UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Ingeniería en Informática

Proyecto de Fin de Carrera

BlackRose: Un modelo de razonamiento con incertidumbre en juegos de estrategia

Autor: Sergio Núñez Covarrubias

Director: Carlos Linares López

Noviembre, 2009

A Rosalía Framil, porque a veces los
sueños se convierten en realidad

Agradecimientos

A mi niña, por su paciencia, comprensión y apoyo incondicional que han convertido un sueño en una realidad. Gracias por darme la mano cada vez que he tropezado con una piedra del camino, gracias por todo.

A los mejores padres que uno puede tener, por su dedicación, trabajo y sacrificio, porque sin ellos esto no hubiese sido posible.

A mi hermano, por ayudarme cada vez que lo he necesitado, por compartir conmigo sus metas y por darme su apoyo en cada momento de mi vida.

A José Framil y Remedios Carpeño, por acogerme y apoyarme durante todo este tiempo, por hacer que me sienta en casa.

A mis amigos, especialmente a Carlos Sánchez, por estar conmigo, por entenderme, por ser amigo, confidente, y en definitiva por estar ahí a cada paso.

A Paola López, Pilar Abad y Raquel Abad, por acompañarme desde el principio y darme todo el apoyo que un amigo puede necesitar.

A mis compañeros, por compartir conmigo mis días de Universidad, por esas risas, por dar margen a mis agobios, y por supuesto, a mis errores.

A mi tutor, Carlos Linares López por su tiempo, dedicación y comprensión, y por encima de todo, por su confianza.

Simplemente, GRACIAS.

Tabla de contenido

1. Introducción 15
2. Estado de la cuestión 19
2.1. ¿Qué es un juego de estrategia en tiempo real? 19
2.1.1. Subgéneros de juegos RTS 20
2.1.2. Primeros juegos RTS 22
2.1.3. Actualidad de los juegos RTS 23
2.1.4. Juegos RTS no comerciales 24
2.2. Mundo ORTS 26
2.2.1. Edificios Terrans 27
2.2.2. Unidades Terrans 30
2.3. Torneo ORTS 33
2.3.1. Primer juego 33
2.3.2. Segundo juego 34
2.3.3. Tercer juego 35
2.3.4. Cuarto juego 36
2.4. Modelos de toma de decisiones 37
2.4.1. Reglas 38
2.4.2. Redes Bayesianas 39
2.4.3. Lógica difusa 42
2.5. Consideraciones finales 44
3. Objetivos 47
4. Desarrollo 51
4.1. Introducción 51
4.2. Análisis 52
4.2.1. Casos de uso 52
4.2.2. Requisitos de usuario 54
4.3. Capa Táctica 56
4.3.1. Estrategias para el juego Starcraft 56
4.3.2. Proceso de abstracción y adaptación de estrategias 60
4.3.3. Filtrado de estrategias 65
4.3.4. Estudio exhaustivo de las estrategias resultantes 69
4.4. Capa General 75
4.4.1. Red Bayesiana de ataque 76
4.4.2. Red Bayesiana de defensa 79
4.4.3. Red Bayesiana de construcción de *barracks* 83
4.4.4. Red Bayesiana de construcción de *Factory* 88
4.4.5. Red Bayesiana de entrenamiento de w*orkers* 91
4.4.6. Red Bayesiana de entrenamiento de unidades militares 94
4.4.7. Red Bayesiana para la construcción de nuevas bases 99

4.5. Capa Estructuras ... 103
4.5.1. Ataque, defensa y creación de nuevas bases ... 104
4.5.2. Construcción de edificios ... 106
4.5.3. Entrenamiento de unidades ... 107
4.5.4. Gather y scouting ... 108
4.6. Consideraciones finales ... 109
5. Resultados ... 115
5.1. Red de ataque ... 116
5.2. Red de defensa ... 117
5.3. Red para fundar nuevos asentamientos ... 119
5.4. Red de construcción de *barracks* ... 120
5.5. Red de construcción de *factory* ... 122
5.6. Redes de entrenamiento de unidades ... 123
5.7. Análisis de rendimiento ... 127
5.8. Análisis de efectividad ... 129
5.9. Torneo ORTS 2009 ... 130
6. Conclusiones ... 135
7. Líneas futuras ... 139
8. Bibliografía ... 143
9. Anexo A: Diagrama de clases ... 147
10. Anexo B: Descripción de los módulos utilizados e implementados ... 153
11. Anexo C: Tutorial de iniciación a ORTS ... 161
11.1. Introducción ... 161
11.2. Instalación de ORTS en Mac OS X ... 162
11.2.1. Instalación de las librerías necesarias ... 162
11.2.2. Instalación del ORTS ... 165
11.3. ¿Cómo ejecutar ORTS? ... 171
11.4. Programación de un cliente ORTS ... 172
11.4.1. Implementación del escenario ... 173
11.4.2. Implementación del primer cliente ... 175
11.4.3. Implementación del segundo cliente ... 179
11.5. Apéndice: Acrónimos y definiciones ... 180
11.6. ANEXO ... 182
11.6.1. Código de static void generate_game4_map (Stringstream &world) ... 182

Índice de tablas

Tabla 1: Requisito UR - 001 54
Tabla 2: Requisito UR - 002 54
Tabla 3: Requisito UR - 003 54
Tabla 4: Requisito UR - 004 55
Tabla 5: Requisito UR – 005 55
Tabla 6: Requisito UR – 006 55
Tabla 7: Requisito UR – 007 55
Tabla 8: Requisito UR – 008 55
Tabla 9: Requisito UR – 009 55
Tabla 10: Resultado del proceso de adaptación y abstracción 65
Tabla 11: Resultados de la red bayesiana para ataque 79
Tabla 12: Resultados de la red bayesiana para defensa 83
Tabla 13: Resultados de la red bayesiana para construcción de *Barracks* 88
Tabla 14: Resultados de la red bayesiana para construcción de *Factory* 91
Tabla 15: Resultados de la red bayesiana para entrenamiento de *Workers* 94
Tabla 16: Resultados de la red bayesiana para entrenamiento de unidades militares 98
Tabla 17: Resultados de la red bayesiana para construcción de nuevas bases 102
Tabla 18: Datos estadísticos de los resultados obtenidos 129
Tabla 19: Resultados del torneo 129
Tabla 20: Resultados torneo ORTS 2009 130
Tabla 21: Definiciones de los acrónimos 180

Índice de ilustraciones

Ilustración 1: De izquierda a derecha, Command & Conquer: Red Alert 3, ORTS 20
Ilustración 2: Close combat II: A Bridge Too Far 21
Ilustración 3: Comandos 2: Men of Courage 21
Ilustración 4: De izquierda a derecha, Dune II, Command & Conquer: Red Alert y Starcraft 22
Ilustración 5: Interfaz de Halo Wars 23
Ilustración 6: Situación inicial del primer juego 34
Ilustración 7: Situación inicial del segundo juego 35
Ilustración 8: Situación inicial del tercer juego 36
Ilustración 9: Situación inicial del cuarto juego 36
Ilustración 10: Ejemplo de red bayesiana 40
Ilustración 11: Resultado de moralizar un grafo 40
Ilustración 12: Resultado de la triangulación del grafo moralizado 41
Ilustración 13: Lista de *cliques* obtenida del grafo obtenido de la triangulación 41
Ilustración 14: Árbol de *cliques* obtenido aplicando MST 41
Ilustración 15: Ejemplo de variable borrosa 43
Ilustración 16: Estructura del cliente 51
Ilustración 17: Casos de uso del cliente 53
Ilustración 18: Casos de uso del servidor 53
Ilustración 19: Situación de una partida sin límite de tiempo 67
Ilustración 20: Topología de la red Bayesiana para ataque 76
Ilustración 21: DPC del nodo *Primary Condition* de la red Bayesiana para ataque 78
Ilustración 22: DPC del nodo *Attack* de la red Bayesiana para ataque 78
Ilustración 23: Topología de la red Bayesiana para defensa 80
Ilustración 24: DPC del nodo *Primary condition* de la red Bayesiana para defensa 82
Ilustración 25: DPC del nodo *Exception* de la red Bayesiana para defensa 82
Ilustración 26: DPC del nodo *Defense* de la red Bayesiana para defensa 82
Ilustración 27: Topología de la red Bayesiana para construcción de *Barracks* 84
Ilustración 28: DPC del nodo *Primary condition* de la red Bayesiana para construcción de *Barracks* 86
Ilustración 29: DPC del nodo *Base Attack* de la red Bayesiana para construcción de *Barracks* 87
Ilustración 30: DPC del nodo *Precondition* de la red Bayesiana para construcción de *Barracks* 87
Ilustración 31: DPC del nodo *Building Barracks* de la red Bayesiana para construcción de *Barracks* 87
Ilustración 32: DPC del nodo *Building Barracks* de la red Bayesiana para construcción de *Barracks* 87
Ilustración 33: Topología de la red Bayesiana para construcción de *Factory* 89
Ilustración 34: DPC del nodo *Primary condition* de la red Bayesiana para construcción de *Factory* 90
Ilustración 35: DPC del nodo *Precondition* de la red Bayesiana para construcción de *Factory* 90
Ilustración 36: DPC del nodo *Building factory* de la red Bayesiana para construcción de *Factory* 90
Ilustración 37: Topología de la red Bayesiana para entrenamiento de *Workers* 92
Ilustración 38: DPC del nodo *Precondition* de la red Bayesiana para entrenamiento de *Workers* 93
Ilustración 39: DPC del nodo *Training worker* de la red Bayesiana para entrenamiento de *Workers* 94

Ilustración 40: Topología de la red Bayesiana para entrenamiento de unidades militares 95
Ilustración 41: DPC del nodo *Primary condition* de la red Bayesiana para entrenamiento de unidades militares 97
Ilustración 42: DPC del nodo *Precondition* de la red Bayesiana para entrenamiento de unidades militares 97
Ilustración 43: DPC del nodo *Training unit* de la red Bayesiana para entrenamiento de unidades militares 98
Ilustración 44: Topología de la red Bayesiana para construcción de nuevas bases 99
Ilustración 45: DPC del nodo *Primary condition* de la red Bayesiana para construcción de nuevas bases 101
Ilustración 46: DPC del nodo *Secondary condition* de la red Bayesiana para construcción de nuevas bases 101
Ilustración 47: DPC del nodo *Precondition* de la red Bayesiana para construcción de nuevas bases 101
Ilustración 48: DPC del nodo *New base* de la red Bayesiana para construcción de nuevas bases 102
Ilustración 49: Diseño de la capa Estructuras 103
Ilustración 50: Evolución del tiempo de respuesta de la red de ataque (cliente 1) 116
Ilustración 51: Evolución del tiempo de respuesta de la red de ataque (cliente 2) 117
Ilustración 52: Evolución del tiempo de respuesta de la red de defensa (cliente 1) 118
Ilustración 53: Evolución del tiempo de respuesta de la red de defensa (cliente 2) 118
Ilustración 54: Evolución del tiempo de respuesta de la red de fundar nuevos asentamientos (cliente 1) 119
Ilustración 55: Evolución del tiempo de respuesta de la red de fundar nuevos asentamientos (cliente 2) 120
Ilustración 56: Evolución del tiempo de respuesta de la red de construcción del edificio *barracks* (cliente 1) 121
Ilustración 57: Evolución del tiempo de respuesta de la red de construcción del edificio *barracks* (cliente 2) 121
Ilustración 58: Evolución del tiempo de respuesta de la red de construcción del edificio *factory* (cliente 1) 122
Ilustración 59: Evolución del tiempo de respuesta de la red de construcción del edificio *factory* (cliente 2) 123
Ilustración 60: Evolución del tiempo de respuesta de la red de entrenamiento de la unidad *worker* (cliente 1) 124
Ilustración 61: Evolución del tiempo de respuesta de la red de entrenamiento de la unidad *worker* (cliente 2) 124
Ilustración 62: Evolución del tiempo de respuesta de la red de entrenamiento de la unidad *marine* (cliente 1) 125
Ilustración 63: Evolución del tiempo de respuesta de la red de entrenamiento de la unidad *marine* (cliente 2) 126

Ilustración 64: Evolución del tiempo de respuesta de la red de entrenamiento de la unidad *tank* (cliente 1) 126
Ilustración 65: Evolución del tiempo de respuesta de la red de entrenamiento de la unidad *tank* (cliente 2) 127
Ilustración 66: Evolución del tiempo de ejecución del método *compute_actions()* (cliente 1) 128
Ilustración 67: Evolución del tiempo de ejecución del método *compute_actions()* (cliente 2) 128
Ilustración 69: Diagrama de clases 147
Ilustración 70: Comando para instalar la librería Glew 164
Ilustración 71: Comando locate glew 165
Ilustración 72: Actualización del módulo ORTS 165
Ilustración 73: Actualización del módulo ORTS terminada 166
Ilustración 74: Actualización del módulo ORTS_DATA 166
Ilustración 75: Actualización del módulo ORTS_DATA terminada 166
Ilustración 76: Makefile original 166
Ilustración 77: Makefile modificado 167
Ilustración 78: sh_array.h original 167
Ilustración 79: sh_array.h modificado 167
Ilustración 80: sr_bv_math.cpp original 167
Ilustración 81: sr_bv_math.cpp modificado 168
Ilustración 82: orts_main.c original 168
Ilustración 83: ortsg_main.c original 168
Ilustración 84: sampleai_main.c original 168
Ilustración 85: orts_main.c modificado 169
Ilustración 86: ortsg_main.c modificado 169
Ilustración 87: sampleai_main.c modificado 169
Ilustración 88: Inicializar la compilación de ORTS 170
Ilustración 89: Compilar ORTS 170
Ilustración 90: Estructura del mapa del juego 181

Capítulo 1

Introducción

1. Introducción

Decía McLuhan (1973) que el afán puesto en el empleo de la técnica no nos ha dejado tiempo para considerar sus implicaciones. Hay que remontarse a 1958 para poder entender no sólo el concepto de lo que hoy en día se conoce como *videojuego*, sino también su importancia, la esencia que lo constituye y lo que le ha llevado a ser considerado en el siglo XXI (además de instrumento de entretenimiento) un arte: el arte de los videojuegos.

Desgraciadamente, los grandes avances tecnológicos están pensados en su gran mayoría para una aplicación puramente militar. Fue sólo después de dos grandes guerras mundiales, cuando numerosas empresas retomaron los conocimientos que habían sido aplicados por entonces en un increíble desarrollo balístico y de encriptación de mensajes, para encaminarlos a la creación de un nuevo mercado de componentes electrónicos orientados al consumo doméstico, y dando lugar a una revolución tecnológica impulsada en los años 70 y 80.

La base de los videojuegos viene impulsada por el nacimiento de la primera computadora, ENIAC, en 1946, que da lugar 12 años más tarde a la creación de lo que muchos consideran el primer videojuego: *Tennis for Two*, resultado de las investigaciones de Willy Higgimbothan, jefe de Diseño de Instrumentación de Brookhaven National.

A raíz de ahí, la evolución fue espectacular: desde el primer videojuego creado por Steve Russel (*Spacewar*), pasando por Nolan Bushnell y su *Computer Space* (que puede considerarse el primer arcade), hasta lo que hoy conocemos como el arte de los videojuegos.

Muchos aficionados recurren a Scott McCloud para definir los videojuegos en una generalización del proceso artístico en 6 pasos: propósito, forma, estilo, estructura, destreza y superficie; y consideran que la lacra de los mismos reside en la falta de su armonía.

Desde el ámbito de la informática, podemos considerar los videojuegos como un programa que está diseñado para ejecutarse en dispositivos dotados de una pantalla y un conjunto de controles, mediante los cuales el jugador puede comunicarse con el sistema. Su estructura está compuesta principalmente de cuatro módulos: apartado gráfico, sistema de control, inteligencia artificial (IA) y lo que se conoce como apartado sonoro.

El mundo de los videojuegos evoluciona continuamente y de una manera asombrosa, evolución que depende en su gran mayoría de mucho esfuerzo e inversión.

Este proyecto le dedica básicamente su atención al caso de la IA, aspecto invisible para la *cuasi* totalidad de jugadores, pese al sacrificio que supone lograr una IA trabajada y depurada.

Analizando la IA ofrecida en videojuegos actuales [7], se puede comprobar que los adversarios siguen rutinas prefijadas y no suelen aprender de la estrategia aplicada por el usuario. Además, hay casos dónde se recurre a las trampas para lograr aumentar la dificultad del videojuego (inteligencia de los oponentes), como tener una puntería casi perfecta en los juegos de acción en primera persona, ver toda la situación actual en los juegos de estrategia en tiempo real y conducir quebrantando las leyes de la física en los juegos de carreras.

Los juegos que ofrecen un mayor nivel de IA [8], son los juegos de estrategia en tiempo real; aunque también es cierto que el dominio en el que se desarrollan, se considera o puede considerarse, más sencillo que el de otros géneros, como la acción en primera persona.

A día de hoy, existen muchas técnicas de IA que se podrían aplicar a los videojuegos, pero por unas u otras razones, no se utilizan. Esto es una de las motivaciones de este proyecto, el cual va a implementar técnicas no comerciales en un juego de estrategia en tiempo real (ORTS).

Se pretende así estudiar el rendimiento de una de estas técnicas con el objetivo de determinar su eficiencia en los juegos comerciales.

Capítulo 2

Estado de la cuestión

2. Estado de la cuestión

En este apartado, se van a comentar aspectos relacionados y pertenecientes al dominio del proyecto, con el objetivo de facilitar la comprensión del desarrollo del mismo.

2.1. ¿Qué es un juego de estrategia en tiempo real?

Es aquel en el que el tiempo transcurre de forma continua, sin pausas. Además, se requiere por parte del jugador habilidad en la forma de pensar y capacidad de planificación para conseguir la victoria.

Normalmente, el jugador al inicio de la partida sólo puede observar la zona del mundo en la que se encuentra, pero puede explorar el resto del escenario con el objetivo de tener una visión absoluta. Desde un tiempo atrás, este tipo de juegos implementan la característica denominada *fog of war* (niebla de guerra), la cual consiste en que las zonas del mundo que han sido descubiertas son visibles, pero si no hay ningún objeto del jugador en éllas, no se puede ver el estado actual de las mismas.

Hay otro tipo de juegos de estrategia, los llamados estrategia por turnos, dónde la acción no se desarrolla de forma continua, sino que se divide en turnos. En cada uno, el jugador al que le corresponde el turno tiene que realizar sus acciones (movimientos, ataques, etc.). Cuando termina, se actualiza la situación del juego y le corresponde al siguiente jugador realizar sus acciones.

Entre estos dos tipos de juegos de estrategia hay mucho debate, debido a las posturas que toman los amantes de cada uno. Por un lado, los defensores de la estrategia por turnos alegan que el juego en tiempo real es de menor profundidad estratégica. Y por el otro, los puristas del tiempo real consideran que los juegos por turnos tienden a ser muy lentos y aburridos.

Entre estas dos posturas de ver cómo se debe desarrollar un juego de estrategia, hay una tercera variante, *el tiempo real regulable*, en el que los jugadores pueden realizar acciones mientras el tiempo está parado. En comparación con los juegos en tiempo real convencionales, se tiene menos acción pero ofrecen una mayor cantidad y variedad de opciones. En este tipo de juegos, el tiempo se para por determinas condiciones preestablecidas, pero hay algunos en los que es posible configurar las condiciones en las que el juego se parará.

Estos tres tipos de estrategia tienen representación comercial en los videojuegos. En el caso del tiempo real, entre las sagas más conocidas se encuentra Warcraft y Starcraft, ambos desarrollados por Blizzard y Age of Empires, desarrollado por Microsoft Game Studios. Por parte de la estrategia por turnos, la saga más destacada es Civilization (Firaxis games). Y por último, los juegos más conocidos de estrategia en tiempo real regulable son los de Paradox Interactive.

2.1.1. Subgéneros de juegos RTS

La estrategia en tiempo real tiene muchos subgéneros y gran parte de ellos son modificaciones, uniones[1] o derivaciones de otros, por lo que sólo se van a comentar brevemente los que se consideran más importantes o representativos en el ámbito del proyecto.

- **Build and Battle (B&B):** El objetivo consiste en recolectar todo tipo de recursos, construir con ellos edificios, trabajadores, guerreros, carros de combate, etc. y por último la batalla. El comercio, la diplomacia y la política suelen destacar por su ausencia.

Ilustración 1: De izquierda a derecha, Command & Conquer: Red Alert 3, ORTS

Este tipo de juegos se suele relacionar directamente con el género padre, estrategia en tiempo real, aunque en verdad no deja de ser un subgénero. Esto se debe a que es el más conocido y el que mayor número de ventas consigue. Tanto ORTS como Command & Conquer[2] (Westwood Studios[3], Electronic Arts) son claros ejemplos de B&B (ambos pueden verse en la *ilustración 1*).

1 No sólo con subgéneros de juegos de estrategia en tiempo real, sino que hay casos en los que se unen con otros géneros, por ejemplo, con los juegos de acción.

2 La saga Command & Conquer fue creada por Westwood Studios y adquirida por Electronic Arts después del fallido Command & Conquer: Renegade.

3 Westwood Studios destacó desde el principio en los juegos de estrategia en tiempo real, ya que son los creadores de Dune II, Command & Conquer y Lands of Lore, entre otros.

- **Wargames:** Aunque la estrategia en tiempo real tenga presencia en este subgénero con juegos más tácticos (Close combat, Atomic, *ilustración 2*), la realidad es que la mayoría de juegos suelen ser estratégicos y por turnos.

  Este tipo de juegos simulan una batalla o campaña bélica de forma abstracta, dónde el objetivo es alcanzar la victoria con los recursos disponibles. La trama principal es la batalla, por lo que no suele ser necesario recolectar recursos ni construir edificios.

Ilustración 2: Close combat II: A Bridge Too Far

- **Tácticos hombre a hombre:** Son juegos tácticos, pero aplicar una buena estrategia es imprescindible para lograr el objetivo. No suele haber ni recolección de recursos, ni construcción de edificios, ni tampoco se suele contar con dos unidades iguales. Suelen estar ambientados en guerras reales, aunque no es imprescindible, a diferencia de los *wargames*.

  Cada unidad tiene una serie de habilidades distintas al resto y por ello, se suelen aplicar órdenes individuales para ir logrando objetivos individuales[4] (normalmente definidos por el usuario), con el fin de alcanzar la victoria. Un buen ejemplo de este subgénero es la saga Comandos (Pyro Studios, *ilustración 3*).

Ilustración 3: Comandos 2: Men of Courage

[4] Los objetivos individuales definidos por el usuario a los que se hace referencia no son objetivos identificados por el juego y tampoco hay que comunicárselos al mismo, sino que son los pasos que el usuario cree que tiene que realizar para alcanzar la victoria.

2.1.2. Primeros juegos RTS

El primer juego de estrategia en tiempo real fue Dune II, desarrollado por Westwood Studios y publicado por Virgin Interactive en el año 1992. Dune II está basado en el mundo creado por Frank Herbert y fue un juego revolucionario, significó lo que DOOM (iD Software) para los juegos en primera persona. Sus gráficos eran en 2D y su interfaz era muy incómoda (puede observarse en la *ilustración 4*), pero sirvió para dar el primer paso en este género de juegos y convertirse así en el precursor de otros juegos como Starcraft (Blizzard), Age of Empires (Microsoft), etc.

El siguiente juego que vió la luz fue Warcraft: Orcs & Humans a manos de Blizzard, en el año 1994. La principal mejora que ofrecía respecto al anterior, era la jugabilidad y la interfaz. Como su nombre indica, sólo había dos razas, los orcos y los humanos, a diferencia de Dune II, en el que había tres razas disponibles, los Atreides, los Harkonnen y los ordos.

En 1995 nace Command & Conquer con unos gráficos mejorados (respecto a los dos anteriores juegos) y un punto de vista mas realista, ya que estaba basado en la guerra. Pocos meses después, se publicó la secuela de Warcraft, Warcraft II: Tides of Darkness, el cuál supero por mucho a su predecesor. Actualmente, hay muchas personas que siguen jugando a este juego por Internet, lo cual indica la gran calidad del juego.

Al año siguiente, en 1996, se publicó la secuela del primer Command & Conquer, Command & Conquer: Red Alert (*ilustración 4*), aunque no fue hasta en 1997 cuando hubo un cambio cualitativo con Total Annihilation (Cavedog) y Dark Reign (Auran) y se culminó en 1998, con el gran juego de Blizzard, Starcraft (*ilustración 4*). Este juego generó aficionados en todo el mundo y al igual que el anterior título de Blizzard, se siguen disputando partidas por Internet e incluso torneos. Fue denominado como obra maestra por gran parte de los aficionados y analistas de juegos.

Ilustración 4: De izquierda a derecha, Dune II, Command & Conquer: Red Alert y Starcraft

En la ilustración anterior, se puede observar a simple vista la evolución de la interfaz y la de los gráficos.

2.1.3. Actualidad de los juegos RTS

Históricamente, los juegos de estrategia han sido desarrollados para ordenadores personales, debido a las ventajas que ofrece el uso del ratón. Gracias a dicho periférico, el control del juego resulta sencillo e intuitivo. Por esta razón, se han realizado pocas conversiones[5] y desarrollos para las videoconsolas. Una medida que se tomó para intentar fomentar los juegos RTS en las videoconsolas, fue el desarrollo de ratones para las mismas.

Con el paso del tiempo, el número de juegos RTS para videoconsolas ha crecido. Un ejemplo es el caso de la saga Age Of Empires en la Nintendo DS. En este caso, el control es sencillo debido a la pantalla táctil que ofrece el dispositivo.

Además, en 2009 se ha publicado el juego de estrategia en tiempo real Halo Wars (Ensemble Studios, *ilustración 5*), desarrollado exclusivamente para la videoconsola Xbox 360 de Microsoft. Para la realización del juego, se ha creado un elaborado sistema, con el objetivo de convertir el mando de control en la herramienta perfecta para dar órdenes a las tropas. Si se realiza un click sobre un soldado, se elegirá a su batallón y si se realiza un doble click, se permite dibujar con el *stick* una figura que encierre a un grupo de soldados concreto del batallón. Una vez que se tiene el control de una unidad, bastará con pulsar un botón para abrir un versátil menú con todo tipo de acciones.

Ilustración 5: Interfaz de Halo Wars

[5] Se refiere a la creación de un juego ya existente en otra plataforma. Por ejemplo, Warcarft 2 fue desarrollado para los ordenadores personales y posteriormente se realizó una conversión a la videoconsola PlayStation.

En cuanto al ordenador personal, actualmente hay varios proyectos en desarrollo, entre los cuales se encuentra el esperado Starcraft 2. En él, se incorporarán nuevas unidades a las clásicas de Starcraft, que reaparecerán con nuevas habilidades. Además, lo más novedoso será el aspecto gráfico del juego, ya que el motor 3D con el que está trabajando Blizzard pondrá en escena unidades que se apreciarán con todo lujo de detalles. Otra novedad es la integración de Havok[6], el cual ofrece un comportamiento físico milimétrico en toda clase de interacciones de las unidades con el escenario o con otras unidades.

La base del juego seguirá siendo la supervivencia, ya que el usuario tendrá que ponerse al frente de un ejército que debe combatir por la gestión de recursos y el dominio de la mayor parte del escenario, con la ayuda de las unidades disponibles.

2.1.4. Juegos RTS no comerciales

Hay juegos que se desarrollan sin carácter lucrativo, ya sea con objetivos de investigación o simplemente por *hobby*. La mayoría de este tipo de juegos son de código abierto, lo que significa que cualquier persona puede modificarlo a su gusto, mejorarlo y por supuesto jugarlo.

También hay que tener en cuenta los juegos comerciales, ya que en algunos casos, pasados unos años liberan el código fuente.

Algunos ejemplos de estos videojuegos son los siguientes:

- **ORTS:** Es el que se va a utilizar para el desarrollo del presente proyecto (véase 2.2 Mundo ORTS).

- **Warzone 2100[7]:** En un juego RTS desarrollado pon Pumpkin Studios y publicado por Eidos Interactive en 1999 para PC y PlayStation. El 6 de Diciembre de 2004 el código fuente fue liberado bajo la licencia GPL[8].

6 Havok Game Dynamics SDK, es un motor físico utilizado en videojuegos. Recrea las interacciones entre objetos y personajes y además, detecta colisiones, gravedad, masa y velocidad en tiempo real, llegando a recrear ambientes realistas y naturales.

7 http://wz2100.net/

8 Es una licencia creada por Free Software Foundation y está orientada principalmente a proteger la libre distribución, modificación y uso de software.

Es similar al videojuego Earth 2150 (TopWare Interactive) en muchos aspectos, aunque contiene otros que son innovaciones propias, como es el caso de las tecnologías radar, tecnologías anti baterías, etc. Está ambientado en los restos de una sociedad que se ha destruido a sí misma alrededor del año 2085. Los jugadores deberán construir una base central, factorías, armas, tanques y otras unidades de combate. Una vez que se disponga del ejercito necesario, el objetivo es único, destruir los ejércitos enemigos y tomar el control del planeta. Cuenta con una gran cantidad de tecnologías y unidades con vistas 3D.

- **Spring**[9]**:** También conocido como TA Spring o Total: Spring, era un juego comercial cuyo código fue liberado por Swedish Yankspanker con la intención de recrear la experiencia de juego vivida por muchos jugadores con Total Annihilation.

  El juego no ha sido desarrollado en exclusiva por Swedish Yankspanker, ya que existe una comunidad volcada en el proyecto, en el que han colaborado traduciendo textos, modificando unidades de combate y creando mapas. Lo que nació con el objetivo de ampliar un juego comercial se ha convertido en un juego nuevo y diferente. Está enfocado hacia las partidas multijugador a través de Internet y recientemente hacia un modo historia, ya que se han añadido misiones para un jugador, mejorando la IA de las unidades controladas por la máquina.

9 http://spring.clan-sy.com/

2.2. Mundo ORTS

A día de hoy, ORTS ofrece la posibilidad de elegir entre dos razas (*Terrans* y *Pantheon*) y entre los cuatro escenarios de la competición. Los escenarios son independientes de las reglas que se le aplican, por lo que se puede elegir un escenario de la competición sin la necesidad de aplicar las mismas reglas que se aplican en la misma. Todo escenario tiene unas reglas básicas, pero lo que se quiere matizar es que se puede jugar al escenario 3, por ejemplo, pudiendo construir todos los edificios y unidades disponibles en ORTS y no sólo los específico del escenario en cuestión de la competición.

En el mundo, hay dos clases de recursos para recolectar, minerales y gas procedente de los géiseres. Para la recolección de este último, es necesario la construcción de un edificio específico.

Además, hay que controlar la población. Para ello, se utiliza una unidad numérica que indica el número de habitantes que se pueden mantener. Cada unidad tiene asignado el número de habitantes que consume, ya que no todas las unidades consumen lo mismo. Por ejemplo, en el caso de la raza *Terrans*, el *worker* consume por un habitante, mientras que un *hoverbike* consume por dos. Hay edificios por parte de las dos razas destinados a aumentar el número de habitantes que se pueden mantener.

Cada unidad y edificio tiene asignada una cantidad de puntos de salud. Estos puntos simulan la vida que tiene el elemento y cuando estos puntos se acaban, el elemento muere. Adicionalmente, hay unidades y edificios que tienen asignados una cantidad de puntos de maná. El maná se utiliza para realizar acciones especiales y se va recuperando con el tiempo.

Todo edificio y unidad tiene una armadura que diminuye el daño que le provoca un ataque enemigo. Esta disminución depende de la armadura, ya que en el caso de las unidades, no todas tienen la misma. La armadura se define por el número de puntos de salud que evita al elemento que protege en cada ataque.

Para tener un conocimiento completo sobre el mundo ORTS, es necesario conocer todos los elementos de las dos razas. Debido a que el proyecto sólo se va a centrar en la raza *Terrans*, se va a dejar el estudio detallado de la raza *Pantheon* para futuros proyectos.

2.2.1. Edificios Terrans

Los *Terrans* poseen principalmente doce edificios, los cuales tienen que ser construidos por la unidad *Worker*. Adicionalmente, hay edificios que pueden realizar pequeñas construcciones complementarias, en las que no tiene que intervenir ningún trabajador (del inglés, *worker*). Todos los edificios y construcciones de la raza *Terrans* se describen a continuación:

- **Control Center:** Es el edificio principal de los *Terrans*, dónde se almacenan los recursos recolectados. Este edificio incrementa el número de habitantes que se pueden mantener. Las principales acciones que permite realizar son las siguientes:
  - **Worker:** Crea un nuevo trabajador. Se pueden solicitar varias peticiones a la vez, formando una cola de creación de trabajadores.
  - **Build Nuke Silo:** Construye el edificio *Nuke Silo*, el cual complementa al *Control Center*. La principal acción que permite realizar es *Build Nuke*, que construye un arma nuclear.
  - **Build Comsat Station:** Construye el edificio *Comsat Station*, el cual complementa al *Control Center*. La principal acción que permite realizar es *Scan*, que permite ver una zona del mapa que aún no había sido explorada o que está cubierta por la niebla de guerra. Para ello, se requieren puntos de maná.
- **Supply Cache:** Edificio que incrementa el número de habitantes que se pueden mantener. Este edificio no permite realizar ninguna acción.
- **Barracks:** Es un cuartel que permite crear diferentes unidades terrestres, en el que se pueden solicitar varias peticiones a la vez, formando una cola de creación de unidades. Entre las acciones que permite realizar, destaca el entrenamiento de las unidades *Marine*, *Toaster*, *Medic* y *Spy*.
- **Engineering:** Es un edificio de ingeniería que permite realizar mejoras sobre los escudos y armas de la infantería. Dichas mejoras son *Infantry weapon upgrade* (permite

aumentar el daño que provocan las armas de infantería) e *Infantry Armor Upgrade* (permite aumentar la resistencia de las armaduras de infantería).

- **Extractor:** Es el edificio que permite la recolección de gas, por lo que debe ser construido sobre un géiser. Este edificio no permite realizar ninguna acción, sólo permite el paso a los trabajadores para que recolecten el gas.

- **Turret:** Es una torre de defensa que puede atacar a las unidades aéreas. La principal acción que permite realizar es *Attack*, la cual realiza un ataque sobre una determinada unidad.

- **Academy:** Es un edificio que permite realizar actualizaciones sobre determinadas unidades, las cuales son *Range upgrade* (aumenta el rango de acción de las armas de los marines, tanto para ataques a unidades terrestres como aéreas), *Medic mend ability* (habilita la acción *Mend* de los médicos), *Medic blind ability* (habilita la acción *Blind* de los médicos), *Medic Energy upgrade* (incrementa la cantidad de maná que pueden tener como máximo los médicos) e *Infantry adrenal boost* (habilita la acción *Combat Boost* de las unidades *Marine* y *Toaster*).

- **Bunker:** Edificio que permite la entrada a cuatro unidades terrestres como máximo, para poder atacar desde dentro a los enemigos que se acerquen. La principal acción que permite realizar es *Empty units*, la cual ordena a las unidades que estén en el interior del bunker salir del mismo.

- **Mech Bay:** Edificio que permite la construcción de unidades terrestres mecánicas, en el que se pueden solicitar varias peticiones a la vez, formando una cola de creación de unidades. Entre las acciones que permite realizar, destacan el entrenamiento de unidades (*Hoverbike*, *Tank* y *Mech)* y *Build Shop*, la cual construye el edificio *Shop*.

- **Shop:** Es una construcción que complementa al *Mech Bay*, y permite realizar actualizaciones sobre las unidades terrestres mecánicas, las cuales son *Tank Siege*

Upgrade (actualiza los tanques, permitiéndoles realizar las acciones *Anchor* y *Release), Mech Air Range Upgrade* (aumenta a la unidad *Mech* su rango de ataque sobre unidades aéreas), *Hover Bike Speed Upgrade* (aumenta la velocidad de desplazamiento de la unidad *Hover bike*) y *Hover Bike Land Mines* (habilita la acción *Place Mine* de la unidad *Hover bike*).

- **Armory:** Permite realizar diversas actualizaciones sobre las armaduras y armas de las unidades terrestres mecánicas y aéreas, las cuales son *Air Armor Upgrade* (permite aumentar la resistencia de las armaduras de las unidades aéreas), *Air Damage Upgrade* (permite aumentar el daño que provocan las armas de las unidades aéreas), *Ground Armor Upgrade* (permite aumentar la resistencia de las armaduras de las unidades terrestres mecánicas) y *Ground Damage Upgrade* (permite aumentar el daño que provocan las armas de las unidades terrestres mecánicas).

- **Air Base:** Es una base aérea en la que se pueden crear diversas unidades. Además, se pueden solicitar varias peticiones a la vez, formando una cola de creación de unidades. Entre las acciones que permite realizar, destacan el entrenamiento de unidades (*Striker*, *Destroyer*, *Dropship*, *FlyingLab* y *Raptor*) *y Build Tower*, la cual construye el edificio *Control Tower*.

- **Control Tower:** Es una construcción que complementa al edifico *Air Base* y permite realizar actualizaciones sobre las unidades *Striker*, las cuales son *Research Striker Cloak* (permite que la unidad *Striker* tenga la capacidad de volverse invisible) y *Research Striker Energy* (incrementa la cantidad de maná que pueden tener como máximo la unidad *Striker*).

- **Theoretical Lab:** Permite realizar actualizaciones sobre la unidad *Flying Lab*, además de poder construir el edificio *Physics Extension* o *Infiltration Extension*. Las actualizaciones son *Flying Lab EMP* (habilita la acción *EMP* de la unidad *Flying Lab*), *Flying Lab Irradiate* (habilita la acción *Irradiate* de la unidad *Flying Lab*) y *Flying Lab Energy* (incrementa la cantidad de maná que pueden tener como máximo la unidad *Flying Lab*).

- **Physic Extension:** Es una construcción que complementa al edifico *Theoretical Lab* y permite realizar actualizaciones sobre la unidad *Destroyer*, las cuales son *Research Destroyer Energy* (incrementa la cantidad de maná que pueden tener como máximo la unidad *Destroyer*) y *Research Destroyer Fire Wall* (habilita la acción *Fire Wall* de la unidad *Destroyer*).

- **Infiltration Extension:** Es una construcción que complementa al edificio *Theoretical Lab* y permite realizar actualizaciones sobre la unidad *Spy*, las cuales son *Spy Sight Upgrade* (aumenta el rango de visión de la unidad *Spy*), *Spy Energy Upgrade* (incrementa la cantidad de maná que pueden tener como máximo la unidad *Spy*), *Research Spy Lockdown* (habilita la acción *Lockdown* de la unidad *Spy*) y *Research Spy Cloak* (permite que la unidad *Spy* tenga la capacidad de volverse invisible).

Además de las acciones descritas para cada una de las construcciones, hay algunas que son comunes para parte de los edificios (*Control Center*, *Barracks*, etc.), como es el caso de *Liftoff* (permite al edificio levantarse del suelo; una vez que se encuentra en el aire, se puede desplazar a otro lugar), *Set Down* (cuando el edificio se encuentre en el aire, le permite regresar al suelo si la superficie sobre la que se encuentra está disponible.) y *Stop* (cancela la última solicitud recibida; si el proceso necesitaba recursos y no se ha llegado a completar, se devuelven los recursos solicitados).

2.2.2. Unidades Terrans

Los *Terrans* cuentan con doce unidades de ataque y una unidad de recolección y construcción, aunque dicha unidad también puede realizar ataques, su arma es muy poco efectiva. Las unidades con las que cuentan los *Terrans* son las siguientes:

- **Worker:** Unidad terrestre que forma parte de la infantería. Es el encargado de la construcción y la recolección. Las principales acciones que puede realizar el trabajador son *Gather Resource* (recolectar recursos, minerales o gas), *Return Resource* (llevar los recursos recolectados al *Control Center* indicado), *Repair* (reparar un edificio o una unidad mecánica), *Build* (construir un edificio entre los que no se encentran *Mech B*ay, *Armory*, *Air Base* y *Theoretical Lab*) y *Advanced Build* (construir un edificio avanzado, es decir, *Mech Bay*, *Armory*, *Air Base* o *Theoretical Lab*).

