

**UNIVERSIDAD AUTÓNOMA METROPOLITANA  
UNIDAD AZCAPOTZALCO  
DIVISIÓN DE CIENCIAS BÁSICAS E INGENIERÍA**

**INGENIERÍA EN COMPUTACIÓN**

**Reporte DE PROYECTO TERMINAL:**

**Implementación de motor de juego para  
juegos tipo “Tower Defense”**

**Montaño Ayala Oscar Xahil  
Matrícula 205302776**

**TRIMESTRE 11-P**

## Tabla de contenido

<b>Introducción</b>	<b>2</b>
<b>De las unidades</b>	<b>3</b>
<b>De las balas</b>	<b>6</b>
<b>De la lista ligada de unidades</b>	<b>7</b>
<b>Recolectar entradas del jugador</b>	<b>8</b>
<b>Procesamiento lógico o lógica del juego</b>	<b>10</b>
<b>Redibujar la pantalla (Renderización)</b>	<b>16</b>
<b>Menú Principal</b>	<b>18</b>
<b>Referencias</b>	<b>19</b>

## Introducción

Como ya se había descrito en la *propuesta de proyecto terminal*, este proyecto se trata de la realización de un videojuego tipo Tower Defense[1], pero, como ya se explico tendrá algunas variantes, en este reporte se describirá como se realizaron cada uno de los 3 módulos principales (Recolectar entradas del jugador, procesamiento lógico o lógica del juego, redibujo de pantalla o renderizado).

Por otro lado se tratara de explicar cómo se alcanzaron los objetivos establecidos en la propuesta, primero los particulares y por último el general.

Como ya se mencionó se tomaron como punto de referencia algunos juegos comerciales entre ellos: *Rampart* [2], *Master of Defense* [3], *Desktop Tower Defense*[3], *Fieldrunners* [4], *Trenches*[5].

La motivación para realizar este proyecto era la poca producción de este tipo de software en México y su alto consumo, se citó anteriormente al Universal [6] y al Financiero [7] para consultar sus cifras en cuanto a consumo de videojuegos.

## De las unidades

Como se planteó en un inicio hay cuatro diferentes tipos de unidades, 2 de ataque y 2 de defensa, para representar cada unidad se creó una clase, estas clases heredan de una clase base llamada *Entidad\_Virtual*, la cual contiene métodos y atributos que varían según el tipo de unidad, se puede observar el diagrama de las clases a continuación:

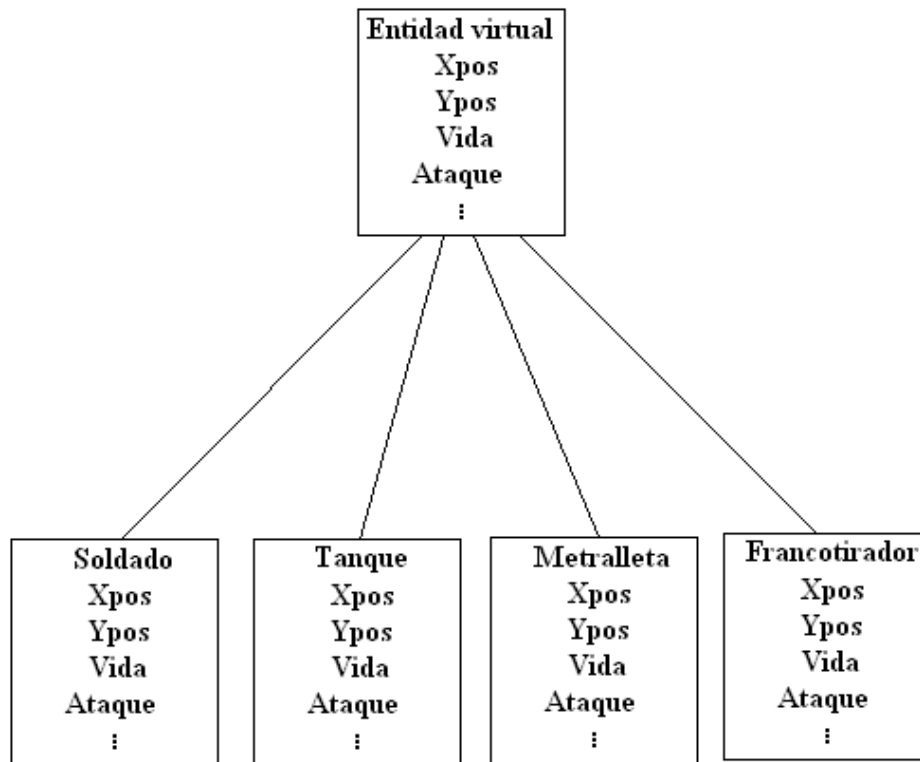


Figura 1: Entidades virtuales

Como ya se mencionó, cada unidad tiene diferentes atributos y respuestas a los eventos, estos comportamientos son lo que los hace únicos y diferentes entre sí.

### Los soldados o unidades de infantería:



Figura 2: Entidad soldado

Es la unidad más pequeña en el terreno de juego, ocupa un solo espacio en el quadtree de ubicación espacial. Además de ser el único que necesita crear un camino para llegar a su

destino, es la unidad protagonista del árbol de decisiones de la IA siendo la mayoría de las decisiones tomadas en respuesta a los datos obtenidos por esta unidad, como la creación de otras unidades, localización de enemigos, etc.

