UNIVERSIDAD NACIONAL DE QUILMES
Departamento de Ciencia y Tecnología
Tecnicatura Universitaria en Programación Informática

Drenar's World

Desarrollo de un juego de tipo RPG en Haskell

Trabajo de Inserción Profesional

Mieres Federico
Suárez Cristian

Director: Pablo E. Martínez López, Dr.

17 de diciembre de 2012

Índice

1. Introducción

En el negocio de los videojuegos existen en la actualidad tipos muy variados de juegos. Uno de estos tipos, que es muy relevante por su cantidad de ventas, es el de los juegos llamados RPG. Los Videojuegos de RPG (Role-Playing Games, o en castellano Juego de Interpretación de Papeles) son juegos en los que el jugador controla un personaje (muchas veces llamado Héroe) el cual va adquiriendo habilidades u objetos conforme va progresando el juego. Además, el personaje tiene un Rol específico, que en algunos juegos va cambiando a medida que se avanza en el mismo. Este rol indica qué cosas puede hacer el personaje y qué cosas no[1].

Algunas características comunes en este estilo de juegos son, por ejemplo, el desarrollo estadístico del personaje, la obtención de objetos determinados, la posibilidad de cumplir misiones y obtener recompensas por ello, y el manejo de una economía personal. Algunos juegos tienen la posibilidad de poder jugar indefinidamente, lo cual quiere decir que, o bien no tienen fin, o bien dan la libertad de adquirir todo lo que se puede conseguir aunque se haya cumplido el objetivo principal. Al ser necesario mucho tiempo de juego, se puede lograr fácilmente que las personas que lo juegan no dejen de hacerlo por mucho tiempo, y ésto hace que las empresas le presten especial atención en su desarrollo.

En la actualidad, la mayoría de los juegos de este estilo son MMORPG (Massive Multiplayer Online Role-Playing Game, o Juego de Interpretación de Papeles en Línea con Múltiples Jugadores), es decir, son jugados mediante Internet y se puede interactuar con otras personas. Esto constituye otra fuente de ingresos para las empresas, y dan relevancia a esta temática.

El desarrollo de Videojuegos RPG suele estar hecho en los lenguajes C [KR88] (programación estructurada) o C++ [Str85] (programación orientada a objetos híbrido). La razón de que se utilicen estos lenguajes es la eficiencia y el manejo de muy bajo nivel que poseen. Dado que nuestro enfoque se basa en el prototipado de un videojuego, estas ventajas no son relevantes. Sí lo son la facilidad de abstracción y modelado. Y siendo así optamos por el paradigma funcional, con Haskell como lenguaje con el cual trabajar.

En la actualidad los lenguajes funcionales todavía no alcanzan la eficiencia de los lenguajes de bajo nivel, pero son lo suficientemente eficientes como para realizar juegos de mediana escala, y un ejemplo de ellos es Haskell [Jon02]. Haskell es un lenguaje de Programación Funcional no estricto y fuertemente tipado. Sus ventajas son que permite código muy conciso y compacto, facilidad de comprensión, manejo de tipos de datos, reutilización de código, polimorfismo y funciones de alto orden. Un aspecto importante en Haskell es que es muy fácil modelar y prototipar, algo muy útil a la hora de programar juegos. Además, teniendo en cuenta que las computadoras en la actualidad suelen tener mejores recursos disponibles nos permitimos olvidarnos del manejo manual de la RAM y dejamos que el lenguaje haga lo que crea más conveniente en ese aspecto. Con estas características, creemos que se podrían realizar videojuegos RPG en lenguajes funcionales como Haskell, utilizando cómodas abstracciones y conservar niveles razonables de eficiencia.

Para el desarrollo de un juego son necesarias diversas áreas que lo definen en su conjunto. La programación es una de ellas: los códigos que modelan las mecánicas, las entidades y su interacción. También tenemos el área gráfica que provee los diseños e ilustraciones de entornos, personajes y animaciones necesarias para darle atractivo al juego. Otra área que encontraremos es la de la musicalización, la cual provee la música y los sonidos necesarios para el juego. También tenemos los eventos, que mezclan todos los aspectos anteriores

[1] Juegos RPG: Se han hecho informes sobre su historia y sus características en internet en distintas páginas de Videojuegos.

para dar un resultado jugable. Es importante que los eventos queden separados de los anteriores aspectos, y se utilize alguna forma de robusta y fácil configuración. Finalmente, la sección comercial, que publique el juego y le haga propaganda.
Este informe presenta nuestro Trabajo de Inserción Profesional (TIP) para culminar la carrera de Técnico Universitario en Programación Informática de la UNQ. La temática de este TIP fue el desarrollo de un juego de RPG desde cero usando el lenguaje Haskell. Describimos los diferentes conceptos involucrados en el desarrollo de un juego, las decisiones tomadas para su realización, así como las herramientas que escogimos para completar la tarea. También incluímos la intención original de la elección, los problemas que surgieron y las decisiones que tomamos para solucionarlos.
Para diseñar el juego nuestra tarea se dividió en varias partes. Por un lado modelamos la idea del juego y la implementamos en Haskell. Por otra parte digitalizamos ilustraciones provistas por terceros no expertos para incluirlas en la interfaz. También armamos un pequeño sistema de configuraciones para especificar las entidades y algunos eventos de manera dinámica. Y finalmente mostramos que todo esto funciona preparando una Demo del juego, con algunas interacciones básicas y un ejemplo de un evento principal.
Como futuros programadores le dimos importancia a la programación y configuración, pero no dejamos de lado los otros aspectos y le dedicamos algo de tiempo a las ilustraciones y la musicalización (esto último no pudo ser agregado al juego, más adelante se menciona el por qué), sin buscar un aspecto totalmente profesional de los mismos. Al único aspeco que no le dimos atención fue al relacionado con propaganda o publicación, ya que no entraba en nuestros objetivos.
El documento está organizado de acuerdo a las diferentes etapas y aspectos abarcados durante el proyecto. Comenzaremos hablando del modelo, las entidades del juego y sus interacciones. Pasaremos entonces a la externalización de datos y las configuraciones que dan modularidad a las etapas de prueba. Seguiremos con el modelo de vista, y su interacción con el usuario. Cerramos con las conclusiones que pudimos obtener durante este trabajo.

