UNIVERSIDAD DEL BÍO-BÍO

Diseño e Implementación de un Lenguaje de Programación de Tipo Declarativo Inspirado en Prolog para Control de Agentes en Videojuegos

Christopher Cromer y Martín Araneda Acuña

8 de enero de 2023

Índice

Índice de figuras

Índice de cuadros

Agradecimientos

From Chris,

I would like to thank my wife Alejandra. Her constant help and support is what has allowed me to go back to college and to achieve my goals and become the professional I am today. I owe her everything and am very lucky to have had someone on my side through this whole process during all these years of working, studying, and investigating.

I would also like to thank our thesis supervisor, Clemente Rubio-Manzano who inspired me to dig deeper into video game design and artificial intelligence. His support during this thesis was indispensable and helped us to bring this project to life.

De Martin,

Agradezco a mi Padre, por su constante apoyo, esfuerzo y sabiduría. A mi madre, por su cariño, paciencia y siempre la disposición a escucharme.

Por otro lado, quiero agradecer a nuestro profesor guía Clemente Rubio-Manzano, por darnos todo el apoyo necesario y tener siempre la disposición de ayudarnos para poder hacer de esta idea realidad.

Composición del Informe

El presente informe se encuentra dividido en cuatro capítulos. A continuación se describe brevemente el contenido de cada uno de ellos.

1. **Introducción:** Se presentan los conceptos básicos más importantes para este proyecto, como son el Desarrollo de Videojuegos y la Inteligencia Artificial. También se explican los motivos principales y específicos para el desarrollo de la investigación. Por último, se detalla la metodología de trabajo a aplicar.

2. **Marco Teórico:** Explicación de las herramientas a utilizar, análisis de rendimiento de software similares y descripción de conceptos abarcados en el proyecto.

3. **Estado del Arte:** Se describen las principales ideas de trabajos de investigación similares, contrastándolo con este trabajo. Se han dividido estos en dos categorías: basado en el lenguaje de programación Prolog y GOLOG, con descripciones acerca de metodología e implementación de agentes en videojuegos

4. **Descripción del Ambiente del Software:** Se explica el software a utilizar, incluyendo la versión y el entorno de desarrollo.

Además, al final del informe se adjuntan las referencias con los artículos utilizados en el proceso de investigación.

1. Introducción

La Inteligencia Artificial (I.A) es una disciplina científica relativamente nueva que nació de los avances en las ciencias de la computación y/o la informática. De forma intuitiva, la I.A puede entenderse como la capacidad que tiene una máquina para tomar decisiones de forma autónoma a partir de los estímulos que recibe del exterior. Una de las primeras veces que se hablo de la I.A como disciplina fue en el año 1956 cuando John McCarthy, importante matemático e informático, la definió como *"La ciencia e ingeniería de hacer máquinas inteligentes, con mayor énfasis en programas computacionales inteligentes"* [1].

Actualmente, la I.A puede considerarse también una tecnología moderna que ha ido evolucionando y se ha ido incorporando a nuestra vida cotidiana. Un ejemplo bastante importante es el reconocimiento facial introducido en teléfonos inteligentes, lo que permite agregar ciertos efectos a las imágenes, ajustar automáticamente el enfoque o balancear la exposición en condiciones de poca luz.

En el campo del desarrollo de videojuegos, la I.A. ha tenido mucha presencia. Por ejemplo, se ha empleado con éxito para programar los personajes o las entidades de los mismos, haciéndolos más dinámicos y divertidos. Permitiendo que el juego aprenda del usuario y creando adversarios artificiales del nivel de un humano.

Dado lo anterior, tendrá un gran beneficio implementar una I.A. empleando un lenguaje declarativo ya que no indicamos el cómo se debe computar, solo indicamos el qué se quiere computar. Esto último es importante ya que conseguimos que la tarea de programación se aproxime a la lógica humana y el programador pueda implementar el comportamiento de los agentes de una forma más intuitiva y empleando reglas simples. En este sentido, uno de los lenguajes declarativos más importantes ha sido Prolog demostrando sus buenas propiedades en este ámbito. Por lo tanto, vamos a crear un lenguaje declarativo inspirado en Prolog que se pueda utilizar para programar el comportamiento de los personajes de un videojuego que tienen, al menos, que realizar las siguientes acciones: i) superar obstáculos, con la opción de saltar con velocidad; ii) recoger items que se agregan a la puntuación final; iii) evitar enemigos.

Los objetivos del proyecto de investigación son los siguientes:

1.1. Objetivos

1.1.1. Objetivo General

Hemos desarrollado un lenguaje de programación de tipo declarativo inspirado en Prolog que permite modelar el comportamiento de los agentes de un videojuego empleando reglas lógicas.

1.1.2. Objetivos Específicos

1. Se revisó la bibliografía sobre Prolog, el motor Godot y programación de videojuegos.
2. Se analizó la información recopilada de la bibliografía investigada.
3. Se creó el lenguaje de programación de tipo declarativo inspirado en Prolog, de nombre *"Obelisk"*.
4. Se implementó una inteligencia artificial basada en el lenguaje anteriormente creado en un videojuego.
5. Se evalúo del desempeño del lenguaje inventado, con verificación del cumplimiento exitoso de la superación de obstáculos por parte del agente.

1.2. Metodología de Trabajo

La metodología utilizada es la de investigación experimental, la cual consiste en ir implementando ciertas acciones en la que el agente debe reaccionar de acuerdo a los estímulos que están presentes en un ambiente preestablecido, que en este caso es un nivel presente en el videojuego. Con esto, se harán pruebas de causa y efecto, recompilando información en su efectivo comportamiento racional frente a obstáculos para posterior comparación con un jugador real.

Por tanto, el plan de trabajo consistirá en las siguientes fases:

- **Fase 1:** Revisión y descarte de bibliografía relacionada al desarrollo de videojuegos con implementación de inteligencia artificial inspirado en Prolog y motor Godot.

- **Fase 2:** Estudio de la información recopilada para posible implementación en el software.
- **Fase 3:** Creación del lenguaje de programación lógico de tipo declarativo inspirado en Prolog.
- **Fase 4:** Implementación de un videojuego en el motor Godot de estilo plataforma formado por agentes inteligentes programados usando el lenguaje recientemente creado.
- **Fase 5:** Evaluación del correcto desempeño del agente en la superación de obstáculos, con el cumplimiento del nivel versus un jugador real y la comparación de la puntuación y tiempo total en completar el juego entre ambos jugadores.

2. Marco Teórico

Este capítulo tiene como propósito describir los conceptos más importantes en el desarrollo del lenguaje de programación para control de la Inteligencia Artificial, así como también su implementación en un juego estilo plataformas.

