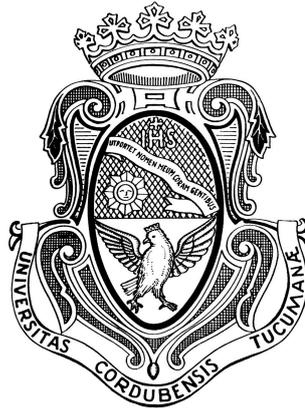


UNIVERSIDAD NACIONAL DE CÓRDOBA

FACULTAD DE MATEMÁTICA, ASTRONOMÍA Y FÍSICA



Trabajo final de la Licenciatura en Ciencias de la Computación

**Botrix: Plugin de *bots* colaborativos
dentro de un juego 3D**

Boris Godin

Luciana Benotti

CÓRDOBA

ARGENTINA

Noviembre 2013

Dedico esta tesis a las personas que amo.

A mi esposa y a mi hija por motivarme y ayudarme.

*A mis padres y mi hermano por ser
una parte importante de mi vida.*

Agradecimientos.

*Agradezco especialmente a Luciana,
por el apoyo que me dió en todo este tiempo.*

*A todos los profes de FaMAF,
por dedicarnos una gran parte de sus vidas.*

*A la comunidad de Valve que me
ayudó con la tecnología Source.*

- **Clasificación de Biblioteca:**

I.2.7

- **Palabras Clave:**

bots, colaborativos, waypoint, semántica, oración, planificación, planning, FF, plugin, 3D, motor, engine, Valve, Source, Half-Life, Deathmatch.

RESUMEN

Botrix es un plugin (un módulo de software) que ofrece creación y manejo de *bots colaborativos*. Un **bot** es un jugador artificial manejado por la computadora. Originalmente el plugin está hecho para el juego *Half-Life 2: DeathMatch*, pero puede ser adaptado para algunos otros juegos del motor Source. Los bots, dependiendo del juego, pueden jugar cada uno por su cuenta o en equipo, colaborando uno con el otro para lograr objetivos comunes. Para la navegación por el mapa se usaron técnicas de *waypoints* y *pathfinding*. Para que los bots puedan interactuar uno con el otro o con jugadores humanos se implementó un *sistema de chat*. Además los bots pueden usar distintos tipos de armas y esquivar (usar, romper o tirar) objetos. Los bots adaptan técnicas de *planificación*, más específicamente, extendiendo el planificador FF (Fast-Forward Planning System) para armar *planes colaborativos*, para lograr objetivos comunes entre varios jugadores, en un entorno dinámico y desconocido.

ABSTRACT

Botrix is a plugin (software module) that provides creation and management of *collaborative bots*. A **bot** is an artificial player handled by computer. Originally the plugin is made for the game *Half-Life 2: DeathMatch*, but it can be adapted for other games of Source Engine. Bots, depending on the game, can play each one on its own or in teams, collaborating with each other to achieve common goals. For map navigation *waypoints* and *pathfinding* techniques were used. We implemented a *chat system*, so bots can interact with each other or with human players. Furthermore bots can use different types of weapons and dodge (use, break or pull) objects. Bots adapt *planning techniques*, more specifically, they extend the FF planner (Fast-Forward Planning System) for the creation of *collaborative plans* to achieve common goals among several players, in a dynamic and unknown domain.

Índice general

1. Introducción	1
1.1. Objetivos generales	1
1.2. Motivación y descripción del tema	2
1.3. Contribuciones de la tesis	3
1.4. Mapa de la tesis	4
2. Motores de videojuegos	7
2.1. Definiciones básicas del área	7
2.2. Motores de videojuegos	8
2.2.1. Modificaciones de juegos	9
2.2.2. Motor <i>Source</i> de Valve Corporation	9
2.2.3. Source SDK	10
2.2.4. Consola en juegos Source	11
2.2.5. Sólidos y Entidades	12
2.3. Plugins	13
2.3.1. Servicios de motor Source	14
2.3.2. Plugins de motor Source	14
3. Navegación en un mapa	17
3.1. Waypoints	17
3.1.1. Waypoints en Botrix	18
3.1.2. Movimiento entre waypoints adyacentes	20
3.2. Búsqueda de camino	21
4. Chateando con bots	25
4.1. Introducción	25
4.2. Variables de la oración	27
4.3. Matching de oraciones	28
4.3.1. El algoritmo	29
4.3.2. Ejemplos	30
5. Planificación, PDDL y FF	33
5.1. Planificación y el lenguaje STRIPS	33
5.2. Extensión de STRIPS: ADL	35
5.3. Búsqueda de plan en el espacio de estados	35
5.3.1. Planificación con búsqueda hacia adelante en el espacio de estados	36
5.3.2. Planificación con búsqueda hacia atrás en el espacio de estados	36
5.3.3. Función heurística para la búsqueda	37
5.4. Planificación de orden parcial	38
5.5. Grafos de planificación	39
5.5.1. Grafos de planificación para estimación de heurísticas	41
5.5.2. El algoritmo GRAPHPLAN	42
5.5.3. Búsqueda de solución en el grafo	43
5.6. Sistema de planificación FF	44
5.6.1. Arquitectura del sistema	45

5.6.2. Función heurística	46
5.6.3. La búsqueda	47
5.6.4. Acciones útiles	47
6. Planificación colaborativa en un juego	49
6.1. El juego	49
6.1.1. Objetivos y comunicación	49
6.1.2. Representación de dominio del juego	50
6.2. Tareas de bots	52
6.3. Planificación colaborativa	54
6.3.1. Dominio en PDDL	55
6.3.2. Generación de problema en PDDL	58
6.4. Proposición y ejecución de un plan	60
6.5. La estrategia de los bots colaborativos	60
6.6. Generación automática del dominio de juego	61
7. Experimentos realizados	63
7.1. Ejecución 1, dos bots colaborativos	63
7.1.1. Investigación de un área nuevo	64
7.1.2. Tarea de investigar qué puertas abre un botón	66
7.1.3. Uso de planificador FF	67
7.1.4. Usando caja	69
7.1.5. Resultados de la ejecución	71
7.2. Ejecución 2, con un humano mentiroso	72
7.3. Ejecución 3, un humano colaborando con un bot	79
7.3.1. Expresiones que no están en el sistema	79
7.3.2. Cancelación de tareas	80
7.3.3. Llegando al objetivo	81
8. Conclusiones y trabajo futuro	83
8.1. Conclusiones	83
8.2. Trabajo futuro	84
Apéndices	89
.A. Configuración de Botrix	89
.B. Comandos de consola	94
.B.1. Comandos de waypoints	94
.B.2. Comandos de conexión entre dos waypoints	94
.B.3. Comandos de entidades	95
.B.4. Comandos de bots	95
.B.5. Comandos de configuración	95
.B.6. Comandos de PDDL	95
.C. Dominio del problema en PDDL	96
.D. El problema de 82 pasos en PDDL	101
.E. Diálogo entre dos bots	103
.F. Diálogo entre un humano y un bot	112
Bibliografía	133

Capítulo 1

Introducción

En este capítulo hablaremos sobre los objetivos de esta tesis, dando una breve introducción a inteligencia artificial en los videojuegos. Describiremos las contribuciones de esta tesis para el área de Inteligencia Artificial en Juegos y luego veremos el mapa de la tesis.

1.1. Objetivos generales

La meta principal del área de computación **Inteligencia Artificial** (IA) es crear una persona artificial. Los juegos ofrecen la posibilidad de implementar jugadores artificiales que puedan imitar el comportamiento humano hasta cierto punto. A estos jugadores artificiales se les dió el nombre **bot**, que viene de robot. Un bot es un programa controlado por la computadora que imita a un jugador y puede implementar técnicas de Inteligencia Artificial, adaptadas a las reglas del juego en general y al mapa jugado en particular.

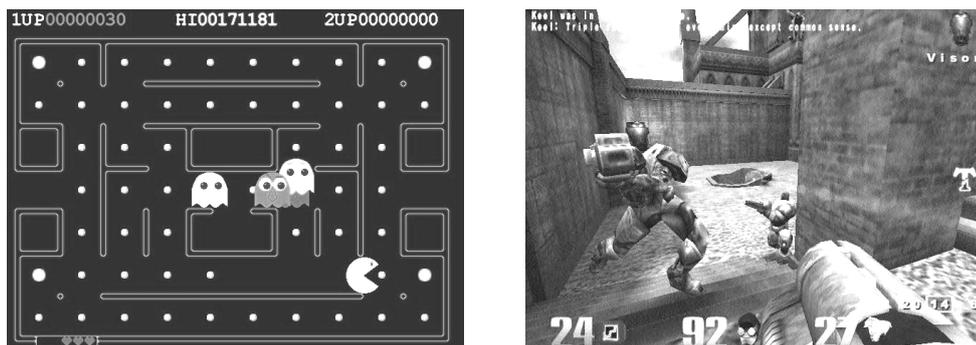


Figura 1.1: Pacman vs Quake III Arena

Se sabe de la importancia de la IA en los videojuegos modernos. Con el tiempo el rendimiento del microprocesador (frecuencia del reloj interno y la cantidad de núcleos), velocidad de memoria y discos duros fueron mejorando [Mal95] y gracias a eso los algoritmos usados en la IA; la inteligencia artificial en el juego

Pacman es mucho más simple que la inteligencia artificial de un bot del juego *Quake 3 Arena*: un monstruo de pacman camina hasta que choca una pared, aleatoriamente eligiendo la próxima dirección (al menos que el jugador esté cerca), en cambio el bot de *Quake 3 Arena* usa redes neuronales para definir el movimiento siguiente [Wav01].

Las tareas básicas de los desarrolladores de videojuegos son las siguientes:

- Proveer a los diseñadores herramientas para construir el mundo virtual.
- Implementar partes gráficas y físicas, algunas de las cuales proveen placas de video o librerías estándar (como OpenGL y NVidia PhysX).
- Implementar la IA de los caracteres del juego.

Últimamente hay muchas investigaciones sobre Sistemas Inteligentes Colaborativos, IA para trabajo en equipo (juegos como *Sniper* y *Mafia 2*). Es impresionante cuando el jugador sale junto con su equipo contra otros soldados, se puede observar como su equipo lo cubre, o alguien le grita “Abajo!” cuando cae una granada cerca. Entre todos los efectos de balas, humo y fuego el juego parece muy real e interesante. Esto es importante en esta tesis, porque queremos lograr que varios bots trabajen en forma colaborativa con un objetivo común. Vamos a ver situaciones donde los bots necesiten armar un plan conjunto para pasar de nivel, es decir, hablando uno con el otro, averiguan información sobre el entorno o se ponen de acuerdo para definir la secuencia de acciones a realizar. Los bots también podrán colaborar con uno o varios jugadores humanos, haciéndoles preguntas o pidiéndoles ayuda para hacer ciertas cosas.

En los videojuegos de disparos en primera persona, conocidos como **FPS** (First Person Shooter), las acciones básicas son mover al personaje y usar un arma. Perspectiva de primera persona tiene por meta dar la impresión de ser el personaje y los gráficos en tres dimensiones aumentan esta impresión. Vamos a usar un videojuego de este tipo para crear jugadores artificiales: se llama *Half-Life 2: Deathmatch* [H12Dm04].

Por supuesto que es complicada la tarea de modelar el comportamiento de la computadora para que simule al humano. Y todavía más compleja es la simulación de interacción entre máquina-máquina o máquina-humano para conjuntamente lograr objetivos comunes. En este trabajo primero veremos las técnicas básicas usadas por los bots: navegación por el mapa, el uso de armas y objetos, simulación de visibilidad y módulo de chat. Después nos concentraremos en trabajo cooperativo [Cla96] entre varios jugadores (máquina-máquina o jugador-máquina) y veremos como las técnicas de planificación automatizada pueden ser adaptadas de mono-agente a multi-agente [NGT04], para que los bots puedan lograr objetivos comunes.

1.2. Motivación y descripción del tema

En juegos FPS, tales como *Counter-Strike* o *Quake*, el jugador ve el mundo del juego desde perspectiva de primera persona y se mueve en un mundo 3D virtual de tiempo real. Teniendo distintos tipos de armas y defensa, la tarea en los juegos FPS online es quedarse vivo la mayor parte del tiempo, además de lograr objetivos tales como liberar rehenes, plantar una bomba, tomar una bandera y llevarla a la base, etc. Hay personas que no tienen acceso a internet para jugar con otros jugadores humanos online, por lo que es deseable tener bots que puedan jugar con ellos sin tener que acceder a una red de computadoras. Además los bots pueden ayudar a uno a aprender a jugar antes de empezar a jugar con jugadores más avanzados.

En este proyecto se diseñarán y se implementarán bots colaborativos, jugadores artificiales manejados por la computadora, que podrán jugar en equipo para lograr ciertos objetivos. Vamos a utilizar el motor de videojuegos llamado *Source* [Src04], del juego *Half-Life 2: Deathmatch* [Hl2Dm04], y además crear algunos mapas de interés. Se decidió utilizar este entorno porque es fácil de usar y es un entorno tridimensional de interacción en el cual la variedad y cantidad de acciones que el bot puede realizar constituye un dominio de planificación desafiante para el área de planificación.

El sistema tiene que cumplir una serie de requisitos a nivel técnico, esto es hacer uso eficiente de los recursos, tanto el uso de CPU como de la memoria. Dado que el código de implementación de bot está ejecutándose al mismo tiempo que el motor del juego, la ejecución del código de bot no debería consumir más de un porcentaje bajo del tiempo de CPU disponible. Picos en el uso de CPU producen interrupciones en el juego, que se refleja en una experiencia negativa de los jugadores humanos. En nuestro caso, esta restricción es severa porque habrá más de un bot ejecutándose al mismo tiempo en el juego. Por lo tanto, además de ver los algoritmos de planificación y pathfinding existentes que resultaron ser los más apropiados para bots colaborativos en términos de eficiencia, veremos como se puede adaptarlos a los juegos.

1.3. Contribuciones de la tesis

Esta tesis abarca tres temas distintos: *navegación automática*, *comunicación* a través del lenguaje natural y *planificación* automatizada colaborativa. A continuación veremos en detalle en qué la tesis contribuye a estas áreas:

- Se propone dividir en *varios frames* el cálculo del camino entre 2 puntos del mapa arbitrarios, por cuestiones de eficiencia para la navegación por un mapa.

- Vamos a ver una forma de *identificar el significado de una oración y detectar valores de las variables* de la misma (por ejemplo, el número de puerta de la que se está hablando).
- *Planificación colaborativa* entre varios bots sin tener una entidad que los controle. Esto es, cada bot “piensa” por sí mismo, pero se ponen de acuerdo para alcanzar el objetivo del juego.
- *Generación automática de dominio* de un juego de tipo *laberinto* usando planificación automatizada.
- Usando todo lo anterior se arma una arquitectura híbrida que permite *colaboración entre humanos y bots* para alcanzar un objetivo en común.

1.4. Mapa de la tesis

Primero se verá lo que es un *motor gráfico*, mods y plugins en el capítulo 2. En particular vamos a hablar sobre características del motor Source de Valve y juegos hechos con el mismo. Explicaremos para que sirven los comandos de consola, veremos en que consiste un mapa de un juego tridimensional.

Se usarán técnicas de *pathfinding* [MilFun09], es decir encontrar el camino a seguir entre dos puntos del mapa distintos, para que los bots puedan moverse en terrenos complejos con elevaciones, escaleras, habitaciones, ventanas, etc. Tales puntos del mapa reciben el nombre de waypoint [MilFun09] y los veremos en el capítulo 3.

Si en el mundo real dos o más personas quieren lograr un objetivo común deben usar un medio de *comunicación* apropiado. Lo mismo se aplica a los bots, van a usar el chat del juego para expresar sus opiniones (creencias), necesidades (pedidos de acciones) o solamente para saludar al jugador. En el capítulo 4 veremos una técnica de matching de cadenas para poder darle semántica (significado) a las oraciones.

En el capítulo 5 vamos a ver distintas técnicas de *planificación*. Introduciremos el lenguaje STRIPS y su extensión, ADL, veremos búsquedas del plan en el espacio de estados, hacia-adelante y hacia-atrás con función heurística. Brevemente veremos planificación de orden parcial y grafos de planificación. En especial describiremos como funciona el planificador FF [Hof01], el cual se usó en este trabajo.

En el capítulo 6 primero definiremos el juego en sí y como el bot lo ve. Veremos las tareas que puede realizar el bot, como hace para proponer y ejecutar un plan. Adaptaremos las *técnicas de planificación* para que los bots puedan colaborar, definiremos las acciones y sus efectos para los bots en un lenguaje llamado PDDL, veremos como se forman los estados inicial / final y también los posibles estados intermedios. También se verá generación automática del dominio para este tipo de juegos.

Describiremos tres *experimentos* realizados en el capítulo 7. El primer experimento muestra como los bots eligen una tarea, la llevan a cabo y como usan el planificador. En este experimento los bots descubrirán todo lo posible sobre el nivel jugado. En el segundo experimento vamos a ver como un humano miente a un bot. Y en el tercer experimento veremos colaboración de un bot con un humano.

Por último, hablaremos sobre *conclusiones* en el capítulo 8. Veremos los problemas encontrados y como podrían ser resueltos.

Describiremos la configuración del plugin en el apéndice .A. Los comandos de consola para manejo de bots están descriptas en el apéndice .B. Dominio del problema en PDDL para los bots está en el apéndice .C y un dominio del juego generado automáticamente se describe en el apéndice .D. El chat completo del experimento 1 se muestra en el apéndice .E y el chat del experimento 3 se puede ver en el apéndice .F.

El código fuente del plugin y algunos videos de muestra se puede ver en [\[Botrix13\]](#).

Capítulo 2

Motores de videojuegos

En este capítulo primero veremos las definiciones básicas del área que usaremos a lo largo de este capítulo, luego daremos una introducción a motores de videojuegos, y en particular veremos el motor Valve Source Engine [Src04]. Descubriremos que son los mods (modificaciones de juegos), plugins, consola, de que están compuestos los mapas o niveles del juego. Pero antes de todo esto vamos a ver algunas definiciones básicas, usadas en el área de los juegos que usaremos a lo largo de este capítulo.

2.1. Definiciones básicas del área

Un videojuego para **jugador-único** (inglés *single-player*), es un videojuego donde se espera que lo juegue un único jugador humano durante la sesión del juego. Por otro lado, videojuegos tipo **multi-jugador** (inglés *multiplayer*) permiten la interacción de dos o más jugadores al mismo tiempo, ya sea de manera física en una misma consola, o mediante servicios en línea u otro tipo de red con personas conectadas a la misma. También hay juegos que tienen los dos modos; en modo jugador-único el usuario normalmente juega contra bots, mientras que en modo multi-jugador cada jugador se enfrenta a otros jugadores humanos.

Para que varios jugadores humanos puedan jugar juntos e interactuar uno con el otro, existen los llamados **servidores**. Son ordenadores que tienen una dirección pública en la red, para que los jugadores puedan conectarse con ellos. Básicamente en los juegos multi-jugador, un jugador envía su movimiento al servidor, y este se encarga de enviarlo a otros jugadores conectados. El servidor además puede tener otras funcionalidades, por ejemplo controlar que el jugador envíe movimientos válidos al servidor y si el jugador hace trampa desconectarlo temporalmente o banearlo (agregarlo a la lista negra) permanentemente.

Un **mapa** o **nivel** de un videojuego es una parte del juego, un escenario con sus objetos, personajes y objetivos propios. Es un área específica dentro del “mundo” ficticio de un videojuego (por ejemplo un edificio o un campo de batalla), y cada nivel puede tener una *misión*, por ejemplo recoger un objeto señalado o llegar

hasta un punto determinado. En videojuegos de jugador-único una vez que la misión ha sido completada, el jugador pasa al próximo nivel; si falla, normalmente el jugador deberá repetirla completamente, o recargar el juego desde un punto guardado anteriormente, si el juego lo permite.

Los **NPC** (inglés Non-Player Character) o personaje no jugador es un personaje no humano dentro del juego, manejado por la computadora. En el capítulo anterior habíamos hablado sobre bots, ellos son NPCs. En juegos de rol, el término NPC se refiere a aliados o personajes con una actitud neutral hacia el jugador, mientras que a los personajes hostiles se les conoce como **mobs**.

Se denomina **frame** (*cuadro* en español), a una imagen particular dentro de una sucesión de imágenes que componen la animación del juego. La continua sucesión de estas imágenes producen la sensación de movimiento, fenómeno dado por las pequeñas diferencias que hay entre cada uno de ellas. La velocidad gráfica del juego se mide en **fps** (frames por segundo). Hoy en día la velocidad de un juego debe ser al menos de 60 fps para que el jugador tenga una buena experiencia jugando.

2.2. Motores de videojuegos

Un **motor de videojuego** (inglés *engine*) es un término que hace referencia a una serie de rutinas de programación que permiten el diseño, la creación y la representación de un videojuego. Existen motores de juegos que operan tanto en consolas de videojuegos como en diferentes sistemas operativos. La responsabilidad de un motor es proveer al videojuego las siguientes funcionalidades (entre otras):

- **Administración de memoria.** Dependiendo del sistema operativo, el manejo de memoria, esto es, reservación de un bloque de memoria y su liberación puede ser lento. Este componente es el encargado de optimizar este manejo. Normalmente lo que se hace es calcular cuanta memoria se va a necesitar, reservar todo el bloque y manejarlo internamente.
- **Renderizado para imágenes/modelos 2D/3D** y sus animaciones. Normalmente es un envoltorio para las operaciones gráficas en DirectX u OpenGL.
- **Motor físico** o detector de colisiones. Su propósito es hacer que los efectos físicos de los objetos tengan las mismas características que en la vida real, teniendo en cuenta, por ejemplo, gravedad, masa, fricción, restitución, etc.
- **Reproducción de música y sonidos.**
- **Sistema de scripting**, que normalmente se usa para presentar las escenas de la historia del juego. Por medio de los comandos del script se controlan los personajes del juego y de esta forma la escena forma parte del juego y no un video aparte.

- **Inteligencia artificial** del juego. En este módulo va todo relacionado con los NPCs y los mobs, formas de movimiento, lo que es manejo de armas, vehículos, etc. Cabe destacar que el término *Inteligencia Artificial en Juegos* se aplica a algoritmos usados en juegos que *pueden no pertenecer* al área de investigación conocida como *Inteligencia Artificial*. De aquí en más usaremos el término IA que se aplica a Inteligencia Artificial en *Juegos*.
- **Redes** para juego online. Este módulo se usa en juegos multi-jugador: es el encargado de transmitir los movimientos del jugador al servidor y de aplicar los movimientos de otros jugadores recibidos del servidor.

El proceso de desarrollo de un videojuego puede variar notablemente por re-utilizar o adaptar un mismo motor de videojuego para crear diferentes juegos. De hecho, hoy en día, es la base del diseño de gran mayoría de los videojuegos.

2.2.1. Modificaciones de juegos

En el mundo de los videojuegos un **mod** (del inglés *modification*) es una modificación del juego original, posiblemente con nuevos personajes, mapas, diálogos, objetos, etc. Prácticamente todos los juegos modernos incorporan herramientas y documentación necesaria para poder modificarlos y así crear el mod. Actualmente las compañías grandes tienen mucho interés en que los fans no solo jueguen, sino que creen sus propios mods, dándoles las herramientas y tutoriales, para así darle más fama al juego original y extender su tiempo de vida.

Una **conversión total** es un mod que cambia el juego por completo. Es como crear y jugar un juego completamente nuevo. El mejor ejemplo de las conversiones totales es *Counter-Strike* [Cs99], mod de *Half-Life* [H198], que incluso llegó a ser más popular que el juego original. Las **conversiones parciales** en cambio son aquellas que agregan cosas o modifican el juego original, por ejemplo modificar objetos, alterar sonidos, cambiar algunos diálogos o texturas.

Para hacer experimentos en los que nuestros bots puedan colaborar en resolver un problema (que es el objetivo de esta tesis), se creó un mod, una conversión parcial a partir de *Half-Life 2: Deathmatch* (HL2:DM). En nuestro mod, **BorzhMod**, el objetivo es salir del laberinto, esto es, llevar a todos los jugadores al área-objetivo. La única diferencia entre BorzhMod y HL2:DM es que el mod detecta cuando todos los jugadores estén en el área-objetivo y termina el mapa. Además del mod, se crearon algunos mapas de interés para que los bots puedan tener un desafiante dominio de planificación.

2.2.2. Motor *Source* de Valve Corporation

Valve Source Engine [Src04] es un motor de videojuegos desarrollado por la empresa Valve Corporation para las plataformas Microsoft Windows (32 y 64 bits), Mac OS X, Xbox, Xbox 360, y PlayStation 3. Debutó en 2004 con los videojuegos

Half-Life 2 [Hl2-04] y Counter-Strike: Source [Css04]. Entre sus características se destacan: la flexibilidad de uso, división por módulos, sincronización de movimientos de labios con el discurso vocal, tecnología para expresar las emociones faciales a partir del sonido y el sistema de física. La parte física del motor Source incluye los principios del llamado “muñeco de trapo”, esto es una técnica de la relajación de un cuerpo después de morir, por ejemplo si tal cuerpo es movido, no queda duro sino que se mueve como si realmente fuera un muñeco de trapo.



Figura 2.1: Juego Half-Life 2 de Valve Corporation

Se eligió este motor porque es fácil de usar, está bastante bien documentado y es muy completo en cuanto a las operaciones gráficas y físicas permitidas. Además incluye editor de mapas que se usó en la creación de niveles necesarios para definir dominios de planificación interesantes.

2.2.3. Source SDK

Source SDK (Software Development Kit: Herramientas para Desarrollo de Software) es un conjunto de herramientas que se puede usar para crear mods de motor Source. Las herramientas más importantes de Source SDK son:

- Herramienta *Valve Hammer Editor* para crear o modificar mapas.
- Herramienta *Model Viewer* para ver los modelos tridimensionales y sus animaciones correspondientes, los puntos de fijación, huesos, etc. También se puede utilizar para ver accesorios, armas, edificios, y casi cualquier otro objeto 3D.
- Herramienta *Face Poser* para editar las expresiones faciales, los gestos y también para sincronizar el habla (de un archivo de sonido) con expresiones faciales de los personajes.
- Aplicación que permite la creación de un mod vacío con el código fuente para librerías *cliente* y *servidor*. La librería cliente se ejecuta en la máquina del cliente, es decir jugador y la librería servidor se ejecuta en el servidor.

En este contexto es difícil hablar sobre modificación de algún juego en particular, ya que el código fuente del Source SDK puede usarse tanto para crear mod parecido a *Half-Life 2* (que es single-player) como para mod parecido a *Half-Life 2: Deathmatch* (que es multi-player), sin embargo, los mods originales son de código cerrado (privado).

Vimos que hay varias etapas en la creación de un mod de un juego de motor Source, que puede incluir:

- Modificar gráficamente el juego, se puede crear nuevos objetos, personajes, mapas, texturas, sonidos, etc. De hecho solamente creando mapas, videos, texturas y el menú nuevo, el mod va a ser completamente diferente del juego original.
- Toda la IA, esto es, manejo de NPCs, armas, objetos, etc. está en el código de las librerías cliente (el código que se ejecuta en la PC del usuario) y servidor (el código que se ejecuta en el servidor), así que todo lo necesario para crear bots se puede hacer modificando el código de estas librerías. Esto sería lo más fácil para hacer, ya que estas librerías tienen acceso al código fuente de los personajes, armas y objetos (tales como botiquín, armadura, caja, etc.), mientras que un plugin tiene acceso limitado a los objetos y personajes del mapa. Sin embargo, si creamos el mod con los bots, esto sería nuestro propio juego, idéntico al juego modificado con la diferencia que se podrá agregar bots, cuando en realidad estamos interesados en poder agregar bots al juego mismo sin modificarlo. Además solamente se puede modificar el código fuente de las librerías si estas librerías no son de código cerrado.
- Afortunadamente podemos crear un *plugin* que permitirá agregar bots a algún juego sin modificar el juego en sí. El juego va a cargar al plugin como si fuera su propio código. Por esta razón nuestro plugin, **Botrix** (BOT's matRIX), puede ser agregado a juegos hechos con el motor Source 2013 con algunas modificaciones del código del plugin, ya que el motor está compartido entre todos estos juegos. Hablaremos más sobre plugins en la sección 2.3.

Antes de hablar sobre funcionalidades que provee el motor Source a los plugins, veremos que es la consola y de que está compuesto un mapa del motor Source.

2.2.4. Consola en juegos Source

Los juegos hechos con Valve Source Engine tienen consola, donde uno puede escribir comandos o cambiar valores de las variables del juego. Estos comandos hacen distinto tipo de trabajo, por ejemplo se puede cambiar la fuerza de gravedad en el servidor, cantidad de daño recibido de un disparo, expulsar a un jugador del servidor, etc.



Figura 2.2: Consola de Half-Life 2

El motor permite al plugin crear variables y comandos para consola. Los vamos a utilizar por ejemplo para crear bots, mostrar alguna información de debug, ver donde se dirige un bot, etc.

Para poder ejecutar comandos o modificar variables de consola que afectan al servidor, un cliente que se conecta a este servidor tiene que tener permisos para poder hacerlo. Algunos usuarios pueden tener más permisos que otros, por una simple cuestión de seguridad. A continuación vamos a llamar a los usuarios que tienen acceso a los comandos del servidor **administradores**. Este es el caso de un juego online, pero cuando el cliente juega en modo jugador-único, él mismo es administrador con permisos máximos.

Por estas razones, en el plugin se incluirán comandos de consola para agregar o modificar administradores del servidor. Todos los comandos del plugin Botrix se incluyen en el Apéndice .B.

2.2.5. Sólidos y Entidades

Los **sólidos** (inglés *brush*) son objetos 3D convexos, que son creados durante el diseño del mapa con la herramienta Valve Hammer Editor. Los diseñadores del mapa usan estos sólidos para definir la forma del mundo. Los sólidos son inanimados, de bajo detalle, iluminados con mapas de iluminación pre-computados y rígidos (no se deforman). Se separan en 4 tipos básicos: *bloques*, *rampas*, *cilindros* y *agujas*. Puede no parecer gran cosa pero es la arquitectura primaria de todos los mapas creados con Hammer Editor: los sólidos se pueden combinar para hacer casi cualquier forma posible. Esto es conocido como *geometría constructiva de*

sólidos (inglés *constructive solid geometry*) y se usa en muchos otros juegos de otros motores.

Para el motor Source todas las cosas existentes en el mapa son **entidades**: sillas, mesas, jugadores, armas, eventos, etc. Son los objetos que se mueven, tienen sonido o son interactivos. Una entidad es cualquier cosa que hace cualquier tipo de operación o tarea en un nivel. Es más, el mundo, es decir el conjunto de sólidos que componen el mapa, también es una entidad.

Las entidades pueden ser de alguno de los siguientes tipos:

- Entidades basadas-en-punto: son entidades que tienen una posición determinada en la grilla del mapa. Se separan en:
 - ◊ Entidades lógicas, invisibles: son controladores de AI, luces, eventos, coreografía, etc.
 - ◊ Entidades con modelo: son objetos del mundo tales como ropero, mesa, ventana, pueden ser interactivos, por ejemplo, un vehículo.
 - ◊ Entidades NPC, son entidades que tienen su propia AI (por ejemplo bots lo son) y si son visibles tienen modelos asociados.
- Entidades basadas-en-sólidos: son entidades que dependen de un sólido para su presencia física. Normalmente son *disparadores*, es decir tienen un efecto cuando el jugador se posiciona en su área o los toca, por ejemplo puertas, plataformas o elevadores.

En caso de un juego multi-jugador, las entidades pueden ser *transmisibles en red* (inglés *networkable*), por ejemplo, la entidad “jugador” puede transmitir su posición, sus ángulos o cuando cambie de arma. Otro ejemplo es el resultado de disparar una bala: huellas en la pared o animación de un jugador herido. Hay entidades que no son transmisibles ya sea porque son fijas, como el mundo o porque la transmisión en red tiene costo elevado y solamente se transmite lo primordial y necesario. Por ejemplo, en Half-Life 2 no se transmiten objetos chicos, como botellas y pequeñas piedras, entonces si un jugador esta disparando a una botella, la cual se rompe, esto es visible solamente por el jugador mismo, los otros jugadores lo ven al primero disparar al vacío.

2.3. Plugins

Un **plugin**, **add-on** o **complemento** es un componente de software (generalmente una librería) que agrega funcionalidades a una aplicación de software más grande. Un ejemplo conocido es Adobe Flash Player, un complemento que carga animaciones multimedia interactivas y se usa, por ejemplo, para ver videos dentro de un navegador web.

El **plugin** (de inglés “enchufado”) se relaciona con la aplicación principal o *host* por medio de un conjunto específico de funciones. Este conjunto de funciones

recibe el nombre de API: Application Programming Interface, lo que significa Interfaz de Programación de Aplicaciones. Básicamente la aplicación principal define funciones abstractas (API) que el plugin tiene que implementar para que la aplicación principal pueda adquirir funcionalidad extra. Si una aplicación admite plugins, esto permite que los *desarrolladores externos* (3rd party) colaboren con la aplicación principal extendiendo sus funciones.

2.3.1. Servicios de motor Source

Los servicios que provee el Valve Source Engine al plugin son los siguientes:

- Manejo del juego: acceso a los comandos y variables de consola, aviso de eventos en el servidor.
- Acceso a la lista de los **diccionarios** de todas las entidades existentes en el mapa. Un diccionario son las cosas más importantes y genéricas de las entidades, como ser su modelo, su posición y área, ángulos, etc. El plugin solamente tiene acceso a los diccionarios de las entidades, no a las entidades mismas. Por ejemplo, de la entidad “jugador” se puede obtener su arma actual, pero no del diccionario de esa entidad. Esto genera dificultades adicionales para adquirir información del dominio del juego, sin embargo, veremos luego como superarlas.
- Un conjunto de funciones útiles para emular visibilidad: *hojas de visibilidad* y *trazo de rayos*. Las veremos más adelante cuando describiremos como se emula la visibilidad del bot (capítulo 6).
- Manejo de jugadores artificiales: teclado virtual y funciones que provee el motor para agregarlos o sacarlos del servidor. También las veremos más adelante, en el capítulo 6.

2.3.2. Plugins de motor Source

Los juegos hechos con el motor Source admiten 3rd party plugins, por lo tanto Botrix puede funcionar en algunos de estos juegos. Solamente hay que implementar un conjunto de funciones, esto es, una interfaz (es decir, una clase abstracta), e indicarle al motor donde esta la librería implementada. El juego al iniciar va a buscar la librería en el lugar indicado y va a llamar a sus funciones en su debido tiempo.

El motor Source va a avisarle al plugin sobre los eventos del juego por medio de las funciones de la interfaz del plugin. En la figura 2.3 se pueden ver algunos de estos eventos y la respuesta del plugin a los mismos, como para dar una intuición sobre como funcionan los plugins en los juegos Source.

Evento	Respuesta de plugin Botrix
Inicio de un nuevo nivel	El plugin carga los waypoints y la lista de entidades del mapa.
Nuevo frame del juego	Este evento se recibe cuando se produce nuevo frame del juego. El plugin lo usa para detectar los cambios en el dominio del juego y avisarles a los bots, los cuales determinan su próximo movimiento.
Un nuevo jugador entra al juego	El plugin busca a este jugador en la lista de administradores y si está en la lista, le da permisos para ejecutar los comandos de consola.
Un jugador introduce un comando en la consola	Si el comando es un comando del plugin y si el jugador tiene permisos administrativos, el plugin ejecuta al comando.
Fin de nivel	El plugin libera los recursos y elimina a los bots del juego.

Figura 2.3: Algunos eventos que el motor Source notifica al plugin.

En el siguiente capítulo hablaremos sobre como los bots hacen para poder ir a cualquier lugar del mapa. Veremos que son los waypoints y el algoritmo A*.

Capítulo 3

Navegación en un mapa

En este capítulo primero definiremos a los waypoints, la técnica usada en Botrix para que los bots puedan llegar a un lugar arbitrario dentro del mapa jugado. Después veremos distintos tipos de waypoints, tipos de conexiones entre ellos y un algoritmo de búsqueda para encontrar un camino entre 2 waypoints arbitrarios, si tal camino existe.

3.1. Waypoints

A diferencia de los humanos los bots no pueden “ver” el nivel jugado y analizar lo que ven. Necesitan algo que les indique donde hay una puerta, una escalera, un botón, una tubería, por donde pueden pasar y por donde no pueden. El uso de **waypoints** [MilFun09] es una de las técnicas más usadas de navegación por mapas, ayuda al bot encontrar un camino a través del mapa jugado. Son coordenadas para ubicar puntos de referencia (que son tridimensionales en el juego usado en esta tesis), utilizados por el bot en la navegación en el mapa del juego. La palabra viene compuesta del inglés *way* (camino) y *point* (punto). Se emplean para trazar caminos mediante agregación secuencial de puntos. En estos puntos puede haber objetos como armas o botones, pero en general sirven para guiar a los bots por los mismos.

