Sistema multiagent aplicat a l'extracció de coneixement de videojocs MOBA

Pol Major Munich

Grau d'Enginyeria Informàtica

Intel·ligència Artificial

David Isern Alarcón

Carles Ventura Royo

02/01/2018

FITXA DEL TREBALL FINAL

Títol del treball:	*Sistema multiagent aplicat a l'extracció de coneixement de videojocs MOBA.*
Nom de l'autor:	*Pol Major Munich*
Nom del consultor/a:	*David Isern Alarcón*
Nom del PRA:	*Carles Ventura Royo*
Data de lliurament (mm/aaaa):	*01/2018*
Titulació o programa:	*Grau d'Enginyeria Informàtica*
Àrea del Treball Final:	*Intel·ligència Artificial*
Idioma del treball:	*Català*
Paraules clau	*Sistema multiagent.* *Extracció de coneixement.* *Multiplayer Online Battle Arena (MOBA).*

Resum del Treball (màxim 250 paraules): *Amb la finalitat, context d'aplicació, metodologia, resultats i conclusions del treball*

Aquest treball es basa en l'aplicació de tècniques de *Big Data* i mineria de dades a un conjunt de dades sobre un videojoc MOBA, el *League of Legends*. Per això, s'utilitza un sistema multiagent que és capaç de, sense intervenció humana, interactuar amb la API de *Riot Games* per extreure dades rellevants, processar-les, analitzar-les i presentar els resultats obtinguts en format text i Web (*html*).

La utilització d'agents permet, entre altres coses, dissenyar un sistema descentralitzat, autònom i amb una alta cohesió. D'altra banda, per a les tasques de mineria i anàlisi, s'han elegit dues eines molt potents: R i *Splunk*.

Com a resultat, s'obtenen un conjunt de regles d'associació que descriuen les

relacions entre els campions utilitzats a les partides i les seves victòries i derrotes. Entre aquestes, se n'ha trobat una que ha permès validar-ne els resultats, ja que descriu un cas oficialment acceptat. Així doncs, el sistema supera tots els jocs de proves i mostra resultats d'allò més interessants. No obstant, el projecte té encara moltes línies de futur en les quals aprofundir.

En conclusió, s'han obtingut regles rellevants que, d'una banda, ajudaran als desenvolupadors del joc a anivellar-lo i, de l'altre, als jugadors per a què facin les millors eleccions de campions possibles.

Abstract (in English, 250 words or less):

This project is based on the application of Big Data and data mining techniques to a set of data from the video game League of Legends (MOBA). For that, a multiagent system is used, which is capable to interact with the Riot Games API without any human intervention. Then, it does extract, process and analyze the data, presenting the results on text and Web format (*html)*.

Using agents allows, among other things, design a decentralized and autonomous system with a high cohesion. Furthermore, a couple of powerful tools have been chosen to analyse the data: R and Splunk.

As a result, a set of association rules are obtained. These rules, describe the relations between the used champions in a game and their victories and defeats. Among the rules, there is one that describes an officially accepted case, thus validating all the other results. Therefore, the system surpasses all the tests, showing pretty interesting results. However, the project still has many lines of future in which deepen.

In conclusion, there have been obtained relevant rules that may help the developers to balance the game. Also, the players may also be benefited, since they might choose better champion combinations.

Índex de continguts

Llista de figures

1. INTRODUCCIÓ

1.1 CONTEXT I JUSTIFICACIÓ DEL TREBALL

La temàtica escollida per aquest treball és l'aplicació de tècniques de la intel·ligència artificial en l'extracció de coneixement sobre videojocs MOBA[1] (Multiplayer Online Battle Arena), concretament mitjançant un SMA[2] (Sistema MultiAgent).

Aquest tipus de videojocs es caracteritzen per:

- Implicar diferents jugadors en una mateixa partida en línia.
- Moltes partides diferents però de relativa curta duració.

Com que aquests tipus de jocs evolucionen constantment, normalment degut a actualitzacions, les dades queden obsoletes ràpidament. Per tant, cal extreure constantment noves dades i mirar d'inferir-ne nou coneixement. L'ús d'agents serveix per a l'automatització i optimització d'aquest procés.

Actualment, existeixen una gran quantitat de videojocs MOBA. Alguns exemples en serien: Dota 2, Smite, League of Legends o Clash Royale. Aquest treball es centra principalment en el League of Legends[3] (LoL), deixant oberta la possibilitat d'ampliar-lo a altres jocs. Per fer-ho, utilitza la API que ofereix Riot Games[4].

La idea és disposar d'un agent encarregat d'extreure dades rellevants sobre els milions de partides que es juguen diàriament a tot el món. Després, altres agents les demanen i analitzen, per extreure'n coneixements útils per l'experiència dels jugadors.

Alguns dels fets destacables que s'espera trobar durant l'anàlisi són:

- Sinèrgies entre campions.
- Campions de moda entre els usuaris.
- Informació rellevant per equilibrar l'experiència de joc.

Més endavant, es detalla un exemple concret que servirà com a joc de proves.

1.2 OBJECTIUS DEL TREBALL

1.2.1 OBJECTIUS GENERALS

- Introduir el món dels videojocs MOBA.
- Investigar i estudiar els sistemes multiagent i la seva implementació en JADE[5].
- Introduir la mineria de dades[6] i els algorismes que utilitza.
- Estudiar les possibles interaccions entre agents i programes que ofereixen algorismes de mineria de dades (Splunk[7], R[8]). Estudiar la opció d'implementar els algorismes directament.
- Extreure coneixement útil i constantment actualitzat per als jugadors a través de mètodes estadístics i de mineria de dades sobre grans conjunts de dades.
- Estudiar els resultats, treure'n conclusions i proposar millores futures.

1.2.2 OBJECTIUS ESPECÍFICS

- Implementar un primer agent que interaccioni amb la API de Riot Games, capaç d'extreure'n dades rellevants i guardar-les, tot esperant peticions d'altres agents. Aquestes dades són:
    - Dades estàtiques del joc.
    - Estadístiques dels campions.
    - Dades específiques de cada jugador.
    - Dades específiques de cada partida jugada.
- Implementar diferents agents analitzadors, especialitzats en una tasca específica:
    - Anàlisis estadística.
    - Mineria de dades:
        - Algorismes descriptius.
        - Algorismes classificadors o predictius.
        - Algorismes d'agregació.

- Els agents han de poder comunicar-se entre ells per optimitzar la cerca de coneixement útil.

Com a mínim s'ha d'implementar l'agent principal (gestor de dades), un agent d'anàlisis estadística i un agent de mineria descriptiva (*apriori*)[9]. Aquests són doncs, els punts principals del treball.

Després, de manera opcional i sempre depenent del temps disponible, s'implementaran més agents que afegiran funcionalitats al sistema, com ara altres algorismes de mineria (classificació, agregació). La planificació del treball es detalla més endavant a l'apartat 1.4.

1.3 ENFOCAMENT I MÈTODE SEGUIT

1.3.1 PER QUÈ UN SMA?

Un SMA ofereix les característiques necessàries per a resoldre el problema plantejat:

- La capacitat de gestió, extracció i comunicació de dades en temps real.
- L'alta comunicació entre agents permet optimitzar els recursos.
- Cada agent es pot especialitzar en la seva tasca concreta. D'aquesta manera, un agent pot optar per demanar una informació a un altre agent enlloc d'haver d'inferir-la ell mateix.
- El coneixement descobert per un agent pot servir per a què un altre agent es faci noves preguntes.

1.3.2 SMA VS MÈTODES TRADICIONALS

Si bé és cert que el problema pot ser resolt mitjançant altres mètodes (com ara mineria de dades en programació orientada a objectes), s'ha decidit implementar un SMA per múltiples raons.

La primera, és que ofereix de forma natural una alta cohesió i un baix acoblament entre agents. És a dir, permet especialitzar cada agent en una petita tasca concreta. Facilita doncs, la separació entre l'extracció i manteniment de les dades i els diferents tipus d'anàlisi.

A més, també de forma natural, cada agent pot comunicar tot allò que sigui necessari a altres agents. Així, un agent podrà fer peticions concretes a un altre agent, optimitzant el procés d'anàlisi entre tots.

Després, i pel que fa a l'actualització de les dades, l'agent encarregat de demanar-les pot fer-ho segons els seus propis criteris. Aquests poden ser: demandes d'altres agents, falta de dades d'un camp determinat, renovació de dades antigues (per exemple, de més de dues setmanes), etc.

Finalment, també servirà per estudiar les capacitats d'un SMA en aquest àmbit, per tal de veure realment què poden oferir, a la pràctica, aquest tipus de sistemes.

Per tot això, s'ha considerat més adequada la implementació d'un SMA com a resolució del problema plantejat.

1.4 PLANIFICACIÓ DEL TREBALL

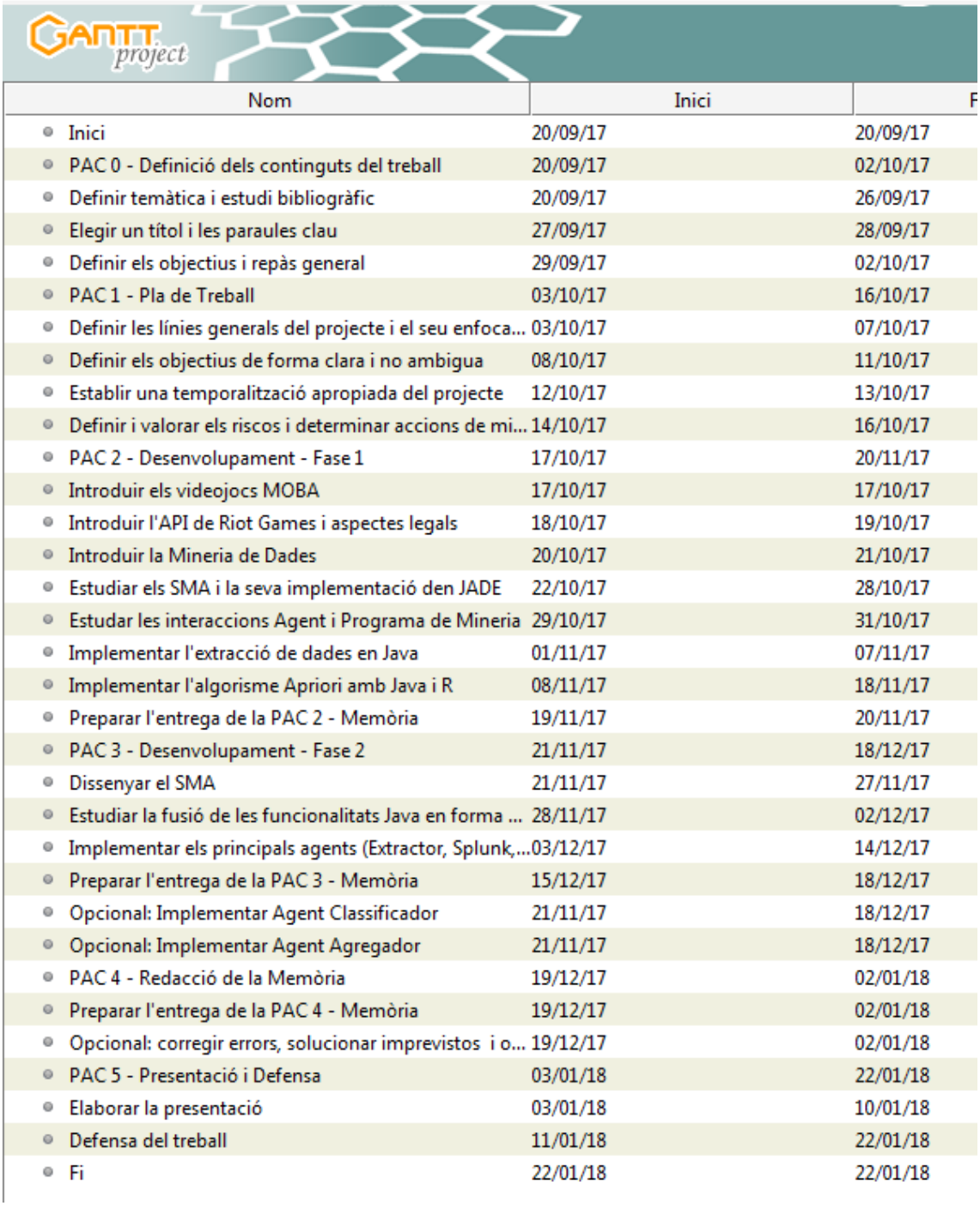
1.4.1 DIAGRAMA DE GANTT

Aquest treball es divideix en 6 PACs [0,5], totes elles amb uns objectius i unes dates d'inici i fi. Utilitzant-les com a base general per a la planificació, s'ha elaborat un diagrama de Gantt per a aquest projecte.

Durant la primera fase, s'espera arribar a implementar el primer agent, *ExtractionAgent*, encarregat de gestionar les dades. Es dedica un temps considerable l'estudi de nous conceptes.

En la segona fase, l'objectiu central és la implementació del l'agent *apriori*, *AprioriAgent*, i de l'agent estadístic, *StatisticAgent*. A més, com s'ha mencionat anteriorment, s'implementaran altres agents de forma opcional, depenent de

l'avanç del projecte. Durant la fase de redacció de la memòria, s'ha reservat un petit període de temps per a solucionar possibles imprevistos, corregir errors i optimitzar el SMA, opcionalment i sempre que sigui necessari.



