

ERM dla opornych

autorstwa Jakuba Grzany

Wstęp

Event Related Model, w skrócie ERM, to skryptowy język programowania dodany w modyfikacji In the Wake of Gods. Umożliwia bardzo głębokie modyfikowanie sposobu, w jaki zachowuje się gra: od podmiany zachowania obiektów, przez programowanie nowych, aż po zmianę tak podstawowych mechanik tak podstawowych jak sposób obliczania obrażeń w trakcie bitew. Do nauki wymagana jest jedynie dobra znajomość angielskiego.

Przykłady działania skryptów ERM



Od lewej: 1) Wiedźma nie wmusza nam już np. sokolego wzoru, gracz może odmówić przyjęcia umiejętności. 2) Siedlisko po kliknięciu PPM wyświetla typ generowanych stworów oraz ilość dostępną do rekrutacji. 3) Nowy tryb gry: Dominacja. Wygrywa ten, kto pierwszy dobije do 100 punktów. Punkty otrzymuje się za kontrolę obiektów: w tym przypadku zamku.

Narzędzia

Do sprawnego wykorzystania ERM będziemy potrzebować: gry Heroes 3 z modem In the Wake of Gods 3.58f albo ERA, edytora map h3, edytora tekstuowego erm_s.exe oraz dokumentacji ERM Help.

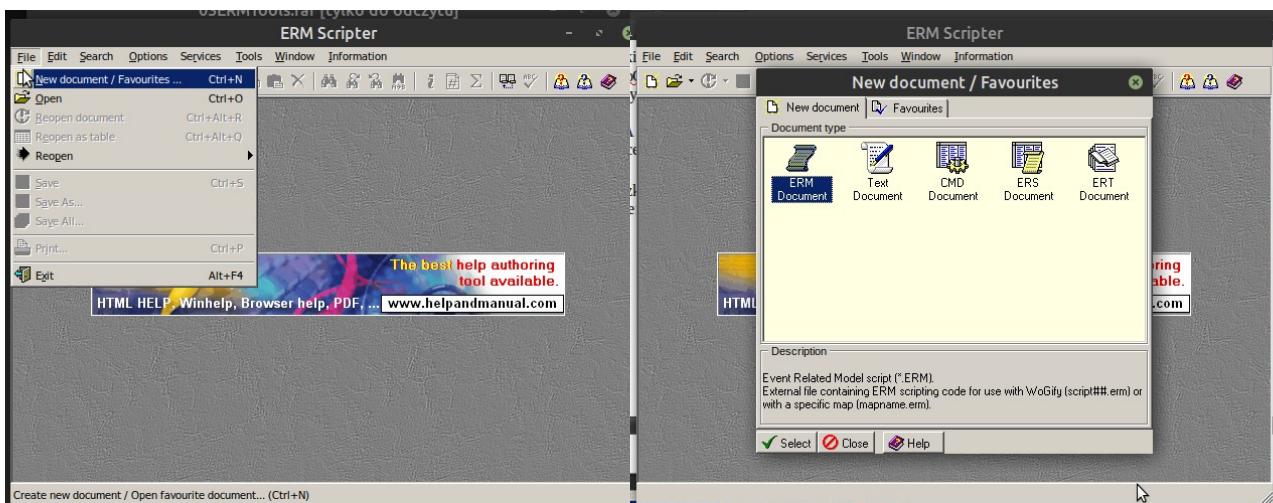
Linki:

- WoG 3.58f, wraz z instrukcją instalacji: [klik](#)
- Edytor i dokumentacja: [klik](#)

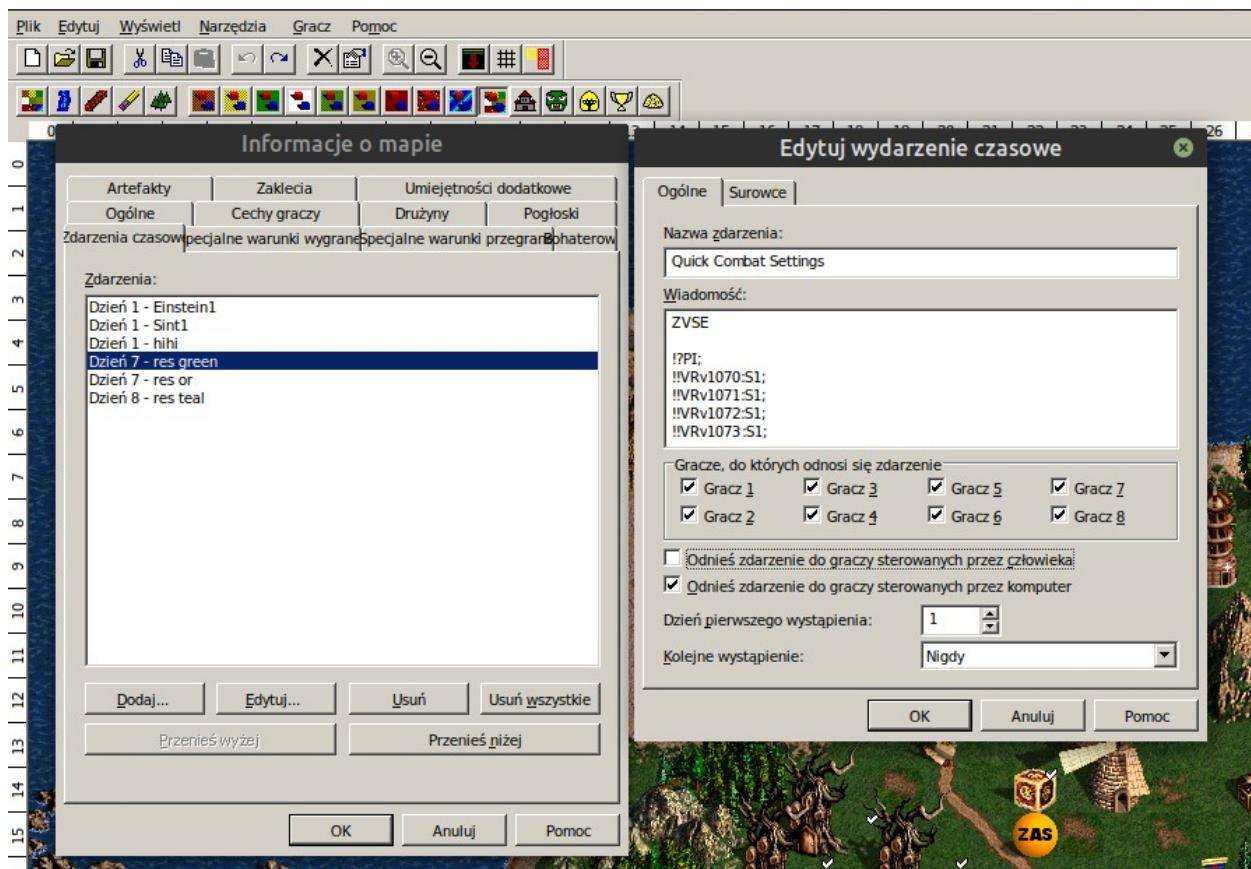
ERA to kontynuator projektu WoGa, pisanie dla niej prawie nie różni się od pisania dla 3.58f, więcej znajdzicie w internecie, ale nie zaprzatajcie sobie tym głowy na starcie.

