



ESCUELA SUPERIOR DE INGENIERÍA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE GESTIÓN

Dominous: simulador libre de dominó

Ignacio Palomo Duarte

19 de septiembre de 2011



ESCUELA SUPERIOR DE INGENIERÍA

INGENIERO TÉCNICO EN INFORMÁTICA DE GESTIÓN

DOMINOUS: SIMULADOR LIBRE DE DOMINÓ

- Departamento: Lenguajes y Sistemas Informáticos
- Autor del proyecto: Ignacio Palomo Duarte
- Directores del proyecto: Inmaculada Medina Bulo y Manuel Palomo Duarte

Cádiz, 19 de septiembre de 2011

Fdo: Ignacio Palomo Duarte

Agradecimientos

Me gustaria agradecer y/o dedicar este texto a ...

Licencia

Este documento ha sido liberado bajo Licencia GFDL 1.3 (GNU Free Documentation License). Se incluyen los términos de la licencia en inglés al final del mismo.

Copyright (c) 2011 Ignacio Palomo Duarte.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Notación y formato

Aquí incluiremos los aspectos relevantes a la notación y el formato a lo largo del documento. Para simplificar podemos generar comandos nuevos que nos ayuden a ello, ver `comandos.sty` para más información.

Cuando nos refiramos a un programa en concreto, utilizaremos la notación: *emacs*.

Cuando nos refiramos a un comando, o función de un lenguaje, usaremos la notación: `quicksort`.

Extractos de ficheros con texto plano o código aparecerán

Listado: Ejemplo de listado

- 1 dentro de un recuadro como este, con numeracion a la izquierda
- 2 Y sin acentos, por compatibilidad con cualquier codificacion

Índice general

1. Introducción	1
1.1. Objetivo	1
1.2. Estructura de la memoria	2
2. Conceptos básicos	3
2.1. El dominó	3
2.1.1. Historia del dominó	3
2.1.2. Reglas básicas	7
2.1.3. El dominó es un juego de señores	8
2.1.4. Juego por parejas	9
2.1.5. Técnicas avanzadas	10
2.2. Inteligencia artificial	10
2.2.1. Sistemas expertos	10
2.2.2. Sistemas expertos basados en reglas	11
3. Planificación	15
3.1. Organización de las partes básicas de la aplicación	16
4. Análisis	19
4.1. Toma de requisitos	19
4.1.1. Requisitos de interfaces externas	19
4.1.2. Requisitos funcionales	20
4.1.3. Requisitos de rendimiento	20
4.1.4. Restricciones de diseño	21
4.1.5. Requisitos del sistema software	21
4.2. Modelo de casos de uso	21
4.2.1. Diagrama de casos de uso	21
4.2.2. Descripción de los casos de uso	21
4.3. Modelo conceptual de datos	26
4.3.1. Diagrama de clases conceptuales	26
4.3.2. Modelo de comportamiento del sistema	27
5. Diseño	31
5.1. Introducción	31
5.2. Definición de los requisitos del sistema	31
5.3. Herramientas utilizadas	32
5.3.1. Librería gráfica	32
5.3.2. Lenguaje de programación	32
5.3.3. Diseño y estilo visual de interfaces	34

5.3.4. Documentación del código	34
5.3.5. Sistema de control de versiones	35
5.4. Interfaz gráfica	35
5.4.1. Diagrama de interacción entre interfaces gráficas	36
5.5. Diagrama Entidad – Relación	36
5.6. Diagrama de clases de diseño	36
5.7. Análisis de las principales clases de la aplicación	38
5.7.1. Clase dominous	38
5.7.2. Clase dominoes_game	38
5.8. Sistemas expertos	40
5.8.1. Estrategia según la posición	41
5.8.2. Reglas de obligado cumplimiento	41
5.8.3. Pesos de las diferentes reglas	41
5.8.4. Aleatoriedad	41
5.8.5. Generación de errores - sistemas imperfectos	42
5.8.6. Diferentes jugadores	42
5.8.7. Modelado	42
6. Implementación	45
6.1. Implementación	45
6.1.1. Entorno gráfico	45
6.1.2. Interfaz de sonido	47
6.1.3. Configuración de la aplicación	48
6.1.4. Inteligencia Artificial	50
6.1.5. Música y efectos	53
7. Pruebas	55
7.1. Pruebas	55
7.1.1. Pruebas unitarias	56
7.1.2. Pruebas de integración	57
7.1.3. Pruebas de Jugabilidad, usabilidad y experiencia de usuario	58
8. Conclusiones	61
8.1. Resultados obtenidos	61
8.1.1. Primer simulador de dominó libre	61
8.1.2. Aplicación de lo aprendido en la carrera	61
8.1.3. Aprendizaje del juego de dominó	62
8.1.4. Primer proyecto que realizo de manera íntegra	62
8.2. Trabajos futuros	63
8.2.1. Juego en red	64
8.2.2. Servidor web para compartir sistemas expertos	64
8.2.3. Analizador de partidas con aprendizaje automático o semiautomático	64
8.2.4. Migración de la aplicación a otros sistemas	65
A. Manual de instalación	67
A.1. Obtener Dominous	67
A.2. Instalación y ejecución en GNU/Linux	67
A.3. Instalación y ejecución en Windows	67

B. Manual de usuario	69
B.1. Ejecución	69
B.2. Menú principal	69
C. Difusión	71
C.1. Difusión	71
Bibliografia y referencias	73
GNU Free Documentation License	75
1. APPLICABILITY AND DEFINITIONS	75
2. VERBATIM COPYING	76
3. COPYING IN QUANTITY	76
4. MODIFICATIONS	77
5. COMBINING DOCUMENTS	78
6. COLLECTIONS OF DOCUMENTS	79
7. AGGREGATION WITH INDEPENDENT WORKS	79
8. TRANSLATION	79
9. TERMINATION	79
10. FUTURE REVISIONS OF THIS LICENSE	80
11. RELICENSING	80
ADDENDUM: How to use this License for your documents	80

Índice de figuras

2.1. Martin Gardner, divulgador científico y filósofo de la ciencia estadounidense.	4
2.2. Cartas de dominó chino “ Double Happiness ” — los símbolos representan diferentes circunstancias y bendiciones de la vida	5
2.3. Fichas de dominó europeo	7
2.4. Ejemplo de partida terminada en cierre. Se han colocado todas las blancas, por lo que no es posible que algún jugador continúe la partida	8
2.5. Esquema conceptual de un sistema experto	10
2.6. Desarrollo de un sistema experto	11
2.7. Arquitectura general de un sistema experto basado en reglas.	12
3.1. Diagrama de Gantt	17
4.1. Diagrama de casos de uso del sistema	22
4.2. Diagrama de clases para los requisitos obtenidos	27
4.3. Diagrama de secuencia para el caso de uso Partida simple	28
4.4. Diagrama de secuencia para el caso de uso del modo laboratorio	28
5.1. Logotipo de la librería Simple DirectMedia Layer	32
5.2. Logotipo de Python - Copyright Python Software Foundation	33
5.3. Diferentes elementos utilizados en la interfaz	35
5.4. Varias pantallas con la interfaz de Dominous	36
5.6. Diagrama de diseño del sistema	37
5.5. Diagrama de interacción entre interfaces	44
5.7. Diferentes imágenes que identifican al conjunto de jugadores de Dominous	44
6.1. Diferentes posiciones en las que se puede colocar una ficha en el tablero	48
6.2. La música se obtuvo de Jamendo, un servicio de música libre	54
7.1. Esquema de relaciones entre módulos para las pruebas de integración	57
7.2. Superficie sobre la cual el usuario puede dejar la ficha satisfactoriamente	58
7.3. Como vemos, la superficie en la cual el usuario puede realizar click de forma satisfactoria es más amplia que el simple texto que incluye	59
8.1. Dominous ejecutándose en un entorno GNU/Linux, distribución Ubuntu 10.04	65
8.2. Dominous ejecutándose en un entorno Microsoft Windows 7	66
A.1. Página oficial de Dominous	68
C.1. Finalista en la Categoría Proyectos Libres de Ocio de la Universidad de Cádiz 2009-2010	72
C.2. Accésit al Mejor Proyecto Libre de Innovación en la Universidad de Cádiz 2010-2011 .	72

Índice de tablas

Índice de listados

2.1. Regla en pseudocódigo	13
2.2. Regla en lenguaje natural	13
6.1. Código Python de cambio de sección	45
6.2. Autómata finito de estados	46
6.3. FicheroINI de ejemplo	49
6.4. Código Python de lectura de ficheroINI	49
6.5. Información global de la partida	50
6.6. Información de la mano actual	51
6.7. Estructura básica de regla	51
6.8. Regla para colocar ficha doble	51
6.9. Base de conocimiento simple	52
6.10. Base de conocimiento compleja	53
A.1. Instalación de dependencias	67

Capítulo 1

Introducción

El Proyecto Fin de Carrera es el culmen a un largo período de aprendizaje, exámenes, vivencias y experiencias, y por estas razones la elección de una temática para el proyecto es compleja, ya que tenemos diferentes necesidades, limitaciones y deseos:

1. Por una parte el proyecto es una facción más de nuestros estudios universitarios, que debemos solventar con éxito, y esta circunstancia nos puede llevar a buscar un proyecto más recortado o limitado en cuanto a requerimientos de tiempo y conocimiento.
2. Pero por otra parte nuestra faceta de ingenieros nos impulsa a aprender, a enfrentarnos con nuevos problemas y dificultades, a atacar ejercicios mentales duros e interesantes para hacer sudar nuestra mente.

1.1. Objetivo

Después de tantear varios proyectos que tenía en mente, mis tutores me presentaron la posibilidad de embarcarme en el desarrollo de un simulador de dominó. Al principio tomé la idea un poco en broma, ya que la temática en principio puede parecer poco tecnológica, demasiado localizada o con escaso atractivo, pero una vez analizado, el proyecto tenía todo lo que le podía pedir:

1. El apartado de Inteligencia Artificial es muy complejo, con lo cual se puede abordar de diferentes maneras. Es un problema de elevada complejidad computacional si intentamos resolverlo mediante simples árboles de decisión: como el juego se desarrolla dentro de un marco de conocimiento limitado (no conocemos las fichas de los demás jugadores) se produce una explosión combinatoria que nos obliga a buscar otros métodos y herramientas, como técnicas de sistemas expertos. Esta búsqueda de nuevas técnicas para la resolución de un problema concreto es la base misma de la Ingeniería Informática, y es un clarísimo ejemplo de Proyecto Fin de Carrera.
2. Por otro lado el desarrollo de videojuegos relaciona multitud de aspectos que resultan interesantes a la hora de ser abordados, como pueden ser:
 - Programación gráfica, un asunto complejo, que no se suele abordar dentro del plan de estudios de una Ingeniería Técnica en Informática de Gestión, y que presenta un gran número de dificultades, como resoluciones de pantalla, velocidad y optimización del sistema, diferencias sustanciales entre sistemas operativos, tratamiento de excepciones, entre otros.
 - Diseño de interfaces, haciendo que la aplicación sea fácil de usar, divertida, sencilla y atractiva para el usuario final, y controlando diferentes opciones y dispositivos de entrada.
 - Sistema de audio, ya que nuestra aplicación debe sincronizar actividad gráfica y ejecución de música y efectos de sonido.

- Aspecto visual, manteniendo una coherencia en cuanto a diseño gráfico de todas y cada una de las páginas, secciones y menús de toda la aplicación, guardando una uniformidad y buscando que la belleza de la aplicación se apoye en requerimientos orientados al usuario.
3. Por último, tras una concienzuda búsqueda en la red, no se ha encontrado ningún simulador de dominó de código abierto, lo que creemos que es esencial para difundir el conocimiento sobre este juego.

1.2. Estructura de la memoria

Esta memoria se estructurará de la siguiente forma:

1. Introducción
2. Conceptos básicos
3. Planificación
4. Análisis
5. Diseño
6. Implementación
7. Pruebas
8. Conclusiones
9. Manual de instalación
10. Manual de usuario
11. Difusión
12. Bibliografía y referencias

Capítulo 2

Conceptos básicos

En este capítulo se comentarán las bases del proyecto: fundamentos del dominó y diseño de sistemas expertos.

2.1. El dominó

A continuación se detalla una breve historia del juego del dominó y similares y después se comentan las estrategias fundamentales para jugar el dominó internacional (el que normalmente suele jugarse en España e Iberoamérica).

2.1.1. Historia del dominó

El dominó, deporte y pasatiempo a un tiempo, es un juego que ha ganado adeptos a lo largo de la historia. Según explica **Miguel Lugo** en su libro Dominó Competitivo [Lug08] éste “es entretenido y fácil de aprender. Ya desde pequeños comenzamos a jugar al dominó con frutas o con animales en lugar de hacerlo con puntos”. Además, éste ha ganado popularidad durante los últimos años hasta el punto de televisar torneos en Europa, Norteamérica y Latinoamérica.

“En 2001 la Federación Internacional de Dominó (con sede en Barcelona, España) celebró el primer Congreso Internacional. El año siguiente se llevó a cabo el primer Campeonato del Mundo de Dominó en La Habana (Cuba). El ‘Mundial’ continúa celebrándose anualmente a partir de este punto, en España, México, Venezuela y EEUU”(Lugo, 2008: 1).

Así pues Lugo señala que el dominó nunca antes ha estado tan reconocido en todo el mundo como ahora, que incluso se puede jugar por Internet.

La página web Dominó en Línea [Sch] recoge que en la actualidad el dominó se juegan en todo el mundo aunque resalta que es “especialmente popular en América Latina, donde los dominós se consideran como el juego nacional de numerosos países del Caribe”. Así, también menciona los torneos anuales y los clubes locales de dominó.

El origen del dominó parece ser muy antiguo, al menos en lo que se refiere a juegos similares y quizá pretérritos al actual” (González Sanz, 2010:22)

Cuenta **González Sanz** en su libro El arte del dominó: teoría y práctica [GS00] que algunos historiadores creen que puede tener origen chino, ya que éstos jugaban a un juego parecido con impresiones en piedra. Este pudo llegar a Europa a través de mercaderes y viajeros, entre los que se cita al célebre Marco Polo, los cuales y fruto de los intercambios culturales de la época, trajeron el dominó a este lado

del mundo, más en concreto a la Península Itálica, primer lugar de Europa donde se ha datado la práctica de este juego.

Benito Ruipérez [RM90] también señala que se trata de un juego muy antiguo, de unos 400 años de historia, aunque dice que se desconoce su origen y etimología (1990:7). Por otro lado, este autor comenta que llegó a Italia desde China en el siglo XVIII, sin embargo, relata que no está probado. “Versiones coincidentes aseguran que el dominó entró en Europa a través de Italia (con lo que también se puede atribuir su invención a los italianos, al menos el sistema de juego europeo)”, recoge Ruipérez.

Los italianos lo pusieron de moda introduciéndolo en España y Francia a mediados del siglo XVIII, llegando más tarde a popularizarse de modo extraordinario. A finales del mismo siglo apareció en Inglaterra, donde fue calificado de ‘juego infantil’, según Ruipérez “nada más lejos de la realidad”. **Joseph Strutt** publicó un libro Deportes y Pasatiempos en 1801, en el que, entre otros disparates, demostró un gran desconocimiento del juego. Así, escribió: “El dominó no tiene mayor interés que el ponerlo en conocimiento de las personas mayores de este país” (Ruipérez, 1990:7).

Precisamente **Martin Gardner**, experto en juegos, explica que en la literatura occidental no hay referencias a este juego hasta mediados del siglo XVIII, en que empezaron a jugarse en Italia y Francia las primeras partidas. Desde ahí, el juego se extendió al resto del continente, y más tarde, a Inglaterra y América. En Occidente, la colección normal de piezas de dominó ha consistido siempre en 28 teselas o losetas formadas por dos cuadrados adyacentes, que contienen todos los posibles pares de dígitos, de 0 hasta 6.

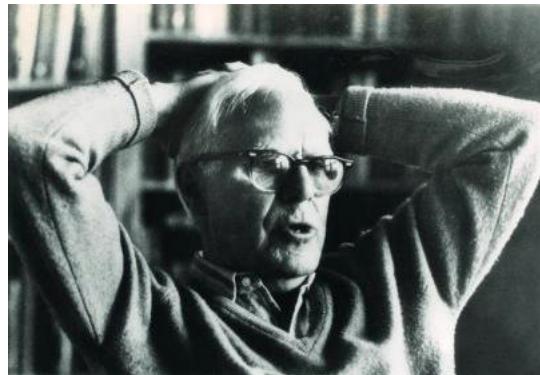


Figura 2.1: Martin Gardner, divulgador científico y filósofo de la ciencia estadounidense.

Respecto al juego chino, Gardner, que colaboró más de 20 años en la sección “Juegos Matemáticos” de la revista **Scientific American**, comenta en su libro Circo Matemático que en los dominós chinos, llamado kwat p’ai, no existen piezas con caras en blanco. Éstos contienen todas las combinaciones por pares desde el (1-1) hasta el (6-6), donde tres de los seis puntos de cara son también rojos. Los dominós coreanos son idénticos, con la única particularidad de que en el as, el punto es mayor que en las demás piezas. En los dominós chinos, cada pieza tiene un nombre pintoresco: el (6-6) es el “cielo”; el (1-1) es “la tierra”, el (5-5) es la “flor del ciruelo”, el (6-5), “la cabeza de tigre”, etc. Los nombres de las piezas son iguales a los que reciben los 21 resultados posibles del lanzamiento de un par de dados.

Ruipérez añade que el invento se les achaca a los chinos “pero no ha de ser muy fiable, ya que el dominó de chinos y coreanos es muy distinto al que se practica en Europa” (1990:7). Así lo describe en su Libro del dominó:

“El dominó oriental consta de 21 fichas, que representan las permutaciones matemáticas resultantes de tirar dos dados (cada mitad de una ficha vale por un lado), el ‘uno’ y el ‘cuatro’ son rojos; además, en los dados chinos intervienen 11 fichas repetidas, con lo que suman un total de 32 fichas el conjunto de, y para, este juego. Las fichas repetidas se llaman ‘civiles’ y las otras ‘militares’, distinción importante para ciertos juegos de dominó en China y Corea”.

González Sanz (2010:22) señala que los dominós chinos “suelen ser en la actualidad de cartón, en vez de la madera, marfil, pasta, o ébano que es lo habitual en los occidentales, y se manejan como naipes”. Y al igual que ocurre en Europa y América, con estas piezas se realizan numerosos juegos. Respecto a los distintos juegos de dominó chinos y coreanos, este autor nos remite a Games of the Orient de Stewart Culin, obra de 1895 reimprresa en 1958 por Charles Tuttle como la mejor referencia. Asimismo apunta que no existe un dominó propio en Japón – frente al resto de países asiáticos – y dice que en este país se juega con el sistema occidental.



Figura 2.2: Cartas de dominó chino **“Double Happiness”** — los símbolos representan diferentes circunstancias y bendiciones de la vida

Sin embargo, y aunque por lo que señalan algunos autores el origen asiático del dominó es el más extendido, también existen otras versiones que atribuyen el invento del juego a árabes o egipcios, “sosteniendo que no hay pruebas para relacionar claramente el dominó europeo con el chino, pudiendo ser dos invenciones independientes y separadas en el tiempo” (2010:23). González también cuenta que se conocen otras versiones del juego como la esquimal o la coreana, con distinto número de fichas y palos al clásico.

Por otro lado, otros autores (desde la página Dominó en Línea [Sch]) apuntan que el dominó de origen más antiguo se habría encontrado en la tumba de Touthankamon en Egipto. Y que los dominós nacieron de la derivación del juego de dados indio, conocido en Europa bajo el dado a seis-cara, los chinos modificaron este dado en parte plana reversible representando puntos, de 1 en 6 puntos. En Europa sería apareció una cara suplementaria, el blanco.

En el apartado de curiosidades que se apunta en esta web podemos destacar que la palabra “dominó” sería a causa de la semejanza entre las partes de las fichas del juego y la ropa de las religiosas de las Dominicas (blanco cubierto de un cabo negro). Sin embargo, el autor Ruipérez en el Libro del Dominó apunta otros posibles orígenes al término, algunos muy parecidos al ya apuntado. De hecho, existe una

variante del juego que se juega con fichas de tres caras y se llama *Trimino*, haciendo un juego de do(s)-mino por tri-mino [Boa]

En este sentido, señala este autor que su nombre se debe, según unos, al revestimiento negro que llevan sus fichas por el reverso (la espalda), como si fueran cubiertas por un dominó (capuchón que usaban en el coro, durante el invierno, los monjes). Y según otros, a que tal juego, por su sencillez, “prueba inequívoca de que no se conocían las técnicas que se usan hoy en día, o de alta escuela, que son las que se explican en este libro”, estuvo muy en uso en los conventos, y cuando uno de los jugadores ganaba una partida decía: ‘Benedicamus Dómino’. Una tercera versión para explicar el nombre afirma que por aquel entonces alguien ya detectó que para ‘manejar’ bien estas fichas tenía que tener ‘dominio’ de sí mismo (control de lo jugado y de lo pendiente por jugar) en cada mano, y podría decir, cada vez que querían jugar con estas fichas, ‘vamos a practicar unas manos al juego del dominio’, habiendo degenerado o perfeccionado el dicho hasta quedar en el juego del dominó (Ruipérez, 1990:9).

