

A ti te va a caer el Axl

Daniel Barreto - #04-36723

daniel@ac.labf.usb.ve

13 de marzo de 2009

Índice general

Introducción	2
1. Descripción del Juego	4
1.1. Metáfora	4
1.2. ¿Cómo jugar?	4
2. El juego y sus objetos	6
2.1. Personajes	6
2.1.1. Slash	7
2.1.2. Los Reggaetoneros	7
2.2. Escenario o tarima	7
3. Comportamientos y movimientos desarrollados	8
3.1. Los comportamientos no tan inteligentes de Slash	8
3.2. Los reggaetoneros	8
3.3. Comportamiento general, unión de varios comportamientos . .	9
4. Dándole inteligencia a los reggaetoneros	11
4.1. Encontrar caminos (A^*)	11
4.1.1. Malla de Polígonos: División del mapa	11
4.1.2. Enumeración de sectores, nodos y caminos	13
4.1.3. Encontrando el camino inteligentemente	13
4.2. Toma de decisiones	14
4.2.1. Estados de un reggaetonero	14
4.2.2. Árbol de decisiones para el movimiento	15
4.2.3. Máquina de estados para la pelea	16

5. Detalles de Implementación	17
5.1. Herramientas Utilizadas y Requerimientos	17
5.2. Modelos Geométricos Implementados	17
5.3. Blending de comportamientos	18
5.3.1. Prioridades de los comportamientos	19
5.4. Manejo de fuerzas y movimiento sobre los reggaetoneros	19
6. Problemas encontrados	21
6.1. Velocidades después de una colisión	21
6.2. Problemas con la cohesión de los “boids”	21
6.3. Muchas balas = Juego lento	22
6.4. Optimización del algoritmo de búsqueda de caminos mínimos .	22
6.5. Choques contra obstaculos	23

Introducción

“*A ti te va a caer el Axl*”¹² es un juego desarrollado en 2D+1/2 que emula juegos de pelea entre un personaje principal y varios controlados artificialmente, sobre un “stage” o campo de tamaño limitado.

El objetivo principal del juego es vencer a todos los contrincantes que sean encontrados, para esto es necesario atacarlos y herirlos de forma que sean más susceptibles a ser lanzados fuera de la plataforma sobre la que pelean.

Para esta entrega se implementaron diferentes *Steering Behaviors* como persecución y evasión (*Seek*, *Flee*, *Pursue* y *Evade*), evasión de obstáculos (*Obstacle Avoidance*), comportamientos grupales o de manada (*Flocking*) que incluyen Separación, Cohesión y Alineación.

El juego está construido en 3D usando OpenGL como herramienta para realizar el “rendering” de todos los objetos que este incluye (personajes, obstáculos, plataforma, etc).

¹Axl Rose, vocalista principal de la banda de Heavy Metal Guns n’ Roses

²Nombre provisional

Capítulo 1

Descripción del Juego

1.1. Metáfora

La metáfora del juego se trata de la difícil tarea que tiene **Slash** (personaje principal), el guitarrista principal de la banda de heavy metal *Guns n' Roses*, de acabar con los máximos exponentes del *Reggaeton*, a los cuales debe atacar con la música producida por su guitarra.

La trama se desarrolla en una tarima de conciertos de *Guns n' Roses*, en donde se encuentran obstáculos como amplificadores, una batería y otros instrumentos. **Slash** tendrá que sacar a los reggaetoneros invasores lanzándolos fuera de la tarima, pero este objetivo no es tan fácil como parece, pues cada enemigo viene lleno de energía y su capacidad de quedarse dentro de la tarima está ligado a su nivel de energía actual, por esto, **Slash** tendrá que lastimarlos con su guitarra antes de intentar arrojarlos hacia afuera, de forma parecida al juego de Nintendo © *Super Smash Bros*.

1.2. ¿Cómo jugar?

Los controles y sus acciones se encuentran representadas en las siguientes tablas:

Cuadro 1.1: Controles de **Slash**

Tecla	Acción
a	Acelerar hacia la izquierda
d	Acelerar hacia la derecha
w	Acelerar hacia el fondo del escenario
s	Acelerar hacia el frente del escenario

Nota: Varios movimientos pueden ser ejecutados simultáneamente para acelerar hacia varias direcciones al mismo tiempo.

