T.E.G:
Plan táctico y estratégico de la Guerra.

Luis Emiliano Sánchez[1],

[1] Estudiante de ingeniería de sistemas, UNCPBA.
lsanchez@alumnos.exa.unicen.edu.ar

Resumen. El presente trabajo se realizó como trabajo final de la cátedra Análisis y Diseño de Algoritmos 2, de la carrera Ingeniería de Sistemas de la Universidad Nacional del Centro de la Provincia de Buenos Aires (UNCPBA). Trata el desarrollo de un video juego, adaptación del clásico juego de mesa argentino *T.E.G*. La aplicación [1] destaca la forma de modelar la inteligencia artificial del ordenador, considerando básicamente la separación del proceso de toma de decisiones (o plan de acción) en sus dos componentes metodológicos: *Táctica* y *Estrategia*. Por último, se llevó a cabo un análisis comparativo de los resultados obtenidos simulando una serie de partidas entre jugadores virtuales.
La aplicación fue construida en lenguaje C++ con el IDE Code::Blocks [2], usando el framework de video juegos HGE [3].

Keywords: Inteligencia Artificial, Patrones de diseño, HGE, TEG, Algoritmos de Grafos.

1 Introducción

El TEG [4] es un clásico juego de mesa Argentino que surge como una variante muy interesante al mundialmente conocido Risk [5]. Se trata de un juego de estrategia por turnos, de 2 a 6 jugadores, que propone un conflicto bélico que tiene lugar sobre un planisferio dividido en 50 países o regiones geográficas, agrupados en 6 continentes, en donde cada jugador tiene una misión secreta que cumplir, u optar por la misión global de conquistar 30 de los 50 países. Estas misiones secretas pueden ir desde eliminar a determinado oponente o conquistar determinadas regiones y países del mundo. Para ello se debe actuar inteligentemente, deducir las intenciones de nuestros enemigos, administrar razonablemente nuestras tropas (fichas), y atacar y reagrupar en los momentos y de la manera adecuada. Todas estas consideraciones fueron tomadas en cuenta al momento de modelar la inteligencia artificial de los jugadores virtuales.

La aplicación permite cualquier combinación de rivales humanos y/o virtuales, como así también asignar distintos niveles de inteligencia a estos últimos, desde los dos aspectos de su IA: *táctica* y *estrategia*. Con táctica nos referimos a la capacidad del jugador de definir un plan de acciones a corto plazo, es decir, la unidad de tiempo que corresponde a un turno de juego. Con estrategia nos referimos a definir el plan de acciones a largo plazo, es decir, la totalidad de la partida. La relación entre los dos

conceptos es fundamental. No es posible aplicarlos en forma independiente. Sin táctica la estrategia nunca podría concretarse, ya que no encontraríamos el camino para coronar con éxito los planes que diseñamos. Sin estrategia ni lineamientos generales, la táctica no tendría objetivos claros y su aplicación sería errónea.

Es interesante mencionar la existencia de otros videojuegos clones del T.E.G, como antecedentes a esta aplicación, que proporcionan además modo multijugador en red, como el software *Tegnet* [6] y *TEG: "Tenés Empanadas Graciela"* [7].

2 Diseño e implementación del video juego

El diseño e implementación del video juego se puede analizar desde 3 aspectos o problemas de desarrollo:

- Primero, como *modelar e implementar los componentes lógicos del juego*, sus reglas y dinámica, con el fin de que estos definan una interfaz apropiada y sencilla para solucionar el segundo problema: la inteligencia artificial.
- Segundo, la *inteligencia artificial* es el principal desafío del proyecto, mayormente por la complejidad que representa la multitud de decisiones que pueden tomar los jugadores. En este sentido, se desarrolló un conjunto de clases y algoritmos organizados en capas o niveles de abstracción, con el fin de modelar los aspectos tácticos y estratégicos de la inteligencia, y que además se pueda seguir mejorando sencillamente extendiendo el código.
- Por último, la *interfaz gráfica de usuario*. El framework HGE ofrece un mecanismo de flujo de control clásico de video juegos interactivos: una etapa de inicialización de los recursos, seguida de una iteración continua de una función para la captura de eventos y actualización de estados (*update(float dt);*) y renderizado de los componentes del juego (*render();*). Por otro lado, la lógica del juego es más fácil de implementar si esta mantiene un flujo de control propio para manejar las rondas y asignar los turnos a los objetos que representan a cada jugador. Cómo combinar estos dos flujos de control, bajo un solo hilo de ejecución representa también otro desafío importante.