En cuanto a la documentación escrita de este juego, este mismo autor, Ruipérez, señala que el primer libro que hace referencia al juego del dominó data del año 1786, editado en Amsterdam, según consta en la Biblioteca Nacional de Bruselas. En total, este autor señala que hay más de un centenar de libros hasta la fecha (1984), según consta también en distintas bibliotecas nacionales de Europa y Latinoamérica. Aunque para este autor no han tenido suficiente éxito porque no se profundiza en el tema sino que se relatan someramente las reglas básicas del mismo. En una consulta en la Biblioteca Nacional aparece como primer libro en español sobre el tema **Tratado del Juego del Domino, sus Reglas, Combinaciones y Preceptos para ser un buen jugador** de Martínez [Mar72], publicado en 1872.

En 1982 y después en 1984, más reformado y actualizado, aparece en Barcelona el libro **ABC del dominó**, de J.M. Vilabella, que denota un conocimiento más amplio del tema comparado con lo que se había escrito hasta ese momento, dando explicaciones más amplias de cómo se desarrollan las jugadas, “aunque muy pocas y con muy pocas variantes”, (Ruipérez, 1990:8)

Tipología de dominós

Como hemos visto, a lo largo y ancho del mundo existen diferentes tipos de dominó de los que no siempre los autores coinciden con un solo origen. Aunque para catalogar tales juegos como dominó sí deben tener una serie de características en común.

Generalizando el concepto de dominó, podríamos decir que es un juego cuyas fichas se encuentran divididas en dos partes, las cuales señalan mediante incisiones o muescas, un número concreto entre los posibles palos o números admitidos. Estas fichas contendrán en su totalidad todas las combinaciones posibles de estos palos, comenzando por la ausencia de puntos o palo de blancos (González Sanz, 2010:24).

La Real Academia de la Lengua Española en su primera acepción define este juego como aquel que “se hace con 28 fichas rectangulares divididas en dos cuadrados, cada uno de los cuales lleva marcados de uno a seis puntos, o no lleva ninguno. Cada jugador pone por turno una ficha que tenga número igual en uno de sus cuadrados al de cualquiera de los dos que están en los extremos de la línea de las ya jugadas, y gana quien primero coloca todas las suyas o quien se quede con menos puntos, si se cierra el juego”. De este modo, la RAE acota algo más el término acercándose a lo que conocemos como dominó occidental.

Según la página Dominó en Línea [Sch] “los dominós son de simples bloques de construcción que pueden armarse de innombrables maneras con el fin de crear una gran variedad de juegos. Es un juego que



Figura 2.3: Fichas de dominó europeo

exige mucha capacidad y de estrategia”.

Ruipérez define el domino europeo en “un conjunto de veintiocho fichas, por lo general negras y totalmente lisas por un lado, el reverso o la espalda, y por el otro lado, el anverso o la cara, divididas en dos mitades con fondo blanco y señaladas con agujeros o pinitos negros; en el centro de la cara llevan un tornillito con cabeza redondeada, que es el que apoya en la mesa y facilita el movimiento de las fichas cuando éstas han de ser movidas (barajadas), para que los participantes cojan sus fichas e inicien la jugada o la mano correspondiente. Tiene siete fichas denominadas dobles, por en sus medias partes de la cara la misma cantidad de pinitos, a excepción de la doble blanca, que no lleva ninguno”.

2.1.2. Reglas básicas

Aunque las reglas del dominó son sencillas y conocidas por un gran conjunto de lectores, daremos una serie de pinceladas rápidas sobre las reglas más básicas, siempre recordando que estamos desarrollando una partida de dominó siguiendo las reglas de la modalidad *Dominó Internacional*.

El objetivo del juego es alcanzar una determinada puntuación previamente fijada, jugando para ello las manos o rondas que sean precisas. En el caso del Dominó Internacional, el número de puntos son 200. En esta modalidad se enfrentan dos equipos, cada uno formado por una pareja de jugadores dispuestos en la mesa de forma alternativa.

Antes de empezar, las fichas se colocan boca abajo sobre la mesa y se revuelven para que los jugadores las recojan al azar en igual número cada uno. Cada jugador cogerá 7 fichas. La primera ronda la comenzará el jugador que posea el seis doble. En las siguientes rondas, empezará el jugador a la derecha del que empezó la ronda anterior. Podrá comenzar usando cualquier ficha, no tiene porqué ser doble.

En su turno cada jugador colocará una de sus piezas con la restricción de que dos piezas sólo pueden colocarse juntas cuando los cuadrados adyacentes sean del mismo valor. Si un jugador no puede colocar ninguna ficha en su turno tendrá que pasar el turno al siguiente jugador.

La mano continúa hasta que se da alguna de las dos situaciones:

- Alguno de los jugadores se queda sin fichas por colocar en la mesa. En este caso el jugador se dice que dominó la partida y la pareja ganadora sumará la totalidad de los puntos no jugados, es decir, la suma de los puntos en las fichas que resten por jugar a ambas parejas.
- En caso de cierre — es decir, cuando a pesar de quedar fichas en juego ninguna pueda colocarse — ganará la pareja cuyas fichas sumen menos puntos. Esta situación solamente ocurre cuando el mismo número está en ambos extremos del juego, y las siete fichas de ese número ya han sido jugadas. En este caso gana la pareja/jugador que menos puntos tenga en sus fichas, y se le suman los puntos del perdedor al ganador. Al igual que en el anterior caso, la pareja ganadora sumará la totalidad de los puntos no jugados, es decir, la suma de todos los puntos en las fichas que resten por jugar a ambos equipos.



Figura 2.4: Ejemplo de partida terminada en cierre. Se han colocado todas las blancas, por lo que no es posible que algún jugador continúe la partida

La partida finaliza una vez que un equipo ha alcanzado o superado los 200 puntos.

2.1.3. El dominó es un juego de señores

Una gran frase que resume la filosofía del Dominó es que *el dominó es un juego de señores*. Esta frase viene a describir tanto la filosofía del juego como parte de las reglas que lo normalizan, y tendrá mucha repercusión en cuanto al diseño del sistema experto, por lo que pasamos a explicarla a continuación.

Durante el transcurso de una partida de dominó las parejas no pueden comunicar ningún tipo de información que pueda afectar al desarrollo normal de la partida, estando totalmente prohibidos cualquier tipo de gestos entre jugadores destinados a comunicar futuras intenciones, fichas o estrategias conjuntas. Pero existe una excepción a esta regla.

La única señal o gesto válido en el juego del dominó es *la pensada*. Cuando toca el turno de jugar, se tiene la opción de pensar durante un tiempo relativamente largo para hacerle entender al compañero que

se tienen varias fichas del mismo número que va a tapar o que va a cuadrar. O por el contrario, jugar de inmediato, sin pensar, indica que no se tienen más fichas de ese número.

Por lo tanto, a la hora de colocar una ficha sobre la mesa podemos — aunque quizás tendríamos que decir que debemos — comunicar cierto tipo de información a todos los jugadores de la mesa. Esta información resulta tremadamente valiosa, y puede hacer que la partida se decante sobre un equipo o sobre el otro.

Hemos resaltar y aclarar que, como bien dicta la frase de que *el dominó es un juego de señores*, esta seña no puede utilizarse para confundir o engañar al contrario, y en los diferentes torneos o campeonatos de dominó supone la descalificación inmediata.

2.1.4. Juego por parejas

El juego por parejas que caracteriza el dominó internacional, a diferencia de otros tipos de modalidades, determina las estrategias que han de seguirse para que un equipo consiga la victoria, ya que esta únicamente se consigue trabajando en equipo; como juego de mesa por equipos, necesita imprescindible colaboración mutua entre compañeros si se quiere conseguir la victoria común.

En el desarrollo de una partida los jugadores van desempeñando consecutivamente diferentes roles. Para designar a cada jugador, es común utilizar la siguiente nomenclatura:

j1 Jugador mano que inicia la mano

j2 Jugador a la derecha del que inicia la partida

j3 Compañero del jugador que inicia la mano

j4 Compañero del jugador j2, tiene a su derecha al jugador que inicia la mano.

Desde el punto de vista de un jugador concreto, las posibilidades son:

Ser el jugador mano — jugador que inicia la partida Este jugador es el que debe dominar la mano, ya que al comenzar está orientando la partida hacia la posición que le interesa; normalmente, intentará que el juego se mueva por el palo del que tenga más fichas. Este jugador también cuenta con la ventaja de que, al ser el primero que coloca ficha (salvo que no pueda jugar algún turno), es el que tiene siempre igual o menor número de fichas que cualquier otro jugador y puede quedarse sin ninguna antes.

Por estas circunstancias, es el único jugador que puede (y debe) jugar para beneficio propio.

Segundo jugador — jugador a la derecha del que inicia la partida La labor de este jugador es evitar que el jugador mano desarrolle su estrategia, *matando* las fichas con las que inicie e intentando reorientar la partida hacia una posición más ventajosa para él o para su compañero.

Tercer jugador — compañero del jugador mano Este jugador debe apoyar en la medida de lo posible la estrategia del jugador mano. Su juego está supeditado en origen a colocar fichas del palo con el que haya comenzado su compañero, con la intención de facilitarle el juego y que pueda dominar a ese palo.

Cuarto jugador — a la izquierda del jugador mano este jugador debe intentar matar las fichas que coloque la pareja del jugador mano, y evitar que el juego se desarrolle por el palo al que abrió el jugador mano.

Más tarde, dependiendo de las circunstancias de la partida, los roles pueden ir cambiando al no poder jugar algunos jugadores sus turnos. En todo caso, el jugador referencia será el que tenga menos fichas que el resto, pues, si no pasara ningún turno, ganaría la partida. Este puede intentar ganarla por sí mismo si contempla la posibilidad y estima que sus adversarios están a la espera de sus movimientos.

2.1.5. Técnicas avanzadas

- Soltar dobles
- Cierres
- Salida en falso
- Jugar para el otro
- Exceso de fichas de un tipo (de un sólo número, dobles, etc).

2.2. Inteligencia artificial

A la hora de afrontar un proyecto que simule cierto comportamiento *humano*, debemos usar técnicas de Inteligencia Artificial, en busca de herramientas y metodologías que nos ayuden a afrontar este difícil problema, probablemente uno de los más complicados dentro de la Ingeniería Informática.

2.2.1. Sistemas expertos

Para implementar la inteligencia de los contrincantes de Dominous se utilizará un **sistema experto** [Gia89]. El desarrollo de sistemas expertos es una rama de la Inteligencia Artificial, que imita los mecanismos y la forma de pensar de un experto en cierta materia para resolver problemas de su campo de aplicación. Esto lo hace adecuado para un juego como el dominó, cuyas estrategias ganadoras son conocidas. Estos sistemas tienen una gran implantación en diversas ramas de la ciencia, como medicina, ingeniería o sistemas de decisiones para negocios.

Arquitectura general

Si consideramos el sistema una caja negra, el usuario del sistema experto sólo tiene que proporcionarle información sobre el problema y recibirá como respuesta el consejo del sistema. Para ello el sistema incorpora internamente dos elementos: la base de conocimiento (donde se almacena la información conocida) y el motor de inferencia (que permite obtener las respuestas adecuadas a partir de la información que el usuario introdujo y la base de conocimiento). La figura 2.5 muestra esta arquitectura.

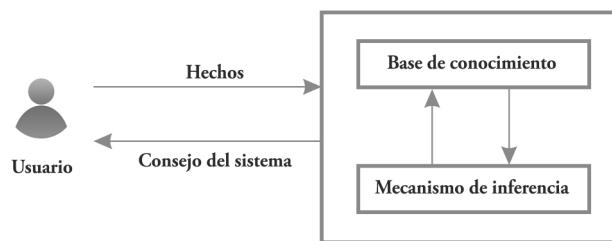


Figura 2.5: Esquema conceptual de un sistema experto

Desarrollo de un sistema experto

El proceso de construcción de un sistema experto se denomina ingeniería del conocimiento. Se basa en una serie de fases que se iteran hasta conseguir un sistema adecuado (figura 2.6). En una primera etapa el ingeniero de conocimiento tiene una entrevista con el experto, del que intenta obtener sus conocimientos. Después los implementa en el sistema. Y, por último, el experto evalúa el sistema. Si la evaluación no es del todo satisfactoria, se repite el proceso para refinar el resultado.

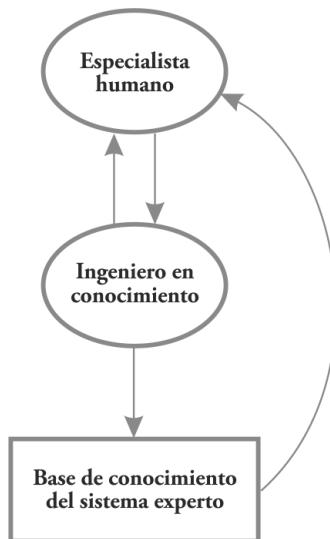


Figura 2.6: Desarrollo de un sistema experto

2.2.2. Sistemas expertos basados en reglas

De los diversos tipos de sistemas expertos que existen [Lia05] este trabajo se centrará en los sistemas expertos basados en reglas. Estos son los más adecuados para un simulador de dominó al ser este un problema con conocimiento parcial del entorno (esto es, se saben las fichas que hay en la partida, pero no qué jugador tiene cada una de ellas) y cuya inteligencia está bien estudiada y expresada en reglas concretas [Bor90]. Aún así, existe en la bibliografía una aproximación basada en redes bayesianas [Tur06].

No es este el primer trabajo en este sentido, pues existen diferentes aplicaciones de los sistemas expertos basados en reglas a todo tipos de juegos, desde sistemas en tiempo real [Fat05] a diversos juegos de tablero (como [Qui10] o [Vá11]).

Los sistemas expertos basados en reglas (sistemas expertos a secas a partir de ahora) suelen estar dividido en seis partes principales, como se observa en la figura 2.7. A continuación se detallan cada una de ellas.

Base de conocimiento Contiene conocimiento extraído del experto en forma de reglas.

Base de hechos (o Memoria activa de trabajo) Contiene los hechos sobre un problema que se conocen.

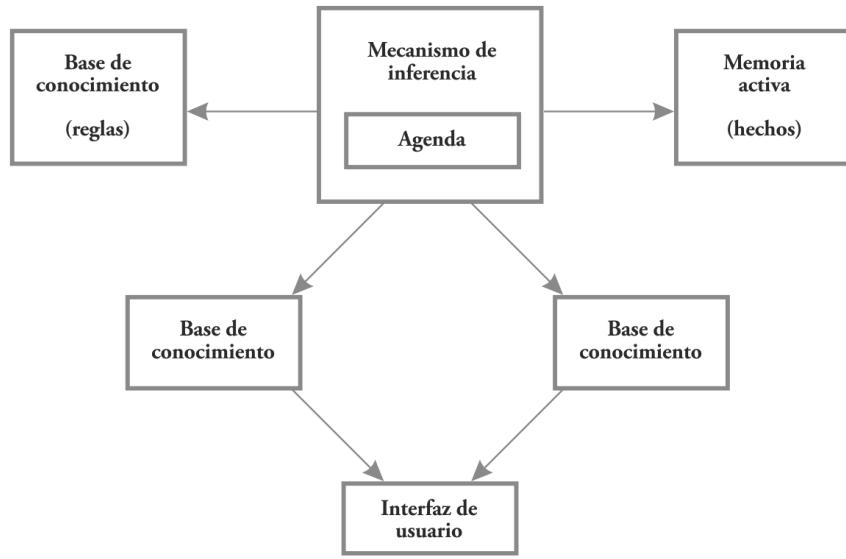


Figura 2.7: Arquitectura general de un sistema experto basado en reglas.

Mecanismo de inferencia Motor que implementa el proceso de razonamiento humano. Comprueba las reglas que satisface la memoria activa y ejecuta la que corresponda. Incluye la Agenda, que es el conjunto de reglas satisfechas en un momento dado.

Módulo de justificación (o Medio de explicación) Detalla el razonamiento utilizado por el sistema para llegar a la conclusión proporcionada al usuario.

Medio para la adquisición de conocimiento Permite al usuario introducir nuevo conocimiento en el sistema sin tener que mediar el ingeniero de conocimiento. El sistema suele recibir ejemplos para aprender de ellos.

Interfaz de usuario Implementa la interacción del usuario con el sistema. La interacción la puede realizar tanto una persona como otro sistema que deseé hacer uso del sistema experto.

En los siguientes apartados se desarrollan los elementos más importantes: la memoria activa, la base de conocimiento y el motor de inferencia.

Memoria activa

Mantiene una base de datos con la información del sistema. Esta varía constantemente, tanto por información que se reciba del exterior (del problema, en general) como por la información interna que modifiquen las reglas de la base de conocimiento.

Está formada por dos tipos de elementos: las plantillas y los objetos. Las plantillas definen la estructura de la información que se almacena. En concreto especifican los atributos (del inglés *slots*) que podrán tener los objetos. Y los objetos en sí almacenan información de acuerdo a la estructura indicada en las plantillas.

Base de conocimiento

La base de conocimiento (también llamada *sistema de producción*) de un sistema experto es el conjunto formado por todo el conocimiento que tiene el sistema en un momento dado. Existen sistemas en los que el conocimiento se mantiene constante a través del tiempo (la mayoría), y otros en los que el sistema aprende conocimiento a lo largo de su vida.

El conocimiento se almacena en forma de reglas. Las reglas suelen estar almacenadas como expresiones “Si CONDICIÓN entonces ACCIÓN”. Un ejemplo sería:

Listado 2.1: Regla en pseudocódigo

```
1 (defrule biblioteca-regla-1
2   (libro (titulo ?X) (estado retraso) (prestado-a ?Y))
3   (persona (nombre ?Y) (domicilio ?Z))
4   =>
5   (mandar-nota-retraso ?X ?Y ?Z))
```

Esta regla en lenguaje natural se podría leer línea a línea:

Listado 2.2: Regla en lenguaje natural

```
1 Regla 1:
2 Si
3   hay un libro, titulado X, con retraso, y
4   prestado a una persona de nombre Y
5   Y
6   el domicilio de la persona Y es Z
7 Entonces
8   mandarle una nota de retraso a Y en Z del libro X
```

El libro y persona en concreto deben de estar en la memoria activa. Y las acciones (como “mandar-nota-retraso” del ejemplo) se definen en funciones escritas por el usuario.

En la bibliografía se suele denominar antecedente (o parte izquierda de la regla¹) a la condición que tiene el “Si”, y consecuente (o parte derecha de la regla²) a su acción.

Motor de inferencia

Su cometido es aplicar el conjunto de reglas del tipo “si-entonces” de la base de conocimiento al conjunto de datos que están en la memoria activa en un momento dado.

La *agenda* es el conjunto de reglas que se pueden ejecutar en un momento del tiempo dado. Hay que tener en cuenta que no es lo mismo la activación que la ejecución de una regla. La activación indica que la regla se puede disparar (porque se satisface su condición en el “Si”). Pero en un instante del tiempo puede haber más de una regla activa. Y si una de ellas se ejecuta (se dispara) puede provocar cambios en la memoria activa e introducir o eliminar reglas de la agenda. Además hay que tener en cuenta que en algunos sistemas (sobre todo en sistemas en tiempo real) la memoria activa puede estar recibiendo información constantemente, lo que provoca muchas modificaciones en la agenda tras disparar una regla

¹En inglés *Left-Hand-Side (L.H.S.)*.

²En inglés *Right-Hand-Side (R.H.S.)*.

y actualizar la información del mundo real.

En concreto, el método seguido para determinar qué regla de la agenda se dispara se denomina *estrategia de resolución de conflictos*. Existen muchas de ellas, que pueden ser combinadas: asignando prioridades a las reglas, disparar primero las reglas menos usadas (o más las usadas), disparar las reglas con mayor (o menor) número de condiciones, etc.

El algoritmo más simple para implementar el motor de inferencia (llamado *regla de aproximación a la búsqueda de hechos*) sería recorrer circularmente de modo indefinido el conjunto de reglas, buscando reglas que se satisfagan y ejecutándolas. El problema es que tiene una complejidad computacional de orden $O(R * F^P)$, siendo R el número de reglas, P la media de comprobaciones por el *L.H.S.* de cada regla y F la cantidad de reglas de la base de conocimiento. Evidentemente, este orden no es deseable para sistemas grandes, pues se dispara al incrementarse las comprobaciones por regla, por lo que pueden necesitarse optimizaciones [Pal05]. Sin embargo, como veremos, es suficiente para problemas pequeños como puede ser jugar una partida de dominó.

Capítulo 3

Planificación

Para el desarrollo de **Dominous** se decidió utilizar el modelo evolutivo iterativo incremental para el ciclo de vida del proyecto. La decisión fue tomada ya que, a pesar de tener acotado el ámbito y los requisitos del programa, la funcionalidad concreta de cada uno de los apartados se desconocía en un principio.

El modelo iterativo incremental es un modelo de tipo evolutivo que está basado en varios ciclos en cascada realimentados aplicados repetidamente, con una filosofía iterativa. Como se comenta en [Som05], al final de cada iteración se le realiza una entrega al cliente final; en este caso, los clientes han sido los tutores del proyecto, que a cada iteración iban dictando las líneas maestras generales a tomar a cada nuevo paso.

