EFECTO DE LA PRECISIÓN DE LAS SEÑALES EN LA EVOLUCIÓN DE LA AGRESIÓN RITUALIZADA

Andrés Sanín Montoya

Director:
Adolfo Amézquita, PhD.

Jurados:
Oscar J. Ramos, MSc.
Orlando Martínez, PhD.

UNIVERSIDAD DE LOS ANDES

FACULTAD DE CIENCIAS

DEPARTAMENTO DE CIENCIAS BIOLÓGICAS

BOGOTÁ, 2004

Resumen

Los modelos de teoría de juegos permiten realizar predicciones sobre la elección de comportamientos, con base en la predicción de estrategias evolutivamente estables (ESS). Una estrategia pura consiste en realizar siempre un mismo comportamiento, y una estrategia mixta consiste en realizar diferentes comportamientos con determinada probabilidad. Se ha probado que cuando existe una asimetría en una interacción agonística, únicamente son estables las estrategias puras. Sin embargo, se ha sugerido que las estrategias mixtas pueden ser estables si la asimetría no es observada de forma perfecta por los individuos, y una forma de observar imperfectamente la asimetría podría ser la imprecisión que usualmente tienen las señales que sirven como indicadores honestos de alguna propiedad. En el presente estudio se construyó un programa que permite simular la evolución de la estrategia de reacción, en una población en la que sus individuos participan por un recurso y modifican su *fitness* según el beneficio de ganar y el costo de cada enfrentamiento. Se realizaron simulaciones variando la precisión con la que los individuos señalan su tamaño, con diferentes valores de la relación beneficio/costo, para observar el efecto de la precisión de las señales, como indicadores honestos del tamaño, sobre las estrategias de reacción. Posteriormente, se realizaron simulaciones utilizando valores de precisión dentro del rango observado en diferentes poblaciones de *Epipedobates femoralis*, tomando como medida de precisión el coeficiente de determinación resultante de la regresión frecuencia-longitud corporal. Los resultados de las simulaciones se compararon con los resultados de experimentos de playback realizados en las mismas poblaciones, que permiten estimar las estrategias de reacción. Se encontró que la imprecisión puede generar estrategias mixtas estables, y que al aumentar la precisión de las señales, la estrategia resultante consiste en: aumentar la proporción de comportamientos agresivos, si el oponente es más pequeño según su tamaño señalado, y disminuir la proporción de comportamientos agresivos en el caso contrario. La misma relación se encontró al utilizar los rangos de precisión observados para *E. femoralis*. No se observó una relación significativa entre la precisión observada en cada población y los comportamientos observados en los experimentos de playback. Sin embargo, la latencia de orientación hacia la fuente de sonido, que probablemente es la variable que mejor se relaciona con la agresividad, parece disminuir en poblaciones donde la precisión observada es mayor, al utilizar estímulos de alta frecuencia. Por esta razón, no se puede descartar la posibilidad de que los individuos de *E. femoralis* utilicen la información del tamaño corporal codificada en la frecuencia dominante de sus llamadas.

Introducción

Durante la segunda mitad del siglo pasado, el estudio del comportamiento animal presentó un cambio de énfasis desde la causa hacia la función (Huntingford 1991). La causa, se refiere a los mecanismos que operan sobre un individuo para producir un comportamiento determinado, mientras que la función, se refiere al efecto que produce el comportamiento escogido por el individuo, en términos de costos y beneficios. Esta tendencia hacia el estudio de la función de los comportamientos se relaciona en parte con la utilización de modelos de teoría de juegos, los cuales intentan explicar cada comportamiento desde un punto de vista evolutivo, y en la mayoría de los casos lo hacen mejor que los modelos etológicos tradicionales (Caryl 1979). Los modelos basados en teoría de juegos buscan una estrategia especial para elegir comportamientos, la cual, al ser adoptada por los miembros de una población no puede ser reemplazada por ninguna otra estrategia alternativa que pueda surgir. Dicha estrategia especial es conocida como 'estrategia evolutivamente estable' o ESS (Maynard Smith 1974).

Otra forma de predecir comportamientos es utilizar simulaciones. Por medio de un programa que simule las interacciones entre los individuos de una población, es posible observar el cambio en la estrategia de reacción durante varias generaciones. Hay algunas ventajas al utilizar simulaciones, como la facilidad de incluir características particulares del grupo de estudio y la facilidad de variar las mismas para observar el efecto sobre la situación final. Además, cuando se incluyen varios factores, usualmente, los modelos matemáticos se vuelven muy complejos y en estos casos una simulación puede tener mayor claridad. Un grupo de simulaciones, muy utilizado en estudios de inteligencia artificial, son las 'redes neurales artificiales', donde una serie de unidades computacionales realizan operaciones análogas a las de los sistemas nerviosos. Las redes neurales resultan especiales para simular la forma en que un receptor es estimulado por una señal (e.g. Ryan *et al.* 2001, Phelps & Ryan 1998), y permiten hacer predicciones de sus respuestas. Al poner a prueba las predicciones de las redes neurales con experimentos reales, se han obtenido resultados satisfactorios (e.g. Phelps 2001), por lo cual las simulaciones pueden ser valiosas para predecir comportamientos.

En varios grupos animales, los individuos utilizan señales ritualizadas que les permiten resolver conflictos evitando las agresiones físicas (Maynard Smith & Price 1973). En este tipo de interacciones, usualmente el comportamiento más adecuado (en términos de *fitness*) que pueda escoger un individuo depende del comportamiento del otro individuo (Maynard Smith 1974), y en esta situación es ideal la teoría de juegos. La clave para estudiar las interacciones agonísticas y lograr predicciones acertadas de los comportamientos utilizados, es modelar de forma detallada y precisa la situación particular que se esté estudiando (Austad 1983).

Cuando se utilizan las señales ritualizadas, los individuos que participan en un enfrentamiento obtienen algún tipo de información. Los modelos que toman en cuenta cualquier tipo de información, que permita diferenciar los oponentes en un enfrentamiento, se conocen como modelos asimétricos (Maynard Smith 1982). Generalmente en estos casos, la estrategia para elegir el comportamiento más adecuado cambia según la posición que ocupa un individuo en la asimetría, es decir, si se trata de una asimetría en el tamaño corporal, el individuo actuará de una forma si es más grande que su oponente y de otra forma si es más pequeño (Maynard Smith 1982). Selten (1980) demostró que únicamente las estrategias puras pueden ser evolutivamente estables en juegos asimétricos. Una estrategia pura consiste en actuar siempre con un comportamiento determinado, contrario a lo que ocurre en una estrategia mixta, que consiste en escoger entre diferentes comportamientos, cada uno con una probabilidad determinada (Maynard Smith 1982). Los resultados de Selten implican que nunca se deberían observar estrategias mixtas, ya que en las interacciones entre animales es muy probable que siempre exista alguna asimetría, sobre la cual los animales puedan condicionar su comportamiento (Binmore & Samuelson 2001). Como consecuencia, se favorecerían estrategias que evitan la posibilidad de que dos oponentes opten por comportamientos agresivos simultáneamente, porque usualmente esta situación es desfavorable para ambos, y por lo tanto no se observarían conflictos.

A pesar de las inferencias de los resultados de Selten, en la naturaleza ocurren conflictos, y es común observar que un mismo individuo opte por diferentes comportamientos en situaciones similares, razón por la cual parece más probable que se estén utilizando estrategias mixtas (Binmore & Samuelson 2001). Se ha observado que si los individuos ignoran parte de la información que genera la asimetría, pueden existir estrategias mixtas que son evolutivamente estables (Crowley 2000, Binmore & Samuelson 2001).

En las interacciones agonísticas, las señales ritualizadas proporcionan la información que genera la asimetría. Tanto emisor como receptor se benefician de un sistema de comunicación en el cual se utilizan señales que trasmiten información honesta, sobre los atributos del emisor (Kodric-Brown & Brown 1984, Zahavi & Zahavi 1997). Sin embargo, el costo que puede tener un receptor en términos de energía, tiempo, o riesgo, para evaluar la información, puede causar una preferencia por señales menos elaboradas y menos precisas (Dawkins & Guilford 1991). Además, puede surgir deshonestidad por parte de los individuos que siempre serían perdedores del enfrentamiento (Grafen 1987). Como consecuencia, existe imprecisión en la información de las señales, y esta imprecisión podría tener el mismo efecto que ignorar parte de la información, permitiendo la utilización de estrategias mixtas.

En el presente estudio, se creó un programa que simula las interacciones entre los individuos de una población, los cuales compiten por un recurso utilizando señales ritualizadas, y tienen una estrategia de reacción que puede variar entre generaciones. La estrategia de reacción de un individuo consiste en la

probabilidad con la cual se escoge cada uno de los comportamientos que puede presentar. El programa permite variar la precisión de la información incluida en las señales. Aunque el modelo permite plantear hipótesis generales, resulta adecuado utilizar un grupo particular para probar predicciones más específicas. Se utilizó como modelo de estudio una especie de rana venenosa, *Epipedobates femoralis*, cuyos machos participan en interacciones agonísticas, y normalmente responden al canto de otro macho de dos formas: se aproximan a éste último con el propósito de combatir físicamente, o cantan 'antifónicamente' de tal forma que intercalan sus llamadas con las llamadas del otro macho (Hödl 1987). Como ocurre en varias especies de anuros (e.g. Sullivan 1982, Ryan 1985, Robertson 1986, Howard & Young 1998), la frecuencia dominante de las llamadas presenta una correlación negativa con el tamaño corporal, y por lo tanto, la frecuencia dominante de las llamadas emitidas por un macho, tiene el potencial de servir como indicador honesto de su tamaño.

El objetivo del estudio es predecir el efecto que tiene la precisión de las señales, que sirven como indicadores honestos del tamaño de los individuos, sobre la estrategia de elección entre los comportamientos posibles y compararlo con el efecto observado en diferentes poblaciones de *E. femoralis*. Con este estudio se tratará de comprobar si las estrategias mixtas pueden ser estables, cuando la asimetría no es completamente conocida para los individuos. Al comparar las predicciones con los datos empíricos, se podrá determinar la importancia real que tiene la precisión de las señales en la evolución de la agresión ritualizada. Finalmente, los resultados aportarán evidencia acerca de la posibilidad de que los machos de *E. femoralis* utilicen la información del tamaño corporal, proporcionada por la frecuencia dominante de sus llamadas, durante las interacciones agonísticas.

Metodología

Construcción del programa

El programa se escribió en Java™, utilizando la plataforma Eclipse v.2.1.3. El propósito del programa es simular la evolución de la estrategia de reacción en una población, al ocurrir interacciones agonísticas entre los individuos. Para cada individuo se almacena información acerca de: tamaño, *fitness*, probabilidad de reproducirse y estrategia de reacción. Cada individuo tiene dos comportamientos posibles: agresivo y no agresivo. La estrategia de reacciónes la probabilidad con la que el individuo escoge cada uno de los comportamientos, según su tamaño relativo (oponente más pequeño – oponente más grande); como hay dos comportamientos, se tomó como estrategia de reacción la frecuencia del comportamiento agresivo, para cada tamaño relativo (la frecuencia del comportamiento no agresivo corresponde a la frecuencia restante). Luego se crea una población, según las variables que introduzca el usuario: número de individuos (n), distribución de los tamaños corporales (μ y σ). La población tiene una dinámica en donde cada individuo se enfrenta con otro individuo de la población y se calcula el cambio en el *fitness* para cada uno, según el comportamiento escogido por ambos. En un enfrentamiento, cada individuo estima su tamaño relativo, comparando su tamaño real con el tamaño señalado por su oponente. Cada individuo produce un número de descendientes proporcional a su nuevo valor de *fitness*, el cual depende únicamente de los resultados de las interacciones en que participó (porque inicialmente el *fitness* es 0). Sus descendientes heredan la estrategia de reacción, que pueden cambiar según la tasa de mutación introducida por el usuario. Una vez todos los individuos se han reproducido, sus descendientes reemplazan la generación actual. Si se excede el tamaño de la población (n), se eliminan los sobrantes escogiendo individuos fortuitamente, de tal forma que el individuo con más descendientes tiene mayor probabilidad de 'transmitir' su estrategia de reacción a la siguiente generación.

El tamaño señalado por un individuo se calcula según el valor de precisión introducido por el usuario. El método para calcular el tamaño señalado según la precisión, fue calibrado para que el valor de precisión coincidiera con el coeficiente de determinación, calculado de la regresión entre el tamaño señalado y el tamaño real (Figura 1). Esto se hizo con el propósito de poder comparar los valores de precisión utilizados en el modelo, con valores de precisión observados en poblaciones reales, usualmente estimados con el coeficiente de determinación resultante de una regresión.

El programa también permite al usuario variar los siguientes parámetros: el número de generaciones por las que se repite el ciclo, el valor (en unidades de *fitness*) para el beneficio de ganar un enfrentamiento, el valor (en unidades de *fitness*) del costo de un enfrentamiento, y el modelo a utilizar en la simulación. Con los valores de costos y beneficios, el programa genera una matriz con los resultados (cambio en el *fitness*) para todas las posibles combinaciones de

enfrentamientos, según el modelo. En el presente estudio, se utilizaron dos modelos basados en el juego de Halcón/Paloma (Apéndice A): 1. modelo Asimetría de Tamaños Desconocida, y 2. modelo Precisión Fija.

Comprobación del programa

Todas las simulaciones del estudio utilizaron 500 generaciones, para garantizar que se pudiera llegar a la ESS, y la estrategia resultante se calculó como el promedio de la estrategia obtenida para las últimas 100 generaciones (para no tomar los casos antes de llegar a la ESS). El modelo Asimetría de Tamaños Desconocida (Apéndice A, parte b) se utilizó para comprobar el correcto funcionamiento del programa. Con éste modelo se realizaron varias simulaciones variando los parámetros: beneficio (V) y costo (K), para comparar las estrategias obtenidas por la simulación, con las ESS calculadas matemáticamente utilizando el modelo de teoría de juegos (Apéndice B). Las estrategias obtenidas por ambos métodos coincidieron (Figura 2).

Adicionalmente, se realizaron simulaciones variando los parámetros: tamaño de la población (n) y tasa de mutación (t.m), para ver si éstos afectan la estrategia de reacción resultante. Se observó que aunque pueden aumentar la fluctuación de la estrategia de reacción entre generaciones, la estrategia promedio resultante no depende de dichos parámetros (Figura 3).

Efecto de la precisión de las señales

El modelo Precisión Fija (Apéndice A, parte c) se utilizó para determinar el efecto de la precisión de la información del tamaño corporal, sobre la evolución de la estrategia de reacción. Éste modelo es asimétrico, porque incluye señales que le indican a cada individuo el tamaño de su oponente. La estrategia de reacción de cada individuo se divide en dos, una estrategia que utiliza cuando el oponente es más pequeño, y otra que utiliza cuando el oponente es más grande.

Con el fin de determinar la relación entre la precisión y la estrategia de reacción, se realizaron simulaciones variando los valores de precisión, desde 0%, incrementando en 10% hasta 100%, tomando el valor de la estrategia de reacción resultante. Se realizó una regresión lineal, tomando la estrategia de reacción resultante como variable dependiente, y la precisión como variable independiente. El procedimiento anterior se repitió con diferentes valores de la proporción beneficio/costo (V/K), donde V es el valor del beneficio de ganar el recurso y K es el costo de un enfrentamiento, con el propósito de observar la interacción entre la precisión y la relación V/K. Se utilizaron los siguientes valores de la proporción V/K: 1/3, 1/2, 2/3, 1/1, 3/2, 2/1 y 3/1. Se observó que con valores más extremos se produce únicamente un comportamiento (agresivo si $V/K > 1/4$, y no agresivo si $V/K < 1/4$).

