Robo-maze Blast Report

Dogà Sara
Widmann Lukas
Askar Sami

Abstract

Bomberman + Evolutionary algorithmens + Tournament Arc goes brr

ACM Reference Format:
Dogà Sara, Widmann Lukas, and Askar Sami. 2025. Robo-maze Blast Report. In . ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

1.1 The game

Robo Maze Blast, created in 2008 by Kai Ritterbusch and Christian Lins, is a clone of the Bomberman game. Also known as Dynablaster, it is a strategy maze-based video game franchise originally developed by Hudson Soft in 1985.

The general goal of Bomberman is to complete the levels by strategically placing bombs in order to kill enemies and destroy blocks. Some blocks in the path can be destroyed by placing bombs near it, and as the bombs detonate, they will create a burst of vertical and horizontal lines of flames. Except for indestructible blocks, contact with the blast will destroy anything on the screen.

1.2 Our Goal

The aim of our project is to explore the efficiency of different Genetic Algorithms to develop 3 agents with strategic competence in the Robo Maze Blast scope, and to compare them by making the agents fight against each other and observe which agent outlives the others more frequently.

2 Background

2.1 Genetic Algorithms

Genetic Algorithms (GA) are optimization algorithms inspired by the process of natural selection and biological evolution. They are widely used to solve complex optimization and search problems in various domains. One of those domains is the gaming side as in this paper. Due to constrained optimization (e.g., state/action of the game), Genetic Algorithms are a perfect choice for this task [**?**]: *"Genetic Algorithms (GAs) were selected for their ability to handle complex combinatorial optimization problems [...] and to encode domain-specific constraints"* (Section 1).

Conference'17, Washington, DC, USA
© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

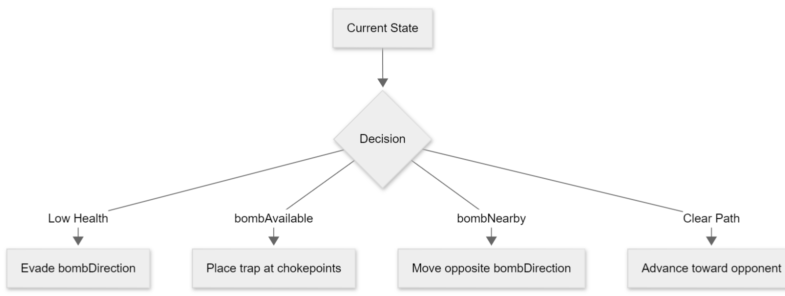

Figure 1: A diagram on the possible actions of a player

The core steps of a typical genetic algorithm can be described as follows:

- **Population Base**: Initialize a population from valid chromosomes, i.e. a set of strings that encodes any possible solution. Usually, the initial population is chosen randomly.
- **Evaluation**: Each population solution is evaluated on the basis of a predetermined fitness function.
- **Selection**: Reproductive opportunities are allocated to the chromosomes that represent a better solution to the target problem, and such solutions are selected to form a 'mating pool' for the next generation.
- **Crossover and Mutation**: The selected individuals are then combined to produce offspring by exchanging genetic material. Sometimes small changes can happen in the genetic material, such as bit flips. All of this ensures good exploration of the solution space and diversity.

These steps are repeated for a number of times until an ending criterion is reached.

2.2 Robo Maze Blast's Default AI Agent

The game has its own AI agents that will play against the player in the absence of in-real-life adversaries. They share a common behavior and reasoning that can be visualized with the finite-state machine, as shown in Figure 3.

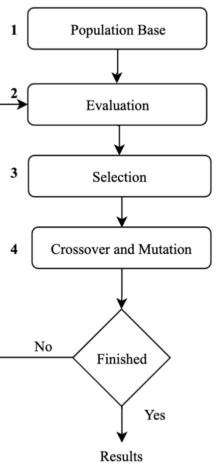

Figure 2: A diagram on the steps of a genetic algorithm

3 Fine-tuning agent behaviors with jenetics

3.1 Differential Evolution

3.2 Agent Behavior

3.3 The Reward Metrics

The fitness function determines which agent behaviors are rewarded, and which are penalized.

Table 1: Agent Reward and Penalty Values

Action	Points
Movement	+1 (per step)
Place Bomb	+75 (per bomb)
Blow Wall	+150 (per wall)
Kills	+750 (per player)
Death	-500
Suicide	-500
Win without kills	+200
Win with kills	+1000

4 Supervised Learning with Jenetics

4.1 Introduction

My objective was to create an AI player using a genetic algorithm (GA) trained on human gameplay data, with an exploration of skill transferability across gaming domains. This approach was motivated by competitive gaming frameworks where player strength is quantified through Elo rating systems [Elo78]. As an experienced fighting game player, my current Tekken 8 Elo rating stands at **1940** (profile: https://wank.wavu.wiki/player/3nyHJQr8Gq6Q, accessed August 8, 2025). While acknowledging that direct skill transfer between a 3D fighting game (Tekken) and a 2D grid-based strategy game (Robo Maze Blast) may be limited, this project tests the hypothesis that:

Hypothesis: Strategic decision-making patterns in high-Elo players exhibit domain-agnostic qualities, such that:

$$\text{Elosource} \geq 1900 \xrightarrow{\text{transfer}} \text{AItarget} > \text{AI}_{\text{baseline}} + \Delta$$

where Δ represents measurable skill advantage in Robo Maze Blast.