Se puede pensar en los waypoints como un grafo dirigido, donde los waypoints forman los vértices o nodos del grafo. Las **conexiones** entre waypoints (aristas del grafo) indican qué waypoints son alcanzables desde el actual. El bot camina desde un waypoint al otro en línea recta y *cambia de waypoint* cuando la distancia del bot al waypoint destino se hace chica. Un **camino** es una *secuencia* de waypoints, donde siempre existe una conexión entre 2 waypoints adyacentes en la secuencia.

Normalmente los waypoints (por ejemplo en PodBot [Flo06], un plugin de bots de juego Counter-Strike [Cs99]) tienen lista de waypoints adyacentes, y etiquetas que indican el tipo de waypoint: puede ser un waypoint para agacharse (por ejemplo para entrar a una tubería) o saltar sobre una superficie alta, en este caso

tendrá también la dirección adonde saltar o ir después de agacharse. Algunas veces las conexiones también tienen sus etiquetas respectivas.

Los waypoints no son visibles durante el juego, pero los diseñadores del mapa los marcan de alguna forma mientras los crean (normalmente con una línea vertical).

3.1.1. Waypoints en Botrix

En Botrix, los waypoints además de tener coordenadas de posición (x, y, z) tienen **etiquetas** que indican el tipo de waypoint y estas etiquetas puede tener varios argumentos. Por ejemplo un waypoint de tipo arma, balas o medicina indica que hay uno de esos objetos en ese waypoint, waypoints camper o sniper indican que es un buen lugar de escondite o de francotirador. Cada waypoint tiene un conjunto de conexiones a otros waypoints. Se puede pensar en una conexión que es una línea recta que une a los waypoints *vecinos*. El bot para llegar al waypoint vecino tiene que correr en línea recta hacia ese vecino. Las conexiones también tienen etiquetas, las cuales también pueden tener argumentos. Por ejemplo, la etiqueta puede ser de tipo “salto”, es decir, que para llegar al waypoint destino el bot tiene que saltar desde el waypoint origen (ver figura 3.1). Saltará cuando llega el tiempo de inicio de la acción (que se guarda en el argumento de la etiqueta de la conexión), contando desde el momento de llegada al waypoint origen.



Figura 3.1: Conexión entre 2 waypoints con etiqueta “salto”.

Tipos de etiquetas que puede tener un waypoint en Botrix son los siguientes:

- **Stop.** Hay que parar antes de llegar a ese waypoint. Sirve para caminar con cuidado en lugares altos.
- **Camper.** Indica que es buen lugar de escondite.
- **Sniper.** Indica que es buen lugar para francotirador.

- **Arma.** Se puede agarrar un arma al llegar a ese waypoint. el tipo de arma está en el argumento del waypoint.
- **Botiquín.** Hay un botiquín cerca. Argumento indica cantidad de vida que restaura.
- **Chaleco.** Hay un chaleco allí.
- **Botón.** Desde ese waypoint se ve un botón al cual se le puede *presionar*. El argumento indica el ángulo para mirar al botón.
- **Vista del botón.** Desde ese waypoint se ve un botón al cual le se puede *disparar*. El argumento indica el ángulo para mirar al botón.

Tipos de etiquetas que puede tener una conexión en Botrix se muestran a continuación:

- **Agacharse.** Hay que agacharse para poder pasar entre los waypoints vecinos. El argumento indica cuando agacharse.
- **Salto.** Hay que saltar (ver figura 3.2). El argumento indica cuando saltar.



Figura 3.2: Bot usando conexión entre waypoints con etiqueta “salto”.

- **Romper.** Hay un objeto que molesta pasar, hay que romperlo. El argumento indica cuando disparar/golpear.
- **Salto largo.** Hay que tomar carrera para saltar. El argumento indica cuando saltar.
- **Escalera.** Hay que usar la escalera.
- **Daño.** Tomar esta conexión provoca un daño corporal. El argumento indica la cantidad de vida que se pierde. Por ejemplo se puede caer desde un lugar alto.

- **Linterna.** Un lugar oscuro. Es una simulación, ya que los bots no ven.
- **Puerta.** Hay una puerta entre 2 waypoints. Argumento es el número de puerta.

Puede ser que una conexión tenga 2 etiquetas, por ejemplo romper y agacharse. El bot al llegar al waypoint origen toma el tiempo en el cual tiene que disparar y agacharse. Seguirá corriendo hacia el waypoint destino y en el momento indicado usa el arma y al mismo tiempo se agacha para entrar a una tubería (ver la figura 3.3).

Cada waypoint además tiene asociada un área. Cada área contiene múltiples waypoints. Esto se hizo para que los bots puedan ubicarse en el mapa. Por ejemplo, un bot podría preguntar al otro donde está y el otro responderle algo como “estoy en el área del botón 8”. Es la tarea del diseñador de mapa asignar áreas a cada waypoint.



Figura 3.3: Bot rompiendo la reja para entrar a una tubería.

3.1.2. Movimiento entre waypoints adyacentes

Cuando un bot aparece en el mapa, se toma su posición y se calcula la distancia de este punto a *todos* los waypoints. El waypoint con la menor distancia se considera el actual, siempre que ésta no supere una **distancia crítica**. Vamos a llamar a esta operación **cálculo del waypoint actual**. Si la distancia supera la distancia crítica, se considera que el bot no tiene waypoint actual y esto hace que el bot no se pueda mover. El diseñador del nivel tiene que ubicar los waypoints correctamente para que esto no pase.

Cabe destacar que en un mapa con muchos waypoints, el cálculo del waypoint actual puede llegar a ser una operación bastante costosa. Por esta razón, una vez calculado el waypoint actual, para saber si hubo cambio del waypoint en los frames subsiguientes solamente se toman las distancias a los waypoints *vecinos* del actual.

Si la distancia desde la posición del bot al waypoint vecino es más chica que la distancia al waypoint actual, se cambia el waypoint actual. Cuando la distancia del bot al waypoint actual supera la distancia crítica, el bot se considera perdido y se hace cálculo completo del waypoint actual.

Por más que se cambie el waypoint actual, esto no significa que el bot llegó al waypoint destino, solamente está más cerca del waypoint destino que del waypoint actual. El bot llega al waypoint destino cuando el bot **toca** al waypoint, es decir que la distancia del bot al waypoint es más chica que la **distancia del toque**, que es una constante definida por el programador del juego. En este momento se analizan las etiquetas del waypoint y las etiquetas de la conexión al siguiente waypoint, y se toma alguna acción dependiendo de las mismas.

3.2. Búsqueda de camino

Siendo que el conjunto de los waypoints con conexiones es un grafo dirigido, podemos aplicar algoritmos de búsqueda existentes para encontrar un camino entre 2 waypoints dados. El más famoso por ser uno de los más rápidos es el **algoritmo A*** [RusNor03]. Presentado por primera vez en 1968 por Peter E. Hart, Nils J. Nilsson y Bertram Raphael, el algoritmo A* encuentra, siempre y cuando se cumplan unas determinadas condiciones, el camino de menor costo entre dos vértices.

El algoritmo A* utiliza una función de evaluación $f(n) = g(n) + h(n)$, que determina el orden en el cual se visitan los nodos del grafo. Esta función de evaluación es una estimación del camino de costo mínimo que va desde el inicio hasta el objetivo y pasa por el nodo n . $h(n)$ es la llamada *función heurística*, se define como una estimación de costo entre el nodo n y el nodo destino. $g(n)$ es el costo real del camino recorrido para llegar a dicho nodo n desde el nodo inicial, esto es, la suma de los pesos de las aristas del camino calculado hasta el momento. En nuestro caso la función heurística es la distancia euclídea (la longitud de la línea recta) desde waypoint a evaluar n al waypoint destino y $g(n)$ es la longitud del camino actual desde el waypoint origen hasta el waypoint n .

Una heurística será **admisible** cuando el costo estimado $h(n)$ sea siempre menor o igual que el costo mínimo de alcanzar el nodo destino desde n . Con una heurística **no admisible**, el algoritmo A* *podría pasar por alto la solución óptima* durante la búsqueda debido a una sobre-estimación de $f(n)$. Como $g(n)$ es el costo exacto de alcanzar n , tenemos como consecuencia inmediata que $f(n)$ nunca sobreestima el costo verdadero de la solución a través de n si la heurística es admisible. De estar $h(n)$ definida por una heurística admisibles, el algoritmo de búsqueda A* devolverá una *solución óptima*. Claramente, la distancia euclídea es una función admisibles, ya que no existe un camino más corto entre dos puntos que una línea recta, por lo que A* siempre nos devolverá el camino óptimo, esto es, el más corto.

A* mantiene 3 estructuras de datos auxiliares:

- *openset*, una cola de prioridad de nodos a visitar (ordenada por el valor $f(n)$ de cada nodo).
- *closedset*, un conjunto donde se guardan los nodos que ya han sido visitados.
- *path*, un mapa (o función asociativa) que para cada nodo devuelve el nodo predecesor. Sirve para recuperar el camino de menor costo.

En cada paso del algoritmo, se expande el nodo que esté primero en el *openset*, y en caso de que no sea un nodo visitado, calcula la $f(n)$ de todos sus vecinos, los inserta en el *openset*, y agrega el nodo evaluado *closedset*. Mientras A* atraviesa el grafo, sigue un camino de costo total más bajo, manteniendo la cola de prioridad ordenada y alternando los segmentos del camino a lo largo del mismo.

Nosotros vamos a ver el algoritmo de A* modificado que separa la búsqueda en frames, para no sobrecargar al microprocesador. *MAX_NODES* aquí indica la cantidad máxima de waypoints a procesar en cada frame. *start* es el nodo origen, *goal* es el nodo destino y *reiniciar* es una variable booleana que indica si la búsqueda recién empieza o es una búsqueda nueva. El algoritmo en pseudo-código se da a continuación:

```
// Calcula el camino con mejor costo desde origen a destino.
// Devuelve secuencia de nodos.
función A*(origen, destino, reiniciar)
    procesados := 0
    si reiniciar
        reiniciar := falso // Para que no se reinicie en el frame siguiente.
        closedset := conjunto vacío // El conjunto de nodos visitados.
        openset := start // El conjunto de nodos a evaluar.
        path := mapa vacío // Mapa de nodos visitados.

        g[origen] := 0 // El costo desde start hasta el nodo actual.
        // Costo total estimado desde origen a destino a través del nodo dado.
        f[origen] := g[origen] + heuristica(origen, destino)

    mientras (openset no está vacío) ^ (procesados < MAX_NODES)
        actual := el nodo en openset que tiene f[] más bajo
        si actual = goal
            reiniciar := verdadero // Encontramos el camino, próxima búsqueda se reiniciará.
            retornar reconstruir_camino(path, goal)

        remover actual del openset
        agregar actual a closedset

    para cada vecino en vecinos(actual)
        g_tentativo := g[actual] + distancia(actual, vecino)
        f_tentativo := g_tentativo + heuristica(vecino, goal)
        si (vecino está en closedset) ^ (f_tentativo >= f[vecino])
            continuar

        si (vecino no está en openset) v (f_tentativo < f[vecino])
            path[vecino] := actual // Nodo previo del vecino es el actual.
            g[vecino] := g_tentativo
            f[vecino] := f_tentativo
            si vecino no está en openset
```

```
        agregar vecino a openset

si openset está vacío
    retornar ''error''
sino
    retornar ''continua en frame siguiente''

// Reconstruye el camino hasta nodo, usando mapa de nodos previos.
// Devuelve secuencia de nodos.
función reconstruir_camino(mapa, nodo)
    si nodo está en mapa
        previo = mapa[nodo]
        p := reconstruir_camino(mapa, previo)
        retornar (p + nodo)
    sino
        retornar nodo
```

En el capítulo que sigue veremos una técnica de reconocer patrones de texto para que los bots puedan interactuar entre sí o con un humano.

Capítulo 4

Chateando con bots

En este capítulo se describirá como hacen los bots para entender las oraciones que reciben por medio de chat. Primero vamos a ver la terminología básica del área y describimos a las variables de la oración. Luego mostraremos el pre-procesamiento de la oración, algoritmo aplicado y ejemplos de uso.

4.1. Introducción

El *lenguaje* es una de las herramientas que los seres humanos usamos para comunicar o intercambiar pensamientos, creencias, emociones, etc. Un **acto de habla** o **acto verbal** (speech act en inglés) es la realización de un determinado tipo de acción (informar, saludar, invitar, aconsejar, disculparse, etc.) a través del lenguaje. En la interacción cotidiana afirmamos, aseguramos, negamos (actos asertivos); o pedimos, suplicamos, damos órdenes (actos directivos); o expresamos diferentes emociones (actos expresivos). Cada una de estas acciones es un acto de habla.

Austin [Aus62] afirma que todo enunciado tiene carácter de acción, y diferencia distintos tipos de actos:

- **Acto locutivo** es el que se lleva a cabo al emitir una cadena de sonidos ligada a un significado de acuerdo con las reglas de una lengua.
- **Acto ilocutivo** es el que se realiza cuando se dice algo (informar, sugerir, solicitar, etc) y el hablante lo realiza con una intención concreta que se llama fuerza ilocutiva.
- **Acto perlocutivo** es el efecto que se causa en el destinatario del mensaje (convencer, divertir, asustar, informar).

Searle [Sea69], además, propone una clasificación de estos actos en las siguientes cinco clases. Esta clasificación esta basada en la *intención* del acto de habla:

- **Representativos o asertivos:** el hablante se comprometa que un comentario se refiere a la realidad y que es un hecho. Por ejemplo: afirmar, negar, confesar, admitir, notificar etc.
- **Directivos:** intentan obligar al oyente hacer una cosa. Solicitar, requerir, ordenar, prohibir, aconsejar etc.
- **Compromisorios:** obligan al hablante hacer una cosa. Prometer, jurar, ofrecerse, garantizar etc.
- **Expresivos:** expresan el estado de ánimo del hablante. Agradecer, felicitar, condolerse, dar la bienvenida, disculparse etc.
- **Declaratorios:** cambian el estado de alguna cosa. Nombrar, bautizar, rendirse, excomulgar, acusar etc.

En lo que sigue describiremos como los bots usan actos de habla para poder colaborar en objetivos comunes.

La **oración** es el constituyente sintáctico más pequeño, capaz de realizar un enunciado o expresar el contenido de un acto de habla. Algunos ejemplos: “El botón 3 abre puerta 5” es un acto asertivo (una aserción), “¿Podrías abrir la puerta?” es un acto directivo (petición de acción), “¿La puerta está abierta?” también es un acto directivo (petición de información). En este trabajo modelaremos *actos asertivos* y *actos directivos*.

La **semántica lingüística** es un subcampo de la semántica general y de la lingüística que estudia la codificación del significado dentro de las expresiones lingüísticas, es decir de las oraciones. Viene del griego “semántikos”, que quiere decir “significado relevante”.

El **chat** es una conversación escrita que se realiza entre varias personas, aunque también se puede tener conversaciones con un bot, por ejemplo A.L.I.C.E. [Alice04] es un bot de chat. A.L.I.C.E. fue finalista de la competencia internacional “Loebner Prize” en los años 2000, 2001 y 2004. El formato de la competición sigue el estándar establecido en el test de Turing, que consiste en que una persona no pueda diferenciar si está hablando con otra persona o con un chatbot. En nuestro caso queremos que el jugador humano converse con los bots durante el juego, o que los bots conversen entre ellos. La conversación se basa en una secuencia de oraciones cada una de las cuales tiene un significado (semántica). En esta tesis nos limitaremos a conversaciones de 2 integrantes, es decir bot-bot o persona-bot.

Cada bot tiene sus propios *objetivos* y *creencias*, estas últimas pueden ser verdaderas o falsas. Para que una persona cambie sus creencias, puede pasar una de dos cosas: esta persona vió algo que hizo cambiar sus creencias o alguien le dijo a esta persona que sus creencias son falsas y la persona le creyó. También puede pasar que una persona cambie su objetivo solamente porque alguien le pidió ayuda y esta persona deja de hacer lo que estaba haciendo y ayuda a ese alguien (cambió de objetivo). Nosotros vamos a emular este comportamiento de

los humanos, restringido al dominio del juego, para que los bots puedan colaborar para alcanzar un objetivo común. Recordemos que muchas veces un solo jugador no podrá realizar el objetivo completo solo, necesitando la colaboración de otro jugador. De esta forma un jugador humano o bot podrían solicitar mediante chat la colaboración de otro bot o jugador humano.

Por esta razón es importante que los bots puedan interpretar la *semántica de las oraciones*. A través de chat ellos pueden cambiar sus creencias/objetivos, pero sólo si pueden entender la oración. Los bots solamente van a entender un conjunto predefinido de actos de habla. Nuestro sistema tiene asociaciones de conjuntos de oraciones a sus respectivas semánticas o significados. Por ejemplo la oración “«nombre del bot», I am busy right now” tiene asociado el significado “ocupado”, y el efecto en el sistema es que bot con nombre «nombre del bot» anota al estado del jugador que habló como “ocupado”.

Las personas no siempre dicen las cosas de la misma forma, es decir son no deterministas. Por ejemplo, para indicar que otra persona se acerque, uno puede decir “Vení aca”, “Podrías acercarte?”, “Acércate, por favor”, etc. Los *sinónimos*, *detección de errores de sintaxis* y cambio de *orden de las palabras* son importantes para interpretar la semántica de la oración. En este trabajo se toma en cuenta un conjunto de sinónimos y el orden de las palabras, pero no hay detección de errores de sintaxis. Es decir si uno dice “open” en vez de “opens” el sistema dará un error. Si bien es un enfoque simple, y el formato es bastante estricto en cuanto a las oraciones aceptadas, en el capítulo 7 veremos que el sistema funciona bastante bien para su propósito, esto es, adquisición de información y armado de peticiones.

La conversación la puede iniciar tanto una persona como un bot. Para iniciar la conversación con un bot hay que saludarlo. Una vez recibido el saludo, el bot supone que el jugador es colaborativo y usará a este jugador en las acciones de planificador. Veremos esto más en profundidad en el capítulo 6.

4.2. Variables de la oración

El sistema filtra signos de puntuación, reemplaza sinónimos e identifica valores de **variables de la oración**. Es más fácil explicar que son las variables de la oración con un ejemplo. En la oración “I can see a \$door_status door \$door” que tiene semántica “ver puerta” hay dos variables: \$door y \$door_status. \$door es un número y \$door_status puede ser “opened” o “closed”. Cuando el sistema procesa la oración “I can see a closed door 4” devuelve la semántica “ver puerta” con dos variables: \$door = 4 y \$door_status = closed.

Cada variable de la oración tiene un conjunto finito de **valores** que la variable puede tomar. Las variables de la oración de Botrix y descripción de sus posibles valores se muestran a continuación:

- \$player representa el nombre de un jugador. Posibles valores son nombres de jugadores que están conectados actualmente. Se usa para detectar a quien está dirigida la oración.

- *\$area* representa nombre de un área. Puede tomar valor de cualquier área del mapa jugado (ver capítulo 3).
- *\$button* - número de un botón. Los valores posibles son números de 1 a *#botones* en el mapa.
- *\$door* - número de una puerta. Los valores posibles son números de 1 a *#puertas* en el mapa.
- *\$door_status* - estado de una puerta. Puede ser “opened” o “closed”.
- *\$weapon* - nombre de un arma. Los nombres de armas están definidos en el archivo de configuración del plugin (ver apéndice .A). Se le elimina el prefijo “weapon_”, por ejemplo “weapon_physcannon”, arma de gravedad, se transforma en *physcannon*. Un chat de ejemplo: “I just found weapon *physcannon*”.
- *\$box* - número de una caja. Puede tomar valores de 1 a *#cajas* en el mapa.

4.3. Matching de oraciones

Para cada semántica admitida (que corresponde a una aserción o petición de acción) nuestro sistema tiene un conjunto de oraciones que la representan. Cada oración es una secuencia de palabras, cada una de ellas puede ser **requerida** u **opcional**. Entre las palabras obligatorias están los verbos, que dirigen la sintaxis y la semántica de la oración. También permitimos reórdenes de palabras en la oración. Se puede ver nuestro sistema como n-gramas de orden de 1 a 4 con penalización en caso de que las palabras se reordenan.

En la oración de entrada se reemplazan los sinónimos y luego el sistema compara la oración de entrada con cada una de las oraciones del sistema. A cada par $\langle \text{oración de entrada}, \text{oración del sistema} \rangle$ le da un puntaje que indica la medida de similitud entre las oraciones. El sistema dará como salida la oración con el puntaje máximo, la semántica respectiva y valores de variables en la oración. El puntaje se calcula teniendo en cuenta las proporciones de:

- Palabras requeridas.
- Palabras opcionales.
- Palabras bien ordenadas.
- Palabras sobrantes (las que están en la oración de entrada pero no en la oración del sistema).

4.3.1. El algoritmo

En lo que sigue se muestra el algoritmo usado:

```

// Toma como entrada oración O, que es una secuencia de palabras.
// Devuelve la semántica y el puntaje máximo.
función dar_semantica(Oi)

    Puntaje_Maximo := 0
    Semantica_Maxima := desconocida
    Valores_Maximos := mapa vacío // Valores de variables.

    para cada semántica posible sem
        para cada oración Os dentro del conjunto Oraciones(Sem) // Oraciones del sistema.
            valores_variables := mapa vacío // Valores de variables para esta oración.
            requeridas := 0 // Cantidad de palabras requeridas.
            opcionales := 0 // Cantidad de palabras opcionales.
            ordenadas := 0 // Cantidad de palabras ordenadas.

            para cada palabra P de Os
                si P empieza con '$'
                    para cada V en el conjunto valores_de_variable(P)
                        si V está en Oi
                            valores_variables(P) := V
                            si P es una palabra opcional
                                opcionales := opcionales + 1
                            sino
                                requeridas := requeridas + 1
                            si P está bien ordenada en Oi
                                ordenadas := ordenadas + 1
                        sino
                            si P está en Oi
                                si P es una palabra opcional
                                    opcionales := opcionales + 1
                                sino
                                    requeridas := requeridas + 1
                                si P está bien ordenada en Oi
                                    ordenadas := ordenadas + 1

            N_requeridas := #palabras requeridas de Os
            N_opcionales := #palabras opcionales de Os
            N_extra := #Oi - (opcionales + requeridas) // Palabras que están en Oi pero no en Os.

            req := requeridas / N_requeridas // Proporción de palabras requeridas.
            si N_opcionales == 0
                opt := 1 // Sumar 1 si no hay palabras opcionales.
            sino
                opt := opcionales / N_opcionales // Proporción de palabras opcionales.
            ord := ordenadas / #Os // Proporción de palabras bien ordenadas.
            extra := N_extra / #Oi // Proporción de palabras que no dieron matching.

            puntaje := 6*req + 1*opt + 3*ord - 4*extra

            si puntaje > Puntaje_Maximo
                Puntaje_Maximo := puntaje
                Semantica_Maxima := sem
                Valores_Maximos = valores_variables

    retornar (Semantica_Maxima, Puntaje_Maximo, Valores_Maximos).

```

Como se ve en el algoritmo, cada proporción es un número entre 0 y 1. Cuando *req*, *opt* y *ord* son iguales a 1 y *extra* es igual a 0, esto nos da el *puntaje* máximo 10. De la misma forma el *puntaje mínimo* es -4, cuando ninguna palabra coincide. Los valores 6, 1, 3 y 4 en la multiplicación de las proporciones representan la “importancia” de cada proporción. Es decir, nos importa más la proporción de palabras requeridas (obligatorias) que la de opcionales, el orden importa más que las palabras opcionales, pero palabras sin coincidir tienen más importancia que el orden de las palabras. El signo positivo representa aumento de puntaje, mientras que el negativo resta: es intuitivo, teniendo muchas palabras obligatorias aumenta el puntaje, mientras que teniendo muchas palabras sin matching debería bajar el puntaje. Estos valores de multiplicación fueron elegidos experimentalmente.

Notar que si hay dos oraciones con el mismo puntaje, se devuelve el significado de la primera oración, sin embargo, es bastante improbable que dos oraciones distintas den el mismo puntaje si ellas no tienen el mismo significado.

Puede darse que el puntaje de matching de la oración de entrada sea menor que el mínimo aceptado por el sistema (es 6.0 en Botrix). En este caso el bot puede responder algo como “Sorry, what did you say?” dando a entender que no logró obtener el significado de la sentencia anterior.

Todas las oraciones del sistema de chat están en el archivo de configuración del plugin y se describen en el apéndice .A.

4.3.2. Ejemplos

Vamos a ver algunos ejemplos de matching. La oración usada en el chat se da al principio y luego explicamos que match dió el sistema, si es que hubo. En la oración del sistema las palabras opcionales están entre paréntesis y las variables empiezan con el símbolo '\$’.

- “let’s try to push button 4” dió match “let’s try (to push) button \$button” con puntaje máximo 10, ya que en la oración están todas las palabras requeridas y opcionales y además 4 es un valor válido para variable \$button de la oración.
- “let’s try button 4” dió match “let’s try (pushing) button \$button” con puntaje 9. La única palabra que faltó es la palabra opcional *pushing*, esto hace que la variable *opt* del algoritmo tome valor 0. *req* y *ord* valen 1 ya que todas las palabras requeridas están presentes y las palabras en la frase están bien ordenadas. *extra* vale 0, porque no hay palabras sobrantes.
- “go to the area area-respawn1 please” dió match “\$player, (please,) go to area \$area” con puntaje 7.63. *extra* vale 1/6 (hay 1 palabra extra de 6 palabras del sistema), *req* tiene valor 4/5 ya que falta la variable \$player en la oración, *opt* es 1 y *ord* es 5/6 (*please* está mal ubicada en la oración). Puntaje entonces es $6 * 4/5 + 1 + 3 * 5/6 - 4 * 1/6 = 7,63$. Variable de la oración \$area toma valor area-respawn1.

- Para la oración “I can see weapon crossbow” el mejor match es “I (just) found weapon \$weapon” con puntaje 5.9. Aquí *req* vale $3/4$ porque falta la palabra *found*, *opt* vale 0 (falta *just* en la oración), *ord* es 1, porque de las palabras que hay, todas están bien ordenadas y *extra* vale $2/5$. El puntaje entonces es $6 * 3/4 + 0 + 3 * 1 - 4 * 2/5 = 5,9$. De esta forma el sistema rechaza la oración de entrada (porque $5,9 < 6$) y el bot no entenderá que le están diciendo.
- Para la oración “there is a button 8” el mejor match es “I (have) found (a) button \$button”. El puntaje nos da $6 * 2/4 + 1 * 1/2 + 3 * 1 - 4 * 2/5 = 4,9$. El sistema rechaza la oración.
- Para la oración “let’s see what do the button 4” el mejor match es “let’s try going to (the) button \$button”. El puntaje nos da $6 * 3/6 + 1 + 3 * 1 - 4 * 3/7 = 5,28$. El sistema rechaza la oración.
- Un ejemplo interesante: “button 4 doesn not opens door 1” dió match “(so,) button \$button opens door \$door”! El puntaje es $6 * 5/5 + 0 + 3 * 1 - 4 * 2/7 = 7,85$, por lo que el sistema acepta la oración, pero la semántica que le da el sistema es completamente distinta a la que intentó dar el usuario! Por esta razón es importante no cometer errores de sintaxis ya que nuestro sistema no los detecta, sin embargo, aquí tenemos 2: *doesn* en vez de *does* y *opens* en vez de *open*.

El chat sirve para que un jugador pueda lograr un objetivo usando ayuda de otro jugador. Sin embargo, ¿cómo sabe un bot que tiene que usar ayuda de otro jugador? ¿Cómo sabe que tiene que decir algo para estar más cerca del objetivo? En el capítulo 6 se describe como se usa la planificación y su relación con el chat, pero antes de esto hay que entender que es la planificación: esto se da en el capítulo siguiente.

Capítulo 5

Planificación, PDDL y FF

En este capítulo definiremos planificación, sintaxis del lenguaje STRIPS (sección 5.1) y su extensión ADL (sección 5.2). En la sección 5.3 se mostrarán algunos algoritmos de búsqueda de plan en el espacio de estados: búsquedas hacia adelante y hacia atrás, además veremos heurísticas para las mismas. Hablaremos brevemente sobre planes de orden parcial en la sección 5.4. Veremos grafos de planificación en la sección 5.5 y el algoritmo GRAPHPLAN en la sección 5.5.2. Finalmente en la sección 5.6 explicaremos como funciona el planificador FF, el cual se usó en los experimentos realizados.

5.1. Planificación y el lenguaje STRIPS

Llamaremos **planificación** (inglés *planning*) al proceso de búsqueda de una secuencia de acciones que permitan alcanzar un objetivo desde un estado inicial. Para representar un problema, necesitamos representación para *estados*, *acciones* y *objetivos*. Usaremos el lenguaje *STRIPS* [RusNor03], el cual se usó mucho en representaciones de problemas clásicos.

Representación de **estados**: usaremos *literales proposicionales* y *literales de primer-orden*, por ejemplo *Bot1* y *En(Bot1, Waypoint1)*. Los literales deben ser simples y sin dependencias funcionales, por ejemplo *En(x, y)* o *En(Ayudante(Bot1), Waypoint2)* no se permiten. Se asume la hipótesis de un mundo cerrado, por eso es que todas las condiciones no mencionadas en un estado son falsas.

Representación de **objetivos**: un objetivo es un estado parcialmente especificado. Un estado proposicional *S* **satisface** un objetivo *G* si *G* está incluido en *S* ($G \subseteq S$). Por ejemplo, el estado $En(Bot1, Waypoint1) \wedge Tiene(Bot1, ArmaDeGravedad)$ satisface el objetivo *En(Bot1, Waypoint1)*.

Representación de **acciones**: una acción se representa mediante un *esquema de acción* que consta de tres partes:

- El *nombre* de la acción y la lista de *parámetros* de los que depende la acción (por ejemplo, *Moverse(bot, desde, hasta)*).

- La *precondición*: es la unión de literales positivos sin dependencia funcional estableciendo lo que debe ser verdad en un estado antes de que una acción sea ejecutada. Todas las variables en las precondiciones también deben aparecer en la lista de parámetros de acción.
- El *efecto*: es la unión de literales sin dependencia funcional describiendo cómo el estado cambia cuando la acción es ejecutada. Un literal positivo P que aparece en el efecto será verdadero en el estado resultante de la acción, mientras que un literal negativo $\neg P$ será falso. Las variables en el efecto también deben pertenecer a la lista de parámetros de acción.

Por ejemplo:

```

Acción Moverse (
  Parámetros: bot, desde, hasta
  Precondición: Waypoint(desde)  $\wedge$  Waypoint(hasta)  $\wedge$ 
                Bot(bot)  $\wedge$  En(bot, desde)
  Efecto:  $\neg$  En(bot, desde)  $\wedge$  En(bot, hasta)
)

```

es una acción que representa a un bot que se mueve de un waypoint hasta otro.

Diremos que una acción es **aplicable** en cualquier estado que satisface sus precondiciones; en otro caso, la acción no podrá ser aplicada. La *instanciación*, para un esquema de acción de primer orden, se reduce a la sustitución de las variables en la precondición y efecto.

Por ejemplo, supongamos que el estado actual está descrito por:

$$Bot(Bot1) \wedge Bot(Bot2) \wedge Waypoint(Waypoint1) \wedge Waypoint(Waypoint2) \wedge \\ En(Bot1, Waypoint1) \wedge En(Bot2, Waypoint2)$$

Este estado satisface la precondición de Moverse con la sustitución bot/Bot1, desde/Waypoint1, hasta/Waypoint2. De esta forma, la acción concreta Moverse(Bot1, Waypoint1, Waypoint2) es aplicable.

Comenzando en el estado S , el resultado de ejecutar una acción aplicable A nos lleva a otro estado S' , que es el mismo que S excepto que cualquier literal positivo P en el efecto de A es añadido a S' , y cualquier literal negativo $\neg P$ es eliminado de S' . Siguiendo con el ejemplo, después de $Moverse(Bot1, Waypoint1, Waypoint2)$, el estado actual se convierte en:

$$En(Bot1, Waypoint2) \wedge En(Bot2, Waypoint2) \wedge Bot(P1) \wedge \\ Bot(P2) \wedge Waypoint(Waypoint1) \wedge Waypoint(Waypoint2)$$

Notemos que si un efecto positivo ya está en S , no se añade dos veces, y si un efecto negativo no está en S , entonces esa parte del efecto es ignorada.

La **solución** de un problema de planificación, es una *secuencia de acciones* que, ejecutada en el estado inicial, da como resultado un estado final que satisface el objetivo.

5.2. Extensión de STRIPS: ADL

STRIPS tiene ciertas restricciones para poder diseñar algoritmos de búsqueda más simples y más eficientes. Todas estas restricciones hacen que STRIPS sea un lenguaje simple, y en los últimos años se ha demostrado que de hecho carece de expresividad para ciertos dominios. Por esta razón se desarrolló el Lenguaje de Descripción de Acciones (Action Description Language, ADL). En ADL, la acción *Moverse* podría ser escrita como:

```

Acción Moverse (
  Parámetros: (bot, desde, hasta),
  Precondición: Waypoint(desde) ∧ Waypoint(hasta) ∧
                Bot(bot) ∧ En(bot, desde) ∧ (desde ≠ hasta)
  Efecto: ¬En(b, desde) ∧ En(b, hasta)
)

```

En la figura 5.1 se da la tabla de diferencias entre STRIPS y ADL.

STRIPS	ADL
Sólo literales positivos en estados: $Pobre \wedge Desconocido$	Literales positivos y negativos en estados: $Rico \wedge \neg Famoso$
Hipótesis de Mundo Cerrado: los literales no mencionados son falsos	Hipótesis de Mundo Abierto: Los literales no mencionados son desconocidos
El efecto de $P \wedge \neg Q$ significa añadir P y eliminar Q	El efecto de $P \wedge \neg Q$ significa añadir P y $\neg Q$ y eliminar $\neg P$ y Q
Sólo literales simples en objetivos: $Rico \wedge Famoso$	Variables cuantificadas en objetivos: $\exists x \bullet En(P1, x) \wedge En(P2, x)$ es el objetivo de tener P1 y P2 en el mismo lugar
Los objetivos son conjunciones: $Rico \wedge Famoso$	Se permiten conjunciones y disyunciones en los objetivos: $\neg Pobre \wedge (Famoso \vee Inteligente)$
Los efectos son conjunciones	Se permiten efectos condicionales: $when P : E,$ significa que E es un efecto sólo si P es satisfecho
No permite igualdades o desigualdades	Tiene igualdades o desigualdades
Lenguaje no tipado	Las variables pueden tener tipos, por ejemplo ($b : Bot$)

Figura 5.1: Diferencias entre STRIPS y ADL.

5.3. Búsqueda de plan en el espacio de estados

El enfoque más simple y más intuitivo que llega a la mente al pensar en un problema de planificación es la búsqueda en el espacio de estados del problema dado. Ya que las acciones tienen tanto precondiciones como efectos, las búsquedas pueden ser tanto hacia-adelante en el espacio de estados partiendo desde el estado inicial, como hacia-atrás partiendo desde el estado final. Ya conocemos un algoritmo de búsqueda: A* descrito en el capítulo 3. Sabiendo que necesita de una

función heurística, veremos algunas formas automáticas de obtener la misma en la sección 5.3.3.

5.3.1. Planificación con búsqueda hacia adelante en el espacio de estados

También se le llama planificación de **progresión**, porque mantiene una dirección de avance. Comenzamos en el estado inicial del problema, considerando secuencias de acciones hasta que encontremos una secuencia que alcance un estado objetivo. En ausencia de dependencias funcionales, el espacio de estados de un problema de planificación es finito.

Se puede ver que es ineficaz en la práctica:

- Permite uso de acciones **irrelevantes**: todas las posibles acciones aplicables son consideradas desde el estado actual. Una acción es relevante para una secuencia encadenada de objetivos si alcanza un conjunto de ellos.
- En la práctica el árbol de búsqueda no se genera todo, ya que puede ser muy grande, sino se genera una parte del mismo. Sin una buena **heurística** la parte generada del árbol de búsqueda se hace muy grande. Veremos el uso de heurística en la sección 5.3.3.

5.3.2. Planificación con búsqueda hacia atrás en el espacio de estados

Comenzamos en el estado *objetivo* del problema y generamos estados *predecesores* hasta encontrar el estado inicial. Tiene la ventaja que solamente considera acciones *relevantes*, aunque a veces no siempre es obvio cómo generar estados predecesores. Además de insistir en que las acciones alcancen algunos literales deseados, hay que insistir en que las acciones no deshagan ningún literal deseado. Una acción que satisfaga dicha restricción es llamada **consistente**.