- **Marine:** Unidad terrestre que forma parte de la infantería. Tiene la capacidad de atacar a unidades terrestres y aéreas. Se entrena en el edificio *Barracks*. Las acciones que puede realizar el *marine* son las comunes, es decir, moverse, atacar, etc.

- **Toaster:** Unidad terrestre que forma parte de la infantería. Se entrena en el edificio *Barracks*. Sólo puede atacar a unidades terrestres. Las acciones que puede realizar el *toaster* son las comunes, es decir, moverse, atacar, etc.

- **Medic:** Unidad terrestre que forma parte de la infantería. Se entrena en el edificio *Barracks*. Las acciones más destacadas que puede realizar son *Heal* (recupera a una unidad amiga puntos de salud a cambio de maná), *Mend* (elimina la mayoría de los estados negativos de una determinada unidad, aunque hay estados que no se pueden eliminar y requiere el consumo de maná) y *Blind* (ciega a un enemigo, reduciendo su campo de visión al mínimo, y tiene un consumo alto de maná).

- **Spy:** Unidad terrestre que forma parte de la infantería y tiene la capacidad de atacar a unidades terrestres y aéreas. Se entrena en el edificio *Barracks*. Las acciones más características que puede realizar son *Cloak* (se vuelve invisible consumiendo maná), *Uncloak* (abandona el modo invisible), *Lockdown* (inutiliza una unidad mecánica durante sesenta segundos, consumiendo altas cantidades de maná) y *Mark for Nuke* (señaliza un punto del escenario dónde se va a aterrizar el arma nuclear construido en el edificio *Nuclear Silo*).

- **Hover Bike:** Unidad terrestre mecánica que se entrena en el edificio *Mech Bay* y sólo tiene la capacidad de atacar a unidades terrestres. Las acción más interesante que puede realizar la unidad es *Place Mine* (puede colocar tres minas terrestres de tal forma que persigan y se detonen contra las unidades enemigas cuando éstas entren en el rango de actuación de la mina).

- **Tank:** Unidad terrestre mecánica que se entrena en el edificio *Mech Bay* y sólo tiene la capacidad de realizar ataques a unidades terrestres. Las acciones distinguidas que puede realizar son *Anchor* (se ancla en el suelo para poder utilizar un arma más grande y

poderosa, aumentando así el daño que causa a las unidades enemigas) y *Release* (se libera del suelo y vuelve a utilizar su arma normal).

- **Mech:** Unidad terrestre mecánica que se entrena en el edificio *Mech Bay*. Las acciones que puede realizar el *mech* son las comunes, es decir, moverse, atacar, etc.

- **Striker:** Unidad aérea que se entrena en el edificio *Air Base*. Las acciones más relevantes que puede realizar la unidad son *Cloak* (se vuelve invisible consumiendo maná) y *Uncloack* (abandona el modo invisible).

- **Destroyer:** Unidad aérea que se entrena en el edificio *Air Base*. Las acciones más características que puede realizar la unidad son *Missile – Object* (lanza un misil a un determinado objetivo, consumiendo puntos de maná), *Missile – Ground* (lanza un misil a un determinado objetivo terrestre, consumiendo puntos de maná) y *Firewall* (lanza diez misiles a un determinado objetivo, consumiendo una cantidad elevada de maná).

- **Dropship:** Unidad aérea de transporte que se entrena en el edificio *Air Base*. Puede cargar con ocho unidades siempre y cuando sean del mismo jugador, móviles y terrestres. Las acciones más interesantes que puede realizar son *Load Passenger* (carga a una unidad terrestre) y *Unload All* (expulsa a todas las unidades que contenga).

- **Flying lab:** Unidad aérea que se entrena en el edificio *Air Base*. Las acciones más interesantes que puede realizar la unidad *Flying Lab* son *Irradiate* (causa daño a las unidades biológicas que se encuentren dentro de la nube que genera la acción), *EMP* (elimina todo el escudo y maná de las unidades que se encuentren dentro del área afectada por la acción) y *Wall* (crea un muro alrededor de una determinada unidad absorbiendo todo el daño que la estuviesen provocando).

- **Raptor:** Unidad aérea que se entrena en el edificio *Air Base*. Su arma primaria lanza un pequeño número de misiles simultáneos. Las acciones que puede realizar el *raptor* son las comunes, es decir, moverse, atacar, etc.

Hasta aquí se han descrito las acciones más características de cada unidad, pero no sólo pueden realizar las comentadas, sino que existe un conjunto de acciones que suelen ser comunes a la mayoría de unidades. Dichas acciones son *Move* (la unidad se mueve a un punto concreto del escenario), *Stop* (la unidad deja de realizar la acción que estaba desempeñando) y *Attack* (la unidad ataca a una unidad o edificio enemigo). Hay que tener en cuenta que estas acciones no las pueden realizar todas las unidades, por ejemplo, ni la unidad *Medic*, ni la unidad *Drop Ship*, ni la unidad *Flying Lab* pueden realizar ataques.

Además de estas acciones, existe otra que sólo la pueden realizar las unidades *Marine* y *Toaster*. Dicha acción en *Combat Boost* (la unidad pierde puntos de salud a cambio de duplicar su velocidad actual durante un pequeño periodo de tiempo).

Por último, hay un conjunto de acciones ligadas al edificio *Bunker*, por lo que no todas las unidades pueden realizarlas (sólo las unidades terrestres). Dichas acciones son *Bunker* (la unidad se introduce en un determinado *Bunker*) y *Exit* (la unidad abandona el *Bunker* en el que se encuentra).

2.3. Torneo ORTS

El torneo ORTS se divide en cuatro categorías, en cada una de las cuales se presenta un reto diferente. La topografía de cada escenario es prácticamente idéntica, la cual presenta pequeñas elevaciones aleatorias del terreno y pequeños obstáculos móviles, excepto en el cuarto escenario, cuyo terreno no presenta ninguna elevación.

2.3.1. Primer juego

El objetivo del primer escenario es la recolección del mayor número posible de minerales en diez minutos, sobre un terreno de 32 x 32 casillas. Al inicio del juego se dispone de un *Control Center* y de veinte unidades *Worker* colocadas cerca del mismo (como se puede ver en la *ilustración 6*). Sobre el escenario hay una gran cantidad de minerales para ser recolectados, pero sobre un mineral, sólo puede haber un máximo de cuatro *Workers* en el mismo instante de tiempo.

En el primer juego sólo está disponible la unidad *Worker*, la cual puede realizar las acciones *Move* (moverse a una determina coordenada del escenario, con la posibilidad de indicar la velocidad de movimiento del trabajador), *Stop* (parar de mover y de recolectar), *Gather* (recolectar minerales; como máximo, un trabajador sólo puede cargar con diez unidades de minerales y la velocidad de recolección es de un mineral por cada cuatro ciclos de juego) y *Return* (trasladar los minerales recolectados al *Control Center*).

Ilustración 6: Situación inicial del primer juego

Uno de los principales retos del escenario es la coordinación de movimientos de las unidades con el entorno y el resto de unidades para evitar colisiones. Además, es muy importante lograr una buena distribución de las unidades en los *clusters* de minerales.

2.3.2. Segundo juego

El segundo juego se desarrolla durante quince minutos, con el objetivo de destruir tantos edificios del oponente como sea posible. Inicialmente, sobre el terreno de 64 x 64 casillas, hay cinco *Control Centers* colocados aleatoriamente (pero simétricos), donde cada uno está acompañado de diez tanques (como puede observarse en la *ilustración 7*).

En esta ocasión, sólo está disponible la unidad *Tank*, la cual puede realizar las acciones *Move*, *Attack* (atacar a una determinada unidad enemiga) y *Stop* (parar de mover y de atacar).

En este escenario es muy importante contar con una buena estrategia, pero también hay que tener en cuenta los movimientos que esta realizando el oponente, ya que es muy difícil lograr una estrategia que sea capaz de ganar en la mayoría de los casos. Por ello, hay que tener en cuenta las unidades del rival y jugar con la incertidumbre del desarrollo de la partida.

Ilustración 7: Situación inicial del segundo juego

2.3.3. Tercer juego

Destruir todos los edificios del oponente en un tiempo máximo de veinte minutos es el objetivo de este escenario, con la dificultad añadida de la existencia de niebla de guerra en el terreno del juego, formado por 64 x 64 casillas. Cuando una partida comienza, cada jugador dispone de un *Control Center*, seis unidades *worker* y un conjunto de minerales colocados a poca distancia (como puede verse en la *ilustración 8*). Aproximadamente existen cuatro zonas de minerales colocadas aleatoriamente por el escenario.

Los objetos disponibles son los siguientes:

- **Worker:** La unidad está capacitada para realizar las acciones *Move*, *Stop, Attack*, *Gather*, *Return*, *Build Control Center* (construir un *Control Center*), *Build Barracks* (construir el edificio *Barracks*, con la condición de que exista un *Control Center* perteneciente al jugador) y *Build Factory* (construir el edificio *Factory*, con la condición de que exista un *Control Center* y *Barracks* pertenecientes al jugador).

- **Tank/Marine:** Las acciones que pueden realizar son *Move*, *Attack* y *Stop*.

- **Control Center/Barracks/Factory:** Pueden realizar dos acciones: entrenar una unidad (*worker*, *marine*, *tank* respectivamente), o cancelar el entrenamiento.

Este escenario es el más completo del torneo, ya que podría decirse que abarca el resto de escenarios y los amplia. El principal reto del escenario es conseguir una toma de decisiones coordinada y eficiente para lograr el objetivo. Hay que tener en cuenta, que este escenario es una simplificación de una partida real, pero que aún así se tienen que realizar todas las actividades (atacar, defender, scouting, etc.).

Ilustración 8: Situación inicial del tercer juego

2.3.4. Cuarto juego

Cada jugador debe destruir tantas unidades del oponente como sea posible durante cinco minutos, en un terreno de 64 x 48 casillas. Al inicio de cada partida, cada jugador cuenta con veinte unidades *Siege Tank* y cincuenta *marines* colocados aleatoriamente, ocupando un cuarto del mapa, extremo derecho o extremo izquierdo (como puede apreciarse en la *ilustración 9*). Todas las unidades están colocadas de forma diagonalmente simétrica respecto al oponente.

Los objetos que se permiten utilizar son los siguientes:

- **Marine:** Las acciones que puede desempeñar son *Move*, *Attack* y *Stop*.

- **Tank:** En este escenario, la unidad está capacitada para realizar las acciones *Move*, *Attack* (atacar a una determinada unidad enemiga; este tipo de ataque sólo está activo en el modo *Unsieged*), *Stop*, *Attack2* (crea una explosión de radio quince centrada en una coordenada concreta del escenario, repartiendo el daño; este tipo de ataque sólo está activo en el modo *Sieged*) y *Switch* (origina la transición entre los modos *Sieged*/*Unsieged*; no tiene efecto si ya se está transitando).

Ilustración 9: Situación inicial del cuarto juego

2.4. Modelos de toma de decisiones

La toma de decisiones es un campo grande e importante dentro de la Inteligencia Artificial, ya que para que una máquina sea inteligente debe poder tomar decisiones inteligentes, es decir, decisiones coordinadas (con otras relacionadas) de un modo eficiente.

Enseñar a una máquina a tomar decisiones no es una tarea sencilla, ya que no es nada fácil representar y/o tener en cuenta la incertidumbre existente a la hora de tomar una decisión. El problema no sólo recae en la incertidumbre, ya que para que una máquina sepa el estado actual, todos los valores deben ser cuantificables y en la vida real esto no suele ocurrir.

Por ejemplo, tomar la decisión de qué quieres comer en un restaurante es fácil para un ser humano, pero en absoluto para una máquina. En este ejemplo podemos observar los dos problemas mencionados. Por un lado, la máquina debe saber qué comida le gusta, que temperatura hace (no es igual una comida en verano que en invierno), cuanta hambre tiene, etc. En definitiva aspectos complejos de cuantificar (hambre). Por otro lado, existe la incertidumbre de muchos aspectos; si me pido el plato A ¿me quedaré con hambre?, ¿seré capaz de comerme todo?, ¿me gustará la forma en la que preparan el plato en este restaurante?, etc.

No existe un modelo genérico que sirva para todas las situaciones, si no que en función del tipo de decisiones a tratar se debe utilizar un tipo u otro. Un aspecto muy importante en el diseño de estos modelos es el tiempo de respuesta, ya que no sirve de nada tener un modelo muy bueno si tarda una semana en tomar una decisión como la del ejemplo anterior. Aunque también es verdad que este aspecto depende del ámbito, ya que hay veces que el tiempo de respuesta no es tan importante (una toma de decisión sobre qué coche comprar no tiene por qué ser inmediata, es preferible que sea la acertada).

En este apartado vamos a tratar tres de los modelos más populares para la toma de decisiones: Reglas, Redes Bayesianas y lógica difusa.

2.4.1. Reglas

Este es uno de los modelos más antiguos y rudimentarios de toma de decisiones. Simplemente consiste en tomar decisiones en base a unas reglas determinadas donde no existe incertidumbre. El proceso es sencillo, se parte de un conjunto de datos y se obtiene una salida resultante de aplicar las reglas pertinentes.

Este modelo es poco flexible y no funciona en ámbitos dónde la incertidumbre forma parte de él. Además, no es nada fácil de utilizar, ya que cuando el conjunto de datos de entrada está formado por varios elementos, el número de reglas suele ser muy elevado (al igual que la complejidad de las mismas) y por lo tanto difícil de gestionar.

Las reglas son del tipo *Si condición entonces A, sino B* ó anidadas, es decir, *Si condición1 entonces A, si no, si condición2 entonces B, si no, si condición3 entonces C si no*, etc.

Pongamos un ejemplo, imaginemos que queremos tomar decisiones en un ámbito con pocos valores de entrada como dirigir un robot (con cuatro sensores sónar[10] que se mueve mediante dos ruedas) para que no se choque. Como valores de entrada tenemos los valores de los cuatro sensores y como salida, la velocidad que hay que aplicar a cada rueda. Para simplificar el problema supongamos que el rango de valores de cada sensor es [0.0, 10.0] con la precisión de un decimal y dividimos ese rango en 4 subrrangos a tener en cuenta. Por lo tanto, de cada sensor distinguimos cuatro grupos, pero en las reglas hay que tener en cuenta todas las posibilidades, es decir, hay que tener en cuenta el valor del resto de sensores, por ello, tendríamos un sistema formado por 256 reglas, dónde la condición de cada una estaría formada por cuatro subcondiciones.

Como se puede observar, para un ejemplo sencillo el sistema resultante es bastante complejo. Por ello, este modelo se utiliza cuando se está totalmente seguro de que no va a existir incertidumbre y el número de reglas es reducido.

[10] Es una técnica que utiliza la propagación del sonido para detectar obstáculos. Su funcionamiento (aplicado al ámbito del ejemplo) es sencillo. Consiste en emitir un sonido y medir el tiempo que tarda en escucharse el eco. A partir de esa medición, se pude calcular la distancia a la que se encuentra el objeto.

2.4.2. Redes Bayesianas

Formalmente, una red *Bayesiana* ([5] y [6]) es un grafo[11] acíclico[12] dirigido[13] en el que cada nodo representa una variable y cada arco representa relaciones de dependencia directas entre las variables. Los nodos pueden representar cualquier tipo de variable, ya sea un parámetro cuantificable, una variable latente o una hipótesis.

Una red *bayesiana* está compuesta por:

1. Un grafo dirigido acíclico (GDA) donde cada nodo representa una variable aleatoria y los arcos representan dependencias probabilísticas entre variables. A esta parte de la red se la denomina estructura o modelo.

2. Una distribución de probabilidades condicionadas de la forma P ($x \mid \pi_x$) para cada nodo x dado su conjunto de padres π_x. Estos son los llamados parámetros de la red bayesiana.

En la *ilustración 10* podemos ver un ejemplo de red bayesiana. En este caso, los nodos *D*, *A* y *F* serían los nodos de entrada a la red, mientras que el nodo *I* sería el nodo de salida. Por otro lado, el nodo *S* sería el nodo intermedio.

La tabla que aparece al lado de cada nodo, es la distribución de probabilidades condicionales. Cuando todos los nodos están perfectamente definidos, se calcula el valor del nodo de salida (y de los nodos intermedios), en este caso, el nodo I, el cual contemplará la probabilidad de que se cumpla *i*.

El cálculo de las probabilidades de los nodos intermedios y de salida se puede realizar aplicando diversos algoritmos (*Junction tree*, *Variable elimination*, *Pearl´s polytree*, *Exact inference*, *Metropols Hastings*, *Importance sampling*, etc., etc.).

Programas como *GeNIe & SMILE*, *XBAIES 2.0*, *MSBNx*, *Huginn*, *UnBBayes*, etc., etc. implementan el algoritmo más utilizado para el cálculo de las probabilidades de redes bayesianas, *Junction Tree*. El cual consiste [4] en transformar la estructura de la red para obtener un árbol, mediante agrupación de nodos usando la teoría de grafos. Para ello, se hace

11 Es un par G = (N, A) dónde N es un conjunto de nodos y A un conjunto de arcos definidos sobre los nodos.

12 Grafo que no contiene ciclos. Un ciclo es un camino no dirigido que empieza y termina en el mismo nodo.

13 Es un grafo en el que los arcos tienen definido una dirección.

una transformación de la red a un árbol de uniones (grupos de nodos) mediante el siguiente procedimiento:

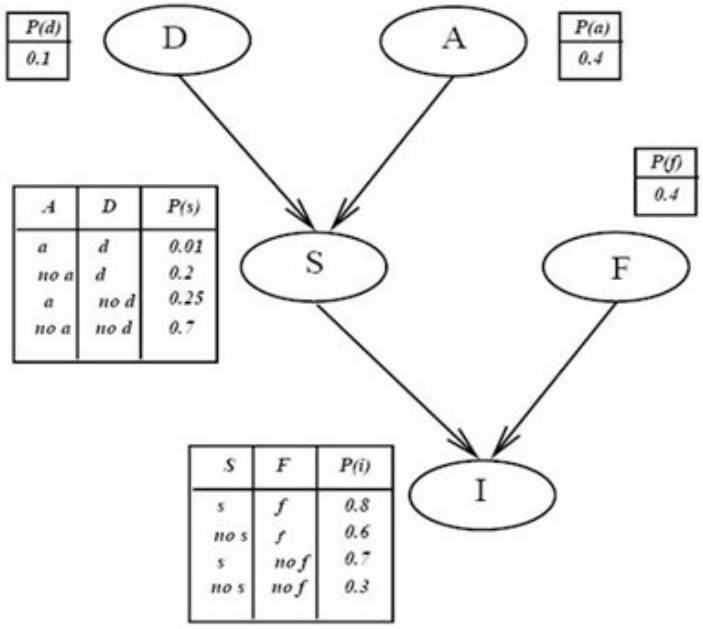

Ilustración 10: Ejemplo de red bayesiana

1. **Moralizar el grafo.** Es un concepto utilizado en la teoría de grafos para encontrar la forma no dirigida equivalente de un grafo acíclico dirigido. Para ello, se añade un arco entre cada par de nodos con algún hijo común. Por último, se elimina la dirección de todos los arcos. En la *ilustración 11* se puede ver el resultado de moralizar un grafo.

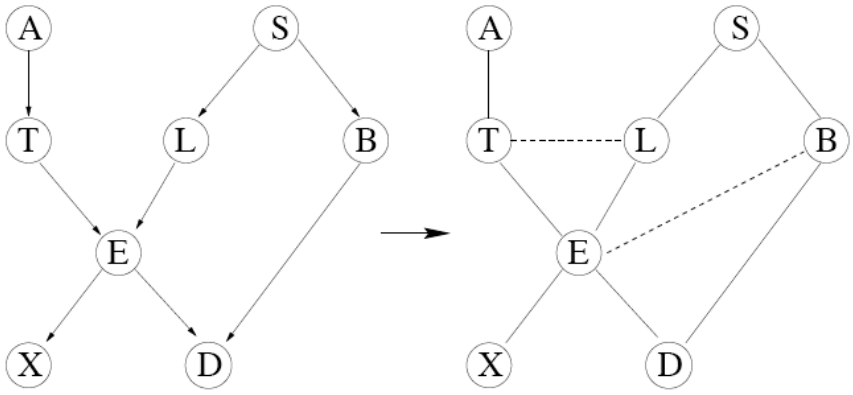

Ilustración 11: Resultado de moralizar un grafo

2. **Triangular el grafo obtenido en el paso anterior.** Un grafo no dirigido es triangular si para cada ciclo de longitud mayor que cuatro existe al menos un arco entre dos nodos no consecutivos del ciclo. Por tanto, se puede obtener un grafo triangular añadiendo

arcos al grafo moral mediante el algoritmo *fill-in* (en español, rellenado de aristas). El resultado de triangular el grafo moralizado se puede analizar en la *ilustración 12*.

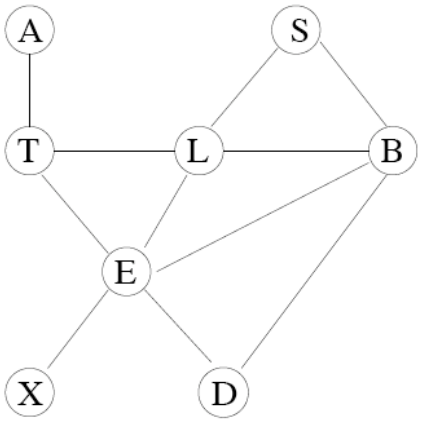

Ilustración 12: Resultado de la triangulación del grafo moralizado

3. **Obtener los *cliques*.** En teoría de grafos, un *clique* en un grafo no dirigido es un conjunto de nodos N, tal que para todo par de nodos de N, existe un arco que las conecta. Los cliques obtenidos del grafo anterior se pueden observar en la *ilustración 13*.

$$\{A,T\}$$
$$\{T,L,E\}$$
$$\{X,E\}$$
$$\{S,L,B\}$$
$$\{L,B,E\}$$
$$\{E,B,D\}$$

Ilustración 13: Lista de *cliques* obtenida del grafo obtenido de la triangulación

4. **Construir árbol de *cliques*.** Obtenido la lista de *cliques*, hay que estructurarlos para que formen un árbol de intersecciones. Principalmente hay dos opciones para obtener el árbol de *cliques*: búsqueda por máxima cardinalidad (MCS) y árbol de peso máximo (MST). La *ilustración 14* ilustra el árbol obtenido aplicando MST.

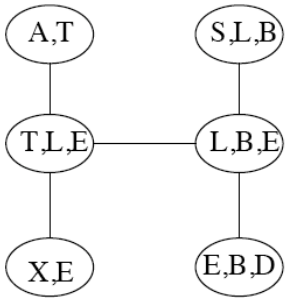

Ilustración 14: Árbol de *cliques* obtenido aplicando MST

Una vez transformado el grafo, la propagación se realiza mediante el envío de mensajes en el árbol de uniones o *cliques*. Inicialmente se calcula la probabilidad conjunta de cada *clique*, y la condicional dado el padre. Dada cierta evidencia se recalculan las probabilidades de cada *clique*. La probabilidad individual de cada variable se obtiene de la del *clique* por marginalización.

2.4.3. Lógica difusa

La lógica difusa no es una lógica precisa, a diferencia de la clásica. Esto se debe en cierta medida a que trabaja con información imprecisa, aunque el resultado, en cierto modo, es muy preciso.

Se puede definir como un método o técnica que traduce a lenguaje matemático reglas u órdenes imprecisas, del tipo, *vamos muy lento, por lo que debes acelerar un poco más* ó *lava la ropa un poco más*, lo que permite a las lavadoras lavar según la suciedad de la ropa.

Es una lógica que se amolda a la forma en la que los humanos tomamos decisiones, de hecho, es considerada por muchos la lógica que utilizamos los seres humanos.

La principal diferencia entre ambas lógicas, es que en la clásica solo tienes como posible resultado verdadero o falso, 0 ó 1, mientras que en la lógica difusa, algo puede ser falso pero cierto en cierta media. Por ejemplo, si una persona dice *Yo juego al tenis todos los días*, pero en realidad solo juega entre semana, en la lógica clásica eso sería falso, ya que sólo juega cinco días en vez de siete. Pero en la difusa, no sería totalmente falso, pero tampoco verdadero. Se podría decir que es verdad al 71%.

La lógica difusa es una generalización de la lógica clásica, en la que un concepto puede tener un determinado grado de verdad en cualquier lugar entre 0 y 1. Sin embargo, los grados difusos no son lo mismo que porcentajes de probabilidad. La probabilidad mide si algo puede ocurrir o no. Las mediciones difusas miden el grado en que una condición existe o algo ocurre. Es decir, las cosas pueden ser verdaderas o falsas con diverso grado.

Un punto muy favorable de este modelo es la facilidad de realizar las reglas, ya que se pueden implementar en lenguaje natural, aunque es necesario un paso previo. Dicho paso consiste en convertir las variables de entrada en rangos. Por ejemplo, imaginemos un ámbito en el que una de las entradas sea la edad. Lo que habría que hacer sería definir rangos con los valores de la edad de tal forma que cada rango recibe un nombre (joven, adulto, etc.) con la

característica de que los rangos están superpuestos. En la *ilustración 15* se puede ver un ejemplo de variable borrosa con la velocidad.

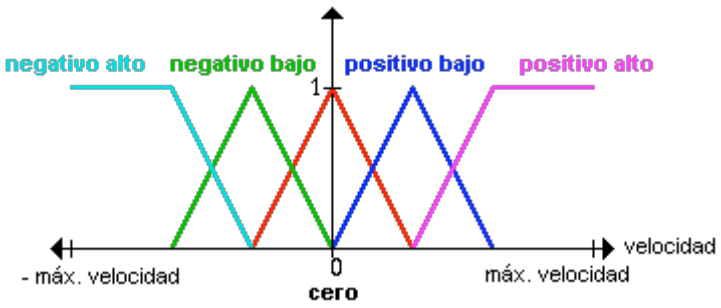

Ilustración 15: Ejemplo de variable borrosa

Para tomar decisiones con lógica difusa hay que seguir los siguientes pasos:

- **Modelizar las variables de entrada y salida.** Cada variable de entrada debe ser definida mediante una variable borrosa, como el ejemplo anterior. Además, hay que definir las variables borrosas de salida, que indicarán la decisión o decisiones que hay que tomar.

- **Modelizar las reglas.** Definir las reglas borrosas necesarias utilizando las variables definidas en el punto anterior. Toda regla tiene un antecedente y un consecuente. En el antecedente se utilizarán las variables borrosas de entrada y en el consecuente las de salida.

- **Borrosificar los datos de entrada.** Definidas todas las variables y reglas borrosas, hay que convertir los datos numéricos de entrada en datos borrosos.

- **Realizar el razonamiento.** Se aplican todas las reglas necesarias a las variables de entrada y salida.

- **Desborrosificar.** Una vez que se han aplicado las reglas, se debe convertir los datos de las variables de salida en datos numéricos, los cuales indicarán la decisión o decisiones que hay que llevar a cabo.

2.5. Consideraciones finales

En el segundo capítulo se ha realizado una introducción al dominio del proyecto, donde el primer paso fue definir el concepto "juego de estrategia en tiempo real", junto con la descripción de los subgéneros más representativos, una breve evolución de los juegos RTS y un pequeño estudio de juegos no comerciales, como ORTS.

Para profundizar más, se analizó su mundo (características, razas, etc.) y los cuatro juegos del torneo, pues el proyecto se va a desarrollar en el escenario del tercer juego. Pero el dominio no es únicamente ORTS, sino también los modelos de toma de decisión, por lo que se estudiaron tres de los más populares. Curiosamente, dichos modelos no se han implementado antes en ORTS.

La dificultad de ORTS no reside únicamente en el razonamiento con incertidumbre, sino que dicho razonamiento debe ser en tiempo real, lo cual aumenta en gran medida la complejidad del dominio. Además, implementar un modelo que no se ha utilizado antes sobre un dominio del que apenas existe documentación, incrementa su dificultad.

Capítulo 3

Objetivos

3. Objetivos

Los principales objetivos de este proyecto son: el diseño e implementación de un cliente funcional para el tercer dominio de ORTS mediante técnicas de IA y la documentación de todo el proceso. Con esto, se pretende demostrar que se puede utilizar técnicas de IA en los videojuegos con resultados decentes y además, se pretende fomentar el uso de ORTS.

A continuación, se detallan todos los objetivos del proyecto:

1. **El cliente debe ser funcional.** Uno de los principales objetivos de este proyecto, es que el cliente sea funcional de forma eficiente, es decir, no limitarse a hacer un buen diseño e implementarlo al margen de los resultados obtenidos, sino que los resultados que obtenga el cliente, se van a intentar maximizar todo lo posible.

2. **El cliente debe utilizar técnicas de IA.** Se pretende crear un cliente que utilice técnicas de IA para la toma de decisiones. Aunque no todas se tomarán como resultado de dichas técnicas, ya que se utilizarán reglas predeterminadas para acciones que carezcan de incertidumbre.

3. **El cliente debe participar en el torneo de ORTS del año 2009.** Con el objetivo de agradecer a Michael Buro y a todo su equipo el gran esfuerzo que están realizando para sacar el proyecto ORTS hacia delante, el cliente debe estar terminado (o al menos una versión funcional) para verano de 2009, ya que tiene que participar en el torneo del presente año. Además, dicha competición también servirá para obtener resultados del cliente contra otros clientes de todo el mundo, incluyendo los del propio equipo de desarrollo de ORTS.

4. **El cliente debe estar bien estructurado.** Conseguir una buena estructuración del cliente es imprescindible para que se puedan reutilizar partes de su comportamiento. Además, puede servir como punto de partida para principiantes en este mundo, ya que pueden intentar mejorarlo o incluso extraer comportamiento aislados y crear con ellos un nuevo cliente para un dominio diferente.

5. **Generar la documentación de todo el proceso de diseño e implementación.** A día de hoy, no existe ningún manual que enseñe/explique cómo se crea un cliente ORTS, por lo que se quiere documentar todo el proceso de diseño e implementación para que sirva

de punto de partida para las personas interesadas en el proyecto, pero que por las dificultades que supone empezar casi sin ayuda, terminan abandonando la idea de unirse a ORTS.

6. **Generar la documentación del API[14] utilizado para la implementación del cliente.** Uno de los principales defectos de ORTS, es que el API que ofrece no está documentado, lo cual no favorece en absoluto la expansión del proyecto. Por esto, se pretenden documentar todas las librerías utilizadas e implementadas, con el objetivo de contribuir en este gran proyecto y de esta forma, a que el número de usuarios crezca cada vez más.

[14] Interfaz de programación de aplicaciones (del inglés, Application Programming Interfaces).

Capítulo 4

Desarrollo

4. Desarrollo

En este apartado, se van a describir todos los pasos realizados en la implementación del cliente. Se empezará con un breve análisis, que abarcará casos de uso y requisitos de usuario. A continuación, le seguirá un estudio de las estrategias a implementar. Dicho estudio, partirá de un listado de estrategias extraídas del juego *Starcraft*, las cuales serán la entrada de un proceso de abstracción y adaptación (para el tercer escenario de ORTS). La salida del proceso anterior será filtrada, para posteriormente realizar un análisis exhaustivo de las estrategias a implementar.

El siguiente paso consistirá en una breve conclusión sobre qué modelo de toma de decisión se ha elegido en este proyecto y cuales han sido las motivaciones para tomar dicha decisión. Acto seguido, se detallará la implementación del modelo.

Por último, se explicarán las estructuras intermedias utilizadas, las cuales servirán de puente entre las decisiones tomadas por el general (modelo de toma de decisión) y las estrategias implementadas.

4.1. Introducción

Como podemos observar, se pueden distinguir tres capas o niveles en la estructura de nuestro cliente, la cual se corresponde con la *ilustración 16*. La capa *General* es la implementación del modelo de toma de decisión de carácter estratégico y decidirá qué hay que ejecutar. Estas decisiones serán procesadas por las estructuras intermedias (capa *Estructuras*) y llevadas a cabo por la capa Táctica (implementación de las diferentes estrategias).

Ilustración 16: Estructura del cliente

Hay que tener en cuenta que el general tomará decisiones globales, es decir, decidirá si hay que atacar, pero no el lugar ni las unidades que deberán llevar a cabo dicha tarea. Esa decisión será tomada por la capa *Estructuras*, en función del resto de decisiones, de la situación geográfica y de los recursos disponibles. Además, hay determinadas situaciones donde es

imprescindible omitir temporalmente algunas decisiones del general, para poder ejecutar otras. Por ejemplo:

- La partida ha comenzado hace escasos minutos y ya se ha construido el edificio *Barracks*.
- El general decide entrenar una unidad *marine*, porque dispone de poco poder militar y construir un edificio *Factory*, para poder entrenar otro tipo de unidades más poderosas.
- Se dispone de recursos suficientes para ejecutar la primera decisión pero no la segunda.

En este punto, si se decide entrenar la unidad militar, el general volverá a pedir una nueva unidad *marine* (porque el poder militar es escaso, sólo se cuenta con *marines*) y la construcción del edificio, por lo que nos encontraremos en la situación anterior, lo que conlleva a entrenar muchas unidades *marine* antes de construir el edificio *Factory*. El problema de este planteamiento radica en el tiempo de ejecución, es decir, antes de poder empezar a construir el edificio, lo más probable es que se reciba un ataque, por lo que se perderán unidades militares y se tendrán que reponer. Por ello, es necesario omitir la decisión de entrenar la unidad militar hasta que se construya el edificio. Obviamente, antes de omitirla, hay que asegurarse que se cuente con un mínimo de poder militar (cinco o seis marines). La responsabilidad de omitir determinadas decisiones bajo situaciones predefinidas, recae en la capa *Estructuras*.

4.2. Análisis

Como se ha comentado anteriormente, se presenta a continuación un análisis del sistema con sus casos de uso y requisitos de usuario.

4.2.1. Casos de uso

Se distinguen dos autores diferentes: cliente y servidor. El cliente es el programa que vamos a implementar, es decir, el programa que recibe la situación actual del juego y envía las acciones que se deben ejecutar de cada objeto. Y el servidor, es el programa que recibe acciones de los objetos del juego, las procesa, las ejecuta y manda a los clientes la nueva situación del juego.

Los casos de uso del cliente se pueden ver en la *ilustración 17*.

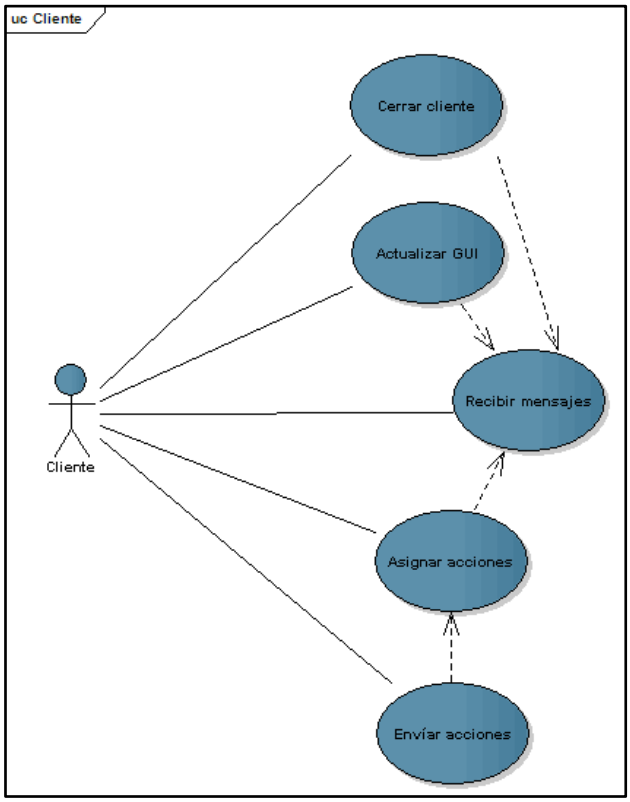

Ilustración 17: Casos de uso del cliente

Por su parte, los casos de uso del servidor se pueden ver en la *ilustración 18*.

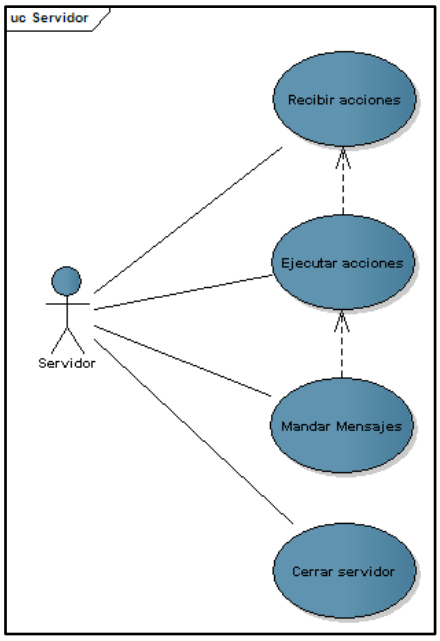

Ilustración 18: Casos de uso del servidor

4.2.2. Requisitos de usuario

Los requisitos de usuario se van a presentar en tablas, dónde cada tabla contiene los siguientes campos:

- **Identificador:** Se utiliza para identificar de forma unívoca cada uno de los requisitos. El identificador tendrá el formato *RU – XXX*, dónde *XXX* es el número del requisito.

- **Nombre requisito:** Título dado al requisito que pueda definir en pocas palabras para qué sirve.

- **Descripción:** Explicación del requisito de forma clara y breve que recoja la funcionalidad plasmada en el mismo.

A continuación se listan todos los requisitos de usuario:

UR – 001	
Nombre Requisito	Tasa de error.
Descripción	El número de fallos del sistema debe ser inferior al 10% de las ejecuciones.

Tabla 1: Requisito UR - 001

UR – 002	
Nombre Requisito	Entorno heterogéneo.
Descripción	El sistema debe funcionar correctamente en los sistemas operativos Mac OS X y Linux.

Tabla 2: Requisito UR - 002

UR – 003	
Nombre Requisito	Sistema dinámico.
Descripción	El sistema debe ser capaz de adaptarse a los diferentes estados que se puedan presentar y actuar de forma eficiente en cada uno de ellos.

Tabla 3: Requisito UR - 003

UR – 004	
Nombre Requisito	Técnicas de Inteligencia Artificial.
Descripción	El sistema debe implementar al menos una técnica de inteligencia artificial para la toma de decisiones.

Tabla 4: Requisito UR - 004

UR – 005	
Nombre Requisito	Acciones tácticas.
Descripción	El sistema debe ser capaz de recolectar minerales, defenderse de ataques, elaborar ataques, entrenar nuevas unidades, construir nuevos edificios, realizar la acción de scouting y fundar nuevas bases.

Tabla 5: Requisito UR – 005

UR – 006	
Nombre Requisito	Tiempo de respuesta.
Descripción	El sistema debe procesar la situación actual del juego, asignar a cada objeto la acción que debe ejecutar y enviar las acciones de los objetos al servidor en un tiempo inferior a ciento veinticinco milisegundos.

Tabla 6: Requisito UR – 006

UR – 007	
Nombre Requisito	Ejecución concurrente.
Descripción	El sistema debe ejecutar de forma concurrente todas las acciones tácticas necesarias en cada ciclo de la simulación.

Tabla 7: Requisito UR – 007

UR – 008	
Nombre Requisito	Arquitectura cliente-servidor.
Descripción	El sistema debe implementar la arquitectura cliente-servidor, permitiendo que la conexión con el servidor sea local o sobre cualquier máquina conectada a Internet.

Tabla 8: Requisito UR – 008

UR – 009	
Nombre Requisito	Sockets.
Descripción	El sistema debe intercambiar información con el servidor mediante sockets.

