

**Instituto Tecnológico de Costa Rica**  
**Escuela de Ingeniería en Computación**

**Segunda Tarea Programada**

**Estudiante:**

Carlos Jenkins  
200769999

**Curso:**

Algoritmos y Estructuras de Datos 2

**Grupo:**

3

**Profesora:**

Julia Espinoza

**Entrega:**

viernes, 30 de mayo de 2008

**Cartago, 2008**

## 1) Descripción general:

El presente proyecto consiste en la creación de un juego clásico conocido como *Galaxy*, *Space Invaders*, *Invaders From Space*, etc. Los objetivos específicos son la implementación de dos estructuras de datos, árboles B y tablas Hash, y un algoritmo de búsqueda conocido como A\* (A estrella).

## 2) Problemas encontrados:

La dificultad del proyecto fue importante debido a la mezcla de varios factores:

- Fue necesario aprender un nuevo lenguaje de programación para realizar el proyecto. En particular, C++ es un lenguaje complejo que no ofrece facilidades que otros lenguajes de más alto nivel ofrecen, como es la recolección automática de basura y una segura administración de la memoria y posee carencias importantes como la extrema falta de información en el proceso de depurado.
- Junto al lenguaje fue necesario adaptarse a un nuevo ambiente de desarrollo. En la sección de investigación se explica en qué entorno se desarrolló el proyecto y por qué.
- El esquema de clases fue confuso al inicio por la definición un archivo de encabezado y otro archivo de cuerpo.
- Nunca se había desarrollado un juego de esta envergadura. El paradigma de desarrollo de juegos supuso un reto importante pues no se tenía experiencia previa en el desarrollo de este tipo de aplicaciones.
- Un tercio del tiempo del proyecto se dedicó a investigación sobre el lenguaje, el entorno y las librerías gráficas. A causa del tipo de librerías gráfica seleccionadas, que ofrecen acceso de bajo nivel al hardware multimedia del equipo, el proceso de creación de la interfaz gráfica fue muy lento comparado a lenguajes que ofrecen su propio Toolkit visual (p.e Java Swing) y muchísimo más lento que lenguajes visuales que incorporan diseñador de interfaces (p.e C#).
- C++, por ser evolución de C y poseer tanta flexibilidad carece de coherencia en la forma de escribir código. A causa de esto a nivel de escritura y creación de código nunca se tuvo un objetivo concreto de presentación y desarrollo (existencia de varios estándares, carencia de funciones básicas, escritura de código tipo C).
- Por el tipo de proyecto, para utilizar las estructuras complejas es necesario tener un motor de juego funcional casi al 100% ya que estas no son indispensables para el funcionamiento del mismo por lo que el árbol se pospuso hasta lograr un engine de juego aceptable. Debido a lo anterior mencionado no se pudo implementar en el tiempo estimado para la realización del proyecto el árbol B. Debido a que el árbol no se logró implementar en su totalidad, la clase Árbol B del proyecto utiliza una lista para su funcionamiento.