### **Los tanques:**



Figura 3: Entidad tanque

Esta unidad, de ataque al igual que los soldados es mucho más grande y no ocupa un lugar en el quadtree de ubicación, ocupa toda una “Caja” (una matriz del tamaño de la imagen que se desplaza conforme la unidad lo hace), por razones de tamaño esta unidad no crea un camino como el soldado, llega a su destino probando los caminos que tiene disponibles, esto es si: colisionó con otra unidad regresa por donde llegó y prueba una forma diferente de llegar a su destino, esta es la única unidad que cuenta con dos tipos de armas: la metralleta (más rápida pero menos poderosa) y el cañón (más lento pero más poderoso, además de contar con un número limitado de balas).

### **La torre de metralleta:**



Figura 4: Entidad metralleta

Esta unidad es fija, por lo que su posición desde que es creada no cambia, pero a diferencia de las unidades móviles se puede crear donde el jugador lo desee, o la IA lo decida, una vez

seleccionado el lugar de su ubicación, si es posible colocarla ahí de igual manera que el tanque se creará una caja que ocupe todo el lugar que la imagen necesite en el quadtree de ubicación, pero tardará un tiempo en funcionar a partir de su creación. Esta es la única unidad que puede cambiar de bando, el soldado controlando la metralleta muere antes que ésta sea destruida, una vez que la metralleta esté sola cualquier soldado de cualquier bando puede tomarla y utilizarla para atacar a los enemigos.

### **Los francotiradores:**



Figura 5: Entidad francotirador

De igual forma esta unidad es fija, permanecerá en el lugar donde se decida crear y también tiene un tiempo antes de ser útil, esta unidad es la única con la habilidad de esconderse de las unidades enemigas, para esto tiene tres estados; en el primero está oculto(al igual que su matriz de riesgos) y no ataca a las unidades, en el segundo puede atacar a una unidad enemiga seleccionada y también permanece oculto, en el tercero ataca cualquier unidad enemiga a su alcance pero ya no está oculto y su matriz de riesgos tampoco. En el caso de los francotiradores controlados por la IA siempre están ocultos y las unidades del jugador no los atacarán a menos que reciban la orden explícitamente.

## De las balas

Cada unidad puede disparar un tipo de bala diferente, en el caso de los tanques son dos tipos las que puede disparar, por lo tanto se tienen diferentes tipos de balas, para esto, al igual que las entidades virtuales se creó una clase base llamada *Entidad\_bala*, de la cual heredan todos los tipos de balas, siendo la única diferencia la forma de dañar a la unidades, y algunos atributos como el daño y la velocidad.

La mayoría de las balas dañan al enemigo al impactarlo, sólo a la unidad que impactaron y la única diferencia es en cuanto a la cantidad de vida que le quitan, pero hay dos tipos de balas que tienen un comportamiento diferente: la bala de cañón, la bomba y la granada(ésta exclusiva de la base del bando controlado por la IA), estas balas dañan a toda unidad alrededor del impacto, la bala de cañón en menor magnitud y la bomba abarca un área más grande.

### Tipos de balas:

Soldado/Francotirador:



Torre Metralleta:



Tanque Metralleta:



Tanque Cañón:



Bombas:



Granada:



Para calcular la trayectoria de la bala, se cuenta con la posición inicial y la posición final, con estos datos se obtiene la pendiente de la recta con la fórmula:

$$m = |(Y^1 - Y)/(X^1 - X)|$$

una vez obtenida la pendiente, la componente en X y Y se calcula con:

$$\begin{aligned} C_x &= \cos(\arctan(m))V \\ C_y &= \sin(\arctan(m))V \end{aligned}$$

Siendo  $V$  la velocidad correspondiente a cada bala, el resultado se muestra a continuación.

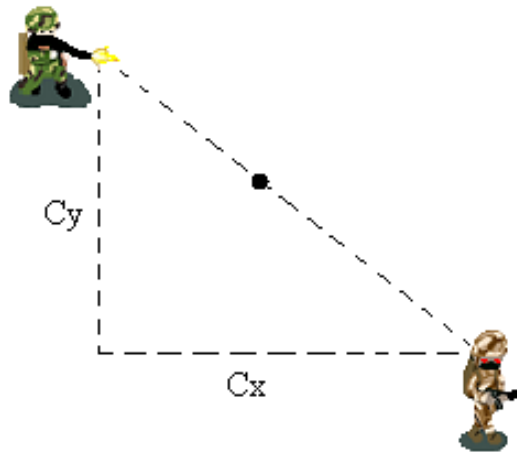


Figura 6: Trayectoria de las balas

## De la lista ligada de unidades

Todas las entidades del juego están contenidas en una lista ligada, esta lista es de tipo Entidad\_Virtual, cada unidad creada debe ser añadida a la lista, cuando dicha unidad sea destruida y después de un tiempo debe de ser sacada de la lista para que ésta no crezca demasiado.