4.2 Jenetics

Jenetics is an open-source Java library that provides a genetic Algorithm (GA) framework for solving optimization problems. It abstracts biological evolution principles, such as selection, crossover, and mutation, into reusable software components, enabling users to evolve solutions without implementing a GA from scratch.

4.3 Jenetics Framework

We selected it for supervised learning because:

- **Prior experience**: We'd successfully used it in previous labs
- **Java compatibility**: Integrated smoothly with our game codebase
- **Rapid adjustments**: Changing parameters takes minutes instead of hours
- **Easy versioning**: Git checkpoints for different configurations

Key Features We Utilized:

- **Evolutionary engine**: Automatically handles generations and survival mechanics
- **Domain flexibility**: Worked directly with our game strategy optimization
- **Pre-built operators**:
  - Tournament selection (picks winners from random groups)
  - Gaussian mutation (makes small, smart adjustments)
  - Single-point crossover (combines parent solutions)

Agile Implementation: The library let us quickly test configurations by reducing parameters:

- Population size: 100 $\longrightarrow$ 500
- Mutation rate: 0.1 $\longrightarrow$ 0.05
- Training generations: 50 $\longrightarrow$ 200

This enables an iterative improvement cycles:

0. Gather human data
1. Train initial AI
2. Identify weaknesses
3. Adjust parameters
4. Retrain (under 30 minutes)

Reference: Jenetics User Manual [Wil24].

4.4 Implementing Gameplay Recording: Code Changes for Data Collection

To train the AI using Jenetics, we first needed good quality gameplay data. The simplest way to get this data was to record what a human player does during a game - both their actions and the game situation at that moment. This required changes to the Player class, which led to creating the new `RecordablePlayer` class. The main changes are shown in Algorithm 1 and include:

Algorithm 1: RecordablePlayer Modifications

```
  Class RecordablePlayer extends Player
1 New Data: recordings = []
  isRecording = false        // Recording on/off switch
2 Key Changes:
3 move(dx, dy): super.move(dx, dy)
  if isRecording then
      LogMovement(dx, dy)            // Records moves
4 placeBomb(): super.placeBomb()
  if isRecording then
      LogAction(BOMB)                // Records bombs
5 New Methods:
6 LogMovement(dx, dy): Convert dx/dy to direction
  LogAction(direction)
7 LogAction(action): state ← [normX, normY, bombAvail,
    bombNear]
  recordings ← recordings + (state, action)
8 Save(filename): Write all recordings to file
```

- **Recording trigger**: Making the game log actions when players move or place bombs
- **State capture**: Saving player position and bomb status in simple numbers
- **Data export**: Saving all records to a CSV file for AI training

Example from actual gameplay: Imagine this situation during a game:

- Player is at position (7, 12) in a 20x20 grid
- Player can place a bomb (has bombs available)
- There's a bomb nearby above the player

The `RecordablePlayer` would save this information as:
`0.35, 0.60, 1.0, 1.0, -0.4, 5`

Value	Meaning
0.35	Player is 35% across the level (X position)
0.60	Player is 60% down the level (Y position)
1.0	Bomb available
1.0	Bomb nearby
-0.4	Bomb is above player
5	Player placed a bomb

This simple number format allows the AI to learn from many such records. In the future I would collect a sample trainings data set to train

4.5 How to add Citations and a References List

You can simply upload a `.bib` file containing your BibTeX entries, created with a tool such as JabRef. You can then cite entries from it, like this: [Gre93]. Just remember to specify a bibliography style, as well as the filename of the `.bib`. You can find a video tutorial here to learn more about BibTeX.

If you have an upgraded account, you can also import your Mendeley or Zotero library directly as a `.bib` file, via the upload menu in the file-tree.

4.6 Good luck!

We hope you find Overleaf useful, and do take a look at our help library for more tutorials and user guides! Please also let us know if you have any feedback using the **Contact us** link at the bottom of the Overleaf menu — or use the contact form at https://www.overleaf.com/contact.

References

[Elo78] Arpad E. Elo. *The Rating of Chessplayers, Past and Present.* Arco Publishing, 1978. Original Elo system formulation.
[Gre93] George D. Greenwade. The Comprehensive Tex Archive Network (CTAN). *TUGBoat*, 14(3):342–351, 1993.
[Wil24] Franz Wilhelmstötter. *Jenetics User Manual (Version 8.2.0)*, 2024. Accessed: 07.08.2025.