Trabajo de Fin de Grado

Grado en Ingeniería Informática

Desarrollo de un Videojuego 2D Procedural

Development of a Procedural 2D Videogame

Luis David Padilla Martín

La Laguna, 4 de septiembre de 2018

D. **Estévez Damas, José Ignacio**, con N.I.F. 43786097-P profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor.

C E R T I F I C A

Que la presente memoria titulada:

“Desarrollo de un Videojuego 2D Procedural”

ha sido realizada bajo su dirección por D. **Padilla Martín, Luis David** ,

con N.I.F. 79070828-W .

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 4 de septiembre de 2018

Agradecimientos

En primer lugar me gustaría agradecer a mi tutor José Ignacio Estévez Damas por su colaboración a lo largo del desarrollo de este proyecto. Además durante el trabajo he adquirido numerosos conocimientos sobre el mundo de los videojuegos y sobre el desarrollo gracias a su inestimable ayuda. Su dedicación ha sido de gran importancia para la realización de este proyecto.

También me gustaría agradecer al resto de profesorado del centro que me han guiado por estos años de carreras y me ha llevado a estar donde estoy.

A mi compañeros de carreras por todos esos buenos momentos que hemos pasado durante estos años juntos.

Por último y no por ello menos importantes a mi familia por todo el sacrificio que han hecho para hacerme llegar a donde estoy.

Licencia

Resumen

El objetivo que se propone para este proyecto es la creación de un videojuego que se genere completamente de manera procedural, haciendo especial hincapié en que todos los puzles que aparecen en este sean completamente resolubles. Mediante la utilización de un lenguaje declarativo se generarán una serie de puzles de manera aleatoria, a partir de las soluciones obtenidas y mediante un motor gráfico serán convertidas en una pantalla del videojuego.

Palabras clave: Videojuego, Procedural, puzles, Lenguaje declarativo.

Abstract

The objective of this project is create a procedural videogame, with special emphasis on which all puzzles that appear in the game have a possible solution. With a declarative language we will make a series of random puzzle. Using a game engine we will take one solution a make a game level of the game.

Keywords: Videogame, Procedural generation, puzle, Declarative Language.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1
Introducción

1.1 Motivación Personal

En el año 2013 comencé la carrera de Ingeniería Informática en la Universidad de La Laguna debido a que tenía interés en todo lo aquello que tenía que ver con el desarrollo de aplicaciones, programas y sobretodo juegos. Por ello mi ambición es conseguir llegar a ser desarrollador de videojuegos. Un Trabajo de Fin de Grado relacionado con la programación de los mismos es una buena antesala a futuros proyectos personales o profesionales. Además los videojuegos me han acompañado a lo largo de mi vida y se han convertido en parte imprescindible de mi día a día y me ha influenciado y convertido en la persona que soy yo ahora. No solo me han proporcionado horas y horas de entretenimiento sino me han permitido interaccionar y conocer muchas personas de distintas partes del mundo.

Por otro lado el mercado de los videojuegos es uno de los más influyentes y que más dinero mueven de la actualidad. Conocer su funcionamiento es importante para ser un buen profesional en el mundo del desarrollo de software. Los videojuegos son un mundo complejo y que se dividen en muchos campos como el desarrollo de motor gráfico,diseño de modelos y niveles y la realización de sistemas scripts que permitan la realización de acciones durante el tiempo de juego.

1.2 Relevancia de los Videojuegos

Los videojuegos según el diccionario de la RAE son “Juegos electrónicos que se visualizan en una pantalla”. Estos tienen un orígen en la década de los 50 ,cuando en los primeros ordenadores se empezaron a crear programas de carácter lúdico. A partir de 1970 empezaron a crearse las primeras “consolas” y juegos dirigidos a la audiencia general. A medida que pasaron los años las tecnologías de desarrollo se fueron refinando y creando nuevos juegos más sofisticados. Un ejemplo de esta mejora fue la aparición en los 90 de la tecnología de 16-bit que trajó a este campo un avance en los gráficos. En las últimas décadas con la aparición de smartphones, tablet y el abaratamiento de los componentes de los PC los videojuegos se han expandido a una gran audiencia, haciendo que aumente su importancia en la actualidad. Tanto es así

que ahora el mundo de los videojuegos es uno de los mercados más rentables.

Además los videojuegos no solo tienen gran importancia en el ámbito económico sino que también destacan en otros campos como la educación. Varios estudios han demostrado que los niños responden mejor en la enseñanza si se utilizan juegos ya que estos consiguen captar mejor el interés de los mismos. Un ejemplo de la utilización de videojuegos para ayudar a los niños en el aprendizaje es la enseñanza de niños con déficit de atención ya que mejoran la atención es estos casos, a diferencia de la enseñanza clásica.

Otro campo donde destacan los videojuegos es en la medicina donde son utilizados como terapia ya que resulta más sencillo de entender y realizar ciertos ejercicios(de fisioterapia) si se plantean como juegos. Además los juegos sirven para hacer más entretenida las largas estancias de algunos pacientes en los hospitales y les permiten interactuar con personas fuera de este entorno. Otra utilidad de los videojuegos es que pueden servir como apoyo en las relaciones sociales lo que los hacen ideales para ayudar a las personas con problemas para las interacción social.

Por último cabe destacar que los videojuegos al ser tan relevantes en tantos campos y tener tanta importancia en el mundo actual, es necesario el estudio de los mismos para conocer cuáles han sido los factores que han permitido su desarrollo hasta llegar al presente.

1.3 El concepto de generación procedural en los videojuegos

La utilización de sistemas de generación procedural para la creación de videojuegos es una técnica clásica empleada desde los años 70 y que con el tiempo ha sufrido enormes sofisticaciones. El interés en la misma se deriva del objetivo de crear videojuegos no repetitivos que no impliquen costes adicionales tanto en la fase de desarrollo (tiempo de desarrollo) como en la utilización de los propios recursos de las plataformas donde se juegan.

Entre los métodos más simples de sistemas pseudoaleatorios con comprobación de restricciones hasta los más complejos donde se deben cumplir complejas restricciones y condiciones, existe un amplio abanico de posibilidades intermedias. En este proyecto se pretende ensayar la generación procedural de un videojuego 2D donde la fuente de complejidad vendrá dada por la elección y ubicación de un conjunto de puzles. Una posible solución para este tipo de problemas es la utilización de resolvedores de problemas con restricciones, descriptores declarativos y dependientes de múltiples parámetros.

La generación procedural en videojuegos es una técnica que se remonta a los 70 donde se generaban mazmorras pseudoaleatorias para videojuegos tan antiguos que sus representaciones se basan en caracteres ASCII. El “Roguelike”, es un claro ejemplo de la utilización de generación procedural en videojuegos, ya que es un género donde se aprovechan las distintas técnicas de generación aleatoria para la creación de pasillos, monstruos y tesoros de manera compleja sin la necesidad de tener que gastar recursos en la creación del mundo. Rogue, el mayor referente de este género, consistía en la exploración de una mazmorra donde cada uno de los niveles se genera al azar.

Otro ejemplo de la aplicación de esta técnica es Spelunky. Se trata de un juego indie open source cuya trama consistía en explorar una caverna consiguiendo tesoros, luchando contra enemigos y esquivando trampas. Este juego tiene varios tipos de habitaciones donde cada una sigue diferentes reglas. En función de estas reglas se van eligiendo cada una de las habitaciones para rellenar una matriz de 4x4.

En los últimos años en el mercado se puede encontrar juegos como Dead Cells que mediante la parametrización de la generación procedural se controla el nivel de verticalidad u horizontalidad de la misma en función del nivel en el que se encuentre el jugador.

Actualmente a pesar del refinamiento de la generación procedural no es una técnica muy utilizada ya que es difícil encontrar la manera de que el jugador no pierda el interés en el juego. A pesar de esto hay ámbitos donde se usa constantemente. Como por ejemplo en las fase de desarrollo se utiliza la generación procedural para diseñar paisajes naturales con terrenos, bosques, plantas y caminos.

La generación procedural plantea problemas cuando el mundo a generar incluye restricciones. Por ejemplo, podría necesitar un laberinto generado proceduralmente, pero con características acordes al nivel del juego y el problema a resolver. También podría ser necesario generar puzles aleatorios, pero obviamente con la restricción de que puedan ser resolubles por el jugador. Este tipo de generaciones pueden enfocarse de dos modos. La primera consiste en programar algoritmos específicos para cada tipo de problema. La segunda, se basaría en sistemas generales de resolución de problemas con restricciones, para lo que sería necesario codificar el problema de modo que se adaptara a los requerimientos del resolvedor.

1.4 Objetivos

El objetivo principal de este trabajo es la realización de un videojuego en 2D donde cualquier nivel se genere de manera aleatoria siguiendo una serie de parámetros y que todos los puzles que aparezcan en el mismo puedan ser resueltos.

- Algoritmo de generación de laberintos: Este objetivo consiste en la realización de un laberinto en el entorno gráfico Unity donde ocurrirá la acción del videojuego
- Codificación de un puzle básico mediante un lenguaje declarativo(ASP): Para realizar esta tarea se usará el sistema Clingo, que se describe más adelante en esta memoria.
- Integración Unity-clingo: Con un sistema para generar instancias de un puzle básico el siguiente paso consistirá en su aprovechamiento desde un motor gráfico, en este caso Unity. Esta implementación debe caracterizarse por la obtención del puzle en tiempo de ejecución del juego mediante una orden a la que se le pueden pasar parámetros.
- Implementación de nuevos puzles: Mediante clingo se realizarán nuevos puzles para darle mayor complejidad al juego.
- Implementación de demo jugable: Añadiendo un personaje jugable y

otros elementos como una interfaz y así realizar un prototipo demo del futuro juego.

- La última tarea será añadir algún puzle más complejo, como por ejemplo que suponga cambiar el estado del laberinto.

Capítulo 2
Antecedentes

2.1 Antecedentes en la generación procedural aplicada a videojuegos

2.1.1 Rogue

Rogue es un videojuego de mazmorra creado en 1980 [1]. Este consistía en que el jugador debía explorar una mazmorra generada al azar, recuperar un amuleto y volver al principio del juego. A partir de este juego se inspiró todo un género el “Roguelike” [2], que al igual que su predecesor se regía por una serie de normas:

- Juegos para un jugador.
- Basados en sistemas por turnos.
- Énfasis en la generación procedural: la disposición de las habitaciones, los enemigos y los objetos.
- Énfasis en la jugabilidad.
- Muerte permanente: tras fracasar el jugador debe empezar una partida completamente nueva sin recuperar ninguno de los logros obtenidos.
- Una alta dificultad.
- Premisa sencilla, el único objetivo es llegar al final.

A pesar de que estas premisas son las que definen este género no todos los juegos pertenecientes al mismo cumplen todos las reglas. Un ejemplo claro es Diablo donde la muerte solo penaliza en algunos aspectos del juego y solo es permanente si se juega en el modo más difícil. En cuantos a los gráficos la mayoría de casos están basado en mazmorras y en el caso de Rogue comenzó siendo una matriz de símbolos ASCII ya que estaba pensado para ser jugado en una terminal Unix. Posteriormente y con el avance del género se fueron añadiendo plantillas mejorando el aspecto gráfico hasta llegar a tener componentes gráficos de última generación.

2.1.2 Spelunky

Spelunky es un videojuego indie lanzado en 2008 que consistía en un aventurero cuyo objetivo es explorar una serie de cavernas [3]. La principal característica de este juego es que cada nivel es generado de manera completamente aleatoria dando así gran variedad al juego. Otra característica fundamental de este juego es que a pesar de ser generado completamente al azar los niveles siempre tienen una solución. Para la generación del nivel se creaba una matriz de 4x4 que se rellenaba con distintos tipos de habitación. Los tipos de habitación son lo siguientes [4]:

- Tipo 0: habitaciones que no son parte de la solución y solo están ahí para completar cada nivel.
- Tipo 1: habitaciones que poseen una salida tanto a la izquierda como a la derecha.
- Tipo 2: habitaciones que poseen una salida tanto a la izquierda como a la derecha y además una salida abajo. Si además se encuentra debajo de otra habitación de tipo 2 también dispondría de una salida adicional en la parte superior.
- Tipo 3: habitaciones que poseen una salida tanto a la izquierda como a la derecha y además un salida arriba.

Para asegurar que siempre haya una solución al problema estas habitaciones se disponían de la siguiente manera:

La habitación inicial siempre se elegía de manera aleatoria y se encontraba en la primera fila de la matriz. Posteriormente se seleccionaba un número aleatorio del 1 al 5. Si este número era 1 o 2 la siguiente habitación debe estar colocada a la izquierda, si el número es 3 o 4 la siguiente habitación debe estar colocada a la derecha y si el número es un 5 la siguiente habitación debe estar abajo. Además el sistema se aseguraba de que siempre en los extremos de la matriz debe salir un 5. Por último la habitación colocada al principio es de tipo 1, salvo en el caso de que el número que salga sea 5. En este caso la habitación inicial debe ser del tipo 2 y la habitación siguiente del tipo 3 o 2(con una salida superior). Tras esto y al asegurar que siempre hayan salidas a la izquierda y la derecha se puede repetir el algoritmo una y otra vez para generar el camino solución. Este algoritmo paraba y colocaba una habitación solución siempre que llegara a la fila de abajo y saliera el número 5 (indicando que el siguiente camino es abajo). El resto de habitaciones que no había sido completada se rellenaba con habitaciones del tipo 0. Si varias de estas habitaciones eran colocadas de manera vertical, era sustituidas por habitaciones especiales predefinidas de tipo 7, 8, y 9 para crear un “foso” lleno de enemigos y colocando un tesoro en la parte inferior para llamar la atención del jugador. Además en algunas habitación del tipo 0 se añaden un amuleto o algún personaje en apuros para añadir más objetivos al juego y que la puntuación de cada nivel varía en función de los objetivos conseguidos. Finalmente para dar aún más variedad al juego para la selección de el “layout” de cada habitación había una serie de plantillas determinadas. Estos los layout se dividía mediante bloques y espacios en una matriz de 10x10. En el centro de cada habitación se coloca un obstáculo aleatorio. Además cada uno de los

“sprite”que componen la matriz son elegidos de manera aleatoria [5].

En conclusión es importante estudiar el sistema que utiliza Spelunky ya que a pesar de la sencillez de su algoritmo ha sido uno de los juegos más populares con generación aleatoria y todavía a día de hoy sigue siendo un juego de gran éxito.

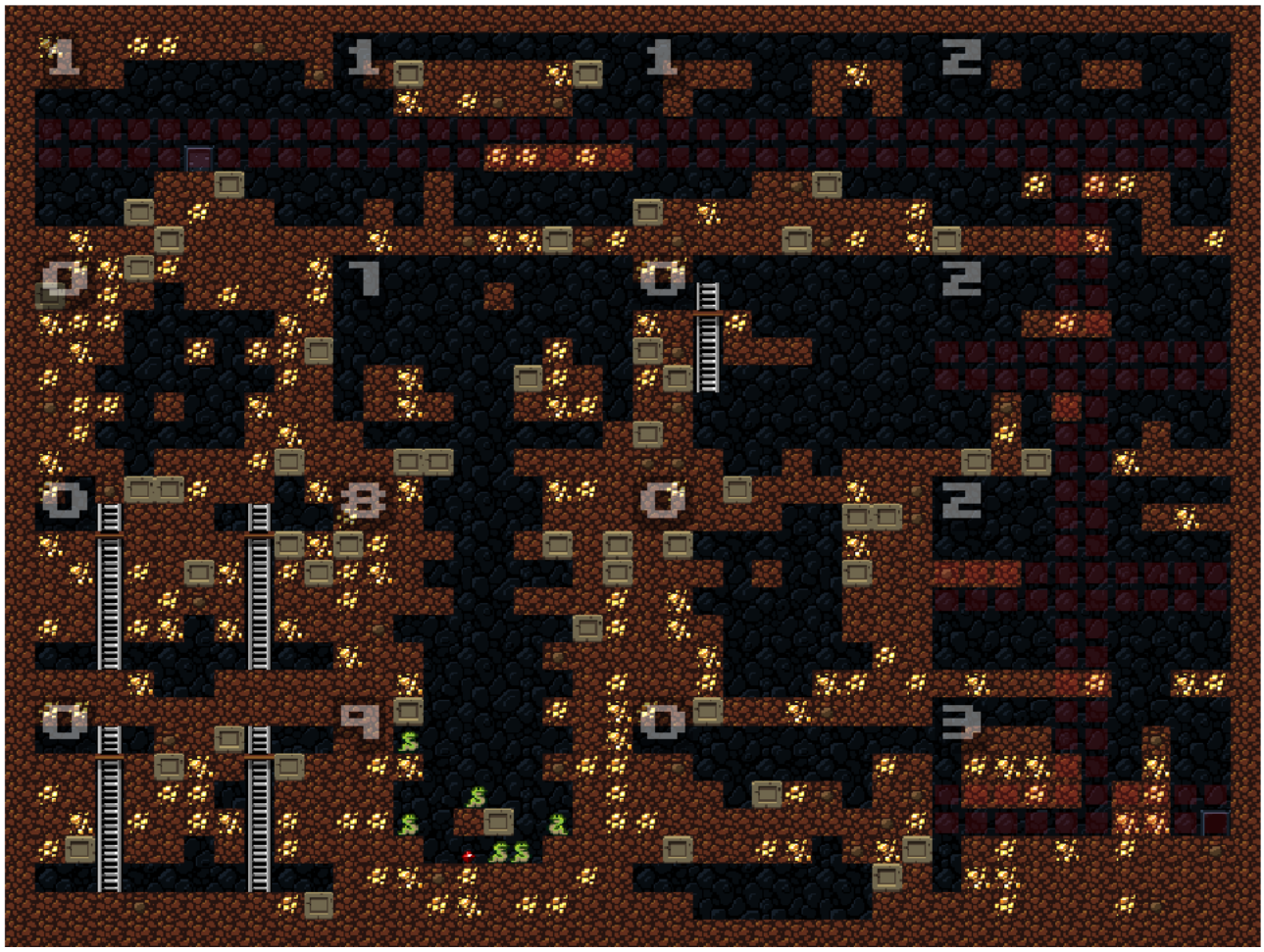

Figura 2.1: Spelunky Generator Lessons.

