

Aprendizaje por Refuerzo: Uso de Framework para robots en RealTime Battle

RealtimeBattle es un juego en el que robots controlados por programas software luchan entre sí y que proporciona un protocolo extremadamente sencillo para recibir información del entorno de la batalla y enviar las acciones de nuestro robot. RTB es software libre por lo que todo su código es susceptible de ser estudiado y/o modificado.

En este trabajo se ha desarrollado un motor genérico para aprendizaje de políticas de acciones basadas en el algoritmo de Sutton y Barton para el cálculo de la tabla-Q óptima. Dicho motor se ha utilizado y probado en las luchas en RTB, variando entorno y parámetros, y se muestran los resultados y primeras mejoras perceptibles, abriendo además la investigación de la propia tabla-Q para saber más sobre el entorno. Se exponen además las técnicas e ideas utilizadas para la selección de sensores, estados y acciones sobre el entorno.

Autor: José Luis Marina

Esta es una obra "Creative Commons" con algunos Derechos Reservados



<http://creativecommons.org/licenses/by-nc/2.5/es/>

Contenido

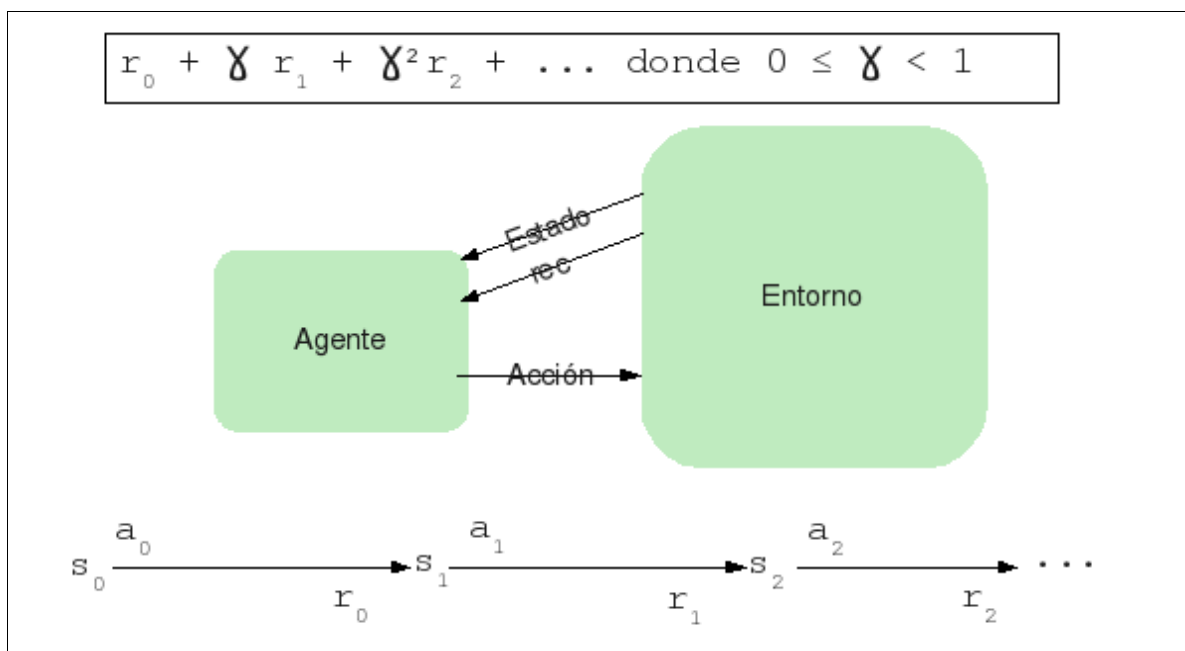
| | |
|--|----|
| Introducción..... | 3 |
| Trabajo Realizado..... | 6 |
| Estudio y diseño de universo de Estados y Acciones:..... | 7 |
| Implementación de algoritmo de función o tabla Q:..... | 9 |
| Resultados y Experimentos..... | 11 |
| Conclusiones..... | 13 |
| Trabajo Futuro..... | 14 |
| Principales Referencias..... | 15 |

Introducción

El Aprendizaje por Refuerzo está teniendo buenos resultados en el campo del aprendizaje de políticas de acciones correctas en la consecución de objetivos en juegos entre agentes o bots, con la ventaja añadida de que puede interactuarse con los entornos sin problemas de riesgos de otros sistemas reales, y utilizar al mismo tiempo una gran cantidad de datos sobre la que aplicar los métodos estadísticos necesarios.

