Grado en Ingeniería Informática

Inteligencia Artificial

Curso 2019/2020

Guión 3
Java
Juegos IC

Profesor: Cristóbal J Carmona

Grupo: B

Rafael Osuna Ventura

31024321N

rov00005@red.ujaen.es

José Antonio Morales Velasco

76593264K

jamv0007@red.ujaen.es

Índice

Puntuación del jugador obtenida

Esta es la tabla de resultados tras 5 min jugando al juego del Flappy Bird

Número de Partidas	Puntuación Máxima	Minuto en el que se alcanzó la puntuación
32	6	2:28

Estudio del Juego

En la siguiente tabla se resume la ejecución del juego 10 veces en las que se cumplen las condiciones de 10000 puntos o más o su generación es 25 o mayor.

Partida	Generación	Puntuación
1	13	10095
2	4	10194
3	25	328
4	20	10107
5	18	10077
6	17	10442
7	1	10243
8	25	415
9	25	308
10	12	10529
Media	16	7273.8

Como se puede observar las puntuaciones aceptadas los valores se obtienen en generaciones alrededor de la 20 en algunos casos y puede que hasta en generaciones menores a esta. También se da el caso de que no se obtiene puntuación mayor o igual que 10000 aunque su puntuación en este caso es bastante baja, y su generación es 25 o más, no consigue evolucionar para conseguir más puntuación independientemente de las generaciones que pasen.

Modificación de Operadores del Juego

Operador de cruce (Genetic Crossover)

El cruce en dos puntos como lo hemos implementado coge el vector data con el genoma 1 convertido a JSON y genera dos variables a y b inicializadas a 0 que serán las posiciones entre las cuales se ejecutará el cruce con el genoma 2. Después hemos implementado un bucle do-while para controlar que las variables no tengan el mismo valor.

```
Generation.prototype.breed = function (g1, g2, nbChilds) {
    var datas = [];
    for (var nb = 0; nb < nbChilds; nb++) {
        // Deep clone of genome 1.
        var data = JSON.parse(JSON.stringify(g1));
        var a = 0;
        var b = 0;
        do{
            a = numeroAleatorio(1,data.network.weights-2);
            b = numeroAleatorio(1,data.network.weights-2);

            if(a < b){
                for (var i = a; i < b; i++) {
                    data.network.weights[i] = g2.network.weights[i];
                }
            }else if(a > b){
                var aux = a;
                a = b;
                b = aux;

                for (var i = a; i < b; i++) {
                    data.network.weights[i] = g2.network.weights[i];
                }

            }
        }while(a == b);

        //for (var i in g2.network.weights) {
            // Genetic crossover
            // 0.5 is the crossover factor.
            // FIXME Really should be a predefined constant.
```

*Sin título: Bloc de notas
Archivo Edición Formato Ver Ayuda
Jose Antonio Morales Velasco 76593264K
Rafael Osuna Ventura 31024321N
Ln 2, C 100% Windows (CRLF) UTF-8

Generamos las posiciones de a y b de forma aleatoria con una función auxiliar que devuelve un valor aleatorio entre dos números dados:

```
function numeroAleatorio(min,max){
    return Math.floor(Math.random() * (max - min) + min);
}

/**
 * Breed to genomes to produce offspring(s).
 *
 * @param {g1} Genome 1.
 * @param {g2} Genome 2.
```

Archivo Edición Formato Ver Ayuda
Jose Antonio Morales Velasco 76593264K
Rafael Osuna Ventura 31024321N
Ln 1, C 100% Windows (CRLF) UTF-8

Tras obtener las dos posiciones de forma aleatoria comprobamos las posiciones , si a es menor que b, recorremos los pesos del genoma 2 e intercambiamos estos por los del genoma 1 en las mismas posiciones del genoma 2.

Si b es mayor que a intercambiamos los valores y hacemos el mismo procedimiento que antes.

Si en caso son iguales, no nos dejaría salir del bucle while por lo que buscaría de nuevo el intervalo a cruzar.

