UNIVERSIDAD AUTONOMA DE MADRID

Escuela Politécnica Superior

2021

Trabajo fin de grado

Un entorno para generación de juegos colaborativos

Adrián García García

Campus Internacional excelencia UAM CSIC

www.uam.es

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Un entorno para generación de juegos colaborativos

Autor: Adrián García García
Tutor: Pablo Gómez Abajo

julio 2021

Adrián García García
Un entorno para generación de juegos colaborativos

Adrián García García
Escuela Politécnica Superior
Universidad Autónoma de Madrid

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

Un entorno para generación de juegos colaborativos

Adrián García García

Tutor: Pablo Gómez Abajo

Departamento de Ingeniería Informática
Universidad Autónoma de Madrid

Se presenta esta memoria para la obtención del título de
Grado en Ingeniería Informática

Escuela Politécnica Superior Julio 2021

Dedico este trabajo de fin de grado a Felipe y Susana.
Aspira alto en la vida, que soñar es fácil.
Algún día llegará algo que soñaste y nunca te arrepentirás.

Declaración

Declaro que los contenidos de esta memoria son originales, a excepción de las referencias hechas a otros trabajos. Los contenidos no se han enviado en ninguna otra universidad ni se han enviado en conjunto o en parte para consideración en ningún otro título o cualificación. Esta memoria es el resultado de mi propio trabajo y contiene nada más que el trabajo realizado en colaboración con el tutor, a excepción de lo especificado en el texto.

Adrián García García
Julio 2021

Agradecimientos

En primer lugar, quiero agradecer de forma infitina el esfuerzo de mis padres y mi hermano por convertirme en lo que soy ahora. Sin ellos nunca hubiera llegado hasta aquí. Han sido, son y serán mi pilar fundamental en todo. Ellos siempre han estado en los momentos más duros de mi vida y me han enseñado los valores y los conocimientos más importantes. En los momentos más difíciles donde no veía ninguna solución, me han orientado y nunca dudaron de lo que era capaz, incluso cuando yo mismo no lo veía. Podría decir que el esfuerzo que he desempeñado en la universidad es más cosa de ellos que mía. Sé que el camino ha sido duro, pero juntos siempre hemos sabido encontrar lo mejor de nosotros mismos. Os quiero.

En segundo lugar, quiero agradecer a Asmaa, mi compañera de vida. Ella me ha enseñado a valorar lo que soy, sin importar lo que piensen los demás. Con una persona así, nada podía salir mal a lo largo del camino. Eres el mejor apoyo que he tenido nunca.

En tercer lugar, quiero a agradecer a Jorge Durán, Jorge Hernán, Jaime Enríquez y Marcos de las Heras, mis mejores compañeros universitarios. Ellos han conseguido que lo más difícil del grado no fueran los examenes ni las prácticas, sino reirme en clase. Sinceramente, sin vosotros el camino hubiera sido largo y aburrido. Unos buenos camperos y alguna que otra salida es lo mejor que me llevo de la universidad. Nunca me voy a olvidar de lo que he vivido con vosotros.

En cuarto lugar, quiero agradecer a Pablo Gómez Abajo, mi tutor. Tú has hecho que este trabajo fuera divertido y llevadero. Desde el primer día sentí que eras sincero y abierto conmigo, dándome cuenta de que teniamos más cosas en común de lo que pensabamos. Nunca hemos tenido una reunión aburrida ni cordial, y eso ayuda a desestresar las cosas. No cambies nunca tu forma de trabajar. Puedes con todo.

Por último quiero agradecer a todas las personas que me he cruzado en este camino, sobre todo a los que me han apoyado. Estoy seguro de que una parte de este trabajo también es gracias a vosotros.

Resumen

A día de hoy, nos encontramos con un gran abanico de posibilidades a la hora de elegir y desarrollar un juego. Ya sea en móviles, juegos de mesa tradicionales, recreativos o videoconsolas, todos hemos jugado alguna vez y en el futuro todos acabaremos jugando. Sin embargo, existe un problema en el desarrollo de juegos en dispositivos electrónicos. La mayoría de los entornos capaces de generar juegos en estos dispositivos tienen un enfoque único al tipo de juego en desarrollo, acotando tanto el número de juegos finales como los enfoques de cada uno de ellos. En la sociedad actual y futura donde se cree que el crecimiento de los juegos va a aumentar de forma exponencial, sería ideal disponer de algún entorno capaz de generar juegos de forma automática a partir de una serie de parámetros sencillos, incluso sin necesidad de tener conocimientos de programación.

Una de las técnicas de desarrollo software más popular es la Ingeniería Dirigada por Modelos. En esta ingeniería el nucleo del desarrollo radica en un modelo, utilizado durante la especificación, la simulación, la generación de código y la ejecución final. Este modelo puede implementar un lenguaje de dominio específico que facilite su manipulación. Implementar este lenguaje en el modelo nos puede permitir desarrollar de forma automática, rápida e intuitiva aplicaciones software de diferentes ámbitos. Además, el uso de modelos y de lenguajes de dominio específico aumenta la abstracción del desarrollo, por lo que cualquier persona sin conocimientos de programación ni experto en el ámbito podría diseñar y desarrollar aplicaciones.

En base a esto, hemos desarrollado un entorno sencillo capaz de generar juegos colaborativos gracias a una sintaxis y un modelo enfocado a dos tipos de juego: juegos tipo comunio y juegos de mesa sencillos. Junto a este modelo, se presenta un lenguaje de dominio específico con una sintaxis similar a Python, lo que implica que sea fácil, intuitiva y legible. Adicionalmente, se han añadido las herramientas necesarias para que realice comprobaciones del modelo antes de su compilación, por lo que cualquier aplicación generada es funcional y compatible con el modelo propuesto. Para poner a prueba este entorno, se han desarrollado tres juegos diferentes: el primero de ellos es el gato y el ratón, un juego de mesa sencillo; el segundo consiste en el famoso juego de las damas y el tercero coincide con las mismas características que el Comunio.

Abstract

Today, we find a wide range of possibilities when choosing and developing a game. Whether on mobile phones, traditional board games, recreational or video consoles, we have all played at some time and in the future we will all end up playing. However, there is a problem in developing games on electronic devices. Most of the environments capable of generating games on these devices have a unique approach to the type of game in development, limiting both the number of final games and the focus of each of them. In today's and future society where the growth of games is believed to increase exponentially, It would be ideal to have an environment capable of generating games automatically from a series of simple parameters, even without the need for programming knowledge.

One of the most popular software development techniques is Model Driven Engineering. In this engineering, the core of development lies in a model, used during specification, simulation, code generation, and final execution. This model can implement a domain-specific language that makes it easy to manipulate. Implementing this language in the model can allow us to develop automatic, fast and intuitive software applications from different fields. In addition, the use of models and domain-specific languages increases abstraction. of development, so that anyone without programming knowledge or expert in the field could design and develop applications.

Based on this, we have developed a simple environment capable of generating collaborative games thanks to a syntax and a model focused on two types of games: community-type games and simple board games. Along with this model, a domain-specific language is presented with a syntax similar to Python, which implies that it is easy, intuitive and readable. Furthermore, the necessary tools have been added to verify the model before its compilation, so that any application generated is functional and compatible with the proposed model. To test this environment, three different games have been developed: the first one is the cat and the mouse, a simple board game; the second consists of the famous game of checkers and the third coincides with the same characteristics as the Comunio.

Índice general

Índice de figuras

Índice de tablas

Nomenclatura

Acrónimos / Abreviaturas

DSL Domain-Specific Language - Lenguaje de Dominio Específico

MDE Model-Driven Engineering - Ingeniería Dirigida por Modelos

Capítulo 1

Introducción

En esta primera sección, se detallan los principios introductorios a la tesis. En la sección 1.1 se tratará los antecedentes al trabajo realizado. Posteriormente, la sección 1.2 explica los motivos por los que se ha desarrollado el trabajo. En el apartado 1.3 nos encontramos con las contribuciones que se han dado en el desarrollo. Para finalizar, en el apartado 1.4 se explica la estructura que seguirá la memoria.

1.1. Antecedentes

La Ingeniería Dirigida por Modelos (Model-Driven Engineering, MDE) [10] se trata de un enfoque en el desarrollo de software donde la salida principal del desarrollo es un modelo y no unos programas, ya que a partir del modelo se genera de forma automática el código de estos. De esta manera, consigue una mayor abstracción durante el desarrollo software, pues el modelo supone el rol principal a lo largo de todo el desarrollo. [13]

En esta clase de ingeniería, se suele implementar un modelo que permita su reutilización con el fin de evitar codificar programas u otros modelos semejantes. Para la implementación de estos modelos esenciales en este tipo de desarrollo software, se recurre normalmente a los lenguajes de dominio específico (Domain-Specific Languages, DSLs). Los DSLs se consideran lenguajes especializados en resolver y modelar un conjunto de tareas, por lo que al añadirlo al modelo se consigue una manipulación excepcional [14].

Para disponer de un modelo completamente reutilizable, debe ser lo más general posible y abarcar el mayor número de secciones posibles. Sin embargo, no existe el modelo definitivo en cuanto a reutilización. Cuanto más general y más secciones abarque, el modelo será más complicado y puede suponer una gran inversión de tiempo, dinero y esfuerzo. Podríamos analizar la curva de estas tres variables como una gráfica exponencial. El modelo reutilizable

idóneo estaría situado en el punto de la curva más equilibrado, pero es muy dificil de conseguir.

Un ejemplo sencillo lo podemos encontrar en los videojuegos. Si quisieramos modelar únicamente un tipo de juego, el esfuerzo, tiempo e inversión sería más pequeño que un modelo que englobe dos o más tipos de juegos. Por el contrario, si quisieramos englobar en un modelo todos los tipos de juego que existen, las tres variables crecen de forma exponencial y alcancaría unos valores desorbitados. Sin embargo, no significa que un modelo enfocado a tres tipos de juegos disponga de un coste mayor que un modelo enfocado a dos tipos de juegos.

