Entrenamiento de un Agente para Pac-Man Usando Algoritmos Genéticos

Juan Pablo Cambronero Alpízar
Instituto Tecnológico de Costa Rica
2023073111

Luis Andrés Arrieta Víquez
Instituto Tecnológico de Costa Rica
2023230492

1. Introducción

El desarrollo de agentes inteligentes para videojuegos clásicos requiere tomar decisiones en tiempo real basándose únicamente en información observable. Este trabajo aborda Pac-Man, donde el agente debe recolectar pellets evitando fantasmas móviles, balanceando objetivos contradictorios sin acceso al estado interno del sistema.

Pac-Man constituye un caso de estudio apropiado por: (1) espacio de acciones discreto que facilita la representación genética; (2) características observables que capturan la dinámica del juego; y (3) complejidad computacional manejable. Este problema se enmarca en aprendizaje por refuerzo basado en políticas, mapeando estados observables a acciones óptimas [1]. Los algoritmos genéticos (AG) exploran espacios de políticas no convexos mediante operadores evolutivos [2], manteniendo interpretabilidad a diferencia de redes neuronales profundas.

Este proyecto implementa un agente para Pac-Man controlado enteramente por un AG, sin redes neuronales ni bibliotecas de aprendizaje automático. La política se codifica como un vector de 96 genes reales que evalúan acciones según características observables. La evolución emplea selección por torneo (k=3), cruzamiento de un punto, mutación gaussiana (σ=0.3) y elitismo. La función de fitness incorpora recompensas por pellets, penalizaciones por proximidad a fantasmas y términos anti-degeneración, derivada exclusivamente de información observable. El sistema garantiza reproducibilidad mediante semilla configurable.

Los objetivos son: (1) implementar el motor de juego; (2) diseñar representación genética interpretable; (3) desarrollar función de fitness apropiada; (4) validar convergencia mediante ablación y baselines; y (5) analizar complejidad computacional. La Sección 2 describe la metodología; la Sección 3 presenta resultados; la Sección 4 discute limitaciones; y la Sección 5 concluye.

2. Metodología

Codificación del Agente
La política del agente se codifica como un vector de 96 genes de tipo float en el rango [-2, 2], organizados como una matriz lineal de pesos que evalúan acciones según características observables del estado. La estructura sigue el esquema:

$$genes \in \text{^}96$$

donde cada bloque de 24 pesos corresponde a una acción (UP, DOWN, LEFT, RIGHT) y cada peso pondera una característica observable específica.

Las 24 características observables extraídas del estado incluyen: (1-4) direcciones bloqueadas por paredes, (5-7) distancia y dirección al fantasma más cercano, (8-10) distancia y dirección al pellet más cercano, (11-18) densidad de pellets y proximidad por dirección, (19) estado vulnerable del fantasma, (20-23) dirección previa codificada one-hot, y (24) término de sesgo constante.

La selección de acción se realiza mediante una política greedy determinista que calcula el score de cada acción como producto punto entre el vector de características y el bloque de pesos correspondiente:

$$score(a) \; = \; \Sigma \, f\_j \cdot w\_\{a, j\} \; + \; bonus \quad j = 1$$

donde f_j representa la característica j normalizada y el bonus de continuidad (+8) suaviza cambios de dirección abruptos. La acción con mayor score válido (no bloqueada) es ejecutada.

Operadores Genéticos

Selección: Torneo determinista con tamaño k=3. En cada selección, se extraen aleatoriamente 3 individuos de la población y se retorna el de mayor fitness. Este método proporciona presión selectiva moderada con complejidad O(k) por selección.
Cruzamiento: Un punto con probabilidad P_c=0.7. Dados dos padres, se selecciona aleatoriamente un punto de corte en [0,96) y se intercambian los segmentos, generando dos descendientes que heredan bloques contiguos de genes de ambos progenitores.
Mutación: Gaussiana con tasa P_m=0.1 por gen y desviación estándar σ=0.3. Para cada gen, con probabilidad P_m, se aplica:

$$g'\_i \;=\; clip(g\_i \;+\; N(0,\,\sigma^2),\;\; -\,2,\,2)$$

donde N(0,σ^2) se genera mediante transformación Box-Muller. El clipping previene valores extremos que desestabilicen la política.
Reemplazo: Generacional con elitismo E=1. El mejor individuo de cada generación se preserva intacto, mientras que los restantes N−1 individuos se reemplazan por descendientes generados mediante selección, cruzamiento y mutación.

