AI Project 2

Kiran Hedge (KXH230029)

CS 6364

April 2, 2025

An implementation of MiniMax and Alpha-Beta algorithms with static estimation for the Jumpy3 board game.

Table of Contents

a Implementation Details

a.1 1. Algorithms

- **MiniMax**: Classic algorithm with depth-limited search (White perspective)

- **Alpha-Beta**: Optimized MiniMax with pruning (35-60% fewer evaluations)
- **MiniMaxBlack**: MiniMax variant for Black's moves using board flipping
- **Improved Estimator**: Enhanced static evaluation with:
  - 3× King position weighting
  - Pawn positional bonuses
  - Path clearance analysis
  - Material advantage tracking
  - King exit proximity bonuses

a.2 2. Key Components

- **Board Class**: Represents the game board. Handles move generation, position searching, and other key functions like flipping the board for Black's move generation.
- **StaticEstimator Interface**: Core evaluation contract for board assessment.
- **BasicEstimator**: Implements the project handout's baseline heuristic for evaluating board states given in handout.
- **ImprovedEstimator**: A custom, advanced heuristic that enhances evaluation using positional and strategic factors.
- **AlphaBetaAlgorithm**: Implements the Alpha-Beta pruning optimization for MiniMax search.
- **MiniMaxAlgorithm**: Implements the classic MiniMax search algorithm.
- **Piece**: Defines the game pieces and their properties. Contains Black-/White kings and pawns also an empty space character.
- **Player**: Represents a player in the game, either White or Black.

- **ProblemType**: Defines different types of evaluation like MiniMax and AlphaBeta.
- **Result**: Stores and manages game outcomes after move calculations.
- **Utils**: Provides helper functions for I/O and validations.

b Setup & Execution

b.1 Requirements

- Java JDK 8+

b.2 Compile & Run

```
# Compile all files
javac -d bin/ src/models/*.java src/*.java

# Run programs
java -cp bin/ MiniMax input.txt output.txt <depth>
java -cp bin/ AlphaBeta input.txt output.txt <depth>
java -cp bin/ MiniMaxBlack input.txt output.txt <depth>
java -cp bin/ MiniMaxImproved input.txt output.txt <depth>
```

c Javadocs Generation Procedure

The code in this project is well-documented, with Javadocs provided for all key classes, interfaces, and methods. These Javadocs explain the purpose, parameters, return values, and usage of various components, making it easier for developers to understand the design and functionality of the project.

To generate the Javadocs for the project, you can run the following command:

```
# Execute to generate java docs
javadoc -d docs -sourcepath src -subpackages models src/*.java -private
```

Once the Javadocs are generated, you can open the `index.html` file located in the `docs` folder in a browser to view the full documentation.

d Jumpy Test Suite Setup and Usage

The JumpyTestSuite class provides a comprehensive test suite to validate the correctness of all AI algorithm variants (MiniMax, AlphaBeta, MiniMaxBlack, MiniMaxImproved) for the Jumpy3 board game. The suite automates the validation of:

- **Input/Output board comparisons**: Ensures that the generated board matches the expected output for a given input.
- **Position evaluation count**: Verifies that the number of evaluated positions matches the expected value.
- **Static estimate value**: Ensures that the static evaluation estimate matches the expected estimate.

d.1 Requirements

- **Java JDK 8+**: Ensure you have Java Development Kit (JDK) version 8 or higher installed.
- **Project Compilation**: All source files must be compiled before running the test suite.

d.2 Test Suite Overview

- **Automated Tests**: The test suite runs a series of predefined test cases, each containing:
  - An initial board configuration.
  - Expected results for each algorithm type (MiniMax, AlphaBeta, MiniMaxBlack, MiniMaxImproved).
  - A specified search depth.
- **Test Execution**: The suite compares the results from running the algorithms against the expected output, positions evaluated, and estimate values.
- **Results**: The test suite will output a summary of the test results, including the total number of tests, passed tests, and failed tests. It also provides a breakdown by algorithm type.

d.3 Running the Test Suite

```
# Compile all files
javac -d bin/ src/models/*.java src/*.java
```

To execute the test suite, run the following command:

```
# execute the test suite
java -cp bin/ JumpyTestSuite
```

d.4 Customizing the Test Cases

You can modify the predefined test cases by updating the TEST_CASES list in the JumpyTestSuite class.

Each test case consists of an initial board configuration, a search depth, and expected results for each algorithm type.

Example -

```
new TestCase("WwwwxxxxxxxxbbbB", 2, new HashMap<ProblemType, ExpectedResult>() {{
    put(ProblemType.MIN_MAX, new ExpectedResult("xwwwWxxxxxxxbbbB", 16, 0));
    put(ProblemType.MIN_MAX_BLACK, new ExpectedResult("WwwwxxxxxxxBbbbx", 16, 0));
    put(ProblemType.MIN_MAX_IMPROVED, new ExpectedResult("xwwwWxxxxxxxbbbB", 16, 12));
    put(ProblemType.ALPHA_BETA, new ExpectedResult("xwwwWxxxxxxxbbbB", 7, 0));
}})
```

e Example Comparisons (Basic vs Improved)