Comparación con datos de E. femoralis

De la base de datos del Proyecto Femoralis (ver Hödl *et al*. 2004, Narins *et al*. 2003 y 2005, Amézquita *et al*. en prensa), se utilizaron los siguientes datos de ocho poblaciones de *Epipedobates femoralis* del Amazonas: 1. un muestreo del tamaño corporal (SVL) y la frecuencia dominante (de la última nota) de las llamadas de los machos de la población; y 2. los resultados de un experimento de playback, en el cual se utilizaron como estímulo llamadas sintéticas que varían en su frecuencia dominante, sumando o restando entre 0 y 6 desviaciones estándar (de la especie). La reacción de cada individuo al que se le presentó el estímulo, se midió con las siguientes variables: 1. *Respuesta*, 1 si se llegó hasta la fuente de sonido y 0 si no llegó. 2. *Velocidad de aproximación*, la distancia entre el macho y la fuente de sonido dividida por el tiempo que tardó el macho en llegar a la fuente de sonido. 3. *Latencia de orientación*, el tiempo desde el inicio del estímulo hasta la primera reorientación del macho hacia la fuente de sonido. 4. *Latencia de salto*, el tiempo desde el inicio del estímulo hasta el primer salto del macho hacia la fuente de sonido.

Como medida de precisión de la información del tamaño corporal, se utilizó el coeficiente de determinación (r^2) resultante de la regresión entre la frecuencia dominante de las llamadas y el SVL, para cada población. Como los valores estimados de la precisión de las señales de las ocho poblaciones se encontraron entre 0 y 0.3, se realizaron simulaciones variando los valores de precisión, desde 0%, incrementando en 5% hasta 30%, tomando el valor de la estrategia de reacción resultante, con el fin de realizar predicciones específicas para el rango observado. El procedimiento anterior se repitió para el mismo rango de valores de la relación V/K utilizado en la anterior sección. El programa utiliza (por defecto) como parámetros la media y la desviación del SVL estimadas para *E. femoralis* (μ = 28.7 mm y σ = 1.3 mm).

Como medida de la estrategia de reacción, se utilizaron las respuestas medidas en el experimento de playback. Puesto a que el experimento contiene estímulos supranormales con respecto a la frecuencia dominante, se escogieron únicamente los casos en los cuales la probabilidad de respuesta fuera mayor a 0.5. La probabilidad de respuesta se calculó utilizando una regresión logística entre la variable *respuesta* y la frecuencia del estímulo. En el experimento de playback no se midió el tamaño de los individuos, por esta razón, los estímulos con frecuencia dominante mayor a la media poblacional se tomaron como estímulos que representan un oponente pequeño, y los estímulos con frecuencia dominante menor a la media poblacional se tomaron como estímulos que representan un oponente grande. La estrategia de reacción se calculó con el promedio de cada una de las cuatro variables medidas (porcentaje en el caso de la variable *respuesta*), en cada población, para cada tipo de estímulo (alta frecuencia, baja frecuencia).

Con el fin de observar el efecto de la precisión de las señales potenciales de los individuos de *E. femoralis*, sobre su estrategia de reacción, se examinó la relación entre cada variable respuesta y la precisión observada (r^2) para cada población, por medio de una regresión lineal tomando cada variable respuesta como variable dependiente y la precisión observada como variable independiente. En cada caso, se realizó un análisis separado para cada tipo de estímulo (alta frecuencia, baja frecuencia).

Finalmente, para cada variable respuesta, se comparó el resultado obtenido con los dos tipos de estímulos (alta frecuencia, baja frecuencia) con una prueba T de muestras independientes, con el fín de determinar si los individuos responden de forma distinta a estímulos con frecuencia diferente. Para todos los análisis estadísticos utilicé el paquete estadístico SPSS 10.0.

Resultados

Efecto de la precisión de las señales

Cuando el individuo es más grande que el tamaño señalado por su oponente, aumenta la probabilidad de producir el comportamiento agresivo (Figura 4, parte a); mientras que cuando el individuo es más pequeño que el tamaño señalado por su oponente, disminuye la probabilidad de producir el comportamiento agresivo (Figura 4, parte b).

Existe una interacción entre la precisión de las señales y la proporción beneficio/costo, que depende del tamaño relativo del individuo. Cuando el oponente es más pequeño, el efecto de la precisión sobre la estrategia de reacción disminuye al aumentar la relación V/K; si ésta relación es 2/1 o más, únicamente se produce el comportamiento agresivo. Cuando el oponente es más grande, el efecto de la precisión sobre la estrategia de reacción aumenta al aumentar la relación V/K; si ésta relación es 1/3 o menos, únicamente se produce el comportamiento no agresivo. Los resultados de los análisis de regresión se muestran en la Tabla 1.

Comparación con datos de E. femoralis

Al utilizar valores de precisión dentro del rango 0-30%, se mantiene la relación entre la probabilidad de producir el comportamiento agresivo y la precisión utilizada (Figura 5). Sin embargo, la interacción entre la precisión de las señales y la proporción beneficio/costo no es la misma que se observó para el rango entero de valores de precisión (0-100%). Los resultados de los análisis de regresión se muestran en la Tabla 2.

Entre las variables medidas como respuesta en los experimentos de playback, únicamente la latencia de salto parece disminuir en poblaciones donde la precisión de las señales es mayor (Figura 6), sin embargo la relación no es significativa (reg. lineal, $r^2 = 0.41$, $n = 8$, $p = 0.08$). En los demás casos, no se presentó una relación significativa ($n = 8$, $p > 0.3$ en todos los casos).

De igual forma, la latencia del salto fue la única respuesta que pareció variar al utilizar estímulos con frecuencia diferente; sin embargo la diferencia no es significativa (prueba T, $t = -2.03$, g.l. = 14, $p = 0.06$). En los demás casos, no se presentó una diferencia significativa (prueba T, g.l. = 14, $p > 0.1$ en todos los casos).

Discusión

Los resultados de la simulación permiten determinar si la estrategia de reacción en una población es evolutivamente estable, según la variación que se presente entre las diferentes generaciones. El programa funciona de forma acertada, ya que al aplicar un modelo teórico, cuyos resultados se pueden calcular matemáticamente, los resultados de la simulación concuerdan con los resultados teóricos. La precisión de las señales como indicadores del tamaño corporal tiene un efecto significativo sobre la evolución de la estrategia de reacción, y el efecto depende de la relación entre el beneficio de ganar un recurso y el costo de un enfrentamiento. Según los resultados de las simulaciones, cuando las señales en una población son más precisas indicando el tamaño de los individuos, la estrategia de reacción evolutivamente estable de un individuo consiste en: ser más agresivo si interactúa con un oponente de menor tamaño, y ser menos agresivo si interactúa con un oponente de mayor tamaño. Si el beneficio es mayor que el costo, la estrategia de reacción tiende a presentar en gran proporción comportamientos agresivos, cuando el oponente es más pequeño, por esta razón, el efecto de la precisión disminuye. Por otro lado, si el costo es mayor o igual que el beneficio, la estrategia de reacción tiende a presentar en gran proporción comportamientos no agresivos, cuando el oponente es más grande; por esta razón, el efecto de la precisión disminuye también en este caso. Los resultados obtenidos de datos empíricos no presentan evidencia a favor de los resultados obtenidos por las simulaciones.

El modelo Precisión Fija fue utilizado para las simulaciones que incluyen la información del tamaño en las interacciones de la población. En este modelo, se utiliza un valor de precisión para calcular el tamaño señalado por los individuos de una población. En la naturaleza, es común que las señales que sirven como indicadores honestos de alguna característica estén ligadas a la misma; esto se ha sugerido, por ejemplo, para animales que producen señales auditivas como anuros (e.g. Ryan 1988, Gerhardt 1994, McClelland *et al.* 1996) y aves (e.g. Ryan & Brenowitz 1985), en los cuales el tamaño corporal puede imponer restricciones sobre los aparatos que producen los sonidos. Por ejemplo, en los anuros, la frecuencia de la llamada se determina en gran medida por la masa de las cuerdas vocales (Martin 1972, Ryan 1986), de tal forma que el tamaño de los individuos, al relacionarse con la masa de sus cuerdas vocales, restringe la frecuencia de sus llamadas. Según esto, el rango de variación de las propiedades de las señales, estaría restringido por la morfología de los individuos de una población, y podría afectar la precisión observada de las señales en la población.

Cuando las señales que indican el tamaño corporal del individuo son totalmente precisas (100%), los individuos siempre producen el comportamiento agresivo cuando su oponente es más pequeño, y el comportamiento no agresivo cuando oponente es más grande. Los porcentajes de cada comportamiento no alcanzan a llegar a cien por ciento, porque la

simulación incluye la tasa de mutación que genera variaciones en la estrategia de reacción en cada generación. Este resultado concuerda con anteriores estudios, los cuales concluyen que al existir una asimetría, únicamente las estrategias puras pueden ser estables (Selten 1980, Enquist & Leimar 1983). Si la naturaleza presentara este tipo de señales, los individuos siempre presentarían el mismo comportamiento y nunca se presentarían enfrentamientos.

Sin embargo, cuando la información de las señales es menos precisa, pueden existir estrategias mixtas estables. En realidad, las señales que tienen el potencial para indicar las propiedades de un individuo suelen ser poco precisas. Por ejemplo, la mayoría de los coeficientes de determinación resultantes de la regresión entre la frecuencia dominante de las llamadas de los anuros y su tamaño corporal, se encuentra por debajo de 0.50 (revisión hecha por Amézquita, 2001). Algunas explicaciones para la existencia de señales imprecisas son: 1. El riesgo que puede tener un receptor en términos de energía, tiempo, o riesgo para evaluar la información, hace que prefiera señales menos elaboradas y menos precisas (Dawkins & Guilford 1991). 2. Pueden generarse sistemas de comunicación deshonestos, porque los individuos tienden a no respetar una asimetría que siempre los coloque como perdedores de un enfrentamiento (Grafen 1987, Bee *et al*. 2000). 3. La misma señal puede utilizarse para transmitir un segundo tipo de información, de tal forma que el individuo modifica las características de la señal. Aunque se genera una imprecisión con respecto a la información original, no existe una 'intención' deshonesta (Wagner 1992).

Anteriormente, se había demostrado que las estrategias mixtas pueden ser estables cuando la asimetría es observada de forma imperfecta (Binmore & Samuelson 2001) o cuando existe una simetría mixta en la cual únicamente se conoce parte de la asimetría (Crowley 2000). En ambos casos, se sugiere que no existe una clara división entre los juegos simétricos y los juegos asimétricos. El cambio en la precisión de las señales en este estudio, se puede interpretar como un cambio en la asimetría: al disminuir la precisión, cada individuo tiene menor certeza de la información del tamaño de su oponente, y cuando la precisión es nula, la información dejaría de ser útil de tal forma que corresponde a un modelo simétrico. Por esta razón, es posible que la precisión de las señales genere un continuo entre una situación asimétrica (100% de precisión) y una situación simétrica (0% precisión).

El comportamiento promedio de un individuo, cuando es más grande que su oponente, disminuye en agresividad a medida que las señales son menos precisas, ya que aumenta la posibilidad de que en realidad no sea más grande que el oponente. Cuando el individuo es más pequeño que su oponente, ocurre lo contrario, aunque el efecto de la precisión generalmente es menor, posiblemente porque el error más común en este caso (enfrentarse con un oponente mayor) es más costoso que el error más común en el caso anterior (dejar de enfrentarse con un oponente menor).

El efecto de la relación entre el beneficio de ganar un recurso y el costo de un enfrentamiento ya se ha probado con otros modelos, soportados por datos empíricos (Enquist & Leimar 1987). En este estudio se observó que cuando un individuo es más grande que su oponente, y el beneficio del recurso supera el costo del enfrentamiento, se observa en la población un alto porcentaje del comportamiento agresivo, sin importar el valor de la precisión. Por lo tanto, ocurre una disminución del efecto de la precisión de las señales debido al alto valor del recurso, el cual hace que los individuos busquen siempre un enfrentamiento, ya que el beneficio de ganar supera el costo de perder una, o varias veces. Por otro lado, cuando un individuo es más pequeño que su oponente, y el beneficio del recurso no supera el costo del enfrentamiento; en la población se observa un alto porcentaje del comportamiento no agresivo, sin importar el valor de la precisión. Por lo tanto, ocurre una disminución del efecto de la precisión de las señales debido al alto costo del enfrentamiento, el cual hace que los individuos eviten participar en enfrentamientos, ya que el costo de perder supera el beneficio de ganar una, o varias veces.

No se observó un efecto significativo de la precisión de las señales de *E. femoralis* (estimada por el coeficiente de determinación resultante de la regresión entre la frecuencia dominante de las llamadas y el SVL), sobre su estrategia de reacción (estimada por las respuestas promedio observadas en experimentos de playback). Sin embargo, entre las variables respuesta medidas en experimentos de playback, la única variable que parece presentar diferencias en la respuesta según la frecuencia del estímulo es la latencia del salto. Por esta razón, este comportamiento puede ser el que mejor refleja la agresividad, de forma que una reacción agresiva consiste en presentar un menor tiempo de latencia, es decir, saltando más rápido hacia la fuente de sonido que presenta el estímulo. Esa misma variable es la única que indica una posible relación con la precisión observada de las señales. Cuando se utilizaron estímulos de alta frecuencia, los individuos de poblaciones con un mayor coeficiente de determinación (de la regresión frecuencia-SVL) presentaron una tendencia a disminuir la latencia del salto. Este resultado estaría de acuerdo con la predicción obtenida por las simulaciones, en donde el comportamiento agresivo aumenta con la precisión de las señales, cuando los individuos son más grandes que sus oponentes.

Al tomar como modelo de estudio *E. femoralis*, se esperaba que presentara comportamientos discretos que se pudieran categorizar como agresivos y no agresivos. La variable *respuesta* (llegar o no a la fuente de sonido), es poco informativa y parece depender únicamente del reconocimiento de la señal por parte del receptor. Las variables continuas son más informativas y parecen ser un mejor indicativo de la agresividad de la respuesta. Probablemente los individuos de esta especie no presentan comportamientos agresivos o no agresivos, sino un continuo de comportamientos que varían en niveles de agresividad (e.g. la latencia del salto). Este fenómeno es muy común en los grupos que presentan sistemas de comunicación graduados (e.g. Wagner 1989, Wells 1989, Grafe 1995). Aunque el modelo utilizado podría ajustarse a este tipo de situaciones, de forma que el aumento en el porcentaje del

comportamiento agresivo se tome como el aumento en el nivel de agresividad, lo ideal sería aplicar un modelo más adecuado, como la guerra de desgaste ('war of attrition', Maynard Smith 1982).