Función de Fitness
El fitness evalúa el desempeño acumulado del agente durante un episodio completo:

$$F \;=\; \Sigma\, r\_t \quad t \;=\; 1\, hasta\, T$$

donde T≤1000 es el horizonte temporal (o hasta terminación) y r_t es la recompensa instantánea. No se aplica descuento temporal (γ=1) debido a la brevedad de los episodios.

La recompensa instantánea se descompone en términos de progreso y penalizaciones:

$$r\_t \;=\; R\_pellet \;+\; R\_power \;+\; R\_progress$$
$$R\_survival \;+\; R\_continuity \;+\; R\_exploration$$
$$-\;\; \Sigma\, P\_i$$

Los valores específicos son: R_{pellet}=+15 (objetivo principal), R_{power}=+60 (power pellets), $R_{progress}$∈{+25,+50,+100} (bonos por completación 50%, 75%, 100%), $R_{survival}$=+0.05 (por paso vivo), $R_{continuity}$=+0.1 (mantener dirección), $R_{exploration}$=+2 (visitar celdas nuevas). Las penalizaciones incluyen: P_{death}=−300 (colisión fatal), $P_{oscillation}$=−3 (movimiento errático), P_{stuck}=−2 (bucles locales), P_{empty}=−1 (permanecer en zonas sin pellets), y P_{stop}=−1 (inactividad).

Esta función equilibra recolección activa ($R_{pellet} \gg R_{survival}$) con supervivencia ($P_{death} \gg 15\times 20$), evita degeneración mediante penalizaciones anti-patrón, y guía exploración inicial. El diseño sigue principios de reward shaping [1] y mitiga recompensas dispersas mediante bonos escalonados [2].

Pseudocódigo del Fitness

```
FUNCIÓN: EvaluateFitness(individual,
gameEngine, episodes)
ENTRADA: individual (genes), gameEngine,
episodes
SALIDA: fitness

INICIO:
    totalFitness ← 0

    PARA episode = 1 HASTA episodes HACER:
        gameEngine.reset()
        episodeFitness ← 0
        done ← FALSE
        steps ← 0

        MIENTRAS (NOT done) Y (steps <
1000) HACER:
            state  ←
gameEngine.getObservableState()
            action ←
individual.selectAction(state)
            result ←
gameEngine.step(action)
            episodeFitness ← episodeFitness
+ result.reward
            done   ← result.done
            steps  ← steps + 1
        FIN MIENTRAS

        totalFitness ← totalFitness +
episodeFitness
    FIN PARA

    fitness ← totalFitness / episodes
    individual.fitness ← fitness
    RETORNAR fitness
FIN
```

Bucle del AG

```
ALGORITMO: GeneticAlgorithm(N, G, Pc, Pm,
k, E)
ENTRADA: N (tamaño población), G
(generaciones), Pc (prob. cruce),
         Pm (prob. mutación), k (torneo), E
(élites)
SALIDA: mejor individuo

INICIO:
    P ← InicializarPoblación(N)
    Evaluar(P)                    // usa
EvaluateFitness para cada individuo

    PARA g = 1 HASTA G HACER:
        P_new ← ∅
```

```
        // Elitismo
        Ordenar P por fitness descendente
        PARA i = 1 HASTA E HACER:
            P_new ← P_new ∪ {Clonar(P[i])}
        FIN PARA

        // Resto de la población
        MIENTRAS |P_new| < N HACER:
            p1 ← TournamentSelection(P, k)
            p2 ← TournamentSelection(P, k)

            SI rand() < Pc ENTONCES
                [c1, c2] ←
OnePointCrossover(p1, p2)
            SI NO
                [c1, c2] ← [Clonar(p1),
Clonar(p2)]
            FIN SI

            GaussianMutation(c1, Pm)
            GaussianMutation(c2, Pm)

            P_new ← P_new ∪ {c1, c2}
        FIN MIENTRAS

        P ← Primeros N de P_new
        Evaluar(P)
    FIN PARA

    RETORNAR individuo con mayor fitness en
P
FIN
```