2.1.3 The binding of Isaac

The binding of Isaac es uno de los juegos con generación procedural más importantes y exitoso de la actualidad. El juego es un “Roguelike” donde el jugador encarna a Isaac que usando sus lágrimas como armas debe avanzar por una mazmorra y enfrentarse a sus miedos con el objetivo de llegar al final [6].

El objetivo que se persigue mediante la generación procedural de los niveles es buscar el interés del jugador por explorar. Para ello cada nivel dispone de 4

habitaciones fundamentales: la habitación del tesoro, la tienda, la habitación secreta y una habitación de jefe. Cada una de ellas ofrecen distintas recompensas. Estás 4 habitaciones pueden contener en su interior distintos objetos, enemigos y sorpresas que se añaden de forma aleatoria. Además estas 4 habitaciones son los cimientos de cada nivel.

El resto de habitaciones son seleccionadas de manera aleatoria de una serie de “layouts” predefinidos (más de 1000). Esta selección aleatoria de habitaciones se hace de manera parametrizada según la dificultad para que haya un progreso en el avance del jugador en cada nivel. Además hay una serie de habitaciones bonus que añaden premios extras o nuevos riesgos que debe afrontar el jugador.

Otro factor importante en el juego es la generación de enemigos. Estos se seleccionan de manera aleatoria en cada habitación del conjunto de niveles. Hay variantes de enemigos llamados Campeones con una pequeña probabilidad de aparecer. Estas variantes tienen un color elegido también de manera aleatoria y con habilidades extra a su versión normal.

Todos este sistema se complica con la adición de objetos, ya que algunos objetos pueden alterar el valor de suerte del jugador haciendo que en cada nivel aparezcan menos enemigos y más recompensas alterando su generación.

Figura 2.2: The Binding of Isaac
Gameplay explained.

Por último los “jefes” como todo lo demás, son también generados proceduralmente. Al igual que los enemigos normales hay distintas versiones de cada jefe y una serie de ayudantes con los que combatir. Hay una serie de jefes especiales que siempre aparecen en niveles concretos del juego. A medida que se aumenta de nivel la dificultad de los jefes aumenta no dependiendo del tipo de “boss” sino aumentando su nivel de vida con respecto al anterior [7].

Todos estos elementos unidos crean un experiencia única cada vez que se comienza el juego. Esto hace que el juego sea “fresco” y poco repetitivo convirtiéndolo en uno de los juegos más populares de la actualidad y haciendo que sea uno de los más grandes referentes en el mundo de la generación procedural de los videojuegos.

2.1.4 No Man's Sky

No Man's Sky es un videojuego de mundo abierto creado por la compañía Hello Game lanzado en 2016. Este juego pone al jugador en el papel de un explorador espacial que deberá viajar a lo largo del espacio con el objetivo de

descubrir los secretos del universo. Toda la información que obtenga el jugador a lo largo de su viaje sobre los distintos planetas y especies que habitan en ellos quedarán registrado en una base de datos particular llamada “Atlas”. Además al descubrir nuevos planetas y especies cada jugador podrá darles nombre y esta información se compartirá con el resto de jugadores ya que los creadores siempre han definido su universo como un “universo compartido “ [8].

La principal particularidad de este juego es la magnitud de los número que mueve, ya que sus creadores han dicho que en su universo hay unos 18,446,744,073,709,551,616 (más de 18 trillones) de planetas [9]. Esto es debido a su método de generación. Para ello el juego dispone de un algoritmo de generación que crea el universo. A su vez, dentro de cada planeta que componen el universo, tenemos características generadas aleatoriamente como geología, flora y fauna. Lo más importante de este algoritmo generador es que cada una de las partes que componen el juego solo son generadas en el momento que se crean. Esto ocurre cuando un jugador descubre un planeta, animal o planta. En ese momento este se genera y su información se guarda en una semilla. En el momento que el jugador se aleja o apaga el juego estos mundos y especies desaparecen y vuelven a estar a la espera de que el jugador vuelva o aparezca otro jugador que visite este mundo (caso muy poco probable debido al tamaño del mundo creado). Por último para dar un interés adicional a los mundos y especies generadas, los creadores tienen acceso a una serie de herramientas de modificación que permiten editar algunas especies y planetas y poder parametrizar algunos puntos de su universo creado.

Desafortunadamente y a pesar de ser un proyecto muy ambicioso, el resultado de su lanzamiento no dejó contento a los jugadores. En su momento, cuando se presentó el juego lo hizo con un trailer donde se veía un mundo lleno de vida y criaturas increíbles. Esto hizo crecer la expectación de los jugadores ya que estas imágenes deberían ser un reflejo de cómo iba a ser el juego. La problemática llegó cuando debido a el método de generación la mayoría de los mundos y animales que los habitaban no eran tan asombrosos como los del trailer haciendo que aumentara la decepción en los compradores del juego. Esta situación se agravó ya que en ese mismo trailer aparecían varias naves en una misma zona lo que hizo pensar a los jugadores, que el juego dispondría de un modo multijugador. Este hecho fue negado por los creadores del juego poco días antes de su lanzamiento, lo que aumentó el descontento general de los jugadores. La excusa que dieron los desarrolladores fue que “en un universo tan grande, las posibilidades de que dos jugadores pudieran reunirse en un mismo punto eran muy pequeñas”. El problema fue que el mismo día de lanzamiento dos jugadores(que además se encontraba transmitiendo el juego) estuvieron en el mismo planeta al mismo tiempo, pero ninguno de los dos podía verse.

Figura 2.3: No Man's Sky - Expectation Vs. Reality.

Conscientes de la problemática y polémica que generó el juego, el equipo de Hello Games han ido añadiendo de manera gratuita contenido al juego para intentar compensar a los jugadores que lo habían comprado. Además en una de estas actualizaciones (la denominada Next) se añadió el modo multijugador donde varios jugadores pueden disfrutar de la experiencia de No Man's Sky juntos.

A modo de conclusión, y a pesar de todos los intentos de mejoras y que el juego ahora tiene muchos más contenido y ha mejorado como experiencia con respecto a su lanzamiento, se puede ver en él un claro ejemplo de la problemática de la generación procedural de lo juegos, ya que muestra como la aleatoriedad puede generar mundos sin ningún tipo de interés. Esto como consecuencia desemboca en que los jugadores no disfruten de la experiencia del juego y que no continúen jugando.

2.2 Aportaciones a los juegos de la actualidad

En la actualidad a pesar de las mejoras y refinamientos de las técnicas de generación procedural no es muy usada en la creación de videojuegos (pero si en la generación de escenarios dentro de los mismos) ya que tiene el principal inconveniente de que es difícil captar la atención de los jugadores y mantenerla en cada partida y conseguir que el jugador siga teniendo interés en continuar jugando. The binding of Isaac consigue esto parametrizando gran parte de la generación y para que siempre haya recompensas y que el jugador

continúe jugando. Además incentiva al jugador a continuar jugando haciendo que después de ganar una partida se desbloquean nuevos niveles y más objetos para mejorar la siguiente partida.

A pesar de esto la mayoría de los sistemas que aparecen en los videojuegos consisten en la exploración donde la dificultad se encuentra en evitar trampas y vencer enemigos generados de manera aleatoria. El objetivo de este proyecto es buscar la creación de un juego donde la dificultad se encuentre en la distribución de puzles por un escenario y que siempre exista una solución para el mismo.

Además para conseguir este objetivo se plantea crear un juego mediante la utilización de un sistema de resolución de problemas denominado “Answer Set programming”(ASP), que a pesar de ser utilizado en muchos campos no es muy popular en el mundo de los videojuegos aunque sí existen estudios previos de aplicación, como por ejemplo para el diseño de niveles [10][11]. Mediante estas herramientas se plantea integrar un sistema de generación de puzles resolubles con un motor gráfico. Este juego pretende aportar una mejora a la generación aleatoria manteniendo la atención de los jugadores añadiendo niveles distintos en cada ejecución del juego.

Otro objetivo a lograr es realizar un acercamiento de este tipo de tecnologías(ASP) al mundo de los videojuegos y aumentar su viabilidad y visibilidad en este campo de la informática. Mediante la integración del lenguaje clingo con c++ se generará una librería dinámica(dll) que permitirá al motor gráfico obtener soluciones planteadas mediante código clingo. Posteriormente mediante la utilización de una script de construcción se generará el nivel con puzles. Esto hace que simplemente teniendo que cambiar pocos elementos se pueda reutilizar el código fácilmente y ayudar a popularizar estas técnicas en futuros juegos.

La generación procedural plantea problemas cuando la creación aleatoria del mundo debe estar acotada a restricciones ya que dentro de este nivel deben generarse una serie de puzles y que estos sean resolubles. Por ello la generación de puzles no debe afectar a la estructura del escenario principal del juego. Se plantea la separación de la creación del escenario de la del puzle. Posteriormente se deberán juntar ambos elementos en entornos controlados seleccionados dentro del propio mundo.

Capítulo 3
Herramientas y metodología

3.1 Unity

3.1.1 Descripción de unity

Unity es un motor de videojuego lanzado en 2005 por la compañía Unity Technologies y cuyo principal característica es que es multiplataforma [12]. Esto significa que el código realizado por los desarrolladores no varía en función de la plataforma donde se realiza. Además es muy popular entre los programadores debido a su sencillez de uso ya que dispone de una versión gratuita que permite a los nuevos desarrolladores aprender a utilizar su funcionalidades y sacar juegos sin necesidad de tener un alto presupuesto. Este motor está compuesto por distintas partes:

- Un editor (“Scena”) de escena que permite al jugador arrastrar los objetos y ser colocados en el escenario de un nivel del juego según la posición que este crea conveniente. Otra utilidad del editor es que permite alterar los objetos tanto en posición como en tamaño. Además permite al jugador visualizar el juego desde distintos puntos de vista.
- Una pestaña “Game” que añade una cámara de juego que permite representar como quedaría esa escena dentro del juego. Además una vez se ejecute el juego esta pestaña permite probar una “ demo” de ese nivel.
- La pestaña consola permite ver al programador los errores y warning que tiene cada uno de los códigos que componen el juego.
- En “Hierarchy” el programador puede ver los distintos objetos de la escena y de qué están compuestos cada uno de estos objetos.
- El inspector permite parametrizar algunas cosas como la posición de cada uno de los componentes que aparecen en “Hierarchy”. Además desde aquí es donde se añaden las script a los objetos, se controlan las texturas, sistema de colisiones y permite darle prioridad si existen conflictos entre estos elementos.
- Por último en “project” se puede explorar las distintas carpetas del proyecto para añadir script, assets, texturas y sonidos y luego poder añadirlo al editor del juego.

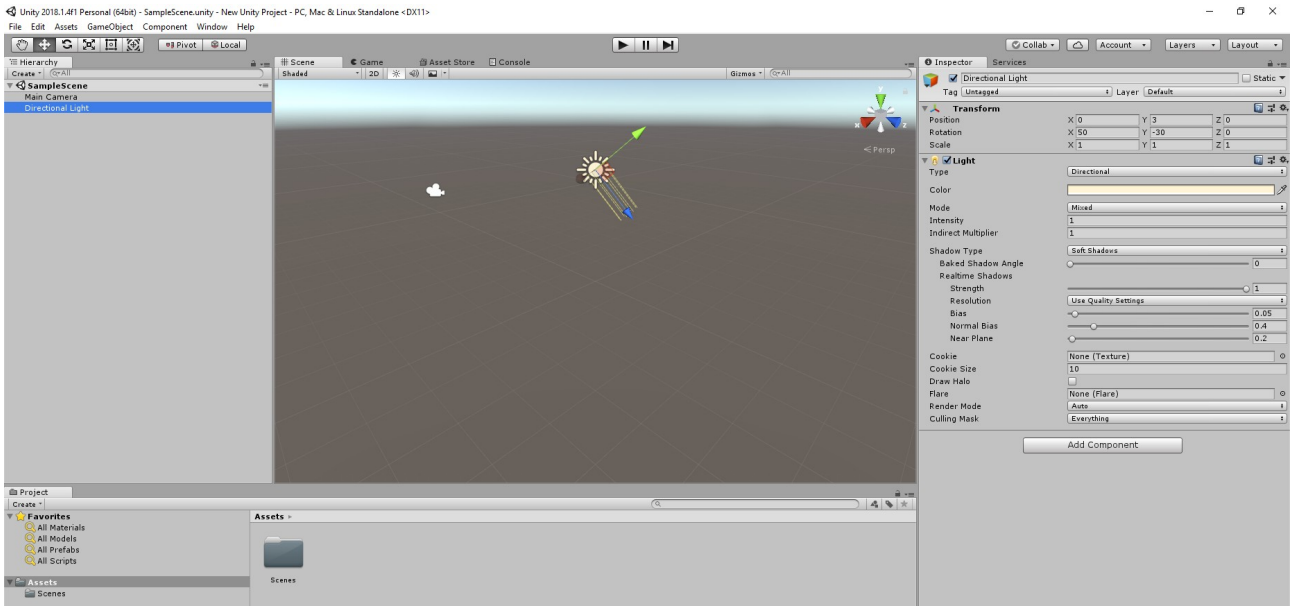

Figura 3.1: Editor de Unity.

Para la realización de scripts en Unity se utilizan dos lenguajes de programación el C# y el Javascript. Además funciona con la programación orientada a eventos que es un paradigma de la programación que consiste en que la estructura y la ejecución va determinada por los sucesos que ocurren y que son declarados por el programador. Este tipo de programación es fundamental para el desarrollo de interfaces y juegos ya que permiten al creador del programa definir cada una de las acciones que realizará el usuario final y tener una respuesta a cada una de estas acciones. Para ello es necesario definir un administrador de eventos que permita controlar cada una de las acciones que ocurran [13].

3.1.2 Administrador de eventos

Para la realización de este trabajo dispondremos de un GameObject vacío al que se le añadirá la script principal del juego. Esta script generará en el momento de inicialización del juego cada uno de los elementos que lo componen. Adicionalmente está script controlará parte de los eventos que ocurran en el juego así como las acciones al pulsar las teclas de acción.

En cuanto al control de colisiones del juego se realizará mediante una malla de colisiones (“collider”) en los objetos que componen el juego. Este collider está completamente vacío y se puede atravesar (los objetos dispondrán de otra malla adicional para disimular la física real y que no pueda atravesarse) para activar el evento. Dependiendo del evento y del estado del jugador ocurrirá la acción necesaria. Por ejemplo si el jugador dispone de la llave para abrir una puerta y se encuentra cerca de ella se activa la acción de abrir la puerta.

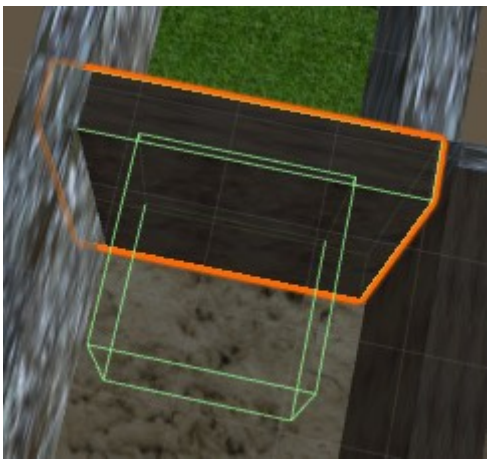

Figura 3.2: Ejemplo de collider en Unity.

Para que las script pueda localizar un objeto en concreto dentro de la escena se utilizan “tags”. Estas etiquetas permiten definir objetos (como la etiqueta player para el jugador) y así controlar cuáles son los componentes que intervienen en cada uno de los eventos del juego.

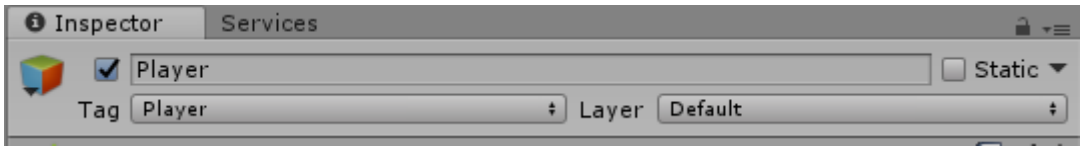

Figura 3.3: Ejemplo de Tag en Unity.

Otra funcionalidad que realiza el controlador principal de este juego es que tras llegar a la salida del nivel activará una función que permite recargar el nivel cuando se llegue a la solución. Al generar proceduralmente el juego cada vez que se ejecuta esta función permite cargar un nivel nuevo completamente distinto al anterior.

3.1.3 Interfaz de usuario

Para la realización de la interfaz que verá el usuario, Unity permite crear un tipo de objeto especial llamado “Canvas”. La principal característica de este objeto es que es en 2D y se coloca automáticamente en relación a la propia cámara del juego de forma que todos los elemento (sprite y textos) se verán como interfaz del juego.

Este objeto es muy útil para la creación de interfaz del usuario y para añadir elemento importantes en el juegos como menús e inventarios.

Para la realización de esta interfaz se colocarán todos los elementos que la forman en el “canvas” y en función de los eventos que ocurran en el juego se irán activando y desactivando objetos haciendo que aparezcan o desaparezcan en la pantalla del usuario. De esta manera el juego puede hacer diversas

acciones como añadir llaves al inventario o actualizarse el contador de pasos cada vez que avanza el jugador.

