RICOCHET RIVALS

Introduction to Artificial Intelligence

Dinglasa, Roanne Maye B.

Espartero, Rasty C.

Mahayag, David Geisler M.

Placente, Yesa V.

JUNE 2025

I. Introduction and its Background

RICOCHET RIVALS is a competitive 2D arena shooter where two opponents — either player versus player or player versus AI — compete to outwit and outlast each other by launching bouncing projectiles through a shifting field of obstacles.

Each participant has **three lives** and must strategically fire projectiles that can bounce off walls up to a maximum number of times depending on charge time. Players can shoot at any moment without turn limits, but are limited by an interval, adding intensity and unpredictability to each match.

Unlike traditional shooting or pong-like games, RICOCHET RIVALS introduces five rows of dynamic, vertically moving walls that constantly reshape the battlefield. Success depends not just on aim but on mastering timing, bounce angles, and prediction under pressure.

To create a richer and more challenging experience, the game integrates three distinct AI difficulties that mirror the evolution of real-world AI technologies:

- **Easy AI** uses a basic **Monte Carlo simulation**, firing with minimal calculation.

- **Medium AI** employs a **Monte Carlo simulation** approach, testing several random shot scenarios before deciding on the most promising one.

- **Hard AI** utilizes a combination of the **Monte Carlo method and precise raycasting**, calculating optimized angles and reflections to strike the opponent with a heavily deadly accuracy.

This layered AI design allows players to experience the growing complexity of AI thinking — from random behavior, to probabilistic planning, to ruthless geometric precision — within a single, fast-paced competitive environment.

II. Game Proper

a. Objective of the Game

The main objective of the game is to eliminate the opposing player by successfully landing **3 hits** using bouncing projectiles, reducing their **lives from 3 to 0**.

Players must **strategically use the bounce mechanics**, **time their shots carefully**, and **anticipate obstacles** as they fire across a field of **five constantly moving vertical walls**. Precision, timing, and prediction are key to victory.

Victory Conditions:

- A player wins by reducing the opponent’s lives from 3 to 0.
- If time runs out (5-minute match limit), the player with **more remaining lives** wins.
- If both players are tied in lives, the winner is determined by the player who **fired the fewest shots** throughout the match.

b . Discuss the Rule of the Game

- Each player starts with **3 lives**.
- Players **cannot move**; their only offensive and defensive tool is shooting.
- **Projectiles bounce up to 3 times** by default before disappearing.
- Players have an unlimited number of shots, but firing has a limiting interval: **2 seconds per shot.**
- **Shots can collid**e mid-air and cancel each other out with an explosion.
- **Four rows of randomly sized vertical walls** move up and down in the center of the arena, blocking or deflecting shots.
- If a projectile hits the opposing player, they **lose one life.**
- The match has a **5-minute timer.**
- In the final **30 seconds**, the game enters Hyper Shooting Mode:
    - Shots bounces are increased up to 8
    - Firing interval is reduced to **1 second**
- In case of a draw in lives at the end of 3 minutes, **the player with the least number of shots fired wins.**

c. Game Modes

- **Player vs AI (PvAI)** – Player faces one of **three AI difficulty levels**:
    - **Easy AI**: Basic Monte Carlo simulation that is slow and inaccurate
    - **Medium AI**: Uses Monte Carlo simulation to test several random shots and pick the best
    - **Hard AI**: Uses a combination of Monte Carlo simulation and raycasting to calculate optimal shots with near-perfect accuracy

III. Implementation of AI Algorithm

a. AI Intelligence Implemented in the Game

RICOCHET RIVALS features a three-tiered AI system that progressively showcases smarter behaviors by combining projectile simulation and geometry-based prediction. All AI opponents in the game use variations of simulation-based decision-making, but differ in intelligence, accuracy, and response time.

b. Easy AI – Simplified Monte Carlo Simulation

Approach: Basic Monte Carlo-inspired AI

Description:

This AI simulates a **small number** of random projectile angles and fires one at the best route. It does not track wall timing or bounce accuracy closely.

