Aprendizaje a partir de Redes Neuronales: Nidhogg.

Descripción del juego:

Nidhogg es un juego de acción en 2D, en el que dos jugadores se baten en duelos con espada. La función del jugador es elegir entre tres opciones disponibles para **colocar la espada(arriba, medio, abajo),** un ataque que le dejará inmóvil, y un desplazamiento lateral con el que nunca das la espalda al contrincante.

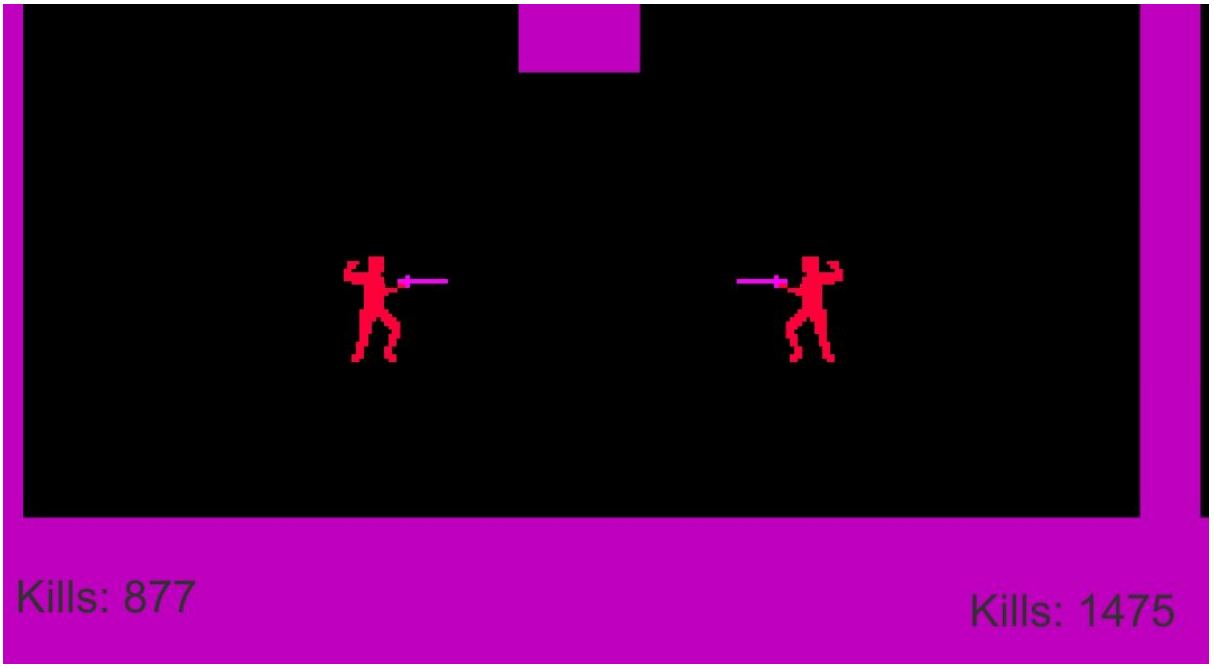
En este juego la situación inicial se vuelve muy compleja, ya que el resultado de los movimientos es ligeramente caótico y determinista.

Documentación útil para el entendimiento del proyecto:

Al comienzo el proyecto estaba planteado para que un jugador se enfrentará a una IA con una red neuronal formada por un aprendizaje base y una serie de refuerzos. Sin embargo tras avanzar un poco con este trabajo, nos dimos cuenta que es muy difícil demostrar que la red neuronal aprende mientras juega contra un humano, ya que este cambia patrones de ataque constantemente y la IA tiene que jugar MUCHAS partidas para llegar a aprender mínimamente.

Por esta razón, buscamos una manera de solucionar esto, y para ello decidimos crear una pequeña IA basada en decisiones. Sin embargo, no nos convencía ya que pensábamos que muchas veces ganaba por los propios refuerzos en vez de por un aprendizaje basado la sucesión de partidas.

Esta idea la descartamos, y se nos ocurrió una mucho mejor: usar la misma red neuronal para las dos IAs, basadas en el mismo entrenamiento pero una de ellas sin refuerzos. De esta forma podemos ver como al principio su comportamiento es básicamente igual, y tras el paso de jugadas, la IA con aprendizaje acaba por distanciarse en el marcador.



Para terminar, hay que destacar que para crear los refuerzos nos hemos centrado en intentar hacer los menos posibles y que no sean muy determinantes para ganar la partida, ya que de esta forma en vez de crear una red neuronal basada en el aprendizaje, básicamente estás creando una IA por estados.