El modelo utilizado asume que la información de las señales se utiliza en las interacciones entre los individuos. Para que esto ocurra, no basta con que exista una señal que codifica la información de alguna propiedad. Se requiere que los receptores del grupo de estudio tengan la capacidad de distinguir de forma acertada pequeños cambios en dichas señales (Ryan 1988). Otro aspecto a tener en cuenta con respecto a la aplicabilidad del modelo, es la estimación de la precisión de las señales de una población, ya que los parámetros de la regresión, incluyendo el coeficiente de determinación y la pendiente, pueden variar según el tamaño de muestreo (e.g. Amézquita 2002). Por esta razón, es mejor probar las predicciones del modelo utilizando un grupo de estudio cuyas poblaciones presenten diferencias más amplias en la precisión de la información de sus señales.

En conclusión, la simulación utilizada sugiere que pueden existir estrategias mixtas estables cuando la información de las señales no es del todo precisa. Además, sugiere que la precisión de las señales tiene un efecto sobre la estrategia de reacción, dependiendo del tamaño relativo del receptor. Estas predicciones sirven como herramientas para examinar la probabilidad de que una señal, que tiene el potencial de ser un indicador honesto, sea realmente utilizada por los receptores en una población. Aunque los datos empíricos no presentan relaciones significativas que comprueben las predicciones, hay indicios que sugieren que la precisión de la información si podría tener un efecto, y sería incorrecto concluir que los receptores de *E. femoralis* no utilizan la información del tamaño corporal codificada en la frecuencia dominante de sus llamadas. Se recomienda que el modelo sea probado en otros grupos con mayor variación en la precisión de la información de las señales, y que se incluya la utilización de niveles de agresividad.

Referencias

Amézquita, A. 2002. Signal diversity and the evolution of the vocal communication system in the high-Andean frog *Hyla labialis*. Ch. 3. Calling performance and the potential for honest signalling in the communication system of the high-Andean frog *Hyla labialis*. PhD Thesis, Universidad de los Andes, Bogotá.

Amézquita, A., L. Castellanos & W. Hödl. (en prensa): Auditory tuning of male *Epipedobates femoralis* (Anura: Dendrobatidae) under field conditions: the role of spectral and temporal call features. Anim. Behav.

Austad, S. N. 1983. A game theoretical interpretation of male combat in the bowland doily spider *(Frontinella pyramitela)*. Anim. Behav. 31: 59-73.

Bee, M. A., S. A. Perrill & P. C. Owen. 2000. Male green frogs lower the pitch of acoustic signals in defense of territories: a possible dishonest signal of size? Behav. Ecol. Vol. 11 No. 2: 169-177.

Binmore, K. & L. Samuelson. 2001. Can mixed strategies be stable in asymmetric games? J. Theor. Biol. 210: 1-14.

Caryl, P. G. 1979. Communication by agonistic displays: what can games theory contribute to ethology? Behaviour. 68: 136-169.

Crowley, P. H. 2000. Hawks, doves, and mixed-symmetry games. J. Theor. Biol. 204: 543-563.

Dawkins, M. S. & T. Guilford. 1991. The corruption of honest signalling. Anim. Behav. 41: 865-873.

Enquist, M. & O. Leimar. 1983. Evolution of fighting behaviour: decision rules and assessment of relative strength. J. Theor. Biol. 102: 387-410.

Enquist, M. & O. Leimar. 1987. Evolution of fighting behaviour: the effect of variation in resource value. J. Theor. Biol. 127: 187-205.

Gerhardt, H. C. 1994. The evolution of vocalization in frogs and toads. Annu. Rev. Ecol. Syst. 25: 293-324.

Grafe, T. U. 1995. Graded aggressive calls in the African painted reed frog *Hyperolius marmoratus* (Hyperoliidae). Ethology 101: 67-81.

Grafen, A. 1987. The logic of divisively asymmetric contests: respect for ownership and the desperado effect. Anim. Behav. 35: 462-467.

Hödl, W. 1987. *Dendrobates femoralis* (Dendrobatidae): a handy fellow for frog bioacoustics. In: Proceedings of 4th Ord. Gen. Mtg. Soc. Eur. Herpetol. Pp: 201-204.

Hödl, W., A. Amézquita & P. M. Narins. 2004. The rôle of call frequency and the auditory papillae in phonotactic behavior in male Dart-poison frogs *Epipedobates femoralis* (Dendrobatidae). J. Comp. Physiol. A 190: 823-829.

Howard, R. D. & J. R. Young. 1998. Individual variation in male vocal traits and female mating preferences in *Bufo americanus*. Anim. Behav. 55: 1165-1179.

Huntingford, F. A. 1991. War and peace revisited. In: The Tinbergen legacy (ed R. Dawkins). Chapman & Hall. London. Pp: 40-59.

Kodric-Brown, A. & J. H. Brown. 1984. Truth in advertising: the kinds of traits favored by sexual selection. Am. Nat. 124: 309-323.

Martin, W. F. 1972. Evolution of vocalizations in the genus *Bufo*. In: Evolution in the genus *Bufo* (ed W. F. Blair). University of Texas Press. Austin. Pp: 279-309.

Maynard Smith, J. 1974. The theory of games and the evolution of animal conflicts. J. Theor. Biol. 47: 209-221.

Maynard Smith, J. 1982. Evolution and the theory of games. Cambridge University Press. New York.

Maynard Smith, J. & Price, G. R. 1973. The logic of animal conflict. Nature, Lond. 246: 15-18.

McClelland. B. E., W. Wilczynski & M. J. Ryan. 1996. Correlation between call characteristics and morphology in male cricket frogs (*Acris crepitans*). J. Exp. Biol. 199: 1907-1919.

Narins, P. M., W. Hödl & D. S. Grabul. 2003. Bimodal signal requisite for agonistic behavior in a dart-poison frog *Epipedobates femoralis*. Proc. Nat. Acad. Sci. USA 100: 577-580.

Narins, P. M., D. S. Grabul, K. K. Soma, P. Gaudner & W. Hödl. 2005. Cross-modal integration in a dart-poison frog. Proc. Nat. Acad. Sci. USA (en prensa).

Phelps, S. M. 2001. History's lessons: A neural network approach to receiver biases and the evolution of communication. In: Anuran Communication (ed M. J. Ryan). Smithsonian Institution Press. Washington and London.

Phelps, S. M. & M. J. Ryan. 1998. Neural networks predict response biases of female túngara frogs. Proc. R. Soc. Lond. B. 265: 279-285.

Robertson, J. G. M. 1986. Female choice, male strategies and the role of vocalizations in the Australian frog *Uperoleia rugosa*. Anim. Behav. 34: 773-784.

Ryan, M. J. 1985. The Túngara Frog. A Study in Sexual Selection and Communication. University of Chicago Press. Chicago.

Ryan, M. J. 1986. Factors influencing the evolution of acoustic communication: biological constraints. Brain Behav. Evol. 28: 70-82.

Ryan, M. J. 1988. Constraints and patterns in the evolution of anuran acoustic communication. In: The evolution of the Amphibian Auditory System (eds B. Fritzsch, M. J. Ryan, W. Wilczynski, T. E. Hetherington & W. Walkowiak). John Wiley & Sons. New York. Pp: 637-675.

Ryan, M. J. & E. A. Brenowitz. 1985. The role of body size, phylogeny, and ambient noise in the evolution of bird song. Amer. Nat. 126: 87-100.

Ryan, M. J., S. M. Phelps & A. S. Rand. 2001. How evolutionary history shapes recognition mechanisms. Trends in Cognitive Science. Vol.5 No.4: 143-148.

Selten, R. 1980. A note on evolutionarily stable strategies in asymmetric animal contests. J. Theor. Biol. 84: 93-101.

Sullivan, B. K. 1982. Significance of size, temperature and call attributes to sexual selection in *Bufo woodhousei australis*. Herpetol. 16: 103-106.

Wagner, W. E., Jr. 1989. Graded aggressive signals in Blanchard's cricket frog: vocal responses to opponent proximity and size. Anim. Behav. 38: 1025-1038.

Wagner, W. E. 1992. Deceptive or honest signaling of fighting ability? A test of alternative hypotheses for the function of changes in call dominant frequency by male cricket frogs. Anim. Behav. 44: 449-462.

Wells, K. D. 1989. Vocal communication in a neotropical treefrog, *Hyla ebraccata*: responses of males to graded aggressive calls. Copeia. 1989(2): 461-466.

Zahavi, A. & A. Zahavi. 1977. The handicap principle: A missing piece of Darwin's puzzle. Oxford U. Press. Oxford, U.K.

Figuras y Tablas

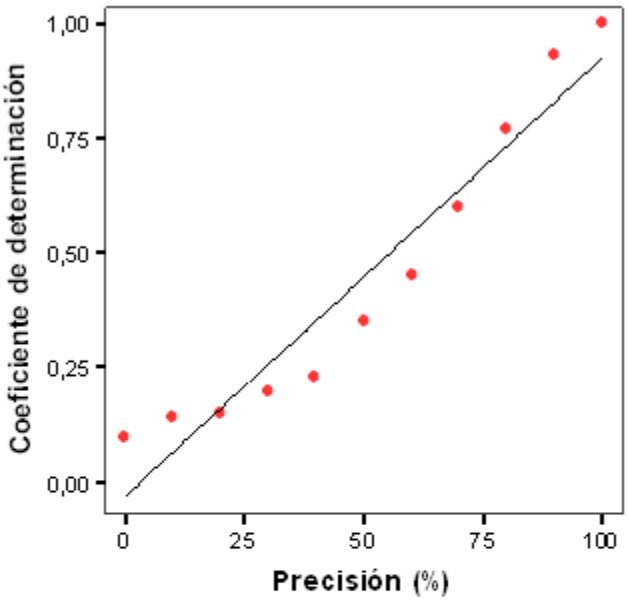
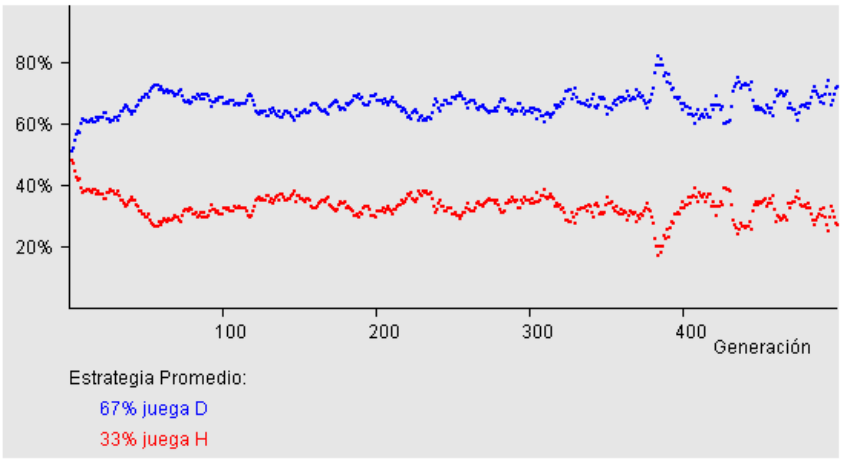


Figura 1. Relación entre el valor de precisión que utiliza el programa para calcular el tamaño señalado de los individuos de una población y el coeficiente de determinación, calculado de la regresión tamaño señalado-tamaño real en la misma población. (reg. lineal; $p < 0,01$, $n = 11$, $r^2 = 0,93$; ecuación: coef. det. = -0,03 + 0,01 prec.).

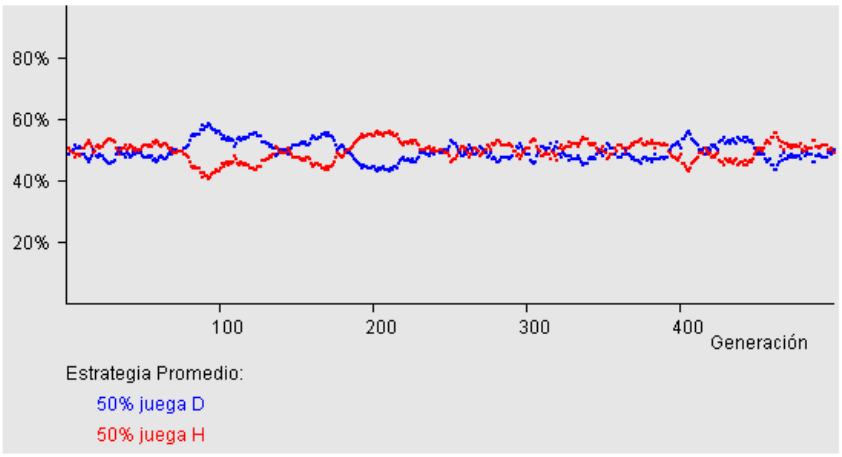
a.) V = 2 K = 3



Valor teórico:

2/3 juega D (67%)
1/3 juega H (33%)

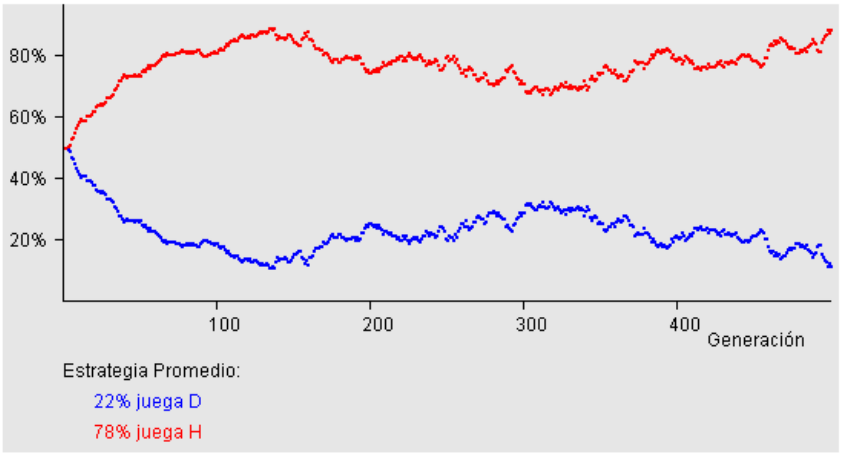
b.) V = 2 K = 2



Valor teórico:

1/2 juega D (50%)
1/2 juega H (50%)

c.) V = 3 K = 2

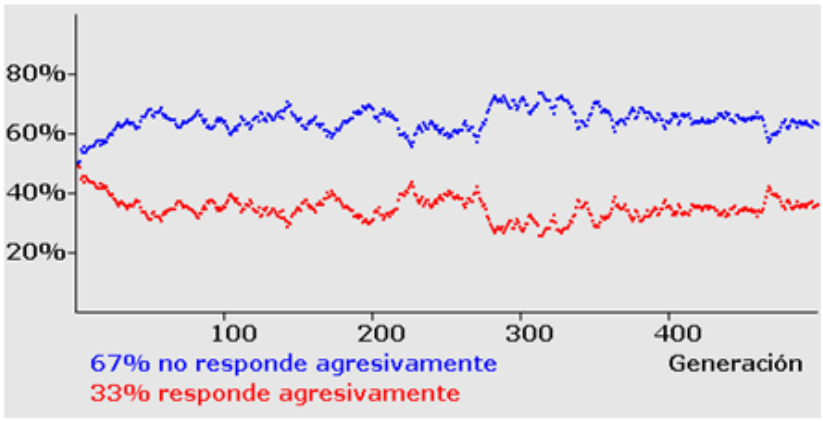


Valor teórico:

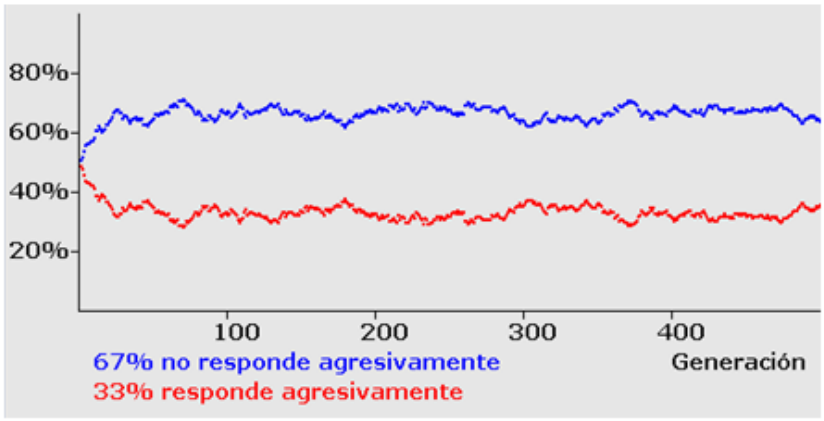
1/4 juega D (25%)
3/4 juega H (75%)

Figura 2. La simulación puesta a prueba. A la izquierda, resultados de la simulación de la evolución de la estrategia de reacción, utilizando el modelo Halcón/Paloma. A la derecha, cálculo teórico de la ESS según el modelo de Maynard Smith (Binmore & Samuelson 2001). Se muestran tres casos con diferentes valores del beneficio de ganar el enfrentamiento (V), y del costo del enfrentamiento (K). Cada punto corresponde al porcentaje promedio con el que se presenta cada comportamiento en una generación. Azul: paloma (D). Rojo: halcón (H).

a.) **n = 1000** V = 1,6 K = 2,4 **t.m = 0,05**



b.) **n = 1000** V = 1,6 K = 2,4 **t.m = 0**



c.) **n = 200** V = 1,6 K = 2,4 **t.m = 0**

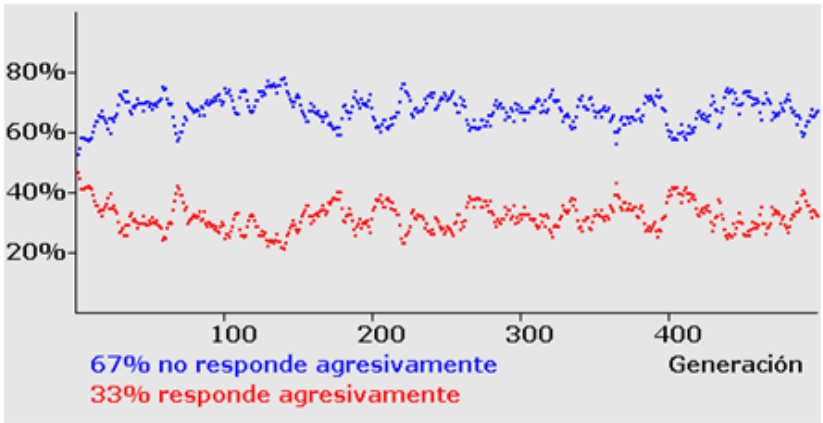


Figura 3. Resultados de diferentes simulaciones de la evolución de la estrategia de reacción, variando los parámetros: tamaño de la población (n), y tasa de mutación de la estrategia (t.m). Se mantienen constantes: el beneficio de ganar el enfrentamiento (V), y el costo del enfrentamiento (K). Cada punto corresponde al porcentaje promedio con el que se presenta cada comportamiento en una generación. En este caso se utilizó el modelo Asimetría de Tamaños Desconocida. Azul: comportamiento no agresivo. Rojo: comportamiento agresivo.

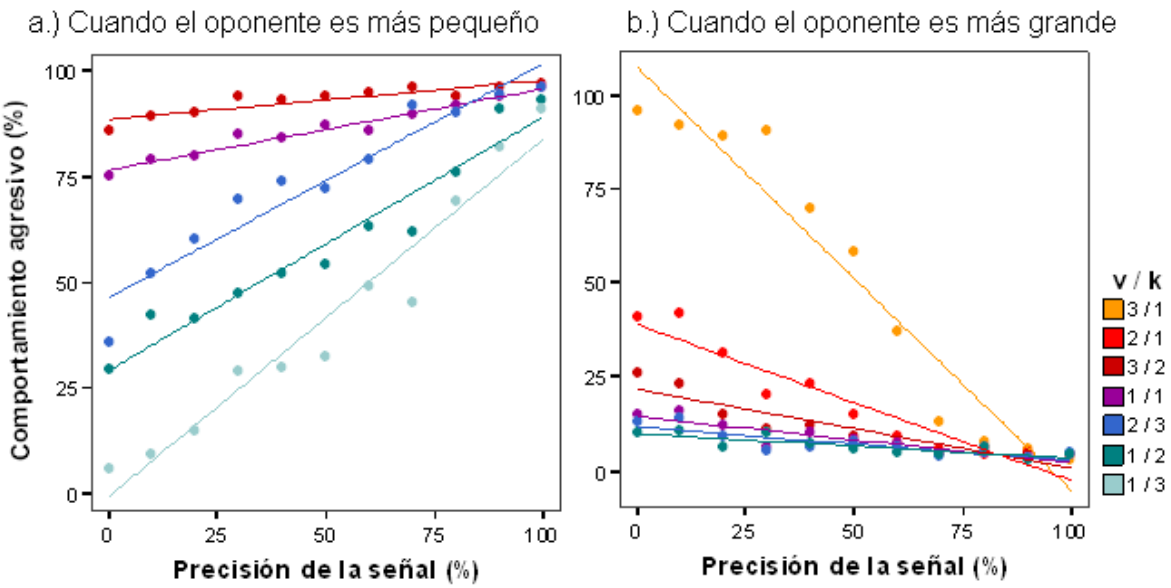


Figura 4. Efecto de la precisión de las señales, como indicadores honestos del tamaño, sobre la estrategia de reacción, utilizando diferentes valores de la relación beneficio/costo (V / K).

Tabla 1. Resumen de los análisis de regresión entre la estrategia de reacción y la precisión de las señales, como indicadores honestos del tamaño corporal, al utilizar diferentes valores de la relación beneficio/costo (v/k). Se muestran los resultados para las dos posibilidades de la estrategia de reacción: cuando el oponente es más pequeño y cuando el oponente es más grande. El coeficiente de determinación (r^2) y la pendiente (β) se incluyen si la relación es significativa ($p < 0,05$).

	oponente pequeño			*oponente grande*		
v / k	**r^2**	**β**	**p**	**r^2**	**β**	**p**
1 / 3	0,95	0,84	< 0,01			0,101
1 / 2	0,94	0,60	< 0,01	0,73	-0,06	< 0,01
2 / 3	0,93	0,55	< 0,01	0,68	-0,09	< 0,01
1 / 1	0,96	0,19	< 0,01	0,81	-0,12	< 0,01
3 / 2	0,82	0,09	< 0,01	0,85	-0,21	< 0,01
2 / 1			0,054	0,90	-0,42	< 0,01
3 / 1			0,087	0,93	-1,13	< 0,01

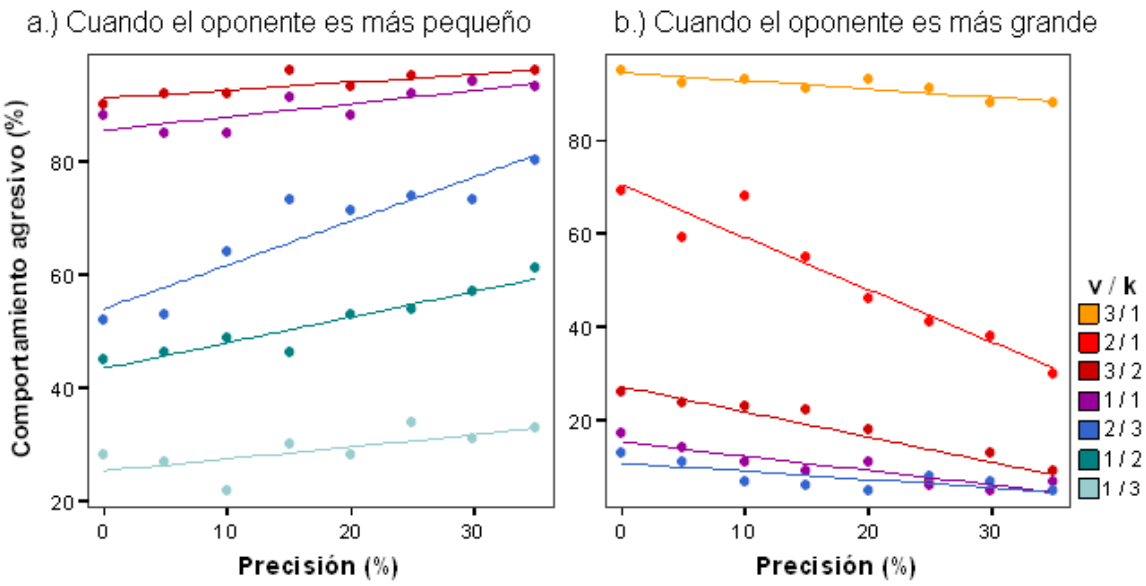


Figura 5. Efecto de la precisión de las señales, como indicadores honestos del tamaño, sobre la estrategia de reacción, utilizando diferentes valores de la relación beneficio/costo (V / K). El rango de valores de precisión, corresponde al rango de valores de los coeficientes de determinación obtenidos de la regresión frecuencia-SVL, para diferentes poblaciones de *E. femoralis* .

Tabla 2. Resumen de los análisis de regresión entre la estrategia de reacción y la precisión de las señales, como indicadores honestos del tamaño corporal, al utilizar diferentes valores de la relación beneficio/costo (v/k), cuando se toman valores de precisión dentro del rango observado para diferentes poblaciones de *E. femoralis* . Se muestran los resultados para las dos posibilidades de la estrategia de reacción: cuando el oponente es más pequeño y cuando el oponente es más grande. El coeficiente de determinación (r^2) y la pendiente (β) se incluyen si la relación es significativa ($p < 0,05$).

	oponente pequeño			*oponente grande*		
v / k	**r^2**	**β**	**p**	**r^2**	**β**	**p**
1 / 3	0,72	0,34	< 0,01			0,06
1 / 2	0,89	0,45	< 0,01			0,27
2 / 3	0,85	0,77	< 0,01	0,57	-0,18	0,03
1 / 1	0,66	0,23	0,01	0,83	-0,30	< 0,01
3 / 2	0,63	0,14	0,02	0,83	-0,54	< 0,01
2 / 1			0,47	0,91	-1,11	< 0,01
3 / 1			0,29	0,76	-0,17	< 0,01

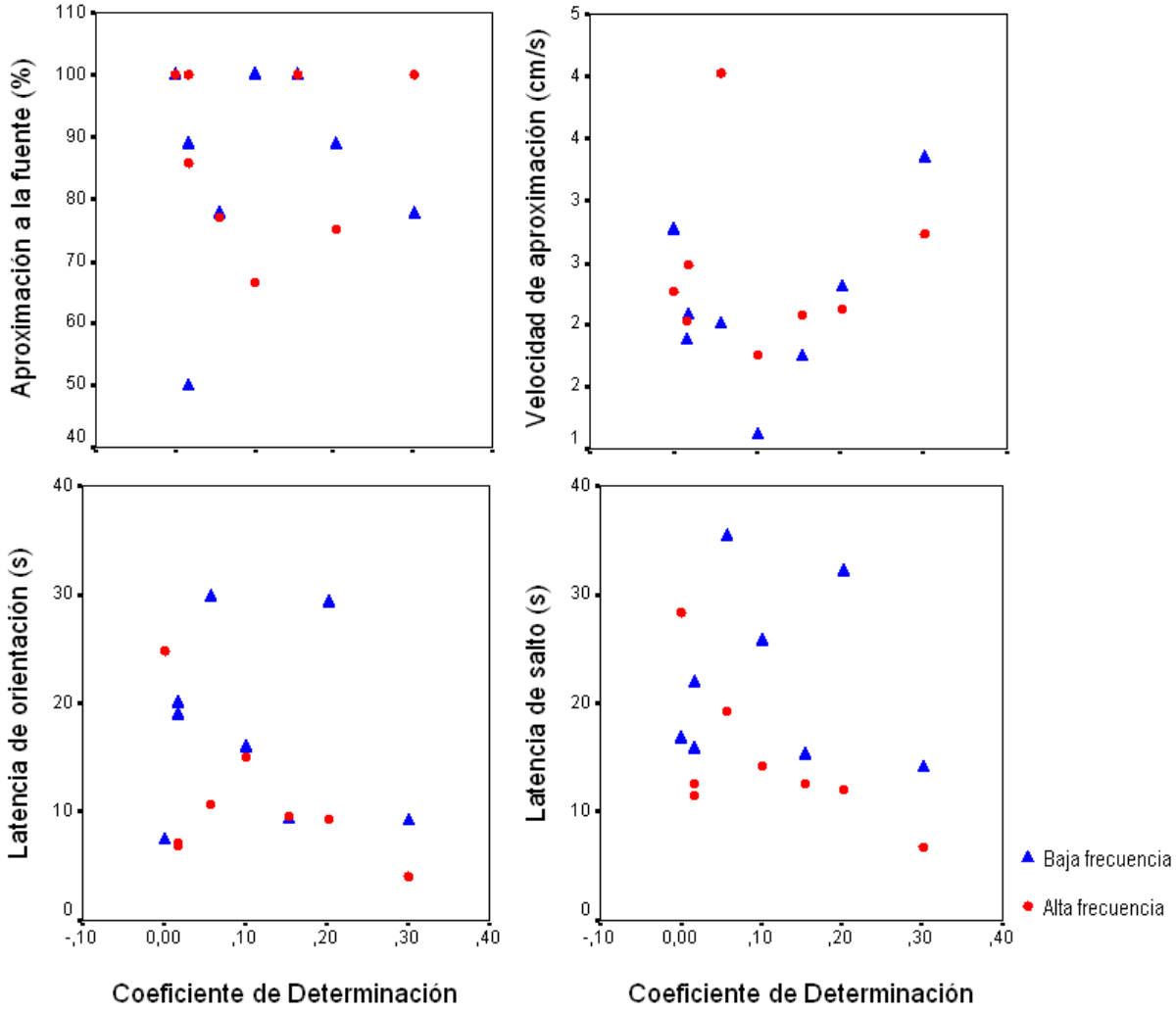


Figura 6. Relación entre la precisión observada en diferentes poblaciones de *E. femoralis* y la respuesta promedio en experimentos de playback. Las respuestas se midieron de cuatro formas: 1. Porcentaje de veces que se observó a los individuos llegar hasta la fuente del estímulo (arriba a la izquierda); 2. Velocidad con la que los individuos se aproximaron a la fuente del estímulo (arriba a la derecha); 3. Tiempo transcurrido desde el inicio del estímulo hasta que se observara una reorientación de los individuos hacia la fuente del estímulo(abajo a la izquierda); 4. Tiempo transcurrido desde el inicio del estímulo hasta que se observara una reorientación de los individuos hacia la fuente del estímulo(abajo a la derecha). Círculo rojo: respuestas observadas al utilizar estímulos con una frecuencia dominante mayor a la media de la población (representan oponentes pequeños). Triángulo azul: respuestas observadas al utilizar estímulos con una frecuencia dominante mayor a la media de la población (representan oponentes grandes).

