Makarena: Deducción evolutiva de funciones de evaluación aplicadas a *Minimax*

Propuesta de tesis

Santiago, Chile
2022-07-03
v1.0.0

Índice

1. Introducción

El concepto de *máquinas que aprenden* fue propuesto por primera vez por *Alan Turing* en 1950 con una simple pregunta: «¿Pueden las máquinas pensar?» [2]. Siguiendo esa pregunta, no es descabellado preguntarse: «¿Puede un computador jugar como una persona?»
El inicio del estudio de *teoría de juegos* se le atribuye a *von Neumann* en su publicación de 1928 [1] y desde entonces se ha desarrollado como un área de investigación. Entre los avances de la teoría de juegos se encuentra el desarrollo de algoritmos capaces de idear estrategias similares a las de los jugadores humanos, llamaremos a estos «*jugadores artificiales*».
En este documento nos centraremos en un algoritmo en particular, *Minimax*, debido a que ha sido ampliamente estudiado y utilizado [1, 3, 10, 13, 17]. Sin embargo, este algoritmo depende de una *función de evaluación* que otorga un valor numérico a cada estado del juego para discriminar qué jugada es «mejor» que otra, pero esta función se desprende de estudiar en detalle el juego [4] o por «tanteo» dado que la función de evaluación es **específica del dominio**[19]. Este trabajo propone una ***manera automática*** *de encontrar la función de evaluación del estado del juego sin importar el dominio de éste.*

2. Contexto

Esta sección introducirá varios conceptos necesarios para plantear el problema, y para esto se dividirán las definiciones en dos partes: *Teoría de juegos* (donde nos centraremos en juegos en los que los jugadores tomen turnos para jugar) y *Algoritmos evolutivos.*

2.1. Teoría de juegos

2.1.1. Función de evaluación

Definición 2.1 (**Recurso**[1])**.** Se le dirá recurso a cada elemento en el juego que acerque a un jugador a una jugada óptima. Formalmente, diremos que un *recurso* es un cambio en el estado del juego que incrementa las probabilidades de que el jugador que lo obtiene gane la partida.
Ejemplo: En el ajedrez a cada pieza se le otorga un valor, siendo las «mejores piezas» las que tienen un valor más alto. La pieza con valor más bajo es el *peón* con un valor de 1; así podemos definir el valor de cada pieza en base al valor de un peón (diciendo que una pieza **X** tiene un valor equivalente a x *peones*). En este caso diremos que los recursos son los peones.

Se le dirá **función de evaluación**[16] $v_i(s)$ a la función que determina la cantidad de recursos que tiene el jugador i en el estado s. A su vez, el **estado del juego** se define como el par a_i, a_{-i} que representa las acciones de los jugadores que llevan a una configuración específica del juego (un ejemplo de configuración puede ser la posición y cantidad de piezas en el tablero).

2.1.2. Juegos de suma cero

Los ***juegos de suma cero*** [18] (*zero sum games*) son una representación matemática usada en *teoría de juegos* y *teoría económica* para describir una situación donde existen dos «jugadores» que se enfrentan entre sí. Un *juego es de suma cero* si la ventaja que lleva un jugador es igual a la pérdida del otro.
Los *juegos de suma cero* son un tipo de juegos de suma constante donde la suma de ganancia y pérdida es igual a cero.
En un juego de suma cero, todos los jugadores perciben la misma ganancia de cada recurso. Para entender esto podemos considerar un juego con tres estados: empate, ganador y perdedor, con puntajes 0, 1 y -1 respectivamente.

- **Empate:** Ningún jugador tiene ventaja; ambos jugadores tienen 0 puntos.
- **Ganador:** El jugador ganó, por lo que tiene 1 punto; el otro jugador pierde así que tiene -1 puntos.
- **Perdedor:** El jugador perdió, por lo que tiene -1 puntos; el otro jugador ganó así que tiene 1 punto.

De esta forma, para todo estado del juego, la suma de los puntajes es 0.

[1] Término propio.

2.1.3. *Minimax*

Minimax [3] (*MM*) es un algoritmo de decisión utilizado en múltiples ámbitos para minimizar la pérdida posible para el «peor caso» (*maximum loss*). Se formuló originalmente para juegos de suma cero de n jugadores, pero se ha extendido a juegos más complejos y a problemas de decisión con incertidumbre.