Why Chosen:

- Mimics an inexperienced player with delayed and uncertain shots.
- Introduces new players to the core mechanics without overwhelming them.

How It Works:

- Simulates 3–5 random angles.
- Chooses one without deep evaluation.

c. **Medium AI – Optimized Monte Carlo Simulation**

Approach: Refined Monte Carlo simulation with evaluation

Description:

The AI simulates **multiple bounce shots** based on current wall layouts and picks the best-performing angle after evaluating outcomes (e.g., likelihood of hit).

Why Chosen:

- Balanced difficulty that feels human-like.
- Provides realistic challenge through strategic trial-based behavior.

How It Works:

- ○ Simulates 10–20 possible angles.

    - Tracks which shots come closest to hitting the player within bounce limits.
    - Chooses the one with the best success rate.
    - Adjusts shot power (bounces) and timing based on situation.

d. Hard AI – Monte Carlo + Raycasting Precision

Approach: Advanced AI using targeted raycasting within simulations

Description:

The Hard AI uses **full-scale projectile prediction** — combining Monte Carlo simulation **with raycasting techniques** to test high-precision bounces. It looks for direct kill shots and reacts instantly if one is possible.

Why Chosen:

- Provides a true "sniper-level" challenge.
- Reflects intelligent decision-making through accurate angle calculation.
- Forces players to block, fake, and think strategically.

How It Works:

1. Maps wall and player positions.
2. Runs raycasting-based bounce simulations within allowed bounce count.
3. Prioritizes guaranteed hits or intercepts.

4. In **Hyper Mode**, it fires immediately at full power.

Easy AI – Basic Monte Carlo (Light Evaluation)

```
function EasyBehavior()
    aimDirection ← normalize(player.position - ai.position)
    bestScore ← -infinity
    bestDirection ← aimDirection

    repeat 5 times:
        randomAngle ← random number between -45° and 45°
        testDirection ← rotate(aimDirection, randomAngle)

        hit ← simulateRaycast(from ai.position in testDirection)

        if hit exists:
            if hit.target is Player:
                score ← 10  // Best outcome
            else:
                score ← -1  // Hit wall or obstacle
        else:
            score ← 1  // Free space, no hit
        if score > bestScore:
            bestScore ← score
            bestDirection ← testDirection

    rotate AI to face bestDirection
    shootBullet in bestDirection
end function
```

Medium AI – Monte Carlo with Evaluation

```
function MediumBehavior()
    if shoot cooldown is active:
        return

    log "Medium AI calculating..."

    bestScore ← -infinity
    bestAngle ← 0

    baseAngle ← angle from AI firePoint to player position
```

```
// ----- Monte Carlo Simulation -----
for offset from -10 to 10:
    angle ← baseAngle + (offset × 2°)
    direction ← vector from angle

    origin ← firePoint.position + (direction × 0.1)  // avoid self-collision
    hits ← performRaycastAll(origin, direction, range = 50)

    score ← 0

    for each hit in hits:
        if no hit.collider:
            continue

        if hit.collider is Player:
            score += 50  // high reward

        else if hit.collider is Wall or Boundary:
            score -= 20  // blocked

        else:
            score += 10  // minor hit (e.g., floor)
            break  // only consider the first object hit


        if score > bestScore:
            bestScore ← score
            bestAngle ← angle
    // ----- End Simulation -----


    bestDirection ← vector from bestAngle
    FireBullet(bestDirection)
end function
```

