UNIVERSIDAD NACIONAL DE CORDOBA

FACULTAD DE MATEMATICA, ASTRONOMIA Y FISICA

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACION

TESIS PARA LA CARRERA
LICENCIATURA EN CIENCIAS DE LA COMPUTACION

Inteligencia Artificial en juegos: Desarrollando un game partner utilizando generación de lenguaje natural

Alumno: Bertoa Nicolás

Directora: Benotti Luciana

CÓRDOBA ARGENTINA

18 de Noviembre de 2013

Página para los Evaluadores

Calificación:

Comentarios:

....

.....

.....

Lugar para la Fecha de la Evaluación

a mis padres, hermano e hijo.

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Mapa de tesis

El desarrollo de videojuegos vive en su propio mundo técnico. Tiene sus propios idiomas, habilidades, y desaf´ıos. Esa es una de las razones por las cuales es muy divertido estar dentro de él. A pesar de los numerosos esfuerzos para emparejarse con el resto de la industria, el estilo de programación en un videojuego es todavía muy diferente de aquel en cualquier otra esfera del desarrollo. Hay una focalización sobre la velocidad, pero no es muy similar a la programación de aplicaciones embebidas o de control. Hay una focalización sobre los buenos algoritmos, pero no comparte el mismo rigor que en la ingenier´ıa de los servidores de bases de datos. Se toman técnicas de un gran rango de fuentes diferentes, pero casi sin excepción se las modifica. Y, para agregar una capa extra de intriga, los desarrolladores hacen sus modificaciones de maneras diferentes.

Actualmente la mayoría de los tutoriales en los videojuegos están hechos para que el jugador siga un *script* fijo. Por lo tanto, que un tutorial sea bueno o malo es todavía un arte que depende de cuán bueno sea el *script* para hacernos creer que como jugadores tenemos libertad (por ejemplo, prediciendo posibles reacciones del jugador) con el fin de hacer el juego más entretenido. Para la industria de videojuegos, los tutoriales son caros de desarrollar porque los programadores deben predecir todas las posibles reacciones del jugador. La generación de lenguage natural puede proveer a los *game designers* de técnicas que, en el futuro, podrían transformar el arte del diseño de tutoriales en una ciencia que ofrezca diferentes puntos de vista y algoritmos eficientes para el cálculo del conocimiento del jugador y así poder lidiar con sus reacciones de forma dinámica.

En el afán de concretar el objetivo de esta tesis, se realizaron numerosas contribuciones que intentan mejorar y ampliar la visión del campo teórico y práctico actual de los temas anterioremente mencionados. Podemos enumerar las siguientes contribuciones:

- Diseño de algoritmos de replanificación para entornos no determinísticos: El entorno en el que nos focalizamos es un entorno dinámico, es decir, posee diversos elementos que pueden cambiar de estado, con o sin intervención por parte del jugador, y que no siempre lo hacen de la misma manera. Por lo tanto, se diseñaron algoritmos para cambiar el plan a seguir en el lugar y momento óptimos.
- Inclusión de un planificador estático dentro de un entorno dinámico: En nuestro caso, elegimos un planificador que asume un entorno estático, es decir, que en tiempo real los elementos del entorno no cambian, sino únicamente mediante la intervención del jugador. Como parte de nuestro trabajo, se implementó un sistema para utilizar este planificador dentro de un entorno dinámico.

- Producción automática de *scripts* que cambian de acuerdo al comportamiento impredecible del jugador: Como mencionamos anteriormente, la mayoría de los tutoriales actuales están implementados mediante un *script* fijo. Cada vez que el programador encuentra un nuevo comportamiento del jugador, debe ir e actualizar el *script*. En nuestro caso, cuando el plan a seguir debe cambiarse ya que el jugador reaccionó de forma impredecible e inesperada, dinámicamente se genera un nuevo plan a seguir sin necesitar intervención por parte del programador.

Estas contribuciones fueron publicadas en [LN11].

Además, se realizaron evaluaciones con jugadores humanos tomando métricas objetivas (calculadas en tiempo real en cada partida)y subjetivas (cuestionario completado por el jugador luego de la partida). A partir de dichas evaluaciones se confeccionó una lista de conclusiones y posibles mejoras para nuestro sistema. A continuación resumiremos cada uno de los cápitulos de nuestro trabajo.

1.1.1. Capítulo 2

La inteligencia artificial[Int] (IA) tiene como fin hacer que las computadoras sean capaces de realizar las tareas de pensamientos de las que son capaces los humanos y animales. Es un área muy importante en el desarrollo de videojuegos ya que mejora de forma considerable la experiencia de juego. A pesar que las computadoras resuelven tareas sobrehumanas como por ejemplo la resolución de problemas aritméticos o de búsqueda, hay muchas cosas que no resuelven correctamente y que nosotros encontramos triviales: reconocimiento de caras familiares, hablar nuestro propio lenguaje, etc. La inteligencia artificial intenta descubrir los algoritmos necesarios para lograr tales fines.

Pac-man (Midway Games West, 1979)[Gamb] fue el primer juego que muchas personas recuerdan haber jugado con inteligencia artificial, que se basaba en una máquina de estados. La inteligencia artificial en los videojuegos no cambió mucho hasta mediados de 1990, época en la cual empezó a ser una razón de venta de juegos. La IA en la mayoría de los juegos actuales posee tres necesidades básicas: la capacidad de mover los personajes, la capacidad de tomar decisiones sobre donde moverlos, y la capacidad de pensar tácticamente o estratégicamente[MF09].

Una de las tareas relevantes para los videojuegos es el proceso de toma de decisiones. Éste involucra a un personaje resolviendo lo próximo que hará. El sistema de toma de decisiones necesita decidir cual es el comportamiento más apropiado para el personaje en cada momento del juego. La importancia de esta tarea radica en simular que los personajes realmente piensan y razonan como un ser humano.

Un subcampo de la inteligencia artificial relevante para nuestro trabajo es la generación de lenguaje natural[RN03] (NLG). Se focaliza en la construcción de sistemas de computadora que puedan producir textos entendibles en inglés o en otras lenguas humanas, a partir de una representación no lingüística de información. Intenta responder preguntas como las siguientes: ¿cómo deberían interactuar las computadoras con la gente?, ¿cuál es la mejor manera de comunicar información a un humano por parte de una máquina?, ¿qué constituye que un lenguaje sea entendible o apropiado en una situación dada?, ¿cómo pueden ser convertidas las representaciones de información por computadora en representaciones apropiadas para los humanos, como por ejemplo un número de conceptos simbólicos de alto nivel?. La mayoría de los sistemas de NLG actuales se utilizan para presentar información a los usuarios o automatizar parcialmente la producción de documentación rutinaria. Algunas razones para utilizarla en lugar de la autoría humana son: la consistencia, la conformidad con los estándares, la velocidad de la producción de documentos, la generación de documentos en múltiples lenguajes y el mantenimiento del staff.

La aplicación más relevante para nuestro trabajo es la generación de instrucciones en entornos

virtuales. Uno de los ejemplos es GIVE[Cha] (*Giving Instructions in Virtual Environments*), que es una competencia de tecnologías de NLG en la cual se aplican técnicas al problema de generar instrucciones en lenguaje natural en un entorno virtual 3D. Como resultado, varias técnicas investigadas por la comunidad de GIVE son directamente aplicables a nuestro trabajo.

Para aplicar las técnicas de NLG para la generación de instrucciones en entornos virtuales desarrollamos un juego del género *First Person Shooter* [Sho]. En este tipo de juegos, el jugador ve al mundo a través de una cámara en primera persona, de este modo se siente inmerso dentro de él. El jugador camina por un castillo medieval habitado por un monstruo enemigo que persigue lo persigue con la intención de matarlo. El objetivo del juego es eliminar al enemigo y para ello el jugador debe utilizar sus armas. El enemigo es invulnerable hasta que el jugador recoja una secuencia de rayos de color en el orden correcto, a partir de lo cual puede eliminarlo. Si el jugador recolecta un rayo incorrecto debe volver a realizar el procedimiento desde el principio. Mientras el monstruo sea invulnerable, el jugador únicamente puede alejarlo dañándolo con sus armas, luego de lo cual el enemigo se aleja para recargar energ´ıas y vuelve a perseguirlo.

1.1.2. Capítulo 3

Las tareas y técnicas de la generación de lenguaje natural[RN03] que son relevantes para nuestro trabajo son la determinación de contenido a través de *planning*, la generación situada de expresiones referenciales y la gestión del *common ground* [CB91].

La determinación del contenido es el nombre que le damos al problema de decidir la información que debería ser comunicada. Algunas veces la aplicación proveerá una especificación de la información a ser comunicada, pero en muchos casos el sistema de NLG será el responsable de seleccionar algún subconjunto apropiado de información disponible. Esta decisión depende de una variedad de factores: los diferentes objetivos comunicativos, dependencia de caracter´ısticas asumidas o conocidas por el oyente o lector, las restricciones sobre el texto final, la fuente de información, etc.

Muy pocos sistemas de generación de lenguaje natural simplemente generan mensajes que comunican todos sus datos de entrada. En su lugar, realizan un procesamiento de los datos que incluye las siguientes etapas: selección, sumario, razonamiento y adaptación de la salida. En muchos casos, la parte más difícil de construir un sistema para la determinación de contenido es determinar cuales son las reglas para identificar los mensajes que deberían ser incluidos en un texto. Este problema es muy parecido al análisis de requerimientos en la ingeniería de software convencional[CB91].

La segunda tarea de la NLG es la generación de expresiones referenciales que afecta a la forma en la cual se produce una descripción de una entidad que permita al oyente identificarla en un contexto dado. Se relaciona con la NLG ya que la misma entidad puede ser referida de diferentes maneras. Esto es as´ı tanto cuando una entidad es mencionada por primera vez (referencia inicial) como cuando la entidad es referida posteriormente luego de que ha sido introducida dentro del discurso (referencia subsecuente)[CB91].

La función de la referencia inicial es introducir a alguna entidad dentro del discurso. La generación de expresiones referenciales para referencias iniciales está relativamente inexplorada en la literatura de NLG. Por otro lado, las referencias subsecuentes se utilizan generalmente para referirnos a entidades que ya han sido introducidas en el discurso. Estas referencias son anafóricas en cuanto a que sus interpretaciones son dependientes del material precedente. Para referirnos a entidades que han sido recientemente mencionadas en el discurso utilizamos los pronombres. En español y en otros lenguajes, se distinguen por género, persona, y número. El pronombre que se utilice dependerá de las propiedades del antecedente y el contexto sintáctico[CB91].

La última tarea es la gestión del *common ground*. Dos personas que se están comunican-

do necesitan coordinar el contenido y el proceso de dicha comunicación para tener éxito. Para ello deben asumir una basta cantidad de información compartida o *common ground*. Además, para coordinar en el proceso, necesitan actualizarlo constantemente. Todas las acciones colectivas se construyen en base a la acumulación de *common ground*. Éste no puede ser actualizado apropiadamente sin un proceso, al cual llamamos *grounding*. En una conversación, por ejemplo, los participantes intentan lograr que lo que se ha dicho haya sido entendido. En nuestra terminolog´ıa, intentan hacer *grounding* sobre lo que se dijo, esto es, hacerlo parte de su *common ground*.

La mayor´ıa de las conversaciones comienzan con un potencial contribuyente que presenta un enunciado a su compañero. Una contribución generalmente se divide en dos fases: la fase de presentación en la cual el emisor presenta un enunciado esperando evidencias por parte del receptor para asegurarse que entendió lo que quiso decir; y la fase de aceptación en la cual el receptor acepta un enunciado por parte del emisor y da evidencias de su entendimiento. Se necesita de ambas fases para que una contribución se considere completa. Una vez que decimos algo en una conversación, todo lo que tenemos que hacer es buscar evidencias negativas, es decir, evidencias que nos digan que no se entendió lo que dijimos. Si encontramos alguna, reparamos el problema, pero si no, asumimos por defecto que se nos entendió. Pero si la evidencia negativa es lo único que buscamos, puede darse el caso de que aceptemos información que tiene poca justificación para ser aceptada. Por esta razón, la gente últimamente busca evidencia positiva de entendimiento cuyas tres formas más comunes son los *acknowledgements*, la iniciación del próximo turno relevante y la atención continua[CB91].

Algunas restricciones que el medio puede imponer en la comunicación entre dos personas son: copresencia, visibilidad, capacidad de audición, cotemporabilidad, simultaneidad, secuenciabilidad, capacidad de revisión de mensajes recibidos y capacidad de revisión de mensajes a enviar. Cuando un medio sufre escasez de una de estas características, éste generalmente fuerza a las personas a usar técnicas alternativas para el proceso de *grounding*. Se hace esto por que los costos de las técnicas del proceso de *grounding* cambian. Los once costos son: costo de formulación, de producción, de recepción, de entendimiento, de iniciación, de *delay*, de asincronización, por cambio de emisor, de indicación, de falla y de reparación[CB91].

1.1.3. Capítulo 4

El *game partner* es un agente experto que ayuda al avance del jugador, en nuestro caso en un juego del género *First Person Shooter* [Sho]. En la mayoría de los juegos, el proceso de toma de decisiones de los jugadores es o bien implementado de forma ad-hoc que es mediante la creación de un *script* para ese juego en particular, o diseñado como una máquina de estados finitos[MF09] que necesita ser pequeña debido a problemas de escalabilidad. Un *game partner* creíble no puede conseguirse de esta forma. Primero, el agente necesita reaccionar apropiadamente a las infinitas reacciones de los jugadores que no pueden ser predecidas y colocadas en un *script*. Y segundo, el agente necesita razonar sobre los complejos estados del juego hacia un objetivo dirigido con el fin de ser capaz de dar instrucciones que son causalmente apropiadas en el momento en el cual se expresaron y relevantes con respecto al objetivo del juego.

Proponemos utilizar un planificador[GK10] para implementar el proceso de toma de decisiones del *game partner* experto. Los planificadores automatizados han alcanzado un nivel de madurez en el que pueden ser usados en aplicaciones de tiempo real incluso si la necesidad de replanificación es alta. En nuestra configuración hay una necesidad alta de replanificación debido a que el comportamiento del jugador es no determinístico e impredecible. Además, los planificadores son independientes del dominio, y con el fin de obtener un plan que gane el juego, sólo necesitan una especificación de las acciones que son posibles en el juego y una representación discretizada de lo que el agente sabe sobre el estado del juego.

El agente de NLG conoce la secuencia correcta de recolección de rayos así como también la

posición de cada uno de ellos. Además posee información del jugador, el enemigo, y los demás items del juego. Toda esta información está almacenada en la especificación que se le envía al planificador, y el planificador retorna una secuencia de acciones que, si fuesen ejecutadas en el estado actual del juego, lograr´ıan que el jugador cumpla el objetivo del juego. Dada la secuencia anterior de acciones, el agente necesita decidir las acciones a verbalizar. En nuestro agente hemos decidido verbalizar la ultima acción en la secuencia que contiene una expresión referencial que identifica unívocamente al objeto involucrado en la acción.

El agente necesita información sobre el entorno del juego para poder generar un problema de *planning*. Dicho problema va a servir para que el planificador encuentre un plan que pueda ser utilizado por el agente. Primero el agente debe obtener información sobre el dominio en lenguaje PDDL[Gha98] asociado al problema. Como mencionamos anteriormente, el agente necesita información del entorno para elaborar un problema de *planning* acorde y luego de obtener un plan, poder chequear si el plan se ha cumplido o no, y en que situaciones debe replanificar. Además de la información de las entidades del juego el agente utiliza algoritmos para calcular los objetos visibles a 360 grados del jugador como así también los que estan bajo la mira.

El agente tendrá dos formas de darle instrucciones al jugador para guiarlo a cumplir el objetivo del nivel: mediante instrucciones por pantalla o mediante instrucciones reproducidas por audio. El jugador puede seguirlas al pie de la letra o realizar cualquier otra acción que desee. En algunos casos, el agente puede guiarlo a seguir el plan ya calculado, pero hay casos en los que se necesita generar un nuevo plan, es decir, replanificar. En nuestro caso, decidimos replanificar por dos razones: reposicionamiento del item a recoletar o desv´ıo del camino del plan.

1.1.4. Capítulo 5

El objetivo de la evaluación es testear nuestro sistema con humanos y ver los resultados que obtenemos. Para esto usaremos métricas objetivas y subjetivas, mediante las cuales obtendremos datos mucho más precisos para luego analizarlos y sacar conclusiones. Además basándonos en los resultados podremos evidenciar aspectos y situaciones no contempladas anteriormente que enriquecerán nuestro sistema una vez integradas.

Compararemos dos planificadores: BlackBox[Bla] y FastForward[Hof01]. El objetivo de la comparación es justificar la elección del planificador que utilizaremos en nuestro juego.

También detallaremos las métricas que utilizaremos para analizar el agente. Las métricas son parámetros a tener en cuenta para evaluar el sistema. Es muy importante la selección de dichas métricas para que evaluen aspectos claves del sistema y a través de su análisis se puedan ver los puntos fuertes y débiles del mismo, pudiendo así sacar conclusiones y mejorarlo. Las hay de dos tipos: objetivas y subjetivas. El primer paso para poder evaluar el sistema es recolectar datos importantes sobre las partidas jugadas por humanos. En nuestro caso hacemos dos tipos de registro de datos, uno en tiempo real, es decir, mientras se juega la partida, y el otro al final de la misma.

Cabe mencionar que para la evaluación emplearemos dos sistemas diferentes. El primero, al cual denominaremos sistema escrito, consiste en un sistema en el cual el agente muestra instrucciones por pantalla al jugador. El segundo, al cual denominaremos sistema oral, consiste en un sistema en el cual el agente reproduce las instrucciones mediante archivos de audio. Luego de ambas evaluaciones compararemos los puntos fuertes y débiles de ambos sistemas.

Además, presentaremos tres casos de estudio que mostrarán comportamientos claves del agente. Los casos de estudio son importantes porque proveen una manera sistemática de observar eventos, recolectar datos y analizar la información. Por último, analizaremos los resultados de las partidas efectuadas por humanos, explicando de que consta el proceso, que datos analizaremos y que conclusiones podemos obtener.

Capítulo 2

Trabajo previo

La inteligencia artificial (IA) tiene como fin hacer que las computadoras sean capaces de realizar las tareas de pensamientos de las que son capaces los humanos y animales. En el campo de los videojuegos es un tópico muy importante ya que mejora de gran manera la experiencia de juego por parte del jugador. No es igual de divertido un juego en el cual los enemigos son fáciles de eliminar, ya que el jugador no encontrará desafío alguno y se aburrirá, aunque tampoco es bueno el otro extremo, ya que sintiéndose muy inferior a los enemigos, también dejará de jugarlo. Es muy importante comprender como ha ido evolucionando la inteligencia artificial en los videojuegos, para comprender el estado del arte y el porqué de las decisiones que se toman hoy en d´ıa con respecto a este campo.

Como mencionamos anteriormente, una de las tareas relevantes es el proceso de toma de decisiones. Éste involucra a un personaje resolviendo lo próximo que hará. El sistema de toma de decisiones necesita decidir cual es el comportamiento más apropiado para el personaje en cada momento del juego. La importancia de esta tarea radica en simular que los personajes realmente piensan y razonan como un ser humano.

Nos centraremos en el subcampo de la inteligencia artificial llamado generación de lenguaje natural[GK10], ya que es el que nos permitirá realizar la tarea propuesta para nuestro trabajo: la generación de instrucciones. Todas las técnicas y el estado del arte de este campo afectarán de manera directa a nuestro trabajo. En particular, la aplicación en entornos virtuales. Desarrollamos nuestro propio entorno virtual, en este caso un juego, por dos razones: para conocer totalmente nuestro sistema y para complicar tanto como querramos el juego. Además conoceremos ejemplos de entornos virtuales previamente desarrollados para ver como aplican las técnicas de generación de lenguaje natural.

En la sección 2.1 veremos la historia de la inteligencia artificial en los videojuegos y su importancia, como así también numerosos ejemplos de juegos que utilizan este tipo de técnicas. En la sección 2.2 veremos de que se trata la generación de lenguaje natural, sus diversas aplicaciones, sus tareas principales y porque es relevante para nuestro trabajo. En la sección 2.3 veremos dos ejemplos de entornos virtuales que utilizaron técnicas de generación de lenguaje natural. Por último, en la sección 2.4 describiremos a grandes rasgos el entorno virtual del juego usado en esta tesis, es decir, los objetos que lo constituyen y la función que cumplen.

2.1. Inteligencia artificial en juegos

El desarrollo de videojuegos vive en su propio mundo técnico. Tiene sus propios idiomas, habilidades, y desaf´ıos. Esa es una de las razones por las cuales es muy divertido estar dentro de

él. A pesar de los numerosos esfuerzos para emparejarse con el resto de la industria, el estilo de programación en un videojuego es todavía muy diferente de aquel en cualquier otra esfera del desarrollo. Hay una focalización sobre la velocidad, pero no es muy similar a la programación de aplicaciones embebidas o de control. Hay una focalización sobre los buenos algoritmos, pero no comparte el mismo rigor que en la ingenier´ıa de los servidores de bases de datos. Se toman técnicas de un gran rango de fuentes diferentes, pero casi sin excepción se las modifica. Y, para agregar una capa extra de intriga, los desarrolladores hacen sus modificaciones de maneras diferentes.

En la sección 2.1.1 veremos que es la inteligencia artificial y cuales son los principales desafíos con los que se encuentra el programador. En la sección 2.1.2 veremos como se aplicó la inteligencia artificial en los primeros juegos y su evolución. Por último, en la sección 2.1.3 veremos en qué se basa el proceso de toma de decisiones y cómo influye en los videojuegos y qué puede aportar la Inteligencia Artificial a este proceso.

2.1.1. Qué es la inteligencia artificial?

La inteligencia artificial (IA) tiene como fin hacer que las computadoras sean capaces de realizar las tareas de pensamientos de las que son capaces los humanos y animales.

Ya podemos programar computadoras para que tengan habilidades sobrehumanas en la resolución de muchos problemas: aritméticos, de ordenación, de búsqueda, etc. Incluso podemos hacer que las computadoras jueguen algunos juegos de mesa mejor que un ser humano (por ejemplo, Reversi[Rev]). Muchos de estos problemas fueron originalmente considerados como problemas de IA, pero a medida que han sido resueltos de formas más comprensivas, han salido del dominio de los desarrolladores de IA.

Pero hay muchas cosas que las computadoras no resuelven correctamente y que nosotros encontramos triviales: reconocimiento de caras familiares, hablar nuestro propio lenguaje, decidir que hacer después, y ser creativos. Éstos son dominios de la IA.

En la academia, algunos investigadores son motivados por la filosof´ıa: el entendimiento de la naturaleza del pensamiento y la naturaleza de la inteligencia y la construcción de software que modele el proceso de pensamiento. Algunos son motivados por la psicolog´ıa: el entendimiento de los mecanismos del cerebro humano y los procesos mentales. Otros son motivados por la ingeniería: la construcción de algoritmos para la realización de tareas humanas. Esta triple distinción es el corazón de la inteligencia artificial académica.

Los desarrolladores de videojuegos se interesan principalmente en la parte ingenieril: la construcción de algoritmos que hagan que los personajes del juego se parezcan a humanos o animales. Los desarrolladores siempre han tomado ideas de las investigaciones académicas, donde esas investigaciones los ayudan a completar su trabajo.

2.1.2. Aplicación en video juegos

Pac-Man [Man] (Midway Games West [Gamb], 1979) fue el primer juego que muchas personas recuerdan haber jugado con inteligencia artificial. Hasta ese punto hab´ıan habido clones del Pong [Pon] con oponentes controlados por computadora que básicamente seguían a la bola arriba y abajo, e incontables *shooters* en el Space Invaders [Inv]. Pero Pac-Man tenia definitivamente enemigos que parecían conspirar contra ti, moviéndose alrededor del nivel de la misma forma en que el jugador lo hac´ıa, y pareciendo tener vida propia.

Pac-Man se basaba en una técnica de IA muy simple: una máquina de estados [MF09]. Cada

uno de los cuatro monstruos o bien te perseguía o escapaba. Para cada estado ellos seguían una ruta semi-aleatoria en cada unión. En el modo de persecución cada uno tenía una chance diferente para perseguir al jugador o elegir una dirección aleatoria. En el modo de evasión o bien escapaban o elegían una dirección aleatoria.

La inteligencia artificial en los videojuegos no cambió mucho hasta mediados de 1990. La mayoría de los jugadores controlados por computadora anteriores a esa época, fueron tan sofisticados como los enemigos del Pac-Man.

Tomemos un clásico como Golden Axe [Axe] (SEGA Entertainment [SEG], 1987). Los enemigos se quedaban quietos (o caminaban hacia atrás y hacia adelante una distancia corta) hasta que el jugador estaba lo suficientemente cerca de ellos, después de lo cual se dirigían a él. Golden Axe tuvo una innovación con los enemigos en la cual el enemigo arrazaba con el jugador y luego lo atacaba por detrás. La sofisticación de su IA es solo un paso más que la de Pac-Man.

A mediados de 1990 la inteligencia artificial empezó a ser una razón de venta de juegos. Juegos como Beneath a Steel Sky [aSS] (Revolution Software [Sof], 1994) incluso mencionó inteligencia artificial en la parte trasera de su caja. Desafortunadamente, su sistema de IA permitía que los personajes caminaran hacia atrás y adelante a través del juego, difícilmente un avance real.

Goldeneye 007 [007] (Rare [Rar], 1997) probablemente le mostró a los jugadores la forma en que la inteligencia artificial podía mejorar la experiencia de juego. Aunque sus personajes tenían un número pequeño de estados bien definidos, Goldeneye agregó un sistema de simulación de sentidos: los personajes podían ver a sus colegas y notar si ellos fueron aniquilados. La simulación de los sentidos fue el tópico del momento, con Thief: The Dark Project [Pro] (Looking Glass Studios [Stub], 1998) y Metal Gear Solid [Sol] (Konami Corporation [Kon], 1998) basando sus diseños de juegos enteros sobre esta técnica.

A mediados de 1990, los juegos de estrategia en tiempo real [RTS] (RTS) hab´ıan comenzado a destacarse. Warcraft [Wara] (Blizzard Entertainment [Ent], 1994) fue uno de los juegos en los que por primera vez era muy notorio el uso de *pathfinding* [MF09]. Los investigadores sobre IA estuvieron trabajando con modelos emocionales de soldados en un simulador de un campo de batalla militar en 1998 cuando ellos vieron al juego Warhammer: Dark Omen [Ome] (Mindscape [Min], 1998) haciendo lo mismo. Esta fue una de las primeras veces que la gente vio el movimiento de formaciones robustas en acción.

Recientemente, un creciente número de juegos han situado a la inteligencia artificial como su punto principal. Creatures [Cre] (Cyberlife Technology [Tec], 1997) hizo esto en 1997, pero juegos como The Sims [Sim] (Maxis Software [Max], 2000) y Black and White [BW] (Lionhead [Stua], 2001) han llevado la antorcha. Creatures tiene uno de los sistemas de IA más complejos vistos en un juego con un cerebro basado en una red neuronal [RN03] para cada criatura.

Ahora tenemos una diversidad masiva de IA en juego. Muchos géneros éstan todavía usando las técnicas simples de 1979 porque es todo lo que necesitan. Los *bots* en los *First Person Shooters* [Sho] (FPS) han visto mucho interés en la inteligencia artificial académica más que cualquier otro género. Los RTS han utilizado mucha de la IA utilizada para construir simuladores de entrenamiento para los militares (cabe mencionar que Full Spectrum Warrior [Warb] (Pandemic Studios [Stuc], 2004) comenzó siendo un simulador de entrenamiento militar).

Los juegos de deportes y de autos tienen en particular sus propios desaf´ıos con respecto a la IA, alguno de los cuales permanecen largamente sin resolverse (por ejemplo, el cálculo dinámico del camino más rápido alrededor de una pista de carreras), mientras que los juegos de rol [RPG] (RPG) con interacciones complejas entre los personajes y todavía implementadas como árboles de conversación, necesitan de una mejor inteligencia artificial. Varias lecturas y artículos en los últimos cinco o seis años han sugerido mejoras que todavía no han sido materializadas en la producción de juegos.

La inteligencia artificial en la mayoría de los juegos actuales posee tres necesidades básicas: la capacidad de mover a los personajes, la capacidad de tomar decisiones sobre donde moverlos, y la capacidad de pensar tácticamente o estratégicamente. Incluso cuando hemos migrado desde el uso de IA basada en estados por todas partes, hacia un rango amplio de técnicas, todavía ellas cumplen los tres requerimientos básicos. [MF09]

2.1.3. El proceso de toma de decisiones

El proceso de toma de decisiones involucra a un personaje resolviendo lo próximo que hará. Comúnmente, cada personaje tiene un rango de comportamientos diferentes que pueden ejecutar: atacar, mantenerse quieto, esconderse, explorar, patrullar, etc. El sistema de toma de decisiones necesita decidir cuál de estos comportamientos es el más apropiado en cada momento del juego. El comportamiento elegido puede ser ejecutado usando IA de movimiento y tecnología de animación.

En su nivel más simple, un personaje puede tener reglas muy simples para seleccionar un comportamiento. Los animales de granja en varios niveles de los juegos de Zelda [Zel] se mantienen quietos a menos que el jugador se acerque demasiado, después de lo cual se alejarán una distancia pequeña.

En el otro extremo, los enemigos en Half-Life 2 [Lif] (Valve [Cor], 2004) muestran un proceso de toma de decisiones complejo, donde intentarán varias estrategias diferentes para alcanzar al jugador: concatenación de acciones intermedias tales como arrojar granadas y disparar fuego de supresión con el fin de lograr sus objetivos.

Algunas decisiones pueden requerir de IA de movimiento para llevarlas a cabo. Una acción de ataque cuerpo a cuerpo requerirá que el personaje se acerque a su víctima. Otras son manejadas puramente por animaciones (por ejemplo, el Sim comiendo) o simplemente mediante la actualización del estado del juego directamente, sin ningún tipo de respuesta visual (cuando la inteligencia artificial de un pa´ıs en Sid Meier's Civilization III [III] (Firaxis Games [Gama], 2001) elige investigar una nueva tecnolog´ıa, por ejemplo, esto pasa simplemente sin respuesta visual).

2.2. Generación de lenguaje natural

La generación de lenguaje natural (NLG) es el subcampo de la inteligencia artificial [RN03] y la lingüística computacional que se focaliza en la construcción de sistemas de computadora que puedan producir textos entendibles en inglés o en otras lenguas humanas. Típicamente se empieza a partir de una representación no lingüística de información como entrada, los sistemas de NLG usan el conocimiento sobre un lenguaje y el dominio de aplicación para producir automáticamente documentos, reportes, explicaciones, mensajes de ayuda, y otros tipos de textos.

La generación de lenguaje natural es tanto un área de investigación fascinante como una tecnología emergente con muchas aplicaciones en el mundo real. Como un área de investigación, NLG nos brinda una única perspectiva sobre temas fundamentales de la inteligencia artificial, ciencia cognitiva, e interacción humano-computadora. Éstos incluyen preguntas tales como la forma en que la lingüística y el conocimiento del dominio deberían ser representados y como razonar con ellos, que significa que un texto esté bien escrito, y como la información es comunicada de la mejor manera entre máquina y humano. Desde una perspectiva práctica, la tecnología de NLG es capaz de automatizar parcialmente la rutina de creación de documentos, removiendo la mayoría de las cosas tediosas asociadas con tal tarea. Además se utiliza en los laboratorios de investigación, y esperamos que pronto se utilice en aplicaciones reales, para presentar y explicar información compleja a gente que no tiene el conocimiento o tiempo requeridos para entender los datos en bruto. A largo plazo, es también probable que NLG juegue un rol importante en las interfaces humano-computadora y permitirá mucha más rica interacción con las máquinas de la

que es posible hoy en d´ıa.

En la sección 2.2.1 veremos las preguntas fundamentales que los investigadores se hacen con respecto a la generación de lenguaje natural. En la sección 2.2.2 veremos los beneficios de los sistemas de generación de lenguaje natural y las tareas que pueden facilitar. En la sección 2.2.3 veremos los casos en los cuales es recomendable que utilicemos sistemas de generación de lenguaje natural y cuando no.

2.2.1. La perspectiva de investigación

Desde una perspectiva de investigación, NLG es un subcampo del procesamiento del lenguaje natural [dln] (NLP), que a su vez puede ser visto como un subcampo de la ciencia de la computación y la ciencia cognitiva. Desde la amplia perspectiva de la ciencia de la computación y la ciencia cognitiva como un todo, NLG provee una importante y única perspectiva sobre muchos problemas y preguntas fundamentales, incluyendo las siguientes:

- ¿Cómo deberían interactuar las computadoras con la gente? ¿Cuál es la mejor manera de comunicar información a un humano para una máquina? ¿Qué tipo de comportamiento lingüístico espera una persona de una computadora con la que se está comunicando, y cómo puede este comportamiento ser implementado? Estas son preguntas básicas en la interacción humano-computadora, un área de la ciencia de la computación que se está convirtiendo en algo tan importante como la calidad de un sofware de computadora.
- ¿Qué constituye que un lenguaje sea entendible o apropiado en una situación comunicativa dada? ¿Cómo pueden ser formalizadas apropiadamente las restricciones pragmáticas, semánticas, sintácticas, y psicolinguísticas? ¿Qué rol juega el contexto en la elección de un lenguaje apropiado? Estas son preguntas básicas en la linguística, y además importantes en la filosof´ıa y psicolog´ıa, tal como lo son en la ciencia cognitiva.
- ¿Cómo pueden ser convertidas las representaciones de información por computadora en representaciones apropiadas para los humanos, como por ejemplo un número de conceptos simbólicos de alto nivel? ¿Qué tipo de dominios y modelos del mundo y razonamientos asociados se requieren para traducir información a partir de representaciones de computadora a lenguaje natural, con su vocabulario y estructura? Estas preguntas son aspectos de una pregunta aun mayor sobre como la inteligencia humana puede ser modelada y simulada en una computadora, el cual es uno de los objetivos principales de la inteligencia artificial.

Aunque los trabajos sobre generación de lenguaje natural puedan proveer pistas en estos campos relacionados, además acuden a estos mismos campos en busca de ideas. Por ejemplo, muchos sistemas de NLG usan ideas desarrolladas dentro del campo de la inteligencia artificial, tales como técnicas de *planning* y reglas de producción, para determinar el contenido informativo de un texto; y la mayoría de los sistemas de NLG usan modelos de lingüística formal de sintaxis para asegurar que su texto de salida es gramaticalmente correcto.

2.2.2. La perspectiva de aplicaciones

Desde una perspectiva de aplicaciones, la mayor´ıa de los sistemas de NLG actuales se utilizan para presentar información a los usuarios o automatizar parcialmente la producción de documentación rutinaria. La presentación de la información es importante porque las representaciones internas utilizadas por sistemas de computadoras requieren una experiencia considerable para interpretarlas. La producción automatizada de documentos es importante porque mucha gente invierte grandes proporciones de su tiempo en la producción de documentos, incluso cuando no es su responsabilidad principal. Un programador de computadora puede invertir tanto tiempo

escribiendo un texto (documentación del codigo, descripción de la lógica del programa, revisiones del código, reportes de progreso, etc) como escribiendo código. Las herramientas que ayudan a tales personas a producir buenos documentos rápidamente pueden mejorar considerablemente la productividad y la moral.

