Controlador inteligente para el juego de la serpiente

Rodrigo A. Ortiz de Zarate, y Gisela A. Roncaglia
Catedra de Inteligencia Computacional, FICH, Universidad Nacional del Litoral

***Resumen*—El juego de la serpiente consiste en controlar un reptil de forma tal que, se alimente lo máximo posible, evitando golpear las paredes de su entorno o a sí misma para no morir. Como muchos juegos de Arcade, resulta de interés crear un agente inteligente que permita actuar sobre este. En este artículo, exploramos la aplicación de un controlador para el juego que permita obtener el mayor puntaje posible. Para realizar esto, se utilizaron tres funciones de evaluación, las cuales, son combinadas mediante una suma ponderada con diversos pesos asociados. Posteriormente, cada uno de estos pesos es optimizado mediante un algoritmo evolutivo, con el fin de obtener un controlador óptimo. Finalmente, se examinaron diferentes variantes del algoritmo evolutivo, aplicando distintos tipos de operadores y se realizaron diversos experimentos respecto a la función de aptitud utilizada comprobando que nuestro método de diseño es capaz de crear un controlador que realiza un buen trabajo e incluso es capaz de llegar al máximo puntaje del juego.**

***Palabras clave*—Videojuego, Serpiente, Algoritmo Evolutivo, Búsqueda en Amplitud, Inteligencia Artificial.**

I. INTRODUCCIÓN

Los algoritmos evolutivos, son utilizados en muchas aplicaciones de la Inteligencia Computacional [8][9], estos son métodos de optimización y búsqueda basados en la teoría de Darwin y se caracterizan por la existencia de una población de individuos, que, frente a la presión del ambiente y la selección natural, los más aptos prevalecen a través del tiempo, evolucionando hacia individuos más capaces que resuelvan de mejor forma un problema concreto. Dadas estas particularidades de los algoritmos evolutivos, se puede notar que, en muchas ocasiones, podemos encontrar ciertos parámetros que definen el funcionamiento de un juego que, de ser ajustados correctamente, podríamos crear una inteligencia artificial que resuelva cualquier tipo de problema.

El éxito de estos algoritmos puede verse en muchos juegos populares. Para juegos relativamente simples como *Lemmings*, Kendall y Spoerer [2] propusieron un controlador que permitía realizar un agente basado en algoritmos evolutivos. Este juego consiste en roedores que deben hallar la salida de un escenario cerrado en un cierto límite de tiempo, atravesando ciertos obstáculos. Para resolverlo, en cada instante, a cada roedor se le asigna una tarea en particular (cavar, construir, bloquear, caminar, etc.) ligada a diversos pesos, y se buscan los individuos óptimos para encontrar la salida en el menor tiempo posible. Por otro lado, juegos complejos que involucran estrategias complicadas, como el Ajedrez o las Damas, también han recibido suficiente atención como para que se lleguen a realizar controladores inteligentes. Para el primer caso, se han desarrollado métodos que, brevemente consisten en asignar a cada jugada (dependiendo de la pieza) cierto puntaje [4], devolviendo al finalizar la partida, la cantidad de movimientos y su resultado. En el caso del juego de las Damas, se han realizado controladores extremadamente optimizados, como en el caso de Schaeffer [5] que llegó a realizar un jugador inteligente que fue capaz de ganar a Marion Tinsley, considerado el mejor jugador de damas de la historia.

A pesar de que hay infinidad de artículos sobre desarrollo de controladores para diferentes tipos de juegos, se realizó una búsqueda enfocada sobre el problema planteado en este artículo. Kong y Mayans [2], proponen una forma de resolver el juego de la serpiente, mediante 5 métodos de búsqueda, cada método ofrece sus propias ventajas y desventajas, unos son más rápidos, otros no llegan a un puntaje alto, etc. La eficiencia de sus algoritmos se encuentra muy relacionada con el tamaño del escenario. Por otro lado, artículos como el de Ma, Tang y Zhang [7], intentan aplicar técnicas de *aprendizaje por refuerzo*, aunque sus resultados establecen que el algoritmo no produce individuos estables, sumado a en ningún caso la serpiente pudo alcanzar el puntaje máximo del juego.

