# *Modding* con *Dota 2*

PID_00247902

David León Molero

Tiempo mínimo de dedicación recomendado: 3 horas

Universitat Oberta de Catalunya

# Índice

# Introducción

### ¿Qué es un *mod*?

Los *mods* o modificaciones son alteraciones sobre videojuegos hechas por usuarios y fanes. Hay distintos tipos de *mods*, desde los que actúan como «trucos» para añadir más vida al personaje, hasta los que modifican el arte del videojuego o incluso construyen nuevos modos de juego.

Originalmente, los *mods* surgieron como maneras de alargar la vida útil de un juego mediante la incorporación de nuevos niveles y enemigos, o para cambiar su temática a otra más acorde con los gustos del *modder*.

Los videojuegos de id Sofware fueron los precursores de la cultura del *modding*. *Wolfenstein 3D*, *Doom* o *Quake* cuentan con innumerables *mods* públicos en la web, y su influencia en la popularidad de dichos juegos fue tal que sus creadores dieron el visto bueno a la modificación de estos videojuegos por parte del público e incluso crearon herramientas para hacerla más fácil.

Imagen 1.

Con *Half-Life*, *Warcraft 3* y las comunidades sociales en línea llegó el auge del *modding* para el público general. En esta época dorada, surgieron juegos tan míticos como *Counter-Strike* o *Dota*, con los que se inventaron nuevos géneros y maneras de jugar a partir de *mods*.

Hoy en día el *modding* forma parte de la cultura del jugador de PC, y plataformas como Steam Workshop otorgan herramientas para que desarrolladores y usuarios puedan colaborar generando nuevo contenido (*user generated content*).

En este módulo, analizaremos las herramientas de *modding* de *Dota 2* y las utilizaremos para crear nuestro propio *mod*.

**Clasificación**

Los *mods* se pueden clasificar en:

- Parches no oficiales

Los propios fanes crean una actualización no oficial para arreglar *bugs* del juego, mejorar el *gameplay* o extender funcionalidades técnicas (como soporte para monitores de alta resolución). Los parches no oficiales suelen ser habituales en juegos antiguos que ya no tienen soporte oficial por parte del desarrollador. También existen los parches que simplemente actualizan contenido del juego para ponerlo al día, algo muy típico en juegos de deporte, donde los jugadores crean *mods* anuales para actualizar las equipaciones y la lista de jugadores de cada equipo de fútbol, básquet, etc. del juego de acuerdo con las alineaciones de las ligas en curso.

- *Mods* artísticos

Los *mods* más típicos son aquellos que simplemente cambian el aspecto de componentes específicos del juego. Cambiar los *posts* FX de pantalla, añadir texturas de alta resolución, modificar el aspecto de los enemigos o los modelos de las armas..., las posibilidades son infinitas. Juegos como *Skyrim* se nutren principalmente de una comunidad que trabaja en este tipo de *mods*.

Imagen 2.

- *Add-ons*

Los *add-ons* son *mods* que modifican o añaden nuevos elementos de juego. Puede tratarse de nuevas armas, más variedad de enemigos, mayor biblioteca de conjuros o incluso soporte para nuevos idiomas o nuevos modos de juego o mapas. *Dota* comenzó originalmente como un nuevo mapa para *Warcraft 3*.

- *Remakes* u *overhauls*

A veces la comunidad coge un juego y rehace todos sus componentes manteniendo el juego que hay detrás. Sería un buen ejemplo de ello *Black Mesa*, resultado de la modificación que la comunidad hizo de *Half-Life 1*, modificación basada en gráficos y mejoras de *gameplay* actualizados a nuestra época, pero en la que la historia, los niveles y el juego en sí se dejaron intactos.

- Conversiones totales

Una conversión total modifica todos los aspectos del juego y lo convierte, por definición, en uno totalmente nuevo, eso sí, aprovechando el motor y muchos otros elementos del original. *Counter-Strike* es una conversión total de *Half-Life 1* que transformaba el horror y la ciencia ficción del original en un juego multijugador de policías contra terroristas. *DayZ* convirtió un simulador de guerra en un juego de supervivencia en un apocalipsis zombi.

Imagen 3.

### *Dota 2*

*Dota* (*Defense of the Ancients*) fue un *mod* de *Warcraft 3* que convertía el juego de estrategia en tiempo real (RTS) en un tipo de juego totalmente distinto, lo que hoy se conoce como MOBA (*multiplayer online battle arena*).

En los MOBA, dos equipos de jugadores luchan para destruir la base del otro equipo, con la peculiaridad de que los jugadores no controlan a una facción o ejército de unidades; solo controlan a un único héroe con habilidades especiales, mientras que las unidades básicas son controladas por la IA.

Imagen 4.


*Dota* fue el precursor del género, *League of Legends* lo llevó al público general, y Valve desarrolló *Dota 2* como su apuesta para intentar desarrollar el MOBA más popular. Como baza principal, comparado con sus competidores, *Dota 2* incluye una serie de herramientas para permitir a sus jugadores crear modificaciones y nuevos mapas y tipos de juego para *Dota 2*. Además, utiliza Steam Workshop como nexo de unión entre creadores y jugadores.

En este módulo, aprenderemos a utilizar las herramientas de *modding* de *Dota 2* para crear nuestro propio *mod*.

# 1. Herramientas de *modding* de *Dota 2*

## 1.1. Estructura

Se pueden generar distintos tipos de *mods* para *Dota 2*. Para cada uno de ellos tenemos herramientas distintas:

- **Diseño de nivel**. Se utiliza la herramienta llamada Hammer.
- ***Scripting***. Principalmente se modifica código en lenguaje Lua.
- ***Mods* artísticos**. Se pueden importar nuevos modelos (*meshes*) o texturas.
- **Edición de sistemas de partículas**.Se utiliza Particle Editor Tool (PET) o se editan sistemas de partículas existentes.
- **Modificación de interfaz**. Se hace mediante el *script* Panorama o modificando la UI existente de *Dota 2*.

En esta unidad, nos centraremos en el diseño de nivel con Hammer.

Para más información, podéis consultar la wiki oficial de Valve:

https://developer.valvesoftware.com/wiki/Dota_2_Workshop_Tools

## 1.2. Configuración del proyecto

Para instalar las herramientas de *modding*, seguiremos los siguientes pasos:

**1)** Instalar Steam y crear una cuenta en esta plataforma en el caso de que no la tengamos.
**2)** Descargarnos e instalar *Dota 2* desde Steam.
**3)** Hacer clic derecho en *Dota 2* en la Biblioteca y hacer clic en View Downloadable Content.
**4)** Marcar la casilla Dota 2 Workshop Tools DLC y hacer clic en Cerrar. Las herramientas de *modding* se empezarán a instalar.