El proceso de búsqueda entonces será el siguiente. Sea G la descripción del objetivo y supongamos que A es una acción que es relevante y consistente. El correspondiente predecesor se calcula de la siguiente forma:

- Cualquier efecto positivo de A que aparezca en G es eliminado.
- Cada precondition literal de A es añadida, a no ser que ya apareciese.

Cabe destacar que usando sólo acciones consistentes no es posible resolver gran parte de los problemas de planificación, en las que hace falta deshacer literales deseados en algún punto del plan.

5.3.3. Función heurística para la búsqueda

Cualquier algoritmo de búsqueda en grafos que es completo (por ejemplo, A*) será un algoritmo de planificación completo, ya sea hacia-adelante o hacia-atrás. Algunos de ellos requieren función *heurística* adecuada, para encontrar más rápido la solución. Recordemos que una función heurística estima la distancia desde un estado cualquiera al estado objetivo; en la planificación STRIPS, el coste de cada acción es 1, de modo que la distancia es el número de acciones. La idea básica es observar los efectos de las acciones y los objetivos que deben ser alcanzados y estimar cuántas acciones son necesarias para alcanzar todos los objetivos. Encontrar el número exacto es un problema NP-completo, pero es posible encontrar razonables estimaciones en muchos casos sin demasiado gasto computacional. De hecho, el planificador FF (usado en los experimentos de este trabajo) usa una heurística que no es admisible, y vamos a verla en detalle en la sección 5.6.2.

Hay dos enfoques para mencionar:

- Generar **problema aproximado**. El costo de la solución óptima del problema aproximado (que suponemos más sencillo de resolver que el original) nos da una heurística admisible para el problema original. A continuación vamos a ver las ideas más usadas:
 - ◊ La *eliminación* de todas las *precondiciones* de las acciones. En esta situación, todas las acciones serán aplicables, y cualquier literal puede ser alcanzado en cada etapa. Hay que tener en cuenta dos cosas: a) que una acción puede alcanzar muchos objetivos y b) puede haber dos acciones, una de las cuales elimine el objetivo alcanzado por la otra.
 - ◊ La *eliminación de precondiciones y de efectos negativos*. Después, se cuenta el mínimo número de acciones requeridas tales que la unión de los efectos de acciones positivas satisfagan el objetivo. Lamentablemente, este algoritmo calcula heurística que no es admisible, es decir, puede sobreestimar los costos para llegar al objetivo.
 - ◊ La *eliminación de efectos negativos sin eliminar precondiciones*. El costo de la solución del problema resultante aproximado da lo que se conoce como **heurística para suprimir listas vacías**. La heurística es bastante precisa, pero ponerla a funcionar lleva asociado ejecutar un algoritmo de planificación simple. Aunque, en la práctica, puede ser suficientemente rápida como para que el costo valga la pena. Es la más usada actualmente, por ejemplo la usa el planificador FF (vea sección 5.6.2).
- Asumir la *independencia de sub-objetivos*. Entonces, el costo de resolver un conjunto de sub-objetivos sería la suma de los costos para resolver cada sub-objetivo. La hipótesis de la independencia de sub-objetivos puede ser *optimista* o *pesimista*. Se llama **optimista** cuando existen interacciones negativas entre los subplanes de cada sub-objetivo, por ejemplo, cuando una acción en un sub-plan hace fracasar el objetivo a alcanzar por otro sub-plan.

La heurística es pesimista, y por tanto inadmisibile, cuando los sub-planes contienen acciones redundantes, por ejemplo, dos acciones que podrían ser reemplazadas por una acción sencilla en un plan conjunto.

5.4. Planificación de orden parcial

Las búsquedas en el espacio de estados hacia-adelante y hacia-atrás son tipos de planes de búsqueda totalmente ordenados. Sólo exploran secuencias estrictamente lineales de acciones conectadas directamente al inicio o al objetivo. Esto significa que estas búsquedas no pueden sacar provecho de la descomposición del problema. En cambio, la **planificación de orden parcial** (POP) es un enfoque que trabaja en varios sub-objetivos independientemente, los soluciona con varios sub-planes, y por último, combina el conjunto de sub-planes utilizados.

Un ejemplo:

Inicio:

$(\text{En}(\text{Bot1}, \text{Area1}) \wedge \text{En}(\text{Bot2}, \text{Area1}) \wedge$
 $\text{Adyacente}(\text{Area1}, \text{Area2}) \wedge \text{Adyacente}(\text{Area2}, \text{Area3}))$

Objetivo:

$(\text{En}(\text{Bot1}, \text{Area3}) \wedge \text{En}(\text{Bot2}, \text{Area3}))$

Acción Moverse (

Argumentos: bot - Bot, desde, hasta - Area

Precondición: $\text{En}(\text{bot}, \text{desde}) \wedge \text{Adyacente}(\text{desde}, \text{hasta})$

Efecto: $\neg \text{En}(\text{bot}, \text{desde}) \wedge \text{En}(\text{bot}, \text{hasta})$

)

El plan de orden parcial para el problema dado es el de la figura 5.2.

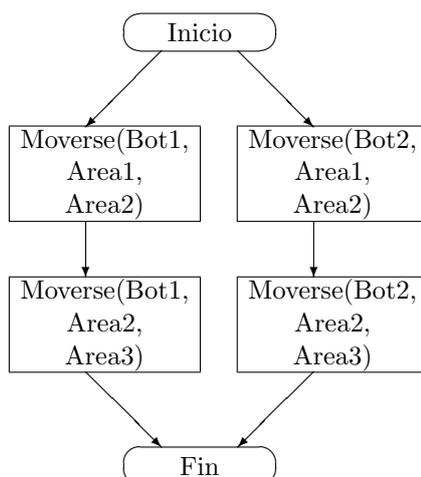


Figura 5.2: Plan de orden parcial para el problema de llevar a 2 bots al área 3.

Destaquemos que la solución es representada como un grafo de acciones y no como una secuencia. La solución de orden-parcial se corresponde con seis posibles

planes de orden total, cada uno de ellos se conoce como una **linealización** del plan de orden parcial. Por ejemplo, Inicio - Move(Bot1, Area1, Area2) - Move(Bot1, Area2, Area3) - Move(Bot2, Area1, Area2) - Move(Bot2, Area2, Area3) - Fin, es una linealización del plan.

A pesar del auge inicial de las técnicas POP, la investigación de orden parcial se abandonó parcialmente por ineficiencia.

5.5. Grafos de planificación

Todas las heurísticas que hemos sugerido para planificación pueden sufrir imprecisiones. Veremos como una estructura especial llamada **grafo de planificación** puede ser usada para dar mejores estimaciones heurísticas. Estas heurísticas pueden ser aplicadas a cualquiera de las técnicas de búsqueda que hemos visto hasta ahora. Alternativamente, podemos extraer solución directamente del grafo de planificación, usando un algoritmo especializado llamado **GRAPHPLAN**.

El grafo de planificación contiene dos tipos de nodos: *nodos de proposición* y *nodos de acción*, organizados en niveles. Los niveles con numeración par contienen nodos de proposición (es decir, literales) y, en particular, el nivel cero consiste precisamente de las proposiciones que son verdaderas en el estado inicial del problema de planificación. Los nodos presentes en niveles con numeración impar corresponden a instancias de acción, cuyas precondiciones (que son mutuamente consistentes) están en el nivel previo.

Los nodos correspondientes a instancias de acciones están conectados mediante *arcos* a los nodos de proposición (en el nivel anterior) que constituyen las *precondiciones* de la acción. Existen arcos adicionales (*arcos de postcondición*) que conectan los nodos de acción con los nodos de proposiciones (en el nivel siguiente) que se hacen verdaderos por efecto de la acción. Es importante destacar que el grafo de planificación representa acciones que pueden desarrollarse en paralelo en cada nivel de acciones. Sin embargo, el hecho que dos acciones estén presentes en el mismo nivel de un grafo de planificación no necesariamente significa que sea posible ejecutar ambas a la vez.

En grafos de planificación es importante la relación binaria de exclusión mutua, denominada **mutex**, entre nodos presentes en el mismo nivel del grafo de planificación. La relación se define recursivamente como sigue:

- Dos instancias de acción en el nivel i son mutex si ocurre alguno de los casos siguientes:
 1. El efecto de una acción es la negación del efecto de la otra (efectos inconsistentes).
 2. El efecto de una acción elimina la precondición de la otra (interferencia).
 3. Las acciones tienen precondiciones que son mutuamente exclusivas a nivel $i - 1$ (necesidades conflictivas).

- Dos proposiciones en el nivel i son mutex si:
 1. Una es la negación de la otra.
 2. Todas las formas de llegar a estas proposiciones (es decir, las acciones en el nivel $i - 1$) son mutex tomadas de a dos (soporte inconsistente).

El grafo de planificación se construye de la siguiente manera:

- Todas las condiciones iniciales son ubicadas en el primer nivel de proposiciones (nivel cero) del grafo.
- Construir un nivel genérico de acciones consiste en lo siguiente: para cada acción y para cada forma de instanciar las precondiciones de esta acción a proposiciones del nivel previo, se inserta un nodo de acción si no existen dos precondiciones que sean mutuamente exclusivas. Además, se insertan todas las acciones de persistencia (acciones nulas o no-op actions, en forma de nodo nulo, que tienen la misma postcondición que la precondición) para cada una de las precondiciones en el nivel previo. Luego, se chequea la relación de exclusión mutua entre los nodos de acción y se crea una lista, que mantiene estas relaciones, para cada acción (en el grafo se representa como un enlace desde una acción a otra). Los nodos de acciones nulas son mutuamente excluyentes para cualquiera de las acciones no nulas que haya.
- Para crear un nivel de proposiciones, simplemente se tienen en cuenta todos los efectos de las acciones presentes en el nivel previo (incluyendo las acciones de mantenimiento, nulas) y se los ubica en el siguiente nivel como proposiciones, conectándolos a través de los apropiados arcos de postcondición. Finalmente, dos proposiciones son marcadas como mutuamente exclusivas, si todas las formas de generar la primera son mutuamente exclusivas con respecto a todas las formas de generar la segunda de las proposiciones marcadas.
- Continuaremos de este modo, alternando entre el nivel de proposiciones S_i y el nivel de acción A_i , hasta que alcancemos un nivel donde dos niveles consecutivos sean idénticos. En esta situación, decimos que el grafo está **estabilizado**.

Por ejemplo, para el problema de tener y comer un pastel:

```

Inicio : (Tener(Pastel)  $\wedge$   $\neg$ Comido(Pastel))
Objetivo : (Tener(Pastel)  $\wedge$  Comido(Pastel))

Acción Comer (
  Argumentos : Pastel
  Precondición : Tener(Pastel)
  Efecto :  $\neg$ Tener(Pastel)  $\wedge$  Comido(Pastel)
)
```

```

Acción (Cocinar
  Argumentos: Pastel
  Precondición: ¬Tener(Pastel)
  Efecto: Tener(Pastel)
)

```

el grafo de planificación es el que se muestra en la figura 5.3.

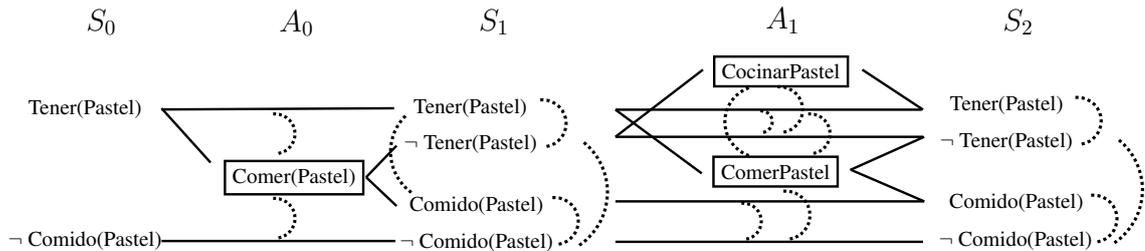


Figura 5.3: Grafo de planificación del problema “tener y comer pastel”. Los rectángulos indican acciones, las líneas rectas - aplicación de alguna acción (puede ser acción persistente). Los enlaces de exclusión mutua son representados como curvas punteadas.

5.5.1. Grafos de planificación para estimación de heurísticas

Un **grafo de planificación**, una vez construido, es una fuente rica de información acerca de un problema. Podemos estimar el costo que supone alcanzar cualquier literal del objetivo como el nivel en el cual aparece por primera vez en el grafo de planificación. Lo llamaremos *costo de nivel* del objetivo.

Por ejemplo, Tener(Pastel) tiene costo de nivel 0 y Comido(Pastel) tiene costo de nivel 1. Esta estimación es admisible para objetivos individuales. La estimación puede no ser muy buena, ya que los grafos de planificación permiten varias acciones en cada nivel mientras la heurística tiene en cuenta simplemente niveles (y no la cantidad de acciones para alcanzar al literal).

Por esta razón, es común usar un *grafo de planificación serial* para las heurísticas. Un grafo serial exige que sólo una acción pueda ocurrir en una etapa de tiempo dada; esto se logra añadiendo enlaces de exclusión mutua entre cada par de acciones excepto acciones persistentes. Los costos de nivel extraídos de grafos seriales son frecuentemente estimaciones razonables de costos reales.

Para estimar el costo de una secuencia de objetivos, existen tres enfoques simples:

- La heurística de *nivel máximo* simplemente toma el costo de máximo nivel de cualquiera de los objetivos; esto es admisible pero no necesariamente preciso.
- La heurística de *nivel suma* (asumiendo la hipótesis de independencia de objetivos) devuelve la suma de los costos de los niveles objetivo. Esto no

es admisible pero funciona bien en la práctica en problemas que se pueden descomponer en partes. Es más precisa que la aplicación de heurísticas de número de objetivos insatisfechos presentada anteriormente.

- La heurística de *nivel de conjunto* encuentra el nivel en el que todos los literales de la secuencia de objetivos aparecen en el grafo de planificación, sin que ningún par de ellos sean mutuamente excluyentes. Es admisible y más precisa que la heurística de nivel máximo.

5.5.2. El algoritmo GRAPHPLAN

Veremos como extraer un plan directamente de un grafo de planificación. El algoritmo **GRAPHPLAN** tiene dos etapas fundamentales, las cuales se alternan dentro del ciclo del algoritmo:

- Se chequea si todos los literales del objetivo están presentes en el nivel actual sin que existan enlaces mutuamente excluyentes entre cualquier par de ellos. En este caso se extrae la solución.
- Se extiende el grafo añadiendo acciones y literales de estado para los siguientes niveles.

A continuación se da el algoritmo GRAPHPLAN:

```
función GRAPHPLAN(problema):
  grafo := Grafo-Planificación-Inicial(problema)
  objetivos := Objetivos(problema)
  repetir
    si (todos los objetivos están en el último nivel del grafo) ∧
      (son enlaces mutuamente excluyentes del grafo) entonces
      solución := Extraer-Solución(grafo, objetivos, Longitud(grafo))
      si (solución ≠ error)
        retornar solución
      sino si (¬ Solución-Posible(grafo))
        retornar error
  grafo := Grafo-Expandir(grafo, problema)
```

Vale la pena ver algunos ejemplos de relaciones de exclusión mutua y sus causas. Para esto vamos a introducir un problema de cambiar la rueda del auto:

```
Iniciar( En(Rueda, Eje) ∧ En(Repuesto, Baúl) )
Objetivo( En(Repuesto, Eje) )
```

Acción Quitar-Repuesto

Precondición: En(Repuesto, Baúl)

Efecto: ¬ En(Repuesto, Baúl) ∧ En(Repuesto, Suelo)

Acción Quitar-Rueda

Precondición: $\text{En}(\text{Rueda}, \text{Eje})$

Efecto: $\neg \text{En}(\text{Rueda}, \text{Eje}) \wedge \text{En}(\text{Rueda}, \text{Suelo})$

Acción Colocar-Repuesto

Precondición: $\text{En}(\text{Repuesto}, \text{Suelo}) \wedge \neg \text{En}(\text{Rueda}, \text{Eje})$

Efecto: $\neg \text{En}(\text{Repuesto}, \text{Suelo}) \wedge \text{En}(\text{Repuesto}, \text{Eje})$

Acción Dejar-Auto-Sin-Protección

Precondición:

Efecto: $\neg \text{En}(\text{Rueda}, \text{Suelo}) \wedge \neg \text{En}(\text{Rueda}, \text{Eje}) \wedge$
 $\neg \text{En}(\text{Repuesto}, \text{Suelo}) \wedge \neg \text{En}(\text{Repuesto}, \text{Eje}) \wedge$
 $\neg \text{En}(\text{Repuesto}, \text{Baúl})$

A continuación se dan ejemplos de exclusión mutua:

- *Efectos inconsistentes*: *Quitar-Repuesto* es una exclusión mutua con *Dejar-Auto-Sin-Protección* porque una tiene el efecto $\text{En}(\text{Repuesto}, \text{Suelo})$ y la otra su negación.
- *Interferencia*: *Quitar-Repuesto* es una exclusión mutua con *Dejar-Auto-Sin-Protección* porque uno tiene la precondición $\text{En}(\text{Repuesto}, \text{Baúl})$ y la otra tiene su negación como un efecto.
- *Necesidades conflictivas*: *Colocar-Repuesto* es una exclusión mutua con *Quitar-Rueda* porque uno tiene $\text{En}(\text{Rueda}, \text{Eje})$ como una precondición y el otro tiene su negación.
- *Soporte inconsistente*: $\text{En}(\text{Repuesto}, \text{Eje})$ es mutuamente excluyente con $\text{En}(\text{Rueda}, \text{Eje})$, porque el único modo de alcanzar $\text{En}(\text{Repuesto}, \text{Eje})$ es mediante *Colocar-Repuesto*, y esto es mutuamente excluyente con la acción persistente (nula) que es el único modo de alcanzar $\text{En}(\text{Rueda}, \text{Eje})$.

A continuación veremos como se busca la solución en el grafo de planificación (función Extraer-Solución del algoritmo GRAPHPLAN).

5.5.3. Búsqueda de solución en el grafo

- El *estado inicial* es el último nivel del grafo de planificación S_n , junto con el conjunto de objetivos del problema de planificación.
- Las *acciones disponibles* en un estado de nivel S_n están para seleccionar cualquier subconjunto de acciones libre de conflicto en A_{i-1} , cuyos efectos cubran los objetivos en el estado. El estado resultante tiene nivel S_{i-1} , y tiene como conjunto de objetivos las precondiciones para el conjunto seleccionado de acciones. Por “libre de conflicto” queremos decir conjunto de acciones tales que ningún par de ellas es mutuamente excluyente, y tampoco cualquier par de sus precondiciones.

- El *objetivo* es alcanzar un estado a nivel S_0 , tal que todos los objetivos sean satisfechos.
- El *costo* de cada acción es 1.

Se sabe que la planificación es PSPACE-completa (orden en tiempo polinomial) y puede darse la situación en la que la búsqueda de la solución sea muy larga. Por lo tanto, necesitamos alguna orientación heurística para la selección entre acciones durante la búsqueda hacia atrás. Un enfoque que trabaja bien en la práctica es un algoritmo basado en el costo de nivel de los literales. Para cualquier conjunto de objetivos, procedemos de la siguiente manera:

1. Seleccionar primero el literal con el costo de nivel más alto.
2. Para alcanzar este literal, elegir la acción con la precondition más sencilla. Esto es, elegir una acción tal que la suma (o el máximo) del nivel de costo de sus preconditiones sea la más pequeña.

Se puede demostrar que GRAPHPLAN no caerá en un bucle, extendiendo el grafo de planificación en cada iteración. La idea de demostración es que las acciones y los literales crecen y las relaciones mutuamente excluyentes decrecen. Esto quiere decir, una vez que una acción o un literal aparece en cierto nivel de grafo, aparecerá en los niveles siguientes debido a acciones persistentes (nulas) que preservan el conjunto de literales en niveles subsiguientes. Los enlaces de exclusión mutua decrecen monótonamente: si dos acciones son mutuamente excluyentes en un nivel dado A , entonces también lo son para todos los niveles previos en los cuales ambas aparezcan. Lo mismo sucede para exclusiones mutuas entre literales.

Como las acciones y los literales crecen y las relaciones mutuamente excluyentes decrecen, y como sólo existe un número finito de acciones y literales, cada grafo de planificación se estabiliza (todos los niveles siguientes serán idénticos).

5.6. Sistema de planificación FF

El algoritmo de planificación usado por el planificador **FF** [Hof01] (fast forward - búsqueda rápida hacia adelante) es un sistema de planificación que, según nuestro conocimiento, es el mejor sistema hasta la actualidad, por esta razón lo usamos en esta tesis. Este sistema y sus variaciones ganaron las últimas 13 competencias de planificación AIPS.

El planificador usa una adaptación del lenguaje ADL, llamado **PDDL**, que viene de las sílabas en inglés “Planning Domain Definition Language”, esto es, lenguaje para definir dominio de planificación. Se usó por primera vez en la competición de planificación AIPS-98 para formalizar la especificación de los problemas de planificación. Este lenguaje separa el *dominio del problema* de un *problema particular* de este dominio. El dominio del problema *define* los **tipos**, **predicados**

y **acciones**, mientras que un problema en particular define los **objetos** y estados **inicial** y **final**.

A continuación explicaremos como funciona el planificador FF.

FF resuelve los problemas de planificación con búsquedas hacia adelante en el espacio de estados. Estas búsquedas están guiadas por una función heurística que es automáticamente extraída del dominio. La heurística se obtiene a partir de la especificación de la instancia de planificación y sirve para guiar la búsqueda en el espacio de estados. En el capítulo 3 vimos el algoritmo A*, que también usa función heurística. Para conseguir esta función heurística el sistema relaja el problema de planificación ignorando partes de la especificación (ver sección 5.6.2).

FF usa una técnica llamada **enforced hill-climbing** combinada con dos potentes **técnicas de poda** (prunning), que veremos en breve. La estrategia de búsqueda y las técnicas de poda fueron motivadas por observación de distintos tipos de ejemplos de problemas de planificación, tales como Assemble, Blocks World, Grid, Gripper, Logistics, Mystery y otros [Hof01].

La técnica *enforced hill-climbing* trata de encontrar la solución aproximándose al estado objetivo, es como escalar la colina acercándose al estado objetivo hasta encontrar el plan para lograr tal objetivo (un máximo global) o hasta hallar el máximo local. Las técnicas de *poda* cortan ramas del árbol de planificación sacando estados que probablemente no lleven al estado objetivo. Una de estas técnicas selecciona un conjunto de sucesores prometedores de cada estado, y otra que poda las ramas en las que algún objetivo se alcanza demasiado rápido.

5.6.1. Arquitectura del sistema

El algoritmo de búsqueda enforced hill-climbing es una *búsqueda hacia-adelante*, que para el estado actual llama a **GRAPHPLAN relajado** (GRAPHPLAN está explicado en la sección 5.5.2), el cual devuelve la distancia estimada al objetivo, y además da también las acciones sugeridas, las llamadas **acciones útiles** (helpful actions en inglés).

También se integran algunas *optimizaciones* que fueron descubiertas durante los tests:

- Si el problema de plan contiene estados, de los cuales el objetivo es *inalcanzable* (callejones sin salida), entonces enforced hill-climbing *puede fallar* en encontrar la solución, en este caso se hace una búsqueda en anchura completa (breadth-first search) desde principio.
- Cuando hay *ordenamiento de los objetivos*, enforced hill-climbing a veces *demora mucho* logrando objetivos que se deberían haber hecho más tarde. Así, se integran dos técnicas adicionales:
 1. **Eliminación de objetivo agregado**, que corta las ramas, donde algún objetivo fue logrado demasiado temprano.
 2. **Agenda de objetivos**, adaptada del trabajo de Jana Koehler [KH11], provee los objetivos al planificador en el orden predeterminado.

5.6.2. Función heurística

Tratando de crear un sistema de planificación independiente del dominio usando *búsqueda heurística*, la dificultad principal radica en la *obtención automática* de la función heurística. Un enfoque común para derivar una heurística es *relajar* el problema P usando un problema más sencillo P_0 , el cual puede ser resuelto eficientemente. Se puede entonces utilizar la longitud de la solución de P_0 para estimar la dificultad del problema original. Bonet et al. [BLG97] han propuesto una manera de aplicar esta idea a la planificación independiente del dominio. Se relaja la descripción del problema *omitiendo las listas de eliminación* de las acciones. En el problema relajado, todas las acciones sólo añaden nuevos átomos al estado siguiente, no eliminan ninguno. Durante la ejecución de una secuencia de acciones de plan relajado el estado solamente crece (en cantidad de átomos lógicos) y el problema se resuelve cuando cada literal-objetivo es añadido por alguna acción.

Vamos a ilustrar un problema relajado. Digamos que tenemos una acción Mover, que mueve a un bot de un punto A a otro punto B. La precondition dice que el bot tiene que estar en la posición A para que la acción sea aplicable. Después de aplicar la acción Mover, la *lista de átomos agregados* produce un hecho que sitúa al bot en B, y la *lista de eliminación* quita el hecho de que estaba en A. En la relajación, el borrado no se tiene en cuenta, por lo después de aplicar la acción el bot se encuentra en A y B simultáneamente.

La longitud de una *solución óptima* del problema relajado es una heurística *admisible* (porque nunca sobreestima), y en teoría podría ser usada con el *algoritmo* A^* . Sin embargo, el cálculo de la longitud de una solución óptima del problema relajado es NP-complejo [Byl94]. A continuación explicamos como se estima la longitud del plan en FF.

Usando GRAPHPLAN podemos encontrar una solución al problema relajado en tiempo polinomial [HN01]. Recordemos que el grafo de GRAPHPLAN consiste en alternar niveles de literales con niveles de acciones. Primer nivel de literales es S . Primer nivel de acciones son todas las acciones aplicables en S . Aplicando estas acciones junto con literales del primer nivel formamos el segundo nivel, etc. hasta llegar al nivel donde están todos los literales del estado objetivo. En *GRAPHPLAN relajado* solamente hay interacciones positivas, no existen interferencias y por esta razón tampoco hace falta tener en cuenta las exclusiones mutuas (véase sección 5.5.2).

Una vez que llegamos al nivel S_m donde están todos los literales del estado objetivo (literales-objetivos) la *solución relajada* se extrae de la siguiente forma. Empezando por S_m , trabajamos con literales-objetivos. Para cada nivel S_i si el literal-objetivo está presente en nivel anterior S_{i-1} , entonces lo insertamos a literales-objetivos que tienen que ser logrados en nivel $i - 1$. Si no tomamos una acción en el nivel $i - 1$ tal que agrega al literal-objetivo en S_i e insertamos sus precondiciones a literales-objetivos que tienen que ser logrados en nivel $i - 1$. Una vez que terminamos con literales-objetivos en nivel i , seguimos con el nivel $i - 1$, y

así sucesivamente hasta llegar al primer nivel. El proceso genera un plan relajado $\langle O_1, O_2, \dots, O_m \rangle$, donde cada O_i es el conjunto de acciones aplicadas en el nivel i y m es el nivel máximo. Entonces estimamos la longitud de la solución contando la cantidad de acciones en el plan de solución aproximada:

$$h_{FF}(S) := \sum_{i=1, \dots, m} |O_i|$$

5.6.3. La búsqueda

Dado un estado S , FF evalúa los estados sucesores. Si ninguno de ellos tiene una mejor heurística, FF continúa con sucesores de sucesores, así siguiendo hasta encontrar un estado S' con mejor heurística. Una vez encontrado, se agregan las acciones que llevan de S a S' al plan y FF sigue todo el proceso de vuelta, pero ahora desde S' . Si el problema no contiene situaciones de callejón sin salida, esta estrategia siempre encuentra una solución [HN01].

Se ha reconocido en la comunidad SAT (viene de “satisfiability” - problema de satisfacibilidad booleana) que el comportamiento de un método de búsqueda local depende de manera crucial de la estructura del problema que trata de resolver [FCS97]. Las características importantes aquí son: la cantidad y distribución de soluciones, así como el tamaño de mínimos locales y llanuras. La observación de los autores del FF es que las llanuras y mínimos locales tienden a ser pequeños en muchos problemas de planificación, cuando se usa la heurística del FF. Por lo tanto, es un enfoque adecuado tratar de encontrar salida de tales regiones haciendo búsqueda en anchura completa (BFS - breath-first search). Volveremos a esto más adelante.

5.6.4. Acciones útiles

El plan relajado que FF extrae de cada estado de búsqueda también puede ser usado en determinar al sucesor más útil y obtener información sobre orden de literales-objetivos [HN01]. Explicaremos como se identifican los estados sucesores útiles usando las llamadas **acciones útiles**. Para dar una intuición de que se trata veremos un ejemplo del dominio “Pinza”. Hay dos habitaciones A y B, dos pelotas y un robot que tiene dos pinzas para levantar o soltar pelotas. Supongamos que el robot está en la habitación A con las dos pelotas en las pinzas, y tiene que soltarlas en la habitación B. La solución relajada que extrae heurística de FF en este ejemplo es la siguiente:

$$\langle \{MOVE A B\}, \{DROP ball1 B left, DROP ball2 B right\} \rangle$$

Aquí podemos ver que la única acción que ayuda en el estado inicial es la acción MOVE, ya que tirando las dos pelotas en la habitación A nos aleja del estado objetivo. Entonces los autores del FF definen el conjunto de **acciones útiles** $H(S)$ como sigue:

$$H(S) := \{ o \mid pre(o) \subseteq S, add(o) \cap G1 \neq \emptyset \}$$

Aquí $G1$ representa el conjunto de los objetivos que la extracción del plan relajado construye un nivel por delante del nivel inicial del grafo. Por consiguiente, se consideran como útiles aquellas acciones que suman al menos un literal-objetivo en el nivel más bajo de la solución relajada. Estas son las acciones que *pueden* ser seleccionados para el primer conjunto en la solución relajada. Los sucesores de cualquier estado S de búsqueda en anchura se limitan a $H(S)$. Si bien esta búsqueda *no es completa*, este enfoque funciona bien en la mayoría de los problemas de planificación actuales. Si *enforced-hill climbing* no logra encontrar una solución, simplemente se ejecuta algoritmo A^* (con la misma heurística) que es completo.

En el capítulo que sigue vamos a mostrar para que se usa el planificador FF dentro del juego y también daremos ejemplos de dominios y problemas de planificación colaborativa en PDDL.

Capítulo 6

Planificación colaborativa en un juego

En este capítulo se describirá el objetivo del juego y como el bot lo “ve” al dominio (sección 6.1), luego en la sección 6.2 explicaremos las posibles tareas que pueden realizar los bots y que subtareas incluyen. Para las tareas que usan planificador, en la sección 6.3 se darán ejemplos de PDDL generado y veremos la forma en que los bots proponen, coordinan y ejecutan el plan obtenido (sección 6.4). Veremos la estrategia que utilizan los bots para pasar el nivel jugado en la sección 6.5. Por último, en la sección 6.6 vamos a describir como se generaron distintos dominios del problema para los experimentos usando planificación.

6.1. El juego

Nuestro juego consiste en encontrar la salida de un **laberinto**. Las habitaciones, o mejor dicho **áreas** están divididas por **puertas** o **paredes**. Para abrir las puertas existen **botones**, de hecho, un botón puede abrir varias puertas. Hay botones que no pueden ser alcanzados, para poder usarlos hay que dispararles usando **armas** con *miras ópticas*. Para poder subirse a una pared, se necesita tener una **caja** cerca. Para poder levantar una caja y llevarla cerca de alguna pared se necesita un **arma de gravedad**, que tiene la habilidad de levantar objetos pesados.

6.1.1. Objetivos y comunicación

El objetivo del juego es tener todos los jugadores (humanos o bots) en la salida del laberinto, un lugar definido por el diseñador del mapa, desde ahora lo llamaremos **área objetivo**. Los jugadores comienzan en lugares distintos, separados por puertas, áreas y paredes. Empiezan investigando el área en la cual están, usando *nombres de áreas* definidas por el diseñador del mapa para poder distinguirlas. Luego tienen que coordinar sus movimientos para investigar qué puertas abren qué botones descubiertos hasta el momento. Las puertas, botones y cajas tienen

un *número* asignado y los bots saben esta información de antemano, es decir cuando un bot se acerca a una puerta o un botón, sabe su número. Un humano también puede acceder a esta información fácilmente, ya que está visualmente disponible. Se podría haberlo hecho de otra forma, por ejemplo el bot al ver una puerta asignarle un número que la identifique unívocamente, pero la idea es la misma: tener cada puerta, caja o botón numerado, así poder distinguirlos y ubicarlos. Esta característica la poseemos los humanos: al encontrarnos en un laberinto, teniendo solamente un lápiz y un cuaderno, empezamos investigando el área en el cual estamos, tratando de dibujar un mapa del lugar, numerando puertas y botones que encontramos y tratando de llegar a nuevas áreas y si no es posible, investigar que puertas abre algún botón que vimos durante la exploración del laberinto.

Además, los jugadores se pueden hablar entre sí por medio del chat, posiblemente tratando de coordinar sus movimientos. Por ejemplo uno de los jugadores puede proponer presionar un botón para ver que puertas abre el mismo o proponer llevar alguna caja cerca de alguna pared para poder subirse a la misma. También se puede usar el chat para notificar a los otros jugadores sobre cambios en el dominio, por ejemplo, al ver cambio de estado de una puerta o entrando en nueva área, viendo un botón nuevo, etc. En nuestro juego la comunicación es muy importante, ya que solamente teniendo la información precisa sobre el dominio del juego, se puede armar un plan para poder pasar el nivel.

A continuación vamos a ver como el bot representa y usa la información del dominio de juego.

6.1.2. Representación de dominio del juego

Al iniciar un mapa, el plugin lee los waypoints y los objetos de interés, en nuestro caso son las áreas, puertas, cajas, botones y armas. Se considera que entre los dos waypoints más cercanos a la puerta se encuentra la puerta misma. Entonces para cada puerta el plugin sabe entre qué waypoints está la misma. Para saber si la puerta está abierta, se traza un rayo de visibilidad entre estos dos waypoints: si el rayo toca la puerta, entonces está cerrada, sino está abierta.

En su estado interior los bots tienen anotados los objetos del mapa descubiertos y otra información adicional: cajas (y en que áreas se encuentran), puertas (y sus estados: abierta-cerrada), botones, áreas etc. Cuando el bot está próximo al waypoint que tiene un objeto cerca, el bot observa su estado y determina si tiene que hablar o no sobre este objeto. Por ejemplo cuando el bot está próximo a uno de los dos waypoints de la puerta, realiza el test de visibilidad entre los dos waypoints de la puerta. Si esta puerta no fue descubierta por él u otro jugador, o si el estado de la puerta es distinto del que el bot tenía registrado, usa el chat para notificar a otros jugadores sobre este hecho, por ejemplo puede decir “I just found door 6, it is closed” al ver que la puerta 6 está cerrada. Con el botón pasa lo mismo: si el bot pasa por un waypoint que tiene tipo “botón” y además nadie nunca habló del mismo, notifica a los jugadores que encontró un botón nuevo. También al estar próximo a una pared, el bot habla sobre esto.

Es un poco distinto con la caja, ya que su posición puede cambiar: el bot “anota” el área en la cual está la misma en el momento de verla o escuchar hablar de ella, en este último caso asume que el área donde está la caja es la misma que el área del jugador que habla. Si el área de la caja no coincide con el área que tiene registrada el bot, también lo notifica, salvo que algún bot esté usando arma de gravedad sobre la misma.

El bot siempre notifica cambio de área: al pasar por un waypoint, se compara el área del mismo con el área del waypoint previo, y si son distintas, el bot comunica que cambió de área. Esto se hace para que los jugadores siempre tengan el estado más actualizado del dominio de juego, así pueden armar planes válidos. Un jugador humano debería hacer lo mismo, si es que quiere salir del laberinto.

Los bots distinguen entre jugadores *colaborativos* y *no colaborativos*. Suponen que un jugador es colaborativo, en el momento que *les saluda*. Esta decisión del diseño del bot fue tomada solamente por su simpleza. Hay muchas otras formas de determinar si un jugador es colaborativo, pero normalmente se resolvería al momento de preguntar al jugador si desea colaborar para llegar al área objetivo. En los planes los bots pueden usar a estos jugadores colaborativos en las acciones, esto solo si el jugador acepto a participar en el plan. Ya veremos de que se trata más adelante.