Initial Board	Depth	MiniMax Move	Positions evaluated	Estimate	MiniMaxImproved Move	Positions evaluated	Estimate
xwwxxbxxwBxxWxbb	11	xwwxxbxxwBxxxWbb	3201293	0	xwwxxbxxxxwxWBbb	3201293	7
xxwxxxWwBxwxbbbx	9	xxwxxxWxxwwxbbbB	254650	5	xxwxxxxwBWwxbbbx	254650	54
xwwxxxWwBxxxbbbx	9	xwwxxxWxxwxxbbbB	261922	4	xxwwxxWwBxxxbbbx	261922	24

f Why the Improved Estimator is Better

f.1 1. Multi-Factor Heuristic Analysis

- The **basic estimator** only considers immediate win/loss states and a simple heuristic $(i + j - 15)$, which evaluates king positions.
- The **improved estimator** incorporates multiple factors:
  - Weighted king advancement (3x multiplier)
  - Pawn positioning bonuses
  - Path blocking and capturable pawn analysis
  - Pawn quantity advantage
  - King exit proximity bonuses

f.2 2. Weighted King Advancement (3x Multiplier)

The improved estimator **prioritizes king movement**, ensuring that advancing White's king and pushing back Black's king have a more pronounced impact on the evaluation.

f.3 3. Pawn Positional Bonuses

Unlike the basic estimator, which **ignores pawns**, the improved function rewards White pawns for advancing and penalizes Black pawns for progressing.

f.4 4. Path Blocking and Capturable Pawn Analysis

Evaluates **whether pawns block White's king progress** or whether Black's pawns are vulnerable to being jumped over. This directly affects **mobility**, which is crucial in Jumpy3.

f.5 5. Pawn Quantity Advantage

Considers **pawn superiority**, rewarding positions where White has more pawns than Black.

f.6 6. King Exit Proximity Bonuses

- White King $\geq 13 \Rightarrow +50$ bonus (favors near victory states)
- Black King $\leq 2 \Rightarrow$ -50 penalty (punishes near Black victory states)

Ensures the function **strongly favors winning positions**.

f.7 Conclusion

The improved estimator is **superior** because it provides a **more comprehensive and strategic evaluation of board positions**, leading to significantly better decision-making in Jumpy3.

g Example Comparisons (MiniMax vs Alpha-Beta)

Initial Board	Depth	MiniMax Move	Positions evaluated	Estimate	AlphaBeta Move	Positions evaluated	Estimate
xwwxxbxxwBxxWxbb	11	xwwxxbxxwBxxxWbb	3201293	0	xwwxxbxxwBxxxWbb	152741	0
xxwxxxWwBxwxbbbx	9	xxwxxxWxxwwxbbbB	254650	5	xxwxxxWwBxxwbbbx	21617	5
xwwxxxWwBxxxbbbx	9	xwwxxxWxxwxxbbbB	261922	4	xwwxxxWxxwxxbbbB	23705	4

In every case we can see that the number of positions evaluated in AlphaBeta is less than MiniMax by a significant margin.

g.1 Conclusion

Hence, AlphaBeta improves the overall performance of the algorithm by using pruning techniques.

h Example Cases

h.1 Example 1

Input: ‘WwwwxxxxxxxxxbbbB‘
Depth: 2

Algorithm	Output Board Position	Positions Evaluated	Estimate
MiniMax	xwwwWxxxxxxxbbbB	16	0
AlphaBeta	xwwwWxxxxxxxbbbB	7	0
MiniMaxBlack	WwwwxxxxxxxxBbbbx	16	0
MiniMaxImproved	xwwwWxxxxxxxbbbB	16	12

h.2 Example 2

Input: ‘xwwwxxxxxxxxxbbbB‘
Depth: 3

Algorithm	Output Board Position	Positions Evaluated	Estimate
MiniMax	xwwwxxxxxxxxxbbbB	1	100
AlphaBeta	xwwwxxxxxxxxxbbbB	1	100
MiniMaxBlack	xwwwxxxxxxxxxbbbB	1	100
MiniMaxImproved	xwwwxxxxxxxxxbbbB	1	100

h.3 Example 3

Input: ‘xwwxxxwBxbxxbbWx‘
Depth: 5

Algorithm	Output Board Position	Positions Evaluated	Estimate
MiniMax	xwwxxxwBxbxxbbxW	919	100
AlphaBeta	xwwxxxwBxbxxbbxW	234	100
MiniMaxBlack	wwwxxBxxxbxxbbWx	958	100
MiniMaxImproved	xwwxxxwBxbxxbbxW	919	100

h.4 Example 4

Input: ‘xwwxxxWwBxxxbbbx‘
Depth: 9

Algorithm	Output Board Position	Positions Evaluated	Estimate
MiniMax	xwwxxxWxxwxxbbbB	261922	4
AlphaBeta	xwwxxxWxxwxxbbbB	23705	4
MiniMaxBlack	xwwxxxWwBxxbxbbx	261901	1
MiniMaxImproved	xxwwxxWwBxxxbbbx	261922	24

h.5 Example 5

Input: ‘xxwxxxWwBxwxbbbx‘
Depth: 9

Algorithm	Output Board Position	Positions Evaluated	Estimate
MiniMax	xxwxxxWxxwwxbbbB	254650	5
AlphaBeta	xxwxxxWwBxxwbbbx	21617	5
MiniMaxBlack	xxwxxBWwxxwxbbbx	259499	0
MiniMaxImproved	xxwxxxxwBWwxbbbx	254650	54

h.6 Example 6

Input: ‘xwwxxbxxwBxxWxbb‘
Depth: 11

Algorithm	Output Board Position	Positions Evaluated	Estimate
MiniMax	xwwxxbxxwBxxxWbb	3201293	0
AlphaBeta	xwwxxbxxwBxxxWbb	152741	0
MiniMaxBlack	wwwxxbxBxxxxWxbb	3189633	-3
MiniMaxImproved	xwwxxbxxxxwxWBbb	3201293	7