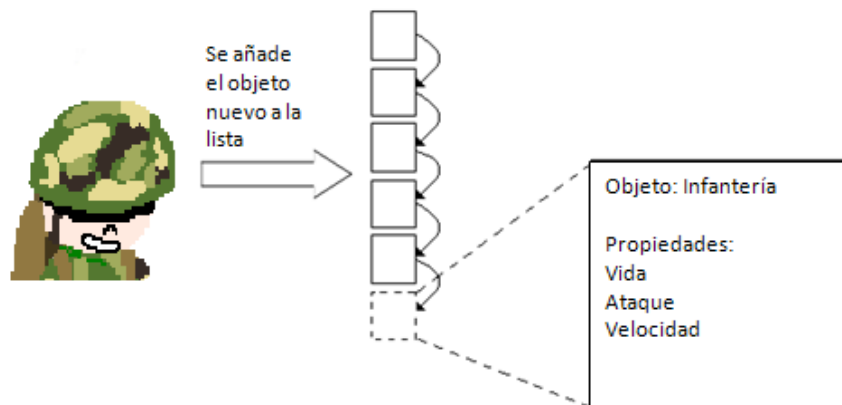


Figura 7: Añadiendo nueva unidad a la lista ligada



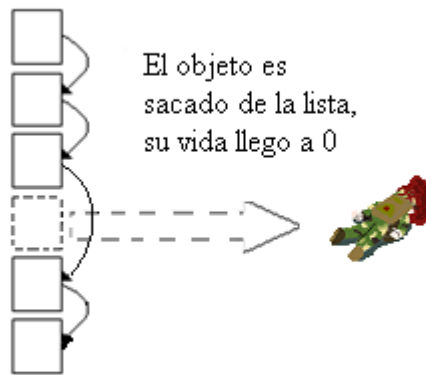


Figura 8: Sacando un objeto de la lista ligada

## Recolectar entradas del jugador

Para que haya entidades en el mundo virtual que interactúen, primero se deben crear. Esto lo pueden hacer tanto el jugador como la IA del juego, pero lo segundo se tratará en el apartado de Procesamiento Lógico, en este apartado sólo se hablará de la interacción del jugador con el mundo virtual.

### Creación de nuevas unidades:

El jugador puede crear nuevas unidades cuando así lo desee, aunque también hay que tomar otros factores en cuenta, como son el dinero del que dispone y el número de unidades actuales con el que cuenta, suponiendo que cumple con todos los requisitos puede crear los 4 tipos de unidades de los que dispone, ya sea con el uso de teclas o botones que se presentan en pantalla.

Para crear unidades de infantería se puede hacer con el uso de la tecla “V” o apretando el botón ubicado en el extremo inferior de creación de soldados, dicho botón es como se presenta en seguida:



Figura 9: Botón soldado

Si cualquier evento de los antes mencionados es verdadero, el método *Crear\_Sniper\_Vrd*, de la clase *Funciones\_C* creará una nueva entidad llamada soldado, la cual se añadirá a la lista ligada de unidades

Para crear tanques, al igual que para la infanteria se tienen dos formas, con la tecla “T” o con el boton de creación de tanques que es como se muestra en seguida:



Figura 10: Botón tanque

Estos eventos con el método *Crear\_Tanq\_Vrd* de la misma clase *Funciones\_C* crearán una entidad de tipo tanque que se añadirá a la lista ligada.

Para crear torres de metralleta se puede usar la tecla “M” o con el uso del botón de creación de metralletas que aparece en seguida:



Figura 11: Botón metralleta

Ambos eventos con el método *Crear\_Metrallata\_Vrd*, de la clase *Funciones\_C* crearán una entidad de tipo metralleta que se añadirá a la lista ligada.

Por ultimo para crear francotiradores se usara la tecla “S”, o el boton que lo cree, el mismo se muestra abajo:



Figura 12: Botón francotirador

Estas son todas las unidades jugables disponibles y que el jugador puede crear, pero no son todas las acciones que el jugador puede realizar, además de la creación de elementos el jugador puede:

- Desplazarse en el mapa : Esto con las teclas arriba, abajo, derecha e izquierda, o con la ayuda del minimapa, para usar este sólo es necesario dar click dentro del minimapa y el mapa se desplazará a donde se desee, el desplazamiento se controla con la clase *Desplazamiento\_Fondo*, las variables que contienen los datos de la ubicación actual respecto al fondo son FondoX y FondoY, declaradas en la clase Game1. El minimapa es controlado por el método navegar de la misma clase Desplazamiento\_Fondo, para hacer que el minimapa funcione bien se necesitó hacer un mapeo del tamaño del minimapa, al tamaño del terreno de juego.
- Seleccionar unidades : Esto simplemente haciendo click sobre la unidad deseada, el método *Buscar\_mouse* de la clase Funciones\_C es el encargado de buscar en el quadtree tomando en cuenta la posición del mouse en el momento de dar click, para buscar en el terreno de juego se divide cada vez el mapa en 4 hasta llegar al quad más chico, en este caso si se llegara a topar con quad aun no creado regresará que no encontro unidad en esa posición y no se seleccionar nada.
- Mover unidades : Teniendo unidades seleccionadas, siendo estas de infantería o tanques pueden desplazarse haciendo click derecho en el lugar al que se desee mandar, también funciona con el minimapa, para hacer esto bastó con asignarle la posición del mouse al momento de dar click a las variables que controlan el punto a seguir de cada unidad, además de poner la variable *seguir*(de tipo bool) a true para que el procesamiento sepa que esa unidad se está moviendo.
- Activar las diferentes características de cada unidad: Cada unidad tiene un comportamiento diferente como ya se describió en *De las unidades* con el uso de teclas o botones estos comportamientos pueden cambiar como se desee.

## Procesamiento lógico o lógica del juego

En esta etapa la lista ligada que se tiene de la creación de unidades(así como de la eliminación de las mismas), será procesada para calcular el nuevo estado de cada unidad dentro del terreno de juego.

### Quadtrees:

Para calcular las interacciones de las unidades se hicieron dos quadtrees[8] como se explica a continuación:

- Quadtree de ubicación: Se utiliza para tener la ubicación espacial de las entidades que interactúan en el mundo, es la unidad mínima de espacio que puede ser ocupada por una entidad y es exclusivo, esto es, no puede haber más de 1 unidad en el

mismo quad, cuando 2 o mas unidades tratan de entrar al mismo quad se crea una colisión y las unidades tendran que crear una ruta para resolver su camino como se vera mas adelante.

Cuando una unidad es creada debe mapearse en en quadtree de ubicación y poner las banderas de ocupado a verdadero, de igual forma cada quad hoja debe saber que unidad es la que contiene, para meter una unidad en el quadtree se invoca cualquiera de los métodos *Anadir\_Sprite*(con sus 2 sobrecargas) y *Ocupar\_Lugar*, ambos métodos declarados en la clase *Quadtree*, los cuales dividen recursivamente en 4 la pantalla hasta llegar a los quad's mas pequeños, en el caso de que no exista el quad aun(por que no ha habia ninguna unidad ahí), el nuevo quad debe ser instanciado.