Naturalmente los bots suponen que todo lo que se dice por medio del chat es *una verdad* absoluta, hasta que se den cuenta de lo contrario. Cabe destacar, que si un jugador quiere “mentir” a los bots, lo puede hacer, pero obviamente esta estrategia no es la más inteligente para poder salir del laberinto. Decir la verdad es una necesidad básica de todo agente colaborativo [Gri75]. Por ejemplo un jugador puede decirle al bot que una puerta está abierta cuando en realidad está cerrada o mentirle sobre su posición actual (siendo que está en otra área), esto logrará que el bot arme planes que no puedan llevarse a cabo, porque están basados en hechos falsos. Sin embargo, hay que darse cuenta que un ser humano tampoco sabría si otro jugador le dice la verdad o no, ya que no lo ve y no sabe lo que hace, así que probablemente actuaría de la misma forma que nuestro bot (por lo menos hasta que se de cuenta de que lo engañan).

Si un jugador dice que la puerta N está abierta, el bot “anotará” el estado de la puerta, y sólo lo va a cambiar si algún otro jugador le dice o el mismo bot descubre que está cerrada. La acción de mentirle al bot es reversible. Si el jugador dijo alguna vez (equivocadamente tal vez) que el botón N abre puerta M y el bot lo “anotó” sin darse cuenta que puede ser mentira, el jugador puede decir que botón N no afecta a la puerta M, lo que cambiará el estado “interior” del bot. Los bots fueron diseñados para no mentir, ya que su meta es salir del laberinto.

Los bots tienen cierta cantidad de tareas que pueden realizar, y cada de estas tareas tiene su prioridad. Al terminar una tarea, el bot busca nueva tarea de acuerdo a las prioridades de las mismas. Veremos en que consisten las tareas y explicaremos el algoritmo que emplea el bot para poder salir del laberinto.

6.2. Tareas de bots

El bot en cada momento tiene una **tarea** fija que está realizando o puede estar libre, sin tarea. Las tareas no son interrumpibles, es decir que el bot responderá que está ocupado a cualquiera que trate hablar con él mientras está realizando alguna tarea. Solamente puede aceptar una propuesta de un plan, cuando no tiene nada para hacer o está esperando a otros jugadores. A continuación se dan las posibles tareas de bots junto con sus respectivas descripciones:

- **Investigación de un área.** Esta tarea consiste en recorrer todos los waypoints de un área. La tarea se lleva a cabo de la siguiente forma: se elige un waypoint al azar del área de interés y usando el algoritmo A* se calcula el camino hacia el mismo. Al llegar a ese waypoint, se elige otro waypoint no visitado del área, así sucesivamente. Cuando bot “toca” a waypoint por el cual pasa, éste se marca como visitado. Mientras el bot pasa por los waypoints, describe objetos desconocidos o de diferente estado. La tarea se completa en el momento de visitar el último waypoint del área.
- **Probar una configuración botón-puerta.** La tarea, en general, consiste en apretar el botón N (o dispararle) y ver si este botón afecta o no la puerta M. Una variación es solamente apretar el botón y ver que puertas abre (cuando el botón nunca fue presionado). Para que pueda llevarse a cabo, uno de los jugadores tiene que estar en el área del botón y otro jugador tiene que estar en una de las áreas de la puerta. Uno de los jugadores tiene que presionar el botón y decirlo, luego todos los jugadores recorren puertas cercanas viendo si cambian su estado. Si el estado de la puerta no cambia, el jugador deduce que el botón no afecta a la puertas, en caso contrario deduce que el botón sí afecta a la puerta. El bot siempre avisa a los otros jugadores sobre esto, por ejemplo puede decir: “The door 2 stays closed. So, button 6 doesn’t affect door 2.”; así todos los bots que reciben este chat registran el estado de la puerta 2 (cerrada) y el hecho de que el botón 6 no afecta a la puerta 2.
- **Ejecución de un plan.** Esta tarea tiene varios pasos: generación de dominio y objetivo en PDDL, ejecución del planificador FF, parseo del resultado y en el caso de exista el plan, su proposición y ejecución. Veremos bien de que se trata en la sección 6.3.

Las tareas de investigación de un área y probar una configuración son acciones que no requieren modificar el mundo sino el conocimiento del jugador. Son acciones **no deterministas**, es decir su resultado es desconocido a priori, por eso no pueden ser modelados completamente en FF, dado que el último no soporta tales acciones. Un planificador que soporta acciones no deterministas es **PKS** [Pet04].

Hace falta indicar que si la tarea de probar una configuración botón-puerta requiere uso de planificador (esto es, no hay jugadores en el área del botón o en el área de la puerta), entonces primero se hace *la ejecución del plan* (que tiene como

objetivo dejar un jugador en el área de la puerta y otro en el área del botón) y una vez que se detecta que el objetivo de planificador ha sido logrado, la tarea se cambia a *probar la configuración dada*.

Una vez que el planificador devuelve un plan válido, es decir una *secuencia de acciones PDDL*, cada acción PDDL se transforma en **subtarea**. Estas **subtareas** son secuencias de **acciones atómicas** y se guardan en el **stack** (o pila) de las acciones actuales del bot. Una acción atómica es un comportamiento indivisible, ejecutado por el motor del juego con una animación particular. En la figura 6.1 se muestran todas las posibles acciones atómicas.

Nombre de acción	Argumentos	Descripción
Retraso	T - cantidad de segundos	Espera el tiempo T antes de seguir.
Moverse	W - número de waypoint	Calcula una ruta al waypoint W usando algoritmo A* y corre hasta ahí.
Mirar	O - objeto (botón, caja, etc.)	Cambia la mira para apuntar al objeto O.
Usar		Usar el objeto al cual está mirando (en caso de botón, lo presiona).
Seleccionar-arma	T - tipo de arma	Cambia de arma a T.
Disparar		Dispara el arma actual.
Disparo-secundario		Usa acción secundaria del arma: para arma de gravedad es levantar el objeto y para arma con mira óptica es usar/sacar la mira.
Chat	C - Número de chat y sus parámetros (número de botón, puerta, etc.)	Usa chat para notificar a otros jugadores sobre algo.

Figura 6.1: Acciones atómicas y sus argumentos.

Distinguimos las subtareas de las acciones PDDL por el hecho de que el bot no usa el planificador todo el tiempo. Puede tener como tarea investigar un área o probar una configuración botón-puerta, sin embargo en todo momento tiene una subtarea para realizar. Ejemplos de subtareas: apretar un botón, correr hacia una puerta, levantar una caja, disparar un arma, etc. En la figura 6.2 se pueden observar dos subtareas: disparar a un botón y levantar una caja, con sus respectivas acciones atómicas.

En la próxima sección vamos a ver el armado del problema en PDDL. El planificador FF tratará de obtener un plan, que es una secuencia de acciones PDDL para resolver un problema dado (por ejemplo llevar a todos los jugadores al área objetivo). El bot, teniendo esta secuencia de acciones, las procesa una por una. Dada una acción PDDL, el bot la transforma en subtarea correspondiente e inserta las acciones atómicas de la subtarea en el stack de tareas. Por ejemplo la acción PDDL *SHOOT-BUTTON* que veremos en la siguiente sección se transforma en subtarea *Disparar-botón*, mostrada en la figura 6.2.

Subtarea y argumentos.	Secuencia de acciones atómicas.
Disparar-botón. Argumentos: <ul style="list-style-type: none"> • Botón B. • Waypoint W, desde el cual se ve el botón B. • Arma A con mira óptica. 	<ol style="list-style-type: none"> 1. Moverse(W). 2. Seleccionar-arma(A). 3. Mirar(B). 4. Disparo-secundario: para usar el aumento de la mira óptica. 5. Disparar. 6. Disparo-secundario: para sacar el aumento de la mira óptica. 7. Retraso(2). 8. Chat(el botón ha sido disparado).
Levantar-caja. Argumentos: <ul style="list-style-type: none"> • Caja C. • Waypoint W1, el más cercano a la caja C. • Arma de gravedad A. 	<ol style="list-style-type: none"> 1. Moverse(W1). 2. Seleccionar-arma(A). 3. Mirar(C). 4. Chat(ahora voy a levantar la caja C). 5. Retraso(2). 6. Disparo-secundario: para levantar la caja.

Figura 6.2: Subtareas de disparar a un botón / levantar una caja.

6.3. Planificación colaborativa

Habíamos visto en el Capítulo 5 el lenguaje PDDL usado por el planificador FF. También vimos que el planificador FF toma como entrada dos archivos PDDL: el archivo del dominio del problema y el archivo de un problema en particular. Recordemos que el dominio del problema incluye definiciones de los tipos, predicados y acciones, en cambio un problema en particular define los objetos, sus relaciones (usando predicados) y los estados inicial y final. En lo que sigue describiremos el dominio del problema y como hace el bot para generar el problema en PDDL automáticamente en cada estado del juego.

Dentro del juego el planificador se usa para lograr alguno de los 3 objetivos (recordemos que la tarea correspondiente es *ejecutar un plan*):

- Llevar a todos los jugadores colaborativos al **área objetivo**. Este es el objetivo principal del juego. Recordemos que puede no existir un plan para lograr esto.
- El bot quiere investigar una nueva área, pero para pasar ahí necesita subirse a una pared. El objetivo entonces es **llevar una caja** a esa pared, es decir que exista una caja en el área de la pared para poder subirse arriba y saltar la pared.
- Probar una **configuración botón-puerta** desconocida, esto es si el bot no sabe si un botón afecta o no a una cierta puerta. El objetivo es entonces que exista un jugador colaborativo cerca del botón y otro cerca de la puerta. Otra

variación es cuando el botón nunca fue presionado, entonces el objetivo es solamente tener a uno de los jugadores en el área de ese botón.

Ya habíamos visto en el Capítulo 3 que cada waypoint pertenece a una sola área. Entonces, si los waypoints están correctamente ubicados, la puerta separa dos áreas adyacentes, es decir que los dos waypoints de la puerta deberían pertenecer a dos áreas distintas. La cantidad de áreas es mucho más chica que la cantidad de waypoints, y por esta razón en vez de waypoints vamos a usar áreas en el armado de planes. Para comparar, en el mapa de los experimentos hay más de 100 waypoints y solamente 11 áreas, lo que permite reducir drásticamente el espacio de estados durante búsqueda de un plan y de esta forma se reduce el tiempo de búsqueda. El planificador es usado para moverse entre áreas y el algoritmo A* para moverse dentro de un área.

6.3.1. Dominio en PDDL

A continuación vamos a definir los posibles tipos, predicados y acciones usados por el planificador. Los tipos usados son: **bot**, **area**, **weapon** (arma), **door** (puerta), **button** (botón), **box** (caja). En lo que sigue definimos a los predicados y explicamos para que sirve cada uno:

- (*physcannon ?weapon - weapon*) - indica si un arma es un arma de gravedad.
- (*sniper-weapon ?weapon - weapon*) - indica si un arma es un arma francotirador, para usar el aumento y disparar botones.
- (*can-move ?from ?to - area*) - sugiere que se puede pasar de área *?from* a área *?to*. Usamos este predicado para indicar que una puerta está abierta.
- (*can-shoot ?button - button ?area - area*) - dice que el botón *?button* puede ser disparado del área *?area*.
- (*wall ?from ?to - area*) - significa que hay una pared que hay que subir para pasar de área *?from* a área *?to*.
- (*at ?bot - bot ?area - area*) - indica que *?bot* está en un área particular.
- (*box-at ?box - box ?area - area*) - indica que la caja está en el área dado.
- (*button-at ?button - button ?area - area*) - el botón está en el área dado.
- (*weapon-at ?weapon - weapon ?area - area*) - el arma está en el área dado.
- (*has ?bot - bot ?weapon - weapon*) - el bot tiene el arma dada.
- (*carry ?bot - bot ?box - box*) - el bot está usando el arma de gravedad para levantar la caja.
- (*empty ?bot - bot*) - el bot no está levantando ninguna caja.
- (*between ?door - door ?area1 ?area2 - area*) - indica el hecho de que una puerta separa dos áreas.
- (*toggle ?button - button ?door1 ?door2 - door*) - el botón *?button* afecta dos puertas. Cabe destacar que un botón puede abrir a lo sumo dos puertas. Si botón afecta una sola puerta, la puerta se repite en este predicado.

Por último enumeramos todas las acciones usadas en el juego:

- *move* (?bot - bot ?to ?from - area) - bot pasa a otra área.
- *take-weapon* (?bot - bot ?weapon - weapon) - bot agarra un arma.
- *carry-box* (?bot - bot ?box - box ?area - area ?weapon - weapon) - bot levanta la caja con arma de gravedad.
- *drop-box* (?bot - bot ?box - box ?area - area) - bot deja la caja en el piso.
- *climb-box* (?bot - bot ?box - box ?from ?to - area) - bot sube a la caja y pasa al área destino.
- *fall* (?bot - bot ?to ?from - area) - bot se tira desde la pared a otra área.
- *push-button* (?bot - bot ?button - button ?door1 ?door2 - door) - bot aprieta un botón que cambia el estado de una o dos puertas.
- *shoot-button* (?bot - bot ?button - button ?door1 ?door2 - door ?weapon - weapon) - bot dispara a un botón.

Vamos a ver las acciones de subir caja y disparar un botón en PDDL. La primera acción, *climb-box* cambia estado del juego en el cual el jugador pasa de un área a otra usando una caja para subirse arriba y saltar la pared:

```

;-----
; Bot sube la caja para pasar a otro área.
;-----
(:action climb-box
  :parameters
    (?bot - bot
     ?box - box
     ?from ?to - area)

  :precondition
    (and
      (at ?bot ?from)
      (box-at ?box ?from)
      (wall ?from ?to)
      (empty ?bot)
    )

  :effect
    (and
      (not (at ?bot ?from))
      (at ?bot ?to)
    )
)

```

Ahora vamos a ver como cambia al mundo la acción de disparar un botón lo que cambia el estado de a lo sumo dos puertas. Esto es, si la puerta estaba abierta se cierra y si estaba cerrada se abre. En PDDL, al abrirse una puerta, se inserta el predicado *can-move* con áreas que separa la puerta como argumentos. Al cerrarse, se borra dicho predicado. En ADL, un efecto puede ser condicionado, esto se expresa usando la palabra *when* en PDDL. Usamos efectos condicionados para saber si la puerta estaba abierta o cerrada antes de cambiar su estado:

```

;-----
; Bot dispara botón que cambia estados de dos puertas.
;-----
(:action shoot-button
  :parameters
    (?bot - bot
     ?button - button
     ?door1 ?door2 - door
     ?sniper-weapon - weapon
     ?area - area
     ?door1-area1 ?door1-area2 - area
     ?door2-area1 ?door2-area2 - area)

  :precondition
    (and
      (sniper-weapon ?sniper-weapon)
      (at ?bot ?area)
      (has ?bot ?sniper-weapon)
      (empty ?bot)
      (can-shoot ?button ?area)
      (toggle ?button ?door1 ?door2)
      (between ?door1 ?door1-area1 ?door1-area2)
      (between ?door2 ?door2-area1 ?door2-area2)
    )

  :effect
    (and
      (when
        (can-move ?door1-area1 ?door1-area2)
        (and
          (not (can-move ?door1-area1 ?door1-area2))
          (not (can-move ?door1-area2 ?door1-area1))
        )
      )
      (when
        (not (can-move ?door1-area1 ?door1-area2))
        (and
          (can-move ?door1-area1 ?door1-area2)
          (can-move ?door1-area2 ?door1-area1)
        )
      )
      (when
        (can-move ?door2-area1 ?door2-area2)
        (and
          (not (can-move ?door2-area1 ?door2-area2))
          (not (can-move ?door2-area2 ?door2-area1))
        )
      )
      (when
        (not (can-move ?door2-area1 ?door2-area2))
        (and
          (can-move ?door2-area1 ?door2-area2)
          (can-move ?door2-area2 ?door2-area1)
        )
      )
    )
)

```

A continuación veremos como se genera automáticamente el dominio del problema y los objetivos en lenguaje PDDL considerando el estado actual del juego.

6.3.2. Generación de problema en PDDL

El bot genera al archivo de problema en PDDL y luego ejecuta el planificador FF, parseando el resultado. La generación de archivo PDDL se hace de la siguiente forma: primero se generan los objetos del dominio de planificador (bots, áreas, botones, etc.), luego el estado inicial (el mundo percibido por el bot) y después el estado objetivo o estado final. Es muy importante entender que el bot solamente usa los objetos y los hechos (en predicados) *descubiertos hasta el momento*, por ejemplo no puede definir el objeto *área objetivo* dentro de planificador, si nunca la investigó o nunca escuchó hablar de la misma.

Un ejemplo de generación de los objetos se muestra a continuación. Es bastante simple, su generación no requiere comentarios.

```
(:objects
  bot0 bot1 - bot
  area0 area1 area2 area3 area4 area5 area6 - area
  door0 door1 door2 door3 door4 - door
  button0 button1 button2 button3 button4 - button
  weapon0 weapon1 - weapon
  box0 box1 - box
)
```

En lo que sigue se puede observar el PDDL generado para el estado inicial. El bot accede a la información que tiene registrada, por ejemplo, posiciones de jugadores, posesiones de armas, posiciones de las cajas / puertas / botones, configuraciones conocidas de botón-puerta, etc., y la usa para generar el PDDL del estado inicial:

```
(:init
  ; Armas.
  (sniper-weapon weapon0) ; Arma con aumento óptico.
  (phycannon weapon1) ; Arma de gravedad para levantar cajas.

  ; Posiciones y armas de los bots.
  (at bot0 area1) (empty bot0) (has bot1 weapon0)
  (at bot1 area2) (empty bot1)

  ; Posiciones de las cajas.
  (box-at box0 area1) ; ...

  ; Posiciones de los botones descubiertos.
  (button-at button0 area8) ; ...

  ; Botón 9 no se puede alcanzar, pero sí disparar.
  (can-shoot button9 area9)

  ; Que áreas separan las puertas.
  (between door0 area8 area2) ; ...

  ; Que áreas son adyacentes (puertas abiertas).
  (can-pass area3 area2) ; ...
```

```

; Posición de paredes (que áreas separan).
(wall area10 area11) ; ...

; Configuraciones botón-puerta descubiertos.
(toggle button5 door7 door7) ; ...
)

```

El PDDL del objetivo final del juego, mover a todos los jugadores (bot0, bot1) al área-objetivo (area13) se arma como sigue:

```

(:goal
  (and
    ; Area13 es el area-objetivo.
    (at bot0 area13)
    (at bot1 area13)
  )
)

```

El objetivo de dejar la caja 0 cerca de la pared en el área 4 se muestra a continuación:

```

(:goal
  ; El area 4 es el área de la pared que hay que saltar.
  (exists (?box - box) (box-at ?box area4))
)

```

En lo que sigue veremos el objetivo de probar configuración botón-puerta en PDDL. Para hacer eso uno de los jugadores tiene que estar en el área del botón (área4) y otro en una de las dos áreas de la puerta:

```

(:goal
  (and
    (exists (?bot0 - bot) (at ?bot0 area4)) ; Área del botón.
    (or
      (exists (bot1 - bot) (at ?bot1 area5)) ; Área 1 de la puerta.
      (exists (bot1 - bot) (at ?bot1 area7)) ; Área 2 de la puerta.
    )
  )
)

```

Como mencionamos en la sección 6.2 no se puede modelar *completamente* la tarea de investigar un botón en FF porque es no determinista. Por esta razón usamos al planificador para ubicar a los jugadores antes de probar dicha configuración.

6.4. Proposición y ejecución de un plan

La **proposición** de colaborar en un objetivo común se hace de la siguiente manera. El bot propone el objetivo a los otros jugadores. Si alguno de los jugadores dice que no quiere colaborar, o que el bot espere, éste quedará en espera del jugador hasta que este último se desocupe o alguien le proponga hacer algo. Una vez que todos los jugadores colaborativos aceptan, el bot ejecutará el plan. Claramente el bot que tiene el plan, es el que manda, y los demás bots esperan sus comandos. De aquí en más llamaremos a este bot o jugador como **jefe**. Claro está que si un humano tiene un plan, puede ser jefe: puede proponer al plan y darles instrucciones a los bots.

Cuando el bot recibe un chat, interrumpe la acción atómica actual: la guarda en el stack, y hace que acción atómica Retraso(2) sea la acción actual (para simular el procesamiento). Luego procesa el chat, probablemente insertando acciones de Chat(respuesta) en el stack. De esta forma el bot continuará haciendo lo que hacía después de hablar. Si el chat recibido cancela la tarea actual el stack se limpia antes de insertar la acción de Chat(respuesta). Por ejemplo un bot al ejecutar subtarea Move(W) puede encontrar en el camino una puerta cerrada. Lo que hace el bot es notificar a otros jugadores sobre este hecho y cancelar la tarea actual. Los bots que reciben este chat también limpian su stack. Luego buscarán otra tarea para hacer.

Ejecutar el plan significa que el bot una por una ejecutará las acciones. Si el bot es el que figura en la acción, lo hace, pero si es otro jugador que lo tiene que hacer, entonces el bot-jefe le indicará y esperará que el otro jugador termine, esto es, el otro jugador le tiene que decir que terminó.

Si un bot recibe una proposición, responde de acuerdo a lo que estaba haciendo. Si tiene tarea, es decir si está investigando algún área, probando algún botón o está dentro de ejecución de un plan, entonces rechaza la propuesta, lo que pondrá al bot que hizo la propuesta en espera. Si no estaba haciendo nada, o estaba esperando a alguien, acepta. Rechaza proposiciones que vienen de jugadores no-colaborativos, recordemos que son los que no saludan.

Si durante la ejecución de un plan alguno de los bots encuentra una anomalía, esto es una contradicción a alguna de sus creencias (puerta cerrada en el camino, o que de alguna forma la caja deja de estar cerca de la pared que estaba) el bot lo notifica a los demás y el jefe tendría que indicarles a todos que el plan falló y proponer hacer alguna otra cosa. No propondrá hacer la misma cosa hasta que no tenga nada para hacer. Veremos mejor de que se trata al describir las ejecuciones en el capítulo 7.

6.5. La estrategia de los bots colaborativos

Cada uno de los bots tiene una estrategia para poder pasar el nivel jugado, la cual se resume a continuación:

1. El bot trata de armar un plan para que todos los jugadores lleguen al área objetivo. Como habíamos visto en la sección 6.3.2, para poder usar el área-objetivo dentro de planificador, la misma tiene que ser descubierta anteriormente. Si el bot encuentra un plan, lo propone, sino sigue en 2.
2. Si el bot tiene alguna nueva área para investigar, la investiga, nombrando por el camino los objetos de interés: puertas, sus estados, botones, cajas, paredes, etc. Sino sigue en 3. Cabe destacar que esta tarea no es colaborativa, cada uno de los bots lo hace por separado.
3. Si hay un área sin investigar, para llegar a la cual hay que subir una pared y además hay una caja vista anteriormente, el bot tratará de armar el plan para llevar la caja y dejarla cerca de esa pared. El objetivo de esto es poder investigar el área nuevo. Si hay tal plan, lo propone, sino sigue en 4.
4. Si hay un botón que no se sabe si afecta o no alguna puerta:
 - Si el botón está en el área del bot y hay otro jugador en el área de la puerta, el bot propone presionarlo. La tarea de investigar un botón es colaborativa, esto es, el bot necesita ayuda de otros jugadores (por lo menos para que le confirmen si puede presionar o no al botón), sin embargo no necesita usar el planificador.
 - En caso contrario el bot ejecuta planificador FF con objetivo de dejar a uno de los jugadores en el área del botón y a otro en el área de la puerta. En el caso de encontrar un plan, propone presionar al botón y ver si cambió el estado de la puerta. Sino sigue en 5.
5. En este punto el bot supone que no hay nada más para hacer, lo dice y quedará en espera hasta que alguien propone hacer algo o hasta que alguien hable sobre un cambio en el dominio (por ejemplo alguien dice que cambió el estado de alguna puerta), en este caso empieza todo de nuevo desde 1.

Como se ve aquí, cada uno de los bots “piensa” por sí mismo, es decir que no hay una entidad central que los controla. Son colaborativos en el sentido que el bot puede usar jugadores colaborativos dentro del planificador, y así armar secuencias de acciones que incluyen varios jugadores. La única condición aquí es que el planificador puede ser usado solamente por un bot al mismo tiempo, pero esta decisión solamente fue tomada por una cuestión de eficiencia.

6.6. Generación automática del dominio de juego

Una vez armado el mapa, el problema es definir que botón abre que puerta, para que sea desafiante poder pasar el nivel jugado. Para esto se guardó en un archivo PDDL la información completa del mapa: posiciones de botones, puertas, áreas, cajas, etc. Además a los objetos PDDL se agregaron 2 bots, para poder hacer la simulación. En el estado inicial todas las puertas están cerradas. El estado final es que los 2 bots estén en área-objetivo. Luego se armó un script, que básicamente repite los siguientes pasos:

- Para el estado inicial aleatoriamente se generan las configuraciones de los botones, es decir para cada botón X se agrega predicado (*toggle buttonX doorY doorZ*), donde Y y Z son puertas aleatorias (por supuesto son números de puertas válidas).
- Se ejecuta el planificador FF con el archivo generado y se parsea su salida.
- Si planificador encuentra un plan válido se cuenta la cantidad de pasos y se guarda la secuencia de acciones.

Haciendo 10000 ejecuciones de generaciones aleatorias de configuraciones de botones del mapa usado, obtuvimos 81.1 % de ejecuciones exitosas, es decir dominios del problema válidos que tenían un plan para llegar al área-objetivo. Los planes generados varían de 26 a 83 pasos.

Elegimos dos configuraciones, una de 82 y otra de 83 pasos. La configuración de 83 pasos se modificó, eliminando el acceso al área-objetivo (es decir se eliminó la puerta que lleva al área-objetivo de todos los *toggle* de PDDL) para que los bots fallen, pero antes de eso puedan descubrir lo máximo que se pueda de la configuración de botones. La de 82 pasos se dejó intacta, se puede pasar el nivel.

En el capítulo siguiente vamos a ver ejecuciones con estos 2 dominios generados.

Capítulo 7

Experimentos realizados

En este capítulo veremos 3 casos de estudio de ejecución de los bots colaborativos. Daremos las partes más importantes partes del chat generado, explicando paso a paso cual es la siguiente decisión del bot y porque fue tomada.

Las ejecuciones se realizan sobre el mismo mapa, pero con dos configuraciones distintas de botón-puerta, es decir que los botones cambian estados de puertas distintas. Vamos a ver 3 ejecuciones del juego:

1. Colaboración de *dos bots*. *No pueden llegar al objetivo*, porque la configuración de los botones fue diseñada para que los bots descubran todo lo posible del mapa jugado.
2. Colaboración entre *un humano y un bot*, en la cual *el humano miente*. Por esta razón el bot saca conclusiones incorrectas y crea planes inválidos.
3. Colaboración entre *un humano y un bot*, en la cual *pueden llegar al objetivo*. Esta ejecución también ayudó a descubrir algunos problemas en el módulo de chat.

El plan esquemático del mapa se muestra en la figura 7.1. La configuración 1 de los botones se muestra en la figura 7.2; no se puede pasar el nivel ya que ningún botón abre la puerta del área-objetivo. Con ✓ se marcan las puertas a las que afecta el botón dado. La configuración 2 de los botones se muestra en la figura 7.5.

7.1. Ejecución 1, dos bots colaborativos

Se usó configuración 1 de los botones. Hay 2 bots en juego: Jason y George. Los bots empiezan saludándose uno al otro (de esta forma se marcan como jugadores colaborativos), luego investigan las áreas en donde están. A continuación mostraremos partes del diálogo entre ellos, explicando las acciones internas que se ejecutan al recibir el chat del colaborador. El chat completo se encuentra en el Apéndice .E.

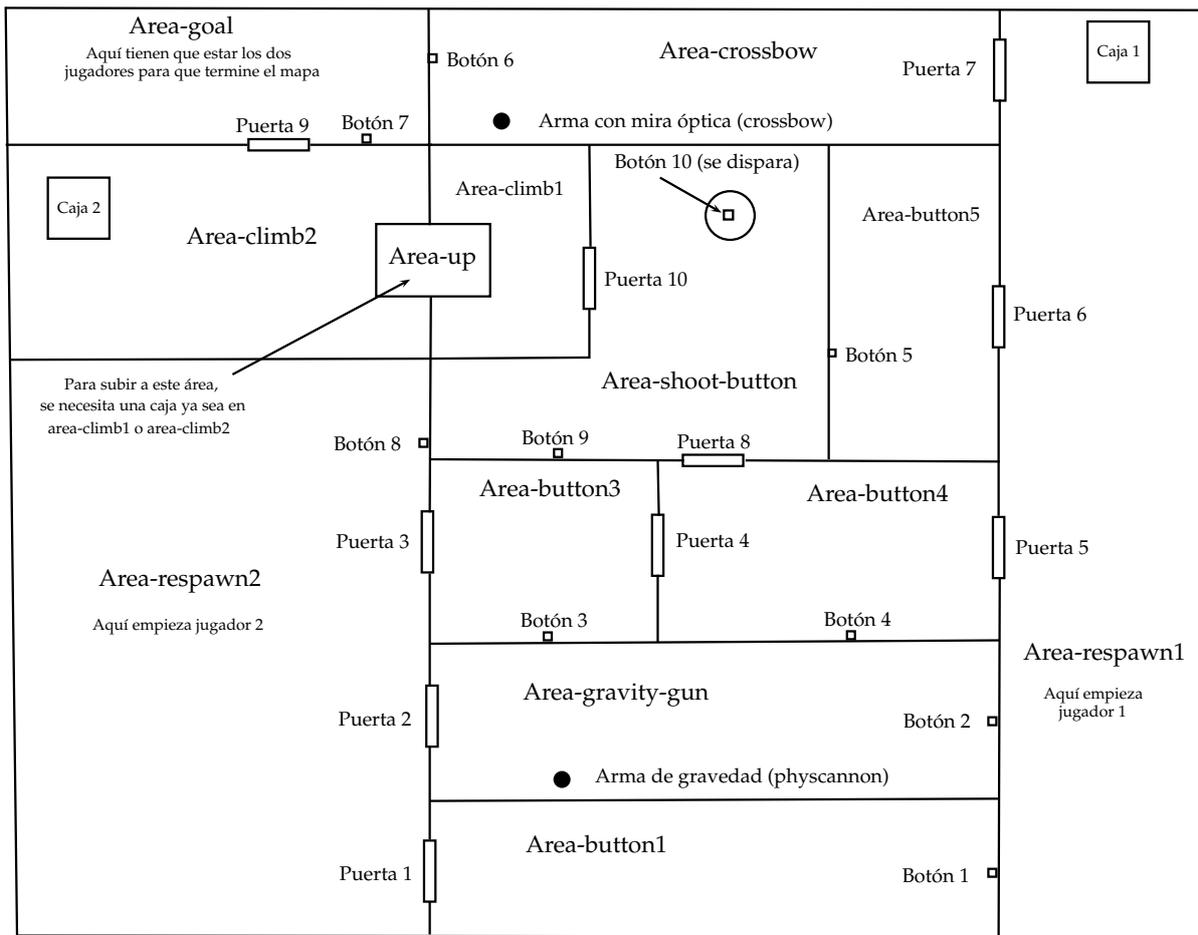


Figura 7.1: Plan esquemático del mapa.

7.1.1. Investigación de un área nuevo

001. Jason: Hello, George.

George marca a Jason como jugador colaborativo.

002. George: Hi there, Jason.

003. George: I'm in new area, let's call it area-respawn2.

Jason anota posición de George: area-respawn2.

004. Jason: I am at new area, call it area-respawn1.

005. George: I will explore new area.

Jason anota que George "está ocupado".

006. Jason: I will investigate new area.

007. George: I'm seeing closed door 3 here.

Los 2 bots anotan puerta 3 como vista y su estado: cerrada, para no volver a hablar de ella hasta que detecten el cambio de su estado.

008. George: I am seeing closed door 2 here.

009. Jason: I'm seeing closed door 5 here.

Puerta	1	2	3	4	5	6	7	8	9	10
Botón 1								✓	✓	
Botón 2			✓							✓
Botón 3			✓			✓				
Botón 4		✓						✓		
Botón 5		✓	✓							
Botón 6		✓						✓		
Botón 7							✓			✓
Botón 8				✓		✓				
Botón 9		✓			✓					
Botón 10						✓			✓	

Figura 7.2: Configuración 1 de los botones.

010. George: I'm seeing closed door 1 from here.

011. Jason: I'm seeing closed door 6 here.

012. George: All right.

Cuando el bot recibe información sobre el dominio, puede responder:

- Afirmativamente, como para indicarle al otro jugador que lo entiende. Se elige aleatoriamente una de las frases predefinidas para responder afirmativamente (ok, all right, cool, got it, etc.).
- Puede no decir nada, en este caso el bot entendió lo que el jugador le dijo.
- Si el bot no lo entiende al jugador se lo dice explícitamente. Hay que destacar aquí que los bots entienden uno al otro perfectamente, ya que usan las mismas oraciones del sistema.

013. Jason: I am seeing closed door 7 from here.

014. George: I see button 8 over here.

Los 2 bots anotan botón 8 como visto, para no volver a hablar de él al verlo otra vez.

015. Jason: Okey.

Hasta ahora los bots descubrieron las **puertas 1, 2, 3, 5, 6, 7** y el **botón 8**.

016. George: I explored the area i'm in currently.

George termina de investigar el área. Jason anota que George no tiene tareas.

017. George: Let's try figuring out which door opens button 8.

George al terminar la tarea busca una nueva, de acuerdo a la estrategia vista en la sección 6.5. Como no visitó el área-objetivo, no hay nuevas áreas para investigar y no hay ninguna pared vista hasta ahora sobre la que pueda saltar; se da cuenta que hay un botón para explorar: botón 8, el único botón visto que está en su área, propone presionarlo y ver que puertas afecta.

018. Jason: Please, no, wait.

Jason está investigando su área, rechaza tarea colaborativa. George anota a Jason como ocupado, y queda en espera.

019. Jason: I can see box 1 from here.

Los 2 bots anotan caja 1 como vista, para no volver a hablar de ella, hasta que la necesiten.

020. Jason: I need a gravity gun to use it.

Jason no tiene arma de gravedad para poder levantar la caja, lo avisa al otro jugador.

021. Jason: Ok, i finished investigating the area i'm in currently.

George anota que Jason no tiene tareas. En este momento Jason termina la tarea actual: investigar área y busca nuevas tareas para hacer. Nunca vió área-objetivo, tampoco tiene áreas nuevas para investigar y nunca vió una pared para escalar. Pero hay un botón sin investigar, el botón 8. Como Jason no está en el área del botón, trata de armar un plan para que alguien colaborativo esté cerca del botón. Al correr el planificador FF, éste termina con un plan válido: plan vacío, ya que George está en el área del botón.

022. Jason: I am stuck.

Jason dice que no tiene nada para hacer, para que otro bot tome la iniciativa de apretar el botón 8.

7.1.2. Tarea de investigar qué puertas abre un botón

023. George: Let's try figuring out what doors closes button 8.

George propone apretar el botón 8. Queda en espera de respuestas de bots colaborativos: en este caso queda esperando a Jason.

024. Jason: Good.

Jason responde afirmativamente. George ahora puede presionar el botón, ya que todos los bots colaborativos respondieron afirmativamente. Corre hacia el botón para presionarlo.

025. George: I will press button 8.

George llegó al botón, y antes de presionarlo avisa a otros jugadores de que va a hacerlo. Los dos bots ahora saben que el botón 8 fue presionado, empiezan a recorrer puertas vistas anteriormente y a las que pueden llegar, para ver si su estado cambió. Hay que destacar que si el bot sabe que el botón 8 afecta a puerta X, entonces anota que la puerta X cambia su estado en el momento de presionar al botón o recibir el chat correspondiente. Además, no va a ir hasta la puerta para ver su estado, ya que sabe que cambió.

026. Jason: Door 5 isn't changed.

Jason llegó a la puerta 5, la cual no cambió.

027. Jason: So, button 8 doesn't toggle door 5.

Jason deduce que el botón 8 no afecta a la puerta 5. Lo “anota” en su estado. Lo mismo hace George al recibir el chat.

028. George: Door 3 stays closed.

029. George: So, button 8 isn't toggling door 3.

030. George: Door 2 did not change, it is still closed.

031. George: Button 8 isn't toggling door 2.

032. Jason: Door 6 is now opened.

Jason llegó a la puerta 6, la cual se abrió.

033. Jason: So, button 8 toggles door 6.

Jason deduce que el botón 8 cambia el estado de la puerta 6. Lo “anota” en su estado. Lo mismo hace George al recibir el chat.

Hasta ahora vimos como los bots investigan *nuevas áreas*, tomando nota de los *estados de las puertas*. Además vimos como los bots descubren qué puertas abre *un botón* al presionarlo. Ahora veremos que pasa cuando un bot encuentra un arma:

077. George: I found weapon physcannon.

George encontró el arma de gravedad, que sirve para poder mover cajas pesadas. Jason “anota” que George tiene arma de gravedad, para poder usar este hecho dentro de planificador.