En base a esto, se consideró la descomposición del proyecto en estos 3 aspectos o paquetes de desarrollo: *Interfaz grafica de usuario*, *Lógica del juego* e *Inteligencia artificial*.

2.1 Interfaz gráfica de usuario

La interfaz de usuario se implementó con HGE [3], que ofrece una capa de abstracción sobre DirectX, con un completo conjunto de clases que implementan componentes gráficos y demás recursos comunes a los videojuegos 2D.

Frente a otras bibliotecas de igual propósito como SDL, Allegro, entre otras [8], HGE presenta una interfaz muy sencilla y reducida que motivó a usarla en este proyecto. Lamentablemente se descontinuó su desarrollo desde Marzo de 2008,

aunque una amplia comunidad de usuarios siguen extendiéndola de forma independiente [9].

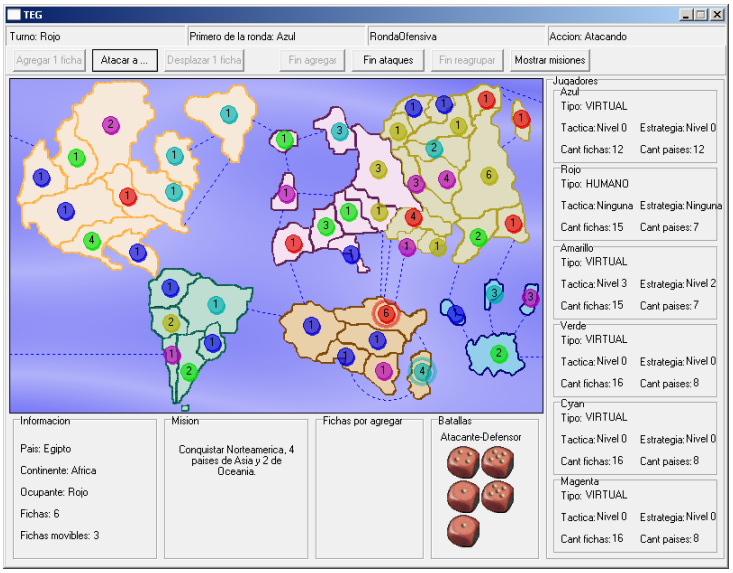

Fig. 1. Captura de pantalla de la ventana principal de la aplicación, con una partida en curso.

2.2 Lógica del juego

La lógica del juego hace referencia al conjunto de clases y objetos que representan los componentes del juego (tablero, jugadores, misiones) y manejan la dinámica y reglas del mismo (controlador del juego). Definen una interfaz sencilla e intuitiva, como así también una implementación eficiente y robusta para comunicarse con los componentes de la *interfaz grafica de usuario* y la *inteligencia artificial*.

Estos componentes (Fig. 2) son:

- *ControladorJuego*: es la clase que maneja las rondas y estados del juego, llamando a las funciones que implementan la interfaz gráfica, y a las acciones (métodos) de los jugadores. También se encarga de la administración de los demás recursos de la lógica del juego.
- *TableroTEG*: este objeto es un contenedor de *casilleros*. Se modela como un grafo no dirigido plano, cuyos nodos son los casilleros que representan países y regiones geográficas.
- *Jugador*: esta clase abstracta contiene y brinda información sobre los jugadores, implementa las *acciones elementales* que los jugadores pueden ejecutar sobre el tablero (agregar ficha, desplazar ficha y batalla) y define *acciones generales* (un grupo de métodos virtuales puros) que los jugadores concretos, ya sea virtual (controlado por el ordenador) o humano, deben ejecutar en sus turnos.
  La implementación de la clase se inspira en el patrón de diseño *Facade* (Fachada) [11], al brindar un grupo de métodos protegidos *(acciones elementales)* para proporcionar una interfaz unificada de alto nivel y controlar y facilitar el acceso