El modelo más equilibrado en este contexto sería aquel que englobe de la forma más general posible aquellos juegos que dispongan de características parecidas. Por ejemplo, un modelo que englobe juegos de tipo shooter y de futbol (en adelante modelo A) va a presentar unos valores más elevados en cuanto a esfuerzo, tiempo de desarrollo e inversión de capital que uno enfocado a juegos de futbol, baloncesto y balonmano (en adelante modelo B), ya que los juegos que engloba tienen más diferencias. Sin embargo, a la hora de extender los modelos es más costoso añadir shooters en el modelo B que añadir juegos deportivos en el modelo A, ya que el modelo A ya cuenta con características muy similares para su expansión mientras que el modelo B no. En definitiva, para generar un modelo reutilizable y equilibrado, debemos enfocar el desarrollo en conseguir que sea general y que abarque todas las secciones posibles, contando con el mayor número de características similares en dichas secciones.

1.2. Motivación

Hoy en día, nos encontramos que los videojuegos están al alcance de cualquiera, ya sea desde un dispositivo móvil, un pc o una videoconsola. Existen una gran cantidad de tipos de juegos pero, analizándolos, nos encontramos que existen una gran cantidad de similitudes entre ellos, como pueden ser los jugadores (uno o más), artículos de los que disponen, marcadores, etc.

En el entorno de los videojuegos, existen una gran multitud de herramientas enfocadas a su creacción y desarrollo. Entre estas destaca Unity, Unreal Engine o Frostbite entre muchas otras. Todas estos motores presentan una gran cantidad de opciones para desarrollar cualquier tipo de videojuego. Sin embargo, se enfocan en el desarrollo individual de un determinado juego. Estos motores nos permiten reutilizar parte de los videojuegos, obviamente de forma individual en cada desarrollo, pero ninguno de ellos nos permite crear de forma automática varios juegos desde una misma "plantilla". Partiendo de un juego A de tipo Acción, para desarrollar otro juego B del mismo tipo con una historia diferente y los mismos escenarios,

personajes, texturas, etc, nos encontramos que podemos reutilizar el código del juego A pero debemos retocarlo para incorporar la nueva historia. Sin embargo, al utilizar la MDE, podemos favorecer el desarrollo ágil y, a partir de un mismo modelo, generar otro juego completamente diferente de forma automática variando la historia. De esta forma obtenemos un desarrollo automático de videojuegos a petición del cliente, con un desarrollo ágil de software y un ahorro en el coste de tiempo, esfuerzo y capital.

Ante esta falta de recursos de desarrollo automático, se ha decidido implementar un entorno de desarrollo enfocado a la creacción automática de juegos corporativos mediante el uso de la Ingeniería Dirigida por Modelos y un Lenguaje de Dominio Específico que facilite su manipulación. La elección de los juegos corporativos se debe a su gran cantidad de juegos y de tipos que podemos encontrar. Con este entorno, seremos capaces de desarrollar de forma automática juegos corporativos diferentes gracias a un mismo modelo que reutiliza todas las características comunes que pueden presentar.

1.3. Contribución

Con el fin de facilitar la lectura de la memoria y su comprensión, hemos desarrollado un entorno reducido mediante DSL enfocado a juegos de mesa sencillos (por ejemplo las damas) y juegos de tipo comunio. Este entorno nos permite manipular facilmente el modelo principal, consiguiendo desarrollar juegos de una forma fácil e intuitiva. Gracias a este DSL, podemos diseñar juegos completamente diferentes utilizando siempre la misma sintaxis. Esta sintaxis es muy similar a la de python, con la excepción de que no es necesario tabular para su correcto compilado y ejecutado.

Cada juego generado mediante este DSL garantiza su correcta adaptación al meta-modelo del entorno, asegurando que cumple todas las restricciones y peculiaridades que figuran en el. También se asegura de que todos los valores no declarados y necesarios para la generación automáticática del juego se puedan obtener gracias al meta-modelo y se autorrellenen de acuerdo a unos valores predeterminados.

El entorno generado, dados unos parámetros fijados en el DSL asociado al entorno, genera juegos de forma automática utilizando python como lenguaje de programación. Como se ha mecionado anteriormente, este entorno es reducido por lo que se puede extender de diversas formas de cara al futuro. Se pueden reutilizar en futuras extensiones varios atributos del metamodelo para generar, por ejemplo, juegos de mesa más complicados (el ajedrez). Esta posible extensión se encuentra detallada en el capítulo 5.

A lo largo de la memoria, se describe el meta-modelo implementado con sus reglas de validación y generación de código además de detallar el lenguaje DSL que se ha desarrollado

para la interacción con el usuario final. El meta-modelo presenta características comunes a ambos tipos de juegos y propias de cada uno, por lo que no todos los juegos necesitan utilizar todos los campos del modelo. Las reglas de validación se han estimado de acuerdo a los dos tipos de juego que se han tenido en cuenta. Seguramente, si en una extensión se quieren añadir nuevos tipos de juegos, debería extenderse junto al meta-modelo. La plantilla de generación de código se ha desarrollado en python, donde se encuentra todo el núcleo de los juegos finales. Es completamente reutilizable a todos los tipos de juego destinado en este entorno. Esto se debe a que utiliza clases para la implementación de los atributos del meta-modelo y el núcleo de juego se ha desarrollado de la forma más general posible. Al ser tan general, la mayoria de los juegos a los que se enfoca el entorno no utilizan todas las prestaciones que ofrece.

1.4. Estructura de la memoria

La estructura que acompaña al resto de la memoria se resume en los siguientes apartados:

El **Capítulo 2** contiene información sobre los conceptos previos de la Ingeniería Dirigida por Modelos, concretamente información sobre la pripia MDE, los Lenguajes de dominio específico, la generación de texto y las tecnologías utilizadas en el desarrollo del entorno.

El **Capítulo 3** explica la implementación del DSL elaborado para la generación de los juegos corporativos. Además, explica las herramientas proporcionadas para trabajar sobre el DSL.

El **Capítulo 4** muestra tres ejemplos desarrollados utilizando el DSL explicado en el Capítulo 3. Concretamente, muestra el ejemplo del juego de mesa el ratón y el gato, las damas y un juego tipo comunio.

El **Capítulo 5** expone una serie de conclusiones finales en base al entorno generado y un análisis sobre el trabajo futuro o extensión del modelo desarrollado.

Capítulo 2

Conceptos Previos y Trabajo Relacionado

A lo largo de este capítulo se van a detallar los conocimientos fundamentales sobre MDE (apartado 2.1) que se han necesitado para el desarrollo del entorno. Concretamente, se detallarán los conceptos de la ingeniería dirigida por modelos, lenguajes de dominio específico, generación automática de texto y las tecnologías utilizadas durante el desarrollo del entorno. Para cerrar el capítulo, se van a explicar en el apartado 2.2 una serie de técnicas que se han utilizado para facilitar varios aspectos del desarrollo.

2.1. Ingeniería dirigida por modelos

La ingeniería Dirigida por Modelos es una rama dentro de la Ingeniería del Software enfocada al uso de modelos y sus transformaciones a lo largo de todo el ciclo de vida software [10] . Concretamente, estos modelos son utilizados para probar, especificar, generar y simular código de la aplicación final desarrollada de forma automática.

En el ámbito de MDE, un sistema se encuentra definido como concepto genérico para el diseño de una aplicación, plataforma o artefacto software [13] . En base a esto, un modelo se define como la abstracción sobre un sistema actual y futuro. El uso de modelos en este paradigma nos ayuda a especificar y definir un sistema y a simular diferentes aspectos sin la necesidad de utilizarlo directamente.

Durante décadas, la misión principal en cualquier ingeniería software es conseguir un desarrollo de gran calidad a un bajo coste. En la ingeniería dirigida por modelos, el núcleo base se encuentra en el uso de modelos a lo largo de todo el ciclo de vida. Gracias a estos modelos, la MDE consigue un gran nivel de abstracción (elevando a su vez la automatización durante y después del desarrollo), junto a una reducción de la complejidad y la mejora de una gran cantidad de atributos de calidad del software final, como el mantenimiento o la productividad. Además, los beneficios que otorga no solo se aplican al desarrollo, si no al

proceso de reingeniería y a la configuración dinámica en tiempo de compilación. Concretamente, esta ingeniería nos permite generar aplicaciones a través de una transformación de modelos expresados con un DSL específico a código.

A diferencia de la ingeniería basada en objetos, la MDE presenta tres paradigmas de desarrollo software. El primer paradigma se basa en el desarrollo de nuevas aplicaciones mediante el uso de ingeniería directa y recibe el nombre de Model Driven Development (MDD), donde destacan las factorias de software, la arquitectura dirigida por modelos (MDA) y la programación orientada a lenguajes (DSM). El segundo corresponde a la reingenieria o la modernización de entornos basados en modelos (MDR). El último se trata de Models@runtime y se basa en el uso de modelos con el fin de representar información del contexto de ejecución para realizar una configuración o adaptación en el software que se encuentra en ejecución. De todas ellas, la última es la que menos uso tiene y, por lo tanto, la menos extendida. [13]

Todos estos paradigmas constan de cuatro principios básicos, que corresponden con los pilares de esta disciplina. El primero de ellos determina que un modelo representa uno o varios aspectos de sistemas software de forma parcial o total. El segundo expresa que todos estos modelos deben ser representados mediante un DSL específico. El tercero consiste en que un meta-modelo se emplea para representar formalmente un DSL determinado. Para finalizar, el cuarto y último pilar, determina que el concepto de automatización tan importante en esta disciplina se consigue mediante transformaciones de modelos a código.

Los lenguajes específico de dominio constan de 3 elementos principales. El primero de ellos es la sintaxis abstracta, que consiste en la definición de las reglas para estimar que un modelo está bien formado junto a sus conceptos del lenguaje y las relacciones entre ellos. El segundo elemento es la sintaxis concreta empleada para establecer la notación del lenguaje. Por último, el tercer atributo es la sintaxis semántica con la que se desarrolla la semántica del propio lenguaje.

La principal ventaja de utilizar DSL en nuestro desarrollo es que nos permite encontrar soluciones software a un nivel mayor de abstracción que los clásicos lenguajes utilizados. A su vez, para conseguir que sea efectivo, se aumenta el nivel de automatización junto al de abstracción.

En MDE, este aumento en el nivel de automatización se consigue generando de forma automática el código final a partir de modelos ya existentes, ya sea de forma directa mediante transformaciones modelo a texto (m2t) o de forma indirecta mediante la generación de modelos intermedios con transformaciones modelo a modelo (m2m). En el entorno desarrollado, se obtiene esta mejora de automatización de forma directa con el uso de transformaciones modelo a texto.