2.1. Desarrollo de Videojuegos

2.1.1. Motor de Videojuegos

El motor de videojuegos es, en pocas palabras, un conjunto de rutinas de programación que tiene como objetivo proveer facilidad en la utilización de elementos gráficos, sonidos, físicas, redes y varios otros aspectos que ayudan en el desarrollo de un videojuego. El motor que se utilizará en este proyecto tiene por nombre ”Godot”. [2, 3, 4]

Figura 1: Godot Engine

El motor Godot tiene un gran valor en el desarrollo del proyecto, principalmente por su capacidad para implementar arte de píxel, en comparación con motores similares como Unity o Unreal, porque en Godot las distancias entre los elementos es medido en píxeles mientras que otros motores utilizan metros, lo que facilita en gran medida la eficacia al trabajar en juegos 2D, que usualmente para este género de videojuegos se usan píxeles y no metros.

Otro factor importante es que Godot posee un elemento llamado ”GDNative”. GDNative es una API que permite usar lenguajes de programación como C, C++ o Rust, siendo estos capaces de compilar código a lenguaje máquina. Lo anterior posee especial importancia para el proyecto, puesto que permite implementar otro lenguaje de programación que hará interacción con GDNative. [5]

También la API permite que la Inteligencia Artificial tome decisiones con mayor rapidez dado que el código objeto de máquina se ejecuta con mayor velocidad que un lenguaje interpretado o que se compila a bytecode, como se puede notar en los siguientes gráficos que comparan GDScript(lenguaje interpretado similar a Python), C# y C++.

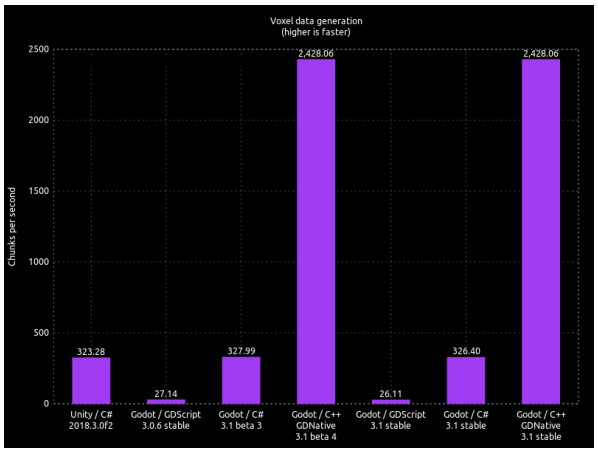

Figura 2: Voxel Data Generation
[6]

En el caso de generación de datos voxel, existe un aumento de 8,846.43 % en su rendimiento comparando GDNative con GDScript y un aumento de 651.07 % comparando GDNative con C#.

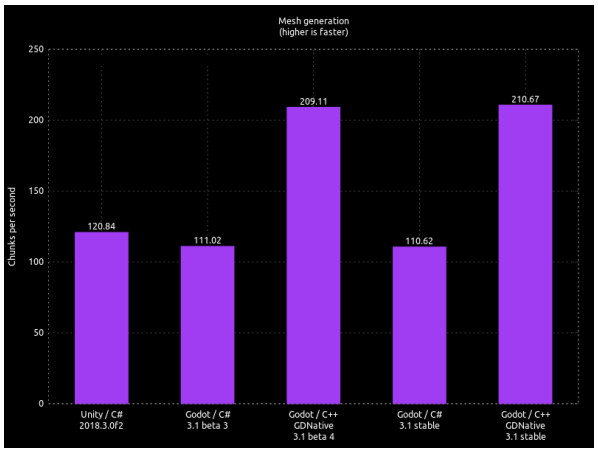

Figura 3: Mesh Generation
[6]

En la prueba de generación de mallas, GDNative tiene un aumento de 651.07 % en su rendimiento comparado con C#.

2.1.2. Game I.A.

El lenguaje responsable que se encargará de la toma de decisiones por parte de la Inteligencia Artificial será de carácter propio e inspirado en Prolog, un lenguaje declarativo y lógico. [7]

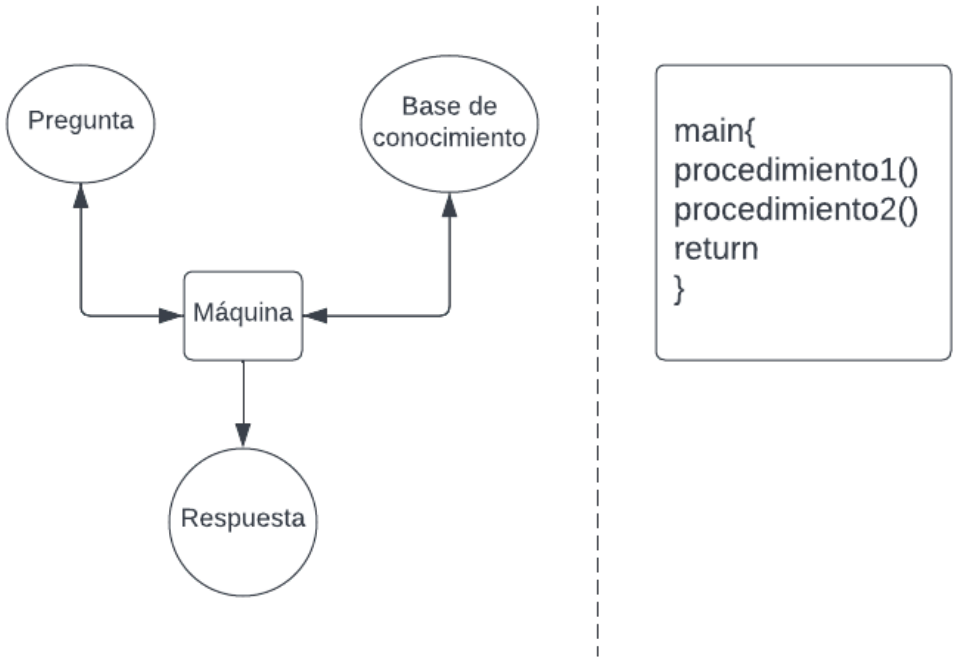

Figura 4: Programación Lógica vs. Funcional

Al hablar de programación funcional, la idea principal es que todos los elementos sean funciones y estos sean capaces de poder ejecutarse de manera secuencial, por lo cual se utiliza la lógica paso por paso para resolver el problema.

Por otro lado, la programación lógica utiliza una base de conocimiento para hacer preguntas y recibir respuestas que se utilizarán para resolver el problema. [8]

El lenguaje tipo Prolog debe tener 3 elementos importantes para funcionar:

1. **Hechos:** Son datos verdaderos, como por ejemplo ”el español es una idioma”.
2. **Reglas:** Son cláusulas condicionales que conectan los hechos. Un ejemplo es: ”si vives en Chile hablas el español”.