Definición 2.2 (**Valor *Maximin***)**.** El valor *maximin* de un estado del juego es la máxima cantidad de recursos que puede obtener un jugador, sin conocer las jugadas de su oponente.
Formalmente [17]:

$$\underline{v_i} = \max_{a_i} \min_{a_{-i}} v_i(a_i, a_{-i})$$

Donde:

- i es el índice del jugador actual.
- $-i$ representa a el conjunto de todos los jugadores oponentes ($-\text{i} \implies \text{j} \in \mathcal{J} \land \text{j} \neq \text{i}$ con $\mathcal{J}$ el conjunto de todos los jugadores).
- a_i es la jugada del jugador i.
- a_{-i} son las jugadas de todos los oponentes.
- v_i es la función de evaluación del estado del juego en el turno del jugador i.

Definición 2.3 (**Valor *Minimax***)**.** El valor *minimax* de un estado del juego es la mínima cantidad de recursos que un oponente puede forzar al jugador a perder, sin saber las jugadas del jugador.
Formalmente:

$$\overline{v_i} = \min_{a_{-i}} \max_{a_i} v_i(a_i, a_{-i})$$

Nota: Las definiciones de $\underline{v_i}$ y $\overline{v_i}$ se proveen solamente para mantener la formalidad de la función presentada a continuación.

Definición 2.4. Se define la función MM[2] para un estado del juego s como:

$$MM_i(s) = \begin{cases} v_i(s) & \text{si } s = F \\ \underline{v_i}(s) & \text{si es el turno del jugador } i \\ \overline{v_i}(s) & \text{si es el turno del oponente } -i \end{cases}$$

Con F representando el estado final del juego.

Definición 2.5 (**Algoritmo *Minimax***)**.** Sea `t` un árbol de estados del juego, donde cada nodo del árbol es un estado del juego evaluado con la función de evaluación v_i, y donde cada arco del árbol es una jugada de un jugador (para simplificar diremos que cada jugada representa un turno y que los jugadores toman turnos alternados, primero i y luego $-i$). Luego, cada nodo tendrá tantos hijos como jugadas pueda realizar cada jugador en su turno.
Se define el algoritmo *minimax* para un árbol de estados `t` como:

```
fun minimax(t: StateTree, player: Player in [i, -i]): Double =
  if (t.isTerminal()) {
    t.value
  } else if (player == -i)  {
    t.children.maxOf { child -> minimax(child, i) }
  } else {
    t.children.minOf { child -> minimax(child, -i) }
  }
```

Para entender mejor el algoritmo, considere el árbol de estados de la fig. 1. Cada nodo del árbol representa un estado del juego, y cada arco representa una jugada de un jugador. Con esto podemos definir una función de evaluación arbitraria donde el valor de cada nodo es la suma de puntajes acumulados en el camino que conecta la raíz del árbol con el nodo. Aplicando esta función de evaluación al algoritmo *minimax* se toman las decisiones indicadas por las flechas rojas en la figura, tomando el mínimo o máximo dependiendo de quién sea la jugada. Finalmente se escoge la secuencia de jugadas marcadas en verde.

[2] O *función Minimax*; se prefiere MM para evitar confusiones con el *algoritmo Minimax*

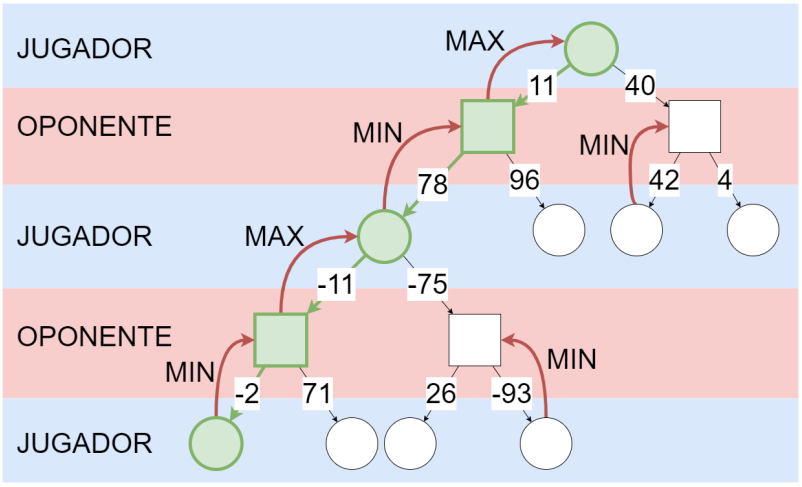

Figura 1: Árbol de estados de un juego.

