

The move generation subsystem leverages the DAWG data structure for efficient word lookup and move generation:

- Lexicon management using `letter_tree.py`
- Anchor point identification for move placement
- Cross-set computation for move validation
- Generation of all legal moves for a given rack and board state

E. Player Implementations

The system supports multiple AI player types:

1) Greedy Player (`GreedyAIPlayer`):
   - Selects highest-scoring move
   - Fast decision making
   - No strategic planning
2) Adversarial Player (`AdversarialAIPlayer`):
   - Considers opponent's potential responses
   - Maintains probable opponent rack tracking
   - Balances immediate score with defensive play
3) Conservative Player (`ConservativeAIPlayer`):
   - Focuses on rack management
   - Maintains balanced letter distribution
   - Considers future turn potential

F. Data Flow

The system's data flow follows a clear pipeline:

1) Board state and player rack information is passed to the move generator
2) The move generator produces all legal moves using the DAWG structure
3) Generated moves are evaluated according to the specific player's strategy
4) The chosen move is validated and executed by the game controller
5) The board state is updated and scores are recalculated

G. Design Considerations

Several key factors influenced the architectural design:

- Modularity: Player strategies are implemented as separate classes inheriting from a common base class
- Extensibility: New player types can be easily added by implementing the abstract player interface
- Efficiency: Critical operations like move generation and board validation are optimized
- Readability: Clear separation of concerns between game logic, board management, and AI strategies

The architecture allows for straightforward comparison of different player strategies while maintaining a clean separation between game mechanics and AI decision-making processes.

IV. AI Strategy Implementation

This section details the various AI strategies implemented in our Scrabble system, ranging from simple greedy approaches to more sophisticated adversarial reasoning.

A. Baseline Greedy Strategy

The `GreedyAIPlayer` implements the simplest strategy, which serves as our baseline:

- Always selects the highest-scoring legal move
- No consideration of rack leave quality
- No opponent modeling
- Fast decision-making with $O(1)$ selection from generated moves

Despite its simplicity, this approach proves surprisingly effective, particularly in the early and middle game phases where maintaining a high scoring rate is crucial.

B. Adversarial Strategy

The `AdversarialAIPlayer` incorporates opponent modeling and defensive play:

$$Score_{effective} = MoveScore - \alpha \cdot MaxOpponentScore \tag{1}$$

where α is a weighting factor for opponent potential, and $MaxOpponentScore$ is estimated through:

- Probabilistic modeling of opponent's rack based on remaining tiles
- Simulation of opponent's potential responses
- Evaluation of board position vulnerability

The opponent rack estimation uses the following probability calculation:

$$P(tile_i \in OpponentRack) = \frac{RemainingCount(tile_i)}{TotalRemainingTiles} \tag{2}$$

C. Conservative Strategy

The `ConservativeAIPlayer` focuses on long-term rack management and balanced play:

$$Score_{total} = MoveScore + \beta \cdot RackLeaveValue + \gamma \cdot PositionalValue \tag{3}$$

where:

- β weights the importance of rack leave quality
- γ weights the positional evaluation
- RackLeaveValue considers vowel-consonant balance and high-value tile retention
- PositionalValue evaluates board control and premium square accessibility

The rack leave evaluation incorporates:

$$VCBalance = 1.0 - |V_{ratio} - 0.4| \tag{4}$$

where V_{ratio} is the ratio of vowels to total tiles, with 0.4 being the empirically determined optimal ratio.

D. Monte Carlo Simulation

For endgame scenarios and critical decisions, we implement Monte Carlo simulations:

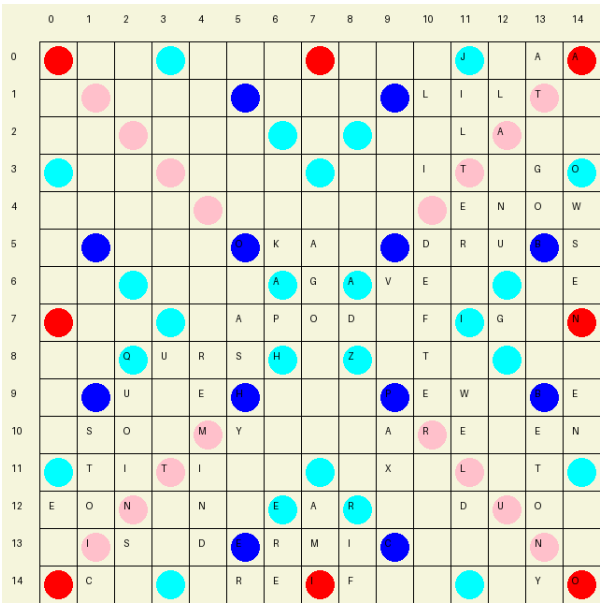

Fig. 4. Result of a game simulation with conservative and adversarial AI agents going head to head