Edytor map do WoGa znajdziemy w katalogu głównym Heroes 3, nazywa się H3WMPED.exe.

Paczka edytor + dokumentacja zawiera zmodyfikowaną przeze mnie dokumentację 3.58f. Skrypty piszemy za pomocą programu erm_s.exe w katalogu erm_s. Nowy skrypt tworzymy wybierając New -> New Document -> ERM Document. Do dokumentacji erm_help wróćmy później.



Skrypty umieszczamy w zdarzeniach czasowych na mapie. Ich nazwa, dzień pierwszego wykonania, gracze do których się odnosi nie mają znaczenia. Osobiście polecam ustawić pierwsze wystąpienie na 500 dzień – aby nie mieszały się z innymi zdarzeniami czasowymi na mapie – oraz odnieść zdarzenie tylko do graczy komputerowych. Uwaga: skrypty działają tylko jeśli mapa jest w formacie WoGa!

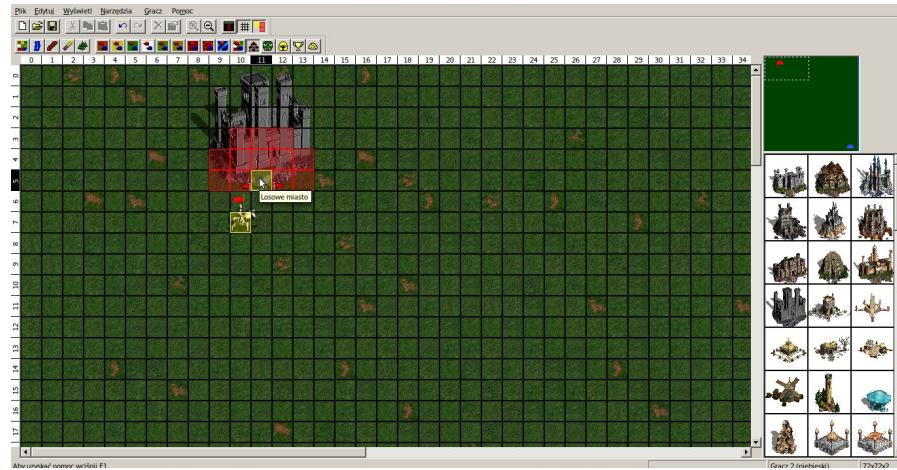


Tworzenie zdarzenia czasowego: Narzędzia -> Zdarzenia Czasowe -> Dodaj

Lekcja pierwsza – Witaj świecie?

Skoro środowisko jest już gotowe, czas na pierwszy skrypt. Nie będę póki co tłumaczył zbyt dokładnie jak to działa – ale domyślam się że Ty chciałbyś zobaczyć efekty na własnym ekranie, nieprawdaż? :)

Stwórz nową mapę i wypełnij trawą. W lewym górnym rogu umieść zamek gracza czerwonego tak, by jego wejście (żółty kwadrat) znajdowało się na koordynatach 11/5/0. W prawym dolnym rogu umieść gracza niebieskiego, by dało się mapę uruchomić. Całość ma prezentować się tak:



Teraz czas na skrypt. W zdarzeniu czasowym mapy umieść to:

ZVSE

```
! ?0B11/5/0;  
!!IF:Q1/21/172/1^You are going to have baaaad time^;
```

Gotowe. Wystarczy zapisać mapę do katalogu Maps, i uruchomić grę, po czym odwiedzić bohaterem zamek. Efekt? Wait what



Dobra żartuję, tak naprawdę zobaczycie to. Ale spokojnie, do bardziej złożonych skryptów też przejdziemy (btw. zwrócić uwagę na balistę w creature banku)



Krótkie wyjaśnienie działania:

- ZVSE to nagłówek skryptu. KAŻDY skrypt ERM musi się od niego zaczynać. Daje znać grze, że „oho to jest skrypt ERM, muszę go wykonać”.
- !?OBx/y/z; reaguje na bohatera odwiedzającego obiekt położony na koordynatach x/y/z. Gdy to nastąpi, powoduje wykonanie kolejnych poleceń.
- !!IF: to polecenie odnoszące się do „InterFace’u” użytkownika. Wyświetla wiadomość. Nie wykona się jednak, jeżeli nie jest poprzedzone polecienniem wyzwalającym. 21/172 oznacza „załącz obrazek potwora(21), konkretnie Nightmare(172). ^text^ to tekst do wyświetlenia. Możesz umieszczać w nim wszystko (wliczając entery) poza znakiem ^.

Lekcja druga – Składnia

W tym dziale – sucha teoria, którą musisz znać i rozumieć by pisać większe skrypty ERM. Jak już wiesz, wszystkie skrypty (a konkretnie – pliki/zdarzenia ze skryptami) MUSZĄ zaczynać się od napisu **ZVSE**. Jest to jedyne słowo kluczowe. Oprócz tego, ERM operuje jedynie na poleciennach, dzielących się na trzy grupy:

- Triggery – Zaczynają się od znaków **!?** lub **!\$**, rozpoczynają „blok” poleceń. Po wystąpieniu zdarzenia powiązanego z danym triggerem (np. odwiedzenie obiektu) gra wykonuje występujące po nim polecenia, aż trafi na koniec pliku lub inny trigger.
- Receivery – Zaczynają się od znaków **!!** i wykonują się tylko jeśli występują po „załączonym” triggerze. To one wpływają na zachowanie gry, np. wyświetlając okienka czy rozpoczynając walki.
- Instructions (instrukcje) – Zaczynają się od znaków **!#** i pod względem składni są identyczne z receiverami. Natomiast w przeciwieństwie do nich są niezależne od triggerów. Wykonują się tylko raz, przy rozpoczęciu gry. Rzadko wykorzystywane.

Każde polecenie składa się z przedrostka określającego typ(!?,!!), dwuznakowej nazwy polecenia(IF,OB), opcjonalnych parametrów (11/5/0 dla triggera OB) i średnika (;) kończącego polecenie. Składnia receiverów oraz instrukcji jest rozszerzona o komendy, występujące po dwukropku (:), dla przykładu !!IF:Q..., wraz z argumentami (np. 1/21/172/1^^);

Bardzo ważnym elementem składni, który jednak nie zalicza się do poleceń, są komentarze. Są to fragmenty skryptów które nie są wykonywane i służą wyłącznie poprawieniu czytelności. Przynajmniej składnia jest prosta: wszystko co nie jest polecienniem jest komentarzem ;)
Dobrze jest komentować wszystko, linijka po linijce, zwłaszcza gdy wprowadzamy nową zmienną do skryptu – należy opisać co przechowuje. ERM nie należy do języków które łatwo się czyta.