de sus subclases al tablero y sus casilleros.
De esta, extienden las clases *JugadorHumano* y *JugadorVirtual*. Cada una implementa los métodos virtuales puros que corresponden a las acciones generales de los jugadores: *TomarMision*, *AgregarFichas*, *Atacar* y *Reagrupar*. La primer clase (*JugadorHumano*) lo hace interactuando con la interfaz gráfica para capturar los eventos del usuario, mientras la segunda, a través de sus objetos *Táctica* y *Estrategia*, como se explica posteriormente.

- *Misión*: es una simple estructura que modela los 3 tipos de misiones disponibles: conquista de regiones, eliminación de jugador, o conquista de N casilleros (misión global).

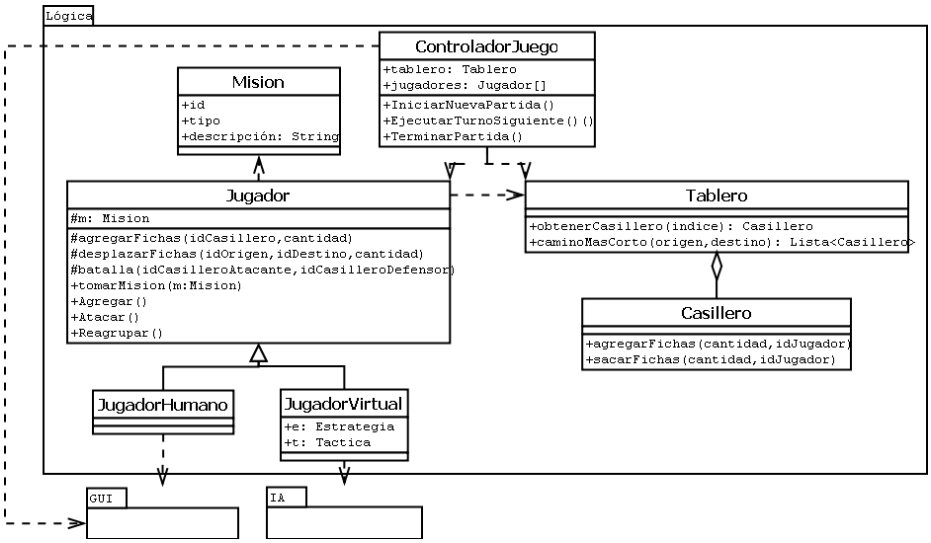

Fig. 2. Diagrama de clases simplificado correspondiente al paquete de la lógica del juego.

2.3 Inteligencia artificial

Este paquete involucra los algoritmos y clases que modelan, de una manera ordenada y sencilla, la toma de decisiones inteligentes en la búsqueda de un objetivo final. Esta se encuentra organizada en niveles de abstracción con el fin de simular, de cierta forma, a un ser humano en su proceso de toma de decisiones.
Para la IA, se consideran las siguientes cuestiones:

- Definir de la manera más sencilla y flexible posible las interfaces de las clases que representan la lógica del juego, para que la implementación de la IA se simplifique.
- Organizar la IA en niveles o capas de abstracción, con sus límites y responsabilidades bien definidos, y de esta manera tener un mejor entendimiento del todo y sus partes y una mayor modificabilidad del código.
- Desarrollar las clases y algoritmos de IA con la intención de que resulten fácil de extender.

Así, la IA se descompone en 3 niveles o capas: *estrategias, tácticas y algoritmos auxiliares***.** Un usuario de la aplicación puede seleccionar, para cada jugador virtual, 1 de 3 niveles de capacidad estratégica y 1 de 4 niveles de capacidad táctica. Estos niveles representan mejoras sucesivas de la inteligencia artificial.