Cuadro 1.2: Acciones de **Slash**

Tecla	Acción
Espacio	Salto
Enter	Disparar
]	Aumentar ángulo de disparo
[Disminuir ángulo de disparo
'	Aumentar fuerza de disparo
\	Disminuir fuerza de disparo

Capítulo 2

El juego y sus objetos

El juego consta de varios objetos que se describen a continuación

2.1. Personajes

Los personajes que participan en el juego pueden ser divididos en dos grupos: El personaje principal (**Slash**) y sus enemigos los reggaetoneros. Ambos cuentan con restricciones similares en cuanto a su máxima velocidad y aceleración, tanto lineal como angular, que puedan tener. Además cada personaje tiene una cierta cantidad de masa que lo hace más o menos susceptible a ser desplazado por golpes de otros jugadores.

Todos los personajes cuentan con un nivel de vida o energía, el cual es multiplicado por su masa cada vez que se detecta una colisión de algún vector de fuerza que se le este siendo aplicado. El máximo nivel de energía para todos los personajes es de 100 puntos, y los golpes o ataques que reciban de otros personajes irán disminuyendo poco a poco dicho nivel.

La fórmula para calcular la masa de un personaje cuando reacciona frente a una colisión realizada por algún arma o bala es la siguiente:

$$M_c = M_p x E / 100$$

Donde M_c es la masa referente a una colisión con un arma o proyectil lanzado por otro personaje, M_p es la verdadera masa del personaje y E es

su nivel de energía.

Para otro tipo de colisiones como los que vendrían dados por el choque con algún otro personaje, se usa la verdadera masa del personaje M_p .

2.1.1. Slash

El personaje principal del juego sólo posee un arma: su guitarra, con la cual puede golpear a los reggaetoneros como si fuera una espada, pero también puede disparar “balas” a sus enemigos más lejanos. Actualmente no existen restricciones a cerca del número de balas con las que puede contar **Slash**.

2.1.2. Los Reggaetoneros

En el estado en el que se encuentra el juego para esta entrega, los reggaetoneros no cuentan con ningún arma para defenderse. Su única función es demostrar comportamientos inteligentes desarrollados artificialmente. Estos comportamientos serán detallados más adelante.

2.2. Escenario o tarima

El escenario del juego es un espacio rectangular de tamaño fijo, en el cual se encuentran objetos que funcionarán como obstáculos a evitar para los personajes llevados por la inteligencia artificial.

Capítulo 3

Comportamientos y movimientos desarrollados

3.1. Los comportamientos no tan inteligentes de Slash

El comportamiento de **Slash** viene dado por la interacción del usuario con el juego, pero posee la posibilidad de realizar las acciones “Saltar” y “Disparar”.

Al saltar, se le da un valor predeterminado a la componente Y de su velocidad. En el momento del salto, la velocidad en el eje Y se ve afectada por la acción de la gravedad que actúa siendo una aceleración constante y negativa a lo largo de dicho eje.

La acción de disparar es dejada a manos del jugador. El usuario podrá elegir una fuerza y un ángulo para realizar los disparos mediante el uso del teclado. La trayectoria que siguen las balas variarán dependiendo de esas dos variables.

3.2. Los reggaetoneros

Las acciones que pueden desempeñar los reggaetoneros son las siguientes:

- **Vagar** (Wander): Los reggaetoneros pueden vagar por la tarima cada

vez que se encuentren sin ningún otro comportamiento con más importancia que tengan que realizar. Para simular este comportamiento se toma eventualmente una nueva dirección a seguir y se devuelve una fuerza o “steering” que lo acelera hacia dicha dirección.

- **Perseguir y Evadir** (Pursue and Evade): Con estos comportamientos, los reggaetoneros persiguen (o evaden) un objetivo, estimando la posición del objetivo 1 segundo mas adelante del tiempo actual, asumiendo que el objetivo permanece con la misma velocidad durante dicho tiempo.
- **Evasión de obstaculos** (Obstacle Avoidance): Los reggaetoneros tratarán de evadir, siempre que se pueda, los obstaculos que se les presenten en el camino. Entre los obstaculos a evitar, se fijarán también en los bordes del escenario, tomándolos como paredes imaginarias que tendrán que esquivar.