Tabla 9: Requisito UR – 009

4.3. Capa Táctica

En un juego RTS hay que contemplar diferentes estrategias para lograr la victoria. Poniendo como punto de partida el juego Starcraft, se van a enumerar las estrategias necesarias para alcanzar la victoria en un escenario, en el que se permite la construcción de todos los objetos del juego y el único objetivo es la completa destrucción del oponente. Acto seguido, se va a realizar un proceso de abstracción y adaptación, en el que el resultado será el conjunto de estrategias que hay que contemplar para lograr la victoria en el tercer escenario del torneo ORTS. Después, se aplicará un filtro al resultado obtenido, utilizando como criterio el tiempo de la duración de la partida y la complejidad de las estrategias. Por último, se realizará un estudio de cada una de las estrategias, centrado en las precondiciones, las post-condiciones y las relaciones existentes entre éllas.

4.3.1. Estrategias para el juego Starcraft

El conjunto de estrategias contempladas para el juego Starcraft son las siguientes:

1. Recolección
    a. **Una única fuente en el asentamiento**. Recolectar recursos de una única fuente del mismo en el asentamiento.

    b. **Varias fuentes en el asentamiento.** Recolectar recursos contando con varias fuentes del mismo en el asentamiento.

2. Defender la base principal
    a. Colocación estratégica de torretas
        i. **En la entrada a la base**. Colocar torretas de defensa en las posibles entradas de la base, con el objetivo de rechazar los posibles ataques enemigos.

        ii. **En las zonas de minerales dentro de la base**. Para proteger las unidades de recolección, construir torretas de defensa cerca de los recursos del asentamiento,

iii. **Entre los edificios de la base**. Distribuir torretas de defensa entre las construcciones de la base, intentando repeler ataques que hayan superado la defensa exterior.

b. Colocación estratégica de unidades militares

i. **Junto a las entradas de la base**. Para rechazar los posibles ataques enemigos, colocar unidades militares en las posibles entradas de la base.

ii. **Dentro de la base**. Con el objetivo de repeler ataques que hayan superado la defensa exterior, se distribuyen unidades militares entre las construcciones de la base.

3. Defender base secundaria

a. Colocación estratégica de torretas

i. **Cercando la base**. En el perímetro del asentamiento se construyen torretas de defensa, intentando repeler los posibles ataques enemigos.

ii. **Alguna en el interior**. Distribuir estratégicamente torretas de defensa en el interior del perímetro del asentamiento, para rechazar los ataques enemigos que hayan superado la defensa exterior.

b. Colocación estratégica de unidades militares

i. **Alrededor de la base, entre medias de las torretas**. Colocar unidades militares junto a las torretas de defensa, con el objetivo de obtener una defensa del asentamiento más poderosa.

4. Expansión de los asentamientos

a. **Base principal**. Construir todo tipo de edificios (para mejorar unidades, para entrenar unidades y para aumentar la capacidad de la población).

b. **Base de ataque**. Construir edificios que permitan el entrenamiento de unidades militares.

5. Examinar mapa
    a. **Forma suicida**. Examinar el mapa con una unidad y con independencia de los lugares por los que pase, es decir, aunque pase por una base enemiga la unidad sigue su camino.

    b. **Forma gallina**. Examinar el mapa con una unidad y si se encuentra con alguna unidad enemiga o asentamiento enemigo, cambiará su rumbo para evitar ser atacado.

    c. **Forma colectiva**. Examinar el mapa con varias unidades y en el caso de encontrase con enemigos, atacarán si disponen de superioridad numérica. En caso contrario, cambiarán su rumbo para evitar ser atacados.

6. Crear nuevos asentamientos
    a. **Extracción de minerales**. Crear un nuevo asentamiento para explotar una o varias fuentes de recursos.

    b. **Posiciones tácticas para elaborar ataques**. Crear un asentamiento con un doble objetivo, creación de unidades militares y posicionamiento táctico de las tropas para realizar ataques.

7. Atacar
    a. Grupo de unidades militares
        i. **Técnica gallina**. Colocar un conjunto de unidades militares cerca de un grupo de unidades enemigas y utilizar una unidad como cebo, para que los enemigos se acerquen a nuestras unidades.

        ii. **Guerra campal por flancos**. Atacar de forma directa a un grupo de enemigos mediante varios subgrupos, cada uno por un flanco diferente.

    b. Asentamiento enemigo principal
        i. **Ataque a entrada más débil**. Atacar a la entrada más débil de un asentamiento enemigo, con el objetivo de acceder a su interior.

ii. **Ataque a entrada más táctica**. Atacar a la entrada más táctica del asentamiento enemigo, es decir, la entrada que disponga de edificios de entrenamiento de unidades cercanos.

iii. **Ataque por varias entradas a la vez**. Atacar un asentamiento enemigo por diferentes entradas a la vez.

iv. **Ataque campal**. Atacar el asentamiento enemigo con todos los efectivos disponibles.

c. Asentamiento enemigo secundario
   i. **Ataque gallina**. Aplicar la técnica gallina sobre una entrada de una base secundaria enemiga.

   ii. **Ataque campal**. Atacar un asentamiento secundario enemigo con una gran cantidad de efectivos.

   iii. **Ataque por flancos**. Atacar un asentamiento secundario enemigo por varios flancos a la vez.

8. Construcción de unidades
   a. **Unidades de recolección y construcción**. Entrenar unidades capaces de recolectar recursos y construir edificios.

   b. **Unidades militares terrestres**. Entrenar unidades militares terrestres.

   c. **Unidades militares aéreas**. Construir unidades aéreas.

9. Realizar mejoras de las unidades construidas
   a. **Unidades terrestres**. Realizar mejoras sobre las armas y escudos, y habilitar acciones especiales de las unidades terrestres.

   b. **Unidades aéreas**. Aumentar la potencia de las armas, reforzar la capacidad de defensa de los escudos y habilitar las acciones especiales de las unidades aéreas.

4.3.2. Proceso de abstracción y adaptación de estrategias

A continuación, se va a realizar el proceso de abstracción y adaptación de las estrategias, para que el resultado sea perfectamente válido para el tercer escenario de ORTS.

Para llevarlo a cabo, se van a definir tres grupos de estrategias. El primer grupo va a estar formado por las que directamente no se pueden aplicar por las restricciones del escenario, el segundo, por las estrategias que tienen que sufrir modificaciones, y por último, las que son perfectamente válidas debido a la abstracción contenida en su definición, formarán el tercer grupo.

En este proceso se va a realizar un pequeño análisis de cada estrategia, asignando cada una al grupo al que pertenece. Además, las estrategias del segundo grupo (estrategias que necesitan modificaciones para ser válidas), serán modificadas.

La primera estrategia consiste en recolectar minerales y tiene dos variantes. La primera es el caso de que sólo exista una única fuente de minerales en el asentamiento, por lo que todos los obreros recolectarán de la misma fuente. La otra variante, corresponde al caso de que exista más de una fuente de minerales en el asentamiento, por lo que habrá que repartir los obreros en función de las necesidades actuales y las distancias a las fuentes. Esta estrategia pertenece al tercer grupo, ya que es perfectamente válida, aunque habría que aplicarle la restricción de recolección de minerales del escenario.

El objetivo de la segunda estrategia es la defensa de la base principal y cuenta con dos variantes. La primera variante consiste en la colocación estratégica de torretas de defensa, pero debido a que en el tercer escenario de ORTS está prohibida la construcción de dichas torretas, esta variante pertenece al primer grupo (estrategias que no se pueden aplicar por las restricciones del escenario). La segunda variante, consiste en la defensa de la base principal mediante unidades militares. Dichos objetos se pueden colocar principalmente en dos zonas. La primera se corresponde con la/s entrada/s a la base, es decir, las regiones de la base que están expuestas a posibles ataques. Y la segunda se corresponde con las zonas internas de la base, con el objetivo de contrarrestar un ataque que haya conseguido romper la protección de una entrada a la base. Esta segunda variante, al igual que la estrategia anterior, cumple con los requisitos del tercer escenario del torneo ORTS, por lo que corresponde al tercer grupo.

La tercera estrategia tiene como objetivo la defensa de las bases secundarias y al igual que la estrategia anterior, dispone de dos variantes. La primera de ellas, consiste en la colocación

estratégica de torretas de defensa, por lo que pertenece al primer grupo. Por otro lado, la segunda variante, consiste en la colocación estratégica de unidades militares alrededor de la base, entre medias de las torretas de defensa. Tal y como está definida esta variante, pertenece al segundo grupo, por lo que habría que modificarla para que perteneciese al tercero, que es el objetivo. En este caso la modificación es muy sencilla, ya que basta con eliminar las torretas de defensa de la variante.

La siguiente estrategia (la cuarta), tiene como objetivo la expansión de los asentamientos, es decir, la construcción de edificios en función del objetivo y tipo de asentamiento. Se distinguen tres tipos de asentamientos:

1. **Base principal:** Su objetivo es la recolección de minerales, construcción de edificios (para mejorar las unidades, para poder crear nuevas unidades, para aumentar la capacidad de la población) y el entrenamiento de unidades militares (para la defensa de la base y para realizar ataques sobre los oponentes).

2. **Base de ataque:** El objetivo de esta base, es la creación masiva de unidades militares para realizar ataques estratégicos. Parte de esas unidades se destinaran para la defensa del asentamiento.

Hay que tener en cuenta que una base creada originalmente con un propósito puede cambiar a lo largo del tiempo. Por ejemplo, una base de recolección puede utilizarse también como base de ataque, si se descubre que está localizada en un punto estratégico. Tal y como está definida la estrategia debería pertenecer al tercer grupo, pero debido a los objetivos de los asentamientos, pertenece al segundo. El problema se encuentra localizado en el objetivo de la base principal, ya que en el tercer escenario de ORTS no se pueden construir edificios para tales fines. Por ello, la modificación consiste en disminuir los objetivos de la base principal, convirtiéndola en la unión modificada de los otros dos tipos de asentamientos. Los nuevos objetivos de la base principal serían los siguientes:

- Recolección de minerales.
- Construcción de unidades militares para la defensa de la base.
- Construcción de unidades militares para realizar ataques a los oponentes.
- Construcción de edificios para construir unidades militares.

Con estos nuevos objetivos, y teniendo en cuenta que no se dice de forma explícita qué unidades o tipo de unidades se tienen que entrenar, la estrategia pertenece al tercer grupo.

La quinta estrategia tiene como objetivo explorar las zonas del mapa que no han sido descubiertas. Se distinguen tres formas de realizar la exploración:

1. **Forma suicida:** Consiste en mandar a un soldado a zonas ocultas en el mapa, sin importar lo que se pueda encontrar, dándole sólo nuevas coordenadas cuando haya llegado a su destino.

2. **Forma gallina:** Es una variación de la forma suicida, pero en este caso en cuanto el soldado descubra una unidad enemiga, se le asignarán nuevas coordenadas en sentido contrario a la unidad enemiga localizada.

3. **Forma colectiva:** En este caso, se manda a un conjunto de soldados a zonas sin descubrir y si por el camino se descubren enemigos, se pueden realizar dos acciones. La primera es atacar si disponen de superioridad numérica (teniendo en cuenta los puntos de salud de cada soldado) y si no es así, se asigna al grupo de soldados una nueva coordenada en sentido contrario al grupo de enemigos.

La quinta estrategia no incumple ninguna de las restricciones del tercer escenario, por lo que pertenece al tercer grupo.

Con la sexta estrategia se pretenden crear nuevos asentamientos, ya sea de tipo ataque o de recolección. Esta estrategia pertenece al tercer grupo porque es compatible con el dominio que se está utilizando. Pensando esta estrategia en el dominio del tercer escenario ORTS, puede ser necesaria la inclusión de crear una nueva base principal, ya que sus objetivos son la unión de los dos anteriores. Aun así, se ha decidido no incluirla porque una base creada para un propósito, puede cambiar con el tiempo.

La séptima estrategia se basa en realizar ataques y en ella se distinguen tres tipos. El objetivo del primer tipo es realizar ataques a pequeños grupos de unidades enemigas. Dichos ataques se pueden realizar aplicando dos técnicas:

1. **Técnica Gallina:** Consiste en colocar de forma estratégica un conjunto de unidades militares cerca del grupo que se desea atacar sin ser vistos. Una vez que todos estén colocados, una unidad se acerca al grupo enemigo hasta que sea visto y éstos se muevan

hacia él. En ese momento, a la unidad se le ordena volver con el resto de compañeros. Con esto se consigue que el grupo que se desea atacar caiga en la trampa y la eliminación del mismo sea más sencilla que con un ataque directo.

2. **Guerra campal por flancos:** Se forman varios grupos de unidades y se realiza un ataque directo sobre el grupo enemigo por varios flancos.

El siguiente tipo, tiene como objetivo el ataque a la base principal del enemigo y se puede realizar aplicando varias estrategias:

1. **Ataque a la entrada más débil:** Consiste en realizar un ataque a la entrada de la base principal más débil, es decir, la entrada que cuente con menos efectivos para defender el ataque.

2. **Ataque a la entrada más táctica:** Consiste en realizar un ataque a la entrada que permita hacer más daño táctico, por ejemplo, atacar a la entrada que tenga cerca un edificio de creación de unidades, teniendo la opción de destruirlo antes de que el enemigo envíe todos sus efectivos para desbaratar la maniobra.

3. **Ataque por varias entradas a la vez:** Consiste en lanzar un ataque por varias entradas de forma simultánea, con el objetivo de que el enemigo tenga que repartir sus efectivos y poder romper y/o debilitar con más facilidad su defensa de la base.

4. **Ataque campal:** Consiste en realizar un ataque a la base principal enemiga con todos los efectivos disponibles.

El último tipo de ataque está enfocado sobre los asentamientos secundarios. Dicho ataque se puede llevar a cabo de tres formas:

1. **Ataque gallina:** Es el resultado de aplicar la técnica gallina (explicada en el primer tipo de ataque) sobre una entrada de una base secundaria enemiga.

2. **Ataque campal:** Es el resultado de realizar un ataque con una gran cantidad de efectivos. No hay que confundir este ataque campal con el del tipo anterior, ya que el anterior, se realiza con todos los efectivos disponibles y está motivado por una situación desesperada. Por ejemplo, porque se han agotado todas las fuentes de minerales.

3. **Ataque por flancos:** Es el equivalente al ataque por varias entradas a la vez del anterior tipo de ataque.

La estrategia de ataque está redactada de forma genérica, es decir, especifica el cómo llevarla a cabo, pero no explica quién debe ejecutarla, por lo que no viola ninguna de las restricciones del escenario (tercer grupo).

La octava estrategia consiste en la creación de unidades, ya sea para recolección de recursos y construcción de edificios, defensa de la base o para realizar ataques (mediante unidades militares terrestres y/o aéreas). Esta estrategia no es válida en el contexto en que está definida, ya que dice explícitamente el tipo de unidad, y en el dominio de interés no está permitido la construcción de unidades militares aéreas, por lo que habría que suprimir este tipo para que la estrategia perteneciese al tercer grupo.

La última estrategia consiste en realizar mejoras de las unidades disponibles, con el objetivo de aumentar el poder de ataque, de defensa y de optar a acciones que antes no se podían realizar. Debido a las limitaciones del tercer escenario, esta estrategia pertenece al primer grupo.

Para finalizar la sección, se va a realizar un breve resumen con el resultado del proceso de abstracción y adaptación. Dicho resumen está contenido en la *tabla 10*.

Identificador de la estrategia	Grupo
1. a. Recolección de una única fuente	Tercer grupo
1. b. Recolección de varias fuentes	Tercer grupo
2. a. i. Defender base principal con torretas en la entrada	Primer grupo
2. a. ii. Defender base principal con torretas junto a los minerales	Primer grupo
2. a. iii. Defender base principal con torretas entre los edificios	Primer grupo
2. b. i. Defender base principal con unidades militares en la entrada	Tercer grupo
2. b. ii. Defender base principal con unidades militares en el interior	Tercer grupo
3. a. i. Defender base secundaria con torretas en el perímetro	Tercer grupo
3. a. ii. Defender base secundaria con torretas en el interior	Tercer grupo
3. b. i. Defender base secundaria con unidades militares entre las torretas	Segundo grupo
4. a. Expansión de base principal	Segundo grupo
4. b. Expansión de base de ataque	Segundo grupo
5. a. Examinar mapa de forma suicida	Tercer grupo
5. b. Examinar mapa de forma gallina	Tercer grupo
5. c. Examinar mapa de forma colectiva	Tercer grupo
6. a. Crear nuevo asentamiento de extracción de minerales	Tercer grupo
6. b. Crear nuevo asentamiento para ataques	Tercer grupo
7. a. i. Ataque a un grupo enemigo con técnica gallina	Tercer grupo
7. a. ii. Ataque a un grupo enemigo por flancos	Tercer grupo

Identificador de la estrategia	Grupo
7. b. i. Ataque a la base principal enemiga por la entrada más débil	Tercer grupo
7. b. ii. Ataque a la base principal enemiga por la entra más táctica	Tercer grupo
7. b. iii. Ataque a la base principal enemiga por varias entradas	Tercer grupo
7. b. iv. Ataque campal a la base principal enemiga	Tercer grupo
7. c. i. Ataque a una base secundaria enemiga mediante técnica gallina	Tercer grupo
7. c. ii. Ataque campal a una base secundaria enemiga	Tercer grupo
7. c. iii. Ataque por flancos a una base secundaria enemiga	Tercer grupo
8. a. Construcción de unidades de recolección y construcción	Tercer grupo
8. b. Construcción de unidades militares terrestres	Tercer grupo
8. c. Construcción de unidades militares aéreas	Primer grupo
9. a. Realizar mejoras de unidades terrestres	Primer grupo
9. b. Realizar mejoras de unidades aéreas	Primer grupo

Tabla 10: Resultado del proceso de adaptación y abstracción

4.3.3. Filtrado de estrategias

A continuación, se va a aplicar un filtro con el objetivo de obtener las estrategias más adecuadas para el dominio. Atendiendo al criterio del tiempo de la partida, la estrategia que se podría suprimir sería la sexta, es decir, la creación de nuevos asentamientos, debido a que el tiempo y esfuerzo necesario para su creación puede que no se rentabilice durante la partida. Aún así, debido a que nos encontramos en una etapa de diseño y no se pueden simular los resultados de aplicar o no esta estrategia, se va a dejar con el objetivo de realizar un análisis sobre la conveniencia de aplicarla.

Atendiendo al segundo criterio del filtro (la complejidad de las estrategias) y teniendo en mente el primero, se ha decidido reducir la séptima estrategia (atacar), ya que se considera que es demasiado compleja y en algunos casos se necesitan muchos recursos y tiempo para poder llevarlas a cabo. En un principio se había decidido prescindir de las siguientes estrategias de ataque:

- **Del ataque al asentamiento enemigo principal:** Ataque a la entrada más táctica.
- **Del ataque el asentamiento enemigo secundario:** Ataque gallina y ataque por flancos.

Además, se había decidido combinar el ataque a entrada más débil con el ataque por varias entradas a la vez, obteniendo así un mejor rendimiento de los ataques. Pero debido a la complejidad de las estrategias de ataque, el tiempo y esfuerzo que hay que dedicar a su

implementación y teniendo el cuenta que el proyecto está más orientado a una toma de decisiones eficiente (de forma que las unidades interactúen entre sí), que a elaborar estrategias base (tácticas) complejas, se ha decidido dejar sólo un tipo de ataque por cada grupo, es decir, la séptima estrategia quedaría así:

a. Grupo de unidades militares
    i. **Guerra campal**. Atacar de forma directa a un grupo de enemigos.

b. Asentamiento enemigo principal
    i. **Ataque campal**. Atacar el asentamiento enemigo con todos los efectivos disponibles.

c. Asentamiento enemigo secundario
    i. **Ataque campal**. Atacar un asentamiento secundario enemigo con una gran cantidad de efectivos.

Teniendo el cuenta los motivos que han llevado a la modificación de la estrategia de ataque, es obvio que también hay que reducir la estrategia de defensa, ya que algunas técnicas eran similares en complejidad a las excluidas en el ataque. En concreto, la estrategia que se va a excluir es *Defender la base principal – Colocación estratégica de unidades militares – Junto a las entradas de la base*, ya que localizar las entradas o posibles entradas a una base no es nada sencillo.

Por los mismos motivos, se va a modificar la estrategia de defensa de una base secundaria, pasando de colocar unidades militares en las entradas de la base, a colocarlas en el interior del perímetro, pero estando alerta de cualquier movimiento enemigo en el exterior de la base, para deshacer lo más rápido posible el ataque.

Otra estrategia que se va a ver reducida es la número cinco (examinar el mapa), debido al tiempo de implementación, ya que es preferible reducirla e invertir ese tiempo en el desarrollo de un general eficiente e inteligente. La estrategia elegida para su implementación es la forma suicida de examinar el mapa, ya que se considera que es la mejor para localizar la base enemiga, aunque conlleve la pérdida de alguna unidad.

La última estrategia que se ha decidido reducir, es la referente a la creación de nuevos asentamientos, concretamente el asentamiento de ataque. El principal motivo, es que esa estrategia está orientada a desarrollos de partidas más complejas de las que se van a poder ver

en el tercer escenario de ORTS (principalmente, debido al tiempo de la partida). En la *ilustración 19*, se plasma una situación de una partida en la que no existe restricción de tiempo, donde los hexágonos y cuadrados representan asentamientos y los círculos, *clusters* de minerales. Los asentamientos en forma de hexágono pertenecen al enemigo y el tamaño representa la proporción de los mismos.

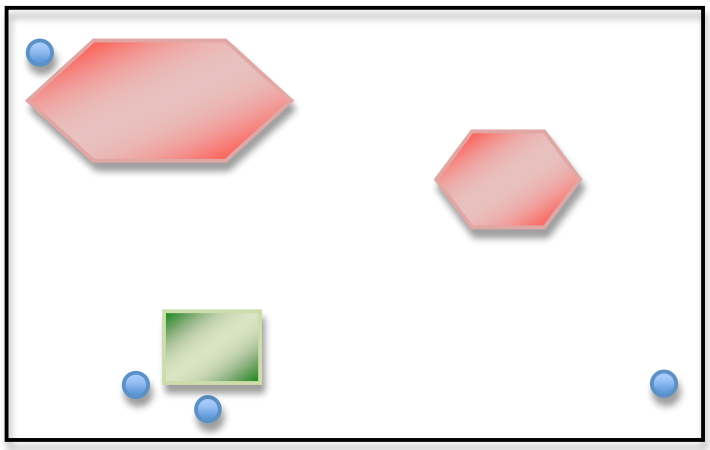

Ilustración 19: Situación de una partida sin límite de tiempo

En esta situación, se estarían recibiendo ataques por parte de los dos asentamientos enemigos, por lo que si se quiere vencer, primero se debería eliminar el asentamiento enemigo derecho. Una vez conseguido, habría que levantar en ese mismo lugar una base secundaria de ataque, ya que el enemigo tiene un asentamiento muy grande y sería muy beneficioso tener uno secundario más cerca y mejor colocado para realizar los ataques. Hay que tener en cuenta, que levantar y mantener la base secundaria es costoso (ya que hay que defenderla, crear los edificios de entrenamiento de unidades y entrenar unidades de ataque), y que en esta situación se puede llevar a cabo porque la principal dispone de dos *clusters* de minerales. De lo contrario, habría que levantar uno nuevo de recolección en el extremo inferior derecho del escenario.

Como se puede observar, en esta situación sí que es rentable la creación del asentamiento de ataque, pero el tiempo requerido para llevar a cabo todo el plan es muy alto, por lo que en una partida de veinte minutos no es posible.

La solución que se ha tomado no es la exclusión de la estrategia, sino una solución intermedia. Es decir, todo asentamiento secundario se va a fundar con el objetivo de recolectar minerales, pero en el momento en el que se cuente con más de una base, se va a tener en cuenta la distancia de la base con la base enemiga y en función de dicha distancia, se aplicará o no la estrategia de expansión de asentamientos secundarios de ataque.

Por último, se va a realizar el estudio de cada una de las estrategias resultantes de los procesos anteriores, las cuales son:

1. Recolección
    a. **Una única fuente en el asentamiento**. Recolectar recursos de una única fuente del mismo en el asentamiento.

    b. **Varias fuentes en el asentamiento**. Recolectar recursos contando con varias fuentes del recurso en el asentamiento.

2. Defender la base principal
    a. Colocación estratégica de unidades militares
        i. **Dentro de la base**. Con el objetivo de repeler ataques que hayan superado la defensa exterior, se distribuyen unidades militares entre las construcciones de la base.

3. Defender base secundaria
    a. Colocación estratégica de unidades militares
        i. **En el interior del perímetro de la base**. Colocar unidades militares en el interior del perímetro del asentamiento, con el objetivo de repeler los posibles ataques enemigos.

4. Expansión de los asentamientos
    a. **Base principal**. Construir todo tipo de edificios (para mejorar unidades, para entrenar unidades y para aumentar la capacidad de la población).

    b. **Base de ataque**. Construir edificios que permitan el entrenamiento de unidades militares.

5. Examinar mapa
    a. **Forma suicida**. Examinar el mapa con una unidad y con independencia de los lugares por los que pase, es decir, aunque pase por una base enemiga la unidad sigue su camino.

6. Crear nuevos asentamientos
    a. **Extracción de minerales**. Crear un nuevo asentamiento con el objetivo de explotar una o varias fuentes de recursos.

7. Atacar
    a. Grupo de unidades militares
        i. **Guerra campal**. Atacar de forma directa a un grupo de enemigos.

    b. Asentamiento enemigo principal
        i. **Ataque campal**. Atacar el asentamiento enemigo con todos los efectivos disponibles.

    c. Asentamiento enemigo secundario
        i. **Ataque campal**. Atacar un asentamiento secundario enemigo con una gran cantidad de efectivos.

8. Construcción de unidades
    a. **Unidades de recolección y construcción**. Entrenar unidades capaces de recolectar recursos y construir edificios.

    b. **Unidades militares terrestres**. Entrenar unidades militares terrestres.

4.3.4. Estudio exhaustivo de las estrategias resultantes

A continuación se presenta el análisis definitivo en el que analizan las precondiciones, post-condiciones y relaciones de cada una de las estrategias resultantes:

Recolección – Una única fuente en el asentamiento

- **Precondiciones:** Disponer de un edificio *Control Center*, disponer de al menos una unidad *worker* (porque es la única que puede recolectar) y tener localizado una fuente de mineral, que no esté controlada por los enemigos y en la que no se encuentren recolectando cuatro unidades *worker*.

- **Post-condiciones:** Una vez que la unidad *worker* ha recolectado los minerales y los ha depositado en el edificio *Control Center*, la cantidad de minerales disponible ascenderá el número de unidades de minerales que la unidad *worker* haya depositado, siendo diez unidades el máximo de minerales que se pueden recolectar de una sola vez. Además, la cantidad de minerales disponibles en la fuente se verá reducido en la misma cantidad.

- **Relaciones:** Esta estrategia está relacionada con todas las estrategias que necesiten minerales para poderse llevar a cabo.

Recolección – Varias fuentes en el asentamiento

- **Precondiciones:** Disponer de un edificio *Control Center*, disponer de al menos dos unidades *worker* (ya que con una sola no se puede recolectar de dos fuente a la vez) y tener localizado un mínimo de dos fuentes de minerales, que no estén controladas por los enemigos y en las que no se encuentren recolectando cuatro unidades *worker* en cada una.

- **Post-condiciones:** Las mismas que para el caso de recolección con una única fuente en el asentamiento, por cada unidad de recolección.

- **Relaciones:** Esta estrategia está relacionada con todas las estrategias que necesiten minerales para poderse llevar a cabo.

Defender la base principal – Colocación estratégica de unidades militares – Dentro de la base

- **Precondiciones:** Disponer de una base principal y de unidades militares para poder colocarlas de forma estratégica en el interior de la base. Opcionalmente, también se podrían utilizar unidades *worker*, ya que pueden realizar ataques aunque no sean muy efectivos.

- **Post-condiciones:** Una vez que las unidades estén colocadas, aumentará la capacidad de defensa interna de la base y disminuirá la cantidad de unidades militares sin acciones asignadas.

- **Relaciones:** Esta estrategia está relacionada con la creación de unidades.

Defender base secundaria – Colocación estratégica de unidades militares – En el interior del perímetro de la base

- **Precondiciones:** Disponer de al menos una base secundaria y de unidades militares (opcionalmente también se pueden utilizar unidades *worker*) para poder colocarlas de forma estratégica en el interior del perímetro.

- **Post-condiciones:** Una vez que las unidades estén colocadas, aumentará la capacidad de defensa de la base secundaria y disminuirá la cantidad de unidades militares sin acciones asignadas.

- **Relaciones:** Esta estrategia está relacionada con la creación de unidades y con la creación de nuevos asentamientos.

Expansión de los asentamientos – Base principal

- **Precondiciones:** Disponer de una base principal, de minerales suficientes para poder realizar la construcción de edificios y tener el espacio suficiente para realizar las construcciones. Además, es indispensable contar con una unidad *worker*, ya que será ella la encargada de realizar la construcción.

- **Post-condiciones:** Disminuirá la cantidad de minerales disponibles dependiendo de la construcción realizada y aumentará la capacidad de creación de unidades y/o se disminuirá el tiempo de recolección de minerales, si se ha construido un *Control Center* próximo a la fuente de minerales. Por último, la unidad *worker* estará preparada para recibir nuevas ordenes.

- **Relaciones:** Esta estrategia está relacionada con la recolección de minerales y las estrategias de defensa de la base principal y de ataque.

Expansión de los asentamientos – Base de ataque

- **Precondiciones:** Disponer de al menos una base secundaria y minerales suficientes para poder realizar la construcción del edificio *Barracks* o *Factory,* así como la superficie necesaria para realizar la construcción. Como en los casos anteriores de construcción de edificios, es necesario disponer de una unidad *worker*.

- **Post-condiciones:** Disminuirá la cantidad de minerales disponibles en función del coste del edificio construido, aumentará la capacidad de creación de unidades *marine* ó *tank* y volverá a estar disponible la unidad que realizó la obra.

- **Relaciones:** Esta estrategia está relacionada con la recolección de minerales, construcción de nuevos asentamientos y la estrategia de ataque.

Examinar mapa – Forma suicida

- **Precondiciones:** Disponer de al menos una unidad, independientemente del tipo, ya que para realizar la exploración no se necesita capacidad de ataque. También se precisa la necesidad de descubrir zonas del escenario ocultas y/o de descubrir la ubicación de algún asentamiento enemigo, ya se principal o secundario.

- **Post-condiciones:** Aumenta el conocimiento del escenario y existe una probabilidad alta de que se pierda a la unidad que estaba realizando la exploración, pero también hay una probabilidad alta de localizar una base enemiga.

- **Relaciones:** Esta estrategia está relacionada con la creación de unidades, la creación de nuevos asentamientos y la estrategia de ataque.

Crear nuevos asentamientos – Extracción de minerales

- **Precondiciones:** Necesidad de recolectar minerales de fuentes alejadas de todos los asentamientos, tener localizado una fuente de minerales que no esté controlada por los enemigos, disponer de los minerales suficientes para poder crear el *Control Center* y disponer de las unidades necesarias para su defensa o la cantidad de

minerales necesarios para la construcción de dichas unidades. Por último, es esencial contar con una unidad capaz de construir el *Control Center*.

- **Post-condiciones:** Aumenta la capacidad de recolección de minerales y de construcción de unidades *worker*. Disminuye la cantidad de minerales disponibles y de unidades sin acciones asignadas. Además, se vuelve a contar con una unidad *worker* sin acciones asignadas.

- **Relaciones:** Esta estrategia está relacionada con la recolección de minerales.

Atacar – Grupo de unidades militares – Guerra campal

- **Precondiciones:** Tener localizado un grupo de enemigos y disponer de un conjunto de unidades militares con más unidades que el grupo enemigo que se pretende atacar. Hay que tener en cuenta los puntos de salud de las unidades, ya que aunque se cuente con superioridad numérica, si las unidades cuentan con pocos puntos de salud, la probabilidad de éxito sería muy baja.

- **Post-condiciones:** Destrucción del grupo enemigo y la posibilidad de perder parte de las unidades del grupo atacante. También se ganaría en terreno controlado, es decir, en parte del escenario que no domina el enemigo.

- **Relaciones:** Esta estrategia está relacionada con la exploración del mapa y con la construcción de unidades militares terrestres.

Atacar – Asentamiento enemigo principal – Ataque campal

- **Precondiciones:** Tener localizada la base principal enemiga y disponer de al menos una unidad que permita realizar la acción de atacar. Estos son los requisitos mínimos, ya que el ataque se debe realizar con todos los efectivos disponibles.

- **Post-condiciones:** Destrucción total de la base enemiga o destrucción parcial y pérdida de todos los efectivos mandados al ataque, lo que puede conducir a la derrota.

- **Relaciones:** Esta estrategia está relacionada con la exploración del mapa, la construcción de unidades militares terrestres y la construcción de nuevos asentamientos de posiciones tácticas para elaborar ataques.

Atacar – Asentamiento enemigo secundario – Ataque campal

- **Precondiciones:** Tener localizado un asentamiento enemigo secundario y disponer de un conjunto numeroso de unidades militares. Hay que tener en cuenta que lo que puede parecer un asentamiento secundario puede convertirse en su base principal, ya que puede que antes de realizar el ataque, no se haya descubierto todo el asentamiento.

- **Post-condiciones:** Destrucción total o parcial de la base secundaria enemiga y la posibilidad de perder parte o todos los efectivos que han realizado el ataque. Probabilidad alta de determinar con certeza el tipo de asentamiento enemigo.

- **Relaciones:** Esta estrategia está relacionada con la exploración del mapa y con la construcción de unidades militares terrestres y con la construcción de nuevos asentamientos de posiciones tácticas para elaborar ataques.

Construcción de unidades – Unidades de recolección y construcción

- **Precondiciones:** Disponer de un edificio *Control Center* y tener disponible la cantidad de minerales necesaria para la construcción de las unidades. Es necesario recordar que en el escenario actual, no se tienen que controlar las unidades que se pueden mantener, por lo que a la hora de construir unidades esta restricción no hay que tenerla en cuenta.

- **Post-condiciones:** Disminución de la cantidad de minerales disponible y aumento de las unidades *worker*.

- **Relaciones:** Esta estrategia está relacionada con la recolección de minerales y con la expansión o creación de bases.

Construcción de unidades – Unidades militares terrestres

- **Precondiciones:** Disponer de un edificio *Barracks* y/o *Factory*, tener disponible la cantidad de minerales necesaria para la construcción de las unidades y tener la necesidad de aumentar la capacidad de defensa de algún asentamiento o la necesidad de elaborar un ataque en el que se necesiten unidades con las que no se cuenta en el momento actual.

- **Post-condiciones:** Disminución de la cantidad de minerales disponible y aumento de las unidades militares disponibles.

- **Relaciones:** Esta estrategia está relacionada con la recolección de minerales, la defensa de la base principal o la base secundaria, la elaboración de ataques y la creación de una nueva base de recolección, ya que para fundarla es necesario enviar unidades militares para asegurar el nuevo asentamiento frente a ataques por parte del enemigo.

4.4. Capa General

Uno de los principales objetivos del proyecto es que el cliente sea eficiente y para ello, se necesita que la toma de decisiones sea rápida y coordinada. Además, hay que tener en cuenta que en el ámbito actual, la incertidumbre juega un papel muy importante. Por estas razones, se ha decidido implementar el modelo de redes Bayesianas mediante *GeNIe & SMILE* [1].

La idea inicial consiste en utilizar una única red con varias salidas, donde cada una sea una decisión diferente. Esta idea fue desechada en detrimento de implementar varias redes más sencillas y pequeñas, con el objetivo de aumentar la velocidad y disminuir la complejidad.

Se ha decidido implementar Redes Bayesianas para todas las acciones tácticas, excepto para dos. Ambas carecen casi por completo de incertidumbre, ya que la acción *gather* se va a realizar siempre (sólo hay que decidir dónde realizarla) y la acción *scouting* se realiza bajo dos situaciones bien definidas (ubicación desconocida de la base enemiga y carencia de escenario conocido).

Por cada acción (atacar, defender, construir edificios, entrenar unidades y fundar nuevas bases) se va a utilizar una única red, excepto para la construcción de edificios y el entrenamiento de unidades, donde se van a utilizar dos.

De cada red se va a mostrar su topología, una breve explicación de las entradas y la definición de la DPC (distribución de probabilidades condicionales) de los nodos intermedios y de salida, acompañada de una breve descripción. Por último, se van a mostrar ejemplos para comprobar que el comportamiento de cada red es el deseado, aunque hay que tener en cuenta que un buen comportamiento individual no implica que en el colectivo se comporten igual, por lo que no se descarta un posterior reajuste de las mismas.

Para todas las redes hay que tener en cuenta que todos los nodos de entrada tienen definidas dos variables, cuyos valores son probabilidades complementarias entre sí. Además, en los casos que hay que calcular valores en función de rangos, si el valor actual es superior o inferior a los límites, se cambiará dicho valor actual por el límite más cercano.

4.4.1. Red Bayesiana de ataque

La red bayesiana de ataque aportará el porcentaje con el que deberíamos realizar un ataque a una base enemiga, ya sea secundaria o principal. Como se puede apreciar en la *ilustración 20*, es una red sencilla con sólo cuatro nodos de entrada y un nodo intermedio.

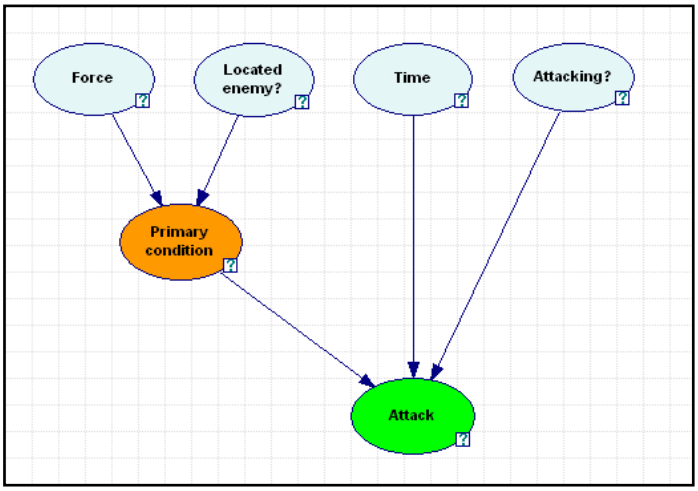

Ilustración 20: Topología de la red Bayesiana para ataque

Antes de comenzar a explicar el diseño interno de la red, hay que conocer los datos que recibe, para así poder comprender mejor la definición de los DPC de los nodos. La información recibida es la siguiente:

- **Force:** Para medir la fuerza militar de la que se dispone, se ha creado la siguiente formula: *Force = Número_de_marines + (número_de_tank * 3)*[15]. Además, se ha decidido limitar el número máximo de unidades militares disponibles en cuarenta, ya que se considera que un número mayor puede provocar problemas de coordinación en los movimientos de las unidades. Si a esto le sumamos el ratio definido anteriormente entre las unidades *marine* y *tank*, obtenemos que la fuerza militar máxima de la que se va a disponer es de sesenta y la mínima de cero. El procedimiento para calcular los valores de las variables *force* y *noForce* del nodo es el siguiente:

    - Aplicar la fórmula del calculo del poder militar actual.

    - Dividir el resultado entre sesenta (poder militar máximo) y almacenar ese valor en la variable *force* del nodo.

    - Calcular el valor de la variable *noForce* mediante la propiedad de probabilidades complementarias.

- **Located enemy?:** Para poder atacar una base enemiga, es necesario saber su ubicación, ya que en caso contrario, el ataque no puede llevarse a cabo.