2. Drenar's World - Modelado del Juego

En esta sección explicaremos a grandes rasgos el diseño que elegimos para nuestro juego y las decisiones tomadas en cada paso. Como ya mencionamos, elegimos hacer un videojuego del estilo RPG, porque presenta un grado de dificultad que lo hace no trivial y tiene muchas características para poder configurar a gusto. Además, un juego RPG puede expandirse indefinidamente si se quisiera.
Nuestro juego consiste en manejar un personaje, el cual sigue una determinada historia, y va cumpliendo misiones a medida va progresando el juego. Tiene un inventario donde guardará cartas que obtendrá en el progreso. Además, existen otros personajes, los cuales son manipulados por el programa e interactúan con el usuario.
En el mundo en el que el personaje se encuentra hay criaturas fantásticas, las cuales denominamos Drenars, que se pueden encontrar en estado salvaje o pueden ser compañeros del protagonista o de otros personajes virtuales. La acción que permite relacionar a los Drenars es denominada Batalla. Por el momento los Drenars del protagonista podrán entablarán unicamente Batallas Salvajes. El protagonista se encontrará con Drenars salvajes a medida que avanza en el juego, los cuales lo retarán a luchar. A este tipo de batalla es a las que denominamos Batallas Salvajes. Si los Drenars del protagonista llegan a vencer a sus enemigos, se podrá obtener el botín que posean. Además, en las Batallas Salvajes el protagonista puede atrapar algunos Drenars salvajes, los cuales al ser capturados se

guardarán en su inventario. El protagonista podrá capturar tantos Drenars como desee.
La acción básica en una batalla está representada por un hechizo que los Drenars pueden utilizar en combate. Estos hechizos serán representados como cartas en el inventario del personaje y las denominaremos SpellCards o Cartas Mágicas. En batalla, el protagonista ordenará a un determinado Drenar que utilice una SpellCard. Las cartas podrán conseguirse como premio tras avanzar con la historia del juego, u obtenerlas como botín en una Batalla Salvaje.
Las batallas que sucedan en el juego estarán organizadas por turnos del usuario y su oponente. Ambos deciden qué Spellcards serán utilizadas por sus Drenars en su turno y luego observarán las consecuencias de sus decisiones. Esto se repite hasta que haya un vencedor o alguno abandone. En otras palabras, si los Drenars de uno de los lados resultan muertos o capturados, el otro participante vencerá. El único que puede capturar otros Drenars es el protagonista.
Nuestro trabajo consistió en desarrollar un modelo básico sobre el que se construyen diversos escenarios con las características mencionadas, en los cuales la jugabilidad cambia dependiendo de qué cosas se le agregan al juego, ya sea dificultad a medida se avanza en el mismo, o la creación de muchos y diversos escenarios, o variedad en los Drenars que uno encuentra. Se pueden crear tantos escenarios como se desee y crear tantas cadenas de misiones como se quiera utilizando el sistema de configuraciones. Para analizar como funciona cada parte del juego describimos los conceptos que se modelaron. Dividimos este análisis en, por un lado, las entidades y, por el otro, su interacción.

2.1. Modelo - Historia

El Protagonista será guiado por una historia principal. Debido a que estamos presentando una demostración del juego, no hay una historia que lo complete. Sí presentamos un ejemplo con una demo consistente en una historia corta introductoria. Mediante los archivos de configuración se pueden crear historias con facilidad, algunas sencillas que no involucren demasiados pasos a seguir hasta otras más robustas que exigirán viajar y hablar con NPCs y cumplirles misiones.

2.2. Modelo - Entidades

2.2.1. Drenars, Wild Drenars, Battle Drenars

La entidad más importante del juego el Drenar. Los Drenars son criaturas que el protagonista puede obtener y coleccionar, haciendolas interactuar con otros Drenars de cierta forma que explicaremos más adelante. Cada Drenar está armado con diferentes características que en su variación darán como resultado una gran diversidad de Drenars (en el apéndice se puede ver un diagrama de Entidades).
Como ya mencionamos, la interacción básica entre Drenars consiste en batallas, que pueden ser iniciadas tanto por el jugador como por alguna entidad virtual del juego. Los Drenar pueden poseer o no un dueño, que es una entidad que puede ser encarnada por un jugador, o ser la realización virtual de uno. Los Drenars Salvajes pueden ser capturados por un jugador y dejan de ser Salvajes, y a partir de allí pueden ser entrenados por el dueño para ser mejores en las batallas.
Un Drenar Salvaje es la representación de un Drenar que no tiene dueño. Contiene información de qué Drenar representa y características extra que lo definen como salvaje, como sus propios ataques y la recompensa que da por ganarle en una batalla. También contiene información sobre la captura y las condiciones que deben darse en ese Drenar Salvaje para

que sea definitiva su captura, con información para el caso de no cumplir las condiciones, es decir, el índice básico de captura.
Cuando un Drenar está en batalla, se convierte en una entidad que tiene más información, al cual llamamos Drenar de Batalla. Un Drenar de Batalla contiene la información que se necesita guardar sólo para la batalla actual, como sus valores de características modificados en la contienda actual. Una Batalla está compuesta entre otras cosas, por la representación de combate de cada drenar que participa.

2.2.2. Personajes y Misiones

Al personaje que utiliza el protagonista antes mencionado se le da el nombre de Player o Jugador. El Jugador contiene información para representar al usuario dentro del juego, tal como su nombre, sus diferentes Drenars y su progreso en el juego.
El Player puede ser manipulado por el usuario, haciendo que camine en las direcciones que desee (Norte, Sur, Este, Oeste). Interactúa con personajes que son manejados por el programa, y cumple misiones que éstos le dan.
Los personajes que son manejados por el programa son denominados NPC (del inglés Non-Player Character) y son los que interactúan con el usuario de diversas maneras. Pueden dar misiones al jugador, establecer diálogos o darle regalos. Las misiones, denominadas Quests, son la forma de hacer que un NPC tenga un determinado diálogo en un determinado momento, o que ofrezca alguna recompensa al Player si se quiere. Dependiendo en qué parte de una Quest se encuentre el Player, determinados NPC actuarán de distinta forma, mientras que a otros no les afectará en nada.
Una Quest que el protagonista podrá realizar en la demo que presentamos en la primer zona donde se encuentra será hablar con determinados NPCs en un determinado orden para obtener una SpellCard como recompensa. Los NPCs ofreceran ayuda al jugador indicándole con quién debe hablar a continuación.

