Entrenamiento de un agente de Arkanoid mediante Algoritmos Genéticos y Políticas Heurísticas Ponderadas

Mishelle Orellana y Nathaniel Motykiewicz
Escuela de Computación

***Resumen*—En este trabajo se estudia el uso de un Algoritmo Genético (AG) para entrenar un agente que juega Arkanoid en un entorno implementado en JavaScript y *canvas* HTML5. La política del agente se codifica como un vector de pesos que pondera características del estado del juego (posición y velocidad de la bola respecto a la barra), dando lugar a una política lineal tipo *policy-based*. Se diseña una función de fitness que combina número de ladrillos destruidos, rebotes en la barra, tiempo de supervivencia y métricas de seguimiento de la bola. Experimentalmente se compara el desempeño del agente genético frente a un agente heurístico base y se realiza un estudio de ablación sobre distintos términos del fitness y sobre el uso de elitismo. Los resultados muestran que el AG es capaz de descubrir políticas competitivas y que ciertos términos del fitness son críticos para evitar comportamientos degenerados (ciclos sin destrucción de ladrillos).**

I. Introducción

Los juegos arcade clásicos como Arkanoid constituyen un banco de pruebas atractivo para el diseño y análisis de algoritmos, al combinar dinámica continua, colisiones y recompensas escasas. Desde la perspectiva de diseño de algoritmos, entrenar un agente que controle la barra puede verse como un problema de optimización sobre un espacio de políticas parametrizadas, donde cada política induce una secuencia de decisiones y, por tanto, una recompensa total.

En este contexto, el objetivo de este trabajo es entrenar un agente capaz de controlar la barra de Arkanoid utilizando un Algoritmo Genético (AG) que optimiza una política parametrizada por un vector de pesos. El enfoque se relaciona con problemas clásicos de búsqueda en espacios de soluciones y optimización de funciones objetivo, como se discute en la literatura de diseño de algoritmos y análisis de heurísticas [1], [2].

A diferencia de enfoques basados en redes neuronales profundas, aquí se emplea una política lineal simple (*policy-based* con parámetros $\mathbf{w}$) que decide acciones a partir de un conjunto reducido de características del estado. Esta elección permite estudiar con claridad el efecto de los operadores genéticos y del diseño de la función de fitness sobre el comportamiento final del agente, conectando con principios de diseño de algoritmos: descomposición del problema, análisis de complejidad e impacto de las decisiones de diseño en el rendimiento.

Las contribuciones principales del artículo son:

- La implementación de un entorno de Arkanoid en JavaScript con un motor de juego determinista y una interfaz web para entrenar y visualizar agentes evolutivos.
- El diseño de una codificación de política basada en un vector de pesos sobre características observables (distancias y velocidades relativas).
- Una función de fitness compuesta que recompensa destrucción de ladrillos y supervivencia, penalizando a la vez la distancia barra–bola y episodios largos sin progreso.
- Un análisis experimental que compara el agente genético con un agente heurístico base, incluye estudios de ablación de términos del fitness y discute sensibilidad a hiperparámetros y costes computacionales.

II. Metodología

II-A. Juego y motor de simulación

El juego implementado es una variante simplificada de Arkanoid: una bola que rebota contra paredes, barra y ladrillos, con colisiones elásticas y un sistema de puntuación basado en ladrillos destruidos. El motor está programado en JavaScript sobre un elemento `canvas` 2D, con un bucle de juego controlado por `requestAnimationFrame` y un paso de integración $\Delta t \approx 1/60$ s.

El estado observable en cada instante incluye, desde la perspectiva de un jugador legal:

- Posición de la bola (x_b, y_b).
- Velocidad de la bola (v_x, v_y).
- Posición de la barra (x_p, y_p).

No se utilizan atajos no observables (por ejemplo, contar ladrillos leyendo directamente estructuras internas sin pasar por colisiones), en cumplimiento de las restricciones del enunciado.