Hard AI – Monte Carlo + Raycasting Prediction

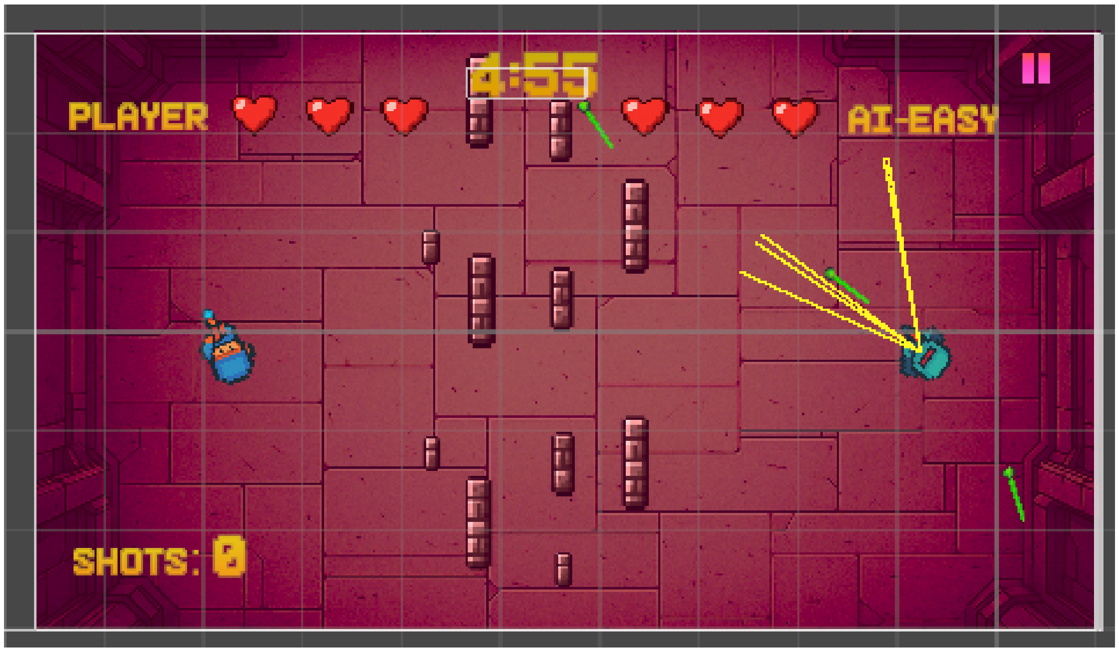
```
function HardBehavior()
    if not ready to shoot:
        return

    bestScore ← -infinity
    bestAngle ← 0

    baseAngle ← angle from AI to player

    for angleOffset from -45° to +45°:
        angle ← baseAngle + angleOffset
        direction ← vector from angle

        score ← SimulateCombatBounce_TIME(origin = AI.firePoint, direction, remainingBounces = 3,
elapsedTime = 0)

        if score > bestScore:
            bestScore ← score
            bestAngle ← angle
    finalDirection ← vector from bestAngle
    FireBullet(finalDirection)
end function
```

Common Methods (Implemented these in the engine logic)

