Facultad de Informática

Trabajo fin de grado
Grado en Ingeniería Informática

Mención en Computación

Generación de Escenarios de un Videojuego 2D mediante Programación Lógica

Autor: Rafael Alcalde Azpiazu
Director: José Pedro Cabalar Fernández

A Coruña, 25 de julio de 2018

Resumen

Abstract

Palabras clave

Keyword one, keyword two, etc.

Índice general

Capítulo 1

Introducción

Capítulo 2

Contexto

2.1. Juegos de estrategia por turnos

También conocidos como *turn-based strategy* (TBS), son juegos donde el tiempo de juego no transcurre de forma continua, si no que se divide en partes bien definidas llamadas turnos. En un turno, un jugador dispone de un periodo de análisis antes de realizar una acción, lo cual realiza un salto de turno al siguiente jugador. Cuando todos los jugadores han terminado su turno, se comienza una nueva ronda.

Existen varios géneros dentro de la estrategia por turnos:

- **Juegos clásicos de tablero**: Son los primeros juegos de este tipo, padres del resto de géneros. La idea es controlar la acción del propio juego mediante turnos. En este género tenemos como ejemplos el ajedrez, el Go o el Revesi.
- **TBS and RTT** (Estrategia por turnos y táctica en tiempo real): Añade un componente en tiempo real al permitir que en cada turno se generen batallas en tiempo real contra los oponentes. Un ejemplo de este género es la saga de videojuegos Total War.
- **Man-to-man**: Se basa en el uso de unidades muy pequeñas, en donde controlamos cada uno de los individuos de nuestra partida. En esta categoría está la saga de videojuegos XCOM.
- **TRPG** (RPG táctico): También conocidos en Japón como Simulation RPG (SRPG), se basan en incorporar elementos de estrategia por turnos al combate clásico de los RPG. En la parte de tablero, un ejemplo clásico es Dragones y Mazmorras (*Dungeons & Dragons*), mientras que en el caso de los videojuegos, los más conocidos son Darkest Dungeon, algunas entregas de Megami Tensei como Shin Megami Tensei y Persona, la saga Disgaea o Fire Emblem.
- **4X**: Llamado así por la idea principal del género (*eXplore, eXpand, eXploit and eXterminate*, en castellano "Explora, expande, explota y extermina"). Se basan en el control de un imperio, profundizando en el

desarrollo económico, tecnológico y militar, con la idea final de que crear a largo tiempo un imperio sostenible. Existe una profunda complejidad debido a la cantidad de elementos que hay que tener en cuenta. Un ejemplo en tablero de este género es Risk, mientras que en videojuegos destaca la saga Master of Orion y Sid Meier's: Civilization [ver sección 2.1.1].

2.1.1. Sid Meier's: Civilization

Como ya comenté, Civilization es un videojuego de estrategia por turnos que cae dentro del género X4, lanzado por primera ver en 1991 por el programador y diseñador Sid Meier. El objetivo del juego es construir una civilización y avanzarla desde la prehistoria y hasta un futuro cercano. En cada turno, el jugador puede mover unidades por el mapa, construir o mejorar ciudades y unidades, y realizar negociaciones con el resto de jugadores. Una vez se realiza una ronda, el juego avanza unos años en la historia. Actualmente existen seis títulos principales (el último, Civilization VI, lanzado en 2016) y varios spin-off que se centran en potenciar distintos aspectos de los títulos principales.

Al empezar la partida, el jugador se encuentra en un mapa basado en casillas generado proceduralmente, el cual contiene distintos biomas (praderas, desierto, montañas, ríos, lagos, etc...) con una serie de recursos (vino, caballos, trigo, fuel, carbón, etc...). La posición inicial normalmente es aleatoria, y el jugador puede controlar con una serie de colonos. En el resto del mapa cercano a los colonos, existe una niebla de guerra para evitar que el jugador pueda ver que hay cerca de el.

Si el jugador decide fundar una ciudad con los colonos, esta empieza a general recursos que pueden ser los propios del mapa que hay en las casillas cercanas a la ciudad o simplemente mano de obra, cultura, ciencia o dinero. La cantidad de recursos que se puede generar en una ciudad se basa en como de grande es esta. Con estos recursos, se puede producir nuevas unidades como colonos, guerreros, distintos elementos de guerra como barcos, aviones y maquinaria pesada, o unidades con las que mantener comercio con otras civilizaciones.

