Enhanced MCTS for Sushi Go - Implementation Guide

Table of Contents

1. Overview
2. Prerequisites
3. File Structure
4. Step-by-Step Implementation
5. Running the Enhanced Agent
6. Adding the Pre-Configured JSON File
7. Configuration Options
8. Troubleshooting

Overview

This guide provides complete instructions for implementing the Enhanced Basic MCTS agent for Sushi Go. The enhancements include:

- **Heuristic-Guided Rollouts**: Domain-specific evaluation function for strategic play
- **Determinization**: Handling imperfect information through multiple world sampling
- **Parameter Tuning**: Optimized rollout length, tree depth, and exploration constants

Performance Improvement: Significant win rate increase over baseline vanilla MCTS and random players.

Prerequisites

Required Software

- Java Development Kit (JDK) 8 or higher
- TAG Framework (Tabletop Games Framework)
- IDE (IntelliJ IDEA, Eclipse, or similar) - Optional but recommended

Required Knowledge

- Basic understanding of MCTS algorithm
- Java programming fundamentals
- Familiarity with the TAG framework structure

File Structure

The enhanced MCTS implementation consists of three main files and one heuristic file:

```
TAG-Framework/
└── src/
```

```
└── main/
    └── java/
        └── players/
            ├── readyBasicMCTS/
            │   ├── BasicMCTSPlayer.java (Modified)
            │   ├── BasicMCTSParams.java (Modified)
            │   └── BasicTreeNode.java (Modified)
            └── heuristics/
                └── SushiGoHeuristic.java (New File)
```

Step-by-Step Implementation

Step 1: Create the Sushi Go Heuristic

File: `src/main/java/players/heuristics/SushiGoHeuristic.java`

Purpose: Domain-specific evaluation function for Sushi Go game states.

Instructions:

1. Navigate to the `players/heuristics/` directory
2. Create a new file named `SushiGoHeuristic.java`
3. Copy the complete heuristic code (provided in your documents)
4. Ensure the package declaration matches: `package players.heuristics;`

Key Features:

- Evaluates set completion (Tempura, Sashimi, Dumplings)
- Assesses Maki race positioning
- Evaluates Wasabi-Nigiri combinations
- Considers Pudding strategy (end-game scoring)
- Evaluates Chopsticks utility

Tunable Weights (in the heuristic):

```
private double COMBO_FOCUS = 1.0; // Aggressive combo hunting
private double SAFETY_FOCUS = 1.0; // Safe immediate rewards
private double DENIAL_FOCUS = 0.5; // Block opponents
private double MAKI_PRIORITY = 1.0; // Compete for Maki
private double PUDDING_PRIORITY = 0.7; // Pudding importance
```

Step 2: Modify BasicMCTSParams.java

File: `src/main/java/players/readyBasicMCTS/BasicMCTSParams.java`

Purpose: Add new tunable parameters for heuristic rollouts and determinization.

Instructions:

2.1 Add New Parameter Fields

Add these fields to the class (after existing parameters):

```
// (1) Adding Heuristics Rollouts
public boolean useHeuristicRollouts = true;
public double heuristicRolloutProbability = 0.9;

// (2) Adding determinization
public int nDeterminizations = 5;
```

2.2 Update Constructor

Add parameter registration in the constructor:

```
public BasicMCTSParams() {
    // ... existing parameters ...

    // Add new parameters
    addTunableParameter("useHeuristicRollouts", true);
    addTunableParameter("heuristicRolloutProbability", 0.9);
    addTunableParameter("nDeterminizations", 5);
}
```

2.3 Update _reset() Method

Add parameter retrieval in the _reset() method:

```
@Override
public void _reset() {
    super._reset();
    // ... existing parameter resets ...

    // Reset new params
    useHeuristicRollouts = (boolean)
getParameterValue("useHeuristicRollouts");
    heuristicRolloutProbability = ((Number)
        getParameterValue("heuristicRolloutProbability")).doubleValue();
    nDeterminizations = (int) getParameterValue("nDeterminizations");
}
```