2.2.3. Auras

Las Auras representan modificaciones de las características de un Drenar. Los hay de dos tipos: Buffs y Special Buffs.
Los Buffs son los utilizados en batalla. Estos modifican los atributos de los Drenars de tal forma que los hacen más fuertes o más débiles. Los Buffs tienen un Tipo, que los diferencian del resto de los Buffs.
Los Special Buffs son diferentes a los anteriores porque actúan o importan en cualquier momento que no sea una batalla. Representan un estado extra resultante de una batalla. Los Special Buffs son la consecuencia de un Buff en la batalla, aunque no todos los Buffs se transforman en Special Buffs. Un ejemplo de un Buff que se transforma en Special Buff y perdura luego de la batalla podría ser la muerte del Drenar: si muere en batalla, seguirá estando muerto luego de ésta.

2.2.4. Cartas Mágicas

Como se mencionó anteriormente, los ataques de los Drenars se representan con Cartas Mágicas, o Spell Cards, que son las entidades que representan toda acción que puede tomar un Drenar en batalla siendo ésta su única manera de alterar la misma. Hay cartas de diferentes tipos, acordes a los roles que puede tener un Drenar. Una Spellcard entonces

contiene información de su nombre, el rol y su nivel requerido para utilizarla, la cantidad de turnos que deben esperarse para volver a usarla (Cooldown), el poder básico de la carta y algunos datos extra sobre la acción que realizan al utilizarse.

2.3. Modelo Batalla

Como se menciona anteriormente, los Drenars fueron creados para dar lugar a batallas. Por el momento solo hay un tipo de batalla, la cual es denominada Batalla Salvaje.

2.3.1. Batalla Salvaje - (Wild Battle)

Las Wild Battles o Batallas Salvajes son la forma de representar el Player y sus Drenars combatiendo contra Drenars Salvajes. Las Batallas de este tipo se pueden dar o bien porque el usuario caminó sobre un área de Drenars Salvajes con chances aleatorias de que surgiera una o porque un Drenar salvaje ligado a la historia indujo la batalla. El objetivo de estas batallas es darle la posibilidad de atrapar nuevos Drenars o simplemente permitirle entrenar los que ya tiene. En estos tipos de lucha, si el usuario se queda sin Drenars, perderá el juego automáticamente. Para representar una Wild Battle se pensó en una agrupación de todos los Drenars de batalla, los salvajes y los del usuario, las SpellCards del Player y su Cooldown y finalmente los Drenars salvajes que intervienen. Esto último sucede porque se requiere información de los Drenars salvajes al terminar la batalla con éxito. Los Drenars salvajes que fueran atrapados en estas batallas serán retirados de las mismas y agregados al Jugador en su inventario. Cuando ya no queden Drenars salvajes en la pelea, el jugador habrá ganado la batalla.

2.4. Ciclo de una Batalla típica contra Drenars Salvajes

Ahora que mencionamos las batallas podemos describir como es el ciclo típico de una (Dirigirse al Apéndice si se desea visualizar un diagrama de flujo de una Batalla).
Cada vez que el protagonista camina por áreas con Drenars Salvajes existe una chance de que se tope con alguno o algunos aleatoriamente. Si esto sucede, tendrá inicio una Batalla Salvaje. El protagonista podrá optar por luchar contra los Drenars Salvajes con los que se encontró o podrá huir de la contienda. Si decide huir, no obtendrá penalización pero tampoco ninguna ventaja ni experiencia.
Si en cambio decide luchar, se habilitará el menú de SpellCards. Las cartas se encuentran en la parte izquierda de la pantalla mientras que en la derecha se mustran sus características. Allí el protagonista eligirá una carta que pueda ser usada por alguno de sus Drenars. Al elegir una carta usable, el menú de SpellCards se ocultará para permitirle elegir qué Drenar usará esa carta. Los Drenars del jugador se muestran en la parte inferior de la pantalla y debajo de cada uno se ubica su salud; los Drenans enemigos se encuentran en la parte superior de la pantalla, y sobre ellos se muestra su salud. En la parte superior izquierda de cada Drenar se muestran los Buffs de cada uno. Luego de seleccionar al Drenar que utilizará la SpellCard elegida, el protragonista deberá pasar a elegir los objetivos de dicha SpellCard. Todas las cartas pueden ser usadas sobre aliados o enemigos, y en el caso de que posean varios objetivos, deben pertenecer al mismo grupo.
Esta acción se deberá repetir hasta que el protagonista haya asignado una tarea a cada Drenar que posea o hasta que decida dejar de elegir cartas a utilizar. En ese momento el usuario perderá control del juego y pasará a observar las consecuencias de sus decisiones. Se mostrará en pantalla cada acción que los Drenars tengan y cómo modifican la salud o los atributos de sus objetivos. Cuando ya no queden acciones (ya sean de los Drenars del

protagonista o los Drenars salvajes), el usuario volverá a tener control del juego. Si ya no quedan Drenars enemigos con vida para combatir, el protagonista habrá ganado la batalla, y si tuvo un poco de suerte con el azar se mostrará en pantalla el botín que pudo obtener de sus enemigos y se actualizan los estados de sus Drenars, ya que obtuvieron experiencia de la pelea. Si en cambio los que quedaron sin vida son los Drenars del protagonista, éste habrá perdido la pelea y el juego. También está la posibilidad de que queden Drenars con vida en ambos bandos. En este caso la batalla debe continuar. Se le ofrece al jugador huir o pasar a elegir las cartas nuevamente.
Este ciclo se repetirá hasta que la pelea concluya.