Apéndice A

Modelos utilizados en la simulación

1. Modelos simples:

   El individuo que participa en una interacción, no tiene indicativos del tamaño de su oponente, por lo cual su estrategia siempre es la misma sin importar su tamaño relativo.

   a. Modelo Halcón/Paloma

   Reproduce las características del modelo de teoría de juegos Hawk/Dove de Maynard Smith (1982). La estrategia de reacción consiste en la probabilidad de escoger cada uno de los dos comportamientos: *halcón*, que interactúa con el oponente e intenta atacarlo hasta que éste se retire o se produzca un enfrentamiento; y *paloma*, que se retira si el oponente lo intenta atacar, o lo tolera si éste continúa la interacción pero sin atacarlo.

   Si un individuo actúa como *halcón* y el oponente actúa como *paloma*, el individuo gana el recurso, sumando el valor del recurso (V) a su *fitness*, y el oponente no sufre cambios. Si ambos actúan como *paloma*, se reparten el recurso entre sí. Por último, si ambos actúan como *halcón* ocurre un enfrentamiento, donde ambos sufren un costo (K), cuyo valor se resta a su *fitness*, pero sólo el individuo que gana el enfrentamiento obtiene el recurso (se asume que, en promedio, cada individuo gana la mitad de los enfrentamientos en que participa). La Tabla A1 resume los resultados de las posibles interacciones.

   Tabla A1. Resultados de las posibles interacciones de un individuo, según el modelo Halcón/Paloma. La matriz contiene los resultados en el fitness del individuo, según el comportamiento escogido por éste y por el oponente.

	Oponente	
Individuo	**halcón**	**paloma**
halcón	(V / 2) - K	V
paloma	0	V / 2

   V es el valor (en términos de fitness), del beneficio de ganar en enfrentamiento.
   K es el valor (en términos de fitness), del costo del enfrentamiento.

   b. Modelo Asimetría de Tamaños Desconocida

   En este modelo, la estrategia de reacción y los comportamientos son los mismos del modelo Halcón/Paloma, aunque el comportamiento *halcón* es

llamado comportamiento agresivo, y el comportamiento *paloma* es llamado comportamiento no agresivo.

La diferencia en éste modelo, es que se toman en cuenta los tamaños del individuo y del oponente que participan en una interacción, para decidir quién gana el recurso si se produce un enfrentamiento. Siempre el individuo con mayor tamaño gana el enfrentamiento. La Tabla A2 resume los resultados de las posibles interacciones.

Tabla A2. Resultados de las posibles interacciones de un individuo, según el modelo Asimetría de Tamaños Desconocida. La matriz contiene los resultados en el fitness del individuo, según el comportamiento escogido por éste y por el oponente, y dependiendo del tamaño relativo del individuo.

	Oponente Menor		Oponente Mayor	
Individuo	**agresivo**	**no agresivo**	**agresivo**	**no agresivo**
agresivo	V - K	V	- K	V
no agresivo	0	V / 2	0	V / 2

V es el valor (en términos de fitness), del beneficio de ganar en enfrentamiento.
K es el valor (en términos de fitness), del costo del enfrentamiento.

2. Modelos asimétricos:

El individuo que participa en una interacción, tiene información sobre el tamaño de su oponente, por lo cual su estrategia varía según su tamaño relativo. La estrategia de reacción para cada tamaño relativo, y los comportamientos posibles son los mismos del modelo Asimetría de Tamaños Desconocida.

c. Modelo Precisión Fija

Todos los individuos de la población tienen el mismo valor de precisión, según el cual señalan su tamaño al oponente. Cada que un individuo participa en un enfrentamiento, el tamaño que le señala a su oponente varía aleatoriamente en un porcentaje contrario a la precisión, es decir, si la precisión es 80%, su tamaño señalado puede variar hasta en un 20%. Cada individuo estima su tamaño relativo según su tamaño real, y el tamaño señalado del oponente.

Apéndice B

Obtención de los valores teóricos de las ESS, utilizando el modelo Halcón/Paloma de teoría de juegos

Siguiendo la descripción del modelo Halcón/Paloma del Apéndice A, las estrategias más simples consisten en escoger (o jugar) siempre el mismo comportamiento: *halcón* (H), o *paloma* (D). Aquellas estrategias se conocen como estrategias puras. Aparte de las estrategias puras, puede existir una estrategia mixta I, que juega H con una probabilidad P y que juega D con una probabilidad 1 – P. Para que la estrategia I sea evolutivamente estable (ESS), deben ocurrir dos cosas:

1. Ninguna de las estrategias puras es mejor para jugar contra I, es decir, siendo E (A, B) el resultado de jugar A contra B:

$$E\,(H,\,I) = E\,(D,\,I)$$

2. Jugar I contra una estrategia pura, es mejor que jugar la misma estrategia pura, es decir:

$$E\,(I,\,H) > E\,(H,\,H)$$
y $$E\,(I,\,D) > E\,(I,\,I).$$

A continuación, se calculan las ESS para los valores del beneficio de ganar el recurso (V) y del costo de un enfrentamiento (K), utilizados para probar el funcionamiento de la simulación:

	Reemplazando $E\,(H,\,I) = E\,(D,\,I)$ según la Tabla A1.	Si $E\,(I,\,H) > E\,(H,\,H)$ y $E\,(I,\,D) > E\,(I,\,I)$ es estable
• V=2 y K=3	-2P + 2(1-P) = 1-P **P = 1/3 (33%)**	-2/3 > -2 4/3 > 1
• V=2 y K=2	-P + 2(1-P) = 1-P **P = 1/2 (50%)**	-0.5 > -1 1.5 > 1
• V=3 y K=2	-0.5P + 3(1-P) = 1.5(1-P) **P = 3/4 (75%)**	-0.375 > -1 2.625 > 1