En el caso más general, nuestra implementación consiste en la serpiente moviéndose dentro del área limitada, tratando de comer la mayor cantidad de manzanas posibles sin morderse a sí misma. Cada vez que come una manzana, una nueva será depositada en el escenario de forma aleatoria. Cuando la serpiente ya no tenga más posibilidades de avanzar el juego finaliza y se devuelve su puntaje.

En la Sección II del presente trabajo se desarrolla el funcionamiento del controlador basado en tres funciones de evaluación específicas [1], este definirá cual es la dirección óptima para que la serpiente se mueva a continuación, otorgándole mayor posibilidad de mantenerse con vida. Luego, en la sección III, se desarrolla el algoritmo evolutivo utilizado para optimizar los pesos que se proveen al controlador. Los experimentos y resultados de las pruebas son presentados en la Sección IV. Finalmente, concluimos nuestro informe en la Sección V, presentando nuestras conclusiones y posibles enfoques para una futura mejora al desarrollo.

II. CONTROLADOR INTELIGENTE

El controlador inteligente es el módulo encargado de recibir información sobre el estado del juego y de elegir en cada instante de tiempo cual es el mejor movimiento que podría realizar la serpiente en el siguiente paso, tal como se puede apreciar en la parte superior de la Figura 1.

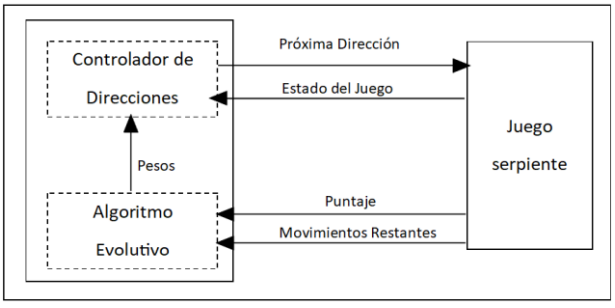

Fig. 1: Módulos generales

En síntesis, debe elegir si esta se mantendrá en dirección recta, si girara a su izquierda o a su derecha. Para realizar esta tarea, se verifica cada una de las tres potenciales elecciones de la serpiente mediante tres funciones de evaluación: suavidad, espacio y distancia al alimento, las cuales serán presentadas a continuación.

A. Evaluación de la Suavidad

La evaluación de la suavidad se relaciona con el riesgo de la serpiente a realizar muchos giros, si se evitan estos giros hay más probabilidades de que la serpiente se mantenga a salvo por más tiempo. La evaluación consiste en calcular la mínima distancia (en unidades de tiempo) a todas las posiciones del escenario, basándose en la futura decisión a tomar: doblar relativamente a la izquierda, a su derecha, o mantenerse recto.

En la Fig.2 se puede ver la serpiente en un instante de tiempo específico. Para cada posición del área, se calcula la cantidad de unidades de tiempo requeridas para llegar allí (utilizando Búsqueda en Amplitud). Si esta continua su trayecto en dirección recta en el próximo instante, la mayor distancia alcanzada será de 9, si gira a su izquierda será de 8 y a su derecha, 9 nuevamente.

B. Evaluación del Espacio

La función de evaluación de la suavidad ayuda a identificar una manera de que la serpiente viva cómodamente. Sin embargo, considerar solo la evaluación de la suavidad, en ciertas ocasiones, puede conducir a callejones sin salida. Al implementar la evaluación del espacio, podemos determinar cuántas posiciones alcanzables existen para una posible futura decisión.

Esto puede notarse en la Fig. 3, aquí la evaluación de la suavidad de girar a la derecha da por resultado 6, mientras que ir hacia la izquierda daría como resultado 5, por lo que, solo considerando el primer término, iríamos a un camino sin salida. En cambio, si tenemos en cuenta el espacio disponible, la elección más predominante seria ir hacia la izquierda y la serpiente estaría a salvo.