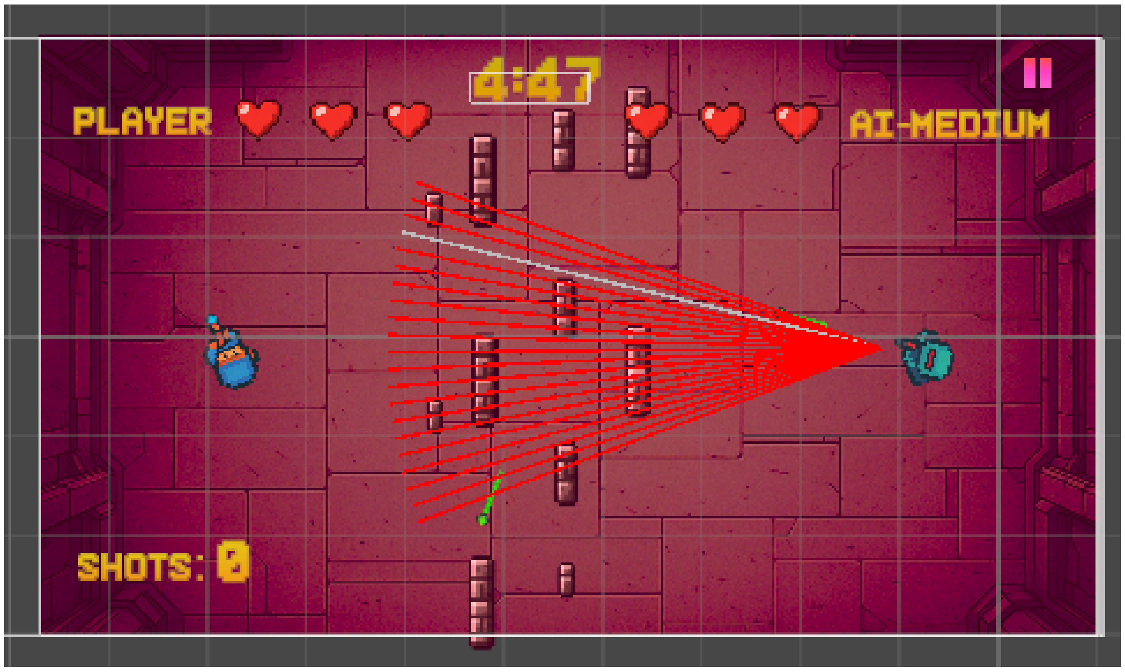
```
void FireProjectile(float angle, float chargeTime) {
    // Instantiate projectile, set bounce count based on charge time
}

bool SimulateBouncePath(float angle, float chargeTime) {
    //  imulation to test if projectile reaches player
    return true; // or false
}

bool SimulateRaycastBounces(float angle, float chargeTime, int maxBounces) {
    // Recursive raycast logic with reflection physics
    return true; // if target can be hit within bounces
}
```