Nom	Inici	F
Inici	20/09/17	20/09/17
PAC 0 - Definició dels continguts del treball	20/09/17	02/10/17
Definir temàtica i estudi bibliogràfic	20/09/17	26/09/17
Elegir un títol i les paraules clau	27/09/17	28/09/17
Definir els objectius i repàs general	29/09/17	02/10/17
PAC 1 - Pla de Treball	03/10/17	16/10/17
Definir les línies generals del projecte i el seu enfoca...	03/10/17	07/10/17
Definir els objectius de forma clara i no ambigua	08/10/17	11/10/17
Establir una temporalització apropiada del projecte	12/10/17	13/10/17
Definir i valorar els riscos i determinar accions de mi...	14/10/17	16/10/17
PAC 2 - Desenvolupament - Fase 1	17/10/17	20/11/17
Introduir els videojocs MOBA	17/10/17	17/10/17
Introduir l'API de Riot Games i aspectes legals	18/10/17	19/10/17
Introduir la Mineria de Dades	20/10/17	21/10/17
Estudiar els SMA i la seva implementació den JADE	22/10/17	28/10/17
Estudar les interaccions Agent i Programa de Mineria	29/10/17	31/10/17
Implementar l'extracció de dades en Java	01/11/17	07/11/17
Implementar l'algorisme Apriori amb Java i R	08/11/17	18/11/17
Preparar l'entrega de la PAC 2 - Memòria	19/11/17	20/11/17
PAC 3 - Desenvolupament - Fase 2	21/11/17	18/12/17
Dissenyar el SMA	21/11/17	27/11/17
Estudiar la fusió de les funcionalitats Java en forma ...	28/11/17	02/12/17
Implementar els principals agents (Extractor, Splunk,...	03/12/17	14/12/17
Preparar l'entrega de la PAC 3 - Memòria	15/12/17	18/12/17
Opcional: Implementar Agent Classificador	21/11/17	18/12/17
Opcional: Implementar Agent Agregador	21/11/17	18/12/17
PAC 4 - Redacció de la Memòria	19/12/17	02/01/18
Preparar l'entrega de la PAC 4 - Memòria	19/12/17	02/01/18
Opcional: corregir errors, solucionar imprevistos i o...	19/12/17	02/01/18
PAC 5 - Presentació i Defensa	03/01/18	22/01/18
Elaborar la presentació	03/01/18	10/01/18
Defensa del treball	11/01/18	22/01/18
Fi	22/01/18	22/01/18

FIGURA 1. TAULA DE GANTT

Per a la realització d'aquest treball, no són necessaris recursos fora dels temporals. No requereix materials ni cap llicència de pagament.

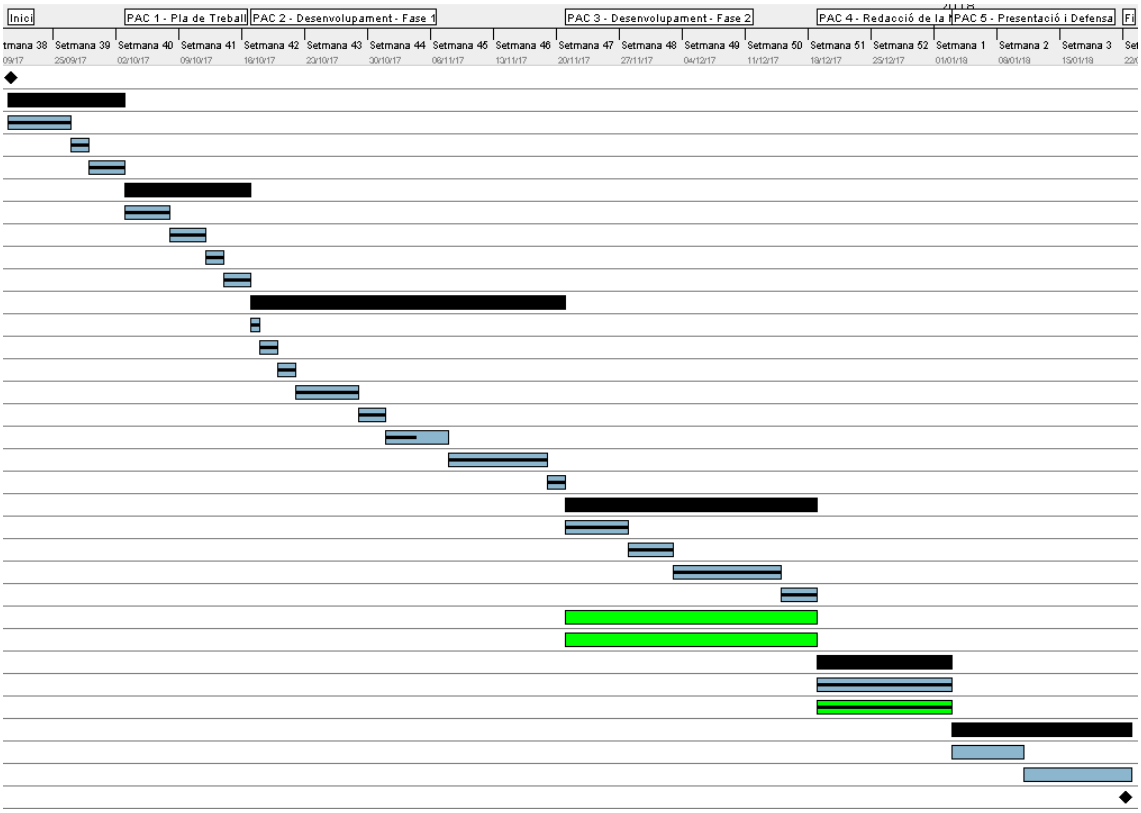


FIGURA 2. DIAGRAMA DE GANTT

En el diagrama:

- Els rombes negres marquen l'inici i el final del treball.
- Els rectangles negres marquen el període de duració de cada PAC.
- Els rectangles blaus marquen les fites a assolir obligatòriament per a l'èxit del projecte. La línia negre que els travessa marca el progrés de cada fita.
- Els rectangles verds marquen les fites opcionals, corresponents a la implementació de noves funcionalitats i gestió d'imprevistos.

Idealment, s'hauria de complir amb les fites marcades. No obstant, sempre cal tenir en compte possibles imprevistos. Per això, s'ha elaborat també un pla de gestió de riscos.

1.4.2 PLA DE GESTIÓ DE RISCOS

Risc	Conseqüències	Probabilitat	Impacte	Mesures correctives
Problemes legals amb l'API de Riot Games.	No es podrien extreure les dades necessàries.	Baixa.	Alt.	Abans de plantejar el treball, es va investigar la viabilitat legal d'utilitzar aquesta API satisfactòriament.
Canvis en la política de Riot Games.	El sistema podria deixar de funcionar correctament.	Mitjana.	Baix/Mitjà	El disseny del sistema ha de permetre una ràpida adaptació si es dóna el cas.
Problemes en la temporització del treball.	El treball no es podria finalitzar a temps.	Mitjana.	Alt.	La planificació té en compte els imprevistos. Per això, conté fases d'implementació opcionals, així com un marge de correcció d'errors.
Incompatibilitat entre agents i programes de Mineria de Dades.	Dificultat d'implementació dels algorismes, incrementant el temps necessari.	Baixa.	Alt.	Al utilitzar JADE (Java), la probabilitat d'aquest risc és molt baixa. En tot cas, s'ha previst un estudi previ d'alternatives.
Mala interpretació de l'abast del treball.	El projecte podria quedar-se curt o, d'altra banda, llarg.	Alta.	Mitjà	La segona fase del desenvolupament és flexible al canvi, segons els resultats de la primera fase.

FIGURA 3. TAULA DE GESTIÓ DE RISCOS

1.5 JOCS DE PROVES

Per confirmar que el sistema funciona s'han dissenyat unes proves que haurà de passar una vegada sigui implementat. Les característiques principals són:

- L'*ExtractionAgent* és capaç d'extreure les dades rellevants i d'oferir-les sota la demanda d'altres agents.
- L'*StatisticAgent* és capaç de demanar dades (segons el seu criteri) i d'inferir-ne coneixement estadístic.
- L'*AprioriAgent* implementa l'algorisme amb el mateix nom. Aquest algorisme permet crear regles d'associació amb un cert grau de rellevància a partir d'unes dades d'entrada. Aquestes dades es demanaran, principalment, a l'*ExtractionAgent*.

El sistema ha de ser capaç de resoldre els problemes següents:

1.5.1 PROBLEMA DE SINÈRGIES

Es vol saber quines sinèrgies existeixen entre els campions del joc. A cada partida, s'utilitzen 5 campions diferents per equip, fent un total de 10. Només un dels dos equips guanya la partida; no és possible empatar.

1. L'*ExtractionAgent* ha extret prou dades de partides recents, com per inferir-ne coneixement rellevant. Llavors, separa els campions de l'equip guanyador i perdedor en dues taules que contenen: (idPartida, idCampió). Aquesta tasca la duu a terme el mateix agent perquè és qui disposa de les dades, segons indica el patró expert.
2. L'*AprioriAgent* es disposa a treballar. Primerament, fa una petició a l'*ExtractionAgent* per obtenir les taules de campions. A continuació, aplica l'algorisme *apriori* i n'obté regles d'associació. Si es comparen els resultats de guanyadors i perdedors, es donaran dos casos:
    a. Les regles d'associació són molt diferents entre guanyadors i perdedors. És el cas ideal, ja que indica fortes sinèrgies (positives o negatives) entre campions.

b. Les regles obtingudes no varien de forma rellevant entre guanyadors i perdedors, cosa que indica una manca de sinèrgies. Això és, que importen més altres factors (com ara l'habilitat del jugador).

1.5.2 PROBLEMA DELS CAMPIONS DE MODA

Es busquen els campions més i menys utilitzats en l'entorn de joc actual. És justificada aquesta elecció?

1. L'*ExtractionAgent* disposa de prous dades sobre partides recents.
2. L'*StatisticAgent* demana totes les dades sobre les partides jugades a l'*ExtractionAgent*. Llavors, ja pot calcular el percentatges d'utilització i de victòria per a cada campió. Es poden donar dos casos:
    a. L'ideal, on el joc es troba equilibrat i s'utilitzen tots els campions per igual amb resultats similars. Aquest seria l'objectiu perseguit, però difícilment mai assolit, per els desenvolupadors del joc.
    b. El més probable, on es detecten fortaleses i debilitats diverses entre campions. En aquest cas, són els campions més utilitzats també els que més percentatge de victòria tenen?
        i. No, els usuaris tenen un comportament d'elecció que sembla aleatori o, si més no, respon a altres motivacions. L'agent pot donar per solucionat el problema o, d'altra banda, considerar cercar per altres vies.
        ii. Sí i, per tant, l'agent dóna per solucionat el problema: l'elecció dels usuaris és justificada.

1.5.3 BUSCANT L'EQUILIBRI

En l'apartat anterior (1.5.2) s'han detectat dos campions (X,Y) amb un percentatge d'utilització i de victòria massa elevat. Per a la bona experiència de l'usuari, s'ha

decidit realitzar actualitzacions per mirar d'equilibrar la situació actual del joc. Per on s'hauria d'atacar el problema?

1. S'utilitza l'*AprioriAgent* per detectar sinèrgies. Concretament, es busquen les relacionades amb (X,Y):

    a. Es detecta una forta sinèrgia positiva per (X,Y). En aquest cas, cal estudiar la relació d'ambdós campions i prendre mesures col·lectives.

    b. No es detecten sinèrgies positives per (X,Y). En aquest cas, la força de (X,Y) no està relacionada, i cal estudiar-los i prendre mesures de forma individual.

1.6 BREU SUMARI DE PRODUCTES OBTINGUTS

Fitxers:

- *timestamp.txt*, *matchIdHistory.txt, matchIds.txt* i *accIds.txt*, que contenen el temps de vida actual del servidor, els identificadors de les partides i els dels comptes d'usuari, respectivament.
- *winners.csv* i *losers.csv,* que contenen l'identificador de la partida i el llistat de campions segons si guanyen o perden la partida, respectivament.
- *winRules.html* i *loseRules.html*, que representen les regles obtingudes amb l'algorisme *apriori* aplicat per l'agent *AprioriAgent*.
- *winInteractive.html* i *loseInteractive.html*, que mostren un gràfic descriptiu de les regles anteriors.
- *problematicChamps.txt* i *checkedRules.txt*, amb les dades dels campions detectats com a més problemàtics.

Producte:

- Sistema multiagent, implementat amb JADE, que s'encarrega de mantenir les dades actualitzades i d'inferir-ne coneixement.

1.7 BREU DESCRIPCIÓ DELS ALTRES CAPÍTOLS DE LA MEMÒRIA

A continuació, es detalla el procés d'estudi del problema i implementació de la solució.

A l'apartat 2, es comença explicant la base teòrica sobre la que es sustenta el projecte, que comprèn els àmbits dels videojocs MOBA, l'ús d'APIs, la mineria de dades, la gestió de Big Data, els SMA i la seva implementació en JADE. També, es mostrarà com es relacionen tots aquests elements a través del disseny del sistema.

Després, a l'apartat 3 es detalla la implementació de la solució proposada. Concretament, les característiques de cada agent i els algorismes que implementen, així com els resultats que s'obtenen i la solució dels jocs de proves definits.

Finalment, a l'apartat 4 es tracten les interaccions entre els agents i la temporització que utilitzen per a executar els algorismes, mentre que a l'apartat 5 es detallen els resultats i les conclusions del treball.

2. ANÀLISI DEL PROBLEMA I LES SOLUCIONS

2.1 INTRODUCCIÓ ALS MOBA

En els videojocs *Multiplayer Online Battle Arena (MOBA)* s'enfronten dos equips d'un o diversos jugadors. Cada equip comparteix un objectiu general comú: derrotar a l'altre equip mitjançant algun tipus d'acció (eliminar l'enemic, destruir la seva base...). Per aconseguir-ho, és necessari cooperar amb els companys i elegir la millor estratègia cap a la victòria. El mapa de joc sol ser limitat (arena) i l'acció quasi immediata.

Tanmateix, una de les característiques més importants dels MOBA és la relativament curta duració de les partides, que sol rondar entre els 3 i els 75 minuts. A més, les tries de jugadors per equip sol seguir un algorisme aleatori basat en l'habilitat dels mateixos. Aquesta, al seu torn, se sol calcular a partir de les estadístiques de l'historial de partides de cada jugador. El nivell d'habilitat se sol reflectir en un número o rang simbòlic (número de trofeus, medalles...). Tot això resulta extremadament atractiu per els jugadors, que gaudeixen del joc sense pressions.

Pel que fa al cas d'aquest treball, aquest sistema de joc brinda una perfecta base de dades de partides esperant a ser analitzada a fons.

Dit això, el funcionament d'una partida qualsevol sol ser el següent:

1. Creació dels equips. S'emparellen els jugadors tal i com s'ha explicat al paràgraf anterior.

2. Cada jugador tria un campió que el representarà. En aquest punt és important elegir una estratègia de joc, buscant una bona combinació de campions.

3. La partida comença i els dos equips s'enfronten, fins que un dels dos assoleix l'objectiu que determina la victòria. També és possible que el resultat final esdevingui un empat.

4. Els jugadors solen rebre alguna recompensa, dins el joc, per haver jugat (experiència, diners...). Això els permet obtenir millores per a futures partides, creant un incentiu per seguir jugant i guanyant.