Cuando construimos un sistema completo de NLG, una decisión importante de diseño que debe ser tomada es si el sistema operará en el modo *standalone*, generando textos sin ninguna información humana, o si operará produciendo lo que en efecto son borradores de textos que serán modificados por un autor humano. Esta distinción es la misma que la encontrada en sistemas expertos, donde es muy frecuente que se encuentre la presencia de un humano, especialmente en contextos donde un error en alguna parte del sistema puede ser riesgoso para la vida o desastrozo de alguna otra forma.

2.2.3. ¿Cuándo son apropiadas las técnicas de NLG?

El desarrollo de un sistema basado en técnicas de NLG no es siempre la mejor manera de cumplir las necesidades del usuario. En algunos casos, puede ser mejor presentar la información de forma gráfica en lugar de presentarlo en forma textual. En otros casos, la mejor solución es contratar a una persona para que escriba los documentos o le explique cosas al usuario. Que opción es mas apropiada en una circunstancia dada depende de un número de factores, incluyendo el tipo de información siendo comunicada, la calidad del texto requerido y el volumen del texto a ser producido/citeclarkbrennan01.

Es importante reconocer que muy pocos usuarios finales explícitamente perciben la necesidad de utilizar un sistema basado en tecnología de NLG. Es mucho más común que la gente perciba la necesidad de un sistema que genere documentos o presente información. La tecnología de NLG es una de las maneras posibles de cumplir sus necesidades, y sólo es apropiada si provee la solución más barata o más efectiva que cumpla los requerimientos para la generación de documentos o presentación de información.

Texto vs gráficos

Hay muchas situaciones donde las imágenes, diagramas esquemáticos, mapas, y ploteos pueden ser usados para comunicar información más efectivamente o más eficientemente que si se utilizaran textos. Cualquier persona que desarrolle un sistema para la presentación de información necesita considerar si la utilización de texto, gráficos, o alguna mezcla de los dos es la mejor manera de cumplir los requerimientos del usuario.

No hay principios perfectos para decidir cuando la información debería ser presentada de forma textual o gráfica. Un criterio que parece afectar esta decisión es el tipo de información siendo comunicada. En general, la información sobre un lugar físico puede ser mejor comunicada mediante una imagen en lugar de palabras. Sin embargo, puede ser difícil comunicar conceptos abstractos tales como la causalidad, de forma gráfica, mientras que esto se puede lograr de forma simple mediante un texto.

Otra cuestión que puede tener impacto sobre que modo de presentación es mejor es el nivel de experiencia de los usuarios que consumirán los textos producidos. En particular, aunque los gráficos son presentados algunas veces como algo fácil de entender, lo cual es adecuado para los novatos, investigaciones en psicología muestran que en muchos casos una cantidad considerable de experiencia y conocimiento es necesaria para interpretar un gráfico correctamente, y los novatos pueden desempeñarse mejor con una representación textual de la información. Esto en parte se debe al hecho de que las presentaciones textuales pueden explotar la acumulación de convenciones compartidas que los usuarios de un lenguaje comúnmente tienen (sin considerar si son novatos o expertos en un dominio en particular), un vocabulario de miles de palabras,

además de un rico conjunto de reglas sintácticas, semánticas, y pragmáticas para combinar e interpretar estos elementos. En el mundo de la representación gráfica, sin embargo, mientras los expertos en un dominio en particular pueden tener un rico conjunto de convenciones gráficas compartidas con respecto a un dominio específico, éstos pueden no ser conocidos por los novatos en el dominio; y convenciones gráficas independientes del dominio que son conocidas para el público en general, y que pueden ser asumidas en presentaciones que apuntan a novatos, son relativamente pocas.

Desde una perspectiva práctica, una buena manera de determinar si el texto o los gráficos deberían ser usados es examinar los documentos existentes que presentan la información en cuestión. Esto no es infalible, por supuesto, porque no hay garantía de que la manera en que las cosas se hagan sea la mejor posible. Sin embargo, en muchas áreas la tarea de presentación de información se ha convertido en un arte bien desarrollado, y así vale la pena seguir las buenas prácticas en el área. En algunos casos, la elección se dicta no a partir de que medio es el más efectivo, sino a partir de los requerimientos legales, convenciones, o distribución de restricciones. Por ejemplo, el uso de gráficos no será una opción si el material debe ser enviado via teletexto, un enlace lento de internet, o mediante una línea telefónica [CB91].

Generación de lenguaje natural vs autoría humana

Los sistemas de software son caros de construir, y los desarrolladores necesitan demostrar que el desarrollo de un sistema de NLG para producir documentos o presentar información es una mejor solución que contratar y entrenar a alguien que manualmente escriba los documentos o presente la información. Una gran parte de esta decisión se basa en la economía: ¿Es más barato contratar y entrenar a una persona, o crear y mantener un software? La decisión económica dependerá largamente en el volumen de texto producido. Un sistema de NLG que produce millones de páginas de texto por año será fácilmente justificable como efectivo con respecto al costo que un sistema que produce sólo cientos de páginas de texto por año.

Por supuesto que el costo no es el único factor influyente para decidir sobre la automatización de la tarea de producción de documentos, y además en muchos casos tampoco es el factor predominante. Otras razones para utilizar tecnolog´ıa de NLG son:

- Consistencia: Se puede requerir de consistencia entre los textos y los datos de entrada. Ya que los sistemas de NLG generan textos directamente desde los datos de entrada, siempre comunicarán los datos de forma precisa. Autores humanos, en cambio, puede cometer errores como resultado de un descuido o fatiga. Tales errores puedes ser detectados mediante un chequeo de calidad, pero esto puede ser caro y no detectar todos los errores.

- Conformidad con los estándares: En muchos casos, los documentos deben realizarse conforme a estrictos estándares de escritura y contenido. Los autores humanos pueden encnotrar muy difícil la tarea de escribir los textos constantemente de acuerdo a los estándares, pero un sistema de NLG puede ser programado para que obedezca tales estándares, dándole las reglas necesarias de gramática, estilo y contenidos.

- Velocidad de la producción de documentos: Por ejemplo, Canadá investigó sobre sistemas de NLG para producir actualizaciones de reportes del clima bajo condiciones severas tales como tornados o huracanes. La ventaja de adoptar un software para este caso, es que puede producir un reporte en segundos, mientras que un humano necesitará de varios minutos para escribir el mismo reporte. En cualquier caso, en una situación de clima severo, se puede aprovechar más el tiempo del humano para analizar los reportes.

- Generación de documentos en múltiples lenguajes: Algunos sistemas de NLG, pueden producir versiones de un texto en diferentes lenguajes. Esto permite a los usuarios producir documentos en lenguajes que ellos no conocen, sin necesitar la intervención de un traduc-

tor. Esto sólo es útil, por supuesto, si la calidad de los textos generados es suficientemente buena para remover cualquier necesidad de tener un humano que chequee los resultados.

- Mantenimiento del *staff* : Si un tipo particular de documento es muy aburrido de escribir, puede ser muy dif´ıcil contratar o retener a gente del *staff* que realize esta tarea. Los sistemas de computadoras no se aburren e incluso trabajan mejor en este tipo de tareas monótonas que no le gustan a la gente.

Por supuesto, hay también razones para preferir escritores humanos en lugar de tecnologías de NLG. Por ejemplo, si la utilización de un sistema de NLG requiere que la gente cambie la manera de trabajar o requiere que una organización cambie sus procesos, el sistema podría no ser aceptado a menos que se perciban beneficios muy altos.

Otra cuestión que puede hacer que la gente no quiera utilizar sistemas de generación de texto basados en computadoras tiene que ver con la responsabilidad del texto generado. En particular, en muchos casos un experto humano puede ser responsabilizado por errores en un documento producido por un sistema de NLG. A mucha gente no le gusta ser responsables de documentos que ellos no escribieron.

2.3. Generación de lenguaje natural en entornos virtuales

La Generación de Lenguaje Natural (NLG) puede ser utilizada dentro de entornos virtuales para generar instrucciones que ayuden a cumplir un propósito u objetivo. El entorno virtual que utilizaremos no es el primero ni el único que fue desarrollado para este propósito.

En la secciónes 2.3.1 y 2.3.2 veremos dos ejemplos de entornos virtuales que utilizan técnicas de generación de lenguaje natural. En 2.3.1 veremos principalmente como influyen las tareas de la generación de lenguaje natural en este entorno y en 2.3.2 veremos conclusiones interesantes acerca de los datos que se analizaron luego de varias partidas.

2.3.1. Generando instrucciones en entornos virtuales (GIVE)

El área de NLG es una nueva área de investigación, con un par de décadas de experiencia [RD00]. De todas formas es un campo de rápido desarrollo que ha puesto adelante técnicas prometedoras en los últimos años, en particular gracias a los desafíos comunes propuestos para la comparación de las actuales tecnologías de NLG. Uno de estos desafíos se llama GIVE [Cha] (*Giving Instructions in Virtual Environment*). GIVE es particularmente relevante para nuestro trabajo ya que aplica técnicas de NLG al problema de generar instrucciones en lenguaje natural en un mundo virtual 3D. Como resultado, varias técnicas investigadas por la comunidad de GIVE son directamente aplicables a este trabajo. En GIVE, jugadores humanos intentan resolver una búsqueda del tesoro en un mundo virtual 3D que no han visto antes. La computadora tiene la representación simbólica completa del mundo virtual. El desafío para el sistema de NLG es generar, en tiempo real, instrucciones en lenguaje natural que guiarán al usuario a la finalización exitosa de su tarea. Sólo el jugador puede efectuar cambios en el mundo, moviéndose por los alrededores, manipulando objetos, etc. La Figura A.1 muestra un *screenshot* de la vista del usuario en el mundo 3D. En la parte superior de la figura se muestra la instrucción actual dada por el sistema de NLG.

Las tareas y técnicas de la NLG que son relevantes para GIVE y para este trabajo son la determinación del contenido a través de *planning* [GK10], generación situada de expresiones referenciales y la gestión del *common ground* [CB91]. La tarea de determinación del contenido [RD00] consiste en decidir que información comunicar al jugador de tal manera que la información sea

Figura 2.1: La vista del jugador durante la competencia GIVE

adecuada en el momento actual de la interacción. Dicha tarea puede ser implementada usando la técnica de inferencia de *planning*. Como resultado, las instrucciones son relevantes para el objetivo del juego y causalmente apropiadas en el momento en que se expresan. El área de la generación situada de expresiones referenciales [GK10] provee algoritmos para dar con la descripción de objetos, así el jugador puede identificarlos (e.g. "la puerta de enfrente"), teniendo en cuenta propiedades estáticas (e.g. color) y dinámicas (e.g visibilidad) de un objeto. Finalmente, la gestión del *common ground* es la tarea de generar actos de *grounding* cuando sea apropiado de acuerdo al comportamiento del jugador. Los actos de *grounding* son expresiones que, en un sentido estricto, no agregan nueva información al discurso pero en su lugar refuerzan la información existente. Por ejemplo, si el sistema de NLG le dice al jugador "toma el kit verde" y el jugador gira ligeramente hacia la izquierda para ver un objeto, el sistema puede generar un acto de *grounding* positivo tal como "s´ı" si ese era el objeto correcto, o "no" si no lo era [Ben09]. Todas estas técnicas y algoritmos puede ser aplicados para generar pistas que transmitan información útil al jugador y que consideren la reacción del jugador con el fin de reforzar la información.

2.3.2. Aprendiendo el comportamiento y lenguaje social a través de jugadores en línea

La representación del *common ground* para escenarios que podemos encontrarnos todos los d´ıas es esencial para aquellos agentes que pretenden ser colaboradores y comunicadores efectivos. Los colaboradores efectivos pueden inferir los objetivos del *partner* y predecir acciones futuras. Los comunicadores efectivos pueden inferir el significado de expresiones basadas en el contexto semántico. Jeff Orkin y Deb Roy [OR07] introdujeron un modelo computacional del *common ground* llamado *Plan Network*, que es un modelo estad´ıstico que codifica patrones de comportamiento y lenguaje sensibles al contexto, con dependencias en roles sociales. Describe una metodología para el aprendizaje no supervisado de un *Plan Network* usando un videojuego multijugador, la visualización de esta red, y la evaluación del modelo aprendido con respecto al juicio humano sobre conductas típicas. Específicamente, describe el aprendizaje del *Restaurant Plan Network* sobre datos recolectados a partir de 5000 sesiones de un juego de rol de tipo MIMO (*minimal investment multiplayer online*) llamado *The Restaurant Game*. Los resultados demuestran un tipo de sentido común social por parte de los agentes virtuales y tienen implicaciones para la creación automática de contenido en el futuro.

Es muy relevante para nuestro trabajo porque continua demostrando que hay l´ımites para el rango de comportamientos que los *scripts* humanos pueden anticipar. Los *scripts* hechos a mano son frágiles cuando se encuentran con comportamientos no anticipados y es muy poco probable que respondan apropiadamente frente al gran rango de comportamientos exhibidos cuando los jugadores han recibido una serie m´ınima de instrucciones para jugar un rol en un en-

torno abierto. Además, los personajes manejados a través de *scripts* no tienen forma de detectar comportamientos inusuales.

Los resultados que se obtuvieron demuestran que es posible no sólo recolectar cantidades gigantes de datos de videojuegos multijugador, sino también datos de alta calidad que reflejan de manera precisa el comportamiento y lenguaje humano más típico. Con cientos de millones de personas jugando en l´ınea a juegos como *Second Life*[Sec], *World of Warcraft* [oW] o *The Sims*[Sim], el potencial uso de los videojuegos para enseñarle a personajes, controlados mediante inteligencia artificial, sobre los humanos, es enorme. Con respecto a nuestro trabajo, esta forma de recolección de datos nos puede ayudar providencialmente ya que podemos tener conocimiento sobre:

- Sectores del mapa más transitados por el jugador.
- Porcentaje de ejecución exitosa de las instrucciones indicadas por el agente.
- Tiempo de finalización del juego con y sin instrucciones del agente.
- Porcentaje de finalización exitosa del juego.

Existen limitaciones con respecto al enfoque de aprendizaje de planes. Un t´ıpico juego corto compuesto de lenguaje y comportamiento coloquial puede ser problemático. Los humanos evaluan a los juegos basándose en que ocurrió y que no ocurrió, y pueden considerar una partida como extraña debido a esto. El sistema necesita alguna representación de la estructura de alto nivel del juego para reconocer comportamientos fuera de lo común. Actualmente, se tiene acceso únicamente a la representación local de una secuencia esperada de eventos, no se puede identificar ausencias importantes.

2.4. Entorno del Juego

El juego que usaremos en esta tesis para desarrollar el *game partner* capaz de generar lenguaje natural es un t´ıpico *First Person Shooter* [Sho]. En este tipo de juegos el jugador ve al mundo a través de una cámara en primera persona, de este modo se siente inmerso dentro de él.

El jugador es personaje principal del juego y puede realizar las siguientes acciones:

- Disparar con sus armas al enemigo.
- Trasladarse a través del mapa. Esto incluye caminar, saltar y trepar escaleras.
- Recolectar items.

Los tipos de armas son los siguientes:

- Pistola: Es el arma con el daño más bajo pero contiene municiones infinitas.
- Cañon: Produce un daño muy alto pero tiene municiones limitadas. Para recargar este arma el jugador debe recolectar el item de munición en el mapa.

En la Figura 2.2 podemos verlas.

El mapa es una especie de fuerte medieval que se caracteriza por tener:

Figura 2.2: Armas del juego.

- Escaleras para subir y bajar los pisos.
- Muchas habitaciones comunicadas mediante puertas o puentes.
- Habitaciones sin salida, es decir, con una sola entrada.

En la Figura 2.3 podemos ver diferentes partes del mapa.

Figura 2.3: Mapa.

Los items se encuentran ubicados por todas partes en el mapa y una vez que el jugador los recoge se regeneran automáticamente. Los tipos de items y efectos que le producen al jugador son:

- Salud: Es una cruz roja que aumenta la salud del jugador. Si la salud ya está al máximo no se modifica.
- Veneno: Es una cruz verde que disminuye la salud del jugador. Si el jugador tiene la salud demasiado baja, tomar este item puede ser mortal.
- Munición: Es una caja amarilla que aumenta la munición del arma secundaria o cañon. Si el arma tiene munición máxima no se modifica.
- Rayos: Son items claves que recogidos en el orden correcto, vuelven vulnerable al enemigo. Los hay de 4 colores: rojo, azul, verde y violeta.

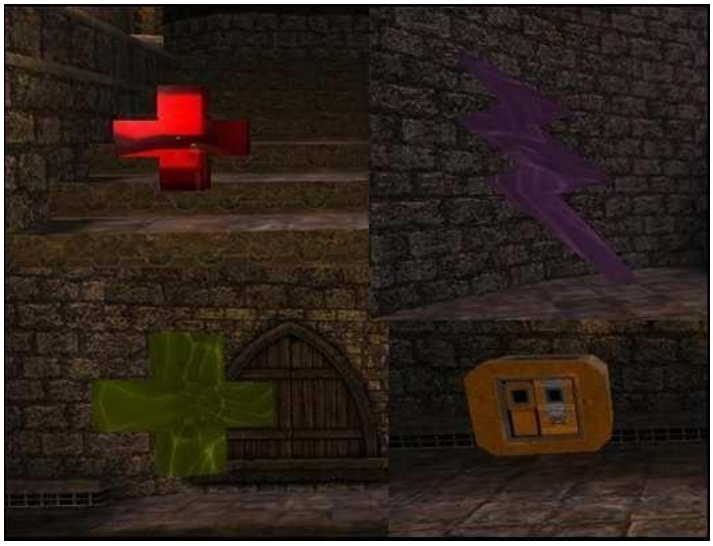

Figura 2.4: Items.

En la Figura 2.4 podemos ver los diferentes tipos de items.

El enemigo posee un arma con la cual produce daños al jugador. Las acciones que puede realizar son las siguientes:

- Perseguir al jugador cuando le dispara o se encuentra dentro del radio de detección.
- Atacar al jugador.
- Mantenerse quieto pero alerta de la presencia del jugador.
- Evadir al jugador para recuperar energ´ıas y as´ı poder volver a atacar. Esto sucede cuando la salud del enemigo se ve disminuida en gran parte.

Figura 2.5: Enemigo.

El objetivo del juego es eliminar al enemigo del fuerte medieval. Para tal fin el jugador posee

dos armas con las cuales le inflingirá daños al enemigo. El jugador pierde cuando su salud llega a cero, ya sea porque el enemigo lo eliminó o porque recogió demasiado veneno.

El enemigo es invulnerable hasta que el jugador recoja los items rayo en el orden correcto que se indica en la pantalla del juego. Una vez logrado esto, el enemigo se vuelve vulnerable y el jugador puede eliminarlo. Si el jugador recoge algún item rayo en orden incorrecto, debe volver a realizar el procedimiento desde el principio.

Mientras el jugador realiza la tarea de recoger los items rayo, el enemigo va a perseguirlo para intentar eliminarlo. El jugador podrá dispararle y producirle daños pero nunca matarlo. Cuando le produzca suficiente daño, el enemigo se dirigirá a algún punto del mapa para recargar energías y continuar con su propósito, eliminar al jugador.

Capítulo 3

Generación de lenguaje natural

En este capítulo veremos las tareas y técnicas de la generación de lenguaje natural que son relevantes para nuestro trabajo. Ellas son: la determinación del contenido a través de *planning*, la generación situada de expresiones referenciales y la gestión del *common ground*.

La tarea de determinación del contenido consiste en decidir que información comunicar al jugador de tal manera que la información sea adecuada en el momento actual de la interacción. Como resultado, las instrucciones son relevantes para el objetivo del juego y causalmente apropiadas en el momento en que se expresan. El área de la generación situada de expresiones referenciales provee algoritmos para dar con la descripción de objetos, así el jugador puede identificarlos teniendo en cuenta propiedades estáticas y dinámicas de un objeto. Finalmente, la gestión del *common ground* es la tarea de generar actos de *grounding* cuando sea apropiado de acuerdo al comportamiento del jugador. Los actos de *grounding* son expresiones que, en un sentido estricto, no agregan nueva información al discurso pero en su lugar refuerzan la información existente.

Todas ellas son muy importantes ya que inciden directamente sobre las instrucciones que queremos que nuestro *game partner* genere. De esta manera, las instrucciones serán más efectivas, eficientes y en definitiva, ayudarán radicalmente al jugador.

En las secciones 3.1, 3.2 y A.5.3 describiremos en detalle estas tres tareas de la generación del lenguaje natural, explicándolas e indicando la forma en la que nos ayudarán.

3.1. Determinación del contenido

Determinación del contenido es el nombre que le damos al problema de decidir la información que debería ser comunicada. Algunas veces la aplicación proveerá una especificación de la información a ser comunicada, pero en muchos casos el sistema de NLG será el responsable de seleccionar algún subconjunto apropiado de información disponible.

La elección del contenido que debería ser expresado en un texto depende de una variedad de factores, incluyendo al menos los siguientes:

- Diferentes objetivos comunicativos pueden requerir que se exprese diferente información. Por ejemplo, si el sistema de NLG es capaz de describir el clima durante un período y además de proveer definiciones y explicaciones de fenómenos meteorológicos, se requerirá de contenido muy diferente en cada caso.

- El contenido requerido puede depender de caracter´ısticas asumidas o conocidas por parte del oyente o lector. Por ejemplo, alguien que es considerado un novato en el dominio de aplicación puede requerir más información explicativa que alguien que es considerado un experto.

- Las restricciones sobre el texto final pueden jugar un rol importante en la determinación del contenido. Por ejemplo, puede ser necesario que el texto producido quepa dentro de un espacio restringido.

- La fuente de información juega un rol significante en la determinación de cual debería ser el contenido del texto.

Por último, las preguntas sobre la información que debería ser incluida en un texto y las circunstancias bajo las cuales debería ser incluida son muy dependientes de la aplicación. Obviamente, lo que es digno de reportar en el contexto de un sumario de clima está totalmente fuera de relación con, por ejemplo, aquello que es notorio en el contexto de la generación de una carta que incite al lector a dejar de fumar. Por esta razón, no es posible especificar reglas generales para la determinación del contenido, aunque existen principios definidos de forma vaga que uno podría utilizar como guía para la construcción de un mecanismo que determine el contenido apropiado, tal como *reportar lo que es significante* o *no colocar lo que es obvio o fácilmente inferible*.

En la sección 3.1.1 veremos diversos sistemas de generación de lenguaje natural que utilizaremos a lo largo del capítulo. En la sección 3.1.2 veremos características importantes como la selección, sumario, razonamiento y adaptación de la salida de datos. En la sección 3.1.3 veremos un procedimiento para el análisis de los datos. En la sección 3.1.4 veremos posibles formas de implementar este tipo de sistemas.

3.1.1. Sistemas de generación de lenguaje natural

En esta sección describimos distintos sistemas de generación de lenguaje natural que nos servirán para entender los conceptos que se presentarán. Los sistemas son:

- WEATHER REPORTER: Su propósito es proveer de reportes retrospectivos del clima durante períodos cuya duración es un mes calendario. Hace esto tomando como entrada un conjunto grande de datos numéricos recolectados automáticamente por dispositivos meteorológicos, desde los cuales produce textos cortos de uno o dos párrafos de longitud.

- FOG: Genera previsiones climatológicas en un texto a partir de simulaciones numéricas del clima producidas por una super-computadora y anotadas por un humano. Más precisamente, toma como entrada una predicción numérica de la velocidad del viento, tipo de precipitación e intensidad, y otros fenómenos meteorológicos varios sobre una región dada en un intervalo de tiempo determinado, y produce como salida un sumario representado en un texto que incluye dicha información.

- IDAS: Produce mensajes de ayuda online para usuarios de maquinaria compleja, usando información alojada en una base de datos que describe dicha maquinaria.

- MODEL EXPLAINER: Genera descripciones textuales de información en modelos de software orientado a objetos. Más especificamente, toma como entrada una especificación de un modelo de clase orientada a objetos y produce como salida un texto describiendo el modelo o porciones de el.

- PEBA: Es un sistema de generación de lenguaje natural que describe interactivamente entidades en una base de conocimientos taxonómicos mediante la generación dinamicá de documentos de hipertexto, presentados como páginas web.

- STOP: Es un sistema de generación de lenguaje natural que produce cartas personalizadas para dejar de fumar. La personalización está basada en un cuestionario de "Actitudes Hacia El Tabaco" que completan los fumadores; esto incluye preguntas sobre tópicos tales como problemas de salud, previos intentos para dejar de fumar, y sobre lo que le gusta o no a la persona con respecto al consumo.

- EPICURE: Genera recetas de cocina usando un modelo de acciones que se asume que el usuario sabe como realizar.

3.1.2. Aspectos de la determinación del contenido

Como mencionamos anteriormente, la determinación del contenido es el nombre que le damos al problema de decidir que información debería ser comunicada. Esto depende en mayor parte de la aplicación en particular que se desarrolla, su dominio, y el género de los textos que están siendo generados; esto hace posible especificar un algoritmo general para la determinación del contenido. De todas formas, la determinación del contenido incluye seleccionar, sumariar, y razonar con los datos. Muy pocos sistemas de generación de lenguaje natural simplemente generan mensajes que comunican todos sus datos de entrada. En su lugar, procesan estos datos de alguna manera, y este procesamiento es el valor agregado de la determinación del contenido.

Selección de los datos

Un tipo de procesamiento realizado en la determinación del contenido es la selección de un subconjunto de información en la base de datos del dominio del sistema o en la base del conocimiento para comunicar al usuario.

IDAS responde a preguntas cortas sobre un objeto, representado mediante preguntas como *¿DondeEsta?* y *¿CualesSonSusPartes?*. Aquí, la determinación del contenido típicamente incluye el uso de un conjunto de reglas que identifican, en la base del conocimiento del dominio, atributos específicos del objeto en cuestión que necesita ser comunicado para responder apropiadamente a la pregunta.

El objetivo de la selección del contenido puede ser resumido a algo como "Proveer información relevante"; lo que cuenta como relevante depende mucho del contexto.

Sumario de los datos

Otra actividad típica en la determinación del contenido es la creación de un sumario de alguna porción de los datos o conocimientos del sistema subyacente. Esto es necesario si los datos tienen una granularidad muy fina para ser reportados directamente o si alguna abstracción o generalización a través de los datos contituye la información interesante o importante.

Por ejemplo, desde la perspectiva de la determinación del contenido, la tarea principal en FOG es la elección de un conjunto pequeño de conceptos los cuales sumarian los aspectos más importantes de los datos numéricos del clima que usa como entrada. Por ejemplo, un conjunto de velocidades y direcciones del viento a partir de las 6 a.m. hasta la medianoche podría ser descripto como vientos del sureste de 15 a 20 nudos disminuyendo a finales de esta tarde. En este ejemplo, FOG ha sumariado un conjunto de datos de 38 números (velocidad del viento a las 6 a.m., dirección del viento a las 6 a.m., velocidad del viento a las 7 a.m., dirección del viento a las 7 a.m., y así sucesivamente) declarando que inicialmente el viento tiene dirección sureste y velocidad de 15 a 20 nudos, y que la velocidad disminuirá a finales de la tarde. Obviamente

tal sumario transmite menos información que la presentada en el conjunto original de datos; el objetivo es sumariar la información de tal manera que no reduzca su utilidad.

Razonamiento con los datos

Los procesos de selección y sumarización son sólo casos especiales de un proceso de razonamiento con los datos subyacentes. Otros tipos de razonamiento son también posibles. Por ejemplo, si STOP produce una carta para un fumador quien ha intentado dejar de fumar antes sin éxito, esto usualmente incluye en la carta generada un mensaje que intenta dar coraje al fumador mostrándole que la mayoría de la gente intentó varias veces dejar de fumar sin éxito al principio. Este razonamiento imita al razonamiento realizado por profesionales de la salud quienes trabajan con fumadores. Tales tipos de razonamientos se vuelven más sofisticados y específicos del dominio, por lo tanto es muy fructífero ver el desarrollo del proceso de determinación del contenido como el desarrollo de un sistema experto.

Adaptación de la salida para diferentes usuarios

Muchos sistemas de generación de lenguaje natural toman en cuenta al modelo de usuario cuando realizan la selección del contenido. Esto puede ser hecho o bien automáticamente o dándole al usuario control explícito sobre el proceso de adaptación.

Por ejemplo, si un usuario de EPICURE quiere saber como hacer una sopa de poroto, EPICURE le explica uno de los pasos como *Preparar los porotos* si cree que el usuario ya sabe como prepararlos. Por el contrario, si el sistema cree que el usuario no sabe como preparar los porotos, luego esta parte de la receta será descripta con algo como "Remojar, escurrir y enjuagar los poroto".

Una forma alternativa de adaptación es darle al usuario el control explícito sobre el proceso de determinación del contenido: en lugar que el sistema de generación de lenguaje natural haga asunciones incorrectas sobre la información que necesita el usuario, el usuario puede especificar directamente que contenido querría ver en el texto generado.

Algunos sistemas soportan diferentes tipos de usuarios implícitamente, siendo capaces de generar distintos textos de salida cada uno de los cuales apunta a una comunidad de usuarios distinta. El sistema FOG, por ejemplo, puede producir pronósticos generales que son dirigidos al público en general, y pronósticos marítimos, que son dirigidos a los marineros. Entre otras diferencias, los pronósticos marítimos dan más detalle sobre la velocidad y dirección del viento que los pronósticos para el público en general, debido a que la información detallada sobre el viento es importante para los marineros pero no usualmente para el público en general.

Desde una perspectiva más teórica, mucha gente ha argumentado que los sistemas de determinación de contenido deberían estar basados en un modelo donde el sistema primero intente reconocer el objetivo del usuario y luego planee una respuesta apropiada usando principios básicos sobre creencia y acción. Por ejemplo, Allen y Perrault (1980) describen un sistema de respuesta de un tren donde si el usuario pregunta,"Cuándo parte el tren de Glasgow", el sistema inferirá que el usuario tiene el objetivo de ir a Glasgow y el plan de lograr este objetivo tomando el próximo tren. Habiendo determinado esto, el sistema analiza este plan y se da cuenta que, con el fin de satisfacer su objetivo, el usuario necesitará tanto un número de plataforma como un tiempo de partida. Luego el sistema genera una respuesta que incluye ambas piezas de información.

3.1.3. Derivando reglas para la determinación del contenido

El objetivo de muchos sistemas de generación de lenguaje natural es producir documentos que sean tan similares como si fuesen producidos por humanos expertos. Cómo duplicar el rendimiento de los humanos expertos en dominios particulares es uno de los objetivos de los sistemas expertos, la dependencia del dominio de la tarea de determinación del contenido nos alienta a ver esto como un tipo específico de tarea de los sistemas expertos. Independientemente del tipo de tecnolog´ıa usada para implementar un sistema experto, una etapa clave en tal desarrollo es la adquisición de conocimiento. Desde la perspectiva de la ingeniería de software convencional, esto es parte del problema de análisis de requerimientos.

A continuación presentamos un procedimiento que puede ser usado para analizar un conjunto de textos representativos:

- Paso 1: Descomponer el texto en mensajes. Si es necesario, revisar el conjunto de definiciones de mensajes durante este proceso.
- Paso 2: Relacionar cada mensaje con la fuente de datos. ¿Transmite esto directamente los datos de entrada, o se realizó alguna sumarización o procesamiento? Si una sumarización u otro procesamiento es realizado, intentar determinar los algoritmos utilizados por los humanos expertos. Para este ejercicio, puede ser útil preguntarle a los expertos cómo crean los documentos.
- Paso 3: Será posible categorizar los textos que serán generados, en un número pequeño de tipos generales. Intentar caracterizar que clases de mensajes aparecen en que tipos de textos, e intentar determinar las condiciones bajo las cuales aparecen los mensajes. El resultado de este proceso será una colección de reglas de la forma “Un texto de tipo t contendrá un mensaje de tipo m bajo condiciones c1, c2, ..., c”. Como en el Paso 1, puede ser apropiado revisar el conjunto de definiciones de mensaje, y cualquier estructura taxonómica impuesta sobre este conjunto, durante este proceso.
- Paso 4: Discutir este análisis con expertos del dominio, y modificarlo apropiadamente. En la mayoría de los casos, esto requerirá una segunda pasada a través de los Pasos 1-4. Puede ser útil presentar las reglas de determinación del contenido a los expertos en una notación semiformal que los expertos puedan entender. En algunos casos, el análisis puede sugerir la modificacíon del corpus de texto objetivo, porque algunos textos son o bien subóptimos o dif´ıciles de generar.
- Paso 5: Cuando estés satisfecho con tu análisis, repite los pasos 1-4 con una selección más grande de textos del corpus de texto objetivo.

Las reglas identificadas mediante este procedimiento se convierten en una especificación para un proceso de determinación del contenido que indica que mensajes deberían estar contenidos en el texto en circunstancias específicas. Sin considerar si se utiliza el método descripto aquí o alguna otra técnica de adquisición de conocimiento, un componente esencial de la adquisión de conocimiento exitosa se familiariza con el dominio y con los datos, que aqu´ı se refiere al corpus. Se necesita una cantidad considerable de tiempo para aprender sobre el dominio, para analizar el corpus cuidadosamente, y para discutir y verificar las observaciones hechas con los expertos del dominio. El costo del recurso de esto no debería ser subestimado; sin embargo, no existe una alternativa fácil si el objetivo es construir un sistema que devuelva resultados apropiados.

En algunos casos, diferentes expertos del dominio pueden tener sustancialmente diferentes opiniones sobre lo que debería haber en las reglas de determinación del contenido. Se sugiere que si esto pasa, puede ser útil describir estas diferencias en términos de parámetros y luego preguntarle a los expertos que discutan los mejores valores para estos párametros.

3.1.4. Implementando la determinación del contenido

En muchos casos, la parte más difícil de construir un sistema para la determinación de contenido es determinar cuales son las reglas para determinar los mensajes que deberían ser incluidos en un texto. Una vez que estas reglas han sido determinadas, hay una variedad de maneras en que pueden ser implementados.

La forma más simple de implementación de determinación de contenido es escribir un código que aplique las reglas a los datos de entrada y produzca el correspondiente conjunto de mensajes. Esto puede ser hecho o bien en un lenguaje de programación convencional o en uno de los lenguajes especiales desarrollados por la comunidad de inteligencia artificial para soportar programación basada en reglas. Esto se utilizó en MODEL EXPLAINER (escrito en C++, un lenguaje de programación convencional) y en FOG (escrito en Prolog, un lenguaje para inteligencia artificial).