Las ventajas de utilizar un modelo iterativo incremental son básicamente los siguientes:

1. Construir un sistema pequeño es siempre menos costoso en términos de riesgo.
2. Al ir desarrollando parte de las funcionalidades, es más fácil determinar si los requisitos planeados para los niveles subsiguientes son correctos.
3. Si se comete algún error grave, sólo la última iteración necesita ser descartada.
4. Reduciendo el tiempo de desarrollo de un sistema (en este caso en incremento del sistema) decrecen las probabilidades que esos requerimientos de usuarios puedan cambiar durante el desarrollo.
5. Los errores de desarrollo realizados en un incremento, pueden ser arreglados antes del comienzo del próximo incremento.

El proyecto **Dominous** consta de tres subsistemas que son los que ocupan el grueso del desarrollo:

1. El primero es el motor de la partida: Controla los jugadores, las fichas en la mesa, la partida, situaciones irregulares y cualquier otro elemento referente únicamente al ámbito del dominó.
2. Por otro lado está el motor gráfico, que será la interfaz entre la partida y el usuario, permitiendo el movimiento fluido por las diferentes secciones del programa e interactuando de forma directa con el motor de la partida, mostrando las fichas actuales y habilitando la interacción del jugador con el mundo.
3. Y por último tenemos el motor de Inteligencia Artificial. Este motor será el que alimente la inteligencia y las acciones y decisiones de los jugadores controlados por el ordenador.

3.1. Organización de las partes básicas de la aplicación

Después de analizar minuciosamente los requisitos de la aplicación, se decidió atacar al grueso de la misma, es decir, la gestión de una partida de dominó en sí, y se estimó que esta debería dividirse en los siguientes apartados:

Subsistema de gestión de la partida Este subsistema se encarga de gestionar todo lo relacionado con el control de la partida que se desarrolla actualmente. Desde el conteo de puntos por parte de cada equipo hasta las fichas de cada jugador, pasando por comprobaciones de que la partida se desarrolla con normalidad y no se producen violaciones de las reglas y normas, barajar las fichas, comprobar que se cierra la partida o pedir fichas a los diferentes jugadores.

Subsistema gráfico Su labor es mostrar por pantalla la información que controla el subsistema de gestión de la partida. Debe dibujar en la pantalla el tablero con las fichas colocadas actualmente, las fichas de cada jugador, los puntos de cada equipo y demás información que sea necesario mostrar.

Subsistema de Inteligencia Artificial Este último apartado se encarga de dotar de inteligencia al conjunto de jugadores que intervienen en la partida y que están controlados por el ordenador. Incluye crear los sistemas expertos, las bases de conocimiento, la librería de reglas y las diferentes interfaces entre el juego y el sistema de gestión de la partida.

Una vez dividido todo el proyecto en módulos, podemos obtener el siguiente diagrama de Gantt [3.1](#):

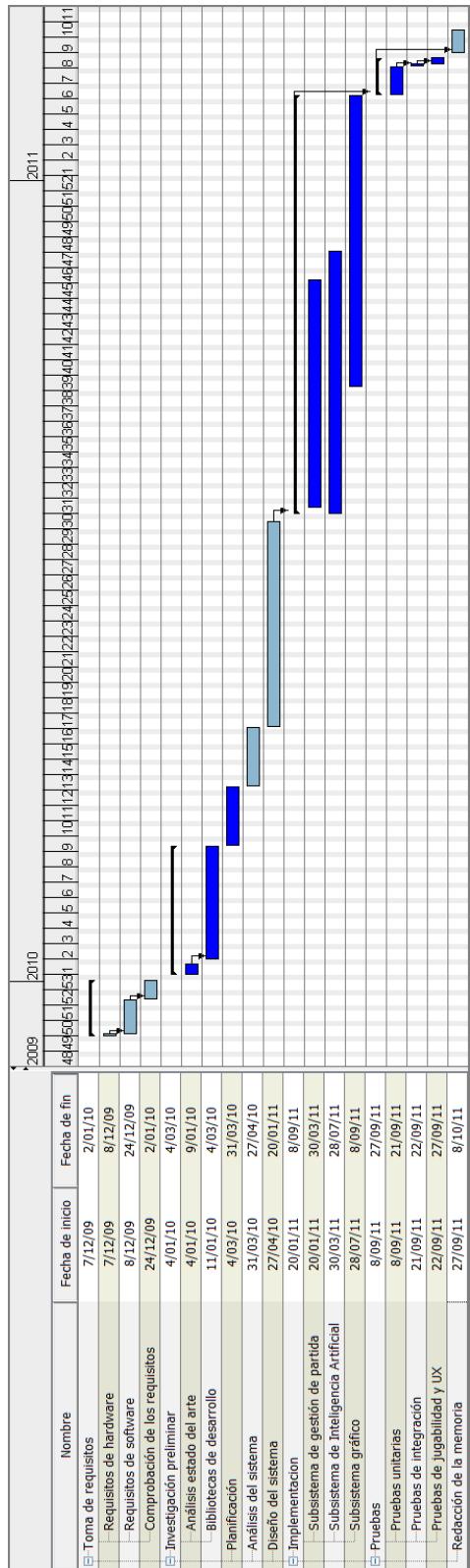


Figura 3.1: Diagrama de Gantt

Capítulo 4

Análisis

4.1. Toma de requisitos

En el desarrollo de esta aplicación la toma de requisitos se hizo mediante reuniones con los tutores del proyecto, que realizaban el papel de cliente potencial de la misma. Después de varias reuniones se obtuvo el listado de requisitos que se muestra en las siguientes secciones:

4.1.1. Requisitos de interfaces externas

En este apartado se van a describir los requisitos de conexión entre el software y el hardware, así como la interfaz del usuario. Así que pasamos a definir el interfaz entre el videojuego y el usuario. Todas las ventanas de la aplicación podrán ser mostradas a pantalla completa o en formato de ventana con una resolución de 800 por 600 píxeles. A continuación se definen las distintas ventanas con las que el usuario se puede encontrar:

Ventana de introducción Esta primera ventana mostrará únicamente el logotipo de Dominous, situando al usuario en contexto para iniciarla en la ejecución del programa

Ventana del menú principal La ventana del menú principal muestra el menú de inicio de Dominous, en el que el usuario podrá elegir entre las opciones más generales del juego:

- Partida clásica
- Laboratorio
- Opciones
- Tutorial
- Salir

Este menú y los siguientes que se describan serán completamente manejados por el ratón y bastará un clic encima de una opción para acceder a ella.

Ventana de selección de personaje Esta ventana mostrará una interfaz que permite al usuario elegir los diferentes participantes que se enfrentarán en la siguiente partida. En caso del modo laboratorio se elegirán los cuatro jugadores, y en caso de partida clásica serán tres jugadores controlados por el ordenador más el jugador humano.

Ventana de partida Esta será la ventana principal de todo el juego. Mostrará una partida de dominó de dos equipos, el tablero e información de la partida, e irá actualizando el tablero según se vaya desarrollando la misma partida. Mediante la pulsación de la tecla ESC o clic sobre el botón *menú*

se desplegará el menú interno de la partida, que permitirá abandonarla a pesar de no haberse terminado la partida actual.

Ventana de laboratorio La ventana de laboratorio proporciona una interfaz para que el usuario de la aplicación pueda generar un conjunto elevado de partidas entre dos equipos definidos, con la idea de poder decidir qué pareja presenta las mejores características de Inteligencia Artificial.

Ventana del modo tutorial Por último la ventana del modo tutorial mostrará al usuario información sobre el juego del dominó mediante un conjunto de pantallas explicativas, para que pueda aprender más sobre el mundo del dominó sin necesidad de salir de la aplicación.

4.1.2. Requisitos funcionales

Los requisitos funcionales que el sistema debe ofrecer al usuario son los siguientes:

- Poder jugar una partida de dominó con tres jugadores más controlados por el ordenador.
- Enfrentar a dos parejas de jugadores controlados por ordenador, haciendo que jueguen un número de partidas seguidas en modo automático (esto es, sin visualizar la partida que se desarrolla y mostrando únicamente los resultados), con la finalidad de poder decidir qué pareja posee una Inteligencia Artificial más avanzada.
- Acceder al modo tutorial, para realizar un aprendizaje de las normas, técnicas y usos del dominó.
- Cambiar el tipo de juego para que cuatro jugadores manejados por la máquina puedan desarrollar una partida en modo visual.
- Seleccionar otro tema gráfico para que, tanto fichas como tablero como otros elementos gráficos, cambien a gusto del usuario, eligiendo entre ciertos temas.
- Cambiar de modo ventana a modo pantalla completa.

4.1.3. Requisitos de rendimiento

El rendimiento de la aplicación debe ser tal que permita un desempeño agradable de la partida. Este requisito hace referencia principalmente a los siguientes asuntos:

- El sistema de inteligencia artificial debe ser lo suficientemente ágil y estar ajustado y perfeccionado para que los tiempos empleados en cálculos de toma de decisiones no ralenticen la partida. Se cuenta como asunto el que, en el desarrollo de una partida de dominó, los tiempos de espera también se interpretan, por lo que debemos realizar los cálculos dentro de un cierto margen de tiempo.
- Por otro lado, el motor gráfico debe estar optimizado para que el usuario no aprecie movimientos bruscos a la hora de manejar la aplicación. No olvidemos que estamos desarrollando un videojuego, así que el programa debe mostrar cierta agilidad a la hora de realizar movimientos y transiciones entre los diferentes estados de la partida, incluyendo menús, fichas, o asuntos relativos a la interfaz, como puede ser el arrastrar una ficha a su lugar correspondiente.

Es importante recordar en todo momento que estamos desarrollando una aplicación en tiempo real, por lo que debe primar la velocidad sobre otros factores como el consumo de memoria principal.

4.1.4. Restricciones de diseño

Como bien comentábamos en el punto anterior, a la hora de realizar el diseño de la aplicación tienen que primar los tiempos de respuesta sobre el consumo de recursos de la memoria principal o secundaria. Esta es la principal restricción que tendrá el diseño de nuestra aplicación.

Los videojuegos están pensados para ejecutarse como aplicación principal, no para compartir recursos con otros programas; por esta razón se permite que consuman muchos recursos.

4.1.5. Requisitos del sistema software

La aplicación debe cumplir con los siguientes requisitos de sistema:

- La aplicación debe ejecutarse de forma multiplataforma, incluyendo como mínimo los sistemas operativos:
 - En Microsoft Windows — Realizándose las pruebas en la versión Windows 7 con las últimas actualizaciones.
 - En sistemas GNU/Linux — Utilizando la distribución Ubuntu en su versión 10.04 con su instalación por defecto y con todas las actualizaciones del sistema.
- El código de la aplicación no debe ser dependiente del sistema operativo en el que se desarrolle la aplicación, y debe ser un código mantenable y fácilmente ampliable para futuras mejoras y versiones.

4.2. Modelo de casos de uso

Para describir los comportamientos que tendrá el sistema, utilizaremos el lenguaje guaje de modelado de sistemas UML [Pre02]; éste representa los requisitos funcionales de todo el sistema, centrándose en qué hace pero no en cómo lo hace.

A continuación describimos uno por uno cada caso de uso.

4.2.1. Diagrama de casos de uso

Como primer paso, debemos mostrar el diagrama de casos de uso que representa la funcionalidad completa de la aplicación. El esquema utilizado es el siguiente:

1. Identificar los usuarios del sistema y sus posibles roles.
2. Para cada rol definido, identificar todas las formas que tiene de interactuar con el sistema. En el caso de Dominous, existe un único rol de acceso a la aplicación, por lo que la especificación de usuario será única.
3. Crear todos los casos de uso para poder describir los objetivos que se desean cumplir.
4. Estructurar y definir esos casos de uso.

4.2.2. Descripción de los casos de uso

A continuación pasamos a la descripción de los casos de uso. Para ello se va a utilizar una notación formal usando plantillas, con la intención y finalidad de que este texto sea legible y comprensible por un usuario que no sea experto.

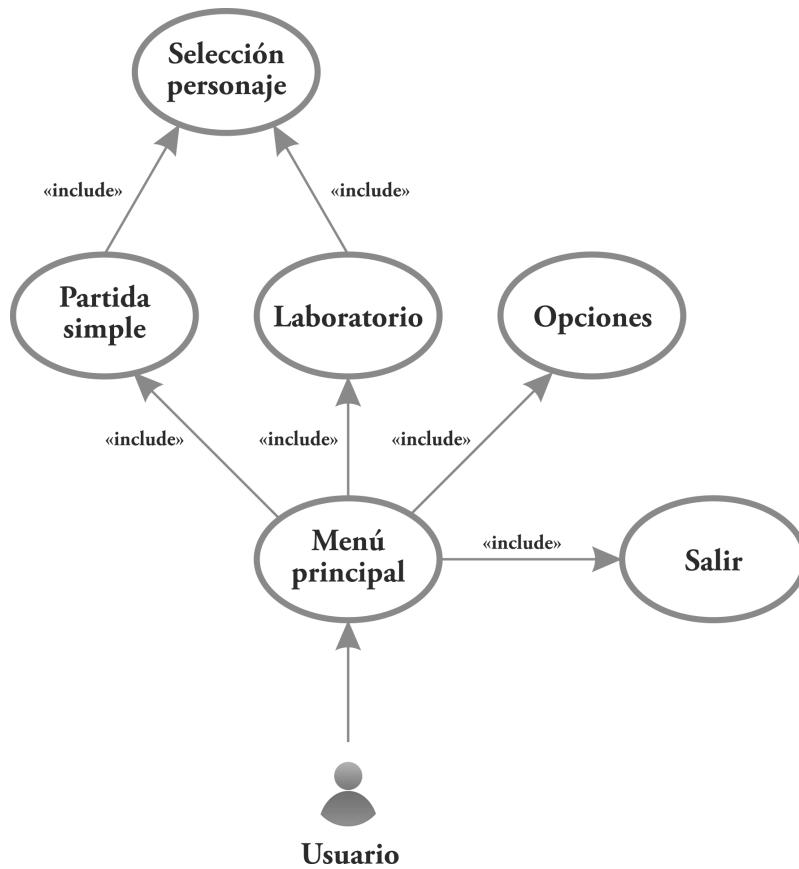


Figura 4.1: Diagrama de casos de uso del sistema

Caso de uso: Menú principal

Caso de uso Menú principal

Descripción Se muestra el menú principal de la aplicación, desde donde es posible acceder a los diferentes modos de juego y a las opciones.

Actores Usuario

Precondiciones Ninguna

Postcondiciones Ninguna

Escenario principal

1. El sistema muestra el menú principal del juego en la pantalla
2. El usuario selecciona el modo **partida simple**
3. El sistema inicia el modo de elección de jugadores

Extensiones — flujo alternativo

- ***a** El usuario cierra la ventana de la aplicación y sale de la aplicación
- 2a** El usuario pulsa sobre el botón de laboratorio, dirigiéndose a ese apartado de la aplicación
- 2b** El usuario pulsa el botón de tutorial, dirigiéndose a ese apartado de la aplicación

2c El usuario pulsa sobre las opciones, dirigiéndose a ese apartado de la aplicación

2d El usuario pulsa sobre el botón de salir, cerrándose la aplicación

Caso de uso: Salir

Caso de uso Salir.

Descripción Primeramente se muestra la pantalla de información de la aplicación — con datos sobre desarrolladores, licencias y cualquier otra información que pueda resultar de interés para el usuario.

Actores Usuario.

Precondiciones Ninguna.

Postcondiciones Se sale de la aplicación.

Escenario principal

1. El sistema muestra la pantalla de información.
2. El usuario pulsa sobre cualquier lugar de la aplicación.
3. La aplicación se cierra.

Extensiones — flujo alternativo

***a** El usuario cierra la ventana de la aplicación y sale de la aplicación.

Caso de uso: Partida simple

Caso de uso Partida simple.

Descripción El usuario pulsa el botón partida simple con la intención de comenzar una partida de dominó con las opciones por defecto, esto es, jugando por parejas con tres jugadores más controlados por el ordenador. El usuario selecciona su pareja de equipo y sus dos adversarios, y da comienzo la partida

Actores Usuario.

Precondiciones Ninguna.

Postcondiciones Se jugará una partida de Dominous

Escenario principal

1. El usuario desea jugar una partida.
2. El usuario selecciona la opción de menú **partida simple**.
3. include SELECCIÓN DE JUGADORES.
4. El sistema inicializa y muestra la partida actual por pantalla
5. Por cada mano que se desarrolle:
 - a) El usuario y el sistema interactúan durante la partida.
 - b) El sistema muestra quién ha ganado la mano.
6. El sistema muestra quién ha ganado al final de la partida.

7. El sistema cierra la partida y muestra de nuevo el menú principal.

Extensiones — flujo alternativo

- ***a** El usuario cierra la ventana de la aplicación y sale de la aplicación.
- 2a** El usuario pulsa sobre cualquier otra opción del menú que no sea la de realizar una partida simple — se rompe el flujo y se redirige al usuario a la opción elegida.
- 5a** El usuario pulsa el botón de menú y pulsa en salir de la partida. Se termina la partida actual y se vuelve al menú principal.

Caso de uso: Laboratorio

Caso de uso Laboratorio

Descripción El usuario desea realizar pruebas y análisis sobre las diferentes habilidades de cada jugador controlado por la Inteligencia Artificial del programa, enfrentando a dos equipos de jugadores a un número de partidas, y calculando el mejor equipo según el número de victorias alcanzadas.

Actores Usuario.

Precondiciones Ninguna.

Postcondiciones Se realizará una competición a 100 partidas, cada partida jugada a 200 puntos, enfrentando a las dos parejas de jugadores seleccionados previamente.

Escenario principal

1. El usuario desea acceder al modo laboratorio.
2. El usuario selecciona la opción de menú **laboratorio**.
3. include SELECCIÓN DE JUGADORES.
4. El sistema comienza con los enfrentamientos entre ambas parejas
5. El sistema finaliza los enfrentamientos y destaca al equipo ganador
6. El sistema cierra el modo laboratorio y muestra de nuevo el menú principal.

Extensiones — flujo alternativo

- ***a** El usuario cierra la ventana de la aplicación y sale de la aplicación.
- ***b** El usuario pulsa el botón de salir del modo laboratorio y se vuelve al menú principal.
- 2a** El usuario pulsa sobre cualquier otra opción del menú que no sea la de realizar una partida simple — se rompe el flujo y se redirige al usuario a la opción elegida.
- 4a** El usuario pulsa el botón de pausa — se realiza una pausa en el desarrollo de las partidas, hasta el momento en el que el usuario vuelve a pulsar el botón de pausa y se reanudan los enfrentamientos
- 4b** El usuario pulsa el botón de reiniciar — se resetean a cero los contadores de partidas, puntos y cualquier otra estadística sobre la que se realice el conteo, y se comienza a desarrollar de nuevo un nuevo conjunto de enfrentamientos.
- 6a** El usuario pulsa el botón de salir del modo laboratorio y se vuelve al menú principal.

Caso de uso: Opciones

Caso de uso Opciones

Descripción El usuario decide cambiar alguna elemento de la configuración con la que se desarrollan las partidas y que modifican el comportamiento transversal de la aplicación.

Actores Usuario.

Precondiciones Ninguna.

Postcondiciones Se cambian las opciones que se desean modificar por parte del usuario, y se vuelve de nuevo al menú principal.

Escenario principal

1. El usuario desea acceder y cambiar las opciones del juego.
2. El usuario selecciona la opción de menú **opciones**.
3. El sistema muestra todas las opciones.
4. El usuario pulsa en volver.
5. El sistema guarda en la configuración general del juego las opciones seleccionadas, para que en posteriores ejecuciones se mantengan las mismas opciones previamente seleccionadas.
6. El sistema muestra de nuevo el menú principal.

Extensiones — flujo alternativo

- ***a** El usuario cierra la ventana de la aplicación y sale de la aplicación.
- ***b** El usuario pulsa el botón volver y se retorna al menú principal.
- 2a** El usuario pulsa sobre cualquier otra opción del menú que no sea la de realizar una partida simple — se rompe el flujo y se redirige al usuario a la opción elegida.
- 3a** El usuario pulsa en **modo de juego** — el sistema va rotando entre las diferentes opciones de juego, que son dos: **modo de juego un jugador** (el jugador humano participa de la partida) y **modo de juego solo computadora** (todos los jugadores participantes estarán controlados por la máquina).
- 3b** El usuario pulsa en **tema gráfico** — el sistema va rotando entre los distintos temas gráficos que están instalados en la aplicación, y que más tarde cambiarán el aspecto visual de la partida.
- 3c** El usuario pulsa en **velocidad de juego** — el sistema permite elegir entre tres tipos de velocidades para la partida: **velocidad normal**, en el que la partida transcurre a una velocidad pausada y cómoda, **velocidad rápida**, en el que los jugadores manejados por la máquina colocan las fichas sin realizar pausas para pensar, y **velocidad extra rápida**, en el que, además de no realizar pausa, las fichas se mueven cuatro veces más rápido de la velocidad normal de juego.
- 3d** El usuario pulsa en **modo ventana** — El sistema permite cambiar entre dos modos de visualización del juego: **modo ventana**, en el que la aplicación se muestra dentro de una ventana controlada por el gestor de ventanas nativo del sistema, y el **modo pantalla completa**, donde la acción ocupa toda la pantalla activa del usuario.