3.2 Clingo

3.2.1 Descripción de Clingo

Clingo es un sistema para resolver problemas con restricciones. Un problema con restricciones es aquel cuyo planteamiento está formado por un conjunto finito de normas que se han de cumplir para poder considerarse resuelto [14] [15]. Un ejemplo de este tipo de planteamientos, es el problema de las n-reinas. Este consiste en colocar un número determinado de reinas en un tablero de ajedrez de forma que se cumpla la restricción de que ninguna puede comerse a las otras. Para entender el funcionamiento de Clingo el primer paso es conocer qué es el ASP. ASP es un lenguaje de programación declarativo. Estos son aquellos lenguajes donde el programador describe el problema en sí(a diferencia de los lenguajes procedurales donde se describe la resolución del problema), existiendo un método general para encontrar soluciones. En concreto el ASP de la familia de los lenguajes de programación lógica. Esto quiere decir que el programa se divide en una series de reglas del tipo “si-”, “entonces” donde el “sí-” es el conjunto de condiciones o antecedentes(también denominado cuerpo) y el “entonces-” es la consecuencia (también denominado cabeceras). La ejecución del ASP en el sistema Clingo combina dos fases:

- Una fase de “grounding” donde el subsistema denominado “Gringo” crea una versión equivalente del programa “sin variables”, es decir todas las variables que aparecen en la regla son sustituidas por objetos concretos.
- Una fase de obtención de soluciones donde se utiliza Clasp, que es un sistema de búsqueda de soluciones viables para problemas con restricciones una vez están expresados en forma proposicional.

Para la definición del problema en ASP, se utiliza un conjunto de átomos denominado modelo estable. Un átomo es un predicado junto a sus argumentos por ejemplo:

- door_on(habitación1, puerta1): es un átomo basado en el predicado door_on/2 con los argumentos habitación1 y puerta1.

Una regla de inferencias es evaluada con un conjunto de átomos y el resultado final que la regla puede ser satisfecha o no. Se considera que una regla de inferencia es satisfecha cuando se cumple alguna de los siguientes requisitos:

- Alguna de las condiciones del cuerpo de la regla no se cumple.
- Si se cumple todas las condiciones del cuerpo de la regla.

El cuerpo de una regla se componen de literales. Los literales son átomos tanto positivos como negativos(los negativos son átomos precedidos de una partícula not). Los conjuntos de átomos que devuelve ASP como soluciones deben ser

modelos estables. Un modelo estable es aquel modelo en el que todos los átomo que se incluyan en la solución deben tener alguna razón(aparece en la cabecera alguna instancia donde el cuerpo es satisfecho). Para entender mejor este concepto veremos el siguiente ejemplo:

- a:-not b.
- b:-not a.

Estas reglas están basadas en dos átomos a y b. Los modelos para este programa serían los siguientes:

- M1{a}: es un modelo porque ambas reglas se cumplen. La primera porque con el cuerpo not b (se interpreta como que el átomo b no puede aparecer en el modelo) y la cabecera. La segunda regla al no cumplirse el cuerpo la regla se satisface.
- M2{b}: ambas reglas se cumplen por razones similares a la anterior.
- M{a,b}: También es un modelo ya que ambas reglas se cumplen. Esto es debido a que en ninguna de las reglas puede cumplirse el cuerpo.

Viendo los modelos obtenidos comprobamos que tanto en M1 como en M2 se incluyen en el modelo porque los cuerpos de los modelos se cumplen, en el caso de M3 no existe ninguna razón positiva para incluir ambos métodos. Debido a esto ASP usa el concepto de modelo estable. Por lo tanto M1 y M2 son modelos estables, pero M3 no lo es.

Algunos de los tipos de reglas particulares que podemos encontrar en ASP son:

- Hechos(“facts”): Reglas que solo tienen cabecera. Al no tener cuerpo nos indica que este siempre se va a cumplir(el cuerpo). Por lo tanto todos los átomos que deben estar invariables en todas las soluciones propuestas debe introducirse mediantes reglas del tipo “hechos”.
- Restricciones de integridad(“integrity constraints”): Son reglas que no tienen cabecera. Al no haber cabecera, no hay forma de cumplirla en un modelo estable, por lo tanto si se cumplen la regla no sería satisfechas ya que no hay posibilidad de cumplir la cabecera. Estas reglas expresan situaciones que bajo ningún concepto pueden darse en cualquier solución. Un ejemplo de restricción es :-a, b. Mediante esta restricción definimos que en ninguna solución del programa se puede incluir simultáneamente el átomo a y el átomo b.

También en las reglas podemos encontrar agregaciones. Las agregaciones sirven para evaluar un conjunto de átomos. Estas pueden aparecer tanto en la cabecera como en el cuerpo de la regla. Si la agregación se encuentra en el cuerpo se admiten tanto literales tanto positivos como negativos mientras que si aparece en las cabecera solo admite los positivos.Una forma en la que pueden encontrarse las agregaciones en ASP es: L<={l1;l2;..;ln}<=U. L1;l2;..;ln son literales. L y U son valores enteros [16].

3.2.2 Integración con C++

Para poder llevar a cabo una integración de nuestro código en Unity es necesario primeramente convertir el código de Clingo en una dll. La misión de esta dll es proveer al sistema de scripts de Unity de una función externa que genere el puzzle requerido. Esta dll será programada en C++ ya que Clingo dispone de una librería y un API para este lenguaje. Para ello Clingo dispone de una librería para C++. El API de Clingo permite trabajar con programas ASP de dos formas. La primera de ellas consiste en construir lo que sería el equivalente a la salida de un parser de ASP. La segunda permite introducir el programa ASP como una cadena de texto y hacer funcionar el Parser. En este proyecto, la parte del programa dependiente de parámetros(los puzles están parametrizados) se crea mediante el primer método. La parte del programa que no depende de los parámetros se pasa como una cadena de texto.

Además esta librería permite construir un modelo que recorre el conjunto de símbolos del problema de Clingo, el conjunto de soluciones y seleccionar los predicado que interesan. La selección de estos predicados se hará mediante el nombre que se les ha dado y el número de argumentos que disponen.

Finalmente se creará la librería dinámica a partir de la clase que contiene el resolvedor mediante la función de C++ “c __declspec(dllexport)”.

3.2.3 Integración con Unity

Para la integración con Unity es necesario que la dll de Clingo este colocada dentro de la carpeta editor del Unity(esta carpeta contiene todas las dll necesarias para el funcionamiento del editor de unity). Con esto se consigue que Unity pueda acceder a la funciones de Clingo necesarias para ejecutar la dll que ya se ha creado con anterioridad. La script generada a partir de la función “c __declspec(dllexport)” debe ser colocada en cualquier lugar de los asset del proyecto de Unity. Dentro de la script de C# se puede llamar a este script mediante la función [DllImport("Ruta interna de Unity")]. Por último se debe declarar la función de la dll como una función externa al código de Unity.

En este caso para poder trabajar con las mismas variables tanto en la dll como en Unity las variables se pasarán como referencia a la función externa y así se consigue que todos los cambios hechos en los parámetros de la función se apliquen en la variables de Unity.

Capítulo 4
Resultados obtenidos

4.1 Generador de laberintos

El objetivo principal de este código es la creación del escenario del juego. Este escenario debía de ser un laberinto cuyos caminos pueden ser recorridos por el jugador. Otra característica importante de este laberinto es que en cada ejecución debe generarse un laberinto totalmente distinto al anterior.

Para la ejecución de este código es necesario recibir como parámetros del código el número de filas y columnas del laberinto (ya que este será creado a partir de una matriz). Adicionalmente se ha sumado la posibilidad de añadir una “prefabs”(Las “prefabs” son modelo de objetos que del muro que dispondrá los muros del laberinto (permitiendo así modificar tamaño y color del mismo). Debido a esto y cómo se trabajará a partir de este muro es necesario introducir el tamaño de su anchura. La script con el código del juego debe ser añadida a un objeto vacío. Una particularidad importante de Unity es que permite controlar las variables públicas de las script mediante el editor lo que hace posible que el diseñador pueda modificar estas variables de manera sencilla desde el editor.

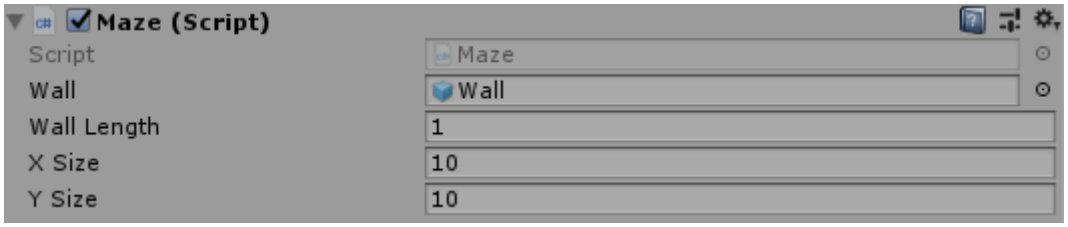

Figura 4.1: Parámetros del laberinto.

Al crear una script en Unity se generarán automáticamente dos funciones. Una primera función “start” que se ejecutará al principio del juego y la segunda función es la “update” que se ejecuta en cada “frame” del juego. En este caso como solo es importante la generación del escenario solo se trabajará en la función “start” por lo tanto la funciones que se han realizado serán llamadas desde esta función.

El primer paso de este código de generación de laberintos es la creación de una matriz con las habitaciones generadas colocando cuatro muros juntos. Para ello lo primero será decidir una posición inicial. A partir de esta y dos bucles “for” anidados (uno con el número de filas y otro con el número de columnas con lo que se generará cada uno de los muros horizontales y verticales del laberinto. Para la generación de estos muros utilizaremos dos bucles “for” anidados para los muros del eje x (horizontal) y dos bucles “for” anidados para el eje y (vertical). Mediante la función “Instantiate” iremos generando los muros y se guardarán todos en un objeto padre. El resultado de esta generación de matriz es el siguiente [17]:

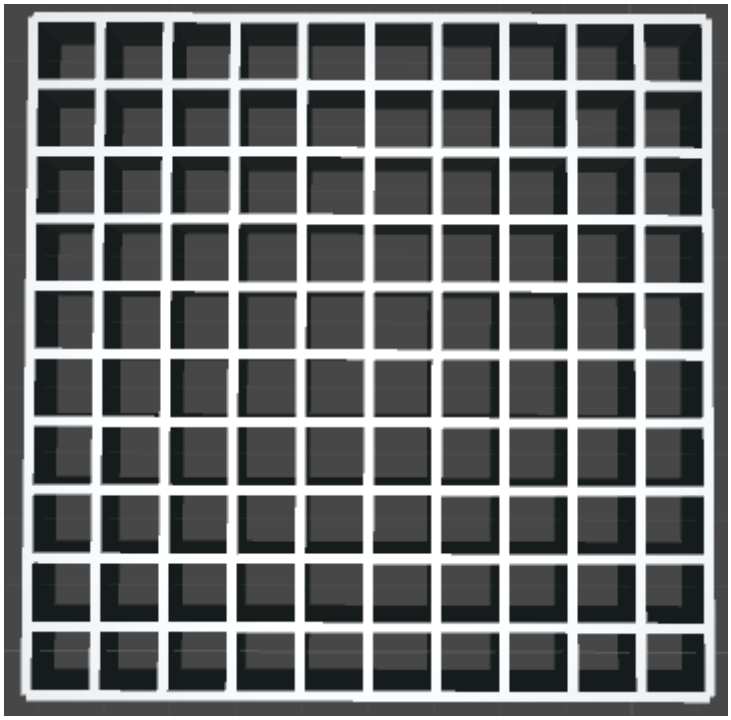

Figura 4.2: Matriz inicial del laberinto

Cada uno de los muros generados son añadidos al juego mediante la función de Unity “Instantiate”. Esta función permite generar objetos a partir de un objeto, una posición y una rotación. Estos muros serán separados de cuatro en cuatro (cada habitación) en una clase anteriormente creada llamada “cell” (El campo “visited” será usado posteriormente para ver qué células han sido usadas con anterioridad, al principio de la ejecución estará en “false”).

```
public class Cell{
    public bool visited;
    public GameObject north;
    public GameObject east;
    public GameObject west;
    public GameObject south;
}
```

Figura 4.3: Clase Cell.

Todas las celdas serán guardadas es una lista para tener una referencia clara de cada una de la células que componen esta matriz [18].

Ahora falta eliminar los muros que componen esta matriz para generar los caminos. Para ello se utilizará un bucle “while” que se ejecutará hasta que todas las células hayan sido visitadas. La primera celda se elegirá al azar. Una vez elegida esta celda se comprobarán donde están los posible vecinos (no visitados). Se elegirá un vecino al azar [19]. Una vez roto el muro que separa la celda actual y la celda vecina, se marcará la célula vecina como visitada y se convertirá en la próxima celda actual. La antigua celda actual se guardará en una lista y en una variable de “backup” se guardará el índice de la última celda añadida. En caso de que no se encontraran vecinos no visitados se extraerá la última celda guardada y se comprobará sus vecinos. De esta manera el código se asegura de que todas las celdas sean visitadas y el laberinto generado no tenga habitaciones cerradas.

```
void CreateMaze(){
    lastCells = new List<int>();
    lastCells.Clear();
    bool startedBuilding = false;
    while (visitedCells < totalCells) {
        if (startedBuilding) {
            GiveMeNeighbour ();
            if (cells [currentNeighbour].visited == false && cells[currentCell].visited == true) {
                BreakWall ();
                cells [currentNeighbour].visited = true;
                visitedCells++;
                lastCells.Add(currentCell);
                currentCell = currentNeighbour;
                if (lastCells.Count > 0) {
                    backingUp = lastCells.Count - 1;
                }
            }
        }
        else {
            currentCell = Random.Range (0, totalCells);
            cells [currentCell].visited = true;
            visitedCells++;
            startedBuilding = true;
        }

    }
}
```

Figura 4.4: Función crear laberinto.

El resultado final de este código será que cada vez que se ejecute el juego se creará un laberinto parametrizado completamente accesible y distinto en cada ejecución [20].

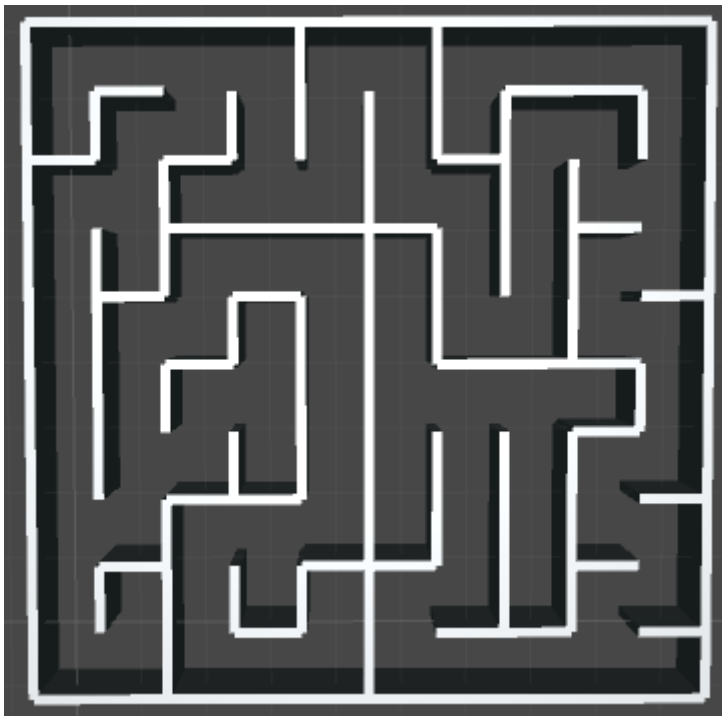

Figura 4.5: Laberinto.

4.2 Mejora del generador de laberintos

Para tener un mejor control de la colocación de puzles en el escenario, se han creado funciones adicionales que permiten a partir de una celda aislar un camino aleatorio del resto del laberinto.

Como entrada se le añadirá a las entradas ya definidas para el laberinto, un nuevo parámetro para controlar el tamaño de este camino nuevo.

El primer paso para conseguir este objetivo es dentro de la lista de celdas resetear el campo visited para así aprovecharlo para este nuevo código. Para ello se recorrerá las listas de celdas y se cambiará este campo de “true” a “false” (ya que tras la creación del laberinto todas las celdas se encuentran visitadas).

```
void resetVisited(){
    for (int i = 0; i < cells.Length; i++) {
        cells [i].visited = false;
    }
}
```

Figura 4.6: Función resetear visitados.

Tras realizar esto se pasará a realizar la función de creación del camino. El primer paso es aislar la celda actual del resto de celdas para ello activaremos todos los muros de la celda mediante la función de Unity “setActive” (true) (Algunos muros de las celdas habían sido desactivados para la creación de los caminos que componen el laberinto). Tras esto se hará una comprobación de las celdas adyacentes. Para facilitar el trabajo con estos vecinos se ha creado una clase “way” para guardar los mismos. Dentro de este tipo se guarda el ID numérico de la celda vecina y el ID del muro que le une con la celda actual. El ID del muro que se ha establecido en este código es el siguiente:

1. Norte
2. Este
3. Oeste
4. Sur

Cada vez que se ejecuta el código de encuentro de vecinos se procederá a guardar todos las variables “way” en una lista. Posteriormente se seleccionará uno de estos posibles vecinos al azar y se deshabilitará el muro que les separa.

```
Neighbourhood ();// Función de obtención de vecinos
int theChosenOne = Random.Range (0, waynei.Count);// Selección aleatoria de unos de los vecinos encontrados
switch (waynei[theChosenOne].wall) {
    case 1: cells[currentway].north.SetActive(false);break;
    case 2: cells[currentway].east.SetActive(false);break;
    case 3: cells[currentway].west.SetActive(false);break;
    case 4: cells[currentway].south.SetActive(false);break;
}
```

Figura 4.7: Selección de muro a romper.

Tras esto se marcará la celda actual como visitada y la celda vecina se convertirá en la nueva celda actual. Por último se activarán todos los muros de la celda antigua salvo el que le separa de la celda anterior y se repite el código desde la selección de vecinos hasta el final tantas veces como tamaño tenga el camino que se quiere aislar.

Como resultado de este código el laberinto dispondrá de un camino aislado sobre el que empezar a trabajar en la adición de puzles al mismo.