- **Time:** Indica la probabilidad que existe de que el tiempo de la partida se haya consumido. Se sabe que toda partida del tercer escenario dura exactamente veinte minutos, por lo que los pasos necesarios para obtener los valores de las variables *finished* y *unfinished*, son los siguientes:

    - Calcular el tiempo transcurrido en segundos.

    - Dividir el tiempo transcurrido entre mil doscientos (veinte minutos expresados en segundo) y almacenar el resultado en la variable *finished*.

[15] Se ha decidido que las unidades *tank* cuentan tres veces más que los marines porque provocan cinco veces mas daño en el ataque, tienen más del triple de puntos de salud y porque no pueden atacar a unidades cercanas a su posición.

- Restar la unidad entre el valor obtenido en el paso anterior y almacenarlo en la variable *unfinished*.

- **Attacking?:** Hacer dos ataque a la vez no suele ser buena idea, ya que cada uno no cuenta con muchas unidades o la bases se quedan muy debilitadas. Por ello, para tomar la decisión sobre si lanzar una ataque o no sobre una base enemiga, es aconsejable saber si ya se está llevando a cabo uno.

La red cuenta con un único nodo intermedio, cuyo DPC está definido en la *ilustración 21*. Dicho nodo, representa la condición primaria para realizar el ataque, es decir, indica si se reúnen las condiciones necesarias para llevarlo a cabo. Como se puede observar, el conocimiento de la ubicación de la base enemiga es muy relevante, aunque también es necesario disponer de un poder militar elevado, ya que es mejor hacer un ataque con diez unidades, que no tres con tres unidades cada uno.

Force	force		noForce	
Located enemy?	localized	nonLocalized	localized	nonLocalized
primaryOK	1	0	0.3	0
primaryKO	0	1	0.7	1

Ilustración 21: DPC del nodo *Primary Condition* de la red Bayesiana para ataque

En la *ilustración 22* se puede observar el DPC del nodo de salida. Su definición muestra la importancia de la condición primaria, ya que si esta no se cumple, el resultado de la red va a ser un valor muy bajo. Por otro lado, cuando la condición se cumple, hay dos aspectos que hacen variar el resultado, el tiempo transcurrido de la partida (a medida que el tiempo avanza, se incita a realizar más ataques) y si se está ejecutando un ataque.

Primary condition	primaryOK				primaryKO			
Time	finished		unfinished		finished		unfinished	
Attacking?	attacking	noAttack	attacking	noAttack	attacking	noAttack	attacking	noAttack
success	0.8	1	0.4	0.8	0.1	0.15	0.05	0.1
failure	0.2	0	0.6	0.2	0.9	0.85	0.95	0.9

Ilustración 22: DPC del nodo *Attack* de la red Bayesiana para ataque

Por último, se van a mostrar los resultados obtenidos de ejecutar un conjunto de pruebas, las cuales se dividen en grupos, dónde cada uno persigue una finalidad diferente. Para llevarlo a cabo, se va a utilizar una tabla, en la que cada columna representa un nodo de entrada, excepto la última, que representa el nodo de salida. Los resultados se muestran en la *tabla 11*.

force	localized	finished	attacking	success
1	0	0.5	0	**0.125**
0.2	1	0.5	0	**0.466**
1	1	0.5	0	**0.9**
0.7	1	0.5	1	**0.49**
0.7	1	0.5	0	**0.73**
0.7	1	0.2	0	**0.68**
0.7	1	0.8	0	**0.78**

Tabla 11: Resultados de la red bayesiana para ataque

El objetivo del primer grupo, es ver cómo se comporta la red cuando no se ha localizado la base enemiga. El resultado es satisfactorio, ya que al ser un valor tan próximo a cero, es imposible que el ataque se lleve a cabo.

El segundo grupo, estudia la relevancia del poder militar a la hora de tomar una decisión. El resultado, confirma que con un valor bajo (0.2), la salida no es satisfactoria (0.466), mientras que si contamos con el máximo poder militar, el resultado asciende más del doble (0.9). Hay que resaltar, que en la situación del segundo caso, no se ha obtenido el valor máximo y esto es debido al tiempo transcurrido, ya que sólo cuando esté prácticamente consumido, la variable *success* de la red tomará el valor 1.0.

En el tercer grupo, se establece una situación constante y se varía el valor de la variable *attacking*, ya que se pretende ver cómo disminuye el resultado, al estar realizando un ataque. El resultado confirma que no es recomendable realizar más de un ataque en paralelo, ya que muestra una diferencia del 24%.

Para finalizar, se quiere estudiar la influencia de la cantidad de tiempo consumido. Concretamente, se pretende verificar la siguiente hipótesis: El aumento del tiempo transcurrido, provoca una mayor firmeza a la hora de tomar la decisión de ataque. A la luz de los resultados, la hipótesis queda confirmada, con una variación notable del tiempo, hay una diferencia en el resultado de 0.1.

4.4.2. Red Bayesiana de defensa

En cuanto a la defensa, se ha implementado una red algo más compleja que la anterior, ya que cuenta con cinco nodos de entrada y dos intermedios. Su topología puede observarse en la *ilustración 23* y es fácil ver a simple vista, que el nodo *Defense* nos indicará la medida en la que tenemos que defender la base.

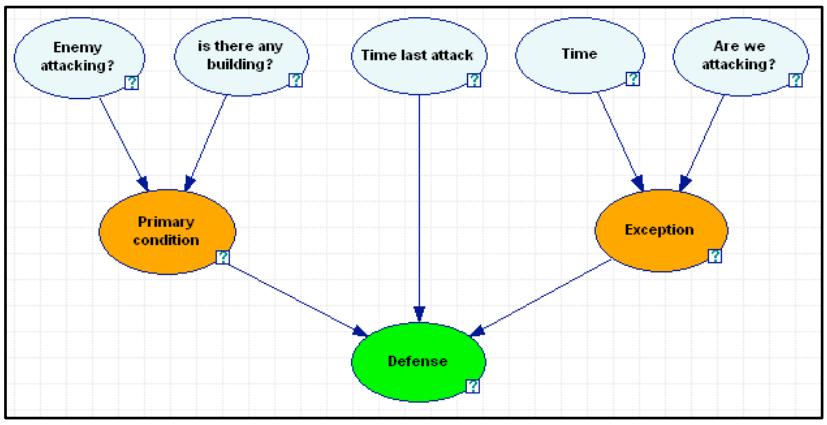

Ilustración 23: Topología de la red Bayesiana para defensa

Una vez expuesta la topología, el siguiente paso es la descripción de los nodos de entrada, para así facilitar la comprensión de la definición de los DPC de cada nodo de la red. Los nodos de entrada son:

- **Enemy attacking?:** El principal motivo para defender una base, es que el enemigo puede realizar ataques en cualquier momento. Por ello, es necesario conocer si se está sufriendo algún ataque.

- **Is there any building?:** Señala si la base cuenta con algún edificio, ya que puede ocurrir que después de repeler un ataque enemigo, no haya quedado ninguna construcción en pie.

- **Time last attack:** Cuando el enemigo realiza un ataque y este es desbaratado, lo normal es que vuelva a entrenar unidades para volver a intentarlo. Por ello, el tiempo transcurrido desde el último ataque es muy importante, ya que gracias a ese dato podemos intuir cuando se tiene más riesgo de recibir un nuevo ataque. Supongamos que un ataque fuerte enemigo está formado por diez unidades *marine* y tres unidades *tank*. Sabiendo que se tarda siete segundos en entrenar una unidad *marine* y diez y seis segundos en entrenar una unidad *tank*, el tiempo que se tarda en entrenar las unidades del ataque, suponiendo que sólo cuentan con un edificio *Barracks* y un edificio *Factory* y además, que cuentan con los recursos necesarios, es de ochenta segundos. A este tiempo hay que añadirle el tiempo que se tarda en tomar la decisión de ataque y movilizar las tropas (supongamos diez segundos). Esto quiere decir que en noventa segundos desde el último ataque se estará sufriendo uno nuevo. Dicha cifra significa, que en una partida de veinte minutos, se pueden llegar a recibir trece ataques del mismo tipo. Este dato no es para nada real,

ya que hay muchos factores que lo pueden alterar (no contar con los minerales necesarios, contar con más de un edificio de cada tipo, estar entrenando todas las unidades mientras se ejecuta el ataque anterior, tiempo que tarda el enemigo en encontrar por primera vez nuestra base, etc.). Aún así, y ya que ese dato nunca va a ser ni constante ni conocido, vamos a utilizar el dato obtenido como el tiempo máximo que puede pasar entre dos ataques.

Para calcular sus valores de las variables *longTime* y *shortTime* hay que realizar los siguientes pasos:

- Obtener el instante de tiempo actual.
- Restar el valor obtenido con el instante en el que el último ataque fue repelido.
- Dividir el resultado anterior entre noventa y almacenar el resultado de la operación en la variable *longTime*.
- Obtener el valor de la variable *shortTime* utilizando la propiedad de probabilidades complementarias.

- **Time:** Su definición se encuentra en la sección 4.4.1 Red Bayesiana de ataque.

- **Are we attacking?:** Indica si se está llevando a cabo un ataque a una base enemiga. Este dato es muy importante, pues forma parte de la condición de la excepción de la defensa de la base. Dicha excepción sucede cuando se está realizando un ataque y el tiempo está muy cerca de terminarse. Es necesario tenerla en cuenta, porque el tiempo que se tarda en hacer regresar a las tropas puede ser superior al tiempo restante de la partida y se estaría desaprovechando la posibilidad de destruir el campamento rival.

Una vez explicadas las entradas de la red, hay que dar paso a detallar los DPC de cada uno de los nodos intermedios y del nodo salida. El primer nodo intermedio que va a ser detallado es *Primary Condition* (¿Nos están atacando? ¿Tenemos edificios que defender?). Como puede observarse en la *ilustración 24*, la entrada *Enemy attacking?* es la más determinante de las dos, puesto que el valor del nodo va a ser muy bajo, cuando la situación que representa no ocurre.

Enemy attacking?	enemyAttack		enemyNoAttack	
is there any building?	building	noBuilding	building	noBuilding
primaryOk	1	0.6	0.2	0
primaryKo	0	0.4	0.8	1

Ilustración 24: DPC del nodo *Primary condition* de la red Bayesiana para defensa

El siguiente nodo representa la excepción en la defensa de la base y como puede apreciarse en la *ilustración 25*, el aspecto más relevante es el tiempo, ya que si y sólo si el tiempo está próximo a finalizar, hay posibilidad de que la situación actual cumpla la condición.

Time	finished		unfinished	
Are we attacking?	playerAttack	playerNoAttack	playerAttack	playerNoAttack
exceptionOk	1	0.2	0	0
exceptionKo	0	0.8	1	1

Ilustración 25: DPC del nodo *Exception* de la red Bayesiana para defensa

Cuando se cumple la excepción, los valores de salida deben ser muy bajos porque no es rentable intentar defender la base. En el caso de que no se cumpla, si se verifica la condición primaria (se esta recibiendo un ataque y se cuenta o no con edificios en el asentamiento), el tiempo desde el último ataque es indiferente, debido a que el nuevo ataque se está produciendo. En caso contrario, el tiempo desde el último ataque juega un papel muy relevante, ya que en función de su valor, el valor final de la red puede variar notoriamente. Estas afirmaciones fueron utilizadas para definir el DPC del nodo de salida (*ilustración 26*).

Primary condition	primaryOk				primaryKo			
Exception	exceptionOk		exceptionKo		exceptionOk		exceptionKo	
Time last attack	longTime	shortTime	longTime	shortTime	longTime	shortTime	longTime	shortTime
success	0.1	0.1	1	1	0.1	0.1	0.8	0.3
failure	0.9	0.9	0	0	0.9	0.9	0.2	0.7

Ilustración 26: DPC del nodo *Defense* de la red Bayesiana para defensa

A continuación, se van a comentar los resultados obtenidos de ejecutar tres grupos diferentes de pruebas, donde el objetivo de todas, es verificar que la red se comporta de la forma esperada. Para mostrar los resultados, se va a utilizar la notación definida en la sección 4.4.1. Red Bayesiana de ataque.

Se han diseñado tres escenarios para a realización de las pruebas, dónde el primero se centra en las variaciones del resultado en función de la condición primaria. El segundo, tiene como objetivo la importancia del tiempo transcurrido desde el último ataque y el último, estudia la condición de excepción. Todas las pruebas se pueden ver en la *tabla 12*.

enemyAttack	building	longTime	finished	playerAttack	success
0	0	0.1	0.5	0	**0.325**
1	0	0.1	0.5	0	**0.676**
0	1	0.1	0.5	0	**0.442**
1	1	0.1	0.5	0	**0.91**
0	1	0.2	0.5	0	**0.478**
0	1	0.9	0.5	0	**0.73**
1	1	0	0.9	1	**0.19**
1	1	0	0.9	0	**0.838**

Tabla 12: Resultados de la red bayesiana para defensa

Los resultados del primer escenario son mejores de lo esperado, ya que como se puede observar, cuando se cumple la condición primaria, el resultado es de 0.91, mientras que cuando no se cumple para nada, el valor desciende a 0.325, tal y como debería suceder. Los otros dos resultados, hacen referencia a cuando se cumple una de las dos entradas, viéndose claramente que tiene más relevancia el hecho de que nos ataquen, a que la base cuente con alguna construcción.

Con los resultados del segundo escenario, se puede afirmar que el tiempo transcurrido desde el último ataque, es muy relevante a la hora de defender la base.

Por último, podemos verificar la gran diferencia de resultados cuando se cumple la condición de excepción, tal y como se esperaba al diseñar la red bayesiana.

4.4.3. Red Bayesiana de construcción de *barracks*

Como se ha comentado anteriormente, la acción de construir edificios de entrenamiento de unidades militares se ha dividido en dos, ya que cada uno tiene unas precondiciones diferentes y además, en la red bayesiana de construcción de *barracks,* se contempla la posibilidad de que la base actual sea una base de ataque, lo que aumentará la medida en la que tenemos que crear el edificio. Para el edificio *Factory* esto no es necesario contemplarlo, ya que si la base reúne las precondiciones necesarias para construir el edificio es porque se trata de una base de ataque, ya sea porque es la única base con la que se cuenta o porque sea un asentamiento secundario con tal fin. Por ello, el resultado ha sido una red más compleja que todas las anteriores, ya que como puede verse en su topología (*ilustración 27*), cuenta con ochos nodos de entrada y tres intermedios.

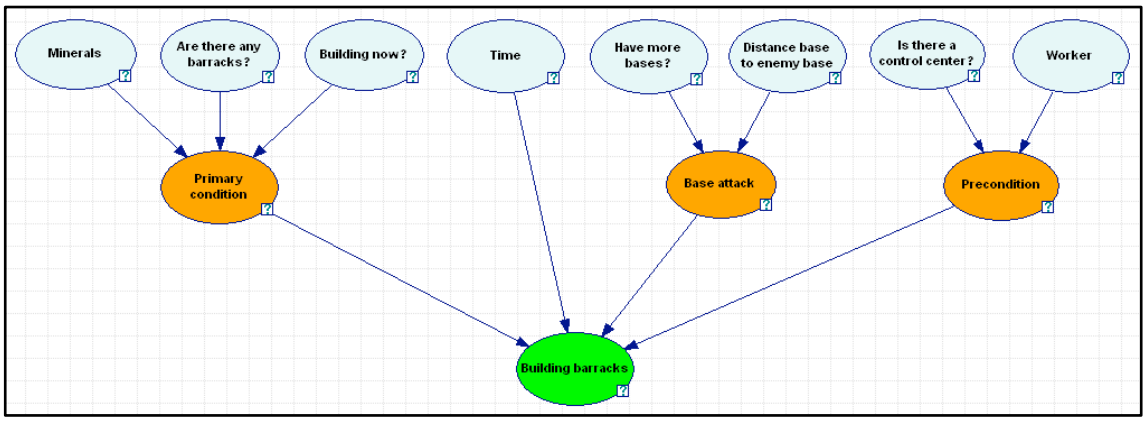

Ilustración 27: Topología de la red Bayesiana para construcción de *Barracks*

Siguiendo con la metodología utilizada en las redes anteriores, el siguiente punto a tratar es la definición de los nodos de entrada:

- **Minerals:** La cantidad de minerales disponible, además de ser un requisito indispensable para la construcción de cualquier edificio o entrenamiento de cualquier unidad, es un factor a tener en cuenta a la hora de realizar una inversión grande, ya que aunque se cuente con la cantidad necesaria, hay que pensar qué va a pasar después. Por ejemplo, si se dispone de la cantidad exacta y lo invertimos todo en la construcción de un edificio, habría que esperar (sin poder hacer nada) a que las unidades *worker* volviesen a recolectar una cantidad significativa de minerales (cien unidades, por ejemplo). Esta es una situación que hay que evitar, ya que es preferible invertir parte del recurso en entrenamiento de más unidades *worker* o en unidades militares y hacer inversiones en edificios cuando se disponga de una economía más holgada.

  Debido a que no existe un baremo para medir el poder económico actual, se han creado dos, uno para inversiones de construcción de edificios y otro para la inversión de entrenamiento de unidades. Ambos utilizan la siguiente fórmula, pero aplicando diferentes valores: *minerals = (Σ coste_individual) * 2*, donde el coste individual hace referencia al coste de realizar cualquier inversión del mismo tipo (edificio o unidad). Por ejemplo, para el caso de los edificios, la fórmula expandida sería la siguiente: *minerals = (coste_control_center + coste_barracks + coste_factory) *2*. Para poder contemplar un rango económico amplio, se decidió incluir la multiplicación en la formula. De esta manera, el límite superior para los edificios es de dos mil ochocientas unidades de minerales, mientras que para el entrenamiento, la cifra desciende hasta las seiscientas unidades.

El nodo está formado por dos variables, las cuales van a almacenar valores de probabilidades complementarias entre sí. Por otro lado, el cálculo de los valores de sus dos variables (*few* y *many*) es sencillo y basta con seguir los siguientes pasos:

- Dividir la cantidad de minerales actual entre el valor máximo contemplado en función de la red actual (dos mil ochocientos para edificios y seiscientos para entrenamiento de unidades).
- Almacenar el resultado anterior en la variable *many*.
- Calcular el valor de la variable *few* en función del resultado anterior.

- **Are there any barracks?:** Antes de construir un edificio, hay que ver si ya está disponible, ya que la construcción de un edificio repetido sólo se llevará a cabo si es totalmente necesario.

- **Building now?:** La construcción de un edificio requiere tiempo, por ello, hay que tener en cuenta si la acción que se va a plantear, ya se está llevando a cabo.

- **Time:** Esta entrada fue definida en el apartado 4.4.1 Red Bayesiana de ataque.

- **Have more bases?:** Las dos variables del nodo responden a la siguiente pregunta: ¿Se cuenta con más bases?. Por este motivo, solo pueden tomar el valor cero o uno.

- **Distance base to enemy base:** Una base de ataque se suele caracterizar por la colocación de la misma con respecto al enemigo. Por este motivo, hay que tener en cuenta la distancia de la base actual, a la base enemiga más cercana. Por la definición del tercer escenario, sabemos que la mayor distancia posible en el mismo es de mil cuatrocientos cuarenta y ocho puntos. Gracias a este dato, se puede establecer la medida de lejanía o cercanía entre las dos bases. Para calcular los valores de las variables (*small* y *big*) del nodo hay que realizar los siguientes pasos:
  - Calcular la distancia euclídea entre las dos bases.
  - Dividir el resultado anterior entre mil cuatrocientos cuarenta y ocho.

- Almacenar el resultado en la variable *big* y calcular el valor de la variable *small*, utilizando la propiedad de probabilidades complementarias.

- **Is there a control center?:** Por restricciones del escenario, no se puede construir el edificio b*arracks* sin disponer de al menos un *control center*.

- **Worker:** La última entrada se corresponde con una precondición importante, la existencia de al menos una unidad *worker*. Si no se cumple, la construcción no puede llevarse a cabo, porque no hay ninguna unidad capacitada para ello.

La red cuenta con tres nodos intermedios, lo cuales se corresponden con la condición primaria (¿debemos construir?), la condición de base de ataque y las precondiciones.

Para responder a la pregunta de la condición primaria, hay que tener en cuenta si se dispone ya del edificio que se quiere construir, si se está construyendo y el poder adquisitivo actual, tal y como puede apreciarse en la *ilustración 28*.

Minerals	few				many			
Are there any barracks?	yes		no		yes		no	
Building now?	yes	no	yes	no	yes	no	yes	no
primaryOk	0	0.2	0.1	0.7	0.2	0.6	0.3	1
primaryKo	1	0.8	0.9	0.3	0.8	0.4	0.7	0

Ilustración 28: DPC del nodo *Primary condition* de la red Bayesiana para construcción de *Barracks*

De estos tres factores, el menos determinante es el poder adquisitivo, ya que aunque sea alto, si se dispone de un edificio y hay otro en construcción, no habría que construir más por el momento. Además, hay que intentar que el valor sea menor cuando no se tiene ninguno y se está construyendo uno, que cuando ya hay uno disponible y no hay ninguno en construcción. El motivo de esta premisa es la paciencia, es decir, hay que mirar al futuro, por lo que si ya estoy construyendo un edificio, hay que olvidarse de la posibilidad de construir otro y con los recursos disponibles realizar otras acciones.

Por otro lado, hay que determinar si una base está bien situada para ser considerada de ataque. Si es así, esta condición empujará a tomar la decisión de construir el edificio, como se puede apreciar en la *ilustración 29*. Como se puede observar, si sólo se cuenta con una base, ésta va a ser considerada de ataque, por lo que no hay que motivar la construcción.

Have more bases?	yes		no	
Distance base to enemy base	small	big	small	big
baseAttack	0.9	0.1	0	0
noBaseAttack	0.1	0.9	1	1

Ilustración 29: DPC del nodo *Base Attack* de la red Bayesiana para construcción de *Barracks*

Cumplir las precondiciones para poder construir el edificio es algo imprescindible, además, hay que cumplirlas todas (tal y como puede verse en la *ilustración 30*), es decir, disponer de un edificio *control center* y de una unidad *worker*.

Is there a control center?	yes		no	
Worker	yes	no	yes	no
preconditionOk	1	0	0	0
preconditionKo	0	1	1	1

Ilustración 30: DPC del nodo *Precondition* de la red Bayesiana para construcción de *Barracks*

Debido al tamaño del DPC del nodo de salida, se ha tenido que dividir en dos (*ilustración 31* e *ilustración 32*). En él, se recogen todas las combinaciones de los tres nodo intermedios, tal y como se puede apreciar. El valor del nodo de las precondiciones es un valor excluyente, es decir, si no se cumplen, el edificio no se construye. Por otro lado, se puede observar la manera en la que influye la condición de base de ataque, ya que cuando se cumple incita más a la construcción del edificio. Por último señalar, que si no se debe construir, el valor de salida va a ser muy bajo, ya que los únicos valores que influyen levemente son el tiempo de la partida y la condición de base de ataque.

Primary condition	primaryOk								primar...
Base attack	baseAttack				noBaseAttack				...
Precondition	preconditionOk		preconditionKo		preconditionOk		preconditionKo		...
Time	finished	unfinished	finished	unfinished	finished	unfinished	finished	unfinished	...
success	0.95	1	0	0	0.8	1	0	0	...
failure	0.05	0	1	1	0.2	0	1	1	...

Ilustración 31: DPC del nodo *Building Barracks* de la red Bayesiana para construcción de *Barracks*

Primary condition	primar...	primaryKo							
Base attack	...	baseAttack				noBaseAttack			
Precondition	...	preconditionOk		preconditionKo		preconditionOk		preconditionKo	
Time	...	finished	unfinished	finished	unfinished	finished	unfinished	finished	unfinished
success	...	0.1	0.3	0	0	0.05	0.2	0	0
failure	...	0.9	0.7	1	1	0.95	0.8	1	1

Ilustración 32: DPC del nodo *Building Barracks* de la red Bayesiana para construcción de *Barracks*

Para finalizar con la red de construcción del edificio *Barracks*, se van a mostrar los resultados de una serie de situaciones, definidas para realizar una pequeña evaluación de la red.

few	yes - barracks	yes - now	finished	yes - bases	small	yes - cc	yes - worker	success
0.7	0	0	0.5	0	0.5	1	1	**0.737**
0.7	0	0	0.5	0	0.5	0	1	**0**
0.7	1	0	0.5	1	0.3	1	1	**0.423**
0.7	1	0	0.5	0	0.3	1	1	**0.373**
0.7	1	1	0.5	0	0.5	1	1	**0.172**
0.7	0	0	0.5	0	0.5	1	1	**0.737**
0.3	0	0	0.5	0	0.5	1	1	**0.83**
0.3	1	1	0.5	0	0.5	1	1	**0.233**
0.8	1	0	0.2	0	0.5	1	1	**0.391**
0.8	1	0	0.8	0	0.5	1	1	**0.293**

Tabla 13: Resultados de la red bayesiana para construcción de *Barracks*

La primera situación que se ha definido, abarca los dos primeras filas de la *tabla 13* y se centra en la diferencia entre cumplir y no cumplir las precondiciones. El resultado es bastante contundente, ya que cuando no se cumplen, la red obtiene como resultado el valor cero.

La siguiente situación, tiene como objetivo estudiar el efecto que provoca el cumplimiento de la condición de base de ataque. Si nos fijamos en el segundo grupo de la *tabla 13*, podemos comprobar que cuando se cumple la condición, el valor de salida aumenta sensiblemente, aunque hay que tener en cuenta, que en la situación actual no se debería construir (porque no se dispone de muchos minerales y además, ya se cuenta con un edificio *barracks*), por lo que el efecto aquí es menor que cuando sí se debe.

En la tercera definición del escenario, se han introducido dos aspectos a estudiar: el efecto del poder adquisitivo y el efecto del número de *barracks* disponible. Si observamos los resultados de la *tabla 13*, podemos ver como varían los resultados al alterar dichos aspectos.

Las dos últimas filas de la *tabla 13*, sirven para comprobar la reacción de la red con el tiempo actual de la partida. Si nos fijamos en los resultados, podemos ver como el tiempo influye negativamente a medida que éste se consume. Este resultado es el esperado, ya que a medida que el tiempo avanza, interesa más el entrenamiento de unidades militares que la construcción de nuevos edificios.

4.4.4. Red Bayesiana de construcción de *Factory*

Como puede observarse en la *ilustración 33*, la red bayesiana para la construcción del edificio *Factory* es similar a la construcción del edificio *Barracks*, a excepción de lo comentado

en el subapartado anterior, es decir, la evaluación de la posibilidad de que la base actual sea de ataque. Por ello, esta es más sencilla y precisa de menos entradas.

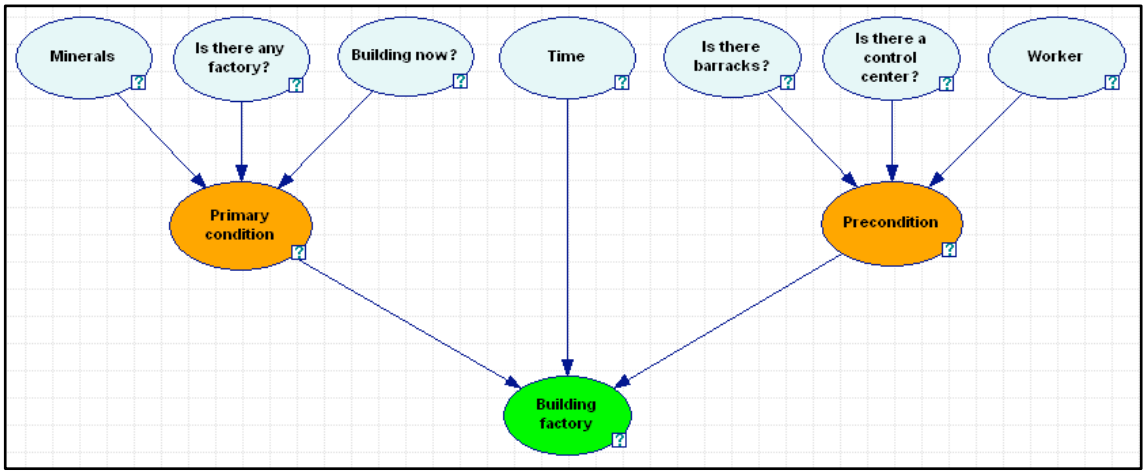

Ilustración 33: Topología de la red Bayesiana para construcción de *Factory*

A continuación, se van a mostrar todas las entradas de la red y sólo se van a explicar tres de ellas, ya que el resto ya han sido detalladas en otros subapartados. Las entradas de la red son:

- **Minerals:** La explicación detallada de esta entrada se encuentra en el apartado 4.4.3 Red Bayesiana de construcción de *barracks*.

- **Is there any factory?:** La construcción de un edificio es una inversión considerable, por lo que hay que estudiar su repercusión y necesidad. Por ello, para tomar una decisión coherente, se necesita saber si el edificio está ya disponible.

- **Building now?:** Para acceder a su definición, véase la sección 4.4.3 Red Bayesiana de construcción de *barracks*.

- **Time:** Esta entrada fue detallada en la sección 4.4.1 Red Bayesiana de ataque.

- **Is there barracks?:** Por restricciones del escenario, no se puede construir el edificio *factory* sin disponer de al menos un *barracks*.

- **Is there a control center?:** Por restricciones del escenario, no se puede construir el edificio *factory* sin disponer de al menos un *control center*.

- **Worker:** Véase el apartado 4.4.3 Red Bayesiana de construcción de *barracks*.

Debido a la gran similitud con la red del apartado anterior, el DPC del nodo *Primary condition* contiene los mismos valores, por lo que simplemente se va a mostrar (*ilustración 34*), pero no se va a comentar nada, con el propósito de repetir la menor información posible.

Minerals	few				many			
Is there any factory?	yes		no		yes		no	
Building now?	yes	no	yes	no	yes	no	yes	no
primaryOk	0	0.2	0.1	0.7	0.2	0.6	0.3	1
primaryKo	1	0.8	0.9	0.3	0.8	0.4	0.7	0

Ilustración 34: DPC del nodo *Primary condition* de la red Bayesiana para construcción de *Factory*

Tal y como se puede deducir de la *ilustración 35*, para que se cumpla la precondición necesaria para la construcción del edificio *factory*, se debe disponer de un *control center*, de un edificio *barracks* y de una unidad *worker*.

Is there barracks?	yes				no			
Is there a control center?	yes		no		yes		no	
Worker	yes	no	yes	no	yes	no	yes	no
preconditionOk	1	0	0	0	0	0	0	0
preconditionKo	0	1	1	1	1	1	1	1

Ilustración 35: DPC del nodo *Precondition* de la red Bayesiana para construcción de *Factory*

La definición del DPC del nodo de salida de la red, es el mismo que el de la red del apartado anterior, excluyendo los valores relacionados con la posibilidad de que la base sea considerada de ataque. Por este motivo, simplemente se va a mostrar mediante la *ilustración 36* su definición.

Primary condition	primaryOk				primaryKo			
Time	finished		unfinished		finished		unfinished	
Precondition	preconditionOk	preconditionKo	preconditionOk	preconditionKo	preconditionOk	preconditionKo	preconditionOk	preconditionKo
success	0.8	0	1	0	0.005	0	0.2	0
failure	0.2	1	0	1	0.995	1	0.8	1

Ilustración 36: DPC del nodo *Building factory* de la red Bayesiana para construcción de *Factory*

No obstante, si se va a proceder a la verificación del comportamiento de la red. Dicho proceso, se va a realizar utilizando la misma notación que en los apartados anteriores. Para esta red, se han definido un conjunto de escenarios en los que se va a estudiar el comportamiento del nodo de salida.

Para ser más exacto, se han definido cuatro escenarios, dónde el primero estudia la respuesta de la red cuando no se cumplen las precondiciones. El segundo se centra en el estudio de la red cuando se varían las preguntas sobre si se posee ya un *factory* y sobre si está en construcción. Por otro lado, el objetivo del tercero es variar el poder adquisitivo de la base. Y

por último, el cuarto se centra en el tiempo transcurrido. Todos los resultados se pueden ver en la *tabla 14*.

few	yes - factory	yes - now	finished	yes - barracks	yes - cc	yes - worker	succes
0.3	0	0	0.5	0	1	1	**0**
0.5	0	0	0.4	1	1	1	**0.8**
0.5	1	0	0.4	1	1	1	**0.441**
0.5	0	1	0.4	1	1	1	**0.282**
0.5	1	1	0.4	1	1	1	**0.202**
0.2	1	0	0.5	1	1	1	**0.517**
0.8	1	0	0.5	1	1	1	**0.326**
0.3	1	0	0.2	1	1	1	**0.545**
0.3	1	0	0.8	1	1	1	**0.426**

Tabla 14: Resultados de la red bayesiana para construcción de *Factory*

Hay que destacar los resultados del segundo escenario, que ocupan el segundo grupo de la tabla. Si nos fijamos bien, se puede observar que el resultado es menor cuando se está construyendo el edificio y no se tiene ninguno disponible, que cuando se tiene disponible y no hay ninguno en construcción. Esto era un comportamiento muy deseado, ya que refleja que es mejor esperar a que se termine de construir e invertir los recursos en otra tarea.

Por otro lado, los resultados del tercer escenario reflejan claramente la importancia del poder adquisitivo a la hora de realizar la inversión. Por último, observando los resultados del cuarto escenario, comprobamos que la red tiende a disminuirlos con el paso del tiempo.

4.4.5. Red Bayesiana de entrenamiento de w*orkers*

La red bayesiana obtenida como resultado de analizar los factores que condicionan el entrenamiento de nuevas unidades *worker,* se puede observar en la *ilustración 37*. Dicha red, a partir de la situación actual, informará del esfuerzo que hay que realizar para el entrenamiento de nuevas unidades, es decir, si la red obtiene valores bajos en su salida, significará que no es necesario el entrenamiento o que las condiciones actuales no son muy favorables para llevarlo a cabo. Por el otro lado, si el valor de salida es elevado, querrá decir que es necesario y que las condiciones actuales son buenas para ello.

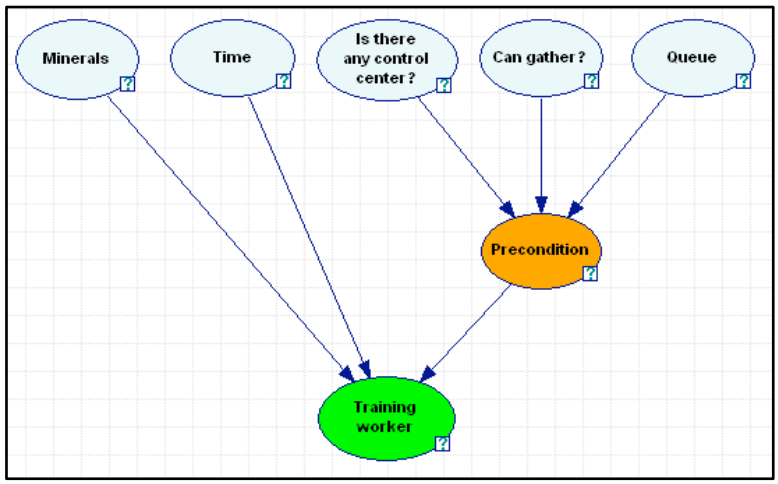

Ilustración 37: Topología de la red Bayesiana para entrenamiento de *Workers*

Antes de comenzar con las definiciones de los DPC de los nodos relevantes de la red, se van a describir brevemente los valores de las entradas, las cuales son:

- **Minerals:** Su definición se encuentra en la sección 4.4.3 Red Bayesiana de construcción de *barracks*.

- **Time:** La descripción de la entrada se encuentra en el apartado 4.4.1 Red Bayesiana de ataque.

- **Is there any control center?:** Para poder entrenar la unidad *worker* se necesita el edificio *control center*, ya que es el encargado de realizar el entrenamiento.

- **Can gather?:** El tercer escenario de ORTS, limita en cuatro el número de unidades que pueden recolectar al mismo tiempo sobre una fuente individual de minerales. Sabiendo que cada *cluster* de minerales está formado por diez fuentes, el número máximo de unidades que pueden recolectar en un *cluster* completo es de cuarenta. El problema comienza cuando las fuentes empiezan a consumirse, ya que habría unidades sin poder recolectar. Además, hay que tener en cuenta que tantas unidades moviéndose en el mismo espacio, pueden provocar problemas de coordinación de movimientos. Una posible solución, es limitar el número de unidades que pueden recolectar en un *cluster*, con el objetivo de evitar la existencia de unidades sin trabajo asignado y evitar el problema de coordinación. El límite de unidades que se ha decidido establecer es diez, ya que de esta forma sólo empezaría a haber unidades sin trabajo, cuando queden menos de tres fuentes. Además, con diez unidades recolectando sin parar, se consigue un buen flujo de entrada de minerales.

Como el nombre de la entrada es una pregunta, los nombres de las variables del nodo son *yes* y *no*. La variable *yes* almacenará el valor uno, cuando en al menos un *cluster* activo, existan menos de diez unidades *worker*.

- **Queue:** En cada edificio de entrenamiento de unidades, sólo se puede entrenar una unidad a la vez, pero se puede crear una cola de unidades a entrenar. Esto es una característica muy útil si se utiliza con coherencia, ya que utilizar una cola ilimitada es un desperdicio de minerales. Por ello, se ha decidido limitar las colas de entrenamiento a seis peticiones, ya que tener una cola de mayor tamaño no es bueno, porque se están paralizando minerales que podrían utilizarse en otras tareas.

  Aunque el nombre de la entrada no es una pregunta, los nombres de sus variables son *yes* y *no*. Para que el valor uno se almacene en la variable *yes*, debe existir un edificio de entrenamiento de la unidad que se desea conseguir cuya cola albergue menos de seis peticiones.

Para tomar una decisión sobre el entrenamiento de una nueva unidad *worker*, hay que tener en cuenta las precondiciones necesarias para poder llevarlo a cabo, la cantidad de minerales disponible y el tiempo transcurrido de la partida.

Como puede observarse en la *ilustración 38*, todas las precondiciones son igual de relevantes, ya que con que sólo una de ellas no se cumpla, el nodo que las representa, toma el valor cero en su variable *preconditionOk*. Este tipo de DPCs implementan lo que se llama propagación discreta.

Is there any control center?	yes				no			
Can gather?	yes		no		yes		no	
Queue	full	noFull	full	noFull	full	noFull	full	noFull
preconditionOk	0	1	0	0	0	0	0	0
preconditionKo	1	0	1	1	1	1	1	1

Ilustración 38: DPC del nodo *Precondition* de la red Bayesiana para entrenamiento de *Workers*

Además, las precondiciones son excluyentes, es decir, si no se cumplen, no se entrena ninguna unidad. Por otro lado, se tiende a entrenar menos unidades *worker*, a medida que avanza el tiempo, tal y como puede deducirse al examinar la *ilustración 39*.

Precondition	preconditionOk				preconditionKo			
Minerals	few		many		few		many	
Time	finished	unfinished	finished	unfinished	finished	unfinished	finished	unfinished
success	0.6	0.8	0.7	1	0	0	0	0
failure	0.4	0.2	0.3	0	1	1	1	1

Ilustración 39: DPC del nodo *Training worker* de la red Bayesiana para entrenamiento de *Workers*

Por último, se van a definir una serie de situaciones con el objetivo realizar un breve análisis de resultados. Concretamente, se van a definir tres situaciones diferentes. La primera tiene como objetivo, observar el valor de salida de la red cuando no se cumple una de las precondiciones. La segunda, se centra en el estudio de la cantidad de minerales disponible en la base. Y la última, en el efecto que provoca el paso de tiempo de la partida. Todos los resultados, se pueden ver en la *tabla 15*.

few	finished	yes – control center	yes - gather	full	success
0.5	0.5	1	1	1	**0**
0.2	0.5	1	1	0	**0.82**
0.5	0.5	1	1	0	**0.775**
0.8	0.5	1	1	0	**0.73**
0.5	0.2	1	1	0	**0.85**
0.5	0.5	1	1	0	**0.775**
0.5	0.9	1	1	0	**0.675**

Tabla 15: Resultados de la red bayesiana para entrenamiento de *Workers*

La principal conclusión que se obtiene al estudiar los resultados, es que el poder adquisitivo de la base no es muy relevante (hay que tener en cuenta que el entrenamiento de la unidad *worker* es de cincuenta unidades de mineral), al contrario que el tiempo, el cual provoca más variación en los resultados al adoptar diferentes valores. Dicha resultados eran los que se pretendían obtener al modelizar la red.