C. Evaluación distancia al alimento

La función que evalúa la distancia al alimento, al igual que antes, es calculada para las posibles futuras decisiones. Esta verifica si el alimento puede ser alcanzado luego de tomar una decisión. Siendo d(a) la distancia al alimento y s el espacio. Si el alimento no es alcanzable, la distancia es infinita y la función de evaluación daría cero; de lo contrario, establecemos la evaluación de distancia alimento como:

$$\mathrm{f}(d) = \frac{s}{d(a)} \qquad (1)$$

Esta evaluación establece una relación entre la distancia hacia la comida si tomamos una dirección en particular y el espacio que tiene disponible en esa misma dirección, ya que,

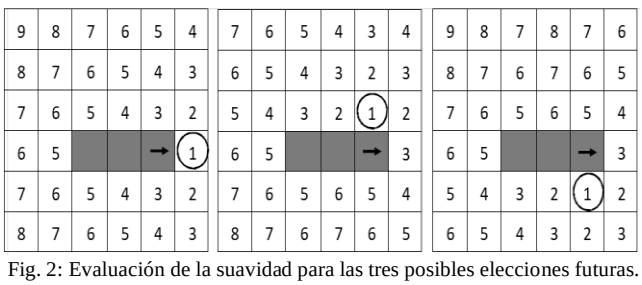

Fig. 2: Evaluación de la suavidad para las tres posibles elecciones futuras.

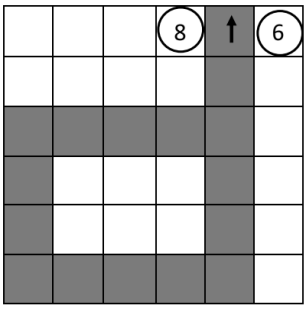

Fig. 3: Evaluación del espacio para sus posibles direcciones futuras.

en muchas ocasiones, ir en una cierta dirección podría acercarnos mucho a el alimento y al mismo tiempo ingresar a un callejón sin salida. Esto puedo verse en la Fig 4, a pesar de que el alimento esté más cercano, si la serpiente gira a su derecha en el próximo instante, se conduce a un pasillo sin salida, evaluando (1), arroja un resultado de 3/1=3.0 , por otro lado, si evaluamos la función con una posible elección hacia su izquierda, el resultado es 20/3=6.66, a pesar de que la víbora no va hacia el alimento, es posible que sobreviva más tiempo debido a la cantidad de espacio disponible para moverse (posteriormente, podría ir hacia la comida cuando sea seguro).

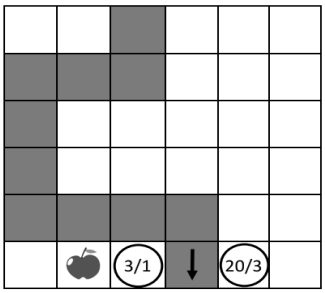

Fig. 4: Evaluación de la distancia hacia el alimento.

D. Definición de la siguiente dirección

Cada una de las posibles decisiones que puede llegar a realizar la serpiente a lo largo de toda su vida es calculada por cada unidad de tiempo y calificada por las tres funciones descritas en las subsecciones anteriores. Las funciones de evaluación se denotan por fi(c), donde i ∈ {1=evaluación suavidad, 2= evaluación espacio, 3=distancia al alimento}, c es una opción (elección sobre camino a seguir), l es la longitud actual y LM la longitud máxima que puede tener la serpiente. Debido a que las diferentes funciones de calificación pueden tener diferente trascendencia, utilizamos una suma ponderada para agregar estos puntajes:

$$F(c) = \sum_{i=1}^{3} f_i(c) \cdot \left(W_{2i-1} + W_{2i}\frac{l}{LM}\right) \qquad (2)$$

La elección que maximiza a la función F(c) es elegida por sobre las demás.

III. OPTIMIZACIÓN DE PESOS MEDIANTE UN ALGORITMO EVOLUTIVO

Dada la ecuación (2), establecemos ciertos pesos W_{2i-1} y W_{2i}, el primero de ellos está relacionado con la influencia de las funciones de evaluación y el segundo representa la importancia del tamaño de la víbora sobre el escenario (a medida que crece la serpiente, se debe brindar mayor importancia a que la víbora se mantenga con vida a que, por ejemplo, se enfoque solo en alimentarse). Cada uno de estos

pesos será optimizado mediante un algoritmo evolutivo dado por seis valores aleatorios entre [-15,15] y los genes que conforman al cromosoma de cada individuo estarán dados por valores reales en el rango [-1,1]. La estructura de los pesos utilizada, se puede observar en la Figura 5.