Caso de uso: Selección de personaje

Caso de uso Selección de personaje.

Descripción El usuario desea seleccionar los jugadores que participarán en el siguiente juego a desarrollar, ya sea **partida simple** o modo **laboratorio**.

Actores Usuario.

Precondiciones El usuario ha pulsado previamente una de estas dos opciones del menú principal: **partida simple** o modo **laboratorio**.

Postcondiciones Se guardará en configuración los jugadores seleccionados por el usuario para el posterior desarrollo del juego.

Escenario principal

1. El usuario debe seleccionar los personajes que participarán en el juego
2. El sistema muestra los jugadores actuales.
3. El usuario selecciona los personajes que jugarán la partida o partidas siguientes.
4. El usuario está satisfecho con su elección y pulsa el botón de jugar.
5. El sistema guarda la información de los jugadores seleccionados.
6. El sistema pasa al siguiente paso, que puede ser **partida simple** o modo **laboratorio**, dependiendo del estado de la precondición.

Extensiones — flujo alternativo

- ***a** El usuario cierra la ventana de la aplicación y sale de la aplicación.
- ***b** El usuario pulsa el botón de volver, con lo que se retorna al menú principal.
- 3a** El usuario pulsa sobre cada jugador, con la intención de seleccionar aquellos que jugarán la partida o partidas siguientes. A cada pulsación, el sistema mostrará el siguiente jugador existente en el sistema, de forma cíclica.

4.3. Modelo conceptual de datos

Este apartado del análisis sirve para especificar los requisitos del sistema y las relaciones estáticas que existen entre ellos.

Para este fin se utiliza como herramienta los **diagramas de clase**. En estos diagramas se representan las clases de objetos, las asociaciones entre dichas clases, los atributos que componen las clases y las relaciones de integridad.

4.3.1. Diagrama de clases conceptuales

En este apartado se presenta una lista de las principales clases que formarán parte del sistema. Junto a cada nombre aparece una pequeña descripción sobre la labor que desempeña cada una, y la jerarquía se describe mediante la indentación de cada ítem del listado.

Juego Clase global, que inicia todo el programa y controla el flujo entre unos apartados y otros.

Motor gráfico Encargado de gestionar el apartado de la representación de la información por pantalla.

Ficha Las fichas que utiliza el motor gráfico no son fichas lógicas, sino que simplemente se emplean para colocarlas por pantalla y que el usuario pueda hacer un uso cómodo de la aplicación

Partida Se encarga de llevar el control de la partida a nivel lógico, vigilando que se cumplan en todo momento las reglas de juego y guardando información detallada sobre toda la partida.

Tablero El tablero contiene las fichas que ya se han colocado sobre la mesa.

Ficha Por otro lado, cada ficha es la unidad mínima para el transcurso del juego, pudiendo encontrarse en manos de los jugadores o bien depositadas sobre la mesa.

Inteligencia artificial Su labor es hacer que los personajes desempeñados por la máquina realicen jugadas y comportamientos simulando la inteligencia humana.

Reglas Las reglas definen diferentes jugadas o estrategias.

Jugadores Los jugadores emplean reglas para participar en la partida.

Música Conjunto de músicas que amenizan la partida.

Sonido Efectos de sonido que añaden feedback al usuario de las acciones realizadas.

Y en la siguiente imagen 4.2 podemos observar el diagrama de clase asociado a los requisitos obtenidos.

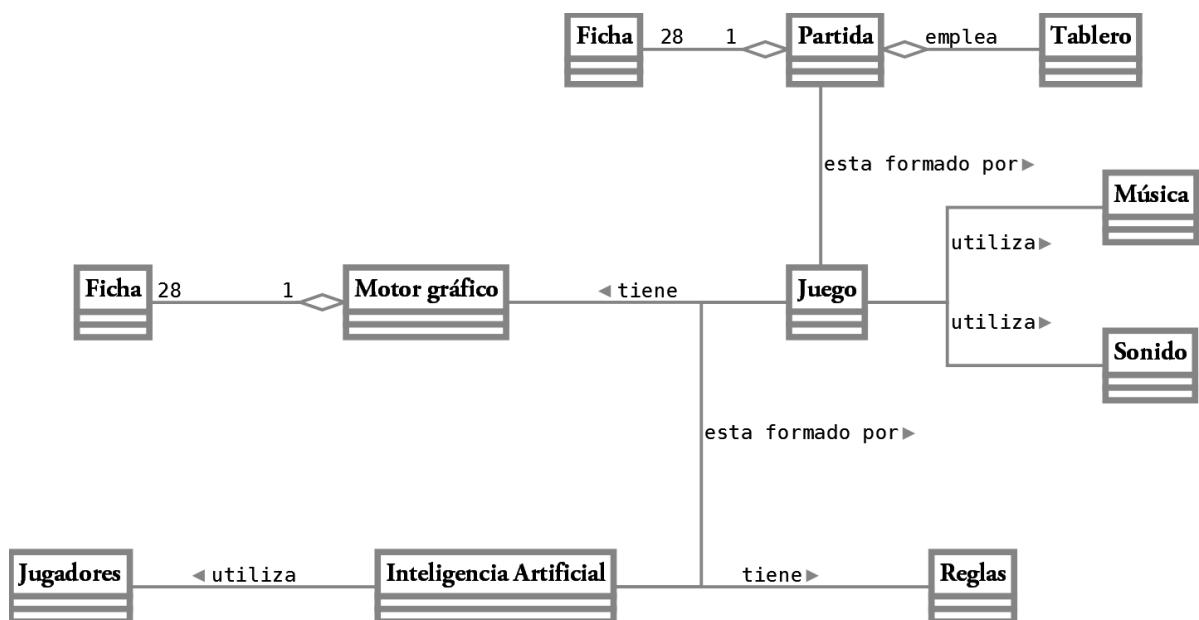


Figura 4.2: Diagrama de clases para los requisitos obtenidos

4.3.2. Modelo de comportamiento del sistema

El modelo de comportamiento define cómo debe actuar un sistema; este sistema a considerar es el que engloba a todos los objetos, y el modelo consta de dos partes:

- El Diagrama de secuencia del sistema, el cual muestra la secuencia de eventos entre los actores y el sistema.
- Los Contratos de las operaciones del sistema, que describen el efecto que producen las operaciones del sistema.

Modelo de comportamiento partida simple

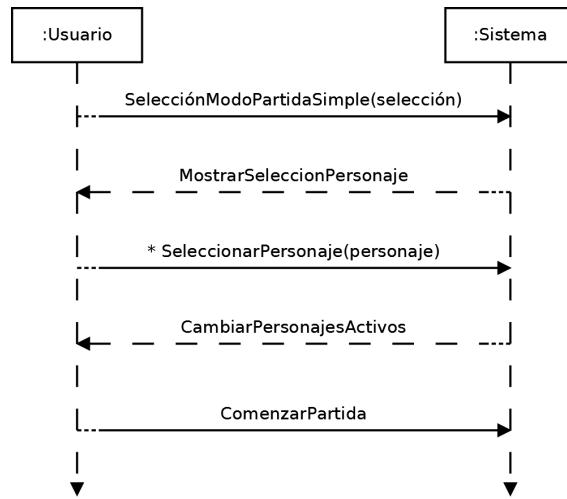


Figura 4.3: Diagrama de secuencia para el caso de uso Partida simple

Contrato de operaciones

Operación Menú principal

Actores Usuario, sistema

Responsabilidades Seleccionar los personajes que intervendrán en la partida actual

Precondiciones Ninguna

Postcondiciones Selección de personajes para la partida, comienzo de partida

Modelo de comportamiento modo laboratorio

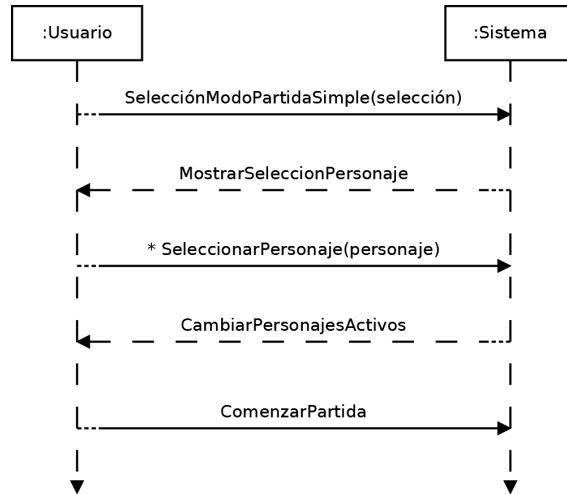


Figura 4.4: Diagrama de secuencia para el caso de uso del modo laboratorio

Contrato de operaciones

Operación Menú principal

Actores Usuario, sistema

Responsabilidades Desarrollar un número elevado de partidas automatizadas

Precondiciones Ninguna

Postcondiciones Selección de personajes para la partida, comienzo de partida

Capítulo 5

Diseño

5.1. Introducción

El primer paso del diseño de la aplicación será describir los requisitos del sistema necesarios para poder ejecutar la aplicación, y posteriormente comentaremos las herramientas que se van a usar para el desarrollo de la aplicación.

5.2. Definición de los requisitos del sistema

Los requisitos hardware y software necesarios para poder ejecutar con soltura la aplicación son los siguientes:

- Sistema operativo Microsoft Windows en su versión Windows 7, o sistemas basados en GNU/Linux tales como la distribución Ubuntu en su versión 10.04.
- Últimas actualizaciones del sistema instaladas, drivers y demás aplicaciones propias del sistema configuradas correctamente.
- Procesador igual o superior a 1,6 GHz.
- Memoria RAM igual o superior a 512 MB
- Tarjeta gráfica con aceleración 3D con un mínimo de 128 MB
- En versiones basadas en Linux, se requieren las siguientes librerías:
 - python-pygame
 - python-setuptools
 - PyOpenGL
 - PyOpenGL-accelerate
- Por otro lado, si el sistema está basado en Microsoft Windows, necesitaremos:
 - PyGame
 - Python OpenGL and Gloss

5.3. Herramientas utilizadas

5.3.1. Librería gráfica

Para todo el tema del aspecto gráfico, mis tutores me recomendaron que utilizara las librerías SDL¹, ya que son unas librerías orientadas al desarrollo de videojuegos con varias particularidades:

- Son completas, ya que permiten gestionar operaciones de dibujo en dos dimensiones, efectos de sonido y música, carga y gestión de imágenes, subsistemas de control de métodos de entrada, etcétera, por lo que contamos con una solución global para desarrollar videojuegos.
- Están programadas en C, por lo que se puede esperar un buen rendimiento de las librerías en diferentes entornos.
- Multiplataforma: es compatible oficialmente con los sistemas Microsoft Windows, GNU/Linux, Mac OS y QNX, además de otras arquitecturas y sistemas menos comunes como Sega Dreamcast, Sony PSP, WebOS, Google Android o Symbian entre otros.
- Tampoco hay que mantener al margen la característica de que cuenta con wrappers a otros lenguajes de programación como entre los que se encuentran C++, Ada, C#, BASIC, Erlang, Lua, Java o Python, por lo que nos da bastante libertad para elegir un lenguaje de programación principal.
- Publicado bajo licencia LGPL, con todas las ventajas que conlleva.
- Y por último no hay que menospreciar que uno de mis tutores emplea SDL a la hora de impartir la asignatura de Diseño de Videojuegos de la UCA, y contar con su conocimiento nos ayudará a desarrollar más rápidamente y solucionar antes nuestros posibles problemas.



Figura 5.1: Logotipo de la librería Simple DirectMedia Layer

En este aspecto, la utilización de las librerías SDL estaba clara. Potencia, comodidad, multiplataforma y con la posibilidad de utilizar diferentes lenguajes de programación.

5.3.2. Lenguaje de programación

Una vez que tocamos el tema de los lenguajes de programación, entra en escena la problemática sobre qué lenguaje utilizar. En principio se pensó emplear el lenguaje C++ por dos sencillas razones:

¹Simple Directmedia Layer

1. Por un lado es un lenguaje que hemos aprendido en la carrera, se ha utilizado en varias asignaturas de diferentes ramas, con lo cual la comodidad y familiaridad que podemos tener a la hora de programar es un punto importante a tener en cuenta.
2. Tampoco podemos olvidar que, al ser un lenguaje compilado, la velocidad de ejecución que se consigue es interesante, y mucho más tratándose de temas como la Inteligencia Artificial (donde puede ser necesario un uso intensivo de los recursos del sistema) o el desarrollo de videojuegos (en el que la potencia del ordenador repercute en una mejor experiencia del usuario)

Pero hay que detenerse un momento y pensar en la naturaleza del proyecto. Aunque el programa a desarrollar sea un videojuego, no hay que olvidar que hay diferentes tipos de juegos, que pueden condicionar o influir en nuestra forma de programarlo. En el caso del dominó, lo primero que debemos tener en cuenta es que el apartado gráfico no va a requerir de una gran potencia o despliegue de efectos: el dominó es un juego pausado y a diferencia de otros videojuegos lo importante en este caso es mostrar al usuario la información de la partida de una forma clara y sencilla, para que el jugador evalúe las posibilidades de acción y actúe en consecuencia.



Figura 5.2: Logotipo de Python - Copyright Python Software Foundation

Si tenemos en cuenta estas circunstancias, existen otros lenguajes que también deben entrar en juego, como por ejemplo **Python**. Buscando las diferencias, ventajas y desventajas de Python frente a C++, obtenemos el siguiente listado:

1. Python es un lenguaje de programación multiparadigma ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional.
2. Al igual que C++ es multiplataforma, y está publicado con la licencia **Python Software Foundation License**, que es una licencia de software libre permisiva, compatible con la GPL.
3. La sintaxis de Python es muy clara, simple, expresiva y legible, con lo cual los programas desarrollados bajo Python son más sencillos de entender [Pil04].
4. Python es un lenguaje interpretado, a diferencia de C++ que es compilado. Este aspecto podría suponer una desventaja ya que al ser interpretado puede resultar más lento, pero analicemos pausadamente estos factores:
 - Como ya hemos comentado previamente, nuestra aplicación, a pesar de enmarcarse dentro de las facciones de un videojuego, no requiere de grandes alardes de potencia gráfica como podría suponerse, ya que es un tipo de juego pausado y donde cómo se muestra la información es mucho más importante que la velocidad o los efectos de vídeo e imágenes.

- A pesar de ser interpretado, un gran conjunto de las funcionalidades de python — como librerías o funciones básicas del lenguaje — están programadas internamente en C, así que podríamos verlo como que estamos utilizando la comodidad de Python sobre la potencia de C, uniendo lo mejor de ambos mundos.

Por otro lado, se consigue la gran ventaja de contar con las librerías PyGame. Pygame es un conjunto de módulos del para Python que permiten la creación de videojuegos en dos dimensiones de una manera sencilla. Funciona como interfaz de las bibliotecas SDL, y está orientado al manejo de sprites. Las grandes ventajas de PyGame son:

- Alta portabilidad entre sistemas – gracias a lo cual cumplimos con el requisito de funcionamiento bajo sistemas Windows y GNU/Linux.
- Libre y gratuito, disponible bajo licencia LGPL — gracias a la cual sería posible incluso lanzar los juegos de forma comercial, gratuita, libre o incluso shareware.
- Soporta CPUs con varios núcleos, incluye funciones altamente optimizadas y escritas en C o directamente en ensamblador, altamente configurable y adaptable al proyecto a desarrollar, muy modular, incluye funciones para trabajar con eventos y sonidos, y muchas más funcionalidades que lo hacen idóneo para un proyecto de estas características.

5.3.3. Diseño y estilo visual de interfaces

El diseño visual que se ha desarrollado para la interfaz se ha creado desde cero, buscando los siguientes objetivos:

- Líneas sencillas, minimalistas, sin recargar innecesariamente la pantalla
- Botones grandes, para que sea fácil de utilizar por usuarios de edad avanzada.
- Textos con un punto de letra elevado, facilitando la rápida lectura y legibilidad del texto.

5.3.4. Documentación del código

Por otro lado, para gestionar toda la documentación del proyecto se decidió utilizar las siguientes herramientas:

- **LATEX** para escribir la memoria, ya que es una forma robusta y fiable de escribir una memoria para un Proyecto Fin de Carrera, descartándose otras posibles opciones por no ser adecuadas para la escritura de un documento de estas características [Mit04]. Para facilitar la compilación dispone de la herramienta GNU Make [Ger].
- Doxygen para la documentación del código fuente, porque además de documentar de manera sencilla y fácil de leer el mismo código fuente, genera una documentación en diferentes formatos [Dim]. Además, Doxygen funciona con lenguajes como C++, C, Java, Objective-C, Python, Fortran, VHDL, PHP o C# (entre otros), por lo que se puede acomodar a nuestras necesidades. Incluso existe una herramienta llamada **Doxypy** que nos permite reutilizar los comentarios *tipo Python* y adaptarlos a Doxygen, con lo cual ahorramos trabajo y cumplimos con la normativa recomendada para el código Python.



Figura 5.3: Diferentes elementos utilizados en la interfaz

5.3.5. Sistema de control de versiones

El código del proyecto Dominous, está alojado por completo dentro de la forja que proporciona Rediris, la Red Española para Interconexión de los Recursos Informáticos de las universidades y centros de investigación, donde tiene su web oficial [Ign]. Esta forja es básicamente una instalación de GForge [GFo] con un repositorio Subversion (SVN) asociado a cada proyecto.

Subversion lleva un control exhaustivo de todos los ficheros e iteraciones de código que se realizan en él, permitiendo volver a versiones anteriores de código, comprobar diferencias entre versiones o ficheros y cualquier otra operación propia de un sistema de control de versiones.

Se evaluaron otros sistemas de control de versiones distribuidos como GIT, Bazaar o Mercurial, pero se desecharon básicamente porque, por un lado, este proyecto cuenta únicamente con un desarrollador, y multitud de ventajas que ofrecen los sistemas de control de versiones distribuidos dejan de tener sentido. Si tenemos en cuenta esta circunstancia del proyecto, y por otro lado la integración de SVN con Rediris (con las ventajas de visualización de código y versiones que ofrece ViewCVS) decantaron la decisión sobre el lado de SVN.

5.4. Interfaz gráfica

Tomando los resultados obtenidos en la fase de análisis es necesario diseñar una interfaz gráfica amigable para el usuario y desde la cual se pueda interactuar con la aplicación. Para el diseño de las interfaces se intentará en todo momento que sean usables además de intentar conseguir que el usuario no pueda introducir datos erróneas para que no produzca comportamientos anómalos.



Figura 5.4: Varias pantallas con la interfaz de Dominous

5.4.1. Diagrama de interacción entre interfaces gráficas

En el siguiente diagrama 5.5 podemos observar la interacción entre las distintas interfaces gráficas desarrolladas para la aplicación.

5.5. Diagrama Entidad – Relación

La aplicación **Dominous** realiza un almacenamiento limitado de información, por lo que no se estima necesario realizar un diagrama Entidad – Relación con este fin.

5.6. Diagrama de clases de diseño

A continuación se muestra el diagrama de clases de diseño para **Dominous** (figura 5.6 en la página siguiente).