### 3) Bitácora de trabajo - Investigación:

- **Domingo 11 de mayo:** Se considera los aspectos generales del proyecto: lenguaje, sistema operativo, bibliotecas gráficas, entorno de desarrollo, compilador, características deseadas (multiplataforma), etc. 2H aproximadamente.
- **Lunes 12 de mayo:** Se investiga sobre el lenguaje. Se escribe el primer “Hello World”. Esto se realiza en el IDE Geany sobre Linux con G++ como compilador. Asimismo, se investiga sobre entornos de desarrollo y compiladores. Dado que se desea desarrollar una aplicación multiplataforma se opta por un compilador y un entorno de desarrollo que cumplan esta característica. Para C++ la mejor opción de compilador multiplataforma es G++, el compilador de C++ del proyecto GNU. Por otro lado, entre las mejores opciones de IDEs para C++ se encuentra Eclipse + CDT, lo que no sólo dota el proyecto de una plataforma muy sólida pero además contempla el aspecto multiplataforma. Lo relacionado al ambiente de desarrollo se puede encontrar en el archivo “Entorno de desarrollo.pdf”. 5H aproximadamente.
- **Martes 13 de mayo:** Se discute el uso de las bibliotecas gráficas wxWidgets, un toolkit de desarrollo de interfaces gráficas multiplataforma, disponible para muchos lenguajes y escrito en C++. Se intenta crear la primera aplicación visual en C++. 3H aproximadamente.
- **Miercoles 14 de mayo:** Sin éxito con wxWidgets, se cuestiona el uso final real. Un juego demanda un único panel de dibujo o *canvas* de interacción. wxWidgets, aunque ideal para creación de interfaces con botones, list-box, etc, no ofrece una ventaja particular para este enfoque. Adicionalmente se tiene dificultades de interacción del entorno y wxWidgets por lo que se decide buscar otra alternativa.
- **Jueves 15 de mayo:** Gracias a investigaciones anteriores sobre software libre, se conoce de la existencia de la biblioteca SDL, por *Simple DirectMedia Layer*. SDL es un conjunto de bibliotecas multiplataforma que ofrecen un API para programación gráfica. Estas librerías proporcionan funciones básicas para realizar operaciones de dibujado 2D, manejo de eventos, gestión de efectos de sonido y música, carga y gestión de imágenes y archivos, etc. La investigación aporta datos muy relevante en el sentido que SDL es usado activamente en el desarrollo de juegos, por mencionar algunos, Quake4 y Neverwinter Nights. 5H aproximadamente.
- **Viernes 16 de mayo:** Se profundiza en el lenguaje, se investiga sobre las clases estándar *std*, las palabras específicas del lenguaje (using, include, etc), los tipos de datos (int, double, float, char), la nueva clase string y la antigua clase *char\**, los punteros y la referencia a datos, manejo de arreglos, etc. 4H aproximadamente.
- **Sábado 17 de mayo:** Se realizan las primeras pruebas con SDL que arrojan excelentes resultados. Se profundiza en el API de SDL y se intenta diseñar la mejor estratégica de programación que se adquee a la especificación de la tarea programada. Se consideran los threads y por consecuencia los semáforos, condiciones de un thread, implicaciones de un thread por animación, etc. 4H aproximadamente.
- **Lunes 19 de mayo:** Se inicia el proceso de codificación, se crea un primer proyecto con

clases básicas y manejo de datos simples. Se investiga sobre procesos de animación, división de una imagen (sprites sheets), sonidos, etc. 3H aproximadamente.

- **Martes 20 de mayo:** Se continúa con el primer proyecto, se crean las primeras clases importantes del juego: nave hero, invasor, etc. Se invierte mucho tiempo en aspectos gráficos y diseño de las gráficas del juego. 4H aproximadamente.
- **Miércoles 21 de mayo:** Se realiza una búsqueda exhaustiva de material gráfico y sonoro abierto. Se construye un pequeño repositorio de recursos gráficos y se indexa y da crédito a los autores (el juego tiene como objetivo ser libre bajo licencia GPL). 5H aproximadamente.
- **Jueves 22 de mayo:** Se descubre que la estructura actual del proyecto es deficiente y que de no cambiarla ahora implicaría un desorden en el código y muchos contratiempos en la implementación. Se decide reescribir desde cero el proyecto. La reescritura toma 8 horas de trabajo. La nueva organización es más eficiente y modularizada, siguiendo fielmente el paradigma orientado a objetos.
- **Sábado 24 de mayo:** Se implementan las clases auxiliares y se da enfoque al manejo de eventos. Se investiga sobre detección de colisiones. Se invierte tiempo en aspectos gráficos generales. Se investiga sobre control del *Frame rate*, *fade*, manejo de tiempo y transparencias en general. 6H.
- **Lunes 26 de mayo:** Se implementan algoritmos de control y de manejo del juego en general, condiciones, colisiones, vidas, efectos de ciertas acciones, game over, etc. 6H aproximadamente.
- **Martes 27 de mayo:** Se investiga sobre estrella y se implementa. Se inicia el manejo de la tabla Hash. Se invierte tiempo en aspectos gráficos. 10H aproximadamente.
- **Miércoles 28 de mayo:** Se hace una mejora importante de eficiencia al algoritmo de la estrella. Se continúa con aspectos del juego en general. Se invierte en aspectos gráficos. 12H aproximadamente.
- **Jueves 29 de mayo:** Se termina el diagrama de clases, el manejo de usuarios y escritura en archivos del top 10, carga de configuración y aspectos gráficos generales. Se avanza con la documentación. 12H aproximadamente.
- **Viernes 30 de mayo:** Se termina el manejo de las tablas Hash y los reportes. Se Implementa el nuevo requisito de poder graficar las escenas guardadas del juego. Se termina la documentación. 14H aproximadamente.