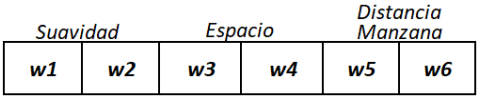

Fig. 5: Representación del cromosoma

A. *Inicialización y Evaluación de la Población*

Al crear la población, cada individuo tendrá una cierta cantidad de pesos asignados con los que comenzará su partida y en cada instante de tiempo el controlador tomará acciones sobre la serpiente, respondiendo a la ecuación (2) considerando los pesos que el algoritmo evolutivo le provee. Esto puede notarse en el módulo de la izquierda presente en la Figura 1. Una vez que finalizada la partida, ya sea porque se mordió a sí misma, chocó o se acabó su límite de movimientos, devolveremos su puntaje y la cantidad de movimientos restantes (en el caso que tenga disponible), por lo que la *función de aptitud* para este trabajo es definida como la cantidad de alimento que consumió cada individuo (puntaje) y la cantidad de tiempo que se mantuvo con vida. Considerando a P y R como el puntaje y la cantidad de movimientos restantes en la partida actual. Además, denotando a Mp como el máximo puntaje alcanzable en el escenario, y a Mr como el máximo de movimientos que tiene disponible cada individuo, representamos la función de aptitud como:

$$g = \frac{P}{\text{Mp}} + \frac{R}{Mr} \qquad (3)$$

Es importante notar que en la ecuación (3) el puntaje y la cantidad de movimientos es normalizada por la cantidad máxima de cada uno de los términos. El máximo fitness que podría obtener un individuo es de 2.

B. *Operador de Selección*

Al finalizar las partidas de toda la población, se producirá la descendencia de los individuos de la actual generación mediante la selección, la cruza y las mutaciones. Para realizar la selección de los individuos se utilizó el *Método de Competencia.* Este método elige k candidatos y realiza una competencia entre ellos (tal como puede notar en la Figura 6) destacando aquellos con mayor aptitud con el fin de trasladarlos a la siguiente generación. De esta manera, procedemos a realizar una cantidad de competencias similar a la cantidad de padres requerida.

C. *Operadores de Cruza, Mutación y Reproducción*

Posteriormente, una vez que se obtuvieron los padres haremos las *Cruzas y Mutaciones* sobre los cromosomas de cada individuo.

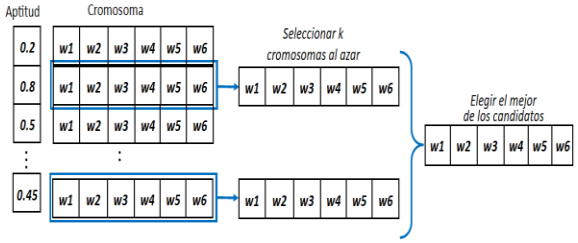

Fig. 6: Método de competencia

Para realizar las Cruzas, se evaluaron dos tipos de operadores, la Cruza Simple, la cual selecciona los cromosomas de dos padres, definiendo un punto aleatorio e intercambiando el material genético antes del punto dado; y la Cruza Aleatoria, que consiste en intercambiar de forma aleatoria los genes de ambos padres formando su descendencia, como se puede ver en la figura 7.

Por otro lado, se estableció el operador de Mutación con cierta probabilidad p_m asociada. Este consiste en agregar a un cromosoma particular un valor aleatorio entre [-1,1] con el fin de mantener la diversidad en la población y promover la búsqueda en el espacio de soluciones.

Finalmente, una vez creada la descendencia y realizadas las mutaciones, definimos aquellos individuos que sobreviven para la próxima generación, por lo que se evaluó la aplicación de diferentes operadores de *reproducción y reemplazo generacional*. En la próxima sección, se detallarán diferentes variantes para los operadores anteriormente mencionados.