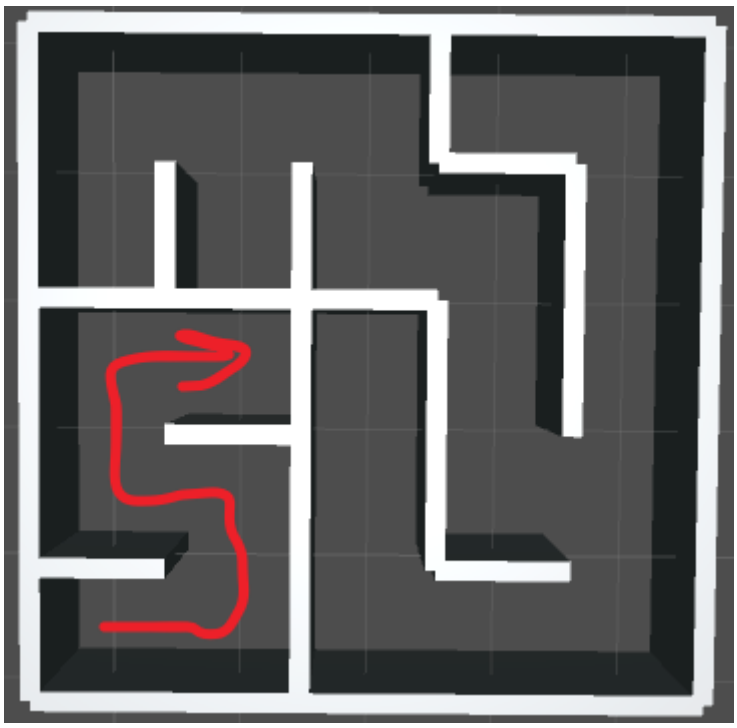

Figura 4.8: Laberinto con camino.

4.3 Definición de un puzle simple mediante Clingo

Todo este código puede ser consultado en el Anexo I.

Para el primer puzle se ha decidido un sencillo rompecabezas de llaves/puertas donde en un pasillo dividido en habitaciones se colocarán una serie de puertas y una serie de llaves. Puesto que cada puerta puede abrirse con una sola de las llaves, la condición para que el puzle sea resoluble es que con anterioridad a que el jugador encuentre una puerta cerrada, debe haber podido visitar la habitación donde se encuentra la llave que la abre.

Este código recibirá el número de habitaciones y el número de llaves/puertas como parámetros. También se le pasará el número de movimientos máximos permitidos para acotar el número de soluciones obtenidas.

Para comenzar se definirán los hechos que van a componer el problema. En este caso serán las habitaciones, las llaves y las puertas.

```
room(1..r).
key(r+1..k+r).
door(r+k+1..r+k+k).
```

Figura 4.9: Definición de hechos de nuestro problema.

Los argumentos que componen los hechos pueden ser tanto números como constantes no númericas. En este caso se ha decidido utilizar números ya que nos permiten tener una correlación directa entre variables de una manera sencilla, por ejemplo podemos definir una sucesión de habitaciones room(1..3). En nuestro problema las variables numéricas quedan de tal manera que el 1 será la primera habitación y r(número de habitaciones) será última. Las llaves irán desde r+1 hasta r+k(número de habitaciones más número de llaves/puertas). Por último las puertas quedarán acotadas entre r+k+1 hasta r+k+k.

También en la parte de definiciones se fijan las acciones que se realizarán para resolver el problema. En este caso se definirán las siguiente acciones:

1. lt(left): movimiento a la izquierda.
2. rt(right): movimiento a la derecha.
3. tk(take): coger una llave.
4. re(release): soltar una llave.
5. uop(unlock and open): abrir una puerta.

Dentro de la parte de definiciones también se determinará el número de transiciones máximo de este problema, el estado inicial (en la habitación inicial, sin ninguna llave, y en la primera transición), y los estados de las puertas (cerradas o abiertas).

En la parte generativa del problema se concretará el estado inicial de las puertas y de las llaves y las acciones que dispondrá el problema.

{key_on(R,K,1) : room(R)}=1:- key(K). En esta regla podemos comprobar que no se sigue la forma habitual para las agregaciones porque se hacen alguna simplificaciones. El símbolo “=” indica que tanto L como U tienen el mismo valor. Por lo tanto esta regla es equivalente a la regla:

- 1<={key_on(R,K,1) : room(R)}<=1:- key(K).

La siguiente parte de la regla key_on(R,K,1):room(R) es lo que se denomina literal condicional. Cuando un literal condicional está dentro de una agregación se examina el átomo que expresa la condición en este caso Room(R) y se determina cada instancia del mismo que aparece en la solución. Por lo tanto si en este ejemplo tenemos que existen los hechos room(1),room(2) y room(3) podemos sustituir el literal condicional por:

- 1<={key_on(1,K,1), key_on(2,K,1), key_on(3,K,1)}<=1:- key(K).

Ahora suponiendo que tenemos los hechos key(1), key(2) y key(3) el “grounding” de la regla sería el siguiente:

- 1<={key_on(1,1,1), key_on(2,1,1), key_on(3,1,1)}<=1:- key(1).
- 1<={key_on(1,2,1), key_on(2,2,1), key_on(3,2,1)}<=1:- key(2).
- 1<={key_on(1,3,1), key_on(2,3,1), key_on(3,3,1)}<=1:- key(3).

Para cada uno de los átomo de esta agregación se establece que solo uno debe ser cierto porque el peso total de la regla debe ser 1. Si interpretamos que el predicado key_on(R,K,S) como la ubicación de la llave K, en la habitación R en el momento S, la primera regla establece que la llave 1:

1. Debe estar en la habitación 1, la 2 o la 3.
2. Que no debe estar en más de dos habitaciones a la vez porque el peso de la regla sería mayor que 1.

Al final las reglas establecen que cada llave tiene que estar en una sola habitación. Además en este caso vemos que en ningún momento se impide que exista más de una llave por habitación.

En el siguiente regla: {door_on(D,R):room(R)}=1:- door(D), vemos un predicado similar al anterior, pero haciendo referencia a la posición de las puertas. En este caso la colocación de una puerta será siempre la misma a lo largo del tiempo por lo tanto el predicado door_on no depende del número de transición en el que nos encontremos. Por lo tanto al expandir esta regla(donde r = 2 y k = 2) quedará de la siguiente manera:

- 1<={door_on(1,1), door_on(1,2)}<=1:- door(1).
- 1<={door_on(2,1), door_on(2,2)}<=1:- door(1).

Por tanto al igual que en el predicado anterior se establece que una puerta tiene que estar en una sola habitación. Sin embargo ahora tenemos el problema de que pueden aparecer más de dos puertas en una misma habitación. Para solventar este problema añadiremos una restricción adicional:

- :- door_on(D1,R),door_on(D2,R), D1!=D2. No debe cumplirse que exista una puerta en una habitación R y exista otra puerta en la misma habitación y que ambas sean distintas. Así se consigue asegurar que no existan dos puertas en la misma habitación.

La siguiente regla importante se refiere a la definición de las acciones. La generación de los predicados de acción se realiza con la regla:

- {actua(A,K,T) : accion(A), takeable(K) }<=1 :- T=1..M-1,moves(M).

Para explicar esta regla supongamos los siguiente hechos:

1. moves(3).
2. accion(lt).
3. accion(rt).
4. takeable(0).
5. Takeable(1).

Instanciando la regla nos quedaría de la siguiente manera:

- {actua(A,K,T) : accion(A), takeable(K) }<=1 :- T=1..2,moves(3).

Mediante el cuerpo de la regla T=1..2, es la comprobación de la igualdad en un rango(1..2), por tanto podemos si sustituimos lo datos la regla quedaría tal que así:

- {actua(A,K,1) : accion(A), takeable(K) }<=1 :- moves(3).
- {actua(A,K,2) : accion(A), takeable(K) }<=1 :- moves(3).

En la agregación tenemos un literal condicional:

- actua(A,K,1): accion(A),takeable(K).

Al realizar la sustitución de la regla obtenemos:

- 0<={actua(lft,0,1), actua(lft,1,1), actua(rgt,0,1), actua(rgt,1,1)}<=1.
- 0<={actua(lft,0,2), actua(lft,1,2), actua(rgt,0,2), actua(rgt,1,2)}<=1.

Mediante es regla expresamos que en la solución para cada valor de T a lo sumo puede producirse una acción. Además debemos asegurarnos haya una sucesión de acciones ya que para pasar de una transición a otra debe existir un cambio de estado. Para resolver este problema podemos utilizar la siguiente restricción:

- :- actua(_,_,T), not actua(_,_,T-1),T>1. Esta restricción nos asegura que puede existir una acción sin que antes hubiera ocurrido otra salvo que estemos en la primera transición.

Por último otro predicado importante que debemos tener en cuenta en las acciones es que no puede existir un movimiento a la derecha si hay una puerta cerrada:

- estado(R,K,T) :- not doorontherightclosed(R-1,T-1),estado(R-1,K,T-1), R<=r, actua(rt,0,T-1),T=1..M,moves(M). doorontherigthclosed es un auxiliar que nos permite saber si la puerta a la derecha está cerrada. Por lo tanto para que se cumpla esta regla y exista un modelo estable es necesario que se cumpla la condición de que no puede haber una puerta cerrada y un movimiento a la derecha en la misma transición.

Después de realizar la generación del problema se añaden las restricciones con el objetivo de establecer la reglas con las que está definido el mismo y acotar el conjunto de soluciones posibles. Como ya hemos visto una de las restricciones más importantes del problema es que las llaves deben colocarse antes de las puertas que abren para así conseguir que todas las soluciones obtenidas definan un puzle con solución. Las restricción obtenida es la siguiente:

- :- key_on(R1,K,1), door_on(D,R2), (D-k)=K, R1>R2. Está restricción indica que no debe cumplirse que exista una llave k en una habitación R1 y una puerta D en una habitación R2 siendo la habitación R1 posterior a la segunda y que además esa llave abra esa puerta. Además utilizando la fórmula D-k=K donde k es el número de llaves y que proviene de la identificación con números de las llaves y las puertas.

Además se añadirán una serie de restricciones sobre las acciones, transiciones, movimientos, coger/soltar llaves, y abrir/cerrar puertas para definir cómo van a funcionar las soluciones obtenidas. Algunos ejemplos de estas restricciones:

- :- actua(re,0,T). No debe cumplirse que exista una acción del tipo realese sin que se tenga objetos.
- :- not estado(r,_,_). Con esta restricción se asegura que siempre debe existir un estado final en el que se llegue a la habitación r.

Tras esta explicación de algunas de las reglas más importantes, es importante señalar que un problema de este tipo en ASP puede ser planteado de dos maneras distintas:

1. La realización de la definición de un problema resoluble. En nuestro caso sería crear una serie de habitaciones con puerta y llaves y que este puzle sea completamente resoluble.
2. Lo anterior añadiendo además las soluciones a nuestro problema.

En este proyecto se ha usado el segundo método ya que a pesar de ser el más difícil de codificar, tiene la ventaja de que posteriormente se puede hacer una selección de resultados y así poder acotar las soluciones obtenidas para adaptarlas mejor a nuestro objetivos.

En concreto en este proyecto las soluciones obtenidas son el conjunto de estado y acciones que ocurren en cada una de las transiciones del problema. Para ello está definido que es necesario que haya una acción para poder pasar de un estado a otro y además en este nuevo estado habrá una parte que varía y una que se mantiene igual. Un ejemplo de esto es que si tenemos el estado incial estado(1,0,1) y un predicado door_on(4,2) realizamos un actua(rt,0,1), obtendremos en la siguiente transición tendremos estado(2,0,2), pero en cambio el predicado door_on(4,2) no se ve afectado por la acción.

Figura 4.10: Cambiar de Estados

Como resultado final Clingo devuelve la definición del problema mediante una serie de predicados. Se puede controlar qué predicados se quieren mostrar mediante la utilización del “show” que recibe como parámetros el predicado a mostrar y el número de argumentos del mismo [16].

```
clingo version 5.0.0
Solving...
Answer: 1
estado(1,0,1) door_on(7,2) door_on(6,3) estado(1,5,2)
estado(1,0,3) estado(2,0,4) estado(2,4,5) estado(2,0,6)
estado(3,0,7) key_on(2,4,3) key_on(2,4,4) key_on(1,5,1)
key_on(2,4,2) key_on(2,4,1)
SATISFIABLE
```

Figura 4.11: Resultado del primer puzzle en Clingo.

4.4 Nuevo puzle: añadiendo una bifurcación

Todo este código puede ser consultado en el Anexo I.

Con el objetivo de aumentar la complejidad de nuestro puzle en Clingo se ha optado por añadirle una bifucarción. Esta bifurcación se colocará de manera aleatoria entre todas las habitaciones existentes del pasillo (salvo la solución final que representa la solución).

Para ello se añadirá al conjunto de parámetros del código anterior al número de habitaciones de la bifurcación. Este se añadirá al conjunto de habitaciones.

```
room(1..r+h).
key(r+h+1..k+r+h).
door(r+h+k+1..r+h+k+k).
```

Figura 4.12: Modificación de los hechos de nuestro problemas.

De esta manera se controla que las habitaciones definidas entre r y h son habitaciones pertenecientes a la bifurcación.

En la parte generativa del problema se añadirá la definición de la bifurcación:

- {bifurcation(R) : room(R),R<r} =1:-secpath(S). Haciendo el “grounding” visto en el punto anterior podemos comprobar que esta regla sirve para indicar que nuestro problema tendrá una bifurcación entre la primera habitación y la última(No incluida la última ya que R debe ser menor que el número de habitaciones).

Adicionalmente se definirá un predicado “jumproom”. Este permite cambiar de el pasillo inicial a la bifurcación. Las “jumproom” siempre serán la habitación bifurcación y la primera habitación del pasillo. Añadiendo una acción adicional “change” que permitirá el paso entre estas dos habitaciones. Dentro de la bifurcación se avanzará con movimientos de izquierda a derecha como en la habitación principal. Gracias a esto se consigue no añadir acciones de

movimientos adicionales en el problema.

Para conseguir que el problema sea resoluble se han añadido restricciones para controlar la posición de las llaves en función de la colocación de las puertas:

- :- key_on(R1,K,1), door_on(D,R2), (D-k)=K, bifurcation(R3), R1<=R3, R2<=R3, R1>=R2. No debe cumplirse que haya una llave en la habitación R1 que corresponda a una puerta que está en una habitación R2 de tal manera que esa llave y esa puerta están antes de la bifurcación y que la habitación de la llave sea posterior a la habitación de la puerta. De esta manera se controla las puertas antes de la bifurcación.
- :- key_on(R1,K,1), door_on(D,R2), (D-k)=K, bifurcation(R3), R1>R3, R1<=r, R2>R3, R2<=r, R1>=R2 . No debe cumplirse que exista una llave en R1 y una puerta en R2 donde esta llave abra esa puerta y que además la habitación de la llave sea posterior a la bifurcación y posterior a la habitación de la puerta. Así se controla las puertas que se generan posterior a la bifurcación.
- :- key_on(R1,K,1), door_on(D,R2), (D-k)=K, R1>r, R2>r, R1>=R2. Esta restricción controla las llaves y las puerta dentro de la propia rama secundaria.
- :- actua(rt,_,T), estado(R,_,T), dooronttherightclosed(R,T). Esta retricción nos indica que no puede existir un movimiento a la derecha y y una puerta cerrada en la misma transición.
- :- actua(ch,_,T), estado(R,_,T), bifurcation(R), dooronbifurcationclosed(T). Similar a la restricción anterior, pero esta vez nos sirve para evitar el paso a la bifurcación si existe una puerta delante de está.
- :- actua(ch,_,T), not jumproom(R), estado(R,_,T). Se controla la acción change para que solo puede realizar en la jumproom.

Con el objtetivo de mejorar la solución obtenida en el problema se han añadido una serie de restricciones que obligan a que exista una llave en la bifurcación y que esta abra una puerta del camino principal. Estas restricciones son las siguientes:

- :- {keyonsec(K) : key(K)} = 0. No puede cumplirse que el número de llaves en el camino secundario sea igual que 0.
- :- {doormainkeysec(D,K) : door(D), key(K)} =0. Esto obliga a que al menos exista una correspondencia entre una llave del camino secundario y una puerta del camino principal.

Como resultado a este nuevo problema se obtienen una serie de predicados que definen las soluciones a este problema añadiendo un camino secundario y dando la posibilidad de que este camino secundario tenga llaves y puertas [16].

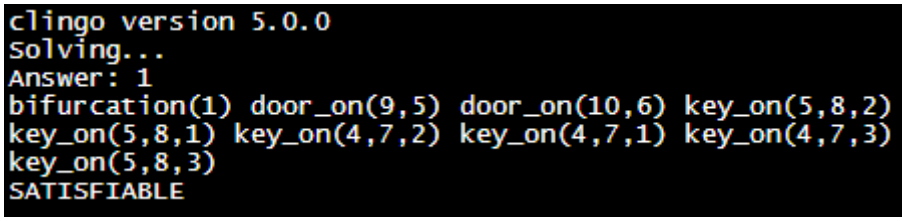

Figura 4.13: Resultado del nuevo puzzle en Clingo.

4.5 Integración del problema de Clingo en C++

Todo este código puede ser consultado en el Anexo II.

Debido a que no hay soporte directo entre Clingo y Unity es necesario realizar un paso intermedio y llevar el código a C++. Clingo dispone de un API y librerías que permiten la definición y resolución de problemas desde un programa C++ que se puede compilar par Windows o Linux. Mediante la utilización de esta API se plantea generar una librería dinámica que permita la obtención de predicados del Clingo y poder trabajar con ellos en Unity.

Como entrada inicial se recibirá el número de habitaciones, el número de puertas y el número de habitaciones de la rama secundaria. De esta manera desde Unity se podrá controlar la generación de habitaciones.