2.2. Algoritmos evolutivos

2.2.1. Algoritmos genéticos

Los *algoritmos genéticos*[12, 11] son una clase de algoritmos evolutivos que utilizan una *población de individuos* para resolver un problema. En este contexto, los individuos son una secuencia de valores que representan una **solución al problema**. A cada individuo se le asigna una **aptitud** (*fitness*) que es un valor numérico que representa la calidad de la solución, de esta forma podremos discriminar qué soluciones son más apta para resolver el problema. Específicamente, los algoritmos genéticos pueden definirse de la siguiente forma:

1. Se crea una población inicial de soluciones aleatorias.
2. Se seleccionan pares de individuos siguiendo alguna *estrategia de selección*.
3. Se cruzan los pares de individuos siguiendo algún *operador de cruza* para obtener nuevos individuos (hijos).
4. Se mutan los nuevos individuos siguiendo algún *operador de mutación* y una *tasa de mutación*.[3]
5. Se evalúa la solución más apta de la nueva población respecto a un criterio de aceptación (e.g., margen de error). Si se cumple el criterio, se termina el algoritmo, si no, se vuelve al paso 2.

2.2.2. Programación genética

La *programación genética* (GP)[9] es un tipo de algoritmo genético donde el espacio de solución son **programas**. Para lograr esto se define a cada individuo como una representación abstracta de un programa, esta representación podría ser un **árbol de sintaxis abstracta**, una **pila de ejecución**, entre otras.
Un caso de uso típico de la *programación genética* es hacer una *regresión simbólica*[8] a partir de un conjunto de valores, donde el objetivo final es encontrar una función que se aproxime bien a los valores de entrada. Esto será relevante para este trabajo, ya que el problema que se quiere resolver es encontrar una función de evaluación para el *algoritmo Minimax*, esto se explicará en más detalle en la section 4.

3. Planteamiento del problema

Para tomar decisiones, el *algoritmo Minimax* utiliza una **función de evaluación** que otorga un valor numérico a cada decisión posible, de esta forma la «mejor» decisión es la que tiene mayor «puntaje».

[3]Probabilidad (usualmente baja) de que un individuo mute.

Si bien el *algoritmo Minimax* ha sido ampliamente estudiado y se ha demostrado que es una solución óptima para problemas de estrategia [1, 3], **no existe una manera estándar de encontrar una función de evaluación** para *Minimax* que sea óptima para todos los problemas de estrategia [19]. Más aún, incluso los juegos para los que el algoritmo ha mostrado resultados favorables no tienen una función de evaluación óptima conocida [4]. Esto muestra un claro problema para desarrollar *jugadores artificiales* en *video juegos*.

4. Estado del arte

4.1. *Artificial Ant Problem*

El **problema de las hormigas artificiales** [5, 6] (AAP) es un problema que busca encontrar un programa que, al ser ejecutado para una hormiga, les permite encontrar toda la comida en una grilla con una cantidad máxima de movimientos, considerando que la comida esté distribuida de acuerdo a un patrón.
En este caso, se espera que el programa sea capaz de reconocer los patrones de comida para encontrarla, podemos interpretar que este problema es un juego de estrategia de un jugador.
Para ver la relevancia de este problema para este trabajo, se debe notar que AAP puede ser reinterpretado como un juego de dos jugadores donde hay un adversario que no puede realizar movimientos y cada pieza de comida es un recurso; dicho de otra forma: el adversario tiene tantos recursos como la cantidad de comida que la hormiga aún no encuentra. De esta forma, el problema de la hormiga se reduce a un *juego de suma zero* de dos jugadores, por lo que encontrar una solución para este problema es equivalente a deducir una función de evaluación para *Minimax*.
En [6] *Koza* propone una solución al problema de las hormigas artificiales que utiliza *programación genética* para generar un programa con las primitivas:

- *Moverse hacia el frente*.
- *Girar a la derecha*.
- *Girar a la izquierda*.
- *Comprobar* si tiene comida en la casilla del frente.
- *Programas o funciones* de 2 ó 3 parámetros.