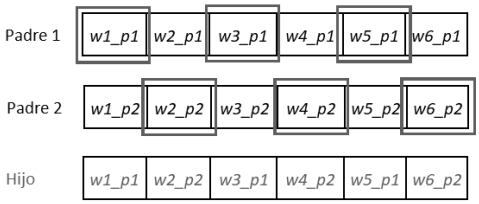

Fig. 7: Cruza Aleatoria

IV. EXPERIMENTOS Y RESULTADOS

Para realizar los experimentos, se utilizó un tamaño de población fija con 20 individuos y 20 generaciones, con una tasa de mutación de 0,05. El escenario utilizado es de 6x6 y la cantidad de movimientos disponibles por cada individuo es de 600 (instantes de tiempo).

Llevamos a cabo 3 experimentos, el primero de ellos consiste en examinar el uso de diferentes operadores de cruza y reproducción, con el fin de evaluar y/o buscar la mejor calidad en la evolución. Posteriormente, evaluamos el impacto de variar la cantidad de candidatos elegidos por el método de la competencia. Finalmente, comprobamos si la utilización de los movimientos restantes en la ecuación (3), provee una ventaja o, por el contrario, resulta mucho más factible utilizar únicamente el puntaje de cada individuo como función de aptitud.

A. *Variación operadores de cruza y reproducción*

Para realizar el primer experimento, se comprobaron cuatro posibles alternativas. En la Tabla 1, pueden verse los diferentes resultados obtenidos (basados en el promedio de 5 ejecuciones para cada variante). Denotamos a CS y CA, como los *operadores de cruza simple y de cruza aleatoria* respectivamente. Así también, definimos RT y BG como los *operadores de reproducción*, correspondientes a reemplazo total y brecha generacional. Para este experimento, se consideró la función de aptitud (3), sin el segundo término. La razón de esto surge del interés de verificar el mayor puntaje que podría llegar la serpiente considerando una combinación de operadores optima que, permita encontrar una buena solución en menor cantidad de generaciones.

Mediante estos datos, se puede notar que la Cruza Aleatoria combinada con la Brecha Generacional, produce mejores resultados que el resto de variantes. La razón de esto,

TABLA I

VARIACIÓN DE DIFERENTES OPERADORES DE CRUZA Y REPRODUCCIÓN

Operadores	CS_RT	CR_RT	CS_BG	CA_BG
MaxAptitud	25	25	28	29
PromAptitud	16.43	17.35	18.10	19.27
PromUltGen	17.552	14.112	16.531	17.234
Máximo	0	0	0	0

recae en el hecho de que la cruza aleatoria produce individuos con mayor variedad frente a la cruza simple. Por otro lado, el uso de la Brecha Generacional, permite que el desempeño sea aún mayor, dado que, padres con buenas aptitudes serán enviados a las generaciones posteriores.

B. Variantes del operador de selección

Este experimento, reflejado en la Tabla II, tiene como objetivo comprobar la utilidad de usar diferente cantidad de candidatos al emplear el *método de competencia*. Nuevamente, sin considerar la cantidad de movimientos restantes en la función (3). Para esto se han recopilado los datos de 10 ejecuciones para cada variante. Los datos de interés son: la Máxima Aptitud y el Promedio obtenido en todas las generaciones, expresados como *MaxAptitud* y *PromAptitud*. El promedio de la última generación, definido como *PromUltGen*. Y la cantidad de veces que se llegó al máximo puntaje, indicado como *Máximo.*

TABLA II

VARIANTES DEL MÉTODO DE SELECCIÓN

Met. Selección	K=2	K=3	K=5	K=10
MaxAptitud	28.18	28.45	30.36	29.00
PromAptitud	17.92	17.86	17.95	17.89
PromUltGen	18.89	18.00	18.78	18.61
Máximo	1	1	2	1

De la tabla anterior, podemos ver que, de 20 generaciones, utilizar 5 candidatos para formar cada uno de los padres, produjo mayores aptitudes que en el resto de las variantes. Además de esto, en tan solo 20 generaciones, la evolución produjo 2 individuos que pudieron obtener el máximo puntaje. Una evolución puntual, utilizando 5 candidatos, puede verse en la Figura 8.