Para la realización de la integración el primer paso será la creación de una clase cuyos atributos más importantes son el número de habitaciones, número de llaves, número de puertas, número de habitaciones de la rama secundaria, además de vectores que guardan el ID de las habitaciones donde se encuentran las llaves, el ID de las habitaciones donde se encuentran las puertas y el vector con la posición de las bifurcaciones. Además es necesario guardar el código Clingo como una cadena de caracteres (char*). Este código de Clingo no debe incluir la definición de variables que dependan de los parámetro de la clase de C++. En este caso serán el predicado “room”, “key”, “door”, “takeable” y “move” (este último se definirá con un número que asegure que el problema tendrá solución que en este caso será el número de habitaciones más el doble del número de llaves). Estos predicados serán añadidos a la ejecución de Clingo, para ello serán guardados en un vector de símbolos.

```
void Room::addsymbolfromkey() {

    for (int i = 0; i < getKey(); i++) {
        clingo_symbol_t skey, stakeable;
        clingo_symbol_create_number(getSCount(), &skey);
        clingo_symbol_create_number(getSCount(), &stakeable);
        clingo_symbol_t skeyh, skey_args[NARGUMENTKEY], stakeableh, stakeable_args[NARGUMENTKEY];
        skey_args[0] = skey;
        stakeable_args[0] = stakeable;

        if (!clingo_symbol_create_function("key", skey_args, NARGUMENTKEY, true, &skeyh))
            throw std::runtime_error{ "Clingo error: creating key predicate" };
        if (!clingo_symbol_create_function("takeable", stakeable_args, NARGUMENTKEY, true, &stakeableh))
            throw std::runtime_error{ "Clingo error: creating takeable predicate" };


        VCSymbols.push_back(skeyh);
        VCSymbols.push_back(stakeableh);
        setSCount();
    }
}
```

Figura 4.14: Función para añadir símbolo llave.

En este ejemplo se recorre mediante un bucle “for” que se repite tantas veces como números de llaves. La variable “sCount” tiene en su interior un contador que va aumentando cada vez que se añade una variable de Clingo, de esta manera se asegura que la asignación de variables sea única. En este caso como la asignación de “key” y “takeable” es la misma se crean y se añaden los dos símbolos a las vez. También es importante en la creación de símbolos el número de argumentos de los predicados. En este caso ese valor se guarda en la constante “NARGUMENTKEY”.

Mediante las funciones que proporciona la librería de Clingo se crea un modelo que permite recorrer los átomos de las soluciones para seleccionar los predicados que interesa. Para ello es necesario crear una “signature” con los datos del predicado a buscar (nombre y número de argumentos). En este caso lo necesario para la creación del puzle en Unity, son la posición de la llaves (una vez obtenidas solo será necesario quedarse con las del estado inicial), la posición de las puertas y la posición de la bifurcación.

```
if (clingo_signature_is_equal_to(sig_key_on, symbolsignature)) {
    clingo_symbol_t const *args;
    size_t nargs;
    if (!clingo_symbol_arguments(*it, &args, &nargs))
        throw std::runtime_error{ "Clingo error: getting symbol arguments" };
    int roomId;
    if (!clingo_symbol_number(*args, &roomId))
        throw std::runtime_error{ "Clingo error: getting obj room id from symbol" };
    int keyId;
    if (!clingo_symbol_number(*(args + 1), &keyId))
        throw std::runtime_error{ "Clingo error: getting obj key id from symbol" };
    int move;
    if (!clingo_symbol_number(*(args + 2), &move))
        throw std::runtime_error{ "Clingo error: getting obj move from symbol" };
    if (move == 1) {
        int pos = keyId - ((Room*)object)->getRoom();
        pos = pos - ((Room*)object)->getHall();
        pos = pos - 1;
        ((Room*)object)->setKey_on(roomId, pos);
    }
}
```

Figura 4.15: Selección de predicado key_on en C++.

Los argumentos obtenidos en este caso son guardados en tres variable locales “roomId”, “keyId” y “move (key_on(R,K,T)”. Debido a que solo interesa la colocación en la posición inicial solo guardaremos aquellos donde “move” sea igual a uno. Todo esto quedará guardado en los distintos vectores para poder acceder a los datos.

El último paso es la creación de la dll. La característica principal es que se pasa por referencia los datos que se quieren llevar a Unity para poder trabajar con ellos.

```
extern "C" __declspec(dllexport) void get_result(int *key, int *door, int *bifurcation, int r, int d, int h) {

    Room room(r, d, h); //Constructo de nuestra clase
    room.placeObjects(); //Función para obtención de soluciones

    for (int i = 0; i < d; i++) {
        key[i] = room.getKey_on(i);
    }
    for (int j = 0; j < d; j++) {
        door[j] = room.getDoor_on(j);
    }
    for (int z = 0; z < 1; z++) {
        bifurcation[z] = room.getBifurcacion_on(z);
    }
}
```

Figura 4.16: Creación dll con nuestro código de C++.

El resultado de este código es una dll que permite ejecutar la función externa

“get_result” que posteriormente será utilizada en Unity para la obtención del puzle.

4.6 Integración del generador de puzzles en un juego Unity.

Tras la obtención de la dll que construya el puzle a partir de los datos, se debe realizar una script de Unity para trasladar este puzle a un entorno gráfico. Para que la dll de C++ funcione en Unity es necesario tenerla en la carpeta assets del proyecto de Unity y la dll general de Clingo dentro de la carpeta editor de Unity. Como objetivo adicional se ha planteado crear una demo jugable de este primer puzle. Para añadir un poco de complejidad a esta demo y que no sea un solo pasillo con una bifurcación se añade la posibilidad de que aleatoriamente se generen caminos predefinidos tanto en el camino principal como en el secundario.

Para la iniciación de este código es necesario que se pase los datos para ejecutar la generación del problema. Estos datos son el número de habitaciones del camino principal, el número de habitaciones del camino secundario y el número de puertas. Además, con el propósito de controlar el aspecto que tendrá el puzle se pasan como parámetro distintos “GameObjects” que serán usados en la construcción como por ejemplo el aspecto de las habitaciones, el asset de las llaves, la puestas y el tamaño (largo) de las paredes de las habitaciones que será necesario para calcular la separación entre las habitaciones.

El primer paso de este código es usar las funciones Unity para ejecutar la función “get_result” y guardar los datos obtenidos. En este caso se guardará la posición de las llaves, la posición de la bifurcación y la posición de las puertas. Para la construcción del puzle el código principal se dividirá en 3 partes, la creación de habitaciones, la colocación de llaves en estas habitaciones y la colocación de puertas. Dentro de la creación de habitaciones, se separará la generación de la rama principal de la secundaria. Para la generación de las habitaciones se recorrerá un bucle “for” que se repetirá tantas veces como habitaciones haya. Este bucle irá generando en cada interacción una habitación. En caso de que la habitación a generar sea la que tiene la bifurcación, se generará una habitación especial abierta (se elegirá aleatoriamente si esta apertura será hacia arriba o hacia abajo) donde se colocarán el resto de habitaciones de la bifurcación. También se hará una separación entre la habitación inicial y final ya que estas tienen unas “prefabs” distintas al resto de habitaciones. Adicionalmente y de manera aleatoria se generará en vez de una habitación, bifurcaciones extra para dar complejidad al problema. Después de colocar las habitaciones de la rama principal (y los caminos extra), se pasará a colocar las habitaciones de la rama secundaria. Estas se colocarán de manera perpendicular a la rama principal (irán hacia arriba o hacia abajo en función de la colocación de la bifurcación en la rama principal). Del mismo modo que en la rama principal, cada vez que se coloca una habitación de la rama de secundaria hay una posibilidad de generar nuevas bifurcaciones.

Tras tener el escenario generado se colocarán en funciones aparte las llaves y las puertas. Para ello hay que tener en cuenta que si la posición de las mismas es menor a el número de habitaciones, esta estará en la rama principal y si es mayor estará en la rama secundaria. Además hay que tener en cuenta si la bifurcación se generó hacia arriba o hacia abajo.

```
void showKey(){
    for(int i = 0;i < n_door; i++){
        Vector3 myPos = new Vector3(initialPos.x, initialPos.y, initialPos.z);
        if (key_on[i] - 1 < n_room){
            myPos = new Vector3(0.5f, -2.0f, (-wallLength / 2) + (key_on[i] - 1) * wallLength);
            tempObject = Instantiate(key, myPos, Quaternion.identity) as GameObject;
            tempObject.name = "Key" + i;
        }
        else{
            if (binary == 0){ //variable que contiene la dirección de la bifurcación
                myPos = new Vector3(0.5f, -2.0f, (-wallLength / 2) + ((bifurcacion_on[0] - 1) * wallLength));
                tempObject = Instantiate(key, myPos, Quaternion.identity) as GameObject;
                tempObject.name = "Key" + i;
            }

            else{
                myPos = new Vector3(0.5f + ((key_on[i] - n_room - 1) * wallLength + 1), -2.0f, (-wallLength / 2) + ((bifurcacion_on[0] - 1) * wallLength));
                tempObject = Instantiate(key, myPos, Quaternion.identity) as GameObject;
                tempObject.name = "Key" + i;
            }
        }
    }
}
```

Figura 4.17: Función mostrar llaves.

Una problemática encontrada a la hora de la generación de este mapeo inicial de la demo jugable es que las las bifurcaciones del camino principal y las bifurcaciones del camino secundario podrían interceptarse entre ellas dando lugar a habitaciones que no podrían ser accesibles de ninguna manera. Para arreglar este problema se creó una matriz cuyos elementos estaban inicialmente a 0. El tamaño de esta matriz acotaba los límites del escenario generado. De esta manera a medida que se iba añadiendo una habitación al juego, se cambiaba su posición dentro de la matriz a 1. En el momento que en el camino secundario se añada alguna bifurcación, se realiza una comprobación para asegurar que el espacio necesario para adicionar habitaciones no esté siendo utilizado actualmente. En caso de que esto ocurra no se añade ningún desvío en esa casilla sino que se añade una habitación normal.

Añadiendo un personaje jugable en tercera persona a la “prefabs” de la habitación inicial se obtiene como resultado del código un escenario con llaves y puertas, aunque todavía no se puede jugar debido a que hace falta crear scripts adicionales para las acciones del juego. Por ahora el resultado obtenido será el mapeo inicial del puzle.

Figura 4.18: Mapeado generado aleatoriamente del juego.

4.7 Código de unity adicionales creados para la demo

Para conseguir una demo jugable es necesario definir códigos adicionales para controlar las acciones del personaje como coger una llave o abrir un puerta. Además es necesario definir una interfaz de usuario que contenga el inventario y los números de acciones (se define esto como reto para que los jugadores intenten completar el puzle en el menor número de pasos posibles). También se definirá en la baldosa final un mensaje de victoria cuando el jugador llegue a esa zona como recompensa final de la demo.

Para las acciones de coger una llave y abrir una puerta es necesario crear una script adicional y que estas sean añadidas a sus respectivos objetos (la script de coger irán en las “prefabs” de cada una de las llaves y la de abrir en cada una de las puertas). Ambas se basarán en la utilización de un sistema de colisiones que detecta cuándo el jugador esté cerca de los objetos. Para ello es necesario añadir un “collider” con la forma que queramos en el objeto.

Figura 4.19: Modelo de llave con collider

Una vez colocado el “collider” se marcará la opción de “is Trigger” para que sea posible atravesar esta malla y pueda activar el evento “trigger” cuando el jugador esté cerca. Para saber que la colisión se activa dentro del “GameObject” del jugador se indicará que tiene el “tag player”.

```
void OnTriggerStay(Collider other){
    if(other.tag == "Player"){
```

Figura 4.20: Función Trigger de Unity.

En el caso de la llave, se activará cuando el jugador esté dentro del “collider” y pulse la tecla “E”. Para ello Unity dispone de la función “Input”. “GetKey (KeyCode.tecla deseada)”. Como es necesario saber qué puerta abre cada llave en el momento de su creación y su colocación en el escenario, se le añadió el número de identificación de la misma al final de su nombre. Cuando el jugador active la colisión y pulse “E”, se extraerá este ID de la llave y se guardará en una variable global (será global debido a que es necesario este número para otros códigos como el de apertura de la puerta) y se destruirá el objeto “llave”. Para el caso de la puerta solo será necesario que se active el evento de colisión y que el jugador tenga en la variable inventario la llave con la misma ID numérica que la puerta (al igual que las llaves, el ID se guarda también al final de su nombre).

Para la acción de soltar la llave, se añadirá en la función “update” del código principal (esta función se ejecuta en cada “frame” del juego), se indicará que si el jugador pulsa la tecla “Q” se crea una “prefab” de llave en esa posición, en cuyo título está el ID de la llave, que se guarda en el inventario ,y dejará el inventario vacío asignándole el valor -1(el 0 es para la primera llave).

Para el control de interfaz, Unity facilita un tipo de “GameObject” llamado “canvas”. Este tiene la particularidad de que todos los objetos que lo componen deben ser en 2D y siempre se verán al frente de la pantalla. Para la creación del inventario se utilizará dos “sprite”. Uno con el inventario vacío y

otro con el inventario lleno.

Figura 4.21:
Inventario
lleno/vacío.

El “sprite” vacío se colocará encima del otro. En el momento que se coja la llave se deshabilita el “gameObject” del “sprite” vacío (se busca mediante el “tag” de “inventory” como se hizo con el jugador), haciendo que desaparezca y se vea el inventario con la llave. Una vez se suelte la llave o se abra la puerta se volverá a habilitar quedando el inventario vacío de nuevo.

Para el contador de acciones se añadirá al “canvas” un “GameObject” del tipo “string” que ponga “Contador de acciones” y debajo del otro, uno distinto que inicialmente ponga 0. Mediante la utilización de una variable global que aumentará cada vez que se ejecute una acción como coger una llave, soltar una llave y abrir una puerta. Para el conteo de pasos se debe añadir un “collider” en cada “prefabs” de habitación y una “script” que actualice el contador en el momento que se salga de la colisión. De esta manera se consigue que este número se actualice con cada una de las acciones. Por último mediante una función “update” se actualiza el “text” en cada “frame” para que se vea en tiempo real el número de acciones realizadas en cada momento.

Por último una vez finalizado el puzle aparecerá un mensaje de victoria. Para ello se añadirá al “canvas” el mensaje de victoria. Este inicialmente estará desactivado y para activarlo es necesario que el jugador llegue a la casilla final. Añadiendo un “collider” en la casilla final y una “script” de control de colisiones como se ha hecho en ocasiones anteriores se conseguirá el objetivo deseado. También se ha añadido una función que permite la obtención de la escena de Unity con la “script” con los parámetros añadidos. Esta escena será de carácter global y en el momento que se activa el mensaje de victoria utilizando la función “Application.LoadLevel(escena a recargar)”. Esto permite cargar de nuevo la escena haciendo que aparezca un puzle nuevo con las mismas condiciones que el puzle anterior, pero totalmente distinto, añadiendo un flujo continuo al juego generado.

Como resultado final, de estos códigos se obtendrá la primera demo jugable que permite realizar todas las acciones posibles y de manera resoluble. Esta demo es totalmente parametrizable por el desarrollador y tiene un flujo continuo, ya que cada vez que se finaliza una pantalla se genera una nueva completamente distinta a la anterior.

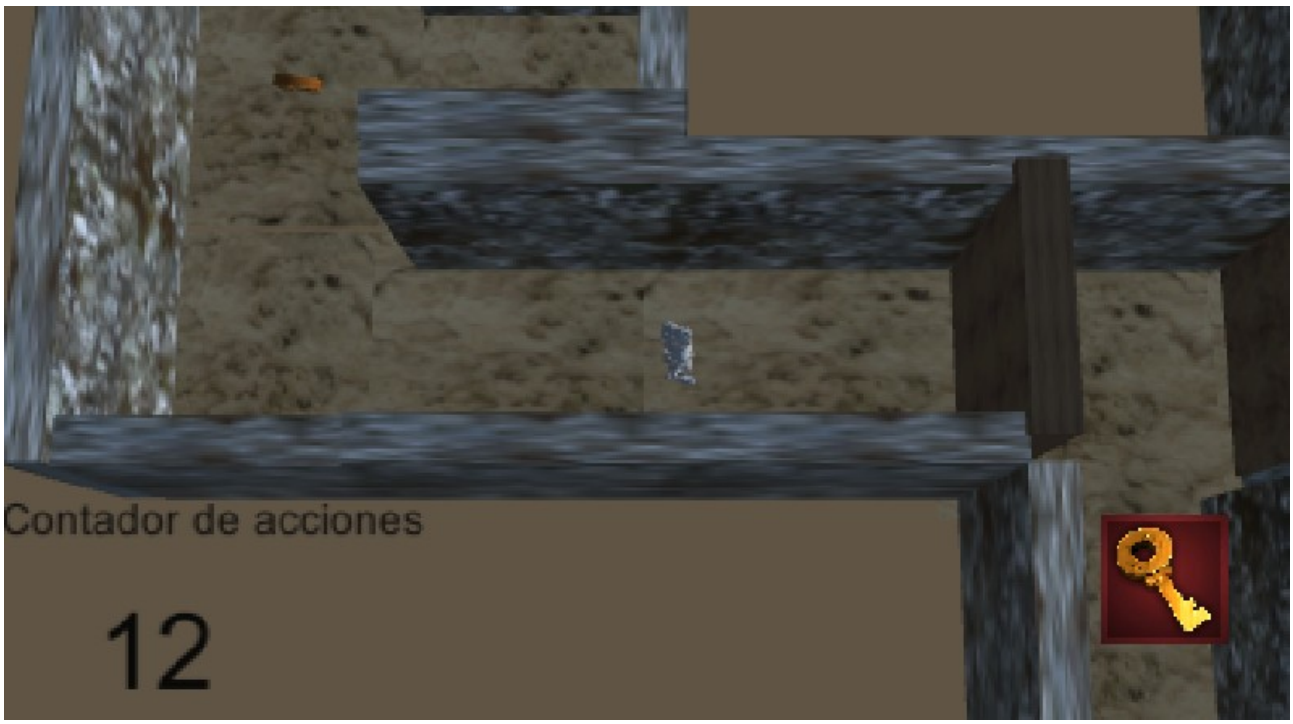

Figura 4.22: Ejemplo de demo jugable.