2.4 Adjust Existing Parameters (Optional but Recommended)

Modify default values for optimal performance:

```
public double K = Math.sqrt(2); // UCB exploration constant
public int rolloutLength = 10; // Truncated rollout depth
public int maxTreeDepth = 100; // Effectively no limit
public IStateHeuristic heuristic = new SushiGoHeuristic(); // Use new
heuristic
```

Step 3: Modify BasicMCTSPlayer.java

File: `src/main/java/players/readyBasicMCTS/BasicMCTSPlayer.java`

Purpose: Implement determinization loop for handling imperfect information.

Instructions:

3.1 Replace the _getAction() Method

Replace the entire `_getAction()` method with:

```
@Override
public AbstractAction _getAction(AbstractGameState gameState,
List<AbstractAction> actions) {
    BasicMCTSParams params = getParameters();

    // Track action visit counts across all determinizations
    Map<AbstractAction, Integer> totalVisits = new HashMap<>();
    for (AbstractAction action : actions) {
        totalVisits.put(action, 0);
    }

    // Run multiple determinizations
    for (int d = 0; d < params.nDeterminizations; d++) {
        // Create a new determinised state
        AbstractGameState determinState = gameState.copy(getPlayerID());

        // Search from the deterministic root
        BasicTreeNode root = new BasicTreeNode(this, null, determinState,
rnd);
        root.mctsSearch();

        // Accumulate visit counts for each action
        for (AbstractAction action : actions) {
            BasicTreeNode child = root.children.get(action);
            if (child != null) {
                totalVisits.put(action, totalVisits.get(action) +
child.nVisits);
            }
        }
    }

    // Return action with the most total visits across all determinizations
    AbstractAction bestAction = null;
    int maxVisits = -1;
    for (AbstractAction action : totalVisits.keySet()) {
        if (totalVisits.get(action) > maxVisits) {
            maxVisits = totalVisits.get(action);
            bestAction = action;
        }
    }

    return bestAction;
}
```

3.2 Add Required Import

Ensure you have the necessary imports at the top of the file:

```
import java.util.HashMap;
import java.util.Map;
```

Step 4: Modify BasicTreeNode.java

File: `src/main/java/players/readyBasicMCTS/BasicTreeNode.java`

Purpose: Implement heuristic-guided rollout selection.

Instructions:

4.1 Add selectHeuristicAction() Method

Add this new method to the BasicTreeNode class:

```
private AbstractAction selectHeuristicAction(AbstractGameState
rolloutState) {
    List<AbstractAction> availableActions = player.getForwardModel()
        .computeAvailableActions(rolloutState,
player.parameters.actionSpace);

    if (availableActions.isEmpty()) {
        return null;
    }

    // If only one action, return it
    if (availableActions.size() == 1) {
        return availableActions.get(0);
    }

    // Use epsilon for heuristic rollouts, sometimes use random action for
exploration
    if (rnd.nextDouble() >
player.getParameters().heuristicRolloutProbability) {
        return availableActions.get(rnd.nextInt(availableActions.size()));
    }

    AbstractAction bestAction = null;
    double bestValue = -Double.MAX_VALUE;

    // Evaluate each action by simulating it and using heuristic
    for (AbstractAction action : availableActions) {
        AbstractGameState nextState = rolloutState.copy();
        player.getForwardModel().next(nextState, action.copy());

        // Use heuristic to evaluate the resulting state
        double value = player.getParameters().getStateHeuristic()
            .evaluateState(nextState, player.getPlayerID());

        // Add small noise to break ties
        value = noise(value, player.getParameters().epsilon,
rnd.nextDouble());

        if (value > bestValue) {
            bestValue = value;
            bestAction = action;
        }
    }

    // Fallback to random if something goes wrong
    return bestAction != null ? bestAction :
        availableActions.get(rnd.nextInt(availableActions.size()));
}
```