Sin embargo, si bien esta solución presentó resultados positivos, un análisis más exhaustivo mostró que la solución no es óptima [7]. Estos estudios mostraron que la solución propuesta presenta un *sesgo evolutivo* que evita la generalización de ésta a otros escenarios, principalmente porque no es capaz de encontrar una solución óptima para grillas que difieran de los datos con que se entrenó a la hormiga. Más aún, este problema de generalización **es común** en GP.
Las siguientes secciones presentan soluciones a este problema de generalización para casos generales y se plantearán como alternativas a la solución original de *Koza*.

4.2. Regresión simbólica

Dado que lo que buscamos es encontrar una función de evaluación, podemos reducir, tanto el problema de la hormiga como la deducción de la función de evaluación, a un problema de regresión simbólica. El tópico de *Programación Genética para Regresión Simbólica* (GPSR) se ha desarrollado ampliamente para buscar soluciones que puedan ser generalizadas para resolver el problema que se planteó en la sección anterior.
Un problema común que limita la capacidad de generalización de la regresión simbólica es que el modelo de GP tiende a aprender modelos demasiado complejos, estos modelos generan soluciones específicas al dominio del set de datos con el que se entrenó (impidiendo la generalización de la solución) [15].
En este tópico se han propuesto múltiples soluciones, Raymond et. al. [14, 15] proponen una solución de interés para este trabajo ya que presenta una generalización con resultados favorables además de evitar que el modelo de GP aprenda modelos demasiado complejos. Para esto los autores utilizan una representación alternativa a la representación clásica de GP utilizando *Splines* en lugar de árboles para representar el modelo de la solución.

5. Hipótesis

Para responder a la pregunta «¿Es posible encontrar la función de evaluación para que el algoritmo *Minimax* produzca estrategias óptimas?» se plantea la hipótesis:

> Dada una representación de su estado, es posible encontrar una función de evaluación para un juego de suma cero genérico mediante GPSR (section 4.2) que permita utilizar el algoritmo *Minimax* para encontrar una estrategia óptima.

6. Objetivos

6.1. Objetivos generales

La investigación propuesta tiene como objetivo principal *diseñar una técnica* (algoritmo) que les permita a lxs desarrolladorxs de videojuegos, además de gente en las áreas de investigación de videojuegos y teoría de juegos, definir una función de evaluación que permita **desarrollar jugadores artificiales** utilizando *Minimax* como algoritmo de decisión.

6.2. Objetivos específicos

Los objetivos específicos son los siguientes:

- Crear una **representación genérica** del *estado de un juego* de suma cero de 2 jugadores en el que los jugadores toman turnos intercalados (e. g. si el jugador J_a realiza una jugada, la siguiente jugada la hará su oponente J_b).
- Implementar el algoritmo *Minimax* que utilice la representación de estados ya mencionada.
- Definir e implementar la **deducción de funciones de evaluación** utilizando *Programación Genética*.
- Utilizar las funciones de evaluación deducidas en dos juegos distintos: *Gato* (*Tic-Tac-Toe*) y *Ajedrez*.

7. Metodología

El desarrollo del trabajo se guía fuertemente en los objetivos específicos de la tesis, de esta forma se propone la división del trabajo en tareas atómicas, no necesariamente secuenciales:

1. Implementar el juego «Gato» que pueda ser jugado por dos jugadores (humanos o artificiales).
2. Implementar el juego «Ajedrez» que pueda ser jugado por dos jugadores (humanos o artificiales).
3. Implementar el algoritmo *Minimax* para el juego «Gato» y el juego «Ajedrez» con funciones de evaluación conocidas.
4. Generalizar el algoritmo *Minimax* para otros juegos (aquí no se busca implementar juegos adicionales a los ya mencionados).
5. Implementar un algoritmo de GP simple (basado en el propuesto por Koza [6]) que pueda encontrar una función de evaluación para los juegos mencionados.
6. Hacer una revisión bibliográfica sobre técnicas de generalización de algoritmos de GP para encontrar la(s) más adecuada(s) para el problema planteado.
7. Reimplementar la solución original con las técnicas de generalización investigadas.
8. Evaluar los resultados obtenidos por las funciones de evaluación conocidas y las obtenidas mediante los algoritmos de GP mediante simulaciones de partidas entre jugadores artificiales para medir su eficacia.