2.1.1 FUNCIONAMENT DEL *LEAGUE OF LEGENDS (LOL)*

Com s'ha dit, aquest treball es centra en un MOBA en concret: el *League of Legends*. Les característiques específiques d'aquest joc i més rellevants per a aquest treball s'exposen a continuació:

- Hi ha dos equips de 5 jugadors, on cada un controla un campió que s'anomena campió.
- Cada campió té habilitats úniques, les quals s'adquireixen al pujar de nivell. Cada partida es comença al nivell 1, podent arribar com a màxim al 18. Actualment, hi ha un total de 135 campions disponibles, tot i que aproximadament cada mes en solen crear un de nou.
- Les característiques de cada campió inclouen: vida, mana/energia, atac físic, poder d'habilitat, velocitat d'atac, velocitat de moviment, crítics, defensa i resistència a la màgia. Durant la partida, s'aconsegueix experiència (per pujar de nivell) i or, que permet comprar diferents objectes i millorar les característiques del campió. Totes aquestes dades estan disponibles a través de la API de Riot Games, tal i com s'explicarà al següent apartat (2.2).
- Una partida sol durar entre 20 i 45 minuts. Acaba quan un equip aconsegueix destruir el nexe de l'altre. Partida rere partida, el que interessa és tot l'historial que es va creant per a cada jugador.
- Es fa, normalment, una actualització del joc cada 2 setmanes. Aquesta inclou canvis en els campions o en el sistema de joc. Aquest fet implica haver de mantenir actualitzada una base de dades per veure com evoluciona el joc, dia a dia.
- Existeixen diversos servidors, que divideixen els jugadors per regions. Aquí es farà servir el servidor *EUW*, que engloba tota l'Europa occidental. Es tracta d'una mostra prou representativa.

2.2 L'API DE RIOT GAMES

Per tal d'utilitzar la API de Riot Games[4], cal tenir-hi un compte d'usuari. Després, només s'ha de fer *login* i generar una clau.

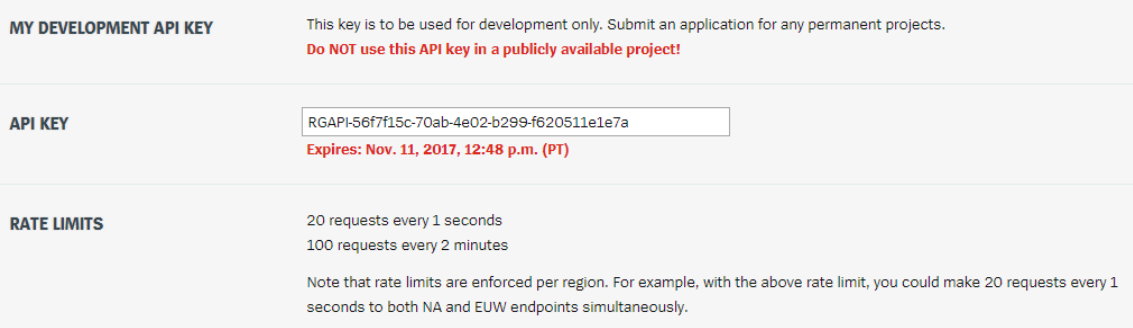


FIGURA 4. CLAU DE L'API PER A DESENVOLUPADORS

Com es veu a la figura anterior, la clau té una data d'expiració. No obstant, només cal regenerar-la quan això passa. Per a obtenir una clau permanent, cal presentar el projecte a Riot Games i no incomplir cap normativa.

L'objectiu que es busca al oferir aquesta API és millorar l'experiència de joc dels jugadors. La idea és donar un conjunt d'eines suficients als desenvolupadors per tal que puguin dur a terme els seus projectes. Per tant, és molt important tenir present aquesta idea i, sobretot, evitar qualsevol projecte que pugui derivar en una experiència negativa per als jugadors.

2.2.1 POLÍTIQUES DE L'API

Totes les polítiques es poden trobar a la web de desenvolupadors, però aquí se'n fa un breu resum:

- Cal protegir la clau de l'API i mai fer-la pública. De moment però, aquí només s'utilitza la clau provisional de desenvolupador, que serveix per iniciar el projecte i fer-ne un prototip. Després, per a fer-lo públic i accessible a la comunitat, cal que sigui revisat i aprovat per Riot Games.
- No s'ha de comprometre la integritat del joc, ni crear avantatges injustos (per exemple, estan prohibits els *scripts* que juguen per si sols).

- Excepte alguns casos molts concrets, no s'ha de buscar monetitzar el projecte, sinó oferir-lo als usuaris de forma gratuïta.
- Sobre la política de dades, el desenvolupador és lliure d'analitzar-les i fer-ne qualsevol tractament, sempre i quan quedi clarament exposat d'on han sortit. Per exemple, si són d'un jugador o d'un campió en concret. L'important és que el context quedi clar.
- Finalment, una de les recomanacions que s'hi fan és la d'avaluar i millorar l'experiència de joc dels jugadors.

Per tant, es pot seguir endavant amb aquest projecte sense problema, ja que no s'està infringint cap normativa; s'està buscant ajudar els usuaris oferint-los-hi coneixements útils (a l'abast de tothom).

2.2.2 DOCUMENTACIÓ DE L'API

Concretament, les dades que s'ofereixen són, entre altres:

- Totes les característiques dels objectes i dels campions.
- Les dades públiques referents als usuaris, incloent tot el seu historial de partides jugades.
- Les partides, disponibles a partir de:
    - El seu número identificador.
    - El compte de l'usuari seleccionat.
- En temps real, el temps de joc d'una partida actual, així com totes les dades de la mateixa.

Les dades de cada partida són:

- Dades generals, com ara tipus de partida, temporada, participants (llista d'usuaris), mapa, equips.
- Dades específiques, entre les quals els campions prohibits i els triats, les torres destruïdes, els monstres èpics morts o la primera sang.

- Estadístiques finals de la partida, que són el nombre de morts, el mal realitzar per cada jugador, les assistències, els objectes, l'or, la puntuació de visió, l'equip guanyador, etc.

2.2.3 EXEMPLE D'UTILITZACIÓ

La sintaxi que segueix és una simple petició "GET" a la referència de l'API corresponent. Per exemple, es vol extreure les dades del campió amb identificador número 23. L'enllaç a seguir és el següent:

https://euw1.api.riotgames.com/lol/static-data/v3/champions/23?locale=en_US &api_key=RGAPI-56f7f15c-70ab-4e02-b299-f620511e1e7a

On:

"euw1.api.riotgames.com" indica el servidor d'Europa occidental i l'API.

"/lol/static-data/v3/champions/23" demana dades estàtiques dels campions, concretament el número 23 (de la versió 3 de l'API).

"?locale=en_US&api_key=RGAPI-56f7f15c-70ab-4e02-b299-f620511e1e7" afegeix paràmetres a la cerca. El més important aquí és "l'api_key=", que assenyala la nostra clau de desenvolupador. Sense la clau, cap petició és possible. És important recordar que aquesta clau és provisional i té data de caducitat, per tant deixarà de funcionar.

La resposta a tota petició és retornada en format JSON. En l'exemple que ens ocupa, s'ha rebut:


```
{
    "title": "the Barbarian King",
    "name": "Tryndamere",
    "key": "Tryndamere",
    "id": 23
}
```


És a dir, el campió demanat és "Tryndamere", el rei dels bàrbars. Si per exemple, se'n vol extreure les característiques, només cal afegir "&tags=stats" a la petició.

```
{
    "stats": {
        "armorperlevel": 3.1,
        "attackdamage": 69,
        "mpperlevel": 0,
        "attackspeedoffset": -0.0672,
        "mp": 100,
        "armor": 33,
        "hp": 625.64,
        "hpregenperlevel": 0.9,
        "attackspeedperlevel": 2.9,
        "attackrange": 125,
        "movespeed": 345,
        "attackdamageperlevel": 3.2,
        "mpregenperlevel": 0,
        "critperlevel": 0,
        "spellblockperlevel": 1.25,
        "crit": 0,
        "mpregen": 0,
        "spellblock": 32.1,
        "hpregen": 8.512,
        "hpperlevel": 98
    },
}
```


2.3 LA MINERIA DE DADES

L'objectiu que persegueix la mineria de dades és el descobriment de coneixement a partir de grans bases de dades. És el procés d'aplicar tècniques d'anàlisi per mirar de crear models que expliquin, de manera útil, uns patrons que descriuen una realitat vàlida, contrastable. Depenent de l'objectiu que es persegueixi, es buscarà un o altre tipus de model de coneixement.

Les fases que sol seguir un procés de mineria solen ser:

- Obtenir i estudiar les dades (anàlisi estadística), per a tal de definir el model de mineria més indicat. És a dir, què és el que es vol buscar o descobrir, a partir del que les dades poden oferir. En aquest sentit, hi ha models descriptius, classificadors, predictius, explicatius i d'agregació de dades.

- Preparar les dades per a l'aplicació de l'algorisme de mineria. Moltes vegades el format de les dades no és el més indicat per a la tasca que es vol dur a terme. Per això, és molt important l'estudi previ i el tractament inicial de les dades obtingudes de la base de dades.

- Construir un o més models de coneixement, interpretar-los i avaluar-ne els resultats. El procés de mineria es tracta d'un procés iteratiu, que no sempre obté resultats satisfactoris: moltes vegades és necessari canviar l'enfocament, estudiar la implementació d'altres models o mirar d'obtenir dades més rellevants.

Normalment, se sol utilitzar algun tipus de programari que facilita aquestes tasques, tot i que també és possible programar-les des de zero. Després d'estudiar-ne diversos, s'ha decidit utilitzar principalment dos programes: R i Splunk. Més endavant, s'entra més en detall en aquest programari i en com interactua amb el sistema.

2.3.1 EXEMPLE DE MINERIA: L'ALGORISME *APRIORI*

Apriori és un algorisme per a trobar regles d'associació dins un conjunt de dades. Aquestes segueixen el format:

"{a,b} → c" o *si a i b, llavors c*.

Cada regla va acompanyada d'un nivell de suport, confiança i pes. S'entén per suport el percentatge total de casos que contenen totes les variables de la regla. D'altra banda, la confiança indica el percentatge de casos en que, essent certa la part esquerra, la regla es compleix. Finalment, el pes és una variant de la confiança i s'obté dividint-la pel suport de la part dreta de la regla.

- Suport = P(a,b,c).

- Confiança = P(c | {a,b}).

- Pes = Confiança/P(c)

Per tant, un pes de 1 indica que les variables són independents entre elles. Un pes elevat, per contra, indica una forta dependència.

2.4 ELS SISTEMES MULTIAGENT

S'entén per agent intel·ligent un sistema que és capaç de rebre i respondre a estímuls de forma relativament racional, autònoma, sempre mirant d'assolir algun tipus d'objectiu. Una de les seves característiques és la capacitat de comunicació amb altres agents. Quan això es dóna, es parla un sistema multiagent (SMA), on diversos agents interaccionen entre ells per tal d'assolir el seus objectius, siguin o no compartits. És a dir, a vegades competiran entre ells, mentre que d'altres treballaran conjuntament.

Així, les característiques principals que ha de tenir un agent són, entre altres:

- Autonomia: una vegada en funcionament, un agent no ha de dependre d'accions humanes, sinó que s'ha de poder valer per ell mateix.

- Comunicació: com s'ha dit, es molt important que un agent pugui comunicar-se de manera eficient amb altres agents del sistema. Per aquesta tasca, existeixen una sèrie de normes de la *Foundation for Intelligent Physical Agents* (FIPA). L'objectiu que persegueix aquesta organització és afavorir la interacció entre agents de qualsevol tipus.

  D'altra banda, un agent també ha de ser capaç de comunicar-se amb humans en cas de necessitat.

- Proactivitat: l'agent ha de ser capaç d'actuar per si sol, prenent la iniciativa, a l'hora d'intentar assolir els seus objectius.

A més, quan diversos agents formen un SMA, apareixen noves característiques interessants:

- Cada agent es pot especialitzar en una tasca determinada i, quan sigui necessari, comunicar-se amb altres agents per compartir informació.

- Un bon SMA és sinèrgic, és a dir el conjunt resultant d'agents pot resoldre problemes que individualment resultarien impossibles.

- Els agents no cal que es trobin en una mateixa localitat, sinó que poden estar distribuïts i interactuar a través de la xarxa.

- Seguint les normes de disseny adequades, com ara les de FIPA, l'abast de comunicació dels agents és de caire mundial.
- És més fàcil de dissenyar petits components individuals que es comuniquin, que no pas un gran sistema complex.

Amb tot, les característiques i els avantatges d'un SMA són ben clares. En aquest projecte, s'utilitzen aquestes capacitats dissenyant un sistema senzill que permet extreure i analitzar dades de forma dinàmica, tal i com s'explica en els següents apartats.

2.5 PROGRAMACIÓ EN JADE

El *Java Agent Development Environment* (JADE), és format per un conjunt de llibreries especialitzades en el desenvolupament i programació de sistemes multiagent. A més, ofereix un entorn que permet executar i provar les interaccions entre els agents, fent-lo ideal per aquest projecte. La versió que s'utilitzarà és la 4.5.0, la qual es pot trobar a la web:

http://jade.tilab.com/

Per utilitzar l'entorn cal executar la classe *jade.Boot* amb l'argument d'entrada -gui. La interfície és força intuïtiva i en facilita molt l'ús. Mitjançant els menús es poden afegir nous agents i plataformes, crear interaccions o un *Sniffer*.

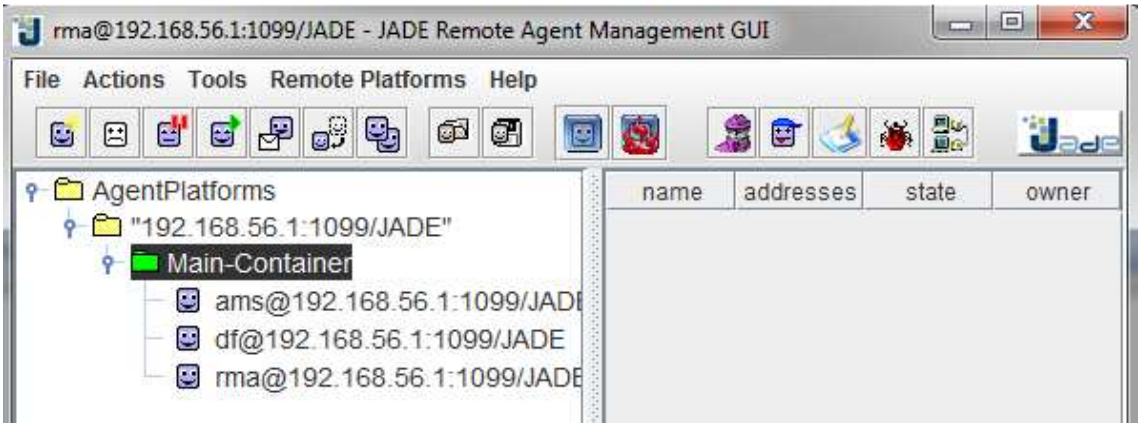


FIGURA 5. ENTORN JADE

En JADE existeixen dos agents principals, anomenats *ams* i *df*, que s'encarreguen de registrar els agents al sistema i d'apuntar-ne els serveis oferts, respectivament.