3. **Preguntas:** Son necesarias para tener una respuesta por parte de la base de conocimiento. Un ejemplo sería ”¿El español es una idioma?”.

2.1.2.1. Lenguaje de Programación Compilado

El proyecto LLVM es un conjunto de tecnologías de compilador y toolchain, el cual permite crear un lenguaje propio de programación. [9]

Figura 5: LLVM

LLVM consiste de varios sub-proyectos, pero el que será utilizado principalmente es ”LLVM Core”. Este sub-proyecto contiene un optimizador y generador de código, siendo este último llamado LLVM Intermediate Representation(LLVM IR). La funcionalidad es similar a una Virtual Machine de bytecode que es portátil y se puede correr en cualquier sistema que posee el LLVM.

Otro aspecto importante de LLVM es que se puede utilizar el LLVM IR, que fue generado anteriormente y así compilarlo a lenguaje máquina para la arquitectura computacional que se desee. Luego, el código objeto generado se puede utilizar con un linker para crear librerías y binarios, lo que tendrá importancia al querer integrar el código que compila nuestra compilador en el motor Godot.

2.1.2.2. Máquinas de Estados

Una máquina de estados se utiliza para controlar el estado de un objeto que tiene una ramificación compleja y estados mutables. [10] La máquina de estados finita es parte de una rama de la ciencia de la computación llamado *”teoría de autómata”*, la cual incluye la famosa máquina de Turing. La máquina de estados es usualmente usada en programación de

I.A., videojuegos y en la creación de compiladores de código, sin embargo fuera de estas 3 áreas, posee muy poca adopción y uso.

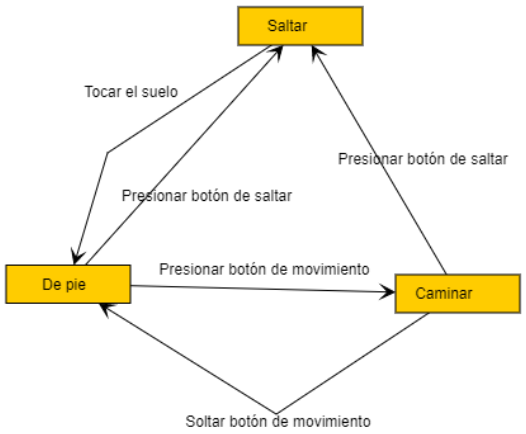

Figura 6: Máquina de Estado Finito

La máquina de estado finita posee un conjunto concreto de estados en la que es capaz de estar, como ”saltar” o ”caminar”, la cual también posee una restricción importante que consiste en que esta máquina solo puede presentar un solo estado en un instante de tiempo, es decir, el jugador no es capaz de saltar y caminar en un mismo momento.

El funcionamiento de la máquina de estados finita consiste en una secuencia de entradas y eventos que son enviadas a esta, los cuales se usan para cambiar entre estados. Cada máquina tiene un conjunto de transiciones y cada una de ellas está asociada con una entrada o un evento, lo que finalmente apunta a otro estado. Por tanto, cuando llega la entrada o evento y este coincide con una transición dentro del estado actual, la máquina cambiará al estado al que apunta la transición, por lo que si un jugador se encuentra ubicado en el estado ”caminar”, se puede presionar el botón de saltar para cambiar al estado de ”saltar”.

3. Estado del Arte

La programación del comportamiento de agentes en videojuegos se ha realizado desde diferentes puntos de vista. Los trabajos de investigación que hemos estudiado se pueden agrupar en dos categorías: (i) basados en el lenguaje de programación lógico Prolog; (ii) basados en el lenguaje de programación lógico GOLOG. A continuación se describirán cada una de las propuestas de forma más detallada.

3.1. Basado en el Lenguaje de Programación Lógico Prolog

El articulo *Prolog-Scripted Tactics Negotiation and Coordinated Team Actions for Counter-Strike Game Bots* [11], explica la creación e implementación de un script basado en el lenguaje Prolog dentro de un juego estilo FPS (Disparos en Primera Persona), con la finalidad de crear diferentes tácticas de comportamiento en un equipo de bots compuesto por agentes. La idea fue inspirada principalmente en la observación de tácticas de equipo presentes en torneos reales de Counter-Strike.

Por otro lado, la construcción de este framework está basado en un proyecto anterior hecho por los autores para crear scripts para bots. El pilar de esta investigación se construye en la premisa de diseñar el framework para utilizar un creador de mapas y así personalizar el comportamiento del bot para mapas nuevos, pues con esto se le entregaría al agente conocimiento necesario del ambiente que le rodea para adquirir una ventaja táctica frente a oponentes humanos. Nos centramos en estudiar dos aspecto: el desarrollo del razonamiento lógica del agente y las características de su I.A.

3.1.1. Desarrollo del Razonamiento Lógico del Agente

Cada bot contiene dos pilas, una de acciones y otra de tareas. Estas siempre ejecutan el bloque que esta en el tope de la pila. Para que un bot sea capaz de cumplir una tarea de manera exitosa, hará ciertas acciones asociadas a esa tarea con tal de cumplirla. Algunas condiciones causarán que se agreguen o quiten acciones dentro de esta pila. Por tanto, cuando se complete una tarea o una acción, se borrará el bloque y luego el bot intentará ejecutar

la acción o tarea siguiente en la pila.

Uno de los mecanismos de razonamiento utilizado es uno llamado ”Razonamiento de reflejos”, el cual sucede en cada actualización del estado actual del juego, prestando gran apoyo cuando ocurren cambios repentinos en un nivel, como en el ambiente o cuando el bot este siendo atacado. En consecuencia, este mecanismo posee una característica poderosa para agregar nuevas tareas o limpiar la pila de tareas, adaptándose a cada situación.

3.1.2. Características de la I.A.

El agente usado en este proyecto es de tipo racional basado en objetivos, con base en un árbol de decisión con instrucciones condicionales. En los trabajos previos hechos por el autor, se implementó una base de datos que contiene puntos de navegación, con el fin de calcular y planificar el camino que debe seguir al moverse el bot. También se usan los puntos de navegación para tomar decisiones racionales de naturaleza táctica. El bot adquiere total conocimiento del mapa, lo que le permite tener una ventaja táctica y así aumentar sus posibilidades para ganar la partida.

3.2. Basado en el Lenguaje de Programación GOLOG

En el articulo *Controlling Unreal Tournament 2004 Bots with the logic-based action language Golog* [12], se describe y explica un lenguaje de programación lógico llamado GOLOG que permite implementar agentes de videojuego del estilo FPS (Disparos en Primera Persona) y enfrentarlos a NPCs (Non-Player Characters). El videojuego en cuestión, llamado ”Unreal Tournament 2004”, es de tipo multijugador, donde los jugadores humanos compiten para alcanzar objetivos. Los oponentes pueden ser tanto humanos o controlados por el computador. Nos centramos en estudiar dos aspectos de la propuesta: el control de los oponentes y la características de la I.A.