Por otro lado, los inputs de información que usamos son los siguientes:

- Distancia (Float : Medidor de distancia).
- Posición de la espada del contrincante (Enum : Arriba, Medio, Abajo).
- Ataque del contrincante (Bool : Ataca o no ataca).

De esta manera, y aunque usemos 3 inputs distintos, el más destacado es el cálculo de la distancia entre los jugadores, ya que a raíz de este, se despliega el juego. Por esta razón, los refuerzos “base” (no son excesivamente relevantes para ganar) creados en la red neuronal están basados sobretodo en esta distancia:

- 1º → Cuando están muy lejos entre ellos, se coloca la espada al medio (mejor posición, ya que solo tiene un paso para llegar arriba o abajo) y se avanza hacia delante.

```
//Cuando la distancia es mayor que 5 reforzamos que la espada este en el centro y la IA avance
if (distancia > 5.0f)
{
    if (actualstate == stateArm.bot)
    {
        output = new float[] { 0.1f, 0.1f, 0.1f, 0.1f, 0.5f, 0.1f, 0.1f };
        cerebro.ReentrenarRed(inputs, output);
    }
    else if (actualstate == stateArm.top)
    {
        output = new float[] { 0.1f, 0.1f, 0.1f, 0.1f, 0.1f, 0.5f, 0.1f };
        cerebro.ReentrenarRed(inputs, output);
    }
    else
    {
        // REFUERZO AL MOVETO
        output = new float[] { 0.1f, 0.9f, 0.1f, 0.1f, 0.1f, 0.1f, 0.1f };
        cerebro.ReentrenarRed(inputs, output);
    }
}
```

- 2º → A media distancia, reforzamos el cambio de espada a otra posición

```
//A media distancia reforzamos un cambio de espada para poder atacar
else if (distancia < 5.0f && distancia > 3.0f)
{
    if (inputs[2] == 0)
    {
        if (inputs[1] == (int)actualstate)
        {
            output = new float[] { 0.1f, 0.1f, 0.1f, 0.1f, 0.6f, 0.8f, 0.1f };
            cerebro.ReentrenarRed(inputs, output);
        }
        else
        {
            output = new float[] { 0.4f, 0.7f, 0.1f, 0.1f, 0.4f, 0.5f, 0.1f };
            cerebro.ReentrenarRed(inputs, output);
        }
    }

}
```

- 3º → A corta distancia, reforzamos el ataque o la huida, dependiendo de si el contrincante está atacando o no.

```
//Cuando la distancia es pequeña reforzamos el ataque dependiedno de la posicion de la espada del enemigo
else if (distancia < 3.0f)
{
    if (inputs[2] == 0)
    {
        if (inputs[1] != (int)actualstate)
        {
            output = new float[] { 0.6f, 0.1f, 0.3f, 0.1f, 0.1f, 0.1f, 0.1f };
            cerebro.ReentrenarRed(inputs, output);
        }
        else
        {
            output = new float[] { 0.4f, 0.2f, 0.1f, 0.1f, 0.6f, 0.4f, 0.1f };
            cerebro.ReentrenarRed(inputs, output);
        }
    }
    else if (inputs[2] == 1)
    {
        if (inputs[1] != (int)actualstate)
        {
            output = new float[] { 0.3f, 0.2f, 0.6f, 0.1f, 0.1f, 0.1f, 0.1f };
            cerebro.ReentrenarRed(inputs, output);
        }
        else
        {
            output = new float[] { 0.4f, 0.5f, 0.6f, 0.1f, 0.1f, 0.1f, 0.1f };
            cerebro.ReentrenarRed(inputs, output);
        }

    }
}
```

Al igual que estos refuerzos son “base” y hemos creado un refuerzo extra que es mucho más importante a la hora del aprendizaje, y consigue al ganar o perder la partida. De esta manera podemos fomentar las jugadas buenas, y castigar las malas.

Refuerzo positivo al matar.

```
if (abuelo.output[(int)abuelo.currentAction] < 0.9f)
{
    Debug.Log("la ia aprende");
    abuelo.output[(int)abuelo.currentAction] = abuelo.output[(int)abuelo.currentAction] + 0.1f;
    abuelo.cerebro.ReentrenarRed(abuelo.inputs, abuelo.output);
}
```

Refuerzo negativo al morir.