2.5.1 EXEMPLE D'UTILITZACIÓ DE JADE

Per comprovar el funcionament de l'entorn, s'ha dut a terme una prova disponible amb la mateixa descàrrega del paquet JADE. Es tracta d'un agent comprador i un venedor de llibres. El primer sempre busca la millor oferta, mentre que el segon disposa d'un llistat de productes a la venta.

Els agents s'implementen a través d'una classe Java que estén la classe *Agent*. En l'exemple que ens empara, es tracta de les classes *BookBuyerAgent* i *BookSellerAgent* que es troben al paquet *Jade.examples.bookTrading*. A més, l'agent venedor implementa una petita classe *JFrame* per oferir una finestra on afegir nous llibres amb el seu preu de venta. D'altra banda, l'agent comprador ha de rebre com a argument d'entrada el nom del llibre a comprar. Per afegir un nou agent, posar-li un nom i indicar la direcció de la classe que l'implementa, tal i com es mostra en la imatge següent:

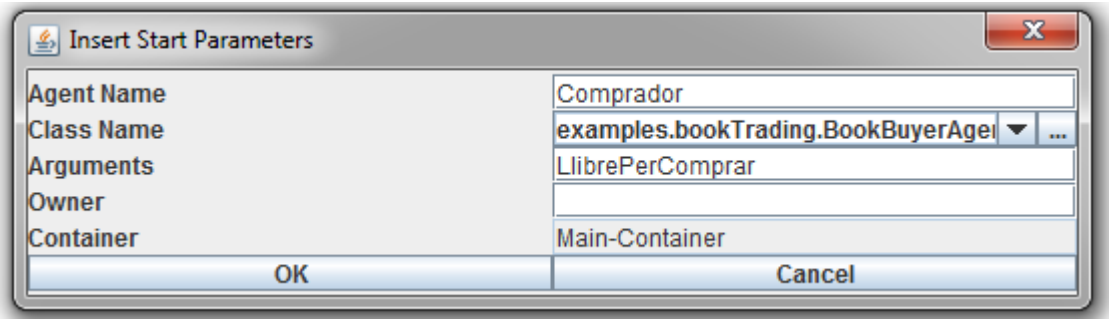


FIGURA 6. AFEGIR UN NOU AGENT

Un cop afegits ambdós agents, es comunicaran entre ells per a assolir els seus respectius objectius: comprar i vendre llibres. L'agent comprador comprova un cop cada minut les diferents ofertes disponibles i, de totes elles, en tria la més barata. El resultat d'aquest procés és:

```
Agent container Main-Container@192.168.56.1 is ready.
---------------------------------------------
LlibrePerComprar inserted into catalogue. Price = 11
Hallo! Buyer-agent Comprador@192.168.56.1:1099/JADE is ready.
Target book is LlibrePerComprar
```

```
Trying to buy LlibrePerComprar
Found the following seller agents:
Venedor@192.168.56.1:1099/JADE
LlibrePerComprar sold to agent Comprador@192.168.56.1:1099/JADE
LlibrePerComprar successfully purchased from agent
Venedor@192.168.56.1:1099/JADE
Price = 11
Buyer-agent Comprador@192.168.56.1:1099/JADE terminating.
```

De manera similar en la d'aquest exemple, s'ha implementat el conjunt d'agents necessaris per aquest projecte.

2.6 INTERACCIÓ AGENTS-*BIGDATA*-MINERIA

El fet d'utilitzar JADE, escrit en Java, obre la porta a moltes opcions a l'hora d'implementar les funcionalitats dels agents. A continuació es detallen les eines que es fan servir, totes elles compatibles amb Java.

2.6.1 SPLUNK

Splunk[7] permet llegir, emmagatzemar i processar tot tipus de dades en temps real, fent-la una eina d'anàlisi de *Big Data* molt potent, ideal per el SMA. Mitjançant informes programables és possible mantenir un conjunt de consultes permanentment actualitzades i accessibles als usuaris. A més, és compatible amb Java. Es tracta, per tant, del model ideal per a tractar i mostrar les dades estadístiques que s'extreguin de l'API. *L'StatisticAgent* és l'encarregat de gestionar aquesta plataforma.

Com que es treballa en un Windows 7, s'utilitza la última versió disponible per a aquest sistema operatiu (6.4.8). Concretament, s'ha instal·lat la versió Free, que permet executar el servidor de Splunk en local de forma gratuïta, per defecte al port 8000. Al manual d'instal·lació que es troba a l'annex se'n donen més detalls.

2.6.2 R PROJECT

R[8] és un programari molt potent utilitzat en l'àmbit d'anàlisis de dades. El que el fa més atractiu és que, al tractar-se de programari lliure, desenvolupadors de tot el món han creat paquets que implementen multitud de funcionalitats interessants a l'abast de tothom.

Així, és possible descarregar i utilitzar paquets que implementen algorismes de mineria de dades, permetent sempre la visualització dels resultats gràficament. Principalment, R serveix l'*AprioriAgent* a través d'*scripts*, el quals poden ser executats fàcilment a través de Java. En aquest cas també, es pot consultar l'annex per veure més detalladament el funcionament i la instal·lació de R.

2.7 DISSENY DEL SMA RESULTANT

Fins ara s'han detallat tots els elements necessaris per a dissenyar el SMA. La millor manera de visualitzar gràficament el sistema resultant, és mitjançant un diagrama de classes UML.

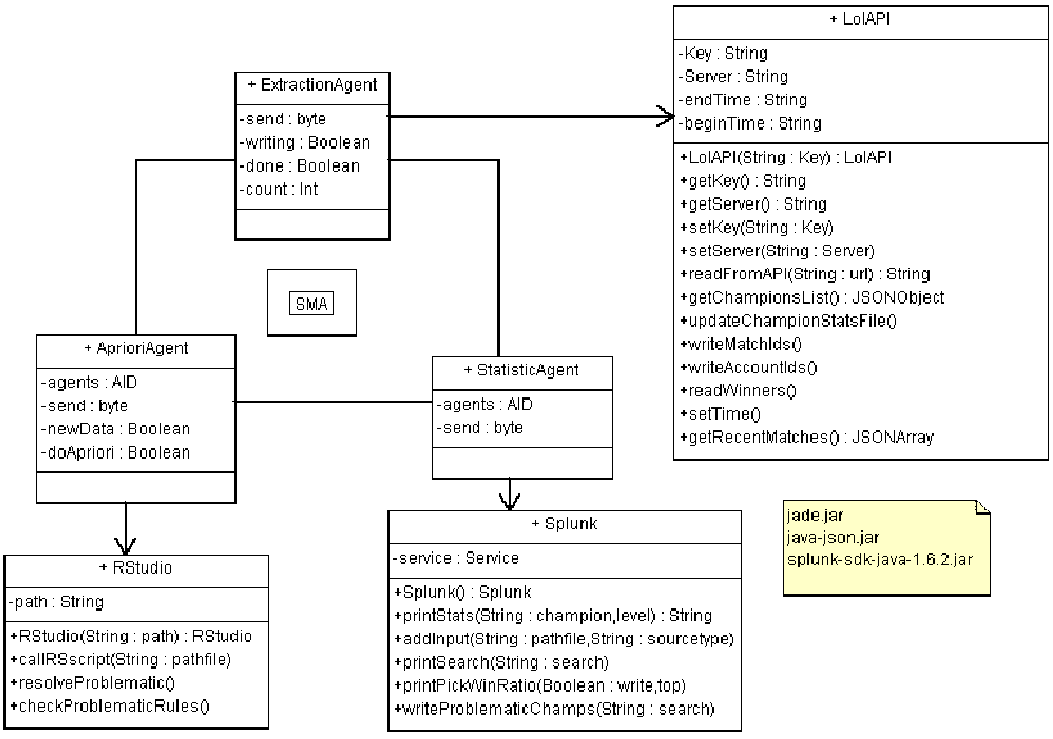


FIGURA 7. DISSENY DEL SMA

3. IMPLEMENTACIÓ DE LA SOLUCIÓ: CARACTERÍSTIQUES DELS AGENTS

3.1 EXTRACCIÓ DE LES DADES: EXTRACTIONAGENT

En l'extracció de les dades és on tot comença. Com s'ha explicat anteriorment, cal interactuar amb l'API de Riot Games, fent-hi peticions i rebent les respostes en format JSON. El primer és doncs, obtenir una clau provisional de l'API i importar la llibreria *java-json.jar*. A l'annex hi ha un manual que detalla aquests procediments. Un cop fet això, tot és qüestió de fer les peticions adequades i tractar-ne les respostes.

L'*ExtractionAgent* utilitza una classe Java, anomenada LolAPI, que implementa totes les funcionalitats necessàries. El constructor d'aquesta classe requereix, com a paràmetre d'entrada, la clau de desenvolupador de l'API.

3.1.1 ALGORISME D'EXTRACCIÓ DE PARTIDES

A continuació es detalla el que és, probablement, l'algorisme més important de tot el sistema: l'extracció de partides rellevants. S'ha dissenyat tenint en compte que ha de complir els requeriments següents:

- Extreure dades actuals, amb un màxim d'una setmana d'antiguitat. Per això, cal detectar el temps actual. El LoL utilitza una marca de temps (*timestamp*) que indica el temps de vida total de cada servidor.
- Només són vàlides les partides classificatòries, la quals s'identifiquen per un codi (420).
- El servidor utilitzat ha de ser el d'Europa occidental (*euw*). La classe LolAPI s'inicialitza per defecte amb aquest servidor, però pot ser modificat mitjançant la funció *setServer()*.

L'algorisme inicia extraient 5 partides aleatòries que s'estan jugant en temps real. D'aquí n'obté la marca de temps actual i en calcula el límit establert d'una setmana. També n'extreu els 50 participants, que amb alta probabilitat seran jugadors actius. De tots ells, n'agafa l'historial de partides (amb un màxim de 100) i les filtra

per codi i temps. De cada partida en guarda la seva identificació, que pot ser utilitzada, posteriorment, per obtenir totes les dades relacionades amb la mateixa.

En pseudocodi:

```
Algorisme ExtreurePartides()
inici
obtenirPartidesActuals()
guardarMarcaTemps()
per cada partidaActual fer
    obtenirJugadorsActius()
fiper
per cada jugadorActiu fer
    filtrarHistorialPartides(temps, codi)
    descartarPartidesRepetides()
    guardarIdentificacióPartides()
fiper
fi
```

Els resultats obtinguts en cada execució varien entre un mínim de 0 i un màxim de 5000 partides rellevants. Això depèn de si els jugadors obtinguts són realment actius, així com del nombre de partides classificatòries que han jugat la última setmana. De mitjana s'aconsegueixen unes 300 partides, amb un marge d'error de ±200.

El cost d'execució es mesura en crides a la API:

- Obtenir les partides actuals té un cost de 1.
- Extreure'n els 50 participants té un cost de 50.
- Filtrar l'historial de partides i guardar-ne l'identificador té un cost de 50.

El cost total és doncs, 1+50+50 = 101 crides a la API. És elevat, però cal tenir en compte les limitacions imposades:

- Com que la clau utilitzada és provisional (de desenvolupador), L'API de Riot Games limita les crides a 100 cada 2 minuts. Això implica haver d'esperar, sempre que sigui necessari, per no sobrepassar els límits establerts. Malauradament, se sobrepassa el límit per 1 crida. Una solució és no llegir l'historial de l'últim jugador. Una altra, és simplement esperar, per defecte, cada 100 crides. S'ha decidit aplicar aquesta segona opció.
- Extreure els participants suposa un cost de 50 degut a que, inicialment, només se'n pot obtenir el seu nom. No obstant, per a consultar l'historial, és necessari l'identificador del compte d'usuari. Obtenir-lo resulta en una crida per cada nom d'usuari, fent un total de 50.

Després d'estar treballant una nit sencera, es pot veure l'evolució de la quantitat de dades obtingudes per cada cicle d'execució.

FIGURA 8. EVOLUCIÓ DE LA QUANTITAT DE DADES EXTRETES AL LLARG DEL TEMPS

Com demostra la imatge, s'observa una forta davallada al llarg de les hores. Això és degut al filtre de partides que s'ha aplicat per evitar les repeticions. Així doncs, arriba un punt on la majoria de les partides obtingudes són repetides i per tant, són descartades. Queda pendent com a futura millora la implementació d'una millor solució a aquest problema.

3.1.2 PREPARACIÓ DE LES DADES

Si la funció principal de l'*ExtractionAgent* és obtenir dades rellevants, també n'és l'encarregat de processar-les i preparar-les per a oferir als altres agents. S'ha decidit fer-ho així ja que, segons el patró d'assignació de responsabilitats Expert, el tractament de les dades li correspon a la classe que en té la informació necessària.

De moment, *ExtractionAgent* implementa un altre algorisme, anomenat *readWinners()*, que llegeix l'identificador de les partides que té guardades i fa crides a la API per obtenir-ne totes les dades. Concretament, consulta quins són els campions que han guanyat la partida i quins els que l'han perduda, tot guardant-los en dues llistes diferents en forma de fitxers csv.

El procés és el següent:

```
Algorisme llegirGuanyadors()
inici
llegirIdentificadorsDelFitxer()
per cada identificador fer
        obtenirPartidaDeAPI(identificador)
        consultarEquipGuanyador()
        consultarCampions()
        escriureLlistes()
        si ((cridesAPI % 100) == 0) llavors
                esperar(120s)
        fisi
fiper
fi
```

El resultat d'aplicar aquest algorisme són dos fitxers *winners.csv* i *losers.csv* que contenen l'identificador de la partida i els campions guanyadors i perdedors, respectivament. El format resultant és:

```
id,champ
3463363495,Jhin
3463363495,Leona
3463363495,Katarina
3463363495,Maokai
3462026830,Nidalee
3462026830,Vladimir
3462026830,Urgot
3462026830,Taric
3462026830,MissFortune
```

El cost és de 1 crida a la API per cada partida llegida. Per tant, s'obtenen aproximadament les dades de 2500 partides cada hora. Altra vegada, això és degut a la limitació imposada per la clau provisional, que només permet 100 crides a la API cada 2minuts. L'algorisme ja preveu aquest fet i espera 120 segons quan s'han assolit les 100 crides.