3.2.1. Control de Oponentes

Originalmente, la toma de decisiones de los bots fue hecha usando un lenguaje orientado a objetos llamado ”UScript”, que es propio del juego y es de tipo script, con lógica ba-

sada en un *"state machine"*, constando de nueve estados controlados por el motor del juego. Sin embargo, el manejo de los bots se realizó en *"ReadyLog"*, un lenguaje basado en GOLOG, que permite el razonamiento basado en acciones.

Con esta información se agregó una interfaz al motor del juego que le permite transmitir información importante relacionada al mapa, tal como items (munición, vida y armas), ubicación de los jugadores y estilo del ambiente en general. Por tanto, la información de estos elementos sera transmitida al bot si este es capaz de percibirlos, lo que da posibilidad a cambiar el comportamiento del bot dentro del framework.

3.2.2. Características de la I.A.

El tipo de I.A. esta basada en objetivos y emplea técnicas de planificación. Con respecto a la planificación de camino y la detección de colisiones debemos mencionar que la interfaz entre el juego, junto con el motor y los bots no permite modificar los algoritmos de la planificación de los caminos ni de la detección de colisiones para permitir el cumplimiento de los objetivos. El agente solo recibe información si está en su campo de visión, por tanto, no es omnisciente y no tiene información suficiente que le permita obtener ventaja contra otros agentes o jugadores.

3.3. Comparación de Trabajos

En el Cuadro 1 se realiza un resumen general de las principales características de proyectos de investigación similares al nuestro, contrastando las diferencias de cada uno de ellos. Cabe destacar que una de las mayores diferencias presentes en relación a los otros trabajos es el tipo de lenguaje y si es un juego propio del autor. Lo primero es porque el uso de lenguaje compilado trae consigo una enorme ventaja en aspectos como por ejemplo la velocidad de toma de decisiones complejas en distintos ambientes en que el agente pueda encontrarse.

	Obelisk	**Prolog**	**ReadyLog**
Basado en	Prolog	Prolog	Golog
Tipo de Lenguaje	Compilado	Scripted	Scripted
Juego Propio	Sí	No	No
Tipo de Juego	Platformer	FPS	FPS
Tipo de I.A.	Racional	Racional	Racional
Uso de *BOT*	No	Sí	Sí
Planificación de Camino	Posible	Sí	Imposible
Detección de Colisión	Posible	Desconocido	Imposible
Agente Omnisciente	No	Sí	No

Cuadro 1: Comparación de Trabajos

El desarrollo de un juego propio permite una mejor integración entre la inteligencia artificial y el juego en cuestión, porque nos da mejor control en todos los aspectos que permiten el control del agente.

4. Diseño del Lenguaje

Nuestro lenguaje de programación, de nombre *"Obelisk"*, tiene como pilar fundamental en su diseño el paradigma declarativo, es decir, posee un fuerte enfoque en definir el resultado al que se desea llegar, en contraste a describir el flujo de control para obtener la solución. En consecuencia, el idioma posee por naturaleza un nivel de abstracción alto.

Por otro lado, las palabras claves utilizadas están inspiradas en el lenguaje Prolog, el cual posee hechos y reglas. Esto es para luego agregar una palabra propia de las acciones a tomar, dependiendo de los hechos y las reglas.

Debido a esto, se abren posibilidades para ser perfeccionado con facilidad ya que al escribir código no es necesario determinar el procedimiento al cual se desea llegar, dándole la poderosa cualidad de ser bastante flexible.

"Obelisk" contiene palabras claves que cuando sean utilizadas permitan que ciertas operaciones lógicas sean declaradas y ejecutadas, y por lo tanto reconocidas para su posterior uso, las cuales serán descritas a continuación.

4.1. Sintaxis

La sintaxis del lenguaje fue diseñado teniendo en mente el asemejarse al lenguaje humano, específicamente el idioma inglés. La razón es debido a que el inglés es utilizado en todas partes del mundo y es muy fácil de aprender y usar.

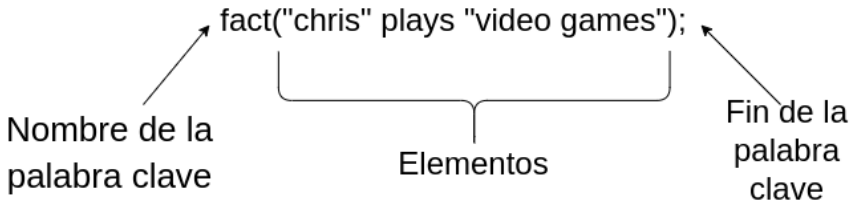

Figura 7: Estructura Básica de una Palabra Clave Lógica

La estructura de la sintaxis del lenguaje de programación *"Obelisk"* se desglosa en tres partes. El nombre, los elementos y el punto y coma (;) componen en su totalidad lo que nuestro idioma comprende como una palabra clave del lenguaje. Cabe señalar que las palabras claves del lenguaje están restringidas solo en el idioma inglés.

En este caso, el idioma posee tres palabras claves implementadas, las cuales son las acciones, los hechos y las reglas. Cada una de ellas tiene un orden ligeramente similar y difieren sólo en el contenido de los elementos, el cual se describirá a continuación.

4.1.1. Hechos

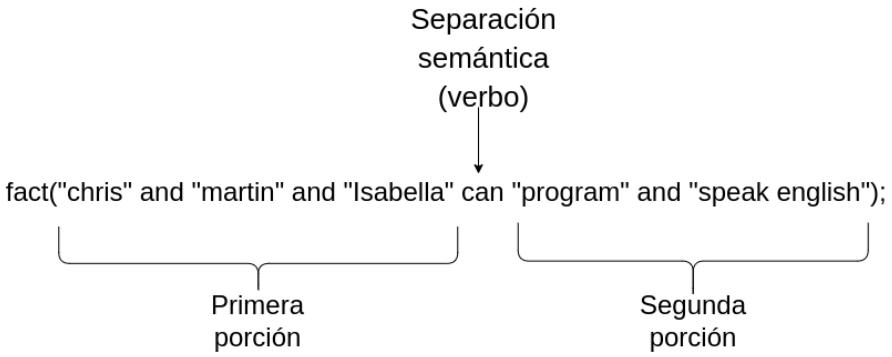

Figura 8: Estructura Básica de un Hecho

Los hechos tienen su contenido dividido en dos porciones. Sin embargo, la separación semántica es hecha por un verbo.