Como ya se mencionó hay unidades que ocupan, por su tamaño mas de un quad, para estas unidades se creo una matriz de quad's, la cual contiene todos los quad's que esta ocupando la unidad.

Al momento de sacar una unidad de la lista ligada, tambien debe ser eliminada del quadtree, esto se hace con el método *Desocupar\_Lugar* de la misma clase *Quadtree*, este método elimina el quad desocupado, así cuando alguna otra unidad intente entrar, el quad estará disponible.

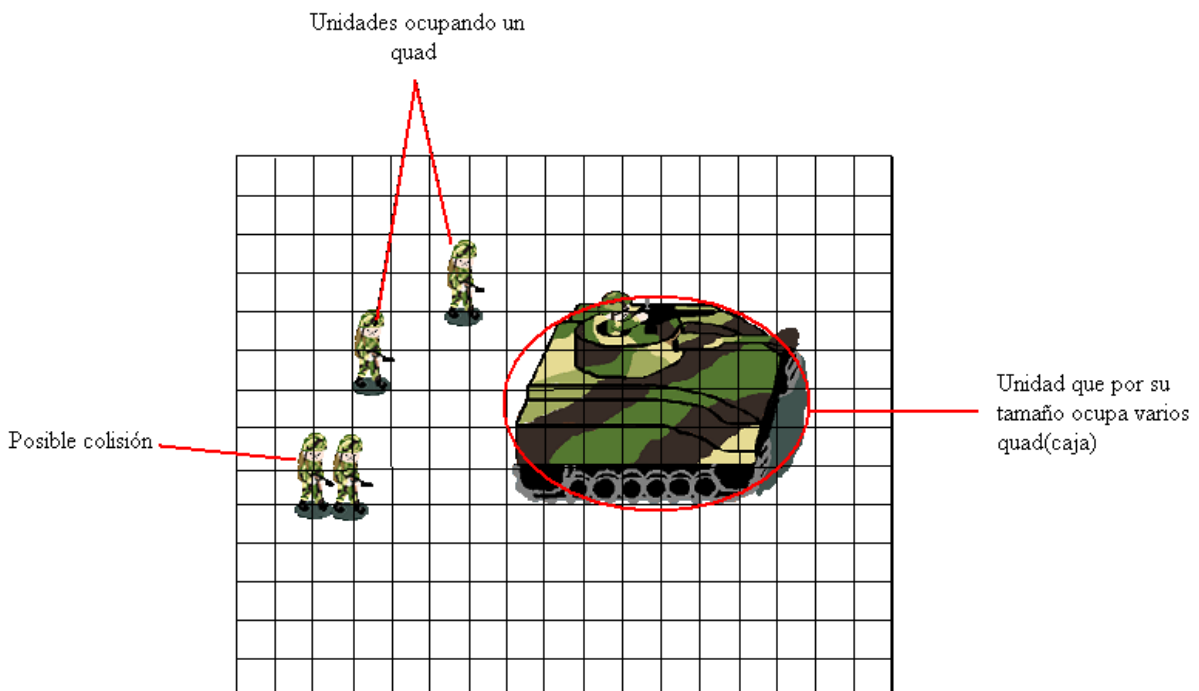


Figura 13: Representación del quadtree de ubicación

- Quadtree para encontrar enemigos: Este es mas grande que el quadtree de ubicación y no es exclusivo, puede almacenar tantas entidades como el quadtree de ubicación lo permita, se utiliza para poder encontrar enemigos al alcance.

El algoritmo que se uso para detectar enemigos fue el siguiente:

Si(unidad no atacando)

    Si(unidad parado o unidad en ataque)

        Por cada quad\_fuego vecino y el propio

            Si(hay enemigos en quad\_fuego y enemigo al alcance)