Anexo

Código del programa

```
package evolution;
public interface EvolConstants {
   public static final int UNKNOWN = 0;
   public static final int BIGGER = 1;
   public static final int SMALLER = 2;
   public static final int EQUAL = 3;
   public static final int HAWK = 0;
   public static final int DOVE = 1;
   public static final String[] MUTATIONS = { "None", "Random",
         "Normal" };
   public static final int NONE = 0;
   public static final int RANDOM = 1;
   public static final int NORMAL = 2;
   public static final int BEFORE = -1;
   public static final int DURING = 0;
   public static final int AFTER = 1;
   public static final String[] MODELS = { "Hawk/Dove",
         "Size Asymmetry (unknown)", "Precision Fixed",
         "Precision Variable", "Deceive" };
   public static final int HAWK_DOVE = 0;
   public static final int SIZE_ASYMMETRY_UNKNOWN = 1;
   public static final int PRECISION_FIXED = 2;
   public static final int PRECISION_VARIABLE = 3;
   public static final int DECEIVE = 4;
   public static final String[] VARIABLES = { "Generation",
         "Individual", "Parent", "Size", "Encounters", "Fitness",
         "Offspring", "ActStrat0", "ActStrat1", "ActStrat2",
         "ReactStrat0", "ReactStrat1", "ReactStrat2" };
   public static final String[] MEAN_VARIABLES = { "Generation",
         "MeanSize", "MeanFitness", "MeanActStrat0",
         "MeanActStrat1", "MeanActStrat2", "MeanReactStrat0",
         "MeanReactStrat1", "MeanReactStrat2" };
   public static final String[] PARAMETERS = { "n", "meanSize",
         "varSize", "generations", "model", "sizeMutation",
         "stratMutation", "k", "v", "inherit", "mutationType" };
   public static final String[] OPTIONS = { "Size?", "Fitness?",
         "Act. Strat.?", "React. Strat.?" };
}
```

```
package evolution;
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.HeadlessException;
import javax.swing.JFrame;
import evolution.controller.EvolController;
import evolution.model.EvolSimulation;
import evolution.view.EvolView;
public class Evolution extends JFrame {
   private EvolSimulation model;
   private EvolView view;
   private EvolController controller;
   public Evolution() throws HeadlessException {
      super("Simulación de la Evolución de la Agresión " +
                        "Ritualizada");
      model = new EvolSimulation();
      view = new EvolView();
      controller = new EvolController(model);
      model.setEvolEventListener(view);
      getContentPane().add(controller, BorderLayout.NORTH);
      getContentPane().add(view, BorderLayout.CENTER);
      setPreferredSize(new Dimension(1000, 730));
   }
   public static void main(String[] args) {
      Evolution evolution = new Evolution();
      evolution.setDefaultCloseOperation(EXIT_ON_CLOSE);
      evolution.pack();
      evolution.setVisible(true);
   }
}
```

```
package evolution.controller;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.io.File;
import javax.swing.ButtonGroup;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JComboBox;
import javax.swing.JFileChooser;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
import javax.swing.JTextField;
import evolution.EvolConstants;
import evolution.model.EvolSimulation;
import evolution.model.ModelDeceive;
import evolution.model.ModelHawkDove;
import evolution.model.ModelPrecisionFixed;
import evolution.model.ModelPrecisionVariable;
import evolution.model.ModelSizeAsymmetryUnknown;
public class EvolController extends JPanel implements
      ActionListener, ItemListener, EvolConstants {
   private JLabel nLabel, genLabel, vLabel, kLabel, sizeMutLabel,
         stratMutLabel;
   private JTextField nTextF, genTextF, vTextF, kTextF,
         sizeMutTextF, stratMutTextF;
   private JComboBox modelList;
   private JRadioButton randSize, inherSize;
   private ButtonGroup sizePassOpt;
   private JRadioButton noneMut, randMut, normMut;
   private ButtonGroup sizeMutOpt;
   private JCheckBox fileOpt;
   private JButton paintOpt;
   private JButton evol;
   private EvolSimulation evolSimulation;
   public EvolController(EvolSimulation simulation) {
      evolSimulation = simulation;
      nLabel = new JLabel("n =");
      nTextF = new JTextField(4);
      nTextF.setText(Integer.toString(evolSimulation.n));
      genLabel = new JLabel("Generations =");
      genTextF = new JTextField(4);
      genTextF
            .setText(Integer.toString(evolSimulation.generations));
      vLabel = new JLabel("Resource value =");
      vTextF = new JTextField(4);
      vTextF.setText(Float.toString(evolSimulation.v));
      kLabel = new JLabel("Fighting cost =");
      kTextF = new JTextField(4);
      kTextF.setText(Float.toString(evolSimulation.k));
      sizeMutLabel = new JLabel("sizeMutation =");
      sizeMutTextF = new JTextField(4);
      sizeMutTextF.setText(Float.toString(evolSimulation.sizeMut));
      stratMutLabel = new JLabel("stratMutation =");
      stratMutTextF = new JTextField(4);
      stratMutTextF
            .setText(Float.toString(evolSimulation.stratMut));
      modelList = new JComboBox(MODELS);
      modelList.addItemListener(this);
      randSize = new JRadioButton("Don´t inherit size", true);
      randSize.setBackground(new Color(100, 100, 250));
      randSize.setForeground(new Color(220, 220, 100));
      randSize.addItemListener(this);
      inherSize = new JRadioButton("Inherit size");
      inherSize.setBackground(new Color(150, 150, 250));
      inherSize.setForeground(new Color(220, 220, 150));
      inherSize.addItemListener(this);
      sizePassOpt = new ButtonGroup();
      sizePassOpt.add(randSize);
      sizePassOpt.add(inherSize);
      noneMut = new JRadioButton("None mutation");
      noneMut.setForeground(new Color(150, 200, 150));
      noneMut.addItemListener(this);
      noneMut.setEnabled(false);
      randMut = new JRadioButton("Random mutation");
      randMut.setForeground(new Color(150, 200, 150));
      randMut.addItemListener(this);
      randMut.setEnabled(false);
      normMut = new JRadioButton("Normal mutation", true);
      normMut.setForeground(new Color(150, 200, 150));
      normMut.addItemListener(this);
      normMut.setEnabled(false);
      sizeMutOpt = new ButtonGroup();
      sizeMutOpt.add(noneMut);
      sizeMutOpt.add(randMut);
      sizeMutOpt.add(normMut);
      fileOpt = new JCheckBox("Save file");
      fileOpt.addItemListener(this);
      paintOpt = new JButton("View");
      paintOpt.addActionListener(this);
      evol = new JButton("Evolution");
      evol.addActionListener(this);
      add(nLabel);
      add(nTextF);
      add(genLabel);
      add(genTextF);
      add(vLabel);
      add(vTextF);
      add(kLabel);
      add(kTextF);
      add(sizeMutLabel);
      add(sizeMutTextF);
      add(stratMutLabel);
      add(stratMutTextF);
      add(modelList);
      add(fileOpt);
      add(paintOpt);
      add(randSize);
      add(inherSize);
      add(noneMut);
      add(randMut);
      add(normMut);
      add(evol);
      setPreferredSize(new Dimension(1000, 70));
   }
   public void actionPerformed(ActionEvent e) {
      boolean ready = true;
      int newN = Integer.parseInt(nTextF.getText());
      int newGen = Integer.parseInt(genTextF.getText());
      float newV = Float.parseFloat(vTextF.getText());
      float newK = Float.parseFloat(kTextF.getText());
      float newSizeMut = Float.parseFloat(sizeMutTextF.getText());
      float newStratMut = Float.parseFloat(stratMutTextF.
            getText());
      int model = modelList.getSelectedIndex();
      if (e.getSource() == paintOpt) {
         boolean options[] = new boolean[OPTIONS.length];
```

```
            for (int i = 0; i < OPTIONS.length; i++) {
               int result = JOptionPane.showConfirmDialog(this,
                     OPTIONS[i]);
               if (result == JOptionPane.YES_OPTION)
                  options[i] = true;
               else
                  options[i] = false;
            }
            evolSimulation.setPaintOptions(options[0], options[1],
                  options[2], options[3]);
            return;
         }
         if ((newN % 2 != 0) || (newN <= 0)) {
            JOptionPane.showMessageDialog(this,
                  "n must be pair and positive (>0)!");
            nTextF.setText(Integer.toString(evolSimulation.n));
            ready = false;
         }
         else
            evolSimulation.n = newN;
         if (newGen < 20) {
            JOptionPane.showMessageDialog(this,
                  "At least 20 generations!");
            genTextF.setText(Integer
                  .toString(evolSimulation.generations));
            ready = false;
         }
         else
            evolSimulation.generations = newGen;
         if (newK < 0) {
            JOptionPane.showMessageDialog(this,
                  "Fighting cost must be positive!");
            kTextF.setText(Float.toString(evolSimulation.k));
            ready = false;
         }
         else {
            evolSimulation.k = newK;
            evolSimulation.model.setPayoffs(evolSimulation.k,
                  evolSimulation.v);
         }
         if (newV < 0) {
            JOptionPane.showMessageDialog(this,
                  "Resource value must be positive!");
            vTextF.setText(Float.toString(evolSimulation.v));
            ready = false;
         }
         else {
            evolSimulation.v = newV;
            evolSimulation.model.setPayoffs(evolSimulation.k,
                  evolSimulation.v);
         }
         if (newSizeMut < 0 || newSizeMut > 1) {
            JOptionPane.showMessageDialog(this,
                  "0 <= SizeMutation <= 1 !");
            sizeMutTextF.setText(Float
                  .toString(evolSimulation.sizeMut));
            ready = false;
         }
         else
            evolSimulation.sizeMut = newSizeMut;
         if (newStratMut < 0 || newStratMut > 1) {
            JOptionPane.showMessageDialog(this,
                  "0 <= StratMutation <= 1 !");
            stratMutTextF.setText(Float
                  .toString(evolSimulation.stratMut));
            ready = false;
         }
         else
            evolSimulation.stratMut = newStratMut;
         switch (model) {
            case HAWK_DOVE:
               evolSimulation.model = new ModelHawkDove();
               evolSimulation.model.setPayoffs(evolSimulation.k,
                     evolSimulation.v);
               break;
            case SIZE_ASYMMETRY_UNKNOWN:
               evolSimulation.model = new ModelSizeAsymmetryUnknown();
               evolSimulation.model.setPayoffs(evolSimulation.k,
                     evolSimulation.v);
               break;
            case PRECISION_FIXED:
               evolSimulation.model = new ModelPrecisionFixed();
               evolSimulation.model.setPayoffs(evolSimulation.k,
                     evolSimulation.v);
               break;
            case PRECISION_VARIABLE:
               evolSimulation.model = new ModelPrecisionVariable();
               evolSimulation.model.setPayoffs(evolSimulation.k,
                     evolSimulation.v);
               break;
            case DECEIVE:
               evolSimulation.model = new ModelDeceive();
               evolSimulation.model.setPayoffs(evolSimulation.k,
                     evolSimulation.v);
               break;
         }
         if ((e.getSource() == evol) && ready) {
            fileOpt.setEnabled(false);
            evolSimulation.evolve();
            evolSimulation.file = null;
            fileOpt.setSelected(false);
            fileOpt.setEnabled(true);
         }
      }
      public void itemStateChanged(ItemEvent e) {
         if (e.getSource() == randSize || e.getSource() ==
           inherSize) {
            if (randSize.isSelected()) {
               evolSimulation.inherit = false;
               noneMut.setEnabled(false);
               randMut.setEnabled(false);
               normMut.setEnabled(false);
               randSize.setBackground(new Color(100, 100, 250));
               randSize.setForeground(new Color(220, 220, 100));
               inherSize.setBackground(new Color(150, 150, 250));
               inherSize.setForeground(new Color(220, 220, 150));
            }
            else {
               evolSimulation.inherit = true;
               noneMut.setEnabled(true);
               randMut.setEnabled(true);
               normMut.setEnabled(true);
               inherSize.setBackground(new Color(100, 100, 250));
               inherSize.setForeground(new Color(220, 220, 100));
               randSize.setBackground(new Color(150, 150, 250));
               randSize.setForeground(new Color(220, 220, 150));
            }
         }
         if (e.getSource() == noneMut || e.getSource() == randMut
               || e.getSource() == normMut) {
            if (noneMut.isSelected()) {
               evolSimulation.mutationType = NONE;
            }
            else if (randMut.isSelected()) {
               evolSimulation.mutationType = RANDOM;
            }
            else {
               evolSimulation.mutationType = NORMAL;
            }
         }
         if (e.getSource() == modelList) {
            if (e.getStateChange() == ItemEvent.SELECTED) {
               if (modelList.getSelectedIndex() == PRECISION_FIXED) {
                  String result;
                  result = JOptionPane
                        .showInputDialog(this, "Precision:", Float
                              .toString(ModelPrecisionFixed.
                                    precision));
                  if (result != null) {
                     float newPrecision = Float.parseFloat(result);
                     if (0 <= newPrecision && newPrecision <= 1)
                        ModelPrecisionFixed.precision = newPrecision;
                  }
               }
            }
         }
         if (e.getSource() == fileOpt) {
            if (fileOpt.isSelected()) {
               JFileChooser chooser = new JFileChooser();
               chooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
               int result = chooser.showSaveDialog(this);
               if (result == JFileChooser.CANCEL_OPTION) {
                  fileOpt.setSelected(false);
                  return;
               }
               evolSimulation.file = chooser.getSelectedFile();
               evolSimulation.fileParameters = new File(
                     evolSimulation.file.getPath() + "Parameters");
               evolSimulation.fileSummary = new File(
                     evolSimulation.file.getPath() + "Summary");
            }
            else
               evolSimulation.file = null;
         }
      }
   }
```

```
package evolution.event;
import java.io.File;
import evolution.model.Model;
import evolution.model.Population;
public class EvolEvent {
   public int state;
   public int generation;
   public int generations;
   public int encounters;
   public boolean saveData;
   public float k, v;
   public Model model;
   public float sValue;
   public float fValue;
   public float pValue1;
   public float hValue1, hValue2;
   public float dValue1, dValue2;
   public boolean inherit;
   public int mutationType;
   public boolean drawSize = false;
   public boolean drawFitness = false;
   public boolean drawActStrat = false;
   public boolean drawReactStrat = true;
   public File fileName, fileParametersName, fileSummaryName;
   public Population pop;
}
```

```
package evolution.event;
public interface EvolEventListener {
   public void evolStepDone(EvolEvent evolEvent);
   public void newPaintOptions(EvolEvent evolEvent);
   public void savePopulation(EvolEvent evolEvent);
}
```

```
package evolution.model;
import java.io.File;
import evolution.EvolConstants;
import evolution.event.EvolEvent;
import evolution.event.EvolEventListener;
public class EvolSimulation implements EvolConstants {
   public Population p;
   public float sizeMut = 0.05f;
   public float stratMut = 0.01f;
   public int generations = 500;
   public int encounters = 4;
   public int sample = 5;
   public int n = 1000;
   public float mSize = 28.7f, dSize = 1.3f;
   public boolean ready = false;
   public Model model;
   public float k = 3, v = 2;
```

```
public boolean inherit = false;
public int mutationType = NORMAL;
public File file, fileParameters, fileSummary;
private EvolEvent event;
private EvolEventListener listener;
public EvolSimulation() {
   event = new EvolEvent();
   model = new ModelHawkDove();
   model.setPayoffs(k, v);
}
public void evolve() {
   p = new Population(model.modelID, n, sizeMut, stratMut,
         mSize, dSize);
   event.model = model;
   event.generations = generations;
   event.encounters = encounters;
   event.k = k;
   event.v = v;
   event.inherit = inherit;
   event.mutationType = mutationType;
   event.fileName = file;
   event.fileParametersName = fileParameters;
   event.fileSummaryName = fileSummary;
   float meanSize = 0;
   float sumSize = 0;
   float meanFitness = 0;
   float sumFitness = 0;
   float meanActStrat1 = 0;
   float sumActStrat1 = 0;
   float meanReactStrat1 = 0, meanReactStrat2 = 0;
   float sumReactStrat1 = 0, sumReactStrat2 = 0;
   int sumCont = 0;
   event.state = BEFORE;
   listener.evolStepDone(event);
   listener.savePopulation(event);
   event.state = DURING;
   for (int i = 0; i < generations; i++) {
      p.confront(model.payoffs, encounters);
      event.pop = p;
      event.generation = i + 1;
      if (event.generation == 1
            || event.generation % (event.generations / 10) ==
               0) {
         event.saveData = true;
         listener.savePopulation(event);
      }
      else {
         event.saveData = false;
         listener.savePopulation(event);
      }
      meanSize = p.genMeanSize;
      meanFitness = p.findMeanFitness();
      if (modelIncludesSignal()) {
         meanActStrat1 = p.findMeanActorStrategy(UNKNOWN);
      }
      if (model.modelID == HAWK_DOVE
            || model.modelID == SIZE_ASYMMETRY_UNKNOWN)
         meanReactStrat1 = p.findMeanReactorStrategy(UNKNOWN);
      else {
         meanReactStrat1 = p.findMeanReactorStrategy(BIGGER);
         meanReactStrat2 = p.findMeanReactorStrategy(SMALLER);
      }
      if ((i + 1) > (generations - (generations / sample))) {
         sumSize += meanSize;
         sumFitness += meanFitness;
         if (modelIncludesSignal()) {
            sumActStrat1 += meanActStrat1;
         }
         sumReactStrat1 += meanReactStrat1;
         if (modelIncludesSignal())
            sumReactStrat2 += meanReactStrat2;
         sumCont++;
      }
      event.sValue = meanSize;
      event.fValue = meanFitness;
      if (modelIncludesSignal()) {
         event.pValue1 = meanActStrat1;
      }
      event.dValue1 = meanReactStrat1;
      event.hValue1 = 1 - meanReactStrat1;
      if (modelIncludesSignal()) {
         event.dValue2 = meanReactStrat2;
         event.hValue2 = 1 - meanReactStrat2;
      }
      listener.evolStepDone(event);
      p.changeGeneration(inherit, mutationType);
   }
   event.state = AFTER;
   meanSize = sumSize / sumCont;
   event.sValue = meanSize;
   meanFitness = sumFitness / sumCont;
   event.fValue = meanFitness;
   if (modelIncludesSignal()) {
      meanActStrat1 = sumActStrat1 / sumCont;
      event.pValue1 = meanActStrat1;
   }
   meanReactStrat1 = sumReactStrat1 / sumCont;