4.4.6. Red Bayesiana de entrenamiento de unidades militares

Fue necesaria la división en dos del entrenamiento de unidades, debido a que ni las precondiciones, ni la influencia de la situación actual, ni los objetivos son los mismos para unidades militares que para unidades *worker*. Además, se quiere mantener un ratio de 3:1[16] en unidades marine y *tank*, es decir, por cada unidad *tank* debería haber tres unidades *marine*. En la *ilustración 40* puede verse la topología de la red bayesiana que se va a utilizar para determinar si hay que entrenar, o no, nuevas unidades militares.

[16] Esta decisión viene motivada por las limitaciones de las unidades *tank*, ya que es una unidad lenta a la hora de disparar y no puede atacar a unidades que estén muy cerca.

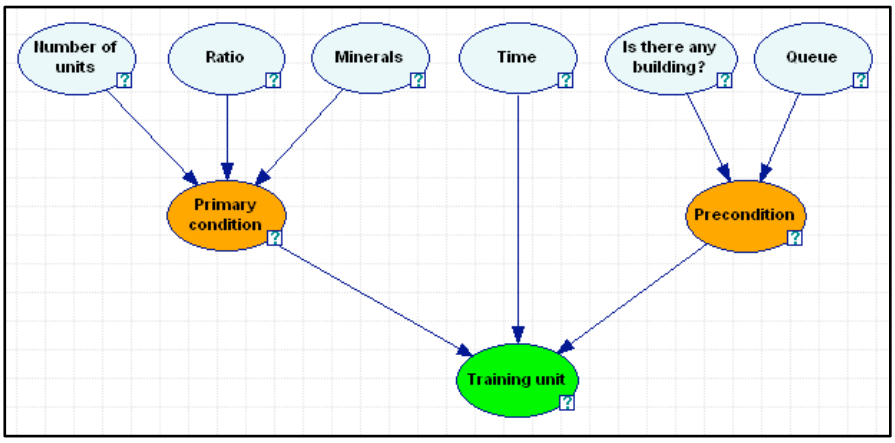

Ilustración 40: Topología de la red Bayesiana para entrenamiento de unidades militares

Algunos de los aspectos que hay que contemplar para entrenar una unidad militar, son diferentes a los contemplados para el entrenamiento de unidades no militares. Por ello, es necesario realizar una breve descripción de sus características. Dichos aspectos son:

- **Number of units:** Al definir la entrada *Force*, se definió el límite máximo de unidades militares que puede haber disponibles, pero en este punto, ese límite se ha aumentado en un tercio, con el objetivo de ganar tiempo entrenando unidades, mientras se empieza o decide realizar el ataque. Además, con esta medida se disminuye la debilitación de la bases al realizar un ataque. Por lo tanto, el número máximo de unidades *marine* es de cuarenta y el de las unidades *tank* de trece. Para calcular los valores de las variables *min* y *max*, se deben seguir las siguientes indicaciones:

  - Obtener el número actual de unidades del tipo que se plantea entrenar.
  - Dividir el valor obtenido entre el valor máximo para el tipo de unidad en cuestión.
  - Almacenar el resultado en la variable *max* y calcular el valor de la variable *min* utilizando las propiedades descritas en su definición.

- **Ratio:** En apartados anteriores, se definió un ratio entre unidades *marine* y *tank*. La entrada actual representa dicho ratio, pero en función del tipo de unidad, los valores se calculan de forma diferente. Para la unidad *marine* se deben realizar los siguientes pasos:

- Dividir el número actual de unidades marine, entre el número actual de unidades *tank*.

- Si el resultado es mayor que tres, se asume que es tres, ya que para la unidad *marine* el ratio es perfecto.

- Si no se dispone de ninguna unidad *tank*, se asume que el resultado es tres, si y sólo si se dispone de alguna unidad *marine*. En caso contrario, se asume que el resultado es cero.

- Dividir el resultado obtenido entre tres.

- Se almacena el resultado obtenido en la variable *ok* y se calcula el valor de la variable *ko*, restando la unidad al valor almacenado en la primera variable.

Cuando la entrada hace referencia a la unidad *tank*, se deben seguir los siguientes pasos:

- Dividir el número actual de unidades *marine*, entre el número actual de unidades *tank*.

- Si no se dispone de ninguna unidad *tank*, se asume que el resultado es seis.

- Si el resultado es menor que tres, se asume que es tres, ya que para la unidad *tank* el ratio es correcto.

- Si el resultado es mayor que seis, se asume que es seis, ya que es el peor resultado que se va a contemplar.

- Dividir el resultado anterior entre tres.

- Restar al resultado la unidad y almacenar el valor en la variable *ko*.

- Calcular el valor de la variable *ok* en función del valor de la variable *ko*.

- **Minerals:** Esta entrada fue definida en el capítulo 4.4.3 Red Bayesiana de construcción de *barracks*.

- **Time:** La explicación detallada de esta entrada se encuentra en la sección 4.4.1 Red Bayesiana de ataque.

- **Is there any building?:** Para poder entrenar una unidad militar, es necesario disponer del edificio que sea capaz de llevarlo a cabo. Debido a que esta red se va a utilizar para decidir si hay que entrenar un *marine* y/o un *tank*, la entrada hará referencia al edificio *barracks* o *factory*.

- **Queue:** Véase la sección 4.4.5 Red Bayesiana de entrenamiento de *workers*.

La condición primaria en el entrenamiento de unidades militares, contesta la siguiente pregunta: ¿Tenemos que entrenar más unidades?. Para responderla, necesitamos saber el número de unidades de las que se dispone, el ratio actual y la cantidad de minerales con los que cuenta la base. Como podemos apreciar en la *ilustración 41*, la entrada mas discriminante para decidir si se tiene que realizar el entrenamiento, es el número de unidades, ya que ese límite no se debería superar.

Number of units	min				max			
Ratio	ko		ok		ko		ok	
Minerals	few	many	few	many	few	many	few	many
primaryOk	0.8	1	0.6	0.8	0.2	0.3	0.05	0.1
primaryKo	0.2	0	0.4	0.2	0.8	0.7	0.95	0.9

Ilustración 41: DPC del nodo *Primary condition* de la red Bayesiana para entrenamiento de unidades militares

Al igual que en la red anterior (entrenamiento de unidades *worker*), es necesario que se cumplan todas las precondiciones, aunque en este caso, como se puede ver en la *ilustración 42*, sólo hay dos.

Is there any building?	yes		no	
Queue	noFull	full	noFull	full
preconditionOk	1	0	0	0
preconditionKo	0	1	1	1

Ilustración 42: DPC del nodo *Precondition* de la red Bayesiana para entrenamiento de unidades militares

Lo más interesante del nodo de salida, cuya definición se puede ver en la *ilustración 43*, es observar la relevancia del tiempo, ya que al contrario que en la mayoría de las redes definidas,

el tiempo cuenta a favor, es decir, mientras menos tiempo reste de partida, más se incita al entrenamiento de unidades militares.

Primary condition	primaryOk				primaryKo			
Precondition	preconditionOk		preconditionKo		preconditionOk		preconditionKo	
Time	finished	unfinished	finished	unfinished	finished	unfinished	finished	unfinished
▶ success	1	0.9	0	0	0.2	0.1	0	0
failure	0	0.1	1	1	0.8	0.9	1	1

Ilustración 43: DPC del nodo *Training unit* de la red Bayesiana para entrenamiento de unidades militares

Para finalizar con el diseño de la red de entrenamiento de unidades militares y siguiendo con la metodología utilizada a lo largo del apartado, se van a definir cuatro posibles situaciones reales, con el objetivo de poder estudiar el comportamiento de la red ante diferentes circunstancias. Los resultados obtenidos se muestran en la *tabla 16*.

Las datos comprendidos entre las filas dos y nueve, corresponden con una situación normal de partida, en la que se ha consumido la mitad del tiempo. El objeto de estudio, es ver la respuesta de la red, a la variación de los datos del número de unidades y del ratio. Como se puede observar, aunque se haya conseguido el ratio deseado, si todavía se pueden entrenar más unidades, la red anima a ello, ya que de esta forma se podrán realizar ataques más poderos y lograr una mejor defensa de la base.

min	ko	few	finished	yes - building	noFull	success
0.5	0.5	0.5	0.5	1	0	**0**
0.2	0.2	0.5	0.5	1	1	**0.339**
0.2	1	0.5	0.5	1	1	**0.454**
0.5	0.2	0.5	0.5	1	1	**0.49**
0.5	1	0.5	0.5	1	1	**0.61**
0.7	0.2	0.5	0.5	1	1	**0.591**
0.7	1	0.5	0.5	1	1	**0.714**
1	0.2	0.5	0.5	1	1	**0.742**
1	1	0.5	0.5	1	1	**0.87**
0.5	0.5	0.2	0.5	1	1	**0.568**
0.5	0.5	0.9	0.5	1	1	**0.491**
0.5	0.5	0.5	0.2	1	1	**0.505**
0.5	0.5	0.5	0.9	1	1	**0.575**

Tabla 16: Resultados de la red bayesiana para entrenamiento de unidades militares

El tercer grupo de la tabla, corresponde a una situación de partida equilibrada, en la que se varía la cantidad de minerales que dispone la base. Tal y como se esperaba, a mayor poder adquisitivo, mayor decisión de entrenamiento de las unidades.

4.4.7. Red Bayesiana para la construcción de nuevas bases

Una de las decisiones más difíciles de tomar en una partida de juegos de estrategia en tiempo real, es la creación de una nueva base, ya que si no sale bien, las consecuencias pueden ser muy perjudiciales. Esta decisión implica invertir muchos recursos (minerales para la construcción de edificios, unidades *worker* para llevarla a cabo y unidades militares para la defensa) y esto suele debilitar la base principal, ya que todos salen de élla. Pero no todo es negativo, porque si no hay problemas y se consigue establecer la base, ésta va a dar muchos beneficios. Para tomar esta decisión se va a utilizar una red bayesiana, cuya topología aparece en la *ilustración 44*.

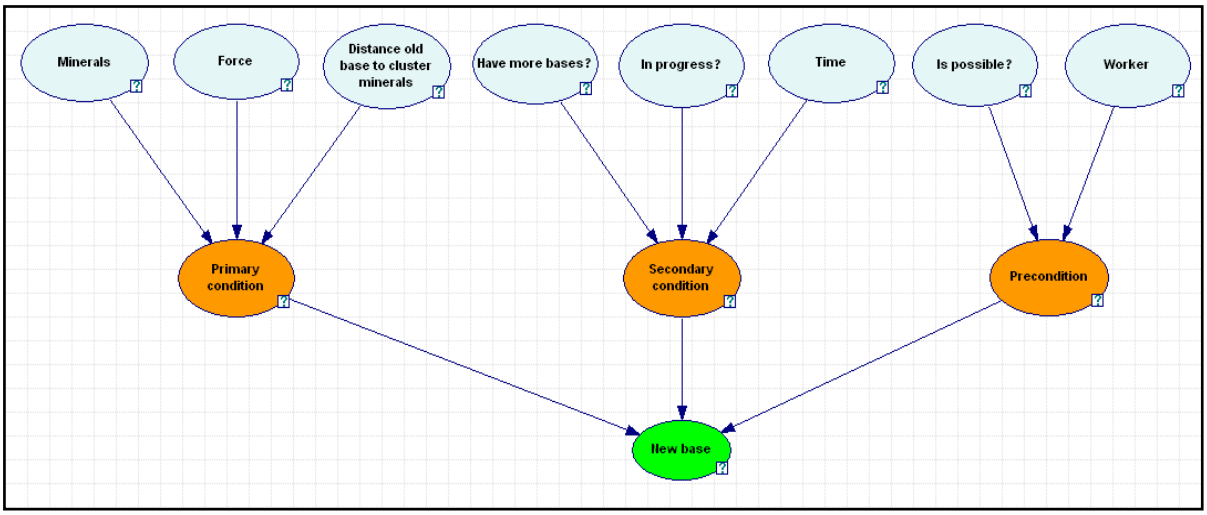

Ilustración 44: Topología de la red Bayesiana para construcción de nuevas bases

A continuación, se van a describir todas las entradas de la red:

- **Minerals:** Su definición se encuentra en la sección 4.4.3 Red Bayesiana de construcción de barracks.

- **Force:** Si se desea leer su descripción, se deberá retroceder al capítulo 4.4.1 Red Bayesiana de ataque.

- **Distance old base to *cluster* minerals:** El principal motivo para fundar una nueva base, es la necesidad de recolectar minerales en fuentes alejadas a todos los *controls centers* disponibles. Hay que tener en cuenta, que cada vez que se funda una base, se le asignan los *clusters* que se consideran cercanos y accesibles. Por ello, dichos

clusters no son computables para determinar la necesidad de recolectar en lugares alejados.

Sabiendo que el *cluster* de minerales inicial, suele estar colocado a una distancia inferior de cien puntos del *control center*, se considerará, que si un *cluster* dista menos de doscientos puntos del *control center* más cercano y es accesible desde el mismo, ese *cluster* será asignado a la base que albergue dicho edificio.

Para tomar la decisión sobre la creación de la nueva base, se necesita saber la distancia del *cluster* de minerales dónde se pretende establecer la nueva, a la base más cercana. Para esta medida, la máxima distancia que se va a contemplar es de cuatrocientos puntos. Por ello, los pasos necesarios para obtener el valor de la variable *big*, son los siguientes:

- Obtener el centro del *cluster* de minerales.
- Calcular la distancia de dicho punto, a todos los *control centers* disponibles.
- Obtener la media aritmética de todos los valores anteriores y dividirla entre cuatrocientos.
- Almacenar el resultado anterior, en la variable *big*.

- **Have more bases?:** Véase el capítulo 4.4.3 Red Bayesiana de construcción de *barracks*.

- **In progress?:** La construcción de un nuevo asentamiento requiere tiempo, por ello, hay que tener en cuenta si la acción que se va a plantear, ya se está llevando a cabo.

- **Time:** En el apartado 4.4.1 Red Bayesiana de ataque, se encuentra la explicación detallada.

- **Is possible?:** El nombre del nodo puede parecer algo ambiguo, ya que pregunta si es posible. Esa pregunta hace referencia a la distancia del *cluster* de minerales, a la base enemiga más cercana, ya que es muy peligroso fundar un nuevo asentamiento al lado de los enemigos. Por ello, la entrada actual indica si es viable la

construcción de la nueva base, basándose en esa distancia. Se considerará, que si la distancia es superior a ciento cuarenta puntos, la variable *yes* del nodo albergará el valor uno.

- **Worker:** Para acceder a su definición, véase la sección 4.4.3 Red Bayesiana de construcción de *barracks*.

En esta red, la condición primaria contesta la pregunta: ¿Podemos fundar una nueva base?. Por ello, se centra en el poder adquisitivo actual, el poder militar y la distancia al *cluster* de minerales dónde se quiere establecer el nuevo asentamiento. Como se puede apreciar en la *ilustración 45*, la cantidad de minerales se nivela con el poder militar, ya que aunque no se disponga de una economía elevada, si el poder militar es evado, el resultado es satisfactorio.

Minerals	few				many			
Force	force		noForce		force		noForce	
Distance old base to cluster minerals	small	big	small	big	small	big	small	big
primaryOk	0.5	0.8	0.25	0.4	0.7	1	0.4	0.6
primaryKo	0.5	0.2	0.75	0.6	0.3	0	0.6	0.4

Ilustración 45: DPC del nodo *Primary condition* de la red Bayesiana para construcción de nuevas bases

Sin embargo, la condición secundaria no contesta ninguna pregunta en especial, más bien se centra en diversos aspectos que pueden hacer variar el resultado final. Como se puede ver en la *ilustración 46*, el tiempo corre en contra de la creación de nuevas bases, como es lógico, porque a medida que el tiempo se consume, la rentabilidad que se puede obtener de la base es menor.

Have more bases?	yes				no			
In progress?	yes		no		yes		no	
Time	finished	unfinished	finished	unfinished	finished	unfinished	finished	unfinished
secondaryOk	0.05	0.2	0.15	0.3	0.1	0.4	0.3	1
secondaryKo	0.95	0.8	0.85	0.7	0.9	0.6	0.7	0

Ilustración 46: DPC del nodo *Secondary condition* de la red Bayesiana para construcción de nuevas bases

En este caso, para satisfacer la precondición obligatoria, sólo se necesita disponer de una unidad *worker* y que la ubicación del nuevo asentamiento sea viable, tal y como se puede apreciar en la *ilustración 47*.

Is possible?	yes		no	
Worker	yes	no	yes	no
preconditionOk	1	0	0	0
preconditionKo	0	1	1	1

Ilustración 47: DPC del nodo *Precondition* de la red Bayesiana para construcción de nuevas bases

Lo más destacado de la *ilustración 48*, es ver la medida en la que afecta la condición primaria, ya que como se ha comentado, la secundaria variará ligeramente el resultado final, pero este depende principalmente de si se puede fundar un nuevo asentamiento.

Precondition	preconditionOk				preconditionKo			
Primary condition	primaryOk		primaryKo		primaryOk		primaryKo	
Secondary condition	secondaryOk	secondaryKo	secondaryOk	secondaryKo	secondaryOk	secondaryKo	secondaryOk	secondaryKo
success	1	0.7	0.2	0	0	0	0	0
failure	0	0.3	0.8	1	1	1	1	1

Ilustración 48: DPC del nodo *New base* de la red Bayesiana para construcción de nuevas bases

En la *tabla 17*, se pueden observar los resultados obtenidos de una pequeña batería de pruebas. Las pruebas se dividen en grupos, dónde en cada uno se persigue un objetivo diferente. Mientras que en el primero se juega con el valor de la precondición, en el segundo se varían todos los valores implicados en la condición primaria, mientras el resto se deja constante. El objeto del estudio del tercer grupo, es la respuesta de la red a diferentes valores en la condición secundaria.

few	force	small	yes - bases	yes - progress	finished	yes - possible	yes - worker	success
0.5	0.5	0.5	0	0	0.5	0	1	**0**
0.2	0.8	0.3	0	0	0.5	1	1	**0.739**
0.9	0.1	0.8	0	0	0.5	1	1	**0.378**
0.6	0.4	0.3	0	0	0.5	1	1	**0.569**
0.3	0.9	0.3	0	0	0.5	1	1	**0.752**
0.6	0.9	0.3	0	0	0.5	1	1	**0.707**
0.3	0.9	0.7	0	0	0.5	1	1	**0.664**
0.3	0.4	0.3	0	0	0.5	1	1	**0.612**
0.3	0.9	0.7	0	0	0.5	1	1	**0.664**
0.2	0.8	0.3	0	0	0.3	1	1	**0.779**
0.2	0.8	0.3	1	0	0.3	1	1	**0.629**
0.2	0.8	0.3	1	1	0.3	1	1	**0.601**
0.2	0.8	0.3	1	1	0.8	1	1	**0.58**
0.2	0.8	0.3	0	1	0.3	1	1	**0.644**
0.2	0.8	0.3	0	1	0.8	1	1	**0.602**
0.2	0.8	0.3	0	0	0.8	1	1	**0.681**
0.2	0.8	0.3	1	0	0.8	1	1	**0.608**

Tabla 17: Resultados de la red bayesiana para construcción de nuevas bases

Los resultados más interesantes son los obtenidos en el segundo grupo, ya que es dónde se ven las condiciones en las que se debe fundar la nueva base. Aunque hay que tener en cuenta, que si la partida se encontrase en una etapa más temprana, los resultados serían mas elevados, ya que el tiempo corre en contra.

4.5. Capa Estructuras

La capa actual tiene un doble objetivo, ofrecer estructuras de apoyo a la implementación de las estrategias y servir de puente entre las decisiones tomadas por el general y las acciones tácticas. La *ilustración 49* refleja el diseño interno de la capa.

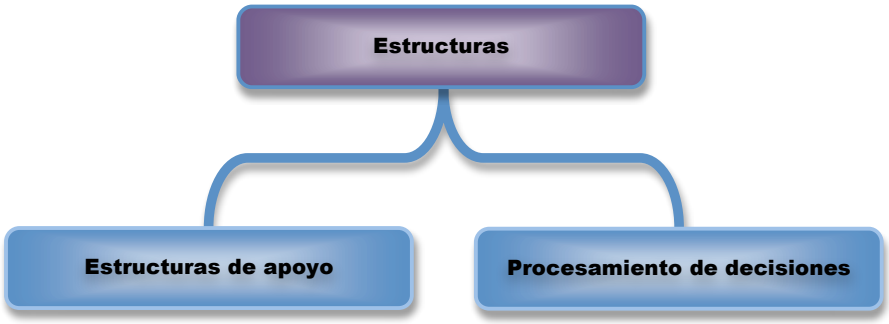

Ilustración 49: Diseño de la capa Estructuras

La subcapa *Estructuras de apoyo*, aporta a cada acción táctica estructuras reales que organizan su información de forma eficiente y lógica, facilitando su ejecución. Por otro lado, la subcapa *Procesamiento de decisiones*, aplica ciertas reglas a las decisiones tomadas por el general, con el objetivo de aumentar el grado de coherencia de las decisiones ejecutadas (ejemplo apartado 4. Desarrollo). Además, selecciona las unidades que deben realizar cada acción y el lugar donde se deben llevar a cabo.

En las acciones *gather* y *scouting*, se omite el procesamiento de decisiones, debido a que el general no toma ninguna. En estos casos, la capa encargada de dicho procesamiento, sólo debe seleccionar las unidades y el lugar donde se debe realizar la tarea.

Para explicar el funcionamiento de la capa actual, se van a formar cuatro grupos de acciones tácticas. El primer grupo está formado por ataque, defensa y creación de nuevos asentamientos, debido a que las estructuras suministradas para el ataque y la defensa son idénticas. Además, dichas estructuras están contenidas en la organización interna de una base, ya que esta debe tener una defensa.

Las construcción de edificios es el único integrante del segundo grupo, porque la diferencia entre la construcción de un edificio *barracks* y uno *factory*, reside en pequeños matices. Aplicando el razonamiento anterior al entrenamiento de unidades, se obtiene el motivo por el cual el tercer grupo se centra en el entrenamiento de unidades militares y no militares. El cuarto y último grupo, esta formado por las acciones que omiten el procesamiento de decisiones.

4.5.1. Ataque, defensa y creación de nuevas bases

La implementación de la estrategia de ataque y defensa es la misma, sólo se diferencian en la forma de ejecutarla. El objetivo de la táctica es eliminar toda unidad enemiga en la región definida mediante un par de coordenadas (x,y) y un radio de acción. Si no existe ningún enemigo en la región definida, todas las unidades deben permanecer dentro de élla.

Por ello, la estructura ofrecida es sencilla. Básicamente, consta de un conjunto para almacenar las unidades militares y su estado, el par de coordenadas (x,y) y el radio de acción. Es muy importante conocer el estado de cada unidad (parado, atacando, moviéndose hacia un enemigo ó moviéndose hacia la región de acción), para poderle asignar la acción que debe realizar en cada momento.

Para defender una base, la región de acción se definirá con las coordenadas del centro de la misma y la distancia al edificio más alejado. Por el contrario, para atacar, se utilizará el centro del edificio enemigo o la posición de una unidad enemiga y una distancia pequeña, con el objetivo de no ser vistos por el resto de enemigos.

La acción de fundar un nuevo asentamiento se divide en dos fases. En la primera, se debe asegurar la ubicación del nuevo asentamiento, mediante un conjunto de unidades militares. Con la región asegurada, se debe movilizar la unidad *worker* encargada de la construcción del edificio *control center*. La fundación de la base es el objetivo de la última fase. En ella, se lleva a cabo la construcción del edificio y la creación de la defensa, utilizando las unidades militares de la fase anterior.

La estructura suministrada para la primera fase, consta de una defensa, una unidad *worker*, un fondo donde reservar la cantidad de minerales necesarios para la construcción del edificio principal, y las ubicaciones del *cluster* de minerales más cercano y de la base enemiga más cercana. Ambas ubicaciones son imprescindibles para calcular la posición más segura donde fundar la nueva base.

Para gestionar una base, se precisa de una estructura más compleja donde se pueda almacenar una mayor cantidad de información de forma eficiente. Por ello, la estructura definida para dicha gestión consta principalmente del centro y radio de la misma, del edificio principal, de una defensa, de un conjunto de edificios y de estructuras de apoyo para la construcción de edificios y entrenamiento de unidades.

Tomadas las decisiones por parte del general, hay que realizar un pequeño procesamiento de las mismas. Concretamente, hay que comprobar si la situación actual corresponde con:

- El general decide defender y fundar un nuevo asentamiento y/o atacar.

- Ninguna base está siendo atacada.

En esta situación, hay que ignorar temporalmente la orden de defender y centrarse en la creación de una nueva base o en atacar. Si en la situación actual, el general no decide ni defender, ni atacar, ni fundar un nuevo asentamiento, todas las unidades militares sin acción asignada, pasarán a formar parte de la defensa de la base más cercana.

Procesadas las decisiones del general, se debe seleccionar la ubicación y las unidades que deben llevar a cabo las acciones tácticas. Dicha selección, varía en función de la acción que se debe ejecutar.

Si se decide defender, cada unidad militar será asignada a la defensa de la base más cercana, excepto las unidades que estén ejecutando un ataque anterior. Dichas unidades sólo cancelarán el ataque y se centrarán en defender si no están atacando en el instante actual (no hay unidades enemigas en su radio de acción), si el enemigo está atacando alguna base y ésta cuenta con un poder militar escaso. De cancelarse, las unidades militares pasarían a defender la base que esta recibiendo el ataque, excepto si se está recibiendo uno en cada base, en cuyo caso, los militares se repartirían entre ambas defensas.

Si la decisión tomada consiste en crear un nuevo asentamiento y se dispone de los recursos necesarios, se utilizará un tercio del poder militar actual y una unidad *worker* seleccionada mediante una jerarquía establecida, la cual consiste en solicitar una unidad sin trabajo asignado. Si en el instante actual, todas las unidades están ocupadas, se escogerá una unidad asignada a la acción *gather*. Si dicha acción, no contase con ninguna unidad, se recurriría a la acción *scouting*. No se debe seleccionar la ubicación de la nueva base, pues el general indica el *cluster* de minerales donde quiere fundarla. Sólo se debe calcular la posición del edificio *control center*, respetando dos restricciones, cercanía al *cluster* de minerales y lejanía a la base enemiga más cercana (intentado ubicar el *cluster* entre ambas bases).

Para atacar, se escoge la base enemiga más cercana a las unidades militares que van a realizar la acción. Dichas unidades, se seleccionan en función del ataque que se va a ejecutar. Se definen tres. El primero se denomina *low* y se ejecuta cuando se omite la decisión de defender.

El ataque *medium* responde a un ataque campal sobre una base secundaria, mientras que el *high*, se corresponde con un ataque campal sobre un asentamiento principal. La diferencia entre los dos primeros tipos, es la cantidad de unidades utilizada: *low* es más conservador.

Cuando se dispone de dos bases y el tipo a ejecutar es *low*, se lanza el ataque con las unidades de la base que dispone menos poder militar y las unidades de la otra se reparten entre ambas para formar las defensas. Por el contrario, si se debe ejecutar el tipo *medium*, la ejecución es igual a la anterior, con la diferencia de utilizar las unidades de la base con mayor poder militar para atacar y las unidades de la otra para defender. Por último, en el ataque *high*, se utilizan todas las unidades militares disponibles.

Si en la situación actual se debe ejecutar un ataque y sólo se dispone de un asentamiento, se utilizan uno, dos o tres tercios del poder militar actual, en función del tipo de ataque a ejecutar: *low*, *medium* y *high* respectivamente.

4.5.2. Construcción de edificios

La construcción de nuevos edificios es una acción que está gestionada por la base donde se van a construir, es decir, la estructura diseñada para esta acción es interna a la estructura de la gestión de la base.

Para gestionar la creación de nuevos edificios se necesita principalmente una cola de peticiones de cada tipo, una unidad capaz de construir un edificio e información relativa a la construcción que se quiera llevar a cabo (tipo de edificio, reserva de minerales para el coste de la edificación, constructor, ubicación, etc.). Con la construcción finalizada, el nuevo edificio pasa a formar parte de la base y la unidad *worker* es liberada para poder realizar otras tareas.

En este caso, el procesamiento de las decisiones tomadas por el general, equivale a comprobar si la situación actual es la siguiente:

- El general decide fundar una nueva base, pero no dispone de recursos suficientes.
- El general decide construir un edificio *barracks* y/o *factory*.
- No es urgente la construcción del edificio *barracks ni factory*.

Debido a que la construcción de los edificios de entrenamiento no es urgente y es necesaria la construcción de una nueva base, se omite la decisión de construcción temporalmente, con el objetivo de reunir los recursos necesarios para fundar el nuevo asentamiento. Si durante la recolección de los recursos, la construcción de uno de los dos edificios se convirtiese en urgente, se atendería la decisión anterior.

La selección de la unidad que va a desempeñar la construcción, es el siguiente paso al procesamiento de las decisiones tomadas por el general. Dicha selección, se realiza aplicando una jerarquía definida para tal fin, la cual consiste en solicitar una unidad que esté desempeñando la acción *gather*. Si en ese instante, no hay unidades asignadas a dicha tarea, se intentará obtener una unidad sin trabajo asignado. Si todas las unidades están realizando la actividad *scouting*, se seleccionará un *scout*.

El edificio se construirá en la posición actual de la unidad *worker*, a menos que no se cuente con el espacio libre necesario. En ese caso, se buscará una ubicación cercana en la que se pueda construir. Salvo excepciones extremas, la unidad encargada de la construcción se escogerá de la actividad *gather*, por lo que el nuevo edificio estará muy próximo al *cluster* de minerales y/o al *control center*.

4.5.3. Entrenamiento de unidades

Al igual que la construcción de edificios, la gestión del entrenamiento de unidades es interna a la gestión de la base, ya que se van a utilizar los edificios disponibles en el asentamiento. Para gestionar la información referente a esta acción, se dispone de una estructura que consta principalmente de una cola de peticiones de cada tipo e información relativa a la unidad que se desea entrenar: tipo de unidad, reserva de minerales necesaria para el entrenamiento, edificio que va a realizar el entrenamiento, estado del entrenamiento, etc.

Procesar las decisiones del general a la hora de entrenar nuevas unidades, consiste en determinar si la situación actual se corresponde con las siguientes afirmaciones:

- El general decide construir un edificio de entrenamiento de unidades de forma urgente y/o fundar un nuevo asentamiento.

- No se dispone de recursos necesarios para llevar a cabo ninguna de las decisiones anteriores.

- El general decide entrenar una unidad militar.

- El poder militar actual es significante: seis marines para el edificio de entrenamiento de unidades y veinte para la creación de la nueva base.

Si las afirmaciones son ciertas para la situación actual, no se debe entrenar ninguna unidad militar hasta reunir los recursos necesarios para la construcción de un edificio que permita el entrenamiento de unidades y/o para fundar un nuevo asentamiento, excepto si el poder militar deja de ser significante.

La acción actual la desempeña un edificio, no una unidad. Por este motivo, sólo hay que seleccionar una construcción capaz de ejecutar la tarea con éxito. Dicho edificio debe cumplir las siguientes condiciones:

- El general indica la base que debe realizar el entrenamiento, por lo cual, debe pertenecer a dicha base.

- Estar disponible, esto es, sin entrenamiento en curso.

- Estar capacitado para entrenar el tipo de unidad demandada.

Si todos los edificios capacitados estuviesen ocupados, el entrenamiento quedaría suspendido hasta encontrar uno disponible.

4.5.4. Gather y scouting

La acción *scouting* consiste en desplazar una unidad *worker*, a una coordenada que se encuentre en una región[17] no explorada. El *scout* será seleccionado del conjunto de unidades sin acción asignada. Si dicho grupo no tuviese ninguna unidad disponible, se utilizaría una asignada a la acción *gather*.

La acción *gather* es independiente y no está relacionada con la gestión de la base. La capa actual ofrece una estructura (denominada *SpecialGather*) que define un grupo de unidades *worker* recolectando en el mismo *cluster* de minerales y depositando los recursos en el mismo

[17] Dicha región debe estar formada por un mínimo de tres *tiles*, es decir, debe constituir más del 7,3 % del mapa.

control center. De la propia definición se deduce que el identificador del grupo será la combinación de *cluster* y edificio principal.

Un grupo será creado cuando se desee que al menos una unidad recolecte de un determinado *cluster* y deposite los minerales en un determinado *control center*, formando entonces un identificador inexistente. Por el contrario, un grupo será eliminado cuando no tenga asignado ninguna unidad.

Del juego se obtiene información individual de cada mineral, por lo que es necesaria la agrupación de los minerales en *clusters*. Para ello, se utiliza una estructura particular que define la capa actual.

Siempre que exista una unidad *worker* sin acción asignada y una vacante[18] en la acción *gather*, dicha unidad ocupará la vacante mencionada. Toda base tiene asignada un conjunto de *clusters* de minerales. Cuando una base dispone de más de uno, las unidades encargadas de recolectar minerales, serán distribuidas uniformemente entre todos los *clusters*.

Al igual que la acción *gather*, la acción táctica *scouting* es independiente y ajena a cualquier gestión de la base. En este caso, la estructura aportada (denominada *Scouting*) por la capa actual es muy simple, debido a que sólo recoge información de cada scout (unidad y destino). Esta acción dispone de un procedimiento que detecta cuando un scout se ha quedado parado. Cuando esta situación es detectada, se le asigna un nuevo destino.

4.6. Consideraciones finales

Para finalizar el capítulo, se va a realizar un breve estudio del grado del cumplimiento de los objetivos definidos en el capítulo anterior, 3. Objetivos. Por ello, se van a volver a enumerar, comentando sobre cada uno de ellos su grado de cumplimiento y verificabilidad. Dicho estudio servirá para comprobar que se ha llegado a los objetivos marcados y también para repasar lo comentado hasta el momento.

1. **El cliente debe ser funcional.** El primer objetivo es fácilmente verificable, pues se cuenta con los resultados obtenidos en el torneo ORTS 2009. Además, como este año sólo ha habido dos participantes, se ha optado por llevar a cabo un pequeño torneo con

[18] En cada cluster de minerales sólo puede haber asignados un máximo de diez unidades *worker*. Dicha restricción se definió en el apartado 4.3.5 Redes bayesianas de entrenamiento de *workers*.

los jugadores de años anteriores, con el objetivo de hacer un estudio de efectividad del cliente. En el capítulo 5. Resultados, se pueden ver los resultados del torneo ORTS 2009 y los estudios de rendimiento y efectividad.

2. **El cliente debe utilizar técnicas de IA.** Para la toma de decisiones se han utilizado redes bayesianas, por lo que el objetivo se ha cumplido.

   Las herramientas utilizadas para la implementación del modelo han sido *GeNIe & SMILE* [1], las cuales permiten y facilitan la definición de las redes bayesianas y su invocación desde el cliente.

3. **El cliente debe participar en el torneo de ORTS del año 2009.** El torneo del año 2009 fue retrasado, pero finalmente se celebró y BlackRose fue uno de los dos participantes. Hay que destacar que la versión final del cliente fue enviada a Michael Buro [9] en las fechas establecidas para la inscripción de los participantes.

4. **El cliente debe estar bien estructurado.** Durante todo el desarrollo, se ha procurado que la estructura del cliente sea clara y modular, con el objetivo de facilitar su utilización por terceras personas.

   Mediante el análisis del diagrama de clases mostrado en el Anexo A, se puede afirmar que sustituir un determinado módulo o introducir uno nuevo es una tarea sencilla. Además, con el objetivo de facilitar la utilización de cada módulo, se han definido dos interfaces: una privada para el funcionamiento interno y una pública con los métodos de acceso a las funcionalidades que ofrece cada uno.

5. **Generar la documentación de todo el proceso de diseño e implementación.** Se ha documentado todo el proceso de diseño e implementación, lo cual no sirve para enseñar cómo implementar un cliente en ORTS, si no que más bien describe la selección de estrategias a implementar, el diseño de las redes bayesianas para la toma de decisiones y las estructuras intermedias que se han utilizado.

Pero dicha descripción unida al Anexo C, forman un buen punto de partida para personas inexpertas, ya que este anexo si explica como implementar un cliente sencillo. Por lo cual, se puede verificar que el objetivo se ha cumplido, ya que además de generar la documentación, se aporta un pequeño tutorial para iniciarse en el mundo ORTS.

6. **Generar la documentación del API utilizado para la implementación del cliente.** El Anexo B contiene la descripción de los módulos utilizados e implementados en el cliente. Dicha descripción está compuesta por el objetivo de cada módulo, una breve explicación de los métodos más relevantes y cómo se debe utilizar el módulo. Toda esta información unida a los comentarios del código fuente, forman una buena referencia para usuarios en proceso de aprendizaje y para usuarios interesados en los módulos implementados.

Capítulo 5

Resultados

5. Resultados

Explicado el proceso de desarrollo, se procede a realizar un estudio de rendimiento del sistema de toma de decisiones implementado. Para ello, se va a lanzar una simulación de una partida bajo las condiciones del tercer escenario del torneo ORTS. En este encuentro, ambos jugadores van a ser el cliente implementado.

Se van a recoger muestras del tiempo invertido en la asignación de los nuevos valores de entrada y la ejecución de cada una de las redes para cada ciclo de la simulación. Hay que tener en cuenta, que hay redes que se ejecutan más de una vez por ciclo, por lo que se recogerá una muestra por cada ejecución. Además, se recogerán muestras del tiempo invertido en el método *compute_actions()* de cada iteración, es decir, en el método responsable de la toma de decisiones de los objetos del jugador.

La simulación se va a lanzar sin interfaz gráfica en un único ordenador con un procesador Intel Core 2 Duo a 2GHz y 4 GB de memoria DDR2 SDRAM a 667 MHz. En estas condiciones, la tarjeta gráfica carece de importancia.

Todos los datos obtenidos han sido recogidos y procesados, generando una gráfica para cada conjunto de muestras. Debido a que la simulación ha sido lanzada con dos clientes, se van a mostrar dos gráficas por cada conjunto de valores. Es preciso conocer que todos los datos obtenidos están expresados en microsegundos, por lo que dicha unidad es la que se va a utilizar en todas las representaciones.

Como complemento al análisis de rendimiento, se esperaba poder hacer un estudio de efectividad con los datos obtenidos en el torneo ORTS 2009. Pero debido a la escasa participación, se ha optado por realizar un análisis con los datos obtenidos de un pequeño torneo formado por jugadores de años anteriores. Para ello, se van a lanzar veinticinco simulaciones de cada enfrentamiento bajo las mismas condiciones del torneo. Hay que destacar que sólo se van a ejecutar las simulaciones en las que participa el cliente implementado, omitiendo el resto de enfrentamientos. No obstante, en la sección 5.9 Torneo ORTS 2009, se comentan los resultados del mismo.

5.1. Red de ataque

Como puede observarse en las *ilustraciones 50 y 51*, en ambas evoluciones se aprecia que el tiempo invertido no es constante y además existen valores atípicos, lo cual es algo normal, pues se está trabajando con tiempos muy pequeños y cualquier retardo durante su ejecución será reflejado. El motivo de estos retardos no es conocido, pero lo más probable es que se deba a la gestión interna de ejecución (recordar que se están ejecutando tres procesos en paralelo) de los procesos (política de ejecución, de recursos, etc.).