De moment, la única dada que extreu són els campions, però és possible utilitzar les mateixes crides a la API per obtenir qualsevol altra dada de la partida. No es guarden les partides senceres perquè la quantitat d'informació que contenen és massa gran, i repercutiria en el cost temporal i espaial de l'algorisme. En tot cas, queda com a futura millora del projecte la utilització d'altres dades a més dels campions, així com la implementació d'una base de dades per a guardar-les.

3.1.3 ALTRES ALGORISMES

Per acabar d'explicar la funcionalitat de l'*ExtractionAgent*, cal comentar uns últims algorismes. Hi ha un tipus de dades que es consideren estàtiques, ja que difícilment varien al llarg del temps. La clau provisional està limitada a només 10 crides a dades estàtiques cada 2 minuts.

Així doncs, per una banda s'ha dissenyat un algorisme que extreu les característiques de cada campió. S'anomena *updateChampionStatsFile()* i no s'aconsella executar-lo massa sovint; la idea és fer-ho cada 2 setmanes

aproximadament, que és quan se sol fer una actualització del joc i, per tant, les dades estàtiques poden canviar

D'altra banda, l'algorisme *readWinners()* fa una crida a *getChampionList()* per traduir els identificadors dels campions al seu nom real. Finalment, la marca de temps es guarda al fitxer *timestamp.txt*, sempre que s'executa l'algorisme *getRecentMatches()*. No obstant, cal executar l'acció *setTime()* per llegir la marca de temps del fitxer sempre que sigui necessari.

3.2 L'ALGORISME APRIORI AMB R: APRIORIAGENT

L'*AprioriAgent* utilitza la classe Java RStudio, que li permet interactuar amb el programa R a través de *scripts*. És indispensable, per tant, que l'agent conegui el camí on R està instal·lat. Per això, el constructor de la classe requereix com a paràmetre una cadena de caràcters indicant-ne la localització, per exemple, "C:/R/bin/Rscript.exe".

Després, s'implementa un únic algorisme, *callRScript(pathfile)*, que executa l'*script* de R indicat com a paràmetre d'entrada. Així doncs, la feina d'aquest agent consisteix, bàsicament, en dos passos:

1. Demanar les dades, ja preparades, a l'*ExtractionAgent*. En aquest cas, es tractaria d'obtenir els fitxers *winners.csv* i *losers.csv.*
2. Executar els *scripts* de R corresponents a l'anàlisi que es vol fer. Per exemple, el fitxer *apriori.R*.

La part complicada, però alhora la més interessant, és el contingut dels *scripts* que es facin servir. Als següents apartats s'explica un exemple pràctic de l'execució d'*apriori.R* i dels resultats obtinguts.

3.2.1 EXECUCIÓ D'UN *SCRIPT* DE R: *APRIORI.R*

A continuació es detallen els passos que segueix l'execució de l'*script* encarregat d'aplicar l'algorisme *apriori* als fitxers *winners.csv* i *losers.csv*.

1. Obrir les llibreries de R necessàries. Per una explicació més detallada, es pot consultar l'annex. Concretament, s'utilitzen tres llibreries:
    a. *arules*[10]: encarregada d'aplicar l'algorisme.
    b. *arulesViz:* genera la visualització dels resultats.
    c. *datasets*: permet llegir els fitxers d'entrada en forma de transaccions, pas previ necessari a l'aplicació de l'*apriori*.
2. Determinar el directori de treball.
3. Llegir els fitxers *winners.csv* i *losers.csv*.
4. Generar els models de mineria aplicant *apriori*. S'obtenen doncs, dos models: un amb les regles que descriuen un equip guanyador, i l'altre amb les que descriuen un equip perdedor.
5. Generar els gràfics i guardar-los en format *html*. En aquest punt, es diferencien dos tipus de gràfics:
    a. *Interactive.html*: gràfic descriptiu que relaciona, en més o menys mesura, la relació entre tots els personatges presents en les regles generades (agafant les 100 millors regles).
    b. *Rules.html:* gràfic de dispersió, amb el suport a l'eix X i el pes a l'eix Y. L'ombrejat vermell indica la confiança de la regla.

3.2.2 ELECCIÓ DEL SUPORT I LA CONFIANÇA ADEQUATS

Potser el punt més important a l'hora de generar regles descriptives rellevants amb l'algorisme *apriori*, és l'elecció d'un límit mínim en termes de suport i confiança. Un límit massa baix pot suposar l'aparició de més de 10.000 regles, mentre que un de massa estricte pot arribar a eliminar totes les regles. Els factors que s'han tingut en compte, a l'hora d'aplicar *apriori*, són:

- Generar entre 100 i 500 regles. Aquesta xifra variarà segons les dades, però és una bona xifra on apuntar.
- Buscar regles de només dues variables. És a dir, del tipus (A=>B).

- Càlcul del suport mínim:
    - Actualment, hi ha un total de 139 campions. Partint de la base ideal, on tots els campions es jugarien per igual, es calcula la probabilitat de que dos d'ells en concret es trobin en el mateix equip. Cal tenir en compte que a cada equip hi ha 5 campions.
    - $suport\ mínim = P(A\&B) = \frac{5/2}{\frac{139!}{2!137!}} = \frac{2.5}{9591} \cong 0.00026$
    - No obstant, el joc dista molt d'aquest cas ideal. Per tant, s'ha decidit relaxar força aquest xifra tan baixa, per obtenir regles més realistes.
    - El suport obtingut per les millors parelles és de 0.02, és a dir, es troben al 2% de les partides. Per obtenir un número considerable de regles rellevants, s'ha decidit aplicar el límit de suport al 0.5%.
- Càlcul de la confiança mínima:
    - La principal funció de la confiança en serà el càlcul del pes. Per tant, s'ha utilitzat per a limitar el número de regles obtingudes, sense que aquestes perdin valor. La confiança elegida és del 0.1.

3.2.3 ANÀLISI DELS FITXERS RESULTANTS

La idea de generar fitxers *html* és poder pujar-los a un servidor web accessible. D'aquesta manera, un usuari tindrà sempre accés a la informació més actualitzada, ja que l'agent per si sol s'encarrega de mantenir els gràfics. De moment però, es treballarà en local.

En el punt anterior s'han generat dos tipus de fitxers *html* i, per tant, ambdós es poden executar directament en un navegador (Google Chrome, FireFox, IE...).

Interactive.html:

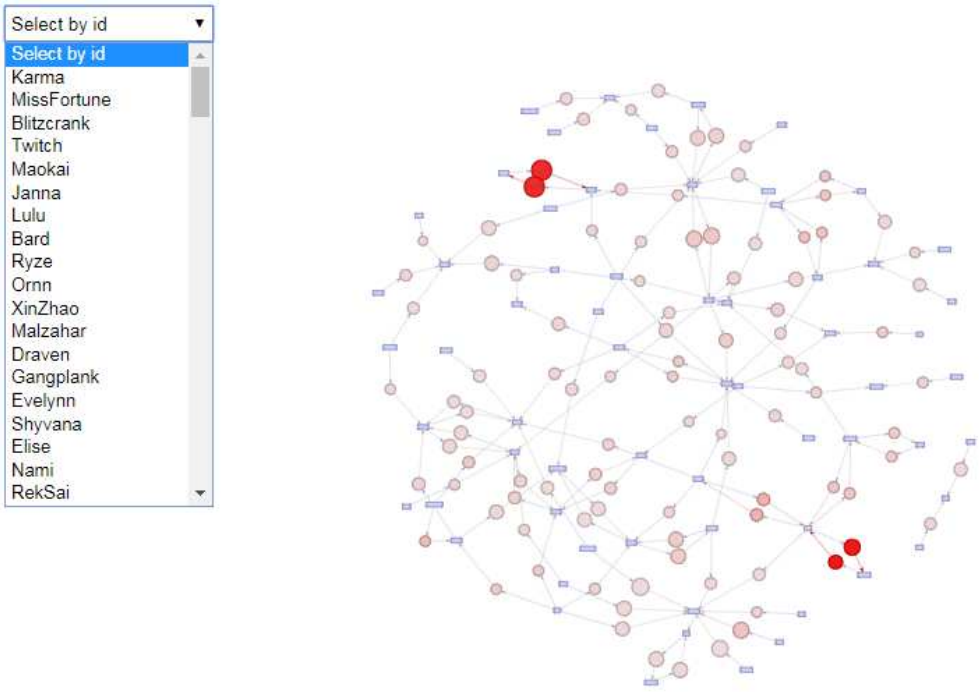


FIGURA 9. GRÀFIC DESCRIPTIU DE LES REGLES *APRIORI (GUANYADORS)*

- Mostra un gràfic descriptiu amb totes les regles generades. La mida dels cercles vermells indica el nivell de suport de la regla, mentre que la intensitat del color n'indica el pes.
- Permet filtrar per campió o per regla. Així, un usuari pot trobar fàcilment aquelles regles que més li interessin.
- És possible fer *zoom* dins del gràfic utilitzant, per exemple, la rodeta del ratolí. Si no es té ratolí, també es pot editar la mida directament. Al fitxer *html*, a la l'última línia, cal editar el: "browser":{"width":960,"height":500}.

Rules.html

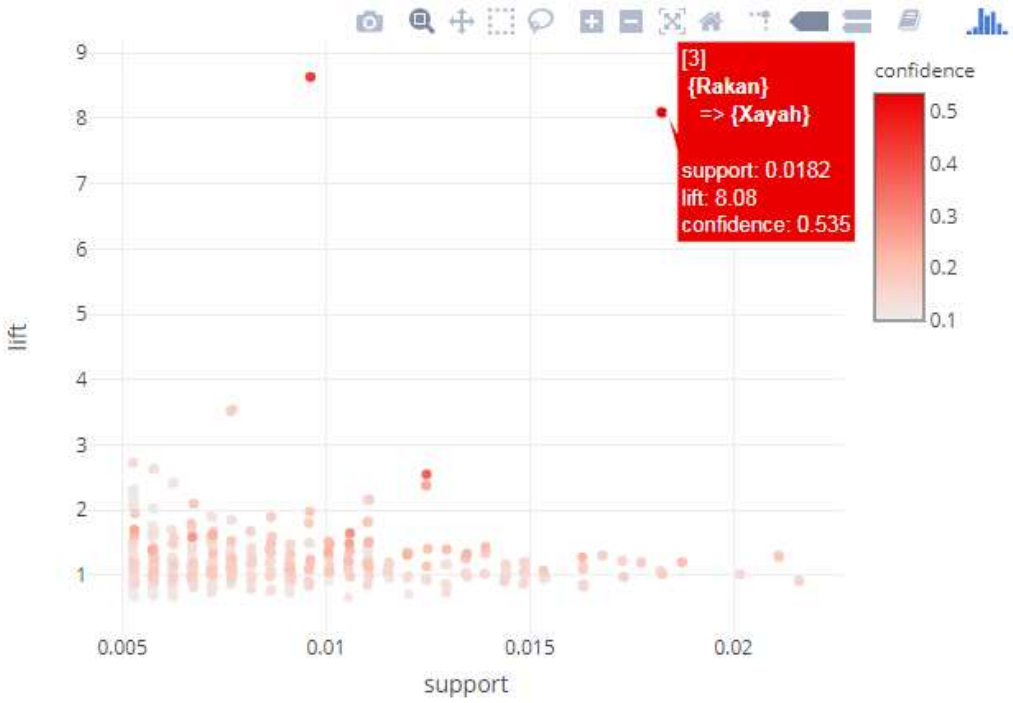


FIGURA 10. GRÀFIC DE DISPERSIÓ DE LES REGLES *APRIORI (GUANYADORS)*

- Conté el gràfic de dispersió de les regles. A més, permet seleccionar cada regla per veure'n els detalls (suport, confiança i pes) en forma numèrica.
- Aquí però, la intensitat correspon a la confiança i sol ser força irrellevant. Cal fixar-se en la posició de la regla: com més pes i suport tingui, més forta és la regla.

3.2.4 DEMOSTRACIÓ: FUNCIONA?

S'hi s'observen els resultats de la figura 9, hi ha dos punts que clarament destaquen de la resta. Un d'ells correspon a la regla: {Rakan} => {Xayah}. La parella té un suport del 1.82%, un dels més elevats que s'han obtingut. A més, té un pes ni més ni menys que de 8.08. Però això, què significa exactament?

1. Que la parella és present, aproximadament, a 1/55 partides en l'equip guanyador. Es tracta d'una xifra molt superior a la mitjana.

2. Que, a més, el fet de que un dels dos membres estigui a la partida, implica que l'altre té fins a un 808% més de possibilitats d'estar-hi també, respecte al que seria normal.

Matemàticament, seria el següent:

$$P(Rakan) = 0.0341$$
$$P(Xayah) = 0.0662$$
$$P(Rakan\&Xayah) = 0.0182 = suport$$
$$confiança(Rakan => Xayah) = \frac{P(Rakan\&Xayah)}{P(Rakan)} = \frac{0.0182}{0.0341} = 0.535$$
$$pes(Rakan => Xayah) = \frac{confiança(Rakan => Xayah)}{P(Xayah)} = \frac{0.535}{0.0662} = 8.08$$

$$confiança(Xayah => Rakan) = \frac{P(Rakan\&Xayah)}{P(Xayah)} = \frac{0.0182}{0.0662} = 0.275$$
$$pes(Xayah => Rakan) = \frac{confiança(Xayah => Rakan)}{P(Rakan)} = \frac{0.275}{0.0341} = 8.08$$

És a dir, si en {Rakan} es troba al 3.41% del total de partides, aquesta xifra augmenta en presència de la {Xayah} fins al 27.5%. D'altra banda, si la {Xayah} és present al 6.62% de les partides, en presència d'en {Rakan} hi és al 53.5%.

Finalment, l'altre punt remarcable fa referència a la regla {KogMaw => Lulu} amb un pes del 8.63 i un suport del 0.0096. Si bé el suport és inferior, el pes no deixa de ser igualment impressionant. Només queda una pregunta per respondre: es pot saber, a l'hora de la veritat, si aquestes conclusions s'apliquen realment? La resposta és, curiosament, sí:

- La {Xayah} i en {Rakan} són dos campions que van ser dissenyats, expressament, per ser sinèrgics entre ells[11]. Al jugar-los junts gaudeixen de petits avantatges que no tenen al jugar per separat. Aquest fet provoca:
    1. Que, independentment dels resultats, els jugadors tendeixin a jugar-los junts.
    2. Que, al jugar-los junts, tendeixin a guanyar més partides de les que perden.
- La {Lulu} i el {KogMaw} van demostrar una forta sinèrgia a les partides disputades als últims tornejos, entre ells el mundial[12]. Per tant, és una

sinèrgia demostrada i explotada, igual que l'anterior, per al conjunt de jugadors.