#### 4) Funcionamiento del juego:

El juego utiliza *Simple DirectMedia Layer*, una biblioteca de programación gráfica multiplataforma de licencia libre que hace alusión a capa de abstracción multimedia.. SDL en si es muy simple; actúa como una delgada capa multiplataforma que provee soporte para operaciones de píxel en 2D, sonido, acceso a archivos, manejo de eventos, tiempo, threading, entre otras. SDL es usado como complemento a OpenGL proporcionando una salida gráfica y entrada por medio del mouse y del teclado, dos de los aspectos más cuestionados de OpenGL. SDL utiliza los componentes nativos del sistema operativo en el que se encuentre como lo ilustra la figura número 1.

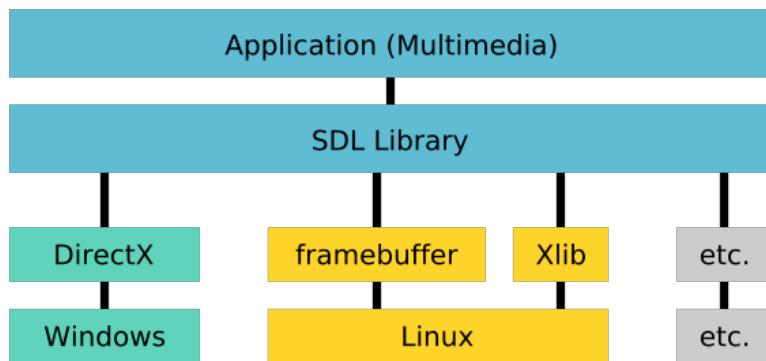


Figura 1: Capas de abstracción de varias plataformas que soporta SDL.

A nivel programático, el funcionamiento global de la tarea programada se puede ilustrar con el diagrama número 1. En la investigación se determinó que no era necesario utilizar múltiples threads ya que uno es suficiente para cumplir con los requerimientos de la tarea programada, incluso las animaciones. Este único hilo entra en un ciclo infinito o hasta que el usuario desee salir o haya perdido. Si ninguna de éstas dos condiciones se cumple entonces el ciclo se repite infinitamente y realiza siempre el mismo proceso:

1. Mueve los objetos en relación a la entrada del mouse.
2. Detecta colisiones entre objetos y toma las decisiones correspondientes.
3. Pinta todos los objetos en la pantalla.
4. Espera a que su cuota de tiempo se haya cumplido, esto para regular la cantidad de cuadros por segundo.

En cambio, el uso de threads obliga al programador a manejar dos o más hilos de procesamiento y debe ser él el encargado de evitar que dos threads colisionen por el mismo dato, para esto se implementan semáforos. Sin embargo, este enfoque no fue necesario.