ZVSE

```
!PI; [Trigger uruchamiajacy się po wywołaniu wszystkich instrukcji]
!IF:Q1/21/43/1^Jestem uzyteczna jednostka^; [To się pojawi jako drugie]
#!IF:Q1/21/37/2^To sie pojawi przy starcie gry^; [To się pojawi jako
pierwsze]
?OB53; [Trigger uruchamiajacy się gdy odwiedzisz dowolna kopalnie]
!IF:Q1/21/192/1^I'm a mistake^; [To się pojawi, gdy odwiedzisz kopalnie]
```



Przy okazji postanowiłem rozstrzygnąć odwieczny spór: według ERM jest KOPALNIA drewna.

Lekcja trzecia – Zmienne w teorii i praktyce

Zmienne w językach programowania służą do przechowywania informacji, głównie liczb oraz napisów. ERM co prawda nie posiada opcji „tworzenia” zmiennych, natomiast mamy kilkanaście tysięcy dostarczonych przez twórców do wykorzystania. Typów zmiennych jest sporo, tutaj omówię tylko te najbardziej podstawowe.

Zmienne ‘v’, v1-v10000 – zmienne globalne, przechowują liczby całkowite (-1,0,1,2 itd.) Ich wartości są zapisywane w pliku zapisu. „Globalność” oznacza że wszystkie skrypty mają do nich dostęp, i np. v12 we wszystkich będzie mieć taką samą wartość.

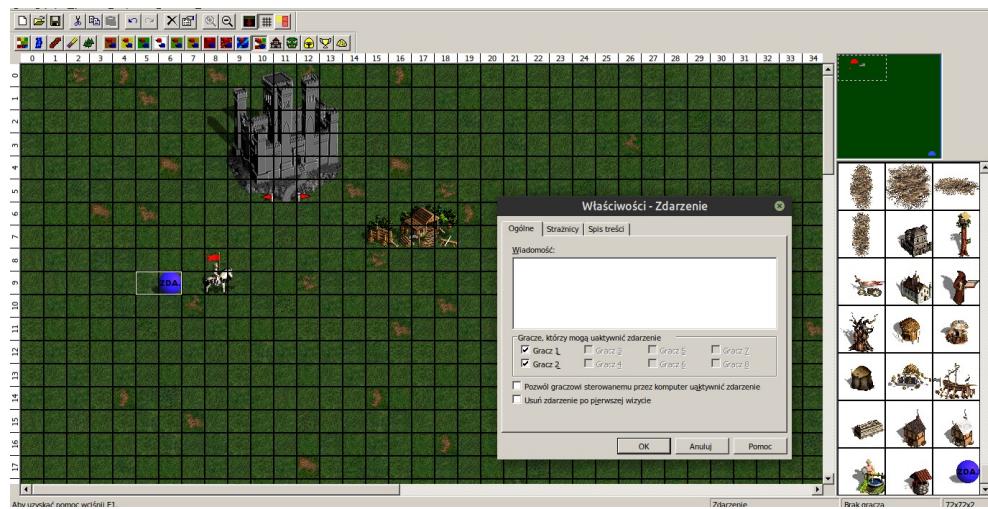
Zmienne ‘y’, y1-y100 – zmienne lokalne, tymczasowe, przechowują liczby całkowite. Każda funkcja, blok w skryptach posiada własny zestaw zmiennych ‘y’. Nie zapisują się w pliku zapisu, służą wyłącznie do tymczasowego przechowywania danych, na których wykonujemy operacje.

Zmienne ‘z’, z1-z1000 – zmienne globalne, przechowują napisy. Zapisują się w pliku zapisu. Mogą mieć maksymalną długość 512 znaków.

Zmienne ‘z-’, z-1 – zmienne lokalne, tymczasowe, przechowują napisy.

Zmienne ‘e’, e1-e100 – zmienne lokalne, tymczasowe, przechowują liczby „zmiennoprzecinkowe”. Innymi słowy, przechowują ułamki (3.69215...)

Huh, sporo tej teorii, czas na praktykę. Wróćmy do naszego testowego scenariusza i umieścmy zdarzenie (event) na mapie na koordynatach 6/9/0. Odznaczamy „usuń zdarzenie po pierwszej wizycie” oraz zaznaczamy wszystkich graczy, dla których skrypt ma działać. Treść zostawiamy całkowicie pustą.



Następnie dodajemy do zdarzeń czasowych taki skrypt:

```
ZVSE
!#VRv2:S0;
!?LE6/9/0;
!!VRv2:+1;
!!VRy1:Sv2;
!!VRz2:S^Number of Qurqirish Dragons' in this tutorial: %Y1^;
!!IF:M^%Z2^; [Stary znajomy, choc w nieco innej formie]
```

Odpalamy grę i przesuwamy bohatera na pozycje zdarzenia. Powtarzamy kilka razy. Jak widać, skrypt zlicza ile razy odwiedziłeś ten konkretny punkt na mapie i wyświetla informacje o tym w wiadomości. Fajne, cnie?



Przejdźmy do omówienia co się tutaj właściwie dzieje. Jak widzisz, w skrypcie pojawia się nowy trigger: **!?**LE, oraz nowy receiver: **!!**VR. Trigger jest zdecydowanie prostszy, więc zacznijmy od niego.

!?LE to trigger powiązany ze zdarzeniem (eventem) na mapie przygody. Przyjmuje trzy parametry: x(6), y(9), l(0). Załącza się tylko gdy zdarzenie jest wykonywane dla danego gracza. Jeżeli zdarzenie zniknie bądź będzie wyłączone dla danego gracza, trigger nie zaskoczy.

!!VR to receiver odpowiedzialny za operacje na zmiennych. Przyjmuje jeden parametr: zmienną na której operujemy, w tym przypadku są to v2, y1 oraz z2. Ma sporo argumentów do wykorzystania, więc jak zwykle omówię te najbardziej podstawowe. Warto zaznaczyć że składnia tych operacji zależy od typu zmiennej na której operujemy.

!!VRv1:S10 +5 -2 *3 :2 R5;

Powysze polecenie oznacza: operuj na zmiennej v1. Ustaw wartość tej zmiennej na 10 (S10), dodaj 5 (+5), odejmij dwa (-2), pomnóż przez 3 (*3), podziel przez 2 (:2), dodaj losową liczbę od 0 do 5 włącznie (R5). Operacje są wykonywane po kolejno, od lewej do prawej, kolejność wykonywania działań tu nie obowiązuje. Wynik: 19+(0..5), przez wartość losową nie da się jednoznacznie odpowiedzieć.

!!VRvy1:Sy2 +3 -y1;

Operuj na zmiennej v o indeksie y1. Ustaw wartość tej zmiennej na y2 (Sy2), dodaj 3 (+3), odejmij wartość y1 (-y1)

!!VRz2:S^Lala %Y2^;

Operuj na zmiennej z2. Ustaw jej wartość na ^Lala %Y2^. W miejscu %Y2 wstawiona zostanie wartość zmiennej y2. W ten sposób możemy do ciągu znaków wpisać wartość dowolnej zmiennej: v, y, e, także z. Wystarczy wpisać znak % oraz dużą literą nazwę zmiennej wraz z indeksem.