7.1.3. Uso de planificador FF

A continuación vamos a mostrar como los bots usan al planificador para lograr un objetivo. George quiere saber si botón 2 abre puerta 4 (de aquí en mas lo denotamos como configuración botón2-puerta4). Ejecuta planificador FF, el objetivo es que exista un bot en el área del botón 2 y que exista un bot en una de las dos áreas de la puerta (ver sección 6.3.2). Output del planificador:

```
ff: found legal plan as follows
step      0: MOVE BOT2 AREA5 AREA9
          1: MOVE BOT1 AREA5 AREA1
          2: MOVE BOT2 AREA6 AREA5
          3: MOVE BOT1 AREA6 AREA5
          4: PUSH-BUTTON BOT1 BUTTON2 DOOR2 DOOR5 AREA6 AREA6 AREA2 AREA4 AREA1
          5: MOVE BOT1 AREA2 AREA6
          6: MOVE BOT1 AREA7 AREA2
          7: REACH-GOAL
```

El planificador terminó con un plan válido, George propone ejecutarlo.

179. George: Let's try to figuring out if button 2 closes door 4.

180. Jason: Yes.

Jason acepta participar en la tarea de ver si botón 2 afecta puerta 4. George ejecuta paso 0 del plan: acción MOVE, argumento 5 (area-button4), bot George. Como bot mismo es el ejecutor de paso del plan, corre hacia el área del botón 4.

181. George: I am in area-button4.

George ejecuta paso 1 del plan: acción MOVE, argumento 5 (area-button4), bot Jason. Como George no es el ejecutor de paso del plan, pide a Jason que vaya al área del botón 4.

182. George: Jason, could you go to area area-button4?

183. Jason: I'm in area-button4.

184. Jason: Done.

Hay que indicar aquí que el jefe (el que armó el plan) no infiere del chat si el paso actual del plan fue ejecutado. Solamente espera que el jugador-ejecutor le indique que terminó.

George ejecuta paso 2 del plan: acción MOVE, argumento 6 (area-button3), bot George. El mismo corre hacia el área del botón 3.

185. George: I'm at area-button3.

George ejecuta paso 3 del plan: acción MOVE, argumento 6 (area-button3), bot Jason.

186. George: Jason, please, move to area area-button3.

187. Jason: I am at area-button3.

188. Jason: Done.

George ejecuta paso 4 del plan: acción PUSH-BUTTON, argumento 2 (botón 2), bot Jason.

189. George: Jason, try to push button 3.

190. Jason: I will try to press button 3 now.

Jason presiona el botón 3, el cual afecta puertas 3 y 6.

191. Jason: I have finished.

Jason indica explícitamente a George que terminó. George ejecuta paso 5 del plan: acción MOVE, argumento 2 (area-respawn2), bot Jason.

192. George: Jason, now, walk to area area-respawn2.

193. Jason: I'm at area-respawn2.

George ejecuta paso 6 del plan: acción MOVE, argumento 7, bot Jason.

194. George: Jason, pass to area area-gravity-gun.

195. Jason: I am in area-gravity-gun.

196. Jason: I'm done.

Ahora los dos bots se dan cuenta que se cumplió el objetivo del plan. Jason está en el área del botón 2 y George está en el área de la puerta 4.

197. Jason: I will try to push button 2.

Botón 2 afecta puertas 3 y 10. George corre hacia puerta 4 para ver su estado.

198. George: Door 4 hasn't changed.

199. George: Button 2 is not affecting door 4.

Deduce que el botón 2 no afecta puerta 4. Los dos bots lo anotan para no volver a probar esta configuración.

7.1.4. Usando caja

Vamos a ver ahora como se usan las cajas dentro del juego. Jason usa planificador: llevar la caja 1 al área area-wall1:

```
ff: found legal plan as follows
step      0: PUSH-BUTTON BOT2 BUTTON8 DOOR1 DOOR4 AREA9 AREA2 AREA7 AREA5 AREA1
          1: MOVE BOT2 AREA5 AREA9
          2: MOVE BOT2 AREA1 AREA5
          3: CARRY-BOX BOT2 BOX0 AREA1 GRAVITY-GUN
          4: MOVE BOT2 AREA5 AREA1
          5: MOVE BOT2 AREA9 AREA5
          6: MOVE BOT2 AREA10 AREA9
          7: DROP-BOX BOT2 BOX0 AREA10
```

248. Jason: Let's try to take box to area-wall1.

249. George: I agree.

Jason ejecuta paso 0 del plan: acción PUSH-BUTTON, argumento 8 (botón 9, afecta puertas 2 y 5), bot George.

250. Jason: George, try to press button 9.

251. George: I will try to press button 9 now.

252. George: Ok, done.

Jason ejecuta paso 1 del plan: acción MOVE, argumento 5, bot George.

253. Jason: George, could you move to area area-button4?

254. George: I am at area-button4.

255. George: Ok, i have finished.

Jason ejecuta paso 2 del plan: acción MOVE, argumento 1, bot George.

256. Jason: George, could you pass to area area-respawn1?

257. George: I am in area-respawn1.

258. George: Ok, done.

Jason ejecuta paso 3 del plan: acción CARRY-BOX, argumento 0, bot George.

259. Jason: George, try to take the box 1 now.

260. George: I will take the box 1.

261. George: Ok, done.

George usa el arma de gravedad para levantar la caja (ver figura 7.3). Jason ejecuta paso 4 del plan: acción MOVE, argumento 5, bot George.



Figura 7.3: Bot usando arma de gravedad con la caja 1.

262. Jason: George, could you walk to area area-button4?

La puerta es estrecha y pasando por la puerta, George golpea la caja contra la puerta, por eso la caja se cae al piso.

263. George: I'm in area-button4.

George se da cuenta de que perdió la caja, trata de volver y levantarla de nuevo:

264. George: I am at area-respawn1.

Esta vez George pasa por la puerta sin problemas.

271. George: I am in area-button4.

272. George: I finished.

Jason ejecuta paso 5 del plan: acción MOVE, argumento 9, bot George.

273. Jason: George, could you walk to area area-shoot-button?

274. George: I am in area-shoot-button.

275. George: Ok, i have finished.

George golpea a la caja contra la pared, la caja se le cae encima. Piensa que alguien le disparo, porque perdió puntos de vida.

276. George: Hey, stop it.

Jason ejecuta paso 6 del plan: acción MOVE, argumento 10, bot George.

277. Jason: George, pass to area area-wall1.

278. George: I'm in area-wall1.

279. George: Ok, i'm done.

Jason ejecuta paso 7 del plan: acción DROP-BOX, argumento 0, bot George.

280. Jason: George, please, release box 1 near the wall.

281. George: I will try to put the box 1 on the ground.

282. George: I have finished.

Los bots se dan cuenta que se cumplió el objetivo del plan. Jason no tiene nada para hacer y George investiga el área detrás de la pared.

283. Jason: I'm stuck.

284. George: I will investigate new area.

7.1.5. Resultados de la ejecución

La ejecución completa está en el apéndice .E. En la tabla 7.4 se puede ver que descubrieron los bots. ✓ indica que el botón afecta a la puerta, ✗ indica que el botón *no* afecta a la puerta y espacio vacío indica que los bots no saben si afecta o no.

Puerta	1	2	3	4	5	6	7	8	9	10
Botón 1										
Botón 2	✗	✗	✓	✗	✗	✗	✗	✗		✓
Botón 3			✓	✗	✗	✓	✗			
Botón 4		✓	✗	✗	✗	✗	✗	✓		
Botón 5	✗	✓	✓	✗	✗	✗				
Botón 6										
Botón 7										
Botón 8	✗	✗	✗	✓	✗	✓	✗			
Botón 9	✗	✓	✗	✗	✓	✗	✗			✗
Botón 10										

Figura 7.4: Lo que saben los bots al final de ejecución 1.

El plano del mapa está en la página 64. Como se ve en la figura 7.2 (configuración de los botones, página 65) ningún botón abre puerta 1. Entonces el botón 1 no puede ser alcanzado, en la tabla 7.4 se ve que los bots no saben nada del mismo. El botón 6 tampoco puede ser alcanzado, ya que el único botón que abre la puerta 7 es el botón 7, pero está detrás de la puerta 9 (que nunca fue vista por los bots). A la puerta 9 la afectan los botones 1 y 10, pero el botón 1 es inalcanzable y para dispararlo al botón 10 hay que tener el arma *crossbow*, el cual está en el área del botón 6 (inalcanzable).

Notar que para el resto los botones, los bots descubrieron que cualquiera de ellos abre 2 puertas, máxima cantidad de puertas para un botón. Por esta razón no siguen investigando, por ejemplo configuración botón3-puerta1.

En la próxima ejecución veremos como un humano miente al bot, y cual es el resultado.

7.2. Ejecución 2, con un humano mentiroso

En esta ejecución vamos a ver como un jugador humano (Boris) miente al bot (Mary), esto hace que el bot arme planes inválidos. Daremos el chat completo, y vamos a ver como falla la ejecución de plan dos veces gracias a que el bot tiene creencias incorrectas sobre el entorno.

Puerta	1	2	3	4	5	6	7	8	9	10
Botón 1	✓					✓				
Botón 2		✓		✓						
Botón 3							✓			
Botón 4			✓					✓		
Botón 5	✓								✓	
Botón 6							✓			✓
Botón 7	✓							✓		
Botón 8		✓			✓					
Botón 9						✓		✓		
Botón 10	✓									✓

Figura 7.5: Configuración 2 de los botones.

Se usó la configuración 2 de los botones que está en la figura 7.5.

001. Mary: Good morning, Boris.
 002. Mary: I am in new area, let's call it area-respawn1.
 003. Boris: hi Mary
 004. Boris: i am at area-respawn2
 005. Mary: I will investigate new area.

 006. Boris: i see closed door 1
 007. Boris: i see closed door 2
 008. Boris: i see closed door 3
 009. Boris: i see button 8

 010. Mary: I am seeing door 6, it is closed.
 011. Mary: I am seeing closed door 7 from here.
 012. Mary: I see a box 1.
 013. Mary: I need a gravity gun to use it.
 014. Mary: I can see door 5, it is closed.
 015. Mary: I finished investigating the area i'm in currently.

 016. Boris: lets try button 8

 Boris propone apretar el botón 8.

 017. Mary: Yes.
 018. Boris: i will press button 8

 Botón 8 abre puertas 2 y 5, y la puerta 5 está en el área de Mary. Pero Boris mintió, dijo que iba a presionar al botón, pero no lo hizo. Mary no lo sabe. Empieza a recorrer las puertas de su área pensando que el botón fue presionado.

 019. Mary: Door 5 stays closed.
 020. Mary: Button 8 isn't changing state of door 5.

Mary dedujo erróneamente que **(1) botón 8 no afecta a la puerta 5**, porque Boris mintió. De acá en más vamos a marcar las creencias falsas del bot con números entre paréntesis.

021. Mary: Door 6 didn't change, it is still closed.

022. Mary: So, button 8 is not toggling door 6.

023. Mary: Door 7 is closed.

024. Mary: Button 8 doesn't close door 7.

025. Boris: door 1 stays closed

026. Boris: button 8 doesn't affect door 1

027. Boris: door 2 stays closed

028. Boris: button 8 doesn't affect door 2

Otra mentira. Botón 8 abre puerta 2 pero Mary no lo sabe. **(2) Botón 8 no afecta a la puerta 2.**

029. Boris: door 3 stays closed

030. Boris: button 8 doesn't affect door 3

031. Mary: Nice.

032. Boris: lets try button 8

Ahora Boris realmente quiere probar botón 8. Mary acepta.

033. Mary: Affirmative.

034. Boris: i will press button 8

Esta vez Boris realmente apretó botón 8. Pero Mary no va a recorrer las puertas esta vez, porque erróneamente cree que el botón 8 no las afecta, ya que había recorrido las puertas anteriormente.

035. Mary: I don't know what can i do.

Mary no está encerrada en su área, la puerta 5 está abierta, pero Mary no lo sabe.

036. Boris: door 2 is opened

037. Boris: button 8 affects door 2

038. Mary: Good.

La acción de mentir se puede corregir, aquí se ve que Mary aceptó el hecho de que botón 8 abre puerta 2. **(2)** quedó anulado. Mary sigue creyendo **(1)** erróneamente.

039. Boris: i am at area-gravity-gun

Boris entró al área detrás de la puerta 2.

040. Boris: i found weapon physcannon

041. Boris: i see button 2

042. Boris: i will press button 2

Boris mintió, no apretó el botón 2. Mary no tiene nada para hacer, entonces empieza recorriendo las puertas para ver si el botón 2 las afecta. Como no las recorrió cuando botón 8 realmente fue apretado, no sabe que la puerta 5 está abierta...

043. Mary: Door 7 has not changed.
 044. Mary: So, button 2 is not affecting door 7.
 045. Mary: Door 6 did not change, it is still closed.
 046. Mary: So, button 2 doesn't open door 6.
 047. Mary: Door 5 is now opened.
 048. Mary: So, button 2 opens door 5.

Deducción errónea. Ahora el bot piensa que:

- **(1) el botón 8 no afecta a la puerta 5.**
- **(3) el botón 2 abre puerta 5.**

La verdad es que es exactamente al revés.

049. Mary: I will explore new area.
 050. Mary: I can see a button 4.
 051. Mary: I can see closed door 8 over here.
 052. Mary: I am seeing door 4, it is closed.
 053. Mary: I investigated.
 054. Mary: Let's find which door closes button 4.
 055. Boris: ok

056. Mary: I will try to press button 4 now.

Botón 4 afecta puertas 3 y 8. Los dos jugadores empiezan a investigar puertas cercanas.

057. Boris: door 3 is opened
 058. Mary: Okey.

Mary infiere del chat que el botón 4 abre puerta 3, ya que la tarea es investigar el botón.

059. Mary: Door 8 is now opened.
 060. Mary: So, button 4 toggles door 8.

La investigación del botón 4 está terminada, ya que los jugadores saben que abre 2 puertas distintas. Detrás de la puerta 8 hay un área nuevo. Mary decide explorarlo.

061. Mary: I will explore new area.
 062. Mary: I'm in new area, let's call it area-shoot-button.
 063. Mary: I see closed door 10 over here.
 064. Mary: I can see button 10 over here.
 065. Mary: I can't shoot it.
 066. Mary: I can see a button 9 from here.
 067. Mary: Ok, i finished exploring the area i'm in currently.

El bot decide investigar lo que hace el botón 9, el cual afecta puertas 6 y 8.

068. Mary: Let's try to find what doors opens button 9.

069. Boris: ok

070. Mary: I will push button 9 now.

Los jugadores investigan puertas.

071. Mary: Door 8 is now closed.

072. Mary: So, button 9 opens door 8.

073. Mary: Door 10 stays closed.

074. Mary: So, button 9 does not change state of door 10.

Mary termina de investigar las puertas. Quiere saber si botón 9 afecta puerta 3 (Boris está cerca).

075. Mary: Let's try to figure out if button 9 open door 3.

076. Boris: button 9 doesn't affect door 1

077. Boris: button 9 doesn't affect door 2

078. Boris: button 9 doesn't affect door 3

079. Mary: Yes.

080. Boris: button 9 doesn't affect door 4

Mary prueba configuración botón 2 - puerta 4. Ejecuta FF:

```
ff: found legal plan as follows
step      0: PUSH-BUTTON BOT1 BUTTON8 DOOR7 DOOR7 AREA-SHOOT-BUTTON AREA-SHOOT-BUTTON AREA-BUTTON4
          AREA-SHOOT-BUTTON AREA-BUTTON4
          1: MOVE BOTO AREA-RESPAWN2 AREA-BUTTON3
          2: MOVE BOT1 AREA-BUTTON4 AREA-SHOOT-BUTTON
          3: MOVE BOTO AREA-GRAVITY-GUN AREA-RESPAWN2
          4: REACH-GOAL
```

081. Mary: Lets try figuring out if button 2 toggles door 4.

082. Boris: ok

Mary ejecuta paso 0, acción PUSH-BUTTON, argumento 8 (botón 9), ejecutor Mary.

083. Mary: I will try to push button 9 now.

Botón afecta puertas 6 y 8. Mary lo hizo para poder salir de su área. Ejecuta paso 1, acción MOVE, argumento 2, ejecutor Boris.

084. Mary: Boris, please, pass to area area-respawn2.

085. Boris: button 2 doesn't affect door 4

086. Mary: Okey, cool.

Boris decide cortar la ejecución del plan y mentirle al bot diciendo que el botón 2 no afecta puerta 4. Bot piensa que **(4) el botón 2 no afecta puerta 4**. Prueba configuración botón 2 - puerta 8.

```
ff: found legal plan as follows
step      0: MOVE BOTO AREA-RESPAWN2 AREA-BUTTON3
          1: MOVE BOTO AREA-GRAVITY-GUN AREA-RESPAWN2
          2: REACH-GOAL
```

087. Mary: Lets try to check if button 2 closes door 8.

088. Boris: button 2 doesn't affect door 8

089. Mary: Good.

Boris está diciendo la verdad ahora. Mary prueba configuración botón 2 - puerta 10.

```
ff: found legal plan as follows
step      0: MOVE BOTO AREA-RESPAWN2 AREA-BUTTON3
          1: MOVE BOTO AREA-GRAVITY-GUN AREA-RESPAWN2
```

090. Mary: Lets try to see if button 2 closes door 10.

091. Boris: button 2 doesn't affect door 10

092. Mary: Affirmative.

093. Boris: lets try button 3

094. Mary: Sure.

095. Boris: i will pree button 3

Boris hizo error sintáctico en la oración (“pree” en vez de “press”), pero el bot lo entendió. Sin embargo, Boris en vez de apretar el botón 3 (afecta puerta 7) aprieta el botón 2 (afecta puertas 2 y 4).

096. Mary: Door 8 did not change, it is still opened.

097. Mary: So, button 3 is not toggling door 8.

098. Boris: i am at area-gravity-gun

099. Mary: I am in area-button4.

100. Mary: Door 4 is now opened.

101. Mary: Button 3 closes door 4.

El bot, pensando que botón 3 fue apretado, deduce equivocadamente que **(5) el botón 3 afecta puerta 4.**

102. Mary: Door 5 stays opened.

103. Mary: So, button 3 is not changing state of door 5.

104. Boris: door 2 is closed

El bot, pensando que botón 3 fue apretado, infiere de esta oración que **(6) el botón 3 afecta puerta 2.** Las creencias incorrectas que tiene el bot son:

- **(1) el botón 8 no afecta a la puerta 5.**
- **(3) el botón 2 afecta puerta 5.**
- **(4) el botón 2 no afecta puerta 4.**
- **(5) el botón 3 afecta puerta 4.**
- **(6) el botón 3 afecta puerta 2.**

Mary decide probar configuración botón 8 - puerta 4.

```
ff: found legal plan as follows
step      0: MOVE BOT1 AREA-BUTTON3 AREA-BUTTON4
          1: PUSH-BUTTON BOT1 BUTTON2 DOOR1 DOOR3 AREA-BUTTON3 AREA-RESPAWN2 AREA-GRAVITY-GUN
              AREA-BUTTON4 AREA-BUTTON3
          2: MOVE BOTO AREA-RESPAWN2 AREA-GRAVITY-GUN
          3: REACH-GOAL
```

105. Mary: Lets try to figure out if button 8 toggle door 4.

106. Boris: ok

Mary ejecuta paso 0, acción MOVE, argumento 6, ejecutor Mary. Corre hacia el área del botón 3.

107. Mary: I am in area-button3.

Mary ejecuta paso 1, acción PUSH-BUTTON, argumento 2 (botón 3, afecta puerta 7), ejecutor Mary.

108. Mary: I will push button 3 now.

Mary ejecuta paso 2, acción MOVE, argumento 2, ejecutor Boris. En este momento el estado del juego es el que se muestra en la figura 7.6.

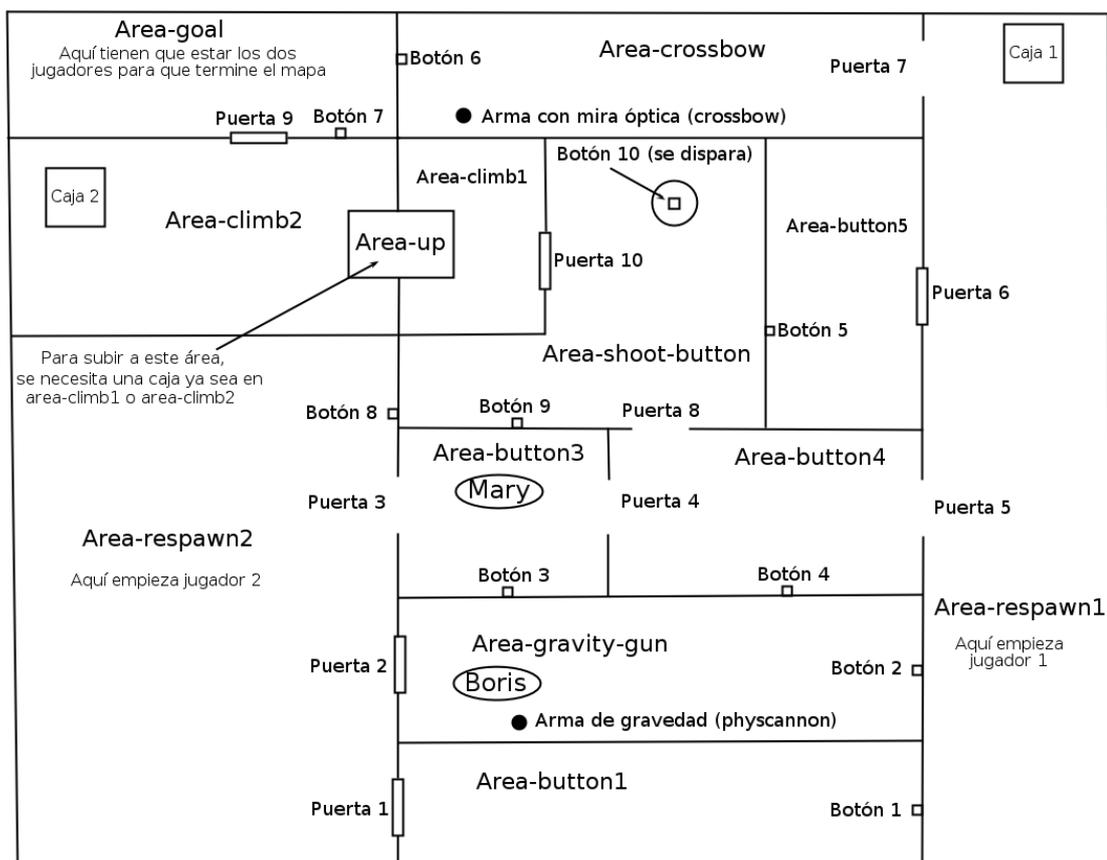


Figura 7.6: Ejecución 2, con un humano que miente.

109. Mary: Boris, please, walk to area area-respawn2.

110. Boris: door 2 is closed

111. Boris: i can't pass to area-respawn2

112. Mary: Sure.

Ejecución del plan falla por la primera vez, gracias a que el bot tiene creencias inválidas.

Mary prueba configuración botón 8 - puerta 8.

```
ff: found legal plan as follows
step      0: PUSH-BUTTON BOT1 BUTTON2 DOOR1 DOOR3 AREA-BUTTON3 AREA-RESPAWN2 AREA-GRAVITY-GUN
           AREA-BUTTON4 AREA-BUTTON3
          1: MOVE BOT1 AREA-BUTTON4 AREA-BUTTON3
          2: MOVE BOTO AREA-RESPAWN2 AREA-GRAVITY-GUN
          3: REACH-GOAL
```

113. Mary: Lets try to figure out if button 8 open door 8.

114. Boris: ok

Mary ejecuta paso 0, acción PUSH-BUTTON, argumento 2 (botón 3, afecta puerta 7), ejecutor Mary.

115. Mary: I will try to push button 3.

Mary ejecuta paso 1, acción MOVE, argumento 5 (area-button4), ejecutor Mary. Corre hacia el área del botón 4, pero en este momento Boris aprieta el botón 2 para cerrar la puerta 4 y confundir al bot.

116. Mary: Door 4 is now closed.

117. Mary: I can't move to area area-button4.

El bot se da cuenta de que la ejecución del plan falló otra vez. Prueba configuración botón 8 - puerta 10.

```
ff: found legal plan as follows
step      0: PUSH-BUTTON BOT1 BUTTON2 DOOR1 DOOR3 AREA-BUTTON3 AREA-RESPAWN2 AREA-GRAVITY-GUN
           AREA-BUTTON4 AREA-BUTTON3
          1: MOVE BOT1 AREA-RESPAWN2 AREA-BUTTON3
          2: PUSH-BUTTON BOT1 BUTTON7 DOOR1 DOOR1 AREA-RESPAWN2 AREA-RESPAWN2 AREA-GRAVITY-GUN
AREA-GRAVITY-GUN
          3: MOVE BOT1 AREA-BUTTON3 AREA-RESPAWN2
          4: MOVE BOTO AREA-RESPAWN2 AREA-GRAVITY-GUN
          5: MOVE BOT1 AREA-BUTTON4 AREA-BUTTON3
          6: MOVE BOT1 AREA-SHOOT-BUTTON AREA-BUTTON4
          7: REACH-GOAL
```

118. Mary: Lets try to see if button 8 closes door 10.

119. Boris: ok

Mary ejecuta paso 0, acción PUSH-BUTTON, argumento 2 (botón 3, abre puerta 7), ejecutor Mary.

120. Mary: I will try to push button 3.

Mary ejecuta paso 1, acción MOVE, argumento 2, ejecutor Mary. Corre hacia el área area-respawn2.

121. Mary: I am at area-respawn2.

122. Mary: Door 2 is now opened.

Pasando cerca de la puerta 2 se da cuenta que su estado es distinto al que tenía anotado. Mary ejecuta paso 2, acción PUSH-BUTTON, argumento 7 (botón 8, afecta puertas 2 y 5), ejecutor Mary.

123. Mary: I will try to push button 8.

Mary ejecuta paso 3, acción MOVE, argumento 6, ejecutor Mary.

124. Mary: I'm at area-button3.

Mary ejecuta paso 4, acción MOVE, argumento 2, ejecutor Boris.

125. Mary: Boris, now, move to area area-respawn2.

Boris corre hacia el área area-respawn2, pero encuentra puerta cerrada en el medio.

126. Boris: door 2 is closed

127. Boris: i can't pass to area-respawn2

Acabamos de ver como **ejecución de un plan falla por la segunda vez** a causa de creencias incorrectas. No vamos a seguir con esta ejecución, ya que se entiende que mentir no ayuda a pasar el nivel jugado.

7.3. Ejecución 3, un humano colaborando con un bot

En esta ejecución hay un bot (John) y un humano (Romina). Se usó la configuración 2 de los botones (ver página 72). Los jugadores pudieron pasar el mapa, es decir llegaron al área-objetivo. Gracias a esta ejecución encontramos algunas limitaciones del sistema de chat. Una forma de corregirlas se describe en profundidad en el capítulo 8.

Aquí daremos solamente partes del chat importantes. El diálogo completo está en el apéndice .F.

7.3.1. Expresiones que no están en el sistema

Vamos a ver ahora algunos ejemplos de expresiones que el sistema de chat no reconoce.

016. Romina: there is a button here, it's number 8

017. John: I can't understand that.

Aquí vemos que el bot no entendió la expresión de Romina ya que tiene vocabulario bastante estricto. Romina se corrige:

018. Romina: i see button 8

Vamos a ver más ejemplos de expresiones que el bot no comprende:

040. Romina: let's see what the button 4 does

041. John: Sorry, Romina i can't understand you.

Romina quiere probar el botón 4, del cual habló el bot anteriormente. Sin embargo la frase no dió ningún match y el bot no la entiende.

042. Romina: push button 4

Esta vez bot entiende el chat, pero lo omite, porque no sabe a quién va dirigido. Para que bot siga las instrucciones hace falta usar su nombre. Romina cede y empieza a investigar nueva área... Después vuelve a intentar.

047. Romina: let's try button 4

048. John: Oh my.

Puede parecer un poco raro pero esta frase indica afirmación. Bot finalmente entendió que es lo que quería Romina.

072. Romina: door 3 is open

073. John: I couldn't understand.

El sistema espera palabra "opened", valor válido para la variable $\$door_status$ en la oración (ver sección 4.2). Otro ejemplo:

077. Romina: button 4 doesn not opens door 1

El bot acepta el chat, pero un error de sintaxis hace que el bot piensa que el botón 4 afecta la puerta 1 (ver página 30).

7.3.2. Cancelación de tareas

A continuación describiremos como un humano puede cancelar la tarea colaborativa que está realizando un bot. John prueba configuración botón3 - puerta5:

```
ff: found legal plan as follows
step      0: PUSH-BUTTON BOT1 BUTTON4 DOORO DOORO AREA-BUTTON5 AREA-BUTTON1
          AREA-RESPAWN2 AREA-BUTTON1 AREA-RESPAWN2
          1: MOVE BOTO AREA-BUTTON1 AREA-RESPAWN2
          2: PUSH-BUTTON BOTO BUTTON0 DOOR5 DOOR5 AREA-BUTTON1 AREA-BUTTON5 AREA-RESPAWN1
          AREA-BUTTON5 AREA-RESPAWN1
          3: MOVE BOTO AREA-RESPAWN2 AREA-BUTTON1
          4: MOVE BOT1 AREA-RESPAWN1 AREA-BUTTON5
          5: MOVE BOTO AREA-BUTTON3 AREA-RESPAWN2
          6: REACH-GOAL
```

249. John: Let's try to check if button 3 toggles door 5.

250. Romina: ok

John: paso 0, acción PUSH-BUTTON, argumento 4, ejecutor John.

251. John: I will try to push button 5 now.

El botón 5 afecta puertas 1 y 9. John: paso 1, acción MOVE, argumento 8, ejecutor Romina.

252. John: Romina, now, pass to area area-button1.

Romina camina hasta el área del botón 1.

253. Romina: i am in area-button1

254. Romina: done

John: paso 2, acción PUSH-BUTTON, argumento 0, ejecutor Romina.

255. John: Romina, try to push button 1.

256. Romina: i will press button 1

El botón 1 afecta puertas 1 y 6. Romina, al presionar el botón, queda encerrada en el área del botón 1 y esto no le agrada. Quiere cancelar la investigación del botón 3, para poder apretar el botón 1 y así salir del área del botón 1:

257. Romina: wait

Esto hace que el bot cancela la tarea actual y espera al jugador. No hacía falta decirle al bot que espere, Romina solamente podía decir que va a apretar el botón 1, el bot detectaría que esto interfiere con el plan y cancelaría la ejecución del mismo.

7.3.3. Llegando al objetivo

Vamos a ver como el bot arma el plan para salir del laberinto. Casi todos los botones están investigados, Romina pudo llegar al área objetivo, pero para que el bot pueda llegar allí, necesita apretar algunos botones para abrir puertas en el medio. John arranca el planificador, el output del mismo es el siguiente:

```
ff: found legal plan as follows
step 0: MOVE BOT1 AREA-BUTTON3 AREA-RESPAWN2
      1: MOVE BOT1 AREA-BUTTON4 AREA-BUTTON3
      2: PUSH-BUTTON BOTO BUTTON6 DOORO DOOR7 AREA-GOAL AREA-BUTTON1 AREA-RESPAWN2
          AREA-SHOOT-BUTTON AREA-BUTTON4
      3: MOVE BOT1 AREA-SHOOT-BUTTON AREA-BUTTON4
      4: SHOOT-BUTTON BOT1 BUTTON9 DOORO DOOR9 CROSSBOW AREA-SHOOT-BUTTON AREA-BUTTON1
          AREA-RESPAWN2 AREA-WALL1 AREA-SHOOT-BUTTON
      5: MOVE BOT1 AREA-WALL1 AREA-SHOOT-BUTTON
      6: CLIMB-BOX BOT1 BOXO AREA-WALL1 AREA-HIGH
      7: FALL BOT1 AREA-WALL2 AREA-HIGH
      8: MOVE BOT1 AREA-GOAL AREA-WALL2
```

500. John: I found a plan to reach the goal!

501. Romina: good

502. Romina: it was a time :)

503. John: Sorry, Romina i didn't understand.

Claramente bot carece de sentido de humor. John: paso 0, acción MOVE, argumento 6, ejecutor John. Corre hacia el área del botón 3.

504. John: I'm at area-button3.

John: paso 1, acción MOVE, argumento 5, ejecutor John.

505. John: I'm at area-button4.

John: paso 2, acción PUSH-BUTTON, argumento 6, ejecutor Romina.

506. John: Romina, push button 7 now.

507. Romina: i will press button 7

508. Romina: done

Botón 7 afecta puertas 1 y 8. John: paso 3, acción MOVE, argumento 9, ejecutor John.

509. John: I am at area-shoot-button.

John: paso 4, acción SHOOT-BUTTON, argumento 9, ejecutor John. Usa el arma con la mira óptica para disparar al botón 10 (ver figura 7.7). El bot necesita abrir la puerta 10 para pasar al área de la pared.



Figura 7.7: Bot apuntando y disparando al botón 10. La puerta 10 está abriéndose.

510. John: I will try to shoot button 10.

El bot dispara botón 10, el cual abre puertas 1 y 10. John: paso 5, acción MOVE, argumento 10, ejecutor John.

511. John: I'm in area-wall1.

John: paso 6, acción CLIMB-BOX, argumento 0, ejecutor John. El bot sube a la caja y así sube la pared .

512. John: I am in new area, call it area-high.

John: paso 7, acción FALL, argumento 12, ejecutor John. La acción FALL es para tirarse de una pared. El bot se tira:

513. John: I'm in area-wall2.

John: paso 8, acción MOVE, argumento 13, ejecutor John. El bot pasa al área objetivo.

514. John: I am in area-goal.

Los dos jugadores llegaron al área-objetivo y así pasan el nivel.

Capítulo 8

Conclusiones y trabajo futuro

Vamos a hablar un poco sobre los objetivos logrados en esta tesis y sobre las mejoras que podrían realizarse en un trabajo futuro.

8.1. Conclusiones

En esta tesis se realizó el estudio de planificación de tareas, para esto se implementó un plugin para creación y manejo de bots colaborativos en un mod del juego Half-Life 2: Deathmatch. El plugin usa técnicas de waypoints para que los bots puedan navegar por el mapa jugado. El plugin también hace uso de un módulo de chat para que los bots puedan comunicarse entre sí y con los humanos, el cual resuelve problemas como por ejemplo cuando una persona o un bot se quedan sin movimientos a realizar. Se investigaron técnicas de planificación eficientes para planificación y coordinación de jugadores colaborativos, lo cual es necesario para poder llegar al objetivo.

Habiendo varias técnicas de navegación por el mapa elegimos usar waypoints, que es una técnica simple, pero potente. Fue interesante descubrir y analizar distintos tipos que puede tener un waypoint o una conexión entre waypoints para poder pasar exitosamente un segmento del camino. Saltos, escaleras, aceleración de paso, necesidad de agacharse, uso de linterna, uso de cajas para llegar a un lugar alto, uso de armas de distinto tipo, todo esto constituyó un desafío particular en esta tesis.

Aunque en este trabajo no se habló de técnicas tridimensionales que usan los bots: como calcular adonde mirar en el frame siguiente, suavidad en cambiar el ángulo, detección de colisiones y detección de visibilidad, fue muy desafiante aprender las técnicas que se usan actualmente para poder armar un dominio de planificación interesante.

N-gramas resultaron ser muy útiles en identificar la semántica de la oración. Vimos muchos ejemplos de procesamiento de las oraciones durante las ejecuciones y, aunque el sistema tiene sus puntos débiles, sirvió con éxito a su propósito. Un

usuario sin experiencia previa en este estilo de juegos pudo comunicarse con el bot sin problemas, parafraseando la oración cuando el bot no lo entendía.

Hoy en día hay mucha discusión si usar o no planificadores en los juegos. Aunque podría parecer que aplicar un planificador STRIPS dentro de un videojuego de tiempo real requiere hacer muchos trucos de ingeniería y optimizaciones de código, sin embargo, el problema principal parece ser que los desarrolladores encuentran los planificadores difíciles de aplicar en un cierto dominio. Al integrar un planificador dentro de un juego en tiempo real nosotros también lo demostramos. El planificador FF resultó ser muy eficiente en el dominio aplicado, el tiempo en el cálculo de planes no superaba 1-2 segundos.

Creamos una metodología híbrida que permite al bot usar la información incompleta sobre el dominio para armar planes y así poder salir del laberinto. Para esto, usamos un planificador estático en un entorno dinámico. Vimos casos de replanificación cuando un plan fallaba, el bot siempre pensaba en probar otras opciones, si es que había.

