UADY
UNIVERSIDAD
AUTÓNOMA
DE YUCATÁN

Proyecto:
Fighting game
Inteligencia artificial

Irving Salomon Azcorra May
Raúl Alejandro Peña Cámara
Manuel Ruiz Ayala

Facultad de Matemáticas
Universidad Autónoma de Yucatán

15 de diciembre de 2015

Índice

Resumen

Basándonos de toda la información encontrada en la pagina de *fighting game AIcompetition*[1], usando los códigos y plantillas para el juego desarrollaremos una IA que pueda competir contra otro jugador o alguna otra IA de las que se encuentran en la referencia [1]. Todo esto mediante la programación en Java y utilizando lo visto en la clase de Inteligencia Artificial y algoritmos de búsqueda como el "Vecino más cercano" el cual fue el que se utilizó en este proyecto.

[1]Ver referencia [1]

1. Introducción

El objetivo del proyecto es lograr crear un agente que sea capaz de competir contra una persona u otra IA en el juego de peleas *FightingICE*.
Como fue mencionado solo crearemos nuestro agente capaz de competir y no programaremos ningún aspecto del juego. Para hacer esto únicamente descargaremos todo el código y el software necesario para el desarrollo de la IA[2], donde únicamente tendremos que agregar el modulo respectivo para la IA.
Todo este proyecto es programado en *JAVA* ya que en este lenguaje están todos los códigos del juego y ademas de que la mayoría de información acerca de la programación de las IA's y del juego se pueden encontrar para este lenguaje de programación.
El juego *FightingICE* es el típico juego de peleas PvP (player vs Player) que para este caso haremos que sea Versus AI-Game, es decir que alguna persona usara a alguno de los personajes y el otro personaje sera controlado por el agente en desarrollo.
Algo muy importante es que para este juego se realizan competencias y torneos, de aquí es que existe mucha información y documentos acerca de este juego y mas aun información acerca del desarrollo de diferentes IA's.

2. Marco Teórico

2.1. Fighting Game AI Competition

Antes de comenzar a hablar acerca de nuestro agente, como fue mencionado existe una competencia de este juego y usaremos las reglas de este torneo para poder desarrollar nuestro agente.

2.1.1. Los frames

Como cualquier otro videojuego ocurre frame por frame, en un segundo tenemos 60 frames es decir que el videojuego corre a *60 fps*. Una ronda del juego dura un minuto es decir que por cada ronda tenemos 3 600 frames[3].

2.1.2. El puntaje

Al ser una competencia es obvio que es necesario una forma de elegir un ganador. Primero a cada jugador se le asigna una cantidad de vida (HP), con

[2]Ver referencia [1]
[3]Mas adelante detallaremos la importancia de detallar el numero de frames.

cada golpe que se acierte se restara puntos de vida al jugador que reciba el golpe. Con estos puntos generamos la siguiente ecuación:

$$puntaje = \frac{OpHP}{SelfHP + OpHP} \cdot 1000$$

Donde:

- **OpHP**.- Son los puntos de vida del oponente
- **SelfHP**.- Son los puntos de vida que tiene el jugador.

Este puntaje lo usamos únicamente para saber como va el desempeño de nuestro agente y como debemos de mejorarlo en cada prueba que hagamos hasta obtener una IA que tenga un buen desempeño.

2.1.3. La memoria

En la competencia se ponen ciertos limites para la memoria de cada IA usada, el limite a usar es de 512 MB de memoria y un uso de *lectura/escritura* que no supere los 10 MB. Para nuestro caso no pondremos limites en la memoria ya que solo desarrollaremos una IA y no pretendemos participar en dicha competencia, aunque para la optimización intentaremos reducir el uso de la memoria.

Importante Todo este apartado únicamente hacemos mención de las reglas mas importantes que se señalan en el reglamento de la competencia y por ello hacemos las presentamos en el reporte, aun cuando no hacemos caso de la puntuación ya que lo que queremos es desarrollar un agente capaz de competir de manera adecuada aunque si esperamos obtener un buen agente que sea capaz de competir contra alguno de los que están en la competencia.

2.2. Instalación y ejecución

Antes de comenzar es necesario instalar los componentes necesarios para compilar y ejecutar el juego. Para ello necesitamos instalar:

- Una version actualizada de **JDK** la cual contiene las librerías necesarias para poder ejecutar el juego.
- El entorno de desarrollo **Eclipse** en el cual trabajaremos y desarrollaremos nuestro agente.

Una vez que se halla instalado el software anterior es necesario descargar el juego *FightingICE* y la plataforma **AITookit** la cual sirve para poder cargar a través de eclipse nuestra IA.
La guía de instalación y carga de la IA paso a paso se encuentra en la referencia [2]. No se presenta esto ya que nuestro tema central es la creación y desarrollo del agente y no la instalación del juego.