El problema genérico que pretende resolver el Aprendizaje por Refuerzo (AR) es el de cómo un agente autónomo que siente y reacciona con su entorno, puede aprender a elegir acciones óptimas para la consecución de sus objetivos [1].

Los agentes ejecutan acciones contra el entorno, y reciben una recompensa (o castigo) numérico que les indica si están consiguiendo o alejándose de los objetivos. Además de ejecutar acciones y recibir recompensas, cuentan con sensores sobre el entorno que les dan una idea del estado actual y por lo tanto de la sucesión de estados a lo largo del tiempo.



La tarea del agente en cada momento, dado un estado, es elegir la acción para maximizar la suma de las recompensas futuras. A las recompensas futuras se les aplica un factor de corrección- γ - para que tengan mayor o menor peso en el estado presente.

Haciendo que el conjunto de estados sea finito y observable, que las acciones sean deterministas y que la recompensa o el nuevo estado resultado de una acción sólo dependan del estado actual, y no de los anteriores, podemos modelar el problema mediante un Proceso de Decisión de Markov (PDM).

Finalmente la tarea del agente deberá ser estimar la tabla-Q de valores para los pares Estado – Acción ($\langle s,a \rangle$) que guarda las recompensas máximas posibles a conseguir para dicho par.

Esta estimación, para un PDM y con $0 \leq \gamma < 1$ converge con el tiempo a la tabla-Q ideal que maximiza las recompensas, aplicando el siguiente algoritmo:

Inicializar la tabla-Q con valor inicial (Hemos utilizado 0)

Mientras no acabe el juego:

$s \leftarrow$ Observar estado.

$a \leftarrow$ Elegir acción (Máximo valor de Q o exploración al azar)

$r, s' \leftarrow$ Ejecutar Acción. Obtenemos recompensa r y nuevo estado s' .

$Q(s,a) \leftarrow$ Actualizamos entrada de tabla-Q estimada con:

$$Q(s,a) + \alpha (r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

Esta es la fórmula de Sutton y Barton con:

α → Tamaño de paso (maneja diferencias entre recompensas)

Si su valor es 1 la fórmula se corresponde con la original explicada en el libro de Mitchell.

r → Recompensa inmediata.

γ → Factor de descuento de recompensas futuras.

RealTimeBattle – RTB – es un entorno en que varios robots pueden luchar en diferentes modalidades con el objetivo de vencer a los demás.

Aunque las reglas de RTB son muy sencillas, construir programas robot con la suficiente inteligencia para adaptarse al entorno cumpliendo su objetivo de destruir a los contrarios no resulta sencillo.

Organizar torneos en RTB es extremadamente fácil. Sólo se le indica al programa cuál es la arena o forma del terrenos de lucha en dos dimensiones, y el número cantidad de los robots en lidia. Pueden variarse multitud de parámetros como el valor de gravedad o la resistencia los choques de los robots, pero para el propósito de este trabajo basta con los valores por defecto.

El programa corre en ventaja a sistemas UNIX y ofrece un interfaz gráfico en tiempo real de lo que ocurre en la arena y con los robots.

La comunicación entre RTB y los robots se realiza utilizando la salida y la entrada estándar de los procesos (un robot = un proceso), por lo que el lenguaje de programación no es un problema siempre y cuando se hable el protocolo a través de dichos descriptores. Existen ejemplos de robots en C, C++, perl, Java o Python.

Programar un interfaz para que los robots puedan comunicarse por red con RTB no parece una tarea excesivamente complicada, aunque no es objetivo de este trabajo.