!#VRv2:S0;

Tym razem instrukcja, nie receiver, ale jak mówiłem są one do siebie bardzo podobne. To konkretne polecenie inicjuje zmienną v2, wpisując do niej na samym początku gry wartość 0. Teoretycznie wszystkie zmienne w języku ERM zaczynają z wartością 0, natomiast inicjowanie ich to bardzo dobry zwyczaj – dotyczy to również zmiennych lokalnych/tymczasowych.

Lekcja czwarta – Pod warunkiem że

Zmodyfikujemy trochę napisany wcześniej skrypt, tak by wyświetlał trzy różne wiadomości: jedną gdy v2 jest większe od 2 a mniejsze od 10, drugą gdy v2 jest większe-równe 10, trzecią gdy v2 równe 10. Przy okazji upraszczamy.

ZVSE

```
!#VRv1:S0;  
!LE6/9/0;  
!!VRv1:+1;  
!!IF&v1>2/v1<10:M^%V1 jest większe niż 2 a mniejsze niż 10^;  
!!IF&v1=10:M^%V1 jest rowne 10^;  
!!IF&v1>=10:M^%V1 jest wieksze-rowne 10^;
```

Większość skryptu powinna być dla Ciebie jasna, to co nas interesuje oznaczyłem tym kolorem. Znak & oznacza „jeżeli wszystkie z podanych warunków spełnione”. Istnieje jeszcze drugi znak | oznaczający „jeżeli jakikolwiek z podanych warunków spełniony”. Po znaczkach & i | występuje ciąg warunków połączonych znakiem / oznacza on i/lub w zależności od kontekstu.

Warunki wstawiamy w poleceniach po nazwie wraz z parametrami, przed dwukropkiem (w przypadku receiveró) lub średnikiem w przypadku triggerów. Np trigger **!?OB1/2/3&v1=10;** uruchomi się gdy odwiedzony zostanie obiekt na koordynatach 1/2/3, a zmienna v1 ma wartość 10.

Warunki do wykorzystania. W miejsce L,P można wstawić dowolnej zmienne liczbowe.

- L równe P: ‘L=P’ – Zadziała również dla napisów (zmienne z)
- L mniejsze od P: ‘L<P’
- L mniejsze-równe P: ‘L<=P’
- L większe od P: ‘L>P’
- L większe-równe P: ‘L>=P’
- L różne od P: ‘L<>P’ – Zadziała również dla napisów (zmienne z)

Lekcja piąta – Wiadomości, zapytania, flagi

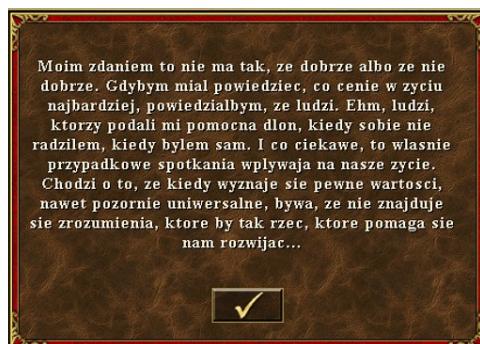
Do komunikacji z graczem wykorzystujemy receiver IF, pozwalający na wyświetlanie wiadomości oraz zadawanie mniej lub bardziej złożonych pytań. Zaczniemy od najprostszego i znanego ci już polecania IF:M.

IF:M to proste polecenie pozwalające wyświetlić wiadomość z przyciskiem „Ok”, tak jak w normalnych zdarzeniach losowych bądź czasowych. Wywołanie wygląda następująco:

!!IF:M^Wersja z ciągiem znaków^;

!!IF:M1/z#;

Obydwa wywołania dają taki sam efekt, różni je tylko sposób dostarczenia ciągu znaków. W pierwszym wykorzystujemy ciąg znaków zawarty wewnętrz ^^, w drugim podajemy go przez zmienną z#. W miejsce # należy wstawić index z-zmiennej w której przechowujemy ciąg znaków. W grze wygląda to tak:



IF:Q jest dużo bardziej złożonym poleceniem. Służy do wyświetlania wiadomości z obrazkami, zadawania pytań zamkniętych (Tak/Nie), wyświetlania tzw. Hintów (opisów dla obiektów wyświetlanych po naciśnięciu prawego przycisku myszy) To nie wszystko, ale póki co wystarczy. Przejedźmy zatem do składni:

!!IF:Q#1 /#2/#3/ ... #4^^;

#1 to flaga, do której zapisana zostanie „odpowiedź” użytkownika. Flagi to specyficzny rodzaj zmiennych które mogą przyjąć wyłącznie wartość 0 (false) oraz 1 (true). W przeciwieństwie do innych zmiennych, nie mają powiązanej z nimi literki, wyłącznie numer, od 1 do 1000. Możemy używać ich w sprawdzaniu warunku: polecenie **!!IF&1:XXXX** wykona się tylko jeśli flaga 1 przechowuje wartość 1 (true). Warto zaznaczyć że w części przypadków flaga ta nie jest tu faktycznie używana.

#2 to typ obrazka, #3 to jego podtyp. Szczegóły na ten temat znajdziesz w dokumentacji do której wróćmy w następnym rozdziale. Póki co niech wystarczy ci wiedza że typy od 0 do 7 oznaczają różnego rodzaju surowce a podtyp ich liczebność, natomiast typ 21 oznacza obrazek potwora.

W miejscu ‘...’ można wstawić do dwóch par „typ/subtyp”. Pozwala to na wyświetlenie do 3 obrazków.

#4 to typ wiadomości (jest ich więcej, tutaj tylko najpopularniejsze)

1 oznacza zwykłą wiadomość, jak w IF:M.

2 oznacza pytanie zamknięte (tutaj wykorzystuje się flagę: OK=1, Cancel=0)

4 oznacza wiadomość bez guzików, wyświetlana do puszczenia przycisku



Wykorzystane polecenia:

```
!!IF:Q1/5/21/1^Jestem pierdolniczkiem^;
!!IF:Q1/2/21/2^Czy jestem kamieniem?^;
!!IF:Q1/7/6/21/112/4^Troche tego jest^;
```

W przypadku pytania zamkniętego możemy sprawdzić odwiedź gracza poprzez warunek , na przykład **!!IF&1:M^Jesli flaga 1= true, wyswietl te wiadomosc^;**

IF:G to chyba mój ulubiony receiver. Pozwala na wybór jednej albo wielu opcji z listy (do 12 pozycji) Tym razem zacznijmy od tego jak to wygląda



Składnia: **!!IF:G#1/#2/#3/#4 /#5/.../#16;**

#1 oznacza pytanie jednokrotnego(1) lub wielokrotnego(0) wyboru.

#2 oznacza numer zmiennej v w której zostanie umieszczona informacja o wybranej/wybranych opcjach. Podajemy sam index, bez literki.

#3 to początkowy stan przycisków, czyli pozycje które są domyślnie zaznaczone.

#4 to napis służący jako nagłówek. Może to być zmienna z bądź ciąg znaków ^^

#5 i dalej to napisy (ponownie z-var albo ^^) które pojawią się jako opcje do wyboru na liście

Odczytywanie informacji, które pozycje zostały wybrane, jest problematyczne. Ponieważ zmienna do przechowywania danych jest jedna a pozycji do 12, informacja ta jest przechowywana w postaci bitowej.