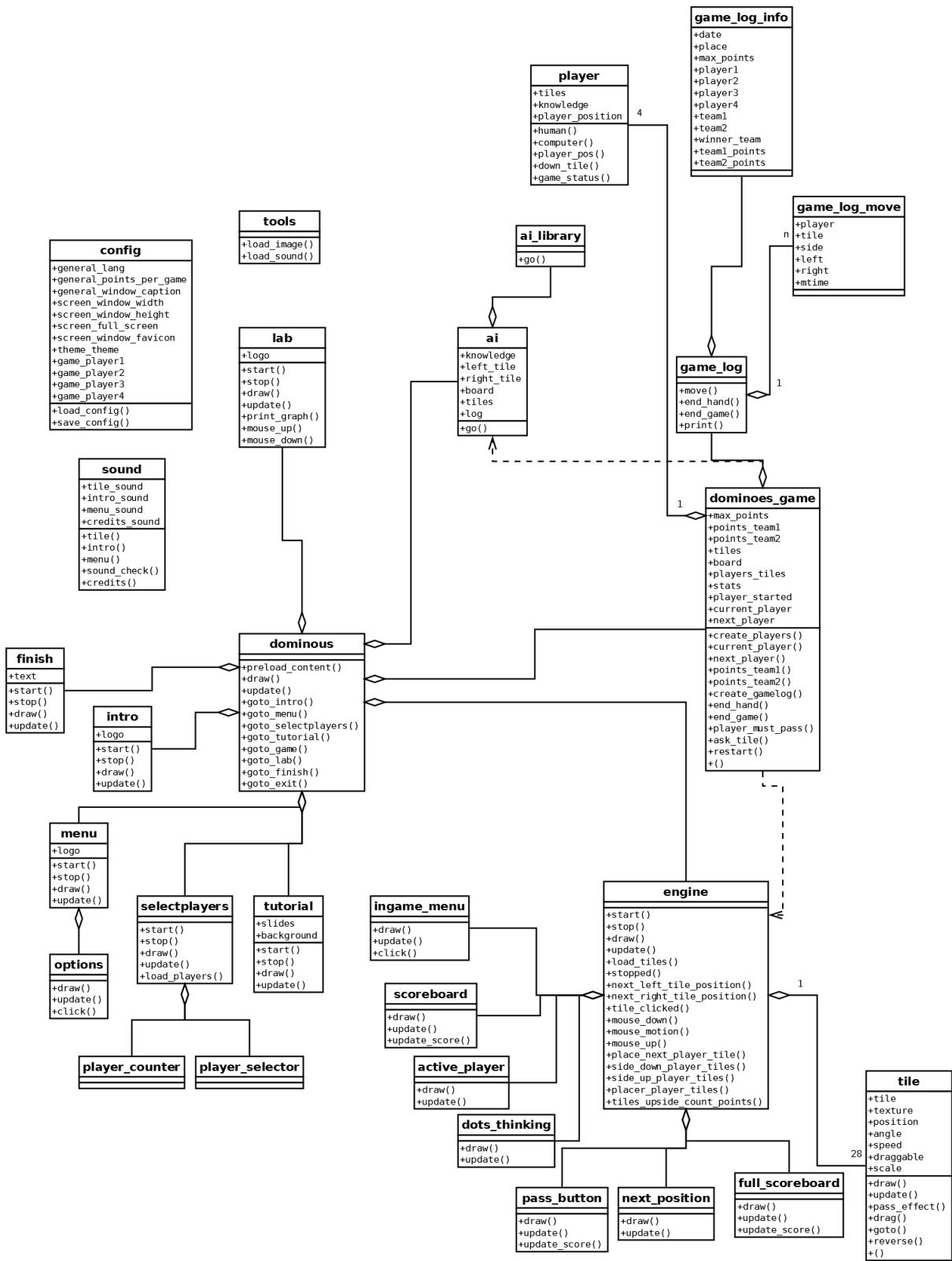


Figura 5.6: Diagrama de diseño del sistema

5.7. Análisis de las principales clases de la aplicación

En este apartado realizaremos un repaso a las principales clases que intervienen en el diseño de **Dominous**, definiendo los métodos y atributos que se requieren para el correcto desarrollo de la aplicación y comentando todos los apartados cuya funcionalidad merezca ser destacada.

5.7.1. Clase dominous

La clase dominous genera el objeto principal que soporta el peso principal de toda la aplicación. Al inicio de la ejecución se crea un objeto de esta clase; este objeto va creando — según las necesidades del flujo que tome la ejecución de la misma — los diferentes objetos, controlando y dando paso a la interfaz activa, llamando a las principales funciones y permitiendo la captura de los eventos de entrada.

Al igual que la gran mayoría de videojuegos, para el desarrollo de Dominous se han utilizado dos métodos principales que permiten controlar la aplicación en tiempo real. Estos dos métodos son:

draw El primer método se encarga de dibujar en pantalla todos los elementos de la interfaz. El funcionamiento básico es recorrer todos los objetos activos y pintarlos según la posición, rotación y escala que tengan en ese preciso momento.

update El segundo método se ocupa de realizar los cálculos para el movimiento de los elementos gráficos. En cada iteración cambiará el estado de los objetos activos, modificando la imagen o los valores posición, rotación y/o escala, estudiando las posibles colisiones, eventos y cualquier otra circunstancia que cambie el estado de la aplicación, para que en el siguiente ciclo se dibujen los elementos en su posición correcta.

Una vez que este objeto principal toma el control de la aplicación, su función es llamar a los métodos `draw` y `update` del objeto encargado de la interfaz que estemos mostrando actualmente, de forma alternativa.

Este juego de llamadas a ambos métodos se realizará hasta un máximo de 60 veces por segundo, dependiendo del rendimiento que se obtenga de los recursos del sistema. De esta forma conseguiremos un máximo de 60 frames o cuadros por segundo, que serán los encargados de generar la sensación de movimiento suave.

Como vemos, el objeto de la clase dominous no realiza ningún dibujado en pantalla ni cálculos, sino que su único cometido es redireccionar el flujo de la aplicación al objeto correspondiente.

5.7.2. Clase dominoes_game

La labor de esta clase es llevar el control de una partida completa de dominó, siguiendo las reglas del **dominó internacional**. Dependiendo de la configuración de la partida, creará los jugadores según la selección que se haya realizado en la pantalla de selección de personajes y comenzará con el bucle principal:

Repartirá las fichas entre los diferentes participantes del juego e irá pidiendo fichas a los jugadores de forma consecutiva, hasta que se de alguna circunstancia de finalización de la mano.

Este bucle se repetirá hasta que alguno de los dos equipos alcance o supere los 200 puntos, momento en el cual la partida se dará por finalizada.

Esta clase también mantendrá activo y actualizado un fichero de registro (*log*) con toda la información generada en la partida. Esta información se utilizará más tarde en el apartado de inteligencia artificial, y almacenará los datos utilizando la siguiente estructura:

Por un lado tenemos la información global de la partida:

date Fecha de desarrollo de la partida.

place Lugar donde tuvo lugar la partida.

max_points Límite superior de puntos necesarios para ganar el encuentro. Como se utilizan las reglas del **dominó internacional**, esta cota estará situada en los doscientos puntos.

player1 Datos del jugador 1.

player2 Datos del jugador 2.

player3 Datos del jugador 3.

player4 Datos del jugador 4.

team1 Pareja que forma el equipo 1.

team2 Pareja que forma el equipo 2.

points_team1 Puntos actuales que tiene el equipo 1.

points_team2 Puntos actuales que tiene el equipo 2.

winner_team Pareja que finalmente ha sido ganadora de la partida.

Y por otro lado tenemos información relativa a la mano que se está desarrollando actualmente. Esta información se repite por cada movimiento que se realice en la partida, y se agrupa por manos:

player Jugador que ha realizado el movimiento.

tile Ficha que ha colocado.

side Lado del tablero por donde ha colocado la ficha.

left Cifra, del cero al siete, que se tiene como resultante del movimiento por el lado izquierdo del tablero.

right Cifra, del cero al siete, que se tiene como resultante del movimiento por el lado derecho del tablero.

mtime Tiempo (en segundos) que ha estado pensando el jugador antes de colocar la ficha.

5.8. Sistemas expertos

En este apartado de la memoria requiere de una especial explicación, ya que a la hora de realizar un videojuego de dominó goza de gran importancia los diferentes sistemas expertos que participarán en la partida. Estos sistemas expertos, como bien su nombre indica, simulan el comportamiento de un ser humano experto en una materia, en este caso del juego de dominó, y su realismo debe hacer sentir al jugador humano que está desempeñando una partida real contra jugadores también humanos.

Enfrentarse a una simulación de inteligencia artificial es una de las empresas más difíciles y complicadas pero a la vez placenteras y satisfactorias de abarcar en un proyecto de Ingeniería del Software. Los seres humanos utilizan multitud de métodos y herramientas para razonar y obtener soluciones a problemas concretos, y en muchas ocasiones este tipo de comportamientos son imposibles de simular, bien por la complejidad computacional que supone, bien por desconocimiento del funcionamiento exacto que sigue el cerebro para obtener esa solución, o bien por limitaciones de recursos, ya sea de disco, tiempo de desarrollo o metodologías que simulen ese comportamiento.

Por otro lado, aunque el público neófito en la materia pueda suponer que el dominó es un juego de suerte, con un abanico muy corto de opciones de juego o con una libertad de movimientos muy limitada, nada más lejos de la realidad.

El dominó es un juego muy profundo, con normas, técnicas y muchas dosis de psicología, un deporte que obliga al jugador a estar concentrado al cien por cien durante todo el desarrollo del juego, y un arte con una curva de aprendizaje poco escarpada, que permite que jugadores nòveles puedan adentrarse en el juego, pero que presenta mucha complicación convertirse en un gran jugador de dominó.

Y, si estas razones que explican la complejidad del mismo no fueran suficientes, no hay más que echarle un vistazo a los torneos locales o de ámbitos más abiertos para descubrir que las parejas de grandes jugadores sabios y con una gran compenetración son los que suelen ganar, descartando de forma tajante la posibilidad de que sea la suerte la que empuje la balanza de la partida hacia un equipo u otro.

A la hora de afrontar este problema, el primer paso fue empaparse de cultura y conocimiento sobre el dominó. Las principales herramientas que se utilizaron fueron:

El Libro del dominó de Benito Ruipérez [RM90], un libro ameno y profundo sobre el mundo del dominó, con multitud de partidas explicadas movimiento a movimiento, trucos, ejemplos, técnicas básicas y avanzadas y mucha información.

Don Manuel Palomo Fernández de Bobadilla –jugador amateur de dominó y participante en torneos durante más de cuarenta años, con un gran conocimiento de técnicas y métodos de juego, mucha experiencia y una gran facilidad para comunicar toda esa sabiduría y conocimiento.

Después de profundizar en el mundo del dominó, y tener cierto conocimiento sobre el mismo podemos intentar definir las pautas sobre las que va a llevarse a cabo la simulación del sistema experto de dominó.

Lo primero que observamos es que la estrategia que siga cada jugador depende enteramente del rol que desempeñe el mismo. Como vimos en el capítulo 2.1.4 es importante definir una estrategia personalizada para cada jugador, dependiendo del lugar respecto al jugador mano.

A continuación mostramos las áreas, campos o ámbitos que debemos tener en cuenta a la hora de resolver con éxito una partida de dominó.

5.8.1. Estrategia según la posición

Lo primero que observamos es que la estrategia que debe seguir cada jugador depende enteramente del rol que desempeñe el mismo. Como vimos en el capítulo 2.1.4 es importante definir una estrategia personalizada para cada jugador, dependiendo del lugar respecto al jugador mano, por lo que nuestro sistema debe tener reglas sensibles a esto.

5.8.2. Reglas de obligado cumplimiento

Si leemos el apartado *Normas fundamentales para el juego del dominó por parejas* de **El Libro del dominó** de Benito Ruipérez [RM90], la primera regla que se destaca es la siguiente:

Defender el doble del palo iniciado por el compañero

Al iniciar, por obligación, el palo más largo que se lleva, se salga o no, sea o no mano. ¿Cuál ha de ser nuestra obligación?

Lo más importante para el éxito de este juego es *volver lo primero* (sin excusas ni pretextos) la que ha iniciado el compañero, aunque sólo lleves una de ellas, seas mano, te quedes fallo o ambas cosas, no importa.

Si tu compañero lleva cuatro de doble, como más normal en el dominó, al volverle una de éstas, lo has hecho *fuerte* en el palo que ha iniciado. Si él ha hecho lo mismo para ti, con otro palo distinto, mandáis en dos palos en esa mano: ¿Qué más se puede pedir? El resto de las fichas que lleváis sirve para contrarrestar a los oponentes en sus palos largos, y ayudar a que el compañero pueda entrar con el suyo: *vuestros palos fuertes*

Como vemos, hay reglas que son de obligado cumplimiento, podríamos decir que *objetivamente*² siempre deben seguirse (sea cual sea la posición del jugador) y, únicamente debemos evitarlas por motivos de fuerza mayor (ausencia de fichas que nos permitan cumplir esa regla).

5.8.3. Pesos de las diferentes reglas

Comparemos la anterior regla con la siguiente, extraída también de **El Libro del dominó** [RM90]:

Si el j1 sale con una ficha que no es doble, y el j2 se dobla a una de ellas, el j3 se debe doblar a la otra, si lleva el doble, de lo contrario, si no lleva el doble, como la que mate por cualquier lado ha de ser de la salida de su compañero, debe poner (cruzar) por el lado de la no doblada.

Podría decirse que existen diferentes reglas que debemos comprobar si se cumplen, y en caso afirmativo, realizar *la más importante de ellas*, es decir, con la que consigamos un mayor distanciamiento o ventaja respecto al equipo contrario.

Por lo tanto nos vemos en la necesidad de establecer pesos o prioridades a cada una de las reglas que se puedan contemplar en cada momento, para que la toma de decisiones esté razonada y sea competitiva.

5.8.4. Aleatoriedad

Uno de los problemas que pueden aparecer a la hora de diseñar este sistema experto es la posibilidad de ser predecible.

² Debemos de reconocer que un recurso válido y muchas veces efectivo a la hora de participar en un juego es utilizar el engaño, pero como norma general lo evitaremos, ya que también estaríamos engañando a nuestro compañero, pudiendo obtenerse resultados nefastos.

Como vemos, los jugadores poseen un conjunto de reglas con prioridades que deben comprobarse y, si se pueden llevar a cabo, realizar el movimiento descrito en ellas. Esto puede llevar a que, dependiendo de los movimientos que realizan los jugadores, saber qué fichas poseen (o qué estrategia están siguiendo).

Una de las formas de evitar este problema es dotar al sistema de cierta aleatoriedad. Una aleatoriedad que no sea completa, sino que, para un determinado momento, elija una entre varias de las mejores opciones. Esto se puede simular (e implementar) de forma sencilla otorgando el mismo peso o prioridad a varias reglas y seleccionando una de ellas. De esta forma los jugadores humanos no podrán saber ni predecir los movimientos de los jugadores controlados por el ordenador, y el sistema experto realizará movimientos que pueden ser igual de buenos pero en diferente orden, añadiendo *humanidad* a la programación de los jugadores.

5.8.5. Generación de errores - sistemas imperfectos

Dominous pretende ser un juego divertido, con el cual aprender a jugar al dominó. Es por ello que una de las prioridades que se buscan es no crear un sistema que sea imposible de vencer, o que ofrezca una resistencia muy fuerte, sino un juego que permita adaptar el nivel del jugador humano al de los jugadores manejados por el ordenador.

Como vemos, el sistema experto debe permitir dos opciones:

- Que los jugadores puedan cometer fallos (simulando por completo el comportamiento de un jugador humano)
- O bien que la descripción de las reglas que siga el jugador (o el conjunto de prioridades que se definen en las reglas) no sea el adecuado para realizar un juego profesional.

Por lo tanto, nuestro sistema experto debe permitir definir cómodamente el conjunto de reglas o los pesos de las mismas.

5.8.6. Diferentes jugadores

Para simplificar y humanizar comportamientos, y representar los diferentes tipos de inteligencias o sistemas expertos, se ha optado por definir un conjunto de jugadores que se podrán elegir como adversario o pareja a la hora de jugar una partida de dominó.

Cada jugador poseerá unas reglas y unos pesos asociados a esas reglas, con la intención de que el jugador humano pueda elegir compañeros y adversarios adaptados al nivel concreto que posea el mismo, y de este modo identificamos de forma sencilla y rápida una *inteligencia* con un avatar definido.

Así pues, nuestro sistema debe permitir la generación de diferentes niveles de dificultad o pilas de reglas y pesos, y asociarlos de forma cómoda a un jugador.

5.8.7. Modelado

Tras realizar un análisis del sistema, se han identificado los siguientes hechos necesarios para que el sistema experto razoné:

Fichas Las fichas con las que cuenta el jugador — en principio todas son posibles candidatas para elaborar el movimiento.

Turno Dependiendo del turno en que nos encontremos.

Jugador que manda Si el jugador que manda es nuestro compañero debemos apoyarle. Si nosotros somos el jugador que manda podemos intentar jugar para nosotros.

Posición del jugador El lugar del jugador, estimando nuestras posibilidades de victoria.

Histórico de la partida En definitiva, se requiere de un histórico con todos los movimientos realizados durante el transcurso de la partida, incluyendo el tiempo *pensando* que ha empleado cada jugador.

En cuanto al motor de inferencia se ha implementado un sencillo motor basado en prioridades de las reglas. Cada jugador (o sistema experto) posee un conjunto de reglas, ordenadas por prioridades, almacenadas en la base de conocimiento. A la hora de mover ficha, se buscará la regla de más peso cuyas condiciones sean satisfechas por la situación de la partida. La regla se dispara y se realiza el movimiento deseado.

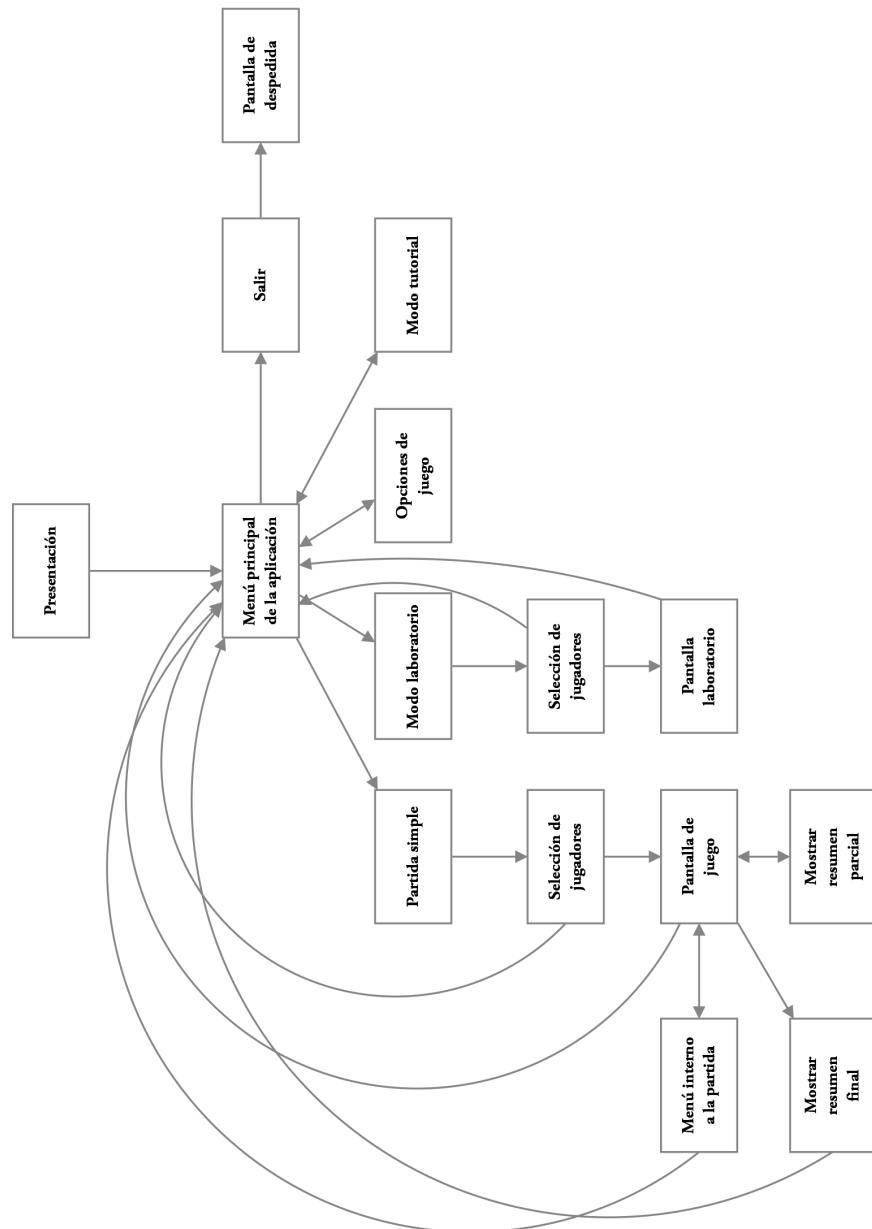


Figura 5.5: Diagrama de interacción entre interfaces

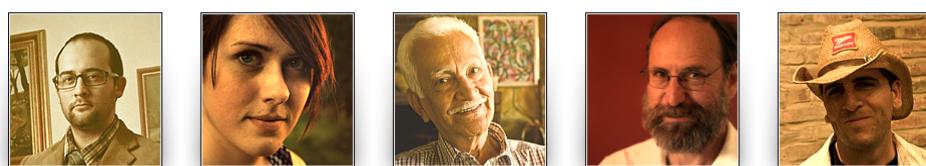


Figura 5.7: Diferentes imágenes que identifican al conjunto de jugadores de Dominous

Capítulo 6

Implementación

6.1. Implementación

Una vez terminadas las dos fases previas — análisis y diseño — es hora de enfrentarnos a la codificación e implementación en sí de la aplicación. Este apartado es largo y entretenido, y va descubriendo a cada momento nuevas problemáticas que no se han tenido en cuenta antes y es necesario resolver, pero también es apasionante, porque vamos viendo de forma empírica cómo nuestra aplicación va tomando forma y se va haciendo tangible a cada momento que avanza.

El desarrollo de Dominous ha presentado muchas características interesantes, problemas, dificultades y otras circunstancias que creemos interesante destacar, y que se pasamos a comentar a continuación.

6.1.1. Entorno gráfico

El entorno gráfico se puede dividir en dos apartados de diferente complejidad: por un lado tenemos la gestión de menús, opciones, pantallas y secciones de la aplicación, y por otro lado tenemos la gestión de una partida de dominó, con todas sus interacciones, animaciones y movimientos de fichas, eventos y demás.