En MDE, los DSL constan de sus elementos ya definidos. La sintaxis abstracta se desarrolla como un metamodelo, es decir, un modelo expresado mediante un lenguaje de metamodelado que posee una serie de reglas y restricciones para determinar si un modelo está bien estructurado y formado. El lenguaje de metamodelado más utilizado a nivel internacional es Ecore y forma el elemento principal en el Eclipse Modeling Framework (EMF) [7] . Ecore es un lenguaje fácil e intuitivo. Proporciona a los desarrolladores los conceptos de los desarrollos enfocados a objetos, permitiendo expresar el lenguaje mediante clases, atributos, agregaciones y referencias entre clases. La sintaxis concreta se puede definir de forma textual o gráfica. En nuestro caso, se ha utilizado una herramienta basada en la creacción de metamodelado para generar DSL textuales denominada Xtext.

Al igual que el modelo se define con una sintaxis abstracta que es el lenguaje de metamodelado, el lenguaje de metamodelado también está descrito por una sintaxis abstracta. Esta sintaxis formaría parte de un metamodelo, que realmente sería un metametamodelo. Para la definición de este metametamodelo, se utiliza la definición de arquitectura de cuatro niveles utilizada para implementar las relacciones entre modelos y metamodelos. Esta arquitectura está formada por los siguientes cuatro niveles:

Nivel de datos del usuario o del mundo real (M0): recoge aquellos objetos que forman parte del mundo real y son utilizados por el software. Por objetos se entiende un gran abanico de posibilidades, incluyendo procesos, conceptos o estados.

Nivel de modelo (M1): formado por aquellos modelos que representan los datos que forman parte del nivel M0.

Nivel de metamodelo (M2): engloba los metamodelos que describen los modelos del nivel M1.

Nivel de metametamodelo (M3): recoge los metamodelos que describen las características de los modelos del nivel M2.

En base a este modelo de arquitectura, un metamodelo es definido mediante una definición circular, es decir, se define con los mismos conceptos que han sido definidos en el meta-meta-modelo.

Existen algunas restricciones en el metamodelo que no se pueden expresar como clases y relacciones. Ante estos casos, se suele recurrir al Object Constraint Language (OCL) [11]. El OCL consiste en un lenguaje declarativo utilizado para describir restricciones de integridad en modelos. Este lenguaje puede ser utilizado en cualquier meta-modelo ECORE y MOF. Las restricciones declaradas con OCL, deben ser definidas y evaluadas en el contexto y en las instancias, respectivamente, de una clase.

2.2. Lenguajes de dominio específico

Un lenguaje de Dominio Específico (DSL) consiste en un lenguaje de especificación o programación enfocado a la representción de una situación en particular, resolución de problemas en un ámbito concreto o el diseño de técnicas para tratar sistemas o resolver problemas particulares [13] . Este tipo de lenguaje no es nuevo, pero goza de gran popularidad en la actualidad gracias al modelaje específico de dominio.

El uso de DSL se encuentra enfocado a la resolución y manejos de problemas repetitivos (el problema reaparece varias veces) cuyos resultados se pueden expresar de una forma más fácil e intuitiva que si son desarrollados con otro tipo de lenguaje. Sin embargo, otorga una gran abstracción sobre el problema principal, por lo que están destinados a usuarios que conocen el problema o son profesionales en su ambito. Con esto se consigue que el DSL se considere un lenguaje más eficiente frente a los lenguajes tradicionales (C, Java, etc) si se utiliza en el mismo ámbito para el que ha sido diseñado. [14]

Como ejemplos de DSL, cabe destacar SQL para manejar bases de datos relaccionales y realizar consultas en ellas; Mathematica [9] para hojas de cálculo como Excell y matemáticas; Csound [8] para crear, editar, componer y analizar sonidos y música.

En la actualidad, las herramientas que permiten desarrollar entornos enfocados al DSL están en continuo desarrollo y permiten a personas con pocos conocimientos o expertos desarrollar DSL invirtiendo poco esfuerzo. Estas herramientas nos permiten desarrollar la sintaxis concreta del DSL de dos formas: gráfica y visual. En el enfoque gráfico, destacan Sirius [1], GMF [3] y Graphiti [4], mientras que en el enfoque textual destaca EMFText [2], Xtext [12] y Spoofax [6]. En el desarrollo de nuestro entorno, hemos utilizado la herramienta textual Xtext. Esta herramienta será descrita con más detalle en el apartado 2.4.2.

A la hora de otorgar semántica al DSL, destacan el enfoque operacional y denotacional. En el enfoque denotacional, el significado de un modelo o programa se otorga como otro formalismo cuya semántica se encuentra bien definida. El enfoque operacional determina que el significado de un modelo o programa bien formado es el resultado representado como trazas de cómputo de su ejecución y simulación. En el trabajo realizado, hemos utilizado el enfoque denotacional en el que se dota al DSL de semántica gracias a su compilación a código Pyhton.

2.3. Generación de texto

El último paso que se debe realizar en MDE es conseguir que el modelo desarrollado sea compilado y ejecutado. Para ello, a partir del modelo, se utilizan sus elementos y atributos

para generar código en un lenguaje de alto nivel, consiguiendo poder ejecutar dicho modelo [10]. En el entorno generado, el lenguaje con el que se genera el código es Python [5].

Los generadores de texto o compiladores de modelos se tratan de una transformación donde la entrada es un modelo y la salida un artefacto textual. En MDE, los generadores de texto se utilizan para la generación automática de código con un nivel más bajo de abstracción partiendo de un modelo determinado. Se suele utilizar para el desarrollo total o parcial de una aplicación. Recibe el nombre de compilador de código ya que su comportamiento se asemeja a los compiladores que, dado un código funte, generan ficheros binarios ejecutales.

La generación automática de código se suele realizar a través de plantillas basadas en reglas. Algunos ejemplos de generación de plantillas son Xtend, JET o Acceleo entre muchos otros. Un generador de código consiste en una serie de plantillas formada por controles en su propio contenido que, al aplicarse sobre los atributos del modelo de entrada, producen el código automático. En nuestro entorno, hemos utilizado como lenguaje de plantillas Xtend y es descrito en el apartado 2.4.2. Con este lenguaje de plantillas, generamos el código Python como resultado del modelo.

Con el código ya generado, simplemente queda optimizarlo, compilarlo, ejecutarlo y/o desplegarlo. Para ello, se suele utilizar las herramientas que proporciona el propio IDE utilizado.

2.4. Tecnologías utilizadas

A lo largo de esta sección se van a detallar las tecnologías utilizadas a lo largo del desarrollo del entorno. En el apartado 2.4.1, se va a explicar la herramienta Eclipse Modeling Framework (EMF). En el 2.4.2, se hablará de Xtext y Xtend. Para finalizar el capítulo, se hablará sobre python y en que ambito del desarrollo se ha utilizado.

2.4.1. Eclipse Modeling Framework

Eclipse Modeling Framework se trata de un framework [7], como su propio nombre indica, donde se proporcionan las herramientas necesarias para desarrollos de aplicaciones basados en MDD. Este nos proporciona lo necesario para desarrollar y definir modelos de un dominio específico utilizando XML Metadata Interchange (XMI), interfaces Java anodadas o diagramas UML. Sin embargo, da igual la metodología que utilicemos para definir modelos, ya que con cualquiera de las tres podemos generar las otras dos restantes de forma automática y, si nuestro objetivo es modificar la implementación del modelo a nivel de código, también genera la implementación del modelo en clases de Java. De esta forma,

proporciona al desarrollador dos formas de desarrollo: modelaje y programación. EMF es una implementación de MetaObject Facility (MOF) de la OMG y supone el estándar en el desarrollo de modelado.

A pesar de las grandes diferencias entre modelado y programación, en EMF ambas opciones se pueden considerar equivalentes. Permite al desarrollador determinar el nivel de separación de abstracción entre ambos desarrollos, es decir, ambas forman parte del mismo trabajo pero se puede conseguir una separación entre ingeniería de alto nivel de modelaje y la programación de bajo nivel.

Para representar meta-modelos en EMF, se utiliza un modelo propio denominado Ecore, formado por un simplificado subconjunto de UML. Este incluye la posibilidad de añadir restricciones OCL y lograr así un meta-modelo más optimizado. En la definición un meta-modelo, nos permite crear sus elementos con clases Ecore. Algunas de estas clases son las siguientes [7]:

- Eclass: nos permite representar el modelo en base a una clase representado por un nombre, cero o más referencias y cero o más atributos.
- EAttribute: permite representar un atributo del modelo, caracterizado por un nombre y un tipo primitivo.
- EReference: representa el extremo origen de una relacción entre clases y se caracteriza por tener un boolean para determinar si la referencia es contenedora, la clase destino de la referencia, etc.

2.4.2. Xtext y Xtend

Xtext es un propio framework de eclipse desarrollado con código abierto y utilizado en la implementación e integración con el propio IDE de Eclipse de DSLs textuales [12]. Xtext permite el desarrollo ágil de lenguajes, cubriendo todos los aspectos que debe tener un entorno completo de edición del lenguaje. Algunos de estos aspectos son el intérprete o generador de código, el analizador sintáctico, el completado de código o marcadores de errores entre muchos otros. Para implementar un lenguaje de Xtext, debemos crear un nuevo proyecto Xtext a partir de un meta-modelo DSL ya implementado. Xtext genera de forma automática una gramática para el DSL del meta-modelo, pero el desarrollador es libre de modificarla con el fin de obtener una gramática ideal en el ámbito del desarrollo. Sin embargo, Xtext no genera de forma automática la generación de código ni la propia funcionalidad del editor, por lo que el desarrollador (en caso de necesitarlo) debe completarlo. Al estar Xtext

implementado a parir de EMF, cualquier programa desarrollado con Xtext es un modelo de EMF y, por lo tanto, puede ser representado como un modelo XMI.

Xtend es un lenguaje de programación muy similar a Java. Permite interactuar con cualquier sistema de tipos de Java y se compila a Java de forma automática. Posee una sintaxis con técnicas avanzadas, donde podemos encontrar expresiones lambda o inferencia de tipos, pero destaca por ser sencilla y compacta. Xtend es muy utilizado por Xtext con el fin de configurar una gran cantidad de servicios asociados al editor del DSL.