Kolejne pozycje na liście reprezentujemy jako kolejne potęgi dwójki zaczynając od zera, tzn 1,2,4,8,16 itd. Tak więc trzecia od góry pozycja ma wartość 4, czwarta ma 8 itd. Do v#2 zostaje wpisana SUMA wartości wszystkich wybranych pozycji.



W tym przypadku do v#2 zostanie wpisana liczba $1+4+8=13$

Do odczytania, czy konkretne pozycja została wybrana najłatwiej użyć receivera VR:

!!VRy1:Sv#2 &#L;

gdzie #L oznacza wartość pozycji (1,2,4,8...) Jeżeli pozycja **nie jest** wybrana, y1 będzie równe zero. Jeżeli **jest** wybrana, przyjmie wartość #L. Zauważ że & nie występuje tutaj jako "sprawdzacz" warunku i realizuje kompletnie inną operację. Aby sprawdzić czy pozycja o wartości #L wybrana, wystarczy sprawdzić czy y1<>0.

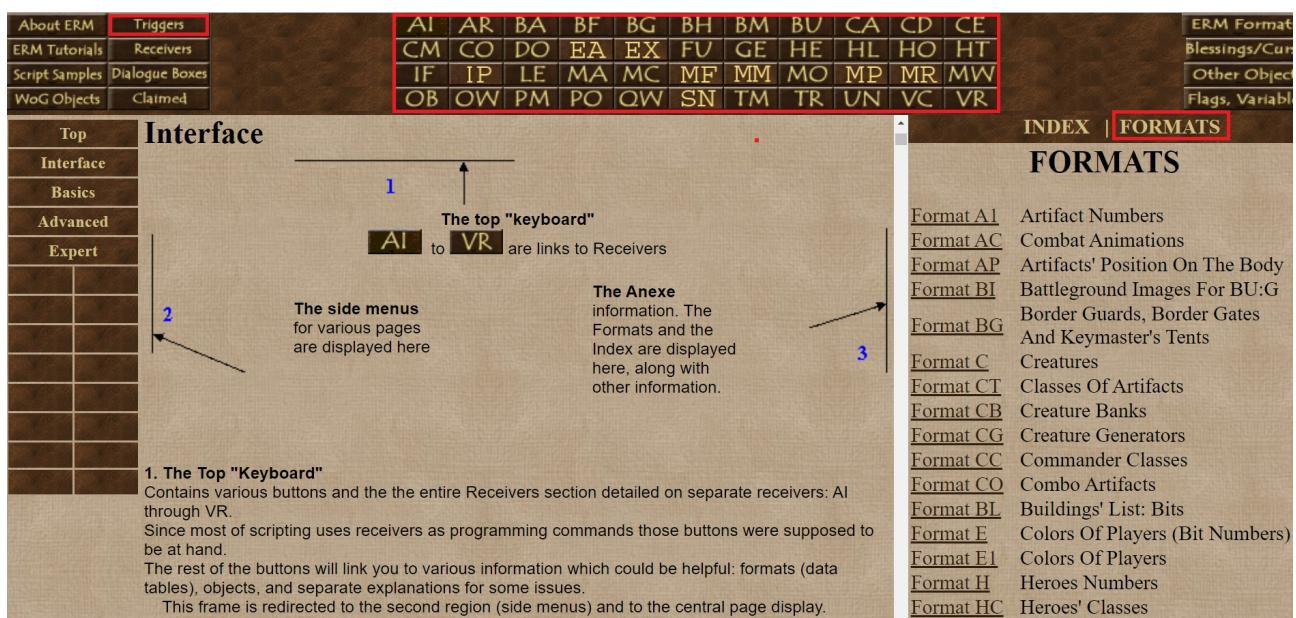
!!VRy1:Sv2 &16;

!!IF&y1<>0:M^Wybrano m.in. pozycje numer 5^;

Lekcja szósta – ERM Help

Podstawowym narzędziem pracy skryptera ERM nie jest program erm_s, a znajdujący się razem z nim w paczce (która pobrałeś przy rozpoczynaniu nauki) ERM Help. Jest to dokumentacja języka, zawierająca ogrom informacji o dostępnych poleceniach, bazach danych wykorzystywanych w Heroes 3 i wiele więcej. Na początku może wydać ci się przytłaczająca, ale bez obaw, jest to naprawdę poręczne narzędzie.

Do uruchomienia ERM Help potrzebna jest tylko przeglądarka internetowa. Wejdź do katalogu erm_tools/erm_help, po czym uruchom za jej pomocą plik „index.htm”. Gotowe!



Na początek nieco o interfejsie. Na obrazku powyżej zaznaczyłem ci czerwonymi prostokątami 3 najistotniejsze sekcje: „Triggers”, gdzie znajdziesz wszystkie dostępne triggery wraz z pełną ich dokumentacją, klawiaturę z „Receiverami” - możesz kliknąć na dowolny i dowiedzieć się o nim nieco więcej, oraz „Formats” gdzie znajdują się bazy danych, np. rozpiska numerów ID wszystkich potworów w grze (Format C)

About ERM	Triggers
ERM Tutorials	Receivers
Script Samples	Dialogue Boxes
WoG Objects	Claimed

AI	AR	BA	BF	BG	BH	BM	BU	CA	CD	CE
CM	CO	DO	EA	EX	FU	GE	LIE	H1	HO	HT
IF	IP	LE	MA	MC						
OB	OW	PM	PO	QW	SN	TM	TR	UN	VC	VR

ERM Formats
Blessings/Curse

Flags, Variables

POSITION INFO (PO) - MAP SQUARE
RECEIVER

Used to set up information for any square on the map.

This receiver may be very useful if you want to control a number of heroes or players who visited any object last (for bonuses and such). Except for calculate, ALL of the categories (options) are for convenience only; they can ALL be used for any numbers and are simply a way to store information about a map location for later use.

!!PO#1/#2/#3:XXXX;	Used to set up or check information on any square of the map. Sets/checks the square at x,y,l coordinates.
!!POS:XXXX;	Alternative method to above. Uses indirect reference: x=v[\$], y=v[\$+1], l=v[\$+2]

OPTIONS

B#/\$; NEW!	Set/check/get a big integer value (-2147483648...2147483647). You can use up to 2 values. # - index of the value to set (0...1) \$ - value to set/check/get
Calculate all objects	

file:///H/erm_help/receivers/receiver.fu.htm #3/\$4/\$5;

Oto strona przykładowego Receivera: PO. Na samej górze widzisz opis czego dotyczy (niezaznaczyłem kolorkiem, sorry) poniżej składnie (parametry), jeszcze niżej opcje (komendy dla receivera). Po lewej stronie ekranu znajduje się „wstążka” zawierająca odnośniki do różnych poleceń. Każde polecenie dla receivera składa się z pojedynczej literki i argumentów.