Figura 4.23: Mapeado r = 4, h = 3, k = 1

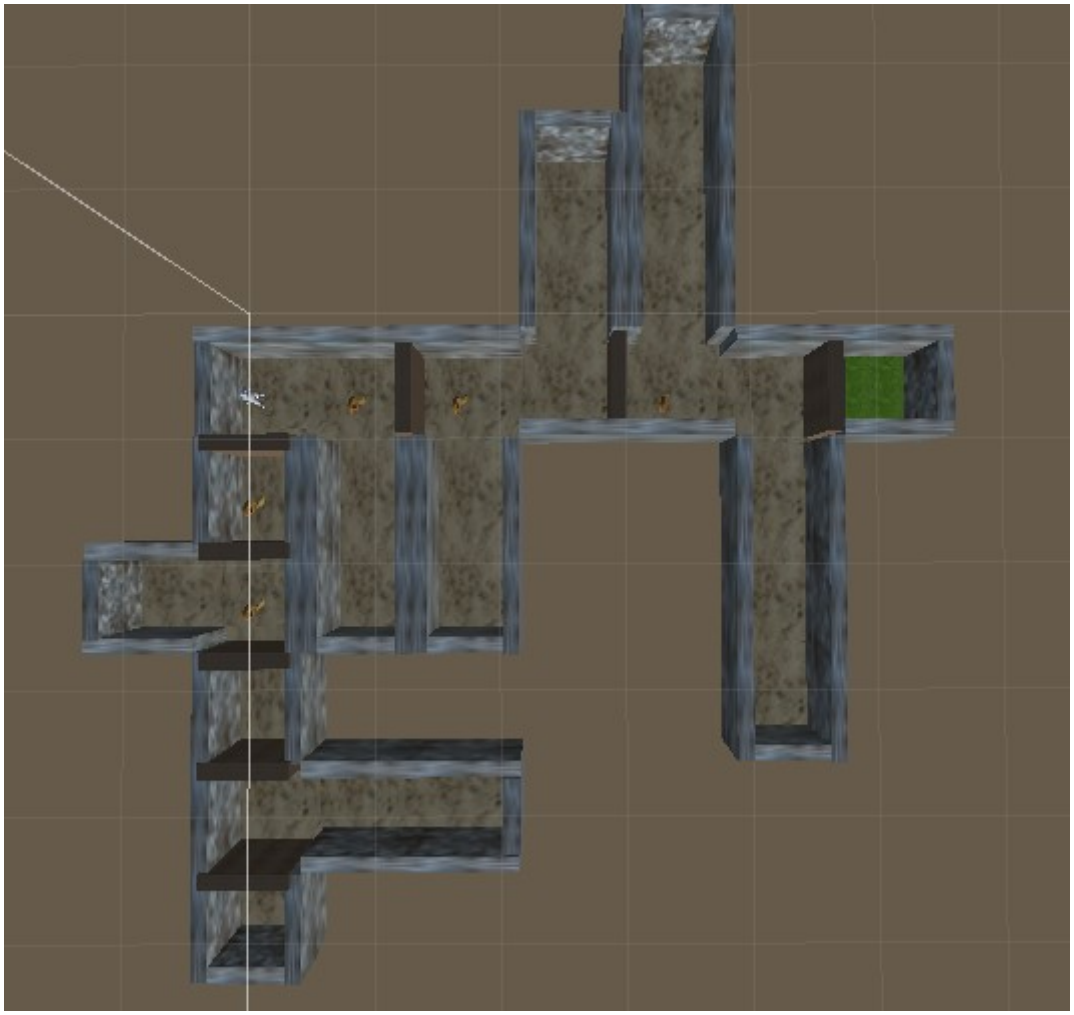

Figura 4.24: Mapeado r = 7, h = 5 , k = 8

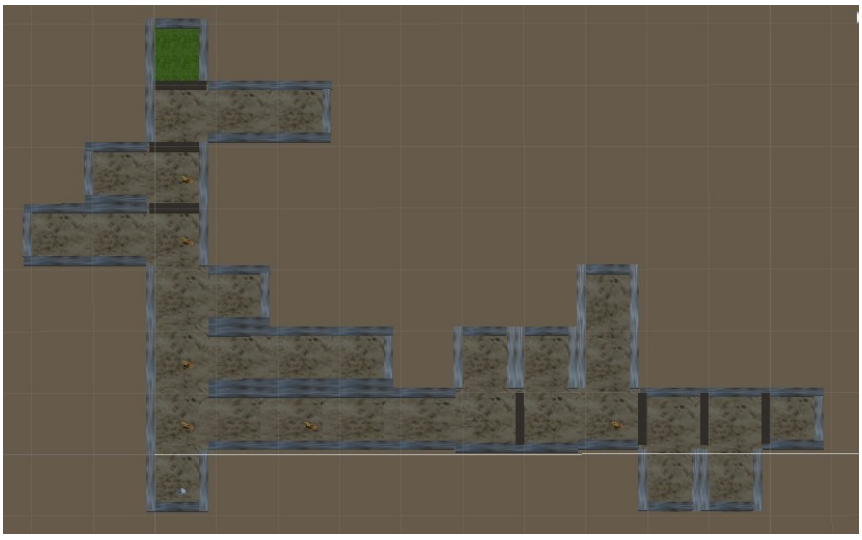

Figura 4.25: Mapeado r = 8, h = 10, k = 7

4.8 Resultados obtenidos

- Algoritmo de generación de laberintos: Se ha realizado un código capaz de generar laberintos. Cada vez que se ejecuta este código se obtiene un resultado distinto. Además este posee una serie de parámetros que se pueden modificar y cambiar así el tamaño del mismo.
- Codificación de un puzle básico mediante un lenguaje declarativo(ASP): Mediante la utilización del lenguaje ASP se ha codificado un problema básico de llaves/puertas. Este consiste en la colocación de una serie de llaves y puertas en un pasillo. La principal característica del puzle generado es que siempre tiene una solución factible.
- Integración Unity-clingo: Gracias a la codificación de una dll en c++ se ha conseguido poder trasladar la el puzle codificado en ASP a un entorno gráfico en Unity.
- Implementación de nuevos puzles: Se ha modificado nuestro problema incial en ASP para que se incluya una bifurcación en la que también puedan aparecer llaves y puerta aumentando la complejidad del problema.
- Implementación de demo jugable: Se han incluido una serie de scripts de Unity para incluir una serie de elementos necesario para poder jugar este proyecto. Entre las mejoras que aparecen cabe destacar la generación de caminos aleatorios a nuestro pasillo.

Capítulo 5
Conclusiones y líneas futuras

Como conclusión, el objetivo de este trabajo era la realización de un videojuego que se generase completamente de manera procedural y que no se viera afectado por la problemática habitual de este tipo de juegos. La problemática consiste en que estos tipos de juegos no logran captar el interés de los jugadores, ya que suelen tener mecánicas repetitivas que acaban aburriendo a los jugadores. La solución planteada a este problema es la creación de un sistema de generación de puzles cuya principal característica es que sean resolubles y que se cambien con cada ejecución del juego. El escenario que se plantea es un laberinto porque esto permite que el propio lugar donde se realiza la acción sea un reto adicional para el jugador. Este está lleno de rompecabezas de distintos tipos, haciendo más entretenido cada nivel superado del juego. Cabe destacar que es importante que por muchos puzles generados en el juego no pueden interponerse los unos con los otros ya que el nivel siempre debe tener alguna solución. Además, con la parametrización del escenario y de los puzles generados se consigue que estos vayan aumentando de dificultad exponencialmente a medida que el jugador avanza para así conseguir una motivación que le incentive a continuar jugando.

El resultado obtenido en este trabajo es aislar una parte del objetivo planteado y llevarla a cabo. De esta manera se obtiene una pequeña demo donde el jugador podrá resolver un puzle de llaves/puertas y que este se genere de manera aleatoria en cada ejecución. Además la solución obtenida ha sido planteada para que pueda ser usada desde el editor de Unity y cambiar cada uno de los aspectos de la solución obtenida sin necesidad de cambiar el código utilizado, por lo que la hace ideal para ser reutilizada en nuevos proyectos. Con esto se consigue un acercamiento de las tecnologías de ASP al entorno de Unity.

Con este trabajo he aprendido cómo funciona Unity y cómo realizar diversas acciones necesarias para el desarrollo de juegos con este motor gráfico, como la utilización de un sistema orientado a eventos. Dentro de este entorno, he adquirido conocimientos tales como el trabajo con “collider” para la activación de eventos que ejecuten diversas acciones en el mundo generado. También cómo se puede controlar los distintos parámetros del código de “script” en Unity mediante la utilización de variables del tipo pública. Otro conocimiento fundamental adquirido es la utilización de variables globales para compartir información entre las “script” de Unity.

En cuanto a Clingo, fue totalmente nuevo para mí, por lo que tuve que adquirir conocimientos generales sobre este lenguaje. Lo fundamental para la utilización del mismo no es tanto conocer cómo funciona sino tener claro cuál

es el problema a tratar y cómo definirlo mediante una serie de sentencias lógicas.

Con este trabajo he fortalecido y mejorado en la utilización del depurado en distintos sistemas. La utilización de la misma es fundamental en cualquier desarrollo ya que te permite ejecutar el código de manera secuencial para facilitar el encuentro de errores y fallos sobretodo en la ejecución del código. Además, te permite seleccionar los puntos de parada en concreto para centrarse solo en la zona de código que interesa. También he visto importante el aprendizaje realizado de Clingo, aprendiendo a trabajar con tecnologías nuevas y adaptarse a ellas lo más rápido posible. Esto es un conocimiento necesario en la época actual ya que la informática es un mundo cambiante donde hay una constante actualización de los recursos y tecnologías usadas.

Durante la realización del trabajo han ido apareciendo una serie de problemas que se han solucionado a medida que iban surgiendo. El primero era encontrar la manera de integrar Unity con Clingo. Para ello se optó por la utilización de un módulo intermedio en C++ que permitiera trasladar los datos obtenidos en Clingo a Unity de una manera sencilla. Posteriormente, al añadir un personaje en tercera persona al escenario apareció un problema con la cámara del juego. Al obligar a la cámara a hacer un seguimiento del personaje, esta se giraba completamente con el movimiento del personaje, lo que también hacía que los controles de este no funcionaran correctamente. Para solucionarlo hacía falta añadir una “script” de “smooth follow” a la cámara. Esta “script” es una de las que componen los “standard assets” de Unity. Otro problema que ocurría fue que a la hora de generar los caminos en la demo jugable, habían algunos que se cruzaban entre ellos, haciendo que hubieran algunas casillas que podrían no ser accesibles por el jugador. Como hemos visto anteriormente este problema se arregló utilizando una matriz y realizando una comprobación antes de añadir ciertos caminos (más información en el punto 4.5). Otro inconveniente encontrado fue que en la demo el jugador podía llegar al final, aparecía el mensaje de victoria, pero no se podía continuar el juego. Mediante la utilización de algunas funciones en Unity, se ha conseguido solventar el problema y se ha añadido flujo continuo al juego. Además, para intentar dar un mayor sentido al juego y un pequeño incentivo extra, se añadió un contador de acciones a modo de desafío.

Una de las mejoras que se le podría agregar al proyecto, es añadir algunos incentivos más al juego como un sistema de puntuación al final de cada nivel. Este se basaría en algunos sistemas del juego como por ejemplo completar el nivel con un número lo más bajo posible de acciones. Una vez obtenido este sistema de puntuación se podría añadir objetivos adicionales (como un tesoro en el escenario) que hagan sumar más puntos al finalizar la pantalla. Otra mejora planteable es la de trasladar el juego a un entorno móvil, ya que sería el entorno ideal para su ejecución debido a que, al ser un juego de niveles independientes y cortos, permite ser jugado en cualquier rato libre y con la posibilidad de abandonarlo en cualquier momento sin ningún tipo de penalización. Teniendo el presupuesto adecuado, sería una muy buena opción la mejora de los “sprite” de los objetos del juego y la mejora tanto del personaje como de las animaciones del propio juego. En cuanto al Clingo se podrían añadir una serie de restricciones adicionales sobre las soluciones(en concreto la secuencia de acciones) para lograr problemas más interesantes, o controlar mejor el grado de dificultad del nivel.

Como conclusión, mi meta es llegar hacer un desarrollador de videojuegos en un futuro, por lo que ha sido importante para mi la realización de este

trabajo, aprendiendo sobre distintas técnicas y tecnologías del desarrollo de videojuegos. Esto me servirá como punto de partida para seguir trabajando en conseguir esta meta.

Capítulo 6
Summary and Conclusions

The main purpose of this project is to create a videogame that is completely generated by a procedural way and making this game not affected by the average problems of this sort of games. The problem seen on these games is the failure they have in terms of capturing the interest of players. They tend to be repetitive, making the players fed up of them. The solution that we carried out in order to solve this problem out was making systems of puzle creation which main feature is the addition of a different solution every time the game starts. The scenario of the game is a maze because it is already a challenge. This fact gives the game extra difficulty making the player eager to keep playing. The maze will be built by different kind of puzzles, making it more interesting. Furthermore, with the parameterization of the scenario and of the puzzles, we can accomplish the increase of the difficulty of the game every time the player overcomes a level. This will give them the motivation to keep playing.

The results obtained in this project is an isolate part of the objective. This means that we have obtained a small demo where the player can solve a puzle of keys/doors and this puzle is randomly generated with every execution of the game. In addition to this,we can parameterize the solution obtained from the editor ,change how the game looks like and create new puzle features without changing the code. This is ideal for reusing our project in new games and approach the Clingo technology to Unity.

In addition to this, the project has helped me to improve my skill as a developer. For example, I have improved the way I debug my code. The debugger process is important because it allows you to execute the code in a sequential way in order to make it easier to find and fix mistakes in the code. Also, I learnt how to use Clingo for building a puzle generator. I believe it is important learning new technologies and techniques because the computer science field is always changing. This means that adaptability is a skill that needs to be existent in a developer.

In conclusion, my goal is to become a videogame developer. Due to this, making this project was very important to me. It has helped me to learn about different techniques and technologies. This project will be my starting point to begin working on achieving this personal goal.

Capítulo 7
Presupuesto

Descripción	Gastos
Horas de análisis	1.120€
Horas de programación	2800€
Licencia de unity	125€
Diseños	200€
Música	200€

Tabla 7.1: Presupuesto del juego

Se ha calculado que el proyecto final serían unas 14 semanas (4 de análisis y 10 de desarrollo), teniendo en cuenta que en cada semana se trabajan 40 horas y que el precio de las horas de un programador junior ronda los 7€, el resultado es que el desarrollo solo costaría unos 3.920€ (1.120€ + 2800€). Además sería necesario obtener la licencia profesional de Unity para poder tener todas la ventajas adicionales que ofrece a la hora de desarrollar.

Por último una parte fundamental de todo videojuego es los diseños que lo componen y la banda sonora del mismo. En este caso se ha establecido que se pagará mayoritariamente por el diseño del personaje principal. En el caso de la banda sonora, al tratarse de un juego simple solo se dispondrá de un “main theme” que se irá repitiendo en bucle durante toda la ejecución del juego.

Bibligrafía

[1]Es.wikipedia.org. (n.d.). Rogue. [online] Disponible en: https://es.wikipedia.org/wiki/Rogue [Fecha de consulta: 03 de junio del 2018].

[2]Es.wikipedia.org. (n.d.). Roguelike. [online] Disponible en: https://es.wikipedia.org/wiki/Roguelike [Fecha de consulta: 03 de junio del 2018].

[3]Es.wikipedia.org. (n.d.). Spelunky. [online] Disponible en: https://es.wikipedia.org/wiki/Spelunky [Fecha de consulta: 05 de junio del 2018].

[4]Kazemi, D. (2013). Spelunky Generator Lessons. [online] Tinysubversions.com. Disponible en: http://tinysubversions.com/spelunkyGen/ [Fecha de consulta: 05 de junio del 2018].

[5]Kazemi, D. (2013). Spelunky Generator Lessons. [online] Tinysubversions.com. Disponible en: http://tinysubversions.com/spelunkyGen2/ [Fecha de consulta: 05 de junio del 2018].

[6]Es.wikipedia.org. (n.d.). The Binding of Isaac. [online] Disponible en: https://es.wikipedia.org/wiki/The_Binding_of_Isaac [Fecha de consulta: 05 de junio del 2018].

[7]McMillen, E. (2011). The Binding of Isaac Gameplay explained. [online] Edmundmcmillen.blogspot.com. Disponible en: http://edmundmcmillen.blogspot.com/2011/09/binding-of-isaac-gameplay-explained.html [Fecha de consulta: 05 de junio del 2018].

[8]Es.wikipedia.org. (n.d.). No Man's Sky. [online] Disponible en: https://es.wikipedia.org/wiki/No_Man%27s_Sky [Fecha de consulta: 10 de junio del 2018].

[9]Martin, A. (2016). El universo infinito de No Man’s Sky es un reflejo del nuestro. [online] Hipertextual. Disponible en: https://hipertextual.com/2016/08/universo-no-mans-sky [Fecha de consulta: 10 de junio del 2018].

[10]Antonova, E. (2015). [online] Aaltodoc.aalto.fi. Disponible en: https://aaltodoc.aalto.fi/bitstream/handle/123456789/19241/master_Antonova_

Evgenia_2015.pdf?sequence=1&isAllowed=y [Fecha de consulta: 3 de Septiembre de 2018].

[11]Smith, A. and Mateas, M. (2011). [online] Course.ccs.neu.edu. Available at: https://course.ccs.neu.edu/cs5150f13/readings/smith_asp4pcg.pdf [Fecha de consulta: 3 de Septiembre de 2018].

[12]Es.wikipedia.org. (n.d.). Unity (motor de juego). [online] Disponible en: https://es.wikipedia.org/wiki/Unity_(motor_de_juego) [Fecha de consulta: 14 de octubre del 2017].

[13]Unity. (n.d.). Unity. [online] Disponible en: https://unity3d.com/es [Fecha de consulta: 14 de octubre del 2017].

[14]Universidad de Potsdam. (n.d.). Potassco - the Potsdam Answer Set Solving Collection. [online] Disponible en: http://potassco.sourceforge.net/ [Fecha de consulta: 24 de marzo del 2018].