Por lo tanto, tenemos:

- 1°ra Entidad(es) (primera porción): Deben estar en comillas dobles, pueden ser más de uno y deben estar separados por la conjunción *and*. Ej. *"chris and martin...etc"*.
- Verbo: Representa la división semántica y la relación entre las dos porciones. Ej. *"can"*
- 2°da Entidad(es) (segunda porción): Deben estar en comillas dobles, pueden ser más de uno y deben estar separados por la conjunción *and*. Ej. *"program and speak english...etc"*

4.1.2. Acciones

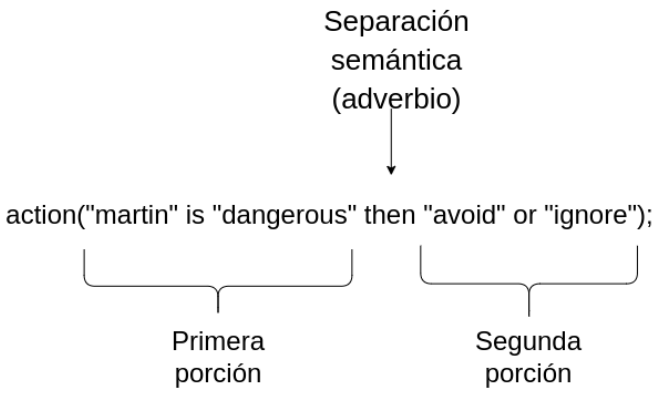

Figura 9: Estructura Básica de una Acción

La sintaxis de las acciones destaca principalmente, como las demás palabras clave, en los parámetros. El contenido esta compuesto por dos porciones, divididos por el adverbio. Usando como referencia el ejemplo anterior, la estructura es:

- 1°ra Entidad (primera porción): Debe estar en comillas dobles. Ej. *"martin"*.
- Verbo: Representa la relación entre las dos entidades de la primera porción. Ej. *"is"*.
- 2°da Entidad (primera porción): Debe estar en comillas dobles. Ej. *"dangerous"*.
- Separación semántica: El adverbio describe la implicación que tiene la primera porción sobre la segunda. . Ej. *"then"*.
- Sugerencia verdadera(segunda porción): Presenta la opción a sugerir en caso en que el hecho sea verdad. Ej. *"avoid"*.
- Sugerencia falsa (segunda porción): Manifiesta la elección a suscitar si es que el hecho sea falso. Ej. *"ignore"*.

4.1.3. Reglas

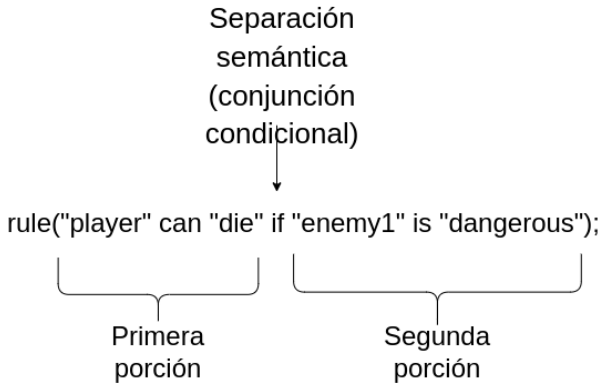

Figura 10: Estructura Básica de una Regla

La sintaxis de las reglas, como en el resto de las palabras claves, posee una división semántica que divide el contenido de los elementos en dos fragmentos. Cabe mencionar que las reglas deben contener si o si dos entidades por cada porción, separados por un verbo

Por tanto, su estructura es:

- 1°era Entidad (primera porción): Debe estar entre comillas dobles. Ej. *"player"*.
- Verbo (primera porción): Es la relación entre ambas entidades de la primera porción.
- 2°da Entidad (primera porción): Debe estar entre comillas dobles. Ej. *"die"*.
- Conjunción condicional: Indica la división semántica entre las dos porciones. En español expresa "cuando, a ser, etc.". Ej. *"if"*.
- 1°ra Entidad (segunda porción): Debe estar entre comillas dobles. Ej. *"enemy1"*.
- Verbo (primera porción): Es la relación entre ambas entidades de la segunda porción.
- 2°da Entidad (segunda porción): Debe estar entre comillas dobles. Ej. *"dangerous"*.

4.2. Semántica

La semasiología de *Obelisk* es en base a las palabras claves contenidas en el lenguaje y en consecuencia, describen la lógica y funcionamiento que tendrán.

Cabe mencionar que cada vez que se agreguen nuevas palabras claves, quedará un registro de estos en la base de conocimientos, el cual almacena toda la información contenida en el lenguaje. Este ultimo será explicado con detalle más adelante.

Finalmente, nuestro lenguaje posee tres palabras claves y su funcionamiento será descrito a continuación:

4.2.1. Hechos

Los hechos, como su nombre indica, describen una afirmación en referencia a la entidad declarada en los elementos de la palabra clave.

Es importante mencionar que si un hecho no esta presente en la base de conocimiento, se deja a entender que por defecto su valor booleano es *false*.

4.2.2. Reglas

Las reglas se utilizan para crear condiciones y el resultado de estas dependen fundamentalmente de los hechos.

Esto quiere decir que una regla tendrá un valor booleano *true* si existe el hecho que respalde tal regla. En caso contrario la regla quedará con valor *false*.

4.2.3. Acciones

Finalmente, las acciones definen que acto(s) se realizará(n) dependiendo si el hecho al cual alude tenga valor booleano *true*.

5. Implementación del Lenguaje

5.1. Arquitectura

5.1.1. Compilador

El compilador de Obelisk tiene como propósito leer el código fuente usando un *Lexer*. Luego se utiliza el *Parser* para analizar y aplicar operaciones en base a los tokens que encontró el *Lexer*. Tras esto, se crea una Base de Conocimiento, el cual posee la característica de poder consultar su información utilizando un software externo. Finalmente, el código relevante se transforma en código intermedio(IR) que luego será transformado en binario.