El fet que el sistema proposat en aquest projecte hagi estat capaç de detectar, tan clarament, aquestes dues parelles, valida per complet els resultats obtinguts.

3.2.5 CÀLCULS FINALS: RESOLUCIÓ DEL PROBLEMA DE SINÈRGIES

Per acabar aquest apartat, el sistema està en condicions de solucionar el joc de proves descrit a l'apartat 1.5.1: problema de sinèrgies.

Si bé al punt anterior s'han detectat fortes relacions entre dues parelles de campions (entre altres), res ens diu que realment siguin sinèrgics. Una opció també vàlida seria que els jugadors, al fer-los creure que la parella funciona bé, decideixin jugar-los junts independentment dels resultats. Si es comparen les regles dels conjunts guanyadors i perdedors. Es poden donar dos casos:

1. La regla existeix en un conjunt, però no en l'altre. Per tant, aquesta indica una forta sinèrgia i el problema està solucionat.

2. La regla existeix als dos conjunts. En aquest cas, s'aplica la fórmula següent:

$$\frac{suport(ReglaGuanyadors) * 100}{suport(ReglaGuanyadors) + suport(ReglaPerdedors)} = \%\ victòria$$

$$En\ el\ cas\ de\ la\ regla\ \{Xayah => Rakan\} = \frac{1.82}{0.0182 + 0.0134} = 57.59\%$$

Si el percentatge és més gran del 50%, existeix una sinèrgia positiva. Si fos molt proper al 50%, es consideraria que el fet de jugar junts no té cap efecte. El problema queda doncs, solucionat.

3.3 SPLUNK: *STATISTICAGENT*

El tercer agent que s'ha decidit implementar, l'*StatisticAgent*, és l'encarregat de l'anàlisi estadística de les dades. Les seves funcions es poden resumir en tres punts:

1. Demanar les dades que l'*ExtractionAgent* té disponibles.
2. Procedir fent-ne un anàlisi estadístic.
3. En cas de trobar alguna estadística interessant, informar-ne a l'usuari o a un altre agent.

3.3.1 COM FUNCIONA?

L'*StatisticAgent* utilitza la classe Java Splunk, que li permet interaccionar directament amb la plataforma *Splunk* instal·lada al sistema. Aquesta, és capaç d'executar cerques sobre les dades. Per entendre'n millor el funcionament, se'n mostra a continuació un exemple pràctic.

L'agent, una vegada establerta la connexió amb la plataforma, li envia una petició de cerca com la següent[13]:

```
"search index=lolsplunk name=Rakan | stats values(stats.hp) as Vida"
```

Llavors, *Splunk* l'executa i en retorna la resposta. En aquest cas, es demana:

- Accedir a l'índex *lolsplunk*, que apunta a les dades disponibles.
- Filtrar per nom, concretament el campió *Rakan*.
- Retornar el valor *stats.hp*, que indica la vida inicial del campió.

De moment, es llegeixen les dades de tres dels fitxers generats per l'*ExtractionAgent*: *winners.csv*, *losers.csv* i *championStats.json*. Al manual d'instal·lació que es troba a l'annex es donen més detalls sobre com pujar i mantenir les dades a la plataforma *Splunk*.

3.3.2 CERQUES ESTADÍSTIQUES IMPLEMENTADES

Bàsicament, l'*StatisticAgent* implementa dos tipus de cerca. La primera utilitza el fitxer *championStats.json* per calcular les estadístiques dels personatges a un nivell en concret. Per fer-ho, es serveix de l'acció *printStats(name, level)*, que té com a paràmetres d'entrada el nom i el nivell del campió desitjat.

D'altra banda, utilitza *winners.csv* i *losers.csv* per fer un càlcul general del percentatge d'utilització i de victòria de tots els personatges. A més, amb la crida a *writeProblematicChamps()*, escriu un fitxer que conté les dades d'aquells personatges que necessiten una revisió. El mètode que s'ha utilitzar és el següent:

- Agafar l'1/3 dels campions més utilitzats, és a dir 47.
- Filtrar per percentatge de victòria. Es considera que és massa elevat si supera el 52.5%. Pot semblar una xifra baixa, però cal tenir en compte que els millors campions assoleixen, normalment, percentatges propers al 57%.
- Escriure al fitxer *problematicChamps.txt* els campions que s'han considerat problemàtics, ordenats per percentatge de victòria.

Finalment, i sempre que s'hagin detectat casos, l'*StatisticAgent* farà una petició a l'*AprioriAgent* perquè en faci un anàlisi més extens, amb l'objectiu de detectar sinèrgies potencials o descartar-les.

3.3.3 RESOLUCIÓ DELS PROBLEMES: CAMPIONS DE MODA I BUSCANT L'EQUILIBRI

Arribats a aquest punt, el sistema està capacitat per superar els últims jocs de proves. Concretament, a l'apartat 1.5.2: problema dels campions de moda, es demana trobar quins són els campions amb un percentatge d'utilització més elevat. Després, es vol saber si aquesta elecció és o no justificada en termes de percentatge de victòria.

- L'*ExtractionAgent* crea els fitxers *winners.csv* i *losers.csv*.
- L'*StatisticAgent* els demana i procedeix a fer-ne l'anàlisi estadístic. Com a resultats, crea el fitxer *problematicChamps.txt*.

- Dels 47 campions més utilitzats, només 9 tenen un percentatge de victòria considerablement elevat.

```
WinRatio,PickRation,Champion
56.986900,17.442303,Khazix
55.717256,18.318227,Camille
54.107981,32.447254,Ezreal
53.770492,11.615508,Evelynn
53.498871,16.871049,Maokai
53.364269,16.414045,Janna
53.246753,20.527077,Xerath
52.592593,20.565161,Alistar
```

- És doncs justificada, l'elecció de l'usuari? En bona part no, ja que 4/5 dels casos el percentatge de victòria no és suficientment elevat. Encara més, en molts d'ells és fins i tot inferior al 50%. És a dir, que tot i perdre més partides de les que guanyen, segueixen sent campions populars.
- No obstant, els altres 9 casos detectats són molt rellevants i es procedeix a fer-ne un anàlisi més a fons.

El problema dels campions de moda queda, doncs, solucionat. Ara bé, a l'apartat 1.5.3: buscant l'equilibri, es busca determinar la forma d'anivellament més adequada per a cadascun dels campions problemàtics. Per a aquesta tasca, L'*StatisticAgent* demana ajuda a l'*AprioriAgent* per a què analitzi la llista del fitxers *problematicChamps.txt*.

- L'*AprioriAgent* rep i accepta la petició d'analitzar més a fons els campions problemàtics de la llista.
- Després, a partir del model de regles d'associació es crea un submodel específic per a cada un dels campions de la llista. És a dir, a la part d'esquerra de les regles de cada submodel hi consta un dels campions de la llista.

L'*AprioriAgent* disposa d'un algorisme anomenat *resolveProblematic()*, de la classe RStudio.java, que implementa precisament aquesta solució. El que fa és crear, dinàmicament, un fitxer de R per a cada campió de la llista. Després, executa aquests fitxers i en guarda el resultat en el format *campió.txt*, dins la carpeta R/problematicChamps.

Una vegada obtingudes les regles resultants, executa l'acció *checkProblematicRules()* que cerca, dins les regles, aquelles amb un pes superior al 1.5. S'utilitza aquest pes perquè es considera una xifra ja considerable a tenir en

compte. A més, com que ja és sabut que els campions són utilitzats i amb un percentatge de victòria elevat, només s'utilitzen els casos de les partides guanyades, disponibles al fitxer *winners.csv*. És a dir, es vol saber perquè han guanyat aquelles partides.

```
Khazix is too strong! No synergy detected.
Camille is too strong! No synergy detected.
Ezreal is too strong! No synergy detected.
Evelynn is too strong! No synergy detected.
Maokai is too strong! No synergy detected.
Janna is too strong! No synergy detected.
Xerath is too strong! No synergy detected.
Synergy found: Alistar=>Kalista with a lift value of: 2.3684947
Synergy found: Alistar=>Kassadin with a lift value of: 1.8288670
```

- En la majoria dels casos no es troben regles significatives. Per exemple, per a *Khazix* la regla de més pes és de només 1.32. Això indica que aquest campió es massa fort per si sol, no depèn de cap sinèrgia.

- A l'últim cas però, es troben un parell de sinèrgies significatives:
  Alistar => Kalista amb un pes de 2.3.
  Alistar => Kassadin amb un pes de 1.8.

Per tant, caldria enfocar l'anivellament del campió *Alistar* de forma col·lectiva amb la *Kalista* i el *Kassadin*. Amb això, l'últim joc de proves queda solucionat.

4. EL SISTEMA MULTIAGENT: INTERACCIÓ ENTRE AGENTS

Fins aquí s'han detallat les característiques dels agents que implementen la solució al problema i, fins i tot, la seva capacitat per a superar els jocs de proves a partir d'exemples reals. No obstant, queda per determinar com interactuen aquests agents entre si i com gestionen les diferents peticions i accions en el temps. En aquest darrera apartat, es dóna resposta a totes aquestes qüestions.

4.1 TEMPORITZACIÓ DELS AGENTS

La temporització fa referència als intervals de temps on cada agent decideix dur a terme alguna acció. Per exemple, cada quan extreure noves dades, demanar-les o processar-les.

En el cas de l'*ExtractionAgent*, el més important serà respectar les limitacions de la clau provisional de desenvolupador de l'API. Alhora, caldrà gestionar les esperes entre obtenció de identificadors de jugadors, l'extracció de partides i el processament de les mateixes. És a dir, no intentar extreure una partida quan encara no s'han obtingut els jugadors.

Per l'*AprioriAgent*, determinar un interval de temps per demanar les dades dels fitxers *winners.csv* i *losers.csv*. Després, aplicar l'algorisme *apriori* sempre que es disposin de noves dades.

Finalment, per l'*StatisticAgent*, caldrà també demanar les dades cada cert interval i configurar les peticions d'anàlisi més extens que es fan a l'*AprioriAgent*.

4.1.1 *EXTRACTIONAGENT*

Per inicialitzar l'agent, es demana com a paràmetre d'entrada la clau de desenvolupador:

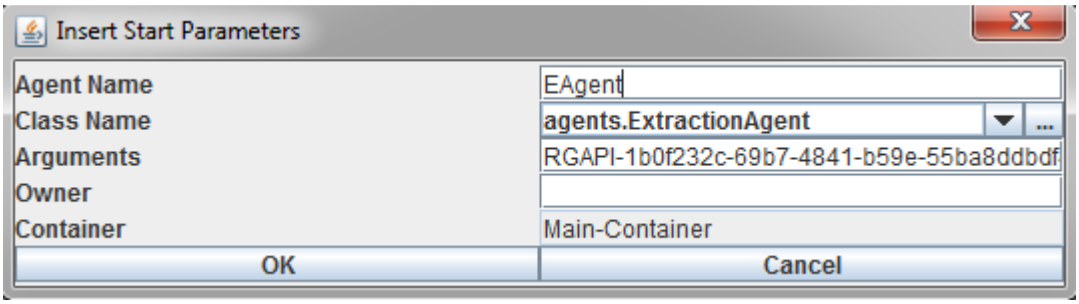


FIGURA 11. CREACIÓ DE L'AGENT *EXTRACTIONAGENT*

En la configuració inicial, s'executen les accions *writeAccountIds()* i *writeMatches()* definides a la classe LolApi.java, amb un descans de 2 minuts entre cada una. Amb això, es creen els fitxers *accIds.txt*, *matchIds.txt* i *timestamp.txt*.

A partir d'aquí, s'han definit dues funcions *onTick()*, de la classe *Agent*, que s'executen cada 30minuts i cada 2 minuts, respectivament. La primera, fa el mateix procediment que en la configuració inicial: actualitzar els fitxers d'usuaris i partides. En la segona, i sempre que hi hagi noves actualitzacions, s'actualitzen els fitxers processats *winners.csv* i *losers.csv* executant *readWinners()*.

D'aquesta manera, es crea un bucle constant on, cada 30minuts aproximadament, es preparen noves dades. L'agent deixa un rastre de missatges, per pantalla, que permet fer un seguiment del que està fent. Com a mostra, aquest és el resultat d'una execució d'una hora de durada:

```
Hello! ExtractionAgent is ready!!
ExtractionAgent: Key setted correctly
ExtractionAgent: Writing new active accounts...
ExtractionAgent: Writing new matches...
Account: 31826820 has no recent ranked matches.
Account: 30571084 has no recent ranked matches.
Account: 228259540 has no recent ranked matches.
691 new matches
ExtractionAgent: New matches available!!
ExtractionAgent: Writing csv files...
Sleeping for 2 minutes... Matches processed: 100
Sleeping for 2 minutes... Matches processed: 200
Sleeping for 2 minutes... Matches processed: 300
Sleeping for 2 minutes... Matches processed: 400
Sleeping for 2 minutes... Matches processed: 500
Sleeping for 2 minutes... Matches processed: 600
ExtractionAgent: csv files updated!!
ExtractionAgent: Waiting for new matches.
ExtractionAgent: Waiting for new matches.
ExtractionAgent: Waiting for new matches.
```

```
ExtractionAgent: Waiting for new matches.
ExtractionAgent: Writing new active accounts...
ExtractionAgent: Writing new matches...
Account: 23658212 has no recent ranked matches.
Account: 26750684 has no recent ranked matches.
Account: 23846686 has no recent ranked matches.
Account: 225191655 has no recent ranked matches.
Account: 29145731 has no recent ranked matches.
Account: 217602122 has no recent ranked matches.
284 new matches
ExtractionAgent: New matches available!!
ExtractionAgent: Writing csv files...
Sleeping for 2 minutes... Matches processed: 100
Sleeping for 2 minutes... Matches processed: 200
ExtractionAgent: csv files updated!!
ExtractionAgent: Waiting for new matches.
Warning: ExtractionAgent terminating.
```

4.1.2 *APRIORIAGENT*

Igual que en l'agent anterior, es demana un paràmetre d'entrada per a inicialitzar l'*AprioriAgent*. En aquest cas, cal indicar la ruta sencera cap a l'executable *RScript.exe*.