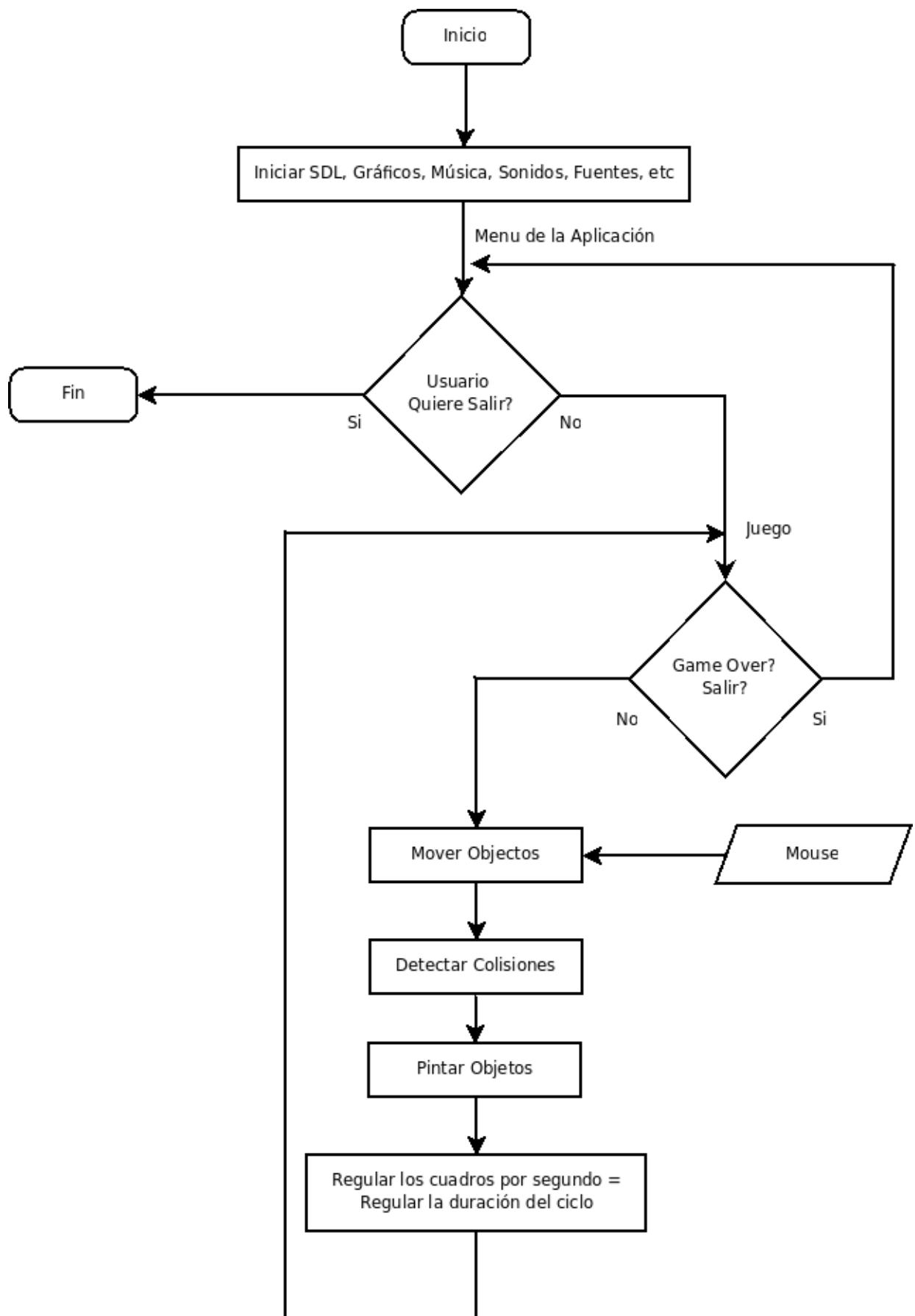


Diagrama 1: Funcionamiento del programa.

Desde el punto de vista de usuario el funcionamiento de la aplicación es muy intuitivo. Al iniciar, un menú permite al usuario seleccionar que desea realizar como muestra la figura número 2.



Figura 2: Menú principal del programa.