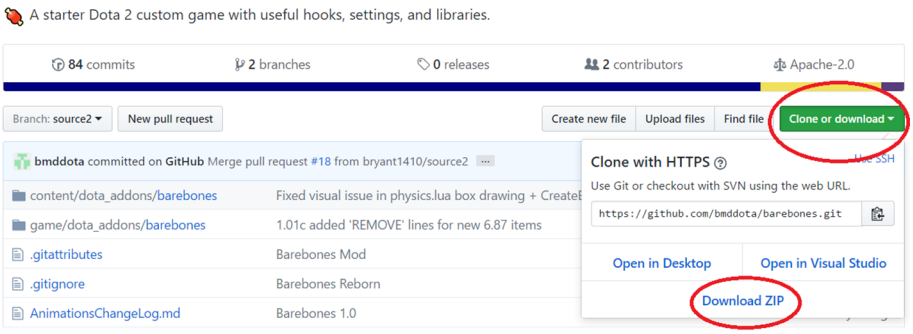
## 1.3. Utilizando la plantilla Barebones

Barebones es una plantilla (*template*) hecha por la comunidad que proporciona una base limpia desde la que comenzar un *mod*. También incluye una serie de *scripts* en Lua que facilitan las acciones por *scripting* más comunes en un *mod*.

Podemos encontrar Barebones en el repositorio oficial: https://github.com/bmddota/barebones

Descargamos todo el repositorio en .zip mediante Clone or download:

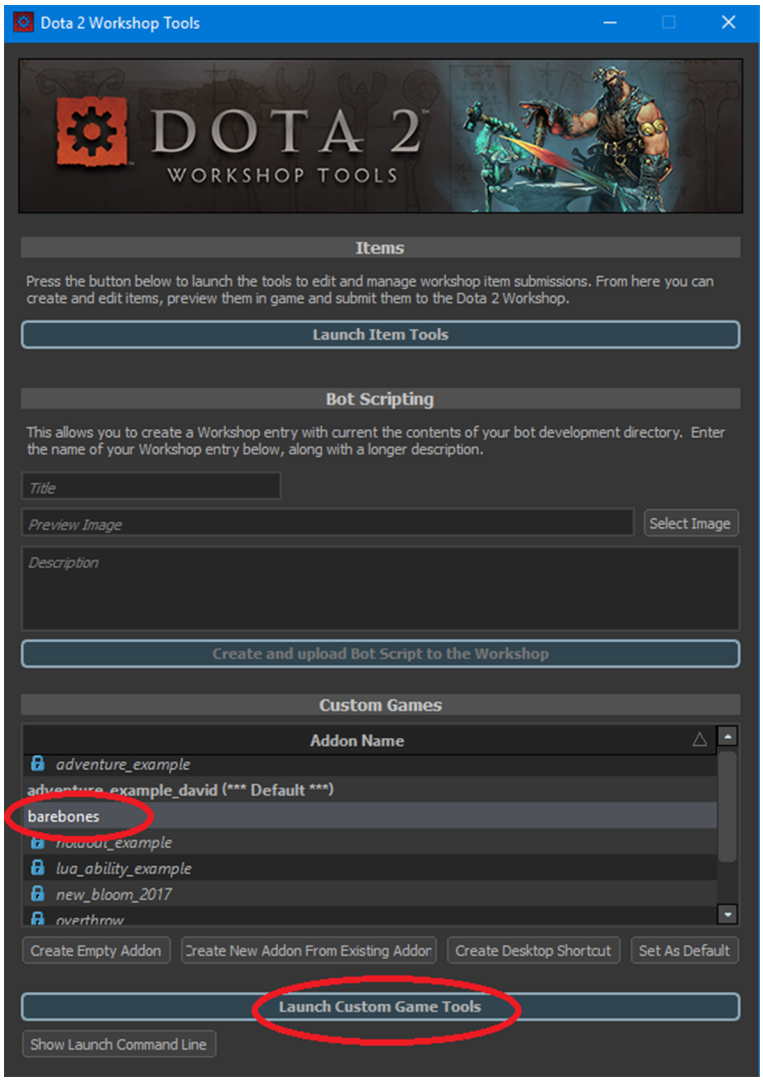
Imagen 5.




Luego seguimos las instrucciones de instalación, por las que debemos copiar las carpetas *content* y *game* dentro de la carpeta \SteamApps\common\dota 2 beta, que es la que contiene la instalación de *Dota 2*. Por lo general, esta se encuentra en C:\Archivos de Programa\Steam, pero variará según el equipo.

Con Barebones instalado, arrancamos las herramientas de *Dota 2* desde Steam haciendo clic derecho en Dota 2 y seleccionando Launch DOTA 2 - Tools. En el lanzador que aparece, encontraremos *barebones* como *add-on*, el cual arrancaremos. Si queremos, podemos hacer clic en Set As Default para establecerlo como *mod* por defecto para el futuro.

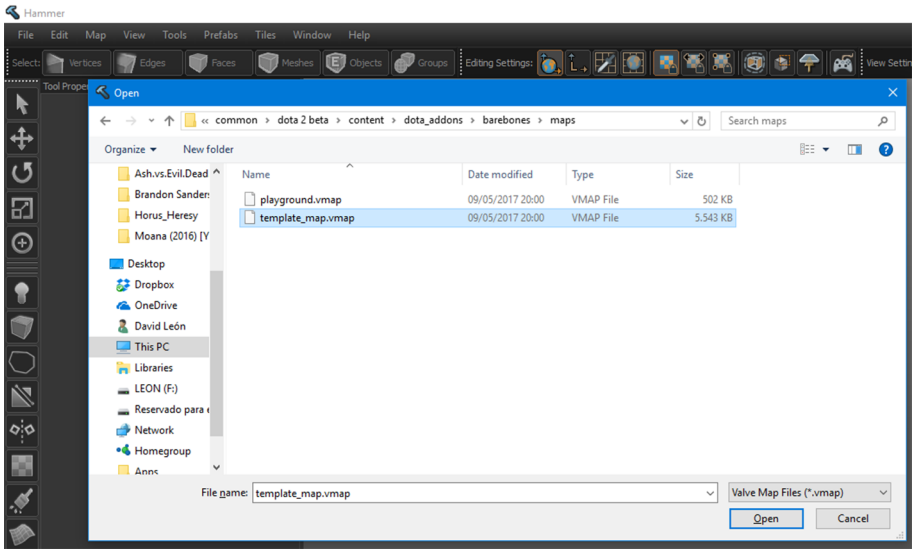
Imagen 6.




Aparecerá una ventana llamada Asset Browser. Hacemos clic en la rueda de la esquina superior derecha: Tools -> Hammer (Map Editor).