                Unidad atacar enemigo

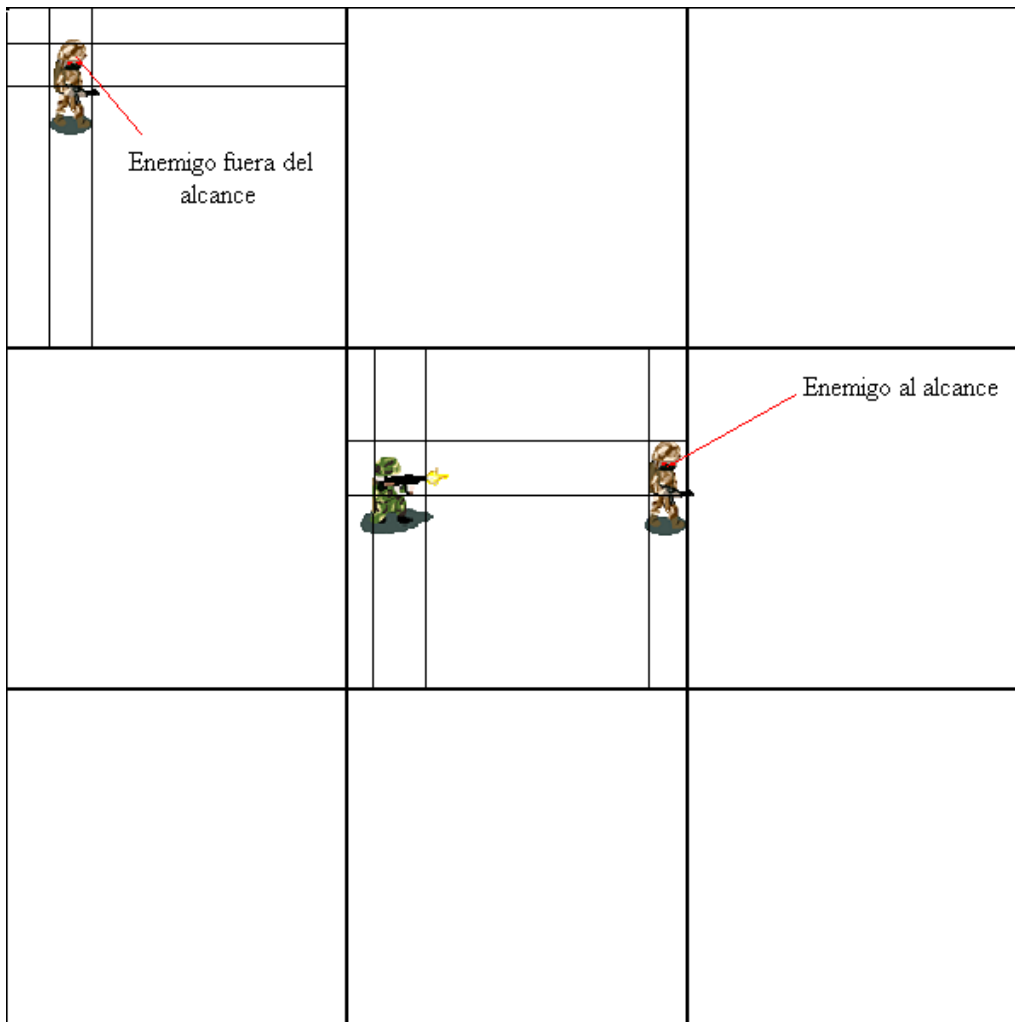


Figura 14: Quadtree de fuego

Y para tener una correcta representación en las imágenes, se tuvo que calcular el ángulo del enemigo con respecto a la unidad controlada por el usuario para así poder decidir que imagen sería la que mejor se adaptara para dar un buen efecto visual, el resultado fue como sigue:

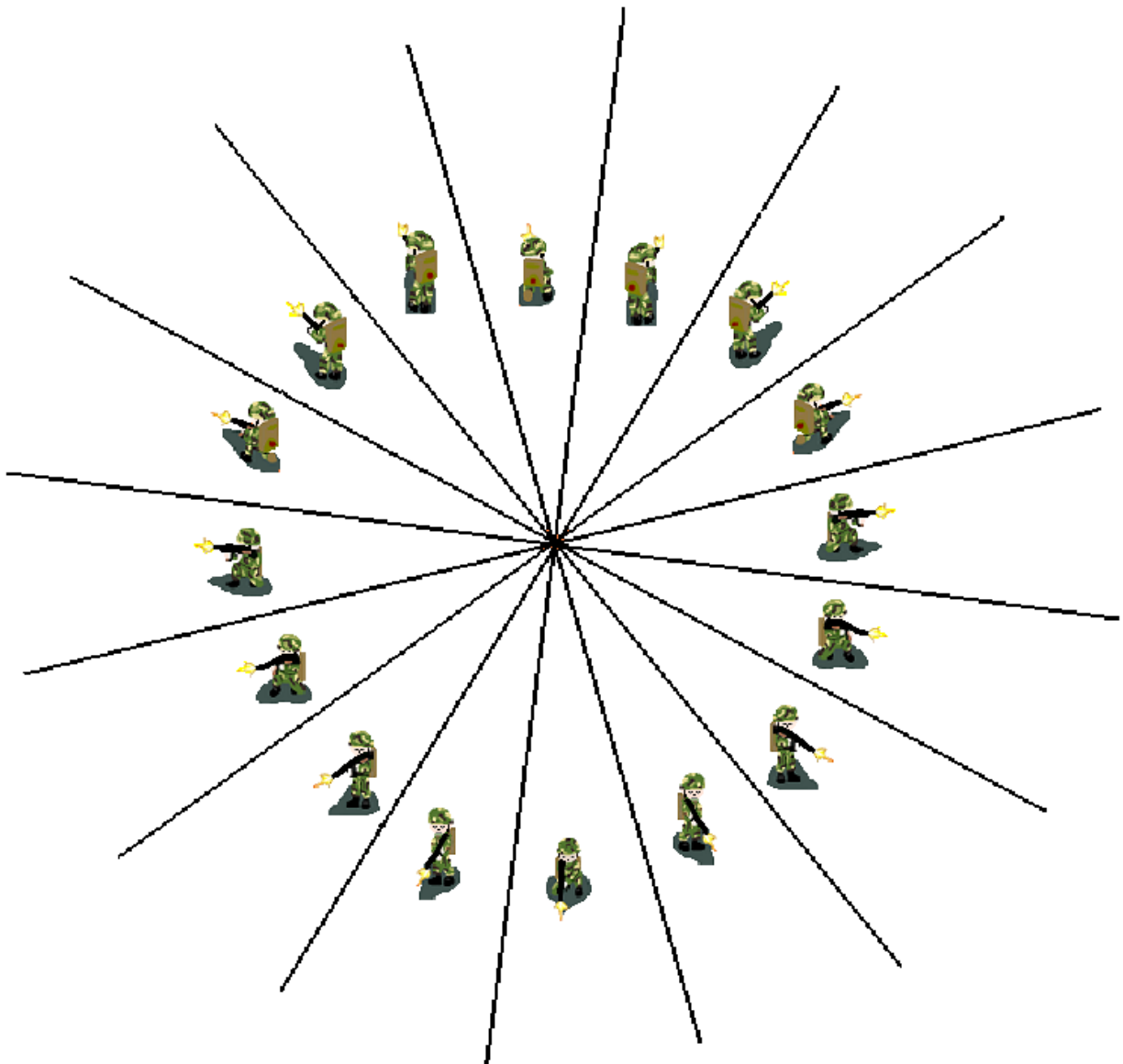


Figura 15: Distribución de los ángulos

### Resolver la ruta a seguir:

Una vez que se haya detectado una colisión, o que es inminente si se sigue el camino actual, la entidad a colisionar deberá crear un nuevo camino a su destino, al principio se planteó que se utilizaría el algoritmo de Floyd, pero resultó no ser apropiado para este problema, ya que este algoritmo encuentra el camino entre todos los vértices, en un tiempo