Algunas veces es útil descomponer los textos en un número de categorías y escribir reglas separadas para cada una, en lugar de un solo conjunto de reglas que las cubra a todas. Las reglas pueden ser compartidas entre múltiples categorías si es necesario, quizás utilizando un mecanismo de herencia. En este caso es además necesario escribir un conjunto adicional de reglas que determine la categoría que debería ser usada bajo qué circunstancias. Esto se utiliza en IDAS y STOP.

Hay muchas técnicas utilizadas para la construcción de sistemas expertos que podrían ser adaptadas para la tarea de determinación del contenido. Por ejemplo, los textos podrían ser generados mediante razonamiento basado en casos, u optimizaciones sofisticadas y algoritmos de satisfacción restringida que podrían ser utilizados para producir un conjunto óptimo de mensajes dado algún conjunto de restricciones. Ésta es un área de investigación relativamente no explorada en la generación de lenguaje natural.

3.2. Generación de expresiones referenciales

Cualquier dominio está poblado por entidades: éstas son las cosas de las que se puede hablar, ya sean concretas o abstractas. La generación de expresiones referenciales afecta a la forma en la cual se produce una descripción de una entidad que permita al oyente identificar esa entidad en un contexto dado. La generación de expresiones referenciales se relaciona con la NLG ya que la misma entidad puede ser referida de diferentes maneras. Esto es así tanto cuando una entidad es mencionada por primera vez (referencia inicial) como cuando la entidad es referida posteriormente luego de que ha sido introducida dentro del discurso (referencia posterior).

En el uso real del lenguaje, hay muchas maneras de introducir una entidad dada en un discurso. Una cuestión importante cuando una entidad es referida por primera vez es la elección de la perspectiva a partir de la cual la presentaremos. Cuando introducimos a alguien en una conversación, por ejemplo, podemos toparnos con la decisión de si debemos presentarlo como un lingüista de la computación, un visitante de otro continente, o alguien interesado en el Tai Chi. Estas o cualesquiera otras descripciones posibles serán seleccionadas dependiendo del motivo por el cual se introduce a la persona en la conversación y de la información que intentamos impartir posteriormente. De todas formas, virtualmente todos los sistemas de NLG operan sólo dentro de contextos limitados, y así la elección de la perspectiva no es generalmente una cuestión: el cuerpo de la referencia inicial está altamente restringido por el dominio de aplicación y así puede lidiarse con él de forma muy simple. Con una referencia posterior, el propósito es distinguir al referente de las demás entidades con las que puede confundirse. Las referencias posteriores comúnmente se abrevian de alguna manera, y el problema se convierte en cómo abreviar una referencia sin al mismo tiempo hacerla ambigua.

El proceso de la generación de expresiones referenciales tiene que considerar preguntas como las siguientes:

- ¿Puede y debería un pronombre tal como *él* o *ella* ser usado para referirnos a una entidad?
- ¿Debería ser usado un nombre propio, tal como *Glasgow*?
- ¿Deberían ser usadas frases tales como *el tren de Córdoba*, *el tren en la plataforma 12* o simplemente *el tren*?

En cada caso, necesitamos decidir sobre el contenido semántico apropiado de la frase que utilizaremos, para obtener el referente. Esto significa, seleccionar aquellas propiedades de la entidad que permitirán al oyente identificar la entidad de la que estamos hablando.

En la sección 3.2.1 veremos el mundo de las frases nominales y sus características. En la sección 3.2.2 veremos el concepto de referencia inicial y posterior y el rol del modelo de discurso. En la sección 3.2.3 veremos el procedimiento para generar pronombres en una frase, teniendo en cuenta el discurso anterior, como así también sus problemas. Por último, en la sección 3.2.4 veremos una serie de consejos a tener en cuenta para identificar referencias posteriores.

3.2.1. Expresiones referenciales y sus usos

Las entidades del dominio son generalmente referidas mediante frases nominales. Con el fin de ver lo que está involucrado dentro de la producción de expresiones referenciales, debemos investigar el mundo de las frases nominales.

Definición e indefinición

Una distinción importante puede ser hecha entre frases nominales definidas e indefinidas. Esta es una distinción en la forma, no en la función. Las frases nominales definidas son prototípicamente identificadas por el uso de adjetivos demostrativos, como en los siguienes ejemplos:

- *El tren* está por partir.
- *Este tren* partirá antes que el nuestro.
- *Aquel tren* partirá antes que el nuestro.
- *Aquellos trenes* partirán antes que el nuestro.

De todas formas, nombres propios y pronombres son también subcategorías importantes de frases nominales definidas:

- *Un tren* está por partir.
- *Algunos trenes* partirán antes que el nuestro.

Generalmente hablando, las frases nominales indefinidas se utilizan para introducir en el discurso a una entidad que no ha sido previamente mencionada (referencia inicial), mientras que una frase nominal definida se utiliza para referencias posteriores, refiriéndose a entidades que ya han sido introducidas en el discurso. De todas formas, hay excepciones a esta regla general; en

particular, las expresiones referenciales definidas pueden usarse como referencia inicial en varias circunstancias. Por ejemplo, una referencia definida puede ser usada para referirse a entidades cuya existencia puede asumirse conocida o inferible por el oyente, como en los siguientes ejemplos en contextos donde las entidades referidas mediante frases nominales italizadas no han sido mencionadas en el discurso anterior:

- ¿Puedes decirme dónde esta *la estación de subte*?
- *La estación donde abordé este tren* estaba desierta.

3.2.2. Requerimientos para la generación de expresiones referenciales

Dado un requerimiento para referirse a alguna entidad, el objetivo es hacer esto de tal forma que el oyente sepa sobre qué entidad se está hablando.

Referencia inicial y posterior

La función de la referencia inicial es introducir a alguna entidad dentro del discurso. La generación de expresiones referenciales para referencias iniciales está relativamente inexplorada en la literatura de NLG. Generalmente hablando, las entidades son introducidas dentro del discurso usando propiedades que puedan ser útiles posteriormente en el discurso, efectivamente indicando la etapa para lo que sigue. Sin embargo, es muy probable que los procesos involucrados aquí sean muy dependientes del dominio y la tarea, y así desde un punto de vista computacional la mejor estrategia es examinar textos en el dominio para ver cuales cuerpos de referencias iniciales se consideran generalmente útiles. Así, por ejemplo en un dominio donde las entidades tienen nombres propios, podríamos elegir utilizar siempre el nombre propio entero como referencia inicial, quizas junto con una frase nominal que indique propiedades que se consideran relevantes en el dominio de aplicación:

- *Tony Blair, el primer ministro británico*, se reunió...

En ciertos contextos puede ser apropiado introducir objetos físicos mencionando su ubicación:

- Tu deberías usar *la bicicleta en el patio*.

Las referencias posteriores, por otro lado se utilizan generalmente para referirnos a entidades que ya han sido introducidas en el discurso. Estas referencias son anafóricas en cuanto a que sus interpretaciones son dependientes del material precedente en el discurso; para el oyente, la interpretación de una expresión referencial anafórica involucra identificar a qué se está refiriendo, comúnmente mediante el reconocimiento del antecedente de la anáfora. El antecedente es alguna expresión referencial temprana que se co-refiere con la forma anafórica.

Para un sistema de NLG existen dos consideraciones que hacer. Por un lado, queremos evitar la ambigüedad, así que debemos decir lo suficiente para permitir que el oyente distinga la referencia de otras entidades con las que puede confundirse. Si hay dos trenes a punto de dejar la estación y le sugerimos a los oyentes que deberían abordar el tren que está a punto de salir, esto no es muy útil.

Por otro lado, deberíamos evitar la redundancia y la inclusión de información innecesaria en las descripciones. La forma descriptiva minimal que la mayoría de los lenguajes ofrecen como

recurso para esto, es el pronombre. Así podemos referirnos a una entidad mediante una expresión que virtualmente no provee contenido, usando las formas *él* o *ella*. Sin embargo en muchos contextos el uso de pronombres es ambiguo; consideremos el siguiente ejemplo:

- Tu puedes sentarte en la primera o segunda fila. Ella es para no fumadores.

Aquí, no queda claro a qué fila se está haciendo referencia mediante el pronombre *ella* en la segunda sentencia. As´ı, necesitamos direccionar dos cuestiones que se dirigen en direcciones opuestas: queremos decir tan poco como sea posible con el fin de no aburrir al lector, pero necesitamos decir tanto como sea necesario para permitir la identificación del referente.

El rol del modelo de discurso

Si un sistema debe determinar como una entidad debería ser descripta tal que pueda ser distinguida de otras entidades con las que puede confundirse, el sistema necesita tener alguna representación del contexto. Hay dos aspectos del contexto que son fuentes de distractores potenciales: el contexto f´ısico inmediato y el discurso anterior.

La mayoría de los trabajos en NLG han explorado la generación de expresiones referenciales con respecto a algunos contextos de discursos más que con respecto al contexto físico. Hay razones obvias para esto. Para un sistema computacional es más fácil tener acceso al modelo del primero más que el del segundo. Los aspectos relevantes del contexto del discurso son mantenidos en lo que llamamos el modelo del discurso. Éste almacena algunas representaciones de las entidades que han sido mencionadas en el discurso, quizás con información adicional sobre cuándo y cómo fueron mencionadas. Al usar esta información, el sistema de NLG puede determinar si una potencial expresión referencial es probable que sea ambigua.

El modelo de discurso más simple posible es una lista histórica de todas las entidades mencionadas a lo largo del texto. Luego, por ejemplo si dos trenes distintos han sido mencionados en el discurso anterior, y el sistema necesita hacer una referencia posterior a uno de ellos, el modelo de discurso deja claro que el tren en particular al que nos estamos refiriendo puede necesitar ser distinguido de otro tren que ya ha sido mencionado.

En efecto, el modelo de discurso es un intento para modelar el estado de atención del oyente: una representación de todas las entidades a las que está atendiendo actualmente el oyente, y así, cuales podrían ser los potenciales distractores para cualquier entidad que necesite ser identificada. Una observación común es que la cantidad de tiempo que ha transcurrido desde que una entidad fue mencionada por última vez, hace la diferencia. Con el tiempo nosotros olvidamos las cosas, y las entidades mencionadas más recientemente son más salientes que aquellas mencionadas hace un tiempo. Más específicamente dentro de la literatura psicológica, frecuentemente se hace una distinción entre memoria a corto y a largo plazo. Parece ser el caso en el que nosotros solemos recordar los detalles sintácticos del enunciado inmediatamente anterior, mientras que hay una tendencia a recordar solamente el contenido, y no el cuerpo de la expresión, del material más reciente en el discurso. Una observación nos dice que los pronombres tienden a referirse a entidades recientemente mencionadas, comúnmente dentro de la claúsula actual o anterior.

Los modelos de discurso intentan capturar estas diferencias imponiendo algunas estructuras sobre la información representada. Al principio, podemos observar una distinción entre aquella parte del modelo de discurso que representa el enunciado inmediatamente anterior y aquella que corresponde al discurso temprano, y as´ı podemos distinguir dos niveles de accesibilidad de entidades del discurso previamente mencionadas.

3.2.3. Generación de pronombres

Se ha observado que los pronombres se utilizan generalmente para referirse a entidades que han sido recientemente mencionadas en el discurso o son salientes de alguna otra forma en particular. En español y en otros lenguajes, los pronombres se distinguen por género (*él* vs *ella*), persona (*yo* vs *tu*), y número (*él* vs *ellos*). El pronombre en particular que se utilice depende de las propiedades que tenga el antecedente. El contexto sintáctico también impacta, ya que los pronombres varían de acuerdo a sus casos gramaticales (*él* vs *a él*).

La pronominalización puede estar sujeta a restricciones sintácticas como en los siguientes ejemplos:

- a. La manzana ten´ıa un gusano dentro de la manzana.
- b. La manzana ten´ıa un gusano dentro de ella.
- c. Ella ten´ıa un gusano dentro de la manzana.
- d. Ella ten´ıa un gusano dentro de ella.

Los ejemplos b y d son aceptables sintácticamente pero el a y el c no lo son.

Este conocimiento gramatical es claramente importante para un sistema de generación de lenguaje natural; pero la pregunta más fundamental es aquella que dice cuando un pronombre puede ser usado de forma segura para referirse a una entidad sin miedo a la ambigüedad en la interpretación. Ya hemos notado que los pronombres se utilizan comúnmente para referirse a entidades mencionadas en la misma sentencia o en la sentencia inmediatamente anterior. Aunque las instancias de la pronominalización a larga distancia pueden ocurrir, es algo muy inusual, por ejemplo usar un pronombre para referirse a una entidad que fue mencionada por última vez hace varias sentencias atrás, incluso en situaciones donde no han sido mencionados otros posibles antecedentes en el inter´ın.

Esta aparente restricción sobre la accesibilidad de las entidades previamente mencionadas por medio de pronombres tiene algunas ventajas. Esto sugiere que podemos desarrollar estrategias de pronominalización que no necesitan tener en cuenta el discurso pasado más allá de lo representado en lo que podríamos pensar como memoria a corto plazo. En primer lugar, podríamos sugerir que una estrategia apropiada de referencia pronominal puede ser capturada por la siguiente regla:

- Si la referencia fue mencionada por última vez en la sentencia anterior, luego usaremos un pronombre.

Esta simple regla parece funcionar en muchos casos tal como lo podemos ver en el siguiente discurso:

- a. El tren está partiendo a las 5 p.m.
- b. Éste llega a Córdoba a las 7 p.m.

Además manejará otros casos donde hay múltiples pronombres involucrados:

- a. Juan dijo que el tren está yéndose a las 5 p.m.
- b. Él piensa que éste llega a Córdoba a las 7 p.m.

Por otro lado, puede ser muy fácil generar referencias pronominales inapropiadas, como en el siguiente ejemplo:

- a. Tú puedes sentarte en la primera o segunda fila.
- b. Ésta es para no fumadores.

Un algoritmo más cuidadoso, usaría sólo un pronombre si ninguna otra entidad dentro del conjunto de potenciales distractores compartiera las mismas propiedades gramaticales. Sin embargo esto nos demuestra que, en realidad es muy restrictivo; la gente adicionalmente utiliza el dominio y el conocimiento del mundo cuando interpreta los pronombres, de tal manera las instancias de la pronominalización que podrían ser pensadas como ambiguas, no lo son generalmente:

- a. Susana invitó a María para la cena.
- b. Ella cocinó su platillo más sabroso.

Es posible, dados los contextos apropiados, forzar interpretaciones alternativas en cada caso; pero generalmente hay una interpretación por defecto con la cual estarán de acuerdo la mayoría de los lectores.

El importante rol del conocimiento del mundo en la interpretación de referencias anafóricas y las dificultades inherentes en el modelado del conocimiento del mundo denotan que no sabemos actualmente como construir un algoritmo que produzca referencias pronominales en todos y sólo aquellos casos donde los pronombres podrían ser usados y entendidos por la gente. Esto significa que los algoritmos de pronominalización de NLG cometerán errores, que pueden ser caracterizados como los siguientes:

- Falta de pronombres: el algoritmo decide no utilizar un pronombre cuando de hecho un pronombre ser´ıa perfectamente aceptado por el lector.
- Pronombres inapropiados: el algoritmo decide utilizar un pronombre de forma inapropiada, en casos donde el referente no está claro para el lector.

3.2.4. Generación de referencias posteriores

Supongamos que la entidad a la que necesitamos referirnos ha sido mencionada anteriormente en el discurso pero somos incapaces de utilizar un pronombre para referirnos a ella. En este caso, nos encontramos con la pregunta de cómo el referente puede ser distinguido de todos los otros posibles. Hay tres consideraciones que necesitamos tener en cuenta:

- La expresión referencial generada debe ser adecuada, de tal forma que provea suficiente información para localizar al referente.
- La expresión referencial debe ser eficiente, de tal forma que no nos provea de más información que la necesaria con el fin de identificar al referente.
- La expresión referencial debería ser sensible a las necesidades y habilidades del oyente, no haciendo uso de propiedades de la entidad que el oyente no pueda ser capaz de determinar.

Empezaremos caracterizando la tarea que consiste en obtener la descripción distinguible del referente. Una descripción distinguible es una descripción precisa de la entidad a la que nos estamos refiriendo pero no de algún objeto en el conjunto de contraste, que es el conjunto de las otras entidades a las que el oyente está prestando atención o se está focalizando. La constitución del conjunto de contraste estará determinada por el modelo de discurso. En el caso más simple, pueden estar todas las otras entidades mencionadas en el discurso.

Bajo este modelo, cada propiedad expresada en una expresión referencial puede tener la función de descartar miembros del conjunto de contraste. Supongamos que el emisor quiere identificar un perrito negro en una situación donde el conjunto de contraste consiste de un gran perro blanco y un gatito negro. El emisor podría elegir el adjetivo *negro* con el fin de descartar al perro blanco, y el sustantivo *perro* con el fin de descartar al gato. Esto resultaría en la generación de la expresión referencial *el perro negro*, que coincide con el referente pero no con otro objeto en el contexto actual. *El perrito* también sería una expresión referencial exitosa en este contexto.

Podemos observar que, en muchas situaciones donde el referente es un objeto f´ısico notorio visualmente, propiedades tales como el color, tamaño y posición tienden a ser muy útiles en la identificación de entidades. En otros contextos, por ejemplo donde el referente no es visible o es un objeto abstracto, otras propiedades pueden ser de uso convencional. Generalizando un poco, podemos sugerir que, dado un dominio, podemos identificar un conjunto convencionalmente útil de propiedades para utilizar en la construcción de expresiones referenciales. Como podrían surgir estas convenciones es un tema abierto al debate, pero claramente factores tales como la probabilidad de éxito sobre la base de uso en el pasado jugarán su rol. Así, por ejemplo, quizás porque hemos aprendido a través de la experimentación u otras maneras, podemos ver que utilizar la posición y el color son maneras útiles para identificar entidades físicas.

3.3. *Grounding* en las comunicaciones

Se necesitan dos personas para hacer un duelo, darse la mano, jugar ajedrez, enseñar o hacer el amor. Para tener éxito, ambos deben coordinar el contenido y el proceso de lo que están haciendo. Ellos no pueden empezar a coordinar contenido sin asumir una vasta cantidad de información compartida o *common ground*. Para coordinar en el proceso, necesitan actualizar su *common ground* momento a momento. Todas las acciones colectivas se construyen en base a la acumulación de *common ground*.

La comunicación es una actividad colectiva fundamental del ser humano. Cuando Alan habla con Bárbara, debe hacer más que simplemente planear y emitir enunciados, y ella debe hacer más que solamente escuchar y entender. Ellos deben coordinar el contenido. El discurso se desvanece rápidamente, y así Alan debe intentar hablar sólo cuando piensa que Bárbara está atendiendo e intentando entender lo que dice, y ella debe guiarlo dándole evidencias de su entendimiento.

En la comunicación, el *common ground* no puede ser actualizado apropiadamente sin un proceso, al cual llamamos *grounding*. En una conversación, por ejemplo, los participantes intentan lograr que lo que se ha dicho haya sido entendido. En nuestra terminolog´ıa, intentan hacer *grounding* sobre lo que se dijo, esto es, hacerlo parte de su *common ground*.

El proceso de *grounding* es algo muy básico para la comunicación, de hecho, para todas las acciones colectivas, as´ı que es importante entender como trabaja. Tomamos dos aspectos principales para esto: uno es el propósito, es decir, que es lo que las personas intentan lograr en su comunicación. La otra es el medio de la comunicación y las técnicas disponibles en él para lograr el propósito y cuales son los costos de utilizarlas.

En la sección 3.3.1 veremos un ejemplo de cómo funciona este proceso y el significado de criterio de *grounding*, así como también las fases en la cual se divide una conversación y qué aporta

el proceso de *grounding* en una conversación entre dos o más personas. Además veremos la diferencia entre evidencias positivas y negativas y el concepto de menor esfuerzo de colaboración. En la sección 3.3.2 veremos el *grounding* de referencias y el *grounding* del contenido verbatim. En la sección 3.3.3 veremos las restricciones y los costos del proceso de *grounding*. Por último, en la sección 3.3.4 daremos una conclusión al tema.

3.3.1. Introducción y definiciones

Consideremos el siguiente ejemplo:

- A: Ahora, tu y tu marido tienen un j-auto?
- B: Si tenemos un auto?
- A: As´ı es.
- B: No, no tenemos.

B indica con la frase *Si tenemos un auto?* que no ha entendido lo que A dijo. Sólo luego de que A ha contestado a su pregunta con *Así es* a lo que ella respondió con *No, no tenemos*, podemos decir que aparentemente ambos creen que han tenido éxito en la conversación.

El entendimiento no puede ser perfecto siempre. Asumimos que el criterio que la gente utiliza para entender las conversaciones es el siguiente: El contribuyente y sus compañeros mutuamente creen que todos entendieron lo que el contribuyente quiso decir con un criterio suficientemente bueno para el propósito actual. Esto se denomina criterio de *grounding* [CB91]. Técnicamente, *grounding* es el proceso colectivo mediante el cual los participantes intentan terminar con una creencia mutua.

Contribuyendo a la conversación

La mayor´ıa de las conversaciones comienzan con un potencial contribuyente que presenta un enunciado a su compañero/a. En nuestro ejemplo, Alan presenta a Bárbara la declaración *Ahora, tu y tu marido tienen un j-auto?*. Pero él no puede saber si tuvo éxito o no, a menos que ella provea evidencias de su entendimiento. Contribuir a la conversación generalmente se divide en dos fases:

- Fase de presentación: A presenta un enunciado U para que B considere. A asume que si B le preporciona una evidencia E o más fuerte, puede asegurarse que entendió lo que quería decir con U.
- Fase de aceptación: B acepta el enunciado U dando evidencia E asegurando que entendió lo que A quiso decir con U. B asume que una vez que A registre esa evidencia, también puede asegurarse que entendió.

Se necesita de ambas fases para que una contribución se considere completa. La fase de presentación puede volverse complicada, debido a que pueden cometerse errores. Una manera es utilizando auto-reparación. La presentación misma puede contener distintas contribuciones, cada una con sus propias fases de presentación y aceptación. El proceso de *grounding* se vuelve más evidente en la fase de aceptación. Luego de la presentación de algún enunciado U por parte de A, tenemos que B puede pensar que está en uno de estos estados para todo o parte de U:

- Estado 0: B no se dio cuenta que A enunció U.
- Estado 1: B notó que A enunció U.
- Estado 2: B correctamente escuchó U.
- Estado 3: B entendió lo que A quiso decir con U.

Evidencia en el *grounding*

Una vez que decimos algo en una conversación, todo lo que tenemos que hacer es buscar evidencia negativa, es decir, evidencia que nos diga que no se entendió lo que dijimos. Si encontramos alguna, reparamos el problema, pero si no, asumimos por defecto que se nos entendió. Pero si la evidencia negativa es lo único que buscamos, puede darse el caso que aceptemos información que tiene poca justificación para ser aceptada. De hecho, la gente comunmente busca criterios más altos. Como el modelo de contribución dice, la gente últimamente busca evidencia positiva de entendimiento. Veamos las tres formas más comunes de evidencia positiva y como funcionan:

Acknowledgements: Son la forma más obvia de evidencia positiva. Son denominados también respuestas del canal de retorno. Incluyen palabras como *uh*, *si*, *aja*. Además incluye gestos como el asentir con la cabeza. Son producidos generalmente sin que el que los realiza tenga que tener el turno para hablar.

Otra forma común de evidencia positiva es la iniciación del próximo turno relevante. Supongamos que A intenta hacerle una pregunta a B. Si B la entiende, puede contestar en el próximo turno. Preguntas y respuestas forman lo que llamamos pares de adyacencias, y una vez que la primera parte de un par de adyacencias está dicha, la segunda parte es condicionalmente relevante para el próximo turno. Así que A no sólo espera que B enuncie algo, sino también que conteste a su pregunta. Si B contesta la pregunta apropiadamente, también evidencia que entendió la pregunta de A. Si la respuesta no es apropiada, evidencia entonces que no la entendió.

¿Qué hace que el próximo turno sea apropiado o relevante? No es difícil decidir la segunda parte de un par de adyacencias, es decir, la respuesta a una pregunta, la respuesta a un pedido o la aceptación de una invitación. Una conversación generalmente se divide en secciones coherentes que tienen entradas identificables, cuerpos y salidas. Estas secciones son devotas a uno u otro proceso social, tales como hacer planes o intercambiar información. La mayoría de los turnos son designados para llevar este proceso adelante y dar evidencia sobre el entendimiento del emisor de la etapa previa en el proceso.

El próximo turno relevante provee evidencia positiva de entendimiento de la fase de presentación que le sigue, pero lo hace mediante la iniciación de la próxima contribución sin un corte. Así que aunque el proceso de aceptación pueda descontrolarse por varios turnos, usualmente termina cuando el compañero inicia el próximo turno relevante.

La tercera y más basica forma de evidencia positiva es la atención continua. En una conversación la gente monitorea lo que hacen sus compañeros momento a momento, en particular, si atienden o no. Si Alan presenta un enunciado mientras Bárbara no estaba prestando atención, él difícilmente asuma que ella lo entendió. Ella debe mostrar que estaba prestando atención, y una forma es a través de la mirada. Supongamos que estaba mirando hacia otro lado, Alan puede intentar capturar su mirada y además su atención mediante un enunciado. En el momento en que ella lo mire, Alan emitirá otro enunciado nuevamente. O también puede empezar un nuevo enunciado, pausarlo hasta que ella le preste atención y luego continuar. Las personas que hablan tienen varias formas de captar la atención.

La evidencia positiva de entendimiento se logra cuando la atención no se rompe o perturba. Alan tiene razones para creer que Bárbara lo sigue. Cuando ella se da vuelta para escuchar a

alguien más o tomar el teléfono, Alan tiene razones para creer que ha perdido su atención.

Menor esfuerzo de colaboración

A la gente aparentemente no le gusta trabajar más duro de lo que es necesario y en el lenguaje este principio también se cumple. El principio de menor esfuerzo lo podemos expresar mediante dos máximas:

- Cantidad: Hacer la contribución tan informativa para que cumpla su propósito, pero no más de lo que se requiere.
- Forma: Ser breve.

De acuerdo a ambas versiones, la persona que habla se supone que crea lo que llamamos enunciados apropiados, una vez que cree que va a ser totalmente entendido por los oyentes. A continuación describimos tres problemas que se presentan con este principio:

- Presión de tiempo: El emisor limita el tiempo y esfuerzo que invierte en planear y emitir cada enunciado, y frecuentemente realiza declaraciones inapropiadas. A menudo emite sentencias o frases que son inadecuadas y luego las repara, puede empezar una frase, pensar una mejor y empezar una frase diferente o crear partes inapropiadas en un enunciado. También suele invitar a los interlocutores a completar sus enunciados.
- Errores: El emisor frecuentemente emite enunciados inapropiados porque comete errores y tiene que repararlos.
- Ignorancia: El emisor algunas veces se da cuenta que no sabe lo suficiente sobre el interlocutor para emitir un enunciado apropiado, así que es forzado a emitir uno inapropiado en su lugar.

El principio de menor esfuerzo, debe ser reemplazado por el principio de menor esfuerzo colaborativo: En una conversación, los participantes intentan minimizar su esfuerzo colaborativo, es decir, el trabajo que ambos deben realizar desde de la iniciación de cada contribución hasta la aceptación mutua. Tal principio nos ayuda en varias cosas. Con respecto a la reparación, el emisor tiene dos preferencias fuertes: (a) prefiere reparar sus propios enunciados en lugar de que el interlocutor lo haga y (b) prefiere iniciar sus propias reparaciones antes que el interlocutor le diga. Aunque ambas preferencias tienen varias causas, el resultado es que minimizan el esfuerzo colaborativo.

3.3.2. El proceso de *grounding* cambia con el propósito

La gente en las conversaciones generalmente intenta establecer fines colectivos. Si están planeando una fiesta, ese puede ser su fin colectivo. Otras veces el propósito general podría ser intercambiar chismes, o aprender sobre alguna actividad. El proceso de *grounding* debería cambiar junto con estos fines. Si se trata de entender lo que el emisor quiere decir, con un determinado criterio, entonces el criterio debe cambiar a medida que cambian sus fines colectivos. Así también, las técnicas que se utilizan. Las técnicas deberían cambiar, por ejemplo, a medida que cambia el contenido de la conversación. De hecho, hay técnicas especializadas que han evolucionado para poder hacer *grounding* sobre diferentes tipos de contenido. A continuación detallamos dos tipos de contenidos:

Grounding de referencias

Muchas conversaciones se focalizan en objetos y sus identidades; cuando lo hacen, se vuelve crucial identificar los objetos de forma rápida y segura. Conversaciones como éstas se llevan a cabo, por ejemplo, cuando un experto le está enseñando a un novato a construir cosas, y ambos se refieren una y otra vez a piezas de construcción. El propósito de interés aquí es establecer una identidad referencial, esto es, la creencia mutua de que se ha identificado correctamente al referente. Hay varias técnicas comunes para establecer esto:

- Descripciones alternativas: Cuando el emisor se refiere a objetos, típicamente utiliza una o más expresiones referenciales, como por ejemplo, una descripción definida o indefinida, nombres propios, pronombres demostrativos o personales. Una forma para que los compañeros puedan demostrar que han identificado al referente o puedan verificar su identidad es mediante una descripción alternativa, como por ejemplo:

  - A: Bueno, aquel joven caballero del parque.
  - B: ¿Te refieres a Joe Wright? jajaja.
  - A: Sí, jajaja.
  - B: Pienso que el viejo Joe Wright fue quien caminó por primera vez allí, jajaja.

  A describe un referente cuando dice *Aquel joven caballero del parque*; B da evidencia de haber identificado al hombre mediante la descripción alternativa ofrecida; él agrega la entonación en la pregunta para obtener información sobre esa descripción; y A acepta esa descripción.

- Gestos indicativos: Cuando el emisor se refiere a un objeto cercano, los compañeros pueden dar evidencia positiva de que han identificado el objeto, al apuntarlo, mirarlo o tocarlo.

- Cuotas de referencia: Es importante establecer la identidad de un referente antes de decir algo sobre él. La razón es simple. Hasta que el referente haya sido identificado apropiadamente, el resto del enunciado será difícil, sino imposible, de entender. El emisor puede asegurar la referencia, al tratarla como una cuota del enunciado a ser confirmada separadamente. Tomemos como ejemplo esta charla entre un experto y un novato que están ensamblando una bomba:

  - S: Toma el tubo de salida, el pequeño que se parece a una lata de aceite.
  - J: Ok
  - S: Y colócalo en la apertura del otro tubo largo.

  En la primera línea, S dice *el tubo de salida, el pequeño que se parece a una lata de aceite.* y luego hace una pausa para obtener evidencia de que J ha identificado el referente. Él continua únicamente cuando la cuota ha sido entendida.

- Juicio de referencias: El emisor puede iniciar el proceso de *grounding* para una referencia en el medio de un enunciado. Cuando el emisor se encuentra a punto de presentar un nombre o descripción que no está seguro que haya sido entendida correcta o comprensivamente, la presenta junto con un marcador, seguida de una pausa pequeña, y espera que sus compañeros la confirmen o corrijan antes de completar la presentación. Consideren este ejemplo:

  - A: Lo llame a Bill a su casa y me atendió, ... Vanina??
  - B: Sí, Marina.
  - A: y Marina me dijo que Bill regresaba a la noche.

A aparentemente quiso decir *me atendió Vanina y me dijo que Bill regresaba a la noche*, pero al estar dudoso sobre el nombre Vanina, A lo presenta con un marcador. B confirma que conoce a quien se está refiriendo, pero luego corrije su nombre a Marina. A acepta esta corrección volviendo a representar a Marina y continuando. La correción entera se hizo de manera paulatina y eficiente.

Grounding del contenido verbatim

Algunas veces es importante registrar el contenido verbatim de lo que se dijo. Cuando un amigo nos dice su número de teléfono, hacemos más que solamente escucharlo. Tu intentas obtener su verbatim así puedes copiarlo o recordarlo hasta que marcas el número. Lo mismo pasa con los nombres, direcciones, títulos de libros, números de tarjetas de crédito y cuentas bancarias. A continuación presentamos algunas de estas técnicas:

- Muestras del verbatim: Cuando los vendedores llaman a los hogares para realizar una venta muchas veces confirman la numeración de la calle con la muestra del verbatim:
  - A: La dirección es fragueiro 1234.
  - B: 1234.
  - A: Es correcto.
  - B: Muchas gracias.

  B confirma el número que A le presentó, repitiendo su verbatim, “1234”.
- Cuotas: Cuando el emisor presenta mucha información a partir de la cual su verbatim debe ser registrado, generalmente la información es dividida en pedazos o cuotas, y muestra el verbatim de cada cuota. Por ejemplo:
  - A: Ah, ¿dónde vives ahora?.
  - B: Calle Urquiza 49.
  - A: Cuarenta y uno.
  - B: Nueve, nueve.
  - A: Calle Urquiza 49.
  - B: Primer piso A.
  - A: Calle Urquiza primer piso A.
  - B: S´ı.
  - A: Calle Urquiza 49 primer piso A.
  - B: S´ı.
  - A: Ok.

  B divide su dirección en pedazos repetibles y A le muestra un verbatim por cada uno. El emisor sabe como dividir la mayoría de los tipos de información en tales pedazos. Él lo hace de forma espontánea.

  La división de una presentación en cuotas está basada en el reconocimiento tácito de que las personas tienen una memoria inmediata limitada. Incluso las compañías telefónicas reconocen esto y dividen los números de teléfono en cuotas convencionales de tres o cuatro d´ıgitos.
- Deletreo: Para muchas palabras, el correcto contenido del verbatim se obtiene a través de un correcto deletreo. Por tal motivo el contribuyente deletrea las palabras dif´ıciles, como en el siguiente ejemplo:

- A: Su nombre es Iain, I A I N.
- B: Ok, ¿y su dirección?.
- A: Larrañaga 1234, L A R R A Ñ A G A.
- B: Ok.

O también existen otros trucos para obtener el deletreo correcto:

- A: Y mi nombre es Nicolás Persa, Persa como la alfombra.
- B: Ok.

Para resumir, las técnicas especializadas han evolucionado para el proceso de *grounding* de piezas especiales de una conversación. Cuando es crítico que una referencia sea bien establecida, la gente usará técnicas propias que están diseñadas para ese propósito. Cuando el contenido del verbatim es lo crucial, se usarán otras técnicas. De esta forma el proceso de *grounding* cambia dependiendo del propósito actual.

3.3.3. El proceso de *grounding* cambia con el medio

La gente debería realizar el proceso de *grounding* con técnicas disponibles en un medio que permita lograr el menor esfuerzo colaborativo. Consideremos el *acknowledgment okay*. En una conversación cara a cara o telefónica, *okay* constituye una evidencia de entendimiento y no una interrupción. En teleconferencia por teclado, es decir, cuando la gente se comunica a través de teclado y monitores, es dif´ıcil interpretar un *acknowledgment* de forma precisa, y enviar varios de ellos puede interrumpir a la otra persona que tipea, ya que se puede interpretar que la persona que lo envió, quiere pausar la comunicación para introducir nueva información, es decir, cambiar el turno.