[15]Es.wikipedia.org. (n.d.). Problema de satisfacción de restricciones. [online] Disponible en: https://es.wikipedia.org/wiki/Problema_de_satisfacci%C3%B3n_de_restricciones [Fecha de consulta: 25 de julio del 2018].

[16]Estevez Damas, J. (2018). Apuntes introductorios sobre ASP. [online][Fecha de consulta 27 de agosto del 2018].

[17]FlatTutorials (2015). Creating a maze generator - Creating a Closed Grid - Unity 3D. [online] YouTube. Disponible en: https://www.youtube.com/watch?v=OzENv_ZRA1g&t=1244s [Fecha de consulta: 17 de diciembre del 2017].

[18]FlatTutorials (2015). Creating a maze generator - Creating Cells - Unity 3D. [online] YouTube. Disponible en: https://www.youtube.com/watch?v=r4fQzkPzNx4&t=1243s [Fecha de consulta: 17 de diciembre del 2017].

[19]FlatTutorials (2015). Creating a maze generator - Finding neighbors - Unity 3D. [online] YouTube. Disponible en: https://www.youtube.com/watch?v=57RHhUXOA60 [Fecha de consulta: 17 de diciembre del 2017].

[20]FlatTutorials (2015). Creating a maze generator - Applying DFS (Depth first search) - Unity 3D. [online] YouTube. Disponible en: https://www.youtube.com/watch?v=z7wHZMB9YYs [Fecha de consulta: 17 de diciembre del 2017].

Anexo I : Código clingo

```
**********************************************************************
**********
*
* problem.lp
*
**********************************************************************
******
*
* AUTORES
*
*     Luis David Padilla Martín
*     José Ignacio Estévez Damas
*
* FECHA
*
*    22 de abril del 2018
*
* DESCRIPCION
*
*    Codificación de un problema simple de llaves puertas mediante la
*    utilización de ASP.
*
**********************************************************************
******/

#const r=3.
#const k=2.
room(1..r).
key(r+1..k+r).
door(r+k+1..r+k+k).
takeable(0).
takeable(r+1..k+r).
doorproperty(op;cl).
movaction(lt;rt).
accion(lt;rt;tk;re;uop).
```

```
moves(8).
estado(1,0,1).

{key_on(R,K,1) : room(R)}=1:- key(K).
{door_on(D,R):room(R)}=1:- door(D).
doorstate(D,cl,1) :- door(D).
:- door_on(D1,R),door_on(D2,R), D1!=D2.
:- door_on(D,1).
:- key_on(R1,K,1), door_on(D,R2), (D-k)=K, R1>R2.

{ actua(A,K,T) : accion(A), takeable(K) }<=1 :- T=1..M-1,moves(M).
:- actua(A,K,T), movaction(A),K>0.
:- actua(_,_,T), not actua(_,_,T-1),T>1.
:- actua(lt,_,T), estado(1,_,T).
:- actua(rt,_,T), estado(r,_,T).
:- actua(tk,0,T).
:- actua(re,0,T).
:- actua(uop,K,T), not key(K).
:- actua(re,K1,T),estado(_,K2,T),K1!=K2.
:- actua(tk,K1,T),not key_on(R,K1,T),estado(R,_,T).
:- actua(tk,_,T), estado(_,K,T),K>0.
:- estado(R,K1,T),actua(uop,K2,T),K1!=K2.
:- estado(R,_,T),not actua(_,_,T-1),T>1.
:- key_on(_,_,T),not actua(_,_,T-1),T>1.
:- doorstate(_,_,T),not actua(_,_,T-1),T>1.
:- not estado(r,_,_).
:- estado(r,_,T), actua(_,_,T).

doorontherightclosed(R,T):-
doorstate(D,cl,T),door_on(D,R+1),estado(R,_,T),T=1..M,moves(M).
estado(R,K,T) :- not doorontherightclosed(R-1,T-1),estado(R-1,K,T-1), R<=r,
actua(rt,0,T-1),T=1..M,moves(M).
key_on(R,K,T):-key_on(R,K,T-1),actua(rt,_,T-1).
doorstate(D,S,T):-doorstate(D,S,T-1), actua(rt,_,T-1).
estado(R,K,T) :- estado(R+1,K,T-1), R>=1, actua(lt,0,T-1),T=1..M,moves(M).
key_on(R,K,T):-key_on(R,K,T-1),actua(lt,_,T-1).
doorstate(D,S,T):-doorstate(D,S,T-1), actua(lt,_,T-1).
estado(R,K,T) :- estado(R,0,T-1), actua(tk,K,T-1),T=1..M,moves(M).
:- key_on(R,K,T),estado(R,_,T-1),actua(tk,K,T-1).
key_on(R,K,T):-key_on(R,K,T-1),actua(tk,K1,T-1), K!=K1.
doorstate(D,S,T):-doorstate(D,S,T-1),actua(tk,_,T-1).
estado(R,0,T) :- estado(R,K,T-1), actua(re,K,T-1),T=1..M,moves(M).
key_on(R,K,T) :- estado(R,_,T-1), actua(re,K,T-1),T=1..M,moves(M).
doorstate(D,S,T):-doorstate(D,S,T-1),actua(re,_,T-1).

doorstate(D,op,T):- door_on(D,R+1), doorstate(D,cl,T-1), estado(R,K,T-1),K==D-k,
actua(uop,K,T-1),T=1..M,moves(M).
estado(R,0,T):- door_on(D,R+1), doorstate(D,cl,T-1), estado(R,K,T-1),K==D-k,
actua(uop,K,T-1),T=1..M,moves(M).
```

```
key_on(R,K,T):-key_on(R,K,T-1),actua(uop,K1,T-1),K!=K1.
doorstate(D,C,T):-doorstate(D,C,T-1),actua(uop,K,T-1),K!=D-k.

**********************************************************************
**********
*
* keysanddors.lp
*
**********************************************************************
******
*
* AUTORES
*
*     Luis David Padilla Martín
*     José Ignacio Estévez Damas
*
* FECHA
*
*    31 de agosto del 2018
*
* DESCRIPCION
*
*    Codificación de una una bifurcación a nuestro puzzle básico de ASP.
*
**********************************************************************
******/

#const r=3.
#const k=2.
#const h=3.
#const m=10.
room(1..r+h).
key(r+h+1..k+r+h).
door(r+h+k+1..r+h+k+k).
secpath(1).
takeable(0).
takeable(r+h+1..k+h+r).

doorproperty(op;cl).

movaction(lt;rt;ch).

accion(lt;rt;tk;re;uop;ch).
moves(m).

{bifurcation(R) : room(R),R<r} =1:-secpath(S).


{key_on(R,K,1) : room(R)}=1:- key(K).

jumproom(r+1):-h>0.
```

```
jumproom(R):-bifurcation(R).

{door_on(R,D):room(R)}=1:- door(D).
doorstate(D,cl,1) :- door(D).
:- door_on(R,D1),door_on(R,D2), D1!=D2.
:- door_on(1,D).
:- key_on(R1,K,1), door_on(R2,D), (D-k)=K, bifurcation(R3), R1<=R3, R2<=R3,
R1>=R2.
:- key_on(R1,K,1), door_on(R2,D), (D-k)=K, bifurcation(R3), R1>R3, R1<=r, R2>R3,
R2<=r, R1>=R2.
:- key_on(R1,K,1), door_on(R2,D), (D-k)=K, R1>r, R2>r, R1>=R2.
:- key_on(r,K,1).

{ actua(A,K,T) : accion(A), takeable(K) }<=1 :- T=1..M-1,moves(M).
:- actua(A,K,T), movaction(A),K>0.
:- actua(_,_,T), not actua(_,_,T-1),T>1.
:- actua(lt,_,T), estado(1,_,T).
:- actua(lt,_,T), estado(r+1,_,T).
:- actua(rt,_,T), estado(r,_,T).
:- actua(rt,_,T), estado(r+h,_,T).
:- actua(rt,_,T), estado(R,_,T), doorontherightclosed(R,T).
:- actua(ch,_,T), estado(R,_,T), bifurcation(R), dooronbifurcationclosed(T).
:- actua(ch,_,T), not jumproom(R), estado(R,_,T).
:- actua(tk,0,T).
:- actua(re,0,T).
:- actua(uop,K,T), not key(K).
:- actua(uop,K,T), estado(R,K1,T), K!=K1.
nextroom(R1,R2) :- room(R1), room(R2), not bifurcation(R1), R2=R1+1.
nextroom(R1,r+1) :- bifurcation(R1).
nextdoor(R,R1,D) :- door_on(R1,D), nextroom(R,R1), room(R), room(R1), door(D).
:- actua(uop,K,T), estado(R,K,T), {nextdoor(R,_,D) : D=K+k}=0.
:- actua(re,K1,T),estado(_,K2,T),K1!=K2.
:- actua(tk,K1,T),not key_on(R,K1,T),estado(R,_,T).
:- actua(tk,_,T), estado(_,K,T),K>0.
:- estado(R,K1,T),actua(uop,K2,T),K1!=K2.
:- estado(R,_,T),not actua(_,_,T-1),T>1.
:- key_on(_,_,T),not actua(_,_,T-1),T>1.
:- doorstate(_,_,T),not actua(_,_,T-1),T>1.
:- not estado(r,_,_).
:- estado(r,_,T), actua(_,_,T).

doorontherightclosed(R,T):-
doorstate(D,cl,T),door_on(R+1,D),estado(R,_,T),T=1..M,moves(M).
dooronbifurcationclosed(T):-doorstate(D,cl,T),door_on(r+1,D),T=1..M,moves(M).
estado(R,K,T) :- estado(R-1,K,T-1),  actua(rt,0,T-1),T=1..M,moves(M).
estado(r+1,K,T) :- bifurcation(R), estado(R,K,T-1),actua(ch,0,T-
1),T=1..M,moves(M).
estado(R,K,T) :- bifurcation(R), estado(r+1,K,T-1),actua(ch,0,T-
1),T=1..M,moves(M).


key_on(R,K,T):-key_on(R,K,T-1),actua(ch,_,T-1).
doorstate(D,S,T):-doorstate(D,S,T-1),actua(ch,_,T-1).
key_on(R,K,T):-key_on(R,K,T-1),actua(rt,_,T-1).
doorstate(D,S,T):-doorstate(D,S,T-1), actua(rt,_,T-1).
estado(R,K,T) :- estado(R+1,K,T-1), R>=1,actua(lt,0,T-1),T=1..M,moves(M).
key_on(R,K,T):-key_on(R,K,T-1),actua(lt,_,T-1).
doorstate(D,S,T):-doorstate(D,S,T-1), actua(lt,_,T-1).
estado(R,K,T) :- estado(R,0,T-1), actua(tk,K,T-1),T=1..M,moves(M).
:- key_on(_,K,T),estado(R,_,T-1),actua(tk,K,T-1).
key_on(R,K,T):-key_on(R,K,T-1),actua(tk,K1,T-1), K!=K1.
```

```
doorstate(D,S,T):-doorstate(D,S,T-1),actua(tk,_,T-1).
estado(R,0,T) :- estado(R,K,T-1), actua(re,K,T-1),T=1..M,moves(M).
key_on(R,K,T) :- estado(R,_,T-1), actua(re,K,T-1),T=1..M,moves(M).
doorstate(D,S,T):-doorstate(D,S,T-1),actua(re,_,T-1).
doorstate(D,op,T):- nextdoor(R,_,D),estado(R,K,T-1),actua(uop,K,T-
1),T=1..M,moves(M).
estado(R,0,T):- nextdoor(R,_,D),estado(R,K,T-1),actua(uop,K,T-
1),T=1..M,moves(M).
key_on(R,K,T):-key_on(R,K,T-1),actua(uop,K1,T-1),K!=K1.
doorstate(D,C,T):-doorstate(D,C,T-1),actua(uop,K,T-1),K!=D-k.
keyonsec(K):-key_on(Rd,K,1),Rd>r,room(Rd),key(K).
dooronmain(D):-door_on(Rd,D),Rd<=r,room(Rd),door(D).
doormainkeysec(D,K):- dooronmain(D), keyonsec(K), D=K+k, door(D), key(K).
:- {keyonsec(K) : key(K)} = 0.
:- {dooronmain(D) : door(D)} = 0.
:- {doormainkeysec(D,K) : door(D), key(K)} =0.
```