C. Comparación entre funciones de aptitud

En este último experimento, quisimos comparar la utilización de la función de aptitud, tal como se indica en la ecuación (3), en relación a la función utilizada para los experimentos anteriores (sin considerar la cantidad de movimientos restantes). Los resultados de 5 ejecuciones del algoritmo para cada tipo de función, se pueden ver en la Tabla III. De estos, podemos destacar que, la utilización de la función de aptitud original, provee soluciones que no obtienen un puntaje realmente alto, pero el puntaje que adquieren lo consiguen en pocos pasos. Este tipo de función, seria muy útil para el caso de un juego de la serpiente que tenga como restricción un límite de tiempo. Debido a que el juego en este trabajo, no presenta un límite de instantes de tiempo, es preferible utilizar la ecuación (3) sin considerar el término de la derecha.

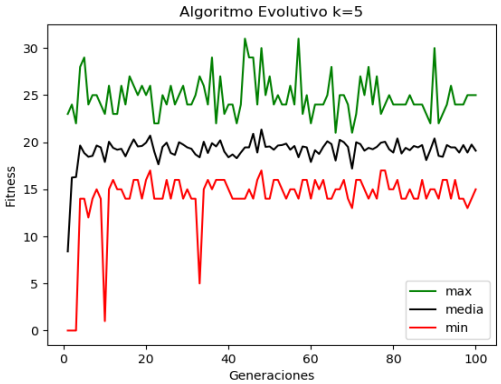

Fig. 8: Algoritmo Evolutivo para k=5

TABLA III

COMPARACIÓN FUNCIONES DE APTITUD

Función fitness	Considerando Movimientos Restantes	Sin considerar movimientos restantes
MaxAptitud	1.34	32
Movimientos Restantes	521	200
Puntaje	18	32

V. CONCLUSIONES

En este trabajo, generamos un controlador basado en 3 funciones de evaluación, asociado a ciertos pesos que fueron optimizados mediante un algoritmo evolutivo. Se evaluaron diferentes variantes de los operadores utilizados y se concluyó que cierta combinación proveía mejores resultados en una menor cantidad de generaciones. Finalmente, se comprobó si es necesario considerar la cantidad de movimientos que realiza y se determinó que, para el juego utilizado, era conveniente obtener un mayor puntaje sin considerar el tiempo que tarda en obtenerlo. En un futuro trabajo, consideraríamos la posibilidad de utilizar otras funciones de evaluación, dado que demanda mucho tiempo la evaluación de los individuos para escenarios muy grandes. Sumado a esto, podría investigarse la resolución del problema, mediante *algoritmos de enjambre de partículas*.

REFERENCIAS

[1] J. F. Yeh, P. H. Siu, S. H Huang, y T. C. Chiang, “Snake Game AI: Movement Rating Functions and Evolutionary Algorithm-based optimization”, *Conference on Technologies and Applications of Artificial Intelligence*, 2016.

[2] S.Kong, y J. A. Mayans, “Automated Snake Game Solvers via AI Search Algorithms”, *CS 271 Introduction to Artificial Intelligence course*, 2014.

[3] G. Kendall, y K. Spoerer, “Scripting the game of Lemmings with a genetic algorithm”, *Proceedings of the 2004 Congress on Evolutionary Computation*, 2004.

[4] O. E. David, H. J. Van Den Herik, M. Koppel, N. S. Netanyahu, “Genetic Algorithms for Evolving Computer Chess Programs”, *IEEE Transactions on Evolutionary Computation*, 2013.

[5] Schaeffer J, Culberson J, Treloar N, Knight B. Lu P, y Szafron D.,” A world championship caliber checkers program”, *Nota de Investigación de Artificial Intelligence*, Pag. 273-289, 1992.

[6] M. Gallager, y A. Ryan, “Learning to play Pac-Man: an evolutionary, rule-based approach”, *Proceedings of IEEE Congress on Evolutionary Computation*, 2003

[7] B. Ma, y M. Tang, J. Zhang, “Exploration of Reinforcement Learning to SNAKE”, *CS 229 Machine Learning Course*, 2016.

[8] P. A. Vikhar, “Evolutionary algorithms: A critical review and its future prospects”, *International Conference on Global Trends in Signal Processing, Information Computing and Communication*, 2016.

[9] D. Câmara, “Evolution and Evolutionary Algorithms”, *ISTE Press Announce Publishing*, Pag.1-30, 2015.