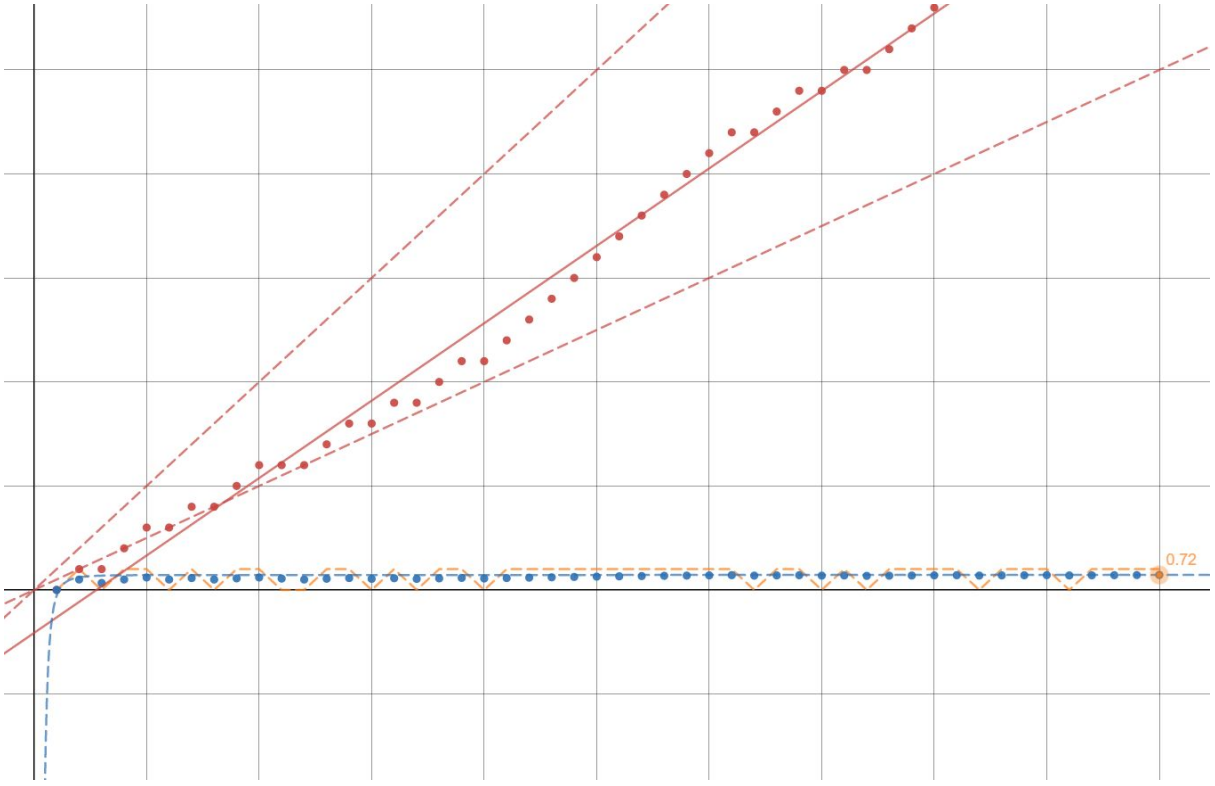
```
if (enemyscr.output[(int)enemyscr.currentAction] > 0.1f)
{
    Debug.Log("La ia desaprende");
    float aux = enemyscr.output[(int)enemyscr.currentAction] - .1f;
    if (aux < 0.1f)
    {
        enemyscr.output[(int)enemyscr.currentAction] = .1f;
    }
    else
    {
        enemyscr.output[(int)enemyscr.currentAction] = enemyscr.output[(int)enemyscr.currentAction] - .1f;
    }
    enemyscr.cerebro.ReentrenarRed(enemyscr.inputs, enemyscr.output);
}
```

Gráficas de datos:

Probando una IA como conejillo de indias contra nuestra IA entrenada obtuvimos unos resultados de mejora paulatina, tal y como predecimos.

Las dos IAs enfrentadas constaban del mismo entrenamiento base, y una de ellas, unos refuerzos en función de las acciones.

	Mejora acumulada.Valores máximos en el límite superior. Valores medios límite inferior.
	Valor medio de la muestra. Regresión logarítmica
	Muestras. 0=Lose 1=Win



Según los datos observado, obtenemos una regresión con la forma:

$$Y = 0.7446*X - 2.067$$

Una pendiente de 0.72 [inversa 1.3]
Significante en un 99.43%

Conclusiones:

Podemos ver como esta metodología de aprendizaje es capaz de asimilar los conceptos para poder entender qué patrones son mejores candidatos a sobrevivir.

Este tipo de aprendizajes son muy sensibles a los refuerzos, y para conceptos muy caóticos puede no llegar a ser tan útil. Sin embargo vemos notablemente los resultados tras muchas “partidas”.