II-B. Codificación del agente (genotipo → política)

Se adopta la opción de **reglas heurísticas ponderadas**. Cada individuo del AG representa una política determinista parametrizada por un vector de pesos

$$\mathbf{w} = [w_{dx}, w_{dy}, w_{vx}, w_{vy}, b] \in \mathbb{R}^5. \tag{1}$$

A partir del estado observable se construye un vector de características normalizadas:

$$dx = \frac{x_b - (x_p + \frac{1}{2}\text{width}_p)}{\text{width}_{\text{canvas}}}, \tag{2}$$

$$dy = \frac{y_b - y_p}{\text{height}_{\text{canvas}}}, \tag{3}$$

$$v'_x = \frac{v_x}{v_{\max}}, \quad v'_y = \frac{v_y}{v_{\max}}, \tag{4}$$

$$\phi = [dx, dy, v'_x, v'_y, 1]. \tag{5}$$

La política lineal calcula una puntuación escalar

$$s = \mathbf{w}^\top \phi \tag{6}$$

y aplica una función de decisión por umbral:

$$a = \begin{cases} -1 & \text{si } s < -\tau \quad \text{(mover barra a la izquierda)}, \\ 0 & \text{si } |s| \le \tau \quad \text{(barra quieta)}, \\ +1 & \text{si } s > \tau \quad \text{(mover barra a la derecha)}, \end{cases} \tag{7}$$

donde τ es un pequeño umbral (por ejemplo $\tau = 0{,}05$) que evita oscilaciones por valores cercanos a cero. Esta acción se traduce en un desplazamiento horizontal de la barra proporcional a una velocidad fija y al Δt del simulador.

Desde el punto de vista de diseño de algoritmos, esta representación corresponde a un espacio de búsqueda continuo de baja dimensión, lo que facilita la exploración mediante operadores simples (cruce de un punto y mutación aditiva) en contraste con representaciones de mayor dimensión que suelen ser más costosas de optimizar.

II-C. Algoritmo Genético

II-C1. Parámetros: El AG opera sobre una población de N individuos (por defecto $N = 30$) durante G generaciones (por defecto $G = 1000$), con horizonte de simulación $T =$ `Max_steps` pasos por episodio. Los parámetros son configurables desde la interfaz y se documentan en el protocolo experimental.

Se exploraron distintas combinaciones de hiperparámetros, variando:

- $N \in \{20, 30, 50\}$ (tamaño de población).
- $G \in \{200, 500, 1000\}$ (número máximo de generaciones).
- Tasa de mutación $m \in [0{,}01, 0{,}1]$.

Con ello se analizó el compromiso entre exploración (poblaciones más grandes y mutación alta) y costo computacional por generación.

II-C2. Operadores genéticos:

Inicialización: Cada gen w_i se inicializa de forma uniforme en $[-1, 1]$, generando políticas inicialmente diversas y sin sesgo hacia una dirección concreta. Esta elección coincide con estrategias estándar de inicialización aleatoria en problemas de optimización [1].

Selección: Se utiliza **selección por ruleta** proporcional al fitness. Sea F_i el fitness del individuo i y $F_{\text{tot}} = \sum_i F_i$. Se elige un número aleatorio $r \sim U(0, F_{\text{tot}})$ y se recorre la población acumulando fitness hasta que la suma excede r. Este esquema se documenta como selección tipo ruleta y puede verse como una implementación simple de un algoritmo aleatorizado donde las probabilidades de elección dependen de la calidad de la solución.

Cruzamiento: Se aplica un **cruzamiento de un punto**. Dado un par de padres (p_1, p_2) con genotipos $\mathbf{w}^{(1)}$ y $\mathbf{w}^{(2)}$, se escoge un punto de corte k y se construye el hijo:

$$w_i^{(h)} = \begin{cases} w_i^{(1)} & \text{si } i < k, \\ w_i^{(2)} & \text{si } i \ge k. \end{cases} \tag{8}$$

Al tratarse de un cromosoma de longitud pequeña, el cruce de un punto resulta suficiente para recombinar soluciones sin introducir gran complejidad adicional.