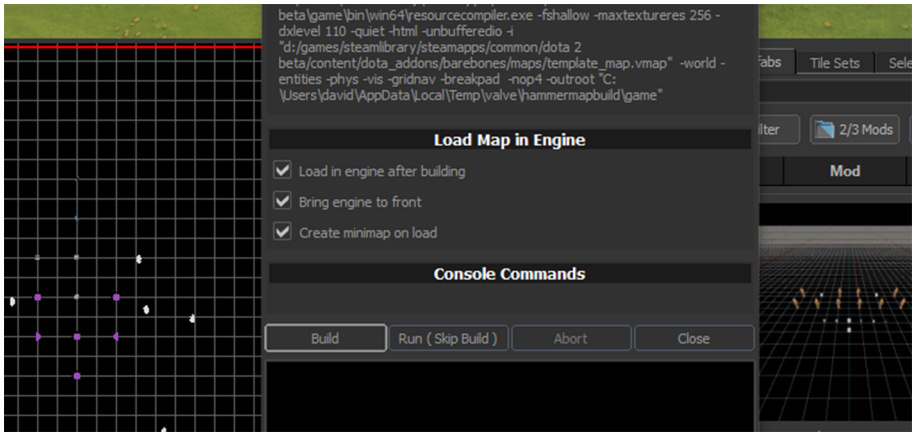
Ya en el editor de mapas Hammer, vamos a File -> Open y abrimos template_map.vmap, situado dentro de la carpeta barebones/maps:

Imagen 7.




Ya tenemos un mapa básico con el que empezar a trabajar. Si presionamos F9, se compilará el mapa y podremos empezar a jugarlo directamente en *Dota 2*.

Imagen 8.




Para no estar trabajando sobre la plantilla Barebones, podemos ir a File -> Save As y guardarnos este mapa en un nuevo fichero antes de empezar a trabajar.

# 2. Aprendiendo Hammer

## 2.1. Tile Editor

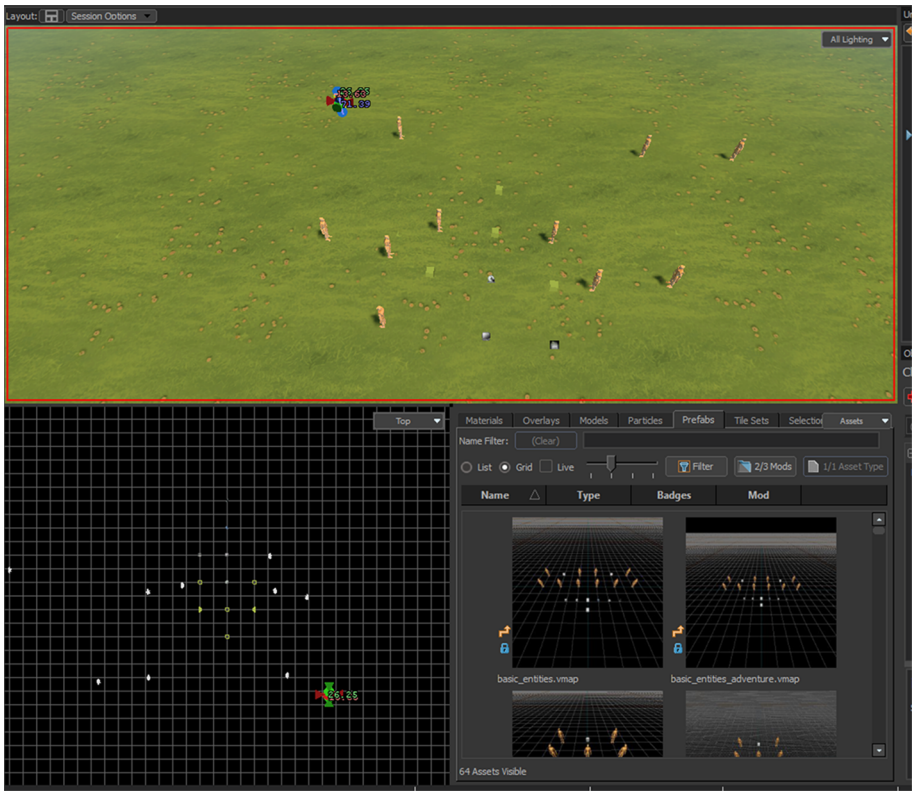
Hammer cuenta con varios tipos de herramientas. Principalmente son las siguientes:

1) Herramientas de creación y diseño del terreno. En esta unidad, nos centraremos en Tile Editor.
2) Herramientas de *mesh* para la creación y el modelado de *meshes*. Usualmente, es más cómodo utilizar Maya u otro programa especializado para crear modelos que luego se pueden importar a Hammer.
3) Herramientas de *entity management* y *scripting* para la gestión de elementos que afectan al *gameplay*. Son similares a los *prefabs* en Unity 3D.

Antes de nada crearemos nuestro primer terreno con Hammer desde cero.

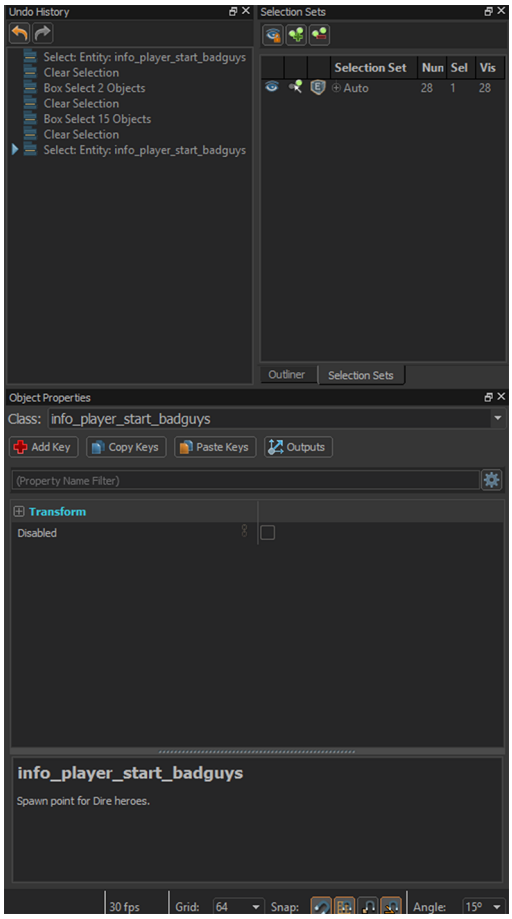
Si observamos la interfaz de Hammer, nos encontramos que en la parte central tenemos un *viewport* principal, otro *viewport* en la zona inferior y un explorador de *assets* del proyecto.

Imagen 9.