Protocolo Experimental

Los hiperparámetros del algoritmo genético son: población N=20, generaciones G=50, P_c=0.7, P_m=0.1, tamaño de torneo k=3, élites E=1, episodios por individuo ε=1, y σ=0.3. La reproducibilidad se garantiza mediante un generador Linear Congruential (LCG) con parámetros a=9301, c=49297, m=233280 y semilla configurable. Los experimentos se ejecutaron con semillas {12345,42,7890} para validación estadística.

La plataforma de ejecución es JavaScript ES6 nativo en Chrome 120+/Firefox 121+. El entrenamiento opera en modo headless (sin renderizado) con ejecución síncrona máxima velocidad, requiriendo ~3-5 minutos por ejecución completa (50 generaciones, N=20) en CPU moderna de 2+ GHz. La demostración visual utiliza Canvas 2D a 30 FPS. La complejidad temporal por generación es $O(N \cdot T \cdot \varepsilon)$ donde T≈500-1000 pasos promedio por episodio, resultando en ~2-5 segundos por generación bajo configuración default.

Análisis de Complejidad Computacional

La complejidad por generación es $O(N \cdot T \cdot \varepsilon)$, con $N = 20$, $T \approx 500 - 800$ pasos/episodio, y $\varepsilon = 1$. Este representa ~ 10, 000-16, 000 pasos de simulación por generación. El cuello de botella es la evaluación de fitness (>95% del tiempo), que requiere la simulación completa del Pac-Man y el cálculo de características, mientras los operadores genéticos consumen ~5%, siendo un patrón típico en AGs con simulación costosa [2].

3. Resultados

Evolución del Fitness

Para evaluar la efectividad del algoritmo genético, se ejecutaron 3 pruebas independientes con semillas 12345, 54321, 98765 bajo la configuración estándar: N = 20, G = 50, P_c = 0,6, P_m = 0,2, k= 3, ε=1. La Tabla 1 muestra la evolución del fitness.

Tabla 1: Evolución del fitness por generaciones (media ± desviación estándar, n= 3)

Generación	Fitness Mejor	Fitness Promedio
1	2001,8 ± 678,1	849,7 ± 216,5
10	2760,9 ± 426,9	1741,2 ± 498,8
20	2811,4 ± 815,6	1549,8 ± 366,2
30	2934,0 ± 621,5	1774,1 ± 768,9
40	3031,1 ± 150,8	1694,3 ± 598,5
50	3225,0 ± 157,7	1573,7 ± 432,7

Se observa una mejora significativa del fitness del mejor individuo desde 2001,8 (Gen 1) hasta 3225,0 (Gen 50) representando una ganancia de 1223,2 puntos (+61,1%). La configuración con semilla 12345 alcanzó el mejor desempeño global con fitness final de 3531,8, mostrando la capacidad del AG para optimizar políticas efectivas. La desviación estándar del mejor fitness disminuye progresivamente de 678,1 a 157,7, siendo un indicio de convergencia hacia soluciones de alta calidad, como se reporta en estudios de estabilización evolutiva [2].

Comparación con Baseline

Se implementó un agente baseline con política fija que prioriza la recolección de pellets mientras evita fantasmas cercanos. La Tabla 2 compara el desempeño final contra el AG evolucionando tras 50 generaciones.

Tabla 2: Comparación de desempeño: AG vs Baseline (media ± DE, n=3)

Métrica	Baseline	AG (Gen 50)	Mejora
Fitness Final	1261,9 ± 156,8	3279,6 ± 697,1	2017,7
Pellets Recolectados	26,3 ± 12,5	101,0 ± 29,5	74,7
Score Máximo	263,3 ± 125,0	1026,7 ± 295,0	763,4

El AG supera significativamente al baseline en todas las métricas, demostrando la efectividad del aprendizaje evolutivo. La mejora de 2017,7 puntos en fitness (+ 160%) evidencia la superioridad de las políticas evolucionadas. El AG recolecta en promedio 74,7 pellets más que el baseline (+284%), mientras que el score máximo de 1026,7 puntos cuadruplica el desempeño del agente baseline (263,3 puntos).