Las características principales de RTB son:

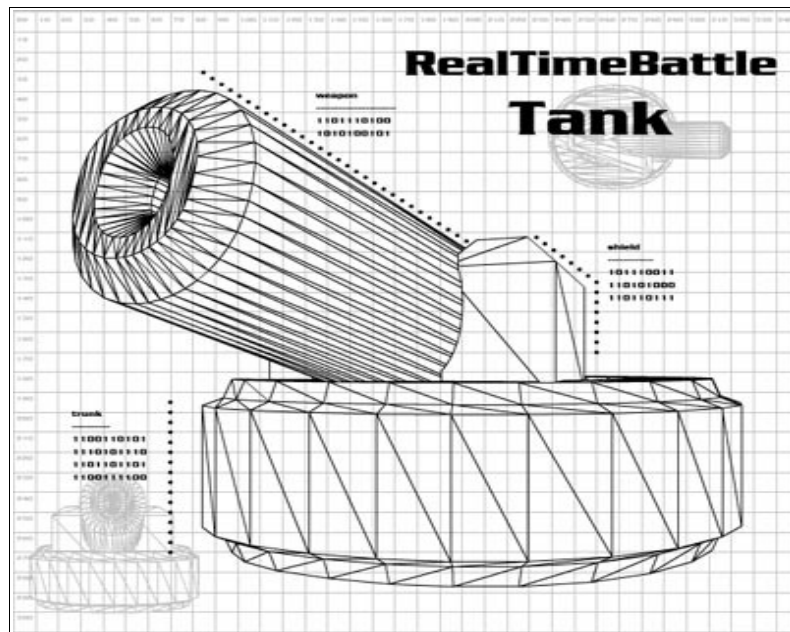
- **Desarrollo del juego en tiempo real.** Cada robots es un proceso hijo de RTB.
- **Cualquier lenguaje de programación** es adecuado. Las comunicaciones RTB–Robot se realizan a través de la salida y entrada estándar.
- **El protocolo** de los mensajes es bastante **sencillo**.
- **Hasta 120 robots** pueden competir en un mismo campo de batalla o arena.
- **Interfaz gráfico** y seguimiento de estado en tiempo real.
- **Poco exigente** en cuanto a requisitos hardware y software. Sistemas UNIX.
- **Los robots están sujetos a leyes físicas** que el entorno simula. Velocidad, aceleración, giros, choques, etc.

Características de los robots:

- **Se comportan como Vehículos con Ruedas:** Ruedan hacia adelante con un pequeño rozamiento de giro y pueden deslizarse lateralmente con una fricción mucho mayor. Un robot puede **acelerar, rotar y frenar**.
- **Radar:** El radar es la forma que tiene un robot de percibir el mundo que le rodea y tiene una dirección. Un robot puede moverse en una dirección y girar el radar en otra diferente. Un robot recibe a lo largo de la contienda mensajes de radar con información sobre el tipo del objeto más cercano en su dirección y la distancia a la que se encuentra.

La información del radar es lo único que percibirá el robot.

- **Cañón:** Un robot puede realizar disparos en la dirección del cañón con la energía que decida, siempre y cuando su propia energía se lo permita. La velocidad del proyectil es la suma de la velocidad del robot más la velocidad del disparo.
- **La salud de un robot se mide por su Energía:** La energía se pierde al ser alcanzado por un disparo, al disparar, al chocar contra paredes otros robots o **minas**. Un robot incrementa su energía comiendo galletas.



Las principales razones para utilizar RTB y frente a otros motores de juegos son su facilidad de uso y su independencia de terceros en cuanto a código o herramientas, además de que proporciona un entorno completo para experimentar con algoritmos de aprendizaje en situaciones muy variadas tanto por terrenos como por contrincantes, y además y para finalizar, consta también de modo colaborativo en el que enfrentar a equipos de robots.

Trabajo Realizado

Los objetivos principales del trabajo son:

1.- Aplicación de Aprendizaje por refuerzo a entornos complejos.

Comprobar empíricamente:

La adaptación a la resolución del problema

Complejidad de los Estados.

2.- Hacer una implementación genérica del algoritmo de aprendizaje de la tabla-Q basado en la fórmula de Sutton y Barton.

Probarlo en RTB.

Prepararlo para otros entornos

Para lograr estos objetivos las tareas se han acometido en las siguientes fases cada una con sus propias tareas y resultados:

Estudio y diseño de universo de Estados y Acciones:

Respecto a las acciones, en este trabajo, se ha optado por dotar al agente de un conjunto de acciones reducido pero más complejo que las ofrece de base el juego.