Las opciones son:

- Jugar (Play)
- Top 10 (Score)
- Acerca de (About)
- Salir (Exit)

Al presionar el botón de jugar se muestra una pequeña animación e inicia el juego. El funcionamiento del juego es simple ya que todo se realiza con el mouse como lo muestra la figura 3.

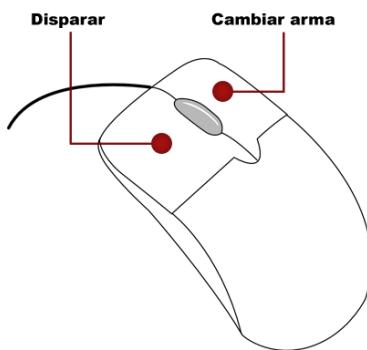


Figura 3: Función de los botones del mouse.

El jugador puede ver en todo momento las variables del juego que le conciernen, un panel situado en la parte inferior le indican el arma seleccionada y el parque de dicha arma, el tiempo de juego, el puntaje, el porcentaje de vida y la cantidad de vidas. La figura 4 muestra la interfaz de juego y sus distintos componentes: enemigos en formación, nave nodriza, disparos, objetos, nave héroe, estado de la nave héroe y el menú de estado mencionado anteriormente.



Figura 4: Interfaz de juego.

El juego posee una cantidad infinita de niveles, pues los enemigos se generan aleatoriamente cuando inicia el nivel. Por lo tanto, el juego termina únicamente si el usuario así de desea (presionando la tecla ESC y confirmando) o si se le acaban las vidas disponibles. En el primer caso el juego regresa al menú principal mientras que en el segundo se informa que el juego a terminado (Game Over) y si se rompe una nueva marca en puntos se le pregunta al usuario el nombre que desea asociar a dicha marca. Al regresar al menú principal, el usuario puede consultar esa información presionando el botón de Marcas (Score). La figura número 5 muestra esta interfaz. Sólo se mantienen las mejores 10 marcas basadas en puntos. Como dato adicional se muestran las naves destruidas, sin embargo, dado que las distintas naves ofrecen distinta recompensa en puntos, es posible tener menos naves destruidas y una mayor cantidad de puntos.

HIGH SCORES			
	NAME:	SCORE:	KILLS:
1	HAVOK	11470	179
2	HAVOK	10180	228
3	NAME	1940	55
4	NOMBRE	1840	51
5	HAVOKXD	1700	49
6	HAVOK	1650	44
7	PATO	1490	41
8	HAVOKXD	1080	29
9	HAVOKXD	1040	28
10	YO	300	6

PRESS LEFT CLICK TO CONTINUE

Figura 5: Top 10 en puntos.

Finalmente, el juego posee la característica de mostrar paso a paso los estados previos al estado de juego actual. Al pausar el juego se puede recorrer segundo a segundo las escenas anteriores. Esto es posible gracias a que el juego registra cada segundo el estado de todos los objetos relevantes dentro de una tabla Hash. El jugador sólo necesita mover el mouse de izquierda a derecha, como cuando está jugando normalmente. Un *slider* le indica la línea de tiempo del juego y cuál segundo está visualizando. Si el usuario desea regresar al juego simplemente se presiona la tecla P (para pausar y resumir se utiliza la tecla P). La figura número 6 muestra este sistema visualizando el segundo 17 de 33 totales.



Figura 6: Sistema de visualización de escenas.

## 5) Estrategias de implementación:

En esta sección se discute las 4 implementaciones principales del proyecto:

- Engine de juego.
- Algoritmo de búsqueda A estrella ( $A^*$ ).
- Tablas Hash y función Hash
- Árboles B.