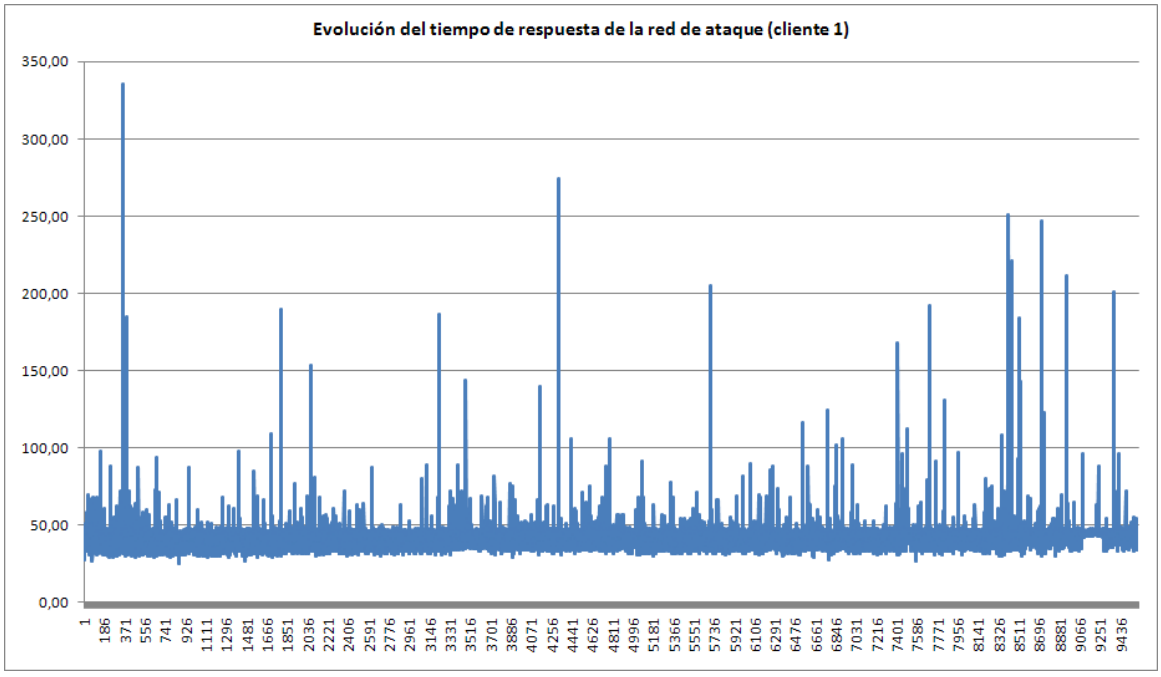

Ilustración 50: Evolución del tiempo de respuesta de la red de ataque (cliente 1)

Por otro lado, hay que indicar que es normal que el tiempo empleado no sea constante, pues hay muchos factores que influyen como el algoritmo de propagación de probabilidades, asignación de los valores de entrada, etc.

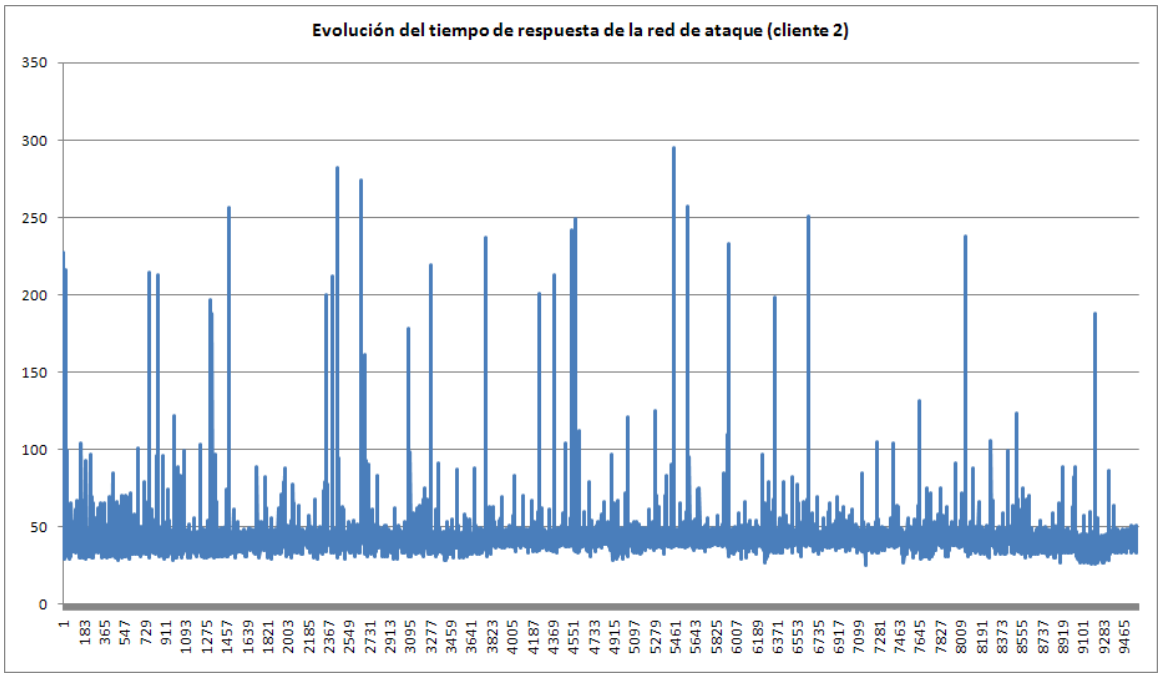

Ilustración 51: Evolución del tiempo de respuesta de la red de ataque (cliente 2)

Como se puede ver, ambas gráficas son muy similares, pues la mayoría de los datos están por debajo de la línea de cincuenta microsegundos.

5.2. Red de defensa

Todas las gráficas van a reflejar lo comentado en las dos anteriores (dispersión y valores atípicos), pues los datos se han obtenido en el mismo escenario. Por otro lado, el tiempo medio invertido en la ejecución de la red de defensa parece ser inferior al de la red de ataque, como se deduce al analizar detenidamente las *ilustraciones 52 y 53*.

La gráfica referente a la red de defensa del cliente 1, muestra un pequeño badén, ya que entre las muestras 3900 y 13000, el tiempo medio utilizado es levemente inferior que en el resto de la simulación.

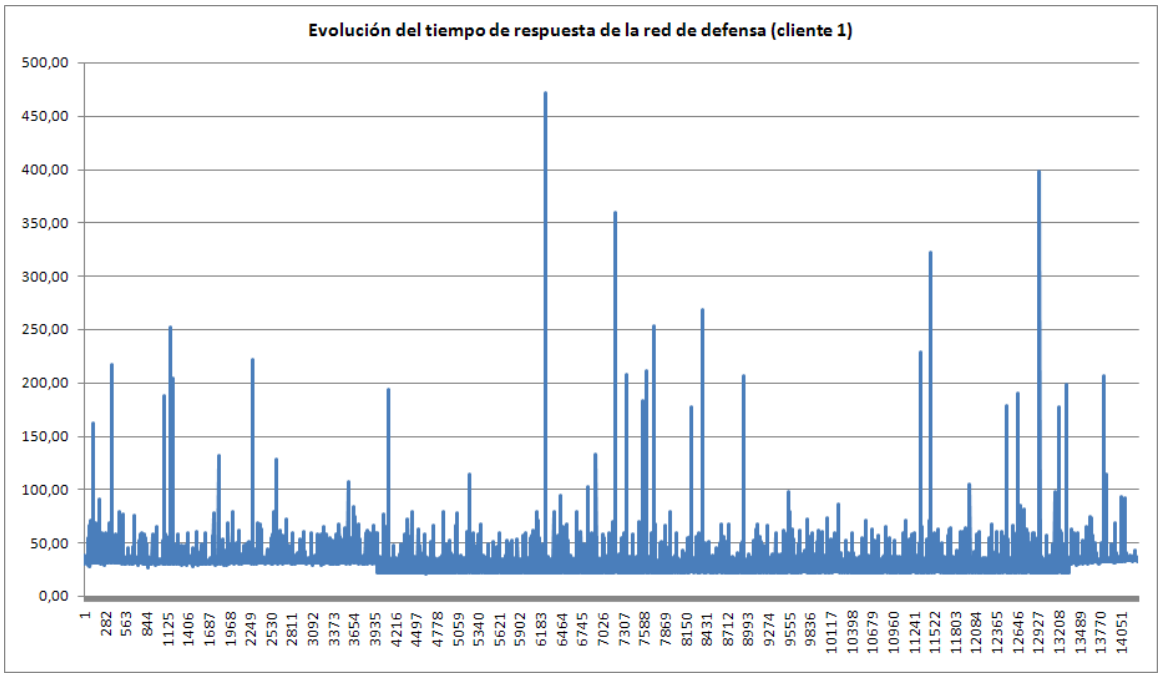

Ilustración 52: Evolución del tiempo de respuesta de la red de defensa (cliente 1)

Analizando ambas gráficas, se puede afirmar que el cliente 1 funda una nueva base durante la simulación, acción que no realiza el cliente 2. Esta afirmación se basa en el número de muestras que contiene cada conjunto, ya que para el cliente 1 se dispone de más de catorce mil, mientras que para el cliente 2, se tienen tantas como ciclos de simulación se han ejecutado: nueve mil seiscientas.

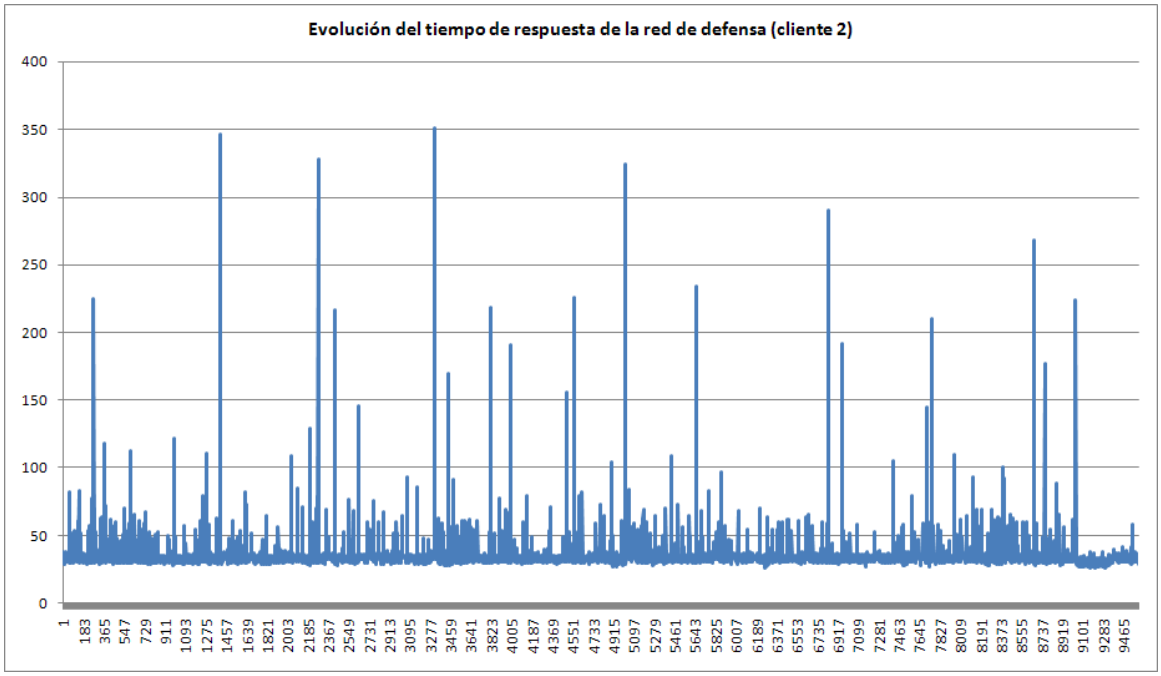

Ilustración 53: Evolución del tiempo de respuesta de la red de defensa (cliente 2)

5.3. Red para fundar nuevos asentamientos

En esta sección, se van a analizar los resultados obtenidos de la red que determina la probabilidad con la que se debería fundar un nuevo asentamiento, sobre un determinado *cluster* de minerales. Las gráficas elaboradas con dichos datos, se pueden observar en las *ilustraciones 54 y 55*.

Lo que más destaca de las gráficas (aparte de la dispersión y los valores atípicos), es el valor medio, el cual gira alrededor de los ochenta/noventa microsegundos. Este valor es superior al de las redes anteriores porque la red actual es más compleja.

Además, hay que señalar que ambos conjuntos de muestras están formados por más de veintitrés mil datos, motivo por el cual, da sensación de haber un mayor número de valores atípicos y una mayor dispersión.

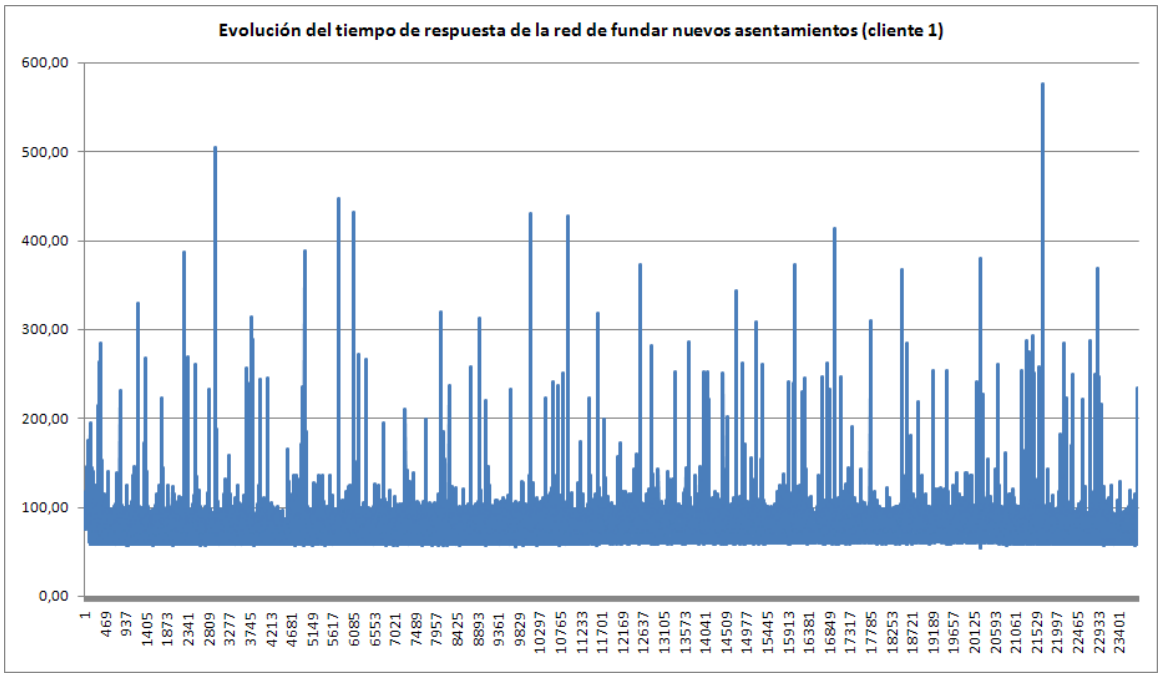

Ilustración 54: Evolución del tiempo de respuesta de la red de fundar nuevos asentamientos (cliente 1)

Para finalizar, indicar que el número de muestras de cada gráfica, da información sobre que cliente ha localizado los *clusters* de minerales más rápidamente y/o que cliente ha descubierto más *clusters*, ya sea por estar localizados en las proximidades de su base, por una buena tarea de las unidades encargadas de la actividad *scouting* o porque uno de los dos clientes no ha explorado zonas del escenario en las que se ubican *clusters* de minerales.

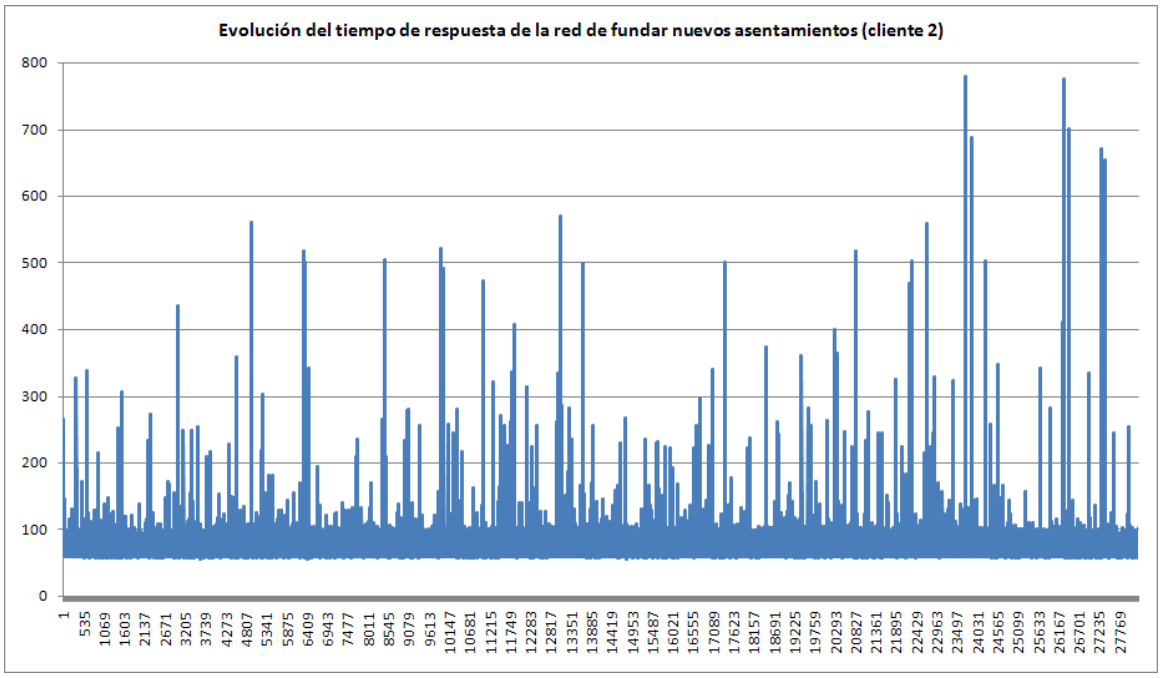

Ilustración 55: Evolución del tiempo de respuesta de la red de fundar nuevos asentamientos (cliente 2)

5.4. Red de construcción de *barracks*

Las *ilustraciones 56 y 57*, reflejan los datos obtenidos al medir los tiempos de la red de construcción del edificio *barracks*. En ellas, se puede ver a simple vista que muestran menos dispersión, pues el rango en el que se encuentran la mayoría de los datos es menos amplio que en los casos analizados hasta ahora (exceptuando las gráficas de la red de defensa, pues son muy similares a las que se están analizando, incluso la *ilustración 56* refleja un badén similar al de la *ilustración 52*).

Al igual que en el caso de la red de defensa, analizando el número de muestras de cada conjunto se puede deducir que cliente ha fundado un nuevo asentamiento.

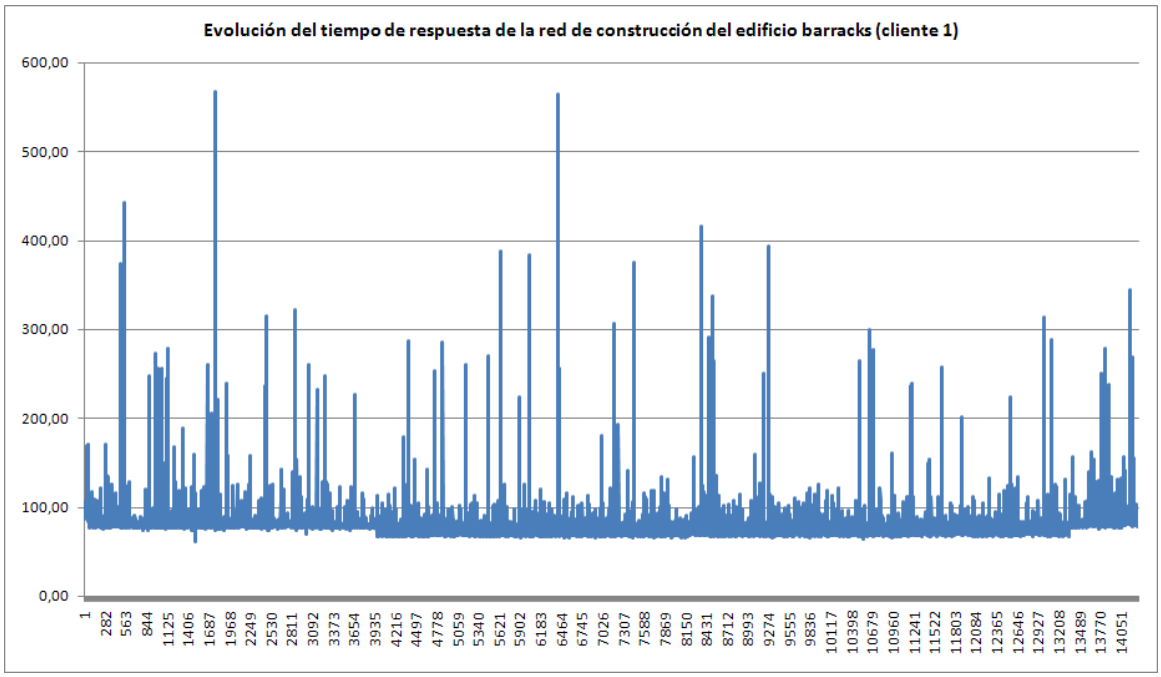

Ilustración 56: Evolución del tiempo de respuesta de la red de construcción del edificio *barracks* (cliente 1)

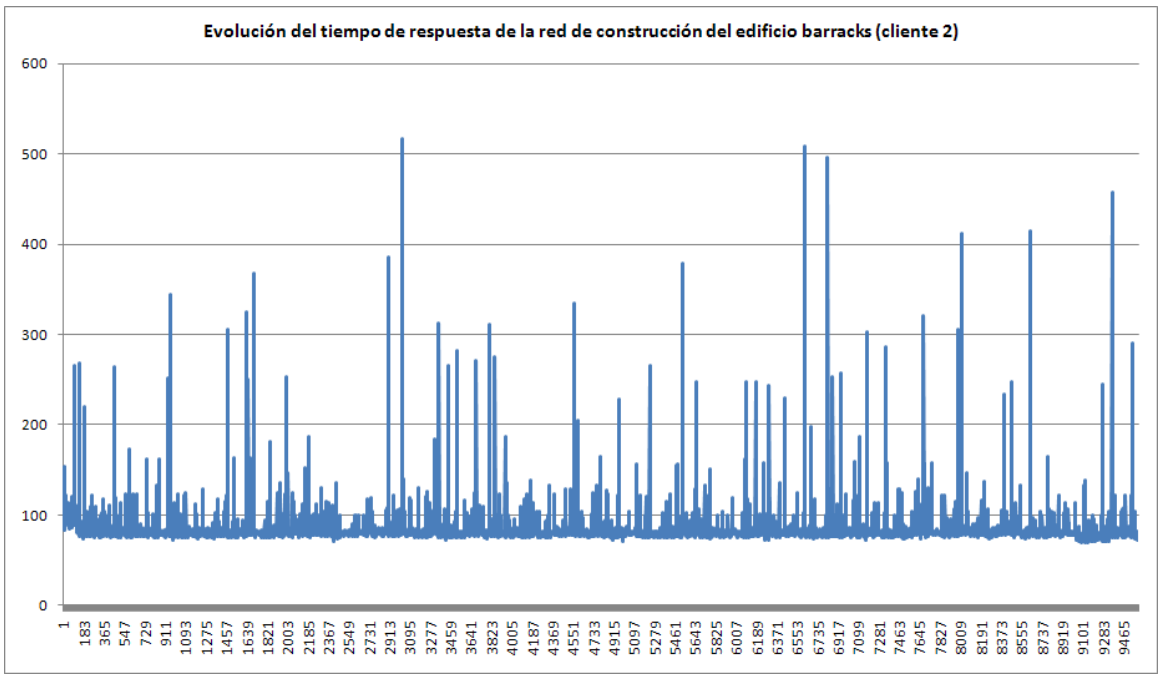

Ilustración 57: Evolución del tiempo de respuesta de la red de construcción del edificio *barracks* (cliente 2)

5.5. Red de construcción de *factory*

Para analizar la evolución de los tiempos de respuesta de la red de construcción del edificio *factory*, hay que estudiar las *ilustraciones 58 y 59*, pues recogen las muestras obtenidas durante la simulación.

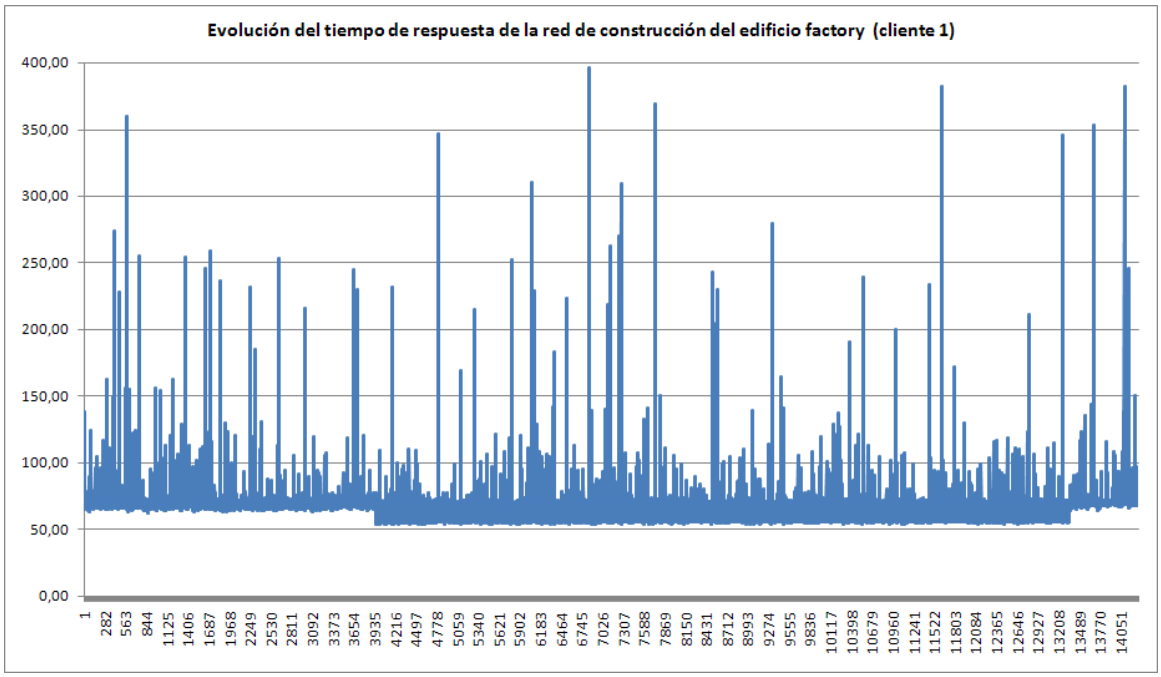

Ilustración 58: Evolución del tiempo de respuesta de la red de construcción del edificio *factory* (cliente 1)

La red bayesiana que indica la probabilidad con la que se debe construir el edificio *factory* en cada base, es más sencilla que la del edificio *barracks*. Por esa razón, los tiempos obtenidos deben ser ligeramente inferiores.

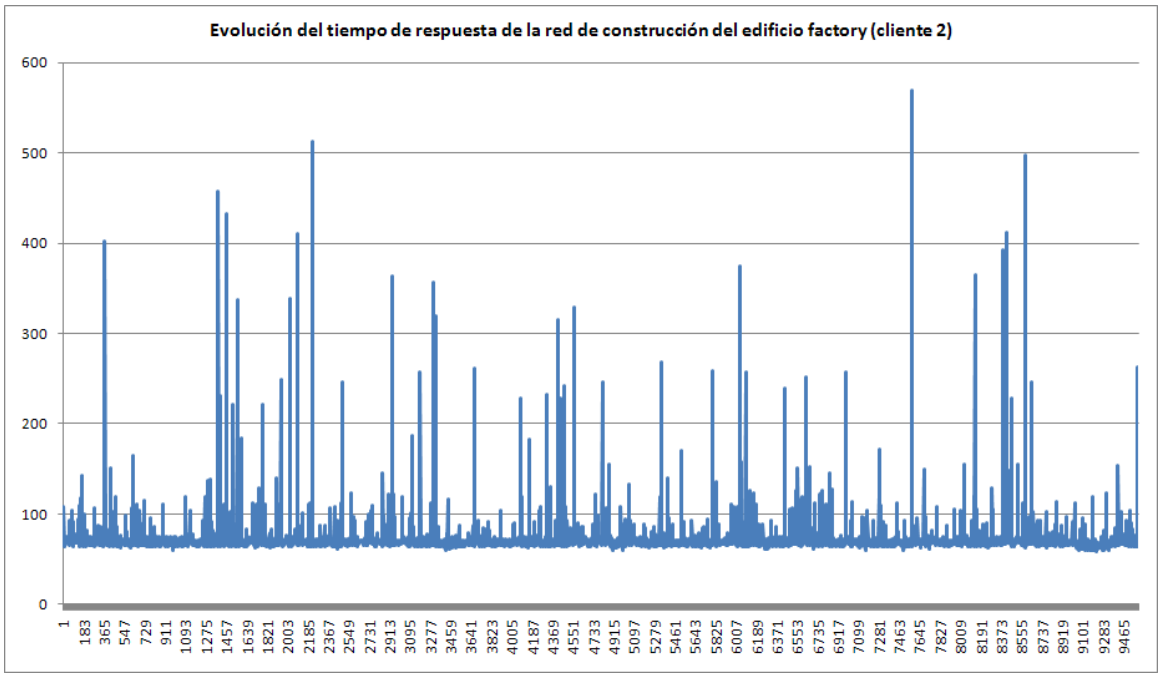

Ilustración 59: Evolución del tiempo de respuesta de la red de construcción del edificio *factory* (cliente 2)

5.6. Redes de entrenamiento de unidades

Para el entrenamiento de los tres tipos de unidades disponibles, se han utilizado dos redes bayesianas. Una para decidir el entrenamiento de las unidades *worker* y otra para determinar si se debe entrenar una unidad militar. Esta última, en función de los valores de entrada, representa un tipo de unidad militar u otro (es decir, representa un *marine* o un *tank*).

Primero se van a analizar los datos obtenidos de la red de entrenamiento de la unidad *worker*, los cuales se pueden observar en las *ilustraciones 60 y 61*. Esta red es la más sencilla de todas, por lo que debe reflejar un valor medio inferior al resto.

Comparando ambas gráficas, obtenemos como conclusión, que los resultados del cliente 1 parecen mejores que los del cliente 2, ya que presenta menos dispersión.

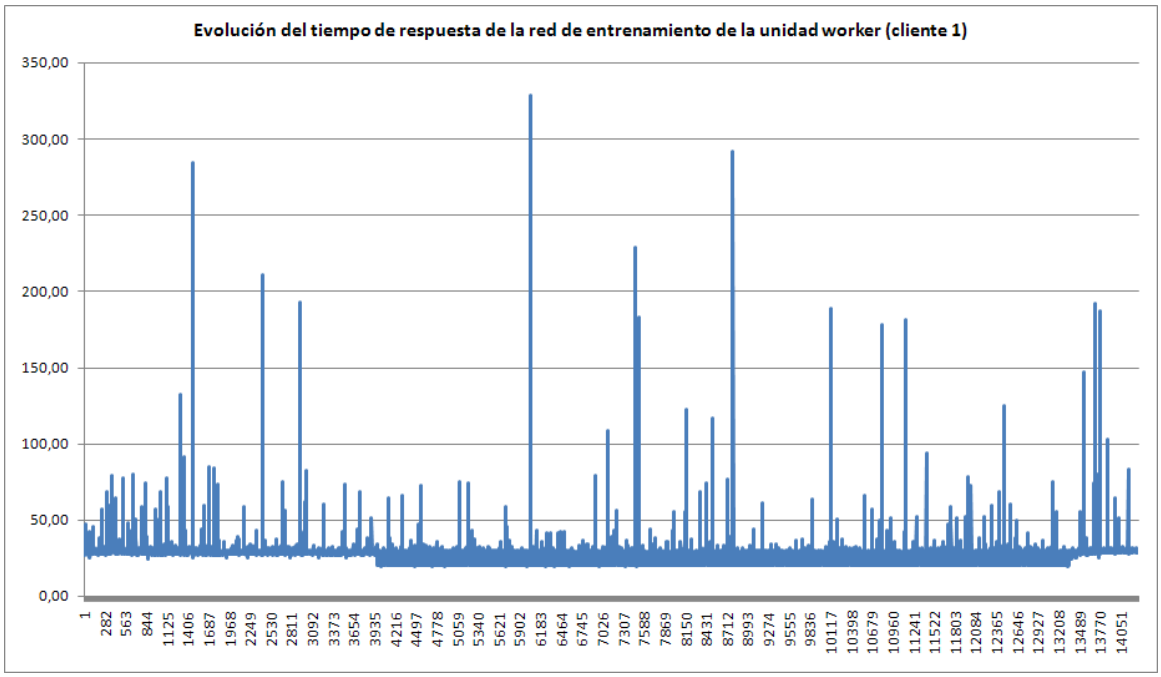

Ilustración 60: Evolución del tiempo de respuesta de la red de entrenamiento de la unidad *worker* (cliente 1)

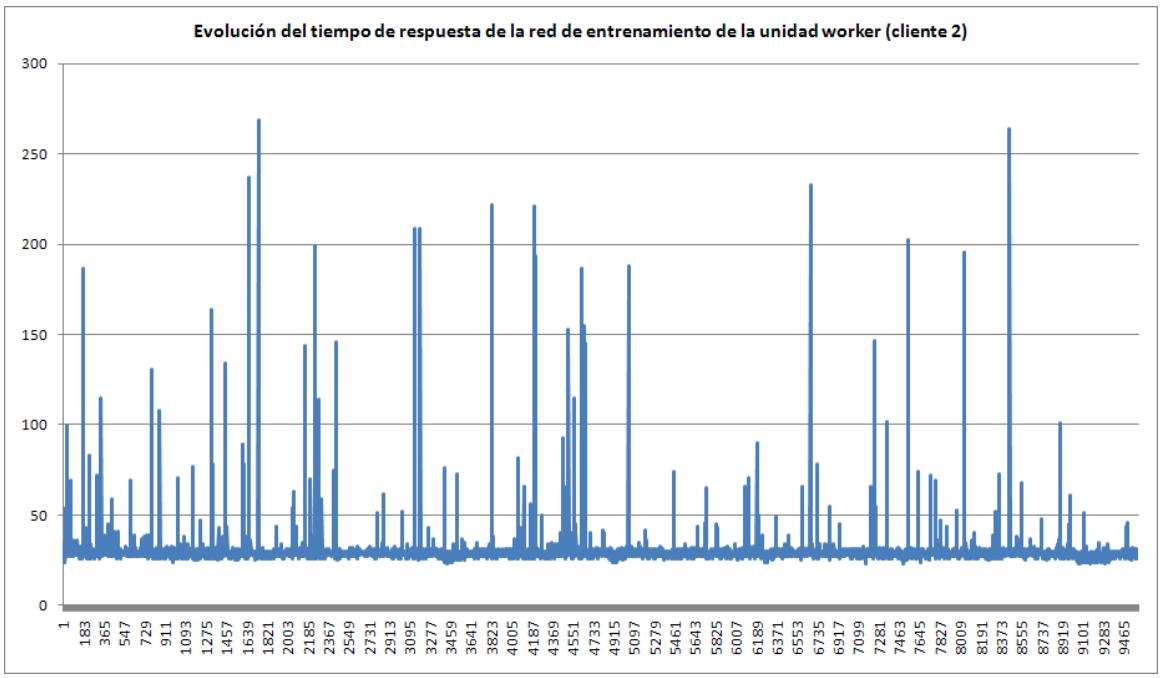

Ilustración 61: Evolución del tiempo de respuesta de la red de entrenamiento de la unidad *worker* (cliente 2)

En las *ilustraciones 62 y 63*, se muestran los resultados de la red de entrenamiento militar para el caso de la unidad *marine*, mientras que en las *ilustraciones 64 y 65*, se reflejan las muestras para la unidad *tank*.

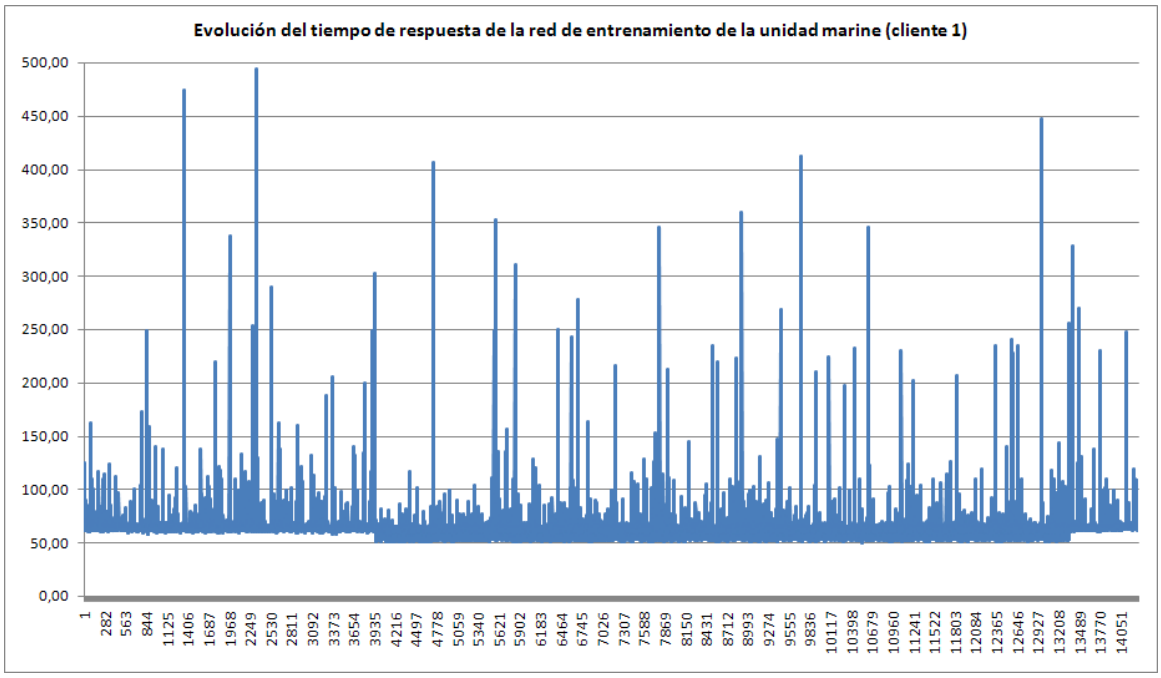

Ilustración 62: Evolución del tiempo de respuesta de la red de entrenamiento de la unidad *marine* (cliente 1)

Es muy curioso ver como se repite un badén similar en cinco gráficas del mismo cliente y más si se tiene en cuenta que suele aparecer en el mismo intervalo. Es posible que estén relacionados y exista un motivo lógico, pero determinar la causa exacta es una tarea excesivamente compleja, ya que los tiempos reflejados dependen de muchos factores, algunos de los cuales son ajenos al propio ORTS.

Es interesante observar que las gráficas de cada tipo de unidad militar presentan diferencias de tiempo entre sí, es decir, utilizando la misma red bayesiana para ambas unidades, se necesita más tiempo para asignar los valores de entrada y calcular la probabilidad del nodo de salida para la unidad *marine* que para la unidad *tank*. Siendo los datos de entrada, la única diferencia entre ambas ejecuciones.