2.3.1 Estrategias

Se trata del aspecto de más alto nivel. Su función es definir un plan de acción a largo plazo (plan estratégico) con el objetivo final de cumplir la misión asignada al jugador. Una buena estrategia debe considerar, además de la misión del jugador, su situación actual y la situación de sus enemigos.

El plan estratégico se modela como un *grafo* de *Objetivos*. Para alcanzar la misión secreta, los jugadores virtuales se plantean una secuencia de objetivos que pretenden ir cumpliendo parcialmente en cada turno.

En su turno, a la estrategia de un jugador virtual le corresponde realizar dos tareas: actualizar su plan estratégico (evaluando el estado actual de la partida) y pasar a los algoritmos tácticos una lista con los objetivos que se deberían cumplir en el turno actual para progresar con el plan estratégico.

Los *objetivos* que forman el plan estratégico se modelan con una serie de atributos que incluyen: el tipo de objetivo (*defender*, *conquistar*, *intervenir* o *expandir*), el conjunto de los casilleros asociados, la importancia (*obligatorio* u *opcional*), y el estado (*cumplido*, *no cumplido*, o *perdido*).

La clase abstracta *Estrategia* define la interfaz que debe tener toda Estrategia concreta. Sus dos métodos principales son virtuales puros: *IniciarPlan* y *ActualizarPlan*. La idea de estos métodos es que sus subclases analicen aquí la situación actual de la partida, para elaborar su plan estratégico. El resto de los métodos se inspiran en el patrón de diseño *Template Method* [11]: los métodos *Agregar*, *Atacar* y *Reagrupar* son métodos plantilla, que definen una estructura algorítmica que delega su implementación a las subclases de la misma. Estos métodos son invocados por los jugadores virtuales en sus turnos.

La clase *EstrategiaNivel1* extiende a Estrategia incorporando un plan de acción simple. *EstrategiaNivel2* extiende al anterior incorporando nuevas características y mejoras.

El primer nivel construye un plan estratégico simple pero suficiente para que el jugador pueda ganar. Este consiste en alcanzar su misión directamente, sin considerar su situación (potencial, dispersión de fichas, etc) ni la de sus oponentes (si ocupan un continente o no, etc). Simplemente descompone su misión en un conjunto de objetivos más simples y los incorpora al plan estratégico. No agrega ningún objetivo indirecto.

El segundo nivel de estrategia, introduce la posibilidad de seguir la misión global en lugar de la misión secreta del jugador. De esta forma, en cada turno, el jugador puede reestructurar su plan estratégico en busca de una u otra misión, haciendo uso de la función heurística *Costo Para Cumplir Misión.*

En caso de que la estrategia siga la misión global, el plan estratégico se arma de la siguiente manera: por cada continente que ocupa completamente el jugador, introduce un objetivo *defensivo* para estos países. También introduce un objetivo de *expansión*

sobre el mayor *"grupo limítrofe"* de países que tenga. Se llama grupo limítrofe a una región del tablero cuyos casilleros pertenecen al mismo jugador y son adyacentes entre si, directa o indirectamente. También introduce un objetivo de *intervención*, seguido de *conquista,* sobre el continente que considere más fácil de dominar, ya que esto implica una ventaja interesante por el ingreso de fichas extras.

Estos planes básicos pueden extenderse en otros niveles. Por ejemplo, se puede considerar evitar zonas conflictivas y ser más conservador. También evitando que un jugador enemigo alcance su objetivo, suponiendo su misión mediante alguna función heurística. Y así también otras tantas posibles extensiones.

2.3.2 Tácticas

Las tácticas son clases que encapsulan los algoritmos que se encargan de interpretar e intentar cumplir los objetivos que determina la estrategia de un jugador virtual. La clase abstracta *Táctica*, define las interfaces de las 3 acciones generales que todo jugador debe realizar en su turno: agregar fichas, atacar, y reagrupar. A partir de esta, extienden una serie de tácticas concretas que mejoran las capacidades de cada una de estas funciones.