MONTE CARLO SIMULATION

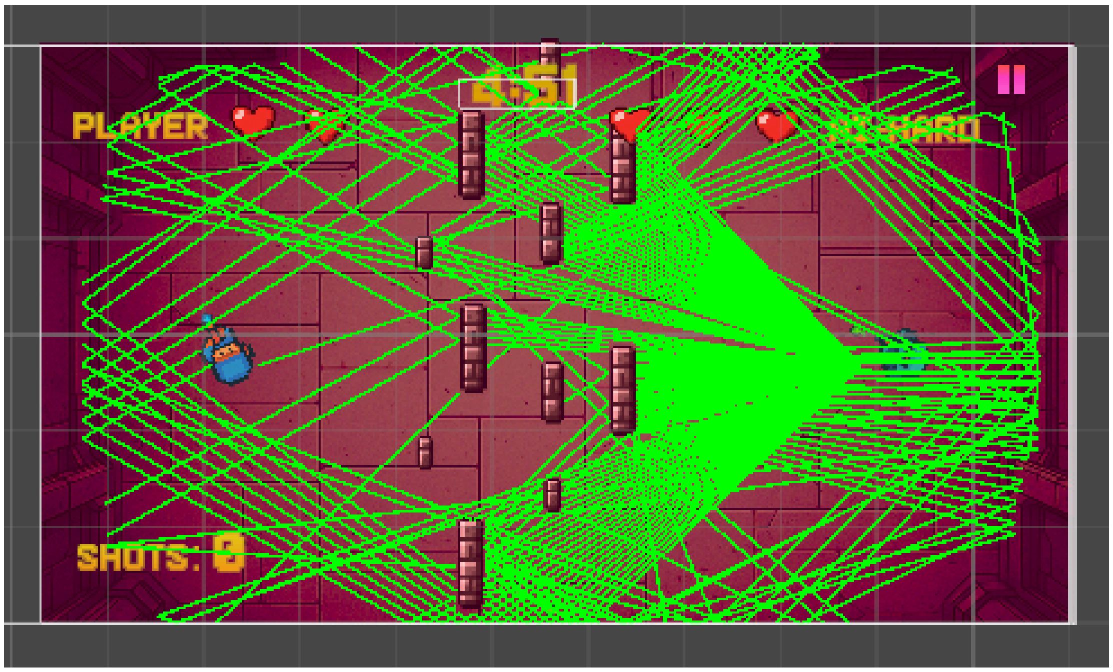
Easy Behavior



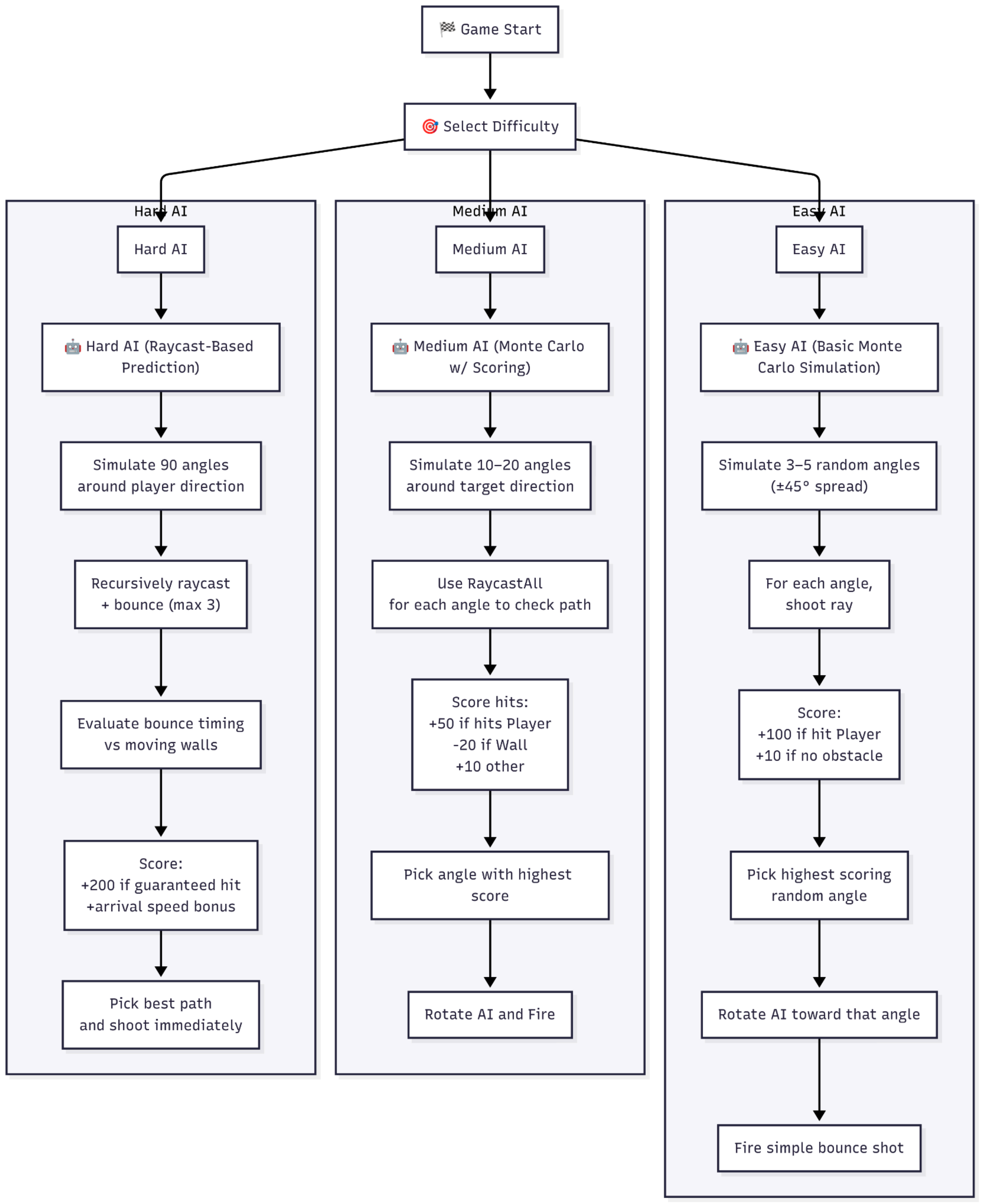
Medium Behavior



Hard Behavior



System Architecture



🏁 Game Start
🎯 Select Difficulty
Hard AI
Hard AI
🤖 Hard AI (Raycast-Based Prediction)
Simulate 90 angles around player direction
Recursively raycast + bounce (max 3)
Evaluate bounce timing vs moving walls
Score: +200 if guaranteed hit +arrival speed bonus
Pick best path and shoot immediately
Medium AI
Medium AI
🤖 Medium AI (Monte Carlo w/ Scoring)
Simulate 10–20 angles around target direction
Use RaycastAll for each angle to check path
Score hits: +50 if hits Player -20 if Wall +10 other
Pick angle with highest score
Rotate AI and Fire
Easy AI
Easy AI
🤖 Easy AI (Basic Monte Carlo Simulation)
Simulate 3–5 random angles (±45° spread)
For each angle, shoot ray
Score: +100 if hit Player +10 if no obstacle
Pick highest scoring random angle
Rotate AI toward that angle
Fire simple bounce shot

The AI system in Ricochet Rivals is structured across three difficulty levels—Easy, Medium, and Hard—each utilizing an increasingly complex version of the Monte Carlo simulation method to determine shot direction and behavior. When the game begins, players select a difficulty, and the chosen AI employs a behavior model tailored to its level.

The Easy AI uses a basic Monte Carlo simulation. It randomly simulates 3 to 5 angles within a ±45° spread from its forward direction. For each angle, it casts a ray and assigns a score—+100 if the shot hits the player, and +10 if the path is clear. It selects the angle with the highest score and fires a simple bouncing projectile. This approach makes the AI feel like a beginner: reactive, slow, and inaccurate, which suits new players.