RTB dispone de una gran cantidad de acciones que enviar al servidor ya que la mayoría de ellas tienen parámetros que varían su comportamiento y las dotan de enorme variedad. Por ejemplo tenemos la acción "Girar" que puede tener como parámetros el elemento o elemento a girar (robot, radar, cañón), velocidad de giro (dependiendo del elemento a girar cambia respecto de qué coordenadas), sentido del giro, etc.

Las acciones básicas para el movimiento y la lucha de los robots que proporciona RTB son principalmente éstas:

```
Rotate [what to rotate (int)] [angular velocity (double)]  
RotateTo [what to rotate (int)] [angular velocity (double)] [end angle (double)]  
RotateAmount [what to rotate (int)] [angular velocity (double)] [angle (double)]  
Sweep [what to rotate (int)] [an vel (double)] [r angle (double)] [l angle (double)]  
Accelerate [value (double)]  
Brake [portion (double)]  
Shoot [shot energy (double)]
```

() Ver todos los comandos en [6]*

Muchas de estas acciones están limitadas por valores máximos o mínimos como por ejemplo la aceleración máxima que puede aplicarse a un robot,

Con combinaciones de las acciones básicas se han construido siete comportamientos más complejos que son los que se han proporcionado como acciones al motor. Los comportamientos son los éstos:

```
// Actions  
void patrol();           // Patrullar evitando paredes y disparar enemigos.  
void fire_around();     // Disparar dando círculos amplios.  
void be_quiet();        // Buscar un sitio seguro.  
void go_for_cookies();  // Ir a por galletas.  
void fire_crazy();     // Disparar constantemente dando vueltas.  
void rambow();         // Buscar e ir a por enemigos disparando.  
void stop();           // Pararse, sólo girar disparando a robots y minas
```

Cada una de estas acciones elaboradas se han probado primero de forma individual comprobando que el comportamiento real se asemeja al descrito. El objetivo principal en esta fase ha sido suministrar un conjunto de acciones lo suficientemente diferentes entre sí para poder utilizarlas en un comportamiento adaptativo.

Un ejemplo de sencillo de acción elaborada es:

```
void RTB_Rebote::fire_crazy()
{
    brake_ = 0.2;
    robot_rotation_ = robot_max_rotate_;
    acceleration_ = 0.2;

    //cout << "Sweep 6 " << 0.0 << " " << 0.0 << " " << 0.0 << endl;
    cout << "Brake " << brake_ << endl;
    cout << "Rotate 1 " << robot_rotation_ << endl;
    cout << "Accelerate " << acceleration_ << endl;

    shoot(0.5);
}
```

Respecto a los estados y los sensores, de toda la información disponible, se han utilizado el radar del robot y cierta información del entorno:

- **Información del radar:**

Radar [distance (double)] [observed object type (int)] [radar angle (double)]

Objeto puede ser Robot – Pared – Galleta – Mina – Bala

Además obtenemos la distancia y el ángulo relativo al frontal del robot.

- **Número de Robots:**

Número de robots sobre el terreno de juego. Es importante para nuestros objetivos que este número baje con el tiempo.

- **Energía:**

Energía del Robot. Lo normal es que comience en 100 y vaya bajando según somos atacados, colisionamos con otros objetos o disparamos. La única manera de que suba es comiendo galletas.

- **Impacto de bala recibidos:**

Podemos medir los impactos de bala recibidos en los últimos dos segundos por ejemplo.

En total se han programado 8 sensores diferentes cada uno con tres valores posibles, cuantificando normalmente “poco o nada”, “medio” y “mucho”.

En combates con 20 robots: Poco < 4, Medio < 8, Mucho < XX.

En este caso la tabla Q de estados por acciones tiene un tamaño de:

$$3^8 \times 7 = 45927$$

Para la función recompensa se ha utilizado una función que tuviera en cuenta de la energía del robot en cada momento y el número de robots restante en lucha, de forma que se castigue la pérdida de energía y se premie el que queden menos contrincantes.

```
reward = this->energy_ + 400.0/(robots_left_ * robots_left_);
```

Implementación de algoritmo de función o tabla Q:

Una vez que tenemos claros los sensores y los estados en que derivan y la función recompensa podemos construir la función de aprendizaje de tablas-Q, que en este trabajo se ha hecho genérica para que puede ser usada en otros entornos.