Sistema de secciones

Todo el sistema de secciones lo gobierna el objeto principal de la aplicación, llamado **dominous**. Este objeto posee diferentes objetos como atributos, y estos atributos son las diferentes secciones de la aplicación. Cuando queremos que el flujo de la aplicación pase de una sección a otra no tenemos más que llamarla; al llamarla estaremos ejecutando el siguiente fragmento:

Listado 6.1: Código Python de cambio de sección

```
1 | def goto_tutorial(self):
2 |     self.flow.stop()
3 |     self.flow = self.tutorial
4 |     self.flow.start()
```

Gracias a este tipo de llamadas podemos subdividir cómodamente la aplicación en módulos muy acotados, a los cuales se aplica primero una función de inicialización — por si es necesario que actualicen estructuras, datos o cualquier otra labor de preparación que puedan requerir — y una vez se termina también se ejecuta una función de finalización.

Y mientras el flujo esté asociado a una sección, todas las funciones de eventos, dibujado y actualización de la pantalla se disparan dentro de la sección correspondiente.

Gracias a esta metodología hemos podido acotar de forma muy cómoda las diferentes secciones (cada una con sus peculiaridades) y gestionar de forma independiente cada apartado, evitando posibles problemas colaterales, asegurando una buena integración y minimizando el hecho de que errores de un apartado puedan afectar a otros.

Gestión de partida

La gestión de la partida se puede dividir en dos apartados principales. El primero de ellos es la gestión de los diferentes estados y transiciones en los que se divide la partida, y el segundo es el motor gráfico en sí, que dibuja y mueve las fichas, el tablero y todos los diferentes elementos interactivos que incluye el juego.

La gestión de estados se ha programado con una máquina finita de estados — o autómata finito — ya que este tipo de juegos por turnos se controla de forma cómoda y fácil con este tipo de estructuras. Contamos con un atributo **status** de la clase principal; este estado se va comprobando y actualizando en los diferentes estados, tal y como se ve en la estructura siguiente:

Listado 6.2: Autómata finito de estados

```
1 """
2 Status = 0 - game start, reset all, create players
3     1 - fade in
4     2 - deal tiles
5     if game is over:
6         999 - end game, goto menu
7     if hand is over:
8         99 - end hand
9         100 - move tiles to center
10        101 - show full screen scoreboard
11        goto 1
12 else:
13     3 - draw available positions
14     4 - ask tile next player
15     if next player is human:
16         if player must pass:
17             20 - human player must pass
18             21 - waiting to click pass button
19             22 - human pressed pass button
20             23 - pass effect
21             goto 5
22 else:
23     6 - start drag n drop player tile
24     7 - dragging tile
25     8 - mousedown released
26     if player can place tile here:
27         goto 5
28     else:
29         goto 6
30 else:
31     if computer must pass:
```

```

32         23 - pass effect
33     else:
34         5 - move tile to its place
35     goto 3
36     """

```

Así, siguiendo este diagrama de flujo con las posibles transiciones entre los diferentes estados, podemos definir y acotar los posibles movimientos que se llevan a cabo a lo largo de la partida.

Por otro lado, el motor gráfico que participa de la partida consta de varios apartados y, probablemente, uno de los más costosos y laboriosos fue el del método que decide en qué lugar se coloca una ficha en el tablero.

Como vemos en el gráfico 6.1 existen multitud de posiciones y orientaciones posibles a la hora de colocar la nueva ficha — hasta un total de 20 diferentes posiciones, dependiendo de las siguientes variables:

- Naturaleza de la ficha que vamos a colocar — recordemos que las fichas dobles se colocan de forma perpendicular al sentido del flujo de las fichas, mientras que las fichas que no son dobles continúan con el sentido de las fichas.
- Orientación de la ficha de enganche.
- Sentido de las fichas en el tablero.
- Posición de la ficha de enganche en el tablero respecto a los límites del mismo.
- Número de fichas colocadas en el tablero — si estamos colocando la primera ficha, ésta debe situarse en el centro justo del tablero.
- Tamaño de las fichas y del área de juego.

Para saber dónde colocar las fichas se necesita información de todas estas variables, y dependiendo del caso, optar por una posición u otra. Dado que existen multitud de circunstancias y posibles opciones, se decidió encapsular en un método y realizar todos los cálculos y comprobaciones para luego mover directamente la ficha.

6.1.2. Interfaz de sonido

La interfaz de sonido se ha diseñado de tal forma que se gestione de forma independiente para que su uso sea sencillo, realizando llamadas al objeto **sound** con los métodos que estimemos oportunos en cada momento, y el propio sistema se encarga de realizar diferentes cálculos y comprobaciones:

- Por ejemplo, a la hora de ejecutar un determinado fragmento musical, controla si ya se está escuchando música en ese momento; si es ese el caso realiza un pequeño fundido de la pista actual, e inicia la ejecución de la nueva, evitando así los desagradables clicks que se podrían escuchar cuando se para y se inicia la canción.
- O también, para ejecutar un sonido de ficha de dominó sobre el tablero, el sistema realiza la pre-carga de todos los sonidos — similares pero no iguales — y dispara uno aleatoriamente, evitando que cada apartado tenga que gestionarlo de forma autónoma

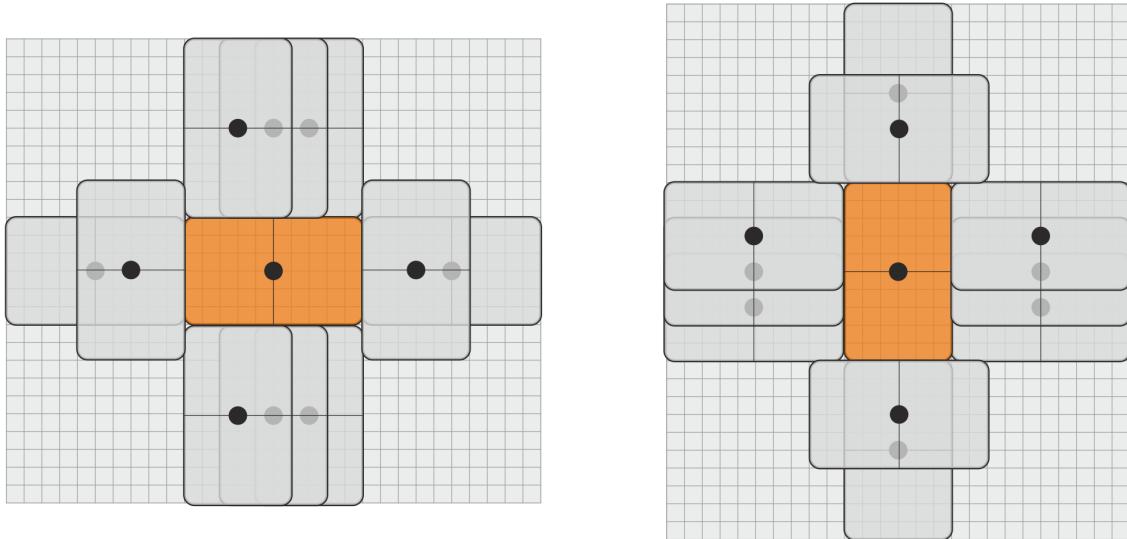


Figura 6.1: Diferentes posiciones en las que se puede colocar una ficha en el tablero

6.1.3. Configuración de la aplicación

La configuración de la aplicación (y almacenaje de la misma) se realiza mediante ficheros de tipo **INI**. Un archivo INI consiste en un simple archivo de texto ASCII que contiene dos tipos de entradas:

Secciones permiten agrupar parámetros relacionados. Por ejemplo: "Parámetros de red".

Valores definen parámetros y su valor. Primero se define el nombre del parámetro y después su valor separado por el signo de igualdad (=).

Comentarios permiten explicar el propósito de una sección o parámetro. Los comentarios comienzan con el carácter punto y coma (;).

El significado de secciones y valores no está bien definido y cada aplicación puede reaccionar de manera diferente ante:

- Secciones duplicadas.
- Parámetros duplicados.
- El carácter de barra invertida (\). A veces se usa para romper una línea en dos.
- Valores. Los valores pueden consistir en texto, números, listas separadas por comas, etc. Esto depende de la aplicación.

Las ventajas de este tipo de ficheros es su estandarización y comodidad a la hora de leer y guardar valores en fichero, de tal forma que podemos guardar información por defecto sobre el juego de la siguiente forma:

Listado 6.3: Fichero INI de ejemplo

```
1 config.add_section('General')
2 config.set('General', 'name', 'Dominous')
3 config.set('General', 'window_caption', 'Dominous')
4 config.set('General', 'lang', 'en')
5 config.set('General', 'points_per_game', '200')
6 config.add_section('Screen')
7 config.set('Screen', 'window_width', '800')
8 config.set('Screen', 'window_height', '600')
9 config.set('Screen', 'full_screen', 'False')
10 config.set('Screen', 'window_fav', os.path.join('images', 'fav.png'))
11 config.add_section('Theme')
12 config.set('Theme', 'theme', 'spanish')
13 config.add_section('Game')
14 config.set('Game', 'player2', 'easy')
15 config.set('Game', 'player3', 'easy')
16 config.set('Game', 'player4', 'easy')
17 # write our configuration file to file
18 with open(file, 'wb') as configfile:
19     config.write(configfile)
```

Y más tarde leerlo simplemente con:

Listado 6.4: Código Python de lectura de fichero INI

```
1 config = ConfigParser.RawConfigParser()
2 config.read(file)
3 config_default = {
4     'name': config.get('General', 'name'),
5     'window_caption': config.get('General', 'window_caption'),
6     'window_width': config.getint('Screen', 'window_width'),
7     'window_height': config.getint('Screen', 'window_height'),
8     'full_screen': config.get('Screen', 'full_screen'),
9     'window_favicon': config.get('Screen', 'window_fav'),
10    'tile_width': 525,
11    'tile_height': 270,
12    'scale': 0.2,
13    'theme': config.get('Theme', 'theme'),
14    'lang': config.get('General', 'lang'),
15    'points_per_game': config.get('General', 'points_per_game'),
16    'gametype': 'human',
17    'gametype_current': 'single',
18    'player1': 'easy',
19    'player2': 'easy',
20    'player3': 'easy',
21    'player4': 'easy',
22    'speed': '1',
23 }
```

Todo ello encapsulado dentro del módulo **config.py** de la aplicación.

Con esto conseguimos un sistema para almacenar la configuración del sistema mediante un método portable, cómodo, sencillo y editable por el usuario mediante cualquier tipo de editor de texto (en caso de que existiera cualquier tipo de problema).

6.1.4. Inteligencia Artificial

En Dominous, el Sistema Experto de Inteligencia Artificial se reparte entre varios módulos, simplificando la implementación de las diferentes partes. Según la estructura clásica en las que se divide un Sistema Experto, podemos ver que cada apartado genera información o conocimiento para la Inteligencia Artificial:

Base de conocimientos Contiene conocimiento modelado extraído del diálogo con un experto — Esta base de conocimiento se almacena, por un lado en la biblioteca de funciones y herramientas para desempeñar una partida, y físicamente se almacena en el módulo de IA

Base de hechos (Memoria de trabajo) contiene los hechos sobre un problema que se ha descubierto durante el análisis — Los hechos se van creando y almacenando gracias al módulo de **Gestión de Partida de Dominó**, que gobierna la partida y guarda toda la información, movimientos, pasos y tiempos al colocar las fichas.

Motor de Inferencia Modela el proceso de razonamiento humano — gracias a la biblioteca de funciones podemos crear un motor de inferencia fácilmente, utilizando pequeñas reglas y dotando de peso o importancia a cada una, modelando exactamente el proceso de razonamiento que queramos, e incluso ofertando la posibilidad de crear cuantos queramos, de una forma fácil y rápida.

Módulos de justificación Explica el razonamiento utilizado por el sistema para llegar a una determinada conclusión — ya que en el dominó no se puede evaluar por sí mismo la efectividad o conveniencia de un movimiento, esta justificación llega del conocimiento del experto en valorar con mayor o menor peso una regla.

Interfaz de usuario es la interacción entre el Sistema Experto y el usuario, y se realiza mediante el lenguaje natural — y, en el caso de Dominous, la interfaz se comunica con el usuario mediante el módulo del motor gráfico del juego, que muestra fichas, movimientos y jugadores.

Pasemos a describir cada apartado para conocer a fondo la implementación del sistema experto.

Base de conocimientos

La base de conocimiento se obtiene desde el módulo **Gestión de Partida de Dominó**, que almacena la información de forma global, esto es, relativa a toda la partida:

Listado 6.5: Información global de la partida

```
1 self.info = {  
2     "date" : date,  
3     "place" : place,  
4     "max_points" : max_points,  
5     "player1" : player1,  
6     "player2" : player2,  
7     "player3" : player3,  
8     "player4" : player4,  
9     "team1" : player1 + " " + player3,  
10    "team2" : player2 + " " + player4,  
11    "description" : description,  
12    "winner_team" : "",  
13    "team1_points" : 0,  
14    "team2_points" : 0,  
15 }
```

El identificador de cada campo es bastante descriptivo

Y también se almacena información de la mano actual y de cada movimiento de la siguiente forma:

Listado 6.6: Información de la mano actual

```
1 movement = {  
2     'player' : player,  
3     'tile' : tile,  
4     'side' : side,  
5     'left' : left,  
6     'right' : right,  
7     'mtime' : mtime,  
8 }
```

Entre los campos almacenados vemos uno que nos resultará de mucho interés en posteriores usos de esta información, y es el campo **mtime**. Este campo almacena el tiempo que ha estado el jugador pensando antes de colocar una ficha, y gracias a que el dominó es un juego de caballeros podemos utilizar esta información para elaborar una estrategia para conseguir la victoria.

Los posibles valores que puede presentar este campo son:

- 0 El usuario no ha requerido tiempo para decidir qué ficha colocaba.
- 1 El usuario ha pensado entre diferentes opciones, y ha necesitado de cierto tiempo para decidir la ficha.
- 2 El usuario ha utilizado una cantidad de tiempo considerable para decidir la ficha a colocar.

Base de Hechos

La Base de Hechos se ha modularizado en pequeñas funciones o reglas, de tal forma que pueden ser fácilmente reutilizadas en el Sistema Experto de cualquier jugador de Dominous. La estructura básica de una regla es la siguiente:

Listado 6.7: Estructura básica de regla

```
1 class nombre_de_la_regla:  
2     def __init__(self):  
3         # inicializamos los atributos que sean necesarios  
4         pass  
5     def go(self, left_tile, right_tile, board, tiles, log):  
6         return ficha, lado, tiempo_pensando
```

Como vemos la estructura es sencilla, implementada mediante clases Python, y tiene una inicialización para definir los atributos que sean necesarios, y por otra parte un método que, para una situación concreta de estado de la partida (tablero, fichas propias, posición del jugador y toda la información generada por la partida).

Veamos un ejemplo práctico. Vamos a definir una regla que intente colocar sobre el tablero una ficha doble, cualquiera que tengamos. Esta regla se implementaría de la siguiente forma:

Listado 6.8: Regla para colocar ficha doble

```
1 | class put_any_double:
```

```

2     def __init__(self):
3         pass
4     def go(self, left_tile, right_tile, board, tiles, log):
5         for item in tiles:
6             if item[0] == item[1]:
7                 if item[0] == left_tile or item[1] == left_tile:
8                     tiles.remove(item)
9                     return item, "left", 1
10                break
11            elif item[0] == right_tile or \
12                item[1] == right_tile or \
13                left_tile == None:
14                tiles.remove(item)
15                return item, "right", 1
16                break
17        return None, "pass", 0

```

En la inicialización no requerimos dar de alta ningún atributo, y el método `go` simplemente comprueba que, para alguna de nuestras fichas dobles, algún lado de la mesa tenga ese mismo valor. Si lo tiene hemos conseguido realizar esta regla, devolvemos el valor y listo. Y si no, devolvemos una señal de que no hemos tenido éxito y salimos.

Gracias a este sistema, simplificamos mucho el desarrollo de nuevos Sistemas Expertos, ya que:

- Es muy sencillo crear nuevas reglas. Como ya vimos en El Libro del dominó de Benito Ruipérez [RM90], muchas reglas, funciones o herramientas se basan en frases o hechos que, si se cumplen, *interesa*¹ cumplirlos. Por poner un caso simple, al comenzar la partida *interesa* comenzar con un doble — o, mejor dicho, hay estrategias interesadas en comenzar con un doble. Para emplear este razonamiento en estas estrategias simplemente tenemos que añadir esta regla con el peso suficiente, y ya lo tendremos implementado.
- Es fácil crear nuevos sistemas, creando pilas de reglas con pesos que se van disparando si pueden cumplirse. Así se van creando diferentes jugadores y dotamos al juego de más variedad y diversión.

Los jugadores poseen una lista de elementos, utilizando normalmente la nomenclatura `self.knowledge`, y van añadiendo las reglas que estimen oportunas como si fuera una lista clásica de Python, utilizando el método `append`.

Si estamos desarrollando un jugador simple, podemos definir su base de conocimiento de la siguiente forma:

Listado 6.9: Base de conocimiento simple

```

1 self.knowledge = []
2 self.knowledge.append([put_any_double()])
3 self.knowledge.append([put_anyone()])

```

Como vemos, estamos definiendo que el jugador intente, inicialmente, colocar una ficha doble. Si esto no es posible, intentará colocar una ficha cualquiera.

¹Como vemos, debemos ser cuidadosos al emplear verbos como *interesa* o *debemos* al hablar de colocar fichas. El juego del Dominó es lo suficientemente complejo y goza de muchas estrategias como para poder simplificarlo o decir objetivamente que *debemos* realizar una acción.

Así, con solamente tres líneas podemos comenzar a desarrollar una inteligencia artificial a nuestro gusto, que juegue y valore cada regla con la importancia que estimemos oportuna.

Por otro lado, también podemos utilizar reglas con el mismo peso, dotando de aleatoriedad al jugador y permitiendo que se comporte de diferente modo en situaciones similares. De este modo, si queremos definir a un jugador un poco más complejo, su base de conocimiento podría escribirse así:

Listado 6.10: Base de conocimiento compleja

```
1 self.knowledge = []
2 self.knowledge.append([
3     put_any_double(),
4     weight_matrix()
5 ])
6 self.knowledge.append([put_anyone()])
```

En un primer momento se ejecutará, aleatoriamente, una regla entre **put_any_double()** y **weight_matrix()**. Si no hay suerte con la primera, se intentará con la segunda, y si no hay éxito con ninguna se ejecutará el siguiente conjunto de reglas — en este caso, **put_anyone()**, es decir, intentar colocar una ficha cualquiera.

El Motor de Inferencia se desarrolla de forma muy sencilla, ya que se utilizan estructuras muy simples proporcionadas por el mismo lenguaje Python.

Motor de Inferencia

El Motor de Inferencia es el encargado de elegir una regla de entre todas las reglas definidas en la Base de Conocimiento del jugador concreto. Este motor, antes de evaluar cualquier regla, realiza diferentes comprobaciones para acelerar los tiempos de cálculo en ciertos casos especiales. Así, las reglas no se evaluarán bajo las siguientes circunstancias:

- El jugador no puede colocar ninguna ficha — si ninguna de las fichas que posee el jugador son candidatas a utilizarse en el tablero, el Motor de Inferencia pasará automáticamente el turno al siguiente jugador.
- El jugador solo puede colocar una ficha — en el caso de que solamente una ficha sea candidata a colocarse en juego, el sistema la colocará automáticamente, dando la información de que el jugador ha estado poco tiempo pensando qué ficha colocar, y pasará el turno al siguiente jugador.

Solamente en el caso de que más de una ficha sea candidata a colocarse, el Motor de Inferencia hará uso de la Base de Conocimiento y aplicará las reglas del sistema experto.

6.1.5. Música y efectos

La música también juega un papel importante en un videojuego, y no debe ser elegida a la ligera, sino debe buscarse una razón y un estilo que encaje con los objetivos definidos. Esta aplicación está orientada a un público muy general, y el ritmo del juego es tranquilo y pausado, así que se eligió el jazz como estilo base para acompañar al juego.

Gracias a la enorme comunidad de músicos que publican sus obras con *licencias libres* — permitiendo que sus obras se utilicen y se adapten a otros entornos — no fue nada complicado buscar un conjunto de canciones que se adaptaran al ritmo y velocidad que se disfruta en Dominous.



Figura 6.2: La música se obtuvo de Jamendo, un servicio de música libre

En cuanto a los efectos de sonido la cosa estuvo un poco más complicada. Al inicio se buscaron sonidos que pudieran encajar con las acciones de colocar la ficha, y se probó con efectos de sonido clásicos de click — sonidos cortos que informan de una acción — pero se descubrió que, aunque dotaban de la suficiente información al usuario, no quedaban bien con la estructura y estilo de Dominous.

La solución que se tomó fue grabar sonidos reales de fichas de dominó golpeando una mesa. Una vez se contó con un buen grupo de sonidos, se eligió uno como candidato un golpe seco, limpio y con sonoridad. Pero descubrimos que escuchar siempre el mismo sonido era molesto para el usuario (y muy poco realista), así que se optó por mejorar la solución final: elegir un gran conjunto de sonidos diferentes de fichas de dominó y ejecutar aleatoriamente uno de ellos cada vez que la ficha golpeaba la mesa.

Así conseguimos una sonoridad y aleatoriedad agradable para el usuario.