2.4.3. Python

Python es un lenguaje de programación interpretado que proporciona una semántica dinámica [5]. Se encuentra orientado a objetos y a la programación de alto nivel. Es comunmente utilizado en el desarrollo rápido de aplicaciones debido a sus estructuras de datos integradas combinadas con tipos y enlaces dinámicos. Además, también es utlizado frecuentemente en scripts como lenguaje o como enlace de conexión a la hora de conectar componentes ya existentes.

Presenta una sintaxis más simple, fácil y limpia que otros lenguajes, lo que hace que el aprendizaje de Python sea rápido e intuitivo. Gracias a esta sintaxis, aumenta la legibilidad y reduce el costo del desarrollo y mantenimiento de los programas. Además, a esto se le suma una gran modularidad y reutilización de código, ya que permite el uso de módulos y paquetes. La mayor virtud a la hora de utilizar Python es su productividad. Al no añadir compilación de código, el ciclo de vida de desarrollo software es increiblemente rápido.

Capítulo 3

Un entorno para generación de juegos colaborativos

En este capítulo, en la sección 3.1 se va a describir el DSL desarrollado para la generación de los juegos y en la 3.2 se va a describir en detalle la herramienta proporcionada, incluyendo el meta-modelo, el código de generación automática de código y la validación adicional añadida.

3.1. El DSL para generación de los juegos

Para el manejo de creación y edición de juegos corporativos en nuestro entorno, hemos desarrollado un DSL con el propósito de que sea fácil e intuitivo en su uso. Xtext genera de forma automática un DSL en formato Java pero, para aumentar su legibilidad y que sea más fácil e intuitivo, hemos realizado modificaciones para que adopte una sintaxis similar a Python.

El resultado final es una sintaxis sin llaves, en la que las separaciones se encuentran definidas por comas en caso de que sea un array o nada en cualquier otro caso. Para detallar los elementos de los que dispone cada clase del modelo, debemos declarar la clase seguido de dos puntos y, posteriormente, dichos elementos. La eliminación de ciertos atributos de Java y la incorporación de características Python obtienen como resultado una sintaxis que desemboca en un menor número de lineas en la descripción de los juegos y aumenta su acceso a cualquier tipo de personas, ya sean profesionales del sector o no. El modelo puede ser descrito en una simple linea o en varias, quedando a decisión del usuario final como quiere detallar el juego.

Todos los elementos junto a sus atributos declarados en el meta-modelo tienen su traducción directa en la sintaxis, es decir, todos los elementos utilizados en el DSL reciben el mismo nombre que en el meta-modelo y todos los atributos del meta-modelo existen en la sintaxis. Los tipos de valores primitivos empleados son los siguientes:

- EString: para atributos que se definan mediante una cadena de caracteres. Su valor por defecto es la cadena vacía.
- EBoolean: para atributos que se definen mediante un valor booleano. Su valor por defecto es verdadero.
- EInt: para atributos que se definan como valores numéricos. Su valor por defecto es -1, valor con el que en la generación de código y validación nos permite identificar si ha sido declarado o no.

Esta sintaxis permite su reutilización para diferentes tipos de juegos. Con esto, utilizando siempre los mismos elementos del modelo y aplicando la misma sintaxis, permite crear juegos de mesa sencillos o tipo comunio y que ambos compartan la misma estructura de desarrollo. De esta forma, podemos cambiar de forma rápida cualquier propiedad del juego ó incluso reutilizar juegos para el desarrollo de otros nuevos o que formen una parte de el. El resultado obtenido en el trabajo se encuentra detallado en el Apéndice 1.

3.2. Herramienta proporcionada

Junto al DSL desarrollado en el punto anterior, el entorno consta de un meta-modelo, un código de validación y un código de generación de texto.

El meta-modelo se ha implementado como Ecore en Eclipse Modeling Framework. Analizando los dos tipos de juegos a los que va destinado el entorno, se han incluido tanto las características comunes como las propias de cada tipo. Estas características son las mostradas en la figura 3.1. Este modelo es simple y completamente extensible de cara al futuro, permitiendo a los futuros desarrolladores seguir reutilizando sus elementos y atributos, como por ejemplo la clase Player, Game o Token.

El meta-modelo está formado por los siguientes elementos:

- Player: elemento que referencia a los jugadores del juego. Estos jugadores se caracterizan por tener un nombre y un id únicos, una lista de tokens (descritos en el siguiente punto), una contraseña (password) que permita iniciar sesión en el juego, una puntuación (score) utilizada por los juegos comunio para la clasificación, un monedero

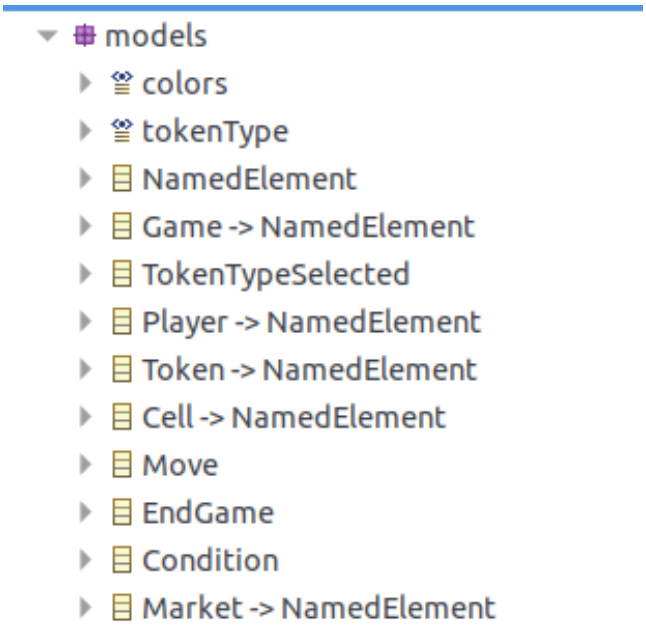

Figura 3.1 Elementos que forman el modelo.

(wallet) utilizado por los juegos tipo comunio para la compra de tokens en el mercado y un array de tokens seleccionados que conforman la plantilla de tokens de cara a la puntuación.

- Token: característica del modelo que denota los diferentes items que pueden tener los jugadores. Un item se caracteriza por tener un nombre y un identificador únicos, un color, un movimiento en caso de juegos de mesa, una puntuación y un tipo enfocado a los juegos tipo comunio.

- Cell: representa las celdas que forman el tablero en los juegos de mesa. Se caracteriza por tener un nombre y un identificador único, además de contar con un color y un token en el caso de que la casilla este ocupada por ese token.

- Move: elemento del modelo que representa los movimientos que pueden realizar los items en los juegos de mesa sencillos. Representa movimientos en dirección norte, sur, este, oeste, noreste, noroeste, sureste y suroeste.

- Game: Elemento principal del modelo que representa el núcleo del juego. Está formado por la lista de jugadores que componen el juego, la lista de celdas que componen el tablero en los juegos de mesa sencillos, un número de filas y de columnas para

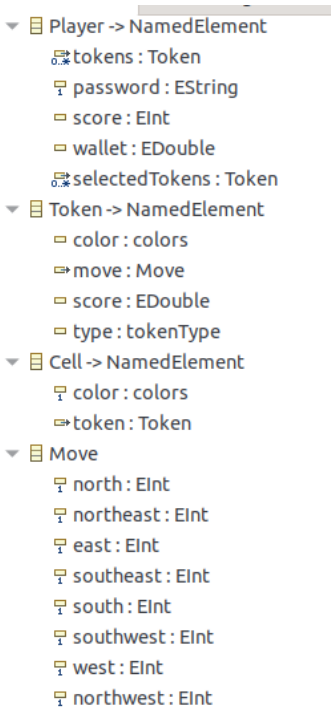

Figura 3.2 Características detalladas de Player, token, Cell y Move.

representar las diferentes dimensiones que puede tener un tablero, el turno del jugador al que le toca mover ficha en juegos de mes sencillos, una lista de condiciones que determinan el final del juego en juegos de mesa, dos propiedades que determinan si en un juego de mesa se puede comer y, en caso de que se pueda, si salta a la siguiente casilla o no (eat y next respectivamente), una lista de ofertas de mercado en el tipo comunio, un administrador en el juego que pueda manejar ciertos criterios, un valor que determine el número máximo de items que un jugador puede selecionar en la jornada de puntuación, una propiedad que determine si el juego está en una jornada de puntuación y un array donde se especifique cada tipo de token junto al valor máximo que puede haber en la jornada (por ejemplo goalkeeper 1)

- TokenTypeSelected: Elemento que determina el número máximo de cata tipo de token que forman parte del juego. Se caracteriza por tener una referencia al ipo del token y el valor máximo.

- EndGame: atributo del modelo que referencia a las condiciones de finalización en juegos de mesa sencillos. Se caracteriza por tener una referencia al jugador ganador junto a dos tipos de condiciones: el número de tokens que deben tener los rivales ó un array de condiciones de final (descritas en el siguiente punto).

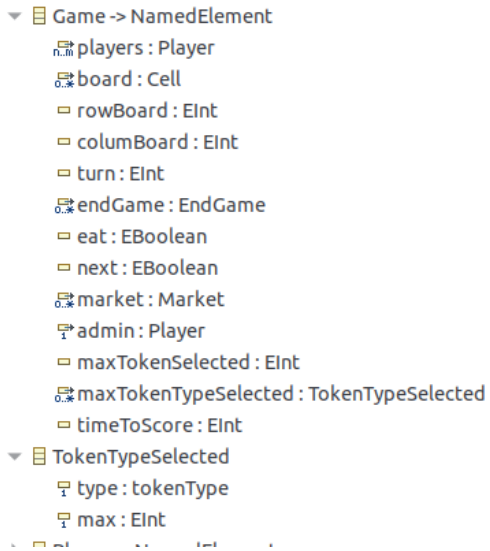

Figura 3.3 Características detalladas de Game y TokenTypeSelected.

- Condition: referencia a las condiciones que se encuentran en el elemento EndGame. Se caracteriza por tener un referencia a un item junto a los posibles movimientos y celdas donde queremos que se compruebe (por ejemplo si queremos que el token no pueda desplazarse hacia el norte bastaria con añadir en el movimiento la característica north con valor -1 y debemos indicar en que celdas se debe comprobar la condición).

- Market: representa las ofertas del mercado en los juegos tipo comunio. Se caracteriza por tener un nombre y un identificador único, el token al que hace referencia la compra, el precio final y el comprador actual (en caso de que no exista vacio).