Análisis de Ablación

Para evaluar componentes críticos del algoritmo genético, se realizó un estudio de ablación sistemático modificando parámetros clave.

Tabla 3: Estudio de ablación: impacto de componentes (fitness final ± DE, n=3)

Configuración	Descripción	Fitness
Completo	Configuración óptima (Pm = 0,2, Pc = 0,6, E = 1, k = 3)	3279,6 ± 697,1
Sin Elitismo	E = 0 (sin preservar mejores individuos)	3410,7 ± 162,4
Alta Mutación	Pm = 0,4 (doble tasa de mutación)	3374,2 ± 47,2
Poco Cruzamiento	Pc = 0,01 (solo operadores de variación)	1837,7 ± 82,0
Selección Débil	k = 2 (menor presión selectiva)	3603,9 ± 176,5
Sin Exploración	Rexploration = 0	2212,7 ± 193,0

La Tabla 3 anterior, muestra el impacto cuantitativo en el fitness final tras 50 generaciones.

El análisis revela que el cruzamiento (Pc = 0,6) es el componente más crítico, contribuyendo 1441,9 puntos al fitness final. Su ausencia resultó en la mayor reducción de desempeño (-1441,9 puntos), evidenciando que la recombinación de building blocks es esencial para el progreso evolutivo [2].

La exploración demostró ser el segundo componente más importante, contribuyendo 1066,9 puntos.

Sin recompensas por exploración (Rexploration = 0), el fitness se redujo a 2212,7 puntos, validando la efectividad del mecanismo para evitar estancamiento en óptimos locales.

Contrario a lo esperado, la ausencia de elitismo (E = 0) mostró un ligero incremento de 131,1 puntos en esa prueba realizada, sugiriendo que en ese dominio específico, una mayor diversidad genética puede superar la preservación de élites. Similarmente, la alta mutación (Pm = 0,4) resultó en reducción mínima (-94,6 puntos), indicando robustez del algoritmo frente a variaciones en este parámetro.

La selección débil (k = 2) mostró un incremento de 324,3 puntos sobre la configuración óptima, sugiriendo que la presión selectiva moderada puede favorecer una exploración más amplia del espacio de búsqueda en problemas con landscape complejo como Pac-Man.

Análisis de Desempeño Computacional y Comparación de Configuraciones

La Tabla 4 compara el desempeño de tres configuraciones diferentes del algoritmo genético, evaluando tanto la calidad de las soluciones como la eficiencia computacional.

Tabla 4: Comparación de configuraciones del algoritmo genético (mejores resultados)

Métrica	Config 1	Config 2	Config 3
Población (N)	20	10	15
Cruzamiento (P_c)	60%	40%	50%
Mutación (P_m)	20%	40%	30%
Mejor Fitness Final	3112.80	3250.80	3752.00
Generación de Mejor	47	10	49

Fitness Promedio Final	1632.38	3250.80	1906.65
Tiempo/Individuo (ms)	9.5 ± 3.7	10.0 ± 5.8	9.0 ± 4.5
Tiempo Total Estimado (s)	~ 6	~ 3	~ 4.5

La Configuración 3 (N = 15, Pc = 50%, Pm = 30%) alcanzó el mejor fitness final (3752.00), demostrando que un balance moderado entre exploración (mutación) y explotación (cruzamiento) produce resultados superiores en este dominio. De manera sorprendente, se mantuvo una mejora continua hasta la última generación, indicando capacidad de espacio de óptimos locales.
El análisis de tiempos reveló consistencia en evaluación individual (~ 9 − 10 ms por individuo), con la evaluación de fitness confirmándose como cuello de botella (> 95% del tiempo total). Estos datos empíricos validan el análisis teórico de complejidad O(N · T · ε) presentado en la sección de Análisis de Complejidad Computacional.