Se realizaron varias pruebas para evaluar el sistema. Encontramos que el sistema responde como era de esperarse en la mayoría de los casos. En primer experimento vimos que los bots descubren todo lo que pueden del mapa jugado, ya que el laberinto no tiene salida. En segundo experimento vimos como bot encontraba evidencias de que la persona que supuestamente colaboraba con el, le mentía. Y en el tercer experimento el bot ayudó a la persona salir del laberinto armando planes para poder abrir puertas en el medio.

8.2. Trabajo futuro

Casos aislados mostraron que el sistema tiene sus fallos que podrían ser arreglados con algunas mejoras. A continuación vamos a ver qué mejoras se podrían realizar en un trabajo futuro:

- Se podría reemplazar la técnica de waypoints por otra técnica llamada **mallas de navegación** (*navegation mesh* en inglés), que se basa en detectar áreas del mapa por las cuales se puede caminar y las conexiones entre las mismas. Source SDK ofrece generación automática de estas mallas. La única mejora aquí es que el bot podrá saber donde puede pisar y donde no puede.
- En este trabajo tenemos *sólo dos niveles de importancia de palabras* en el identificador de oraciones: requerida u opcional. Podría ser mejorado dando diferente nivel de importancia a cada palabra, como ser que la *importancia dependa de la función sintáctica* de la palabra en la oración. Por ejemplo, darle mucha importancia a los verbos y a los sustantivos, menos importancia a los adjetivos y poca importancia a los artículos.
- Agregar un *controlador de sintaxis*, ya que cuando la oración tiene errores ortográficos el sistema puede darle semántica equivocada a la oración. Vimos ejemplos de esto en el capítulo 4. Sin embargo, esto tal vez se resolvería con implementar el inciso anterior.

- En la figura 7.1 (página 64) se ve que hay dos cajas en el juego. Al empezar los experimentos había una sola, la caja 1. El bot, al traer la caja a la pared, se saltaba la pared y se quedaba en el área `area-climb2` sin poder salir. En cambio, un ser humano llevaría la caja con él para poder volver. Tales acciones reciben nombre de “acciones irreversibles” y se podrían investigar los métodos existentes para poder identificarlas y evitarlas.
- Mejorar *el comportamiento del bot*. Esto incluye:
 - ◇ Que el bot indique qué está haciendo cuando está ejecutando un paso del plan. Puede ser que el bot tenga que ejecutar varios pasos del plan en los cuales él es el ejecutor, entonces el otro jugador queda esperando sin saber que está pasando.
 - ◇ Una mejora importante para poder salir del laberinto sería que el bot averigüe el estado de las puertas antes de que se presione un botón. En el apéndice .F se ve que el bot saca conclusiones erróneas gracias a esto.
 - ◇ El plan obtenido para probar si el botón X afecta a la puerta Y puede tener longitud 0. Esto pasa cuando el otro jugador ya está en el área del botón. El bot quedará esperando a que el jugador proponga apretar el botón, lo cual puede ser confuso para un jugador humano. La mejora en este caso es que el bot le indique al jugador de que hay una configuración botón-puerta para probar.
 - ◇ La detección del cumplimiento de la orden que dió el bot podría ser mejorada. En los experimentos se ve que el jugador tiene que avisarle explícitamente al bot que terminó, cuando en realidad esto podría ser inferido de las oraciones anteriores.
- Sería interesante ver un experimento que incluya *más de dos jugadores*, para ver como esto podría acortar el tiempo para poder salir del laberinto. En nuestro caso no se pudo realizar, ya que los bots se confunden en el momento de identificar a quien fue dirigida la oración. En este caso se podría investigar sobre las técnicas existentes para identificar al destinatario de la oración.
- Para lo último se podría investigar sobre planificadores que pueden usar información incompleta sobre el dominio en armado de planes y ver como adaptar alguno de ellos para usar dentro del juego. Un ejemplo de tal planificador es PKS [Pet04].

Apéndices

.A. Configuración de Botrix

A continuación se muestra el archivo de configuración del plugin para el mod de bots colaborativos, **BorzhMod**. Contiene información sobre nombres de bots, modelos de los jugadores, directorios, objetos, armas y chat. Es bastante autoexplicativo, además tiene comentarios. El archivo de configuración para el mod **Half Life 2: Deathmatch** es bastante parecido, así que no se incluye aquí.

```

;*****
; Este archivo de configuración se lee cuando se carga plugin Botrix (BOT's+matRiX).
;
; Atención: si usa el caracter de escape '\ ' para separar la cadena
; en línea múltiples, tenga cuidado de tener el fin de línea inmediatamente después
; del caracter '\ ', sino va a tener errores de parseo.
;*****

;-----
; Sección de bots.
;-----

[Bots]
names = John, Bob, Bart, James, Charles, Michael, Robert, Daniel, Paul, Mark, George, \
       Steven, Kevin, Jason, Brian, Thomas, Mary, Barbara, Susan, Lisa, Betty, Helen

; Mostrar la inteligencia del bot en su nick.
name_with_intelligence = off

;-----
; Acceso del usuario. Da al usuario la posibilidad de ejecutar comandos de consola de bots.
; El formato es <STEAM ID> = <nivel de acceso, separado por espacios>.
; Niveles de acceso:
; - waypoint: da acceso a mostrar/modificar waypoints.
; - bot: da acceso a agregar/remover/testear bots, mostrar el camino actual y objetos cercanos del bot.
; - config: acceso a modificar acceso de usuarios / mostrar entidades / mostrar eventos.
;-----

[User access]
STEAM_ID_LAN = waypoint bot config

;-----
; Secciones de configuración de mod. Cada sección tiene que terminar con .mod
; - games: posibles carpetas del juego (por ejemplo 'half-life 2 deathmatch').
; - mods: posibles carpetas de mods (normalmente están adentro de carpeta del juego, por ejemplo cs)
; - bot: tipo de bot a usar con el mod actual. Hay 2 tipos: hl2dm (para HL2:DM) y borzh (para BorzhMod).
; - teams: posibles equipos ordenados por su número como cuando se usa el comando de consola "jointeam".
;           'unassigned' (asignado automáticamente) y 'spectators' (espectadores) son requeridos, porque
;           son usados para determinar si un jugador es enemigo del bot.
; - models siguiendo por el nombre del equipo, distinto de 'unassigned' y 'spectators':
;           nombres de modelos para usar en ese equipo (separados por coma).
;-----

[BorzhMod.mod]
games = source SDK Base 2013 Multiplayer
mods = borzhmod
bot = borzh
teams = unassigned, spectators, combines, rebels
models combines = models/combine_soldier.mdl, \
                 models/combine_soldier_prisonguard.mdl, \
                 models/combine_super_soldier.mdl, \
                 models/police.mdl
models rebels = models/humans/Group03/male_01.mdl, \
                 models/humans/Group03/male_02.mdl, \
                 models/humans/Group03/male_03.mdl, \
                 models/humans/Group03/male_04.mdl, \
                 models/humans/Group03/male_05.mdl, \
                 models/humans/Group03/male_06.mdl, \
                 models/humans/Group03/male_07.mdl, \
                 models/humans/Group03/male_08.mdl, \
                 models/humans/Group03/male_09.mdl, \
                 models/humans/Group03/female_01.mdl, \
                 models/humans/Group03/female_02.mdl, \
                 models/humans/Group03/female_03.mdl, \

```

```

models/humans/Group03/female_04.mdl, \
models/humans/Group03/female_06.mdl, \
models/humans/Group03/female_07.mdl

```

```

;-----
; Items son entidades que restauran salud (botiquines), armadura, armas, balas y
; otros objetos (como cajas) que pueden ser usados en detección de colisiones.
; Clave indica el nombre de tipo de la entidad, y el valor (argumento) depende del
; tipo de la entidad:
; - health y armor: tienen 1 argumento, 'use' indica hace falta estar cerca y apretar el
;   botón USAR (caso de máquina de restaurar salud, por ejemplo), 'none' indica que no hay
;   argumentos, un número indica cantidad de puntos que el item restaura.
; - button: es un botón, no tiene argumentos
; - door: es una puerta, no tiene argumentos
; - object: puede ser 'none' o 'respawable' (indica que la entidad se creará de vuelta
;   pasado un tiempo).
; - object.models: indican tipos de objetos:
;   * 'explosive' indica que la entidad explota. Esto hará que el bot no lo va a romper
;     si el objeto le molesta para pasar (ya que explotará y lo matará al bot si lo rompe).
;   * 'heavy' es usado para indicar que este objeto no puede ser levantado con el arma de gravedad.
;   * 'box' indica que puede ser usado para subirse, para romperlo o para ser levantado con
;     el arma de gravedad.
;-----
[BorzhMod.items.health]
item_healthvial = none
item_healthkit = none
item_healthcharger = use

[BorzhMod.items.armor]
item_battery = none
item_suitcharger = use

[BorzhMod.items.button]
func_button = none

[BorzhMod.items.door]
prop_door_rotating = none

[BorzhMod.items.object]
prop_physics = none
prop_physics_multiplayer = none
prop_physics_respawable = respawable

[BorzhMod.items.object.models]
crate = box
oildrum001_explosive.mdl = explosive
car003b_physics.mdl = heavy

;-----
; Armas y sus parámetros.
; Clave indica el nombre de la entidad y los argumentos son separados por comas:
; - type: tipo de arma.
; - preference: preferencia, bots inteligentes usarán armas de preferencia alta.
; - delay: tiempo entre disparos.
; - reload: tiempo de recarga.
; - holster: tiempo para sacar (cambiar) el arma.
; - range: el rango (distancia) de uso.
; - damage: cuanto daño hace una bala.
; - default_amm: cantidad de balas que el arma da, cuando se agarra el arma del piso.
; - max_amm: máxima cantidad de balas.
; - amm: nombre de la entidad seguido por cantidad de balas que esa entidad da.
;-----
[BorzhMod.weapons]

; Manuales / armas físicas.
weapon_phycannon = type physics, preference lowest, delay 500, holster 100, range 0-200

weapon_stunstick = type manual, preference lowest, delay 1000, range 0-30, damage 40, holster 1000, team combines

weapon_crowbar = type manual, preference lowest, delay 750, range 0-30, damage 24, holster 1000, team rebels

```

```

; Granadas.
weapon_frag = type remote, preference normal, damage 250, parabolic 50, delay 2000, reload 1000, \
             holster 1000, default_ammo 1, max_ammo 5

weapon_slam = type remote, preference normal, damage 300, parabolic 50, \
             delay 2500, reload 1000, holster 1000, max_ammo 5

; Armas para usar a distancia.
weapon_pistol = type pistol, preference lowest, clip 18, damage 8, delay 200, reload 1500, holster 1000, \
              default_ammo 150, max_ammo 150, ammo item_ammo_pistol 20 item_ammo_pistol_large 100

weapon_357 = type pistol, preference normal, clip 6, damage 75, delay 750, reload 3000, holster 2000, \
            default_ammo 6, max_ammo 12, ammo item_ammo_357 6 item_ammo_357_large 12

weapon_shotgun = type shotgun, preference normal, clip 6, damage 75, delay 750, reload 500, holster 1000, \
                default_ammo 6, max_ammo 30, ammo item_box_buckshot 20, range 0-500, \
                secondary, same_ammo, bullets 2, damage 150, delay 750, reload 500, range 0-500?

weapon_smg1 = type rifle, preference normal, clip 45, damage 5, delay 100, reload 1500, holster 1000, \
             default_ammo 45, max_ammo 225, ammo item_ammo_smg1 45 item_ammo_smg1_large 225, \
             secondary, clip 1, damage 200, delay 1000, reload 0, parabolic 50, \
             default_ammo 0, max_ammo 3, ammo item_ammo_smg1_grenade 1

weapon_ar2 = type rifle, preference high, clip 30, damage 11, delay 100, reload 1500, holster 1000, \
            default_ammo 30, max_ammo 60, ammo item_ammo_ar2 20 item_ammo_ar2_large 60, \
            secondary, clip 1, damage 100, delay 1000, reload 0, \
            default_ammo 0, max_ammo 3, ammo item_ammo_ar2_altfire 1

weapon_crossbow = type sniper, preference high, clip 1, damage 100, delay 500, reload 1500, holster 1000, \
                 default_ammo 4, add_clip, max_ammo 10, ammo item_ammo_crossbow 6, \
                 zoom_distance 500, zoom_time 500

weapon_rpg = type rocket, preference highest, damage 250, delay 2000, reload 1000, holster 1000, \
            default_ammo 3, max_ammo 3, ammo item_rpg_round 1, force_aim

; Armas por defecto que tiene el jugador cuando aparece en el mapa.
default = weapon_stunstick, weapon_crowbar

;-----
; Sentencias de chat.
; '/' separa sentencias, '<>' se usa para juntar palabras separadas por '/',
; '()' indica que son palabras opcionales, '$' indica que es una variable de chat.
;-----
[Chat.sentences]

greeting = hello/<hi (there)>/greetings/<(good) morning>/<(good) evening>/<(good) night> , $player.
bye = cya/<see you>/<see ya> (later,) ($player). goodbye/bye ($player).

error = (sorry, $player) (I) can't/don't/couldn't/didn't understand (you/this/that/<what you said>).
busy = (sorry $player,) <I am>/I'm busy (now/<right now>). \
      (sorry $player,) (I) can't talk/chat now/<right now>.

affirmative = yes. yep. affirmative.
negative = no. nope. negative.

affirm = of course.
negate = (sorry $player,) (I) can't help (you) (right now). (sorry $player,) no/<not now>.

call = (<hey,>/<listen,>) $player.
call response = yes? yeah? hm? what? what do you need?

help = (hey,) ($player,) (can you) help (me)? (hey,) ($player,) (I need) help.

stop = (hey,) ($player,) don't move. (hey,) ($player,) stop (moving). \
      (hey,) ($player,) wait. (hey,) ($player,) stay there.
come = (hey,) ($player,) come (here). (hey,) ($player,) stay here/<at my position>.
follow = (hey,) ($player,) follow me/$player. (hey,) ($player,) escort $player.
attack = (hey,) ($player,) attack.
no kill = (hey,) ($player,) <don't attack>/<no kill>.

```

```

sit down = (hey,) ($player,) sit down.
stand up = (hey,) ($player,) stand up.
jump = (hey,) ($player,) jump.

leave = (<ok,>/<hey,>) ($player,) (you can) go/<go away>/continue/leave (now). \
        (hey,) ($player,) (I) don't need (your) help (anymore).

dont hurt = (Please/hey,) stop it/<hitting me>/<hurting me>/<shooting (at) me>. \
        (Please/hey,) don't hurt/hit/shoot me.

ok = Ok/Okey (, cool/sure). Got it. I understand. I see. Oh my. All right. \
        (It's) good/nice to know (it). Good/Nice/Sure. (I) agree.
done = (Ok,) (<I am>/I'm) done. (Ok,) I (have) finished.
wait = ($player,) (please,) wait until I finish. (Please,) (<don't (do it)>/no,) wait (for me).
no moves = I'm/<I am> stuck/stucked. I don't know what to/<can I> do. I have nothing (left) to do. \
        What should/can I do?
think = I will (try to) think (now). I (will) need (some time) to think (now). (Please,) let me think (now).

new area = I'm/<I am> at/in new area, (let's) call it $area.
change area = I'm/<I am> at/in $area.
area button = I'm/<I am> at/near button $button.
area door = I'm/<I am> at/near door $door.

explore = I will investigate/explore <this area>/<the area I'm in (currently)>.
explore new = I will investigate/explore new area.
finish explore = (Ok,) I <(have) finished investigating>/<(have) finished exploring>/<(have) done \
        exploring>/investigated/explored (<this area>/<the area I'm in (currently)>).

weapon found = I (just) found weapon $weapon.

door found = I (just) found $door_status door $door. \
        <I (can) see>/<I am seeing>/<I'm seeing> $door_status door $door (here/<from here>/<over here>). \
        <I (can) see>/<I am seeing>/<I'm seeing>/<I found> door $door, it is $door_status.
door check = I will (go) check/see door $door.
door change = Door $door is now $door_status.
door no change = Door $door isn't/<is not>/<has not>/hasn't changed, (it is still $door_status). \
        Door $door didn't/<did not> change, (it is still $door_status). \
        Door $door stays/<is (still)> $door_status.

button see = I (can) see (a) button $button (here/<from here>/<over here>). I (have) found (a) button $button.
button can push = I can push/press/reach it.
button cant push = I can't push/press/reach it.

box found = I (can) see (a) box $box (here/<from here>/<over here>).
box lost = I don't see box $box (anymore). The box $box is no more in this area. \
        The box $box have been moved from this area.

gravity gun have = I have (a) gravity gun to use it.
gravity gun need = I need (a) gravity gun to use it.

wall found = I (can) see (a) wall (here/<from here>/<over here>).
box need = I need (a) box to climb it/<this wall>.

box try = let's (try to) bring/take (some/a) box to $area.
box I take = I will grab/take (the) box $box (now). I will (try to) use gravity gun on (the) box $box (now).
box you take = $player, (please,) (try to) grab/take (the) box $box (now). \
        $player, (please,) (try to) use gravity gun on (the) box $box (now).
box I drop = I will drop/release box $box (now/here/<near the wall>). \
        I will (try to) put the box $box on the ground (now/<near the wall>).
box you drop = $player, (please,) (try to) drop/release box $box (you are carrying) (now/<near the wall>). \
        $player, (please,) (try to) put the box $box (you are carrying) on the ground (now/<near the wall>).

button weapon = I can/<have weapon to> shoot it.
button no weapon = I can't/<don't have weapon to> shoot it.

door try = let's/lets <try finding>/<(try to) find> button that opens/closes/toggles door $door.
button try = let's/lets <try finding>/<(try to) find>/<try figuring out>/<figure out> what/which \
        door/doors opens/closes/toggles button $button.
        let's try (<to push>/pushing/activate/<to press>/pressing) button $button.
button try go = let's/lets <try going>/<(try to) go>/<try figuring out how to go>/ \

```

```

    <figure out how to go> to (the) button $button.
button door = let's/lets try <to see>/<to check>/<to figure out>/<to find>/finding/<figuring out> \
    if button $button opens/open/closes/close/toggles/toggle/affect/affects door $door.

button toggles = (so,) button $button opens/closes/toggles door $door.
button no toggles = (so,) button $button doesn't/<does not> affect/open/close/toggle/<change state of> \
    door $door. (so,) button $button isn't/<is not> affecting/opening/closing/ \
    toggling/<changing state of> door $door.

button i push = I will (try to) push/press button $button (now).
button pushed = I (just) pushed/pressed button $button.
button you push = $player, (try to) push/press button $button (now).
button i shoot = I will (try to) shoot button $button (now).
button you shoot = $player, (try to) shoot button $button (now).

area go = $player, (please/now,) go/move/walk/run/pass to area $area. \
    $player, could you go/move/walk/run/pass to area $area?
area cant go = I can't reach/<go to (the)>/<move to (the)>/<pass to (the)> area $area.
door go = $player, (please/now,) go/move/walk/run/pass to door $door. \
    $player, could you go/move/walk/run/pass to door $door?
button go = $player, (please/now,) go/move/walk/run/pass to button $button. \
    $player, could you go/move/walk/run/pass to button $button?

cancel task = let's try doing/<to do> something else (then). \
    we should (try to) do something else (then).
better idea = (wait,) I have (a) better idea/plan.

found plan = I found/<have founded>/have/<have got> <(a) plan>/<sequence of actions> to reach the goal!

```

.B. Comandos de consola

Todos los comandos en plugin Botrix empiezan con la palabra “botrix”. A continuación se dan los comandos de consola del plugin y sus descripciones. Aquí el waypoint actual se refiere al waypoint más cercano al jugador que ejecuta el comando.

.B.1. Comandos de waypoints

Los comandos de waypoints se hacen anteponiendo palabra “waypoint” antes del comando. Por ejemplo, el comando para guardar waypoints en un archivo es “save”, entonces el comando completo que el usuario debería introducir en la consola sería “botrix waypoint save”.

- *addtype* argumento: agregar etiqueta al waypoint actual. Argumento puede ser uno de los siguientes: stop (parar), camper, sniper (francotirador), weapon (arma), ammo (balas), health (botiquín), armor (armadura), health_machine (máquina que da vida), armor_machine (máquina que da armadura), button (botón), see.button (botón que se puede disparar).
- *area*. Subconjunto para comandos de área. Puede ser uno de los siguientes:
 - ◊ *remove* argumento: eliminar el área con el nombre “argumento”.
 - ◊ *rename* argumento1 argumento2: cambiar el nombre del área argumento1 por argumento2.
 - ◊ *set* argumento: asignar área *argumento* al waypoint actual.
 - ◊ *show*: mostrar nombres de todas las áreas.
- *argument*: establecer argumento de la etiqueta del waypoint. Dependiendo de la etiqueta del waypoint puede ser un ángulo, cantidad de balas, índice de arma, cantidad de salud que restaura, etc.
- *autocreate* argument: crear nuevos waypoints automáticamente, cuando el jugador se aleje del waypoint actual por distancia máxima. Argumento puede ser: “off” - deshabilitar, “on” - habilitar.
- *clear*: eliminar todos los waypoints.
- *create*: crear un waypoint nuevo en la posición actual del jugador.
- *destination*: establece el waypoint actual del jugador como “destino”.
- *drawtype* argumento: define como mostrar al waypoint. Argumento puede ser mezcla de: line (línea), beam (rayo), box (caja) o text (texto).
- *info*: mostrar la información del waypoint actual en la consola.
- *load*: cargar a los waypoints desde archivo.
- *move*: cambia posición del waypoint “destino” por la posición actual del jugador.
- *remove*: eliminar al waypoint actual.
- *removetype*: elimina todas las etiquetas del waypoint actual.
- *reset*: hace cálculo completo del waypoint actual (ver sección 3.1.2).
- *save*: guarda a los waypoints en archivo.

.B.2. Comandos de conexión entre dos waypoints

Los comandos de conexión entre waypoint actual y waypoint “destino” se hacen anteponiendo palabra “path” antes del comando.

- *autocreate* argumento: crea conexiones entre waypoints automáticamente. Argumento puede ser: “off” - deshabilitar, “on” - habilitar.
- *addtype* argumento: Agrega etiqueta a la conexión actual (desde waypoint actual al waypoint “destino”). Puede ser mezcla de: crouch (agacharse), jump (saltar), break (romper), sprint (correr), ladder (escalera), stop (pararse), damage (daño), flashlight (linterna), door (puerta).
- *argument* argumento1 argumento2: Establecer argumentos de la etiqueta de la conexión actual. Argumento1 es tiempo de espera antes de realizar la acción (después de llegar al waypoint origen), y segundo es la duración en la acción. El tiempo expresarlo en decisegundos.
- *create*: crear conexión desde waypoint actual al waypoint “destino”.
- *drawtype* argumento: define como mostrar a la conexión. Puede ser mezcla de: line o beam.
- *info*: mostrar información sobre la conexión actual en la consola.
- *remove*: eliminar la conexión actual.
- *removetype*: eliminar todas las etiquetas de la conexión actual.

.B.3. Comandos de entidades

Los comandos de entidades se hacen anteponiendo palabra “item” antes del comando. Las entidades son las que se definen en el archivo de configuración (ver secciones [*items.*] en el apéndice .A).

- *draw* argumento: define que entidades mostrar. Argumento puede ser mezcla de: health (botiquín), armor (armadura), weapon (armas), ammo (balas), button (botón), door (puerta), object (objeto, cajas), other (todas las demás entidades).
- *drawtype* argumento: define como mostrar entidades. Argumento puede ser mezcla de: name (mostrar el nombre de la entidad), box (caja), waypoint (waypoint cercano).
- *reload*: recarga todas las entidades (y recalcula los waypoints cercanos).

.B.4. Comandos de bots

Los comandos de bots se hacen anteponiendo palabra “bot” antes del comando.

- *add*: agrega bot aleatorio.
- *debug* argumento1 argumento2: mostrar información de debug del bot en la consola. Argumento1 es nombre del bot y argumento2 puede ser “on” o “off”.
- *drawpath* argumento: define como mostrar al camino actual del bot (adonde va). Puede ser mezcla de: line o beam.
- *kick*: desconectar a un bot aleatorio del servidor.
- *test*: crear a un bot para probar el camino desde el waypoint actual al waypoint “destino”. El bot se desconectará automáticamente al llegar al waypoint destino.
- *weapon*. Subconjunto de comandos para permitir o prohibir usar armas al bot.
 - ◊ *allow* argumentos: permite a los bot usar armas dadas.
 - ◊ *forbid* argumentos: prohíbe a los bot usar armas dadas.

.B.5. Comandos de configuración

Los comandos de bots se hacen anteponiendo palabra “config” antes del comando.

- *admins*. Subconjunto de comandos para administradores.
 - ◊ *access* argumento1 argumentos: establecer nivel de acceso para usuario “argumento1” (tiene que ser un Steam ID). Argumentos pueden ser mezcla de: waypoint, bot, config.
 - ◊ *show*: muestra administradores que están en el servidor actualmente.
- *event* argumento: mostrar eventos del juego (como ser inicio de nivel, muerte, chat, etc.) en la consola. Argumento puede ser: “off” - deshabilitar, “on” - habilitar.

.B.6. Comandos de PDDL

Hay un sólo comando para generación de PDDL y es: “generate-pddl”. Se usa para generar problema en PDDL completo para el mapa actual, esto es, instancia predicados: box-at, button-at, wall, toggle, etc. Notar que este comando generará el problema del mapa completo; para generar PDDL de las creencias del bot hay que escribir “print-buttons” por chat.

.C. Dominio del problema en PDDL

A continuación se da el dominio del problema para planificador FF. Se definen los tipos, predicados y acciones.

```
(define (domain bots-domain)

  (:requirements :equality :typing :adl)

  (:types bot, area, weapon, door, button, box)

  (:predicates
    (physcannon ?weapon - weapon)           ; Arma de gravedad.
    (sniper-weapon ?weapon - weapon)        ; Arma francotirador.

    (can-move ?from ?to - area)             ; También se usa para emular puerta cerrada.
    (can-shoot ?button - button ?area - area) ; Disparar al botón.
    (wall ?from ?to - area)                 ; Una pared.

    (at ?bot - bot ?area - area)           ; Posiciones.
    (box-at ?box - box ?area - area)
    (button-at ?button - button ?area - area)
    (weapon-at ?weapon - weapon ?area - area)

    (has ?bot - bot ?weapon - weapon)      ; Bot tiene un arma.
    (carry ?bot - bot ?box - box)          ; Bot lleva una caja.
    (empty ?bot - bot)                     ; Bot no está cargando caja.

    (between ?door - door ?area1 ?area2 - area) ; Puertas que separan 2 áreas.
    (toggle ?button - button ?door1 ?door2 - door) ; Botón cambia estados de 2 puertas.
  )

  ;-----
  ; Bot se mueve entre áreas adyacentes.
  ;-----
  (:action move
    :parameters
      (?bot - bot
       ?to ?from - area)

    :precondition
      (and
        (at ?bot ?from)
        (can-move ?from ?to)
      )

    :effect
      (and
        (at ?bot ?to)
        (not (at ?bot ?from))
      )
  )

  ;-----
  ; Bot salta desde una pared.
  ;-----
  (:action fall
    :parameters
      (?bot - bot
       ?to ?from - area)

    :precondition
      (and
        (at ?bot ?from)
        (wall ?to ?from)
      )

    :effect
      (and
```

```

        (at ?bot ?to)
        (not (at ?bot ?from))
    )
)

;-----
; Bot agarra un arma que está cerca de un área.
;-----
(:action take-weapon
 :parameters
  (?bot - bot
   ?weapon - weapon
   ?area - area)

 :precondition
  (and
   (at ?bot ?area)
   (weapon-at ?weapon ?area)
  )

 :effect
  (and
   (not (weapon-at ?weapon ?area))
   (has ?bot ?weapon)
  )
)

;-----
; Bot agarra la caja con el arma phycannon.
;-----
(:action carry-box
 :parameters
  (?bot - bot
   ?box - box
   ?area - area
   ?weapon - weapon)

 :precondition
  (and
   (at ?bot ?area)
   (box-at ?box ?area)
   (has ?bot ?weapon)
   (phycannon ?weapon)
   (empty ?bot)
  )

 :effect
  (and
   (not (empty ?bot))
   (carry ?bot ?box)
   (not (box-at ?box ?area))
  )
)

;-----
; Bot agarra la caja con el arma phycannon desde arriba de la pared.
;-----
(:action carry-box-far
 :parameters
  (?bot - bot
   ?box - box
   ?weapon - weapon
   ?area-bot ?area-box - area)

 :precondition
  (and
   (at ?bot ?area-bot)
   (box-at ?box ?area-box)
   (has ?bot ?weapon)
   (phycannon ?weapon)
   (empty ?bot)
  )
)

```

```

        (wall ?area-box ?area-bot)
    )
    :effect
    (and
        (not (empty ?bot))
        (carry ?bot ?box)
        (not (box-at ?box ?area-box))
    )
)

;-----
; Bot tira la caja que estaba llevando.
;-----
(:action drop-box
  :parameters
    (?bot - bot
     ?box - box
     ?area - area)

  :precondition
    (and
        (at ?bot ?area)
        (carry ?bot ?box)
    )

  :effect
    (and
        (box-at ?box ?area)
        (empty ?bot)
        (not (carry ?bot ?box))
    )
)

;-----
; Bot sube la caja para pasar a otro área.
;-----
(:action climb-box
  :parameters
    (?bot - bot
     ?box - box
     ?from ?to - area)

  :precondition
    (and
        (at ?bot ?from)
        (box-at ?box ?from)
        (wall ?from ?to)
        (empty ?bot)
    )

  :effect
    (and
        (not (at ?bot ?from))
        (at ?bot ?to)
    )
)

;-----
; Bot presiona botón que cambia estados de dos puertas.
;-----
(:action push-button
  :parameters
    (?bot - bot
     ?button - button
     ?door1 ?door2 - door
     ?area - area
     ?door1-area1 ?door1-area2 - area
     ?door2-area1 ?door2-area2 - area)

  :precondition

```

```

    (and
      (between ?door1 ?door1-area1 ?door1-area2)
      (between ?door2 ?door2-area1 ?door2-area2)
      (toggle ?button ?door1 ?door2)
      (at ?bot ?area)
      (empty ?bot)
      (button-at ?button ?area)
    )

:effect
  (and
    (when
      (can-move ?door1-area1 ?door1-area2)
      (and
        (not (can-move ?door1-area1 ?door1-area2))
        (not (can-move ?door1-area2 ?door1-area1))
      )
    )
    (when
      (not (can-move ?door1-area1 ?door1-area2))
      (and
        (can-move ?door1-area1 ?door1-area2)
        (can-move ?door1-area2 ?door1-area1)
      )
    )
    (when
      (can-move ?door2-area1 ?door2-area2)
      (and
        (not (can-move ?door2-area1 ?door2-area2))
        (not (can-move ?door2-area2 ?door2-area1))
      )
    )
    (when
      (not (can-move ?door2-area1 ?door2-area2))
      (and
        (can-move ?door2-area1 ?door2-area2)
        (can-move ?door2-area2 ?door2-area1)
      )
    )
  )
)

;-----
; Bot dispara botón que cambia estados de dos puertas.
;-----

(:action shoot-button
  :parameters
    (?bot - bot
     ?button - button
     ?door1 ?door2 - door
     ?sniper-weapon - weapon
     ?area - area
     ?door1-area1 ?door1-area2 - area
     ?door2-area1 ?door2-area2 - area)

  :precondition
    (and
      (sniper-weapon ?sniper-weapon)
      (at ?bot ?area)
      (has ?bot ?sniper-weapon)
      (empty ?bot)
      (can-shoot ?button ?area)
      (toggle ?button ?door1 ?door2)
      (between ?door1 ?door1-area1 ?door1-area2)
      (between ?door2 ?door2-area1 ?door2-area2)
    )

  :effect
    (and
      (when
        (can-move ?door1-area1 ?door1-area2)

```

```
(and
  (not (can-move ?door1-area1 ?door1-area2))
  (not (can-move ?door1-area2 ?door1-area1))
)
)
(when
  (not (can-move ?door1-area1 ?door1-area2))
  (and
    (can-move ?door1-area1 ?door1-area2)
    (can-move ?door1-area2 ?door1-area1)
  )
)
(when
  (can-move ?door2-area1 ?door2-area2)
  (and
    (not (can-move ?door2-area1 ?door2-area2))
    (not (can-move ?door2-area2 ?door2-area1))
  )
)
(when
  (not (can-move ?door2-area1 ?door2-area2))
  (and
    (can-move ?door2-area1 ?door2-area2)
    (can-move ?door2-area2 ?door2-area1)
  )
)
)
)
)
```

.D. El problema de 82 pasos en PDDL

En lo que sigue vemos un problema para planificador FF. La configuración de botones es generada automáticamente. Este problema es soluble y el plan generado por FF tiene 82 pasos.

```
(define (problem bots)

  (:domain
   bots-domain
  )

  (:objects
   bot0 bot1 - bot

   area0 area1 area2 area3 area4 area5 area6 area7 area8 area9 area10 area11 area12 area13 - area
   door0 door1 door2 door3 door4 door5 door6 door7 door8 door9 - door

   button0 button1 button2 button3 button4 button5 button6 button7 button8 button9 - button
   weapon0 weapon1 - weapon
   box0 box1 - box
  )

  (:init
   (sniper-weapon weapon0)
   (weapon-at weapon0 area3)

   (physcannon weapon1)
   (weapon-at weapon1 area7)

   (at bot0 area1) (empty bot0)
   (at bot1 area2) (empty bot1)

   (box-at box0 area1)
   (box-at box1 area12)

   (button-at button0 area8)
   (button-at button1 area7)
   (button-at button2 area6)
   (button-at button3 area5)
   (button-at button4 area4)
   (button-at button5 area3)
   (button-at button6 area13)
   (button-at button7 area2)
   (button-at button8 area9)
   (can-shoot button9 area9)

   (between door0 area8 area2)
   (between door1 area2 area7)
   (between door2 area6 area2)
   (between door3 area5 area6)
   (between door4 area5 area1)
   (between door5 area4 area1)
   (between door6 area1 area3)
   (between door7 area9 area5)
   (between door8 area12 area13)
   (between door9 area10 area9)

   (wall area10 area11)
   (wall area12 area11)

   ; Configuración de botones auto-generada.
   (toggle button0 door0 door5)
   (toggle button1 door3 door1)
   (toggle button2 door6 door6)
   (toggle button3 door7 door2)
   (toggle button4 door8 door0)
   (toggle button5 door9 door6)
   (toggle button6 door7 door0)
   (toggle button7 door1 door4)
  )
)
```

```
(toggle button8 door5 door7)
(toggle button9 door0 door9)
)

(:goal
  (and
    (at bot0 area13)
    (at bot1 area13)
  )
)
)
```

.E. Diálogo entre dos bots

En esta ejecución hay dos bots: Jason y George. El mapa no se puede ganar, ya que ningún botón abre la puerta del área-objetivo. Por esta razón los bots investigan lo más que pueden sobre botones. Al final de la ejecución saben qué botón abre qué puertas (entre los botones vistos).

001. Jason: Hello, george.

002. George: Hi there, jason.

003. George: I'm in new area, let's call it area-respawn2.

004. Jason: I am at new area, call it area-respawn1.

005. George: I will explore new area.

006. Jason: I will investigate new area.

007. George: I'm seeing closed door 3 here.

008. George: I am seeing closed door 2 here.

009. Jason: I'm seeing closed door 5 here.

010. George: I'm seeing closed door 1 from here.

011. Jason: I'm seeing closed door 6 here.

012. George: All right.

013. Jason: I am seeing closed door 7 from here.

014. George: I see button 8 over here.

015. Jason: Okey.

016. George: I explored the area i'm in currently.

George está en el área del botón 8, propone presionarlo.

017. George: Let's try figuring out which door opens button 8.

018. Jason: Please, no, wait.

019. Jason: I can see box 1 from here.

020. Jason: I need a gravity gun to use it.

021. Jason: Ok, i finished investigating the area i'm in currently.

Jason usa planificador: configuración botón8. Output del planificador:

ff: goal can be simplified to TRUE. The empty plan solves it

El planificador terminó con un plan vacío, Jason espera que el otro bot lo proponga.

022. Jason: I am stuck.

George se da cuenta que Jason no tiene nada para hacer y como lo estaba esperando, 'se despierta''. Está en el área del botón 8, propone presionarlo:

023. George: Let's try figuring out what doors closes button 8.

024. Jason: Good.

025. George: I will press button 8.

(Botón 8 afecta puertas 4 y 6)

026. Jason: Door 5 isn't changed.

027. Jason: So, button 8 doesn't toggle door 5.

028. George: Door 3 stays closed.

029. George: So, button 8 isn't toggling door 3.

030. George: Door 2 did not change, it is still closed.

031. George: Button 8 isn't toggling door 2.

032. Jason: Door 6 is now opened.

033. Jason: So, button 8 toggles door 6.

034. George: Door 1 is closed.

035. George: Button 8 does not close door 1.

036. Jason: Door 7 hasn't changed, it is still closed.

037. Jason: Button 8 is not opening door 7.

038. George: I am stucked.

Jason había visto un área nuevo detrás de la puerta que se abrió.