El código producido como resultado de la generación automática utiliza Python como lenguaje. Esto se debe a que Python es un lenguaje más sencillo que sus rivales (como Java), ahorra más tiempo y espacio durante su ejecución y la biblioteca Tkinter nos ha permitido hacer una aplicación gráfica mediante ventanas. Este código se encuentra formado por clases de Python que están estrechamente relaccionadas con los elementos que forman el meta-modelo junto a una serie de funciones comunes (como iniciar sesión o cerrar sesión) y propias de cada tipo (como comer fichas en juegos de mesa o comprar token en el mercado en juegos tipo comunio). Esto nos facilita la reutilización dentro del propio código, 9ya que los elementos comunes entre los juegos utilizarán siempre las mismas clases y en un desarrollo futuro nuevos juegos también pueden reutilizarlas.

La aplicación generada utiliza como medio de guardado un fichero json con todos los datos que utiliza. Al ser un entorno sencillo, no se ha incorporado ningún medio de seguridad en este bolcado de datos. Si se quisiera extender el entorno, se puede añadir ciberseguridad

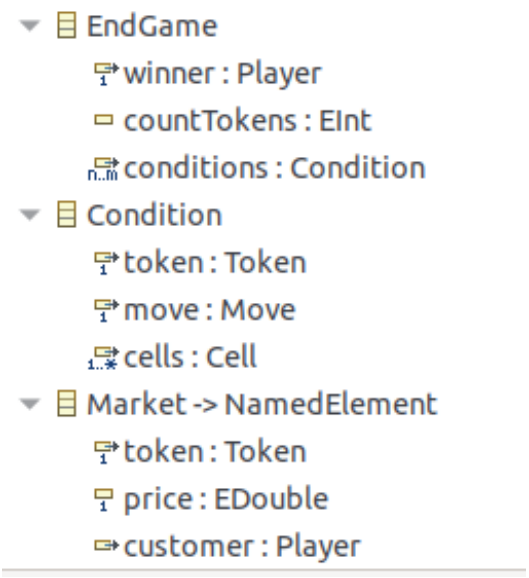

Figura 3.4 Características detalladas de EndGame, Condition y Market.

de muchas maneras, como por ejemplo el encriptado del fichero json o incluso modificar el bolcado y añadirlo a una base de datos local y/o remota ya que Python lo permite fácilemente añadiendo librerias al actual código.

Adicionalmente, se ha añadido un código de validación que aumenta la optimización del modelo y los juegos. No se han añadido restricciones ocl debido a que se estudiaron al final del desarrollo y al ser un entorno sencillo no quisimos modificar la estructura, pero en un futuro desarrollo o un enfoque más elaborado sería interesante añadir las restricciones mediante ocl.

En la validación, se han añadido comprobaciones sencillas que permiten generar un modelo funcional. Algunas de estas restricciones es comprobar si existe Board (tablero del juego) si el juego es de tipo mesa o Market (mercado del juego) si el juego es tipo comunio. Seguramente, las validaciones y comprobaciones que se deban realizar en el modelo en futuras extensiones siempre modifiquen a las actuales, por lo que es una parte del modelo que estará en continuo cambio.

Capítulo 4

Ejemplos de juegos generados

A continuación se muestran varios ejemplos que se han desarrollado utilizando el entorno proporcionado. En concreto, vamos a hablar del juego del gato y el ratón (capítulo 4.1), el juego de las damas (capítulo 4.2) y el juego tipo comunio (capítulo 4.3).

4.1. Juego del gato y el ratón

El primer ejemplo consiste en el juego del gato y el ratón. Este juego está formado por dos jugadores, donde uno de ellos es el ratón y el otro dispone de cuatro gatos. El ratón únicamente se puede mover en dirección noroeste y noreste, mientras que los ratones solo pueden moverse en dirección sureste y suroeste. El juego finaiza cuando el ratón es encerrado, es decir, los gatos consiguen cerrarle el paso (ganando los gatos) ó cuando el ratón consiga llegar al final del tablero. El código de generación se encuentra en el Anexo 2 y su backup en el Anexo 5.

Figura 4.1 Inicio de sesión en el juego.

Como resultado, se obtiene una aplicación, donde lo primero que nos vamos a encontrar es una ventana de inicio de sesión. Al ser un juego de mesa sencillo donde no nos interesa que se registren jugadores, no muestra laopción de registro, pero en el caso del juego tipo comunio si la muestra.

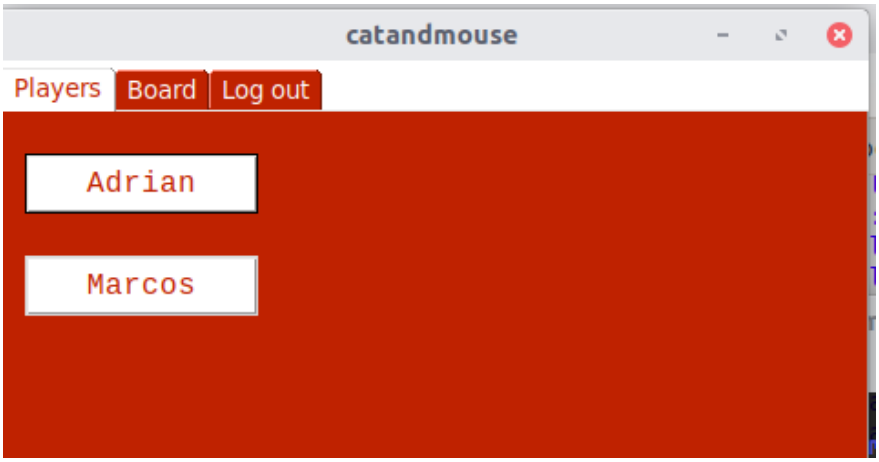

Figura 4.2 Menú donde se muestran los jugadores.

Al iniciar sesión en un juego de mesa sencillo, nos encontramos con tres posibles ventanas. La primera de ellas nos permite visualizar el nombre de los jugadores que forman el juego. Al hacer click en un nombre, se muestra las estadísticas del jugador (en juegos de mesa sencillo se mostrarán el número de fichas restantes mientras que en juegos tipo comunio se muestra la puntuación, los items seleccionados y el valor del monedero).

Figura 4.3 Comienzo del juego mostrando el tablero.

La segunda opción de ventana corresponde al tablero. En ella, nos indica el turno del jugador correspondiente y un botón de recarga. El jugador del que disponga el turno, puede pulsar sobre sus fichas y se mostrarán los posibles movimientos que puede realizar en un color más oscuro que su propia ficha. Al pulsar sobre las fichas del jugador rival o vacías donde no permita movimientos el juego no se actualiza, es decir, solamente se va a actualizar el tablero si el jugador dispone del turno y pulsa sus fichas o una posición del tablero donde se pueda mover.

Figura 4.4 Final del juego mostrando el tablero.

El juego avanza conformemente a las reglas descritas en su declaración. Al finalizar el juego, la aplicación genera un mensaje de aviso donde nos informa de su finalización. Adicionalmente, sustituye el turno del jugador por un mensaje de finalización y bloquea el tablero, de tal forma que cualquier interacción con el no produce ninguna actualización.

Figura 4.5 Cerrar sesión en el juego.

Para finalizar, la última ventana nos permite cerrar sesión y volver a la ventana principal del juego, donde otro usuario puede iniciar sesión y seguir con el juego.

4.2. Juego de las damas

El segundo juego desarrollado es el famoso juego de las damas. En este juego, dos jugadores con 12 fichas cada uno compiten para acabar con las fichas del contrario. Existen dos tipos de finales: un jugador se queda sin fichas y gana el rival ó tablas. En el entorno generado, la opción de acabar en tablas no se ha implementado a excepción de que únicamente quede una ficha a cada jugador. El código de desarrollo de este juego se encuentra en el Apendice 3 y su backup en el Anexo 6.

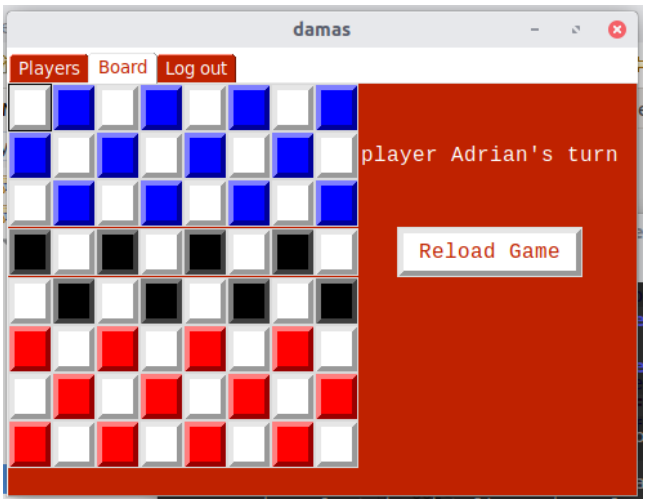

Figura 4.6 Tablero de las damas al inicio del juego.

El resultado obtenido es similar al juego del ratón y el gato, con la misma ventana de login, de vista de los jugadores y de cerrar sesión. Al disponer de los mismos elementos, únicamente cambia el tablero y las condiciones de final del juego.

En este juego se han habilitado las opciones eat y next del modelo para que permita que unas fichas coman a otras del rival siempre y cuando la casilla siguiente este libre, ubicando la ficha en dicha posición.

Al igual que el anterior ejemplo, el tablero muestra únicamente los movimientos permitidos, teniendo en cuenta la propiedad eat, next, los bordes del tablero y las esquinas. En la figura 4.9 se muestra un ejemplo donde el movimiento hacia la izquierda no se muestra ya que la propiedad next obliga a que la siguiente celda esté libre.