En la sección 3.3.3 describiremos ocho restricciones que el medio puede imponer en la comunicación entre dos personas. En la sección 3.3.3 describiremos los costos de las distintas técnicas de *grounding* y su relación.

Restricciones del proceso de *grounding*

Aquí presentamos ocho restricciones que el medio puede imponer en la comunicación entre dos personas, A y B:

- Copresencia: A y B comparten el mismo entorno físico. En una conversacioń cara a cara los participantes usualmente se encuentran en el mismo entorno y pueden ver y oir lo que cada uno está haciendo y observando. En otros medios no tenemos tal posibilidad.
- Visibilidad: A y B son visibles el uno al otro. En una conversación cara a cara, los participantes pueden verse el uno al otro, y en otros medios no pueden. Ellos pueden verse el uno al otro, como en una videoconferencia, sin poder ver lo que el otro está haciendo u observando.
- Capacidad de audición: A y B se comunican mediante el habla. Cara a cara, por teléfono, y con algunos tipos de teleconferencia, los participantes pueden oirse mutuamente y percibir el *timing* y la entonación. En otros medios no pueden. Una máquina contestadora preserva la entonación, pero sólo a algunos aspectos del *timing*.
- Cotemporabilidad: B recibe el producto al mismo tiempo que A lo produce. En la mayor´ıa de las conversaciones, una emisión es producida al mismo tiempo en que es recibida y entendida, sin *delay*. En medios tales como las cartas y correos electrónicos, este no es el caso.

- Simultaneidad: A y B pueden enviar y recibir al mismo tiempo y simultáneamente. Algunas veces los mensajes pueden ser enviados y recibidos por ambas partes al mismo tiempo, como cuando el oyente sonríe durante una emisión. Otros medios son cotemporales pero no simultáneos, tales como el tipo de teleconferencias por teclado que transmiten caracteres sólo cuando la persona que tipea presiona el retorno de carro.
- Secuenciabilidad: Los turnos de A y B no pueden salirse de secuencia. En una conversación cara a cara, los turnos comúnmente forman una secuencia que no incluyen turnos intervinientes de diferentes conversaciones con otras personas. Con el e-mail, las máquinas contestadoras y las cartas, un mensaje y su respuesta pueden estar separados por cualquier número de mensajes o actividades irrelevantes; las interrupciones no tienen la misma fuerza.
- Capacidad de revisión de mensajes recibidos: B puede reveer los mensajes de A. El discurso se desvanece rápidamente, pero en medios como el e-mail, cartas, y mensajes grabados, una emisión se mantiene como algo que puede ser revisado varias veces por cualquiera de los participantes o incluso por una tercera parte. En las teleconferencias por teclado, las últimas emisiones se mantienen visibles en las pantallas por un tiempo.
- Capacidad de revisión de mensajes a enviar: A puede revisar los mensajes para B. Algunos medios, como las cartas y el e-mail, permiten al participante revisar una emisión de forma privada antes de enviársela a un compañero. En conversaciones cara a cara o telefónicas, la mayoría de las autoreparaciones deben hacerse públicamente. Algunos tipos de teleconferencia por teclado se encuentran en medio; lo que una persona tipea aparece en la pantalla del compañero sólo luego de cada retorno de carro, en lugar de letra por letra.

Costos del proceso de *grounding*

Cuando un medio sufre escasez de una de estas características, éste generalmente fuerza a las personas a usar técnicas alternativas para el proceso de *grounding*. Se hace esto porque los costos de las técnicas del proceso de *grounding* cambian. Describiremos once costos que cambian. Los primeros dos, costos de formulación y producción, son pagados por el que habla. Los próximos dos, los costos de recepción y entendimiento, son pagados por la otra parte. El resto son pagados por ambos. Enfatizamos que estos costos no son independientes entre s´ı.

- Costos de formulación: Cuesta tiempo y esfuerzo formular y reformular emisiones. Cuesta más planear emisiones complicadas, cuesta más retribuir palabras poco comunes, y más crear descripciones para objetos poco familiares. Por lo tanto, cuesta más formular emisiones perfectas.
- Costos de producción: El acto de producir una emisión por sí misma tiene un costo que var´ıa de medio en medio. Requiere poco esfuerzo (para la mayor´ıa de nosotros) hablar o gesticular, más esfuerzo tipear en un teclado de computadora o máquina de escribir y muchísimo más esfuerzo escribir a mano.
- Costos de recepción: Generalmente escuchar es fácil, y leer más difícil, aunque puede ser más fácil leer en pantalla las instrucciones complicadas o argumentos abstractos que escucharlos. Además cuesta tener que esperar mientras el emisor produce una instrucción. Este costo es relevante para nuestro trabajo ya que el agente debe tener en cuenta que el jugador receptará las instrucciones mediante la lectura, por lo tanto se deben tener en cuenta factores como la longitud de la instrucción y el tiempo que se mantendrá en pantalla.
- Costos de entendimiento: Es incluso más costoso para la gente entender ciertas palabras, construcciones, y conceptos que otros, sin considerar el medio. Los costos pueden elevarse cuando no se encuentran pistas contextuales, es decir, contenido que podamos obtener del contexto para que nos permita comprender lo que se quiere decir o describir. El e-mail, por ejemplo, no es ni cotemporal ni secuencial. Esto hace que el entendimiento sea más

dif´ıcil porque el receptor tiene que imaginarse contextos apropiados tanto para el emisor como para el mensaje, y recordar a qué fue en respuesta el mensaje, incluso cuando el campo del tipo de mensaje está presente. En nuestro caso, el jugador ve una gran parte del entorno y puede darse cuenta de lo que se le quiere comunicar. Este costo es relevante para nuestro trabajo ya que el agente debe utilizar el entorno disponible, en este caso el mapa y los objetos en él, como pistas contextuales para generar instrucciones que le permitan al jugador identificar objetos o entender la próxima acción que debe realizar.

- Costos de iniciación: Este es el costo de iniciar un nuevo discurso. Este es el costo de que B inicialmente note que A ha emitido algo y acepte que él o ella es el objetivo.

- Costos de *delay*: Son los costos de demorar una emisión con el fin de planearla, revisarla y ejecutarla más cuidadosamente. En las conversaciones cara a cara, como en todos los medios cotemporales y simultáneos, estos costos son altos debido a la forma en que los *delays* son interpretados. Este costo es relevante para nuestro trabajo, ya que una instrucción puede ser errónea o quedar obsoleta si el agente demora demasiado en mostrarla por pantalla.

- Costos de asincronización: En una conversación, la gente mide sus emisiones con gran precisión. Ellos pueden empezar una emisión precisamente en la finalización del turno del emisor anterior. Ellos pueden interrumpir una palabra en particular para mostrar acuerdo o desacuerdo en algún aspecto de ella. En medios sin copresencia, visibilidad, audibilidad, o simultaneidad, el *timing* es mucho menos preciso, y sin cotemporabilidad es además imposible. Así, las técnicas del proceso de *grounding* que se basan en la precisión del tiempo deberían incrementar su costo cuando la producción y recepción son asíncronas.

- Costos por cambio de emisor: En una conversación la regla general es *un emisor a la vez*, y ésta tiende a mantenerse para otros medios. Pero el costo por cambio de emisores varía con el medio. En conversaciones cara a cara es bajo. Los participantes encuentran fácil intervenir para que un emisor pare y otro comience. El costo de cambiar emisores es alto en medios como en las teleconferencias por teclado, donde los puntos de cambio de emisor no son fácilmente marcables o reconocibles.

- Costos de indicación: En las conversaciones cara a cara es fácil apuntar, tocar, o presentar un objeto a los interlocutores. Además es fácil observar a los interlocutores para mostrarles que estamos atendiendo o monitorear sus expresiones faciales. En las videoconferencias podemos usar sólo un rango limitado de gestos y no siempre podemos mirar a alguien y decirle “tu”. Mostrar imágenes es posible en medios tales como el video, las máquinas de fax y las cartas. Este costo es relevante para nuestro trabajo ya que la única forma de indicación que tiene el agente es mediante el texto, es decir, la claridad de las instrucciones y la esperanza de que el jugador las lea, no tiene otra forma de captar su atención.

- Costos de falla: Hay costos asociados con la producción de emisiones fallidas. Algunas fallas derivan en fallas de entendimiento, y es probable que las fallas en una emisión hagan obsoleta la próxima. Supongamos que el jugador no entiende la instrucción actual, si el agente interpreta erróneamente que lo hizo, entonces la próxima instrucción que muestre será inútil. El costo de estas fallas se incrementa con la gravedad de las mismas. Para evitar pagar el costo de las fallas, el emisor puede elegir pagar más en costos de formulación. Por ejemplo, colocando más detalles sobre el objeto o acción descripta, pero esto depende del medio. En una conversación, un receptor puede esperar fallas de un emisor porque la producción del discurso es espontánea, como lo es también en nuestro juego, que es en tiempo real. En el e-mail, las fallas no son fácilmente justificables, porque el emisor ha tenido la chance de revisarlo, y porque el daño hecho no es fácilmente reparable. Este costo es relevante para nuestro trabajo ya que el agente debe darse cuenta si el jugador entendió o no la instrucción, para así agregar más información y no continuar con la instrucción siguiente que quedaría obsoleta.

- Costos de reparación: Algunas reparaciones toman poco tiempo y esfuerzo; otras toman mucho, e incluso otras son imposibles de hacer. Ya que las fallas tienden a convertirse en una bola de nieve, el emisor debería repararlas tan pronto como sea posible.

Debemos tener en cuenta que los costos de *delay*, falla y reparación se relacionan íntimamente, pudiendo provocar costos altísimos. Supongamos que el agente tarda en mostrar una instrucción por pantalla, y en el momento en que la muestra el jugador ya realizó la acción que el agente quería indicar. Luego cuando la muestre, el jugador no va a entender lo que el agente quiere decir, realizando quizás una acción errónea, o molestándose por la instrucción que se le dio. Por lo tanto, el agente deberá reparar esa instrucción errónea generando así, una instrucción correcta.

3.3.4. Conclusión

El proceso de *grounding* es esencial para la comunicación. Una vez que hemos formulado un mensaje, debemos hacer más que sólo enviarlo. Necesitamos asegurarnos que ha sido entendido de la forma en que queremos. En cambio, tenemos poca seguridad que el discurso del que formamos parte procederá de forma ordenada. Para cualquier cosa que digamos, nuestro objetivo es lograr el criterio del proceso de *grounding* : que nosotros y nuestros receptores creamos mutuamente que hemos entendido lo que el uno y el otro han querido decir. Este es el proceso que hemos llamado *grounding*.

Las técnicas que hemos usado cambian tanto con el propósito como con el medio. Las técnicas especiales han evolucionado, por ejemplo, desde las referencias de objetos hasta para el contenido del verbatim de lo que se dijo. El medio difiere en los costos que impone en acciones tales como el *delay* del discurso, el inicio de un turno, el cambio de emisores, la producción de errores, y su reparación. Comúnmente, se pretende pagar tan pocos como sea posible.

La lección es que la comunicación es una actividad colectiva. Requiere la acción coordinada de todos los participantes. El proceso de *grounding* es crucial para mantener esa coordinación bien encaminada.

Capítulo 4

Diseño del agente generador de instrucciones

El *game partner* es el agente experto que guiará al jugador durante todo el juego a través de instrucciones que generará y dará a conocer de dos modos distintos: en un modo mostrará las instrucciones por pantalla y en el otro las reproducirá como audio. El proceso básico que debe realizar es: analizar el entorno, decidir la instrucción a verbalizar, verbalizarla y realizar el procedimiento nuevamente. Por lo tanto, su diseño e implementación es de vital importancia, en cuanto a rendimiento y facilidad de futuras modificaciones. De nada sirve un agente cuya implementación sea tan ineficiente que las instrucciones se generen de forma tardía, no cumpliendo para nada su propósito. Tampoco sirve un agente en el cual agregar nuevos tipos de instrucciones o chequear su cumplimiento, produzca constantes modificaciones en el diseño o arquitectura del sistema. Por todas estas razones es muy importante escoger un buen lenguaje de programación, buenas herramientas para la generación del entorno virtual, como por ejemplo, el motor gráfico. Además de todas las razones anteriores, es importante un diseño robusto para que el agente pueda comunicarse de forma eficiente y directa con los componentes del sistema de los que necesite extraer información para así elaborar y/o actualizar un plan y sus instrucciones.

En la sección 4.1 veremos las funciones que cumple un *game partner*, su definición y ejemplos para entender su comportamiento. En la sección 4.2 veremos las herramientas utilizadas para implementar el agente, el diagrama de clases y su descripción individual, y las propiedades del entorno virtual que inciden directamente sobre el agente. En la sección 4.3 veremos como se define el dominio y el problema de *planning*, a partir de los cuales el agente podrá interactuar con el planificador para obtener un plan propicio. Además explicaremos la interacción con los diversos componentes del sistema para poder extraer la información necesaria para chequear el entorno y así generar las instrucciones. En la sección 4.5 explicaremos las condiciones bajo las cuales el agente solicitará al planificador un nuevo plan debido a que el plan actual ha quedado obsoleto. Es importante mencionar que en el Cap´ıtulo 5 compararemos ambas modalidades de generación de instrucciones.

4.1. Desarrollando un *game partner* experto

La implementación de la toma de decisiones de los personajes del juego es una de las tareas más desafiantes cuando se diseña un juego; el *game partner* experto no es, de ninguna manera, una excepción a esta regla. En la mayoría de los juegos, el proceso de toma de decisiones de los jugadores es o bien implementado de forma ad-hoc que es mediante la creación de un *script* para ese juego en particular, o diseñado como una máquina de estados finitos que necesita ser pequeña

debido a problemas de escalabilidad [MF09]. Un *game partner* creíble no puede conseguirse de esta forma. Primero, el agente necesita reaccionar apropiadamente a las infinitas reacciones de los jugadores que no pueden ser predecidas y colocadas en un *script*. Y segundo, el agente necesita razonar sobre los complejos estados del juego hacia un objetivo dirigido con el fin de ser capaz de dar instrucciones que son causalmente apropiadas en el momento en el cual se expresaron y relevantes con respecto al objetivo del juego.

Proponemos utilizar un planificador para implementar el proceso de toma de decisiones del *game partner* experto. Los planificadores automatizados han alcanzado un nivel de madurez tal que pueden ser usados en aplicaciones de tiempo real incluso si la necesidad de replanificación es alta. En nuestra configuración hay una necesidad alta de replanificación debido a que el comportamiento del jugador es no determinístico e impredecible. Además, los planificadores son independientes del dominio, y con el fin de obtener un plan que gane el juego, sólo necesitan una especificación de las acciones que son posibles en el juego y una representación discretizada de lo que el agente sabe sobre el estado del juego. El conocimiento del agente sobre la localización de las regiones y adyacencias está discretizado utilizando los *waypoints* del juego [MF09].

En la sección 4.1.1 veremos el concepto de *game partner*, cómo puede ayudar al jugador y cómo muestra y se comunica con él mediante instrucciones. Por último, en la sección 4.1.2 veremos las propiedades que caracterizan a nuestro entorno de trabajo, es decir, al juego.

4.1.1. El *game partner*

El *game partner* es un agente experto que ayuda al avance del jugador en un *First Person Shooter* (FPS). El juego y las instrucciones generadas por el agente y mostradas en pantalla están ilustradas en la Figura A.8. Al igual que en la mayoría de los FPS, el jugador está situado en un mundo 3D donde puede realizar varias acciones tales como caminar, saltar, trepar escaleras, disparar y agarrar diferentes items los cuales tienen diferentes efectos. El objetivo del juego es eliminar a una criatura que vaga en el mundo 3D. La criatura no puede ser eliminada solamente disparándole porque tiene un mecanismo de autocuración que necesita ser desactivado mediante la recolección de una serie de rayos de poder en un orden determinado. El agente de NLG conoce la secuencia correcta de rayos así como también la posición de cada uno de ellos. También es posible reconocer otros items (tales como veneno, salud o municiones) y ser consciente de su efecto. Toda esta información está almacenada en la especificación que se le envía al planificador, y el planificador retorna una secuencia de acciones que, si fuesen ejecutadas en el estado actual del juego, lograrían que el jugador cumpla el objetivo del mismo.

Figura 4.1: Instrucciones que pueden ser generadas de acuerdo a las acciones posibles.

Los *screenshots* de la Figura A.8 ilustran como un plan obtenido por el planificador en nuestro juego puede ser usado para generar instrucciones. Podemos ver los *waypoints* visibles

que pertenecen al nivel del juego y sus adyacencias. [1]

Como la representación del mundo 3D que recibió el planificador está discretizada de acuerdo a estos *waypoints*, los arcos entre ellos pueden ser vistos como acciones en el plan. Intuitivamente definiremos la secuencia de acciones posibles en un estado dado del juego como la secuencia de acciones que son visibles en dicho estado. Por ejemplo, en la imagen de la izquierda las acciones posibles son *go down those stairs, go forward, go forward, go up those stairs, go forward*. Dada esta secuencia de acciones, el agente necesita decidir las acciones a verbalizar. En nuestro agente hemos decidido verbalizar la última acción en la secuencia que contiene una expresión referencial que identifica unívocamente al objeto involucrado en la acción. En este ejemplo, este es el caso para la acción *go up those stairs* ya que *those stairs* es una expresión referencial distinguible (dada la posición actual del jugador) para las escaleras más lejanas. Decimos que las primeras acciones llamadas *go down those stairs, go forward, go forward, go up those stairs, go forward* son *left tacit* [Ben07] y se espera que sean inferidas por el jugador dada las limitaciones causales del mundo (en términos más simples, con el fin de subir *those stairs* el jugador tiene que estar allí). La imagen de la derecha de la Figura A.8 (*turn around"*), ilustra una instrucción cuyo objetivo es cambiar la secuencia de acciones posibles.

La Figura A.9 muestra un caso de actos de *grounding* positivos *yeah, that's the ray that we need now* a la cual le sigue la instrucción *we need to find the green ray* cuando el rayo se vuelve visible. Esta pronunciación es un acto de *grounding* positivo porque, en un sentido estricto, no agrega nueva información al discurso; se supone que el jugador es capaz de averiguar que el rayo visible es verde y puede asumir razonablemente que ese es el único rayo en el mundo porque el agente uso la definitiva *the* al referirse al rayo, presuponiendo que el referente es único. De todas formas, el lenguaje natural es ambiguo y el jugador puede no haber hecho estos supuestos tal que el refuerzo logrado por el acto de *grounding* es, en efecto, útil. La otra imagen en la Figura A.9 ilustra un caso de acto de *grounding* negativo *no, don't take that now* al cual le sigue la instrucción *we need to find the green ray* cuando el jugador pasa el puntero del ratón sobre el rayo violeta. Por lo tanto, el agente advierte al jugador para que no reinicie el mecanismo de protección de la criatura como resultado de recoger un rayo incorrecto en la secuencia. Nuevamente, en un sentido estricto, el acto de *grounding* negativo no es necesario porque se le ha comunicado al jugador que el próximo rayo que debe recoger es el verde. De todas formas, el jugador puede no recordar esto y puede intentar tomar el rayo violeta que está directamente en frente.

Figura 4.2: Actos de *grounding* que pueden ser generados como reacción a las acciones del jugador.

Finalmente, los *screenshots* de la Figura A.10 ilustran la generación situada de expresiones referenciales que integra técnicas de la gestión del *common ground*. Por ejemplo, la expresión *the yellow thing is poison* es generada considerando que el veneno es el único objeto amarillo que está visible y puede ser identificado por el jugador. Por otra parte, la expresión es relevante para el jugador porque el juego no ha mostrado esto antes; en términos técnicos, el hecho de que

[1] Ni los *waypoints* ni sus adyacencias serán visibles en el juego, únicamente los mostramos para facilitar las explicaciones.

el objeto amarillo sea veneno no estaba en el *common ground* entre el agente y el jugador antes que esta expresión haya sido mostrada. Similarmente, *the green ray is behind the door on your right* se refiere teniendo en cuenta la posición relativa del jugador con respecto al rayo.

Figura 4.3: Expresiones referenciales que pueden ser generadas de acuerdo al conocimiento del jugador.

4.1.2. Propiedades del entorno de trabajo

Como mencionamos anteriormente, el *game partner* debe analizar el entorno antes de tomar cualquier decisión. Por lo tanto es muy importante que el *game partner* conozca muy bien el entorno con el que se va a encontrar. En nuestro caso, éste es muy espeífico y podemos definir una serie de propiedades para caracterizarlo. Las propiedad del entorno son las siguientes:

- El jugador es el agente principal y el enemigo, el agente secundario.
- La *performance* se medirá de acuerdo a la cantidad de items rayo que el jugador recolecte sin ser eliminado, como as´ı tambien la cantidad de instrucciones exitosas que tenga.
- El entorno es un mapa tridimensional que el jugador transita.
- Los actores son el arma que el jugador dispara y el jugador mismo que se mueve en el entorno.
- Los sensores son la cámara que se muestra a través de la pantalla que el jugador mira y el sonido que recibe del entorno.
- Parcialmente observable: Los sensores no nos muestran la totalidad del entorno. Por ejemplo la pantalla sólo nos permite ver una parte del mapa y no su totalidad.
- Estratégico: Los estados del entorno están determinados por las acciones del jugador y del enemigo, es decir, de otro agente. Por ejemplo, el jugador es quien decide donde moverse y si disparar o no, pero el enemigo puede detectar al jugador y tomar decisiones sin que el jugador se haya dado cuenta de ello.
- Secuencial: Existen estados irreversibles, por ejemplo, si el jugador muere entonces no puede llegar al estado final.
- Continuo: La cantidad de estados posibles es infinita, ya que depende de la posición y velocidad del jugador y el enemigo en cada momento. Por eso mismo también el tiempo es manejado de forma continua. Las acciones del jugador también son continuas (disparar, saltar, correr). Por último como la posición de la cámara varía constantemente debido al movimiento del jugador, también los sensores son continuos.
- Multiagente: Tanto el jugador como el enemigo son agentes.

4.2. Diseño del sistema

El diseño del sistema es muy importante ya que es la estructura que nos brindará la funcionalidad que necesitamos. Para ello debe cumplir con todas las buenas prácticas de programación.

En la sección 4.2.1 veremos la descripción de las herramientas utilizadas como así también la fundamentación de esta decisión. En la sección 4.2.2 describiremos cada una de las clases o componentes del sistema.

4.2.1. Descripción de las herramientas

El juego está implementado en C++ [Str00] y se utilizó el motor gráfico Irrlicht para renderizado y el *framework* Irrwizard como sistema base. Un *framework* es una estructura conceptual y tecnológica de soporte definida, normalmente con artefactos o módulos de software concretos, con base en la cual otro proyecto de software puede ser organizado y desarrollado.

Irrlicht es un motor 3D escrito en C++ y para utilizar tanto con C++ como con lenguajes .NET [Cha06]. Funciona tanto en Windows como en Linux, usa D3D [Lun08], OpenGL [Shr09] y su propio software de renderizado. La elección de este motor se vio influida en gran medida por su facilidad de uso.

Irrwizard es un *framework* orientado a juegos. Genera código fuente, archivos de proyecto y dll's necesarios para crear un *First Person Shooter* totalmente funcional. Elegimos este *framework* por su arquitectura robusta y entendible, a la cual podemos extenderla sin mayores problemas.

4.2.2. Descripción de las clases

En esta sección veremos las clases más importantes de nuestro sistema como así también sus interacciones. Como dijimos anteriormente es muy importante que el agente pueda comunicarse con otros componentes del sistema para obtener información del entorno y así elaborar planes y generar instrucciones, como así también que puedan introducirse nuevas instrucciones y comportamientos con una incidencia m´ınima sobre la arquitectura.

A grandes rasgos, el juego está dividido en diferentes estados, cada uno de los cuales representa un estado en particular del juego y cumple una tarea bien específica, como por ejemplo, mostrar el ménu principal, la pantalla de controles, de créditos, el nivel del juego, etc. De esta forma damos a cada estado una única responsabilidad, principio muy importante dentro de la programación orientada a objetos. A continuación describiremos las clases más importantes y explicaremos su relación con las demás.

La clase *GameManager* controla el flujo del juego. Además inicializa el sistema general, como por ejemplo el motor gráfico y demás configuraciones genéricas. Lo importante aquí es que el *GameManager* no conoce absolutamente nada sobre el estado del juego, sino que delega esta responsabilidad a cada uno de los estados o *GameState*'s como mencionamos anteriormente.

La clase *GameState* representa un estado del juego propiamente dicho. De aqu´ı derivan los estados propiamente dichos como por ejemplo el *IntroState* (pantalla de introducción al juego), el *MenuState* (menú principal), *ControlsState* (menú de los controles del juego), *CreditsState* (pantalla de créditos del juego), *LoadingState* (pantalla de carga del juego antes de comenzar el nivel) y *LevelState* (nivel del juego). Cada estado se inicializa a s´ı mismo y cuando lo cree oportuno cambia a otro estado el cual realiza el mismo procedimiento, liberando antes todos los recursos que utilizó.

La clase *ItemManager* se encarga de actualizar el estado de todos los items del juego, como así también controlar la recolección de los mismos. Básicamente su interfaz permite insertar items y conocer su estado. Cada *GameItem* representa un item del juego, clase de la cual derivan *HealthItem* (item de salud), *VenomItem* (item de veneno), *MunitionItem* (item para recargar las municiones del arma secundaria) y por último y el más importante *KeyItem* (item rayo).

La clase *Enemy* representa al enemigo del juego, del cual podemos saber su salud, posición, velocidad, y lo más importante cambiarle su propio estado. En nuestro juego decidimos que el enemigo puede estar en diferentes estados, representados mediante la clase *EnemyState*. Su funcionalidad es muy parecida a la de la clase *GameState* explicada anteriormente. De la clase *EnemyState* derivan diversos estados como por ejemplo *EnemyPursueState* (estado en el cual el enemigo persigue al jugador), *EnemyAttackState* (estado en el cual el enemigo dispara al jugador), *EnemyStandState* (estado en el cual el enemigo está parado vigilando la zona atentamente hasta que advierta la presencia del jugador), *EnemyEvadeState* (estado en el cual el enemigo hace una retirada luego que el jugador lo hiera considerablemente) y por último *EnemyDeathState* (estado en el cual el enemigo está muerto).

La clase *PathManager* es la responsable de almacenar el grafo de *waypoints* del juego. Como hemos mencionado en capítulos anteriores, este grafo está formado por *waypoints* y sus adyacencias, que son los lugares por donde puede transitar el enemigo, como así también donde pueden posicionarse los items del juego. Además tiene funcionalidades importantes como por ejemplo, saber el *waypoint* más cercano al enemigo o al jugador.

La clase *PlanGenerator* tiene como responsabilidad principal elaborar un plan. Como hemos visto, este plan depende de la especificación de un dominio y un problema en lenguaje PDDL. Por esta razón su interfaz permite ingresarle este tipo de datos y pedirle un nuevo plan, el cual es devuelto como una serie de pasos a seguir para llegar a cumplir el objetivo. Esta clase es la que interactua con el planificador propiamente dicho, parseando su salida y devolviéndosela al usuario de manera amigable.

La clase *Evaluator* es la encargada de hacer la evaluación estadística de cada partida. Constantemente testea el entorno, actualizando datos como por ejemplo la cantidad de *waypoints* por los que transitó el jugador, la cantidad de instrucciones dadas, etc. Luego de la finalización del nivel, escribe en un archivo los datos estadísticos mencionados, para su posterior evaluación.

La clase *Planner* es el agente propiamente dicho. Se encarga de enviar los datos necesarios al *PlanGenerator* para obtener un plan, se encarga de instanciar al *Evaluator* para que evalue la partida y escriba los datos, y por último, y más importante, se encarga de decidir bajo qué condiciones replanificará, cual será la próxima instrucción a verbalizar y chequear si se cumple o no.

Por último, tenemos la clase *Logger* que es utilizada para registrar datos en un archivo. Los datos que registramos son las acciones del juego en tiempo real, como por ejemplo, cambio de *waypoint* del jugador, recolección de un item, verbalización de una instrucción, cumplimiento de una instrucción, etc. Todos los datos los ordenamos cronológicamente.

4.3. Generación automática del problema de *planning*

El agente de *planning* necesita información sobre el entorno del juego para poder generar un problema de *planning*. El problema va a servir para que el planificador encuentre un plan que pueda ser utilizado por el agente. El agente de *planning* está representado por la clase *Planner*.

En la sección 4.3.1 veremos qué es un dominio de *planning*, cuál es su objetivo y cómo el agente obtiene esa información. En la sección 4.3.2 veremos que es un problema de *planning* y los distintos componentes que en este caso lo integran como los *waypoints*, items, personajes,

etc. Además veremos de donde extrae esta información el agente y como es el diagrama de flujo del sistema. Por último, en la sección A.4 veremos el procedimiento que sigue el agente para elegir la próxima instrucción a verbalizar, como así también los posibles problemas con los que se puede encontrar y como los resuelve.

4.3.1. Dominio de *planning*

Primero el agente debe obtener información sobre el dominio en lenguaje PDDL asociado al problema. Esta información se la envía la clase que maneja todo lo relativo al nivel del juego, es decir, la clase *LevelState*. La Figura 4.4 muestra lo que describimos anteriormente.

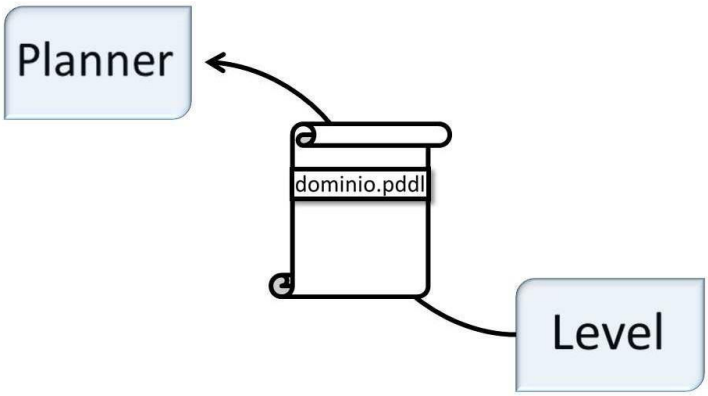

Figura 4.4: Información sobre el dominio en PDDL.

PDDL tiene como objeto expresar la física de un dominio, esto es, qué predicados hay, qué acciones son posibles, cuál es la estructura de las acciones compuestas, y cuáles son los efectos de las acciones. La mayoría de los planificadores requieren algún tipo de asesoramiento, esto es, indicaciones sobre las acciones a usar para lograr determinados objetivos.

4.3.2. Problema de *planning*

El agente necesita información del entorno para elaborar un problema de *planning* acorde y luego de obtener un plan, poder chequear si el plan se ha cumplido o no, y saber bajo que circunstancias se debe replanificar. En las siguientes secciones describimos comó y qué información extrae de su entorno.

Waypoints y adyacencias

El agente debe especificar en el problema la cantidad de *waypoints* que hay y como son las conexiones entre ellos. La clase *LevelState* le proporciona esta información como lo muestra la Figura 4.5.

Los *waypoints* y adyacencias siempre son fijos, por lo tanto el agente no necesita actualizar el problema de *planning* con respecto a esta información.

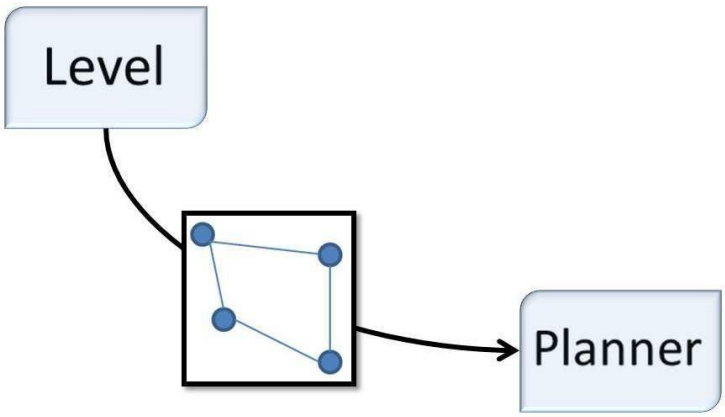

Figura 4.5: Información sobre waypoints y adyacencias.

Orden de recolección de rayos

El agente debe especificar en el problema la cantidad de items rayo que el jugador debe recolectar y además el orden. Nuevamente la clase *LevelState* le proporciona esta información como lo muestra la Figura 4.6.

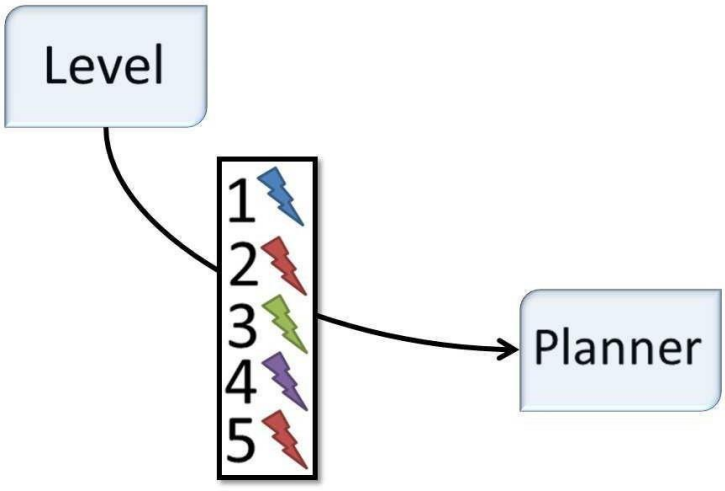

Figura 4.6: Información sobre orden de recolección de rayos.

El orden de recolección de rayos siempre es fijo, es decir, ni la cantidad ni el orden de los rayos cambia en una partida, por lo tanto el agente no necesita actualizar el problema de *planning* con respecto a esta información.

Items

El agente necesita información de los items que el jugador puede recolectar, como por ejemplo rayos, salud, veneno o municiones. El tipo de información que necesita es su ID (identificación), *waypoint* en el que se ubica y el tipo de item. La clase *ItemManager* le proporciona esta información como lo muestra la Figura 4.7.

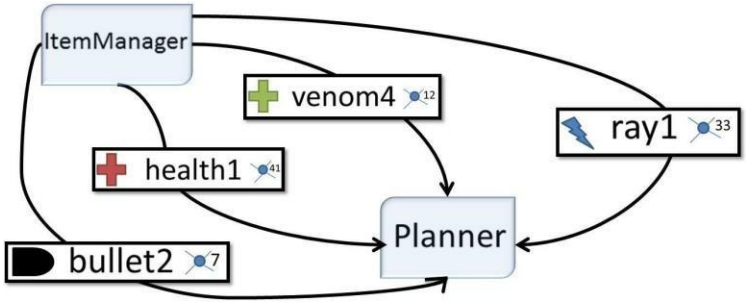

Figura 4.7: Información sobre items.