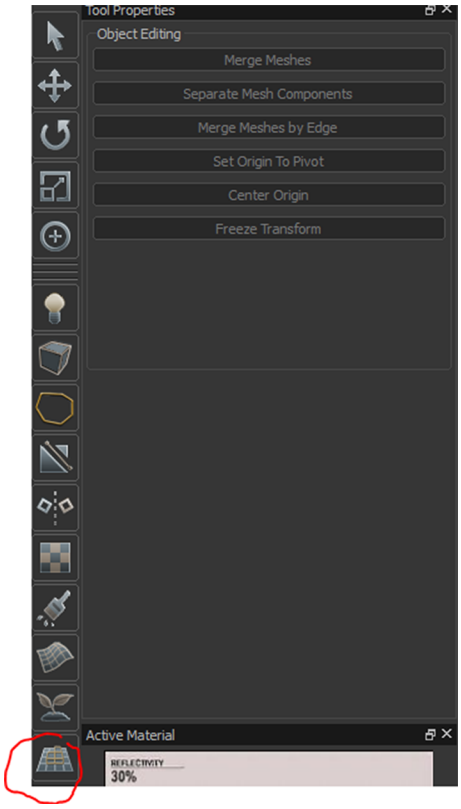
A la derecha tenemos un histórico de acciones, una lista de objetos en escena (en Outliner) y una ventana de propiedades de la selección actual.

Imagen 10.




En la zona izquierda, encontramos lo que más nos interesa actualmente. Primero están las herramientas para controlar nuestra selección (seleccionar, trasladar, rotar, escalar y modificar el pivote), muy similares a las herramientas básicas de Unity 3D. Luego nos encontramos con una serie de herramientas para gestionar el mapa, ya sea añadiendo *entities* (*prefabs*), creando nuevas *meshes* o, en el último botón, modificando el terreno con Tile Editor.

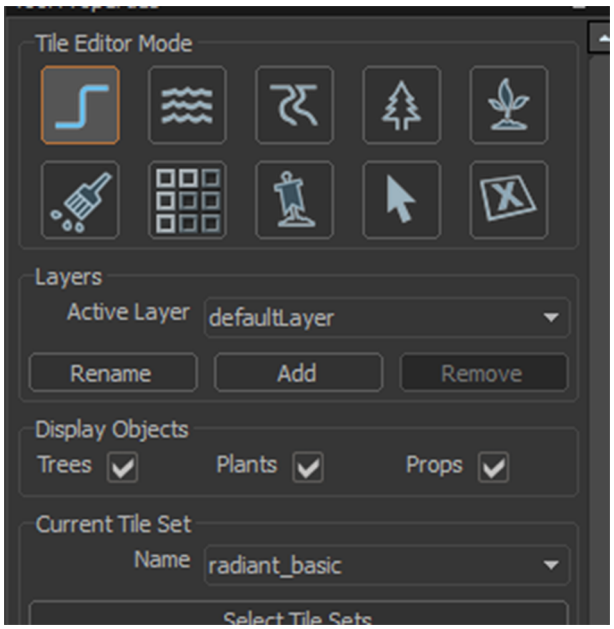
Imagen 11.




Antes de empezar a modificar el terreno, sería buena idea jugar y experimentar un rato con Hammer para familiarizarse con los controles y las herramientas. Aprender a controlar la cámara por el *viewport* será una habilidad fundamental que nos facilitará la vida durante el resto de la unidad.

Si hacemos clic en el último icono, aparecerán las herramientas de edición del terreno.

Imagen 12.



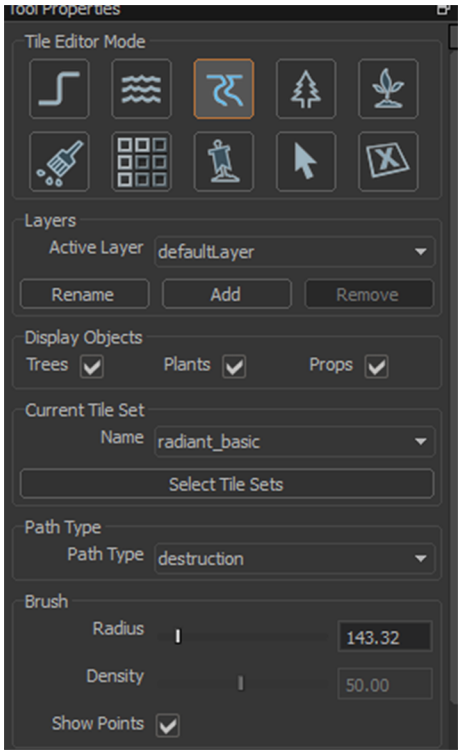
En la parte de abajo, podemos ver cómo el Current Tile Set es radiant_basic. Un *tile set* es el conjunto de piezas o el abanico de *props* y texturas con el que pintaremos nuestro mapa o nivel. Sería el equivalente a la paleta de colores de un pintor. Cambiando el *tile set*, tendríamos un conjunto de piezas diferentes con las que crear nuestro mapa.

Más abajo, encontramos los controles de teclado y ratón para poder utilizar las herramientas de edición del terreno.

En orden, disponemos de las siguientes herramientas:

- **Pintado de terreno**. Con esta herramienta podemos crear unos límites para el nivel en forma de riscos y acantilados. También podemos hacer una topografía más interesante para el interior del nivel. Si miramos los controles en las propiedades de la *tool*, vemos que con clic izquierdo elevamos el terreno; con *control* izquierdo + clic izquierdo lo rebajamos, y con clic central + movimiento del ratón a los lados, cambiamos el tamaño del pincel.
- **Pintado de agua**. Esta herramienta sirve para crear ríos y lagos. Este tipo de superficies serán infranqueables a no ser que dibujemos un camino a través de ellas.
- **Pintado de caminos**. Esta es la herramienta más importante, ya que marcará el diseño de nuestro nivel. Con ella podemos definir los caminos principales de nuestro mapa, por los que queremos que los jugadores se muevan. Veréis que al utilizar una *tool*, estaremos «pisando» lo que había debajo, por lo que, por ejemplo, si creamos un camino encima de agua, este se adaptará el terreno para que esa zona de agua se pueda atravesar. La herramienta de pintado de caminos tiene una opción alternativa en Path Type llamada *destruction* que permite crear grietas intraspasables en el terreno.