Con la ciencia y la cultura se puede aumentar el progreso tecnológico, el cual se expresa en un árbol de tecnologías a desarrollar. El jugador puede ir eligiendo que tecnología se desarrolla en todo momento. Con respecto al dinero se puede usar para mejorar a las ciudades y unidades, crear carreteras, aumentar el ritmo de producción o crear regalos para otras civilizaciones.

En las ciudades se pueden mejorar al crear nuevos edificios, que ayudan a mejorar la producción y aumentar la población, como graneros, universidades, distritos comerciales o librerías. También pueden servir de fortaleza, como murallas, castillos de guardia o escuelas militares. Por último, existen edificios únicos que son las maravillas del mundo, que ayudan a aumentar el ritmo ge-

neral de una civilización y que dan bonus al primero que las complete, de ahí que solo se pueda crear una vez en toda la partida.

Existen múltiples victorias por las que puede ganar un jugador: la victoria por conquista ocurre cuando el jugador toma el control o añade al imperio a todas las ciudades capitales del resto de civilizaciones; la victoria diplomática se obtiene cuando el jugador construye las Naciones Unidas al entablar amistades con todas las civilizaciones; la victoria tecnológica se alcanza cuando el jugador construye una nave para llegar a Alfa Centauri; por último, la victoria cultural se obtiene al acumular la suficiente cultura con respecto a las otras civilizaciones y construir los edificios necesarios para guiar al resto a conseguir un estado utópico con alta cultura.

Debido al éxito de las entregas de Civilization y las diputas legales sobre el derecho de explotación entre Activision y Avalon Hills, se han creado distintos proyectos y secuelas que descienden del oficial, como la saga Call to Power o los proyectos Openciv y Freeciv [ver sección 2.1.2].

2.1.2. Proyecto Freeciv

Freeciv[1][2] es un videojuego open source basado en la serie de Sid Meier's Civilization, sobre todo en Civilization II. Fue creado por tres estudiantes del departamento de computación de la universidad de Aarhus debido al bajo rendimiento de Openciv. Basándose en la arquitectura de X11, escribieron un sencillo cliente en donde se ejecutaría una prueba de concepto, la cual tuvo una enorme acogida y rápidamente se creó una comunidad open source, la cual pasó a gestionar el proyecto para continuar expandiendo el juego al incluir elementos online, mejoras en la interfaz gráfica, mejor rendimiento y nuevos elementos al juego.

El juego está escrito en C y tiene compatibilidad con el estandar POSIX para ser lo más portable posible. Contiene un intérprete de Lua [ver seccion 2.2.2] que permite cargar scripts para expandir el juego y desde el 2006 IANA asignó el puerto TCP/UDP 5556 a Freeciv [1].

2.2. Tecnologías

Debido a las exigencias a la hora de desarrollar el proyecto, se ha optado por elegir un lenguaje de programación lógico sobre el que realizar la base declarativa del proyecto, ya que nos permitirá expresar las reglas de generación del mapa de forma matemática. También se ha escogido un segundo lenguaje multipropósito que nos servirá como soporte para crear un entorno gráfico con el que poder editar, guardar y cargar los mapas creados.

[1] http://www.freeciv.org
[2] https://github.com/freeciv

Figura 2.1: Ejemplo de ejecución de ASP

2.2.1. Answer Set Programming

Answer Set Programming (ASP) es un paradigma enfocado a la resolución declarativa de problemas difíciles, combinando un lenguaje simple con el que modelar los problemas lógicos y herramientas de alto rendimiento para la resolución de estos. ASP está basado en modelos estables [2], que usa para definir la semántica declarativa en los programas lógicos, y lógica no monótono [3], que añade razonamiento por defecto. Con esto, ASP permite resolver problemas *NP-hard* de forma uniforme.