Mutación: Cada gen tiene una probabilidad m de ser perturbado añadiendo ruido uniforme en un pequeño intervalo:

$$w_i \leftarrow w_i + \epsilon, \quad \epsilon \sim U(-0{,}2, 0{,}2). \tag{9}$$

Este operador introduce aleatoriedad controlada en el proceso, permitiendo escapar de óptimos locales y logrando un equilibrio entre intensificación y diversificación, un tema recurrente en el diseño de heurísticas [2].

Reemplazo y elitismo: En cada generación se ordena la población por fitness descendente y se copia sin cambios el mejor $10\,\%$ de individuos (al menos uno), garantizando elitismo. El resto de la nueva población se rellena por selección, cruzamiento y mutación. En los experimentos de ablación se estudia el efecto de desactivar este componente sobre la estabilidad del progreso generacional.

II-D. Función de fitness

Cada individuo se evalúa en M episodios simulados con distintos ángulos iniciales de la bola

$$\theta \in \{-\pi/4,\ -\pi/3,\ -\pi/6\},$$

y su fitness total es la media de los fitness por episodio.

En un episodio, se registran las siguientes cantidades:

- B: número de ladrillos destruidos por colisión.
- R: número de rebotes de la bola en la barra.
- P: número de pasos de simulación sobrevividos (hasta que la bola cae o se agotan los pasos).
- $\bar{d}_x$: distancia media barra–bola en el eje x.
- $\bar{h}$: altura media de la bola (cuánto tiempo pasa lejos del fondo).
- L: máximo número de pasos consecutivos sin destruir ladrillos.

La función de fitness se define como:

$$\begin{aligned} F = {} & 1200B + 300R + 0{,}8P \\ & + 500\frac{\bar{h}}{H} - 400\frac{\bar{d}_x}{W} - 800\frac{L}{T}, \end{aligned} \tag{10}$$

donde W y H son el ancho y alto del `canvas`, respectivamente. Esta combinación busca:

- Premiar destrucción de ladrillos (B) y rebotes controlados en la barra (R).
- Premiar supervivencia (P) y mantener la bola en la zona alta ($\bar{h}$).
- Penalizar distancias grandes barra–bola ($\bar{d}_x$) y episodios largos sin progreso (L), evitando políticas que se limiten a mantener la bola en ciclos sin destruir ladrillos.

La construcción de esta función de fitness es análoga al diseño de funciones objetivo en problemas de optimización multicriterio: diferentes términos capturan objetivos potencialmente en tensión (por ejemplo, supervivencia frente a progreso en el nivel), y los coeficientes actúan como pesos que reflejan la importancia relativa de cada aspecto.

II-E. Pseudocódigo

II-E1. Evaluación de fitness por individuo:

```
function evaluar_individuo(genoma W):
    angulos = [-pi/4, -pi/3, -pi/6]
    total_fitness = 0
    for theta in angulos:
        ent = crear_entorno_simulado(theta)
        pasos = 0
        B = R = 0
        sum_dx = sum_alt = 0
        pasos_sin_brick = 0
        L = 0
        while pasos < T and ent.vivo:
            phi = extraer_features(ent)
            accion = politica_lineal(W, phi)
            mover_barra(ent, accion, DT)
            mover_bola(ent, DT)
            manejar_colisiones(ent, B, R)
            actualizar_L_y_pasos_sin_brick()
            sum_dx += distancia_horizontal(ent)
            sum_alt += altura(ent)
            pasos += 1
        F_theta = combinar_en_fitness(B, R, pasos,
                                      sum_dx, sum_alt, L)
        total_fitness += F_theta
    return total_fitness / len(angulos)
```