Actualmente los reggaetoneros utilizan solo un rayo de visión dirigido hacia donde este dirigida su orientación, dicho rayo de visión se modela como un segmento de linea de tamaño 15.

- **Comportamientos en manada** (Flocking and Swarming): Los reggaetoneros se comportan como un enjambre o “swarm”. Para poder simular este movimiento en conjunto son necesarios los siguientes comportamientos:
 - **Alineación** (Alignment): Se busca mantener una dirección de desplazamiento común para todos los integrantes del enjambre.
 - **Cohesión** (Cohesion): Los miembros del enjambre tratan de mantenerse juntos, tratando de acercarse a un centro de masa del enjambre en general.
 - **Separación** (Separation): Los miembros del enjambre tratan de mantener cierta separación entre ellos.

3.3. Comportamiento general, unión de varios comportamientos

Para lograr un comportamiento más natural en los reggaetoneros se fusionaron los comportamientos anteriormente detallados, utilizando el sistema

de “Blending” con arbitraje, en donde se agruparon los comportamientos y se establecieron prioridades entre dichos grupos.

De esta forma, cada reggaetonero se encarga de aplicar la fuerza dada por el grupo de comportamientos con mayor prioridad y que haya devuelto alguna fuerza superior a un umbral previamente establecido, que tenga entre sus comportamientos asociados.

Dentro de cada grupo de comportamientos se ejecutan los comportamientos pertenecientes a dicho grupo y se suman entre ellos, tomando en cuenta que distintos comportamientos en particular tienen un mayor peso en la suma resultante.

La implementación del “Blending” realizado será detallada posteriormente.

Capítulo 4

Dándole inteligencia a los reggaetoneros

4.1. Encontrar caminos (A^*)

Un problema importante en el juego es dotar a los reggaetoneros de la suficiente inteligencia para poder navegar el mapa completo en búsqueda de caminos. Esta importancia se puede ver en la necesidad de poder encontrar a Slash cuando se esconda detrás de algún obstáculo del juego. En la Figura 4.1 podemos ver una visión superior de la disposición de los obstáculos en el *Stage*.

Para solucionar eficazmente este problema se tomaron y realizaron una serie de decisiones explicadas a continuación.

4.1.1. Malla de Polígonos: División del mapa

El primer paso esencial para proveer navegabilidad del *Stage* a los reggaetoneros fué dividir el mismo en **sectores triángulares** (Figura 4.2) para que la inteligencia artificial pudiese distinguir en que parte del escenario se encuentra.

Esta división se realizó a mano, sin seguir ningún algoritmo específico de división, buscando colocar los polígonos en la mejor disposición para que la inteligencia artificial pueda conseguir los caminos evitando al mismo tiempo los obstáculos.



Figura 4.1: Nuevos obstaculos en el *Stage*

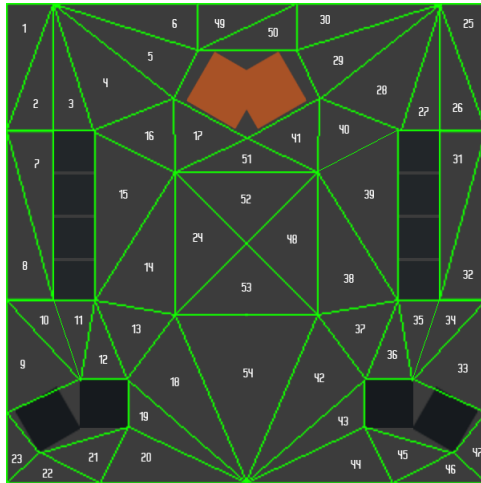


Figura 4.2: División poligonal

4.1.2. Enumeración de sectores, nodos y caminos

Para conseguir una verdadera noción por parte de los reggaetoneros sobre en que lugar del *stage* se encuentran, se enumeraron todos los sectores y se creó un grafo en donde los nodos son los **centroides**¹ de los triángulos (sectores) y los caminos entre dichos nodos fueron definidos arbitrariamente buscando que cada camino evite tener alguna intersección con el área ocupada por algún obstáculo.