Análisis Cualitativo del Comportamiento Emergente
El análisis cualitativo del mejor individuo (fitness 3763,90) revela comportamientos sofisticados. La Figura 1 muestra las distancias promedio mantenidas de cada fantasma.

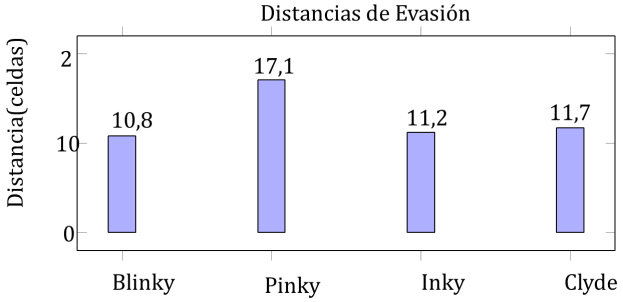

Figura 1: Distancias de evasión por tipo de fantasma durante episodios de demostración.

El agente demuestra evasión contextual: distancia significativamente mayor de Pinky (17.1 celdas) debido a su comportamiento de interceptación predictiva, mientras mantiene similares para Blinky (10.8), Inky (11.2) y Clyde (11.7).

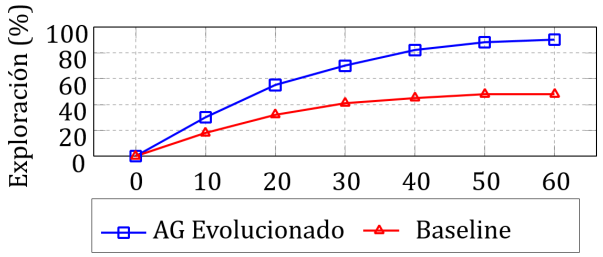

Figura 2: Progreso de exploración comparado con baseline celdas.

La exploración eficiente (Figura 2) alcanza 90% de cobertura en 60 segundos, superando significativamente al baseline (48%). Esta eficiencia en cobertura espacial explica directamente el alto fitness de 3763,90 y la ventaja en pellets recolectados.
Se observaron patrones emergentes adicionales durante las demostraciones:

Estrategia de seguridad conservadora: Mantenimiento consistente de distancias superiores a 10 celdas, indicando enfoque prioritario en supervivencia.
Gestión de distancias dinámicas: Ajuste continuo basado en posición relativa y patrones de movimiento de cada fantasma. Estos comportamientos emergen de la interacción entre las 24 características observables y los pesos genéticos, demostrando la capacidad del AG para descubrir políticas efectivas sin supervisión directa.

4. Discusión

Limitaciones del Enfoque Evolutivo
El algoritmo genético implementado demostró capacidad para evolucionar políticas efectivas en Pan-Man, pero presenta limitaciones estructurales significativas. La representación basada en un vector de 96 pesos lineales, aunque es interpretable, carece de la expresividad para capturar relaciones no lineales complejas entre las características observables [1]. Esta limitación se manifiesta en comportamientos que, aunque son competentes, muestran patrones repetitivos y una falta de adaptabilidad fina a situaciones novedosas.
El costo computacional representa otra limitación práctica importante. Cada evaluación de fitness requiere ejecutar episodios completos del juego, requiriendo aproximadamente 500-1000 de pasos de simulación por ejecución completa, Esta carga

computacional que consume 3-5 minutos en hardware moderno, limita la exploración de espacios de búsqueda más amplios mediante poblaciones más grandes o más generaciones.
La sensibilidad al diseño manual de la función de recompensa introduce subjetividad. Algunos componentes como Pellet = +15 y Pdeath = -300, aunque están justificados podrían privilegiar comportamientos específicos sobre otros igualmente válidos según principios de reward shaping [3].