4.2 Modify rollOut() Method

Update the `rollOut()` method to use heuristic-guided selection:

```
private double rollOut() {
    int rolloutDepth = 0;
    AbstractGameState rolloutState = state.copy();

    if (player.getParameters().rolloutLength > 0) {
        while (!finishRollout(rolloutState, rolloutDepth)) {
            AbstractAction next;

            // Choose rollout action based on parameter policy
            if (player.getParameters().useHeuristicRollouts) {
                // Use heuristic guided selection
                next = selectHeuristicAction(rolloutState);
            } else {
                // Use random rollout selection policy (original behaviour)
                next = randomPlayer.getAction(rolloutState,
                    randomPlayer.getForwardModel().computeAvailableActions(
                        rolloutState,
randomPlayer.parameters.actionSpace));
            }

            advance(rolloutState, next);
            rolloutDepth++;
        }
    }

    // Evaluate final state and return normalized score
    double value = player.getParameters().getStateHeuristic()
        .evaluateState(rolloutState, player.getPlayerID());

    if (Double.isNaN(value))
        throw new AssertionError("Illegal heuristic value - should be a
number");

    return value;
}
```

Running the Enhanced Agent

Method 1: Using Tournament Mode (Recommended for Testing)

1. Set up a tournament configuration file or use the TAG framework's tournament system
2. Add your enhanced agent to the tournament:

```
List<AbstractPlayer> players = new ArrayList<>();
players.add(new BasicMCTSPlayer()); // Your enhanced agent
players.add(new RandomPlayer()); // For comparison
// Add more players as needed
```

3. Run the tournament:

```
# From TAG framework root directory
```

```
java -jar target/TAG-1.0.jar --tournament --game SushiGo --players
BasicMCTS,Random --matches 500
```

Method 2: Single Game Mode

1. Create a main class to run a single game:

```
public class RunSushiGo {
    public static void main(String[] args) {
        GameType gameType = GameType.SushiGo;
        List<AbstractPlayer> players = Arrays.asList(
            new BasicMCTSPlayer(),
            new RandomPlayer(),
            new RandomPlayer()
        );
        Game game = GameType.createGame(gameType, players);
        game.run();
    }
}
```

2. Run the main class from your IDE or command line

Method 3: Using GUI

1. Launch the TAG framework GUI
2. Select "Sushi Go" from the game menu
3. Add "BasicMCTS" as one of the players
4. Configure number of players (2-5)
5. Click "Start Game"

Adding the Pre-Configured JSON File

To use your enhanced MCTS agent with pre-configured optimized parameters in tournaments, you need to create a JSON configuration file and properly reference it in your tournament setup.

Step 1: Create the Player Configuration Directory

Directory: `json/experiments/tournamentplayers/`

Instructions:

1. Navigate to the `json/experiments/` directory in your TAG framework root
2. If the `tournamentplayers` subdirectory doesn't exist, create it
3. This directory will store all your player configuration files

Step 2: Create Your Player Configuration JSON File

File: `sushigo_BasicMCTS_Player.json`

Location: Place this file inside `json/experiments/tournamentplayers/`

Example Configuration:

```
{
  "class": "players.readyBasicMCTS.BasicMCTSPlayer",
  "parameters": {
    "budgetType": "ITERATIONS",
    "budget": 200,
    "rolloutLength": 10,
    "maxTreeDepth": 100,
    "K": 1.41,
    "nDeterminizations": 5,
    "useHeuristicRollouts": true,
    "heuristicRolloutProbability": 0.9,
    "heuristic": {
      "class": "players.heuristics.SushiGoHeuristic"
    }
  }
}
```

Key Parameters to Include:

- `class`: Full package path to your player class
- `budgetType`: Type of computational budget (`"ITERATIONS"`, `"TIME"`, or `"FM_CALLS"`)
- `budget`: Budget amount (e.g., 200 iterations, 1000 milliseconds)
- `rolloutLength`: Depth of rollout simulations
- `K`: UCB exploration constant (typically $\sqrt{2} \approx 1.41$)
- `nDeterminizations`: Number of determinization samples
- `useHeuristicRollouts`: Enable heuristic-guided rollouts
- `heuristicRolloutProbability`: Probability of using heuristic vs random (0.9 = 90%)
- `heuristic`: Reference to your heuristic class

Step 3: Create Your Tournament Configuration File

File: `your_tournament.json` (you can name this anything)

Location: Place in `json/experiments/` directory

Example Tournament Configuration:

```
{
  "gameToRun": "games.sushigo.SushiGoGameState",
  "nPlayers": 4,
  "matchups": [
    {
      "players": [
        "tournamentplayers/sushigo_BasicMCTS_Player.json",
        "players.simple.RandomPlayer",
        "players.simple.RandomPlayer",
        "players.simple.RandomPlayer"
      ]
    }
  ],
  "nRepetitions": 100
```

```
}
```

Important Notes:

- The path to your player config (`"tournamentplayers/sushigo_BasicMCTS_Player.json"`) is **relative to the `json/experiments/` directory**
- You can reference multiple different player configurations in the same tournament
- The `"class"` field in player configs can be used for built-in players like `"players.simple.RandomPlayer"`

Step 4: Directory Structure Summary

Your final directory structure should look like this:

```
TAG-Framework/
├── src/
│   └── main/
│       └── java/
│           └── players/
│               ├── readyBasicMCTS/
│               │   ├── BasicMCTSPlayer.java
│               │   ├── BasicMCTSParams.java
│               │   └── BasicTreeNode.java
│               └── heuristics/
│                   └── SushiGoHeuristic.java
└── json/
    └── experiments/
        ├── tournamentplayers/
        │   └── sushigo_BasicMCTS_Player.json  ← Player config here
        └── your_tournament.json                ← Tournament config here
```

Step 5: Running Tournaments with Your Configuration

From Command Line:

```
# Navigate to TAG framework root
cd TAG-Framework

# Run tournament using your configuration
java -jar target/TAG-1.0.jar --tournament
json/experiments/your_tournament.json
```

From IDE (if using tournament runner class):

```
String tournamentConfigPath = "json/experiments/your_tournament.json";
// Load and run tournament
```

Creating Multiple Player Configurations

You can create multiple JSON files with different parameter settings for comparison:

Example: Conservative Configuration (`sushigo_BasicMCTS_Conservative.json`):

```
{
  "class": "players.readyBasicMCTS.BasicMCTSPlayer",
  "parameters": {
    "budget": 100,
    "nDeterminizations": 3,
    "heuristicRolloutProbability": 0.8
  }
}
```

Example: Aggressive Configuration (`sushigo_BasicMCTS_Aggressive.json`):

```
{
  "class": "players.readyBasicMCTS.BasicMCTSPlayer",
  "parameters": {
    "budget": 300,
    "nDeterminizations": 10,
    "heuristicRolloutProbability": 0.95
  }
}
```

Then compare them in a tournament:

```
{
  "matchups": [
    {
      "players": [
        "tournamentplayers/sushigo_BasicMCTS_Conservative.json",
        "tournamentplayers/sushigo_BasicMCTS_Aggressive.json",
        "players.simple.RandomPlayer",
        "players.simple.RandomPlayer"
      ]
    }
  ]
}
```

Configuration Options

Adjusting Performance vs Speed Trade-offs

For Better Performance (Slower):

```
public int nDeterminizations = 10; // More world samples
public double heuristicRolloutProbability = 0.95; // More strategic play
public int rolloutLength = 15; // Deeper lookahead
```

For Faster Execution (Less Accurate):

```
public int nDeterminizations = 3; // Fewer world samples
public double heuristicRolloutProbability = 0.8; // More randomness
public int rolloutLength = 5; // Shallow lookahead
```

Disabling Specific Enhancements

Test Without Heuristic Rollouts:

```
public boolean useHeuristicRollouts = false;
```