Esta información no sólo la necesita para especificar el problema de *planning*, sino también para monitorear el estado actual del plan. Cada vez que el jugador recolecta un item, sin importar su tipo, el agente actualiza el problema de *planning* con la nueva posición del item. Además, si el item recolectado es de tipo rayo, el agente actualiza el problema de *planning* de la siguiente manera:

- Si el rayo recolectado es correcto, el agente actualiza el problema indicándole en que posición de la secuencia de recolección se encuentra el jugador.
- Si el rayo recolectado es incorrecto, el agente actualiza el problema indicándole al jugador que debe comenzar nuevamente en la secuencia.

Jugador y enemigo

El agente necesita información sobre el jugador y el enemigo, las entidades principales de este juego. El tipo de información que necesita es su ID (identificación) y *waypoint* más cercano donde se ubica. La clase *PlannerFacade* le proporciona esta información como lo muestra la Figura 4.8.

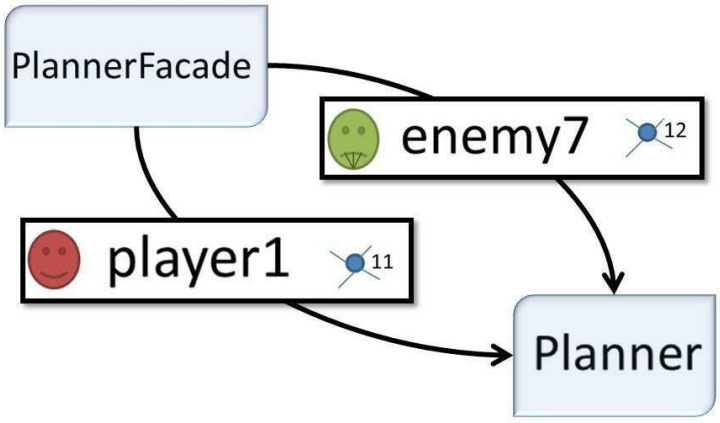

Figura 4.8: Información sobre el jugador y el enemigo.

Esta información no sólo la necesita para especificar el problema de *planning*, sino también para monitorear el estado actual del plan. El agente constantemente verifica el nivel de salud del jugador y del enemigo y actualiza su condición, es decir, vivo o muerto, en el problema de *planning*.

Objetos bajo la mira

El agente necesita información sobre los objetos a los cuales el jugador apunta con la mira de su arma. Esta información es muy importante para poder comunicarle al jugador datos útiles sobre los objetos, como por ejemplo, si apunta hacia un item de salud y su salud es baja, se le informará que recolecte ese item, o por el contrario, que no lo necesita en este momento. La clase *CollisionManager* proporciona esta información al agente. En la Figura 4.9 se muestran dos casos en los cuales el objeto está bajo la mira del arma y no lo está respectivamente.

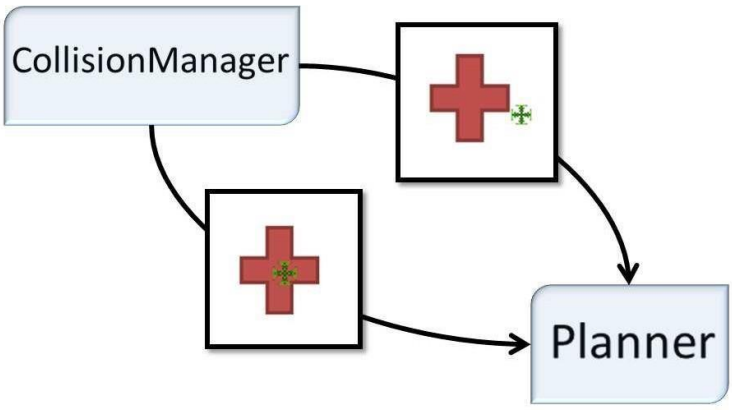

Figura 4.9: Información sobre objetos bajo la mira.

Esta información es irrelevante para el problema de *planning*.

Waypoints y objetos visibles

El agente necesita saber cuando un *waypoint* y/u objeto son visibles para el jugador, no sólo en la cámara sino también a 360 grados. La clase *PlannerFacade* le proporciona al agente dicha información como lo muestra la Figura 4.10.

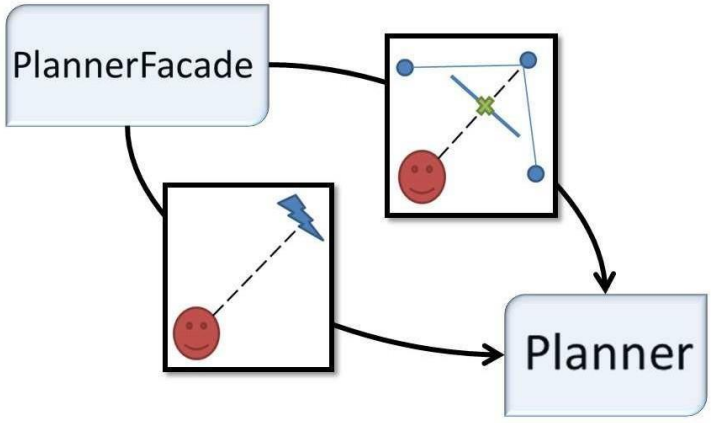

Figura 4.10: Información sobre *waypoints* y objetos visibles.

Esta información es irrelevante para el problema de *planning*.

Diagrama de flujo

Por último, presentamos un diagrama de flujo del sistema. En la Figura 4.11 se pueden apreciar las clases que le proporcionan datos de entrada al *Planner* y la instrucción de salida. Debemos tener en cuenta que este proceso se realiza cada 700 milisegundos, es decir, la instrucción se actualiza en pantalla luego de transcurrido este tiempo. El intervalo de tiempo anterior no fue elegido arbitrariamente, sino que se analizó la velocidad de los jugadores para leer las instrucciones en pantalla, sin que tampoco se muestren por demasiado tiempo en pantalla, fastidiando al jugador.

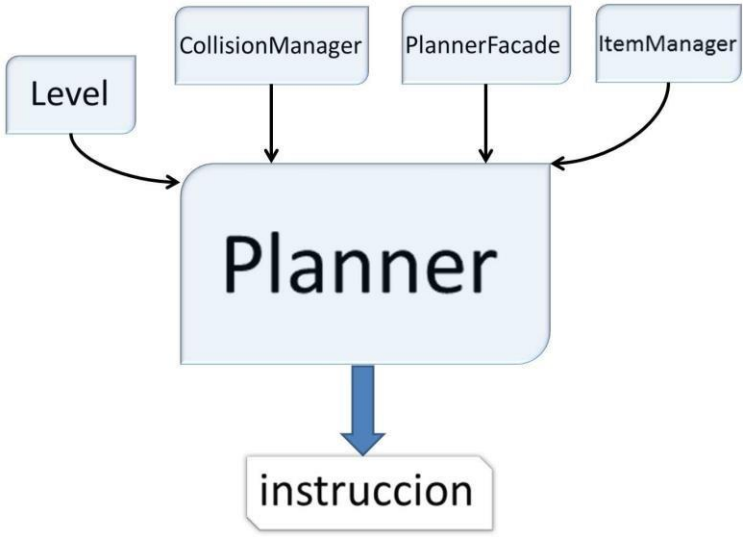

Figura 4.11: Diagrama del flujo para la generación de una instrucción.

4.4. Verbalizando la próxima instrucción

Esta sección explica como el agente usa el plan obtenido del planificador. El plan está compuesto por una serie de pasos que el jugador debe seguir para lograr el objetivo del mismo. Cuando el agente tiene un nuevo plan, debe decidir la próxima instrucción. Supongamos que el planificador nos provee de los pasos del plan que muestra la Figura A.2. Este plan contiene dos tipos de acciones, *MoveTo* y *TakeKey*. The primer tipo de acción indica que el jugador tiene que moverse de un *waypoint* a otro, y el segundo indica el rayo que el jugador debe recoger como también el *waypoint* donde se ubica.

Estos pasos tienen como objetivo recolectar el rayo *blueKey* como se ilustra en la Figura A.3. Ahora, el agente tiene el problema de decidir los pasos del plan que verbalizará, y para esto necesita hacer determinación del contenido basada en los pasos del plan. Notemos que si elegimos verbalizar todos los pasos del plan, tendr´ıamos un agente muy repetitivo y restrictivo. Debido a esto, hicimos que el proceso de determinación del contenido de nuestro agente dependa de las acciones cuyos *waypoints* son directamente visible para el jugador o visibles a 360 grados alrededor de él.

La FiguraA.3 muestra que hay sólo tres *waypoints* visibles pertenecientes al plan a 360 grados de la posición del jugador. Estos son los *waypoints* 4, 5 y 6. Los otros estan bloqueados por el entorno. De los *waypoints* visible, verbalizaremos aquél para el cual exista una expresion referencial que lo identifique unívocamente. En nuestro caso, los *waypoints* 4 y 6 son *waypoints* que están en el medio de habitaciones, por lo tanto, no existen expresiones referenciales que los describan

Figura 4.12: Pasos del plan generados por el planificador.

sin ambig uedad. Pero en el caso del *waypoint* 5, tenemos una puerta, y as´ı podemos usar una expresión referencial para identificar unívocamente al objeto involucrado. Como resultado, en la situación ilustrada en la Figura A.2 la instrucción *go through the door in front of you* será la que generemos.

Pero, Â¿qué pasa en el caso en que tenemos más de un objeto que puede ser identificado unívocamente?. La Figura A.4 muestra este caso. Hay siete *waypoints* del plan a 360 grados desde la posición del jugador, y tres de ellos tienen un objeto que puede ser unívocamente identificado: 5 (tiene una puerta pequeña), 7 (tiene una puerta grande), y 9 (tiene escaleras). En este caso, el agente genera una instrucción que se refiere al objeto visible más lejano en el camino del plan y verbaliza *Do you see those stairs in front of you?* antes de decir *Take them* (esta estrategia es usada cuando nos referimos a objetos que estan lejos del jugador).

Debemos considerar un caso más. Â¿Qué pasa cuando todos los objetos que pueden ser unívocamente identificados están detrás del jugador con respecto al camino de plan? La Figura 4.15 muestra este caso. Hay dos *waypoints* visibles (5 y 6) pertenecientes al plan a 360 grados desde la posición del jugador, y sólo uno de ellos tiene un objeto que puede ser unívocamente identificado, porque hay una puerta en el *waypoint* 5. El agente no puede generar una instrucción que referencie esa puerta porque el jugador estaría volviendo atrás en lugar de avanzar. Cuando tengamos casos como éste, el agente elegirá al último *waypoint* del camino del plan visible a 360 grados desde la posición del jugador, en este caso, el *waypoint* 6.

4.5. Condiciones de replanificación

El agente le mostrará por pantalla instrucciones al jugador para ayudarlo a cumplir el objetivo del nivel. El jugador puede seguirlas al pie de la letra o realizar cualquier otra acción que desee. En algunos casos, el agente puede guiarlo a seguir el plan ya calculado, pero hay casos en los que se necesita generar un nuevo plan, es decir, replanificar.

En la sección 4.5.1 veremos la replanificación a causa del cambio de posición de un rayo. En la sección 4.5.2 veremos el concepto de área de *waypoints* visibles, camino actual del plan, *waypoints* visibles del plan y la replanificación según los conceptos mencionados.Por último, en

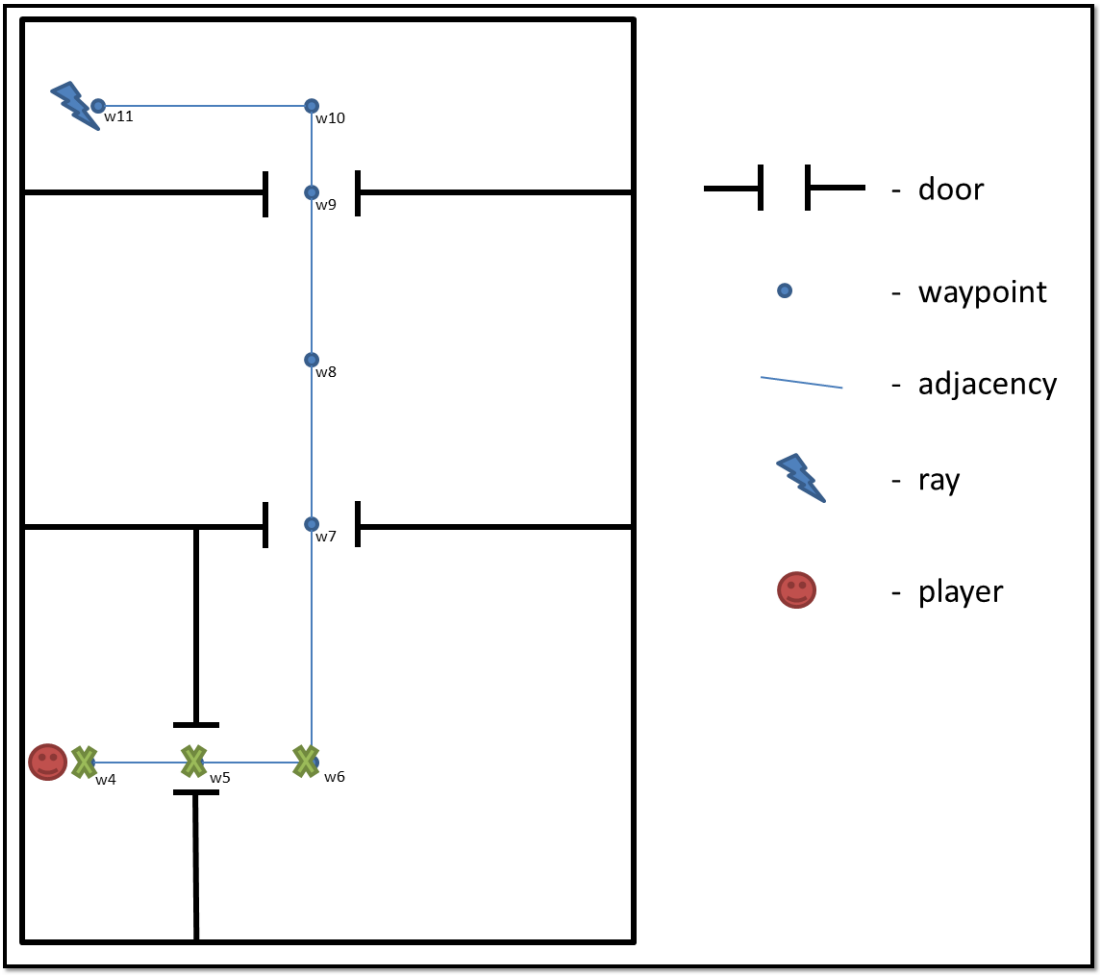

Figura 4.13: Camino del plan de la Figura A.2 con los *waypoints* visibles destacados (sólo uno referible).

la sección 4.25 veremos la importancia de los actos de *grounding* que a pesar de no agregar nueva información, refuerzan la actual.

4.5.1. Replanificación debido al entorno cambiante

La Figura 4.16 muestra la posición inicial de los *waypoints*, sus adyacencias, la posición de los rayos y el jugador. Como podemos ver, tenemos 4 items rayo: azul, verde, violeta y rojo y supongamos que la secuencia correcta para desactivar el mecanismo de defensa de la criatura es rojo, azul y rojo. Cuando el jugador recolecta algún objeto del juego, luego el objeto se reposiciona en otro *waypoint* de manera aleatoria (que puede ser el mismo en el que estaba antes).

Como el planificador cree que las posiciones de los objetos son estáticas, por lo tanto calcula un plan de acuerdo a ese criterio. En la Figura 4.17 podemos ver el plan que obtuvimos del planificador.

Este plan será correcto siempre y cuando el rayo rojo sea reposicionado en el mismo *waypoint* luego de su recolección. En la Figura 4.18 podemos ver que el plan obtenido es incorrecto porque el rayo rojo fue reposicionado en otro *waypoint*.

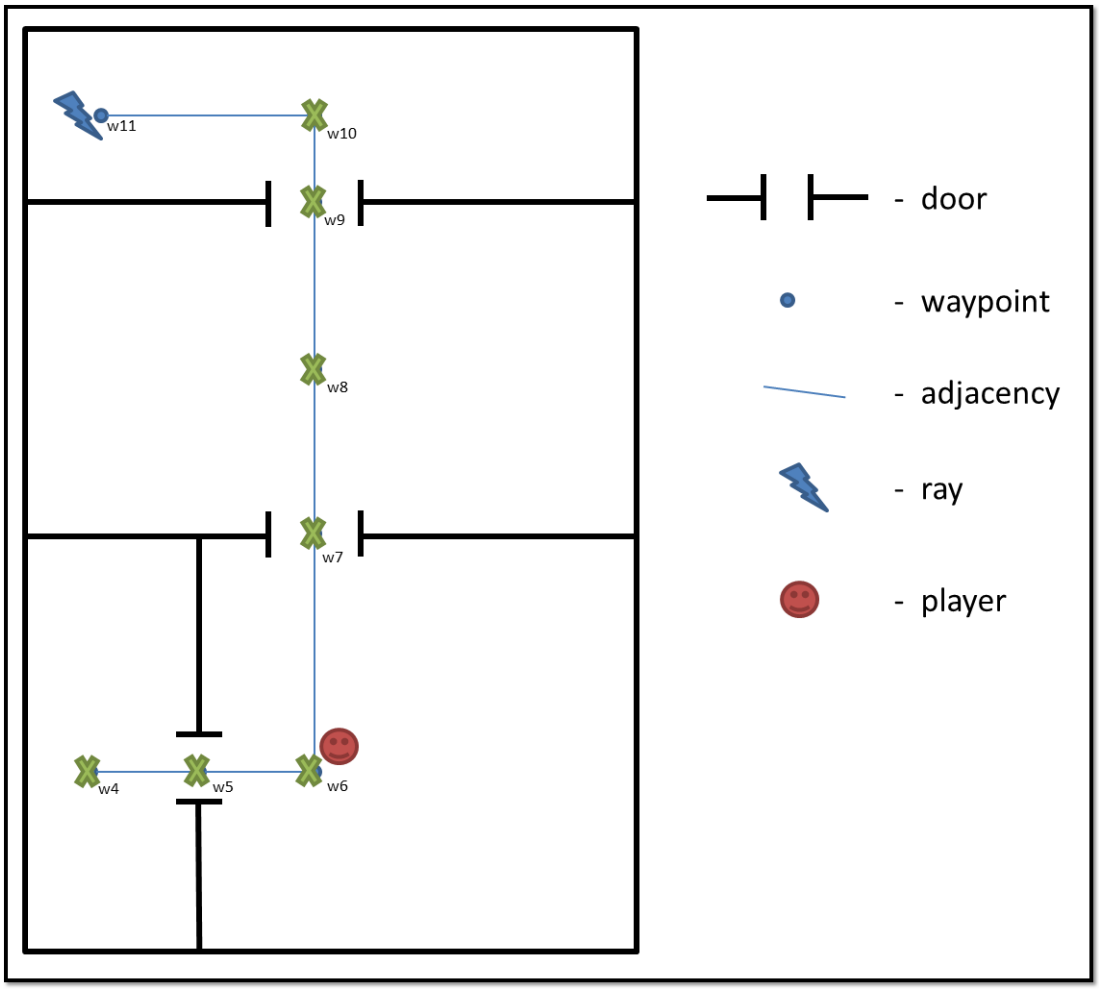

Figura 4.14: El camino del plan de la Figura A.2 con *waypoints* destacados que son visibles a 360 grados (todos referibles unívocamente).

Además puede darse el caso en el que el jugador se equivoca y recolecta un rayo incorrecto, reiniciando as´ı la secuencia de rayos a recolectar para desactivar el mecanismo de defensa. La Figura 4.19 muestra este caso. Por lo tanto, el agente realizará una replanificación cuando la posición de algún item rayo haya cambiado.

4.5.2. Replanificación debido al comportamiento impredecible del jugador

Área de *waypoints* visibles.

Se denomina área de *waypoints* visibles al conjunto de *waypoints* que son visibles desde la posición del jugador en una vista de 360 grados. En la Figura 4.20 y Figura 4.21 podemos ver un escenario de ejemplo y el área de *waypoints* visibles respectivamente.

Para calcular el área de *waypoints* visibles utilizamos una función del manejador de colisiones. La función toma dos coordenadas tridimensionales y chequea si existe algún objeto que se interponga en el segmento entre dichas coordenadas. Por lo tanto, iteramos sobre la función

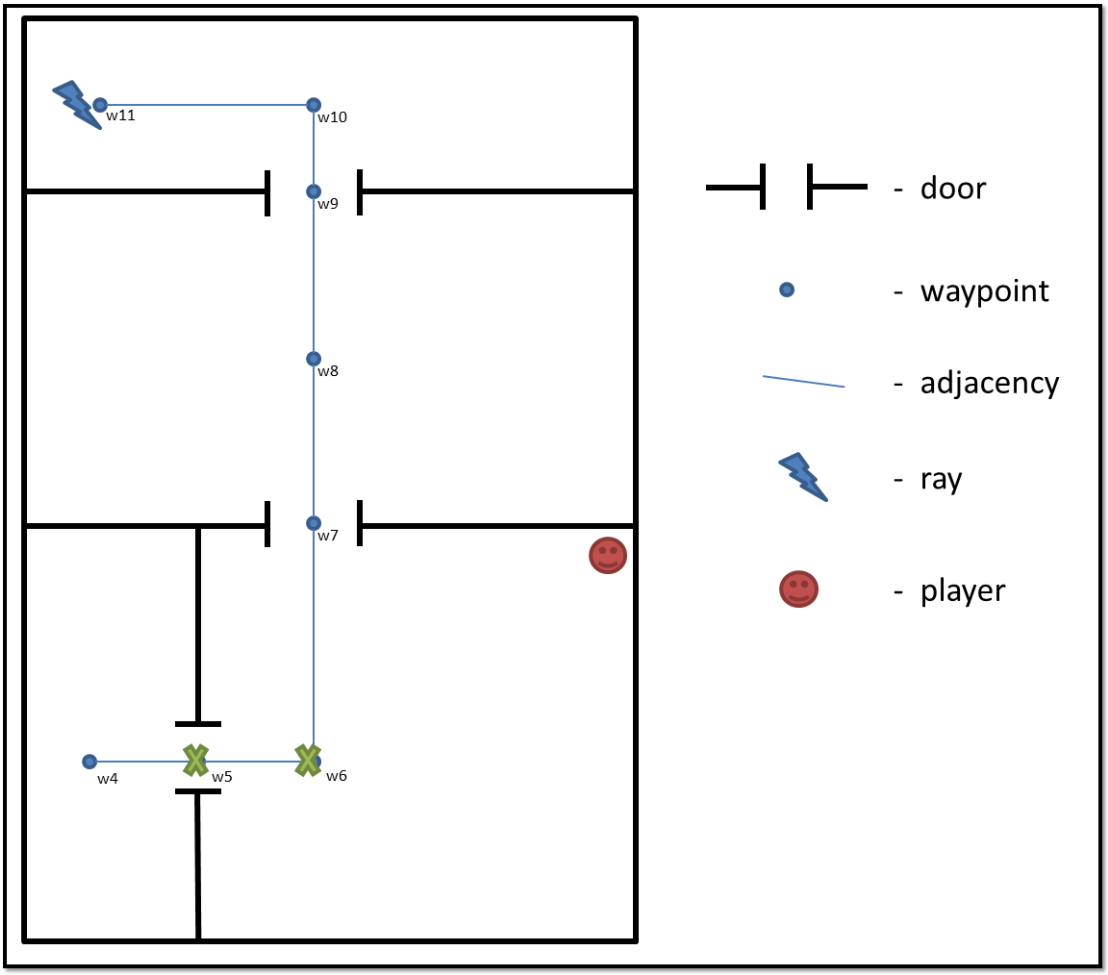

Figura 4.15: *Waypoints* visibles del plan a 360 grados desde la posición del jugador.

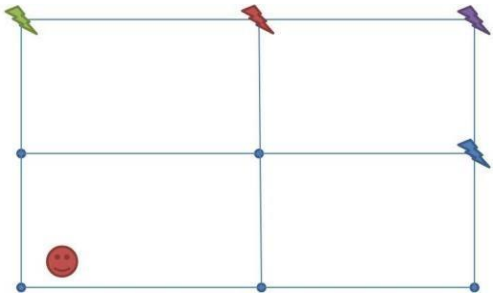

Figura 4.16: *Waypoints*, adyacencias y rayos.

tomando como parámetros la posición del jugador y cada uno de los *waypoints* y agrupamos aquellos cuyo chequeo haya resultado negativo.

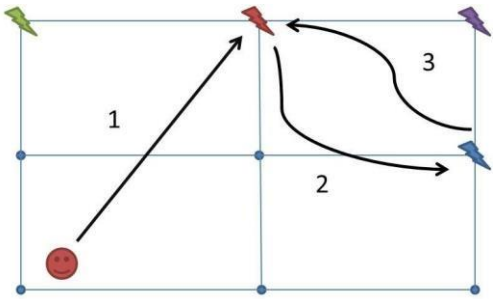

Figura 4.17: Plan obtenido de acuerdo a posiciones estáticas.

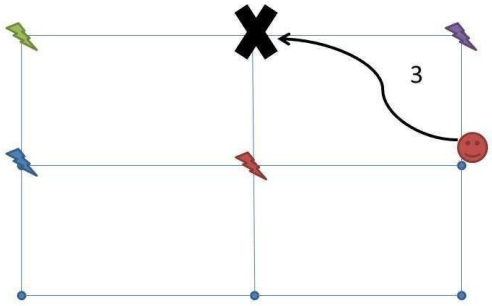

Figura 4.18: Plan Incorrecto.

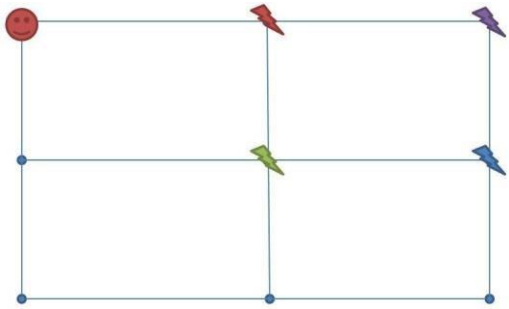

Figura 4.19: El jugador recolectó el rayo verde.

Camino actual del plan

Se denomina camino actual del plan al camino más corto formado por *waypoints* y adyacencias, mediante el cual el jugador, desde su posición actual y en orden, deberá recorrer para llegar al próximo item rayo. La Figura 4.22 muestra el camino actual del plan entre el jugador y el rayo verde.

Waypoints visibles del plan

Se denominan *waypoints* visibles del plan al conjunto de *waypoints* del camino actual del plan que pertenecen al área de *waypoints* visibles. La Figura 4.23 muestra los *waypoints* visibles del ejemplo anterior.

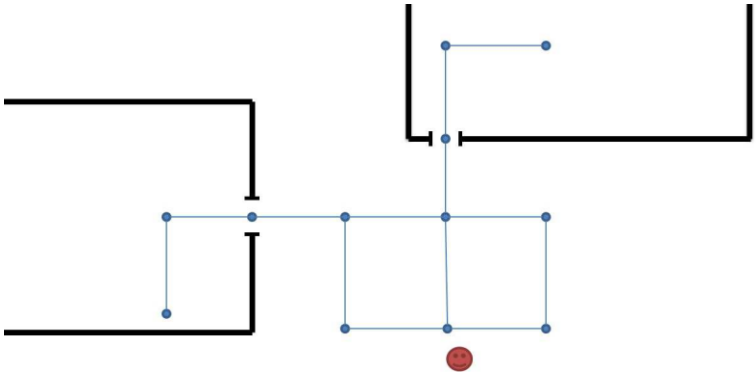

Figura 4.20: Escenario de ejemplo.

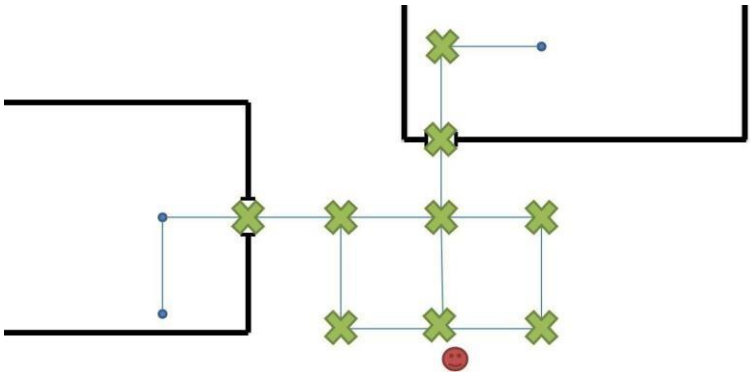

Figura 4.21: Área de *waypoints* visibles.

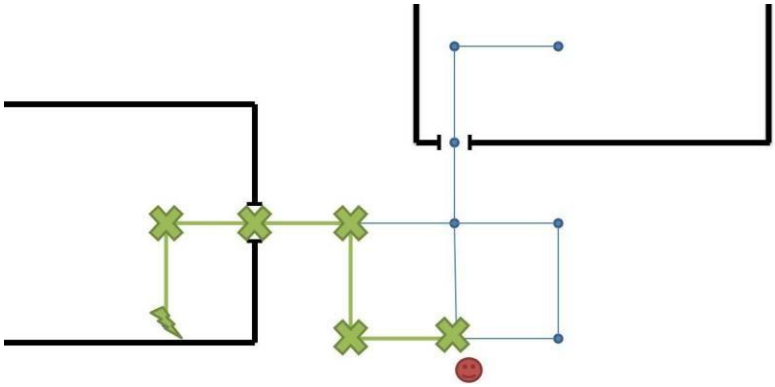

Figura 4.22: Camino actual del plan entre el jugador y el rayo verde.

Por lo tanto, el agente realizará una replanificación cuando no existan *waypoints* visibles del plan. La Figura 4.24 muestra un ejemplo donde la replanificación es necesaria.

4.5.3. La importancia de los actos de *grounding*

Los actos de *grounding* son enunciados que, en un sentido estricto, no agregan nueva información al discurso pero en su lugar refuerzan la información ya conocida. Los actos de *grounding* no

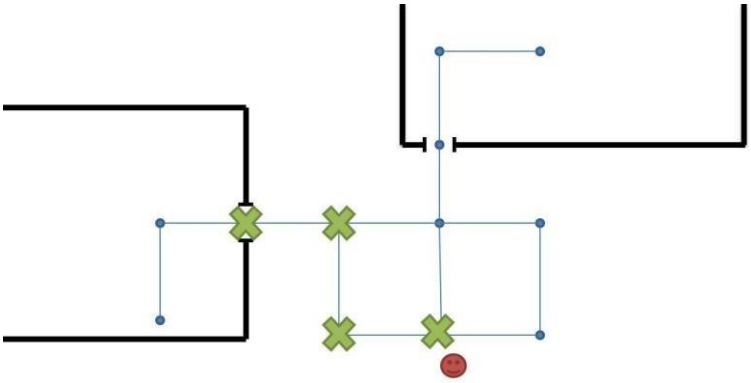

Figura 4.23: *Waypoints* visibles del plan.

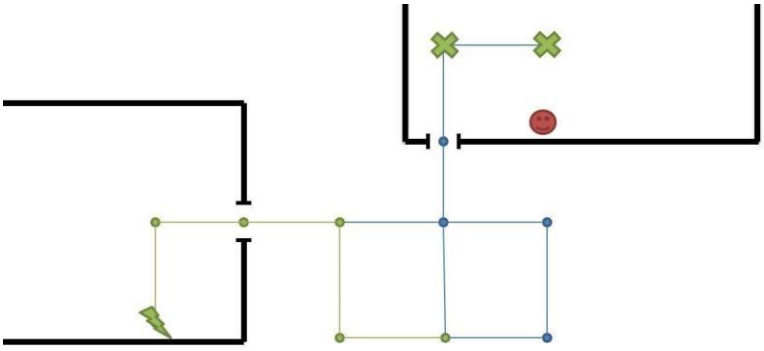

Figura 4.24: Ejemplo de replanificación.

son requeridos desde un punto de vista informacional en la comunicación, pero de todas formas, juegan un rol importante con el fin de lidiar con el entorno cambiante, y el comportamiento impredecible del jugador.

La Figura 4.25 muestra el caso de un acto de *grounding* positivo, *Get this ray* que le sigue a la instrucción *Turn left to see the ray* cuando el rayo se vuelve visible. Este enunciado es un acto de *grounding* positivo porque no agrega nueva información al discurso. El jugador debería ser capaz de notar que debe recoger el rayo, de otra forma, el agente no lo hubiera guiado hasta él. De todas formas, el lenguaje natural es ambiguo y el jugador podría no llegar a esta conclusión, es por esto que reforzamos la información a través del acto de *grounding*.

El otro *screenshot* en la Figura 4.25 ilustra un caso de actos de *grounding* negativos con *This is not the ray you need*, que le sigue a la instrucción *We need to find the green ray*, cuando el jugador pasa el cursor del ratón sobre el rayo rojo. Con esta instrucción, el agente previene al jugador para que no reactive el mecanismo de protección de la criatura, como resultado de recolectar el rayo incorrecto. Nuevamente, en un sentido estricto, el acto de *grounding* negativo no es necesario porque el jugador ya ha sido avisado que el próximo rayo que debe recolectar es el verde. De todas formas, el jugador podría no recordar esto e intentar recolectar el rayo rojo que está enfrente de él.

Los *screenshots* en la Figura 4.26 ilustran un ejemplo típico de creación de *common ground* entre el agente y el jugador. El *screenshot* superior muestra la instrucción *Do you see that ray in front of you?*. Con esta instrucción, el agente pretende que el jugador focalice su atención en el rayo que está enfrente de él. Como resultado se espera que el jugador se acerque al rayo. En esta nueva situación, el agente genera la instrucción *Pick it up*. Está claro que con esta instrucción

Figura 4.25: Actos de *grounding* que pueden ser generados de acuerdo a la reacción del jugador

el agente pretende que el jugador recolecte el rayo azul enfrente de él, pero podemos ver que ni el rayo ni su posición fueron incluidas en la instrucción. Esto se puede hacer porque esta información ya está en el *common ground* entre el jugador y el agente.

Finalmente, el *screenshot* inferior de la Figura 4.26 ilustra la generación situada de expresiones referenciales que integra técnicas del manejo del *common ground*. La expresión *That is venom! be careful* se genera considerando que el veneno es el único objeto visible y por lo tanto puede ser identificado por el jugador. Más aún, la expresión es relevante para el jugador porque no se ha dicho antes en el juego; en términos técnicos, el hecho de que ese objeto amarillo es veneno, no estaba en el *common ground* entre el agente y el jugador antes de que dicha expresión haya sido mencionada.

Figura 4.26: Varias instrucciones que crean y utilizan el *common ground*.

Capítulo 5

Evaluación

En este capítulo describiremos el proceso de evaluación de nuestro agente generador de instrucciones. El objetivo de la evaluación es testear nuestro sistema con humanos y ver los resultados que obtenemos. Para esto usaremos métricas objetivas y subjetivas, mediante las cuales obtendremos datos mucho más precisos para luego analizarlos y sacar conclusiones. Además basándonos en los resultados podremos evidenciar aspectos y situaciones no contempladas anteriormente que enriquecerán nuestro sistema una vez integradas.