II-E2. Bucle del Algoritmo Genético:

```
poblacion = inicializar_poblacion(N)
for g in 1..G:
    for cada individuo i en poblacion:
        i.fitness = evaluar_individuo(i.genoma)
    ordenar poblacion por fitness descendente
    registrar estadisticas (mejor, promedio)
    nueva_poblacion = copiar_elites(poblacion)
    while len(nueva_poblacion) < N:
        p1 = seleccionar_ruleta(poblacion)
        p2 = seleccionar_ruleta(poblacion)
        hijo = cruzar_un_punto(p1, p2)
        mutar(hijo, tasa_mutacion)
        nueva_poblacion.agregar(hijo)
    poblacion = nueva_poblacion
mejor = argmax_i poblacion[i].fitness
```

II-F. Complejidad temporal y cuello de botella

Desde el punto de vista de complejidad, el coste por generación está dominado por la evaluación del fitness. Sea N el tamaño de la población, M el número de episodios por individuo y T el número máximo de pasos por episodio. Cada individuo se evalúa en M episodios y cada episodio recorre a lo sumo T pasos, por lo que el coste de la fase de evaluación es del orden de

$$\mathcal{O}(N \cdot M \cdot T).$$

Adicionalmente, en cada generación se ordena la población por fitness para aplicar el elitismo, lo cual tiene coste $\mathcal{O}(N \log N)$. Dado que en la práctica T es grande comparado con $\log N$, el término $\mathcal{O}(N \cdot M \cdot T)$ resulta claramente dominante.

El principal cuello de botella identificado es, por tanto, la simulación del entorno de juego: actualización de posiciones, detección de colisiones y acumulación de estadísticas en cada paso. Operaciones como la selección, el cruce y la mutación tienen un coste mucho menor en comparación. Como consecuencia, las optimizaciones más efectivas pasan por acelerar el motor de simulación (por ejemplo, reduciendo comprobaciones redundantes de colisión o evitando cálculos innecesarios en cada frame) antes que por microoptimizar los operadores genéticos.

II-G. Reproducibilidad

Para favorecer la reproducibilidad de los experimentos, se fijaron explícitamente varias semillas aleatorias globales:

$$\text{seeds} \in \{42, 123, 999\},$$

aplicadas al generador de números aleatorios del AG y a la inicialización de ángulos de lanzamiento de la bola. De este modo, es posible repetir las corridas y obtener trayectorias de fitness muy similares.

Todos los experimentos se ejecutaron en el navegador Google Chrome versión XX sobre el sistema operativo YY (por ejemplo, Windows 10 o Ubuntu 22.04; se recomienda detallar la versión exacta en el documento final). En el archivo `config_experimentos.json` (incluido en el `.zip`) se documentan los valores de N, G, M, horizonte T y tasas de cruzamiento y mutación.

Adicionalmente, se generan archivos de log en formato CSV (`log_fitness_seed42.csv`, etc.) que contienen, por generación, el mejor y el fitness promedio de la población. Estos logs permiten reconstruir las curvas de la Figura 1, comparar el progreso entre semillas y verificar numéricamente los resultados reportados en las tablas.

III. Resultados

En esta sección se presentan resultados empíricos que ilustran el comportamiento del AG, su comparación con un agente heurístico base y el impacto de las ablaciones sobre términos del fitness y el uso de elitismo.

Figura 1. Mejor y promedio de fitness por generación para el AG con configuración base.

III-A. Desempeño global del AG

La Figura 1 muestra la evolución del mejor y del fitness promedio por generación para el AG con parámetros base. En la primera fase de la ejecución se observa una mejora rápida del fitness máximo, seguida de una fase más lenta donde las ganancias marginales se reducen conforme la población se concentra alrededor de soluciones de alta calidad.

En las corridas reportadas, la curva del mejor individuo tiende a estabilizarse alrededor de la generación $g^\star$ (por ejemplo, entre las generaciones 300 y 500), mientras que el promedio de la población se mantiene por debajo pero con una tendencia ascendente suave. Esto indica que, incluso en etapas avanzadas, persiste cierta diversidad genética, lo cual puede ser beneficioso para evitar convergencia prematura.