El grafo obtenido se muestra a continuación:

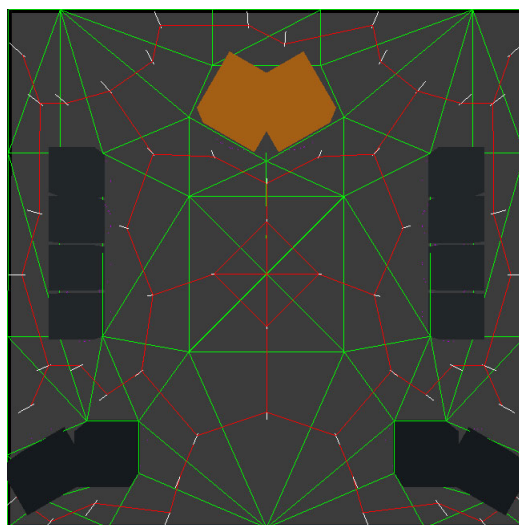


Figura 4.3: Nodos y caminos en el *Stage*

Las líneas rojas representan los caminos.

4.1.3. Encontrando el camino inteligentemente

Una vez obtenido el grafo anterior, el problema presentado fué hacer que los reggaetoneros navegaran con cierta inteligencia los nodos y los caminos presentados y tomaran siempre el camino con menor distancia para llegar

¹El centroide de un polígono es el centro geométrico de el mismo, visto también como el punto promedio de todos los puntos del polígono

desde un punto dado a otro.

Para lograr esta tarea se utilizó el algoritmo A^* como base de búsqueda de caminos del juego dado que, por ser un algoritmo de búsqueda que toma en cuenta una información extra (*heurística*), consigue de forma bastante rápida la solución óptima al problema.

La heurística utilizada en el algoritmo A^* implementado fue tomar la *distancia euclidiana* entre los nodos inicio y final entre los cuales se quiere conseguir el camino óptimo (con menor distancia recorrida).

4.2. Toma de decisiones

Cada reggaetoneros fue dotado de una implementación de toma de decisiones que regula su comportamiento a través del juego dadas las distintas variables con las que se encuentre externas a él mismo.

4.2.1. Estados de un reggaetonero

Los reggaetoneros tienen dos conjuntos de acciones que realizar que se muestran a continuación:

Conjunto	Acciones
Movimiento	Wandering, Pursuing, Evading
Pelea	Hold, Hitting, Shooting

Ambos conjuntos son independientes entre sí, es decir, un reggaetonero puede estar disparando mientras persigue, escapa o vaga por el mapa, de la misma forma que puede golpear o simplemente mantenerse sin estado de pelea.

Para implementar la toma de decisiones en cada uno de los conjuntos se tomaron aproximaciones distintas: para el movimiento se utiliza árbol de decisiones sencillo y para la pelea se implementó una máquina de estados con ciertas transiciones.

4.2.2. Árbol de decisiones para el movimiento

El árbol de decisiones implementado para el movimiento de los reggaetoneros es de la siguiente forma:

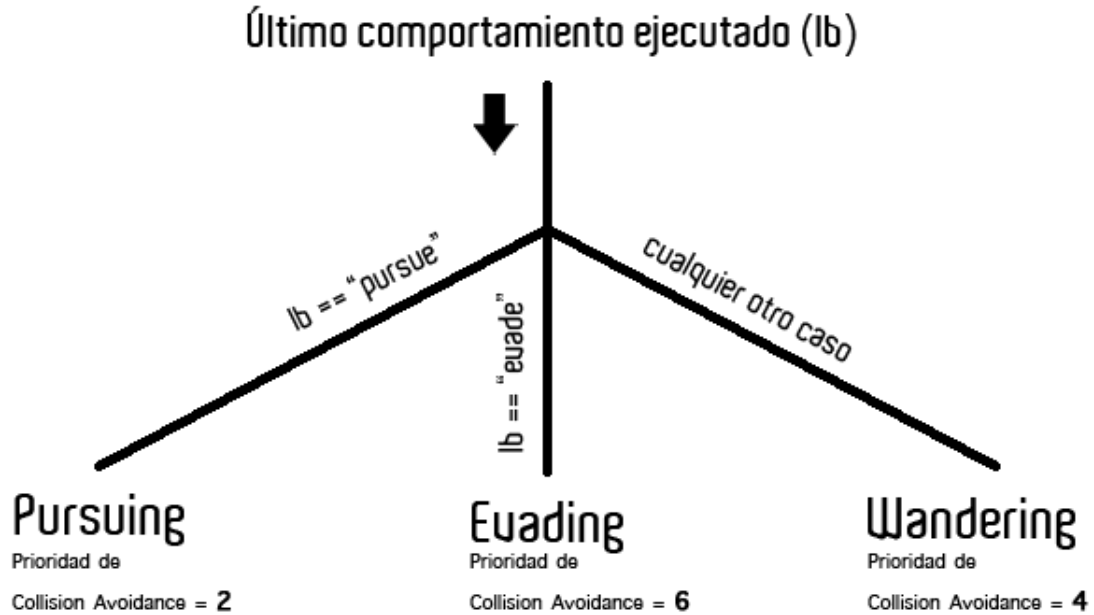


Figura 4.4: Árbol de decisiones para el movimiento

Básicamente el árbol de movimiento afecta la capacidad de esquivar obstáculos del reggaetonero. Cuando el reggaetonero está persiguiendo a **Slash** está siguiendo los nodos de navegación presentados anteriormente, por lo tanto la probabilidad de que choque contra un obstáculo o pegue contra alguna de las paredes del stage es pequeña, por esto, se le baja la prioridad al comportamiento de evasión de obstáculos para que el *Blending* de comportamientos lo tome menos en cuenta.

Por el contrario, cuando el reggaetonero está huyendo, es más probable que en su desesperación se tope con obstáculos y paredes que tenga que evitar, por esto, se le da la máxima prioridad al comportamiento de evasión de paredes y obstáculos.

Por último, cuando el reggaetonero está haciendo cualquier otra cosa que no sea perseguir o evadir, le da una alta prioridad a la evasión de paredes, pero no la máxima prioridad.

Para más información sobre las prioridades de los comportamientos en la inteligencia artificial del juego, ver la Tabla 5.3.1.

4.2.3. Máquina de estados para la pelea

La máquina de estados correspondiente a las acciones de la pelea es la siguiente:

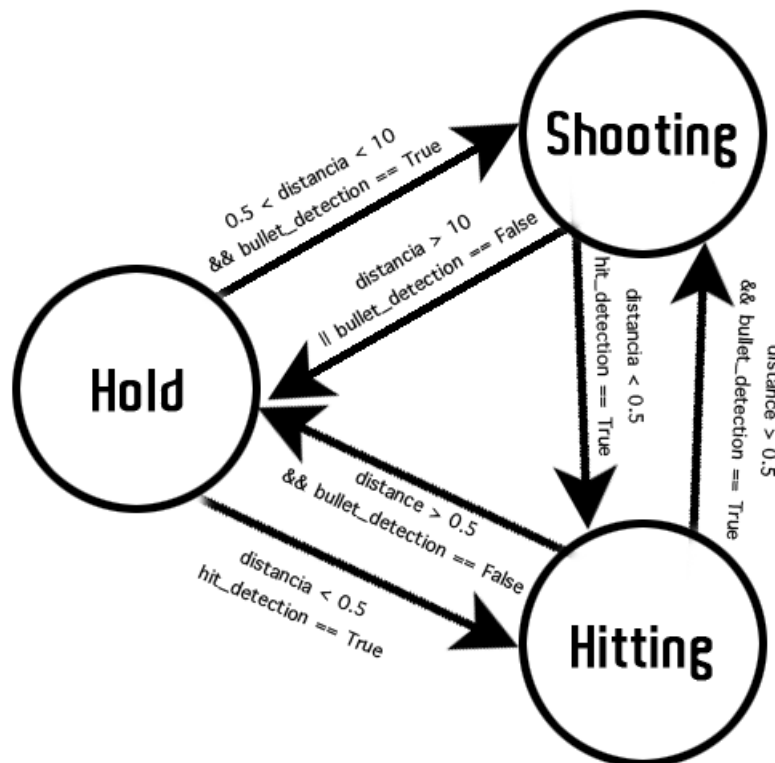


Figura 4.5: Máquina de estados

Capítulo 5

Detalles de Implementación

5.1. Herramientas Utilizadas y Requerimientos

El juego está siendo desarrollado con *Python* como lenguaje de programación y *OpenGL* como librería gráfica, se está usando *Debian Linux* como ambiente de desarrollo y se corre en una computadora con 512Mb de Ram, tarjeta gráfica ATI Radeon Xpress 200M de 128Mb y procesador Intel Celeron de 1.46GHz.