Anexo II: Código C++

```
**********************************************************************
**********
*
* Room.h
*
**********************************************************************
******
*
* AUTORES
*
*     Luis David Padilla Martín
*     José Ignacio Estévez Damas
*
* FECHA
*
*    26 de junio del 2018
*
* DESCRIPCION
*
*    Cabecera de la clase en c++ encargada de obtener las soluciones de
*    nuestro puzle mediante la utilización de la librería de Clingo.
*
**********************************************************************
******/

#pragma once
#include "clingo.h"
#include <vector>
#include <memory>
#include <iostream>
#include <stdlib.h>
#include <string>
#include <sstream>
#include <stdio.h>
#include <time.h>
```

```
class Room {

private:
	int room;
	int key;
	int door;
	int hall;
	std::vector<clingo_symbol_t> VCSymbols;
	int *key_on;
	int *door_on;
	int *bifurcacion_on;
	int sCount;
	char* baseprogram;
	char* cl_argv[4];
	int cl_argc;

public:
	Room(int r, int d, int h);
	~Room();
	static const char* baseprogram1;
	int getRoom();
	void setRoom(int r);
	int getHall();
	void setHall(int h);
	int getKey();
	void setKey(int k);
	int   getDoor();
	void setDoor(int d);
	int getKey_on(int id);
	void setKey_on(int value, int id);
	int getDoor_on(int id);
	void setDoor_on(int value, int id);
	int getBifurcacion_on(int id);
	void setBifurcacion_on(int value, int id);
	int getSCount();
	void setSCount();
	void addsymbolfromroom();
	void addsymbolfromkey();
	void addsymbolfromdoor();
	void addsymbolfrommove();
	void placeObjects();
	void prepareclingooptions(float rfreq, int seed);

private:
	static bool on_model(clingo_model_t* model, void* object, bool* goon);
	static bool on_statement(clingo_ast_statement_t const *stm,
clingo_program_builder *b);

};
```

```
*****************************************************************************
**********
*
* Room.cpp
*
*****************************************************************************
******
*
* AUTORES
*
*     Luis David Padilla Martín
*     José Ignacio Estévez Damas
*
* FECHA
*
*    26 de junio del 2018
*
* DESCRIPCION
*
*    Clase en c++ encargada de obtener las soluciones de nuestro puzle
*    mediante la utilización de la librería de Clingo.
*
*****************************************************************************
#include "room.h"

using namespace std;
const int NARGUMENTROOM = 1;a

const int NARGUMENTKEY = 1;
const int NARGUMENTDOOR = 1;
const int NARGUMENTMOVE = 1;

Room::Room(int r, int d, int h) {
      room = r;
      if (r + h > d) {
            key = d;
            door = d;
      }
      else {
            key = r - 1;
            door = r - 1;
      }
      hall = h;
      key_on = new int[d];
      door_on = new int[d];
      bifurcacion_on = new int[1];
```

```
	sCount = 1;
	std::stringstream ss;
	ss << "#const r=" << r << "." << endl << "#const k=" << d << "." << endl
<<"#const h=" << h <<"."<< endl <<baseprogram1;
	std::string s = ss.str();
	int len = s.length() + 1;
	baseprogram = new char[len];
	strcpy_s(baseprogram, (rsize_t)len, s.c_str());
	std::cout << baseprogram << std::endl;
	float rfreq = 0.50;
	srand(time(NULL));
	int seed = rand() % 10 + 1;
	prepareclingooptions(rfreq, seed);
}

Room::~Room() {
	delete[] key_on;
	delete[] door_on;
}

void Room::prepareclingooptions(float rfreq, int seed) {
	*cl_argv = new (char[100]);
	*(cl_argv + 1) = new (char[100]);
	*(cl_argv + 2) = new(char[100]);
	*(cl_argv + 3) = new (char[100]);
	*(cl_argv + 4) = new (char[100]);
	std::stringstream ss;
	ss << "-t";
	std::string s = ss.str();
	int length = s.length() + 1;
	strcpy_s(cl_argv[0], length + 1, s.c_str());
	std::stringstream ss2;
	ss2 << "2";
	std::string s2 = ss2.str();
	s2 = ss2.str();
	length = s2.length() + 1;
	strcpy_s(cl_argv[1], length, s2.c_str());
	std::string str_rfreq("--rand-freq=");
	str_rfreq.append(std::to_string(rfreq));
	strcpy_s(cl_argv[2], str_rfreq.length() + 1, str_rfreq.c_str());
	std::string str_seed("--seed=");
	str_seed.append(std::to_string(seed));
	strcpy_s(cl_argv[3], str_seed.length() + 1, str_seed.c_str());
	cl_argc = 4;
}

int Room::getRoom() {
	return room;
}
```

```
void Room::setRoom(int r) {
	room = r;
}

int Room::getHall() {
	return hall;
}

void Room::setHall(int h) {
	hall = h;
}

int Room::getKey() {
	return key;
}

void Room::setKey(int k) {
	key = k;
}

int Room::getDoor() {
	return door;
}

void Room::setDoor(int d) {
	door = d;
}

int Room::getKey_on(int id) {
	return key_on[id];
}

void Room::setKey_on(int value, int id) {
	key_on[id] = value;
}

int Room::getDoor_on(int id) {
	return door_on[id];
}

void Room::setDoor_on(int value, int id) {
	door_on[id] = value;
}

int Room::getBifurcacion_on(int id)
{
	return bifurcacion_on[id];
}
```

```
void Room::setBifurcacion_on(int value, int id)
{
	bifurcacion_on[id] = value;
}

int Room::getSCount() {
	return sCount;
}

void Room::setSCount() {
	sCount++;
}

void Room::addsymbolfromroom() {

	for (int i = 0; i < getRoom() + getHall(); i++) {
		clingo_symbol_t sroom;
		clingo_symbol_create_number(getSCount(), &sroom);
		clingo_symbol_t sroomh, sroom_args[NARGUMENTROOM];
		sroom_args[0] = sroom;

		if (!clingo_symbol_create_function("room", sroom_args,
NARGUMENTROOM, true, &sroomh))
			throw std::runtime_error{ "Clingo error: creating room
predicate" };

		VCSymbols.push_back(sroomh);
		setSCount();
	}
}

void Room::addsymbolfromkey() {

	for (int i = 0; i < getKey(); i++) {
		clingo_symbol_t skey, stakeable;
		clingo_symbol_create_number(getSCount(), &skey);
		clingo_symbol_create_number(getSCount(), &stakeable);
		clingo_symbol_t skeyh, skey_args[NARGUMENTKEY], stakeableh,
stakeable_args[NARGUMENTKEY];
		skey_args[0] = skey;
		stakeable_args[0] = stakeable;

		if (!clingo_symbol_create_function("key", skey_args, NARGUMENTKEY,
true, &skeyh))
			throw std::runtime_error{ "Clingo error: creating key
predicate" };
		if (!clingo_symbol_create_function("takeable", stakeable_args,
NARGUMENTKEY, true, &stakeableh))
			throw std::runtime_error{ "Clingo error: creating takeable
```

```
predicate" };


            VCSymbols.push_back(skeyh);
            VCSymbols.push_back(stakeableh);
            setSCount();
      }
}

void Room::addsymbolfrommove() {
      clingo_symbol_t smove;
      clingo_symbol_create_number(getRoom() + getHall() + getKey() + getKey(),
&smove);
      clingo_symbol_t smoveh, smove_args[NARGUMENTMOVE];
      smove_args[0] = smove;

      if (!clingo_symbol_create_function("moves", smove_args, NARGUMENTMOVE,
true, &smoveh))
            throw std::runtime_error{ "Clingo error: creating move predicate" };

      VCSymbols.push_back(smoveh);
}

void Room::addsymbolfromdoor() {

      for (int i = 0; i < getDoor(); i++) {
            clingo_symbol_t sdoor;
            clingo_symbol_create_number(getSCount(), &sdoor);
            clingo_symbol_t sdoorh, sdoor_args[NARGUMENTDOOR];
            sdoor_args[0] = sdoor;

            if (!clingo_symbol_create_function("door", sdoor_args,
NARGUMENTDOOR, true, &sdoorh))
                  throw std::runtime_error{ "Clingo error: creating door
predicate" };

            VCSymbols.push_back(sdoorh);
            setSCount();
      }
}

void Room::placeObjects() {


      clingo_control_t *ctl = nullptr;
      clingo_part_t parts[] = { { "base",NULL,0 } };
      clingo_solve_result_bitset_t solve_ret;

      if (!clingo_control_new((const char**)cl_argv, cl_argc, NULL, NULL, 20,
&ctl) != 0)
```

```
            throw std::runtime_error{ "Clingo error: creation of control failed"
};

      Room::VCSymbols.clear();
      addsymbolfromroom();
      addsymbolfromkey();
      addsymbolfromdoor();
      addsymbolfrommove();

      clingo_program_builder_t *builder;
      if (!clingo_control_program_builder(ctl, &builder))
            throw std::runtime_error{ "Clingo error: creating program
builder" };

      clingo_location_t location;
      location.begin_line = location.end_line = 1;
      location.begin_column = location.end_column = 1;
      location.begin_file = location.end_file = "<string>";

      clingo_ast_rule rule;
      clingo_ast_head_literal_t head;
      clingo_ast_body_literal_t* body;
      clingo_ast_literal_t literal, literal1;
      clingo_ast_statement stm;

      if (!clingo_program_builder_begin(builder))
            throw std::runtime_error{ "Clingo error: starting program
building" };

      if (!clingo_parse_program(baseprogram,
(clingo_ast_callback_t*)on_statement, builder, NULL, NULL, 20))
            throw std::runtime_error{ "Clingo error: parsing program" };

      for (auto sym : VCSymbols) {

            clingo_ast_term_t term;
            term.location = location;
            term.type = clingo_ast_term_type_symbol;
            term.symbol = sym;


            literal.location = location;
            literal.type = clingo_ast_literal_type_symbolic;
            literal.sign = clingo_ast_sign_none;
            literal.symbol = &term;


            head.location = location;
            head.type = clingo_ast_head_literal_type_literal;
```

```
            head.literal = &literal;

            rule.head = head;
            rule.body = nullptr;
            rule.size = 0;

            stm.location = location;
            stm.type = clingo_ast_statement_type_rule;
            stm.rule = &rule;

            if (!clingo_program_builder_add(builder, &stm))
                  throw std::runtime_error{ "Clingo error: adding statement" };


      }

      if (!clingo_program_builder_end(builder))
            throw std::runtime_error{ "Clingo error: ending program building" };

      if (!clingo_control_ground(ctl, parts, 1, NULL, NULL))
            throw std::runtime_error{ "Clingo error: grounding base program" };

      if (!clingo_control_solve(ctl, on_model, this, NULL, 0, &solve_ret))
            throw std::runtime_error{ "Clingo error: solving grounded model" };


      if (ctl) { clingo_control_free(ctl); }


}

bool Room::on_statement(clingo_ast_statement_t const *stm,
clingo_program_builder *b) {
      if (!clingo_program_builder_add(b, stm))
            throw std::runtime_error{ "Clingo error: adding statement to
program" };
      return true;
}

bool Room::on_model(clingo_model_t* model, void* object, bool* goon) {
      bool ret = true;
      *goon = true;
      std::cout << "Modelo encontrado" << std::endl;
      clingo_symbol_t *atoms = NULL;
      size_t atoms_n;

      if (!clingo_model_symbols_size(model, clingo_show_type_shown, &atoms_n))
            throw std::runtime_error{ "Clingo error: obtaining the number of
atoms" };
```

```
	try {
		atoms = new clingo_symbol_t[atoms_n];
	}
	catch (std::bad_alloc& exception){
		throw;
	}
	if (!clingo_model_symbols(model, clingo_show_type_shown, atoms, atoms_n))
		throw std::runtime_error{ "Clingo error: retrieving symbols from
obtained model" };
	clingo_signature_t sig_key_on, sig_door_on, sig_bifurcation_on;
	if (!clingo_signature_create("key_on", 3, true, &sig_key_on))
		throw std::runtime_error{ "Clingo error: creating signature for key
predicate" };
	if (!clingo_signature_create("door_on", 2, true, &sig_door_on))
		throw std::runtime_error{ "Clingo error: creating signature for door
predicate" };
	if(!clingo_signature_create("bifurcation", 1, true, &sig_bifurcation_on))
		throw std::runtime_error{ "Clingo error: creating signature for
bifurcation predicate" };
	size_t str_n = 0;
	char* str = nullptr;
	clingo_symbol_t const *it, *ie;
	for (it = atoms, ie = atoms + atoms_n; it != ie; ++it) {
		size_t n;
		char *str_new;
		if (!clingo_symbol_to_string_size(*it, &n))
			throw std::runtime_error{ "Clingo error: getting symbol string
size" };
		if (str_n<n) {
			if (!(str_new = (char*)realloc(str, sizeof(*str) * n)))
				throw std::runtime_error{ "Error getting memory for
string" };

			str = str_new;
			str_n = n;
		}
		if (!clingo_symbol_to_string(*it, str, n))
			throw std::runtime_error{ "Clingo error: assigning symbol to
string" };
		std::cout << "Symbol: " << str << std::endl;
		clingo_symbol_t stype;
		if (!(stype = clingo_symbol_type(*it)))
			throw std::runtime_error{ "Clingo error: getting symbol
type" };
		if (stype == clingo_symbol_type_function) {
			const char* sname;
			if (!clingo_symbol_name(*it, &sname))
				throw std::runtime_error{ "Clingo error: getting symbol
name" };
			clingo_signature_t symbolsignature;
```

```
                if (!clingo_signature_create(sname, 3, true,
&symbolsignature))
                        throw std::runtime_error{ "Clingo error: creating
expected signature" };
                clingo_signature_t symbolsign;
                if (!clingo_signature_create(sname, 2, true, &symbolsign))
                        throw std::runtime_error{ "Clingo error: creating
expected signature" };
                clingo_signature_t symbolsignb;
                if (!clingo_signature_create(sname, 1, true, &symbolsignb))
                        throw std::runtime_error{ "Clingo error: creating
expected signature" };
                if (clingo_signature_is_equal_to(sig_key_on, symbolsignature))
{
                        clingo_symbol_t const *args;
                        std::cout << "Es un key_on!" << std::endl;
                        size_t nargs;
                        if (!clingo_symbol_arguments(*it, &args, &nargs))
                                throw std::runtime_error{ "Clingo error: getting
symbol arguments" };
                        int roomId;
                        if (!clingo_symbol_number(*args, &roomId))
                                throw std::runtime_error{ "Clingo error: getting
obj room id from symbol" };
                        int keyId;
                        if (!clingo_symbol_number(*(args + 1), &keyId))
                                throw std::runtime_error{ "Clingo error: getting
obj key id from symbol" };
                        int move;
                        if (!clingo_symbol_number(*(args + 2), &move))
                                throw std::runtime_error{ "Clingo error: getting
obj move from symbol" };
                        if (move == 1) {
                                int pos = keyId - ((Room*)object)->getRoom();
                                pos = pos - ((Room*)object)->getHall();
                                pos = pos - 1;
                                std::cout << "Pos:" << pos << "KeyId:" << keyId <<
std::endl;
                                ((Room*)object)->setKey_on(roomId, pos);
                        }
                }
                if (clingo_signature_is_equal_to(sig_door_on, symbolsign)) {
                        std::cout << "Es un door_on!" << std::endl;
                        clingo_symbol_t const *args;
                        size_t nargs;
                        if (!clingo_symbol_arguments(*it, &args, &nargs))
                                throw std::runtime_error{ "Clingo error: getting
symbol arguments" };
                        int doorId;
                        if (!clingo_symbol_number(*args, &doorId))
                                throw std::runtime_error{ "Clingo error: getting
```

```
obj room id from symbol" };
                        int roomId;
                        if (!clingo_symbol_number(*(args + 1), &roomId))
                              throw std::runtime_error{ "Clingo error: getting
obj door id from symbol" };

                        int pos = roomId - ((Room*)object)->getRoom();
                        pos = pos - ((Room *)object)->getHall();
                        pos = pos - ((Room *)object)->getKey();
                        pos = pos - 1;
                        ((Room*)object)->setDoor_on(doorId, pos);
                  }
                  if (clingo_signature_is_equal_to(sig_bifurcation_on,
symbolsignb)) {
                        std::cout << "Es un bifurcation" << std::endl;
                        clingo_symbol_t const *args;
                        size_t nargs;
                        if (!clingo_symbol_arguments(*it, &args, &nargs))
                              throw std::runtime_error{ "Clingo error: getting
symbol arguments" };
                        int roomId;
                        if (!clingo_symbol_number(*args, &roomId))
                              throw std::runtime_error{ "Clingo error: getting
obj room id from symbol" };
                        ((Room*)object)->setBifurcacion_on(roomId, 0);
                  }


            }

      }
      delete atoms;

      return ret;

}

const char* Room::baseprogram1 = R"ZXSP(
secpath(1).
takeable(0).
doorproperty(op;cl).
movaction(lt;rt;ch).
accion(lt;rt;tk;re;uop;ch).
estado(1,0,1).
{bifurcation(R) : room(R),R<r} =1:-secpath(S).
{key_on(R,K,1) : room(R)}=1:- key(K).
jumproom(r+1):-h>0.
jumproom(R):-bifurcation(R).
{door_on(R,D):room(R)}=1:- door(D).
doorstate(D,cl,1) :- door(D).
:- door_on(R,D1),door_on(R,D2), D1!=D2.
:- door_on(1,D).
```

```
:- key_on(R1,K,1), door_on(R2,D), (D-k)=K, bifurcation(R3), R1<=R3, R2<=R3,
R1>=R2.
:- key_on(R1,K,1), door_on(R2,D), (D-k)=K, bifurcation(R3), R1>R3, R1<=r, R2>R3,
R2<=r, R1>=R2.
:- key_on(R1,K,1), door_on(R2,D), (D-k)=K, R1>r, R2>r, R1>=R2.
:- key_on(r,K,1).
{ actua(A,K,T) : accion(A), takeable(K) }<=1 :- T=1..M-1,moves(M).
:- actua(A,K,T), movaction(A),K>0.
:- actua(_,_,T), not actua(_,_,T-1),T>1.
:- actua(lt,_,T), estado(1,_,T).
:- actua(lt,_,T), estado(r+1,_,T).
:- actua(rt,_,T), estado(r,_,T).
:- actua(rt,_,T), estado(r+h,_,T).
:- actua(rt,_,T), estado(R,_,T), doorontherightclosed(R,T).
:- actua(ch,_,T), estado(R,_,T), bifurcation(R), dooronbifurcationclosed(T).
:- actua(ch,_,T), not jumproom(R), estado(R,_,T).
:- actua(tk,0,T).
:- actua(re,0,T).
:- actua(uop,K,T), not key(K).
:- actua(uop,K,T), estado(R,K1,T), K!=K1.
nextroom(R1,R2) :- room(R1), room(R2), not bifurcation(R1), R2=R1+1.
nextroom(R1,r+1) :- bifurcation(R1).
nextdoor(R,R1,D) :- door_on(R1,D), nextroom(R,R1), room(R), room(R1), door(D).
:- actua(uop,K,T), estado(R,K,T), {nextdoor(R,_,D) : D=K+k}=0.
:- actua(re,K1,T),estado(_,K2,T),K1!=K2.
:- actua(tk,K1,T),not key_on(R,K1,T),estado(R,_,T).
:- actua(tk,_,T), estado(_,K,T),K>0.
:- estado(R,K1,T),actua(uop,K2,T),K1!=K2.
:- estado(R,_,T),not actua(_,_,T-1),T>1.
:- key_on(_,_,T),not actua(_,_,T-1),T>1.
:- doorstate(_,_,T),not actua(_,_,T-1),T>1.
:- not estado(r,_,_).
:- estado(r,_,T), actua(_,_,T).
doorontherightclosed(R,T):-
doorstate(D,cl,T),door_on(R+1,D),estado(R,_,T),T=1..M,moves(M).
dooronbifurcationclosed(T):-doorstate(D,cl,T),door_on(r+1,D),T=1..M,moves(M).
estado(R,K,T) :- estado(R-1,K,T-1),  actua(rt,0,T-1),T=1..M,moves(M).
estado(r+1,K,T) :- bifurcation(R), estado(R,K,T-1),actua(ch,0,T-
1),T=1..M,moves(M).
estado(R,K,T) :- bifurcation(R), estado(r+1,K,T-1),actua(ch,0,T-
1),T=1..M,moves(M).
key_on(R,K,T):-key_on(R,K,T-1),actua(ch,_,T-1).
doorstate(D,S,T):-doorstate(D,S,T-1),actua(ch,_,T-1).
key_on(R,K,T):-key_on(R,K,T-1),actua(rt,_,T-1).
doorstate(D,S,T):-doorstate(D,S,T-1), actua(rt,_,T-1).
estado(R,K,T) :- estado(R+1,K,T-1), R>=1,actua(lt,0,T-1),T=1..M,moves(M).
key_on(R,K,T):-key_on(R,K,T-1),actua(lt,_,T-1).
doorstate(D,S,T):-doorstate(D,S,T-1), actua(lt,_,T-1).
estado(R,K,T) :- estado(R,0,T-1), actua(tk,K,T-1),T=1..M,moves(M).
:- key_on(_,K,T),estado(R,_,T-1),actua(tk,K,T-1).
key_on(R,K,T):-key_on(R,K,T-1),actua(tk,K1,T-1), K!=K1.
doorstate(D,S,T):-doorstate(D,S,T-1),actua(tk,_,T-1).
estado(R,0,T) :- estado(R,K,T-1), actua(re,K,T-1),T=1..M,moves(M).
key_on(R,K,T) :- estado(R,_,T-1), actua(re,K,T-1),T=1..M,moves(M).
doorstate(D,S,T):-doorstate(D,S,T-1),actua(re,_,T-1).
doorstate(D,op,T):- nextdoor(R,_,D),estado(R,K,T-1),actua(uop,K,T-
1),T=1..M,moves(M).
estado(R,0,T):- nextdoor(R,_,D),estado(R,K,T-1),actua(uop,K,T-
1),T=1..M,moves(M).
key_on(R,K,T):-key_on(R,K,T-1),actua(uop,K1,T-1),K!=K1.
doorstate(D,C,T):-doorstate(D,C,T-1),actua(uop,K,T-1),K!=D-k.
```

```
keyonsec(K):-key_on(Rd,K,1),Rd>r,room(Rd),key(K).
dooronmain(D):-door_on(Rd,D),Rd<=r,room(Rd),door(D).
doormainkeysec(D,K):- dooronmain(D), keyonsec(K), D=K+k, door(D), key(K).
:- {keyonsec(K) : key(K)} = 0.
:- {dooronmain(D) : door(D)} = 0.
:- {doormainkeysec(D,K) : door(D), key(K)} =0.
)ZXSP";


**********************************************************************
**********
*
* dllClin.cpp
*
**********************************************************************
******
*
* AUTORES
*
*     Luis David Padilla Martín
*     José Ignacio Estévez Damas
*
* FECHA
*
*    28 de junio del 2018
*
* DESCRIPCION
*
*    Código que genera una dll que permite ejecutar código clingo
*
**********************************************************************
#include "Room.h"
extern  "C" __declspec(dllexport) void  get_result(int *key, int *door, int
*bifurcation, int r, int d, int h) {
      Room room(r, d, h);
      room.placeObjects();

      for (int i = 0; i < d; i++) {
            key[i] = room.getKey_on(i);
      }
      for (int j = 0; j < d; j++) {
            door[j] = room.getDoor_on(j);
      }
      for (int z = 0; z < 1; z++) {
            bifurcation[z] = room.getBifurcacion_on(z);

      }
}
```