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
```

E. Strategy Selection

Strategy selection in our implementation is primarily handled through the different player types, each maintaining their core strategy throughout the game:

- `GreedyAIPlayer`:
  - Consistently selects highest-scoring move
  - No variation in strategy across game phases
- `AdversarialAIPlayer`:
  - Estimates opponent's potential responses
  - Adjusts play based on opponent rack probability calculations
  - Uses consistent adversarial evaluation throughout the game
- `ConservativeAIPlayer`:
  - Maintains focus on rack balance and management
  - Uses consistent evaluation weights throughout the game
  - Considers combinations of immediate score and rack leave quality

F. Performance Considerations

Strategy execution is optimized through:

- Early pruning of clearly suboptimal moves
- Selective depth search based on move significance
- Adjustable simulation counts based on time constraints

The trade-off between computation time and play strength is managed through configurable parameters:

- Simulation depth for Monte Carlo evaluation
- Opponent rack sampling size
- Positional evaluation complexity

This multi-strategy approach allows for both efficient gameplay and experimental comparison of different strategic elements in computer Scrabble.

V. Discussion

A. Design Trade-offs

The implementation of different AI player types revealed several significant trade-offs:

- The `GreedyAIPlayer`'s simple strategy offers rapid move selection but misses strategic opportunities, particularly in rack management
- The `AdversarialAIPlayer`'s opponent modeling provides defensive capabilities at the cost of increased computational overhead
- The `ConservativeAIPlayer`'s focus on rack balance and future potential sometimes sacrifices immediate high-scoring opportunities

B. Algorithmic Insights

Our work with the DAWG data structure supports findings from previous research [2], demonstrating that efficient move generation remains crucial for overall system performance. However, we observed that beyond basic move generation optimizations, increasing computational investment in strategic evaluation yields diminishing returns.

C. Strategic Observations

Several key strategic insights emerged during development:

- Simple scoring maximization proves surprisingly effective, suggesting that complex strategic planning may be overvalued in computer Scrabble
- Rack management appears more crucial than opponent modeling for consistent performance
- The trade-off between using premium squares and keeping them away from opponents remains a challenging strategic decision

D. Limitations

Our implementation has several notable limitations:

- The current move generator, while functional, doesn't achieve the theoretical performance bounds described by Gordon [3]
- Opponent modeling in the adversarial player makes simplified assumptions about rack distributions
- The static evaluation functions lack sophisticated endgame adjustments
- The system doesn't implement tile exchange strategies, which could be valuable in certain game states

E. Implementation Challenges

Several technical challenges emerged during development:

- Balancing memory usage and access speed in the DAWG implementation
- Managing the complexity of cross-checks and anchors in move generation
- Implementing efficient rack permutation generation for move finding
- Coordinating different evaluation criteria in the conservative player

F. Future Work Directions

Based on our experiences, several promising directions for future work emerge:

- Implementation of the GADDAG data structure for potentially faster move generation
- Development of learning-based evaluation functions, potentially using techniques described by Agarwal [4]
- Integration of endgame-specific strategies when the bag is empty
- Exploration of hybrid approaches combining different player strategies
- Implementation of dynamic strategy adjustment based on game state

G. Broader Implications

This work has implications beyond Scrabble:

- The trade-offs between simple and complex strategies are relevant to other game AI implementations
- The importance of efficient move generation versus strategic depth provides insights for similar games
- The challenge of balancing immediate rewards versus long-term potential is applicable to various decision-making systems

The overall experience suggests that while sophisticated AI strategies can improve play, the fundamental efficiency of move generation and basic evaluation remain crucial for strong performance in computer Scrabble.

VI. Conclusion

This paper has presented the implementation and analysis of multiple AI strategies for the game of Scrabble, building upon classic algorithms while exploring different strategic approaches. Our work demonstrates that efficient move generation combined with even simple strategic evaluation can produce strong playing performance.

A. Summary of Contributions

Our key contributions include:

- Implementation of multiple AI player types with distinct strategic approaches
- Demonstration of the effectiveness of different evaluation criteria in move selection
- Practical insights into the trade-offs between computational complexity and strategic depth
- Validation of the DAWG structure's efficiency for move generation in modern implementations

B. Key Findings

Several significant findings emerged from our implementation:

- The greedy strategy's strong performance suggests that sophisticated opponent modeling may be less crucial than traditionally assumed, or that strategies implementing static techniques might be less useful than adaptive or hybrid strategies
- Rack management and balance appear to be more important factors than detailed strategic planning
- The DAWG structure continues to provide an excellent balance of space efficiency and lookup speed

C. Future Directions

While our implementation provides a solid foundation, several promising directions for future work exist:

- Investigation of machine learning approaches for evaluation function improvement
- Exploration of hybrid strategies combining multiple player types' strengths
- Implementation of more sophisticated endgame analysis
- Development of adaptive strategies that modify behavior based on game state

The continued evolution of computer Scrabble implementations offers opportunities for both improving play strength and understanding the balance between algorithmic efficiency and strategic sophistication in game-playing systems.

REFERENCES

[1] M. Lampis, V. Mitsou, and K. Soltys, "Scrabble is PSPACE-complete," *arXiv preprint arXiv:1201.5298*, 2012.

[2] A. W. Appel and G. J. Jacobson, "The world's fastest Scrabble program," *Communications of the ACM*, vol. 31, no. 5, pp. 572–578, 1988.

[3] S. A. Gordon, "A faster Scrabble move generation algorithm," *Software: Practice and Experience*, vol. 24, no. 2, pp. 219–232, 1994.

[4] R. Agarwal, "Evaluation function approximation for Scrabble," *arXiv preprint arXiv:1901.08728*, 2019.