$n^3$  con  $n$  aristas en el grafo, para este problema en particular el número de aristas es un  $n$  muy grande, y los pesos de las aristas cambian de manera constante (moviendo entidades y ocupando lugares que hacen que se corten ciertos caminos o que exista un camino más corto, etc.), tomando en cuenta que se tiene que resolver el camino para cada entidad que lo requiera, que el peso de las aristas puede cambiar y que puede haber varias entidades en el mundo virtual, además que en un juego se requiere de respuesta en un tiempo muy corto para que no afecte el desempeño, se llegó a la conclusión de que no era el mejor método para resolver este problema.

El algoritmo que se utilizó se trata de una variación del algoritmo de Dijkstra, este algoritmo se utiliza para la determinación del camino más corto dado un vértice origen al resto de vértices en un grafo con pesos en cada arista. Su nombre se refiere a Edsger Dijkstra, quien lo describió por primera vez en 1959[9].

En esta modificación solo se toman los 2 vértices más cercanos al vértice final, partiendo del vértice actual, una vez creado el camino la unidad recorre este camino y, si su vértice actual no es su vértice final, la unidad simplemente avanzará en línea hacia su destino, si se da el caso de que caiga nuevamente en una colisión, el proceso se repetirá hasta que llegue a su destino, el camino de la entidad se guarda en una pila y conforme vaya llegando a los vértices guardados en la pila, se mandará al siguiente vértice de la pila.

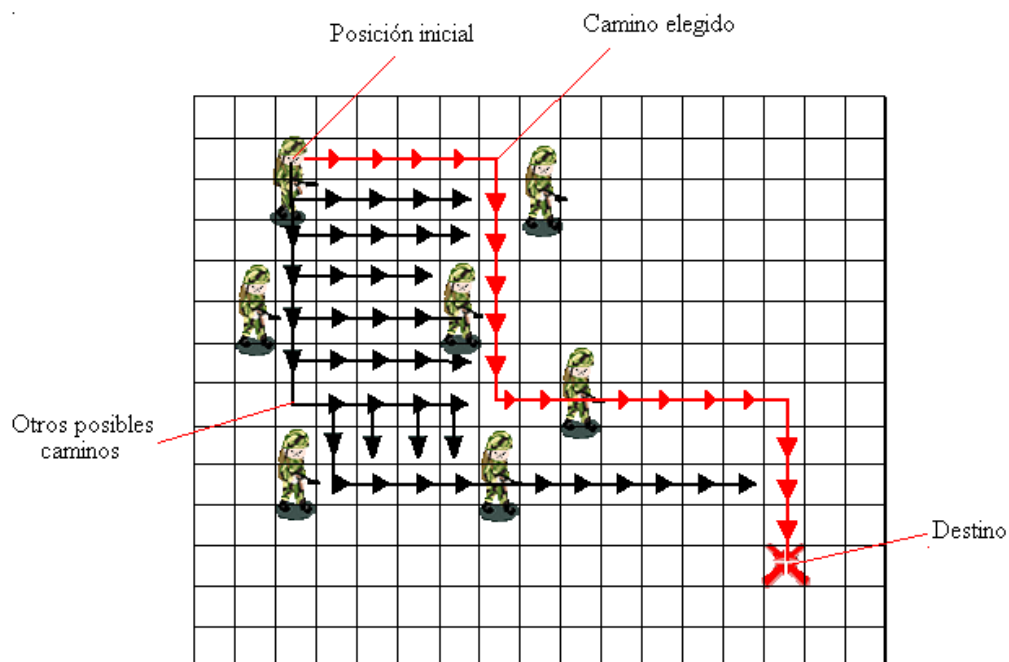


Figura 16: Creación de rutas

Aparte del peso de las aristas también se toma en cuenta (para las unidades controladas por la IA) la matriz de riesgos inherente a cada unidad creada por el jugador, esta matriz afecta sumándose a la distancia que hay entre el vértice actual y el vértice destino, así mientras más poder tiene la entidad, mayor es el número que mapea en la matriz de riesgos y el camino se tomará como más largo.







100	100	100	100	100	100														
100	100		100	100	100														
100	100		100	100	100														
100	100	100	100	100	100														
100	100	100	100	100	100														
1000	1000	1000	1000	1000	1000	1000	1000	1000	1000										
1000	1000	1000	1000	1000	1000	1000	1000	1000	1000										
1000	1000	1000	1000	1000	1000	1000	1000	1000	1000										
1000	1000	1000	1000	1000	1000	1000	1000	1000	1000										
1000					1000	1000	1000	1000	1000										
1000					1000	1000	1000	1000	1000										
1000	1000	1000	1000	1000	1000	1000	1000	1000	1000										
1000	1000	1000	1000	1000	1000	1000	1000	1000	1000										

Figura 17: Matriz de riesgos

**Inteligencia artificial:**

Para la inteligencia artificial del juego se planteó un arbol de decisiones, dicho arbol resulto de la siguiente forma:



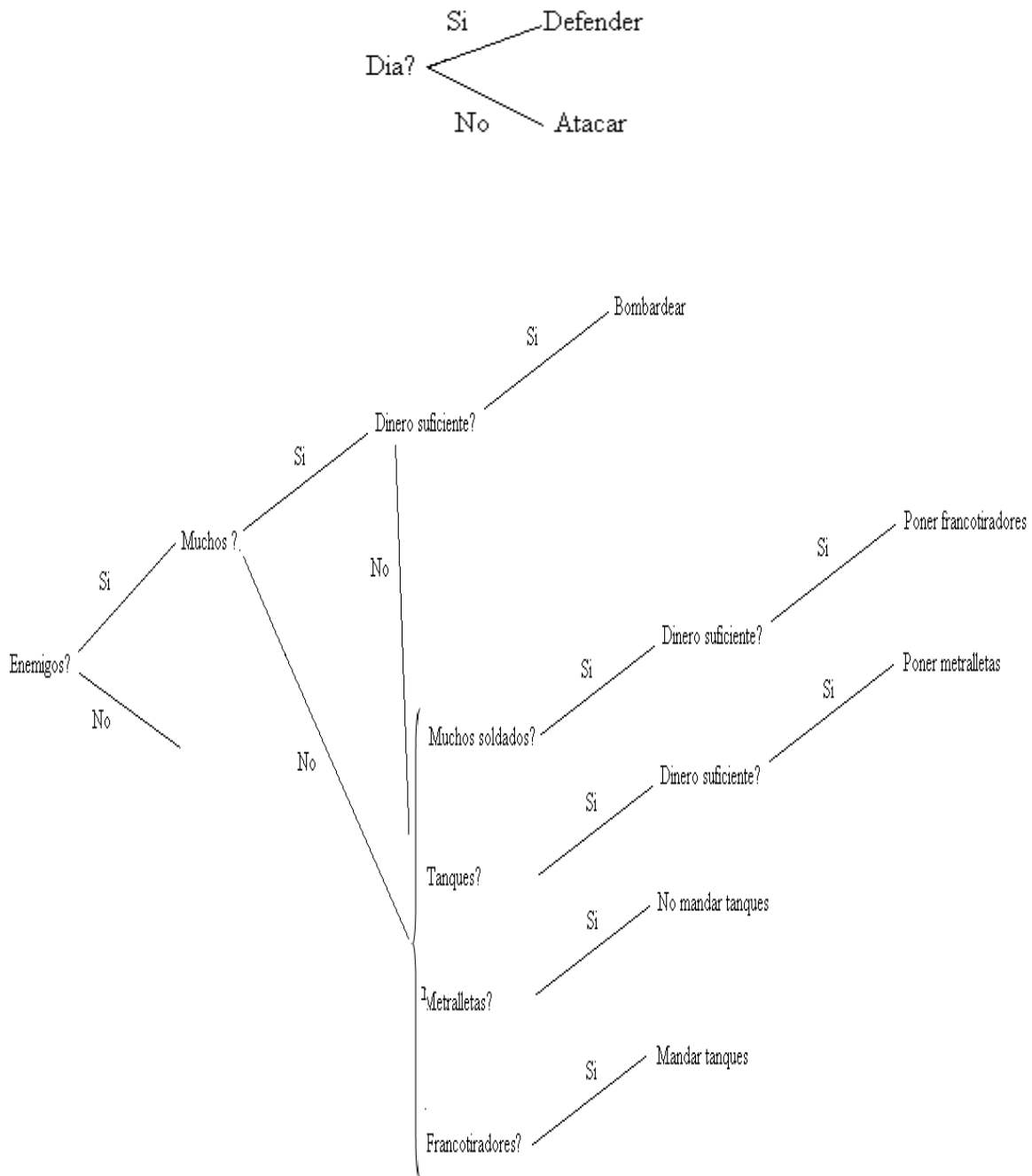


Figura18: Árbol de decisiones

## Redibujar la pantalla (Renderización)

Este último paso es donde se ven reflejadas las acciones y estados de cada entidad en el mundo virtual, si cambio de posición, si cambio de estado atacar a no atacar, si está muerto,

etc. todo esto se ve representado en la pantalla mediante el proceso de renderización, este proceso se lleva a cabo en el método *Draw* de la clase *Game1*, la cual está por defecto en cualquier proyecto de XNA, en dicho método se dibujan todas las unidades, así como el fondo que se encuentren en el área de vista del jugador, las que estén fuera de la vista del jugador no son dibujadas.

Para proyectar un redibujado correcto, las entidades, el fondo y cualquier elemento que se desee dibujar, no se puede hacer en el orden en que se crearon, como es el caso del procesamiento lógico, donde se procesa la lista conforme fueron creados, si esto se hiciera de esa forma el resultado sería que el fondo esté encima de las entidades, por mencionar algo, o que se dibujen de manera inapropiada como se muestra:



Figura 19: Mal renderizado

Para evitar estos problemas, se tienen que ordenar las entidades en la lista ligada conforme a su posición “Y”, esto porque el sistema de coordenadas, llamado coordenadas de pantalla, es diferente al convencional.