Type and subtype of Pictures using the Q command		
	Type	Subtype
Resource	Type (0...7)	Quantity
Artifact	8	Number of artifact For scrolls: (artifact number) + 0x10000*(spell number)
Spell	9	Number of spell
Flag	10	Number of flag
Luck (positive)	11	+ of luck
Luck (neutral)	12	no matter
Luck (negative) Not supported in heroes.	13	- of luck
Morale (positive)	14	+ of Morale
Morale (neutral)	15	no matter
Morale (negative)	16	- of Morale
Experience	17	Quantity
Secondary Skill	20	Skill + level *
Monster	21	Type of monster
Buildings in Towns	22...30 =0..8 by town type (Format T)	Type of building (Format U)

INDEX FORMATS	
FORMATS	
Format A1	Artifact Numbers
Format AC	Combat Animations
Format AP	Artifacts' Position On The Body
Format BI	Battleground Images For BU:G
Format BG	Border Guards, Border Gates And Keymaster's Tents
Format C	Creatures
Format CT	Classes Of Artifacts
Format CB	Creature Banks
Format CG	Creature Generators
Format CC	Commander Classes
Format CO	Combo Artifacts
Format BL	Buildings' List: Bits
Format E	Colors Of Players (Bit Numbers)
Format E1	Colors Of Players
Format H	Heroes Numbers
Format HC	Heroes' Classes

INDEX FORMATS	
FORMAT C Format List	
Creatures' List	
by Numerical Order (Creatures are Type 54)	
By Alphabetical Order	
0	Pikeman
1	Halberdier
2	Archer
3	Marksman
4	Griffin
5	Royal Griffin
6	Swordsman
7	Crusader
8	Monk
9	Zealot
10	Cavalier

A to dokumentacja dotycząca IF:Q i typów oraz podtypów obrazków do wykorzystania. Po kliknięciu na jeden z odnośników w oknie „Formats” pojawi się format z nim powiązany. Formaty to bazy danych (głównie tabelki) w których każdej informacji przypisana jest liczba (index). W ten sposób, chcąc przekazać receiverowi by wyświetlił obrazek potwora, np. Monka, podajemy Typ 21 i sprawdzamy subtyp w Format C - otrzymujemy Subtyp 8.

Ważnym elementem dokumentu jest jeszcze zakładka ERM Tutorials – poziom basic i advanced jest wy tłumaczony tak sobie i raczej nie warto czytać, ale poziom Expert pióra samego Slavy Salnikowa to skarbnica wiedzy dotyczącej m.in. wsparcia dla tłumaczenia skryptów, synchronizacji w trybie sieciowym, podmiana właściwości umiejętności drugorzędnych (chcesz Nauke dającą 20/40/60% dodatkowego expa? Nie ma problemu) natomiast do tego etapu jeszcze ci daleko.

Lekcja siódma – Statystyki jednostek, składnia „set/get”

Do modyfikacji statystyk jednostek wykorzystujemy receiver **!MA**, najczęściej w formie instrukcji. W tym przykładzie użyjemy jednak bloku triggera **?PI**, wyłącznie dlatego że chcę używać zmiennych lokalnych. Także do roboty:

```
ZVSE
!?PI; [Post-instruction trigger]
!!VRy1:S0; [Inicjalizuję zmienną do przechowywania prędkości: y1]
!!MA:S59/?y1[Pobieramy prędkość potwora o numerze 59 (zombie) i
zapisujemy do y1]
!!VRy1:*2; [Mnożymy odczytaną prędkość przez 2]
!!MA:S59/y1;[Race-car zombie]
```

Powyższy skrypt odczytuje jaką szybkość ma zombie i zwiększa ją dwukrotnie. Jest to lepsze podejście niż napisanie po prostu **!MA:S59/8**; ponieważ inne skrypty również mogą modyfikować statystyki jednostek – w takim przypadku ich efekt zostałby nadpisany i pominięty.

Jak zwróciłeś uwagę, to samo polecenie pozwoliło mi na odczyt szybkości zombie oraz na jej ustawienie. Większość poleceń ERM wykorzystuje jedną z trzech składni:

- Set – w takim przypadku możemy tylko ustawiać daną wartość, nie możemy jej odczytać
- Get/Check – możemy tylko pobierać daną wartość, ale nie możemy jej ustawiać
- Get/Check/Set – możemy wartość pobierać i ustawiać.

Oczywiście **!MA:S#/\$/**; wykorzystuje składnie Set na argumencie pierwszym (#) oraz Get/Check/Set na argumencie drugim (\$).

Ustawianie (set) wartości w argumencie odbywa się przez podanie liczby lub po prostu zmiennej, np. **!MA:S59/21**; albo **!MA:Sy1/y2**;

Pobranie wartości (get) odbywa się poprzez podanie zmiennej poprzedzonej znakiem '?', np. **!MA:S59/?y1**;

```
ZVSE
!?PI; [Post-instruction trigger]
!!VRy1:S-4; [Ustaw wartosc zmiennej na -4]
!!MA:A59/dy1;[Dodaj do wartosci ataku potwora o numerze 59 wartosc
zmiennej y1]
```

Tym razem operujemy na ataku zombie, i zamiast pobierać jego aktualną wartość, operować na niej przez receiver VR i po wszystkim ustawiać, wykorzystałem kolejny element składni języka. Użycie litery 'd' przed wartością powoduje, że wartość ataku ZAMIAST być ustawiona (set) na wartość y1, zostaje POWIĘKSZONA o wartość y1 (dodawanie). Ponieważ y1=-4, w tym przypadku skrypt zmniejsza atak zombie o 4.

Powyższy skrypt jest równoznaczny z tym:

```
ZVSE
!?PI;
!!VRy1:S0;
!!MA:A59/?y1;
!!VRy1:-4;
!!MA:A59/y1;
```

Używanie literki ‘d’ jest bardzo użyteczne zwłaszcza, gdy mamy mnóstwo argumentów w polecenie z których faktycznie interesują nas tylko nieliczne. Np.

! !TRx/y/z:T\$1/\$2/\$3/\$4/\$5/\$6/\$7/\$8;

Nie ważne co robi to polecenie. Najczęściej używamy go do pobrania wartości \$1. Niestety, musimy podać wszystkie argumenty, a wpisując 0 **ustawimy** (set) wszystkie pozostałe pola na 0. Bardzo tego nie chcemy. Rozwiązaniem tego problemu jest właśnie ‘d’, konkretnie d0. Podając:

! !TRv1/v2/v3:T?y1/d0/d0/d0/d0/d0/d0;

Pobierzemy wartość \$1 do zmiennej y1 a pozostałe zostaną zwiększone o 0 – czyli niezmienione.

Lekcja ósma – Funkcje, pętle, modyfikacje bohaterów

Do operowania na bohaterach wykorzystujemy receiver **!!HE#1:XXXX**; Na potrzeby tej lekcji napiszemy skrypt rozwijający umiejętność „Finanse” - bohaterowie posiadający tą umiejętność będą generować (70/140/210) dodatkowego złota na każdy poziom doświadczenia. Do roboty!