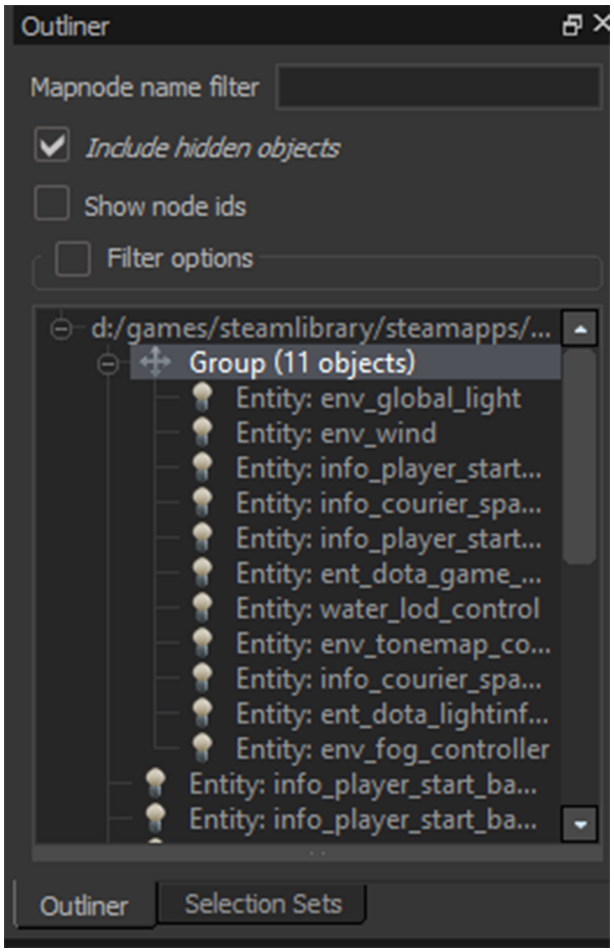
Imagen 13.




- **Pintado de árboles**. Esta herramienta coloca árboles, plantas y otra vegetación.
- **Pintado de plantas**.Mientras que la herramienta anterior prioriza el pintado de árboles, esta solo coloca plantas y hierba.
- **Pintado de *blends***. La herramienta permite elegir entre diferentes *layers* o capas (representadas más abajo) para pintar con espray la zona con otra textura. Se pueden mezclar capas para crear zonas con distintas vegetaciones o roca a fin de darle un detalle más avanzado al mapa.
- **Pintado de GridNav**.La GridNav o *navigation grid* es la malla de navegación del mapa, es decir, zonas por las que los personajes podrán moverse. La GridNav se va generando automáticamente a medida que diseñamos el mapa, y podemos visualizarla con *control* + Q. Si hay algún problema, podemos utilizar esta herramienta para modificar manualmente la GridNav.
- **Colocación de objetos**. Esta herramienta permite colocar objetos decorativos de manera automática (velas, campamentos, calaveras y otros detalles varios que le dan algo más de carisma al mapa).
- **Selección de Tile**.
- **Eliminación de Tile**.

Ahora que conocemos las herramientas básicas de Hammer, podemos ponerlas en práctica para crear nuestro primer mapa. Si vamos a Outliner (a la derecha de la interfaz de Hammer), podemos encontrar el Group 1, donde están las principales *entities* del juego, como la posición inicial del jugador. Debemos asegurarnos de que estas están en una zona transitable del mapa; si no es así, al arrancar el juego nos encontraremos con el personaje encerrado.

Imagen 14.




Presionando F9, montaremos una versión jugable del nivel que podremos testear al momento.

**Reto 01**

Utilizad las herramientas de edición de mapa para crear vuestro propio diseño de nivel. Puede ser un diseño de nivel para un juego de tipo MOBA o algo más arriesgado, como un nivel para un *hack and slash* o un *action RPG*.

A estas alturas, lo que nos interesa es adquirir práctica con las herramientas de edición y poder plasmar nuestro diseño de nivel de un boceto en papel a Hammer, intentando transmitir nuestras decisiones de diseño de juego.

Presionando F9, podemos montar el nivel y jugarlo en el propio *Dota 2* con un personaje jugable. Podemos seguir haciendo modificaciones en el nivel hasta que encontremos un diseño que nos convenza.

Imagen 15.


Podéis encontrar más información sobre las herramientas de Hammer en:

https://developer.valvesoftware.com/wiki/Dota_2_Workshop_Tools/Level_Design/Tile_Editor_Basics

# 3. Entidades y *scripting*

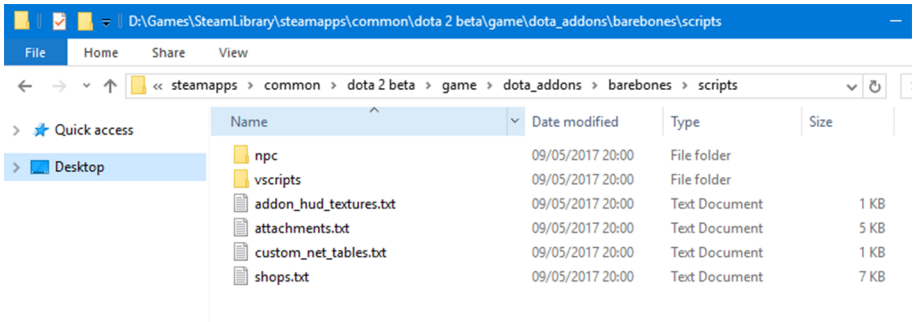
## 3.1. Introducción

Crear un mapa es algo directamente enfocado al diseño de nivel. Un buen mapa puede cambiar radicalmente cómo se juega a un tipo de juego (pensad solo en cómo cambiaría la manera de jugar a *League of Legends* si en vez de tres caminos principales hubiese cinco). Pero un diseño de mapa no cambia las reglas de un juego. Para hacer cambios sustanciales en un juego y llegar a crear uno nuevo totalmente distinto al original, necesitaremos utilizar las herramientas de *scripting*.

El *scripting* en *Dota 2* se divide en *scripting* con Lua y *scripting* con parejas de datos del tipo KeyValue. Cualquier editor de texto, como Visual Studio Code, Notepad++ o Sublime Text, nos sirve para trabajar con *scripting* de *Dota 2*.

Primero de todo, necesitaremos ir a la carpeta *scripts* de nuestro *add-on*. Recordemos que estamos utilizando Barebones como plantilla, de manera que, a no ser que hayamos renombrado la carpeta, estaremos trabajando con el contenido de la carpeta *barebones*, dentro de game/data_addons/. La carpeta que contiene los *scripts* de nuestro *add-on* se llama *scripts*.

Imagen 16.




La carpeta *npc* contiene lo siguiente:

• npc_abilities_custom.txt. Incluye las habilidades y magias de los personajes de nuestro *mod*.

• npc_heroes_custom.txt. Contiene héroes con sus habilidades y estadísticas.

• npc_items_custom.txt. Reúne objetos que un personaje puede llevar en su inventario.

• npc_units_custom.txt. Recoge estadísticas y datos de NPC, edificios y criaturas.

• npc_abilities_override.txt. Incluye versiones «modificadas» de habilidades e ítems ya existentes en *Dota 2*.

• herolist.txt. Es la lista de héroes disponibles para escoger en nuestro *mod*.