La idea ha sido implementar un proceso lo más genérico posible para el recorrido y actualización de tablas Q, que pueda servir tanto para robots en RTB como en otros problemas. Además las tablas pueden guardarse en fichero una vez realizado el aprendizaje y así utilizarse directamente en futuros torneos o adaptaciones a otros tipos de sistemas.

El algoritmo de uso típico de esta clase RTB_Qtable es:

Declarar un objeto Qtable.

Inicializarlo con:

Número de acciones disponibles.

Número de estados diferentes.

Valor de inicialización de la tabla.

Nombre de Fichero de almacenamiento de tabla-Q.

Porcentaje de acciones elegidas al azar (exploración).

Estoy aprendiendo? (Si ó No)

Valor de Alpha en algoritmo Barton & Sutton.

Mientras dure el juego o adaptación:

Recuperar Estado. (Función del sistema)

Recuperar Recompensa. (Función del sistema)

Acción ← Dime_nueva_acción(Estado_Actual, Recompensa_Anterior)

EjecutarAcción. (Función del sistema)

Cerrar Q-Table: si Aprendiendo grabar a fichero

El código de la función Dime_Nueva_Acción es la que implementa el algoritmo de aprendizaje de la tabla-Q:

```

if (am_i_learning_)
{
    // Update Q-table for the last State, Action using the reward.
    // max Q(s', a')
    for (register unsigned int i=0; i < num_actions_ ; i++)
    {
        if (max_value < actions_array[current_state_][i])
        {
            max_value = actions_array[current_state_][i];
        }
    }

    // Q (s ,a)
    double actual_val = actions_array[last_state_][last_action_];

    // -----
    // Q (s, a) = Q(s, a) + alpha (R + gamma max Q(s' , a') - Q(s, a))
    // -----
    actions_array[last_state_][last_action_] =
    actual_val + alpha * (last_action_rew + gamma * max_value - actual_val);
}

// Ok. Choose now the new action:
if(rand()*100.0 < (RAND_MAX * explore_rate_)) // Explore actions randomly
{
    action = rand() % num_actions_ ;
}
else
{
    max_value = -10000000.0;
    // Select the "best" action.
    for (register unsigned int i=0; i < num_actions_ ; i++)
    {
        if (max_value < actions_array[state_now][i])
        {
            max_value = actions_array[state_now][i];
            action = i;
        }
    }
}

last_action_ = action;

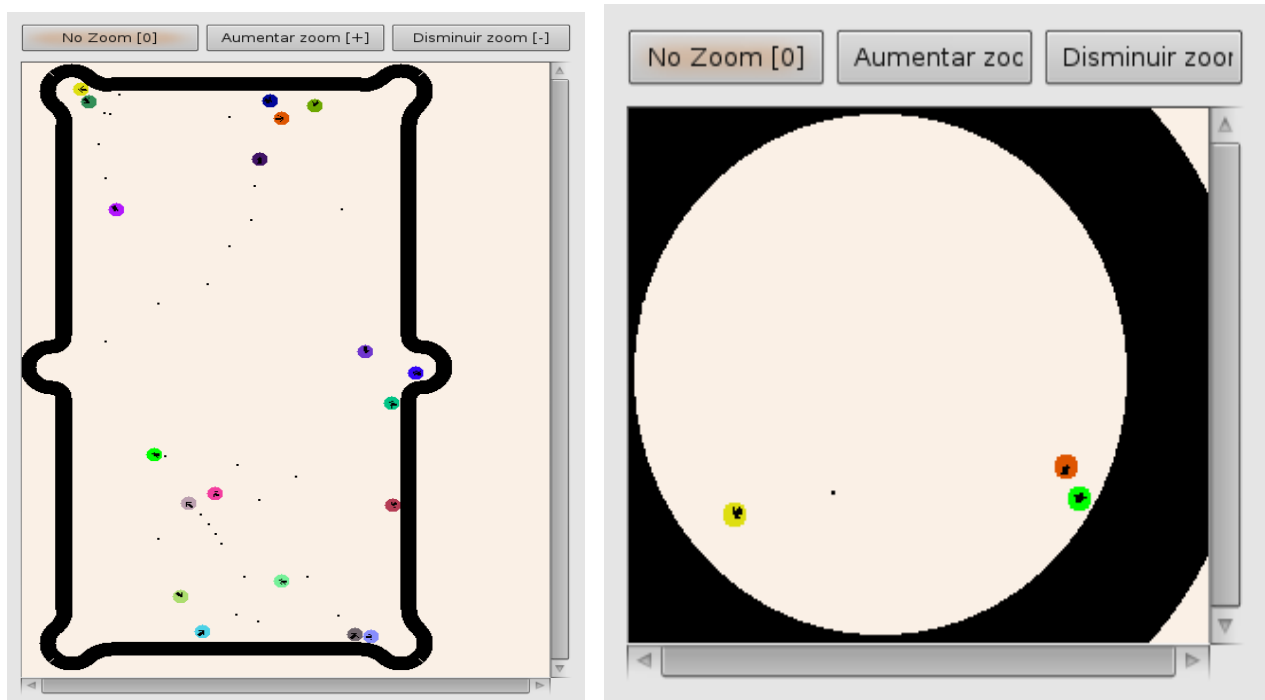
```

