Problema 4: Epic Battle Online Game

Introducción a la Ingeniería

ft. sborquez, gcorrea, mivalenz

UTFSM

5 de octubre de 2016

Departamento de Informática
Universidad Técnica Federico Santa María

Introducción

Juegos

"A game is a system in which players engage in an artificial conflict, defined by rules, that results in a quantifiable outcome." (Katie Salen and Eric Zimmerman)

"[...] all games share four defining traits: a goal, rules, a feedback system, and voluntary participation." (Jane McGonigal)

Computer game: "a game played using a computer, typically a video game."(Google)

Historia

Primer videojuego: *Nought and crosses*, también llamado OXO, desarrollado por Alexander S. Douglas en 1952. El juego es una versión computarizada del *gato*, que se ejecuta sobre la EDSAC y permitía enfrentar a un jugador humano contra la máquina.

Historia

En 1962 Steve Russell, un estudiante del MIT, creó un juego de 2 jugadores para computadora usando gráficos vectoriales: Spacewar!

Historia

1980-1990: La década de los 8 bits. Auge de consolas como NES y Atari. En 1985 nace Super Mario (Nintendo).
1990-2000: La década del 3D. Aparición de consolas como Nintendo 64 y PlayStation, las cuales usaban gráficos poligonales.
2000-Actualidad: Auge de consolas como PlayStation y Xbox. Considerable mejora de gráficas y sonidos gracias al avance de tecnologías asociadas. Masificación de juegos en línea por facilidad de acceso a internet.

Super Mario Bros (1985)

Gráficos poligonales en TLoZ: OoT

Speis Guars

Shooter online multiplayer. En un tablero de bordes periódicos[1] se disparan M jugadores. El jugador que mayor cantidad de disparos haya acertado hasta que todos mueran, gana.

Figura: sborquez©

[1] Ej: al salir por la izquierda se vuelve a entrar al tablero por la derecha

Objetivo del juego

Ya sea un simple koopa de Mario Bros o un impredecible jefe en Dark Souls, en todo videojuego nos encontraremos con algun tipo de bot. Quizás algunos parecerán tan inteligentes que encontrar un patrón para derrotarlo será complicado, pero al final del dia, estos bots solo siguen una lista de instrucciones, es decir, estan *programados*.

Objetivo del juego

Figura: Botdummy ayudantes

Es por esto que ustedes deberán crear un bot lo suficientemente inteligente como para poder derrotar a nuestro *botdummy* y desempeñarse lo mejor posible en contra de los bots de sus compañeros.

Reglas del juego

1. Cada bot consiste en una nave ubicada en una matriz NxN, el cual solo sabe su posición en el espacio.
2. En cada turno el servidor, quien tiene conocimiento de todas las naves, entregará una lista de grados de amenaza a la nave, exigiéndole a esta última una respuesta.
3. El grado de amenaza se calcula en base a qué tan cerca están el resto de las naves.

Reglas del juego: grado de amenaza, pt.1

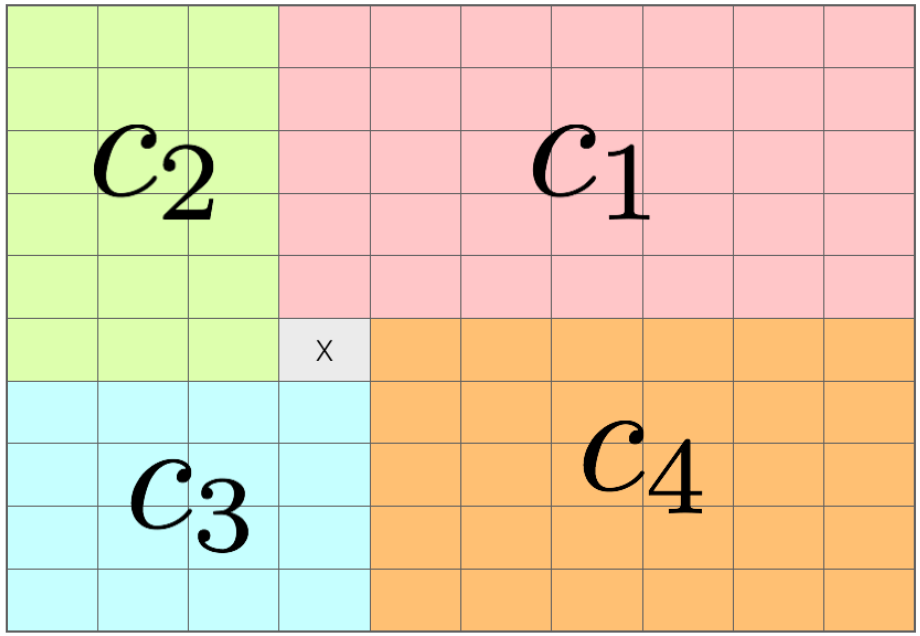
El grado de amenaza es un string de la siguiente forma: "a-g_0-g_1-...-g_n:c_1-c_2-c_3-c_4", donde cada grado de amenaza está representado por g, y g puede tomar cualquiera de los tres valores siguientes:

- `1`: 4 o 5 espacios de distancia.
- `2`: 2 o 3 espacios de distancia.
- `3`: 1 espacio de distancia.