Los anteriores ficheros son datos en formato KeyValue, una manera de almacenar y estructurar información estática. Cada fichero está comentado de manera que es bastante sencillo entender cómo realizar modificaciones sobre los valores por defecto. Si queremos crear comportamientos más interesantes, tendremos que recurrir al *scripting* con Lua, del que hablaremos más adelante.

### 3.2. *Scripting* mediante KeyValue

Es extremadamente recomendable descargar Sublime Text y luego utilizar el siguiente *plug-in* para tener autocompletado de *scripting* de *Dota 2* dentro del editor de Sublime Text:

https://github.com/bhargavrpatel/dota_kv#installation

Si elegís instalar este *plug-in*, acordaos de cambiar el tipo de sintaxis en el selector de la esquina inferior derecha de Sublime Text a Dota KV.

Imagen 17.


También existe otro *plug-in* para autocompletado de *scripting* en Lua, que nos irá bien más adelante:

https://github.com/bhargavrpatel/Dota-2-Sublime-Packages#installation

Vamos a proceder a crear nuestro primer *script* con KeyValue, que definirá una nueva habilidad mágica para nuestro personaje. Dentro de la carpeta /scripts/, creamos un nuevo fichero uoc.txt con el siguiente contenido:

Imagen 18.

```
"habilidad_de_ejemplo"
{
        "BaseClass"                     "ability_datadriven"
        "AbilityTextureName"            "ability_name"
        "MaxLevel"                              "7"

        "AbilityBehavior"                       "DOTA_ABILITY_BEHAVIOR_UNIT_TARGET"
        "AbilityUnitTargetTeam"                 "DOTA_UNIT_TARGET_TEAM_ENEMY"
        "AbilityUnitTargetType"                 "DOTA_UNIT_TARGET_HERO |
DOTA_UNIT_TARGET_BASIC"
        "AbilityUnitDamageType"                 "DAMAGE_TYPE_MAGICAL"

        "AbilityDamage"                                 "500"

        "OnSpellStart" // EVENTO
        {
                "Damage" // ACCIÓN
                {
                        "Target"        "TARGET"
                        "Type"          "DAMAGE_TYPE_MAGICAL"
                        "Damage"        "%AbilityDamage"
                }
        }
}
```

Nuestro *script* comienza con un nombre identificativo: «habilidad_de_ejemplo».

Al empezar a escribir BaseClass sin comillas, el *plug-in* de autocompletado nos ofrecerá automáticamente insertar «BaseClass» y varias líneas más que van asociadas a dicha definición. Para el resto de claves, ocurrirá igual: el *plug-in* escribirá por nosotros la gran mayoría de código.

BaseClass define qué tipo de *mod* o *scripting* tenemos entre manos. En este caso, definimos que nuestro *script* es una «ability_datadriven», es decir, una habilidad definida por datos en formato KeyValue. AbilityTextureName se refiere al identificador de la textura utilizada como icono de la habilidad; se puede utilizar cualquier identificador de icono existente en *Dota 2*.

AbilityBehavior define cómo se comporta nuestra habilidad. En este caso, es una habilidad que afecta a un único objetivo enemigo, ya sea un héroe o una unidad básica.

Cualquier habilidad requiere eventos que definan lo que ocurre al utilizarla. OnSpellStart es un evento que se ejecutará al utilizar la habilidad, e incluye la acción Damage, que define un daño (especificado en la variable AbilityDamage), un tipo de daño (mágico) y el rango (un único *target*). Una misma habilidad puede tener definidos múltiples eventos, que pueden incluir una o más acciones.

La documentación oficial de Valve enumera los distintos tipos de eventos y acciones disponibles: https://developer.valvesoftware.com/wiki/Dota_2_Workshop_Tools/Scripting/Abilities_Data_Driven

Para añadir nuestra nueva habilidad en la lista de habilidades disponibles en nuestro *mod*, debemos incluirla en npc_abilities_custom.txt, de la carpeta / scripts/npc/. Podemos simplemente copiar el contenido de uoc.txt justo después de la quinta línea, encima de containers_lua_targeting.

Ahora que tenemos la habilidad dentro del *mod*, hace falta otorgársela a algún héroe. Abrimos npc_heroes_custom.txt y cambiamos el valor dentro de Ability1 de «example_ability» a «habilidad_de_ejemplo». Guardamos fichero y lanzamos nuestro *mod* desde Hammer con F9.

Al arrancar el juego, escogemos al héroe Ancient Apparition (primer héroe de la penúltima fila), ya que es el que estamos sobrescribiendo en npc_heroes_custom. Al empezar la partida, nuestro héroe debería tener como única habilidad la que acabamos de definir.

Imagen 19.


Si queremos comprobar el efecto de nuestra habilidad, podemos utilizar la consola de comandos (el propio chat) para *spawnear* a un enemigo. Presionamos Enter y escribimos *createhero kobold enemy*. Debería aparecer un enemigo, que empezará a atacarnos. Haciendo clic en nuestra habilidad o presionando Q, podemos después clicar en *kobold* y ver el efecto de nuestra nueva habilidad.

Podéis encontrar más comandos de consola en: http://dota2.gamepedia.com/Cheats

A estas alturas, tenemos los conocimientos para crear nuevas habilidades, ítems e incluso sobrescribir las estadísticas y las habilidades de héroes y monstruos del juego. La gran variedad de valores disponibles y posibilidades se escapa al contenido de este módulo, pero la documentación oficial de Valve y de otras páginas de la comunidad constituye la única ayuda necesaria para construir cualquier tipo de contenido propio.

**Reto 02**

Definid tres nuevas habilidades para nuestro héroe modificado. Cada habilidad debería tener un resultado diferente. Por ejemplo, definid una habilidad con un rango en área que aplique un efecto secundario a los enemigos, otra habilidad que afecte al propio héroe para curarlo, etc.

Utilizad los «trucos» de consola del chat para subir de nivel al personaje y así hacer que pueda emplear todas las habilidades en una misma sesión de juego.

### 3.3. *Scripting* con Lua

Todos los *scripts* en Lua se colocan en la carpeta /scripts/vscripts/.

La utilización de *scripts* con Lua en *Dota 2* tiene las siguientes funciones:

**1)** Definir la lógica de juego.
**2)** Llamar a código en Lua desde eventos de *KeyValue scripting*.
**3)** Permitir la entrada y la salida de datos del editor Hammer.
**4)** Permitir eventos de UI.

Lua es un lenguaje genérico de *scripting* utilizado en innumerables juegos. *Dota 2* posee una poderosa API en Lua que permite controlar cada aspecto del juego. En la carpeta /scripts/vscripts/ encontraréis los ficheros .lua que vienen por defecto en Barebones.

Describir y detallar el uso de Lua se escapa al contenido de este módulo, ya que adquirir un conocimiento avanzado de la materia puede llevar semanas de aprendizaje. De todos modos, vamos a proporcionar un pequeño ejemplo de función en Lua que extenderá nuestra lista de habilidades de héroe.