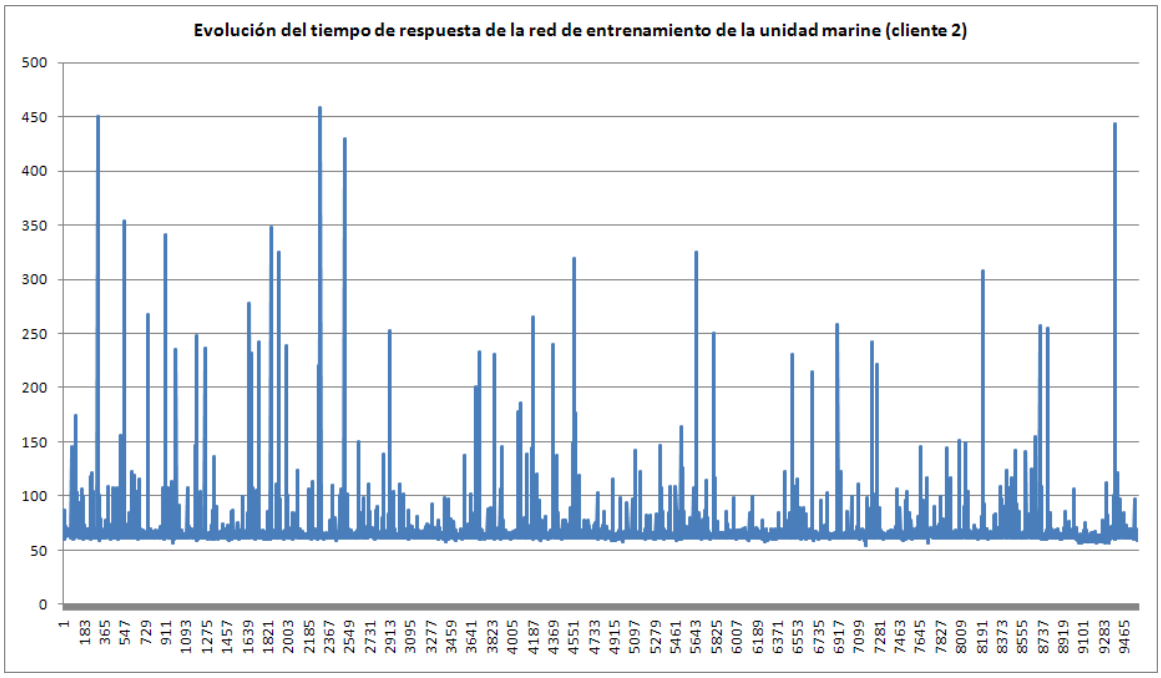

Ilustración 63: Evolución del tiempo de respuesta de la red de entrenamiento de la unidad *marine* (cliente 2)

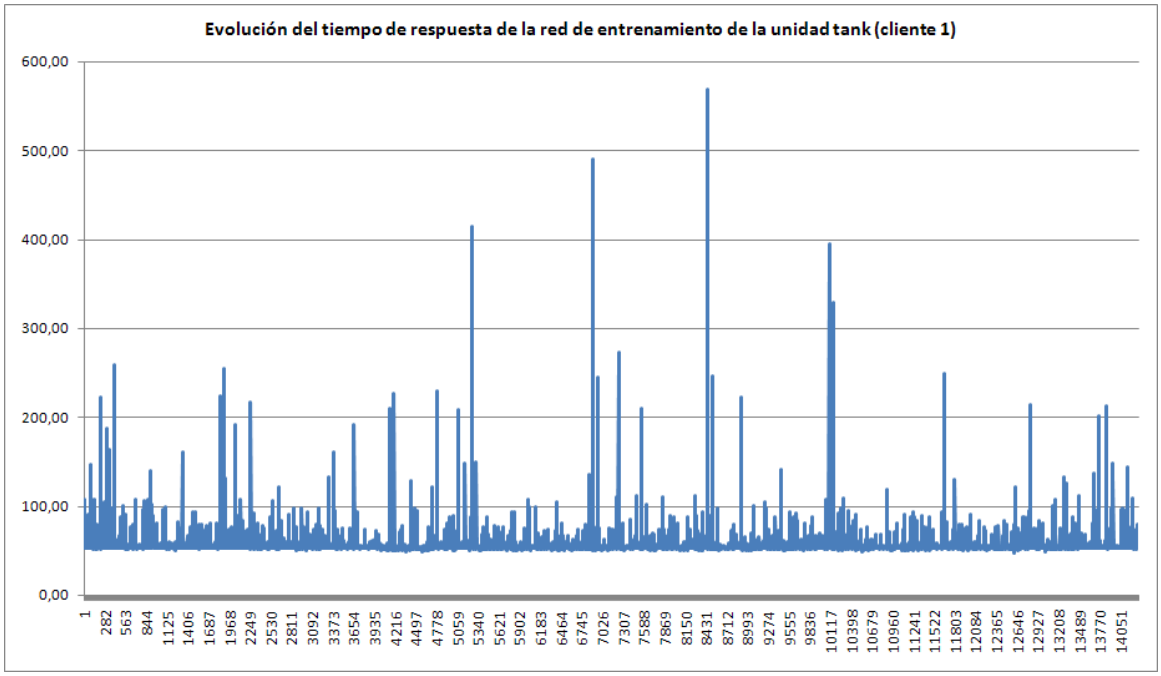

Ilustración 64: Evolución del tiempo de respuesta de la red de entrenamiento de la unidad *tank* (cliente 1)

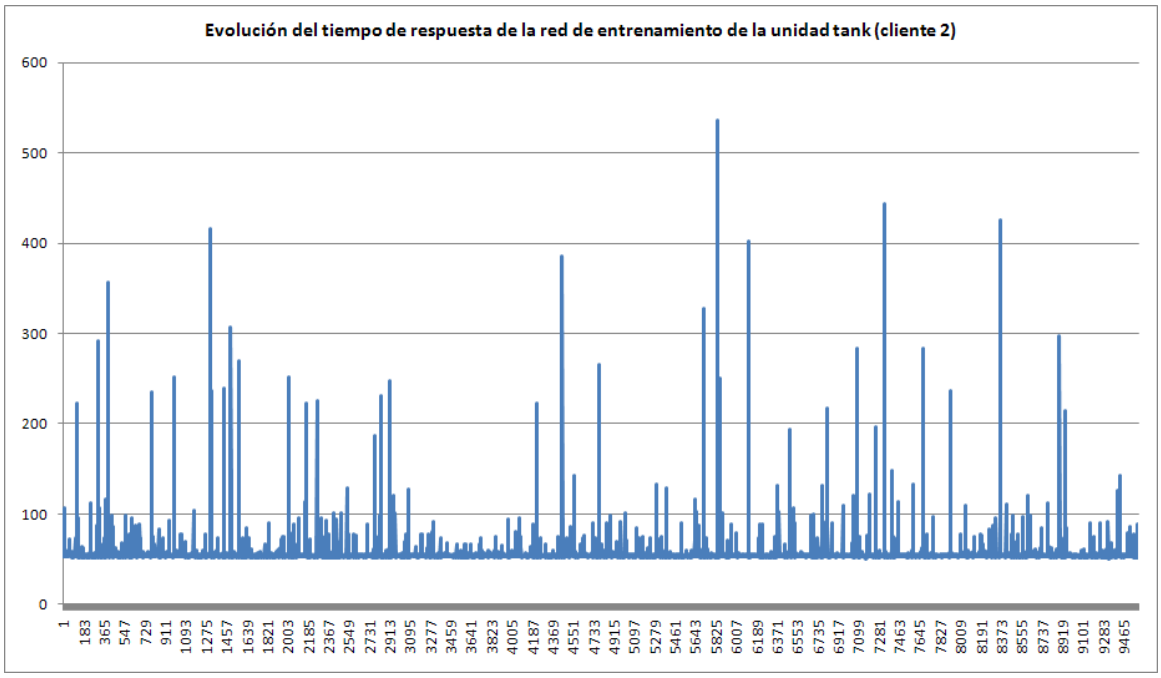

Ilustración 65: Evolución del tiempo de respuesta de la red de entrenamiento de la unidad *tank* (cliente 2)

5.7. Análisis de rendimiento

Para finalizar, se va a analizar el tiempo invertido en el método *compute_actions()*, en el cual reside todo el comportamiento del cliente. Analizando las *ilustraciones 66 y 67*, se puede observar que presentan una mayor dispersión y valores atípicos mas elevados (en proporción) que el resto de las gráficas estudiadas. Esto es un comportamiento esperado, pues por un lado refleja toda la dispersión y datos atípicos de las ejecuciones de las redes bayesianas y por otro, el cálculo y procesamiento de datos que se debe realizar al recibir determinados eventos o para poder ejecutar determinadas acciones. Por ejemplo, cuando se descubren nuevos minerales, hay que agrupar en *clusters* los nuevos minerales encontrados, recalcular todas las rutas de acceso y volver a determinar los *clusters* accesibles desde cada base.

Analizando ambas ilustraciones, se deduce que el cliente 1 realiza más acciones que necesitan un cálculo de datos elevado (fundar un nuevo asentamiento y/o realizar un ataque), ya que se observan intervalos donde el tiempo medio empleado es superior al resto de ciclos de simulación.

En toda partida de ORTS, se ejecutan ocho ciclos de simulación por segundo. Con estos datos, se obtiene que cada ciclo de simulación se debe ejecutar en un tiempo máximo de ciento veinticinco milisegundos. Y analizando esta información, se observa que la media de ejecución

de cada ciclo es inferior a cinco mil microsegundos (cinco milisegundos), por lo que el cliente no consume más tiempo del disponible en cada ciclo de simulación.

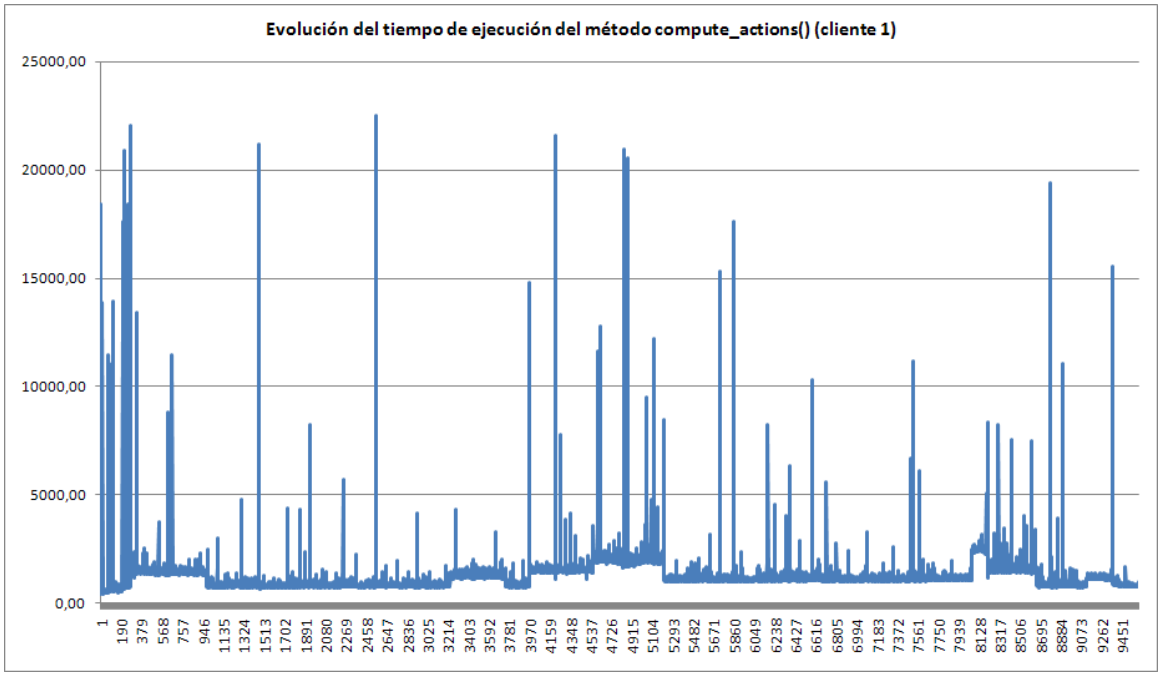

Ilustración 66: Evolución del tiempo de ejecución del método *compute_actions()* (cliente 1)

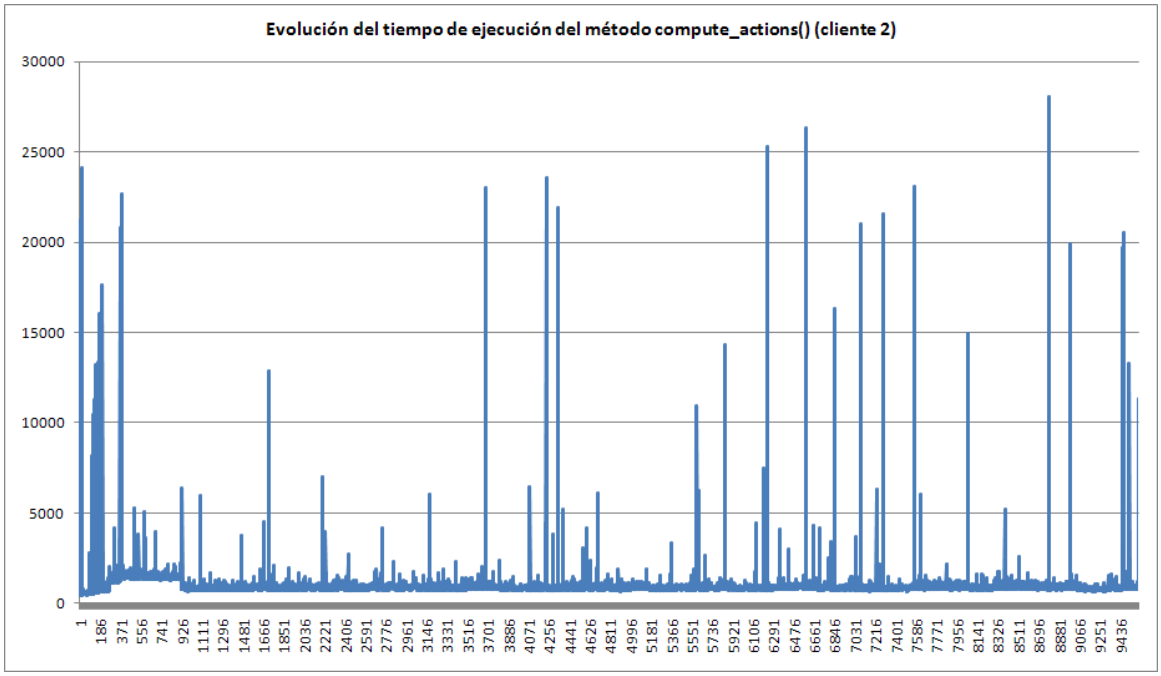

Ilustración 67: Evolución del tiempo de ejecución del método *compute_actions()* (cliente 2)

Para finalizar el estudio de rendimiento, se van a mostrar las estadísticas de cada conjunto de muestras. Dichos datos se pueden analizar en la *tabla 18*.

	Cliente1			Cliente 2		
	Media	Desv. Típica	N. datos	Media	Desv. Típica	N. datos
Ataque	41,52 µs	10,281	9600	43,508 µs	12,159	9600
Defensa	31,40 µs	11,707	14293	34,627 µs	11,069	9600
Nueva base	73,47 µs	20,437	23851	72,549 µs	24,261	28275
Barracks	78,65 µs	16,510	14293	82,145 µs	17,861	9600
Factory	66,76 µs	14,203	14293	71,707 µs	18,268	9600
Worker	26,58 µs	8,203	14293	29,693 µs	9,118	9600
Marine	62,22 µs	15,347	14293	65,681 µs	15,617	9600
Tank	54,72 µs	11,644	14293	55,403 µs	15,346	9600
Método	1268,5 µs	1041,832	9600	996,17 µs	1138,726	9600

Tabla 18: Datos estadísticos de los resultados obtenidos

El dato más relevante que se puede obtener del estudio de la tabla anterior es la pequeña desviación típica que presenta cada conjunto de datos, ya que la red bayesiana que más tiempo (de media) invierte, es la red de construcción del edificio *barracks*, la cual tiene una desviación típica inferior al 22% del valor de la media aritmética.

Realizado el estudio, se puede afirmar que el rendimiento del cliente es mayor de lo esperado, ya que utiliza menos del 1% del tiempo disponible en cada ciclo de simulación.

5.8. Análisis de efectividad

Para poder evaluar la efectividad del cliente se ha realizado un pequeño torneo en el que sólo se han ejecutado las simulaciones que enfrentaban al cliente implementado. Los rivales elegidos han sido los dos participantes en el torneo del año pasado y el participante del equipo *UofA* del año 2007. Los resultados obtenidos se muestran en la *tabla 19*.

Hay que tener en cuenta que estos resultados son reales, pues durante el desarrollo del cliente se ha enfrentado a sí mismo para hacer las pruebas, con el objetivo de no saber a qué nos podíamos enfrentar. La idea era implementar una estrategia genérica con una elevada capacidad de adaptación, lo cual es un problema muy complejo.

	UofA game 3 2007	UofA game 3 2008	Blekinge3 2008
Victorias	25 (100%)	22 (88%)	5 (20%)
Empates	0 (0%)	3 (12%)	3 (12%)
Derrotas	0 (0%)	0 (0%)	17 (68%)

Tabla 19: Resultados del torneo

Los resultados obtenidos son realmente buenos, pues contra el cliente del equipo *UofA* del año 2007, *BlackRose* gana las veinticinco partidas, lo cual es algo increíble, ya que este equipo es el que ha desarrollado ORTS con Michael Buro a la cabeza. La estrategia que ellos implementan consiste en fundar más asentamientos y entrenar sólo unidades marine, con los que atacar al enemigo continuamente.

El jugador del equipo *UofA* del año 2008 es una versión mejorada y más depurada. Los resultados obtenidos contra dicho cliente son excelente, ya que *BlackRose* gana veintidós de las veinticinco partidas, el resto logra el empate, es decir, transcurren los veinte minutos y el servidor no da la victoria a ninguno de los dos jugadores.

El participante *Blekinge3* es el mejor del torneo, implementa una estrategia muy agresiva que consiste en construir los edificios *Barracks* y *Factory* lo antes posible para poder entrenar unidades *tank*, con las que realizar ataques al oponente. En esta ocasión, *Blekinge3* logra el sesenta y ocho por ciento de las victorias, por lo que es el campeón.

Con estos resultados se puede afirmar que *BlackRose* tiene una tasa alta de efectividad, aunque gracias a este torneo se han puesto de manifiesto sus principales puntos débiles: el algoritmo para determinar la ubicación de una nueva construcción y la estrategia para la acción *gather*. Además, puede que la estrategia implementada sea demasiado conservadora.

5.9. Torneo ORTS 2009

Pese al retraso sufrido, finalmente el torneo se llevó a cabo con dos participantes, entre los que no se encuentra el equipo *UofA*. En esta edición, la competición consistió en setenta y dos partidas, cuyo resultados se muestran en la *tabla 20*. No obstante, dichos datos pueden consultarse en la página oficial del torneo[19].

	Puntuación	Ratio	Desconexiones
Blekinge	51	70.8	0
BlackRose	21	29.2	19

Tabla 20: Resultados torneo ORTS 2009

[19] http://www.cs.ualberta.ca/~mburo/orts/AIIDE09/

Llama la atención las desconexiones de BlackRose, pues suponen el 37% de las partidas perdidas, ya que cuando un participante se desconecta, la partida se da por finalizada, dando la victoria al oponente.

Este hecho también sorprendió a Michael Buro, pues comunicó que la salida mostrada no daba pistas sobre el motivo de dichas desconexiones. La única hipótesis que se maneja es que las simulaciones se lanzaron sobre un ordenador de 64 bits y los clientes fueron compilados con la opción *–m32*, por lo que las librerías *SMILE* [1] suministradas con el cliente pudieron provocar dichas desconexiones, debido a que la versión no era apta para 64 bits.

Como se puede apreciar en la tabla anterior, en este torneo no existe el empate, pues en función de las acciones que realice el jugador se asigna una cantidad de puntos y en caso de empate, la victoria corresponde al cliente que más puntos obtenga.

Los resultados obtenidos son mejores que los del torneo realizado en la sección anterior, pues el porcentaje de derrotas aumenta ligeramente, pero el de victorias incrementa casi un 10%, del 20% al 29,2%.

Capítulo 6

Conclusiones

6. Conclusiones

Es imprescindible saber manejar e interpretar las particularidades de razonar con incertidumbre, es decir, la información imprecisa e incompleta, porque un tratamiento erróneo implica tomar decisiones incoherentes y equivocadas. Es muy importante tener siempre presente que la información que se maneja es incompleta y además imprecisa, por lo que debe analizarse con sumo cuidado, dejando margen de error a las interpretaciones. Tratar de suponer los vacíos que presenta suele ser muy peligroso, porque si sobre un dato impreciso se realizan predicciones, el resultado suele ser equivocado o más impreciso que el original.

Cuando se razona con incertidumbre en tiempo real es imprescindible actualizar constantemente los datos con los que se trabaja, ya que en este caso, la información además es temporal y en ocasiones circunstancial. Por ello, muchas veces es aconsejable omitir ciertos datos, pues suelen ayudar a que las decisiones tomadas sean menos precisas.

Por ejemplo, en el dominio del proyecto, durante una partida se ven a muchos enemigos (exploradores, enemigos que huyen al ver como se disuelve el ataque que estaban realizando, enemigos que anulan un ataque, etc.), por lo que se podría almacenar las coordenadas y el instante en el que se ve a cada uno. A esto hay que añadirle el tipo de unidad, pues no se puede identificar a un enemigo concreto. Además, cuando se ve a una unidad enemiga, esta puede que vea a la nuestra, por lo que el oponente puede optar por moverla nada mas irnos del lugar o esconder unidades militares con el objetivo de engañar sobre el poder militar que realmente dispone. Toda esta información es demasiado temporal y circunstancial, por lo que desde mi punto de vista, almacenarla y utilizarla es un gran error.

ORTS no es un simple juego, es algo más, pues permite buscar soluciones a problemas reales como pathfinding o estrategias a llevar a cabo en un enfrentamiento bélico. Es un juego muy similar a Starcraft, aunque en este caso ORTS implementa técnicas reales y no hace trampas. Un ejemplo sería las colisiones de unidades, donde en ORTS suceden y es un problema complejo de solucionar, mientras que en Starcraft esto se omite, es decir, al procesar las rutas se omite la colisión y se dibuja de forma que parece que la evitan. Desde mi punto de vista, la principal limitación es la escasa documentación existente y una posible extensión sería la inclusión de unidades marítimas y recursos marítimos, como la gasaolína.

Capítulo 7

Líneas futuras

7. Líneas futuras

La implementación de un cliente para el tercer escenario de ORTS engloba muchos aspectos y algunos de ellos, se alejan demasiado del ámbito del proyecto, por lo que han tenido que ser simplificados.

Por esta razón, existen muchas posibles mejoras y ampliaciones deseables. Las más relevantes son las siguientes:

- **Ruta accesible entre dos puntos.** Determinar si un *cluster* de minerales es accesible desde un determinado edificio es imprescindible, debido a que existen situaciones donde el *control center* tiene más de uno a su alrededor. En estos casos, saber si son accesibles desde el mismo es muy relevante, pues se podrían explotar de forma paralela sin tener la necesidad de fundar una nueva base.

  En el proyecto, se ha implementado una simplificación al problema. Para determinar si una ruta es accesible, se calculan tres (la propia ruta, una paralela superior y una paralela inferior). Acto seguido, se someten a un estudio para determinar si alguna está libre de obstáculos. En caso afirmativo, la ruta original se declara accesible.

- **Ubicación de los edificios construidos.** Calcular la ubicación de los edificios a construir en una base, es una tarea que conlleva mucha responsabilidad, pues una mala organización de las construcciones puede dividir el asentamiento, inhabilitando[20] unidades.

  Este aspecto ha sido cortado en el proyecto, pues se edifica en la posición actual de la unidad encargada de la construcción. No obstante, siempre se intenta utilizar una unidad asignada a la tarea *gather*, con el objetivo de construir los nuevos edificios relativamente cerca del *control center* y del *cluster* de minerales, obteniendo una organización aceptable, en la mayoría de los casos.

[20] Hace referencia a las situaciones donde hay unidades que se quedan bloqueadas por estar rodeadas de edificios y obstáculos del escenario.

- **Colaboración entre bases.** Cuando se dispone de más de un asentamiento, es importante que exista colaboración entre ambas para intercambiar unidades, ayudar en la defensa de la base en momentos críticos, etc.

  Esta acción no se ha implementado, simplemente se iguala el poder militar de ambas al realizar un ataque. Esto simula una colaboración entre las bases, pero no es óptimo, ya que solo se realiza a la hora de crear un nuevo ataque.

- **Pérdida de edificios *control center*.** Cuando se pierde el edificio principal de una base, es necesaria la evaluación de la situación para determinar si el asentamiento se puede reconstruir o está totalmente perdido.

  En el cliente se ha implementado una simplificación de la evaluación de la situación cuando se pierde un edificio principal de la base, pero es necesario el análisis real para tomar las acciones optimas minimizando los daños.

- **Estrategias desechadas.** En el capítulo 4.3 Capa Táctica, se desechan un conjunto de estrategias por falta de tiempo y complejidad. Dichas estrategias, podrían aumentar el dinamismo de la lógica del cliente, mejorando su rendimiento.

Al margen de lo comentado, sería muy deseable realizar un estudio comparativo entre el modelo de toma de decisión implementado y diferentes técnicas que se utilicen en los juegos comerciales de estrategia en tiempo real, utilizando los tiempos de respuesta como criterio principal.

Capítulo 8

Bibliografía

8. Bibliografía

[1] *GeNIe & SMILE*. http://genie.sis.pitt.edu.

[2] Bruce Eckel. *Thinking in C++: Introduction to Standard C++, Volume One, Second edition*. Prentice Hall, 2000.

[3] A. J. Millán. *Curso de C++*. http://www.zator.com/Cpp/index.htm.

[4] José Antonio Gámez Martín. *Propagación en redes Bayesianas: métodos exactos*. www.dsi.uclm.es/asignaturas/300209/material/prop-exacta.pdf.

[5] Richard E. Neapolitan. *Learning Bayesian Network*. Prentice Hall, 2003.

[6] Kevin B. Korb y Ann Nicholson. *Bayesian Artificial Intelligence*. Chapman & Hall/CRC, 2004.

[7] Ian Millington. *Artificial Intelligence for Games*. Elsevier, 2006.

[8] John David Funge. *Artificial Intelligence For Computer Games: An introduction*. Peters Corp., 2004.

[9] *ORTS*. http://www.cs.ualberta.ca/~mburo/orts.

Anexo A

Diagrama de clases

9. Anexo A: Diagrama de clases

Para facilitar la compresión de la estructura del cliente, en la *ilustración 68* se muestra un diagrama de clases simplificado. En él, aparecen todos los aspectos comentados a lo largo del presente documento.

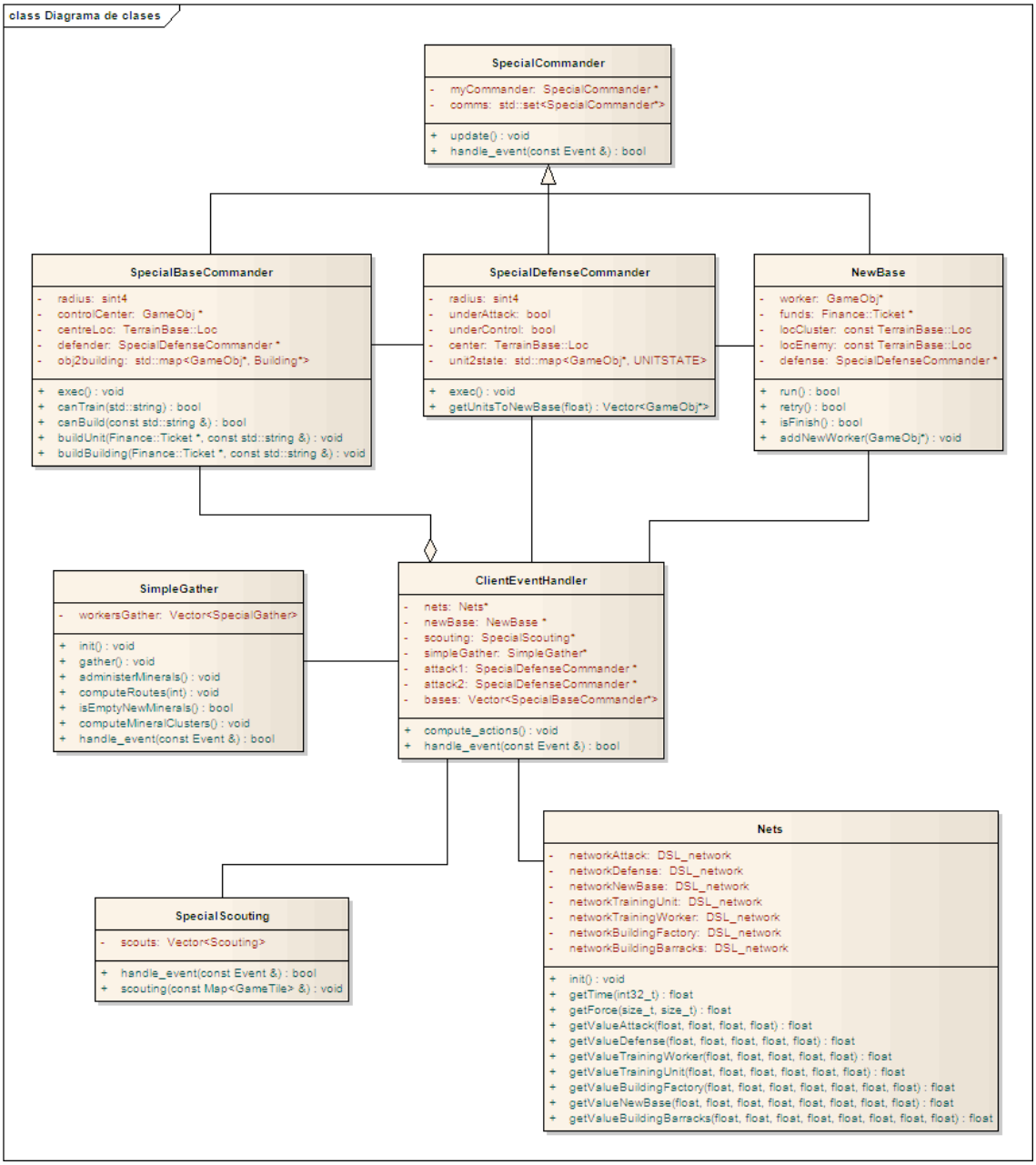

Ilustración 68: Diagrama de clases

Para solicitar un determinado evento, se debe realizar una petición al modulo *Watcher*[21] mediante su interfaz pública. El evento más demandado por parte del cliente, es la notificación de la pérdida de las unidades. Para solicitar dicho evento, se debe invocar el método *notifyOnDeath(EventHandler * handler, GameObj * unit)* del módulo *Watcher*, dónde el primer parámetro indica el objeto al que se debe informar y el segundo, la unidad sobre la cual se solicita el evento.

De la signatura del método, se obtiene que el objeto que va a recibir la notificación debe ser de tipo *EventHandler*. Por ello, los objetos que solicitan eventos deben heredar[22] directamente (o indirectamente) de la clase *EventHandler*.

Cada módulo que solicite el evento de la pérdida de unidades, debe implementar en el método *handle_event(const Event &event)* el tratamiento del mismo[23]. Esto supone repetir el mismo fragmento de código en todos los módulos involucrados. Para evitar la repetición del código, se puede optar por la herencia indirecta de la clase *EventHandler* mediante la clase *SpecialCommander*. De esta forma, se evita la repetición de código, aunque se deben implementar otros métodos que pueden resultar innecesarios.

Una tercera solución, sería la definición de una clase que herede de *EventHandler* e implemente el tratamiento del evento.

Este problema se ha resulto de las dos primeras formas descritas, con el objetivo de mostrar la implementación de ambas. En el diagrama de clases de la *ilustración 68*, se puede observar como las clases *SimpleGather* y *SpecialScouting* implementan el método de tratamiento de eventos, mientras que la clase *NewBase* hereda de *SpecialCommander*, omitiendo dicha implementación.

La clase *SpecialCommander* contiene un atributo denominado *myCommander*. Dicho atributo, representa el comandante inmediatamente superior en la jerarquía de comandantes. Cuando un comandante almacena en dicho atributo el valor *0*, significa que es la raíz de la jerarquía, es decir, el comandante jefe.

21 Es un módulo de apoyo que ofrece ORTS, a través del cual se pueden solicitar diversos eventos, como el descubrimiento de minerales en el escenario.

22 Propiedad de lenguajes de programación orientados a objetos [2] y [3].

23 Siempre que se recibe un evento por pérdida de unidades, se invoca el método *onLoseUnit(GameObj * unit)* con la unidad que se ha perdido.

La lógica del cliente se ejecuta en el método *compute_actions()* de la clase *ClientEventHandler*. Dicho método es invocado desde *handle_event(const Event &e)* (de la misma clase), al recibir un evento de tipo *VIEW_MSG*, es decir, cada vez que el servidor procesa las acciones tomadas en el ciclo anterior de la simulación y envía la nueva situación.

Anexo B

Descripción de los módulos utilizados e implementados

10. Anexo B: Descripción de los módulos utilizados e implementados

Para poder comprender el funcionamiento de los módulos utilizados, es necesario conocer la estructura interna de los comandantes (*SpecialCommander*). Cada uno, tiene una lista de sub-comandantes de los que es responsable, definiéndose así una jerarquía de comandantes, en la que el nodo raíz es el principal (el jefe).

Además, cada comandante puede recibir eventos (por ejemplo, cuando una determinada unidad muere. Normalmente, al insertar una unidad en un módulo, se le asigna un seguimiento para ser informado en el instante de su muerte). Cuando se recibe un evento, es tratado por los métodos *handle_event(const Event &event)* y *onEvent(const Event &event)*, los cuales invocan los métodos necesarios para procesar la información recibida (en el caso del presente ejemplo, se invocaría *onLoseUnit(GameObj* gob)*).

A continuación, se van a describir los módulos implementados y utilizados en el cliente.

- Módulo de ataque/defensa

Formado por los ficheros *SpecialDefenseCommander.C* y *SpecialDefenseCommander.H*, implementa la acción de controlar una región circular mediante un conjunto de unidades. Por controlar, se entiende que al menos el 75% de las unidades estén en el interior de la región definida y no exista la presencia de ninguna unidad enemiga (en el interior de la región). Cuando esto sucede, la variable *underControl* tomará el valor *true*. Además, el modulo nos informa si hay unidades enemigas en el radio de acción, mediante la variable *underAttack*. Es importante conocer la diferencia entre ambas variables, ya que la última sólo informa de la presencia de unidades enemigas, mientras que la primera además informa de la posición actual de las unidades.

Los métodos más relevantes del módulo son *exec()* (encargado de ejecutar la acción, es decir, evaluar la situación actual, dar valores a las variable descritas y dar órdenes a las unidades) y *getUnitsToNewBase(float a)* (obtiene a/3 unidades de cada tipo, eliminando del módulo las unidades obtenidas. Este método suele ser utilizado para crear un ataque y para obtener las unidades militares encargadas de asegurar el terreno al fundar una nueva base).

El módulo se puede utilizar de dos formas. Siendo comandante jefe sin tener en ningún momento sub-comandantes o siendo directamente un sub-comandante. En el primer caso, es

necesario invocar en cada ciclo el método *exec()*, ya sea mediante una invocación explícita o implícita mediante el método *update()*. En el segundo caso, simplemente se crea el objeto indicado su superior, de forma que la responsabilidad de su ejecución dependa de su comandante jefe. Para realizar ataques, se utiliza como jefe, mientras que para la defensa de las bases, se emplea como sub-comandante.

- Módulo de creación de una base

Consiste en asegurar la zona dónde se va a fundar la nueva base, calcular la ubicación más segura para el edificio *control center* y movilizar la unidad worker hasta la posición calculada (moviendo a las unidades militares para que molesten lo menos posible a la hora de construirlo). El módulo está formado por los ficheros *NewBase.C* y *NewBase.H*. La construcción en sí del edificio y la creación del objeto de la nueva base no es tarea del presente módulo (el código de ambas tareas corresponde con el método *buildNewBase* en la clase *ClientEventHandler*).

Los métodos *run()* (encargado de dar ordenes a las unidades que componen el módulo. Además, indica si ha ocurrido algún problema que debe examinarse.), *retry()* (debe ejecutarse cuando se ha detectado algún problema. Evalúa la posibilidad de reintentar el procedimiento, es decir, comprueba si el problema reside en que la unidad *worker* se ha quedado bloqueada o ha muerto), *addNewWorker(GameObj* gob)* (asigna como unidad *worker* la unidad que recibe por parámetros. Además, permite que se siga ejecutando el procedimiento desde un estado seguro) y *isFinish()* (indica si el procedimiento ha finalizado) son los más relevantes.

Utilizar el módulo es una tarea sencilla. Simplemente hay que crear el objeto e invocar el método *run()* en cada ciclo de la simulación. Si el valor devuelto por dicho método se corresponde con el valor *false*, hay que examinar el problema. Hay dos tipos de problemas, el relativo a la unidad *worker* y el relativo a las unidades militares. El primero es salvable, es decir, la unidad puede ser reemplazada por otra y seguir la ejecución desde un estado seguro. Por otro lado, el segundo es definitivo, por lo que hay que intentar salvar la unidad *worker* y devolver el fondo de minerales.

Además, en cada ciclo hay que preguntar (mediante el método *isFinish()*) si el módulo ha terminado su ejecución, para proceder a la construcción del edificio principal y a la creación de la nueva base.

- Módulo scouting

El objetivo es explorar las regiones del escenario que no han sido descubiertas. Para que una región que no ha sido explorada sea de interés, esta debe estar formada por al menos tres *tiles*, es decir, la región debe suponer más de un 7.3% del escenario[24]. Los ficheros que componen el módulo son *SpecialScouting.C* y *SpecialScouting.H*.

El método *scouting()* monitoriza y asigna órdenes a todos los *scout*. Además, detecta cuando una unidad se ha quedado parada, asignándole un nuevo destino.

Para utilizar el módulo que implementa la acción *scouting*, basta con crear al inicio un objeto y ejecutar de forma explícita el método *scouting()*. Dicho método sólo debe ser invocado si se dispone de algún *scout*. El módulo, al igual que el resto, dispone de métodos que insertan y eliminan unidades. Además, todos los módulos aplican seguimiento a las unidades que están bajo su tutela, con el objetivo de desecharlas en el instante de su muerte.

- Módulo gather

Es el módulo más complejo y amplio (formado por *SimpleGather.C*, *SimpleGather.H, SpecialRoute.C*, *SpecialRoute.H*, *SpecialRouteAllocator.C*, *SpecialRouteAllocator.H*, *SpecialWorker.C* y *SpecialWorker.H*).

Mediante eventos, recibe los minerales encontrados en el escenario, agrupándolos en clusters para su almacenamiento. Además, calcula las rutas de acceso a los mismos para cada una de las bases, es decir, para cada *control center*.

La agrupación en clusters y el cálculo de las rutas de acceso sirven para gestionar de forma eficiente y sencilla la actividad de recolección de minerales. El método *handle_event(const Event &event)* recibe y procesa los eventos de minerales encontrados y minerales destruidos. Por su parte, el método *administerMinerals()* es el encargado de procesar los minerales encontrados, identificando si se trata de minerales nuevos o no. En caso afirmativo, se apoya en los métodos *computeMineralClusters()* y *computeRoutes()* para agruparlos en clusters y calcular las rutas de acceso.