En la sección 5.2 compararemos los planificadores FF y BlackBox, para justificar nuestra elección del planificador de nuestro juego. En la sección 5.1 detallaremos las métricas objetivas y subjetivas utilizadas para evaluar nuestro juego, fundamentando nuestra elección y haciendo hincapié en los diferentes aspectos que evaluan. En la sección 5.3 veremos tres casos de estudio que ayudarán al lector a comprender la constante interacción entre el agente y el jugador, como así también, la lógica en acción por parte del agente. En la sección 5.4 explicaremos el proceso de recolección de datos que realizamos en cada partida, indicando el momento en el cual se realizó la recolección, qué datos fueron recolectados y la importancia tras su elección. Por último en la sección 5.5 describiremos el proceso de evaluación para ambos modos del agente, las características de los jugadores, los resultados que arrojó el proceso y un análisis exhaustivo de los mismos.

5.1. Métricas de Evaluación

Las métricas son parámetros a tener en cuenta para evaluar el sistema. Es muy importante la selección de dichas métricas para que evaluen aspectos claves y a través de su análisis se puedan ver los puntos fuertes y débiles del mismo, pudiendo así sacar conclusiones y mejorarlo. Las hay de dos tipos: objetivas y subjetivas. Los datos de las métricas objetivas se obtienen de los registros de las interacciones entre el jugador y el entorno en tiempo real; mientras que los datos de las métricas subjetivas se obtienen de un cuestionario que se le entrega a cada jugador luego de cada partida.

En la sección 5.1.1 el conjunto completo de métricas objetivas y subjetivas como así también su categorización de acuerdo al tipo de característica que evaluan.

5.1.1. Métricas objetivas y subjetivas

Como mencionamos anteriormente, la información para confeccionar las métricas objetivas se extrae de los registros de las interacciones del jugador durante la partida. Las métricas que utilizamos en este caso son:

- Cantidad de *waypoints* transitados.
- Cantidad de objetivos cumplidos: Es la cantidad de instrucciones exitosas.
- Tiempo promedio de completitud por objetivo.
- Cantidad de fallas graves: Fallas generadas por replanificación, como veremos más adelante.
- Cantidad de fallas leves: Fallas generadas por incumplimiento de la instrucción actual, como veremos más adelante.
- Tiempo total de juego.
- Cantidad de instrucciones.
- Eliminación del enemigo.

Las métricas subjetivas se evaluan mediante la confección de un cuestionario por parte del jugador, que será completado luego de cada partida. El cuestionario está constituido por una serie de afirmaciones que el jugador debe puntuar de 0 a 200 según su nivel de aceptación. Estas afirmaciones se denominan métricas subjetivas, porque como podemos ver trivialmente, el nivel de aceptación varía de usuario en usuario. Como evaluaremos el sistema dando instrucciones de forma escrita y mediante audio, hay afirmaciones que posee un sistema que el otro no. Las afirmaciones son:

- El sistema utilizó palabras y frases fáciles de entender.
- Tuve que releer las instrucciones para entender lo que tenía que hacer (únicamente sistema de instrucciones en forma escrita)
- El sistema me dio devoluciones útiles sobre mi progreso.
- Estuve confundido sobre qué hacer proximamente.
- Estuve confundido sobre cual dirección tomar.
- No tuve dificultad para identificar los objetos que el sistema me describió.
- El sistema me dio mucha información innecesaria.
- El sistema me dio demasiada información a la vez.
- El sistema inmediátamente ofreció ayuda cuando estuve en problemas.
- Las instrucciones del sistema fueron dadas muy temprano y/o rápido.
- Las instrucciones del sistema fueron dadas demasiado tarde y/o lentamente.
- Las instrucciones del sistema estuvieron visibles durante suficiente tiempo para que pueda leerlas (únicamente sistema de instrucciones en forma escrita)
- Las instrucciones del sistema fueron redactadas/pronunciadas con claridad
- Las instrucciones del sistema fueron repetitivas.
- Realmente quer´ıa eliminar a la criatura.

- Perdí la noción del tiempo mientras resolvía el nivel.
- Disfruté la resolución del nivel.
- La interacción con el sistema fue realmente molesta.
- Recomendar´ıa el juego a un amigo.
- El sistema fue muy amigable.
- Sentí que podía confiar en las instrucciones del sistema.

A continuación catalogamos las métricas según las diferentes categorías que evaluan.

Eficiencia y eficacia

Las métricas fueron elegidas pensando en la calidad de las instrucciones del sistema, teniendo en cuenta el entendimiento y nivel de ayuda de las mismas.

- Cantidad de objetivos cumplidos: Mientras más tareas indicadas por el agente que complete exitosamente el jugador, mayor será la eficiencia y eficacia de las instrucciones.
- Cantidad de instrucciones: Mientras menos instrucciones utilice el agente para que el jugador cumpla su objetivo, más eficiente será.
- No tuve dificultad para identificar los objetos que el sistema me describió: Nos proporciona información sobre qué tan claras y precisas fueron las instrucciones para describir entidades del juego.
- El sistema utilizó palabras y frases fáciles de entender: Si el agente construyó las instrucciones con palabras y frases claramente entendidas por el jugador, entonces fue eficiente y eficaz.
- Cantidad de fallas graves: Si se produjeron varias de estas fallas, entonces el agente no fue eficaz en la generación de instrucciones que reorienten al jugador nuevamente en el camino del plan, provocando así una replanificación.

Entretenimiento

Las métricas fueron elegidas pensando en la calidad de la experiencia del jugador a nivel de entretenimiento.

- Realmente quer´ıa eliminar a la criatura: Si el jugador quer´ıa eliminar a la criatura a como de lugar, significa que el juego lo entretuvo y estuvo inmerso en él.
- Sentí que podía confiar en las instrucciones del sistema: Si el jugador confió en las instrucciones del sistema, significa que consideró al agente como un compañero dentro del juego, lo cual evidentemente incrementa el entretenimiento.
- Perdí la noción del tiempo mientras resolvía el nivel: Perder la noción del tiempo cuando uno juega es un claro indicador de que uno está inmerso totalmente en el juego.
- La interacción con el sistema fue realmente molesta: Si el agente fue un estorbo en lugar de una ayuda y le provocó molestia al jugador, obviamente provocó que la diversión disminuyera.
- Recomendaría el juego a un amigo: Se sobreentiende que uno recomendaría únicamente juegos que son divertidos.

Generación de *common ground* entre el sistema y el usuario

Las métricas fueron elegidas pensando en la calidad de las instrucciones con respecto a la asunción del contenido de las instrucciones.

- El sistema me dio mucha información innecesaria: Si el sistema satura al jugador con información irrelevante, claramente está incluyendo contenido sin importancia en el *common ground*.
- Cantidad de palabras promedio por instrucción: Si las instrucciones tienen un gran número de palabras es muy probable que el agente esté incluyendo demasiada información en la instrucción y por lo tanto, el jugador quizás pase por alto o no absorba información que es muy importante que incluya en su *common ground*.
- Las instrucciones del sistema fueron repetitivas: Si las instrucciones fueron repetitivas, significa que comunicaban conocimientos que el jugador ya ten´ıa incorporados en su *common ground*, por lo tanto era contenido redundante y por lo tanto indeseable.
- No tuve dificultad para identificar los objetos que el sistema me describió: Si el agente logró que el jugador identificara los objetos descriptos de forma fácil es porque utilizó inteligentemente información ya incorporada previamente en el *common ground*.
- Tuve que releer las instrucciones para entender lo que ten´ıa que hacer: Si el jugador tuvo que releer varias veces una instrucción para entenderla, es muy probable que el agente haya usado información no incluída en el *common ground*.

5.2. Comparación entre planificadores

Compararemos dos planificadores: BlackBox y FastForward. El objetivo de la comparación es justificar la elección del planificador que utilizaremos en nuestro juego. Blackbox es un sistema de *planning* que funciona mediante la conversión de problemas especificados en notación STRIPS a problemas de satisfactibilidad booleana, y luego resolviendo los problemas con una variedad de motores de satisfactibilidad del estado del arte. BlackBox es un planificador basado en grafos. Tiene una alta flexibilidad para la especificación del motor a usar, por ejemplo, se puede utilizar el motor X los primeros 60 segundos y si falla, utilizar un motor Y los próximos 1000 segundos. Esto le da a BlackBox la capacidad de funcionar eficientemente sobre un rango muy amplio de problemas. Su nombre proviene del hecho de que los generadores de planes no saben nada acerca de los *SAT solvers*, y los *SAT solvers* no conocen nada acerca de los planes: cada uno es una *Caja Negra* (BlackBox) para el otro.

Fast-Forward (FF) es un sistema de *planning* independiente del dominio, desarrollado por Joerg Hoffmann. FF puede manejar tareas de *planning* tanto en STRIPS como en ADL, que deben ser especificadas en PDDL. El sistema está implementado en C. Fast-Forward ha obtenido muchos premios y méritos como por ejemplo, en la 2da Competición Internacional de *Planning* o el premio Schindler al mejor sistema de *planning* en el dominio Miconic 10 Elevator. FF es un sistema de planificación que busca en el espacio de estados usando una estrategia de *forward chaining* guiada por una heur´ıstica.

En la sección 5.2.1 describiremos un problema simple y veremos las posibles acciones que se pueden realizar para resolverlo. Además veremos como solucionar el problema de manera óptima y como se desempeñan ambos planificadores para resolver el problema, sus tiempos y la fundamentación en la elección del planificador para nuestro trabajo.

5.2.1. El problema

El problema se denomina *El mono y la banana*. Supongamos que un mono se encuentra en un laboratorio con una banana que cuelga del techo. Hay una caja disponible que permitirá al mono recoger la banana si se trepa a la caja. La Figura 5.1 nos detalla los estados inicial y final.

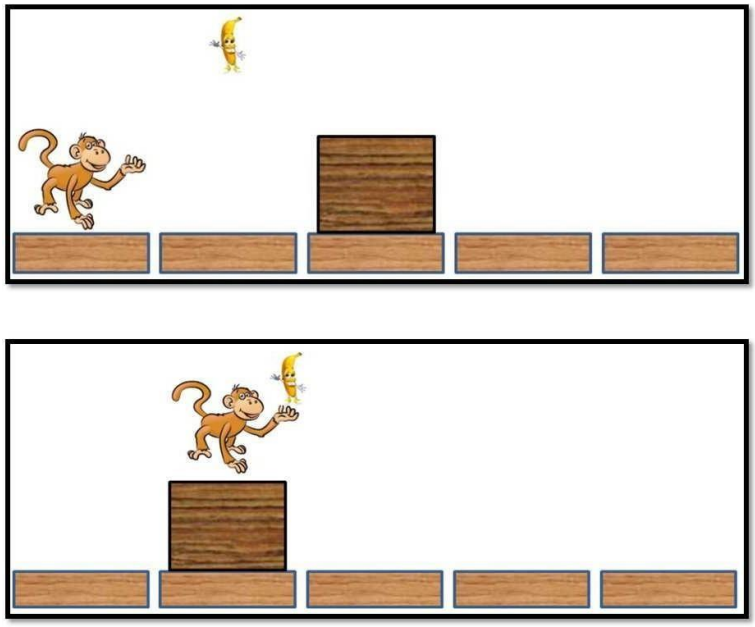

Figura 5.1: Estado inicial y final.

Las acciones que el mono puede realizar son las siguientes:

- *MoveTo*: Moverse de su casillero a otro adyacente, donde no haya ningún otro objeto.
- *ClimbUp*: Trepar a la caja si se encuentra en un casillero adyacente al suyo.
- *ClimbDown*: Bajar de la caja a un casillero adyacente que no esté ocupado por ningún objeto.
- *Push*: Empujar la caja hacia un casillero adyacente a ella, en donde no haya ningún otro objeto.
- *Take*: Tomar la banana si está a su misma altura y en su mismo casillero, es decir, si está subido a la caja en el mismo casillero que la banana.

La Figura 5.2 nos muestra cual es la secuencia más corta de acciones para resolver el problema.

Dadas las correctas especificaciones del dominio y problema en PDDL, ambos planificadores encontraron la solución óptima descripta anteriormente. La diferencia entre ambos radica en que FF resolvió el problema en 0 milisegundos mientras que a BlackBox le tomó 0.17. Obviamente no hay una diferencia sustancial de tiempo, pero debemos tener en cuenta que este problema es de los más simples y en nuestro juego nos encontraremos con problemas que requerirán un número de pasos muchísimo mayor y debemos sumar a esto que el cálculo de nuevas soluciones será muy frecuente. Por lo tanto, para nuestro trabajo elegiremos el planificador FF.

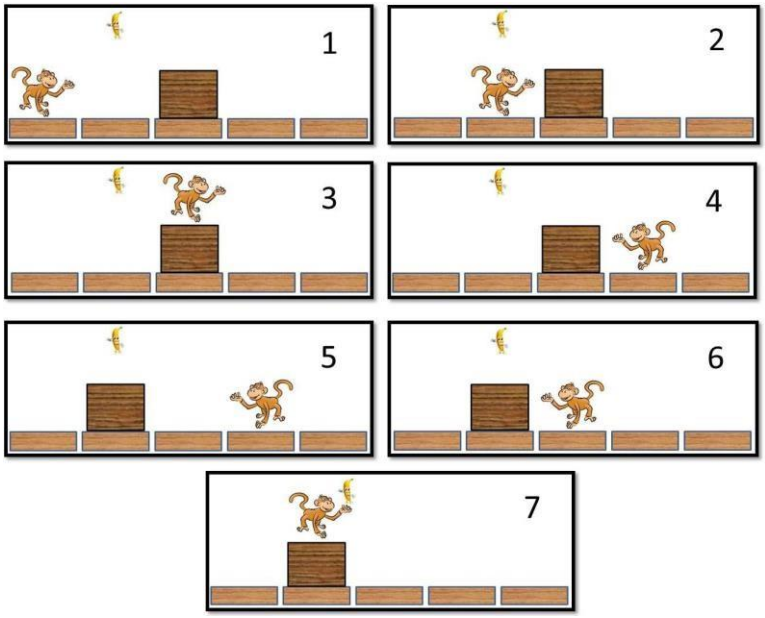

Figura 5.2: Secuencia de acciones para resolver el problema.

5.3. Casos de estudio

En esta sección presentaremos tres casos de estudio que mostrarán comportamientos claves del agente. Los casos de estudio son importantes porque proveen una manera sistemática de observar eventos, recolectar datos y analizar la información. Como resultado el lector entenderá de manera muy precisa como se llevan a cabo todos los eventos, y el investigador podrá tener una mirada más clara sobre las posibles mejoras futuras. Además son una fuerte herramienta para generar y testear hipótesis.

En la sección 5.3.1 veremos como se genera *common ground* entre el agente y el jugador. En las sección 5.3.2 veremos una secuencia completa de instrucciones cuyo fin será que el jugador encuentre un rayo. Por último, en la sección 5.3.3 veremos un ejemplo de replanificación.

5.3.1. Generación de *common ground*

El objetivo principal de este caso de estudio es mostrarle al lector una situación en la cual se generará *common ground* entre el agente y el jugador. Esto se evidenciará en las instrucciones que el agente le dará al jugador.

Supongamos que el jugador debe recolectar el rayo azul, y el agente, mediante el testeo del entorno, detecta que el rayo azul se ubica enfrente del jugador pero en una posición un poco alejada. Como podemos ver en la Figura 5.3, el agente generó la instrucción *Do you see that ray in front of you?*, cuyo fin es que el jugador vea el rayo azul lejano a él.

Posteriormente, el jugador lee la instrucción que generó el agente, la interpreta y entiende y comienza a acercarse al rayo azul. En ese preciso momento, el objeto (el rayo azul) y su posición (enfrente del jugador) son conocimientos que se incorporaron en el *common ground* del jugador y el agente. Como podemos ver en la Figura 5.4, el agente generó la instrucción *Pick it up*, cuyo fin es que el jugador recolecte el rayo azul. En este caso, el agente no mencionó de forma explícita el objeto o su localización, ya que es innecesario y redundante, porque dichos conocimientos ya

Figura 5.3: El agente pretende que el jugador vea el rayo azul.

están integrados en el *common ground* de ambas partes.

Figura 5.4: El agente genera una instrucción que utiliza *common ground*.

5.3.2. Generación de instrucciones de forma colaborativa

En este caso veremos como el agente va generando instrucciones con el objetivo de que el jugador recolecte el rayo rojo. Notaremos claramente como cambian las instrucciones generadas de acuerdo a las acciones que va realizando el jugador y explicaremos el porqué de estos cambios.

Supongamos que el jugador está en la posición inicial indicada por la Figura 5.5. Podemos ver que el agente ha generado la instrucción *Do you see that door in front of you?* pretendiendo que el jugador ubique la puerta que tiene delante suyo.

Como podemos ver en la Figura 5.6, el jugador ha ubicado la puerta y se acerca a ella. En

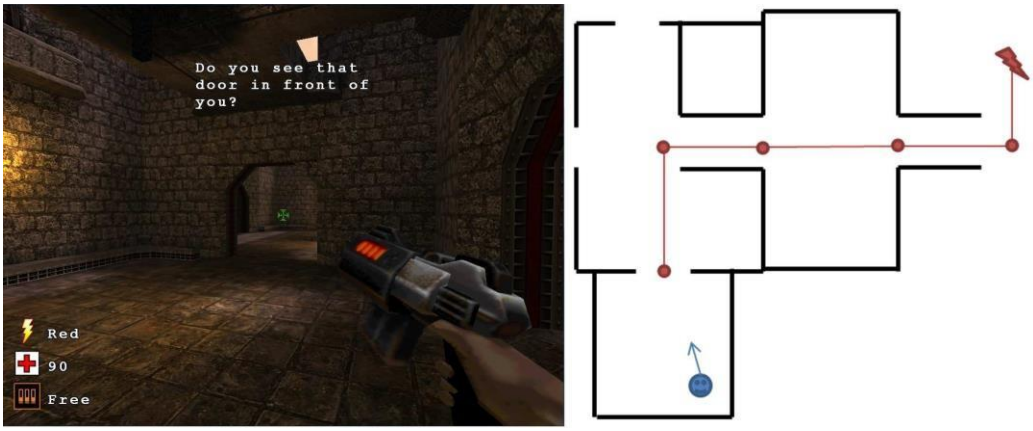

Figura 5.5: Instrucción para localizar la puerta.

este instante, la ubicación de la puerta se integró en el *common ground* de ambas partes, y dado esto, el agente genera una nueva instrucción, evitando mencionar de forma explícita el objeto en cuestión. Como podemos ver el agente genera la instrucción *Cross it*, pretendiendo que el jugador cruce la puerta.

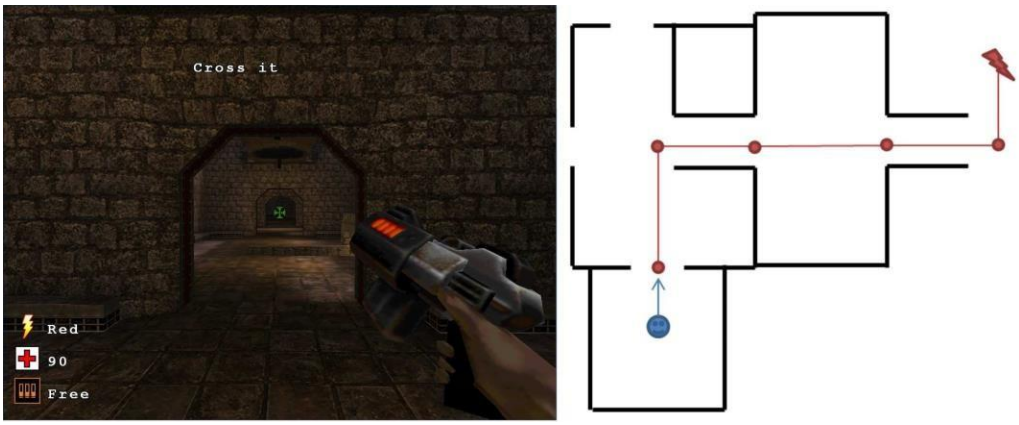

Figura 5.6: Instrucción para cruzar la puerta.

Luego el agente cruza la puerta y ha seguido su rumbo como lo muestra la Figura 5.7. En este caso, el agente le indica mediante la instrucción *Turn right to see the stairs* que voltee hacia la derecha para poder ubicar unas escaleras. De esta forma, el agente alerta al jugador sobre sus movimientos para intentar guiarlo por el camino correcto.

Una vez que el jugador gira y ve las escaleras, como lo podemos ver en la Figura 5.8, el agente genera una nueva instrucción, *Take these stairs* pretendiendo que el jugador continue su rumbo en esa dirección. En este caso menciona de forma explícita a la escalera, porque ella no estaba en el *common ground* de ambos, ya que el jugador ten´ıa una idea de donde se ubicaba la escalera, pero no el lugar exacto.

El jugador de manera correcta baja la escalera como muestra la Figura 5.9 y el agente nuevamente, mediante la instrucción *Do you see those stairs in front of you?*, pretende captar la atención del jugador para que ubique unas escaleras que están frente a él. En este momento la posición de la escalera integra el *common ground* del jugador y el agente.

Luego, como podemos ver en la Figura 5.10, el jugador se acerca a la escalera y el agente

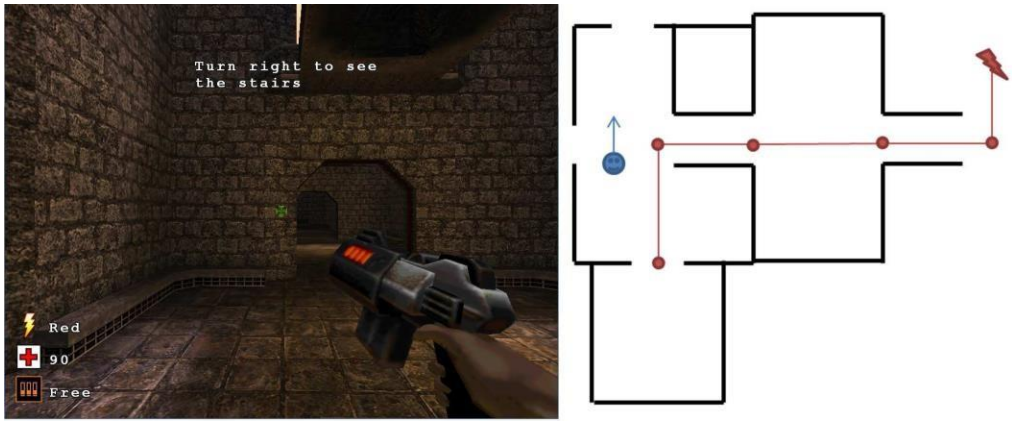

Figura 5.7: Instrucción para girar y ver unas escaleras.

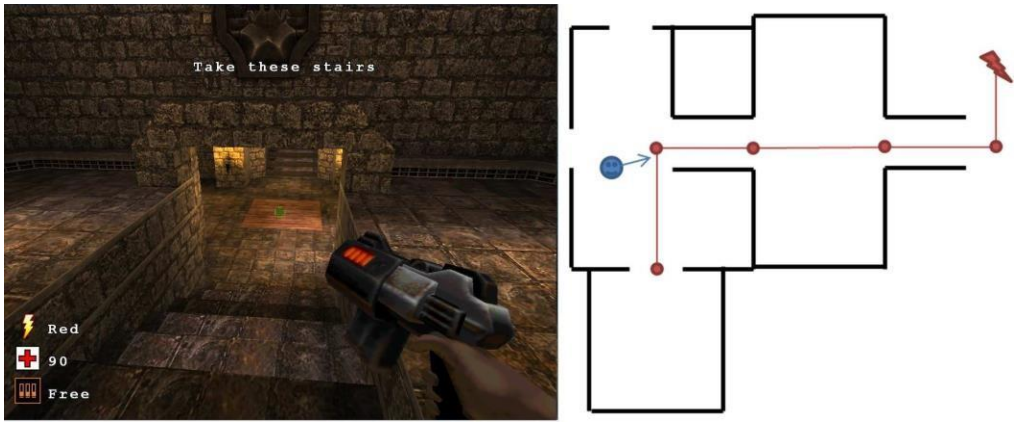

Figura 5.8: Instrucción para cruzar la escalera.

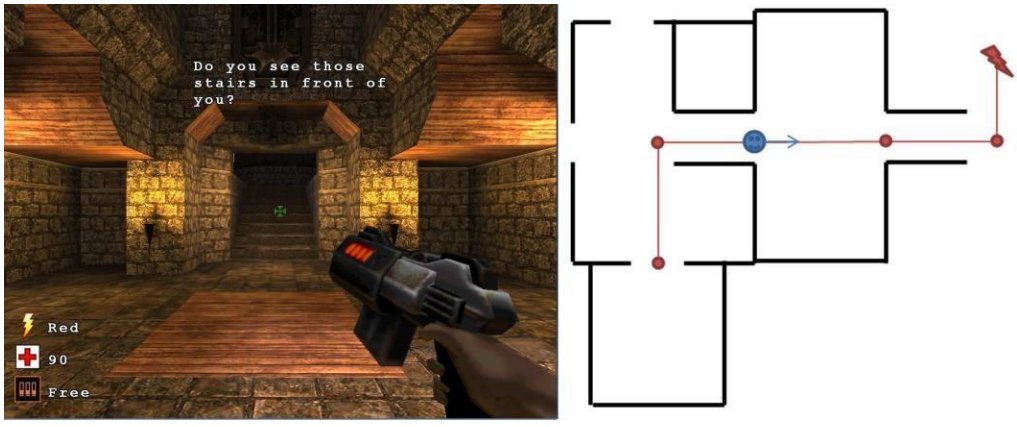

Figura 5.9: Instrucción para ubicar la posición de las escaleras.

genera la instrucción *Take them* con el objetivo de que el jugador la suba. Nuevamente como hemos visto anteriormente se refiere de manera impl´ıcita a ella.

Una vez que el jugador sube las escaleras como se muestra en la Figura 5.11, el agente genera la instrucción *Turn left to see the ray* con el objetivo de que el jugador gire para lograr hacer

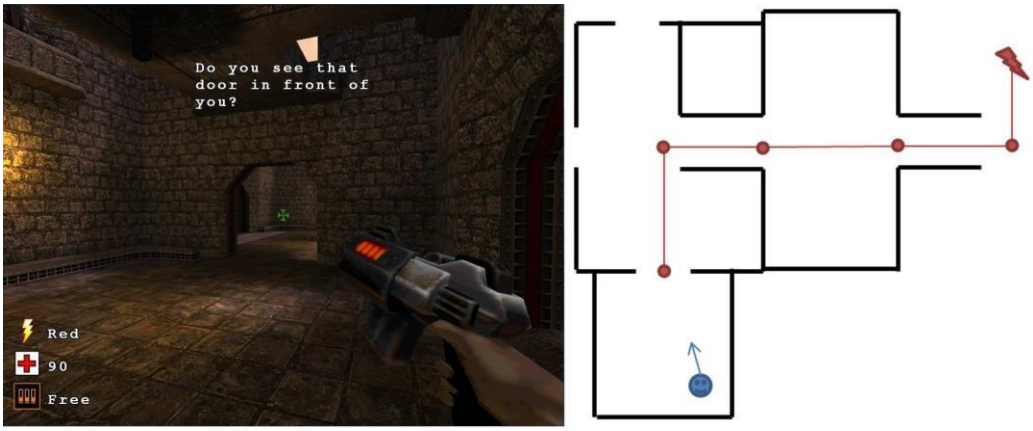

Figura 5.10: Instrucción para subir las escaleras.

visible el rayo buscado.

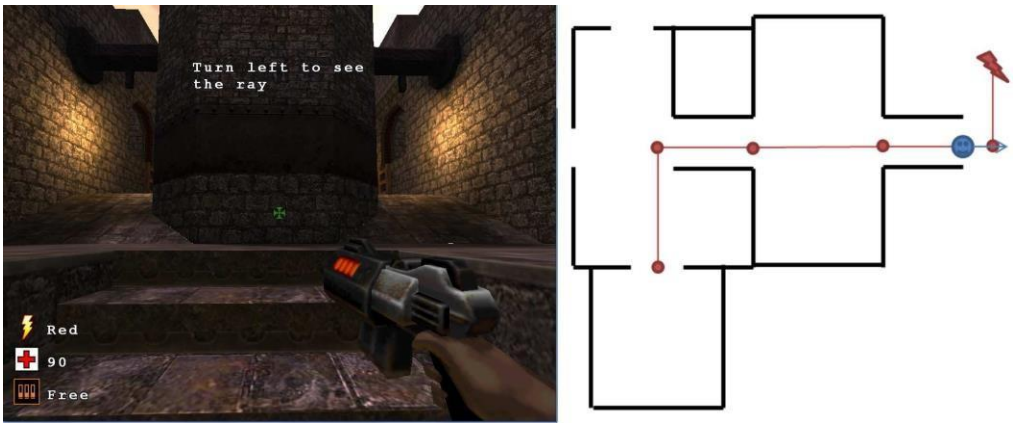

Figura 5.11: Instrucción para girar y ver el rayo rojo.

Finalmente, como vemos en la Figura 5.12, el agente le indica al jugador mediante la instrucción *Get this ray*, que recoja el rayo rojo.

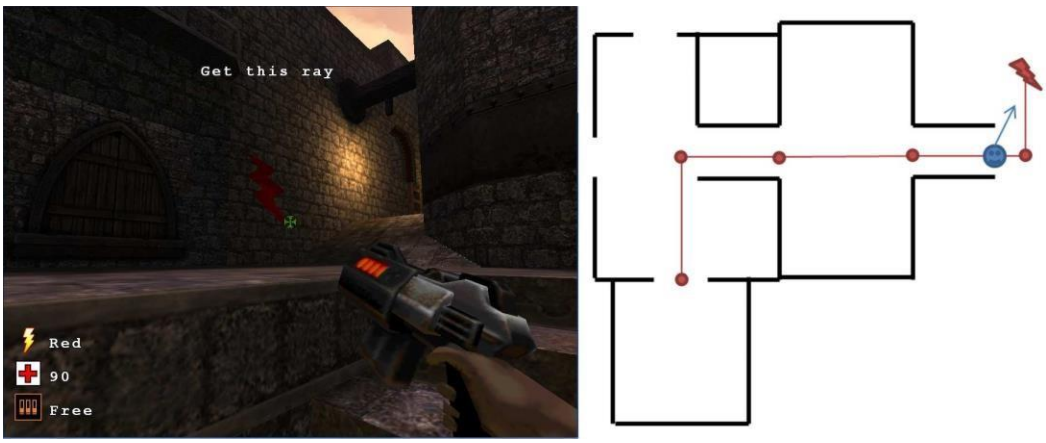

Figura 5.12: Instrucción para recoger el rayo rojo.

5.3.3. Replanificación por desvió del camino del plan

Por último, veremos un caso de estudio en el cual el agente decidirá realizar una replanificación debido a que el jugador se desvía del camino del plan, no pudiendo el agente reubicarlo nuevamente hacia los *waypoints* del camino actual. En nuestro caso, el agente ha confeccionado un plan para guiar al jugador hacia la recolección del rayo azul.

Supongamos que el jugador comienza en la posición que nos muestra la Figura 5.13. Como podemos ver, el agente genera la instrucción *Do you see that ray in front of you?* pretendiendo que el jugador ubique el rayo azul delante de el.

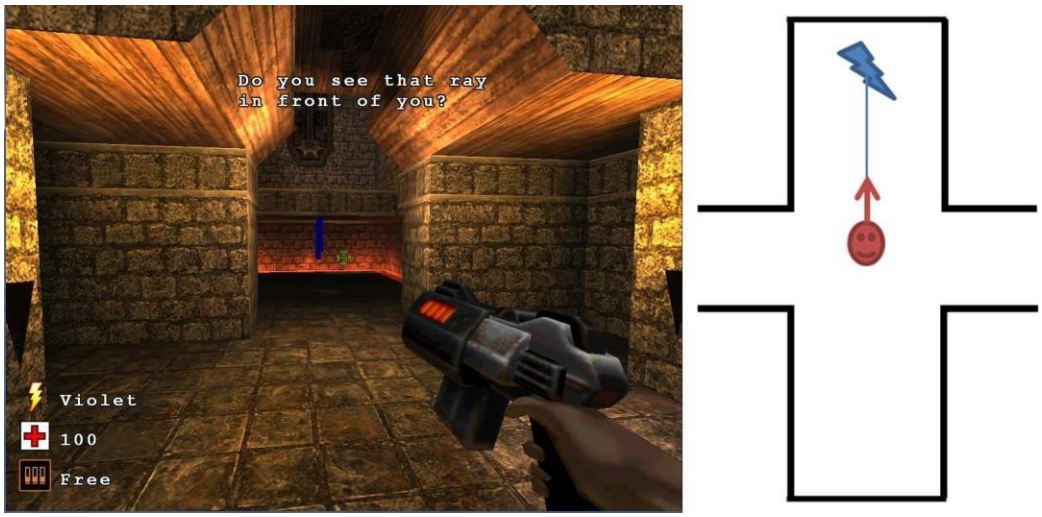

Figura 5.13: Instrucción para ubicar el rayo.

El jugador decide hacer caso omiso a la instrucción gira hacia la izquierda como nos muestra la Figura 5.14. Debido a esta acción por parte del jugador, el agente genera la instrucción *Turn right to see the ray* para intentar que el jugador gire y ubique el rayo azul, aprovechando que todavía está visible a 360 grados de la posición del jugador.

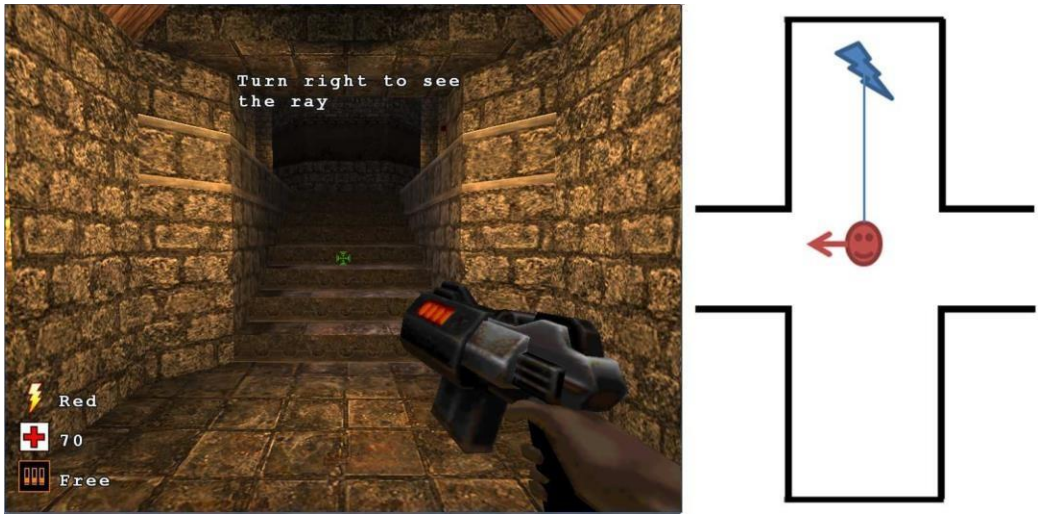

Figura 5.14: Instrucción para girar y ver el rayo.