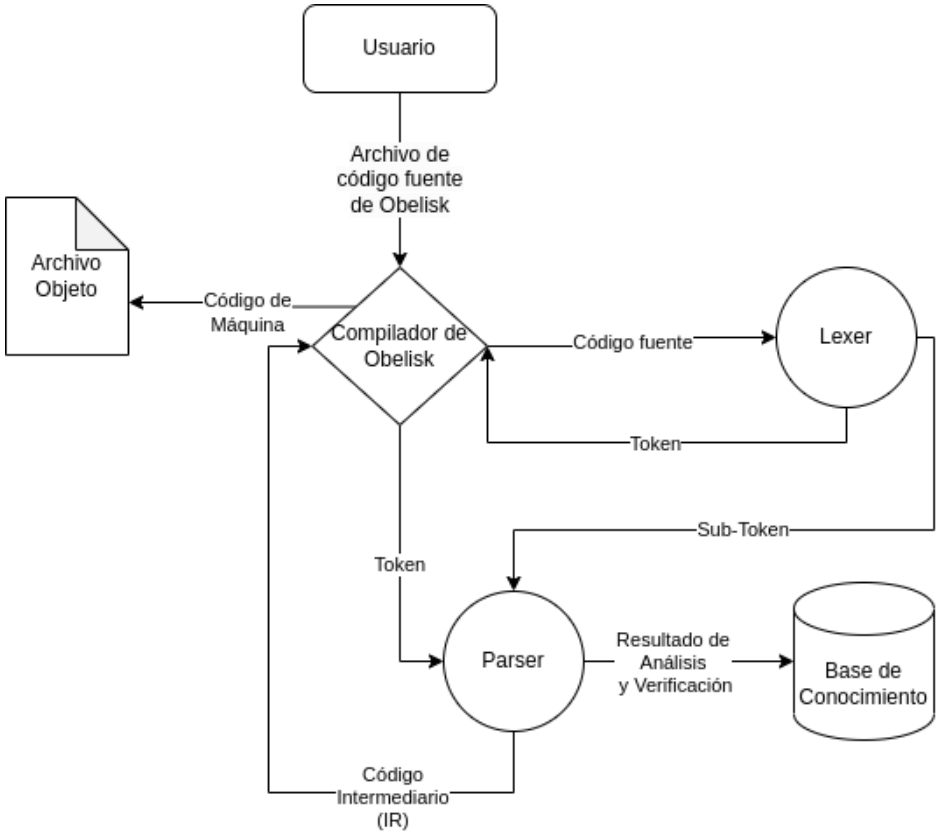

Figura 11: Diagrama Estructural del Compilador

5.1.2. Base de Conocimiento

La Base de Conocimiento es donde se almacena toda la lógica proveniente de los hechos, reglas y acciones. Su implementación fue hecha utilizando SQLite, el cual permite hacer consultas a la base de conocimiento utilizando el lenguaje SQL.

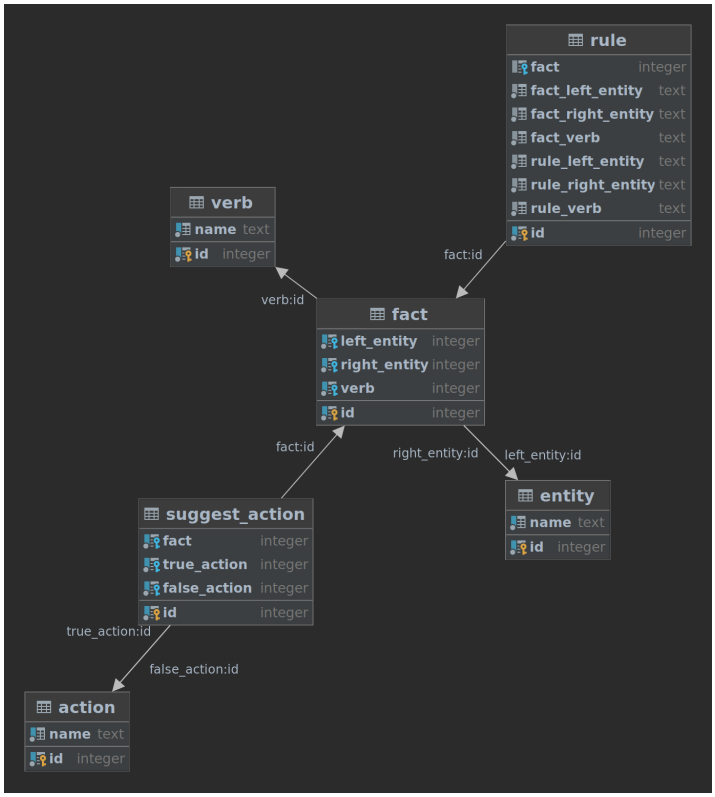

Figura 12: Estructura del Base de Conocimiento

La estructura de la base de conocimientos consiste en seis tablas principales las cuales se describen en el siguiente cuadro:

Nombre de Tabla	Propósito
entity	La tabla entity es usada para almacenar los nombres de las entidades presentes en los hechos, reglas y acciones.
verb	Se utiliza la tabla verb para almacenar los nombres de los verbos usados en los hechos y reglas.
action	La tabla action almacena los nombres de las posibles acciones que se pueden tomar.
fact	La tabla fact tiene los hechos verdaderos. Si existe una fila en esta tabla, la relación entre las dos entidades es verdadera.
rule	La tabla rule contiene reglas. Si una regla resulta ser verdad, se inserta el hecho en la tabla fact.
suggest_action	La tabla suggest_action contiene las dos posibles acciones que se pueden tomar dependiendo si el hecho es verdadero o falso.

Cuadro 2: Estructura del Base de Conocimiento

5.1.3. Librería

La librería de Obelisk permite interactuar y consultar a la Base de Conocimiento de Obelisk, con el uso de un software externo. Hay dos tipos de datos que puede devolver la consulta. Un string que representa la acción a tomar o un numero entre 0 y 1 que representa su valor de verdad.

6. Incorporación del Lenguaje en el Motor de Videojuegos

Para incorporar el lenguaje en un software externo, como por ejemplo nuestro videojuego, se utiliza la librería de Obelisk, que permite hacer consultas a la base de conocimientos y tomar acciones basado en ello.

6.1. Toma de decisiones

En cualquier software que utilice Obelisk, al ser este utilizado, se hacen consultas a la base de conocimientos, siendo la ultima compilada con anterioridad.

Después de esto, son dos los posibles resultados que pueden tener las consultas, siendo estos la acción que debe tomar el agente o un valor numérico que representa su resultado y que se puede interpretarse en el software externo.

6.2. Puesta en marcha

Basado en las decisiones elegidas en relación a la consulta de la base de conocimiento, es la responsabilidad del software poner la acción en marcha. Esto se puede llevar a cabo utilizando varias técnicas de inteligencia artificial tales como A*, maquina de estado, planificación de acciones orientados a metas, etc.

7. Evaluación del Desempeño del Agente

La evaluación del agente tiene como propósito estimar el correcto comportamiento de la inteligencia artificial dentro del videojuego, como por ejemplo la cantidad de monedas recolectadas, veces que murió, etc.

7.1. Sistema

7.1.1. Monitor

Desarrollamos un monitor hecho en el lenguaje GDScript, que es parte del motor de videojuegos Godot y que graba todo la sesión del agente y/o jugadores. La grabación del partido es basado en el ciclo de update del motor Godot, es decir que si la pantalla del jugador tiene 60Hz como su taza de refresco, se grabará 60 cuadros de información cada segundo. Al terminar un partido toda esta información se envía a un servidor donde se procesa y guarda para futuro análisis. Cabe mencionar que se envía todo los datos en formato JSON a un servidor de REST.