Hemos indicado a la función que da los valores del intervalo aleatoriamente que empiece en el valor 1 del vector de pesos y termine en el penúltimo.

Operador de Mutación (Mutation Operator)

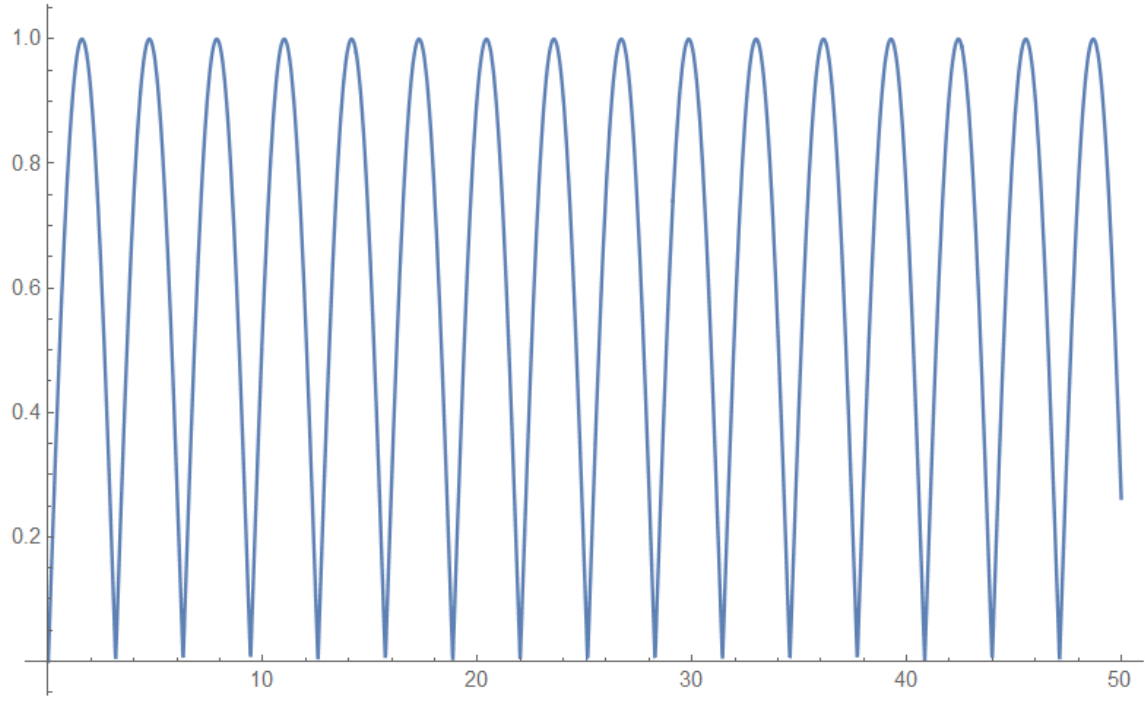
```
*/
Generation.prototype.breed = function (g1, g2, nbChilds) {
    var datas = [];
    for (var nb = 0; nb < nbChilds; nb++) {
        // Deep clone of genome 1.
        var data = JSON.parse(JSON.stringify(g1));
        for (var i in g2.network.weights) {
            // Genetic crossover
            // 0.5 is the crossover factor.
            // FIXME Really should be a predefined constant.
            if (Math.random() <= 0.5) {
                data.network.weights[i] = g2.network.weights[i];
            }
        }

        // Perform mutation on some weights.
        for (var i in data.network.weights) {
            if (Math.random() <= self.options.mutationRate) {
                data.network.weights[i] = Math.abs(Math.sin(numeroAleatorioNoExacto(0,50)));


            }
        }
        datas.push(data);
    }

    return datas;
}
```

*Sin título: Bloc de notas
Archivo Edición Formato Ver Ayuda
Jose Antonio Morales Velasco 76593264K
Rafael Osuna Ventura 31024321N
Ln 2, C 100% Windows (CRLF) UTF-8