3. Diseño de Investigación

3.1. Conceptos Básicos

Antes de comenzar con los algoritmos es necesario analizar conceptos básicos acerca del juego comenzando por los frames. En cada frame se puede hacer una acción, es obvio que en cada frame sucede demasiado rápido y realmente nosotros no podemos percatarnos de que ocurre en cada frame, pero nuestro agente trabaja con cada frame y es importante saber que ocurre en cada frame. Para comenzar tenemos que en un frame se puede ejecutar una acción *básica* (ir hacia adelante, ir hacia atrás, saltar, atacar, defenderse) y solo nos centraremos a estas acciones ya que podemos hacer acciones un tanto mas complejas como lanzar habilidades pero estas se hacen haciendo una çadena"de acciones.

- **Moverse** Para avanzar (ir hacia atrás o hacia adelante) solo es necesario un frame.
- **Defenderse** Cuando el oponente lanza un ataque moverse hacia atrás hará que el personaje adopte la posición defensiva durante los frames que dure el ataque
- **Atacar** Para lanzar un ataque son necesarios 18 frames.

Como se puede ver no hay mucho que analizar en cuanto a los movimientos del personaje o el defenderse ya que únicamente se necesita dar las indicaciones y nuestro agente en teoría debería lograr analizar esto de manera sencilla. El problema se complica al atacar ya que no es solo lanzar un ataque sino que hay que analizar que sucede durante el ataque del personaje y que es lo que realiza el oponente y del mismo modo cuando el oponente ataca hay que analizar que sucede durante su ataque que acción deberíamos tomar.

En un ataque tenemos 3 fases las cuales son cruciales para la toma de decisiones de nuestro agente:

- **Startup** En el momento que se indica dar la acción de atacar comienza esta fase. Ocurre desde el *frame 1* hasta el *frame 6*.
- **Active** Es el momento en que ocurre el hit (se acierta o falla el ataque) y se calcula el daño realizado. Dura únicamente 2 frames.
- **Recovery** Son los últimos 9 frames en los cuales la animación del personaje regresa a su estado habitual.

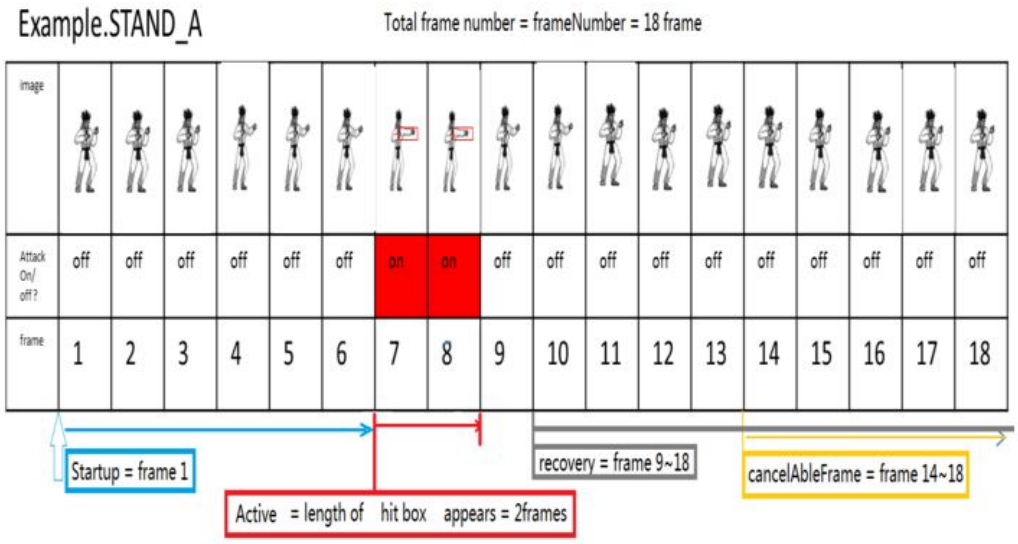

Figura 1: fases de un ataque y sus frames necesarios

En las primeras 2 fases no es posible cancelar la acción, es decir que cuando nuestro agente ejecute un ataque estos 8 frames pueden ser usados únicamente para captar la información del oponente, pero durante el recovery es posible cancelar parte de la animación para ejecutar una acción anticipada.

3.2. El algoritmo

Para poder hacer nuestro agente usaremos el manual de la *referencia [3]* el cual nos indica las principales características del agente así como algunos códigos usando el *algoritmo del vecino mas cercano*.

Ademas de este manual como tuvimos algunos problemas durante la instalación del software necesario para poder desarrollar nuestro agente recurrimos a buscar otro agente el cual nos sirvió de base para identificar ciertos comando y clases que son necesarios para que nuestro funcione correctamente. Posteriormente usamos el manual que mencionamos anteriormente para comenzar a programar nuestro agente.