Resultados y Experimentos

Se han contruido los siguientes robots:

- **Rebote “Normal”**: Con aprendizaje de tabla -Q para la selección de acciones.
- **Rebote “Azar”**: Selección de acciones siempre al azar.
- **Rebote “Rambo”**: Siempre se elegía esta acción.
- **Rebote “Loco”**: Siempre se elegía la acción “*Fire_Crazy*”

Para las pruebas se han utilizado dos arenas o terrenos de juego una más complicada y otra sencilla:



Respecto al número de contrincantes se han utilizado 20 en la arenas complicada y 3 en la sencilla. Rebote siempre ha luchado contra todos los demás que eran del mismo tipo.

Para el tipo de contrincantes se ha utilizado el robot del entorno RTB Seek_and_Destroy por su versatilidad y buen resultado en torneos genéricos.

El total se han desarrollado cerca de 100 torneos de 100 juegos cada uno.

Para el aprendizaje se ha dejado al robot compitiendo durante 100 juegos antes de enfrentarlo en modo “no-aprendizaje” o utilizando sólo los valores de la tabla-Q

De todos los torneos y juegos ejecutados los iniciales se han utilizado principalmente para poner a punto el sistema y diseñar mejor los siguientes experimentos.

Las estadísticas y datos que se muestran a continuación están basados en los casos que presumiblemente existía menos complejidad para poder así medir la convergencia de los

algoritmos. Los resultados aquí mostrados son los de experimentos a 100 juegos con valores dentro de la media de todos los últimos ejecutados.

| Experimento | Robot | Posición | Puntos | Tiempo Vivo |
|--------------------------|---------------|----------|--------|-------------|
| Acción al Azar | S&D | 1 | 2,08 | 72,65 |
| | S & D | 2 | 2,05 | 72,57 |
| | Rebote | 3 | 1,30 | 50,31 |
| Acción Rambow | Rebote | 1 | 2,17 | 57,33 |
| | S & D | 2 | 1,79 | 56,26, |
| | S & D | 3 | 1,74 | 54,79 |
| Acción "Fire Crazy" | S & D | 1 | 2,14 | 72,50 |
| | S & D | 2 | 2,03 | 71,49 |
| | Rebote | 3 | 1,35 | 57.05 |
| Adaptativo | S & D | 1 | 2,04 | 67,68 |
| | S & D | 2 | 1,94 | 65,40 |
| | Rebote | 3 | 1,61 | 53,27 |
| Q-table (sin Learning) | S & D | 1 | 2,00 | 67,68 |
| | S & D | 2 | 1,90 | 65,40 |
| | Rebote | 3 | 1,40 | 59,85 |
| Adaptativo (Gamma = 0,8) | S & D | 1 | 1,97 | 62,93 |
| | S & D | 2 | 1,83 | 61,06 |
| | Rebote | 3 | 1,78 | 59,71 |

Otros resultados:

En cada una de las tablas-Q generadas el número de elementos que han visto variado su valor inicial asciende a una media de 1000 elementos sobre el total de 45927.