3. Implementación

Dedicamos esta sección para hablar de los aspectos más interesantes y sus detalles en la implementación del juego, que se separó en partes. Por un lado se implementó el modelo del juego, donde se representan las entidades, sus características y sus interacciones. Por otro lado se trabajó sobre la vista del juego, manejando los eventos, cambios de escenarios y los cambios de interacción. También se trabajó sobre el motor configurable y las diferentes lecturas de archivos que permiten la configuración de muchos datos del juego. Finalmente se completó el módulo de persistencia, permitiendo al juego la posibiliadad de guardar el progreso del jugador para reanudarlo más tarde.

3.1. Modelo

En esta sección hablaremos sobre las decisiones tomadas a la hora de implementar el modelo. Las entidades básicas del juego como los Drenars y sus variantes están modelados usando tipos algebraicos; por ejemplo `WildDrenar`, `BattleDrenar`, etc. cada uno con la colección de características que le dan significado. Una característica interesante de remarcar es que un `WildDrenar` hace uso de funciones de tipo `BattleDrenar -> Bool` para describir si es atrapable fácilmente, almacenando todas estas funciones en una lista que será recorrida con el `BattleDrenar` contenido para saber si el `WildDrenar` cumple todas las condiciones de captura o no.
Otro aspecto interesante en la implementación son los `Buffs`, que contienen la forma en que afectan a los `Drenars`. En su representación se utilizó un tipo algebraico suma donde algunas de sus opciones utilizan funciones junto a un identificador que indica la manera en que modifica un atributo de un `Drenar`.
Siguiendo con elementos más cercanos a la `Battle`, se continúa con las `SpellCards`. Definimos las acciones que se ejecutan en batalla como funciones y las implementamos de manera que el resultado de una sea fácilmente aplicable como argumento de otra. Imaginemos una tupla que contenga un Drenar de Batalla lanzador, un Drenar de Batalla objetivo y un valor númerico que representa el poder de la acción. Una acción entonces contiene una función de tipo `(BattleDrenar, BattleDrenar, Int) -> (BattleDrenar, BattleDrenar, Int)` que retorna los valores actualizados tras aplicarse. Por ejemplo la acción de daño: `Damage ((BattleDrenar, BattleDrenar, Int) -> (BattleDrenar, BattleDrenar, Int))` devuelve al segundo componente de la tupla (Drenar objetivo) con los daños correspondientes y el tercer componente con el valor del daño que lo afectó, calculado con los valores de ataque del Drenar lanzador y los valores de defensa del Drenar objetivo.
Con un tipo algebraico diferenciamos si se mezclan para obtener sus valores iniciales de otra función o del poder inicial de la carta. Estas funciones son contenidas en un tipo

algebraico suma que encapsula las funciones y las diferencia de otras.

3.2. Vista - FunGen

En esta sección explicamos las decisiones tomadas con respecto a la vista del juego. Decidimos utilizar el framework para videojuegos llamado FunGen [Fur02]. Actualmente se encuentra la versión 0.3.
FunGen proporciona la forma de guardar objetos para el juego, formas de interactuar con el usuario, maneras de manipular los objetos del juego, ya sea cambiando su posición o dándoles una velocidad de desplazamiento, y posibilidad de cargar las imágenes a usar, entre otras. Trabaja sobre HOpenGL[Pan09].
Lo elegimos por las ventajas que ofrece, como por ejemplo, los buenos niveles de abstracción y un sencillo manejo de los objetos para el juego. Pero también posee algunas desventajas, ya que el mismo no está finalizado, y faltan algunas abstracciones necesarias, como por ejemplo, la posibilidad de utilizar sonido, o poder capturar todos los input de los usuarios que fueran necesarios. Es necesario remarcar que la página de FunGen resulta más ambiciosa que la realidad y promete información de abstracciones que no posee; el manejo del sonido es un ejemplo.
FunGen permite manipular Estados, y poder realizar diferentes acciones dependiendo el estado en el que se encuentre el juego. En nuestro juego, podemos encontrar 2 estados principales, y varios Sub-Estados dentro de cada uno de ellos.
El Estado `Moving` representa la posibilidad de moverse libremente por un mapa. Tiene varios Sub-Estados que representan las cosas que se puede hacer en ese mapa, como por ejemplo, poder hablar con los `NPC`, quedarse quieto, y salir de un mapa e ir a otro.
El Estado `Fighting` representa una batalla, por el momento solo existe la `WildBattle`. La batalla tiene varios Sub-Estados, que son cada etapa de la misma. Por ejemplo, tenemos una etapa en la que se decide si se quiere luchar o escapar. También hay una etapa de selección de `SpellCards` para atacar, una de selección de `Drenars` que utilizarán las `SpellCards`, y otra para seleccionar los objetivos. Y por último la etapa de combate, que es en la que todos los `Drenars` hacen sus movimientos, ordenados por su velocidad, y cambian el estado de la batalla, y luego vuelve a empezar el ciclo de la pelea o simplemente termina la batalla, ya sea porque hubo una victoria o una derrota.

El módulo Game es donde se obtiene la información de configuración, carga las imágenes, crea todos los objetos del juego, e inicia el Ciclo del Juego. Puede resultar muy criticable esta manera de cargar en un comienzo toda la información de los archivos de configuración; el buen enfoque consistiría en ir cargando los archivos bajo demanda, por secciones y no completa. Hacerlo de esta manera hubiera implicado extender los tiempos del trabajo.
El módulo más importante de nuestro juego es el Ciclo del Juego, o Game Cycle. Es el que se encarga de mostrar qué va a ver el usuario en la pantalla en determinado Estado del juego. Por ejemplo, si el juego se encuentra en el Estado `Moving`, y en el Sub-Estado correspondiente para hablar con un `NPC`, el Game Cycle mostrará el texto del `NPC` y un recuadro a su alrededor, indicando que están hablando.
FunGen maneja colisiones, pero no se utilizaron en el juego, ya que las colisiones de FunGen se dan entre objetos si se solapan o no, pero necesitamos hacer un esquema distinto para determinar colisiones. Los mapas del juego son matrices de números, y cada número representa un objeto del mapa. Al moverse el `Player`, irá pasando por las coordenadas de la matriz, y puede que encuentra un objeto. Si no está permitido que el `Player` pase por arriba de un objeto, habrá una colisión, y no podrá avanzar más. También se verifica esto para los `NPC` que caminan, y tienen en cuenta las mismas reglas de colisión.