Capítulo 7

Pruebas

7.1. Pruebas

La fase de prueba es de las más importantes de un proyecto software [Mye04]. El objetivo de este paso es, como razón última, la verificación de que el proyecto cumple con todos los requisitos iniciales que se plantearon al comienzo del desarrollo. Según la metodología clásica de desarrollo de pruebas existen diferentes enfoques a la hora de realizar este tipo de pruebas de software, siendo todos complementarios [Bei90], pero para el caso concreto que tenemos entre manos debemos tener en cuenta la naturaleza del proyecto.

Los videojuegos presentan dos facetas distintas que deben ser abordadas de diferentes maneras cuando se realiza la fase de pruebas:

- Por un lado tenemos las pruebas clásicas que se realizan a cualquier desarrollo de software, como pueden ser las pruebas unitarias o de integración, destinadas a verificar el correcto funcionamiento del código.
- Y por otro lado, al requerir de una interacción directa con el usuario (y estar destinado a divertir, entretenir y proporcionar un rato ameno al mismo) se deben realizar varios tipos de pruebas orientadas a comprobar que se cumplen requisitos más abstractos como puede ser el testeo o análisis de la jugabilidad o usabilidad de la aplicación, interfaces, medir el entretenimiento que proporciona la aplicación (relacionado directamente con el desarrollo de la inteligencia artificial de los contrincantes, entre otros)

El primer conjunto de pruebas se pueden realizar con comprobaciones de código, compilación (o en este caso concreto, interpretación) del mismo y diferentes métodos, pero el segundo conjunto de pruebas es necesario que se desarrolle con diferentes tipos de usuarios reales manejando la aplicación y realizando cuestionarios que nos ayuden a valorar el éxito o fracaso de estos apartados.

Para los apartados de pruebas unitarias y de integración se va a trabajar principalmente con los tres módulos que forman el núcleo fuerte de la aplicación (y sobre las que cae el peso de la misma):

- Sistema de gestión de partida de dominó
- Sistema de inteligencia artificial
- Motor gráfico de la aplicación

7.1.1. Pruebas unitarias

Durante la fase de implementación se fueron realizando pruebas unitarias no automatizadas de cada conjunto o subconjunto de módulos que se iban desarrollando, evitando así encontrar errores en las pruebas de integración que no sean propiamente de integración sino de errores en la codificación de los diferentes módulos.

Una herramienta cuyo uso ha resultado muy interesante y útil ha sido **PyFlakes**. Esta herramienta es un programa que analiza otros programas escritos en Python y detecta un gran conjunto de errores analizando el código fuente. Existen otras herramientas de análisis automático de código Python, como PyChecker o PyLint, pero se decidió utilizar PyFlakes principalmente por la rapidez de análisis y seguridad ante caídas, ya que no requiere ejecutar el código, sino que realiza un análisis mediante *parseo* del código.

En cuanto a los diferentes módulos que conforman Dominous, existen varios casos inherentes a cada módulo que deben tratarse de forma individual (ya que dependen de la naturaleza del módulo) y de forma manual (ya que no se puede abordar su solución de forma automática).

Sistema de gestión de partida de dominó

En este módulo las pruebas unitarias están claras: el módulo debe velar por el correcto cumplimiento de todas y cada una de las reglas del dominó internacional. Para ello debe vigilarse cada movimiento de los jugadores, repasando el estado del juego y las diferentes posibilidades que tiene el jugador, entrándose en modo error cuando se realiza una acción ilegal.

En este momento del desarrollo se tuvo que tomar una decisión respecto a cómo contemplar los errores o incumplimiento de normas que puedan producirse (de forma intencionada o no) por parte de los jugadores:

- Una opción era permitir que los jugadores incumplan las normas y actuar en consecuencia contra el jugador. En ningún momento se puede permitir que los jugadores coloquen fichas en lugares donde está prohibido ese movimiento, por lo que, de todas las normas que posee el dominó las únicas candidatas a entrar en este grupo son las que hacen del dominó un **juego de señores**, es decir, todas aquellas que están destinadas a dotar de información a los demás jugadores de forma obligatoria (como puede ser la falta de un palo mediante acortar el tiempo que transcurre en su turno).

Para esta opción teníamos nuevamente otras dos opciones:

- Podemos permitir y *mirar hacia otro lado*, permitiendo que los jugadores engañen de forma clara a los demás jugadores, incluyendo compañeros, o
 - Podemos actuar como jueces y penalizar a los jugadores que cometan este tipo de faltas.
- La otra opción es no permitir este tipo de acciones, volviendo hacia atrás en la acción del jugador y pidiendo de nuevo que actúe, hasta que la acción satisfaga las reglas del juego del dominó.

Finalmente se decidió que, dado que esta aplicación tiene como requisito el favorecer el aprendizaje del juego del dominó, no existe cabida alguna a jugadas deshonrosas o que puedan llevar a equivocación al usuario que desea aprender a jugar.

Por lo tanto, cuando un jugador intenta realizar una acción equivocada, se deshace esa acción y se le pide de nuevo que mueva ficha hasta que ese movimiento sea correcto, momento en el cual se continúa con naturalidad la partida.

Sistema de Inteligencia Artificial

El juego del dominó es un juego con un amplio universo de acciones y con conocimiento limitado de la situación de la partida, por lo que no se puede estimar que una acción sea óptima (puede haber varias opciones válidas). Pero a pesar de esto sí existe un requisito que debe cumplirse (y se exige el cumplimiento) a la hora de realizar un movimiento, y es o pasar o colocar una ficha cualquiera que pueda colocarse.

Esta acción se definió y se tomó como opción por defecto (también llamado fallback) en caso de que todas las demás reglas de Inteligencia Artificial no se cumplan, y antes de que el sistema quede colgado o no se pueda continuar, se toma esta acción y se continúa con la partida.

7.1.2. Pruebas de integración

Según se iban desarrollando módulos y éstos cumplían las pruebas unitarias desarrolladas para estos apartados, era necesario integrar los diferentes módulos para corroborar y contrastar el correcto funcionamiento de la conjunción de ambos módulos.



Figura 7.1: Esquema de relaciones entre módulos para las pruebas de integración

Sistema de Gestión de Partida de Dominó y Sistema de Inteligencia Artificial

Una vez definida la interfaz de comunicación entre el Sistema de Gestión de Partida de Dominó y Sistema de Inteligencia Artificial, para comprobar la consistencia de esta integración se desarrolló un jugador de dominó muy simple que debía cumplir con dos premisas:

1. El jugador debe cumplir con éxito las especificaciones de la interfaz.
2. El jugador debe poner una ficha cualquiera (si puede colocar una ficha) o debe pasar (si no existe la posibilidad de mover).

Durante el desarrollo del Sistema de Gestión de Partida de Dominó se fue comprobando el éxito de la comunicación entre ambos módulos, de tal modo que cuando se realizó la integración con los dos sistemas totalmente finalizados no se encontraron problemas graves, obteniéndose una integración suave y sin errores costosos de solventar.

Sistema de Gestión de Partida de Dominó y Motor gráfico de la aplicación

La integración fue más costosa que la comentada anteriormente, ya que no se pudo realizar un sistema de pruebas que cumpliera las especificaciones de integración, pero gracias a que el Sistema de Gestión

de Partida de Dominó contaba con un gran número de métodos de todo tipo para obtener información del estado del tablero, jugadores y otros, se pudo realizar la integración de forma sencilla y sin retrasarnos en demasiada en la estimación de tiempos.

7.1.3. Pruebas de Jugabilidad, usabilidad y experiencia de usuario

La jugabilidad en Dominous abarca dos ámbitos diferenciados: por un lado se desea que el juego transcurra fluido, sin parones bruscos o bajadas en la velocidad de cálculo y dibujado de la pantalla, y por otro lado que el jugador, al mover fichas y desenvolverse por el juego y los menús.

El desempeño o velocidad de la aplicación está asegurado por varios factores:

- Utilización de una librería de bajo nivel como es PyGame, cuya velocidad está garantizada y testeada en profundidad.
- La naturaleza propia de la aplicación, que se desarrolla a un ritmo bajo, no necesita de grandes cálculos ni uso avanzado de la CPU.
- El motor de inteligencia artificial es rápido, y en casos en los que las posibilidades del usuario se limiten a pasar o colocar una única ficha, el motor salta todas las comprobaciones de reglas que sabe que van a fallar — redundando así en una mayor velocidad y rendimiento.

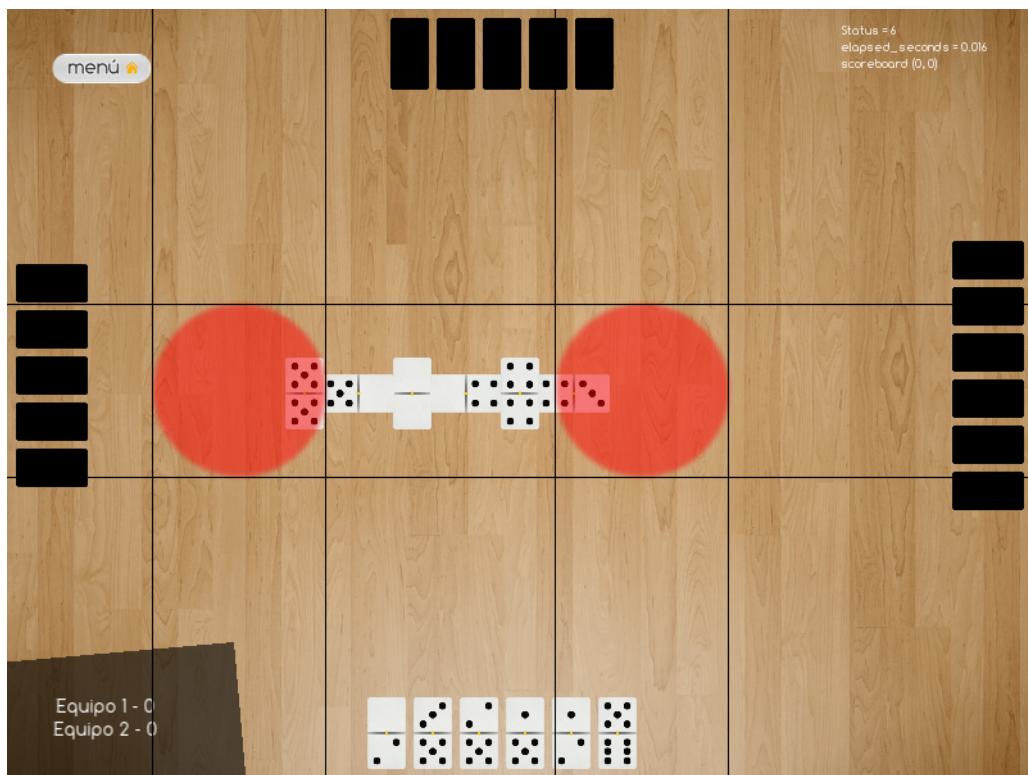


Figura 7.2: Superficie sobre la cual el usuario puede dejar la ficha satisfactoriamente

La jugabilidad de la aplicación, dado que la acción más común que realizará el usuario será mover las fichas, se trabajó haciendo que el movimiento de arrastrar y soltar la ficha sea satisfactorio, cuidando que la ficha siga al cursor de forma fiel, el error de soltar una ficha en un lugar equivocado no sea frustrante

— sino que se convierta en un hecho más de la partida —, que la zona donde el usuario suelta la ficha sea grande y no se produzcan errores no deseados y funcionalidades y pequeñas mejoras que buscan que el jugador se divierta con la aplicación.

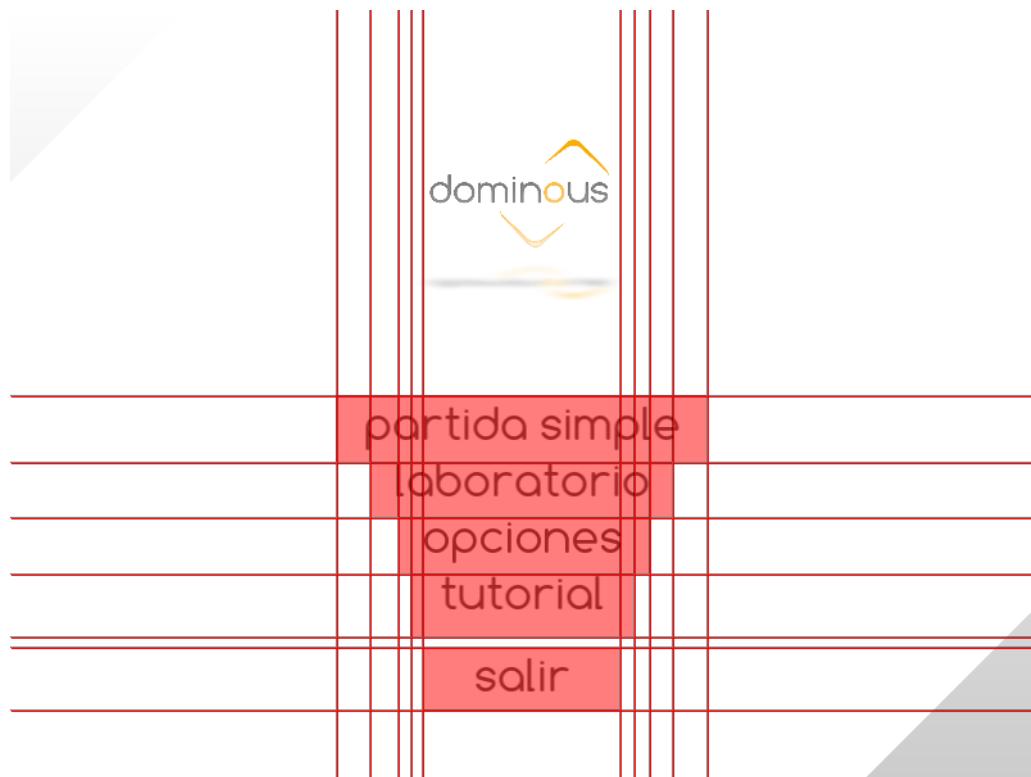


Figura 7.3: Como vemos, la superficie en la cual el usuario puede realizar click de forma satisfactoria es más amplia que el simple texto que incluye

Dado que la aplicación está orientado a un público adulto — que, probablemente, no goce de una habilidad muy desarrollada con los dispositivos de entrada — la zona de click sobre menús y demás elementos de la interfaz se ha agrandado, con lo cual las posibilidades de éxito del click aumentan, así como la satisfacción del usuario con la herramienta.

Por último, se han realizado numerosas pruebas con usuarios reales durante el desarrollo del proyecto, recibiendo feedback real y realizando y estudiando pequeñas modificaciones que hagan que el mismo disfrute de la experiencia de jugar a este producto.

Los usuarios sobre los que se han realizado las diferentes pruebas son los siguientes:

Sujeto 1 Mujer, 28 años, estudios universitarios de la rama de letras, nivel de informática usuario alto, con conocimientos de ofimática, poca experiencia en videojuegos, bajo nivel de las reglas y normas del dominó internacional.

Sujeto 2 Hombre, 63 años, estudios universitarios de economía, nivel de informática usuario medio, nula experiencia en videojuegos, jugador experimentado de dominó.

Sujeto 3 Hombre, 30 años, estudios universitarios de Informática, alto nivel de informática usuario, amplia experiencia en videojuegos, nivel medio de conocimiento del juego de dominó.

Sujeto 4 Hombre, 31 años, estudios universitarios de Informática, alto nivel de informática usuario, amplia experiencia en videojuegos, nivel bajo de conocimiento del juego de dominó.

Se analizó el uso de la aplicación por parte de los sujetos con las siguientes circunstancias:

- Todas las pruebas se realizaron en un ordenador portátil con ratón, ejecutándose la aplicación en un entorno Windows 7.
- Se les proporcionó la aplicación ya en ejecución, mostrándoles la pantalla de inicio.
- No se les facilitó ningún tipo de información adicional sobre cómo funciona la aplicación.

Por otra parte, los objetivos a cumplir eran los siguientes:

Objetivo 1 Comenzar una partida simple, jugar una mano, salir al menú principal y cerrar la aplicación.

Objetivo 2 Cambiar el tema gráfico, seleccionando el tema **fruits** comenzar una partida simple, jugar una mano, salir al menú principal y cerrar la aplicación.

Objetivo 3 Seleccionar el modo **solo computadora** en velocidad rápida, visualizar una mano, salir al menú principal y cerrar la aplicación.

Objetivo 4 Entrar en **laboratorio**, hacer que compitan dos equipos diferentes y comentar con el entrevistador cuál de los dos equipos ha conseguido más victorias.

Y los éxitos de objetivos y resultados obtenidos se detallan a continuación:

Objetivo 1 Todos los sujetos fueron capaces de completar con éxito el objetivo 1. Cabe destacar que los sujetos tres y cuatro salieron al menú principal activando el menú de juego con la tecla **escape**, y el sujeto cuatro salió de la aplicación cerrando directamente la ventana de la misma, sin hacer uso del botón salir que se encuentra en el menú principal de Dominous.

Objetivo 2 Todos los sujetos fueron capaces de completar con éxito el objetivo 2, aunque el usuario dos necesitó de información acerca de qué es un tema gráfico — la información se le proporcionó debido a que el bajo nivel de conocimiento informático no permitía el éxito de este objetivo.

Objetivo 3 Todos los sujetos fueron capaces de completar con éxito este objetivo, pero se produjeron los siguientes hechos meritorios de ser comentados:

- El usuario dos se sintió desconcertado cuando se iban produciendo los movimientos automáticos de los jugadores, y preguntó que qué utilidad podía tener ver una partida a esta velocidad y sin participar.
- El usuario uno se frustró y preguntó cómo podía salir de la aplicación, a pesar de que la interfaz para volver al menú principal era la misma que en el modo de juego simple.

Objetivo 4 Los usuarios uno, tres y cuatro completaron con éxito este objetivo. El usuario dos no fue capaz de comprender el significado de la gráfica o de las barras, y no entendió el uso que podía tener el enfrentar a dos equipos si no puedes ver el transcurso de las partidas.

Capítulo 8

Conclusiones

8.1. Resultados obtenidos

Los resultados obtenidos gracias al desarrollo de esta aplicación han sido tremadamente satisfactorios, tanto en el plano personal como en el profesional.

Como hitos conseguidos podemos destacar los siguientes:

8.1.1. Primer simulador de dominó libre

Sin querer resultar pretencioso — y siempre según la investigación previa que hemos realizado — Dominous es el primer videojuego de dominó internacional bajo licencia libre, con todas las ventajas que ello conlleva:

- Posibilidad de que pueda ser mejorado por la comunidad, añadiendo soluciones a errores de código o refactorizaciones, proporcionando nuevos jugadores, mejorando el sistema y motor de Inteligencia Artificial, añadiendo nuevas funcionalidades que se encuentran fuera de los requisitos iniciales del proyecto, portar Dominous a otros entornos, Sistemas Operativos o incluso consolas de videojuegos...
- Libertad para aprender mediante la lectura de la documentación del proyecto o del análisis del código.
- Facilidad para llegar a más usuarios gracias a la gratuidad del producto y a la disposición del mismo en Internet.

El acto de volcar todo el conocimiento de esta aplicación a la comunidad — código, documentación o arte, por ejemplo — es desde mi punto de vista un acto lógico y natural de intentar devolver a la comunidad del software libre una pequeña parte de todo lo que se ha obtenido de ella. Con un pequeño repaso a las herramientas utilizadas podemos comprobar que la gran mayoría de ellas tienen licencias libres, y gracias a ellas hemos podido desarrollar esta aplicación con éxito, pudiendo elegir entre un gran número de diferentes opciones y tomando la que mejor nos conviene en cada momento.

8.1.2. Aplicación de lo aprendido en la carrera

Existe una creencia extendida sobre que los estudios universitarios técnicos suelen pecar de facilitar conocimientos que resultan muy teóricos, con muy poca aplicación práctica a la hora de desarrollar un proyecto real, así que es muy gratificante poder comprobar que gracias todas las asignaturas de la carrera se consigue un fuerte bagaje y una metodología de trabajo que nos permite llevar a cabo proyectos de

gran envergadura como este.

La principal asignatura que nos ha permitido solventar los principales problemas de este proyecto ha sido la de Inteligencia Artificial, impartida por el profesor Don Ignacio Pérez Blanquer, en especial el apartado de análisis de juegos. Los apartados sobre representación del conocimiento y la base teórica sobre sistemas expertos también han resultado de gran ayuda.

Por otro lado, gracias a las asignaturas relacionadas con la Ingeniería del Software se han podido definir las líneas maestras que gobiernan el desarrollo de la aplicación, afrontando el proyecto con mucha más seguridad y confianza en la buena terminación del mismo.

Y por supuesto, no podemos olvidar las asignaturas de Análisis y Diseño de Algoritmos, Estructura de Datos, Programación orientada a Objetos, Sistemas Operativos y tantas otras que, además de enseñar conocimientos, enseñan cómo pensar y cómo aprender, lo cual es mucho más importante que el simple hecho del conocimiento en sí o aprobar simplemente la asignatura.

8.1.3. Aprendizaje del juego de dominó

Debo reconocer que, cuando mi tutor de proyecto me presentó esta idea, creía que el dominó era un juego sencillo y fácilmente abarcable con las herramientas de Inteligencia Artificial que conocemos hoy día. Y nada más lejos de la realidad.