[24] El escenario esta formado por 4096 *tiles.*

Es imprescindible invocar el método *init()* al inicio de la partida, pues es el encargado de solicitar los eventos sobre los minerales, es decir, solicita que cada vez que se encuentre un mineral en el escenario, se le comunique mediante un evento.

Por último, el método *gather()* asigna a las unidades las órdenes necesarias para realizar la actividad de recolección de minerales en cada ciclo de la simulación.

Utilizar el módulo de recolección es una tarea sencilla gracias a la interfaz que ofrece. Al inicio de la partida es necesaria la creación de un objeto de tipo *SimpleGather* y la invocación al método *init()*. En cada ciclo de reloj hay que realizar dos tareas. La primera consiste en preguntar si se han encontrado minerales mediante el método *isEmptyNewMinerals()*. En caso afirmativo se debe ejecutar el método *administerMinerals()* para agrupar y calcular las nuevas rutas de acceso. La segunda consiste en ejecutar el método *gather()*, si y sólo si se cuenta con al menos una unidad asignada a la tarea.

- Módulo de gestión de una base

La gestión de la base engloba la defensa de la misma, la construcción de nuevos edificios y el entrenamiento de unidades. El módulo está formado por los ficheros *SpecialBaseCommander.C*, *SpecialBaseCommander.H*, *SpecialBuildCommander.C* y *SpecialBuildCommander.H*.

El método principal del módulo se denomina *exec()*, y es el encargado de gestionar y monitorizar la construcción de edificios y el entrenamiento de unidades. Para iniciar un entrenamiento o una construcción, se debe invocar el método *buildUnit(const std::string &unitBPname, Finance::Ticket *funds)* o *buildBuilding(const std::string &buildingBPname, Finance::Ticket *funds)*, donde ambos reciben un parámetro denominado *funds*. Dicho parámetro consiste en una reserva de la cantidad de minerales necesaria para llevar a cabo la acción.

Antes de iniciar una actividad, es recomendable comprobar que mediante los recursos actuales se puede llevar a cabo. Para realizar dicha comprobación, se pueden utilizar los métodos *canTrain(const std::string bp_name)* y *canBuild(const std::string objectBPname),* aunque antes de utilizarlos, se recomienda la comprensión del código de ambos.

El módulo utiliza la estructura jerárquica de los comandantes, dónde el objeto de tipo *SpecialBaseCommander* es el jefe y el resto son sub-comandantes (como la defensa de la base). Por esta razón, es necesaria la invocación del método *update()* de la base en cada ciclo de la simulación.

- Módulo de las redes bayesianas

Permite ejecutar cada una de las redes indicando los datos de entrada. Además, ofrece un conjunto de métodos que realizan el cálculo de algunos valores de entrada a las mismas. El modulo consta de dos ficheros, *Nets.C* y *Nets.H.*

El método *init()* es el responsable de cargar las siete redes bayesianas en memoria, mientras que los métodos *getValue* ejecutan una determinada red, utilizando como valores de entrada a la misma, los valores indicados por parámetros. Por último, los métodos *get* forman el conjunto de métodos de apoyo mencionado anteriormente.

La utilización del módulo es sencilla e intuitiva. Al inicio de la simulación se crea un objeto de tipo *Nets* y se invoca el método *init()*. Cada vez que se necesite ejecutar una red, se invoca su método correspondiente con los parámetros necesarios (calculados previamente en función de la situación actual de la simulación).

Anexo C

Tutorial de iniciación a ORTS

11. Anexo C: Tutorial de iniciación a ORTS

El objetivo de anexo es la creación de un tutorial que describa el proceso completo de creación de un cliente ORTS para la plataforma Mac OS X. Dicho proceso comienza con la instalación del ORTS y termina con la codificación del cliente funcional.

Antes de empezar, vamos a hacer una pequeña introducción a ORTS, con el objetivo de comprender un poco mejor el entorno en el que vamos a trabajar.

11.1. Introducción

Es un entorno de programación abierto (software libre) para juegos de estrategia en tiempo real. El objetivo de dicho entorno es el estudio de problemas de Inteligencia Artificial en tiempo real, ya que no es un simple juego de RTS, sino es un motor de juegos de estrategia en tiempo real, en el que se pueden definir diferentes dominios.

El problema de los actuales juegos de RTS es que son de código cerrado, además de utilizar la tecnología *peer-to-peer*, la cual entorpece más las posibilidades de investigación.

Las principales características del ORTS son las siguientes:

- Es un software gratuito con licencia *GPL*, por lo que todo el código fuente está disponible y se puede distribuir libremente. Además, el código puede ser modificado por cualquiera para posibles mejoras, para ampliar la funcionalidad o lo que se desee.

- Utiliza una topología cliente-servidor, con un protocolo de comunicación abierto.

- Flexibilidad. El usuario puede definir el escenario que desee, indicando en un *script* todos los tipos de unidades, estructuras e interacciones. Dicho *script* será cargado y ejecutado por el servidor ORTS. Posteriormente, el cliente se conectará al servidor generando acciones sobre los objetos del juego.

- El lenguaje de programación utilizado tanto para el servidor como para los clientes es C++.

- ORTS es un software independiente de la plataforma, ya que funciona bajo Windows, Linux y Mac OS X.

- Posee interfaz gráfica, de la cual se puede configurar la resolución y división de la pantalla. Además, para una mayor flexibilidad, se puede ejecutar sin GUI.

- El cliente tiene todo el control de cada una de las unidades, ya que no hay comportamientos de bajo nivel predefinidos, son todo acciones atómicas.

- La IA es local para cada uno de los clientes.

- ORTS permite jugar contra tus propios clientes (o contra cualquier cliente) mediante el teclado y el ratón. Esto permite simular diferentes situaciones sin tener que esperar a que se produzcan por sí solas, ahorrando muchísimo tiempo. De esta forma, podemos observar el comportamiento de las tropas del cliente (rival) para diferentes situaciones de forma rápida y sencilla.

11.2. Instalación de ORTS en Mac OS X

En este apartado vamos a detallar la instalación del ORTS en un MAC OS X Leopard. Además, debido a que se necesitan varias librerías para poder compilar ORTS, también se va a describir brevemente cómo instalarlas.

11.2.1. Instalación de las librerías necesarias

Antes de instalar ORTS hay que instalar un conjunto de librerías para su correcto funcionamiento. Dichas librerías son las siguientes:

- Librería estándar de C++

- Boost

- SDL
- SDL_net
- Glew (con OpenGL)

A continuación vamos a describir brevemente cómo instalar cada una de ellas.

- ***Librería estándar de C++***

Esta librería debería estar ya instalada con el Sistema Operativo. En el caso de que no lo estuviese, se puede instalar de forma sencilla desde el CD de instalación del Mac OS X.

- ***Boost***

Para instalar esta librería primero hay que descargársela desde su página principal[25]. Una vez descargada, se descomprime el fichero, obteniendo una carpeta llamada *boost_X_XX_X*, dónde las X´s representan la versión actual, en nuestro caso *boost_1_37_0*.

Dentro de la carpeta hay un fichero llamado *INSTALL*, el cual indica que las instrucciones para la instalación (entre otras cosas) se encuentran en la sección *Getting Started,* del fichero *index.html* de la misma carpeta.

- ***SDL y SDL_net***

Empezaremos por la librería *SDL*, la cual se puede descargar de la sección *Downloads/SDL 1.2* de su página principal[26]. Deben descargarse la *Runtime Library* y la *Development Library*.

Ambos ficheros son *dmg*, por lo que simplemente hay que hacer doble click para instalarlos. Primero se debería instalar la *Runtime* y luego la *Development*.

Por otro lado, la librería *SDL_net* de puede descargar desde este enlace[27]. Y para instalarla hay que hacer lo mismo que para la *SDL*, ya que también es un fichero *dmg*.

25 http://www.boost.org/

26 http://www.libsdl.org/

27 http://www.libsdl.org/projects/SDL_net/

- ***Glew (con OpenGL)***

Hay dos formas de instalar la librería *Glew*, las cuales se describen a continuación:

- *Primera forma*

Actualmente la librería se encuentra en su versión 1.5.0 y se puede descargar desde el siguiente enlace[28]. Hay que descargarse la versión *darwin*, tanto del fichero *agl* como *glx*.

Una vez descargados ambos ficheros, seguimos las instrucciones descritas en el apartado *Installation* del enlace anterior.

- *Segunda forma*

La librería *Glew* la vamos a instalar utilizando los *darwin ports*, pero antes de utilizarlos hay que instalarlos, a no ser que ya lo estén. El proceso de instalación es sencillo, y está descrito en su página oficial[29].

Una vez instalados, ya podemos instalar la librería ejecutando el comando reflejado en la *ilustración 69*.

```
Macintosh-2:/ sergio$ cd /opt/local/bin
Macintosh-2:bin sergio$ sudo ./port install glew
```

Ilustración 69: Comando para instalar la librería Glew

Para saber cuándo se ha terminado de instalar y no cerrar la consola antes de tiempo, hay que ejecutar el comando mostrado en la *ilustración 70*.

[28] http://glew.sourceforge.net/

[29] http://darwinports.com/

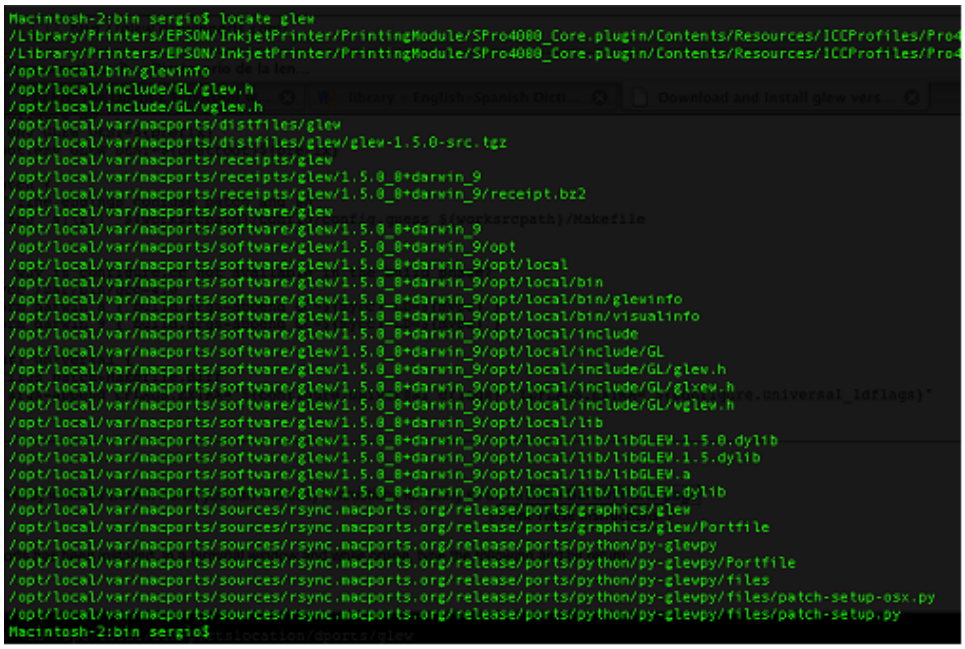

Ilustración 70: Comando locate glew

Si obtenemos la misma salida o una parecida es que la librería ya se ha instalado correctamente.

11.2.2. Instalación del ORTS

Lo primero que hay que hacer es descargar el código fuente, el cual está disponible en varios sitios (snapshot[30], svn, etc.). En este tutorial se opta por utilizar el sistema de control de versiones *svn*.

SVN permite actualizar de forma rápida y sencilla la versión del ORTS. Si no se tiene ninguna, descarga la versión más reciente. Primero, debemos crear una carpeta dónde queremos que se instale ORTS. A continuación, abrimos una consola y accedemos a la carpeta que nos acabamos de crear.

Para actualizar el módulo ORTS tenemos que ejecutar el comando de la *ilustración 71* desde la consola.

```
Macintosh-2:~ sergio$ svn co svn://[anonymous]@bodo1.cs.ualberta.ca:/all/pubsoft/orts
```

Ilustración 71: Actualización del módulo ORTS

[30] http://www.cs.ualberta.ca/~mburo/orts/src_snapshot/snap.html

Cuando aparezca el mensaje de la *ilustración 72*, significará que ya se ha actualizado el módulo ORTS correctamente.

Ilustración 72: Actualización del módulo ORTS terminada

A continuación, tenemos que actualizar el módulo ORTS_DATA, simplemente ejecutando el comando indicado en la *ilustración 73*.

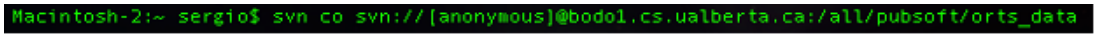

Ilustración 73: Actualización del módulo ORTS_DATA

Al terminar la actualización se mostrará el mensaje contenido en la *ilustración 74*.

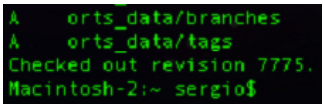

Ilustración 74: Actualización del módulo ORTS_DATA terminada

Una vez que tenemos todo el código fuente, tenemos que realizar unos pequeños cambios en él para asegurarnos que funcione correctamente. Lo primero que tenemos que modificar es el *Makefile*, introduciendo en la parte de la instalación para Mac unos cuantos directorios. Dicho fichero se encuentra en *./orts/trunk/* y el contenido original de la parte que tenemos que modificar es el que contiene la *ilustración 75*.

```
OS_MAC := 1

MACROS    := -DGCC -DOS_MAC -DHAS_TYPEOF=1 -DTRANSFORM_F
DTARGET_API_MAC_CARBON=1
USR_FRAME := /Library/Frameworks
SYS_FRAME := /System/Library/Frameworks
INC_OPTS += -I$(USR_FRAME)/SDL.framework/Headers \
            -I$(USR_FRAME)/SDL_net.framework/Headers \
            -I/usr/include/GL \
            -L$(USR_FRAME) -L$(SYS_FRAME)
```

Ilustración 75: Makefile original

Concretamente tenemos que introducir la ruta de la librería *boost*, *GL* y *lib*. Obteniendo como resultado la *ilustración 76*.

```
OS_MAC := 1

MACROS    := -DGCC -DOS_MAC -DHAS_TYPEOF=1 -DTRANSFORM_MOUS
DTARGET_API_MAC_CARBON=1
USR_FRAME := /Library/Frameworks
SYS_FRAME := /System/Library/Frameworks
INC_OPTS += -I$(USR_FRAME)/SDL.framework/Headers \
            -I$(USR_FRAME)/SDL_net.framework/Headers \
            -I/usr/include/GL \
            -I/Applications/boost_1_37_0 \
            -I/opt/local/include/GL \
            -L/opt/local/lib \
            -L$(USR_FRAME) -L$(SYS_FRAME)
```

Ilustración 76: Makefile modificado

Hay que tener en cuenta, que esta modificación puede variar de una máquina a otra, ya que depende de dónde se hayan instalado las librerías.

El siguiente cambio que tenemos que realizar es en el fichero *./orts/trunk/libs/pathfinding/dcdt/sr/src /sr_array.h*, en el que tenemos que localizar el fragmento de código mostrado en la *ilustración 77*.

```
/*! Operator version of X& get(int i), but returning a non const reference.
    No checkings are done to ensure that i is valid. */
X& operator[] ( intptr_t i ) { return ((X*)_data)[i]; }
```

Ilustración 77: sh_array.h original

Una vez localizado, hay que reemplazarlo por el fragmento señalado en la *ilustración 78*.

```
/*! Operator version of X& get(int i), but returning a non const reference.
    No checkings are done to ensure that i is valid. */
X& operator[] ( int i ) { return ((X*)_data)[i]; }
```

Ilustración 78: sh_array.h modificado

Si observamos ambas ilustraciones, comprobamos que es suficiente con cambiar el tipo de parámetro de la variable *i*, de *intptr_t* a *int*.

En la misma ruta, se encuentra el fichero *sr_bv_math.cpp*, en el cual también tenemos que realizar una pequeña modificación. Por lo cual, hay que localizar el fragmento del código de la *ilustración 79*.

```
#ifdef WIN32
#include <sr_bv_math.h>
#else
#include <SR/sr_bv_math.h>
#endif
```

Ilustración 79: sr_bv_math.cpp original

Una vez localizado, hay que modificarlo obteniendo como resultado la *ilustración 80*.

```
#ifdef WIN32
#include <sr_bv_math.h>
#else
#include <SR/sr_bv_math.h>
#endif
using namespace std;
```

Ilustración 80: sr_bv_math.cpp modificado

Si nos fijamos con un poco de atención, nos damos cuenta de que sólo hay que introducir la última sentencia de la *ilustración 80*.

El cuarto cambio a realizar, se tiene que hacer en tres ficheros diferentes. A continuación se muestran los fragmentos de código de los tres ficheros que hay que localizar (*ilustraciones 81, 82 y 83*)

- *./orts/trunk/apps/orts/src/orts_main.c*

```
};

int main(int argc, char **argv)
{
 try {
#if USE_EFENCE
```

Ilustración 81: orts_main.c original

- *./orts/trunk/apps/orts/src/ortsg_main.c*

```
typedef SimpleTerrain::ST_ForceField Terrain;

int main(int argc, char **argv)
{
  char mydir[81];
  getcwd(mydir, 80);
  glutInit(&argc, argv);
  chdir(mydir);
```

Ilustración 82: ortsg_main.c original

- *./orts/trunk/apps/orts/src/sampleai_main.c*

```
private:
  SampleAIState * state;
};

//==============================

int main(int argc, char * * argv)
{
  signals_shut_down(true);
```

Ilustración 83: sampleai_main.c original

Una vez localizados los fragmentos de código, los modificamos incluyendo la sentencia *#undef main* antes de la definición del método main, obteniendo los resultados contenidos en las *ilustraciones 84, 85 y 86*.

- *./orts/trunk/apps/orts/src/orts_main.c*

```
};

#undef main
int main(int argc, char **argv)
{
 try {
#if USE_EFENCE
```

Ilustración 84: orts_main.c modificado

- *./orts/trunk/apps/orts/src/ortsg_main.c*

```
typedef SimpleTerrain::ST_ForceField Terrain;
#undef main
int main(int argc, char **argv)
{
  char mydir[81];
  getcwd(mydir, 80);
  glutInit(&argc, argv);
  chdir(mydir);
```

Ilustración 85: ortsg_main.c modificado

- *./orts/trunk/apps/orts/src/sampleai_main.c*

```
private:
  SampleAIState * state;
};

//====================================
#undef main
int main(int argc, char * * argv)
{
  signals_shut_down(true);
```

Ilustración 86: sampleai_main.c modificado

Una vez realizados estos cambios ya estamos listos para compilar ORTS. Lo primero que tenemos que hacer es asignar a la variable de entorno *OSTYPE* el valor *DARWIN*, para que ORTS se compile en Mac OS X.

La forma más sencilla es mediante el siguiente comando: "*export OSTYPE=DARWIN*" (para *bash*, en el caso de utilizar otro interprete de comandos, habrá que utilizar el comando apropiado, por ejemplo, para *csh*, se utiliza *setenv* en vez de *export*). El problema que tiene esta

solución, es que cada vez que queramos recompilar tendremos que ejecutar antes este comando. Otra opción más cómoda a medio plazo, es configurar la variable de entorno para que siempre tenga asignado el valor *DARWIN* en el fichero *.profile*.

Una vez configurada la variable de entorno, accedemos a la ruta *./orts/trunk* y ejecutamos el comando mostrado en la *ilustración 87*.

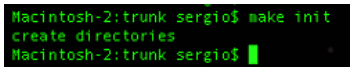

Ilustración 87: Inicializar la compilación de ORTS

Una vez creados los directorios, compilamos ejecutando el comando *make*. Si la compilación se ha realizado correctamente, veremos por pantalla el mensaje de la *ilustración 88*.

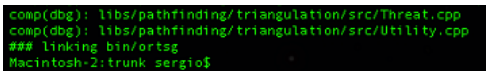

Ilustración 88: Compilar ORTS

11.3. ¿Cómo ejecutar ORTS?

Como ya mencionamos anteriormente, ORTS está basado en la estructura cliente/servidor, por lo cual, primero habrá que ejecutar un servidor y posteriormente todos los clientes que se deseen.

Para ejecutar el servidor por defecto, simplemente hay que utilizar el siguiente comando (desde *./orts/trunk*):

```
./bin/orts
```

El servidor se puede configurar a nuestro gusto mediante opciones a la hora de ejecutarlo. Dichas opciones se pueden ver en el código fuente, pero una forma más sencilla, ejecutarlo con alguna opción incorrecta, como por ejemplo la siguiente:

```
./bin/orts -help
```

Algunas de las opciones del servidor son las siguientes:

- ***-game1*** para cargar el escenario 1 del torneo.
- ***-game2*** para cargar el escenario 2 del torneo.
- ***-game3*** para cargar el escenario 3 del torneo.
- ***-game4*** para cargar el escenario 4 del torneo.
- ***-nplayers n*** indica el número de clientes que van a jugar.
- ***-nfog*** indica que no hay niebla de guerra, es decir, que todos los clientes saben dónde están en cada momento los objetos del resto de clientes.
- ***-fplat n*** indica la fracción del terreno elevado en el escenario. Si *n* es 0, el terreno no tendrá ninguna zona elevada.

- ***-bp ruta*** indica la ruta del fichero *blueprint*, el cual contiene información sobre las características del juego (velocidad de movimiento de los marines, objetivos del escenario, etc.).

Una vez que el servidor se está ejecutando, hay que cargar los clientes. Cada uno puede tener sus propias opciones, ya que éstas las implementa el propio cliente.

Al instalar ORTS, se crea un cliente que permite recibir órdenes mediante el ratón y el teclado. Dicho cliente se ejecuta con el siguiente comando (desde *./orts/trunk*):

```
./bin/ortsg
```

Este cliente tiene sus propias opciones, las cuales se pueden consultar de la misma forma que las del servidor:

```
./bin/ortsg –help
```

ORTSG es muy útil, ya que se puede utilizar para jugar contra nuestros propios clientes y poder observar cómo se comportan en las diferentes situaciones que nosotros mismos provoquemos.

11.4. Programación de un cliente ORTS

El objetivo de este apartado es crear un escenario totalmente liso (sin zonas elevadas, minerales, etc), para dos jugadores, con un marine para cada uno. Además de implementar dos clientes, dónde el primero moverá de forma aleatoria al *marine* y el segundo lo moverá de forma aleatoria hasta que encuentre al marine rival, momento en el cual pasará a seguirle a una cierta distancia.

El *makefile* de ORTS permite compilar clientes de forma rápida y sencilla. La única restricción es que los ficheros fuentes tienen que estar en la siguiente ruta:

```
./orts/trunk/apps/nombre_del_cliente/src/ficheros_fuente
```

Además, en la carpeta src debe haber un fichero *app.mk* con las librerías que utiliza, el nombre del cliente y el fichero de reglas (ver código fuente). También existe la posibilidad de

modificar el *makefile* para poder ubicar los clientes en otra ruta o incluso nos podemos crear nuestro propio *makefile*.

Una vez que tengamos los clientes implementados y compilados y el escenario creado, habría que abrir tres terminales y ejecutar los siguientes comandos desde *./orts/trunk*

- Terminal 1: *./bin/orts –example –nplayers 2 –fplat 0*
- Terminal 2: *./bin/waypoints [-usegfx]*
- Terminal 3: *./bin/cheser [-usegfx]*

La opción *–usegfx* es opcional, por si queremos ver al cliente en acción.

11.4.1. Implementación del escenario

Lo primero que tenemos que hacer, es introducir una nueva opción al servidor, la cual usaremos para ejecutar nuestro escenario. Por ello, tenemos que añadir unas líneas al fichero *./orts/trunk/apps/orts/src/orts_main.c*

- En la función static *void add_options()* (línea 30) tenemos que introducir:

```
o.put("-example", "example game");
```

  para que reconozca el parámetro *-example* como una opción válida. *o* es un objeto de tipo *Options*, el cual utilizamos para ejecutar el método *put*. El primer parámetro de dicho método, representa la cadena de caracteres que se tendrá que insertar como argumento al ejecutar el servidor para que la opción se ejecute. El segundo parámetro, es un texto explicativo de la opción, que se mostrará en la ayuda del servidor.

- En la función *void create_orts_world(stringstream &world)* (línea 64), dentro del *else* del segundo *if* anidado, es decir, en la región de código destinada si no usamos la opción *–loadmap* ni *–hfile,* tenemos que hacer lo siguiente:

- Primero definimos una variable de tipo *bool* que se llame *example*,

```
bool example;
```

- Luego, preguntamos si se ha ejecutado la opción *–example* con la siguiente línea:

```
Options::get("-example", example);
```

  Para ejecutar el método *get* no tenemos que utilizar ningún objeto, ya que dicho método es estático. El primer parámetro indica la opción que se desea preguntar y el segundo, es la variable dónde se almacena el resultado de la pregunta.

- Y por último, si la variable *example* es *true*, llamamos a la función *static void generate_example_map(stringstream &world)* mediante la siguiente línea:

```
MapTool::generate_example_map(world);
```

Una vez hecho lo anterior, tenemos que definir el prototipo de la función static void generate_example_map(stringstream &world) e implementarla, ya que estamos haciendo una llamada a una función que no existe. El parámetro que recibe, creo que es dónde se define el propio mapa, por lo que al finalizar la ejecución de la función, va a contener la definición del mapa.

Para ello, tenemos que introducir el prototipo de la función en el fichero *./orts/trunk/libs/serverclient/src/MapTool.h*, mediante la siguiente línea:

```
static void generate_game4_map(std::stringstream &world);
```

Por último, tenemos que implementar la función, la cual se va a implementar en el fichero *./orts/trunk/libs/serverclient/src/TournamentMapTool.c*, junto a las funciones de creación de los escenarios del torneo (game1, etc.),

Simplemente tenemos que crear un escenario para dos jugadores, en el que cada uno tenga su propio marine (Ver Anexo).

11.4.2. Implementación del primer cliente

En este subapartado vamos a comentar cómo se implementa un cliente que mueva de forma aleatoria a sus unidades y tenga la opción de gráficos en 3D.

Nuestro cliente va a estar formado por tres ficheros:

- *waypoints_main.c*
- *ClientEventHandler.h*
- *ClientEventHandler.c*

El fichero waypoints*_main.c* contiene la función *void add_options()*, la función *int main(int argc, char **argv)* y la clase *Looper*.

La función *void add_options(),* simplemente sirve para incluir las opciones que nuestro cliente va a considerar válidas, en nuestro caso *–usegfx* para el modo gráfico en 3D.

La función *main* realiza todas las comprobaciones e inicializaciones necesarias para poder ejecutar el cliente. Entre ellas, comprueba si el modo gráfico está o no activo para realizar las correspondientes gestiones.

Por último, la clase *Looper* contiene una estructura *ClientAIState* (definida en el fichero *ClientEventHandler.h*), un constructor y el método *void loop()*.

El constructor inicializa el atributo *state* de tipo *ClienteAIState* con la estructura *ClientAIState* que recibe por parámetros. Dicha estructura se explica en detalle más adelante.

El método *loop* se va a ejecutar hasta que el juego termine o el cliente finalice (es llamado desde el *main*). Dicho método busca mensajes entrantes procedentes del servidor, si los hay llama a los *handlers* y si no se duerme un milisegundo. Además, realiza gestiones del modo gráfico.

El fichero *ClientEventHandler.h* contiene la definición de la estructura *ClientAIState* y la definición de la clase *ClientEventHandler*.

La estructura *ClientAIState* contiene la información necesaria para conocer el estado actual del juego. Los elementos que forman dicha estructura son los siguientes:

- ***gsm*** es un puntero a *GameStateModule*, gracias al cual podemos conocer información relativa al estado del juego.

- ***gfxm*** es un puntero a *GfxModule*, gracias al cual se actualiza la imagen gráfica del juego. Si el modo gráfico 3D no está activado, *gfxm* vale 0.

- ***just_drew*** es una variable de tipo *bool* que indica si hay que dibujar en el instante actual.

- ***quit*** es una variable de tipo *bool* que indica si hay que salir o no del juego.

- ***error*** es una variable de tipo *bool* que indica si se ha producido o no un error.

- ***el constructor*** inicializa los punteros *gsm* y *gfxm* con los parámetros que recibe, además de inicializar *just_drew*, *quit* y *error* a *false*.

La clase *ClientEventHandler* contiene un constructor, una referencia a una estructura *ClientAIState*, un objeto *random* y los prototipos de los métodos *bool handle_event(const Event &e)* y *void compute_actions()*.

El constructor inicializa la referencia a la estructura con el parámetro que recibe.

El fichero *ClientEventHandler.c* contiene la implementación de los métodos *bool handle_event(const Event &e)* y *void compute_actions()*.

El primero recibe eventos y los trata. Diferencia entre tres tipos diferentes:

- ***STOP_MSG*** indica que hay que salir del juego.

- ***READ_ERROR_MSG*** indica que se ha producido un error y por ello hay que salir del juego.

- ***VIEW_MSG*** indica que ha recibido un mensaje vista, lo que quiere decir que se ha actualizado el estado del juego. Por ello, se llama al método *void compute_actions(),*

se mandan las acciones al servidor y se realizan unas gestiones de los gráficos, a la vez que se imprimen por pantalla unas estadísticas del estado actual.

El método *void compute_actions()* (aproximadamente línea 86) es el que determina las acciones de cada una de las unidades del cliente. Por ello, es el único método que va a ser diferente al segundo cliente que vamos a implementar. En este método se concentra toda la inteligencia artificial del cliente.

Nuestro método para el primer cliente es muy sencillo, ya que únicamente mueve a todos sus objetos a un punto elegido de forma aleatoria.

Podríamos decir que el método se divide en dos partes. En la primera, obtenemos la información necesaria sobre el estado actual del juego. Dicha información es la siguiente:

- Nuestro id de jugador. Se obtiene mediante la siguiente sentencia:

```
const sint4 cid = game.get_client_player();
```

cid es la variable que va a contener el id del jugador después de ejecutar el método get_client_player mediante el objeto game. Dicho objeto es de tipo Game y hay que actualizarlo en cada iteración[31] mediante la siguiente sentencia:

```
const Game &game = state.gsm->get_game();
```

game es una referencia al propio juego, mediante la cual podemos obtener información relativa al estado del juego (mapa, objetos, etc.). Por otro lado, *state* es el atributo público de tipo estructura *ClientAIState* de la clase *ClientEventHandler*.

- Todos nuestros objetos. Se obtienen mediante la siguiente sentencia:

```
const Game::ObjCont &objs = game.get_objs(cid);
```

objs es una referencia a un vector de punteros, en el que cada uno apunta a un objeto concreto. cid es el id del jugador del que se quieren obtener todos sus objetos. Al igual que la referencia al juego (game), los objetos hay que obtenerlos

[31] Cada vez que se ejecuta el método compute_actions, es decir, cada vez que cambia el estado del juego después de que se hayan procesado las últimas instrucciones enviadas al servidor.

en cada iteración porque pueden cambiar de una iteración a otra (debido a que se hayan creado nuevos objetos, haya cambiado la información del algún objeto concreto, etc.).

- El ancho del mapa, medido en puntos. Se obtiene mediante la siguiente sentencia:

```
const sint4 points_x = map.get_width() * game.get_tile_points();
```

points_x contiene el resultado de multiplicar el número de tiles del mapa (a lo ancho) por el número de puntos/*tile*. Mediante la referencia al juego, podemos obtener cuantos puntos forman una *tile*[32]. Por otro lado, map es una referencia al mapa actual del juego que se obtiene mediante la siguiente sentencia:

```
const Map<GameTile> &map = game.get_map();
```

Por ultimo, el método *get_width* devuelve el número tiles que forman el ancho del mapa.

- El largo del mapa, medido en puntos. Se obtiene mediante la siguiente sentencia:

```
const sint4 points_y= map.get_height() * game.get_tile_points();
```

points_y contiene el resultado de multiplicar el número de tiles del mapa (a lo largo) por el número de puntos/*tile*. El método *get_height* devuelve el número *tiles* que forman el largo del mapa.

Como podemos observar, toda esta información se obtiene gracias a la referencia del juego (*game*) y como ella se tiene que obtener/actualizar en cada iteración, toda la información con la que está relacionada también se tiene que actualizar en cada una de las iteraciones.

La segunda parte, consta de un bucle *for* para cada uno de nuestros objetos. Dentro del bucle, comprobamos que el objeto actual exista, esté en juego, dentro del mapa y no esté muerto. Si es así, preguntamos si nuestro objeto está quieto o en movimiento, ya que si está en movimiento hay que dejarle que llegue a su destino. Si está quieto, calculamos unas nuevas coordenadas de

[32] Se supone que las tiles son cuadradas y que el método *get_tile_points* devuelve el número de puntos (a lo largo o a lo ancho) que forman una tile.

forma aleatoria y le asignamos la acción move. Todo esto se puede ver en el código adjunto, que está detalladamente comentado.

11.4.3. Implementación del segundo cliente

El segundo cliente está formado por los mismos ficheros que el primero, pero el fichero *waypoints_main.c* pasa a llamarse *cheser_main.c*. Además, se cambia el valor de la variable *VERSION* a *"Cheser v1.0"*.

El contenido de los ficheros es el mismo, a excepción del método *void compute_actions()* (aproximadamente línea 86) del fichero *ClientEventHandler.c*

En este segundo cliente, el objetivo es que cada uno de nuestros objetos se mueva de forma aleatoria hasta que encuentre a un objeto rival, momento en el cual pasará a seguirle a una cierta distancia.

La primera parte del método va a ser exactamente igual a la del primer cliente, la diferencia está en la segunda.

La segunda parte, consta de un bucle *for* para cada uno de nuestros objetos. Dentro del bucle comprobamos que el objeto actual exista, esté en juego, dentro del mapa y no esté muerto. Si es así, comprobamos que solo existan dos jugadores en el escenario, para posteriormente entrar en segundo bucle *for* para cada uno de los objetos del rival.

Dentro del segundo bucle, comprobamos que el objeto actual del rival exista, esté en juego, dentro del mapa y no esté muerto, ya que eso significará que hemos encontrado a un enemigo. Por ello, calculamos la distancia hasta él, y si es mayor a 600 puntos vamos hacia él, en caso contrario nos quedamos quietos.

En el caso de que no hayamos encontrado a un enemigo y estemos quietos, nos movemos de forma aleatoria. Una vez que encontremos a un enemigo, el resto nos dan igual, por lo que tendremos que salir del segundo bucle anidado.

Como resultado de esto podemos hacer tres acciones diferentes. La primera es quedarnos quietos si estamos demasiado cerca del enemigo, por lo que no asignaremos ninguna orden. La segunda es que estemos muy lejos del enemigo y tengamos que ir hacia él, por lo que

asignaremos la acción move con las coordenadas del enemigo. Y la tercera, es que no encontremos ningún enemigo y tengamos que seguir moviéndonos de forma aleatoria, en el caso de que estemos parados. Todo esto se puede ver en el código adjunto, que está detalladamente comentado.

11.5. Apéndice: Acrónimos y definiciones

Para facilitar la comprensión del presente tutorial, en este apartado se definen todos los acrónimos que aparecen en el documento. Además, se definen los conceptos que se consideren necesarios para lograr una completa comprensión.

A continuación se muestra una tabla con todos los acrónimos:

Acrónimo	Definición
DMG	(Apple) Disk Image
GLEW	The OpenGL Extension Wrangler Library
GPL	General Public License (Licencia Pública General)
GUI	Graphical User Interface (Interfaz Gráfica de usuario)
IA	Inteligencia Artificial
ORTS	Open Real-Time Strategy
OS	Operating System (Sistema operativo)
RTS	Real-Time Strategy (Estrategia en tiempo real)
SDL	Simple DirectMedia Layer
SRC	Source (Código fuente)
SVN	Subversion

Tabla 21: Definiciones de los acrónimos

Por otra parte, los conceptos más técnicos son los siguientes:

- **Cliente-Servidor (Client-Server):** Arquitectura que básicamente consiste en que un programa/ordenador (cliente) realiza peticiones a otro programa/ordenador (servidor) que le da respuesta.

- **Peer-to-peer:** Arquitectura que no tiene ni clientes ni servidores fijos, sino un conjunto de nodos que se comportan simultáneamente como clientes y como servidores respecto de los demás nodos de la red. En esta definición, nodo puede hacer referencia a un programa o a un ordenador.

- **Tile:** Su traducción literal del inglés es baldosa, azulejo, losa. Creo que el mapa del juego se divide en pequeñas zonas, llamadas *tiles* y cada una de ellas está formada por un número de puntos.

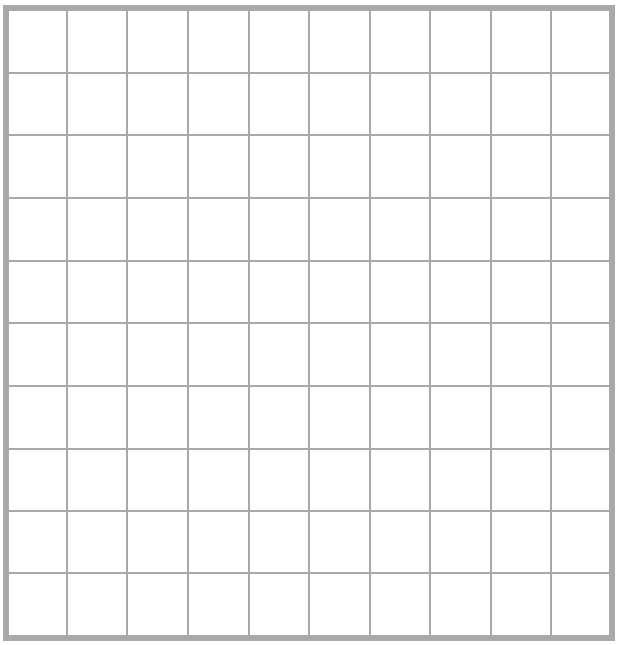

Ilustración 89: Estructura del mapa del juego

Si nos fijamos en la *ilustración 89*, el cuadro grande marcado con un borde gris sería el mapa del juego y cada uno de los cuadrados de su interior sería una *tile*.

- **Blueprint:** Es un lenguaje de *script* que se utiliza para definir los dominios. Los ficheros *blueprint* tienen la extensión *.bp* y no toda la definición del dominio tiene que estar en dichos *script*, parte puede estar en el propio código fuente de la creación del escenario (número de marines, por ejemplo), aunque no es recomendable.

11.6. ANEXO

11.6.1. Código de static void generate_game4_map (Stringstream &world)

```
const sint4 EXAMPLE_NUM_UNITS = 1;
…

void MapTool::generate_example_map(stringstream &output_world){

  MapData md;
  stringstream world;

  init_map_data(md);

  assert(md.num_players <= 2);

  REM("generate symmetric bases");

  md.bad.clear();
  md.bad.reserve(md.tile_n);
  FORS (i, md.tile_n) md.bad.push_back(0);

  Vector<StartLoc> slocs;

  // create center symmetric start locations

  FORS (i, EXAMPLE_NUM_UNITS) {

    sint4 x0, y0;

    FOREVER {
      x0 = md.rand.rand_uint4() % (md.tiles_x / 4);
      y0 = md.rand.rand_uint4() % md.tiles_y;
      if (unit_loc_ok(md, x0, y0)) break;
    }

    mark_unit_loc(md, x0, y0);

    slocs.push_back(StartLoc(0, x0, y0));
    x0 = md.tiles_x - 1 - x0;
    y0 = md.tiles_y - 1 - y0;
    slocs.push_back(StartLoc(1, x0, y0));
  }

  cout<<"Creating example map" << endl;

  generate_game_tiles(md, world, false);

  string fname = opt.bp_file;
  fname = sanitize_path(fname);
  parse_map_include(world, fname);

  world << "<OBJECTS>\n";

  FORALL (slocs, i) {
    place_unit(world, "marine", (*i).player, (*i).x, (*i).y);
  }

  output_world << world.str();
}
```