9. Comparar las funciones de evaluación obtenidas mediante GP con las conocidas en cuanto a similitud y complejidad.

Todo este trabajo sería realizado con el lenguaje de programación *Kotlin* [20] y el framework de algoritmos genéticos *Jenetics* [21].
Adicionalmente se plantea la posibilidad de hacer una revisión bibliográfica sobre usos de otros tipos de *Machine Learning* en videojuegos para tener una visión más completa sobre el estado del arte.

8. Resultados esperados

Para evaluar los resultados de este trabajo se utilizarán las implementaciones de los juegos mencionados en la sección anterior para simular partidas entre jugadores artificiales. Para esto se utilizarán 3 tipos de jugadores artificiales: un jugador que tome decisiones aleatorias, un jugador que tome decisiones basadas en el *algoritmo Minimax* con una función de evaluación tomada de la literatura (conocida), y uno que tome decisiones basadas en el *algoritmo Minimax* con funciones de evaluación obtenidas con el algoritmo de GP.
Los jugadores artificiales luego pueden evaluarse en base a su *ratio de victorias* (*win rate*) enfrentandose a otros jugadores artificiales (tanto del mismo tipo como de los otros). En este caso, se espera que los jugadores basados en *Minimax* tengan un *win rate* superior a los que toman decisiones aleatorias, y que los jugadores con funciones de evaluación deducidas con GP tengan ratios de victoria similares a los que utilizan funciones de evaluación conocidas.
Además, se compararán las funciones *deducidas* con las *conocidas* en cuanto a su similitud y complejidad. En este caso, se espera que las funciones de evaluación deducidas tengan una complejidad similar a las de las funciones de evaluación conocidas.

9. Conclusiones

En esta propuesta se plantea una manera de facilitar el desarrollo de videojuegos mediante la automatización del proceso de encontrar una *función de evaluación para Minimax*. Para hacer esto se propone utilizar *programación genética* con el fin de hacer una regresión simbólica utilizando técnicas del estado del arte.
Adicionalmente, se plantea que esta investigación no solo servirá para el desarrollo de videojuegos, sino que también servirá para validar y evaluar las técnicas de regresión simbólica utilizando *programación genética*, así como su capacidad de generalización de soluciones.
Finalmente, se propone una evaluación de resultados en juegos clásicos mediante la simulación de partidas entre jugadores artificiales, de modo que se tenga un punto de referencia para medir la eficacia de la solución propuesta en comparación con la implementación de jugadores artificiales presentes en la literatura.

Referencias

[1] J. v. Neumann. «Zur Theorie der Gesellschaftsspiele». En: *Mathematische Annalen* 100.1 (dic. de 1928), págs. 295-320. ISSN: 0025-5831, 1432-1807. DOI: 10.1007/BF01448847. URL: http://link.springer.com/10.1007/BF01448847 (visitado 24-04-2022).

[2] A. M. Turing. «I.—COMPUTING MACHINERY AND INTELLIGENCE». En: *Mind* LIX.236 (1 de oct. de 1950), págs. 433-460. ISSN: 1460-2113, 0026-4423. DOI: 10.1093/mind/LIX.236.433. URL: https://academic.oup.com/mind/article/LIX/236/433/986238 (visitado 20-04-2022).

[3] Ky Fan. «Minimax Theorems». En: *Proceedings of the National Academy of Sciences of the United States of America* 39.1 (ene. de 1953), págs. 42-47. ISSN: 0027-8424. pmid: 16589233. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1063722/ (visitado 18-04-2022).

[4] Claude E. Shannon. «Programming a Computer for Playing Chess». En: *Computer Chess Compendium*. Ed. por David Levy. New York, NY: Springer New York, 1988, págs. 2-13. ISBN: 978-1-4757-1970-3 978-1-4757-1968-0. DOI: 10.1007/978-1-4757-1968-0_1. URL: http://link.springer.com/10.1007/978-1-4757-1968-0_1 (visitado 26-04-2022).

[5] David Jefferson y Rob Collins. «The Genesys System: Evolution as a Theme in Artificial Life». En: *Artificial Life - ALIFE* (1 de ene. de 1990).