La investigación sobre el paradigma de desarrollo de juegos advierte que el uso de threads genera complicaciones adicionales y no ofrece ventajas particulares a las implementaciones de un sólo ciclo o hilo, esto incluso en procesadores de varios núcleos. Tomando esta recomendación como paradigma se diseña el engine de juego en un único ciclo que debe incorporar todos los manejos necesarios de condiciones, efectos y eventos de entrada del jugador. El algoritmo principal de juego puede observarse en el diagrama de flujo de la página número 6. Este diagrama muestra la lógica, no así la organización del código real, ya que para cuestión de optimización se diseñó los algoritmos para recorrer la menor cantidad de veces las estructuras del juego, como las listas de objetos y eventos. Por lo tanto, un mismo ciclo puede detectar colisiones, pintar y generar más eventos para un mismo objeto mientras itera en la estructura particular. Las estructuras de datos

usadas para administrar los eventos y los distintos objetos de la escena son listas y colas simples ya que es necesario iterar por toda la estructura al menos una vez por ciclo o cuadro por segundo. Durante el juego se manejan tres tipos de clase de suma importancia: Invader, HeroShip y Object. Estas clases, instanciadas en las diferentes estructuras, y unas pocas clases utilitarias o de organización más, contienen y administran el manejo de variables del juego. El Invader es un objeto, sea nave nodriza o invasor, que posee definición espacial para la detección de colisiones, variables de estado como destruida o vida restante y de definición como la probabilidad de disparo y el tipo de nave. El HeroShip es la clase más importante para el manejo de variables del juego pues contiene métodos y atributos que definen el estado del jugador, sus capacidades, su definición espacial, las estadísticas del usuario, entre otros. El Object es la clase más genérica del juego, pues se instancia por cada artefacto, o objeto, que es liberado durante el juego, ya sea un disparo, un *power-up* como escudo o invisibilidad, un kamikaze, etc.

Para el recorrido del Kamikaze se utiliza un algoritmo basado en el algoritmo de búsqueda A\*. El algoritmo de estrella itera por la estructura hasta encontrar un camino. Sin embargo, ya que se debe realizar un ciclo de juego cada 1/30 segundo (Framerate o cuadros por segundo igual a 30) es obligatorio reducir la duración del mismo. Un recorrido por toda la estructura de la pantalla hasta encontrar el mejor camino provocó un máximo de 28 cuadros por segundo, lo cual para un juego simple es ligeramente por encima de lo aceptable. Adicionalmente, debido a que el kamikaze se mueve a cierta velocidad por la pantalla el camino calculado no puede ser recorrido en su totalidad antes de que otro ciclo obligue la recalculación del trayecto debido al cambio de posición del héroe generada por el usuario. Esta situación no solo provoca una reducción del rendimiento sino que necesita de una mayor cantidad de memoria para mantener el camino en memoria. La adaptación realizada al algoritmo provoca que solo los primeros pasos del mejor camino sean calculados en cada ciclo de juego, lo que a lo sumo provoca el mismo efecto. Adicionalmente, A\* revisa las ocho direcciones posibles a partir de un punto cualquiera de una matriz, sin embargo, el Kamikaze (o el héroe que también utiliza el algoritmo para recoger el botín) sólo se mueve hacia abajo, lo que limita las direcciones a 5 posibles. En un principio debido a que el algoritmo se basa en A\*, utilizaba un ciclo para determinar la dirección, sin embargo se determinó que un ciclo no era necesario y que matemáticamente podía simplificarse el proceso a una simple resta, suma y comparación. Aunque el algoritmo base fue A\*, el implementado en la tarea programada ofrece un rendimiento mejor y es computacionalmente más simple.

El juego es capaz de registrar el estado de la pantalla cada segundo y poder visualizarlo en cualquier momento. Esto es posible ya que el juego agrega un nodo de escena a un árbol B cada segundo que transcurre. Este nodo escena contiene los códigos hash de todos los objetos relevantes de la escena: el héroe, los invasores, la flota, la nave nodriza, los kamikazes y los objetos genéricos. Debido a que la información particular de un tipo de objeto no es indexable dentro de una tabla, pues para los más de 20 atributos sería necesario poder indexar todas las posibles combinaciones que suponen  $N!$  con N el número de atributos ( $20! = 2.43290201 \times 10^{18}$ ). Esto es computacionalmente complejo y requiere un sacrificio importante de memoria el poseer un tabla con esa cantidad de índices. Si a esto se suma que existen al menos 3 tipos de objetos con al menos 20 atributos cada uno el gasto de memoria sería absurdo. Se optó por un código hash que contiene la información en si ordenada de forma particular para que pueda ser reinterpretada al momento de reconstruir una escena basada en el segundo en la que fue grabada. Un ejemplo de esta implementación de tabla hash se da a continuación:

\*\*\*\*\*

Reporte de Tabla Hash

Second/Id: 20

Hero Hash: 710|3|75|12|0|0|1|0|447|604|64|64|463|620|32|32|64|0|64|64

Fleet Hash: 246|104|320|582|3|10|18|

1	Invader Hash:	8 2 1 2 0 192 64 64 64 246 104 64 64
2	Invader Hash:	8 2 1 2 0 192 64 64 64 320 104 64 64
3	Invader Hash:	8 2 1 2 0 128 64 64 64 394 104 64 64
4	Invader Hash:	8 2 1 2 0 128 64 64 64 468 104 64 64
5	Invader Hash:	8 2 1 2 0 64 64 64 64 542 104 64 64
6	Invader Hash:	8 2 1 2 0 192 64 64 64 616 104 64 64
7	Invader Hash:	8 2 1 2 0 192 64 64 64 690 104 64 64
8	Invader Hash:	8 2 1 2 0 192 64 64 64 764 104 64 64
9	Invader Hash:	7 1 1 1 0 64 0 64 64 246 168 64 64
10	Invader Hash:	7 1 1 1 0 192 0 64 64 320 168 64 64
11	Invader Hash:	7 1 1 1 0 192 0 64 64 394 168 64 64
12	Invader Hash:	7 1 1 1 0 64 0 64 64 468 168 64 64
13	Invader Hash:	7 1 1 1 0 192 0 64 64 542 168 64 64
14	Invader Hash:	7 1 1 1 0 64 0 64 64 616 168 64 64
15	Invader Hash:	7 1 1 1 0 192 0 64 64 690 168 64 64
16	Invader Hash:	7 1 0 0 0 0 0 64 64 668 168 64 64
17	Invader Hash:	7 1 1 1 0 64 0 64 64 246 232 64 64
18	Invader Hash:	7 1 0 0 0 64 0 64 64 467 212 64 64
19	Invader Hash:	7 1 0 0 0 64 0 64 64 484 212 64 64
20	Invader Hash:	7 1 0 0 0 128 0 64 64 318 202 64 64
21	Invader Hash:	7 1 0 0 0 64 0 64 64 323 202 64 64
22	Invader Hash:	7 1 0 0 0 320 0 64 64 535 212 64 64
23	Invader Hash:	7 1 0 0 0 320 0 64 64 609 212 64 64
24	Invader Hash:	7 1 0 0 0 128 0 64 64 614 232 64 64
25	Invader Hash:	7 1 1 1 0 192 0 64 64 246 296 64 64
26	Invader Hash:	7 1 0 0 0 256 0 64 64 461 276 64 64
27	Invader Hash:	7 1 0 0 0 256 0 64 64 346 266 64 64
28	Invader Hash:	7 1 0 0 0 64 0 64 64 387 266 64 64
29	Invader Hash:	7 1 0 0 0 64 0 64 64 323 266 64 64
30	Invader Hash:	7 1 0 0 0 128 0 64 64 505 276 64 64
31	Invader Hash:	7 1 0 0 0 256 0 64 64 585 276 64 64
32	Invader Hash:	7 1 0 1 0 192 0 64 64 614 286 64 64
33	Invader Hash:	8 2 1 2 0 128 64 64 64 246 360 64 64
34	Invader Hash:	8 2 0 0 0 320 64 64 64 452 340 64 64
35	Invader Hash:	8 2 0 0 0 64 64 64 64 457 330 64 64
36	Invader Hash:	8 2 0 0 0 0 64 64 64 486 330 64 64
37	Invader Hash:	8 2 0 0 0 128 64 64 64 515 340 64 64
38	Invader Hash:	8 2 0 0 0 0 64 64 64 487 340 64 64
39	Invader Hash:	8 2 0 0 0 0 64 64 64 474 340 64 64
40	Invader Hash:	8 2 0 0 0 64 64 64 64 635 340 64 64
1	Object Hash:	7 12 1 0 0 3 32 96 32 32 250 755 32 32
2	Object Hash:	7 12 1 0 0 3 0 96 32 32 315 682 32 32
3	Object Hash:	7 12 1 0 0 3 160 96 32 32 297 664 32 32
4	Object Hash:	7 12 1 0 0 3 64 96 32 32 308 601 32 32
5	Object Hash:	6 12 1 0 0 -5 128 0 32 32 554 239 32 32
6	Object Hash:	8 12 1 0 0 5 128 128 32 32 46 513 32 32
7	Object Hash:	7 12 1 0 0 3 128 96 32 32 564 495 32 32
8	Object Hash:	7 12 1 0 0 3 96 96 32 32 203 422 32 32
9	Object Hash:	7 12 1 0 0 3 32 96 32 32 360 413 32 32
10	Object Hash:	6 12 1 0 0 -5 224 0 32 32 554 289 32 32
11	Object Hash:	6 12 1 0 0 -5 64 0 32 32 554 344 32 32
12	Object Hash:	6 12 1 0 0 -5 128 0 32 32 554 399 32 32
13	Object Hash:	8 12 1 0 0 5 128 128 32 32 367 348 32 32
14	Object Hash:	6 12 1 0 0 -5 224 0 32 32 554 454 32 32
15	Object Hash:	6 12 1 0 0 -5 64 0 32 32 554 504 32 32
16	Object Hash:	7 12 1 0 0 3 32 96 32 32 359 266 32 32
17	Object Hash:	6 12 1 0 0 -5 160 0 32 32 517 554 32 32