El jugador nuevamente no toma en cuenta la instrucción y decide empezar a subir la escalera

que tiene delante como lo muestra la Figura 5.15. En este caso el agente genera la instrucción *Turn right* para intentar reubicar al jugador en el camino hacia el rayo. Podemos ver que ahora no intenta que el jugador gire como antes y ubique el rayo, esto ocurre debido a que el rayo azul ya no está visible a 360 del jugador, por lo tanto el agente no puede utilizarlo como punto de referencia.

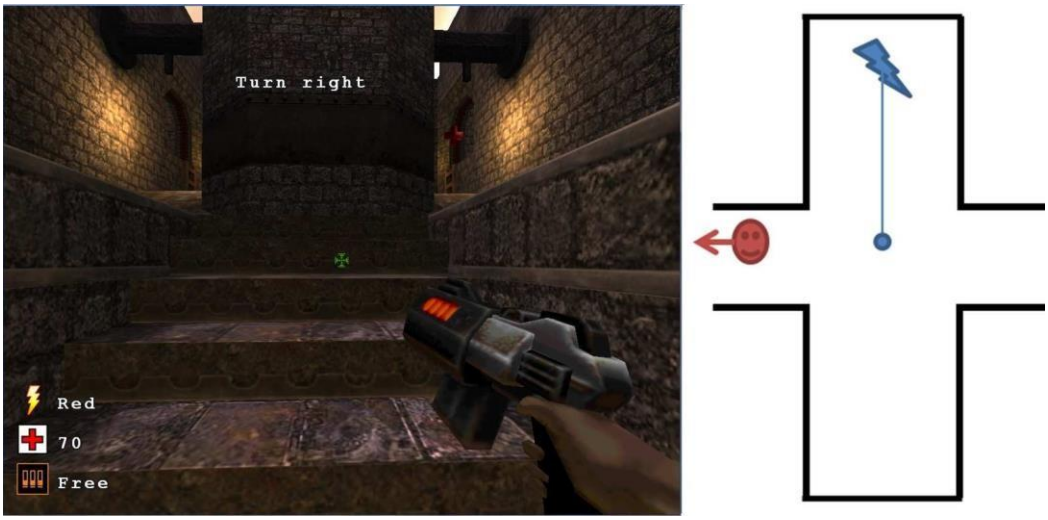

Figura 5.15: Instrucción para girar.

Por último, el jugador sube la escalera y dobla hacia la derecha, acción luego de la cual gira nuevamente posicionándose hacia la escalera, como lo muestra la Figura 5.16. Ya que ningún *waypoint* del plan es visible a 360 grados del jugador, el agente encuentra imposible la reubicación del jugador al camino del plan, por lo tanto decide replanificar. Habiendo hecho esto, se genera un nuevo camino para guiar al jugador y debido a esto, genera la instrucción *Go forward* para que el jugador avance y así dirigirlo al objetivo, es decir, recolectar el rayo azul.

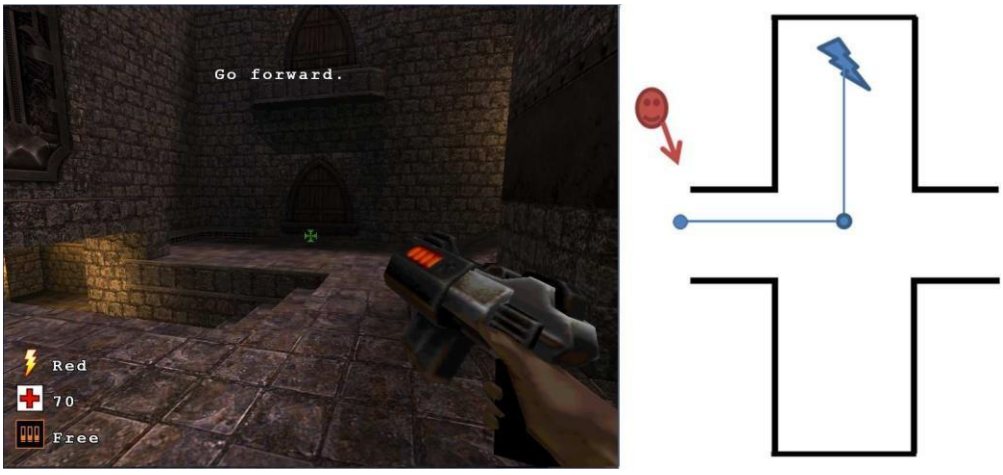

Figura 5.16: Instrucción para avanzar.

5.4. Registro de la partida

El primer paso para poder evaluar el sistema es recolectar datos importantes sobre las partidas jugadas por humanos. En nuestro caso hacemos dos tipos de registro de datos, uno en tiempo real, es decir, mientras se juega la partida, y el otro al final de la misma. El último de ellos no es nada más y nada menos que el registro de las métricas objetivas. Cuando el jugador termina una partida, recolectamos estos dos tipos de datos para incluirlos en el análisis posterior.

5.4.1. Registro en tiempo real

El sistema registra en un archivo de forma secuencial, las acciones relevantes que se están llevando a cabo. Las acciones que registramos son:

- Cambio de posición del jugador: Es importante ver todos los sectores por los que que transitó el jugador.
- Acción de disparo: Esto nos sirve para tener en cuenta que en esos momentos quizas esté disparando al enemigo, siendo normal que se desvíe del plan.
- Recolección de items: Esto es importante para ver si recolectó el rayo correcto, o si se desvió del plan a causa de necesitar recolectar salud o municiones.
- Generación de una nueva instrucción: Esto nos sirve para ver luego como va reaccionando el jugador a cada una de las instrucciones.
- Recolección de rayo correcto.
- Replanificación: Es importante saber cuando hubo una replanificación porque puede deberse a que el jugador se desvió del camino del plan o recolectó un rayo incorrecto.
- Cambio de estado del enemigo: Es importante saber cuando el enemigo está escapando, atacándonos, buscándonos o muerto, para entender las reacciones del jugador en cada momento dado.
- Completitud de instrucción: Cada vez que el jugador completa una instrucción exitosamente, registramos la instrucción en cuestión y el intervalo de tiempo entre la generación de la instrucción y su completitud. Esto es sumamente importante ya que podremos saber si el jugador comprende correctamente una instrucción.

Cabe mencionar que los registros son generados cronológicamente y se incluye su tiempo de generación. La importancia de este tipo de datos radica en que podremos evaluar una partida luego de ser jugada, sin perder detalle. Incluso si pudiésemos ver la partida en vivo, por la rapidez del juego y la cantidad de cosas que ocurren de un momento al otro, es muy probable que el observador no pueda darse cuenta de aspectos importantes que sí se pueden observar en el registro.

5.4.2. Registro de fallas

En el registro final de la partida, donde como dijimos registramos las métricas objetivas, calculamos las fallas graves y las fallas leves entre otros tipos de datos. Una instrucción es exitosa cuando el jugador la ejecuta de forma correcta, esto significa, el jugador hace lo que la instrucción le dice, por ejemplo, en la Figura 5.17 el agente mediante la instrucción *Cross this door* pretende que el jugador atraviese la puerta. Supongamos que el jugador hace lo que

Figura 5.17: Instrucción para cruzar una puerta.

muestra la Figura 5.18, entonces el agente puede inferir que el jugador comprendió correctamente la instrucción y la ejecutó exitosamente.

Ahora, ¿en qué casos el agente inferirá que el jugador falló en la ejecución de esa instrucción y aun más importante, qué tipo de falla fue?. Para ello dividimos las fallas en dos tipos: graves y leves. Se considera falla grave a la replanificación producida por la recolección de un rayo incorrecto o por desvío del camino del plan. Se considera falla leve a las ejecución incorrecta de una instrucción. Supongamos que el jugador hace caso omiso a la instrucción anterior y decide ir por otro camino diferente al del plan, como lo muestra la Figura 5.19. En este caso el jugador no sólo no ejecutó la instrucción exitosamente sino que tambíen al desviarse del camino el plan, provocó que el agente realizara una replanificación por desvío del camino del plan, por lo tanto se la considera una falla grave.

Ahora supongamos que el jugador mira hacia la pared de la derecha como lo muestra la Figura 5.20. En este caso al agente le indica mediante la instrucción *Turn left to see the door* que se vuelva a focalizar en el objetivo. Consideramos que el jugador no ejecutó la instrucción de forma exitosa a causa del desvió de su atención hacia otra cosa, por lo tanto se considerará una falla leve.

Una observación muy importante es que toda falla grave fue ocasionada por una secuencia de fallas leves, ya que secuencialmente el jugador fue desobedeciendo las instrucciones del agente, ocasionando así la replanificación (excepto en el caso de falla grave ocasionada por la recolección de un rayo incorrecto).

5.5. Análisis de los datos

El proceso de evaluación de los datos es una de las partes más importantes de nuestro trabajo ya que se comprobará si todas las hipótesis y suposiciones sobre el sistema actual son ciertas. Comprobaremos la efectividad de las instrucciones generadas por el agente y revelaremos posibles mejoras al sistema.

En la sección 5.5.1 veremos de que consta el proceso de evaluación elegido. En la sección 5.5.2 veremos los resultados del proceso de evaluación y analizaremos los datos obtenidos.

Figura 5.18: El jugador cruza la puerta; la instrucción fue exitosa.

5.5.1. Proceso de evaluación

Para el proceso de evaluación reunimos 10 voluntarios de diferentes edades, que reunían la misma condición de *gamers*, es decir, son jugadores habituales de videojuegos. Se les explicó a cada uno de ellos, las características generales del juego, como por ejemplo su género, controles con los que se ejecutan las diferentes acciones y objetivo del mismo. Además se les explicó el objetivo de dicha evaluación.

Cada uno de los voluntarios jugó una partida de manera aislada, es decir, no pudieron ver partidas ajenas para que no tuvieran la ventaja de conocer más el juego. Como mencionamos en secciones anteriores, mientras la partida se efectuaba, el agente fue recolectando datos estad´ısticos que evaluan las métricas objetivas del sistema. Luego de concluida la partida, cada jugador completó el cuestionario descripto también en secciones anteriores.

Algunas caracter´ısticas sobre los voluntarios fueron las siguientes:

- Todos fueron de género masculino.
- La edad promedio fue de 20 años.
- Todos jugaron anteriormente juegos de este género, es decir, *First Person Shooter*.
- Todos son habituales *gamers*.

Cabe mencionar que realizamos dos tipos de evaluaciones:

- Evaluación del sistema mediante instrucciones escritas: El agente emite instrucciones escritas por pantalla.
- Evaluación del sistema mediante instrucciones orales: El agente reproduce las instrucciones mediante un archivo de audio.

En el último caso, se realizaron varias partidas con diferentes jugadores (previo a la evaluación del sistema) donde una persona que conocía completamente el entorno del juego, emitía

Figura 5.19: El jugador se fue a una habitación lejana.

instrucciones en forma verbal mientras las grababa en archivos de audio. La grabación del audio estaba sincronizada con el tiempo de juego para facilitar su posterior análisis. Luego se analizaron esas instrucciones y se escogieron las que fueron exitosas, es decir, aquellas que el jugador ejecutó inmediatamente después de ser emitidas en forma correcta.

A partir de esto, se obtuvo una base de datos con muchas instrucciones en audio, muchas de ellas emitidas con el mismo objetivo, como por ejemplo, que el jugador recolecte un rayo. Se extendió el agente para que pueda emitir instrucciones oralmente, es decir, reproduciendo audio en lugar de mostrar las instrucciones por pantalla. Además, para aprovechar la diversidad de instrucciones y hacer que el agente no sea monótono en sus instrucciones, para una misma situación, escogemos de un conjunto de instrucciones, una sola que será reproducida en ese momento. De esta forma obtuvimos instrucciones que se refer´ıan a lo mismo pero donde se utilizaban palabras diferentes, oraciones con longitudes variables y entonaciones distintas. Cabe mencionar que las instrucciones fueron emitidas en español.

5.5.2. Análisis y conclusiones sobre los datos

Finalmente llegamos a la parte en la cual sabremos realmente la efectividad de nuestro agente y comprobaremos si realmente ayudó al jugador.

Las caracter´ısticas del sistema que extrajimos de los resultados de cada partida son:

- Promedio de instrucciones generadas.
- Promedio de fallas graves.
- Promedio de fallas leves.
- Promedio de éxito total.
- Tiempo promedio de completitud de instrucciones.
- Promedio de *waypoints* recorridos.
- Promedio de duración de juego.
- Cuestionario.

Figura 5.20: El jugador desvió su atención a la pared de la derecha.

Promedio de instrucciones generadas

La Figura 5.21 muestra la cantidad de instrucciones generadas por jugador en el sistema de instrucciones escritas, cuyo promedio es 73 y su desviación estándar es 21.16 y la cantidad de instrucciones generadas por jugador en el sistema de instrucciones orales, cuyo promedio es 71.5 y su desviación estándar es 7.8. Como podemos ver ambas son bastante similares, aunque el sistema oral tiene una desviación estándar bastante menor. Para poder sacar conclusiones sobre estos datos debemos ver las tasas de objetivos cumplidos y fallas.

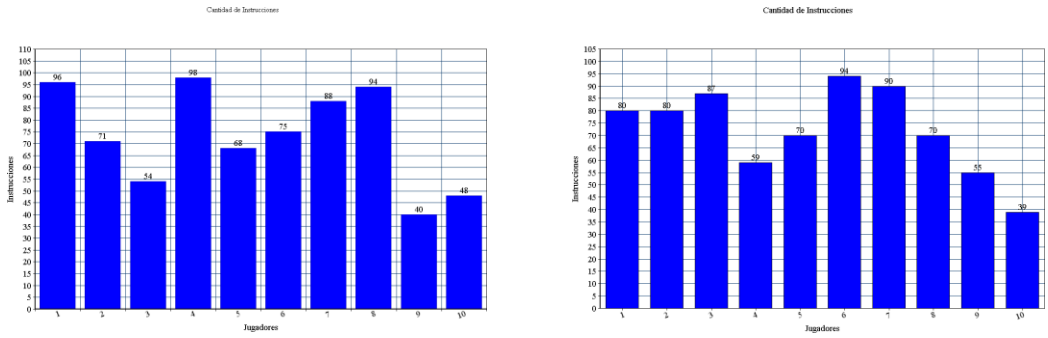

Figura 5.21: Cantidad de instrucciones por jugador (sistema escrito y oral).

Promedio de fallas graves

La Figura 5.22 muestra la cantidad de fallas graves por jugador en el sistema de instrucciones escritas, cuyo promedio es 4.4 y su desviación estándar es 3.92 y muestra la cantidad de fallas graves por jugador en el sistema de instrucciones orales, cuyo promedio es 3.2 y su desviación estándar es 1.87. Por lo tanto podemos apreciar que realmente hubieron en ambos casos muy pocos desvíos del plan que ocasionaran una replanificación, teniendo en cuenta que únicamente hab´ıa que recolectar 4 rayos para volver vulnerable al enemigo. Esto nos sugiere que el agente fue efectivo en la tarea de orientar al jugador hacia el objetivo, en ambos sistemas, aunque nuevamente el sistema oral presenta valores levemente más bajos.

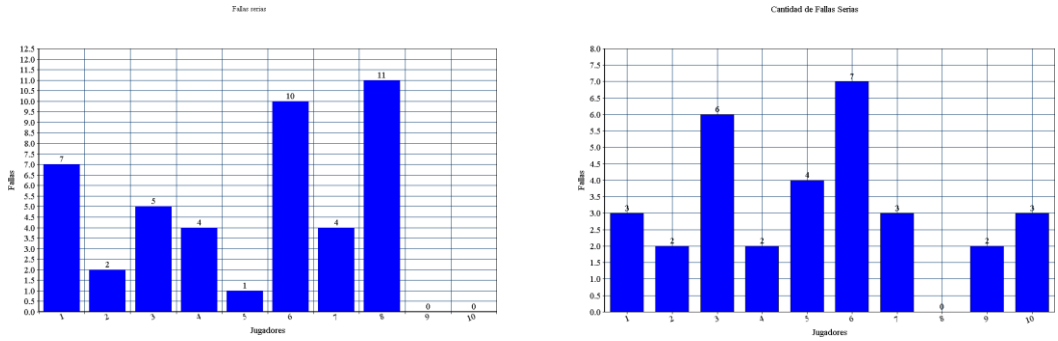

Figura 5.22: Cantidad de fallas graves por jugador (sistema escrito y oral).

Promedio de fallas leves

La Figura 5.23 muestra la cantidad de fallas leves por jugador en el sistema de instrucciones escritas, cuyo promedio es 28.9 y su desviación estándar es 17.04 y muestra la cantidad de fallas leves por jugador en el sistema de instrucciones orales, cuyo promedio es 19.4 y su desviación estándar es 12.34. Por lo tanto los jugadores fallaron en el cumplimiento de un 30 % de las instrucciones en el caso del sistema escrito y en 20 % en el sistema oral que de acuerdo al registro de las partidas se debió en su mayor parte a la pérdida de focalización en el objetivo de la instrucción y por la presencia del enemigo, hecho que obviamente provocará que el jugador no siga la instrucción actual al pie de la letra.

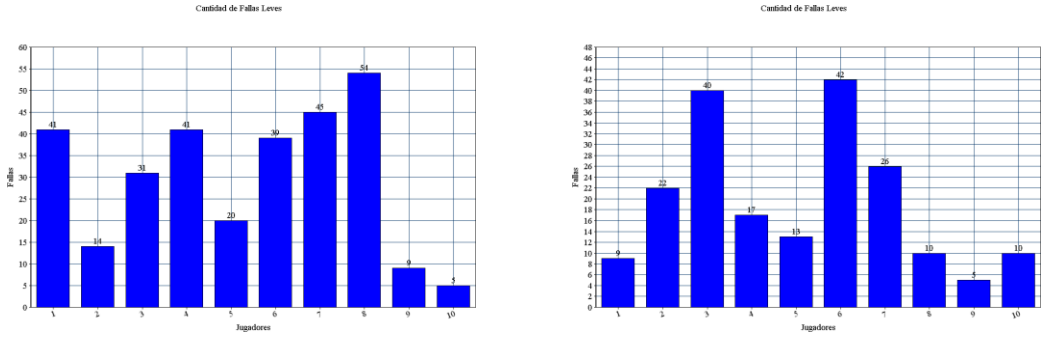

Figura 5.23: Cantidad de fallas leves por jugador (sistema oral).

Promedio de éxito total

Es el promedio de objetivos cumplidos. La Figura 5.24 muestra los objetivos cumplidos de todos los jugadores en el sistema escrito cuyo promedio es 44.1 y su desviación estándar es 9.74 y muestra los objetivos cumplidos de todos los jugadores en el sistema oral cuyo promedio es 54 y su desviación estándar es 7.33. Como podemos ver, casi el 60 % de las instrucciones fueron exitosas en el sistema escrito y aun encontramos un mayor porcentaje en el sistema oral, y teniendo en cuenta las causas de la mayor´ıa de las instrucciones no exitosas, podemos decir que el agente hizo bastante bien su tarea.

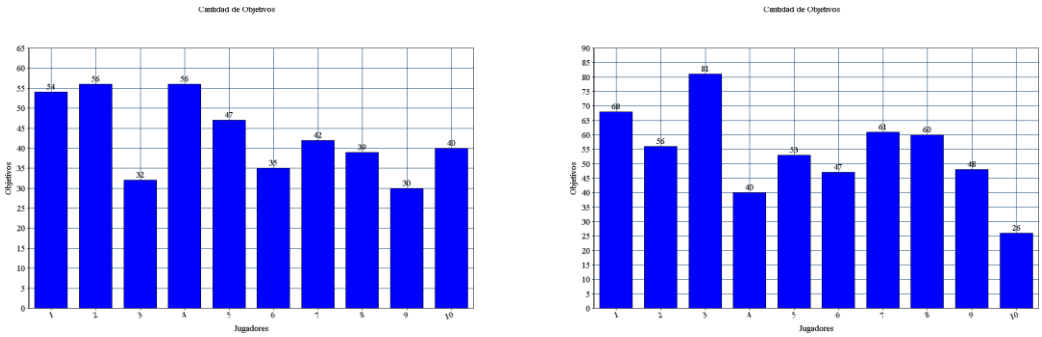

Figura 5.24: Cantidad de objetivos cumplidos por jugador (sistema escrito y oral).

Tiempo promedio de completitud de instrucciones

Es el tiempo promedio que necesitaron los jugadores para cumplir una instrucción. La Figura 5.25 muestra los tiempos de todos los jugadores en el sistema escrito, cuyo promedio es 1.01 segundos y su desviación estándar es de 2.31 segundos y muestra los tiempos de todos los jugadores en el sistema oral, cuyo promedio es 1.13 segundos y su desviación estándar es de 0.3 segundos. Ambos sistemas tienen prácticamente el mismo promedio, pero definitivamente la desviación estándar es mucho menor en el sistema oral. Teniendo en cuenta la velocidad de movimiento del jugador podemos ver que las instrucciones exitosas fueron completadas rápidamente, es decir, las instrucciones fueron de fácil comprensión para los jugadores y de igual manera su ejecución.

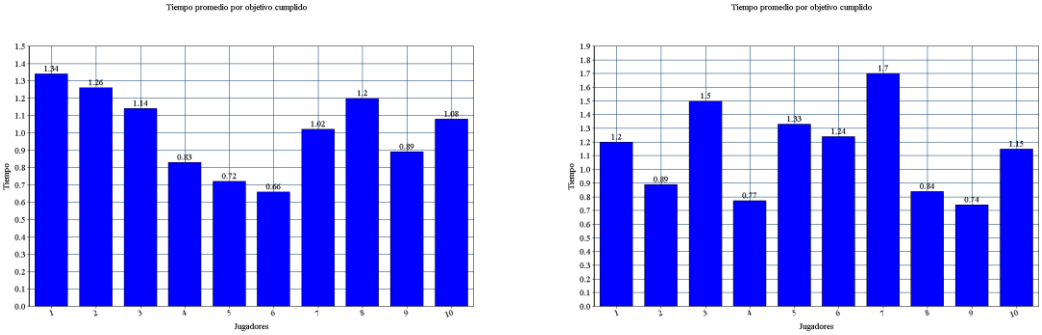

Figura 5.25: Tiempo promedio de completitud de instrucción por jugador (sistema escrito y oral).

Promedio de *waypoints* recorridos

Es el promedio de *waypoints* recorridos durante todo el juego. La Figura 5.26 muestra la cantidad de *waypoints1* recorridos de todos los jugadores en el sistema escrito, cuyo promedio es 65.5 y su desviación estándar es de 18.76 y muestra la cantidad de *waypoints* recorridos de todos los jugadores en el sistema oral, cuyo promedio es 58.1 y su desviación estándar es de 15.02. Nuevamente, el promedio es algo menor en el sistema oral como aí también la desviación estándar. Teniendo en cuenta que el mapa posee exactamente 60 *waypoints* (por lo tanto se recorrieron *waypoints* repetidos), y recolectando la información del registro en tiempo real por partida, podemos ver que en la mayoría de los casos se recorrió aproximadamente un 70 % del mapa.

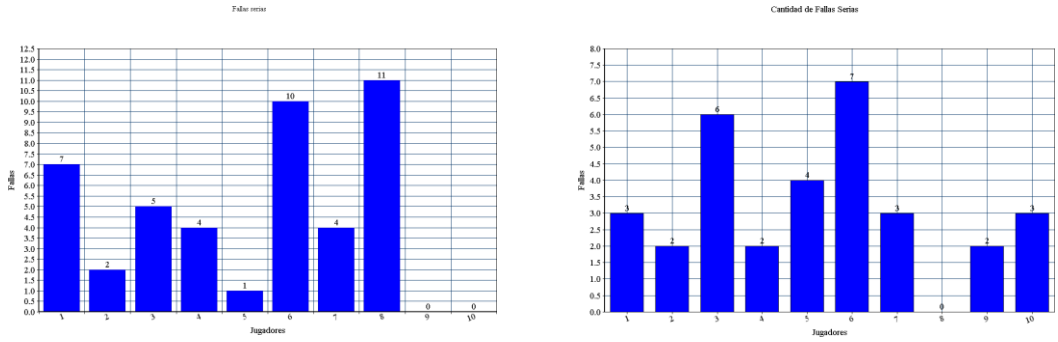

Figura 5.26: Cantidad de *waypoints* recorridos por jugador (sistema escrito y oral).

Tasa de duración de juego

Es el promedio de duración de cada partida. La Figura 5.27 muestra los tiempos de duración de las partidas de todos los jugadores en el sistema escrito, cuyo promedio es 2.4 minutos y su desviación estándar es de 1.29 minutos y muestra los tiempos de duración de las partidas de todos los jugadores en el sistema oral, cuyo promedio es 3.9 minutos y su desviación estándar es de 1.34 minutos. Podemos concluir que los jugadores tardaron poco tiempo en finalizar el juego, ya que el mapa es mediano y los jugadores se mueven a una velocidad lenta comparada con el común de los FPS. De todas formas, en este caso, fue más efectivo el sistema escrito, reduciendo en promedio más de 1.5 minutos.

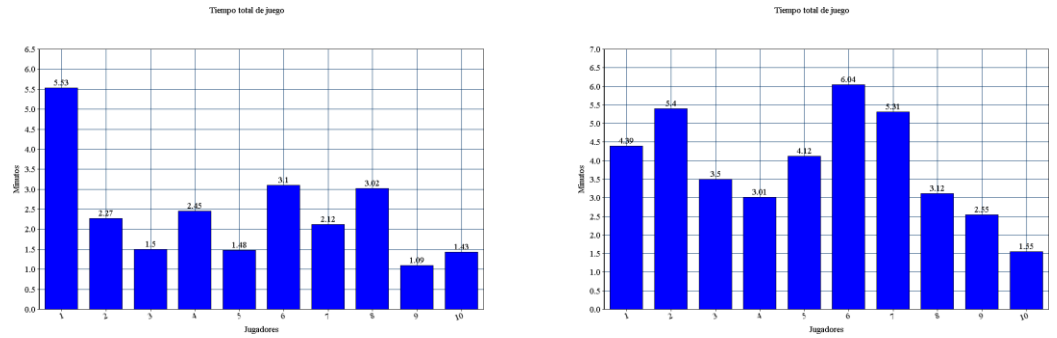

Figura 5.27: Tiempo total de juego por jugador (sistema escrito y oral).

Cuestionario

Los porcentajes de aceptación de las afirmaciones del cuestionario fueron los siguientes:

- El sistema utilizó palabras y frases fáciles de entender - 80 %(escrito) - 90 %(oral)
- Tuve que releer las instrucciones para entender lo que ten´ıa que hacer - 70 %(escrito)
- El sistema me dio devoluciones útiles sobre mi progreso - 50 %(escrito) - 60 %(oral)
- Estuve confundido sobre que hacer proximamente - 75 %(escrito) - 60 %(oral)
- Estuve confundido sobre cual dirección tomar - 80 %(escrito) - 40 %(oral)
- No tuve dificultad para identificar los objetos que el sistema me describió - 80 %(escrito) - 90 %(oral)

- El sistema me dio mucha información innecesaria - 10 %(escrito) - 10 %(oral)
- El sistema me dio demasiada información a la vez - 75 %(escrito) - 45 %(oral)
- El sistema inmediátamente ofreció ayuda cuando estuve en problemas - 80 %(escrito) - 70 %(oral)
- Las instrucciones del sistema fueron dadas muy temprano - 50 %(escrito) - 30 %(oral)
- Las instrucciones del sistema fueron dadas demasiado tarde - 10 %(escrito) - 10 %(oral)
- Las instrucciones del sistema estuvieron visibles durante suficiente tiempo para que pueda leerlas - 35 %(escrito)
- Las instrucciones del sistema fueron redactadas con claridad - 90 %(escrito) - 100 %(oral)
- Las instrucciones del sistema fueron repetitivas - 90 %(escrito) - 50 %(oral)
- Realmente quer´ıa eliminar a la criatura - 95 %(escrito) - 100 %(oral)
- Perdí la noción del tiempo mientras resolvía el nivel - 80 %(escrito) - 80 %(oral)
- Disfruté la resolución del nivel - 90 %(escrito) - 100 %(oral)
- La interacción con el sistema fue realmente molesta - 55 %(escrito) - 25 %(oral)
- Recomendar´ıa el juego a un amigo - 70 %(escrito) - 85 %(oral)
- El sistema fue muy amigable - 70 %(escrito) - 80 %(oral)
- Sentí que podía confiar en las instrucciones del sistema - 60 %(escrito) - 75 %(oral)

Como podemos apreciar, las afirmaciones relacionadas con el entretenimiento obtuvieron altísimos porcentajes de aceptación, lo cual denota que el jugador se divirtió que es el objetivo principal de cualquier juego. Otro punto fuerte tiene que ver con que las instrucciones del agente estuvieron bien redactadas y fueron entendibles y los jugadores no consideraron la información de las instrucciones como innecesaria. Además, un punto importante es que el agente les brindo ayuda cuando se encontraban confundidos, lo cual es un aspecto muy importante. Por otra parte, los jugadores consideraron que las instrucciones les permitieron identificar de forma clara y concisa los objetos a los que se hacía referencia. Además podemos evidenciar que en estos puntos el sistema oral obtuvo porcentajes más altos en casi todos los casos.

Entre los puntos negativos tenemos la consideración de que las instrucciones fueron demasiado repetitivas en el sistema escrito, aspecto que puede molestar al jugador debido a que el agente se puede asemejar más a una máquina. En cambio, esto no se evidenció en el sistema oral, muy probablemente debido a que para un mismo tipo de instrucción se reprodujeron audios diferentes, colapsando la monotonía del sistema escrito. En el sistema escrito, se consideró que el tiempo en pantalla de las instrucciones no fue el suficiente y que también se generaban las instrucciones muy temprano, razón por la cual muchos también concordaron en que el sistema les daba demasiada información a la vez. Gran parte de los jugadores en el sistema escrito se mostraron confundidos con la dirección en la cual tomar, argumentando confundir el verbo *Turn (left or right)* como doblar en lugar de girar. Por lo tanto, esto se vio provocado en gran parte por una mala interpretación del lenguaje inglés. En cambio, no ocurrió nada parecido en el sistema oral, quizás porque las instrucciones se dieron en el idioma nativo de los jugadores y además porque se utilizó un léxico más informal y menos técnico.

Como conclusión podemos observar que el sistema oral definitivamente obtuvo mejores resultados (en algunos casos mucho mejores como promedio de instrucciones generadas, promedio de fallas leves y tasa de duración del juego) que el sistema escrito.

Otras observaciones

En el registro de acciones en tiempo real pudimos observar que sólo un jugador recolectó un rayo incorrecto, que todos los jugadores eliminaron al enemigo y sólo uno de ellos recolectó items de veneno.

Capítulo 6

Conclusiones

6.1. Conclusiones

En este trabajo presentamos un agente que está capacitado para generar pistas que ayuden al jugador a ganar el nivel en un juego del tipo *First Person Shooter*. Dicho agente usa el estado del arte del razonamiento, como *planning*, con el fin de obtener el contenido de las instrucciones; y el estado del arte de técnicas de la generación de lenguaje natural para generar expresiones referenciales relacionadas con el contexto, y actos de *grounding*.

Actualmente, la mayoria de los tutoriales en los videojuegos hacen que el jugador siga un *script* fijo. As´ı, se deben contemplar todas las posibles reacciones por parte del jugador. Para lograr esto, los desarrolladores utilizan frecuentemente máquinas de estados finitos, que necesitan mantenerse pequeñas debido a problemas de escalabilidad[MF09]. Además, es muy difícil predecir todas las reacciones del jugador, por este motivo, cuando una nueva reacción es contemplada, el desarrollador debe agregarla en el código fuente. Nuestra propuesta puede verse como una manera de producir *scripts* automáticamente que cambian de acuerdo al comportamiento impredecible del jugador y el entorno cambiante, reduciendo as´ı, el trabajo para el desarrollador.

Hemos diseñado algoritmos de replanificación para entornos no determinísticos, es decir, donde existen diversos elementos que pueden cambiar de estado, con o sin intervención por parte del jugador, y que no siempre lo hacen de la misma manera. Estos algoritmos tuvieron como principal concepto, la visibilidad del jugador. Además hemos incluido un planificador estático dentro de un entorno dinámico. En nuestro caso, elegimos un planificador que asume un entorno estático.

En cuanto a posibles mejoras que podrían llevarse a cabo podemos encontrar la ambig uedad que existe cuando por ejemplo el agente le da la instrucción al jugador *Get that ray* cuando hay dos rayos visibles delante de él. El agente podría tener para cada instrucción, un conjunto de instrucciones similares que puedan sustituirse entre s´ı aleatoriamente. De esta forma el agente no dará tanta sensación de parecer una máquina que repite constantemente las mismas instrucciones, sino que realmente tiene un lenguaje más rico e incluso se podrían incluir varios idiomas.

Como hemos podido apreciar, en nuestro entorno tenemos un objetivo muy bien marcado: recoger los rayos en la secuencia correcta para poder eliminar al enemigo. Muy bien sabemos que hoy en d´ıa, en los videojuegos nos encontramos con diversos objetivos que var´ıan de un momento al otro de acuerdo a varios factores, como pueden serlo las diferentes reacciones de parte del jugador o del entorno. Por lo tanto, en el futuro debería analizarse como adaptar este tipo de sistemas a cambios dinámicos de objetivos. Podemos citar un ejemplo relacionado con el entorno utilizado: supongamos que el jugador está recolectando los rayos y el agente obviamente

guiándolo para lograr dicho objetivo; si aparece el enemigo, el agente en lugar de seguir dando instrucciones relacionadas a la recolección de rayos, debería darse cuenta que un cambio de objetivo es necesario, y éste debe tener como fin el alejamiento del enemigo. Luego de que dicho objetivo haya sido completado, puede volver nuevamente al objetivo original. Esto puede realizarse con un número ilimitado de objetivos, teniendo en cuenta cuales son más importantes y/o cuales afectan a los demás.

Con respecto a la verbosidad de las instrucciones, sería muy útil que el agente se adapte al conocimiento del jugador. De esta forma, el agente puede asumir que el jugador conoce mucha más información en determinado momento del juego, ya sea porque se familiarizó con el entorno o porque para jugar determinado tipo de juego se asume que el jugador tiene un bagaje teórico previo determinado. Por ejemplo, si el juego es un *First Person Shooter* como en nuestro caso, se asume que el jugador sabe las características del juego y no haŕa falta que el agente le explique los controles o cómo se juega. Para citar otro ejemplo, supongamos que el agente estuvo dando instrucciones para que el jugador vaya por determinados sitios del juego, luego el agente podrá asumir que si le indica al jugador que vaya a la habitación con paredes verdes, el jugador previamente tiene que haberla visto y sabrá como ir, sin necesariamente tener que indicarle el camino como se lo hizo tiempo atrás.