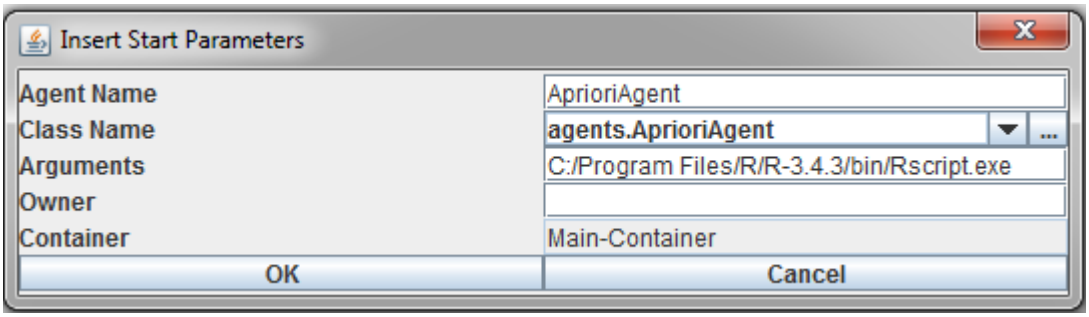


FIGURA 12. CREACIÓ DE L'AGENT *APRIORIAGENT*

D'aquesta manera, podrà executar sense problema els *scripts* de R. Al iniciar-se correctament, fa un intent per obtenir i processar les dades que l'*ExtractionAgent* tingui per oferir. Després, utilitza un total de tres accions *onTick()* per a mantenir-se constantment actualitzat:

1. Demanar dades a l'*ExtractionAgent* cada 14 minuts. És un temps suficient per a fer un parell de peticions per a cada actualitzacions de dades, assegurant així que s'acaben rebent.
2. Comprovar si té peticions per part de l'*StatisticAgent* cada 10 minuts. En cas afirmatiu, obté el fitxer *problematicChamps.txt* i executa l'algorisme *resolveProblematic()*.
3. També cada 10 minuts, comprova si disposa de noves dades i, en aquest cas, actualitza els fitxers de regles *html* generats per l'*apriori.R*.

4.1.3 *STATISTICAGENT*

Finalment, el tercer dels agents implementats no requereix cap paràmetre d'entrada. Això sí, per a una execució més eficient, és recomanable inicialitzar els agents en l'ordre en que s'acaben de descriure. Un cop l'*StatisticAgent* s'inicialitza, intenta obtenir noves dades de l'*ExtractionAgent*. Després, igual que en el cas anterior, utilitza accions *onTick()* per a mantenir-se actualitzat.

Així, cada 16 minuts (per no coincidir inicialment amb l'*AprioriAgent)*, es demanen noves dades a l'*ExtractionAgent*. Si el resultat és satisfactori, les puja automàticament a *Splunk* i en fa l'anàlisi estadístic generant el fitxer *problematicChamps.txt*.

Per a tenir una idea més clara de tot el conjunt, s'ha creat una taula que resumeix totes les temporitzacions:

Agent	**Acció**	***onTick()***
ExtractionAgent	Extreure noves dades.	Inicial i cada 30 minuts.
ExtractionAgent	Si hi ha noves dades, actualitzar els fitxers *winners.csv* i *losers.csv*.	2 minuts.
AprioriAgent	Demanar noves dades.	Inicial i cada 14 minuts.

AprioriAgent	Si se li ha demanat, resol el problema dels campions problemàtics.	Cada 10 minuts.
AprioriAgent	Si té noves dades, actualitza els fitxers *html*.	Cada 10 minuts
StatisticAgent	Demanar noves dades i actualitzar les anàlisis estadístiques.	Inicial i cada 16 minuts.

FIGURA 13. TAULA: RESUM DE LES TEMPORITZACIONS

4.2 PAS DE MISSATGES ENTRE AGENTS

Explicada la temporització de cada un dels agents, només resta comentar com funciona el pas de missatges entre agents. Bàsicament, s'utilitzen dues classes que estenen *Behaviour*[14]: *RequestPerformer()* i *OrdersServer()*. La primera s'utilitza per a enviar peticions, mentre que la segona per a rebre'n. Així doncs, s'han definit dos tipus de missatges, els *data* i els *problematic*.

- L'*ExtractionAgent* està sempre pendent dels missatges que rep. Per fer-ho, ha registrat el servei que ofereix al llibre groc. Aquest servei és del tipus *data*, i espera rebre peticions dels altres dos agents.
- L'*AprioriAgent*, d'una banda, genera i envia missatges *data* per demanar noves dades. De l'altre, registra un servei del tipus *problematic* esperant rebre missatges de l'*StatisticAgent*. Aquest últim, envia missatges d'ambdós tipus: *data* i *problematic*.

Com que els dos serveis estan registrats al llibre groc, l'únic que ha de fer un agent per obtenir l'adreça de contacte és revisar aquest llibre. Així, s'inicia el pas de missatges entre dos agents. A més, es dóna un identificador a cada conversa per a facilitar la recepció de les respostes.

Pel que fa al contingut, els missatges són o bé una cadena de caràcters o un fitxer. En aquest últim cas, el que fa l'agent que envia el fitxer és convertir-lo en una seqüència de *bytes* (serialització[15]) i enviar-los. El receptor, ha de reconvertir aquests *bytes* en el fitxer original.

A continuació, es mostra una execució d'un cicle de pas de missatges:

```
INFO: ----------------------------------------
Agent container Main-Container@192.168.56.1 is ready.
--------------------------------------------
Hello! ExtractionAgent is ready!!
Hello! AprioriAgent is ready!!
AprioriAgent: R path setted. C:/Program Files/R/R-3.4.3/bin/Rscript.exe
AprioriAgent: trying to get apriori data...
ExtractionAgent: AprioriAgent needs data for apriori algorithm.
AprioriAgent: winners.csv file received
AprioriAgent: losers.csv file received
AprioriAgent: data files updated!
Hello! StatisticAgent is ready!!
StatisticAgent: trying to get statistical data...
ExtractionAgent: StatisticAgent needs data for statistic analysis.
StatisticAgent: winners.csv file received
StatisticAgent: losers.csv file received
StatisticAgent: sleeping 5...
StatisticAgent: updating Splunk data...
StatisticAgent: sleeping 10...
StatisticAgent: writing problematic champs file...
AprioriAgent: there are no problematic champions yet.
StatisticAgent: sending problematic champs file...
StatisticAgent: statistic data files updated!
AprioriAgent: problematicChamps.txt file received
AprioriAgent: resolving problematic champs
AprioriAgent: waiting R... (15s)
AprioriAgent: checking rules

Checked rules:
Khazix is too strong! No synergy detected.
Camille is too strong! No synergy detected.
Ezreal is too strong! No synergy detected.
Evelynn is too strong! No synergy detected.
Maokai is too strong! No synergy detected.
Janna is too strong! No synergy detected.
Xerath is too strong! No synergy detected.
Synergy found: Alistar=>Kalista with a lift value of: 2.3684947
Synergy found: Alistar=>Kassadin with a lift value of: 1.8288670
Done!

AprioriAgent: done!
```

- Es comença iniciant els agents. Com que hi ha noves dades disponibles, l'*AprioriAgent* i l'*StatisticAgent* reben una resposta positiva a la petició inicial.

- L'*StatisticAgent* es disposa a pujar les dades al servidor de *Splunk*. Mentre s'espera, s'adorm 15 segons. Després, fa l'anàlisi i envia el resultats, *problematicChamps.txt*, a l'*AprioriAgent*.

- Eventualment, l'*AprioriAgent* comprova si té alguna petició i troba el missatge amb els resultats. Llavors, procedeix a crear regles per als campions problemàtics i les imprimeix.

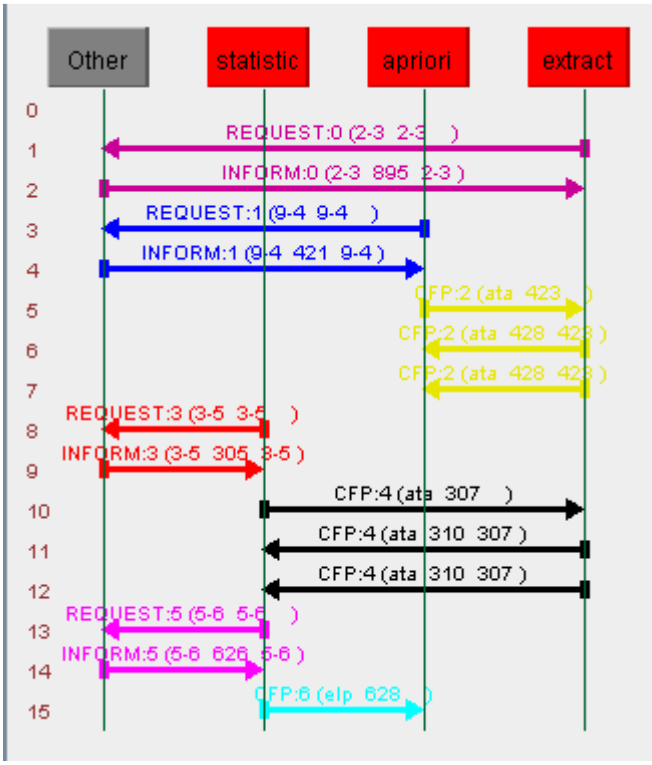


FIGURA 14. PAS DE MISSATGES ENTRE AGENTS

Enumerant cada color per ordre, el pas de missatges és:

1. L'*ExtractionAgent* registra el servei al llibre groc.
2. L'*AprioriAgent* fa una cerca de serveis al llibre groc.
3. Petició i enviament dels dos fitxers *winners.csv* i *losers.csv*.
4. L'StatisticA*gent* fa una cerca de serveis al llibre groc.
5. Petició i enviament dels dos fitxers *winners.csv* i *losers.csv*.
6. L'StatisticA*gent* fa una cerca de serveis al llibre groc.
7. L'StatisticA*gent* envia el fitxer *problematicChamps.txt* a l'*AprioriAgent*.

5. CONCLUSIONS

5.1 SEGUIMENT DE LA PLANIFICACIÓ I METODOLOGIA

En general, es pot afirmar que la planificació i la metodologia han estat adequades: s'han assolit totes les fites obligatòries amb èxit, amb ben pocs entrebancs. No obstant, algun si que n'hi ha hagut:

- En un inici, s'havia planificat el projecte per començar a implementar el SMA a les primeres fases, pensant que seria la part més complicada. No obstant, aviat es va veure que era més adequat implementar primer totes les funcionalitats i, al final de tot, fusionar-les amb els agents.
- Durant la fase final de desenvolupament, hi va haver 1 setmana de retard al subestimar el temps necessari per programar i validar els resultats. En conseqüència, els agents opcionals no s'han pogut implementar i es van haver d'utilitzar les mesures previstes per aquest tipus de risc (apartat 1.4.2).

Fora d'això, el desenvolupament del projecte ha estat, increïblement, molt fluid: allò que es tenia en ment en tot moment s'anava materialitzant satisfactòriament, algunes vegades fins i tot amb millors resultats dels esperats.

Amb tot, la metodologia oberta al canvi ha resultat ser d'allò més encertada, ja que és molt difícil definir amb exactitud una tasca a llarg termini, sobretot quan no és té un domini complet de les àrees del projecte.

Finalment, les tecnologies elegides han complert les expectatives. Els agents han resultat ideals per desacoblar i automatitzar els processos, mentre que els programaris R i *Splunk* s'han comportat perfectament.

5.2 RESULTATS OBTINGUTS

El fet de que s'hagin superat tots els jocs de proves amb èxit indica que els objectius del projecte s'han assolit. A més, el sistema és mínimament capaç de subsistir sense la interacció humana, mostrant els resultats en format web (*html*). A continuació, es fa un anàlisi d'aquests resultats. Per fer-ho, s'han extret i analitzat les dades de 5404 partides classificatòries de la última setmana.

Començant per l'anàlisi estadística, l'*StatisticAgent* ha generat el fitxers *problematicChamps.txt*, mentre que l'*AprioriAgent* n'ha extret i analitzat regles d'associació, generant el *checkedRules.txt*.

```
WinRatio,PickRation,Champion
56.650718,18.709493,Janna
55.965293,8.253661,Rakan
55.136268,8.540122,TwistedFate
54.774536,13.499481,Lulu
53.107345,12.675905,Malzahar
52.771619,16.149246,Nidalee
52.618658,21.878469,Vayne

Janna is too strong! No synergy detected.
Synergy found: Rakan=>Xayah with a lift value of: 6.752098
TwistedFate is too strong! No synergy detected.
Synergy found: Lulu=>KogMaw with a lift value of: 7.5909077
Synergy found: Lulu=>Twitch with a lift value of: 3.0305211
Malzahar is too strong! No synergy detected.
Nidalee is too strong! No synergy detected.
Vayne is too strong! No synergy detected.
```

D'aquí, se n'extreuen dues conclusions significatives, que alhora es poden interpretar des de dos punts de vista oposats:

1. Els campions *Janna, TwistedFate, Malzahar, Nidalee* i *Vayne* són, actualment, els que millors resultats obtenen individualment. Per tant, ara:
    a. El jugador corrent pot aprofitar aquesta informació per mirar de guanyar més partides.
    b. Els desenvolupadors del joc, Riot Games, saben que haurien d'anivellar, i en quina mesura, aquests campions per a què no sobresurtin de la resta.
2. Els campions *Rakan* i *Lulu* obtenen resultats molt bons, però en sinèrgia amb altres campions. Concretament, en *Rakan* amb la *Xayah*[11], mentre que la *Lulu* amb en *KogMaw*[12] i en *Twitch*. Per tant, ara:
    a. Dos jugadors d'un mateix equip poden triar una combinació guanyadora.
    b. Riot Games, a l'hora d'anivellar aquests personatges, sap que els ha d'agafar com a un conjunt i no individualment. Llavors, han de veure perquè funciona la combinació i decidir si cal prendre mesures o no.

Ara bé, no només es disposa d'informació d'aquest petit conjunt de campions guanyadors, sinó que es tenen dades de tots ells. Així, un usuari corrent pot decidir consultar les regles (si n'hi ha), del seu campió preferit. Per això, obre els fitxers *html* generats per *l'AprioriAgent*.

- Si observa el mapa descriptiu guanyador del fitxer *winInteractive.html* i selecciona, posem per cas, en *Thresh*, hi trobarà una forta relació amb la *Kalista* i una més petita amb en *KhaZix*.