Conclusiones

De los datos presentados podemos decir que:

- El Robot con Q-Learning se comporta mejor que el robot que selecciona acciones al azar.
- El Robot con Q-Learning no se comporta como el mejor de los robots con comportamiento fijo pero tampoco como el peor.
- Los parámetros con mejores resultados han sido:
 - Gamma = 0,8
 - Alpha = 0,2
 - Porcentaje de Exploración = 10%

Otros autores indican mejores resultados con Gamma = 1.0, a pesar de que con este valor no se garantiza la convergencia de la estimación de la tabla-Q, pero en entornos tan cambiantes como la lucha de bots, parece adecuado primar la adaptación sobre la estabilidad. Sin embargo en este caso

- Parece que el robot que siempre está aprendiendo, y por lo tanto actualizando los valores de la tabla-Q, tiene un comportamiento mejor que aquél que sólo la usa para seleccionar la mejor acción posible.

En entornos tan cambiantes también parece razonable siempre actualizar y ejecutar las acciones de aprendizaje.

- Uso del Motor de cálculo de Q-table: Queda demostrado en el trabajo su utilidad y facilidad de uso para incorporarlo a nuevas tareas en la que encaje el aprendizaje por refuerzo. Además su diseño facilitaría la inclusión de nuevos algoritmos y parámetros.
- Tabla de Estados: Quedan en la tablas mucho valores por explorar. En el momento de escribir estas líneas no podemos concluir que sea un problema de tiempo y número de ejecuciones, aunque si parece indicar que por lo menos deban extenderse los experimentos para comprobar este punto.

Trabajo Futuro

La líneas abiertas y las nuevas líneas de investigación que se proponen son:

- Pruebas y experimentos ampliados:
 - Acciones:

Las acciones elegidas en el trabajo han sido acciones coordinadas con un punto de mayor complejidad que las básicas ofrecidas por el entorno.

Realizar el mismo trabajo pero eligiendo acciones simples tipo girar a la derecha, frenar, disparar, etc, puede arrojar conclusiones interesante sobre el desarrollo de estrategias por el propio agente.
 - Realizar baterías de pruebas cambiado los parámetros Gamma, Alpha y explore Rate, además del tipo y número de contrincantes y los tipos de arena. Con esto se conseguiría profundizar en las posibilidades del entorno RTB y del algoritmo de aprendizaje reforzado por estimación de tabla-Q.
 - Como aliciente para estos trabajos pueden presentarse Bots a los distintos torneos y concursos que se hacen sobre RTB.

- Motor Q-table:
 - Utilizar en otros entorno:
 - Monitorización de Sistemas. Poder aplicar los conocimientos de un operador humano para aprender las acciones adecuada para administrar una red o unos sistemas. Incluso poder prescindir de la asistencia humana, deduciendo las recompensas del las mejoras o no en los estados de los sistemas monitorizados.
 - Otros sistemas de Bots,

- Matriz Q:

Reutilizar la tabla-Q para extraer información sobre el entorno como:

 - Estados no-alcanzables.
 - Estados que no aportan valor o diferencia para el agente.
 - Reducción de Matrices.
 - Expresión en lenguaje natural de las mejoras en los estados.
 - Acciones sin efecto aparente.
 - Acciones con efectos perjudiciales.

Principales Referencias

- [1] **Machine Learning: C13 – Reinforcement Learning**
Tom Mitchell - 1997

- [2] **RETALIATE: Learning Winning Policies in First-Person Shooter Games**
Megan Smith, Stephen Lee-Urban, Héctor Muñoz-Avila – 2007

- [3] **Learning to be a Bot: Reinforcement Learning in Shooter Games.**
Michelle McPartland and Marcus Gallagher – 2008

- [4] **Sutton, R. S. & Barto, A. G. Reinforcement Learning:An Introduction,**
MIT Press, Cambridge, MA - 1998.

- [5] **Recognizing the Enemy: Combining RL with Strategy Selection using CBR.**
B. auslander, S. Lee-Urban, Chad Hogg and H. Muñoz - 2008

- [6] **Real Time Battle: Web page and documentation.**
<http://realtimebattle.sourceforge.net/>