La táctica de nivel 0 no es más que un grupo de funciones sin inteligencia. Solo se limitan a agregar fichas y atacar de forma aleatoria. Fue implementada para avaluar los demás niveles tácticos.

La táctica de nivel 1 mejora el aspecto de agregar fichas. El jugador con este nivel toma en cuenta los objetivos del plan estratégico para distribuir sus fichas en el tablero, considerando además la distribución de fichas de su rival. Esta clase construye como un grafo el *plan táctico* utilizado por sus subclases. Este plan define la secuencia de ataques que el jugador debería seguir en su turno de ataques.

La táctica de nivel 2 extiende el nivel anterior mejorando el ataque. El jugador con este nivel toma en cuenta el plan táctico para realizar sus ataques, y considera las posibilidades de victoria en las batallas determinada por los dados, mediante una función heurística basada en probabilidades.

Por último, la táctica de nivel 3 extiende el nivel anterior mejorando el reagrupamiento de fichas. Este jugador reagrupa fichas desde sus países "*aislados*" (casilleros sin adyacentes enemigos) hacia las zonas más vulnerables a ataques.

Como con las estrategias, podrían plantearse nuevos niveles de tácticas que mejoren las características de los niveles anteriormente mencionados.

2.3.3 Algoritmos auxiliares

Comprende un conjunto de funciones heurísticas y clases auxiliares para facilitar la implementación de los niveles superiores de IA. Entre ellas se incluye: operaciones sobre conjuntos y arreglos, iteradores para recorrer secuencialmente la estructura del tablero con distintos criterios, funciones heurísticas, y algoritmos de grafos, entre ellos, una adaptación del algoritmo Floyd [10] para obtener el camino más corto entre cualquier par de casilleros del tablero. También se implementó una clase auxiliar *Grafo* y sus operadores, para modelar los planes estratégicos y tácticos.

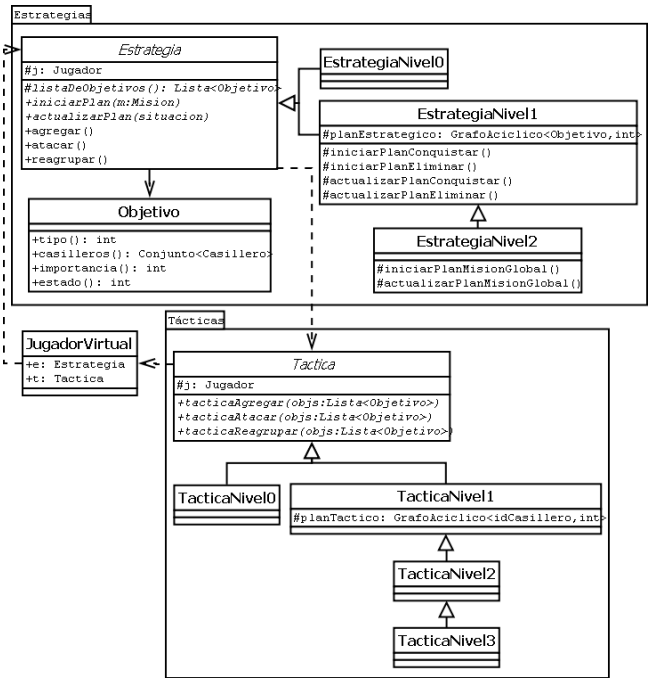

Fig. 4. Diagrama de clases simplificado de los paquetes de componentes de la IA.