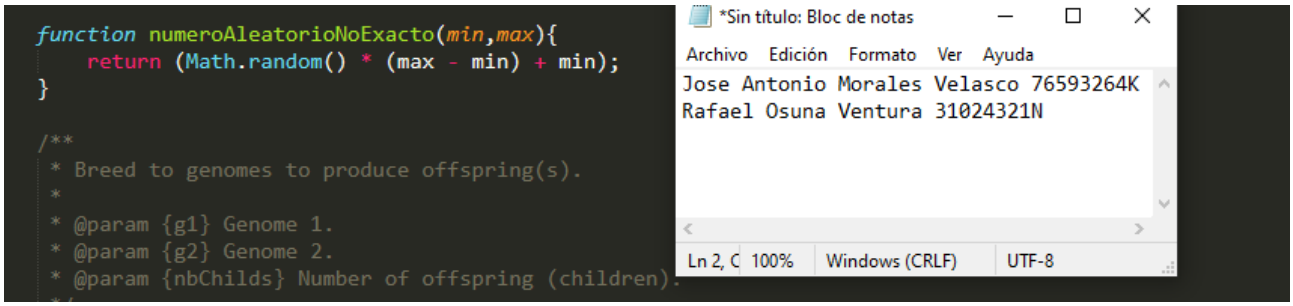
Para modificar el operador de mutación sustituyendo la función, hemos usado la función:

$$F(x) = |Sin(X)|$$

Esta función trigonométrica tiene un valor en el intervalo [-1, 1] y su movimiento es creciente y decreciente continuamente. Por eso ya que los valores pueden llegar a ser negativos usamos el valor absoluto para que siempre sea positivo y su intervalo sea [0, 1], este sería su gráfica:



Para su implementación usaremos como valor de x un valor aleatorio entre los dos pasados donde el valor devuelto será un valor decimal entre los dos pasados. La función es similar a la anterior:



Hemos escogido esta función puesto que tiene la misma probabilidad de que aleatoriamente de que devuelva un valor mayor a 0.5 y de que devuelva uno menor de este además los valores que puede contemplar esta función son muy continuos, varían muy poco de un valor decimal al siguiente lo que hace más exacto el valor del peso lo que hace las mismas posibilidades 50% de cada estado.

Le hemos pasado que los valores entre los que calcule la función sea entre 0 y 50 aunque el intervalo aumenta la probabilidad no cambia.

Tablas de datos de cada Modificación del Operador

A. Nuevo cruce y mutación por defecto.

Partida	Generación	Puntuación
1	25	325
2	25	1587
3	25	1038
4	25	504
5	25	398
Media Aritmética	16	7273.8

B. Cruce por defecto y mutación nueva.

Partida	Generación	Puntuación
1	25	516
2	25	488
3	25	408
4	9	10000
5	16	10000
Media Aritmética	20	4282.4

C. Cruce y mutación nueva.

Partida	Generación	Puntuación
1	25	426
2	6	10000
3	25	398
4	25	1411
5	7	10000
Media Aritmética	17.6	4447

Partida	Generación	Puntuación
MEDIA ORIGINAL	16	7273.8
MEDIA A	25	770.4
MEDIA B	20	4282.4
MEDIA C	17.6	4447

Tras observar los resultados obtenidos podemos ver que respecto a las medias de los juegos con los operadores modificados, la partida C tiene una mejor media que las anteriores puesto que con el cruce modificado tiene más posibilidades de modificar la genética con los valores de los anteriores y al tener también modificado el operador de mutación, los pesos que cambia para saber si va a volar o no son equivalentes ya que debido a la función que hemos usado las probabilidades de volar o no son iguales.

Con respecto a la media original podemos observar que esta última tiene mejor puntuación que las modificadas tanto en generación como en puntuación aunque esto depende también mucho de la ejecución del juego.

Comparando los resultados obtenidos de cualquier partida ya sea del original o modificado con los resultados obtenidos con la partida que realizamos nosotros en el juego vemos que existe una diferencia descomunal puesto que las personas necesitamos un mayor tiempo para el aprendizaje al contrario que usando inteligencia artificial.