El juego finaliza de la misma forma que el anterior. Muestra un mensaje de aviso al jugador donde especifica la finalización del juego y el ganador. Si el usuario no se encuentra

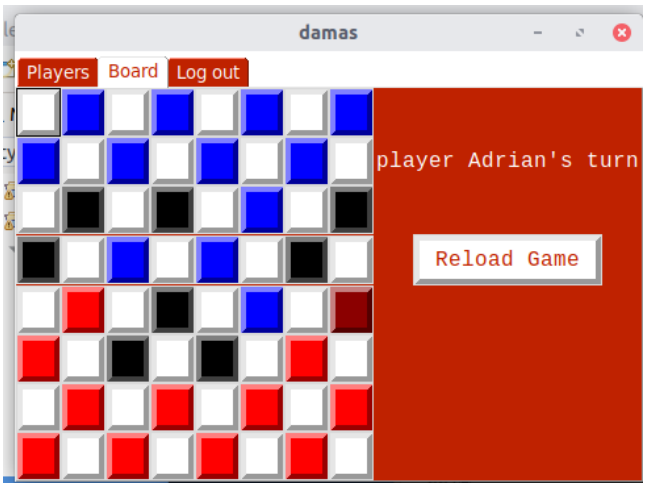

Figura 4.7 Tablero de las damas mostrando movimientos.

en ese mismo momento logeado en el juego, se mostrará la notificación al iniciar sesión. Además, bloquea el tablero y modifica el turno del juego por un mensaje de finalización.

4.3. Juego tipo comunio

El último juego desarrollado se trata de un juego tipo comunio. Este juego se trata de un juego colaborativo de futbol, donde los jugadores compiten para obtener la mejor puntuación. Esta puntuación es manejada por un administrador. Para aumentar la puntuación, los jugadores deben adquirir tokens en el mercado y seleccionarlos de cara a la jornada de puntuación. Durante la jornada de puntuación, los jugadores no podrán cambiar los tokens seleccionados. El administrador determina las nuevas puntuaciones de los tokens, pudiendo subir puntos o incluso disminuirlos en función de la jornada. El código de este ejemplo se encuentra en el Apéndice 4 y su backup en el Anexo 7.

Al contrario que los anteriores ejemplos, este tipo de juego se puede considerar infinito. Por ello, se ha implementado la opción de registro, permitiendo que nuevos usuarios ingresen en cualquier momento. Esta ventana de registro es similar a la de inicio de sesión. Los nuevos usuarios dispondrán de una puntuación inicial de cero y un monedero del mismo valor de monedero por defecto.

Al registrar un jugador, la aplicación genera un backup donde incluye los nuevos datos. Una vez realizado el backup, accede a la ventana principal. En la ventana principal del juego, nos encontramos con un número mayor de pestañas que en juegos de mesa. Las únicas ventanas que comparten son la de inicio de sesión, listado de jugadores y cerrar sesión (añadiendo en este tipo de juegos el número máximo de tokens y de tipos).

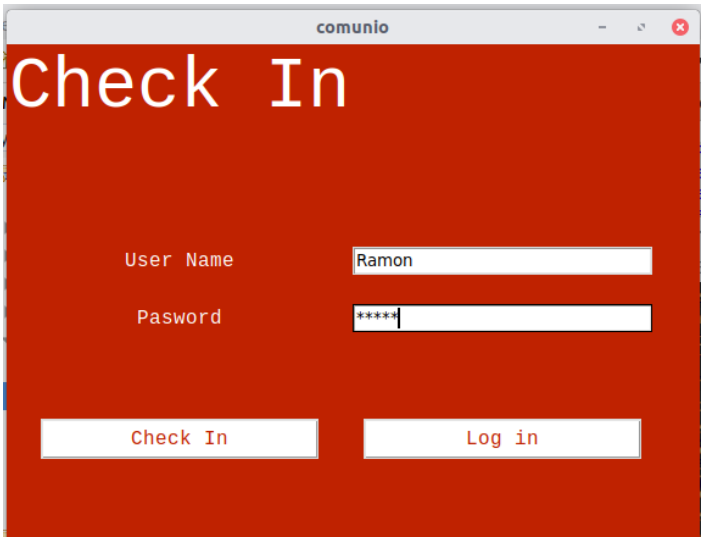

Figura 4.8 Ventana de registro.

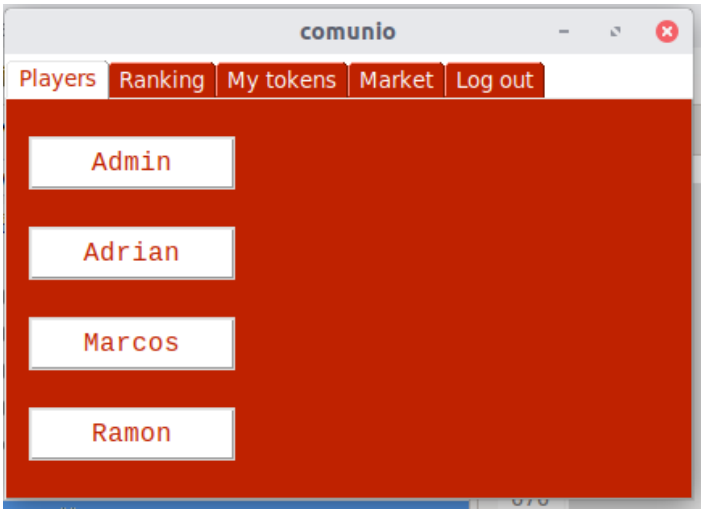

Figura 4.9 Ventana de jugadores con el nuevo usuario.

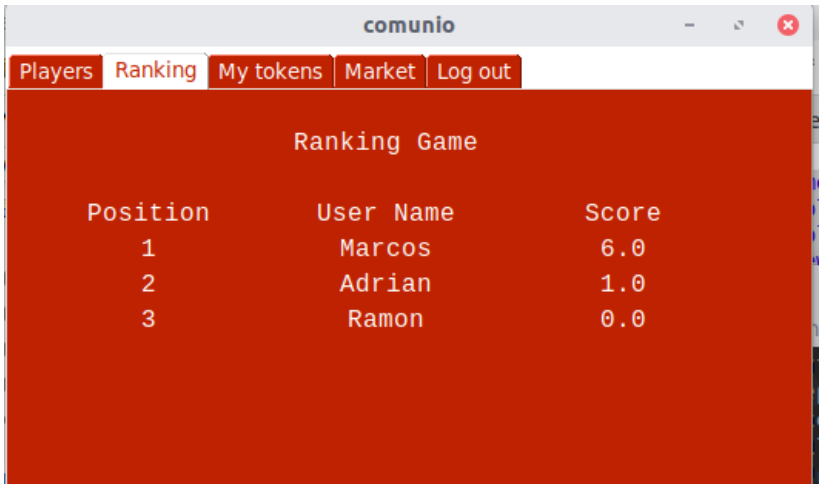

Figura 4.10 Ranking de jugadores.

Una de las nuevas pestañas que se incluyen es el ranking. En el ranking se muestra la información actualizada de la puntuación de los jugadores. Permite analizar la clasificación y determinar quien es el ganador momentaneo del juego. Esta información se actualizará en los periodos de puntuación cuando lo decida el administrador.

Figura 4.11 Ventana de tokens con enfoque de usuario.

De cara a la puntuación, los jugadores pueden seleccionar los items que formen parte de su plantilla en la ventana My tokens. A la hora de seleccionar los tokens, se debe tener en cuenta el límite establecido en cada tipo y en el número máximo entre todos. Al intentar sobrepasar uno de estos valores, la aplicación muestra un mensaje de error y prohibe la operación. En el caso de que se inicie sesión como administrador, dispondrá de otra ventana que será descrita posteriormente.

En la pestaña del mercado, encontramos dos enfoques diferentes: el de administrador y el de usuario. En el caso del usuario, nos encontramos con la posibilidad de añadir nuevos tokens al mercado, fijando su nombre, el tipo, la puntuación hasta el momento y el precio. Además,

Figura 4.12 Ventana de mercado de administrador.

puede finalizar las ofertas actuales del mercado. En el caso de que no exista comprador, se eliminará la oferta del mercado de la aplicación. En caso de que exista un comprador, se añadirá el token a su lista de tokens y se borrará la oferta.

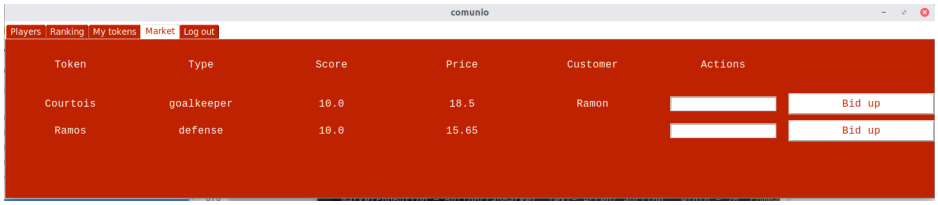

Figura 4.13 Ventana de jugadores con el nuevo usuario.

En el enfoque de usuario, se pueden ver todas las ofertas que existen en el mercado y pujar por ellas. A la hora de la puja, comprueba que el precio puede ser asumido por el comprador y que la puja es mayor que el precio actual. En caso de que no pueda ser asumido, se muestra un mensaje de error. Una vez que se ha pujado por un token, el mercado se actualiza y muestra el nuevo precio del token junto a su nuevo comprador.

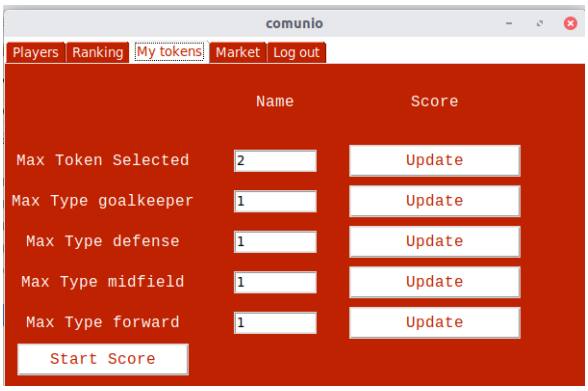

Figura 4.14 Pantalla del administrador antes de iniciar la jornada de puntuación.

Como se ha mencionado anteriormente, la ventana My tokens en el caso de ser administrador también tiene un enfoque diferente. En esta pestaña, el administrador puede personalizar

las plantillas de los jugadores modificando los valores de número máximo de tokens por plantilla y máximo número de tokens de un cierto tipo. Además, puede iniciar el periodo de puntuación, donde la pestaña volverá a cambiar radicalmente.

Figura 4.15 Ventana de score del administrador.

Al inicializar la jornada de puntuación, esta misma pestaña se modifica y muestra un listado con los tokens que disponen de los jugadores junto a su puntuación. El administrador se encarga de modificar las nuevas puntuaciones y de cerrar la jornada. Al modificar la puntuación de un token, si el jugador lo ha seleccionado se actualiza de forma inmediata en el ranking.