Desde la óptica de diseño de algoritmos, este patrón es coherente con lo esperado en heurísticas basadas en población: el elitismo empuja el mejor fitness hacia arriba, mientras que la mutación y el cruce introducen variabilidad que mantiene viva la exploración.

III-B. Comparación con baseline

Como baseline se utiliza un agente heurístico que mueve la barra siempre hacia la posición horizontal de la bola (control *greedy*). Este agente tiende a seguir la bola de manera directa, sin considerar su velocidad ni la altura del rebote, lo que puede producir movimientos bruscos o reacciones tardías frente a trayectorias rápidas.

Cuadro I
COMPARACIÓN DE DESEMPEÑO ENTRE AGENTE GENÉTICO Y BASELINE (MEDIA $\pm$ DESVIACIÓN).

Agente	Score medio	Score std
Baseline greedy	$\mu_b \pm \sigma_b$	
AG (mejor individuo)	$\mu_{GA} \pm \sigma_{GA}$	

Cuadro II
IMPACTO DE LAS ABLACIONES SOBRE EL FITNESS FINAL Y EL SCORE MEDIO.

Configuración	Fitness medio	Score medio
AG base	$F_{\text{base}} \pm \sigma_F$	$S_{\text{base}} \pm \sigma_S$
Sin penalización L	$F_L \pm \sigma_{F_L}$	$S_L \pm \sigma_{S_L}$
Sin elitismo	$F_E \pm \sigma_{F_E}$	$S_E \pm \sigma_{S_E}$

La Tabla I resume el score medio en el juego real (puntuación de Arkanoid) sobre K episodios para cada agente. Se reportan media y desviación estándar sobre distintas semillas.

En los experimentos realizados, el agente genético obtiene, en promedio, una mayor puntuación y una desviación estándar ligeramente inferior, lo que indica un comportamiento más consistente. Cualitativamente, el agente entrenado tiende a posicionar la barra con mayor anticipación respecto a la trayectoria de la bola y a aprovechar mejor ciertos ángulos de rebote para atacar regiones densas de ladrillos.

III-C. Ablaciones

Se diseñaron experimentos de ablación para estudiar el impacto de dos decisiones de diseño clave:

- **Sin penalización por** L: eliminar el término de la Ecuación (10) que penaliza muchos pasos seguidos sin romper ladrillos.
- **Sin elitismo**: ejecutar el AG sin copiar el mejor individuo en cada generación.

La Tabla II ilustra, de forma genérica, el tipo de estadísticas que se reportan (fitness final y score medio), dejando al lector la tarea de rellenar los valores concretos a partir de sus logs experimentales.

En la configuración sin penalización L se observaron políticas que, tras alcanzar un estado en el que la bola rebota de forma segura entre la barra y la parte superior del escenario, dejan de buscar ladrillos nuevos y se limitan a “mantenerse vivas”. Este comportamiento se traduce en episodios largos pero con escaso progreso, y evidencia la importancia de incluir términos de penalización que desalienten ciclos sin avance.

Por otro lado, al eliminar el elitismo, el progreso entre generaciones se vuelve más inestable: en varias corridas la mejor solución encontrada en generaciones intermedias se pierde en generaciones posteriores debido a mutaciones desfavorables. Aunque en algunos casos esto permite escapar de óptimos locales, el coste en términos de variabilidad y tiempo de convergencia resulta significativo.

IV. Discusión

Los resultados obtenidos permiten reflexionar sobre varios aspectos del diseño del AG y de su aplicación a Arkanoid.

En primer lugar, la elección de una política lineal de baja dimensión facilita el análisis y la visualización del comportamiento del agente, pero limita la capacidad expresiva del modelo. Ciertos patrones de movimiento complejos (por ejemplo, secuencias planificadas para limpiar regiones específicas de ladrillos) podrían requerir políticas con memoria o funciones de valor más sofisticadas. Sin embargo, desde el punto de vista pedagógico, esta simplicidad es una ventaja: se pueden vincular directamente cambios en los pesos con cambios observables en la estrategia.