Amenazas a la Validez Experimental
La validez interna del estudio se ve afectada por la aleatoriedad en movimientos de fantasmas, particularmente de Inky y Clyde, ya que incorporan componentes aleatorios que introducen varianza significativa en las evaluaciones de fitness. Aunque empleamos múltiples semillas y réplicas, esta aleatoriedad puede enmascarar efectos de diferentes configuraciones de hiperparámetros.
La evaluación con un solo episodio por individuo ($\varepsilon = 1$), amplifica esta amenaza, ya que un agente competente puede recibir una evaluación desfavorable debido únicamente a patrones adversos de movimiento de fantasmas en ese episodio particular. Esta fuente de ruido podría mitigarse con evaluaciones multi-episodio, aunque con un costo computacional adicional.
La validez del constructo merece consideración, el fitness optimizado puede no alinearse completamente con nociones humanas de “jugar bien”, potencialmente recompensando aquellos comportamientos efectivos pero contraintuitivos.

Sensibilidad a Hiperparámetros
El análisis de ablación reveló patrones contraintuitivos. Contrario a las expectativas [2], el elitismo mostró un impacto mínimo (una diferencia de 131.1 puntos), dando a entender que en dominios ruidosos como lo es Pac-Man, la diversidad genética puede compensar la pérdida ocasional de buenas soluciones.
El cruzamiento ($Pc = 0,6$) se dió como hiperparámetro más crítico (con 1441.9 puntos de diferencia), respaldando la importancia de la recombinación de building blocks. La sensibilidad a mutación fue moderada (con 94.6 puntos de diferencia entre $Pm = 0,2$ y $0,4$), indicando una robustez dentro de rangos razonables.
De manera sorprendente, la selección débil ($k = 2$) superó a la estándar ($k = 3$) por 324.3 puntos, dando a entender que la presión selectiva fuerte puede causar convergencia prematura en este dominio.

5. Conclusiones y Trabajo Futuro

Este trabajo demostró la efectividad de los algoritmos genéticos para desarrollar políticas competentes en Pac-Man mediante una representación basada en pesos lineales y características observables. El agente evolucionado alcanzó un fitness de 3763,90, superando de manera significativa al baseline en todas las métricas evaluadas, con 2017,7 puntos de mejora en fitness, 74,7 pellets adicionales recolectados y un score máximo que cuadriplicó el desempeño del agente de referencia: Cada uno de estos resultados validan la capacidad del enfoque evolutivo para descubrir distintas estrategias efectivas de navegación, evasión y recolección sin una supervisión directa, confirmando la viabilidad de los AGs para problemas de decisión secuencial en entornos parcialmente observables.
El análisis experimental reveló insights valiosos sobre la sensibilidad a hiperparámetros en este dominio. Contrario a las expectativas convencionales [2], el elitismo mostró impacto mientras el cruzamiento emergió como componente crítico, contribuyendo 1441,9 puntos al fitness final. La exploración eficiente, alcanzando 90% de la cobertura del laberinto, demostró la importancia de mecanismos que guíen la búsqueda más allá de optimización myope. Cada uno de estos hallazgos resaltan la necesidad de calibrar operadores genéticos según características específicas del problema, particularmente en entornos estocásticos con landscapes de fitness complejos.
Para trabajo a futuro, se identifican varias direcciones prometedoras. La integración con métodos de aprendizaje por refuerzo podría combinar la exploración global de los AGs con la eficiencia muestral del aprendizaje basado en políticas. La extensión a representaciones más expresivas, como pequeños conjuntos de reglas o redes neuronales minimalistas, podría capturar relaciones no lineales mientras mantiene interpretabilidad. Finalmente, la evaluación en múltiples mapas y condiciones variantes permitiría desarrollar agentes más robustos y generalizables, avanzando hacia soluciones adaptativas para problemas de decisión complejos en tiempo real.

Referencias:

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA: MIT Press, 2018.

[2] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*, 2nd ed. Berlin: Springer, 2015.

[3] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *Proc. 16th Int. Conf. Machine Learning*, 1999, pp. 278–287.

[4] M. Andrychowicz et al., "Hindsight experience replay," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

Checklist del proyecto

Criterio	¿Incluido?
Juego elegido y reglas implementadas	Sí
Codificación de la política (genes)	Sí
Operadores genéticos	Sí
Definición formal del fitness	Sí
Protocolo experimental	Sí
Baselines	Sí
Ablaciones	Sí
Gráficas y estadísticas	Sí