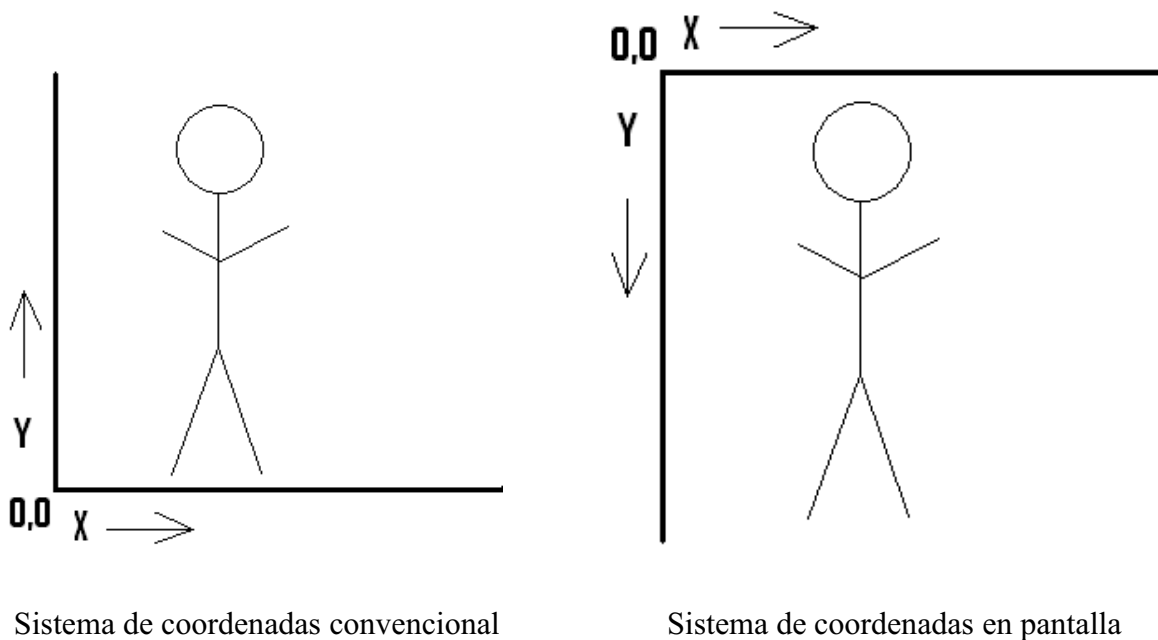


Figura 20: Sistema de coordenadas

Como puede apreciarse mientras más arriba esté la unidad si coordenada Y es menor, por lo tanto si se ordenan las entidades de menor a mayor tomando en cuenta su posición Y, el resultado será que las unidades con un valor en la posición Y serán las primeras en ser dibujadas, las siguientes unidades se dibujarán encima de las ya dibujadas y con eso se tiene el resultado deseado.



Figura 21: Renderizado correcto

Como se tenía contemplado se usaron graficas 2.5D [10], que son con las que los juegos de este tipo suelen trabajar para tener un mundo virtual más amplio.

## Menú Principal

Una vez iniciado el juego, el menú principal aparece dando las opciones de:

**Jugar:** Una vez seleccionada esta opción, todos los sprites y sonidos son cargados a las listas ligadas correspondientes, una vez hecho esto el juego inicia.

**Cargar:** Para esto se debió haber guardado previamente el juego. Para guardar un juego, mientras que la partida está en curso se pone en pausa y da las opciones de salir y guardar, si se selecciona guardar, todas las variables importantes del juego (dinero del jugador y la maquina, si es día o noche, etc.), así como las variables más representativas de todos los elementos de la lista ligada (vida actual, posición, etc.), se guardaran en el archivo **WRMGRS.wrg**, así cuando se quiera cargar solo se inicializara el juego con los valores guardados y se crearan objetos con los valores de vida, posición, etc. que fueron creados.

**Tutorial:** Se cargarán las imágenes y sonidos del tutorial que se reproducirán conforme el jugador lo desee.

**Opciones:** Se desplegarán las opciones de tamaño del mapa, la cual solo modifica una variable, la cual controla el tamaño del mundo virtual y ambos Quadtrees, así como la matriz de riesgos y la opción de dificultad, esta opción solo controla una bandera que determina la cantidad de dinero que se le da a cada parte, si esta en dificultad normal el jugador recibirá más dinero, si por el contrario es difícil, la maquina recibirá más.

**Créditos:** Despliega una pantalla con los créditos.

**Salir:** Termina la aplicación.

## Referencias

- [1] Artículo de Wikipedia sobre Tower Defense  
[http://es.wikipedia.org/wiki/Tower\\_defense](http://es.wikipedia.org/wiki/Tower_defense) 03-Julio-2011
- [2] Artículo de Wikipedia sobre Rampart  
[http://en.wikipedia.org/wiki/Rampart\\_%28arcade\\_game%29](http://en.wikipedia.org/wiki/Rampart_%28arcade_game%29) 03-Julio-2011
- [3] Artículo de Wikipedia sobre “Desktop Tower Defense”  
[http://en.wikipedia.org/wiki/Desktop\\_Tower\\_Defense](http://en.wikipedia.org/wiki/Desktop_Tower_Defense) 03-Julio-2011
- [4] Página oficial de Fieldrunners  
<http://fieldrunners.com/> 03-Julio-2011
- [5] Página oficial de Thunder Game  
<http://www.thundergameworks.com/> 03-Julio-2011
- [6] México, líder regional en consumo de videojuegos; El Universal  
<http://www.eluniversal.com.mx/articulos/40467.html> 03-Julio-2011
- [7] Tiene México 70% del mercado latinoamericano de videojuegos, El Financiero  
<http://www.elfinanciero.com.mx/ElFinanciero/Portal/cfpages/contentmgr.cfm?docId=222068&docTipo=1&orderby=docid&sortby=ASC> 16-Junio-2010
- [8] Artículo de Wikipedia, Quadtree  
<http://en.wikipedia.org/wiki/Quadtree> 03-Julio-2011
- [9] Artículo de Wikipedia Algoritmo de Dijkstra  
[http://es.wikipedia.org/wiki/Algoritmo\\_de\\_Dijkstra](http://es.wikipedia.org/wiki/Algoritmo_de_Dijkstra) 03-Julio-2011
- [10] Artículo de Wikipedia, 2.5D  
<http://en.wikipedia.org/wiki/2.5D> 03-Julio-2011