7.1.2. Servidor

El servidor fue programado en el lenguaje Go. La razón es por su alto nivel de rendimiento y bajo nivel de uso de recursos, tales como CPU y RAM. Esto nos permite recibir muchos más datos simultáneamente de varios partidos al mismo tiempo sin complicaciones. También el servidor provee una API de tipo REST con varios endpoints que permiten almacenar las partidas del juego y consultar la información guardada en ello para futuro estudio.

7.1.2.1. Estructura de Base de Datos

Todo la información de las partidas jugadas se almacenan en un base de datos MySQL.

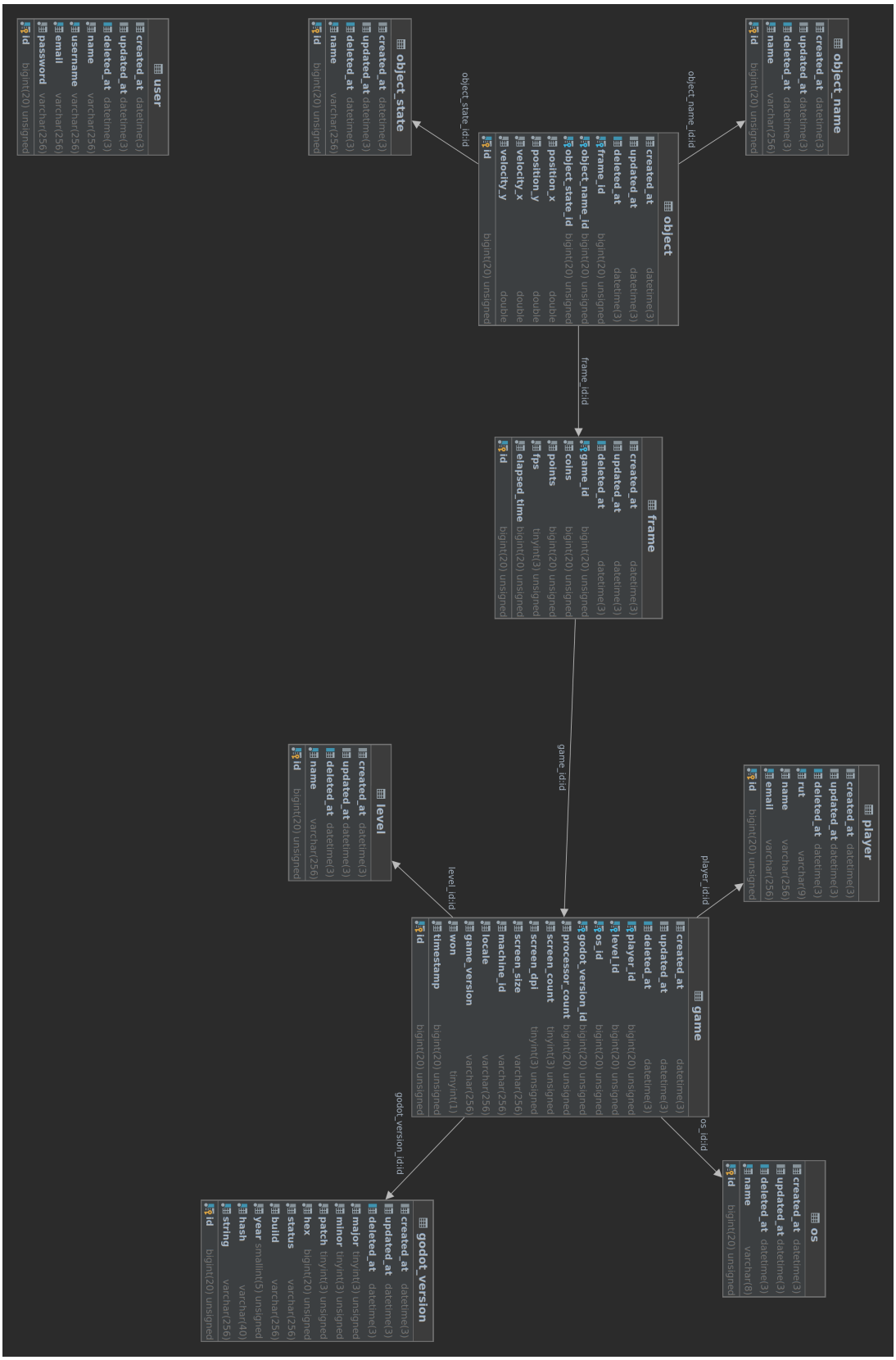

Figura 13: Estructura del Base de Datos del Backend

Nombre de Tabla	Propósito
frame	La tabla frame se utiliza para almacenar la cantidad de puntos, monedas, tiempo que pasó y posiciones de los objetos en el mundo cada cuadro que dibuja el juego.
game	La tabla game contiene información sobre el partido y el equipo que corre tal partido, incluyendo sistema operativo dimensiones de la pantalla y datos similares.
godot_version	La tabla godot_version se usa para almacenar las versiones de Godot que han corrido un partido del juego.
level	La tabla level es una tabla de parametros que contiene los nombres de los niveles que se puede jugar.
object	La tabla object se utiliza para almacenar el estado, posición y velocidad de un objeto en un cuadro especifico del partido.
object_name	La tabla object_name es una tabla de parámetros que contiene los nombres de todos los objetos que existen en un partido del juego.
object_state	La tabla object_state tiene todos los nombres de los estados de un objeto.
os	La tabla os es una tabla de parámetros que contiene los posibles sistemas operativos que pueden jugar el juego.
player	La tabla player es una tabla opcional donde se puede almacenar los datos de la persona que está jugando un partido para poder hacer un análisis sin ser anónima.
user	La tabla user contiene los usuarios y contraseñas de los personas que tienen permiso hacer consultas al API para obtener los datos y usarlos.

Cuadro 3: Estructura del Base de Datos del Backend

7.2. Análisis

Muchos tipos de análisis son posibles con los datos que recolectamos. Por ejemplo en algunas partidas podemos calcular el promedio de veces que ganó el agente, cuantas monedas obtuvo, cuanto demoró realizar todas las acciones, o en tomar una decisión, etc.

8. Trabajo Futuro

Dentro de este aspecto, tenemos varios elementos que proponemos desarrollar en un futuro, con la intención de complementar nuestro lenguaje de programación o su funcionalidad. Estos ya poseen una base y solo requieren su finalización.

- Perfeccionar nuestro lenguaje de programación para que sea *Touring-complete*.
- Análisis del comportamiento del agente dentro de cualquier software, utilizando los datos obtenidos del videojuego.
- Pasar la lógica de consultas proveniente de la base de conocimientos y así utilizar la GPU, con librerías tales como CUDA de Nvidia.
- Implementar lógica difusa en los hechos, reglas y acciones.