Al principio pensaba que el dominó se limitaba a gestionar tus fichas dentro de un radio de acción muy limitado — ya que, comparado con otros juegos como el ajedrez, en éste no podemos colocar o mover la ficha que queramos, sino solamente mover una del pequeño subconjunto formado por las fichas candidatas a colocarse — pero mediante la lectura de libros de estrategias y partidas comentadas empecé a ver las dificultades que entraña el jugar una buena partida de dominó. El dominó es un juego muy profundo, con una componente de suerte mucho más pequeña de lo que la gente cree, mucha psicología, estrategias, tácticas, y sobre todo mucha, mucha diversión.

El dominó es un juego social y de caballeros, que bien jugado afianza y genera confianza y compañerismo entre los jugadores y parejas. Es un juego con reglas simples pero estrategias complejas, que fortalece la memoria (recordando o intentando recordar los diferentes movimientos que ha realizado cada jugador) y cuya diversión se dispara al contar con un compañero, ya que el juego en equipo — sin posibilidad de comunicación entre ellos — añade mucha complejidad y humanidad a un juego que dista muchísimo de estar basado simplemente en estadística o probabilidades.

Y gracias a todo lo aprendido ahora puedo observar el mundo del dominó con otra mirada y, por supuesto, respetar y admirar a los grandes jugadores de dominó.

8.1.4. Primer proyecto que realizo de manera íntegra

Para un estudiante de Ingeniería Informática es una gran satisfacción realizar un proyecto de manera completa, para lo cual se han tenido que aplicar conocimientos de muchas ramas, y ha sido muy divertido mezclar la programación pura y dura de la aplicación con conocimientos más amplios como pueden ser:

Diseño gráfico A la hora de hacer un videojuego un apartado tremadamente importante es el aspecto visual del mismo. En multitud de ocasiones los ingenieros informáticos caemos en la trampa de hacer aplicaciones que solventan los problemas concretos, pero que fallan a la hora de ser

atractivos para el usuario final. Y si no hay usuario final o no le resulta atractivo y no lo usa, la aplicación no tendrá sentido.

Para diseñar el aspecto visual se han definido unas líneas maestras generales y se ha intentado ser consistente con este estilo visual, para dotar de uniformidad y coherencia a la aplicación; los botones tienen el mismo diseño durante toda la aplicación, la tipografía utilizada se limita únicamente a dos fuentes, la paleta de color también se mantiene constante, etcétera.

Por otro lado, ya que no somos estudiantes de bellas artes ni se puede presuponer que un ingeniero informático debe tener conocimientos de diseño gráfico, he optado por una solución que ha dado muy buenos resultados, y es el utilizar una interfaz y diseño minimalista, con líneas y fondos claros, ciñéndonos a jugar con tonos de color blanco, gris, negro y amarillo — para dotar de una fuerte uniformidad de estilo — y sin buscar en ningún momento la espectacularidad gráfica o visual. Gracias a esta solución hemos conseguido que la aplicación no resulte confusa ni pesada, sea legible y tenga un aspecto gráfico bastante profesional.

Música y efectos de sonido La música también juega un papel importante en un videojuego, esta aplicación está orientada a un público muy general, y el ritmo del juego es tranquilo y pausado, así que se eligió el jazz libre como estilo base para acompañar al juego. En cuanto a los efectos de sonido, se grabaron golpes reales de fichas de dominó sobre una mesa y aleatoriamente se producen al poner fichas en el juego.

Gestión de proyectos Para gestionar satisfactoriamente el proyecto se contó con herramientas profesionales como sistemas de control de versiones (SVN), aplicaciones de ticketing para asignar y crear tareas, una página web en la que volcar la información y documentación que se iba generando, aplicaciones de screencast para animar a la comunidad a participar o Latex y Dia para diseño de documentación y diagramas, entre otros muchos.

Usabilidad y Experiencia de Usuario Se ha intentado aplicar mucho sentido común e iteraciones al diseño de la interfaz de la aplicación, para que su facilidad de uso sea un aliciente para usarlo, y no un escollo que dificulte el acceso a ella.

8.2. Trabajos futuros

A la hora de establecer los requisitos técnicos de un proyecto fin de carrera se puede caer en la tentación de querer desarrollar y especificar un proyecto que no sea realista con las limitaciones técnicas, de tiempo y de recursos que se disponen para la realización del mismo.

Este acto puede resultar más flagrante en el caso de que el alumno se sienta muy atraído con el mismo — como es el caso que nos ocupa en este momento — ya que el placer y el interés de desarrollar un producto profesional, similar a los que se pueden encontrar en el mercado, puede llevarnos a una gran frustración por no conseguirlo. Debemos tener en cuenta de que los estudios profesionales de videojuegos hoy en día cuentan con una gran cantidad de recursos, tanto de tiempo como económicos, además de aglutinar profesionales de diferentes áreas (como puede ser el diseño gráfico, directores de arte, expertos en usabilidad o programadores de bajo nivel, por nombrar algunos) y querer conseguir un producto de condiciones similares es disparatado.

Es por estas razones que varias funcionalidades han debido de quedarse — lamentablemente — fuera del pliego de especificaciones de requisitos. Aún así nos gustaría comentarlas a continuación, para que, por un lado, quede constancia del trabajo de estudio preliminar del sector que se ha realizado y, por otro lado, por si alguien quiere recoger el guante y continuar añadiendo nuevas funcionalidades al proyecto, mostrar un conjunto de pasos o características muy interesantes de implementar:

8.2.1. Juego en red

En la fase de especificación de requisitos se barajó la opción de que Dominous presentara opciones para desarrollar una partida en red, pero después de un profundo análisis se desechó esta opción; el juego en red plantea varias problemáticas que resultan verdaderamente difíciles y que explicamos a continuación:

- La principal razón es que existe el problema conceptual de desarrollar una partida a un juego con conocimiento parcial de forma remota — la base del dominó reside en que los jugadores no pueden conocer las fichas de sus compañeros, y si la partida se juega en red nada impide que los equipos intercambien información mediante otras vías de comunicación.

Se podría argumentar que depende del jugador y de la moral y ética que lo gobierne el utilizar estas trampas para conseguir una victoria de forma poco correcta, y que el videojuego no debe entrar en este terreno, pero es cierto que dejar una puerta abierta a este tipo de jugadores es crear un juego con un gran fallo conceptual, que no permitiría las principales premisas de la aplicación, que son entretenir y enseñar el mundo del dominó.

- Tampoco hay que olvidar que, al ofrecer este tipo de juego, estamos cambiando los objetivos principales del juego, que es conseguir profundizar en el mundo de la Inteligencia Artificial orientado al dominó.
- Finalmente tampoco podemos obviar las limitaciones técnicas y de escasez de mercado, que nos obligaría en principio a tener que desarrollar la aplicación para otro tipo de medios con más proyección — como podría ser dispositivos móviles, aplicaciones en flash o directamente orientado a web — que obligaría a reorientar todo el desarrollo del proyecto.

8.2.2. Servidor web para compartir sistemas expertos

También sería muy interesante poder ofrecer una forma de disponer de los diferentes sistemas expertos de forma remota. Así sería sencillo poder realizar torneos entre diferentes jugadores y equipos, cada uno programado por un participante del torneo.

Para ello sería necesario buscar un formato estándar de compartición de sistemas expertos basados en Dominous, desarrollando unas especificaciones lo suficientemente generales para dar potencia a la Inteligencia Artificial, y relativamente simple para que no suponga una curva de aprendizaje agresiva al nuevo jugador.

Todo esto sin olvidar que es necesario que la ejecución de los sistemas expertos se realice en la misma máquina; ya que si se facilita una API para comunicación con otros sistemas estaríamos cayendo en el mismo error que en el juego en red, esto es, permitiendo que equipos pudieran utilizar canales de comunicación alternativos y jugar con ventaja frente a otros.

8.2.3. Analizador de partidas con aprendizaje automático o semiautomático

Existen multitud de herramientas de Inteligencia Artificial orientadas a simular sistemas expertos, comportamientos y razonamiento humano. Una de ellas es el **aprendizaje automático**; sería un proyecto muy interesante generar un sistema que reciba como entrada ficheros con históricos de partidas — como las que genera Dominous, por ejemplo — y obtenga información mediante el análisis de cada movimiento, dependiendo de la posición del jugador en la mesa y de las fichas existentes en cada mano.

Para conseguir este fin se podrían utilizar técnicas como las redes neuronales o árboles de decisión. Y si se quiere dar un paso más se podría apoyar en aprendizaje supervisado — utilizando a algún experto real en el mundo del dominó que evalúe los posibles movimientos, animando o corrigiendo según la valoración del mismo y creando un sistema que vaya mejorando con cada uso.

8.2.4. Migración de la aplicación a otros sistemas

Dominous se ha programado con los requisitos de correcto funcionamiento bajo sistemas operativos basados en GNU/Linux (como Ubuntu) y Microsoft Windows, pero sería un proyecto interesante intentar migrar el sistema a otros entornos como podrían ser dispositivos móviles, consolas portátiles o tablets.



Figura 8.1: Dominous ejecutándose en un entorno GNU/Linux, distribución Ubuntu 10.04

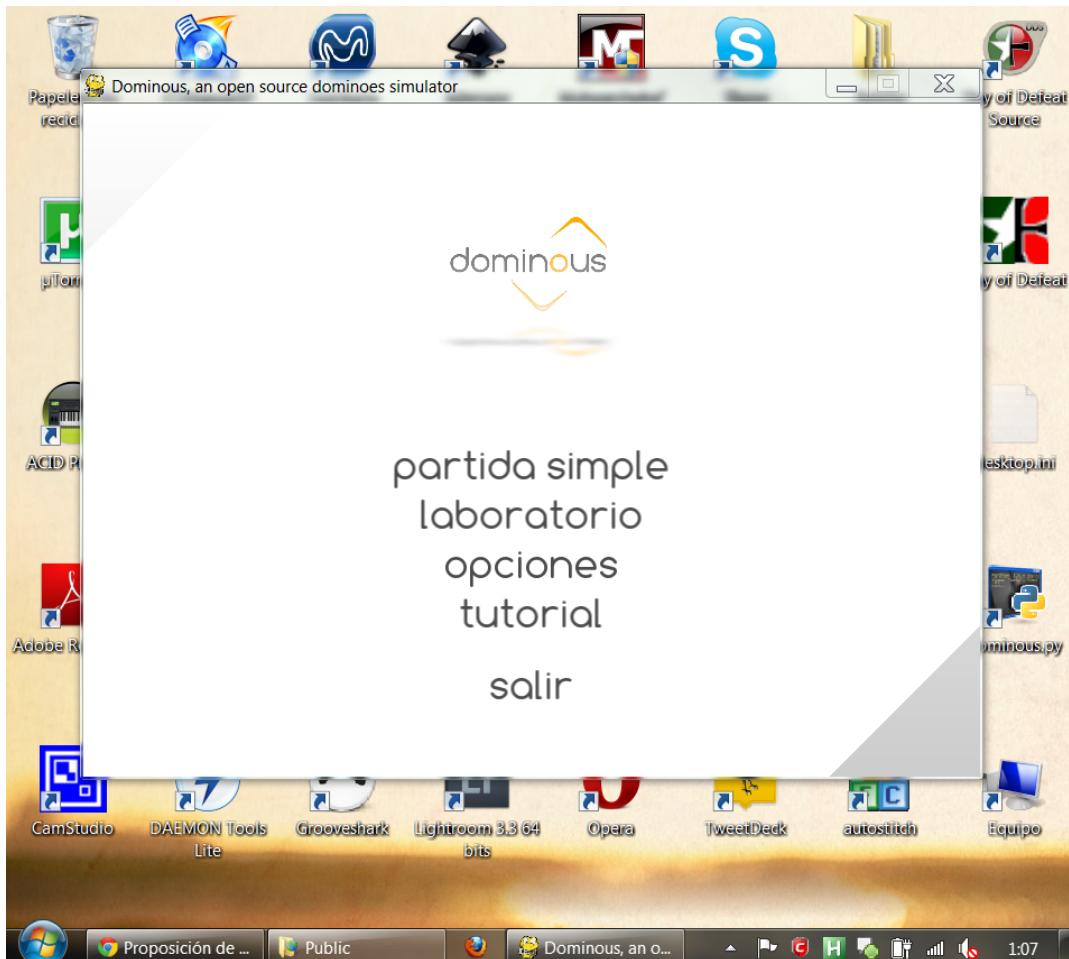


Figura 8.2: Dominous ejecutándose en un entorno Microsoft Windows 7

Apéndice A

Manual de instalación

A.1. Obtener Dominous

Dominous se puede descargar desde varias fuentes principales. Dado que el lenguaje es interpretado y no compilado, el código es válido tanto para sistemas basados en GNU/Linux como para Windows 7.

La descarga se puede hacer desde la web oficial [Ign] o desde los repositorios alojados en la forja de Rediris , en el que además del código fuente se cuenta con herramientas como foros de consulta, descarga de manuales, bug tracker y otros.

A.2. Instalación y ejecución en GNU/Linux

La instalación bajo sistemas GNU/Linux requiere de cierto número de dependencias, que pueden ser instaladas cómodamente con el siguiente comando:

Listado A.1: Instalación de dependencias

```
1 | user@machine:~$ sudo aptitude install python-pygame python-setuptools && sudo easy_install -U PyOpenGL PyOpenGL-accelerate
```

Como vemos, las dependencias que requiere la aplicación son las siguientes:

- python-pygame
- python-setuptools
- PyOpenGL
- PyOpenGL-accelerate

Por último, para ejecutar y disfrutar de Dominous, solamente debemos acceder al directorio y ejecutar:

```
1 | user@machine:~$ python dominous.py
```

A.3. Instalación y ejecución en Windows

Por otra parte, si deseamos instalar la aplicación en un sistema Windows, debemos ejecutar los siguientes pasos:



Figura A.1: Página oficial de Dominous

- Obtener e instalar Python — Para ello nos dirigimos a la página oficial de Python, descargamos la última versión estable (que en el momento de escribir este documento era la 2.7.2) y seguimos los pasos de instalación.
- Obtener e instalar PyOpenGL — descargamos la última versión desde la página oficial de PyOpenGL y seguimos los pasos necesarios hasta instalarla correctamente en nuestro sistema.

Para finalizar, nos dirigimos a la carpeta de Dominous y hacemos doble click sobre el icono de `dominous.py`.

Apéndice B

Manual de usuario

B.1. Ejecución

En este capítulo veremos el funcionamiento de la aplicación desde el punto de vista de un usuario final que ejecutará el videojuego con la intención de disfrutar de las opciones que ofrece.

as Si el programa está instalado, se ejecuta con XXX

B.2. Menú principal

Apéndice C

Difusión

C.1. Difusión

Este proyecto ha recibido difusión en los siguientes medios:

- Forja de RedIRIS, con todo el código, documentación, tareas creadas para el proyecto, bug tracker, foros de ayuda y visor de archivos del sistema de control de versiones.
- Inclusión en la próxima distribución de Guadalinex.
- Finalista en la Categoría Proyectos Libres de Ocio de la Universidad de Cádiz 2009-2010 [C.1](#), dentro del IV Concurso Universitario de Software Libre.
- Accésit al Mejor Proyecto Libre de Innovación en la Universidad de Cádiz 2010-2011 [C.2](#), enmarcado en el V Concurso Universitario de Software Libre.
- Sección especial dedicada al programa dentro del portfolio personal.
- Página web de la Asociación de Desarrollo de Videojuegos de la UCA.



Figura C.1: Finalista en la Categoría Proyectos Libres de Ocio de la Universidad de Cádiz 2009-2010



Figura C.2: Accésit al Mejor Proyecto Libre de Innovación en la Universidad de Cádiz 2010-2011

Bibliografía

- [Bei90] Boris Beizer. *Software Testing Techniques*. International Thomson Computer Press, 2 edición, junio 1990. ISBN 1850328803.
- [Boa] BoardGameGeek, LLC. Trimino en Board Game Geek. <http://boardgamegeek.com/boardgame/97500/trimino>.
- [Bor90] Daniel Borrajo, Juan Ríos, M. Alicia Pérez y Juan Pazos. Dominoes as a domain where to use proverbs as heuristics. *Data and Knowledge Engineering*, 5(2):páginas 129 – 137, 1990. ISSN 0169-023X. doi:DOI:10.1016/0169-023X(90)90009-3.
- [Dim] Dimitri Van Heesch. Pagina oficial de Doxygen. <http://www.doxygen.org>.
- [Fat05] Ramin Fathzadeh, Vahid Mokhtari, Morteza Mousakhani y Alireza Mohammad Shahri. Coaching with expert system towards robocup soccer coach simulation. En Ansgar Bredenfeld, Adam Jacoff, Itsuki Noda y Yasutake Takahashi, editores, *RoboCup*, tomo 4020 de *Lecture Notes in Computer Science*, páginas 488–495. Springer, 2005. ISBN 3-540-35437-9.
- [Ger] Gerardo Aburruzaga García. Make. Un programa para controlar la recompilación. <http://www.uca.es/softwarelibre/publicaciones/make.pdf>.
- [GFo] GForge Group. Pagina oficial de GForge. <http://gforge.org>.
- [Gia89] Joseph C. Giarratano y Gary Riley. *Expert Systems: Principles and Programming*. Brooks/Cole Publishing Co., Pacific Grove, CA, USA, 1989. ISBN 0878353356.
- [GS00] J.L. González Sanz. *El arte del dominó: teoría y práctica*. Paidotribo, 2000.
- [Ign] Ignacio Palomo Duarte. Web oficial de Dominous. <http://dominous.forja.rediris.es>.
- [Lia05] Shu-Hsien Liao. Expert system methodologies and applications—a decade review from 1995 to 2004. *Expert Systems with Applications*, 28(1):páginas 93 – 103, 2005. ISSN 0957-4174. doi:DOI:10.1016/j.eswa.2004.08.003.
- [Lug08] Miguel Lugo. *Dominó Competitivo*. AuthorHouse, 2008.
- [Mar72] E. Martinez. *Tratado del Juego del Domino, sus Reglas, Combinaciones y Preceptos para ser un buen jugador*. Imprenta de Salvador Amarzós, 1872.
- [Mit04] Frank Mittelbach y Michel Goossens. *The LaTeX Companion*. Addison-Wesley, 2004.
- [Mye04] Glenford J. Myers. *The Art of Software Testing*. John Wiley & Sons, 2 edición, 2004. ISBN 0471469122.

- [Pal05] M. Palomo, F. Martín-Mateos y J. Alonso. Rete algorithm applied to robotic soccer. En Roberto Moreno Díaz, Franz Pichler y Alexis Quesada Arencibia, editores, *Computer Aided Systems Theory – EUROCAST 2005*, tomo 3643 de *Lecture Notes in Computer Science*, páginas 571–576. Springer Berlin / Heidelberg, 2005. ISBN 978-3-540-29002-5. 10.1007/11556985_75.
- [Pil04] Mark Pilgrim. *Dive Into Python*. APress, 2004. ISBN 1590593561.
- [Pre02] Roger S. Pressman. *Ingeniería del Software - Un Enfoque Practico*. McGraw-Hill, 2002.
- [Qui10] Pablo Recio Quijano. Ampliación y reingeniería de un sistema experto basado en reglas con fines educativos. <http://dx.doi.org/10498/8484>, 2010. Memoria de Proyecto Fin de Carrera de Ingeniería Técnica en Informática de Sistemas.
- [RM90] B. Ruipérez Moral. *Libro del dominó*. Otero Ediciones, 1990.
- [Sch] Schwartz David. Dominó en línea. <http://www.domino-en-linea.com>.
- [Som05] Ian Sommerville. *Ingeniería del Software*. Pearson Education, 2005.
- [Tur06] Ross Turner. A bayesian dominoes player. <http://www.dcs.shef.ac.uk/intranet/teaching/projects/archive/ug2006/pdf/u2rt.pdf>, 2006. 3rd Year Dissertation Project.
- [Vá11] Daniel Chaves Vázquez. Freerummi: Juego de fichas numeradas. <http://dx.doi.org/10498/11134>, 2011. Memoria de Proyecto Fin de Carrera de Ingeniería Técnica en Informática de Sistemas.

GNU Free Documentation License

Version 1.3, 3 November 2008
Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.

<http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document “free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “**Document**”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “**you**”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “**Modified Version**” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “**Secondary Section**” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “**Invariant Sections**” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “**Cover Texts**” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “**Transparent**” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “**Opaque**”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “**Title Page**” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “**publisher**” means any person or entity that distributes copies of the Document to the public.

A section “**Entitled XYZ**” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “**Acknowledgements**”, “**Dedications**”, “**Endorsements**”, or “**History**”.) To “**Preserve the Title**” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled “History”, Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled “History” in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the “History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled “Acknowledgements” or “Dedications”, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

“Massive Multiauthor Collaboration Site” (or “MMC Site”) means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A “Massive Multiauthor Collaboration” (or “MMC”) contained in the site means any set of copyrightable works thus published on the MMC site.

“CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

“Incorporate” means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is “eligible for relicensing” if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with … Texts.” line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.