3.2.1. Interacción con el Usuario

FunGen da la posibilidad de determinar si se ha presionado una tecla, si se ha soltado, si se sigue presionando, o si se ha clickeado el mouse. Cada uno de estos eventos está relacionado con una acción. En nuestro juego, se pueden utilizar las flechas y algunas teclas extra para cancelar y aceptar opciones. Cada acción determinada se fija en qué Estado se encuentra el juego para decidir qué hacer. Por ejemplo, si el Estado del juego es `Moving`, entonces las flechas permiten que el `Player` se mueva.

3.3. Configuración de datos

En esta sección vamos a explicar cómo hicimos para poder configurar el juego a gusto.
Utilizamos HXT [Sch05], que es una colección de herramientas para el parseo de XML. Elegimos esta herramienta porque provee una rápida lectura de datos de cualquier archivo que tenga la información declarada en forma de tags.
Con HXT, logramos separar cierta especificación de datos, variables relacionadas a la calidad del juego que de otra manera estarían compiladas junto con el código.
Aprovechando esta colección de herraminetas, separamos los siguientes aspectos:

- Especificación de Spellcards, con un módulo que fue escrito para levantar cartas de hechizos de un archivo XML. Con esto se puede experimentar una variedad de cartas que el juego contendrá.
- Especificación de Drenars, detallando sus características principales para luego obtener diferentes variedades de un mismo Drenar.
- Especifiación del Loot o recompensa que un Drenar otorga tras ser derrotado. Utilizando la información parseada de los módulos anteriores, se logró relacionar un Drenar con una serie de Spellcards que contiene como Loot.
- Distribución de Drenars en una zona. De esta manera se puede manipular la zona en la que un Drenar aparecerá para luchar contra él en estado salvaje.
- Condiciones de captura para cada Wild Drenar, con el parseo de dos opciones posibles, una que verifica determinado Stat a un valor, con la relación de mayor, menor, o igual, y la opción que chequea la presencia de determinado buff.
- Especificación de las imágenes de todo el juego
- Especificación de un Player básico, el cual se utilizará la primera vez que se empieza a jugar.
- Especificación de cada NPC del juego.
- Y finalmente, las diferentes SpellCards que están bajo el uso de determinado Wild Drenar

El uso de Arrows nos resultó complejo, y produjo dificultades iniciales que se resolvieron luego de comprender los conceptos básicos. A partir de ahí pudimos construir la capa de configuración, y continuar profundizando en las nociones conceptuales importantes.
La interacción con HXT se resume en: pedir que lea un archivo y arme los árboles de nodos de texto, lo procesamos con reglas propias y devolvemos una estructura que el juego utiliza.

3.4. Persistencia

El concepto de Persistencia en un juego reside en darle la posibilidad al usuario de guardar su progreso. De esta forma puede reanudar su juego en otro momento sin verse obligado a comenzar de nuevo. Para esto decidimos utilizar Acid-State [Him11], una biblioteca de persistencia de datos NoSQL en la memoria RAM. La particularidad por la que la elegimos es su diseño para almacenar tipos de datos arbitrarios de Haskell con código simple.
Para persistir el juego agregamos en otro módulo información de los diferentes tipos algebraicos involucrados con el Player para que fueran instancia de la clase SafeCopy, que es una extensión de serialización binaria con control de versiones.
Finalmente implementamos las funciones que guardan y traen el juego. Los datos relevantes a persistir son el Player, el nombre del mapa donde está ubicado, su posición en el mismo, y qué NPCs están ocultos.

4. Trabajo Futuro

Consideramos nuestro trabajo finalizado, de manera que las mecánicas del mismo están completas, pero aún faltan muchos recursos para que sea considerado un juego terminado, y para poder hacerlo, necesitaríamos agregar más y distintos Drenars para que sea más variado, así también como sus ilustraciones. También harían falta muchas más SpellCards, para abarcar todas las posibilidades que podemos construir con nuestro modelo. Haría falta completar la historia para que el usuario tenga una experiencia más llevadera a la hora de jugarlo. Necesitaríamos agregar misiones, ya que las que aparecen en el modelo de presentación actual son solo de muestra y muy simples, y también faltarían diversas zonas o mapas nuevos para explorar. Uno de los aspectos que no se puede configurar facilmente es la musicalización, ya que no logramos hacerla funcionar. Otro detalle que nos gustaría agregar es una modalidad multijugador, para la cual serían necesarios algunos cambios en el juego, agregando alguna modalidad que no existe, ya sea batallando con otros usuarios o intercambiando SpellCards o Drenars.
Consideramos que nuestros objetivos del juego están cubiertos en su totalidad, puesto que agregar todos los detalles faltantes requieren mucho más tiempo, y no agregarían valor al trabajo en sí con respecto a nuestra formación técnica. Cabe destacar que gracias a los archivos de configuración se pueden seguir agregando Drenars, SpellCards, mapas (siempre agregando sus imagenes correspondientes) e interacciones con más NPCs.

5. Conclusiones

Nuestro desarrollo tuvo como objetivo completar una demo, y no un juego completo, mostrando una esquematización de un juego, elaborando los conceptos básicos, sus conexiones y su configurabilidad. El foco fue prototipar un juego, mas no uno cualquiera. Un juego simple completo podría haberse terminado en el mismo tiempo que duró el desarrollo de nuestro prototipo. Nuestra finalidad fue ofrecer el concepto del mismo con contenido no elemental. Esta característica es una de las razones principales por las cuales elegimos Haskell: la comodidad para modelar y prototipar. Por ello nuestra intención no es terminar de desarrollar una versión completa del juego, sino plantear la forma que debería tener Drenar's World cuando se implemente en versión completa. Debido a esto hubo aspectos que nos vimos obligados a sacrificar. Uno de ellos fue la falta de consideración al respecto del manejo de memoria: no buscamos eficiencia a toda costa.