Capítulo 5

Conclusiones y Trabajo Futuro

A lo largo de este capítulo, se presentan en la sección 5.1 las conclusiones de la memoria en las que se resumen las soluciones y los resultados del trabajo que se ha realizado. En el apartado 5.2, se analiza el entorno y se presentan una serie de futuras contribuciones que se pueden añadir al entorno desarrollado.

5.1. Conclusiones

El principal objetivo de esta tesis es el desarrollo de un entorno para la generación de juegos colaborativos, incluyendo un DSL sencillo para su manipulación. El enfoque de este entorno es generar juegos colaborativos de forma automática utilizando una sintaxis fácil e intuitiva, permitiendo que cualquier persona (ya tenga conocimientos de programación o no) pueda desarrollar juegos. Los tipos de juegos a los que está enfocado este entorno son los juegos tipo comunio o juegos de mesa sencillos. Este entorno facilita las herramientas necesarias para el correcto compilado del modelo o la generación de un ejecutable funcional en Python y optimizado al modelo entre muchas otras. Además, proporciona herramientas de validación y una gestión de errores dentro del propio modelo, consiguiendo de esta forma que cualquier juego compilado sea compatible con el meta-modelo del entorno.

Adicionalmente, esta tesis incluye tres ejemplos que ponen a prueba el entorno. El primero de ellos se trata de un juego sencillo denominado el ratón y el gato. Este juego esta formado por dos jugadores, donde uno dispone de 4 fichas (los gatos) y otro solo de una (el ratón). El juego finaliza cuando los gatos consiguen cerrar el paso al ratón, es decir, el ratón no puede avanzar, o cuando el ratón llega al final del tablero y se proclama ganador. Los gatos únicamente pueden desplazarse en dirección sureste o suroeste, mientras que el ratón puede desplazarse en dirección noreste y noroeste.

El segundo juego generado es el popular juego de las damas. Este juego esta formado por 2 jugadores, donde ambos tienen 12 fichas pero de diferentes colores (blancas y negras por ejemplo). Este juego finaliza cuando un jugador se queda sin fichas, permitiendo eliminar fichas del rival cuando se encuentran en la diagonal y detrás de ella no se encuentra otra ficha.

El último juego desarrollado es un juego tipo comunio. Este juego puede estar formado por un número ilimitado de jugadores, donde cada uno de ellos dispone de tokens propios. Estos tokens deben ser adquiridos en el mercado del juego y se utilizan para calcular la puntuación en el juego. El juego se encuentra manejado por un administrador, que se encarga de validar las compras en el mercado, establecer el número máximo de tokens y sus tipos que van a formar parte de la puntuación, es el encargado de empezar y finalizar la jornada de puntuación. El juego tiene una duración ilimitada y pueden ingresar nuevos jugadores en cualquier momento.

5.2. Trabajo futuro

Analizando el pequeño entorno desarrollado, hay una gran cantidad de extensiones futuras. Entre todas las posibles extensiones, se proponen tres ideas que permiten al entorno ser más completo.

En primer lugar, podemos extender el entorno actual y reutilizar ciertas propiedades para elaborar juegos de mesa más complejos, como por ejemplo el ajedrez o el monopoli. En el caso del ajedrez, podriamos reutilizar el tipo de los tokens y extender los movimientos que se pueden realizar para añadir un diccionario [clave, valor] donde la clave sea el tipo de token y el valor el movimiento permitido para ese tipo de ficha. En el caso del monopoli, podríamos extender el entorno añadiendo tipos en las casillas (como cárcel inicio, suerte, etc) de la misma forma que está en los tokens, podriamos añadir también un atributo de cantidad en los tokens del jugador para determinar los billetes siendo por defecto -1 y significando no declarado, y para finalizar también podriamos añadir a la clase Game dos listas de mazos (suerte y comunidad), donde cada mazo esta formado por un String y su correspondiente acción (por ejemplo, mover la ficha a la carcel).

En segundo lugar, podemos extender el entorno desarrollado añadiendo juegos tipo FIFA Ultimate Team, donde se simulen los partidos. Prácticamente este tipo de juego comparte las mismas características que el juego tipo comunio, pero en vez de esperar a que el administrador fije el inicio y el final de la puntuación y el mismo modifique esos puntos, los partidos se simularán en base al nivel de equipo, puntuación global de los jugadores, potencial defensivo, potencial en el centro del campo, potencial ofensivo, nivel de entrenador y un

factor suerte (para que los equipos más pequeños también puedan competir). Este factor suerte puede ser, por ejemplo, que dos números aleatorios entre 1 y 50 coincidan y que aumente en x cantidad cualquier valor del equipo, por ejemplo el ofensivo. Además, el mercado pasaría de ser manejado por un administrador a ser manejado por los propios usuarios, quedando a disposición del administrador el manejo de los sobres que pueden comprar los jugadores. Estos sobres pueden ser implementados de la misma forma que el mercado y el administrador puede añadir en ellos una serie de tokens (jugadores) de los cuales se eligen algunos de ellos de forma aleatoria. El saldo del que disponen los jugadores incrementaria de forma constante en cada partido. Este saldo puede ser fijo independientemente gane o pierda, puede favorecer más a los jugadores que pierden para igualar siempre los equipos, pueden favorecer más a los equipos ganadores o pueden añadir las tres opciones y que el administrador de la aplicación elija.

En último lugar, podemos simplemente actualizar y mejorar el entorno actual. Por ejemplo, el juego tipo comunio implementado hace que dependa el mercado y la puntuación del administrador del entorno y que los jugadores comiencen con un saldo fijo que nunca va a aumentar, ya que únicamente se reducirá cuando compren tokens en el mercado. En esta futura extensión, podemos dotar al entorno de un algoritmo en el que de forma automática los tokens del mercado finalicen a los 2 días, por ejemplo, y que la jornada de puntuación empiece de forma automática los viernes y finalice los lunes. En cuanto al saldo del que disponen los jugadores, podemos incrementarlo de forma automática dependiendo de la puntuación obtenida o dejarlo en manos del administrador. En el caso de que se quiera hacer de forma automática,si queremos favorecer por ejemplo a los que menos puntuación tienen, podemos multiplicar la puntuación obtenida por 0.3 a los jugadores que se encuentren por debajo de la quinta posición, por 0.15 a los jugadores que se encuentren quintos y cuartos y por cero a los 3 primeros.

Bibliografía

[1] *Eclipse Sirius website*. urlhttps://www.eclipse.org/sirius/.

[2] *EMFText website*. urlhttps://marketplace.eclipse.org/content/emftext.

[3] *Graphical Modeling Framework (GMF)*. urlhttps://www.eclipse.org/gmf-tooling/.

[4] *A Graphical Tooling Infraestructure (Graphiti) website*. urlhttps://www.omg.org/adm/.

[5] *Python website*. urlhttps://www.python.org/doc/essays/blurb/.

[6] *Spoofax website*. urlhttps://www.metaborg.org/en/latest/.

[7] *Eclipse Modeling Framework*. urlhttps://www.eclipse.org/modeling/emf/, 2020.

[8] *CSound*. urlhttps://csound.com/, 2021.

[9] *Mathematica*. urlhttps://www.wolfram.com/mathematica/, 2021.

[10] Brambilla, M., Cabot, J. y Wimmer, M.: *Model-Driven software engineering in practice, Second edition*. 2017.

[11] Group, O. M.: *Object Constraint Language*. 2014.

[12] L., B.: *mplementing domain specific languages with Xtext and Xtend - Second edition*. https://www.eclipse.org/Xtext/, 2016.

[13] Molina, J. G., Rubio, F. O. G., Pelechano, V., Vallecillo, A., Vara, J. M. y Vicente-Chicote, C.: *Desarrollo de software dirigido por modelos*. Ra-Ma, España, 2013.

[14] Volter, M.: *DSL Engineering: Designing, Implementing and Using Domain-specific Languages*. dslbook, 2013.

Anexo A

DSL generado

```
1
2  grammar game.Game with org.eclipse.xtext.common.Terminals
3
4  import "platform:/resource/game.models/model/models.ecore"
5  import "http://www.eclipse.org/emf/2002/Ecore" as ecore
6
7  Game returns Game:
8
9    'Game' name=EString ':'
10
11     'id' id=EInt
12     ('rowBoard' rowBoard=EInt)?
13     ('columBoard' columBoard=EInt)?
14     ('eat' eat=EBoolean)?
15     ('next' next=EBoolean)?
16     ('turn' turn=EInt)?
17     ('maxTokenSelected' maxTokenSelected=EInt)?
18     ('timeToScore' timeToScore=EInt)?
19     ('admin' admin=[Player|EString])?
20     ('players' ':' players+=Player ( players+=Player)* )?
21     ('board' ':' board+=Cell ( board+=Cell)* )?
22     ('endGame' ':' endGame+=EndGame ( endGame+=EndGame)* )?
23     ('market' ':' market+=Market ( market+=Market)* )?
24     ('maxTokenTypeSelected' ':' maxTokenTypeSelected+=TokenTypeSelected ( maxTokenTypeSelected+=
           TokenTypeSelected)* )?
25     ;
26
27
28 EString returns ecore::EString:
29   STRING | ID;
30
31 EInt returns ecore::EInt:
32   '-'? INT;
33
34 Player returns Player:
35   'Player' name=EString ':'
36     'id' id=EInt
```

```
37      'password' password=EString
38      ('score' score=EInt)?
39      ('wallet' wallet=EDouble)?
40      ('selectedTokens' ':' selectedTokens+=[Token|EString] ( "," selectedTokens+=[Token|EString])* )?
41      ('tokens' ':' tokens+=Token ( tokens+=Token)* )?
42    ;
43
44  Cell returns Cell:
45    'Cell' name=EString ':'
46      'id' id=EInt
47      'color' color=colors
48      ('token' token=[Token|EString])?
49    ;
50
51  EndGame returns EndGame:
52    'EndGame' ':'
53      ('countTokens' countTokens=EInt)?
54      'winner' winner=[Player|EString]
55      ('conditions' ':' conditions+=Condition ( conditions+=Condition)* )?
56    ;
57
58  EBoolean returns ecore::EBoolean:
59    'true' | 'false';
60
61  Market returns Market:
62    'Market' name=EString ':'
63      'id' id=EInt
64      'price' price=EDouble
65      ('customer' customer=[Player|EString])?
66      'token' token=Token
67  ;
68
69  TokenTypeSelected returns TokenTypeSelected:
70    'TokenTypeSelected' ':'
71      'type' type=tokenType
72      'max' max=EInt
73  ;
74
75  Token returns Token:
76    'Token' name=EString ':'
77      'id' id=EInt
78      ('color' color=colors)?
79      ('score' score=EDouble)?
80      ('type' type=tokenType)?
81      ('move' move=Move)?
82  ;
83
84  EDouble returns ecore::EDouble:
85    '-'? INT? '.' INT (('E'|'e') '-'? INT)?;
86
87  enum colors returns colors:
88            yellow = 'yellow' | red = 'red' | green = 'green' | blue = 'blue' | white = 'white' | black = 'black';
89
90  Move returns Move:
91    'Move' ':'
```

```
92      'north' north=EInt
93      'northeast' northeast=EInt
94      'east' east=EInt
95      'southeast' southeast=EInt
96      'south' south=EInt
97      'southwest' southwest=EInt
98      'west' west=EInt
99      'northwest' northwest=EInt
100   ;
101
102 enum tokenType returns tokenType:
103           goalkeeper = 'goalkeeper' | defense = 'defense' | midfield = 'midfield' | forward = 'forward';
104
105 Condition returns Condition:
106   'Condition' ':'
107      'token' token=[Token|EString]
108      'cells' ':' cells+=[Cell|EString] ( "," cells+=[Cell|EString])*
109      'move' move=Move
110 ;
```