9. Conclusión

El trabajo realizado y la experiencia que tuvimos con nuestra tesis nos ha enriquecido como futuros profesionales en más de un modo, especialmente en el desarrollo de inteligencia artificial, el cual es una subrama de la programación que ha estado en constante evolución desde ya mas de 50 años.

Cabe destacar de igual forma el arduo camino de aprendizaje que recorrimos para desarrollar este proyecto, conociendo y experimentando con herramientas nuevas, desarrollo de soluciones ante problemas que no conocíamos, entre otros. Todo esto nos dejo valores y maestría necesaria para en un futuro, continuar con esta idea.

Referencias

[1] John McCarthy. «What is artificial intelligence?» En: (2007).

[2] David Vallejo y Cleto Martín. *Desarrollo de Videojuegos: Un Enfoque Práctico*. CreateSpace, 2015. ISBN: 978-1-51730-955-8.

[3] Ernest Adams y Joris Dormans. *Game Mechanics: Advanced Game Design*. New Riders, 2012. ISBN: 0-32182-027-4.

[4] Chris Bradfield. *Godot Engine Game Development Projects*. Packt, 2018. ISBN: 978-1-78883-150-5.

[5] Ariel Manzur y George Marques. *Sams Teach Yourself, Godot Engine Game Development in 24 Hours*. Pearson, 2018. ISBN: 0-13483-509-3.

[6] Royal Donut Games. *CPU Voxel Benchmarks of Most Popular Languages in Godot*. Abr. de 2022. URL: http://www.royaldonut.games/2019/03/29/cpu-voxel-benchmarks-of-most-popular-languages-in-godot/.

[7] Max Bramer. *Logic Programming with Prolog*. Springer, 2013. ISBN: 978-1-44715-486-0. DOI: 10.1007/978-1-4471-5487-7.

[8] tutorialspoint. *Prolog Tutorial*. Abr. de 2022. URL: https://www.tutorialspoint.com/prolog/.

[9] Mayur Pandey y Suyog Sarda. *LLVM Cookbook*. Packt, 2015. ISBN: 978-1-78528-598-1.

[10] Robert Nystrom. *Game Programming Patterns*. Genever Benning, 2014. ISBN: 0-99058-290-6.

[11] Grzegorz Jaśkiewicz. «Prolog-Scripted Tactics Negotiation and Coordinated Team Actions for Counter-Strike Game Bots». En: *IEEE Transactions on Computational Intelligence and AI in Games* 8.1 (2016), págs. 82-88. DOI: 10.1109/TCIAIG.2014.2331972.

[12] «Controlling Unreal Tournament 2004 Bots with the logic-based action language Golog / Jacobs, Stefan ; Ferrein, Alexander ; Lakemeyer, Gerhard». En: *Proceedings of the First AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE).* (2011), págs. 151-152.

Prolog

[7] Max Bramer. *Logic Programming with Prolog*. Springer, 2013. ISBN: 978-1-44715-486-0. DOI: 10.1007/978-1-4471-5487-7.

[8] tutorialspoint. *Prolog Tutorial*. Abr. de 2022. URL: https://www.tutorialspoint.com/prolog/.

[11] Grzegorz Jaśkiewicz. «Prolog-Scripted Tactics Negotiation and Coordinated Team Actions for Counter-Strike Game Bots». En: *IEEE Transactions on Computational Intelligence and AI in Games* 8.1 (2016), págs. 82-88. DOI: 10.1109/TCIAIG.2014.2331972.

[12] «Controlling Unreal Tournament 2004 Bots with the logic-based action language Golog / Jacobs, Stefan ; Ferrein, Alexander ; Lakemeyer, Gerhard». En: *Proceedings of the First AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE).* (2011), págs. 151-152.

Inteligencia Artificial

[1] John McCarthy. «What is artificial intelligence?» En: (2007).

[11] Grzegorz Jaśkiewicz. «Prolog-Scripted Tactics Negotiation and Coordinated Team Actions for Counter-Strike Game Bots». En: *IEEE Transactions on Computational Intelligence and AI in Games* 8.1 (2016), págs. 82-88. DOI: 10.1109/TCIAIG.2014.2331972.

[12] «Controlling Unreal Tournament 2004 Bots with the logic-based action language Golog / Jacobs, Stefan ; Ferrein, Alexander ; Lakemeyer, Gerhard». En: *Proceedings of the First AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE).* (2011), págs. 151-152.

LLVM

[9] Mayur Pandey y Suyog Sarda. *LLVM Cookbook*. Packt, 2015. ISBN: 978-1-78528-598-1.

Godot

[4] Chris Bradfield. *Godot Engine Game Development Projects*. Packt, 2018. ISBN: 978-1-78883-150-5.

[5] Ariel Manzur y George Marques. *Sams Teach Yourself, Godot Engine Game Development in 24 Hours*. Pearson, 2018. ISBN: 0-13483-509-3.

[6] Royal Donut Games. *CPU Voxel Benchmarks of Most Popular Languages in Godot*. Abr. de 2022. URL: http://www.royaldonut.games/2019/03/29/cpu-voxel-benchmarks-of-most-popular-languages-in-godot/.

Videojuegos

[2] David Vallejo y Cleto Martín. *Desarrollo de Videojuegos: Un Enfoque Práctico*. CreateSpace, 2015. ISBN: 978-1-51730-955-8.

[3] Ernest Adams y Joris Dormans. *Game Mechanics: Advanced Game Design*. New Riders, 2012. ISBN: 0-32182-027-4.

[4] Chris Bradfield. *Godot Engine Game Development Projects*. Packt, 2018. ISBN: 978-1-78883-150-5.

[5] Ariel Manzur y George Marques. *Sams Teach Yourself, Godot Engine Game Development in 24 Hours*. Pearson, 2018. ISBN: 0-13483-509-3.

[6] Royal Donut Games. *CPU Voxel Benchmarks of Most Popular Languages in Godot*. Abr. de 2022. URL: http://www.royaldonut.games/2019/03/29/cpu-voxel-benchmarks-of-most-popular-languages-in-godot/.

[10] Robert Nystrom. *Game Programming Patterns*. Genever Benning, 2014. ISBN: 0-99058-290-6.

[11] Grzegorz Jaśkiewicz. «Prolog-Scripted Tactics Negotiation and Coordinated Team Actions for Counter-Strike Game Bots». En: *IEEE Transactions on Computational Intelligence and AI in Games* 8.1 (2016), págs. 82-88. DOI: 10.1109/TCIAIG.2014.2331972.

[12] «Controlling Unreal Tournament 2004 Bots with the logic-based action language Golog / Jacobs, Stefan ; Ferrein, Alexander ; Lakemeyer, Gerhard». En: *Proceedings of the First AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE).* (2011), págs. 151-152.