Mothership Hash: 11|2|1|6|4|128|192|64|128|616|10|64|128

\*\*\*\*\*

## 6) Referencias:

- Wikimedia Foundation, Inc. Artículos: SDL , Collision detection, BTree, Hash function, Hash Table, A\*. *Recursos web tomados el 15 de mayo de 2008 de <http://en.wikipedia.org>*
- Eclipse Foundation. Eclipse documentation. *Recurso web tomado el 16 de mayo de <http://www.eclipse.org/documentation/>*
- CPP Reference Team. CPP Reference website. *Recurso web tomado el 17 de mayo de 2008 de <http://www.cppreference.com/>*
- YoLinux Community Site. C++ STL Tutorial. *Recurso web tomado el 22 de mayo de 2008 de <http://www.yolinux.com/TUTORIALS/LinuxTutorialC++STL.html>*
- Lazy Foo Productions. Beginning Game Programming. *Recurso web tomado el 24 de mayo de 2008 de [http://lazyfoo.net/SDL\\_tutorials/](http://lazyfoo.net/SDL_tutorials/)*
- SDL Project. SDL documentation. *Recurso web tomado el 19 de mayo de 2008 de <http://www.libsdl.org/cgi/docwiki.cgi>*
- Lester, Patrick. A\* Pathfinding for Beginners. *Recurso web tomado el 20 de mayo de 2008 de <http://www.policyalmanac.org/games/aStarTutorial.htm>*
- Bruno R. Preiss, P.Eng. Data Structures and Algorithms with Object-Oriented Design Patterns in C++ [1999]. *Web book tomado el 24 de mayo de 2008 de <http://www.brpreiss.com/books/opus4/index.html>*
- Salinas Caro, Patricio. Tutorial de UML. *Recurso web tomado el 28 de mayo 2008 de <http://www.dcc.uchile.cl/~psalinas/uml/>*