ZVSE

```
!#TM1:S1/-1/1/255; [Inicjalizacja timera - trigger TM1 będzie
zalaczal się kazdego dnia dla kazdego gracza, w nieskonczonosc]
!?TM1; [Trigger ustawiony poleceniem wyzej. Wykonuje się kazdego
dnia]
!!D01/0/155/1:P; [Wywolaj funkcje numer 1 w petli]
!?FU1; [Funkcja numer 1. W zmiennej x16 liczba od 0 do 155: numer
bohatera, Format H]
!!OW:C?y1; [Pobierz aktualnego gracza: y1]
!!HEx16:0?y2; [Pobierz wlasciciela analizowanego bohatera: y2]
!!FU&y1<>y2:E; [Opusc funkcje, jeśli bohater nie należy do
aktualnego gracza]
!!HEx16:S13/?y3; [Pobierz poziom umiejescnosti bohatera: Estate y3]
!!HEx16:Ed0/?y4/1; [Pobierz poziom bohatera: y4]
!!VRy5:S70 *y4 *y3; [Oblicz bonus do złota: y5]
!!OW:Ry1/6/dy5; [Dodaj wlascicielowi bohatera y5 złota]
```

Jest trochę do tłumaczenia. Triggerem i receiverem TM nie będę się tutaj zajmować, po szczegóły zapraszam do ERM Help.

Funkcje

Funkcje to bloki kodu zaczynane triggerem **! ?FU#**; W przeciwieństwie do wszystkich innych triggerów, te nie są załączane z powodu akcji w grze – aby się wykonały, musisz wywołać je sam. Pozwala to na rozbijanie dużych fragmentów kodu na mniejsze, dzięki czemu jest bardziej przejrzysty, a także tworzenie funkcji „ogólnego przeznaczenia” - wykorzystywanych w wielu miejscach, wykonujących jakieś pospolite zadania.

Do tradycyjnego wywołania funkcji wykorzystujemy receiver **!!FU:P**; Nie użyłem go, ponieważ w skrypcie wywołanie NIE jest tradycyjne ;)

Funkcje mogą przyjmować do 15 argumentów, danych jako zmienne ‘x’. Najłatwiej pokazać to na przykładzie: **! !FU1:P4/2/3/9/1/8;**

Przy takim wywołaniu, wewnątrz bloku funkcji **?FU1**; będziemy mieć do dyspozycji argumenty: $x_1 = 4, x_2 = 2, x_3 = 3, x_4 = 9, x_5 = 1, x_6 = 8$. Możemy tak podawać argumenty aż do x_{15} . Jeżeli przy wywołaniu nie podamy któregoś z argumentów, otrzyma on wartość 0, więc w tym przypadku

$x_7 \dots x_{15} = 0$

Poprzez argumenty funkcji możemy nie tylko podawać dane, ale również je pobierać:

!!FU1:P4/1/5/8/?y1;

W takim przypadku: $x1 = 4$, $x2 = 1$, $x3 = 5$, $x4 = 8$, $x5$ natomiast jest zmienną do której możemy zapisać dane. Wówczas zostaną one przekazane do $y1$. Nazywamy to „zwracaniem wartości przez funkcje”.

Każda zmienna x , od indeksu 1 do 15, może być używana zarówno do przekazywania wartości do funkcji oraz odbierania wartości zwracanej przez funkcję, ALE nie jednocześnie

Jeżeli dana zmienna x jest wykorzystywana do zwracania wartości, powinieneś obchodzić się z nią jak z jajkiem. Nie próbuj odczytać jej wartości np. $!!VRy1:Sx5$; i zapisuj wartość do niej tylko raz, na samym końcu funkcji. W moim doświadczeniu wielokrotny zapis powodował nieprawidłowe działanie.

Polecenie **!!FU:E**; pozwala przerwać wykonywanie danej funkcji. Jeśli zostanie wywołane, reszta bloku NIE zostanie wykonana.

Pętle

Jak widzisz w przykładzie z Finansami, odwołuję się do zmiennej $x16$. Ale zaraz, czy nie miało być ich 15? Otóż tak, ale ponieważ wykorzystuję funkcje w pętli mam do dyspozycji dodatkowy argument: numer cyklu.

Pętle pozwalają na wielokrotne wywołanie tej samej funkcji za pomocą jednego polecenia. Do wywołania ich używamy receivera **!!DO**. Składnia:

!!DO#1/#2/#3/#4:P;

#1 – numer funkcji do wywołania

#2 – wartość początkowa zmiennej $x16$ (numeru cyklu)

#3 – wartość zmiennej $x16$, po której przekroczeniu pętla się zakończy

#4 – przyrost na zmiennej $x16$

Po poleceniu ‘P’ można podać argumenty które będą w każdym cyklu przekazywane do funkcji. Składnia jest taka sama jak w przypadku $!!FU#1:P$;

Pętla ustawia wartość zmiennej $x16$ na #2, po czym wywołuje funkcję #1. Następnie zwiększa $x16$ o wartość #4. Gdy wartość $x16$ **przekroczy** #3, pętla dobiera końca.

Można wymusić zakończenie pętli poprzez ustawienie wartości $x16$ powyżej #3: $!!VRx16:S99999$;

W tym przypadku $x16$ oznacza numer bohatera na liście (Format H), jest to pętla „iterującą” po tej liście. „Iteracja” to proces poruszania się po wszystkich elementach listy i wykonująca na nich jakieś operacje. Będziesz z tego często korzystać.

Umiejętność

No, to dwie linijki za nami. Reszta na szczęście pójdzie szybciej. $!!OW:C?y1$; pobiera do zmiennej $y1$ aktywnego gracza (Format E1), następnie za pomocą $!!HEx16:0?y2$; pobieramy właściciela bohatera o numerze $x16$ do zmiennej $y2$. Jeżeli te dwie liczby nie są takie same – przerywamy funkcje. Dlaczego? Otóż trigger **!TM1** w tym przypadku uruchamia się dla każdego gracza osobno, więc bez kontroli właściciela gracza z finansami generowałby szekle X razy na dzień (gdzie X to liczba graczy)

Następnie !!HEx16:S13/?y3; pobieramy poziom umiejętności numer 13 (Format SS) do zmiennej y3 oraz !!HEx16:Ed0/?y4/1; pobieramy poziom bohatera do y4, nie ruszając ilości zebranego expa. !!VRy5:S70 *y3 *y4; pozwala obliczyć ilość złota którą bohater ma wygenerować. Poziom*poziom_umiejki*70. Ostatecznie !!OW:Ry1/6/dy5; zwiększa liczbę surowca 6 (Format R, złoto) gracza y1 o wartość y5.

W sumie warto zaznaczyć, że dla dowolnego bohatera możemy odczytać poziom dowolnej z 27 istniejących umiejętności. Otrzymamy wówczas: 0 gdy nie posiada, 1 gdy podstawowy, 2 gdy zaawansowany, 3 gdy mistrzowski.

Ponieważ !!HE#:S daje Get/Set syntax, możemy również ustawić wszystkie umiejętności zgodnie z własnym widzimisie. Możemy nawet stworzyć bohatera z mistrzem we wszystkich 27 umiejętnościami!