Como hemos podido apreciar con respecto a la evaluación del sistema, se obtuvieron resultados alentadores tanto en el sistema escrito como en el sistema oral. Primero y en principal, el juego atrapó y entretuvo a los participantes, que es nada más y nada menos que su objetivo principal. También en general la tasa de fallas graves fue baja y no se cometieron tantas fallas leves. Un punto en contra del sistema escrito fue la monotoneidad y repetición de instrucciones, situación que no ocurrió en el sistema oral debido a que para cada tipo de instrucción recolectamos varias posibles a ser reproducidas. Comparando ambos sistemas, el oral fue bastante mejor que el escrito en casi todos los puntos, sobretodo con respecto al entendimiento y claridad de las instrucciones, como así tambiíen con respecto a su repetición. Entre otras cosas, se vio una disminución de fallas graves y fallas leves, lo cual indica que el sistema oral fue también más efectivo.

6.2. Trabajo Futuro

Como trabajo futuro existen varias opciones que ser´ıa pertinente analizar. Por ejemplo, ser´ıa interesante realizar pruebas en escenarios más complejos, con múltiples objetivos y enemigos. Además podría utilizarse el agente en un escenario en donde no haya una única solución óptima y que dentro de un conjunto de soluciones aceptables, el agente utilice diferentes heur´ısticas para decidir cual plan seguir.

Por otra parte, en el sistema actual no se tiene en cuenta replanificación por aparición del enemigo, situación que sería importante tener en cuenta. Por último, debería probarse en un juego que sea de llegada masiva así se tiene un conjunto mucho más grande de evaluaciones y comentarios para analizar.

Apéndice A

Apéndice

A.1. Introduction

Nowadays, most game tutorials make the player follow a fixed script. As a result having a good or bad tutorial is still an art that depends on how good the script is at faking some freedom (for instance, by foreseeing possible reactions of the player) in order to make it more entertaining. *Natural language generation* can offer game designers with techniques that, in the future, may transform good tutorial design not in an art but in a science by providing insights and efficient algorithms for calculating the players knowledge and dealing with the players reactions in a dynamic way.

The goal of this work is to design and implement an agent (a computer-controlled character) which is able to generate hints that help a player advance in a game situated in a 3D virtual world. The agent knows information that is necessary to win the level and is not known by the player, hence the player needs to collaborate with the agent in order to finish the level successfully. Such agent uses state of the art reasoning algorithms such as *planning* in order to come up with the content of the instructions which are relevant at each point in the interaction. Furthermore, it applies techniques from situated natural language generation such as *common ground management* to generate the context-aware hints.

We believe that our contribution is interesting for the game industry as well as for the players. For players, this work is a first step towards more flexible, effective and entertaining tutorials. In order to sustain our claims we performed a human based evaluation. The results of the evaluation, though preliminary due to the amount of subjects, are indeed encouraging. For the game industry, good tutorials are expensive to develop because it is costly to script all the appropriate game behaviors caused by different potential player reactions (some of which might never be triggered). Our approach can be seen as a way of automatically producing scripts that change according to the unpredictable player behavior and the changing environment, reducing the burden on the developer.

The paper proceeds as follows. In Section A.2 we briefly introduce the area of natural language generation. Section A.3 describes the game environment where our game partner lives, and discusses the process of planning problem generation from the game environment. Section A.4 presents the process used to generate an instruction according to a plan. Section A.5 describes the integration of our generated instructions into the game interaction as well as the management of common ground between the player and the agent. Section A.6 presents the results of our human evaluation. Section A.7 concludes the paper.

A.2. Related work

Natural language generation (NLG) is a subfield of Artificial Intelligence (AI) and Computational Linguistics. It is concerned with the construction of computer systems which can produce understandable texts in English or other human languages from some underlying non-linguistic representation of information. NLG systems combine knowledge about language and the application domain to automatically produce documents, reports, explanations, help messages, and other kinds of texts.

The area of NLG is a new area of research, with less than a couple of decades of experience [RD00]. However, it is a rapid developing field that has put forward promising techniques in the last few years, in particular thanks to the shared challenges proposed for comparing current NLG technologies. One of such challenges is called GIVE (Giving Instructions in Virtual Environments). GIVE [KSG+10] is particularly relevant for our work since it applies NLG techniques to the problem of generating natural language instructions in a 3D virtual world. As a result several techniques investigated by GIVE's research community are directly applicable to this work. In the GIVE task, human players try to solve a treasure hunt in a virtual 3D world that they have not seen before. The computer has a complete symbolic representation of the virtual world. The challenge for the NLG system is to generate, in real time, natural-language instructions that can guide the player to the successful completion of their task. Only the player can effect any changes in the world, by moving around, manipulating objects, etc. Figure A.1 shows a screen-shot of the user's view on the 3D world. On the top of the picture, the current instruction given by the NLG system is displayed.

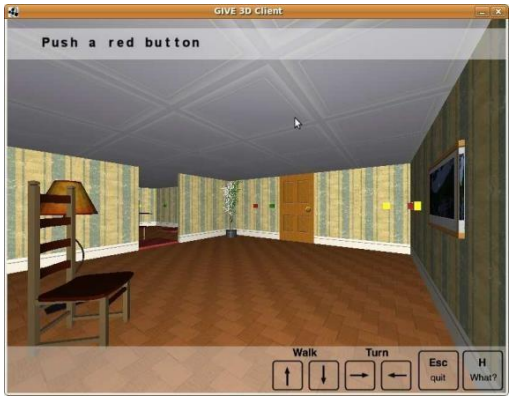

Figura A.1: The player's view in the GIVE Challenge.

The NLG tasks and techniques that are relevant for GIVE and for this work are content determination through planning, situated generation of referring expressions and common ground management; we will briefly discuss each of them.

The task of content determination consists in deciding which information to communicate to the player such that the information is appropriate at the current point in the interaction. Such task can be implemented using the inference technique of planning [Ben10]. As a result, the instructions are both relevant to the goal of the game and causally appropriate at the point in which they are uttered.

The situated generation of referring expressions area [RD00] provides algorithms for coming up with descriptions of objects so that the player can identify such objects (e.g. "the door in front of you") taking into account both static (e.g., color) and dynamic properties (e.g., visibility) of an object.

Finally, common ground management is the task of generating grounding acts when appropriate according to the behavior of the player. Grounding acts are utterances that, in a strict

sense, do not add new information to the discourse but instead they reinforce old information. For example, if the NLG system tells the player "take the left green kit" and the player turns slightly left to face an object, the system may generate a positive grounding act such as "yes" if that was the right object, or "no" otherwise [Tra94].

These three techniques are useful for generating hints that convey appropriate information to the player and that consider the player reaction in order to reinforce the information. In [GK10] the authors propose to cast all three tasks as a planning problem. Such integration is proven to be successful for small and discrete game worlds, however it does not scale to continuous game worlds in which the player can move freely (and continuously) in 3 dimensions. As a result in this paper we propose to use planning for handling only content determination while we use more traditional NLG methods for the other two tasks.

A.3. Finding plans from a game state

The design of game characters decision making is one of the most challenging tasks when designing a game; the expert game partner is, by no means, an exception to this rule. In most games, the characters decision making is either implemented in an ad-hoc way which is scripted for that particular game, or designed as a state machine which needs to be kept small because of scalability issues [MF09]. A believable game partner can not get away with a decision making process that is either scripted or small. First, the agent needs to react appropriately to infinitely many player reactions which cannot be predicted and scripted. And second, it needs to reason over complex game states in a goal directed way in order to be able to give instructions that are both causally appropriate at the point in which they are uttered and relevant to the goal of the game.

Our approach to implement the decision making process of the expert game partner (a.k.a content determination in the NLG area) is to use an off-the-shelf automated planner. The planner that we use is FastForward (FF). FF [Hof05] can handle big planning problems (with over a million objects) and return plans of thousands of actions in seconds and can handle our game environment (with a couple of hundreds of objects and plans with a maximum length of 250 actions) in a few milliseconds. Planning problems are specified in PDDL [GL05].

Automated planners are ready to be used in industry applications. They have reached a maturity level in which they can be used in real time applications even if the need of re-planning is high. In our setup there is a high need for re-planning because the behavior of the player is non-deterministic and unpredictable. Moreover, planners are domain independent, in order to come up with a plan which wins the game they just need an specification of the actions that are possible in the game and a discretized representation of what the agent knows about the state of the game. The agent knowledge about the region locations and adjacencies is discretized using the game waypoints [MF09].

The game environment for which we have implemented our game partner lives is first person shooter game (FPS). The game scenario, called Igor[1], was developed using the Irrwizard framework[2] and extended in C++. As in most FPS games the player is situated in a 3D world where he can perform several actions such as walk, jump, climb stairs, shoot and pick up different items which have different effects. The goal of the game is to kill a creature that is wandering in the 3D world. The creature cannot be killed only by shooting at it because it has a self healing mechanism that needs to be turned off first by deactivating a series of power rays in a given order. The NLG agents knows which is the right sequence of rays as well as the position of each of these rays. It is also able to recognize other items (such as poison or health) and is aware of their effect. All this information is stored in the specification that is sent to the planner and the

[1]https://sites.google.com/site/nicolasbertoa/igor
[2]http://irrwizard.sourceforge.net/

planner returns a sequence of actions that, if executed in the current state of the game, would achieve the goal of the game.

The agent must extract information from the environment and represent it in the planner language: PDDL [GL05]. PDDL is intended to express the physics of a domain, that is, what facts are true in the world, what actions are possible, and what the preconditions and effects of actions are. The agent represents the number of rays that the player needs to pick up and its order. Also, the agent codifies in the planning problem the waypoints and adjacencies which are used by the planner to find the nearest path between the player and the items to pick up. Moreover, the agent needs information about the player and the enemy position and state. The agent constantly verify the health level of the player and enemy and updates its condition (attacking, wandering, dead etc) in the planning problem.

Once the agent has joined all the information about the problem, it generates a planning problem and sends to the planner. Then the planner generates a plan and the agent uses the plan to generate the instructions (see next section). The planner takes, in average, 8.61 milliseconds to find a plan for our game planning domain (with a standard deviation of 7.46 milliseconds). The maximum length of a plan in our game environment is 250 actions (notice that plans and plan length varies depending of the current state of the interaction). Once a plan was obtained, the agent needs information about the environment to check if the plan was accomplished or not, and to know the situations in which must re-plan (we address these issues in Section A.5).

A.4. Using plans to decide what to say

This section explains how the agent uses the plan obtained from the planner. The plan is composed by a series of steps that the player must follow to meet the plan goal. When the agent has a new plan, it must decide what instruction to say next. Suppose that the planner give us the plan steps that shown in Figure A.2. This plan contains two types of actions, *MoveTo* and *TakeKey*. The first type of action indicates that the player has to move from one way-point to another, and the second indicates the ray that the player hast to pick as well as the way-point where it is located.

```
MoveTo  w4    w5
MoveTo  w5    w6
MoveTo  w6    w7
MoveTo  w7    w8
MoveTo  w8    w9
MoveTo  w9    w10
MoveTo  w10   w11
TakeKey blueKey w11
```

Figura A.2: Plan steps generated by the planner.

These steps a plan to pick up the ray *blueKey* as illustrated in Figure A.3. Now, the agent has the problem of deciding which plans steps to verbalize, that is it needs to do content deter-

mination based on the plan steps. Notice that the naive approach of verbalizing all plan steps would result in a very repetitive and over restrictive game partner. Therefore we made the content determination process of our agent dependent of the actions whose waypoints are directly visible to the player or visible by turning around on his current position, we will say that this waypoints are *visible 360*.

The Figure A.3 shows that there are only three visible waypoints of the plan at 360 degrees from the position of the player. These are the waypoints 4, 5 and 6. The others are blocked by the environment. From the visible waypoints we verbalize the one that has a referring expression which uniquely identifies it. In our case, the waypoints 4 and 6 are waypoints that are in the middle of rooms, so, there are no referring expressions to unambiguously describe them. But in the case of the waypoint 5, there is a door, so we can use a referring expression to uniquely identify the object involved. As a result, in the situation illustrated in Figure A.2 the instruction "go through the door in front of you" is generated.

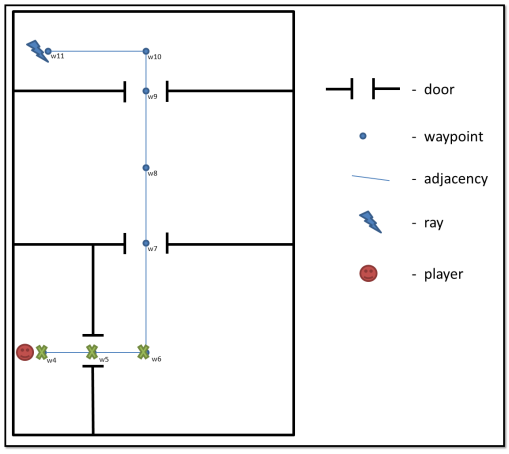

Figura A.3: Path of the plan in Figure A.2 with waypoints visible 360 highlighted (one uniquely referable).

But, what happens in the case when there are more than one object that can be uniquely identified? The Figure A.4 shows this case. There are seven visible waypoints of the plan at 360 degrees from the position of the player, and three of them have an object that can be uniquely identified: 5 (has a small door), 7 (has a big door), and 9 (has some stairs). In this case, the agent generates an instruction which refers to the the furthest visible object in the path of the plan and verbalizes "see those stairs in front of you?" before saying "take them" (these strategy is used when referring to objects that are far away from the player, see Section A.5 for details).

The screen-shots in Figure A.8 illustrate the process we just explained. In the screen-shot we have drawn the visible waypoints and the paths between them.The arcs between waypoints illustrate the actions in the plan. In the screen-shot in the top of Figure A.8 the planned actions could be verbalized as "go forward, go through the door, go forward, go forward, do you see those stairs in front of you?, take them, go forward". As we said, we have decided to verbalize the last action in the sequence which contains a referring expression which uniquely identifies the object involved in the action. In this example, this is the case for the action "do you see those stairs in front of you?" since "those stairs" are a distinguishing referring expression (given the current position of the player) for the stairs that are further away. We say that the first actions, namely "go forward, go through the door, go forward, go forward" are *left tacit* [Ben07] and they are expected to be inferred by the player given the causal constraints of the world (in simpler terms, in order to see "those stairs" better the player will need to approach them). The screen-shot at the bottom of Figure A.8 ("Turn left"), illustrates an instruction whose goal is to make directly visible a waypoint that is visible 360.

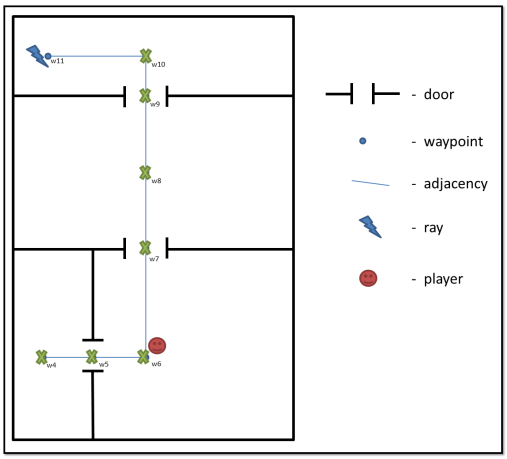

Figura A.4: Path of the plan in Figure A.2 with waypoints visible 360 highlighted (many uniquely referable).

Figura A.5: Instructions generated using the plan and the player's visibility

A.5. Interaction and common ground

Once the agent generates the instruction, it will show that instruction to the player as illustrated in Figure A.8. Of course, the player may decide to follow the instruction or to do something else. A good game partner should handle both situations robustly; we discuss our strategy for doing this in Section A.5.1. Furthermore, not only the player but also our environment can behave in a non deterministic way. The game partner should be able to sense and react to a changing environment in an appropriate way; Section A.5.2 addresses this issue. In both of this situations grounding acts can be use to reinforce instructions that were already communicated but were not successfully accomplished yet; we illustrate how positive and negative grounding acts are used by our agent in Section A.5.3.

A.5.1. Dealing with unpredictable behavior

Our strategy for dealing with the player's unpredictable behavior is to find a new plan (that is, to replan) every time the player gets *too far away* from the current plan. The crucial point here is to define what it means to get too far from the current plan. Since we base our content determination procedure in the player visibility we also use the same strategy to decide when to replan. The agent will replan when all waypoint in the plan are no longer visible 360 for the player.

Let's illustrate our strategy by an example. Figure A.6a shows an example scenario and the waypoints of the plan which are visible 360.

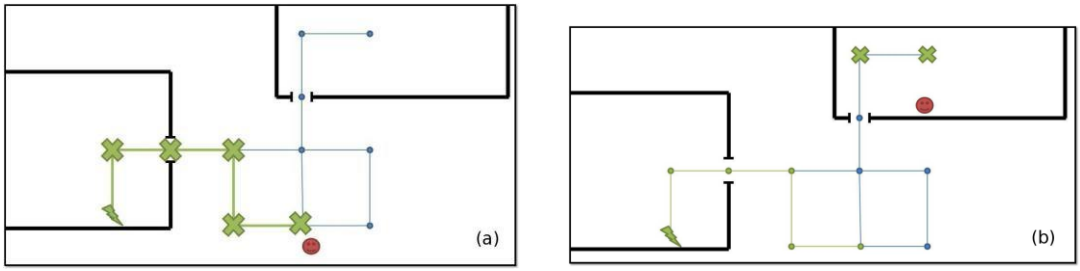

Figura A.6: Re-planning example.

Now suppose that the player does not follow the instructions and ends up in the situation illustrated in Figure A.6b. In this new situation re-planning is needed because none of the waypoints in the plan is visible anymore from the current player position.

Summing up, the agent will re-plan when there are not visible waypoints of the plan, because is very difficult that the agent achieves the reorientation of the player towards the goal. This strategy results effective (and not overly restrictive) while guiding the player to the game goal as shown by our evaluation results in Section A.6.

A.5.2. Dealing with a changing environment

Suppose we have 4 ray items: blue, green, violet and red and the correct sequence to deactivate the enemy's defense mechanism is red, blue and red. The Figure A.7 shows the plan that the agent obtained from the planner (the plan is simplified not to show move actions for presentation simplicity). What the planner cannot handle is the fact that, when the player picks up some object of the game, the object repositions randomly in another way-point. For example, in the Figure A.8 we can see that the obtained plan is incorrect because the red ray was repositioned at a different way-point. The planner assumes a deterministic environment, we deal with a non deterministic environment, such as our game, by means of replanning. That is, when the position the environment changes in a non deterministic way (for example when a ray changes its position), the agent replans.

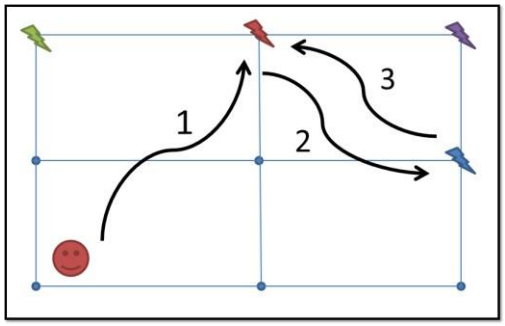

Figura A.7: Steps to get the sequence of rays.

A.5.3. The importance of the grounding acts

Grounding acts are utterances that, in a strict sense, do not add new information to the discourse but instead they reinforce old information. Grounding acts are not required from an

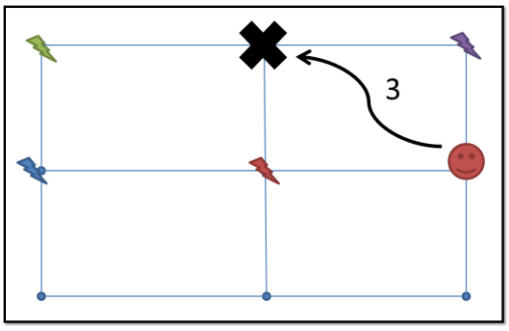

Figura A.8: The plan was incorrect due dynamic properties of rays.

informational point of view of communication, however, they play an important role in order to cope with changing environments, and the unpredictable behavior of the player.

The Figure A.9 shows a case of the positive grounding act “Get this ray” which follows the instruction “Turn left to see the ray” when the ray becomes visible. This utterance is a positive grounding act because, it does not add new information to the discourse. The player should be able to figure out that he should pick up this ray, otherwise, why would have the agent guided him to it? However, natural language is ambiguous and the player may not draw this conclusion so the reinforcing achieved by the grounding act is indeed useful.

The other screen-shot in the Figure A.9 illustrates a case of negative grounding acts with “This is not the ray you need”, which follows the instruction “we need to find the green ray”, when the player hovers the mouse pointer on the red ray. With this instruction, the agent prevents the player from re-activating the protection mechanism of the creature as a result of picking a ray in the wrong sequence. Again, in a strict sense, the negative grounding act is not necessary because the player has already been told that the next ray that is needed is green. However, the player may not remember this and may try to take the red ray which is directly in front of him.

Figura A.9: Grounding acts that can be generated as a reaction to the player actions

The screen-shots in Figure A.10 illustrates a typical example of common ground creation between the agent and the player. The screen-shot in the top shows the instruction “Do you see that ray in front of you?”. With this instruction the agent wants that the player to focus his attention in a ray with is in front of him. As a result it is to be expected that the player gets closer to the ray. In this new situation, the agent generates the instruction “Pick it up”. It is clear from this instruction that the agent wants that the player picks up the blue ray in front of him, but we can see that neither the ray nor its position are included in the instruction. This can be done because this information is already in the the common ground between the agent and the player.

Figura A.10: Multi-utterance instructions which create and use the common ground

A.6. User evaluation

In this section we describe the results of the human evaluation of our agent. You can see a gameplay video at https://sites.google.com/site/nicolasbertoa/ai4games_video.

For our evaluation we used objective and subjective metrics. Objective metrics were collected by logging the player behavior and the subjective metrics by asking players to complete a questionnaire after finishing the tutorial level. We used the same metrics that are used in the GIVE Challenge [KSG+10] to evaluate systems in terms of the effectiveness, naturalness of instructions and the engagement of the interaction. We gathered 10 volunteers for the evaluation. Each of them played the game in his own. The demographic characteristics of the volunteers that we collected are the following: they were all male, the average age was 20 years old and all of them were gamers. This subject population is the target market of the kind of game we implemented.

The results that we obtained using the objective metrics are shown in Figures A.11 and A.12.

The Figure A.11 shows the number of successful instructions, minor faults and serious faults. An instruction is considered successful if the player did exactly what the system asked him to do, an instruction is considered a minor fault if the player deviated from the instruction without causing a replanning, and an instruction is a serious fault if the agent had to replan after uttering it. Their averages are 44.1, 28.9 and 4.4, and their standard deviations are 9.74, 17.04 and 3.92, respectively. We can see that there were very few serious faults, this suggests that the agent was successful in the task of guiding the player. Also, we can see that players deviated from 40 % of the instructions and, according to the game logs, this is due mainly to the presence of enemy which obviously causes the player to not follow the plan. Finally, approximately 60 % of the instructions were directly successful, the player did exactly what the agent asked him to do in more than half of the cases.

The Figure A.12 shows the average of the other objective metrics we collected. The successful instructions were completed quickly, which suggests that the instructions were easy to understand and execute. Also, players were able to complete the level quickly without visiting all the map waypoints (they only visited, in average, 70 % of the map).

The Figure A.13 shows the results of the subjective metrics. On the one hand, the players considered that the instructions were too repetitive. Also, they considered that the instructions were few time at the screen and that the instructions were generated too early. Also, some players said that sometimes they were confused about the direction to go in. On the other hand, the statements that are related to entertainment obtained very high percentages, which suggests that the player had fun, that is the main objective of any game. Another strong point is that the instructions were very clearly worded and were understood. Also, players perceived that most of the time the agent helped to the players when they were confused.

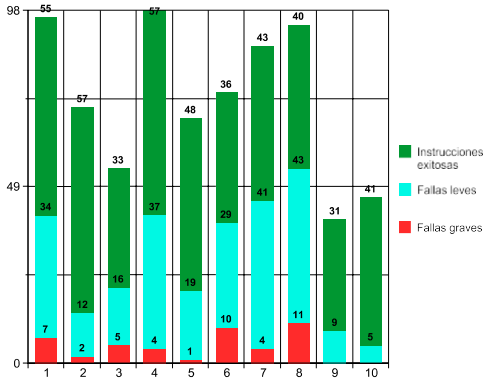

Figura A.11: Objective metrics: Successful instructions, minor and serious faults per player.

	Average	Standard deviation
Time of completeness per instruction (seconds)	28.9	17.04
Waypoints traveled	65.5	18.76
Playing time (minutes)	2.4	1.29

Figura A.12: Objective metrics: average times and distance traveled.

Number	Statement	%
1	The system used words and phrases that were easy to understand	80
2	I had to re-read instructions to understand what I needed to do	70
3	The system gave me useful feedback about my progress	50
4	I was confused about what to do next	75
5	I was confused about which direction to go in	80
6	I had no difficulty with identifying the objects the system described for me	80
7	The system gave me a lot of unnecessary information	10
8	The system gave me too much information all at once	75
9	The system immediately offered help when I was in trouble	80
10	The system sent instructions too late	50
11	The system's instructions were delivered too early	10
12	The system's instruction were visible long enough for me to read them	35
13	The system's instructions were clearly worded	90
14	The system's instruction were repetitive	90
15	I really wanted to kill that creature	95
16	I lost track of time while solving the overall task	80
17	I enjoyed solving the overall task	90
18	Interacting with the system was really annoying	55
19	I would recommend this game to a friend	70
20	The system was very friendly	70
21	I felt I could trust the system's instructions	60

Figura A.13: Results of the subjective metrics.

We consider that these results are encouraging, in particular because one of the main goals of our agent was to be able to participate in an engaging interaction while still providing useful instructions whenever the player needed help. We are considering different techniques to improve the agent in those characteristics in which it did not get such good results (for example, by doing corpus based generation in order to avoid being repetitive [GT07]).

A.7. Conclusions

In this paper we have presented an agent which is able to generate hints that help a player win a level in first person shooter game. Such agent uses state of the art reasoning such as planning in order to come up with the content of the instructions and state of the art techniques from natural language generation to context-aware generate referring expressions and grounding acts.

Currently, most game tutorials make the player follow a fixed script. It must contemplate

as many situations as possible. To achieve that, the developers often use state machines, which needs to be kept small because of scalability issues [MF09]. Also, it is very difficult to predict all the player's reactions, therefore, when a reaction is contemplated, the developer must add new source code. Our approach is a way of automatically producing scripts that change according to the unpredictable player behavior and the changing environment, reducing the burden of the developer. This paper is a first step to motivate the interaction between the game industry and the research area of generation of natural language.

Bibliografía

[007] Wikipedia Golden Eye 007. http://es.wikipedia.org/wiki/GoldenEye 007.

[aSS] Wikipedia Beneath a Steel Sky. http://en.wikipedia.org/wiki/Beneath_a_Steel_Sky.

[Axe] Wikipedia Golden Axe. http://es.wikipedia.org/wiki/Golden_ Axe.

[Ben07] Luciana Benotti. Incomplete knowledge and tacit action: Enlightened update in a dialogue game. In *Workshop on the Semantics and Pragmatics of Dialogue*, pages 17–24, Rovereto, Italy, 2007.

[Ben09] Luciana Benotti. Frolog: an accommodating text-adventure game. In *Proceedings of the Demonstrations Session at the 12th Conference of the European Chapter of the ACL*, pages 1–4, Athens, Greece, 2009. Association for Computational Linguistics.

[Ben10] Luciana Benotti. *Implicature as an Interactive Process*. PhD thesis, Université Henri Poincaré, INRIA Nancy Grand Est, France, 2010. Supervised by P. Blackburn. Reviewed by N. Asher and B. Geurts.

[Bla] Henry Kautz BlackBox. http://www.cs.rochester.edu/u/kautz/satplan/blackbox/index.html.

[BW] Wikipedia Black and White. http://es.wikipedia.org/wiki/Black 26 White.

[CB91] Clark-Brennan. Grounding in communication. pages 57–62, 1991.

[Cha] GIVE Challenge. GIVE Challenge - http://www.give-challenge.org/research/.

[Cha06] David Chapell. *Understanding .NET*. Addison-Wesley Professional, 2nd edition edition, 2006.

[Cor] Wikipedia Valve Corporation. http://es.wikipedia.org/wiki/Valve.

[Cre] Wikipedia Creatures. http://en.wikipedia.org/wiki/Creatures (artificial life program).

[dln] Wikipedia Procesamiento de lenguaje natural. http://es.wikipedia.org/wiki/Procesamiento de lengua

[Ent] Wikipedia Blizzard Entertainment. http://es.wikipedia.org/wiki/Blizzard.

[Gama] Wikipedia Firaxis Games. http://es.wikipedia.org/wiki/Firaxis Games.

[Gamb] Wikipedia Midway Games. http://en.wikipedia.org/wiki/Midway Games.

[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.

[Gha98] Malik Ghallab. Pddl: The planning domain definition language. 1998.

[GK10] Konstantina Garoufi and Alexander Koller. Automated planning for situated natural language generation. In *Proceedings of the 48th ACL*, Uppsala, 2010.

[GL05] Alfonso Gerevini and Derek Long. Plan constraints and preferences in PDDL3. Technical Report R.T. 2005-08-47, Università degli Studi di Brescia, Italy, 2005.

[GNA+98] Malik Ghallab, Ecole Nationale, Constructions Aeronautiques, Craig Knoblock Isi, Keith Golden, Scott Penberthy, David E Smith, Ying Sun, Daniel Weld, and Contact Drew Mcdermott. Pddl - the planning domain definition language, version 1.2. Technical report, 1998.

[GT07] Sudeep Gandhe and David Traum. Creating spoken dialogue characters from corpora without annotations. In *Proceedings of Interspeech*, Belgium, 2007.

[Hof01] Jörg Hoffmann. Ff: The fast-forward planning system. *AI Magazine*, 22(3):57–62, 2001.

[Hof05] Jörg Hoffmann. Where 'ignoring delete lists' works: Local search topology in planning benchmarks. *Journal of Artificial Intelligence Research*, 24:685–758, 2005.

[III] Wikipedia Civilization III. http://es.wikipedia.org/wiki/Civilization_III:_Conquests.

[Int] Wikipedia Artificial Intelligence. http://en.wikipedia.org/wiki/Artificial_intelligence.

[Inv] Wikipedia Space Invaders. http://es.wikipedia.org/wiki/Space Invaders.

[Kon] Wikipedia Konami. http://en.wikipedia.org/wiki/Konami.

[KSG+10] Alexander Koller, Kristina Striegnitz, Andrew Gargett, Donna Byron, Justine Cassell, Robert Dale, Johanna Moore, and Jon Oberlander. Report on the second nlg challenge on generating instructions in virtual environments (give-2). In *Proceedings of the International Natural Language Generation Conference (INLG)*, Dublin, 2010.

[Lif] Wikipedia Half Life. http://es.wikipedia.org/wiki/Half-Life.

[LN11] Benotti Luciana and Bertoa Nicolás. Content determination through planning for flexible game tutorials. lecture notes in computer science. 2011.

[Lun08] Frank Luna. *Introduction to 3D Game Programming with DirectX 10*. Jones and Bartlett Publishers, 1st edition edition, 2008.

[Man] Wikipedia Pac Man. http://es.wikipedia.org/wiki/Pac-Man.

[Max] Wikipedia Maxis. http://es.wikipedia.org/wiki/Maxis.

[MF09] Ian Millington and John Funge. *Artificial Intelligence for Games*. Morgan Kaufmann, Burlington, MA, USA, 2009.

[Min] Wikipedia Mindscape. http://es.wikipedia.org/wiki/Mindscape.

[Ome] Wikipedia Warhammer: Dark Omen. http://en.wikipedia.org/wiki/Warhammer:_Dark_Omen.

[OR07] Jeff Orkin and Deb Roy. The restaurant game: Learning social behavior and language from thousands of players online. In *Journal of Game Development*, pages 39–60, 2007.

[oW] Wikipedia World of Warcraft. http://es.wikipedia.org/wiki/World_of_Warcraft.

[Pon] Wikipedia Pong. http://es.wikipedia.org/wiki/Pong.

[Pro] Wikipedia Thief: The Dark Project. http://es.wikipedia.org/wiki/Thief:_The_Dark_Project.

[Rar] Wikipedia Rare. http://es.wikipedia.org/wiki/Rare.

[RD00] Ehud Reiter and Robert Dale. *Building natural language generation systems*. Cambridge University Press, New York, NY, USA, 2000.

[Rev] Wikipedia Reversi. http://es.wikipedia.org/wiki/Reversi.

[RN03] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition, 2003.

[RPG] Wikipedia RPG. http://es.wikipedia.org/wiki/Juego_de_rol.

[RTS] Wikipedia RTS. http://en.wikipedia.org/wiki/Real-time strategy.

[Sch09] Jesse Schell. *The Art of Game Design*. Elsevier, Burlington, MA, USA, 2009.

[Sec] Wikipedia SecondLife. http://es.wikipedia.org/wiki/Second Life.

[SEG] Wikipedia SEGA. http://es.wikipedia.org/wiki/Sega.

[Sho] Wikipedia First Person Shooter. http://en.wikipedia.org/wiki/First-person shooter.

[Shr09] Dave Shreiner. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Versions 3.0 and 3.1*. Addison-Wesley Professional, 7 edition edition, 2009.

[Sim] Wikipedia Los Sims. http://es.wikipedia.org/wiki/Los Sims.

[Sof] Wikipedia Revolution Software. http://es.wikipedia.org/wiki/Revolution Software.

[Sol] Wikipedia Metal Gear Solid. http://es.wikipedia.org/wiki/Metal Gear.

[Str00] Bjarne Stroustrup. *The C++ Programming Language: Special Edition*. Addison-Wesley Professional, 3rd edition edition, 2000.

[Stua] Wikipedia Lionhead Studios. http://es.wikipedia.org/wiki/Lionhead Studios.

[Stub] Wikipedia Looking Glass Studios. http://en.wikipedia.org/wiki/Looking_Glass Studios.

[Stuc] Wikipedia Pandemic Studios. http://es.wikipedia.org/wiki/Pandemic Studios.

[Tec] Wikipedia Cyberlife Technology. http://en.wikipedia.org/wiki/Cyberlife Technology.

[Tra94] David Traum. *A Computational Theory of Grounding in Natural Language Conversation*. PhD thesis, Computer Science Dept., U. Rochester, USA, 1994. Supervised by James Allen.

[Wara] Wikipedia Warcraft. http://en.wikipedia.org/wiki/Warcraft.

[Warb] Wikipedia Full Spectrum Warrior. http://en.wikipedia.org/wiki/Full Spectrum Warrior.

[Zel] Wikipedia Zelda. http://es.wikipedia.org/wiki/The_Legend_of_Zelda.