Anexo B

Juego del gato y el ratón

```
1
2  Game catandmouse:
3    id 1
4    rowBoard 8
5    columBoard 8
6    eat false
7    turn 1
8    admin Adrian
9    players:
10     Player Adrian:
11       id 1
12       password adrian
13       tokens :
14         Token mouse:
15           id 1
16           color red
17           move Move:
18             north 0
19             northeast 1
20             east 0
21             southeast 0
22             south 0
23             southwest 0
24             west 0
25             northwest 1
26     Player Marcos:
27       ...
28
29   board:
30     Cell cell1:
31       id 1
32       color white
33     Cell cell2:
34       id 2
35       color black
36       token "Marcos.cat1"
37     ...
```

```
38      Cell cell61:
39        id 61
40        color black
41        token "Adrian.mouse"
42      ...
43      Cell cell64:
44        id 64
45        color white
46
47  endGame:
48      EndGame:
49        winner Adrian
50        conditions :
51          Condition:
52            token "Adrian.mouse"
53            cells: cell2, cell4, cell6, cell8
54            move Move:
55              north 0
56              northeast -1
57              northwest -1
58              ...
59      EndGame:
60        winner Marcos
61        conditions :
62          Condition:
63            token "Adrian.mouse"
64            cells: cell9, cell11, cell13 ,cell15, cell18, cell20,
65                 cell22, cell24, cell25, cell27, cell29,
66                 cell31, cell34, cell36, cell38, cell40,
67                 cell41, cell43, cell45, cell47, cell50,
68                 cell52, cell54, cell56
69            move Move:
70              north 0
71              northeast -1
72              northwest -1
73              ...
```

Anexo C

Juego de las damas

```
1
2  Game damas:
3    id 1
4    rowBoard 8
5    columBoard 8
6    eat true
7    next true
8    turn 1
9    admin Adrian
10   players:
11     Player Adrian:
12       id 1
13       password adrian
14       tokens :
15         Token token1:
16           id 1
17           color red
18           move Move:
19             north 0
20             northeast 1
21             east 0
22             southeast 0
23             south 0
24             southwest 0
25             west 0
26             northwest 1
27         ...
28     Player Marcos:
29       id 2
30       password marcos
31       tokens:
32         Token token13:
33           id 13
34           color blue
35           move Move:
36             north 0
37             northeast 0
```

```
38                east 0
39                southeast 1
40                south 0
41                southwest 1
42                west 0
43                northwest 0
44            ...
45    board:
46      Cell cell1:
47        id 1
48        color white
49      Cell cell2:
50        id 2
51        color black
52        token "Marcos.token13"
53      ...
54      Cell cell63:
55        id 63
56        color black
57        token "Adrian.token1"
58      Cell cell64:
59        id 64
60        color white
61    endGame:
62      EndGame:
63        countTokens 0
64        winner Marcos
65
66      EndGame:
67        countTokens 0
68        winner Adrian
```

Anexo D

Juego tipo comunio

```
1
2  Game comunio:
3    id 1
4    maxTokenSelected 2
5    admin Admin
6    players:
7      Player Admin:
8        id 4
9        password administrador
10       score 0
11       wallet 0.0
12     Player Adrian:
13       id 1
14       password adrian
15       score 1
16       wallet 0.0
17       selectedTokens: Oblack
18       tokens:
19         Token Oblack:
20           id 99
21           score 10.0
22           type goalkeeper
23
24         Token Obama:
25           id 99
26           score 10.0
27           type goalkeeper
28
29     Player Marcos:
30       id 2
31       password marcos
32       score 6
33       wallet 12.5
34
35   market:
36     Market market1:
37       id 1
```

```
38        price 15.65
39        token Token Courtois:
40          id 1
41          score 10.0
42          type goalkeeper
43
44      Market market2:
45        id 1
46        price 15.65
47        token Token Ramos:
48          id 1
49          score 10.0
50          type defense
51
52  maxTokenTypeSelected :
53      TokenTypeSelected:
54        type goalkeeper
55        max 1
56
57      TokenTypeSelected:
58        type defense
59        max 1
60
61      TokenTypeSelected:
62        type midfield
63        max 1
64
65      TokenTypeSelected:
66        type forward
67        max 1
```

Anexo E

Backup del juego del gato y el ratón

```
1
2
3  {
4    "id": 1,
5    "rowBoard": 8,
6    "columBoard": 8,
7    "turn": 1,
8    "admin": {
9        "name": "Adrian",
10       "id": 1,
11       "password": "adrian",
12       "tokens": [
13         {
14           ...
15         }
16       ]
17   },
18   "players" : [
19     {
20       "name": "Adrian",
21       "id": 1,
22       "password": "adrian",
23       "tokens": [
24         {
25           "name": "mouse",
26           "id": 1,
27           "score": 0.0,
28           "color": "red"
29           ,"type": "goalkeeper"
30           ,"move": {
31             ...
32           }
33         }
34       ]
35     },
36
37     ...
```

```
38    ]
39    ,"board" : [
40      {
41        "name": "cell1",
42        "id": 1,
43        "color": "white"
44      },
45      {
46        "name": "cell2",
47        "id": 2,
48        "color": "black"
49        ,
50        "token": {
51          "name": "cat1",
52          "id": 2,
53          "color": "blue",
54  +     "move": {
55              ...
56          }
57        }
58      },
59      ...
60    ]
61    ,"endGame" : [
62      {
63        "winner" : "Adrian",
64        "countTokens" : 0
65        ,"conditions": [
66          {
67            "token": "mouse",
68            "move": {
69              ...
70            },
71            "cells" : [
72              "cell2"
73              ,
74              "cell4"
75              ,
76              "cell6"
77              ,
78              "cell8"
79            ]
80          }
81        ]
82      },
83      ...
84    ]
85  }
```

Anexo F

Basckup del juego de las damas

```
1
2  {
3    "id": 1,
4    "rowBoard": 8,
5    "columBoard": 8,
6    "turn": 1,
7    "maxTokenSelectedType": [
8    ],
9    "admin": {
10       "name": "Adrian",
11       "id": 1,
12       "password": "adrian",
13       "score": "-1",
14       "wallet": 0.0,
15       "tokens": [
16         ...
17       ]
18   },
19   "eat": 1,
20   "next": 1,
21   "players" : [
22     {
23       "name": "Adrian",
24       "id": 1,
25       "password": "adrian",
26       "selectedTokens": [
27       ],
28       "tokens": [
29         {
30           "name": "token1",
31           "id": 1,
32           "color": "red"
33           ,"move": {
34             ...
35           }
36         },
37         ...
```

```
        {
          "name": "token12",
          "id": 12,
          "score": 0.0,
          "color": "red"
          ,"type": "goalkeeper"
          ,"move": {
            ...
          }
        }
      ]
    },
    ...
  ]
  ,"board" : [
    {
      "name": "cell1",
      "id": 1,
      "color": "white"
    },
    {
      "name": "cell2",
      "id": 2,
      "color": "black"
      ,
      "token": {
        "name": "token13",
        "id": 13,
        "color": "blue",
        "move": {
          ...
        }
      }
    },
    ...
  ]
  ,"endGame" : [
    {
      "winner" : "Marcos",
      "countTokens" : 0
    },
    ...
  ]
}
```

Anexo G

Backup del juego tipo comunio


```
{
  "id": 1,
  "maxTokenSelected": 2,
  "timeToScore": 0,
  "maxTokenSelectedType": [
    {
      "type": "goalkeeper",
      "max": 1
    },
    ...
    {
      "type": "forward",
      "max": 1
    }
  ],
  "admin": {
      "name": "Admin",
      "id": 4,
      "password": "administrador",
      "score": "0",
      "wallet": 0.0,
      "tokens": [
      ]
  },
  "eat": 0,
  "next": 0,
  "players" : [
    {
      "name": "Admin",
      "id": 4,
      "password": "administrador",
      "score": "0",
      "wallet": 0.0,
      "selectedTokens": [
      ],
      "tokens": [
```

```
        ]
      },
      {
        "name": "Adrian",
        "id": 1,
        "password": "adrian",
        "score": "1",
        "wallet": 0.0,
        "selectedTokens": [
          {
            "name": "Oblack",
            "id": 99,
            "score": 10.0,
            ,"type": "goalkeeper"
          }
        ],
        "tokens": [
          {
            "name": "Oblack",
            "id": 99,
            "score": 10.0,
            ,"type": "goalkeeper"
          },
          ...
        ]
      },
      ...
    , "market": [
    {
      "token": {
        "name": "Courtois",
        "id": 1,
        "score": 10.0,
        ,"type": "goalkeeper"
      },
      "price": 15.65,
      "customer": ""
    },
    ...
    ]
}
```


UAM
UNIVERSIDAD AUTONOMA
DE MADRID