Lekcja dziewiąta – Podmiana muzyki w mieście

Poprzednia lekcja była ciężka jak The Walking Dead więc myślę że czas na coś szybkiego, łatwego i szalenie satysfakcjonującego. Oto skrypcik:

```
ZVSE  
!#MP:S2/^GRASS^;
```

Tak, to naprawdę wszystko. Efekt jest taki, że muzyka w Zamku (Castle) zostanie podmieniona przez ambient odgrywany na trawie. Możesz również dodać muzykę całkowicie nową: musisz tylko pobrać ją w formacie .mp3, umieścić w folderze MP3 w katalogu Heroes, i zamienić GRASS na nazwę pliku z nową muzyką. Nie wpisuj rozszerzenia (.mp3), zostaje ono dodane automatycznie.

Do podmiany muzyki (nie mylić z efektami dźwiękowymi!) używamy receivera !!MP:S#1/^filename^; #1 to indeks danej muzyki (Format MP), natomiast ^ zawiera ścieżkę do pliku z muzyką, bez rozszerzenia. Domyślnym miejscem dla muzyki jest folder MP3 w katalogu heroes i ścieżka musi być pisana względem niego.

Jeśli chcesz wykorzystać muzykę umieszczoną w folderze Data, wpisz ^./Data/nazwa_pliku^.

Lekcja dziesiąta, ostatnia – Walka

Na koniec naszej podróży wybrałem skrypt zamieniający umiejętność „resistance” z wersji znanej z SoD na wersję HotA – zamiast zapewniać % szans na odbicie zaklęcia będzie zmniejszać moc bohatera przeciwnika o 10/20/30%. Od razu mówię że skrypt jest tylko do singleplayer, dałoby się zrobić wersje działającą na multi ale to skomplikowałoby kod. Do roboty:

```
ZVSE  
! ?BA0; [Pre-battle trigger]  
!!BA:H0/?v8 H1/?v9; [Numery bohaterow: attakujacy v8, broniacy v9]  
!!FU&v9=-2:E; [Przerwij jeśli jest tylko jeden bohater]  
!!HEv8:Fd0/d0/?y3/d0; [Moc bohatera atakujacego: y3]  
!!HEv9:Fd0/d0/?y4/d0; [Moc broniacego: y4]  
!!HEv8:S26/?y5; [Odpornosc atakujacego: y5]  
!!HEv9:S26/?y6; [Odpornosc broniacego: y6]  
!!VRy7:S-1; [Do odwracania znaku: y7=-1]  
!!VRv10:Sy6 *10 *y3 :100 *y7; [„Korekta” do mocy atakujacego: v10]  
!!VRv11:Sy5 *10 *y4 :100 *y7; [„Korekta” do mocy broniacego: v11]  
!!HEv8:Fd0/d0/dv10/d0; [Korekta]  
!!HEv9:Fd0/d0/dv11/d0; [Korekta]
```

```
! ?BA1; [Post-battle trigger]
!!FU&v9=-2:E; [Wyjdź, jeśli tylko jeden bohater]
!!VRy1:S-1; [Do odwracania znaku]
!!VRy2:Sv10 *y1; [Korekta dla atakujacego]
!!VRy3:Sv11 *y1; [Korekta dla broniacego]
!!HEv8:Fd0/d0/dy2/d0; [Oddajemy co zabralismy - atakujacy]
!!HEv9:Fd0/d0/dy3/d0; [Oddajemy co zabralismy - obronca]
```

Musimy używać relatywnej składni d przy odejmowaniu/dodawaniu do mocy, ponieważ trigger !?BA1 załącza się po zakończonej walce, ale również po wbiciu levela przez bohatera, więc gdyby dostał z jego okazji moc, nadpisalibyśmy tą zmianę – a tego nie chcemy. Dlatego lepiej dodawać niż ustawać.

Powyższy skrypt dodaje do Resistance dodatkowy efekt: zmniejszanie mocy przeciwnika o 10/20/30%, natomiast nie usuwa standardowego zachowania, wymaga więc rozszerzenia. Doprogramujemy to, dodając dodatkowe bloki. Niestety, ERM nie umożliwia „podmiany” zachowania umiejętności, ani jej wyłączenia – możemy natomiast ją usunąć i przywrócić po zakończonej walce. Ten skrypt należy wkleić PONIŻEJ skryptu wcześniejszego.

```
! ?BA0&1000; [Specjalna flaga - wykonaj jeśli jednym z bohaterow jest gracz]
!!BA:H0/?v8 H1/?v9; [Numery bohaterow: attakujacy v8, broniacy v9]
!!FU&v9=-2:E; [Wyjdź, jeśli tylko jeden bohater]
!!HEv8:S26/?v12; [Odpornosc atakujacego: v12]
!!HEv9:S26/?v13; [Odpornosc broniacego: v13]
!!HEv8:S26/0; [Usuwamy odpornosc atakujacego]
!!HEv9:S26/0; [Usuwamy odpornosc broniacego]

! ?BG1&1000; [Jakakolwiek akcja podjeta podczas bitwy]
!!FU&v9=-2:E; [Wyjdź, jeśli tylko jeden bohater]
!!BU:C?y1; [Sprawdz, czy bitwa skonczyła się w tej turze: y1]
!!FU&y1=0:E; [Wyjdź, jeśli nie]
!!HEv8:S26/v12; [Oddaj atakujacemu co atakucerskie]
!!HEv9:S26/v13; [Oddaj broniacemu odpornosc]

! ?BA1&1000;
!!FU&v9=-2:E; [Wyjdź, jeśli tylko jeden bohater]
!!HEv8:S26/?y1; [Pobierz odpornosc atakujacego: y1]
!!HEv9:S26/?y2; [Pobierz odpornosc obroncy: y2]
!!HEv8&y1<v12:S26/v12; [Jesli utracono odpornosc wskutek ucieczki, przywroc]
!!HEv9&y2<v13:S26/v13; [Jak wyzej]
```

Ta część skryptu powinna być pilnowana specjalną flagą 1000 która (w tym przypadku) przyjmuje 1 (true) jeśli w bitwie uczestniczy gracz. Jest to spowodowane tym, że polecenia dotyczące ekranu bitwy (np. BU, BG) nie mogą być używane w bitwach w których udział bierze wyłącznie AI. Sztuczna inteligencja tak naprawdę nigdy nie wchodzi na ekran bitwy: oblicza wartość swojej armii i umiejek, oblicza wartość armii przeciwnika i na tej podstawie określa wynik bitwy.

W przypadku gdy po jednej stronie jest bohater gracza, bitwa rozgrywana jest normalnie, nawet jeśli włączono szybką walkę (bitwa nadal jest wówczas rzeczywista, nie symulowana, wszystkie polecenie działają)

Podsumowanie

Gratulacje, dotrwałeś do końca mimo moich usilnych prób zanudzenia cię na śmierć! Po takim wyczynie opanowanie ERM do perfekcji powinno być dla ciebie kaszką z mleczkiem. Jednakowoż, czeka cię jeszcze dużo pracy. Dopiero pracując nad tym dokumentem uświadomiłem sobie jak rozbudowany jest Event Related Model. Powodzenia stary. Tylko uważaj, w ERM Help zdarzają się błędy.



I pamiętaj: tutaj da się zrobić niemal wszystko