3.2.1. Recolección de datos

Nuestro agente guarda la información de la posición del oponente con respecto a nosotros, por ello seleccionamos el algoritmo del *vecino mas cercano* para recolectar los datos. En el manual de la IEEE que usamos nos proporciona el siguiente algoritmo:

Algorithm 1 collectData($self$, $opponent$, $data$)

```
if opponent.act is an attack action then
  x ⇐ opponent.x − self.x
  if self is not facing to the right then
    x ⇐ −x
  end if
  y ⇐ self.y − opponent.y
  position ⇐ checkPosition(self, opponent)
  if position is ground − ground then
    data.gg.add(opponent.act, x, y)
  else if position is ground − air then
    data.ga.add(opponent.act, x, y)
  else if position is air − ground then
    data.ag.add(opponent.act, x, y)
  else if position is air − air then
    data.aa.add(opponent.act, x, y)
  end if
end if
```

Figura 2: Código para una implementación del algoritmo del vecino mas cercano

Aunque este algoritmo fue la base para nuestro agente lo único que realmente usamos fue la lógica para poder realizar las acciones. También usamos un el agente *fightingwarrior* para ver algunos comandos que eran necesarios para poder realizar ataques.

Las variables `self` y `opponent` son la posición que nuestro agente y del oponente respectivamente. Los datos de que acción realizan (cada frame) están en `data`. Para el caso de nuestro agente usamos algunas estructuras que ya están en el fighting estas estructuras están en el siguiente código:

```
import structs.CharacterData;
import structs.FrameData;
import structs.GameData;
import structs.Key;
```

Para guardar información del entorno y de los personajes usaremos `CharacterData` para guardar información del oponente(opp) y de nuestro personaje(my). `GameData` es la información de la salud y energía de los personajes que aunque no le da mucha importancia nuestro agente es necesario agregarlo. `FrameData` es el frame que esta ocurriendo y junto con `GameData` podemos obtener datos del espacio y con esta usaremos 2 variables auxiliares `distance`

y `xDifference` para poder tener información de la distancia entre los personajes.Por ultimo tenemos `key` es la tecla que se presiona y por ende es la acción que se va a realizar.

3.3. Objetivo del agente

El principal objetivo de nuestro agente es la recolección de datos del oponente y a partir de su posición decir que acción va a realizar. Estas acciones a realizar son las que creímos mas convenientes.

Buscamos obtener un agente capaz de realizar una pelea que aunque sus resultados (el puntaje como el de la competencia) no sean muy buenos logre hacer acciones pertinentes y bastante buenas.

3.3.1. El siguiente paso

Este agente como se menciono usa el algoritmo del vecino mas cercano como primer instancia para lo toma de decisiones. Pero este algoritmo es meramente de búsqueda y a partir de los arrojados por este algoritmo podemos usarlos para tomar decisiones. Nuestro siguiente paso para mejorar la IA (por el tiempo no se podrá implementar) es unir esto a una red bayesiana. Esta red bayesiana tomaría bastante tiempo diseñarla para que funcione de una manera mas optima.

4. Resultados

A pesar de que no presentamos en este reporte información detallada del avance de nuestro agente ya que lo hicimos a base ensayo y error describir este proceso seria muy largo y abrimos que detallar demasiadas cosas. Por ello solo presentamos información que es de utilidad para la compresión del código.

Nuestro agente final este presentado en el repositorio de *GIT* el cual únicamente tiene el archivo de nuestro agente por lo que hay que instalar adecuadamente el software[4] y cargar nuestro agente para poder ver su ejecución.

Por ultimo es necesario decir que todas las decisiones que toma el agente están que en la función `processing()`. Ademas por el tamaño de todo el proyecto no se pudo subir todos los archivos del fighting y por ello solo subimos el archivo del agente en nuestro repositorio de *GIT*.

[4]Ver apartado 2.2

5. Conclusiones

Podemos decir que este proyecto fue muy bueno, ya que tuvimos que aprender a utilizar Java, un lenguaje de programación que no conocíamos y de esta manera poder implementar la IA de una manera correcta. De igual forma tuvimos que aprender acerca del código del juego que utilizamos y de la IA que nos basamos para poder hacer una nosotros, tuvimos que entender las funciones y ver como mejorarlas, en este caso nos fue muy utilidad un manual acerca de las reglas de FightingICE para poder implementar el algoritmo del vecino mas cercano que es el que utilizamos para implementar nuestra IA, para que tenga un optimo funcionamiento.

En conclusión este proyecto fue difícil al principio, ya que no sabíamos que proyecto elegir pero al elegir esta no conocíamos mucho acerca del tema, pero pudimos resolverlo de manera efectiva gracias a la ayuda del profesor y de nuestras investigaciones.

Por último es necesario decir que el usar la IA *DragonWarrior* no sirvió mucho para tener una base de nuestro agente. Aunque realmente no sabemos que algoritmo usa exactamente ya que no nos dimos el trabajo de analizarlo por completo ya que teníamos el manual de la IEEE como una especie de tutorial para nuestra IA.
Haciendo una especie de comparación lo que nos percatamos del *fightngwarior* es que era una tanto ofensivo a comparación de nuestra IA que depende del posicionamiento del oponente para realizar una acción.

Referencias

[1] http://www.ice.ci.ritsumei.ac.jp/

[2] Manual de instalación

[3] Manual FightingICE IEEE 2014