039. Jason: I will explore new area.

040. Jason: I'm in new area, call it area-button5.

041. Jason: I found button 5.

042. Jason: Ok, i finished exploring the area i'm in currently.

043. Jason: Let's try figuring out what doors toggles button 5.

044. George: Yep.

045. Jason: I will press button 5.
(*Botón 5 afecta puertas 2 y 3*)

046. George: Door 1 stays closed.

047. George: Button 5 is not affecting door 1.

048. Jason: Door 6 has not changed,.

049. Jason: Button 5 is not changing state of door 6.

050. Jason: I'm at area-respawn1.

051. George: Door 2 is now opened.

052. Jason: Ok.

053. George: Button 5 opens door 2.

054. George: Door 3 is now opened.

055. Jason: I agree.

056. George: So, button 5 closes door 3.

057. Jason: Door 5 did not change, it is still closed.

058. Jason: So, button 5 doesn't affect door 5.

059. George: I will investigate new area.

060. George: I'm at new area, call it area-button3.

061. Jason: Door 7 did not change, it is still closed.

062. Jason: Button 5 doesn't open door 7.

063. Jason: I'm stucked.

064. George: I am seeing opened door 4.

065. George: I can see a button 3 over here.

066. George: I investigated.

067. George: I will investigate new area.

068. George: I'm in new area, let's call it area-button4.

069. George: I am seeing closed door 8 from here.

070. Jason: Ok, cool.

071. George: I can see a button 4.

072. George: I explored.

073. George: I will investigate new area.

074. George: I'm in area-button3.

075. George: I am at area-respawn2.

076. George: I am at new area, call it area-gravity-gun.

077. George: I found weapon physcannon.

078. George: I can see button 2 here.

079. Jason: All right.

080. George: I finished exploring.

081. George: Let's find what door closes button 2.

082. Jason: Yep.

083. George: I will press button 2.
(*Botón 2 afecta puertas 3 y 10*)

084. Jason: Door 7 has not changed, it is still closed.

085. Jason: Button 2 is not affecting door 7.

086. Jason: Door 6 stays opened.

087. Jason: So, button 2 does not toggle door 6.

088. Jason: Door 5 is not changed.

089. Jason: So, button 2 is not opening door 5.

Jason usa planificador: configuración botón2-puerta1. Output del planificador:
 best first search space empty! problem proven unsolvable.
 El planificador terminó sin un plan válido.

Jason usa planificador: configuración botón2-puerta2. Output del planificador:
 ff: found legal plan as follows
 step 0: REACH-GOAL
 El planificador terminó con un plan vacío, Jason espera que el otro bot lo proponga.

090. George: Door 2 has not changed, it is still opened.
 091. Jason: Nice.
 092. George: Button 2 isn't affecting door 2.

093. George: I am in area-respawn2.
 094. George: Door 1 hasn't changed, it is still closed.
 095. George: Button 2 does not change state of door 1.

Jason usa planificador: configuración botón2-puerta3. Output del planificador:
 best first search space empty! problem proven unsolvable.
 El planificador terminó sin un plan válido.

096. George: Door 3 is now closed.
 097. George: So, button 2 opens door 3.

098. George: I don't know what can i do.

Jason usa planificador: configuración botón2-puerta4. Output del planificador:
 best first search space empty! problem proven unsolvable.
 El planificador terminó sin un plan válido.

Jason usa planificador: configuración botón2-puerta8. Output del planificador:
 best first search space empty! problem proven unsolvable.
 El planificador terminó sin un plan válido.

Jason usa planificador: configuración botón3. Output del planificador:
 ff: found legal plan as follows
 step 0: MOVE BOT2 AREA7 AREA2
 1: PUSH-BUTTON BOT2 BUTTON1 DOOR2 DOOR2 AREA7 AREA6 AREA2 AREA6 AREA2
 2: MOVE BOT2 AREA2 AREA7
 3: MOVE BOT2 AREA6 AREA2
 El planificador terminó con un plan válido, Jason propone ejecutarlo.

099. Jason: Let's try figuring out what doors opens button 3.
 100. George: Affirmative.

Jason ejecuta paso 0 del plan: acción MOVE, argumento 7, bot George.
 101. Jason: George, run to area area-gravity-gun.
 102. George: I am at area-gravity-gun.
 103. George: Ok, done.

Jason ejecuta paso 1 del plan: acción PUSH-BUTTON, argumento 1, bot George.
 104. Jason: George, try to push button 2.
 105. George: I will press button 2.
 (Botón 2 afecta puertas 3 y 10)
 106. George: I have finished.

Jason ejecuta paso 2 del plan: acción MOVE, argumento 2, bot George.
 107. Jason: George, could you move to area area-respawn2?
 108. George: I am in area-respawn2.
 109. George: Ok, i have finished.

Jason ejecuta paso 3 del plan: acción MOVE, argumento 6, bot George.
 110. Jason: George, please, pass to area area-button3.
 111. George: I am in area-button3.
 112. George: Ok, i'm done.

George se da cuenta que se cumplió el objetivo del plan.
 113. George: I will try to press button 3.
 (Botón 3 afecta puertas 3 y 6)

114. Jason: Door 5 stays closed.
 115. Jason: So, button 3 doesn't open door 5.

116. George: Door 3 is now closed.
 117. George: So, button 3 toggles door 3.

118. Jason: Door 6 is now closed.
 119. Jason: Button 3 closes door 6.

120. Jason: Door 7 hasn't changed,.
 121. George: Door 4 stays opened.
 122. George: Button 3 is not affecting door 4.
 123. Jason: Button 3 is not closing door 7.

*Jason usa planificador: configuración botón4. Output del planificador:
 ff: found legal plan as follows
 step 0: MOVE BOT2 AREA5 AREA6
 El planificador terminó con un plan válido, Jason propone ejecutarlo.*

124. Jason: Let's try figuring out what door closes button 4.
 125. George: Jason, please, wait until i finish.

*George termina recorriendo las puertas, está en el área del botón 4,
 nunca fue presionado, propone presionarlo.*

126. George: Let's try figuring out what door toggles button 4.
 127. Jason: Yes.

128. George: I'm at area-button4.
 129. George: I will try to press button 4 now.
(Botón 4 afecta puertas 2 y 8)

130. Jason: Door 7 has not changed, it is still closed.
 131. Jason: So, button 4 is not changing state of door 7.

132. Jason: Door 6 stays closed.
 133. Jason: So, button 4 doesn't close door 6.

134. George: Door 8 is now opened.
 135. Jason: All right.
 136. George: So, button 4 toggles door 8.

137. George: Door 4 is opened.
 138. Jason: Got it.
 139. George: So, button 4 does not affect door 4.

140. George: I'm in area-button3.

141. Jason: Door 5 didn't change.
 142. Jason: So, button 4 does not close door 5.

*Jason usa planificador: configuración botón4-puerta1. Output del planificador:
 best first search space empty! problem proven unsolvable.
 El planificador terminó sin un plan válido.*

143. George: Door 3 hasn't changed.
 144. George: So, button 4 isn't closing door 3.

*Jason usa planificador: configuración botón4-puerta2. Output del planificador:
 best first search space empty! problem proven unsolvable.
 El planificador terminó sin un plan válido.*

*Jason usa planificador: configuración botón8-puerta4. Output del planificador:
 best first search space empty! problem proven unsolvable.
 El planificador terminó sin un plan válido.*

145. George: I will investigate new area.

*Jason usa planificador: configuración botón8-puerta8. Output del planificador:
 best first search space empty! problem proven unsolvable.
 El planificador terminó sin un plan válido.*

Jason no tiene más configuraciones botón-puerta para testear.

146. Jason: I have nothing left to do.

147. George: I am at area-button4.

148. George: I am at new area, let's call it area-shoot-button.

149. George: I see a button 9 from here.

150. Jason: I agree.

151. George: I can see a button 10.

152. George: I don't have weapon to shoot it.

153. George: I'm seeing closed door 10.

154. George: I explored.

155. George: Let's try to find which door closes button 9.

156. Jason: Ok, cool.

157. George: I will try to press button 9 now.

(Botón 9 afecta puertas 2 y 5)

158. Jason: Door 5 is now opened.

159. Jason: Button 9 closes door 5.

160. George: Nice.

161. Jason: Door 6 didn't change, it is still closed.

162. George: Door 8 is opened.

163. George: So, button 9 isn't affecting door 8.

164. Jason: So, button 9 does not toggle door 6.

165. Jason: Door 7 has not changed, it is still closed.

166. Jason: Button 9 doesn't close door 7.

167. George: I'm in area-button4.

168. George: Door 4 isn't changed,.

169. George: Button 9 is not opening door 4.

170. George: I am in area-button3.

171. Jason: Let's try to figuring out if button 4 opens door 10.

172. George: Sorry jason, can't chat right now.

173. George: Door 3 isn't changed,.

174. George: Button 9 is not toggling door 3.

175. George: I am at area-button4.

176. George: I'm in area-shoot-button.

177. George: Door 10 didn't change.

178. George: Button 9 isn't closing door 10.

George usa planificador: configuración botón2-puerta4. Output del planificador:

ff: found legal plan as follows

```
step 0: MOVE BOT2 AREA5 AREA9
      1: MOVE BOT1 AREA5 AREA1
      2: MOVE BOT2 AREA6 AREA5
      3: MOVE BOT1 AREA6 AREA5
      4: PUSH-BUTTON BOT1 BUTTON2 DOOR2 DOOR5 AREA6 AREA6 AREA2 AREA4 AREA1
      5: MOVE BOT1 AREA2 AREA6
      6: MOVE BOT1 AREA7 AREA2
      7: REACH-GOAL
```

El planificador terminó con un plan válido, George propone ejecutarlo.

179. George: Let's try to figuring out if button 2 closes door 4.

180. Jason: Yes.

George ejecuta paso 0 del plan: acción MOVE, argumento 5, bot George.

181. George: I am in area-button4.

George ejecuta paso 1 del plan: acción MOVE, argumento 5, bot Jason.

182. George: Jason, could you go to area area-button4?

183. Jason: I'm in area-button4.

184. Jason: Done.

George ejecuta paso 2 del plan: acción MOVE, argumento 6, bot George.

185. George: I'm at area-button3.

George ejecuta paso 3 del plan: acción MOVE, argumento 6, bot Jason.

186. George: Jason, please, move to area area-button3.

187. Jason: I am at area-button3.

188. Jason: Done.

George ejecuta paso 4 del plan: acción PUSH-BUTTON, argumento 2, bot Jason.

189. George: Jason, try to push button 3.

190. Jason: I will try to press button 3 now.

(Botón 3 afecta puertas 3 y 6)

191. Jason: I have finished.

George ejecuta paso 5 del plan: acción MOVE, argumento 2, bot Jason.

192. George: Jason, now, walk to area area-respawn2.

193. Jason: I'm at area-respawn2.

George ejecuta paso 6 del plan: acción MOVE, argumento 7, bot Jason.

194. George: Jason, pass to area area-gravity-gun.

195. Jason: I am in area-gravity-gun.

196. Jason: I'm done.

Jason se da cuenta que se cumplió el objetivo del plan.

197. Jason: I will try to push button 2.

(Botón 2 afecta puertas 3 y 10)

198. George: Door 4 hasn't changed.

199. George: Button 2 is not affecting door 4.

200. George: I am in area-button4.

201. George: Door 8 did not change, it is still opened.

202. George: So, button 2 is not toggling door 8.

203. Jason: Let's try to figuring out if button 8 toggles door 4.

204. George: No, wait.

205. George: I'm at area-shoot-button.

206. George: Door 10 is now opened.

207. George: So, button 2 opens door 10.

208. Jason: Let's try to figuring out if button 8 opens door 8.

209. George: Wait.

210. George: I will investigate new area.

211. George: I am at new area, call it area-wall1.

212. George: I see wall.

213. George: I need a box to climb this wall.

214. George: Ok, i done exploring this area.

215. George: Let's try to check if button 4 toggles door 2.

216. Jason: Yes.

217. George: I am at area-shoot-button.

218. George: I'm at area-button4.

219. George: I will try to push button 4.

(Botón 4 afecta puertas 2 y 8)

220. Jason: Door 2 is now closed.

221. Jason: Button 4 toggles door 2.

222. George: I don't know what can i do.

Jason usa planificador: configuración botón8-puerta4. Output del planificador:

ff: found legal plan as follows
step 0: PUSH-BUTTON BOT2 BUTTON3 DOOR1 DOOR7 AREA5 AREA2 AREA7 AREA9 AREA5
1: MOVE BOT1 AREA2 AREA7
2: REACH-GOAL

El planificador terminó con un plan válido, Jason propone ejecutarlo.

223. Jason: Let's try to figuring out if button 8 toggles door 4.
224. George: Yes.

Jason ejecuta paso 0 del plan: acción PUSH-BUTTON, argumento 3, bot George.
225. Jason: George, press button 4.

226. George: I will try to push button 4.
(Botón 4 afecta puertas 2 y 8)
227. George: I am done.

Jason ejecuta paso 1 del plan: acción MOVE, argumento 2, bot Jason.
228. Jason: I'm at area-respawn2.

Jason se da cuenta que se cumplió el objetivo del plan.
229. Jason: I will press button 8 now.
(Botón 8 afecta puertas 4 y 6)

230. George: Door 8 hasn't changed,.
231. George: Button 8 is not changing state of door 8.

232. George: Door 4 is now closed.
233. George: Button 8 opens door 4.

Jason usa planificador: configuración botón9-puerta1. Output del planificador:

ff: found legal plan as follows
step 0: MOVE BOT2 AREA9 AREA5
1: REACH-GOAL

El planificador terminó con un plan válido, Jason propone ejecutarlo.

George se da cuenta que puede llegar al área del botón 9, está pensando proponer a Jason la tarea de configuración botón9-puerta1, ya que Jason está ahí. Pero Jason lo dice primero:

234. Jason: Let's try to check if button 9 opens door 1.

George ya tiene tarea, no se da cuenta que coincide con la de Jason.
235. George: Wait.

236. George: Let's try to figuring out if button 9 toggles door 1.
237. Jason: Yep.

George corre hacía botón 9.
238. George: I'm in area-shoot-button.

239. George: I will try to push button 9 now.
(Botón 9 afecta puertas 2 y 5)

240. Jason: Door 3 has not changed,.
241. Jason: So, button 9 doesn't open door 3.

242. Jason: Door 2 is now closed.
243. Jason: So, button 9 toggles door 2.
244. George: Nice.

245. Jason: Door 1 stays closed.
246. Jason: Button 9 is not closing door 1.

Jason usa planificador: llevar la caja 1 al área area-wall1.

247. George: I am stuck.

George no puede usar planificador, porque Jason lo usa, entonces George no tiene nada para hacer.

Output del planificador:
ff: found legal plan as follows

step 0: PUSH-BUTTON BOT2 BUTTON8 DOOR1 DOOR4 AREA9 AREA2 AREA7 AREA5 AREA1
 1: MOVE BOT2 AREA5 AREA9
 2: MOVE BOT2 AREA1 AREA5
 3: CARRY-BOX BOT2 BOX0 AREA1 GRAVITY-GUN
 4: MOVE BOT2 AREA5 AREA1
 5: MOVE BOT2 AREA9 AREA5
 6: MOVE BOT2 AREA10 AREA9
 7: DROP-BOX BOT2 BOX0 AREA10

El planificador terminó con un plan válido, Jason propone ejecutarlo.

248. Jason: Let's try to take box to area-wall1.

249. George: I agree.

Jason ejecuta paso 0 del plan: acción PUSH-BUTTON, argumento 8, bot George.

250. Jason: George, try to press button 9.

251. George: I will try to press button 9 now.

(Botón 9 afecta puertas 2 y 5)

252. George: Ok, done.

Jason ejecuta paso 1 del plan: acción MOVE, argumento 5, bot George.

253. Jason: George, could you move to area area-button4?

254. George: I am at area-button4.

255. George: Ok, i have finished.

Jason ejecuta paso 2 del plan: acción MOVE, argumento 1, bot George.

256. Jason: George, could you pass to area area-respawn1?

257. George: I am in area-respawn1.

258. George: Ok, done.

Jason ejecuta paso 3 del plan: acción CARRY-BOX, argumento 0, bot George.

259. Jason: George, try to take the box 1 now.

260. George: I will take the box 1.

261. George: Ok, done.

Jason ejecuta paso 4 del plan: acción MOVE, argumento 5, bot George.

262. Jason: George, could you walk to area area-button4?

La puerta es estrecha y pasando por la puerta, George golpea la caja contra la puerta, por eso la caja se cae.

263. George: I'm in area-button4.

George se da cuenta de la pérdida de la caja 1, trata de volver y levantarla de nuevo.

264. George: I am at area-respawn1.

George pierde la caja 1.

265. George: I'm in area-button4.

George se da cuenta de la pérdida de la caja 1, trata de volver y levantarla de nuevo.

266. George: I am in area-respawn1.

Esta vez George pasa por la puerta sin problemas.

271. George: I am in area-button4.

272. George: I finished.

Jason ejecuta paso 5 del plan: acción MOVE, argumento 9, bot George.

273. Jason: George, could you walk to area area-shoot-button?

274. George: I am in area-shoot-button.

275. George: Ok, i have finished.

George golpea a la caja contra la pared, la caja se le cae encima.

276. George: Hey, stop it.

Piensa que alguien le disparo, porque perdió puntos de vida :) .

Jason ejecuta paso 6 del plan: acción MOVE, argumento 10, bot George.

277. Jason: George, pass to area area-wall1.

George pierde la caja 1, trata de volver y levantarla de nuevo.

278. George: I'm in area-wall1.

279. George: Ok, i'm done.

Jason ejecuta paso 7 del plan: acción DROP-BOX, argumento 0, bot George.

280. Jason: George, please, release box 1 near the wall.

281. George: I will try to put the box 1 on the ground.

282. George: I have finished.

Jason se da cuenta que se cumplió el objetivo del plan. No tiene nada para hacer.

283. Jason: I'm stuck.

284. George: I will investigate new area.

George sube la pared.

285. George: I'm in new area, call it area-high.

286. George: Ok, i explored.

George salta de la pared a nuevo área.

287. George: I will explore new area.

288. George: I can see a wall here.

289. George: I need box to climb it.

290. George: I'm in new area, call it area-wall2.

291. George: I can see closed door 9.

292. George: I can see box 2 from here.

293. George: I have gravity gun to use it.

294. George: Ok, i explored.

Jason usa planificador: configuración botón10. Output del planificador:

ff: goal can be simplified to FALSE. No plan will solve it

El planificador terminó sin un plan válido.

George no tiene más configuraciones botón-puerta para testear.

295. George: I am stucked.

.F. Diálogo entre un humano y un bot

En esta ejecución hay un bot (John) y un humano (Romina). Se usó la configuración 2 de los botones. Los jugadores pudieron pasar el mapa, llegaron al área-objetivo, pero tuvieron algunos problemas que veremos a continuación. Gracias a esta ejecución encontramos errores en el sistema de chat y también un error grave que tienen los bots que hace que saquen conclusiones erróneas. Una forma de corregir a estos errores se describe en profundidad en el capítulo 8.

001. John: Greetings, Romina.
 002. John: I'm at new area, let's call it area-respawn1.
 003. John: I will explore new area.
 004. John: I can see closed door 5 over here.
 005. John: I can see door 6, it is closed.
 006. John: I am seeing closed door 7 from here.
 007. John: I see box 1 here.
 008. John: I need a gravity gun to use it.
 009. John: I finished investigating this area.
 010. John: I don't know what can i do.

011. Romina: hi John
 012. Romina: I am in area-respawn2
 013. Romina: door 3 is closed
 014. Romina: door 2 is closed
 015. Romina: door 1 is closed

016. Romina: there is a button here, it's number 8
 017. John: I can't understand that.

Aquí vemos que el bot no entendió la expresión de Romina ya que tiene vocabulario bastante estricto.

018. Romina: i see button 8
 019. John: Got it.

020. Romina: i will push button 8

Romina aprieta botón 8, el cual abre puertas 2 y 5.

021. John: Door 5 is now opened.
 022. John: Button 8 toggles door 5.

023. John: Door 6 is not changed.
 024. John: So, button 8 doesn't close door 6.
 025. John: Door 7 has not changed.
 026. John: Button 8 doesn't open door 7.

027. John: I will investigate new area.
 028. John: I am at new area, let's call it area-button4.
 029. John: I see a button 4 over here.
 030. John: I'm seeing closed door 8 over here.
 031. John: I found door 4, it is closed.

032. John: Ok, i investigated the area i'm in currently.

033. John: Lets figure out which door toggles button 4.
 034. Romina: stop

El bot va demasiado rápido para Romina que todavía está aprendiendo a jugar. Ella recorre su área comentando cosas que ve.

035. Romina: button 8 opens door 2
 036. Romina: door 2 is opened
 037. John: I understand.

Había que indicar a Romina que le comente al bot no solamente los cambios después de apretar el botón, sino también que le indique al bot que puertas no cambian.

038. Romina: the button 8 doesn't affect door 1
 039. Romina: the button 8 doesn't affect door 3

040. Romina: lets see what the button 4 does
041. John: Sorry, Romina i can't understand you.

Romina quiere probar el botón 4, del cual habló el bot anteriormente. Sin embargo la frase no dió ningún match y el bot no la entiende.

042. Romina: push button 4

Bot entiende el chat, pero lo omite, porque no sabe a quién va dirigido. Para que bot siga las instrucciones hace falta usar su nombre. Romina cede y empieza a investigar nueva área.

043. Romina: I am in area-gravity-gun
044. Romina: I can see button 2
045. John: I understand.
046. Romina: I found a weapon physcannon

Romina encontró arma de gravedad para levantar cajas.

047. Romina: lets try button 4
048. John: Oh my.
049. John: I will press button 4 now.

Botón 4 abre puertas 3 y 8.

050. John: Door 8 is now opened.
051. John: Button 4 toggles door 8.
052. John: Door 4 stays closed.
053. John: Button 4 is not closing door 4.
054. John: Door 5 isn't changed.
055. John: So, button 4 isn't affecting door 5.

056. John: I'm at area-respawn1.
057. John: Door 6 stays closed.
058. John: Button 4 is not closing door 6.
059. John: Door 7 hasn't changed, it is still closed.
060. John: So, button 4 isn't closing door 7.

061. John: I will explore new area.
062. John: I'm at area-button4.
063. John: I am in new area, call it area-shoot-button.

064. John: I see button 10.
065. John: I don't have weapon to shoot it.
066. John: I am seeing closed door 10.
067. John: I found button 9.
068. John: I finished exploring.

069. John: Let's try to figuring out if button 4 opens door 2.
070. Romina: door 3 is open
071. John: I couldn't understand what you said.

072. Romina: door 3 is open
073. John: I couldn't understand.

Más ejemplos de que la sintáxis tiene que ser correcta.

074. Romina: door 3 is opened
075. John: I see.
076. Romina: button 4 opens door 3

077. Romina: button 4 doesn not opens door 1

Viendo información de debug nos dimos cuenta de que el bot aceptó el chat, pero entendió que botón 4 afecta puerta 1. Hay que deshacer esta creencia:

078. Romina: button 4 doesn't open door 1

079. Romina: am in area-button3
080. Romina: door 4 is closed

081. Romina: I can see button 3
 082. John: Ok.

083. Romina: button 4 doesn't open door 2
 084. John: Affirmative.

085. John: Lets try to find which doors toggles button 9.
 086. Romina: ok
 087. John: I will press button 9 now.

Botón 9 abre puertas 6 y 8.

088. John: Door 8 is now closed.
 089. John: Button 9 opens door 8.
 090. John: Door 10 did not change.
 091. John: Button 9 doesn't open door 10.

092. John: Lets try to check if button 9 closes door 3.
 093. Romina: no
 094. Romina: button 9 doesn't affect door 1
 095. Romina: button 9 doesn't affect door 2
 096. Romina: button 9 doesn't affect door 3
 097. Romina: button 9 doesn't affect door 4

098. Romina: i will press button 3

Botón 3 abre puerta 7. Pero el bot está encerrado en el área shoot-button entre puertas 8 y 10.

099. John: Door 10 did not change.
 100. John: So, button 3 does not open door 10.
 101. John: Door 8 didn't change.
 102. John: Button 3 isn't closing door 8.

John trata ver si existe un plan para usar botón 2. Output del planificador:

```
ff: found legal plan as follows
step    0: MOVE BOTO AREA-RESPAWN2 AREA-BUTTON3
        1: MOVE BOTO AREA-GRAVITY-GUN AREA-RESPAWN2
```

103. John: Let's try figuring out what doors opens button 2.
 104. Romina: ok

Paso 0, acción MOVE, argumento 2, ejecutor Romina.

105. John: Romina, please, move to area area-respawn2.

106. Romina: i am in area-respan2
 107. John: Sorry, Romina don't understand.

108. Romina: i am in area-respawn2
 109. Romina: done

Paso 1, acción MOVE, argumento 7, ejecutor Romina.

110. John: Romina, please, run to area area-gravity-gun.

111. Romina: i am in area-gravity-gun
 112. Romina: done

113. Romina: i can see button 2
 114. John: Nice.
 115. Romina: i will push button 2

Botón 2 afecta a las puertas 2 y 4.

116. John: Door 8 stays closed.
 117. John: So, button 2 doesn't close door 8.
 118. John: Door 10 has not changed, it is still closed.
 119. John: So, button 2 is not opening door 10.

*John no tiene nada para hacer, ejecuta planificador para configuración botón2 - puerta1.:
 best first search space empty! problem proven unsolvable.*

John prueba configuración botón2 - puerta2.:.

ff: found legal plan as follows
step 0: REACH-GOAL

El planificador terminó con un plan vacío, John espera a que otro jugador lo proponga.

120. Romina: button 2 affects door 2

Notar que Romina quedó encerrada en el área gravity-gun. No sabe que hacer. Ahora cabe destacar que puerta 4 cambió su estado, pero ninguno de los dos lo sabe, ya que no la pueden ver.

121. Romina: i don't know what to do

John prueba configuración botón2 - puerta3:.
best first search space empty! problem proven unsolvable.

John prueba configuración botón2 - puerta4:.
ff: found legal plan as follows
step 0: PUSH-BUTTON BOT1 BUTTON8 DOOR7 DOOR7 AREA-SHOOT-BUTTON AREA-SHOOT-BUTTON AREA-BUTTON4
AREA-SHOOT-BUTTON AREA-BUTTON4
1: MOVE BOT1 AREA-BUTTON4 AREA-SHOOT-BUTTON
2: REACH-GOAL

122. John: Let's try to check if button 2 opens door 4.

123. Romina: ok

Paso 0, acción PUSH-BUTTON, argumento 8, ejecutor John.

124. John: I will press button 9.

Botón 9 cambia estado de puertas 6 y 8.
Paso 1, acción MOVE, argumento 5, ejecutor John.

125. John: I am at area-button4.

Ahora el bot esta esperando que Romina apriete el botón 2 pero Romina no lo sabe. Después de explicarle como funciona el bot se da cuenta que tiene que apretar botón 2, que cambia estado de puertas 2 y 4.

126. Romina: i will press button 2

127. John: Door 4 stays closed.

128. John: Button 2 isn't opening door 4.

Nos damos cuenta de que el bot no sabía del estado de la puerta 4 antes de que Romina apriete el botón. Esto es un error grave, que permite que el bot saque conclusiones erróneas. En el capítulo 8 se muestra como se podría corregir.

129. John: Door 5 didn't change, it is still opened.

130. John: So, button 2 does not close door 5.

131. John: I am in area-respawn1.

132. John: Door 6 has not changed.

133. John: Button 2 doesn't change state of door 6.

134. John: Door 7 is now opened.

135. John: So, button 2 closes door 7.

Aquí se ve como bot saca otra conclusión errónea: botón 2 abre puerta 7, pero en realidad es el botón 3.

136. John: I will investigate new area.

137. John: I am in new area, let's call it area-crossbow.

138. John: I see button 6 from here.

139. John: I just found weapon crossbow.

John encontró arma con mira óptica para poder disparar botones que no pueden ser alcanzados.

140. John: Ok, i finished investigating the area i'm in currently.

141. Romina: i am in the area-button3

142. Romina: i have nothing to do

143. John: Let's figure out which door toggles button 6.

144. Romina: affirmative
 145. John: I will try to push button 6.

Botón 6 afecta puertas 7 y 10.

146. Romina: button 6 does not affect door 4
 147. Romina: button 6 does not affect door 3

148. Romina: i am in area-respawn 2
 149. John: Didn't understand.

150. Romina: i am in area-respawn2

151. Romina: button 6 does not affect door 2
 152. Romina: button 6 does not affect door 1

153. John: Door 7 is now closed.
 154. John: So, button 6 opens door 7.

John prueba configuración botón3 - puerta1:
 best first search space empty! problem proven unsolvable.

John prueba configuración botón3 - puerta2:
 best first search space empty! problem proven unsolvable.

John prueba configuración botón3 - puerta3:
 ff: found legal plan as follows
 step 0: MOVE BOTO AREA-BUTTON3 AREA-RESPAWN2
 1: REACH-GOAL

155. John: Let's try to figuring out if button 3 opens door 3.
 156. Romina: ok

Paso 0, acción MOVE, argumento 6, ejecutor Romina.

157. John: Romina, run to area area-button3.
 158. Romina: i am in area-button3

159. Romina: i will press button 3

Botón 3 abre puerta 7. John está cerca de la puerta 7, pero piensa que el botón 3 no la afecta, por esta razón no va a ver su estado.

160. Romina: button 3 does not open door 4
 161. Romina: button 3 does not open door 3
 162. Romina: button 3 does not open door 2
 163. Romina: button 3 does not open door 1
 164. Romina: i am stuck

John prueba configuración botón3 - puerta5:
 ff: found legal plan as follows
 step 0: PUSH-BUTTON BOT1 BUTTON5 DOOR6 DOOR6 AREA-CROSSBOW AREA-RESPAWN1 AREA-CROSSBOW
 AREA-RESPAWN1 AREA-CROSSBOW
 1: MOVE BOT1 AREA-RESPAWN1 AREA-CROSSBOW
 2: REACH-GOAL

165. John: Let's try to check if button 3 toggles door 5.
 166. Romina: ok

John: paso 0, acción PUSH-BUTTON, argumento 5 (botón 6), ejecutor John.

167. John: Door 7 is now opened.

John pasa cerca de la puerta 7, el botón 3 la abrió, pero John no lo sabe. Solamente notifica el cambio en el estado de la puerta.

168. John: I will press button 6 now.

Botón 6 abre puertas 7 y 10.
John: paso 1, acción MOVE, argumento 1, ejecutor John.

169. John: Door 7 is now closed.
 170. John: I can't reach area area-respawn1.

El plan falló, porque John no sabía de que botón 3 afecta a la puerta 7.

John prueba configuración botón3 - puerta6:

ff: found legal plan as follows

```
step 0: PUSH-BUTTON BOT1 BUTTON5 DOOR6 DOOR6 AREA-CROSSBOW AREA-RESPAWN1 AREA-CROSSBOW
        AREA-RESPAWN1 AREA-CROSSBOW
      1: MOVE BOT1 AREA-RESPAWN1 AREA-CROSSBOW
```

171. John: Let's try to check if button 3 closes door 6.

172. Romina: ok

John: paso 0, acción PUSH-BUTTON, argumento 5, ejecutor John.

173. Romina: i will press button 3

Romina no sabe que John todavía no está cerca de la puerta 6, piensa que tiene que apretar solamente botón 3. Esto hace que el bot corte la tarea de seguir el plan. Trata de hacer otra cosa.

John prueba configuración botón3 - puerta7:

ff: found legal plan as follows

```
step 0: REACH-GOAL
```

John se da cuenta que el plan es vacío, espera que el otro jugador lo proponga.

174. Romina: i will press button 3

175. John: Door 7 hasn't changed, it is still closed.

176. John: Button 3 is not affecting door 7.

Cabe destacar que el bot saca otra conclusión errónea: botón 3 no afecta puerta 7. El error fue no ir a investigar la puerta antes de que se apriete el botón 3. Fue apretado dos veces después de que el bot la vio a la puerta por última vez.

John prueba configuración botón6 - puerta5:

best first search space empty! problem proven unsolvable.

John prueba configuración botón6 - puerta6:

best first search space empty! problem proven unsolvable.

John prueba configuración botón6 - puerta8:

best first search space empty! problem proven unsolvable.

John prueba configuración botón6 - puerta10:

best first search space empty! problem proven unsolvable.

John prueba configuración botón9 - puerta5:

best first search space empty! problem proven unsolvable.

John prueba configuración botón9 - puerta6:

best first search space empty! problem proven unsolvable.

John prueba configuración botón9 - puerta7:

best first search space empty! problem proven unsolvable.

John prueba configuración botón10:

ff: found legal plan as follows

```
step 0: PUSH-BUTTON BOT1 BUTTON5 DOOR6 DOOR6 AREA-CROSSBOW AREA-RESPAWN1 AREA-CROSSBOW
        AREA-RESPAWN1 AREA-CROSSBOW
      1: MOVE BOT1 AREA-RESPAWN1 AREA-CROSSBOW
      2: MOVE BOT1 AREA-BUTTON4 AREA-RESPAWN1
      3: MOVE BOT1 AREA-SHOOT-BUTTON AREA-BUTTON4
```

177. John: Let's figure out what door closes button 10.

177. Romina: ok

John: paso 0, acción PUSH-BUTTON, argumento 5, ejecutor John.

178. John: I will try to press button 6.

El botón 6 afecta puertas 7 y 10.

John: paso 1, acción MOVE, argumento 1, ejecutor John.

179. John: I'm at area-respawn1.

John: paso 2, acción MOVE, argumento 5, ejecutor John.

180. John: I am at area-button4.

John: paso 3, acción MOVE, argumento 9, ejecutor John.

181. John: I am at area-shoot-button.

182. John: I will shoot button 10.

El botón 10 afecta puertas 1 y 10. El bot investiga puertas cercanas.

183. John: Door 8 stays opened.

184. John: Button 10 does not change state of door 8.

185. John: I'm in area-button4.

186. John: Door 4 isn't changed.

187. John: Button 10 is not affecting door 4.

188. John: Door 5 isn't changed, it is still opened.

189. John: So, button 10 is not opening door 5.

190. John: I'm in area-respawn1.

191. John: Door 6 hasn't changed.

192. John: So, button 10 does not toggle door 6.

193. Romina: i am in area-respawn2

194. Romina: button 10 open door 1

Romina se da cuenta que cometió error gramatical y trata de corregirlo.

195. Romina: button 10 opens door 1

196. John: Door 7 has not changed.

197. John: Button 10 isn't opening door 7.

198. Romina: button 10 doesn't affect door 2

199. Romina: button 10 doesn't affect door 3

200. Romina: button 10 doesn't affect door 4

201. John: I am in area-button4.

202. John: I am at area-shoot-button.

203. John: Door 10 hasn't changed.

204. John: Button 10 does not change state of door 10.

Otro error! Botón 10 afecta a la puerta 10, pero como antes se había presionado botón 6, el estado de la puerta 10 es el mismo que tenía bot registrado.

205. John: I have nothing left to do.

Viendo información de debug, nos damos cuenta de que el bot tiene errores.

Vamos a cambiar sus creencias, porque este error no va a dejar que podamos ganar el mapa.

206. Romina: button 10 opens door 10

207. Romina: i am in area-button1

208. Romina: i can see button 1

209. Romina: i will press button 1

Botón 1 afecta puertas 1 y 6.

210. John: Door 10 is not changed.

211. John: Button 1 isn't toggling door 10.

212. John: Door 8 did not change.

213. John: So, button 1 is not toggling door 8.

214. John: I am at area-button4.

215. John: Door 4 didn't change.

216. John: Button 1 doesn't toggle door 4.

217. Romina: door 1 is closed

El bot infiere de que botón 10 afecta puerta 1.

218. John: Got it.

219. John: Door 5 stays opened.

220. John: Button 1 does not toggle door 5.

221. John: I am in area-respawn1.

222. John: Door 6 is now opened.

223. John: Button 1 closes door 6.

224. John: Door 7 is not changed.

225. John: So, button 1 does not affect door 7.

226. John: I will explore new area.

227. John: I am at new area, call it area-button5.

228. John: I can see button 5 here.

229. John: I have done exploring.

230. John: Let's try to find what doors closes button 5.

Romina quedó encerrada en el área del botón 1, quiere salir de ahí, sabe que botón 1 abre puerta 1. El bot está esperando respuesta para investigar el botón 5.