Medium AI takes a more refined Monte Carlo approach. It simulates 10 to 20 angles around the player's position and uses RaycastAll to evaluate each trajectory. The AI scores each result: +50 for hitting the player, -20 for hitting a wall, and +10 for other outcomes. It then chooses the angle with the highest score and rotates to fire in that direction. This gives the impression of a more realistic and human-like opponent that evaluates options before acting.

The Hard AI combines Monte Carlo simulation with raycasting-based precision. It simulates 90 angles and recursively raycasts each path, allowing for up to 3 bounces. It evaluates each shot based on whether it guarantees a hit, how it interacts with moving walls, and how quickly it would reach the opponent. Guaranteed hits earn +200 points with additional bonuses for arrival speed. The AI selects the optimal path and fires immediately. This mode feels like a strategic sniper, using exact calculations to dominate the game.

Together, these three AI levels show a clear progression from randomness to intelligent decision-making, creating a scalable challenge that mirrors real-world AI evolution.

GAME FINAL CHECKING

Group 3

Section: BSCS 3 - 2

Group Member:

- Dinglasa, Roanne Maye B.
- Espartero, Rasty C.
- Mahayag, David Geisler M.
- Placenta, Yesa V.

Contributions:

Member	Document	Program
Dinglasa, Roanne Maye B.	20%	30%
Espartero, Rasty C.	20%	30%
Mahayag, David Geisler M.	30%	20%
Placenta, Yesa V.	30%	20%
TOTAL	**100%**	**100%**

REFERENCES:

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., & Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, *529*(7587), 484–489. https://doi.org/10.1038/nature16961

Gelly, S., & Silver, D. (2011). Monte-Carlo tree search and rapid action value estimation in computer Go. *Artificial Intelligence,* *175*(11), 1856–1875. https://doi.org/10.1016/j.artint.2011.03.007