Creamos un nuevo fichero, ataque_especial.lua, en la carpeta /scripts/vscripts/. Su contenido definirá el comportamiento de nuestra habilidad:

Imagen 20.

```
function MiHabilidad( event )
    local caster = event.caster
    local target = event.target

    if target:GetHealthPercent() > 50 then
        target:Kill(nil, caster) -- Si el objetivo tiene más del 50% de su vida
morirá
    end
end
```

La habilidad simplemente mata al objetivo si este tiene más del 50% de su vida. La función definida utiliza la API en Lua para afectar a elementos del juego.

Ahora modificamos nuestra antigua habilidad de prueba y en OnSpellStart añadimos:

```
"RunScript"
 {
     "ScriptFile"     "ataque_especial.lua"
     "Function"      "MiHabilidad"
 }
```

Podemos eliminar la acción Damage que había definida, ya que la nueva acción RunScript buscará nuestro nuevo *script* en Lua y ejecutará la función MiHabilidad, que matará a la unidad enemiga.

# 4. Diseño de un mapa jugable

## 4.1. Introducción

Pese a todo lo que hemos aprendido de creación de mapas y *scripting*, nos falta llegar al paso esencial de ponerlo en práctica.

Vamos a crear nuestro propio nivel jugable para *Dota 2*. Este mapa enfrentará a dos jugadores en un solo carril o *lane*, las bases y una torre por equipo.

Lanzamos las herramientas de editor y cargamos el mapa que hemos hecho anteriormente.

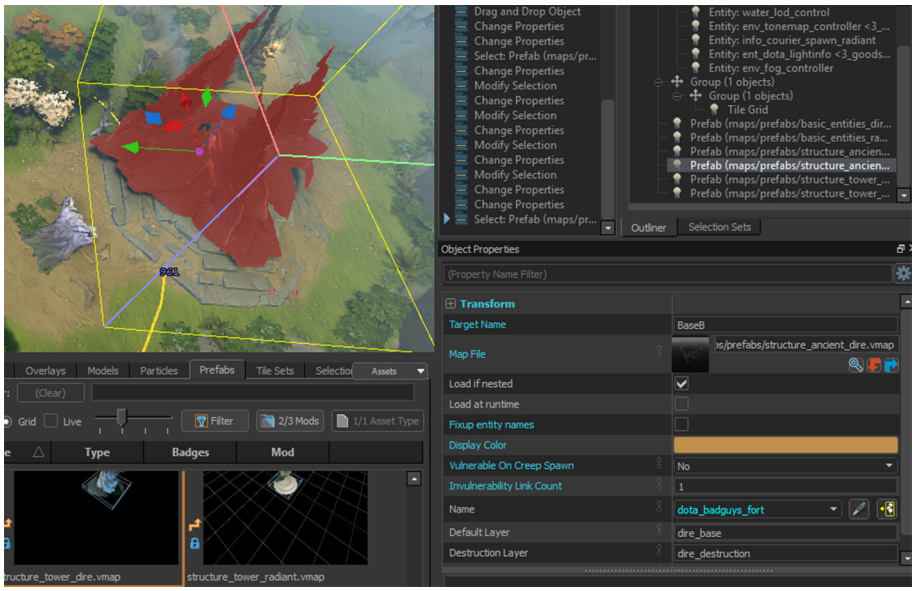
En el Outliner (a la derecha), seleccionamos todos los objetos Entity y los eliminamos. Nos aseguramos de eliminar cualquier entidad de player_start o parecida. Para que el *mod* utilice las «reglas» de *Dota*, debe haber una entidad llamada ent_dota_game_events. Si no la hay, buscadla en Prefabs y arrastradla a la escena.

Abajo, en Prefabs, cogemos basic_entities_dire.vmap y lo arrastramos a la ubicación del mapa donde queremos que una facción empiece. Hacemos lo mismo con basic_entities_radiant.vmap.

Para colocar las bases, arrastramos al mapa structure_ancient_radiant y structure_ancient_dire. En las propiedades del objeto (abajo a la derecha), cambiamos el Target Name a BaseA y BaseB respectivamente. También cambiamos Set Vulnerable On Creep Spawn a No, y Invulnerability Link Count a 1.

El *target name* se utiliza para referenciar este objeto desde otros eventos. Cuando destruyamos una torre del equipo A, enviaremos un evento con el *target name* BaseA para hacer que la base se vuelva vulnerable. Set Vulnerable On Creep Spawn hace que el objeto sea atacable desde el inicio de la partida, por lo que hemos cambiado su estado a No. Invulnerability Link Count define cuántas torres «protegen» a la base de manera directa. Solo tendremos una torre conectada.

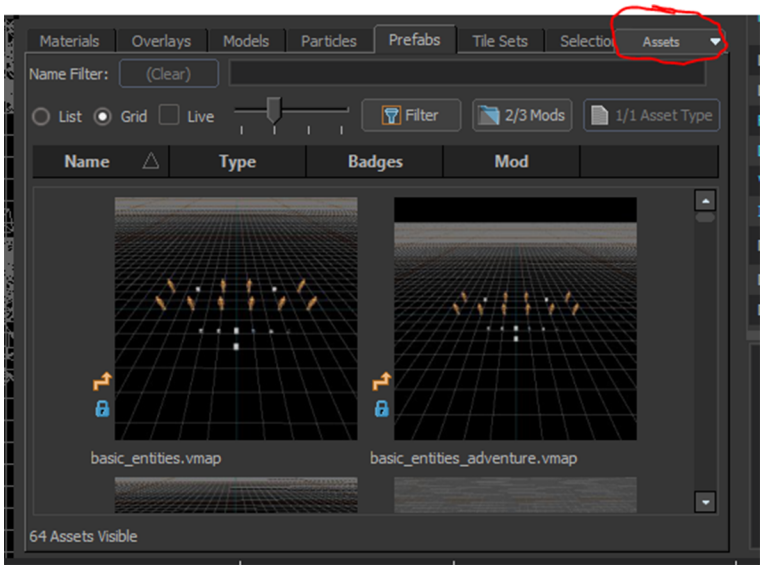
Imagen 21.




Ahora colocamos una torre para cada facción. Arrastramos al mapa structure_tower_radiant y structure_tower_dire, y colocamos a cada una de ellas cerca de la base correspondiente. En las propiedades ponemos TowerLocation a middle_tier_1.

Si lanzamos el *mod*, veremos que todo es jugable, pero al destruir la torre enemiga, la base sigue siendo invulnerable. Necesitamos crear un evento que la haga vulnerable.

Imagen 22.