La distancia solo puede ser vertical o horizontal.

Reglas del juego: grado de amenaza, pt.2

Para el mismo string que indica el grado de amenaza, "a-g_0-g_1-...-g_n:c_1-c_2-c_3-c_4" los c_i correspondientes a la cantidad de enemigos por cuadrante respecto al jugador. Por ejemplo, en la figura el símbolo X corresponde a la ubicación de su robot y c_1 indica la cantidad de robots en el primer cuadrante. Recuerde que esta información es relativa a su posición.



Reglas del juego: acciones

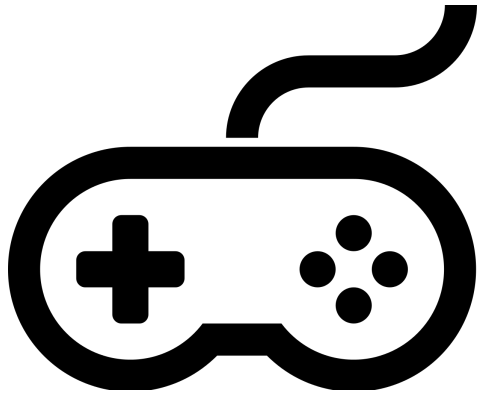
La nave deberá entregar un disparo $D = [-5, 5] \cap \mathbb{Z}$ y un movimiento $M = [-3, 3] \cap \mathbb{Z}$. Este debe ser entregado en el siguiente formato string: D_V, D_H / M_V, M_H. Donde:

$$D_V \text{ y } D_H \in D, D_V D_H = 0 \text{ y } D_V + D_H \neq 0$$

$$M_V \text{ y } M_H \in M, M_V M_H = 0 \text{ y } M_V + M_H \neq 0$$

Los sub-índices V y H indican si el movimiento es vertical o horizontal respectivamente.

Ejemplo: si el jugador quiere disparar 4 espacios a la izquierda, y moverse uno arriba, debería entregar al servidor el siguiente string: `0,-4/1,0`.



Más ejemplos

Si el jugador quiere disparar 2 espacios a la izquierda, y moverse uno abajo, debería entregar al servidor el siguiente string: `0,-2/-1,0`.

Si el jugador quiere disparar 5 espacios hacia arriba, y moverse 2 a la derecha, debería entregar al servidor el siguiente string: `5,0/0,2`.

Si el jugador quiere disparar 4 espacios abajo, y moverse uno arriba, debería entregar al servidor el siguiente string: `-4,0/1,0`.

Palta: $D_V\, D_H = 0$ y $D_V + D_H \neq 0$ dice que uno de los dos es 0 y que uno de los dos es distinto de 0. Lo mismo es válido para M_V y M_H.

Reglas del juego: vidas

* Cada nave tiene 3 vidas. Ser impactada por un disparo le resta una vida. Si la nave, transcurridos 3 turnos, no acierta un disparo, perderá una vida. Finalmente, si una nave X choca con una nave Y, la nave X muere y la nave Y pierte una vida.
* Toda nave que muere, corta conexion con el servidor.
* El juego termina cuando no quedan jugadores vivos.

El código: repositorio

El código deberán clonarlo del siguiente repositorio en `Github`[a], el cual está distribuido en 3 carpetas:

1. Servidor:
   `control.py`, `servidor.py`
2. Cliente:
   `botdummy.py`, `botplayer.py`
   `cliente_alumnos.py`,
   `cliente_dummy.py`
3. Visualización:
   `main.py`
   `/logs`

[a] Hacer click sobre `Github`!

El código: por modificar

Ustedes deberán trabajar principalmente con:

1. `botplayer.py`: Aquí irán todos los algoritmos que usarán para crear su bot (pueden ayudarse con `botdummy.py`).
2. `cliente_alumnos.py`: Contiene todo lo necesario para poder conectarse con el servidor.

El código: Cómo usar

Para poder probar su bot, deberán realizar lo siguiente:

1. Montar el servidor:
   `python servidor.py`
   Entregará la ip y el puerto que estará esperando las conexiones.
2. Conectar un cliente:
   `python cliente_alumnos.py`
   Le pedirá la ip y el puerto del servidor, y además un nombre para identificarlos.

Al terminar la partida, el servidor generará un archivo .log en el directorio /logs de Visualizacion.
Para poder reproducir el resultado: `python main.py`
Seleccionan la partida.

Sockets

Para resolver las conexiones entre clientes y servidor, usaremos la librería `socket` de Python. Para poder llevar a cabo este problema, deberán considerar lo siguiente:

1. El cliente, para conectarse al servidor, necesita la dirección IP y el puerto en el cual se escuchará a nuestro servidor. Es por esto que, antes que todo, deben ejecutar `servidor.py` y recibir la información de la IP y el puerto.
2. El cliente y servidor solo pueden mandarse `strings` entre sí, por lo cual toda la información enviada y recibida por el cliente(o servidor) debe estar en dicho formato.

Grand Finale

Una vez que todos sus bots estén entrenados para la batalla real, se realizará una partida masiva (todos contra todos), en la cual todos sus bots pelearán hasta la muerte para adjudicarse el título de el “Mejor Bot Speis Guars 2016”.