Lo último con lo que queremos concluir es sobre el soporte cognitivo que nos brindó la carrera. El aspecto principal que se transmite es el foco en expresar las ideas independientemente del lenguaje. Con investigación y tiempo, un egresado de TPI podría aprender cualquier lenguaje y paradigma sin mayores dificultades. En nuestro caso, podemos hablar sobre la experiencia con el lenguaje Haskell. Si bien ya habíamos tenido contacto con el lenguaje, no habíamos utilizado ninguna de las bibliotecas que elegimos para programar el juego. Estas bibliotecas involucraron conceptos nuevos, pero no constituyeron un impedimento para las habilidades que adquirimos.

Referencias

[Fur02] Andre Furtado. Functional Game Engine, 2002.

[Him11] David Himmelstrup. Acid-state, 2011.

[Jon02] Simon Peyton Jones. Haskell 98 language and libraries: The revised report, December 2002.

[KR88] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language.* Prentice Hall, Inc., 2nd edition, 1988.

[Pan09] Sven Panne. Haskell opengl bindings, 2002 - 2009.

[Sch05] Uwe Schmidt. Haskell XML Toolbox, 2005.

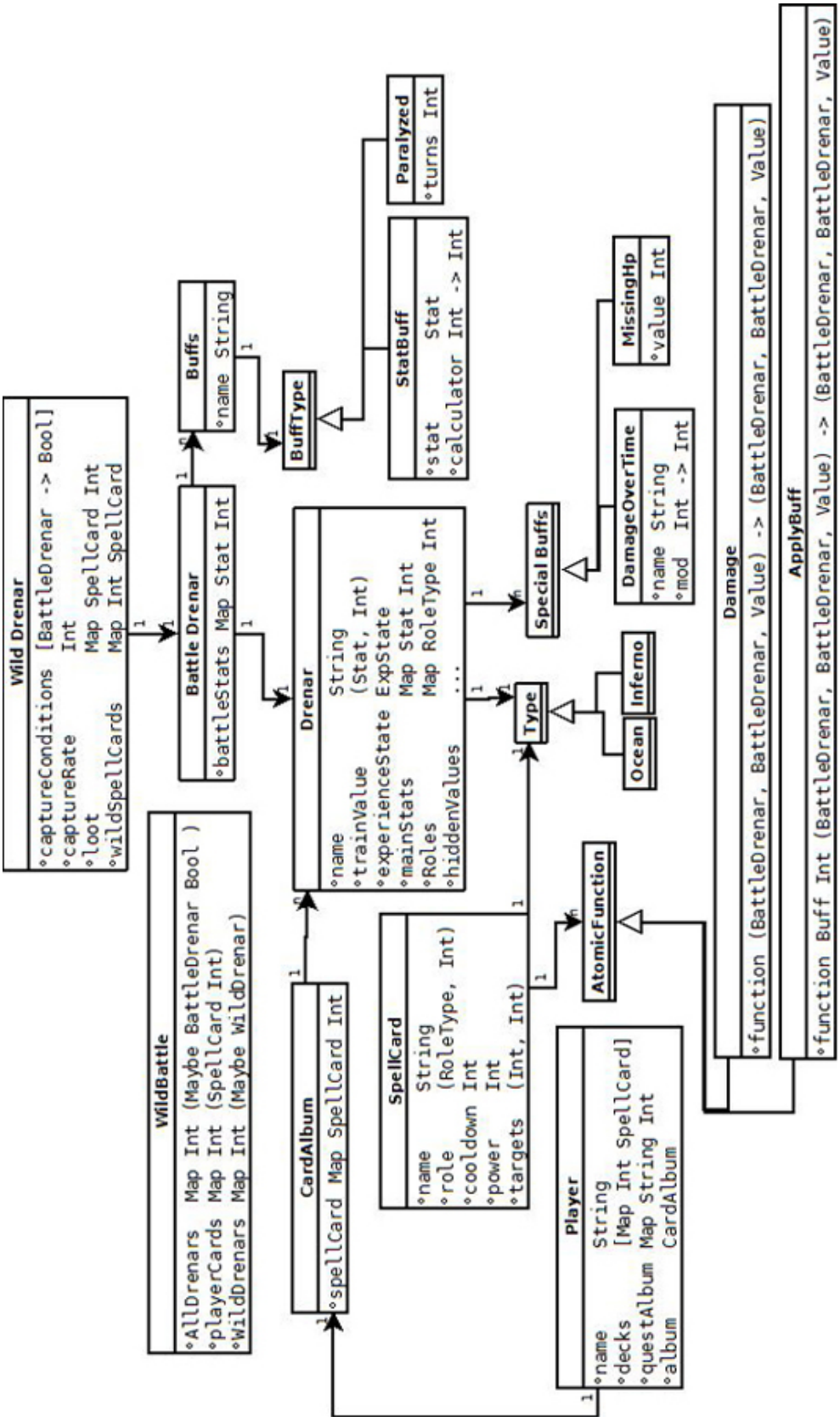
[Str85] Bjarne Stroustrup. *The C++ Programming Language.* AddisonWesley, Inc., 1985.