Mediante estos mecanismos, los problemas lógicos se reducen al cómputo de los modelos estables de un programa lógico dado, lo cual se hace mediante herramientas llamadas *solvers*, que se encargan de utilizar diferentes técnicas para la expansión de lo que se llama Forma Normal Conjuntiva (FNC), que es una conjunción de clausulas:

$$(p \vee v) \wedge (\neg p \vee v) \wedge q \tag{2.1}$$

Para ello, se necesita que los programas lógicos estén expresados con variables libres. Como la forma de expresar un programa lógico en ASP es mediante un lenguaje de alto nivel con ciudadanos de primer orden, muchas de las variables están ligadas. Es por eso que, antes de resolver el programa, se usa unas herramientas llamadas *grounders*, que permiten transformar el programa a su equivalente con variables libres.

Unas de las herramientas más importantes de ASP son la de Potassco[3], la cual son un conjunto de herramientas para ASP desarrolladas en la Universidad de Postdam. Contienen las herramientas fundamentales como un grounder llamado *gringo*; un solver que es *clasp*; y una herramienta que aguntina todo el sistema ASP, *clingo*. Así mismo, añade más funcionalidades al lenguaje ASP, como puede ser al resolución iterativa, debido a que soporta otros lenguajes como Lua [ver sección 2.2.2] o Python, que pueden interactuar con el programa escrito en ASP mediante la API de clingo.

2.2.2. Lua

Lua[4] [4] (del portugués Luna), es un lenguaje de programación de propósito general, de scripting y multiparadigma (procedural, orientado a objetos basado en prototipos, funcional y *data-driven*). Fue pensado para ser embebido en

[3]http://potassco.org
[4]http://www.lua.org

cualquier aplicación independientemente de la plataforma sobre la que se ejecuta. Su intérprete esté escrito en ANSI C y contiene una API en C muy simple.

Uno de los puntos fuertes de Lua es que permite construir nuevos tipos en base a arrays asociativos, que también permiten extender la semántica del lenguaje al aplicar metadatos a estos. A parte de esto, también facilita la tarea al programador al tener un recolector incremental de basura que se encarga de gestionar automáticamente la memoria. Además, Lua es de tipado dinámico, y se ejecuta sobre un intérprete bytecode en una máquina virtual basada en registros, ejecutándose más rápido que otros lenguajes ejecutados sobre una máquina virtual basada en *stack*, como puede ser Java.

A parte de esto, existen otras implementaciones y dialectos basados en Lua, como LuaJIT[5], que es una implementación de la máquina virtual de Lua que compilar en tiempo de ejecución y no antes de ejecutarse; y MoonScript[6], un lenguaje de scripting basado en Python que extiende el lenguaje de Lua al añadir orientación a objetos basados en clases, una forma más eficiente de lenguaje funcional y nuevas estructuras y elementos que no contenía Lua.

Exiten otras implementaciones que lo único en lo que se diferencian de la implementación estándar es que expanden las funciones básicas o implementan bibliotecas por defecto. A estas implementaciones se llaman *Lua Players* o *Lua Engines*, que principalmente son destinadas para la creación de videojuegos, ya que añaden la implementación de las bibliotecas gráficas para una plataforma concreta, como pueden ser SDL[7] o OpenGL[8]. Los más destacados son los intérpretes para consolas de Sony y Nintendo, y los intérpretes para ordenador y móviles como Corona, Cocos-2D, Moai y LÖVE [ver sección 2.3.3].

2.3. Herramientas

2.3.1. Atom

2.3.2. Git

2.3.3. LÖVE

LÖVE[9] (o también conocido como Love2D)

[5] http://luajit.org/luajit.html
[6] https://moonscript.org
[7] https://www.libsdl.org
[8] https://www.opengl.org
[9] https://love2d.org

Capítulo 3

Trabajo desarrollado

Capítulo 4

Evaluación

Capítulo 5

Conclusiones

Índice de tablas

Índice de figuras

Bibliografía

[1] IANA, "Service name and transport protocol port number registry," Jan 2006. [Online]. Available: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml?search=5556

[2] M. Gelfond and V. Lifschitz, "The stable model semantics for logic programming." in *Proceedings of International Logic Programming Conference and Symposium*, vol. 88, 1988, pp. 1070–1080.

[3] S. Hanks and D. McDermott, "Nonmonotonic logic and temporal projection," *Artificial Intelligence*, vol. 33, no. 3, pp. 379 – 412, 1987. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0004370287900439

[4] R. Ierusalimschy, *Programming in Lua, Fourth Edition.* Lua.Org, 2016.