En segundo lugar, el diseño de la función de fitness ilustra el papel central que juegan las funciones objetivo en problemas de optimización. Pequeñas modificaciones en los coeficientes o en los términos incluidos pueden producir cambios cualitativos importantes en el comportamiento del agente. La ablación del término L mostró que, sin una penalización explícita al estancamiento, el AG puede explotar trayectorias seguras pero poco productivas, una manifestación clara de la máxima "el algoritmo optimiza lo que se le pide, no lo que uno cree que le pidió".

Otro aspecto relevante es la sensibilidad a hiperparámetros. Aumentar el tamaño de población N tiende a mejorar la calidad de las soluciones finales, pero incrementa linealmente el tiempo por generación (coherente con la complejidad $\mathcal{O}(N \cdot M \cdot T)$). Tasas de mutación demasiado bajas conducen a una exploración pobre y a convergencia prematura, mientras que tasas muy altas introducen demasiado ruido y dificultan la explotación de soluciones buenas ya encontradas.

Finalmente, la validez de los resultados depende de varias decisiones experimentales: número de semillas (tres en este trabajo), nivel del juego, condiciones iniciales y fidelidad del motor de simulación frente al juego real. Cualquier cambio en estos elementos puede alterar las conclusiones, por lo que es importante documentarlos de forma transparente para facilitar la reproducibilidad.

V. Conclusiones y trabajo futuro

Este trabajo presentó el entrenamiento de un agente de Arkanoid mediante un Algoritmo Genético que optimiza una política lineal definida por un pequeño conjunto de características del estado. A pesar de su simplicidad, el AG fue capaz de aprender políticas que superan a un agente heurístico *greedy* y de exhibir comportamientos razonables en términos de posicionamiento de la barra y aprovechamiento de rebotes.

Se mostró además que ciertos términos de la función de fitness, en particular la penalización por estancamiento (L), son esenciales para evitar comportamientos degenerados. El uso de elitismo se confirmó como un mecanismo eficaz para consolidar el progreso generacional, aunque su ausencia puede abrir oportunidades de exploración más agresivas.

Como trabajo futuro, se plantean varias extensiones naturales:

- Diseñar políticas con memoria (por ejemplo, autómatas finitos o redes recurrentes) que puedan implementar estrategias de mayor nivel.
- Evaluar el AG en múltiples niveles con configuraciones diferentes de ladrillos, para estudiar la capacidad de generalización de las políticas aprendidas.
- Explorar otros esquemas de selección (torneo, selección truncada) y operadores de cruce adaptados a espacios continuos.
- Integrar técnicas de análisis de algoritmos (por ejemplo, cotas probabilísticas sobre el tiempo de convergencia) inspiradas en la literatura de diseño de algoritmos [1], [2].

Cuadro III
Checklist de requisitos mínimos del proyecto.

Ítem	Descripción
Juego	Arkanoid simplificado en JS/canvas
Política	Vector de pesos $\mathbf{w}$ (heurística ponderada)
Selección	Ruleta proporcional al fitness
Cruce	1 punto en el cromosoma
Mutación	Perturbación uniforme en pesos
Reemplazo	Elitismo (10 % mejor población)
Fitness	Ecuación (10) con $B, R, P, \bar{d}_x, \bar{h}, L$
Protocolo	Semillas, N, G, M, tasas documentadas
Baseline	Agente greedy que sigue la bola
Ablaciones	Sin penalización L, sin elitismo

Checklist resumido

La Tabla III resume el cumplimiento de los requisitos mínimos del proyecto.

Referencias

[1] J. Kleinberg y É. Tardos, *Algorithm Design*. Pearson/Addison Wesley, 2006.

[2] S. S. Skiena, *The Algorithm Design Manual*, 2.ª ed. Springer, 2010.