Test Without Determinization:

```
public int nDeterminizations = 1; // Single determinization
```

Test With Original Parameters:

```
public int rolloutLength = Integer.MAX_VALUE; // Full-depth rollouts
public double K = 1.0; // Different exploration
```

Budget Configuration

Control computational resources:

```
// Time-based budget (milliseconds per move)
params.budgetType = BUDGET_TIME;
params.budget = 1000; // 1 second per move

// Iteration-based budget
params.budgetType = BUDGET_ITERATIONS;
params.budget = 500; // 500 MCTS iterations

// Forward model call budget
params.budgetType = BUDGET_FM_CALLS;
params.budget = 1000; // 1000 simulation steps
```

Troubleshooting

Common Issues and Solutions

Issue 1: "Cannot find SushiGoHeuristic"

Solution:

- Verify the file is in `players/heuristics/` directory
- Check package declaration: `package players.heuristics;`
- Rebuild the project to ensure compilation

Issue 2: "NullPointerException in selectHeuristicAction"

Solution:

- Ensure heuristic parameter is set to `new SushiGoHeuristic()` in BasicMCTSParams
- Check that `getStateHeuristic()` returns non-null value

Issue 3: Agent Takes Too Long Per Move

Solution:

- Reduce `nDeterminizations` (try 3 instead of 5)
- Decrease `rolloutLength` (try 5 instead of 10)

- Set a time budget: `params.budgetType = BUDGET_TIME; params.budget = 500;`

Issue 4: Agent Performs Poorly

Solution:

- Verify heuristic weights are appropriate for your game scenarios
- Ensure `useHeuristicRollouts = true`
- Check that `heuristicRolloutProbability` is high (0.9 recommended)
- Increase `nDeterminizations` for more robust decisions

Issue 5: Compilation Errors

Solution:

- Ensure all imports are present:

```
import java.util.HashMap;
import java.util.Map;
import players.heuristics.SushiGoHeuristic;
```

- Check Java version compatibility (JDK 8+)
- Clean and rebuild the project

Issue 6: Agent Plays Randomly Despite Heuristics

Solution:

- Verify `useHeuristicRollouts = true` in parameters
- Check that `selectHeuristicAction()` is being called in `rollOut()` method
- Add debug print statements to confirm heuristic evaluation is occurring

Issue 7: JSON Configuration Not Loading

Solution:

- Verify JSON file is in correct directory: `json/experiments/tournamentplayers/`
- Check JSON syntax is valid (use a JSON validator)
- Ensure tournament JSON references player with relative path: `"tournamentplayers/filename.json"`
- Verify the `"class"` field matches your player's full package path
- Check that parameter names in JSON exactly match those defined in `BasicMCTSParams.java`

Issue 8: Tournament Fails to Find Player Configuration

Solution:

- Ensure path in tournament JSON is relative to `json/experiments/` directory
- Don't include `json/experiments/` in the path - start with `tournamentplayers/`

- Check file extension is `.json` in both filename and reference
- Verify file permissions allow reading

Verification Checklist

Before running experiments, verify:

- [ ] `SushiGoHeuristic.java` is in correct directory and compiles
- [ ] `BasicMCTSParams.java` has all three new parameters added
- [ ] `BasicMCTSPlayer.java` implements determinization loop
- [ ] `BasicTreeNode.java` has `selectHeuristicAction()` method
- [ ] `rollOut()` method uses heuristic-guided selection
- [ ] Parameters are set to recommended values
- [ ] Project compiles without errors
- [ ] Agent can be instantiated and added to games
- [ ] Budget settings are appropriate for testing environment
- [ ] Player configuration JSON file created in `json/experiments/tournamentplayers/`
- [ ] Tournament JSON file created in `json/experiments/`
- [ ] Tournament JSON correctly references player configuration with relative path
- [ ] JSON files have valid syntax and correct parameter names
- [ ] Tournament runs successfully with enhanced agent