En el Asset Browser (pestañas de abajo), hacemos clic en la flecha de Assets y seleccionamos Entity IO. Seleccionamos la TorreA y en la pestaña Outputs clicamos en Add y definimos el siguiente evento:

- My output named: "OnTowerKilled"
- Target entities named: "BaseA"

- Via this input: "ReduceInvulnCount"

Hacemos lo mismo con la TorreB, con un *target entity* que apunte a BaseB.

Cuando una torre caiga, la base recibirá un evento para reducir su invulnerabilidad en 1 y volverse vulnerable.

El juego ya es «ganable», pero nos falta algo elemental en un MOBA: las oleadas de *creeps* o *mobs*. Necesitaremos colocar un edificio (Barracks) que instancia los *creeps*, y luego definir la ruta que seguirán estos hasta la base enemiga.

Colocamos structure_barracks_melee_radiant en el mapa arrastrándolo desde el Asset Browser. Lo pondremos cerca de la base, escondido detrás de la torre. Haremos lo mismo con structure_barracks_melee_dire. Sus propiedades serán las siguientes:

• Target Name = "BarracksA"

• Vulnerable On Creep Spawn = "No"

• Invulnerability Link Count = "1"

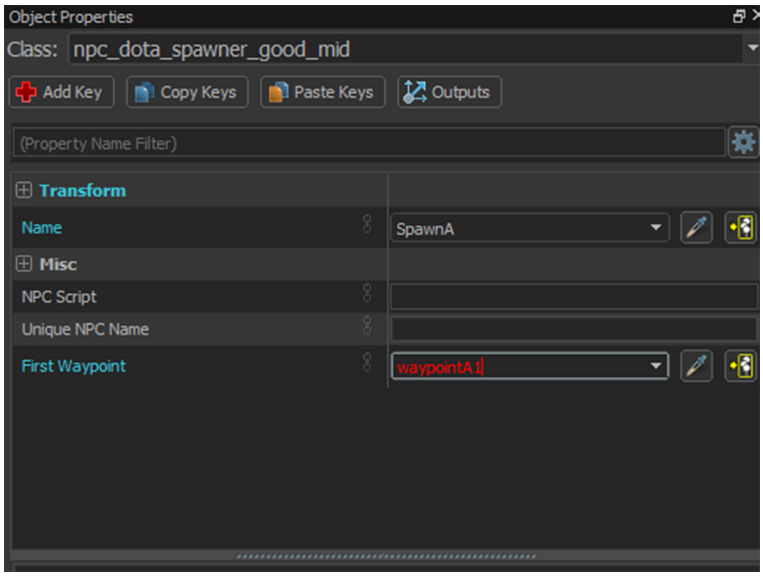
• Barracks Location = "middle"

Utilizamos un evento similar al que hemos hecho para las torres, de manera que al destruir la torre, el Barracks reduzca su *invulnerability link count* en 1 y se vuelva vulnerable.

Abrimos la Entity Tool haciendo clic en la bombilla de la izquierda o presionando *shift* + E. En el *dropdown* seleccionamos npc_dota_spawner_good_mid y clicamos enfrente de nuestros *barracks* para colocar el *spawner*. En las propiedades ponemos:

- Name: "SpawnA"
- First Waypoint: "waypointA1"

Imagen 23.




En la Entity Tool buscamos path_corner y lo colocamos en el mapa. Este será el primer *waypoint* o lugar al que se dirigirán los *creeps* que salgan del *spawner*. En sus propiedades ponemos:

- Name = "waypointA1"
- Next Stop Target = "waypointA2"

Colocamos un nuevo path_corner al lado de la base enemiga y ponemos:

- Name = "waypointA2"

Podemos comprobar cómo una serie de líneas verdes conectan los path_corners o *waypoints*, ya que los hemos ido conectando indirectamente mediante la propiedad Next Stop Target.

Por último, colocamos una entidad info_target; da igual el lugar. Esta entidad es lo que se llama un *staging node*, que simplemente actúa como almacenaje virtual de los *creeps*. Se utiliza para mejorar el rendimiento. En Name ponemos npc_dota_spawner_good_mid_staging.

Repetimos el proceso para la otra facción utilizando un npc_dota_spawner_bad_mid y nuevos *spawners* y path_corners para definir la ruta de los *mobs* de dicha facción. El info_target tendrá un Name = npc_dota_spawner_bad_mid_staging.

Con F9 podemos compilar y ejecutar nuestro mapa. Por fin tenemos nuestro propio *mod* de *Dota 2* listo para jugar y compartir con el mundo.

Para más información y detalles, la wiki oficial ofrece toda la información necesaria: https://developer.valvesoftware.com/wiki/Dota_2_Workshop_Tools/Level_Design/Creating_A_Dota-Style_Map

Imagen 24.

**Reto 03**

Expandid el mapa definiendo tres rutas de *creeps* y colocando un mínimo de una torre por ruta.

# 5. Soluciones a los retos

## 5.1. Reto 01

Solución libre.

## 5.2. Reto 02

Imagen 25.

```
  "regeneracion_divina"
  {
    "BaseClass"                 "ability_datadriven"
    "AbilityTextureName"      "ability_name"
    "MaxLevel"              "1"

    "AbilityBehavior"       "DOTA_ABILITY_BEHAVIOR_UNIT_TARGET"
    "AbilityUnitTargetTeam"     "DOTA_UNIT_TARGET_TEAM_FRIENDLY"
    "AbilityUnitTargetType"     "DOTA_UNIT_TARGET_HERO | DOTA_UNIT_TARGET_BASIC"
    "AbilityUnitDamageType"     "DAMAGE_TYPE_MAGICAL"

    "OnSpellStart"
    {

      "Heal"
      {
        "Target"        "CASTER"
        "HealAmount"   "100"
      }
    }
  }

  "contusion_sismica"
  {
    "BaseClass"                 "ability_datadriven"
    "AbilityTextureName"      "ability_name"
    "MaxLevel"              "1"

    "AbilityBehavior"           "DOTA_ABILITY_BEHAVIOR_AOE |
DOTA_ABILITY_BEHAVIOR_POINT"
    "AbilityUnitTargetTeam"     "DOTA_UNIT_TARGET_TEAM_ENEMY"
    "AbilityUnitTargetType"     "DOTA_UNIT_TARGET_HERO | DOTA_UNIT_TARGET_BASIC"
    "AbilityUnitDamageType"     "DAMAGE_TYPE_MAGICAL"

    "OnSpellStart"
    {
      "Stun"
      {
        "Duration"   "3.0"
        "Target"
        {
          "Center"    "POINT"
          "Radius"   "300"
          "Teams"   "DOTA_UNIT_TARGET_TEAM_ENEMY"
          "Types"   "DOTA_UNIT_TARGET_HERO | DOTA_UNIT_TARGET_BASIC"
        }
      }
    }
  }
```