   event.dValue1 = meanReactStrat1;
   event.hValue1 = 1 - meanReactStrat1;
   if (modelIncludesSignal()) {
      meanReactStrat2 = sumReactStrat2 / sumCont;
      event.dValue2 = meanReactStrat2;
      event.hValue2 = 1 - meanReactStrat2;
   }
   listener.evolStepDone(event);
   listener.savePopulation(event);
}
public void setPaintOptions(boolean s, boolean f, boolean aS,
      boolean rS) {
   event.drawSize = s;
   event.drawFitness = f;
   event.drawActStrat = aS;
   event.drawReactStrat = rS;
   listener.newPaintOptions(event);
   }
   private boolean modelIncludesSignal() {
      if (model.modelID == PRECISION_FIXED
            || model.modelID == PRECISION_VARIABLE
            || model.modelID == DECEIVE)
         return true;
      else
         return false;
   }
   public void setEvolEventListener(EvolEventListener eL) {
      listener = eL;
   }
}
```

```
package evolution.model;
public class Individual {
   private boolean alive;
   private int encounters;
   private int offspring;
   private float size, fitness;
   private Strategy strat = new Strategy();
   public int parent = 0;
   public Individual(float s, float[] aS, float[] rS) {
      setSize(s);
      setFitness(0);
      setOffspring(0);
      setActorStrategy(aS);
      setReactorStrategy(rS);
      setEncounters(0);
      setAlive(true);
   }
   public float getSize() {
      return size;
   }
   public float getFitness() {
      return fitness;
   }
   public float[] getActorStrategy() {
      return strat.getActor();
   }
   public float getActorStrategy(int relSize) {
      return strat.getActor(relSize);
   }
   public float[] getReactorStrategy() {
      return strat.getReactor();
   }
   public float getReactorStrategy(int relSize) {
      return strat.getReactor(relSize);
   }
   public int getOffspring() {
      return offspring;
   }
   public int getEncounters() {
      return encounters;
   }
   public boolean isAlive() {
      return alive;
   }
   public void setSize(float s) {
      size = (s > 0 ? s : 0);
   }
   public void setFitness(float f) {
      fitness = f;
   }
   public void setOffspring(float f) {
      offspring = (f > 0 ? Math.round(f) : 0);
   }
   public void setActorStrategy(float[] aS) {
      strat.setActor(aS);
   }
   public void setActorStrategy(float aS, int relSize) {
      if (aS >= 0)
         strat.setActor(aS, relSize);
      else
         strat.setActor(0, relSize);
   }
   public void setReactorStrategy(float[] rS) {
      strat.setReactor(rS);
   }
   public void setReactorStrategy(float rS, int relSize) {
      if (rS >= 0)
         strat.setReactor(rS, relSize);
      else
         strat.setReactor(0, relSize);
   }
   public void setEncounters(int e) {
      encounters = (e >= 0 ? e : 0);
   }
   public void setAlive(boolean a) {
      alive = a;
   }
   public void copyIndividual(Individual ind) {
      setSize(ind.getSize());
      setFitness(ind.getFitness());
      setOffspring(ind.getOffspring());
      setActorStrategy(ind.getActorStrategy());
      setReactorStrategy(ind.getReactorStrategy());
      setEncounters(ind.getEncounters());
      setAlive(ind.isAlive());
      parent = ind.parent;
   }
}
```

```
package evolution.model;
import java.awt.Dimension;
public abstract class Model {
   public int modelID;
   public float[][] payoffs;
   public abstract void setPayoffs(float k, float v);
   public abstract Dimension getDimension();
   public abstract String getComment(int i);
}
```

```
package evolution.model;
```

```
import java.awt.Dimension;
import evolution.EvolConstants;
public class ModelDeceive extends Model implements EvolConstants {
   public String[] comments = { "% no responde agresivamente",
         "% responde agresivamente",
         "Estrategia Promedio al ser mayor:",
         "Estrategia Promedio al ser menor:" };
   public ModelDeceive() {
      modelID = DECEIVE;
   }
   public void setPayoffs(float k, float v) {
      float matrix[][] = { { v - k, v, -k, v },
            { 0, v / 2, 0, v / 2 } };
      payoffs = matrix;
   }
   public Dimension getDimension() {
      return new Dimension(570, 640);
   }
   public String getComment(int i) {
      return comments[i];
   }
}

package evolution.model;
import java.awt.Dimension;
import evolution.EvolConstants;
public class ModelHawkDove extends Model implements EvolConstants {
   public String[] comments = { "% juega Dove", "% juega Hawk",
         "Estrategia Promedio:" };
   public ModelHawkDove() {
      modelID = HAWK_DOVE;
   }
   public void setPayoffs(float k, float v) {
      float matrix1[][] = { { v / 2 - k, v }, { 0, v / 2 } };
      payoffs = matrix1;
   }
   public Dimension getDimension() {
      return new Dimension(570, 320);
   }
   public String getComment(int i) {
      return comments[i];
   }
}

package evolution.model;
import java.awt.Dimension;
import evolution.EvolConstants;
public class ModelPrecisionFixed extends Model implements
      EvolConstants {
   public String[] comments = { "% no responde agresivamente",
         "% responde agresivamente",
         "Estrategia Promedio al ser mayor:",
         "Estrategia Promedio al ser menor:" };
   public static float precision = 0.8f;
   public ModelPrecisionFixed() {
      modelID = PRECISION_FIXED;
   }
   public void setPayoffs(float k, float v) {
      float matrix[][] = { { v - k, v, -k, v },
            { 0, v / 2, 0, v / 2 } };
      payoffs = matrix;
   }
   public Dimension getDimension() {
      return new Dimension(570, 640);
   }
   public String getComment(int i) {
      return comments[i];
   }
}

package evolution.model;
import java.awt.Dimension;
import evolution.EvolConstants;
public class ModelPrecisionVariable extends Model implements
      EvolConstants {
   public String[] comments = { "% no responde agresivamente",
         "% responde agresivamente",
         "Estrategia Promedio al ser mayor:",
         "Estrategia Promedio al ser menor:" };
   public ModelPrecisionVariable() {
      modelID = PRECISION_VARIABLE;
   }
   public void setPayoffs(float k, float v) {
      float matrix[][] = { { v - k, v, -k, v },
            { 0, v / 2, 0, v / 2 } };
      payoffs = matrix;
   }
   public Dimension getDimension() {
      return new Dimension(570, 640);
   }
   public String getComment(int i) {
      return comments[i];
   }
}

package evolution.model;
import java.awt.Dimension;
import evolution.EvolConstants;
public class ModelSizeAsymmetryUnknown extends Model implements
      EvolConstants {
   public String[] comments = { "% no responde agresivamente",
         "% responde agresivamente", "Estrategia Promedio:" };
   public ModelSizeAsymmetryUnknown() {
      modelID = SIZE_ASYMMETRY_UNKNOWN;
   }
   public void setPayoffs(float k, float v) {
      float matrix[][] = { { v - k, v, -k, v },
            { 0, v / 2, 0, v / 2 } };
      payoffs = matrix;
   }
   public Dimension getDimension() {
      return new Dimension(570, 320);
   }
   public String getComment(int i) {
      return comments[i];
   }
}

package evolution.model;
import evolution.EvolConstants;
public class Population implements EvolConstants {
   public int modelID;
   public int n;
   public float stratMutation;
   public float sizeMutation;
   public float meanSize, destSize, varSize;
   public Individual generation[];
   private Individual nextGeneration[];
   public float genMeanSize, genVarSize;
   public Population(int mID, int pop, float sizeMut,
         float stratMut, float mS, float dS) {
      modelID = ((mID >= 0) && (mID <= 10) ? mID : 0);
      n = ((pop > 0) && (pop % 2 == 0) ? pop : 0);
      sizeMutation = ((sizeMut >= 0) && (sizeMut <= 1) ? sizeMut
            : 0);
      stratMutation = ((stratMut >= 0) && (stratMut <= 1) ?
            stratMut : 0);
      meanSize = (mS > 0 ? mS : 0);
      destSize = ((dS > 0) && (dS < mS) ? dS : 0);
      varSize = (float) Math.pow(dS, 2);
      generation = new Individual[n];
      create();
   }
   private void create() {
      for (int i = 0; i < n; i++) {
         generation[i] = new Individual(randomSize(),
               randomActorStrategy(), randomReactorStrategy());
      }
      findGenSizeDist();
   }
   private float randomSize() {
      double yMax = 1 / (Math.sqrt(varSize * 2 * Math.PI));
      float size = meanSize;
      boolean end = false;
      while (!end) {
         double randX = Math.random() * (meanSize * 2);
         double y = findY(randX, meanSize, varSize);
         if (Math.random() <= (y / yMax)) {
            size = (float) randX;
            end = true;
         }
      }
      return size;
   }
   private float[] randomActorStrategy() {
      float actStrat[] = new float[4];
      if (modelID == PRECISION_FIXED) {
         for (int i = 0; i < 4; i++) {
            actStrat[i] = ModelPrecisionFixed.precision;
         }
      }
      else if (modelID == DECEIVE) {
         for (int i = 0; i < 4; i++) {
            actStrat[i] = 0.5f;
         }
      }
      else {
         for (int i = 0; i < 4; i++) {
            actStrat[i] = (float) Math.random();
         }
      }
      return actStrat;
   }
   private float[] randomReactorStrategy() {
      float reactStrat[] = new float[4];
      for (int i = 0; i < 4; i++) {
         reactStrat[i] = (float) Math.random();
      }
      return reactStrat;
   }
   public void confront(float payoffs[][], int encounters) {
      for (int fightsCount = 1; fightsCount <= encounters;
        fightsCount++) {
         int i = 0, j;
         int attitudeA, attitudeB;
         int actRelSizeA, actRelSizeB;
         int relSizeForA, relSizeForB;
         float sizeA, sizeB;
         float signSizeA, signSizeB;
         float precisionA, precisionB;
         while (i < n) {
            if (generation[i].getEncounters() < fightsCount) {
               j = i + 1 + (int) (Math.random() * (n - (i + 1)));
               if (generation[j].getEncounters() < fightsCount) {
                  sizeA = generation[i].getSize();
                  sizeB = generation[j].getSize();
                  actRelSizeA = findActRelSize(sizeA, sizeB);
                  actRelSizeB = findActRelSize(sizeB, sizeA);
                  if (modelID == DECEIVE) {
                     signSizeA = sizeA
                           * 2
                           * generation[i]
                                 .getActorStrategy(UNKNOWN);
                     signSizeB = sizeB
                           * 2
                           * generation[j]
                                 .getActorStrategy(UNKNOWN);
                  }
                  else {
                     if (modelID == PRECISION_FIXED) {
                        precisionA = ModelPrecisionFixed.precision;
                        precisionB = ModelPrecisionFixed.precision;
                     }
                     else {
                        precisionA = generation[i]
                              .getActorStrategy(UNKNOWN);
                        precisionB = generation[j]
                              .getActorStrategy(UNKNOWN);
```

```
                }
                if (Math.random() < 0.5)
                   signSizeA = sizeA - ((float) Math.random()
                     * (destSize * 5) * (1 - precisionA));
                else
                   signSizeA = sizeA + ((float) Math.random()
                     * (destSize * 5) * (1 - precisionA));
                if (Math.random() < 0.5)
                   signSizeB = sizeB - ((float) Math.random()
                     * (destSize * 5) * (1 - precisionB));
                else
                   signSizeB = sizeB + ((float) Math.random()
                     * (destSize * 5) * (1 - precisionB));
             }
             relSizeForA = findRelSize(sizeA, signSizeB);
             relSizeForB = findRelSize(sizeB, signSizeA);
             attitudeA = findAttitude(generation[i]
                   .getReactorStrategy(relSizeForA));
             attitudeB = findAttitude(generation[j]
                   .getReactorStrategy(relSizeForB));
             generation[i].setFitness(generation[i]
                   .getFitness()
                   + outcome(attitudeA, attitudeB,
                         actRelSizeA, payoffs));
             generation[j].setFitness(generation[j]
                   .getFitness()
                   + outcome(attitudeB, attitudeA,
                         actRelSizeB, payoffs));
             generation[i].setEncounters(fightsCount);
             generation[j].setEncounters(fightsCount);
             i++;
          }
       }
       else
          i++;
    }
  }
}
private int findActRelSize(float sizeA, float sizeB) {
   int result;
   switch (modelID) {
      case HAWK_DOVE:
         result = UNKNOWN;
         break;
      case SIZE_ASYMMETRY_UNKNOWN:
      case PRECISION_FIXED:
      case PRECISION_VARIABLE:
      case DECEIVE:
         if (sizeA >= sizeB)
            result = BIGGER;
         else
            result = SMALLER;
         break;
      default:
         result = UNKNOWN;
         break;
   }
   return result;
}
private int findRelSize(float sizeA, float signSizeB) {
   int result;
   switch (modelID) {
      case HAWK_DOVE:
      case SIZE_ASYMMETRY_UNKNOWN:
         result = UNKNOWN;
         break;
      case PRECISION_FIXED:
      case PRECISION_VARIABLE:
      case DECEIVE:
         if (sizeA >= signSizeB)
            result = BIGGER;
         else
            result = SMALLER;
         break;
      default:
         result = UNKNOWN;
         break;
   }
   return result;
}
public int findAttitude(float d) {
   float motivation = (float) Math.random();
   if (motivation > d)
      return HAWK;
   else
      return DOVE;
}
public float outcome(int a, int b, int relSize,
      float payoffs[][]) {
   float result;
   switch (modelID) {
      case HAWK_DOVE:
         result = payoffs[a][b];
         break;
      case SIZE_ASYMMETRY_UNKNOWN:
      case PRECISION_FIXED:
      case PRECISION_VARIABLE:
      case DECEIVE:
         if (relSize == BIGGER)
            result = payoffs[a][b];
         else
            result = payoffs[a][b + 2];
         break;
      default:
         result = 0;
         break;
   }
   return result;
}
public void changeGeneration(boolean inherit, int
  mutationType) {
   reproduce(inherit);
   replace();
   findGenSizeDist();
   mutate(inherit, mutationType);
   findGenSizeDist();
}
public void reproduce(boolean inherit) {
   float minFitness = findMinFitness();
   for (int i = 0; i < n; i++) {
      generation[i].setOffspring(generation[i].getFitness()
            - minFitness + 1);
   }
   int totalOffspring = findTotalOffspring();
   int cont = 0;
   nextGeneration = new Individual[totalOffspring];
   for (int i = 0; i < n; i++) {
      for (int j = 0; j < generation[i].getOffspring(); j++) {
         float newSize;
         if (inherit)
            newSize = generation[i].getSize();
         else
            newSize = randomSize();
         nextGeneration[cont] = new Individual(newSize,
               generation[i].getActorStrategy(), generation[i]
                     .getReactorStrategy());
         nextGeneration[cont].parent = i + 1;
         cont++;
      }
   }
}
public int findTotalOffspring() {
   int tOffspring = 0;
   for (int i = 0; i < n; i++) {
      tOffspring += generation[i].getOffspring();
   }
   return tOffspring;
}
public void replace() {
   killExcess();
   int j = 0;
   for (int i = 0; i < nextGeneration.length; i++) {
      if (nextGeneration[i].isAlive()) {
         generation[j].copyIndividual(nextGeneration[i]);
         j++;
      }
   }
}
public void killExcess() {
   int i, excess = findTotalOffspring() - n;
   while (excess > 0) {
      i = (int) (Math.random() * nextGeneration.length);
      if (nextGeneration[i].isAlive()) {
         nextGeneration[i].setAlive(false);
         excess--;
      }
   }
}
public void mutate(boolean inherit, int mutationType) {
   float change = 0;
   float newReactStrat[] = new float[4];
   float newActStrat[] = new float[4];
   float mean = findMeanSize();
   float var = findVarSize(mean);
   if (var == 0)
      var = varSize / 100;
   for (int i = 0; i < generation.length; i++) {
      newReactStrat = generation[i].getReactorStrategy();
      newActStrat = generation[i].getActorStrategy();
      if (inherit && (mutationType != NONE)) {
         float newSize;
         newSize = mutateSize(generation[i].getSize(),
               mutationType);
         generation[i].setSize(newSize);
      }
      if (modelID != PRECISION_FIXED) {
         for (int k = 0; k < 4; k++) {
            change = ((float) Math.random() * stratMutation * 2)
                  - stratMutation;
            if (newActStrat[k] + change > 1)
               newActStrat[k] = 1;
            else if (newActStrat[k] + change < 0)
               newActStrat[k] = 0;
            else
               newActStrat[k] += change;
         }
         generation[i].setActorStrategy(newActStrat);
      }
      for (int j = 0; j < 4; j++) {
         change = ((float) Math.random() * stratMutation * 2)
               - stratMutation;
         if (newReactStrat[j] + change > 1)
            newReactStrat[j] = 1;
         else if (newReactStrat[j] + change < 0)
            newReactStrat[j] = 0;
         else
            newReactStrat[j] += change;
      }
      generation[i].setReactorStrategy(newReactStrat);
   }
}
private float mutateSize(float s, int mutationType) {
   double change = sizeMutation * meanSize;
   double xInf = (double) s - change;
   double xSup = (double) s + change;
   double randX;
   if (mutationType == RANDOM) {
      randX = (Math.random() * (xSup - xInf)) + xInf;
      if (randX < 0)
         return 0;
      else if (randX > meanSize * 2)
         return meanSize * 2;
      else
         return (float) randX;
   }
   double y1 = findY(xInf,
         (meanSize - (genMeanSize - meanSize)), (float) Math
               .pow((varSize * (varSize / genVarSize)), 1.5));
```

```
        double y2 = findY(xSup,
              (meanSize - (genMeanSize - meanSize)), (float) Math
                    .pow((varSize * (varSize / genVarSize)), 1.5));
        double yMax = 1 / (Math.sqrt((float) Math.pow(
              (varSize * (varSize / genVarSize)), 1.5)
              * 2 * Math.PI));
        float newSize = s;
        boolean end = false;
        if (xInf <= (meanSize - (genMeanSize - meanSize))
              && (meanSize - (genMeanSize - meanSize)) <= xSup) {
           while (!end) {
              randX = (Math.random() * (xSup - xInf)) + xInf;
              double y = findY(randX,
                    (meanSize - (genMeanSize - meanSize)),
                    (float) Math.pow(
                          (varSize * (varSize / genVarSize)), 1.5));
              if (Math.random() <= (y / yMax)) {
                 newSize = (float) randX;
                 end = true;
              }
           }
        }
        else {
           while (!end) {
              randX = (Math.random() * (xSup - xInf)) + xInf;
              double y = findY(randX,
                    (meanSize - (genMeanSize - meanSize)),
                    (float) Math.pow(
                          (varSize * (varSize / genVarSize)), 1.5));
              if (Math.random() <= (y / Math.max(y1, y2))) {
                 newSize = (float) randX;
                 end = true;
              }
           }
        }