6. Agradecimientos

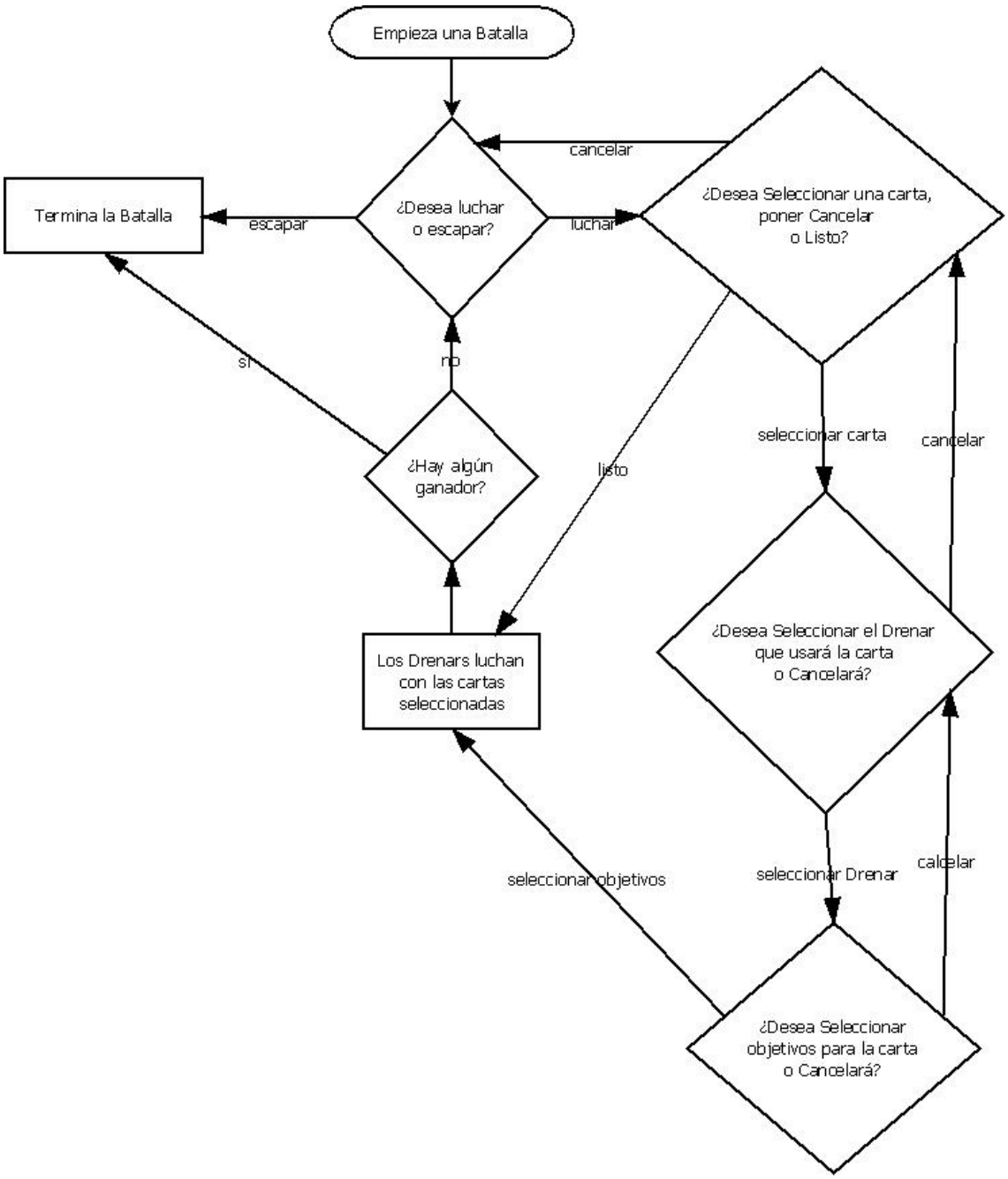
Se agradece la colaboración de las siguientes personas:

- A Maria Laura Medici, por dibujar varios Drenars.
- A Tatiana Ollero, por dar ideas de NPCs.
- A Germán Pérez, por componer una canción.
- A Leonel Martos, por ayudar a componer una canción.
- A Pablo E. "Fidel"Martínez López, por hacerse cargo de la dirección del proyecto, por su ayuda con ideas para la mejora del mismo y por ayudar a redactar el informe.

A. Diagrama de Entidades



B. Diagrama de flujo de una Batalla



C. Sobre los Drenars

En este apéndice describimos de manera completa los atributos de los Drenars.

Afinidad - Tipo de Drenar - (Type)
Cada Drenar tiene un tipo, o Afinidad Elemental. Elegimos Inferno, Ocean, Lighting, Nature, Light, Shadow y Void. Su existencia se debe a la modificación del ataque de un Drenar. Si un Drenar de un tipo determinado usa una habildad del tipo opuesto, su ataque debe verse debilitado, por ejemplo.

Atributos Principales - (Stats)
Los Atributos principales son un conjunto de estadísticas que definen al Drenar, y los llamamos Stats. Los Stats de un Drenar son su vida, su poder, su resistencia mágica, y su velocidad. Diferentes Drenars tienen diferentes Stats, algunos con más vida que otros, más resistencia, etc.

Atributos Ocultos - (Hidden Values)
Los atributos ocultos son valores que representan el crecimiento de un Stat por cuenta del Jugador, aunque no es visible por éste. Contienen un valor básico para cada Drenar, y representan la manera de subir un Stat determinado naturalmente. Creemos, por ejemplo, que es normal que un monstruo con características de defensa tienda a subir más rápidamente su resistencia mágica que otro.

Estado de Experiencia - (ExpState)
El entrenamiento de un Drenar se representa con la mejora de sus Stats y también se le da un número, que determina el Nivel en el que se encuentra. Tener un determinado Nivel da determinados Stats básicos. Para subir el Nivel de un Drenar se deben ganar batallas contra otros Drenars, y obtener experiencia. Cada nivel tiene su monto de experiencia necesaria para superar ese Nivel. Cuando se gana experiencia, al mismo tiempo se actualizan sus Valores Ocultos antes mencionados. Otra característica que cabe destacar es que algunos Drenars ganan experiencia más rapido que otros.

Roles - (Roles)
Dado que las batallas de los Drenars pueden darse de hasta tres contra tres, pensamos que sería correcto que cada uno pueda cumplir ciertos roles en ellas, y lo dividimos en varias categorías: el que cura a los demás Drenars, el que realiza el daño, el que agrega Auras a los Drenars, el que las quita, y el que puede recibir más daño que los demás por tener más Resistencia mágica o vida que el resto. Hay un Rol especial que es el de poder capturar Drenars. La posibilidad de capturar Drenars se da mediante este Rol de algún Drenar que se posea, y la carta de hechizo que se utilize con el Drenar para capturarlo. Estos Roles indican qué Cartas de Hechizo puede usar un Drenar y qué cartas no.

Condiciones de Captura - (Capture Conditions)
Para atrapar un Drenar salvaje con algún grado de éxito se deben cumplir ciertas condiciones. Si las condiciones no se cumplen, la criatura dispone de su índice básico de captura que es mucho más bajo. Un ejemplo de condición de captura puede ser que el Drenar tenga un determinado porcentaje de vida.