Se están usando varias librerías auxiliares de *Python*, entre las cuales vale la pena mencionar:

1. **Pygame:** Conjunto de módulos de *Python* diseñados para escribir juegos.
2. **Psyco:** Módulo de *Python* para agilizar y acelerar la ejecución de código escrito en este lenguaje.

5.2. Modelos Geométricos Implementados

Para facilitar el trabajo en el campo geométrico Euclideano, se realizaron varios tipos abstractos de datos que modelan estructuras importantes como:

- **Vector3:** Tipo abstracto de datos que modela un vector en 3 dimensiones. Este TAD esta implementado para funcionar como un tipo de

datos nativo de *Python*, en donde ha sido implementada la suma y multiplicación por escalares y vectores. A parte de esto, cuenta con otras operaciones importantes como producto cruz, producto punto, longitud, orientación, etc.

Este tipo abstracto se realizó basado en el trabajo de *Will McGugan* (<http://www.willmcgugan.com>), pero se incluyeron algunas modificaciones y correcciones necesarias.

- **Rect:** Tipo abstracto de datos que modela un Rectángulo en 2 dimensiones. Es utilizado para mantener el área ocupada por los personajes del juego, incluye operaciones como: unión, intersección, detección de colisiones entre rectángulos y entre puntos con rectángulos, clipping, entre otros.

Este tipo abstracto de datos esta basado en el Modelo Rect que viene incluido en *Pygame*, pero fue reescrito desde cero para que pudiese trabajar con coordenadas flotantes pues el modelo de *Pygame* sólo trabaja con números enteros.

Para más información sobre estas estructuras, se puede revisar la documentación del código de cada uno de ellos, donde se explica como funcionan cada uno de los métodos.

Ambos se encuentran en la carpeta `physics` del proyecto.

5.3. Blending de comportamientos

Cada comportamiento fue escrito como una función, y para dar una interfaz general a cada comportamiento se escribió la clase `Behavior` que funciona como un contenedor de argumentos necesarios para ejecutar un comportamiento como su peso y el “handler” que hace referencia a la función que debe ser ejecutada para realizar el comportamiento.

Para realizar el “Arbitraje” entre comportamientos se diseño la clase `BehaviorGroup` que funciona como un conjunto de comportamientos que tienen una misma prioridad. Dicha clase define un método `execute` que se encarga de ejecutar uno a uno todos los comportamientos que contiene y de ir haciendo “Blending” entre ellos.

Por último, los personajes manejados por la inteligencia artificial definen un método **behave** que busca todos los **BehaviorGroup**'s asociados al mismo y los ejecutan, buscando el que tenga mayor prioridad y que devuelva una fuerza o “steering” superior a cierto umbral predefinido.

5.3.1. Prioridades de los comportamientos

Cuadro 5.1: Como se comportan los reggaetoneros

Grupo de comportamiento	Prioridad
Wander	1
Pursue	3
Evade	4
Collision Avoidance	5
Flocking	6

5.4. Manejo de fuerzas y movimiento sobre los reggaetoneros

Tras cada ciclo del juego, la velocidad y posición de un reggaetonero es calculada como:

$$p = p_0 + v * t \quad (5.1)$$

$$v = v_0 + a * t^2 \quad (5.2)$$

Donde p representa la nueva posición de un reggaetonero, p_0 la posición del ciclo anterior, v la nueva velocidad, v_0 la velocidad del ciclo anterior, t representa la cantidad de tiempo transcurrida entre cada ciclo y a la aceleración que lleva el reggaetonero en el presente ciclo.