[6] John R. Koza. «Genetic Programming as a Means for Programming Computers by Natural Selection». En: *Statistics and Computing* 4.2 (1 de jun. de 1994), págs. 87-112. ISSN: 1573-1375. DOI: 10.1007/BF00175355. URL: https://doi.org/10.1007/BF00175355 (visitado 22-04-2022).

[7] Ibrahim Kuscu. «Evolving a Generalised Behaviour: Artificial Ant Problem Revisited». En: *Evolutionary Programming VII*. Ed. por V. W. Porto y col. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1998, págs. 799-808. ISBN: 978-3-540-68515-9. DOI: 10.1007/BFb0040830.

[8] Riccardo Poli y col. *A Field Guide to Genetic Programming*. [Morrisville, NC: Lulu Press], 2008. 233 págs. ISBN: 978-1-4092-0073-4.

[9] William B. Langdon y Riccardo Poli. *Foundations of Genetic Programming*. Springer Science & Business Media, 9 de mar. de 2013. 265 págs. ISBN: 978-3-662-04726-2. Google Books: zsaqCAAAQBAJ.

[10] Michael Maschler, Eilon Solan y Shmuel Zamir. *Game Theory*. Cambridge: Cambridge University Press, 2013. ISBN: 978-0-511-79421-6. DOI: 10.1017/CBO9780511794216. URL: http://ebooks.cambridge.org/ref/id/CBO9780511794216 (visitado 18-04-2022).

[11] Milad Taleby Ahvanooey y col. «A Survey of Genetic Programming and Its Applications». En: *KSII Transactions on Internet and Information Systems (TIIS)* 13.4 (2019), págs. 1765-1794. ISSN: 1976-7277. DOI: 10.3837/tiis.2019.04.002. URL: https://www.koreascience.or.kr/article/JAKO201919761177651.page (visitado 20-04-2022).

[12] John H Holland. *Adaptation in Natural and Artificial Systems - An Introductory Analysis with Applications to Biology, Control, and Artificial I*. 2019. ISBN: 978-0-262-27555-2. URL: http://www.vlebooks.com/vleweb/product/openreader?id=none&isbn=9780262275552 (visitado 26-04-2022).

[13] Kiran K Thekumparampil y col. «Efficient Algorithms for Smooth Minimax Optimization». En: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper/2019/hash/05d0abb9a864ae4981e933685b8b915c-Abstract.html (visitado 20-04-2022).

[14] Christian Raymond y col. «Adaptive Weighted Splines: A New Representation to Genetic Programming for Symbolic Regression». En: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. GECCO '20. New York, NY, USA: Association for Computing Machinery, 25 de jun. de 2020, págs. 1003-1011. ISBN: 978-1-4503-7128-5. DOI: 10.1145/3377930.3390244. URL: https://doi.org/10.1145/3377930.3390244 (visitado 29-04-2022).

[15] Christian Raymond y col. «Multi-Objective Genetic Programming for Symbolic Regression with the Adaptive Weighted Splines Representation». En: (7 de jul. de 2021), págs. 165-166. DOI: 10.1145/3449726.3459461.

[16] *Evaluation Function*. En: *Wikipedia*. 27 de mar. de 2022. URL: https://en.wikipedia.org/w/index.php?title=Evaluation_function&oldid=1079533564 (visitado 18-04-2022).

[17] *Minimax*. En: *Wikipedia*. 12 de mar. de 2022. URL: https://en.wikipedia.org/w/index.php?title=Minimax&oldid=1076761456 (visitado 18-04-2022).

[18] *Zero-Sum Game*. En: *Wikipedia*. 5 de abr. de 2022. URL: https://en.wikipedia.org/w/index.php?title=Zero-sum_game&oldid=1081052640 (visitado 18-04-2022).

[19] Charles R. Dyer. *CS 540 Lecture Notes: Game Playing*. URL: https://pages.cs.wisc.edu/~dyer/cs540/notes/games.html (visitado 26-04-2022).

[20] *Kotlin Programming Language*. Kotlin. URL: https://kotlinlang.org/ (visitado 27-05-2022).

[21] Franz Wilhelmstötter. *Jenetics: Java Genetic Algorithm Library*. jenetics.io. URL: https://jenetics.io/ (visitado 20-04-2022).