D. Player

En este apéndice describimos la información que contiene un Player.

Mazos - (Decks)
Los mazos, o Decks, son grupos de SpellCards, que se usarán en las batallas. Cada Deck contiene como máximo 9 cartas. El Player puede tener varios Decks para utilizar, pero solo podrá usar uno por batalla.

Grupo de Drenars - (Party)
El Grupo de Drenars, o Party, está compuesto por hasta 3 de todos los Drenars que tenga el Player. Los Drenars que estén en el Party son los que lucharán cuando se inicie una batalla.

Información personal - (Info)
La Información personal, o Info, contiene algunos datos del Player, como por ejemplo, su nombre, y el rango en el juego que tiene. El rango sube mientras se avanza en la historia del juego. Lamentablemente esta información no se muestra actualmente en la demo.

Álbum de cartas - (Card Album)
El Album de cartas, o Card Album, es el contenedor de todas las cartas que consiga el jugador, que se guardarán aquí hasta que decidamos utilizarlas, ya sea en el Party (los Drenars) o en algún Deck (SpellCards).

Tabla de misiones - (Quest Table)
La Tabla de misiones, o Quest Table, contiene los nombres de todas las Quests que puede realizar el Player, y en qué momento se encuentra de cada una. Esto sirve para que principalmente los NPC cambien sus diálogos dependiendo en qué parte se encuentre el jugador en determinada Quest.

E. Buffs

En este apéndice explicamos los Tipos de Buffs.

Tipo de Buff - (Buff Type)
Un Tipo de Buff, o Buff Type, representa la acción de un Buff. Es la manera en que modifica o trabaja sobre un Drenar. Hay diferentes Buff Types, y por ende, diferentes Buffs, entre los cuales podemos encontrar: modificación de un Stat del Drenar, regeneración de vida, modificación del daño, disminución de vida, etc.

F. SpellCards

En este apéndice describimos las características de las SpellCards.

Recuperación - (Cooldown)
Este concepto representa el tiempo de turnos que se requieren para que la Spellcard vuelva a utilizarse.

Función Atómica o Unidad Elemental - (Atomic Function)
Las funciones atómicas, o Atomic Functions, son la mínima unidad de acción de una SpellCard, y están definidas las suficientes para crear todo tipo de SpellCard. Estas Atomic functions se combinan para formar SpellCards más complejas. Cada Atomic Function necesita un valor de poder, que puede obtenerlo a partir de la SpellCard o de la anterior Atomic Function; esto se explica más adelante. Algunos de los valores que definimos pueden ser: Daño porcentual, curación porcentual, aplicar Buff, capturar Drenar, etc.

Funciones Compuestas - (Composite Functions)
La acción de una carta está definida en una lista de Funciones compuestas, o Composite Functions. Esta información existe para representar la forma en que las acciones de una carta sucedan en la batalla. La manera de establecer estas acciones fue modelada como una función atómica, o Atomic Function, que es la mínima acción que puede realizarse. El objetivo de plantearlo de esta manera fue poder decir que una SpellCard, además de la información antes mencionada, fuera creada a partir de unidades elementales que se componen y dan un resultado mayor.

Formas de Composición
La forma de relacionar las Atomic Functions es mediante composiciones de las mismas. hay 3 formas de composiciones:

Simple - (Single) Una composición Simple, o Single, no tiene relación estrecha con la siguiente Composite Function. En una encadenación de varias Single, cada una tomará como poder el valor original de la carta. Por ejemplo, si una carta hace que el Drenar golpee una vez a N objetivos con el poder básico de la carta, la misma se representa como N singles.

Compuesta - (Composite) Este tipo de Composite Function relaciona dos Composites Functions y hará que el valor resultante de la primera de las acciones sea utilizado por la segunda en la composición. Por ejemplo, si tenemos una SpellCard que hace que un objetivo reciba daño y al mismo tiempo que el Drenar que la utiliza se cure la misma cantidad de daño hecho, se representará como un Composite de dos Singles, uno de Daño y otro de Curación.

Compuesta Distributiva - (Distributive Composite) Finalmente, la necesidad de aplicar un mismo valor resultante de una Función Compuesta a otro par da lugar a este nuevo tipo. Por ejemplo, una SpellCard que haga daño a un Drenar enemigo y al mismo tiempo cure a 2 Drenars aliados el valor del daño que hizo, las acciones están representadas como una Composite de un Single de Daño con un Distributive Composite de dos Single de Curaciones.

G. Battle

En este apéndice explicamos algunos aspectos relacionados a las Batallas.

Battle Actions - Acciones de Battalla
Son la agrupación de la información de las acciones que se realizarán en la batalla. Contiene datos del Battle Drenar lanzador, los objetivos, la carta a utilizarse, la velocidad del lanzador y quién está realizando la acción.

Entorno de Batalla - (Battle Environment)
Un entorno de batalla, o Battle Environment, da fluidez a la misma. Contiene información que representa los turnos en una batalla y su sucesión. Cualquier pelea está representada como un Battle Environment que contenga la batalla correspondiente con los cambios del último turno, una lista de Battle Actions, el Player, y qué tipo de batalla está representando.

Battle Type - Tipo de batalla

El tipo de batalla le indica al Battle Environment cómo ejecutar las acciones y qué respuesta debe dar en caso de derrota o a la finalización de un turno. Un Battle Type puede ser o bien WildBattleType o NPCBattleType y en este último caso contener infomación del NPC originador. Lamentablemente el NPCBattleType no se utiliza en la demo actual, ya que este tipo de batalla es casi idéntico al WildBattleType, y es agrega mucho valor a la demo

Cuando el Battle Environment pase al siguiente turno, ordenará las Battle Actions que contenga en orden por la velocidad del Drenar lanzador y las irá ejecutando sobre la batalla actual. Esto nos devolverá un nuevo Battle Environment con la batalla resultante. En caso de que el jugador perdiera o ganara el turno anterior, no se procede a uno nuevo sino que se termina la batalla con las consecuencias correspondientes.