3 Pruebas y resultados

Para realizar un análisis comparativo de los distintos niveles de inteligencia, se implementó una aplicación por consola que permite simular una determinada configuración de partida, sin los retardos de tiempo que impone la interfaz gráfica. Dado que el factor azar condiciona considerablemente los resultados, la aplicación ejecuta *n* veces la partida con los mismos parámetros, hasta alcanzar la convergencia de las probabilidades de victoria de cada jugador. Se puede definir el número de jugadores, los niveles tácticos y estratégicos para cada uno y el valor ε de precisión o error de convergencia. Como resultado arroja las probabilidades de victoria de cada jugador y el valor *n* de partidas simuladas para alcanzar la convergencia.

Tabla 1. Algunos de los resultados obtenidos en la simulación de partidas entre 2 jugadores virtuales, con ε = 0.0005, bajo distintas configuraciones de sus niveles de inteligencia. Para eliminar la ventaja defensiva del jugador que arranca último, se alterna el orden de los jugadores en la sucesión de partidas simuladas.

Jugador 1			Jugador 2			*N* (partidas simuladas)
Estrategia	Táctica	Prob. de victoria	Estrategia	Táctica	Prob. de victoria	
Nivel 2	Nivel 3	51,833 %	Nivel 1	Nivel 3	48,167 %	995
Nivel 2	Nivel 1	51,120 %	Nivel 1	Nivel 1	48,880 %	982
Nivel 0	Nivel 3	84,112 %	Nivel 0	Nivel 0	15,888 %	321
Nivel 0	Nivel 3	58,696 %	Nivel 0	Nivel 2	41,304 %	346
Nivel 2	Nivel 0	49,910 %	Nivel 0	Nivel 0	50,090 %	990

Los resultados obtenidos revelan algunos datos interesantes:

- La diferencia entre las estrategias de nivel 1 y 2 se acentúa cuando disminuye la cantidad de jugadores, resultando la de nivel 2 claramente superior. Esto se debe a que seguir la misión global puede ser la estrategia más razonable cuando hay pocos enemigos (ver filas 1 y 2 de la tabla 1).
- Sin estrategia (nivel 0), las tácticas de niveles superiores (niveles 2 y 3) tornan a los jugadores más conservadores. La diferencia se nota claramente contra jugadores con tácticas más arriesgadas (niveles 0 y 1) (ver filas 3 y 4).
- Por último, se puede observar como la mejor de las estrategias no se puede aprovechar cuando no hay táctica que la respalde (ver fila 5).

4 Conclusiones

A modo de conclusión, se resalta la utilidad de una metodología *top-down* en la descomposición del desarrollo de la aplicación en porciones más chicas y tratables. Esta descomposición se llevó más allá de la simple división en interfaz gráfica, lógica e IA. La división de la IA en niveles o capas de abstracción representa una manera ingeniosa y apropiada de entender y modelar la solución a este problema, además de las ventajas extras que se obtienen en cuanto a modificabilidad y extensibilidad de las características. También el uso de patrones de diseño contribuye a mejorar estos atributos de calidad.

A todo esto, siempre se encuentran características que pueden mejorarse, y surgen ideas para hacerlo. Quizás sea esto uno de los atractivos que hacen al desarrollo de video juegos más que una industria, sino también un hobby para muchos.

Referencias

1. Acceso a los binarios y código fuente de la aplicación, http://code.google.com/p/teg-plan-tactico-y-estrategico-de-la-guerra/
2. Code::Blocks IDE website, http://www.codeblocks.org/
3. HGE version 1.81 (Haaf's Game Engine), http://hge.relishgames.com/
4. T.E.G website, http://www.yetem.com/teg.html
5. Risk website, http://www.hasbro.com/risk/
6. TegNet, http://www.tegnet.com.ar
7. Tenes Empanadas Graciela, http://sourceforge.net/projects/teg/
8. Frameworks y game engines, http://content.gpwiki.org/index.php/Game_Engines
9. Comunidad HGE: http://relishgames.com/forum/
10. Algoritmo de Floyd, http://es.wikipedia.org/wiki/Algoritmo_de_Floyd-Warshall
11. Design Patterns. Elements of Reusable Object-Oriented Software - Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides - Addison Wesley (GoF- Gang of Four).