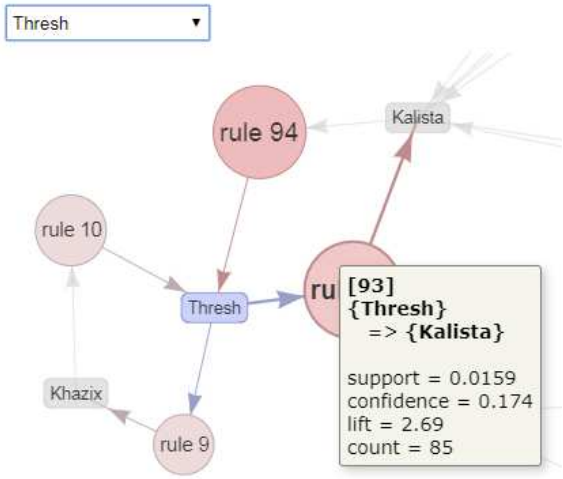


FIGURA 15. RELACIONS DEL CAMPIÓ *THRESH* (GUANYADOR)

- Si observa el mateix cas, però del *loseInteractive.html*, trobarà relacions similars.

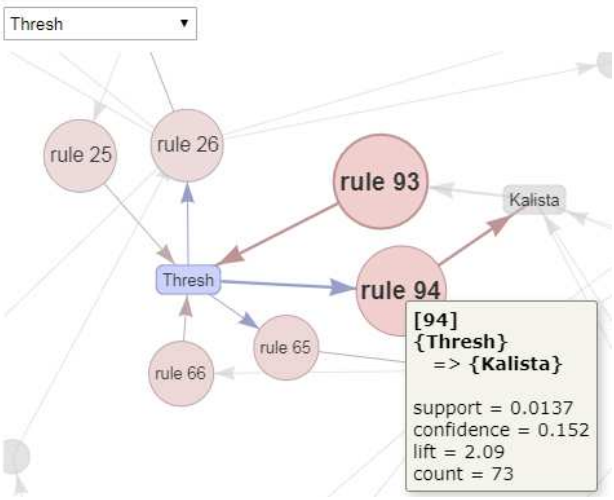


FIGURA 16. RELACIONS DEL CAMPIÓ *THRESH* (PERDEDOR)

- Simplement comparant el número de casos, *count*, es pot saber si la combinació funciona o no. En el cas {Thresh}=>{Kalista}, el percentatge de victòria és 85/85+73 = 53.79%.

La conclusió que es treu d'aquest punt és que es poden consultar totes les relacions existents entre campions, independentment del motiu per el qual es formen. A més, és possible comprovar si són fructíferes (percentatge de victòria positiu) o no. També es pot donar el cas que una regla sigui única per un dels dos fitxers; llavors, no cal fer cap comparació.

Per acabar, també és possible cercar relacions entre campions sense cap mena de prioritat, a partir dels fitxers *winRules.html* i *loseRules.html*. Els gràfics que contenen són fàcilment llegibles: com més s'acosti un punt a la cantonada superior dreta, més rellevant és. A primer cop d'ull, es destaquen:

Campió(1)	**Campió(2)**	**% victòria**
Xayah	Rakan	**56.63%**
Alistar	Tristana	52.07%
LeeSin	Ezreal	53.84%
Lulu	Twitch	54.22%
Lulu	KogMaw	**64.96%**
Thresh	Kalista	53.79%
Vayne	Janna	**65.96%**

FIGURA 17. TAULA: PARELLES SINÈRGIQUES DESTACADES

El més interessant és que, per separat, tenen percentatges de victòria molt inferiors. Per exemple, en Thresh té un 50% i la Kalista un 47.6%, però junts arriben al 53.79%.

En conclusió, és una manera ràpida i efectiva de trobar possibles parelles sinèrgiques rellevants.

5.3 CONCLUSIONS FINALS

Les lliçons apreses amb aquest treball són molt diverses. Per començar, ha permès dissenyar i desenvolupar un projecte, una idea, utilitzant i sintetitzant molts dels coneixements obtinguts durant el grau. Amb àrea principal la intel·ligència artificial, s'han tocat temes com l'estadística, la mineria de dades, el *Big Data*, l'aprenentatge computacional, el disseny i programació orientada a objectes i la gestió de projectes. A més, s'han desenvolupat altres habilitats més personals: la imaginació, per preveure situacions a llarg termini, planificar un projecte clar però subjecte al canvi i fer encaixar totes les peces, que no eren poques; la proactivitat, per a proposar i dissenyar un projecte complet a partir d'una idea més aviat personal, com són els videojocs MOBA; i la constància, per a seguir la planificació i tirar endavant el projecte.

Seguint la mateixa línia, també s'han adquirit coneixements en àrees noves com són la programació d'agents mitjançant JADE i l'ús d'APIs i de la plataforma *Splunk*. El sistema multiagent resultant ha demostrat estar capacitat per extreure i analitzar dades provinents d'una API externa, presentant els resultats sense necessitat d'intervenció humana.

Pel que fa als resultats obtinguts, és molt bona senyal haver detectat la sinèrgia més coneguda i oficialment acceptada del LoL, formada per la parella Xayah i Rakan[11]. Aquesta ha permès validar els resultats, que s'han mantingut estables al llarg del temps i amb diversos conjunts de dades diferents. Per tant, és possible afirmar que, en aquest sentit, s'han obtingut els resultats esperats.

En conclusió, el projecte inicial ha estat un èxit però, a la vegada, té encara un llarg camí per recórrer. Al següent apartat es detallen les línies de futur més importants que s'han anat detectat al llarg del projecte.

5.3.1 LÍNIES FUTURES

Són moltes les idees que han anat sorgint durant la realització d'aquest projecte i ha acabat escapant de l'abast del mateix. Entre elles, es destaquen:

- Millorar l'algorisme d'extracció de dades, amb l'objectiu d'obtenir-ne més a cada cicle d'execució. Actualment es troben moltes partides repetides, que són descartades i suposen un cost temporal molt elevat.
- Establir els paràmetres d'entrada, com ara el suport mínim i el percentatge de victòria mínim de forma dinàmica, a partir d'una funció adequada.
- Obtenir una clau permanent, presentat el projecte a Riot Games, quan aquest estigui a punt per a fer-se públic.
- Pujar els resultats en un servidor web, des d'on els usuaris puguin consultar-los. És a dir, dissenyar una interfície web per al SMA.
- Utilitzar una base de dades, com ara MySQL, o una ontologia.
- Implementar noves funcionalitats al sistema, com ara:
    - Nous algorismes de mineria. Per exemple, el C4.5[16] per a crear models predictius (arbre de decisió) que intentin predir certs resultats, com ara qui guanyarà la partida. Una altra idea, seria utilitzar un algorisme aglomerador per a generar conjunts similars de jugadors, a partir de dades com el rol que solen jugar, la lliga en que estan, els resultats de les partides, etc.
    - La capacitat de calcular automàticament el percentatge de victòria d'una parella concreta, passada com a paràmetre d'entrada, per estalviar a l'usuari el càlcul de *victòries/(victòries+derrotes)*.
    - Filtrar les regles generades segons el rol que juga cada personatge. Per exemple, un personatge de suport és més fàcil que tingui regles sinèrgiques, ja que la seva feina és cooperar estretament amb algun company.
- Treballar amb altres dades a part dels campions, ja que una partida té moltes més variables interessants, com ara els resultats finals.
- Adaptar el sistema per a què pugui treballar amb altres videojocs MOBA. El procés d'adaptació seria:
    - Analitzar a fons quines dades estan disponibles i en quin format.
    - Dissenyar un agent capaç d'obtenir i tractar les dades fins a tenir-les amb el format desitjat.

6. GLOSSARI

agent intel·ligent entitat intel·ligent capaç d'interactuar amb el seu entorn i prendre decisions de forma racional.

API *Application Programming Interface*, especifica la interacció entre diferents components informàtics.

apriori algorisme utilitzat per a crear regles d'associació.

arbre de decisió model predictiu utilitzat per a representar i categoritzar condicions que es donen de forma successiva.

campió personatge del joc *League of Legends*, estudiat en aquest projecte.

clau provisional de desenvolupador codi d'accés a la API de Riot Games, que en permet la utilització durant 1 dia.

confiança percentatge de casos en què una regla d'associació es compleix.

gràfic descriptiu representació, en forma de nodes i vèrtex, de les regles d'associació.

gràfic de dispersió representació, a partir de dues o més variables, dels diferents elements d'un conjunt.

intel·ligència artificial àrea de la informàtica dedicada al desenvolupament d'algorismes de presa de decisions intel·ligents, o que intenten simular la intel·ligència humana.

JADE *Java Agent Development Framework* , plataforma per al desenvolupament d'agents, implementada en *Java*.

joc de proves s'utilitza per verificar el bon funcionament d'un sistema.

línia de futur possibles millores, canvis o idees a implementar dins un projecte a llarg termini.

llibre groc element abstracte on un agent intel·ligent registra els seus serveis, per tal que altres agents els puguin sol·licitar.

LoL *League of Legends*, videojoc desenvolupat per Riot Games, element d'anàlisis d'aquest projecte.

mineria de dades procés d'identificació de nous coneixements, patrons o informació vàlida ocults en un conjunt de dades.

MOBA Multiplayer Online Battle Arena, subgènere basat en batalles en línia a temps real.

navegador programa informàtic que permet reproduir documents d'hipertext.

nexe edifici que cal destruir per guanyar una partida del videojoc *League of Legends*.

ontologia descriu i categoritza conceptes dins una àrea de coneixement específica, donant-los significat i establint relacions entre ells.

pes rellevància d'una regla d'associació, obtinguda a partir de la confiança i la probabilitat de la part dreta de la regla.

puntuació de visió estadística del final d'una partida del *League of Legends*. Indica el nivell de cobertura del terreny durant la partida.

R entorn i llenguatge de programació. En aquest projecte, s'utilitza per executar els algorismes de mineria de dades.

regla d'associació norma generada per un algorisme descriptiu, com ara l'*apriori*. Descriu patrons d'una realitat.

Riot Games empresa que desenvolupa i publica videojocs, creada el 2006.

script llenguatge de programació que interacciona amb les aplicacions, generalment fitxers de text amb un marcatge específic.

serialització transformació d'un objecte informàtic a una cadena de bytes equivalent. També existeix el procés contrari, que permet recuperar l'objecte original.

sinèrgia es dóna quan la integració de dues parts és superior a la simple suma d'ambdós elements.

SMA Sistema MultiAgent, format per un conjunt d'agents intel·ligents.

Splunk empresa propietària de la plataforma d'anàlisi de *Big Data* amb el mateix nom. En aquest projecte, s'utilitza per a fer l'anàlisi estadística de les dades.

suport percentatge total de casos en què els dos elements d'una regla d'associació són presents. *també* pot ser un rol d'un campió del *League of Legends* o d'altres jocs.

7. BIBLIOGRAFIA

[1] Viquipèdia, l'Enciclopèdia Lliure (2017). **MOBA** [en línia]. [Consultat: 4 d'octubre de 2017]. Disponible a Internet a: https://ca.wikipedia.org/wiki/MOBA

[2] Viquipèdia, l'Enciclopèdia Lliure (2017). **Sistema Multiagente** [en línia]. [Consultat: 28 de setembre 2017]. Disponible a Internet a:

https://es.wikipedia.org/wiki/Sistema_multiagente

[3] Riot Games (2017). **League of Legends ES** [en línia]. [Consultat: 28 de setembre 2017]. Disponible a Internet a: http://euw.leagueoflegends.com/es/

[4] Riot Games (2017). **Riot Games API** [en línia]. [Consultat: 28 de setembre 2017]. Disponible a Internet a: https://developer.riotgames.com

[5] Universitat Rovira i Virgili (URV, 2017). **Guia de programació de sistemes multiagent en JADE 3.3** [en línia]. [Consultat: 28 de setembre 2017]. Disponible a Internet a:

http://deim.urv.cat/recerca/reports/DEIM-RT-05-001.html

[6] Viquipèdia, l'Enciclopèdia Lliure (2017). **Mineria de dades** [en línia]. [Consultat: 28 de setembre 2017]. Disponible a Internet a: https://ca.wikipedia.org/wiki/Mineria_de_dades

[7] Splunk (2017). **Splunk: Turn Machine Data Into Answers** [en línia]. [Consultat: 20 de novembre 2017]. Disponible a Internet a: https://www.splunk.com/

[8] R Project (2017). **The R Project for Statistical Computing** [en línia]. [Consultat: 28 de setembre 2017]. Disponible a Internet a: https://www.r-project.org/

[9] Viquipèdia, l'Enciclopèdia Lliure (2017). **Apriori Algorithm** [en línia]. [Consultat: 14 d'octubre 2017]. Disponible a Internet a:

https://en.wikipedia.org/wiki/Apriori_algorithm

[10] Arules package (2017). **Mining association rules and frequent itemsets** [document, en línia]. [Consultat: 18 de desembre 2017]. Disponible a Internet a:

https://cran.r-project.org/web/packages/arules/arules.pdf

[11] Riot Games (2017). **Xayah and Rakan** [en línia]. [Consultat: 31 de desembre 2017]. Disponible a Internet a:

https://euw.leagueoflegends.com/es/news/champions-skins/champion-release/xayah-and-rakan-available-now-launch-promotions

[12] Riot Games (2017). **Kog'Maw and Lulu, a Hjarnan story** [en línia]. [Consultat: 31 de desembre 2017]. Disponible a Internet a:

http://www.lolesports.com/en_US/articles/kogmaw-and-lulu-hjarnan-story

[13] Splunk (2017). **Basic searches and search results** [en línia]. [Consultat: 31 de desembre 2017]. Disponible a Internet a:

http://docs.splunk.com/Documentation/Splunk/7.0.1/SearchTutorial/Startsearching

[14] JADE (2017). **Using Jade Behaviours** [en línia]. [Consultat: 15 de desembre 2017]. Disponible a Internet a:

https://www.iro.umontreal.ca/~vaucher/Agents/Jade/primer6.html

[15] Javatpoint (2017). **Serialization in Java** [en línia]. [Consultat: 21 de desembre 2017]. Disponible a Internet a:

https://www.javatpoint.com/serialization-in-java

[16] Viquipèdia, l'Enciclopèdia Lliure (2017). **C4.5 algorithm** [en línia]. [Consultat: 30 de desembre 2017]. Disponible a Internet a:

https://en.wikipedia.org/wiki/C4.5_algorithm