231. Romina: i will press button 1

232. Romina: i am in area-button3

Romina salio del área del botón 1. Quiere que el bot prosiga con la tarea de investigar el botón 5.

233. Romina: press button 5

El bot no sabe si el chat está dirigido a el.

234. Romina: john press button 5

235. John: I'm busy.

El bot todavía está esperando respuesta para probar el botón 5. Luego de explicarle esto a Romina, ella acepta la tarea.

236. Romina: ok

237. John: I will push button 5.

El botón 5 afecta puertas 1 y 9.

238. John: Door 6 is not changed.

239. John: So, button 5 isn't opening door 6.

240. John: Lets try to figuring out if button 5 toggles door 3.

241. Romina: i am in area-respawn2

242. Romina: button 5 close door 1

243. John: I can't understand you.

Bot no entendió, está esperando la palabra 'closes'.

244. Romina: button 5 closes door 1

245. Romina: button 5 does not affect door 2

246. Romina: button 5 does not affect door 3

247. John: Got it.

Bot cancela la tarea de ver si botón 5 afecta puerta 3, ya que le dieron esta información.

248. Romina: button 5 does not affect door 4

*John prueba configuración botón1 - puerta2:
best first search space empty! problem proven unsolvable.*

*John prueba configuración botón1 - puerta3:
best first search space empty! problem proven unsolvable.*

John prueba configuración botón3 - puerta5:

ff: found legal plan as follows

```
step  0: PUSH-BUTTON BOT1 BUTTON4 DOORO DOORO AREA-BUTTON5 AREA-BUTTON1 AREA-RESPAWN2 AREA-BUTTON1 AREA-RESPAWN2
      1: MOVE BOTO AREA-BUTTON1 AREA-RESPAWN2
      2: PUSH-BUTTON BOTO BUTTON0 DOOR5 DOOR5 AREA-BUTTON1 AREA-BUTTON5 AREA-RESPAWN1 AREA-BUTTON5 AREA-RESPAWN1
      3: MOVE BOTO AREA-RESPAWN2 AREA-BUTTON1
      4: MOVE BOT1 AREA-RESPAWN1 AREA-BUTTON5
      5: MOVE BOTO AREA-BUTTON3 AREA-RESPAWN2
      6: REACH-GOAL
```

249. John: Let's try to check if button 3 toggles door 5.

250. Romina: ok

John: paso 0, acción PUSH-BUTTON, argumento 4, ejecutor John.

251. John: I will try to push button 5 now.

El botón 5 afecta puertas 1 y 9.

John: paso 1, acción MOVE, argumento 8, ejecutor Romina.

252. John: Romina, now, pass to area area-button1.

253. Romina: i am in area-button1

254. Romina: done

John: paso 2, acción PUSH-BUTTON, argumento 0, ejecutor Romina.

255. John: Romina, try to push button 1.

256. Romina: i will press button 1

El botón 1 afecta puertas 1 y 6. Romina queda encerrada en el área del botón 1, y esto no le agrada. Trata de indicarle que el botón 1 afecta puerta 1, aunque el bot ya lo sabe.

257. Romina: wait

El bot cancela la tarea actual.

258. Romina: button 1 closes door 1

John prueba configuración botón3 - puerta6:

ff: found legal plan as follows

```
step  0: MOVE BOTO AREA-RESPAWN2 AREA-BUTTON1
      1: MOVE BOTO AREA-BUTTON3 AREA-RESPAWN2
      2: REACH-GOAL
```

259. John: Lets try to check if button 3 opens door 6.

260. Romina: ok

John: paso 0, acción MOVE, argumento 2, ejecutor Romina.

261. John: Romina, now, walk to area area-respawn2.

262. Romina: door 1 is closed

Algo está mal en el estado del bot. Romina no puede salir del área del botón 1. Decide apretar el botón 1 de vuelta y salir de allí, lo que cancela la tarea actual de bot.

263. Romina: i will press button 1

264. Romina: i am in area-respawn2

Botón 1 afecta puertas 1 y 6.

John prueba configuración botón5 - puerta5:

best first search space empty! problem proven unsolvable.

John prueba configuración botón5 - puerta7:

best first search space empty! problem proven unsolvable.

John prueba configuración botón5 - puerta8:

best first search space empty! problem proven unsolvable.

John prueba configuración botón5 - puerta10:

best first search space empty! problem proven unsolvable.

John prueba configuración botón6 - puerta5:

best first search space empty! problem proven unsolvable.

John prueba configuración botón6 - puerta6:
best first search space empty! problem proven unsolvable.

John prueba configuración botón6 - puerta8:
best first search space empty! problem proven unsolvable.

John prueba configuración botón6 - puerta10:
best first search space empty! problem proven unsolvable.

John prueba configuración botón9 - puerta5:
best first search space empty! problem proven unsolvable.

John prueba configuración botón9 - puerta6:
best first search space empty! problem proven unsolvable.

John prueba configuración botón9 - puerta7:
best first search space empty! problem proven unsolvable.

John no tiene más configuraciones para probar:

265. John: I don't know what to do.

Parece que el bot tiene mal el estado de la puerta 1... Por eso es importante siempre avisarle al bot sobre cambios en el dominio del juego.

266. Romina: door 1 is opened

267. Romina: i am in area-button3

268. Romina: i am stuck

John prueba configuración botón3 - puerta5:

ff: found legal plan as follows

```
step  0: MOVE BOTO AREA-RESPAWN2 AREA-BUTTON3
      1: MOVE BOTO AREA-BUTTON1 AREA-RESPAWN2
      2: PUSH-BUTTON BOTO BUTTON0 DOORO DOOR5 AREA-BUTTON1 AREA-BUTTON1 AREA-RESPAWN2
          AREA-BUTTON5 AREA-RESPAWN1
      3: PUSH-BUTTON BOT1 BUTTON4 DOORO DOORO AREA-BUTTON5 AREA-BUTTON1 AREA-RESPAWN2
          AREA-BUTTON1 AREA-RESPAWN2
      4: MOVE BOTO AREA-RESPAWN2 AREA-BUTTON1
      5: MOVE BOT1 AREA-RESPAWN1 AREA-BUTTON5
      6: MOVE BOTO AREA-BUTTON3 AREA-RESPAWN2
      7: REACH-GOAL
```

269. John: Lets try to figuring out if button 3 opens door 5.

270. Romina: ok

John: paso 0, acción MOVE, argumento 2, ejecutor Romina.

271. John: Romina, could you walk to area area-respawn2?

272. Romina: ok

273. Romina: i am in area-respawn2

274. Romina: done

John: paso 1, acción MOVE, argumento 8, ejecutor Romina.

275. John: Romina, pass to area area-button1.

276. Romina: i am in area-button1

277. Romina: done

John: paso 2, acción PUSH-BUTTON, argumento 0, ejecutor Romina.

278. John: Romina, try to press button 1.

279. Romina: i will press button 1

280. Romina: done

Botón 1 cambia estado de puertas 1 y 6.

John: paso 3, acción PUSH-BUTTON, argumento 4, ejecutor John.

281. John: I will press button 5 now.

El botón 5 afecta puertas 1 y 9. Romina ahora puede salir del área del botón 1.

John: paso 4, acción MOVE, argumento 2, ejecutor Romina.

282. John: Romina, could you go to area area-respawn2?

283. Romina: ok
 284. Romina: i am in area-respawn2
 285. Romina: done

John: paso 5, acción MOVE, argumento 1, ejecutor John.
 286. John: I am in area-respawn1.

John: paso 6, acción MOVE, argumento 6, ejecutor Romina.
 287. John: Romina, could you run to area area-button3?
 288. Romina: ok
 289. Romina: i am in area-button3

El plan terminó satisfactoriamente. Romina puede presionar botón 3 para ver si afecta a la puerta 5, cerca de la cual está John.

290. Romina: i will press button 3

Botón 3 afecta a puerta 7 únicamente.

291. John: Door 5 didn't change.
 292. John: So, button 3 does not open door 5.

293. John: I'm in area-button4.
 294. John: I am in area-respawn1.

295. John: Door 6 hasn't changed.
 296. John: Button 3 does not open door 6.

297. John: I'm in area-button5.

John prueba configuración botón5 - puerta5:
 best first search space empty! problem proven unsolvable.

John prueba configuración botón5 - puerta7:
 best first search space empty! problem proven unsolvable.

John prueba configuración botón5 - puerta8:
 best first search space empty! problem proven unsolvable.

John prueba configuración botón5 - puerta10:
 best first search space empty! problem proven unsolvable.

John prueba configuración botón6 - puerta5:
 best first search space empty! problem proven unsolvable.

John prueba configuración botón6 - puerta6:
 best first search space empty! problem proven unsolvable.

John prueba configuración botón6 - puerta8:
 best first search space empty! problem proven unsolvable.

John prueba configuración botón6 - puerta10:
 best first search space empty! problem proven unsolvable.

John prueba configuración botón9 - puerta5:
 best first search space empty! problem proven unsolvable.

John prueba configuración botón9 - puerta6:
 best first search space empty! problem proven unsolvable.

John prueba configuración botón9 - puerta7:
 best first search space empty! problem proven unsolvable.

John no tiene más configuraciones para testear.
 298. John: I'm stuck.

299. Romina: lets see if button 2 opens door 4

El bot entendió que botón 2 abre puerta 4 (ver ejemplos de la sección 4.3.2).

300. Romina: lets try to check if button 2 opens door 4

301. John: Button 2 toggles door 4.
 302. Romina: ok

Ahora vamos a cambiar la creencia de que el botón 2 abre puerta 7 (el error descrito anteriormente) y luego veremos que va a hacer.

303. Romina: button 2 doesn't affect door 7

John detecta un cambio en el dominio y empieza a ‘pensar’ de vuelta.

Prueba configuración botón6 - puerta5:

ff: found legal plan as follows

```
step 0: MOVE BOT1 AREA-RESPAWN1 AREA-BUTTON5
      1: MOVE BOTO AREA-RESPAWN2 AREA-BUTTON3
      2: MOVE BOT1 AREA-BUTTON4 AREA-RESPAWN1
      3: PUSH-BUTTON BOTO BUTTON7 DOOR1 DOOR4 AREA-RESPAWN2 AREA-RESPAWN2 AREA-GRAVITY-GUN
          AREA-BUTTON4 AREA-RESPAWN1
      4: MOVE BOTO AREA-GRAVITY-GUN AREA-RESPAWN2
      5: PUSH-BUTTON BOTO BUTTON1 DOOR1 DOOR3 AREA-GRAVITY-GUN AREA-RESPAWN2 AREA-GRAVITY-GUN
          AREA-BUTTON4 AREA-BUTTON3
      6: MOVE BOT1 AREA-BUTTON3 AREA-BUTTON4
      7: MOVE BOT1 AREA-RESPAWN2 AREA-BUTTON3
      8: PUSH-BUTTON BOT1 BUTTON7 DOOR1 DOOR4 AREA-RESPAWN2 AREA-RESPAWN2 AREA-GRAVITY-GUN
          AREA-BUTTON4 AREA-RESPAWN1
      9: MOVE BOT1 AREA-BUTTON3 AREA-RESPAWN2
     10: MOVE BOT1 AREA-BUTTON4 AREA-BUTTON3
     11: MOVE BOT1 AREA-RESPAWN1 AREA-BUTTON4
     12: MOVE BOTO AREA-RESPAWN2 AREA-GRAVITY-GUN
     13: MOVE BOT1 AREA-CROSSBOW AREA-RESPAWN1
     14: MOVE BOTO AREA-BUTTON3 AREA-RESPAWN2
     15: MOVE BOTO AREA-BUTTON4 AREA-BUTTON3
     16: REACH-GOAL
```

304. John: Let's try to figuring out if button 6 closes door 5.
 305. Romina: ok

John: paso 0, acción MOVE, argumento 1, ejecutor John.

306. John: I'm at area-respawn1.

John: paso 1, acción MOVE, argumento 2, ejecutor Romina.

307. John: Romina, go to area area-respawn2.

308. Romina: i am in area-respawn2

309. Romina: done

John: paso 2, acción MOVE, argumento 5, ejecutor John.

310. John: I am at area-button4.

John: paso 3, acción PUSH-BUTTON, argumento 7, ejecutor Romina.

311. John: Romina, press button 8.

312. Romina: i will press button 8

313. Romina: done

Botón 8 afecta puertas 2 y 5.

John: paso 4, acción MOVE, argumento 7, ejecutor Romina.

314. John: Romina, could you walk to area area-gravity-gun?

315. Romina: door 2 is closed

316. Romina: i can't go to area-gravity-gun

El plan falló. John prueba configuración botón6 - puerta6:

ff: found legal plan as follows

```
step 0: PUSH-BUTTON BOTO BUTTON7 DOOR1 DOOR4 AREA-RESPAWN2 AREA-RESPAWN2 AREA-GRAVITY-GUN
          AREA-BUTTON4 AREA-RESPAWN1
      1: MOVE BOT1 AREA-RESPAWN1 AREA-BUTTON4
      2: MOVE BOTO AREA-GRAVITY-GUN AREA-RESPAWN2
      3: PUSH-BUTTON BOTO BUTTON1 DOOR1 DOOR3 AREA-GRAVITY-GUN AREA-RESPAWN2 AREA-GRAVITY-GUN
          AREA-BUTTON4 AREA-BUTTON3
      4: MOVE BOT1 AREA-BUTTON4 AREA-RESPAWN1
      5: MOVE BOT1 AREA-BUTTON3 AREA-BUTTON4
      6: MOVE BOT1 AREA-RESPAWN2 AREA-BUTTON3
      7: PUSH-BUTTON BOT1 BUTTON7 DOOR1 DOOR4 AREA-RESPAWN2 AREA-RESPAWN2 AREA-GRAVITY-GUN
```

AREA-BUTTON4 AREA-RESPAWN1
 8: MOVE BOT1 AREA-BUTTON3 AREA-RESPAWN2
 9: MOVE BOTO AREA-RESPAWN2 AREA-GRAVITY-GUN
 10: MOVE BOT1 AREA-BUTTON4 AREA-BUTTON3
 11: PUSH-BUTTON BOTO BUTTON7 DOOR1 DOOR4 AREA-RESPAWN2 AREA-RESPAWN2 AREA-GRAVITY-GUN
 AREA-BUTTON4 AREA-RESPAWN1
 12: MOVE BOT1 AREA-RESPAWN1 AREA-BUTTON4
 13: MOVE BOTO AREA-BUTTON3 AREA-RESPAWN2
 14: MOVE BOT1 AREA-CROSSBOW AREA-RESPAWN1
 15: MOVE BOTO AREA-BUTTON4 AREA-BUTTON3
 16: MOVE BOTO AREA-RESPAWN1 AREA-BUTTON4
 17: REACH-GOAL

317. John: Lets try to figuring out if button 6 toggles door 6.
 318. Romina: ok

John: paso 0, acción PUSH-BUTTON, argumento 7, ejecutor Romina.

319. John: Romina, try to press button 8 now.

320. Romina: i will press button 8 now

321. Romina: done

Botón 8 afecta puertas 2 y 5.

John: paso 1, acción MOVE, argumento 1, ejecutor John.

322. John: I am in area-respawn1.

John: paso 2, acción MOVE, argumento 7, ejecutor Romina.

323. John: Romina, could you pass to area area-gravity-gun?

324. Romina: i am in area-gravity-gun

325. Romina: done

John: paso 3, acción PUSH-BUTTON, argumento 1, ejecutor Romina.

326. John: Romina, press button 2.

327. Romina: i will press button 2

328. Romina: done

Botón 2 afecta puertas 2 y 4.

John: paso 4, acción MOVE, argumento 5, ejecutor John.

328. John: I am in area-button4.

John: paso 5, acción MOVE, argumento 6, ejecutor John.

329. John: I am at area-button3.

John: paso 6, acción MOVE, argumento 2, ejecutor John.

330. John: I am in area-respawn2.

John: paso 7, acción PUSH-BUTTON, argumento 7, ejecutor John.

331. John: I will try to press button 8 now.

Botón 8 afecta puertas 2 y 5.

John: paso 8, acción MOVE, argumento 6, ejecutor John.

332. John: I am at area-button3.

John: paso 9, acción MOVE, argumento 2, ejecutor Romina.

333. John: Romina, move to area area-respawn2.

334. Romina: i am in area-respawn2

335. Romina: done

John: paso 10, acción MOVE, argumento 5, ejecutor John.

336. John: I'm in area-button4.

John: paso 11, acción PUSH-BUTTON, argumento 7, ejecutor Romina.

337. John: Romina, push button 8.

338. Romina: i will push button 8

339. Romina: done

Botón 8 afecta puertas 2 y 5.

John: paso 12, acción MOVE, argumento 1, ejecutor John.

340. John: I'm at area-respawn1.

John: paso 13, acción MOVE, argumento 6, ejecutor Romina.

341. John: Romina, now, pass to area area-button3.

342. Romina: i am in area-button3
 343. Romina: done

John: paso 14, acción MOVE, argumento 3, ejecutor John.

344. John: Door 7 is now closed.
 345. John: I can't pass to the area area-crossbow.

El plan falló, ya que el bot no sabía el estado de la puerta 7.

John prueba configuración botón6 - puerta8:
 ff: goal can be simplified to FALSE. No plan will solve it

John prueba configuración botón6 - puerta10:
 ff: goal can be simplified to FALSE. No plan will solve it

John prueba configuración botón9 - puerta5:
 ff: found legal plan as follows
 step 0: MOVE BOTO AREA-BUTTON4 AREA-BUTTON3
 1: MOVE BOTO AREA-SHOOT-BUTTON AREA-BUTTON4
 2: REACH-GOAL

346. John: Let's try to figuring out if button 9 closes door 5.
 347. Romina: ok

John: paso 0, acción MOVE, argumento 5, ejecutor Romina.

348. John: Romina, move to area area-button4.
 349. Romina: i am in area-button4
 350. Romina: done

John: paso 1, acción MOVE, argumento 9, ejecutor Romina.

351. John: Romina, could you go to area area-shoot-button?
 352. Romina: i am in area-shoot-button
 353. Romina: done

Ahora el bot está esperando que Romina apriete el botón 9.

354. Romina: i will press button 9

Botón 9 afecta puertas 6 y 8.

355. John: Door 7 is closed.
 356. John: So, button 9 is not changing state of door 7.

357. John: Door 6 is now closed.
 358. John: Button 9 closes door 6.

359. John: Door 5 isn't changed, it is still opened.
 360. John: Button 9 is not changing state of door 5.

361. John: I am in area-button4.

John no tiene más configuraciones para testear.

362. John: I have nothing to do.

Para entender que es lo que tiene mal el bot, imprimimos sus creencias.

A = afecta, N = no afecta, - = no importa porque ya abre 2 puertas, vacío = no se sabe:

Puerta	1	2	3	4	5	6	7	8	9	10
Botón 1	A	-	-	-	-	A	-	-	-	-
Botón 2	-	A	-	A	-	-	-	-	-	-
Botón 3	N	N	N	N	N	N	N	N		N
Botón 4	-	-	A	-	-	-	-	A	-	-
Botón 5	A	N	N	N		N				
Botón 6	N	N	N	N			A			
Botón 7	A	-	-	-	-	A	-	-	-	-
Botón 8	-	A	-	-	A	-	-	-	-	-
Botón 9	-	-	-	-	-	A	-	A	-	-
Botón 10	A	-	-	-	-	-	-	-	-	A

Se nota que el bot tiene problemas con el botón 3. Sabemos que botón 3 abre puerta 7,

vamos a indicarle al bot, que no se da cuenta de eso.

363. Romina: button 3 opens door 7
364. Romina: i don't know what to do

John prueba configuración botón5 - puerta5:

ff: found legal plan as follows

```
step 0: MOVE BOT1 AREA-RESPAWN1 AREA-BUTTON4
      1: PUSH-BUTTON BOTO BUTTON8 DOOR5 DOOR7 AREA-SHOOT-BUTTON AREA-BUTTON5 AREA-RESPAWN1
          AREA-SHOOT-BUTTON AREA-BUTTON4
      2: MOVE BOTO AREA-BUTTON4 AREA-SHOOT-BUTTON
      3: MOVE BOT1 AREA-BUTTON5 AREA-RESPAWN1
      4: REACH-GOAL
```

365. John: Let's try to check if button 5 closes door 5.
366. Romina: ok

John: paso 0, acción MOVE, argumento 1, ejecutor John.

367. John: I'm at area-respawn1.

John: paso 1, acción PUSH-BUTTON, argumento 8, ejecutor Romina.

368. John: Romina, press button 9 now.

369. Romina: i will press button 9

370. Romina: done

Botón 9 afecta puertas 6 y 8.

John: paso 2, acción MOVE, argumento 5, ejecutor Romina.

371. John: Romina, could you run to area area-button4?

372. Romina: i am in area-button4

373. Romina: done

John: paso 3, acción MOVE, argumento 4, ejecutor John.

374. John: I am at area-button5.

La ejecución del plan terminó satisfactoriamente.

375. John: I will press button 5.

Botón 5 afecta puertas 1 y 9.

376. John: I am at area-respawn1.

377. John: Door 5 isn't changed.

378. John: Button 5 doesn't toggle door 5.

379. John: I am in area-button4.

380. John: I am at area-respawn1.

381. John: Door 7 stays closed.

382. John: So, button 5 is not toggling door 7.

383. John: I'm at area-button4.

384. John: Door 8 did not change.

385. John: Button 5 isn't closing door 8.

386. John: I'm at area-shoot-button.

387. John: Door 10 did not change.

388. John: Button 5 does not toggle door 10.

John prueba configuración botón6 - puerta5:

ff: found legal plan as follows

```
step 0: MOVE BOT1 AREA-BUTTON4 AREA-SHOOT-BUTTON
      1: MOVE BOTO AREA-BUTTON3 AREA-BUTTON4
      2: MOVE BOT1 AREA-RESPAWN1 AREA-BUTTON4
```

3: PUSH-BUTTON BOTO BUTTON2 DOOR6 DOOR6 AREA-BUTTON3 AREA-RESPAWN1 AREA-CROSSBOW
AREA-RESPAWN1 AREA-CROSSBOW
4: MOVE BOTO AREA-BUTTON4 AREA-BUTTON3
5: MOVE BOT1 AREA-CROSSBOW AREA-RESPAWN1
6: REACH-GOAL

389. John: Lets try to check if button 6 toggles door 5.
390. Romina: ok

John: paso 0, acción MOVE, argumento 5, ejecutor John.

391. John: I am in area-button4.

John: paso 1, acción MOVE, argumento 6, ejecutor Romina.

392. John: Romina, could you run to area area-button3?

393. Romina: i am in area-button3

394. Romina: done

John: paso 2, acción MOVE, argumento 1, ejecutor John.

395. John: I'm in area-respawn1.

John: paso 3, acción PUSH-BUTTON, argumento 2, ejecutor Romina.

396. John: Romina, try to push button 3.

397. Romina: i will press button 3

398. Romina: done

Botón 3 afecta puerta 7.

John: paso 4, acción MOVE, argumento 5, ejecutor Romina.

399. John: Romina, go to area area-button4.

400. Romina: i am in area-button4

401. Romina: done

John: paso 5, acción MOVE, argumento 3, ejecutor John.

402. John: I am in area-crossbow.

La ejecución del plan terminó con éxito.

403. John: I will press button 6.

Botón 6 afecta puertas 7 y 10.

404. Romina: i am in area-shoot-button

405. Romina: door 10 is opened

406. John: Oh my.

407. Romina: button 6 opens door 10

408. Romina: door 5 is opened

John no tiene más configuraciones botón-puerta para probar.

409. John: What should i do?

410. Romina: i am in area-wall1

411. Romina: i see wall

El bot piensa que Romina está diciendo "I see", que tiene el mismo significado que Ok.

412. Romina: i see wall here

413. Romina: i am stuck

John arranca el planificador para traer alguna caja al área wall1:

ff: found legal plan as follows

step 0: MOVE BOTO AREA-SHOOT-BUTTON AREA-WALL1

1: MOVE BOTO AREA-BUTTON4 AREA-SHOOT-BUTTON

- 2: MOVE BOTO AREA-RESPAWN1 AREA-BUTTON4
- 3: CARRY-BOX BOTO BOXO AREA-RESPAWN1 GRAVITY-GUN
- 4: MOVE BOTO AREA-BUTTON4 AREA-RESPAWN1
- 5: MOVE BOTO AREA-SHOOT-BUTTON AREA-BUTTON4
- 6: MOVE BOTO AREA-WALL1 AREA-SHOOT-BUTTON
- 7: DROP-BOX BOTO BOXO AREA-WALL1

414. John: Let's try to bring a box to area-wall1.

415. Romina: ok

John: paso 0, acción MOVE, argumento 9, ejecutor Romina.

416. John: Romina, pass to area area-shoot-button.

417. Romina: i am in area-shoot-button

418. Romina: done

John: paso 1, acción MOVE, argumento 5, ejecutor Romina.

419. John: Romina, please, run to area area-button4.

420. Romina: i am in area-button4

421. Romina: done

John: paso 2, acción MOVE, argumento 1, ejecutor Romina.

422. John: Romina, could you pass to area area-respawn1?

423. Romina: i am in area-respawn1

424. Romina: done

John: paso 3, acción CARRY-BOX, argumento 0, ejecutor Romina.

425. John: Romina, please, take box 1.

426. Romina: i will take box 1

427. Romina: done

John: paso 4, acción MOVE, argumento 5, ejecutor Romina.

428. John: Romina, move to area area-button4.

429. Romina: i am in area-button4

430. Romina: done

John: paso 5, acción MOVE, argumento 9, ejecutor Romina.

431. John: Romina, now, pass to area area-shoot-button.

432. Romina: i am in area-shoot-button

433. Romina: done

John: paso 6, acción MOVE, argumento 10, ejecutor Romina.

434. John: Romina, run to area area-wall1.

435. Romina: i am in area-wall1

436. Romina: done

John: paso 7, acción DROP-BOX, argumento 0, ejecutor Romina.

437. John: Romina, drop box 1 now.

438. Romina: i will drop box 1

439. Romina: done

El bot terminó de probar todas las configuraciones posibles.

440. John: What should i do?

Romina se sube a la caja y pasa al área detrás de la pared.

441. Romina: i am in area-wall2

442. Romina: i can see door 9

443. John: I agree.

444. Romina: door 9 is closed

445. Romina: i can see box 2

446. Romina: i see a wall here

447. Romina: i will climb the wall
448. John: I couldn't understand that.

Bot no entiende que quiere decir Romina. Romina quiere agarrar la caja 2 y dejarla cerca de la pared para poder volver.

449. Romina: i will take the box 2

450. Romina: i will left the box
451. John: Didn't understand what you said.

452. Romina: i will drop the box 2
453. Romina: i will climb the wall
454. John: Sorry, Romina don't understand.

455. Romina: i am stuck
Romina está parada sobre la caja, desde ahí se ve la puerta 9 y la puerta 10.

John prueba configuración botón3 - puerta9:

ff: found legal plan as follows

```
step 0: PUSH-BUTTON BOT1 BUTTON5 DOOR6 DOOR9 AREA-CROSSBOW AREA-RESPAWN1 AREA-CROSSBOW
        AREA-WALL1 AREA-SHOOT-BUTTON
      1: MOVE BOT1 AREA-RESPAWN1 AREA-CROSSBOW
      2: MOVE BOT1 AREA-BUTTON4 AREA-RESPAWN1
      3: MOVE BOT1 AREA-BUTTON3 AREA-BUTTON4
      4: REACH-GOAL
```

456. John: Let's try to figuring out if button 3 closes door 9.
457. Romina: ok

John: paso 0, acción PUSH-BUTTON, argumento 5, ejecutor John.

458. John: I will press button 6 now.

Botón 6 afecta puertas 7 y 10.

459. Romina: button 6 doesn't affect door 9

John: paso 1, acción MOVE, argumento 1, ejecutor John.

460. John: I am in area-respawn1.

John: paso 2, acción MOVE, argumento 5, ejecutor John.

461. John: I am in area-button4.

John: paso 3, acción MOVE, argumento 6, ejecutor John.

462. John: I'm in area-button3.

La ejecución del plan terminó bien. Ahora a probar configuración botón3 - puerta9.

463. John: I will push button 3 now.
El botón 3 afecta puerta 7

464. Romina: button 3 doesn't affect door 9
465. John: Yep.

El bot se da cuenta que puede llegar al botón 5 sin usar el planificador y cerca de la puerta 9 está Romina.

466. John: Lets try to figuring out if button 5 opens door 9.
467. Romina: ok

468. John: I am at area-button4.
469. John: I am at area-respawn1.

470. John: I am at area-button5.

471. John: I will try to press button 5 now.
Botón 5 afecta puertas 1 y 9.

472. Romina: door 9 is opened

473. John: Okey, sure.

474. John: I don't know what to do.

475. Romina: button 5 opens door 9

476. Romina: i am in area-goal!

477. Romina: i can see button 7

478. John: Ok, cool.

479. Romina: i will press button 7

Botón 7 afecta puertas 1 y 8.

480. John: Door 6 isn't changed, it is still opened.

481. John: Button 7 is not changing state of door 6.

483. Romina: button 7 doesn't affect door 9

482. John: I am at area-respawn1.

484. John: Door 5 stays opened.

485. John: Button 7 isn't toggling door 5.

486. John: I am in area-button4.

487. John: Door 8 is now closed.

488. John: So, button 7 closes door 8.

Romina se sube la caja para ver si el botón afectó la puerta 10.

489. Romina: button 7 doesn't affect door 10

490. John: Door 4 isn't changed, it is still opened.

491. John: Button 7 does not open door 4.

492. John: I am at area-button3.

493. John: Door 3 is not changed, it is still opened.

494. John: Button 7 does not toggle door 3.

495. John: I am at area-respawn2.

496. John: Door 2 is not changed, it is still closed.

497. John: So, button 7 doesn't toggle door 2.

498. John: Door 1 is now closed.

499. John: So, button 7 opens door 1.

El área objetivo fue visitada, John arranca el planificador para llegar allí:

ff: found legal plan as follows

```
step  0: MOVE BOT1 AREA-BUTTON3 AREA-RESPAWN2
      1: MOVE BOT1 AREA-BUTTON4 AREA-BUTTON3
      2: PUSH-BUTTON BOTO BUTTON6 DOORO DOOR7 AREA-GOAL AREA-BUTTON1 AREA-RESPAWN2
          AREA-SHOOT-BUTTON AREA-BUTTON4
      3: MOVE BOT1 AREA-SHOOT-BUTTON AREA-BUTTON4
      4: SHOOT-BUTTON BOT1 BUTTON9 DOORO DOOR9 CROSSBOW AREA-SHOOT-BUTTON AREA-BUTTON1
          AREA-RESPAWN2 AREA-WALL1 AREA-SHOOT-BUTTON
      5: MOVE BOT1 AREA-WALL1 AREA-SHOOT-BUTTON
      6: CLIMB-BOX BOT1 BOXO AREA-WALL1 AREA-HIGH
      7: FALL BOT1 AREA-WALL2 AREA-HIGH
      8: MOVE BOT1 AREA-GOAL AREA-WALL2
```

500. John: I found a plan to reach the goal!
501. Romina: good
502. Romina: it was a time :)
503. John: Sorry, Romina i didn't understand.

Claramente bot carece de sentido de humor.

John: paso 0, acción MOVE, argumento 6, ejecutor John.
504. John: I'm at area-button3.

John: paso 1, acción MOVE, argumento 5, ejecutor John.
505. John: I'm at area-button4.

John: paso 2, acción PUSH-BUTTON, argumento 6, ejecutor Romina.
506. John: Romina, push button 7 now.
507. Romina: i will press button 7
508. Romina: done

Botón 7 afecta puertas 1 y 8.

John: paso 3, acción MOVE, argumento 9, ejecutor John.
509. John: I am at area-shoot-button.

John: paso 4, acción SHOOT-BUTTON, argumento 9, ejecutor John.
510. John: I will try to shoot button 10.

El bot dispara al botón 10, el cual abre puertas 1 y 10. Necesita abrir la puerta 10 para pasar al área wall1.

John: paso 5, acción MOVE, argumento 10, ejecutor John.
511. John: I'm in area-wall1.

John: paso 6, acción CLIMB-BOX, argumento 0, ejecutor John.
512. John: I am in new area, call it area-high.

John: paso 7, acción FALL, argumento 12, ejecutor John.
513. John: I'm in area-wall2.

John: paso 8, acción MOVE, argumento 13, ejecutor John.
514. John: I am in area-goal.

Los dos jugadores llegaron al área-objetivo y así pasan el nivel.

Bibliografía

- [Mal95] M.S. Malone. *The Microprocessor: A Biography*. History e-book project. TELOS, The Electronic Library of Science, 1995.
- [Buc04] Mat Buckland. *Programming Game AI by Example*. Jones & Bartlett Publishers, 1 edition, September 2004.
- [BenVil11] Luciana Benotti and Martin Villalba. Inferencia de Waypoints en mundos virtuales. In *Workshop argentino de video juegos*, November 2011.
- [Cla96] H.H. Clark. *Using Language*. Cambridge University Press, May 1996.
- [Aus62] John L. Austin. *How to do things with words*. Harvard U.P., Cambridge, Mass., 1962.
- [Sea69] John Searle. *Speech acts : an essay in the philosophy of language*. Cambridge University Press, London, 1969.
- [Gri75] H. P. Grice. Logic and conversation. In P. Cole and J. L. Morgan, editors, *Syntax and Semantics: Vol. 3: Speech Acts*, pages 41–58. Academic Press, San Diego, CA, 1975.
- [Pet04] Ronald P. A. Petrick. Pks: Knowledge-based planning with incomplete information and sensing, 2004.
- [Gef02] Héctor Geffner. Perspectives on artificial intelligence planning. In *Eighteenth national conference on Artificial intelligence*, pages 1013–1023, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence.
- [Hof01] Jörg Hoffmann. FF: The Fast-Forward Planning System. *AI Magazine*, 22(3):57–62, 2001.
- [BLG97] Blai Bonet, Gábor Loerincs, and Héctor Geffner. A robust and fast action selection mechanism for planning. In *In Proceedings of AAAI-97*, pages 714–719. MIT Press, 1997.
- [HN01] Jörg Hoffmann and Bernhard Nebel. The FF planning system: fast plan generation through heuristic search. *J. Artif. Int. Res.*, 14(1):253–302, May 2001.
- [Byl94] Tom Bylander. The computational complexity of propositional STRIPS planning. *Artif. Intell.*, 69(1-2):165–204, 1994.
- [KH11] Jana Koehler and Jörg Hoffmann. On reasonable and forced goal orderings and their use in an agenda-driven planning algorithm. *CoRR*, abs/1106.0243, 2011.

- [FCS97] Jeremy Frank, Peter Cheeseman, and John Stutz. When gravity fails: Local search topology. *CoRR*, cs.AI/9712101, 1997.
- [NGT04] Dana S. Nau, Malik Ghallab, and Paolo Traverso. *Automated planning - theory and practice*. Elsevier, 2004.
- [Wav01] J.M.P. van Waveren. The Quake III Arena Bot, 2001.
- [MilFun09] Ian Millington and John Funge. *Artificial Intelligence for Games, Second Edition*. Morgan Kaufmann Publishers Inc., 2nd edition, 2009.
- [RusNor03] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: a modern approach*. Prentice Hall, 2nd international edition edition, 2003.
- [Alice04] Richard Wallace. Artificial Linguistic Internet Computer Entity (A.L.I.C.E). <http://alice.pandorabots.com>, 2004.
- [Flo06] Counter Floyd. Ping of Death Bot (PodBot). http://podbotmm.bots-united.com/main_pb_page/index.htm, 2006.
- [Hl98] Valve Corporation. Half-Life. [http://en.wikipedia.org/wiki/Half-Life_\(video_game\)](http://en.wikipedia.org/wiki/Half-Life_(video_game)), 1998.
- [Cs99] Valve Corporation. Counter-Strike. <http://en.wikipedia.org/wiki/Counter-Strike>, 1999.
- [Src04] Valve Corporation. Source, 3d engine. https://developer.valvesoftware.com/wiki/Source_SDK, 2004.
- [Hl2-04] Valve Corporation. Half-Life 2. http://en.wikipedia.org/wiki/Half-Life_2, 2004.
- [Hl2Dm04] Valve Corporation. Half-Life 2: Deathmatch. http://en.wikipedia.org/wiki/Half-Life_2:_Deathmatch, 2004.
- [Css04] Valve Corporation. Counter-Strike: Source. http://en.wikipedia.org/wiki/Counter-Strike:_Source, 2004.
- [Botrix13] Boris Godin. Código fuente del plugin y algunos videos de muestra. <http://www.famaf.unc.edu.ar/~godin/>, 2013.