Al inicio del juego, la posición de cada reggaetonero viene precargada en el programa, y su velocidad es 0. La aceleración es calculada cada ciclo siguiendo la fórmula:

$$a = \frac{\sum F}{m} \quad (5.3)$$

Donde $\sum F$ es la sumatoria de todas las fuerzas que actúan sobre el reggaetonero durante ese ciclo, y m es su respectiva masa.

La fuerza sobre los reggaetoneros se calcula como la resultante de la sumatoria entre 4 fuerzas principales que interactúan continuamente con cada uno de los personajes llevados por la inteligencia artificial. De esta forma podemos ver que:

$$\sum F = F_{bh} + F_{ht} + F_{bl} + F_{fr} \quad (5.4)$$

Donde:

- F_{bh} : Fuerza de comportamiento
- F_{ht} : Fuerza de golpes recibidos por otro personaje o por algún obstáculo.
- F_{bl} : Fuerza de balas recibidas
- F_{fr} : Fuerza de fricción

Capítulo 6

Problemas encontrados

6.1. Velocidades después de una colisión

Al chocar dos personajes el intercambio de velocidades no se realiza como debería, pues todavía no se toma en cuenta la masa del personaje y se asume que su masa “no existe” (o que es igual a 1).

A parte de esto, el choque entre los reggaetoneros hace que su orientación varíe abruptamente, mostrando un comportamiento errático por un período de tiempo corto.

6.2. Problemas con la cohesión de los “boids”

Cuando los reggaetoneros actúa en conjunto como una manada, la cohesión hacia el centro de masa calculado entre ellos genera una fuerza que a veces es mucho mayor a el resto de las fuerzas requeridas en el *Flocking*. Esto genera un comportamiento inestable en el resultado final.

La solución encontrada hasta ahora ha sido darle mucho más peso a la separación en relación con el peso dado a la cohesión.

6.3. Muchas balas = Juego lento

Cuando se disparan muchas balas, el juego empieza a ponerse lento mientras estas balas están en el aire. Una vez que tocan el suelo son eliminadas de memoria y el juego vuelve a recuperar su velocidad. Esto es debido a que las balas, como son esféricas, son pesadas para hacerles “Render”, y probablemente también se debe al procesador de la computadora donde se está desarrollando el juego.

6.4. Optimización del algoritmo de búsqueda de caminos mínimos

Otro de los problemas encontrados es referente a la velocidad de procesamiento del juego: conseguir constantemente el camino mínimo para el movimiento de los personajes llevados por la inteligencia artificial requiere una cantidad de cálculos que se tienen que realizar continuamente.

Esto ocasionaba que, cuando los personajes persiguen o navegan el mapa siguiendo los grafos, el juego se pone lento. Para solucionar esto se realizó un estudio sobre los cálculos requeridos para llevar acabo el algoritmo de búsqueda de caminos mínimos (A^*) y se vió que el cálculo de la heurística (la distancia euclidiana entre los puntos) era preprocesable antes de empezar el juego, debido a que los nodos no cambian de posición a lo largo del desarrollo del juego. De esta forma, al momento de cargar el nivel, los obstaculos y los personajes, también se crea el grafo, con sus sectores y nodos correspondientes, y se calcula la distancia entre cada uno de los nodos.

Esta decisión mejora ampliamente el comportamiento del juego al momento de tener reggaetoneros siguiendo caminos óptimos, debido a que tiene un impacto en el cálculo permanente de la heurística entre cada punto y el nodo destino y además provee de forma directa (acceso lineal sobre una matriz) el costo de trasladarse entre un nodo y otro.

6.5. Choques contra obstaculos

Para cada obstaculo en el juego, y para cada personaje, existe un radio de choque que se utiliza para detectar colisiones entre objetos. En el caso particular de un choque entre un personaje y un obstaculo, se trata de calcular cuanto es la fuerza normal que ejerce dicho obstaculo sobre el personaje que chocó contra él. El problema ha sido que cuando los personajes chocan de forma directa contra alguna de las paredes de los obstaculos, se aplica la fuerza normal que tiene dicha pared contra el personaje, pero si el personaje choca contra alguna esquina del obstaculo entonces se quedan ambos pegados y el personaje no logra salir, al menos de que otras fuerzas interactuen sobre él (por ejemplo, la fuerza de algunas balas).