        if (newSize < 0)
           return 0;
        else if (newSize > meanSize * 2)
           return meanSize * 2;
        else
           return newSize;
     }
     private double findY(double x, float m, float v) {
        double yMax = 1 / (Math.sqrt(v * 2 * Math.PI));
        double exp = (-Math.pow((x - m), 2)) / (2 * v);
        return yMax * Math.pow(Math.E, exp);
     }
     public float getSize(int i) {
        return generation[i].getSize();
     }
     public float findMeanSize() {
        float sum = 0;
        for (int i = 0; i < n; i++) {
           sum += generation[i].getSize();
        }
        return sum / n;
     }
     public float findVarSize(float mean) {
        double sumat = 0;
        for (int i = 0; i < n; i++) {
           float dif = generation[i].getSize() - mean;
           sumat += Math.pow((double) dif, 2);
        }
        float var = (float) (sumat / (n - 1));
        return var;
     }
     public float findMeanFitness() {
        float sum = 0;
        for (int i = 0; i < n; i++) {
           sum += generation[i].getFitness();
        }
        return sum / n;
     }
     private float findMinFitness() {
        float min = 0;
        if (n > 0)
           min = generation[0].getFitness();
        for (int i = 1; i < n; i++) {
           if (generation[i].getFitness() < min)
              min = generation[i].getFitness();
        }
        return min;
     }
     public float findMeanActorStrategy(int relSize) {
        float sum = 0;
        for (int i = 0; i < n; i++) {
           sum += generation[i].getActorStrategy(relSize);
        }
        return sum / n;
     }
     public float findMeanReactorStrategy(int relSize) {
        float sum = 0;
        for (int i = 0; i < n; i++) {
           sum += generation[i].getReactorStrategy(relSize);
        }
        return sum / n;
     }
     public void findGenSizeDist() {
        genMeanSize = findMeanSize();
        genVarSize = findVarSize(genMeanSize);
     }
  }

  package evolution.model;
  public class Strategy {
     private float actor[];
     private float reactor[];
     public Strategy() {
        actor = new float[4];
        reactor = new float[4];
     }
     public float[] getActor() {
        float a[] = new float[4];
        for (int i = 0; i < a.length; i++) {
           a[i] = actor[i];
        }
        return a;
     }
     public float getActor(int relSize) {
        return actor[relSize];
     }
     public float[] getReactor() {
        float r[] = new float[4];
        for (int i = 0; i < r.length; i++) {
           r[i] = reactor[i];
        }
        return r;
     }
     public float getReactor(int relSize) {
        return reactor[relSize];
     }
     public void setActor(float[] acts) {
        for (int i = 0; i < acts.length; i++) {
           actor[i] = acts[i];
        }
     }
     public void setActor(float act, int relSize) {
        actor[relSize] = (act >= 0 ? act : 0);
     }
     public void setReactor(float[] reacts) {
        for (int i = 0; i < reacts.length; i++) {
           reactor[i] = reacts[i];
        }
     }
     public void setReactor(float react, int relSize) {
        reactor[relSize] = (react >= 0 ? react : 0);
     }
  }

  package evolution.view;
  import java.awt.Color;
  import java.awt.Graphics;
  import java.io.FileWriter;
  import java.io.IOException;
  import javax.swing.JOptionPane;
  import javax.swing.JPanel;
  import evolution.EvolConstants;
  import evolution.event.EvolEvent;
  import evolution.event.EvolEventListener;
  import evolution.model.ModelPrecisionFixed;
  public class EvolView extends JPanel implements EvolConstants,
        EvolEventListener {
     private EvolEvent event;
     private boolean ready = false;
     private float[] sizePoints;
     private float[] fitnessPoints;
     private float[] actStratPoints1;
     private float[][] reactStratPoints1, reactStratPoints2;
     private FileWriter out, outParameters, outSummary;
     public EvolView() {
        setBackground(new Color(230, 230, 230));
     }
     protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        if (ready) {
           drawField(g);
           if (event.state == DURING || event.state == AFTER) {
              for (int i = 0; i < event.generation; i++) {
                 drawPoints(g, i);
              }
           }
           if (event.state == AFTER)
              drawText(g);
        }
     }
     public void evolStepDone(EvolEvent evolEvent) {
        ready = true;
        event = evolEvent;
        switch (event.state) {
           case BEFORE:
              drawBack(getGraphics());
              drawField(getGraphics());
              sizePoints = new float[event.generations];
              fitnessPoints = new float[event.generations];
              if (modelIncludesSignal())
                 actStratPoints1 = new float[event.generations];
              reactStratPoints1 = new float[2][event.generations];
              if (modelIncludesSignal())
                 reactStratPoints2 = new float[2][event.generations];
              break;
           case DURING:
              sizePoints[event.generation - 1] = event.sValue;
              fitnessPoints[event.generation - 1] = event.fValue;
              if (modelIncludesSignal())
                 actStratPoints1[event.generation - 1] =
                       event.pValue1;
              reactStratPoints1[0][event.generation - 1] =
                event.hValue1;
              reactStratPoints1[1][event.generation - 1] =
                event.dValue1;
              if (modelIncludesSignal()) {
                 reactStratPoints2[0][event.generation - 1] =
                   event.hValue2;
                 reactStratPoints2[1][event.generation - 1] =
                   event.dValue2;
              }
              drawPoints(getGraphics(), event.generation - 1);
              break;
           case AFTER:
              drawText(getGraphics());
              break;
        }
     }
     private void drawBack(Graphics g) {
        g.setColor(new Color(230, 230, 230));
        g.fillRect(0, 0, 570, 640);
     }
     private void drawField(Graphics g) {
```

```
   g.setColor(Color.black);
   g.drawLine(50, 210, 50, 10);
   for (int posY = 170; posY > 10; posY -= 40) {
      g.drawLine(45, posY, 50, posY);
      g.drawString("" + ((210 - posY) / 2) + "%", 15, posY + 5);
   }
   g.drawLine(50, 210, 550, 210);
   g.drawString("Generación", 470, 240);
   for (int posX = 150; posX < 550; posX += 100) {
      g.drawLine(posX, 210, posX, 215);
      g.drawString("" + ((int) ((posX - 50) * ((float) event.
        generations / 500))), posX - 5, 230);
   }
   if (modelIncludesSignal()) {
      g.drawLine(50, 530, 50, 330);
      g.drawString("Generación", 470, 560);
      for (int posY = 490; posY > 330; posY -= 40) {
         g.drawLine(45, posY, 50, posY);
         g.drawString("" + ((530 - posY) / 2) + "%", 15,
               posY + 5);
      }
      g.drawLine(50, 530, 550, 530);
      for (int posX = 150; posX < 550; posX += 100) {
         g.drawLine(posX, 530, posX, 535);
         g.drawString("" + ((int) ((posX - 50) * ((float) event.
               generations / 500))), posX - 5, 550);
      }
   }
}
private void drawPoints(Graphics g, int i) {
   int x = 0, yS = 0, yF = 0, yP = 0, yH1 = 0, yD1 = 0, yH2 = 0,
     yD2 = 0;
   x = generationValue(i + 1);
   yS = sizeValue(sizePoints[i]);
   yF = fitnessValue(fitnessPoints[i]);
   if (modelIncludesSignal()) {
      yP = strategyValue(actStratPoints1[i]);
   }
   yH1 = strategyValue(reactStratPoints1[0][i]);
   yD1 = strategyValue(reactStratPoints1[1][i]);
   if (modelIncludesSignal()) {
      yH2 = strategyValue(reactStratPoints2[0][i]);
      yD2 = strategyValue(reactStratPoints2[1][i]);
   }
   if (event.drawSize) {
      g.setColor(Color.green);
      g.drawOval(x, yS, 1, 1);
      if (modelIncludesSignal())
         g.drawOval(x, yS + 320, 1, 1);
   }
   if (event.drawFitness) {
      g.setColor(Color.yellow);
      g.drawOval(x, yF, 1, 1);
      if (modelIncludesSignal())
         g.drawOval(x, yF + 320, 1, 1);
   }
   if (event.drawActStrat && modelIncludesSignal()) {
      g.setColor(Color.black);
      g.drawOval(x, yP, 1, 1);
      g.drawOval(x, yP + 320, 1, 1);
   }
   if (event.drawReactStrat) {
      g.setColor(Color.blue);
      g.drawOval(x, yD1, 1, 1);
      if (modelIncludesSignal())
         g.drawOval(x, yD2 + 320, 1, 1);
      g.setColor(Color.red);
      g.drawOval(x, yH1, 1, 1);
      if (modelIncludesSignal())
         g.drawOval(x, yH2 + 320, 1, 1);
   }
}
private void drawText(Graphics g) {
   g.setColor(Color.black);
   g.drawString(event.model.getComment(2), 50, 260);
   g.setColor(Color.blue);
   g.drawString("" + Math.round((event.dValue1) * 100)
         + event.model.getComment(0), 70, 280);
   g.setColor(Color.red);
   g.drawString("" + Math.round((event.hValue1) * 100)
         + event.model.getComment(1), 70, 300);
   g.setColor(Color.black);
   if (modelIncludesSignal()) {
      g.setColor(Color.black);
      g.drawString(event.model.getComment(3), 50, 580);
      g.setColor(Color.blue);
      g.drawString("" + Math.round((event.dValue2) * 100)
            + event.model.getComment(0), 70, 600);
      g.setColor(Color.red);
      g.drawString("" + Math.round((event.hValue2) * 100)
            + event.model.getComment(1), 70, 620);
      g.setColor(Color.black);
   }
}
private int generationValue(int gen) {
   return (int) (50 + gen / ((float) event.generations / 500));
}
private int sizeValue(float size) {
   float value = size / (event.pop.meanSize * 2);
   return 210 - Math.round(value * 200);
}
private int fitnessValue(float fitness) {
   float value = fitness / (event.encounters * event.v);
   return 210 - Math.round((value + 0.5f) * 200);
}
private int strategyValue(float strategy) {
   return 210 - Math.round(strategy * 200);
}
public void newPaintOptions(EvolEvent evolEvent) {
   event = evolEvent;
   repaint();
}
public void savePopulation(EvolEvent evolEvent) {
   event = evolEvent;
if (event.fileName == null) {
   return;
}
if (event.fileName.getName().equals("")) {
   JOptionPane.showMessageDialog(this,
         "Nombre de archivo incorrecto");
   return;
}
switch (event.state) {
   case BEFORE:
      try {
         out = new FileWriter(event.fileName);
         outParameters = new FileWriter(
               event.fileParametersName);
         outSummary = new FileWriter(event.fileSummaryName);
      }
      catch (IOException e) {
         JOptionPane.showMessageDialog(this,
               "Error al abrir el archivo");
         closeFile();
      }
      try {
         for (int i = 0; i < VARIABLES.length; i++) {
            out.write(VARIABLES[i]);
            out.write("\t");
            out.flush();
         }
         out.write("\n");
         out.flush();
         for (int i = 0; i < MEAN_VARIABLES.length; i++) {
            outSummary.write(MEAN_VARIABLES[i]);
            outSummary.write("\t");
            outSummary.flush();
         }
         outSummary.write("\n");
         outSummary.flush();
      }
      catch (IOException e1) {
         JOptionPane.showMessageDialog(this,
               "Error al escribir en el archivo");
      }
      break;
   case DURING:
      try {
         if (event.saveData) {
            for (int i = 0; i < event.pop.generation.length;
              i++) {
               out.write(Integer.toString(event.generation));
               out.write("\t");
               out.write(Integer.toString(i + 1));
               out.write("\t");
               out.write(Integer.toString(event.pop.
                 generation[i].parent));
               out.write("\t");
               out.write(Float.toString(
                     event.pop.generation[i].getSize())
                     .replace(".", ","));
               out.write("\t");
               out.write(Integer.toString(
                     event.pop.generation[i].getEncounters())
                     .replace(".", ","));
               out.write("\t");
               out.write(Float.toString(
                     event.pop.generation[i].getFitness())
                     .replace(".", ","));
               out.write("\t");
               out.write(Integer.toString(
                     event.pop.generation[i].getOffspring())
                     .replace(".", ","));
               out.write("\t");
               out.write(Float.toString(
                     event.pop.generation[i]
                           .getActorStrategy(UNKNOWN))
                     .replace(".", ","));
               out.write("\t");
               out.write(Float.toString(
                     event.pop.generation[i]
                           .getActorStrategy(BIGGER))
                     .replace(".", ","));
               out.write("\t");
               out.write(Float.toString(
                     event.pop.generation[i]
                           .getActorStrategy(SMALLER))
                     .replace(".", ","));
               out.write("\t");
               out.write(Float.toString(
                     event.pop.generation[i]
                           .getReactorStrategy(UNKNOWN))
                     .replace(".", ","));
               out.write("\t");
               out.write(Float.toString(
                     event.pop.generation[i]
                           .getReactorStrategy(BIGGER))
                     .replace(".", ","));
               out.write("\t");
               out.write(Float.toString(
                     event.pop.generation[i]
                           .getReactorStrategy(SMALLER))
                     .replace(".", ","));
               out.write("\t");
               out.write("\n");
               out.flush();
            }
         }
         outSummary.write(Integer.toString(event.
           generation));
         outSummary.write("\t");
         outSummary.write(Float.toString(
               event.pop.findMeanSize()).replace(".", ","));
         outSummary.write("\t");
         outSummary
               .write(Float.toString(
                     event.pop.findMeanFitness()).replace(
                     ".", ","));
```

```
            outSummary.write("\t");
            outSummary.write(Float.toString(
                  event.pop.findMeanActorStrategy(UNKNOWN))
                  .replace(".", ","));
            outSummary.write("\t");
            outSummary.write(Float.toString(
                  event.pop.findMeanActorStrategy(BIGGER))
                  .replace(".", ","));
            outSummary.write("\t");
            outSummary.write(Float.toString(
                  event.pop.findMeanActorStrategy(SMALLER))
                  .replace(".", ","));
            outSummary.write("\t");
            outSummary.write(Float.toString(
                  event.pop.findMeanReactorStrategy(UNKNOWN))
                  .replace(".", ","));
            outSummary.write("\t");
            outSummary.write(Float.toString(
                  event.pop.findMeanReactorStrategy(BIGGER))
                  .replace(".", ","));
            outSummary.write("\t");
            outSummary.write(Float.toString(
                  event.pop.findMeanReactorStrategy(SMALLER))
                  .replace(".", ","));
            outSummary.write("\t");
            outSummary.write("\n");
            outSummary.flush();
         }
         catch (IOException e1) {
            JOptionPane.showMessageDialog(this,
                  "Error al escribir en el archivo");
         }
         break;
      case AFTER:
         try {
            outParameters.write(PARAMETERS[0]);
            outParameters.write("\t");
            outParameters.write(Integer.toString(event.pop.n));
            outParameters.write("\n");
            outParameters.write(PARAMETERS[1]);
            outParameters.write("\t");
            outParameters.write(Float.toString(
                  event.pop.meanSize).replace(".", ","));
            outParameters.write("\n");
            outParameters.write(PARAMETERS[2]);
            outParameters.write("\t");
            outParameters
                  .write(Float.toString(event.pop.varSize)
                        .replace(".", ","));
            outParameters.write("\n");
            outParameters.write(PARAMETERS[3]);
            outParameters.write("\t");
            outParameters.write(Integer
                  .toString(event.generations));
            outParameters.write("\n");
            outParameters.write(PARAMETERS[4]);
            outParameters.write("\t");
            outParameters.write(MODELS[event.model.modelID]);
            outParameters.write("\n");
            if (event.model.modelID == PRECISION_FIXED) {
               outParameters.write("Precision");
               outParameters.write("\t");
               outParameters.write(Float.toString(
                     ModelPrecisionFixed.precision).replace(".",
                     ","));
               outParameters.write("\n");
            }
            outParameters.write(PARAMETERS[5]);
            outParameters.write("\t");
            outParameters.write(Float.toString(
                  event.pop.sizeMutation).replace(".", ","));
            outParameters.write("\n");
            outParameters.write(PARAMETERS[6]);
            outParameters.write("\t");
            outParameters.write(Float.toString(
                  event.pop.stratMutation).replace(".", ","));
            outParameters.write("\n");
            outParameters.write(PARAMETERS[7]);
            outParameters.write("\t");
            outParameters.write(Float.toString(event.k).replace(
                  ".", ","));
            outParameters.write("\n");
            outParameters.write(PARAMETERS[8]);
            outParameters.write("\t");
            outParameters.write(Float.toString(event.v).replace(
                  ".", ","));
            outParameters.write("\n");
            outParameters.write(PARAMETERS[9]);
            outParameters.write("\t");
            outParameters.write(Boolean.toString(event.
              inherit));
            outParameters.write("\n");
            outParameters.write(PARAMETERS[10]);
            outParameters.write("\t");
            outParameters.write(MUTATIONS[event.mutationType]);
            outParameters.write("\n");
            outParameters.flush();
         }
         catch (IOException e1) {
            JOptionPane.showMessageDialog(this,
                  "Error al escribir en el archivo");
         }
         try {
            out.close();
            outParameters.close();
            outSummary.close();
         }
         catch (IOException e) {
            JOptionPane.showMessageDialog(this,
                  "Error al cerrar el archivo");
            System.exit(1);
         }
         break;
   }
   private boolean modelIncludesSignal() {
      if (event.model.modelID == PRECISION_FIXED
            || event.model.modelID == PRECISION_VARIABLE
            || event.model.modelID == DECEIVE)
         return true;
      else
         return false;
   }
   private void closeFile() {
      try {
         out.close();
         outParameters.close();
         outSummary.close();
         System.exit(0);
      }
      catch (IOException e) {
         JOptionPane.showMessageDialog(this,
               "Error al cerrar el archivo");
         System.exit(1);
      }
   }
}
```