TV-Computer

GAME ENGINE unit

TRSE Pascalhoz

Verzió: 0.1

Készítette: Bertók Zsolt, 2023

Tartalomjegyzék

1. Mi az a TRSE

A **TRSE** *(Turbo Rascal Systax Error)* egy Windows / Mac / Linux környezetben futó fejlesztőkörnyezet és **Pascal** fordító 8 bites számítógépekhez.

Eredetileg Commodore 64-es demók fejlesztéséhez készült, de az elmúlt 5 évben számos más platform is támogatásra került, és jó néhány játék is készült vele.

2023 januárjában a VIDEOTON TV-Computer is a támogatott számítógépek közé került.

A TRSE weboldala az alábbi linken található, ahol a **Downloads** menüpontból letölthető a legfrissebb verzió:

https://lemonspawn.com/turbo-rascal-syntax-error-expected-but-begin/

Érdemes a **TRSE Syntax** menüpontot is átnézni a weboldalon, mert vannak sajátosságai a Pascal fordítónak. Mivel első sorban demók fejlesztésére készült a TRSE, ezért csak néhány változó típust ismer a fordító, a string típus kezelése is némiképp egyedi, valamint az egymásba ágyazott feltételeknek is van egy sajátságos szintaxisa.

2. Mi a Game Engine

A *Game Engine (röviden **GE**)* egy TRSE Pascalhoz készült unit, ami TVC játékok fejlesztését támogatja.

A Game Engine a **GE.TRU** nevű fájlban található, ami egy unit fájl *(TRU – Turbo Rascal Unit)*.

A Game Engine az alábbiakat támogatja jelenleg:

- grafikus beállítások kezelése (*SetGraphics. SetPalette, SetBorder*)
- memória lapozás *(SetMemory)*
- sprite kezelés - *inicializálás, mozgatás, megjelenítés (akár csoportosan is)*
- néhány kiegészítő grafikus utasítás *(ClearScreen, FillRect, HbytesLine, VbytesLine)*
- *raszter műveletek (WaitForSync, SetRasterInterruptPos)*
- billentyűzet- és joystick kezelése *(GetJoysticks, Joystick, KeyPressed, SpacePressed, etc.)*
- alapszintű hang kezelés *(SoundOn, SoundOff, Sound, NoSound, Volume)*
- szöveg kiírás saját karakterkészlettel *(WriteText)*
- néhány egyéb funkció *(In, Out, Rnd, Delay, Odd, IntToStr, ByteToStr, FillMem, MemCopy)*
- vertikális hardware scroll *(ScrollUp, ScrollDown)*
- adat kitömörítés ZX7 compressed adatból *(ZX7Decompress)*

Tervben van még:

- horizontális hardver scroll (ez speciális sprite kirakót igényel)
- digitalizált hangeffektek lejátszása

TVC unit

A *Game Engine* mellett készült egy **tvc.tru** unit is, ami a TV-Computer ROM rutinjait implementálja, mint például a grafikus mód, paletta, keretszín beállítása, szövegek kiírása eredeti karakterkészlettel, színek, vonal rajzolás, vonal típus megadása, alakzat kitöltés, karakterek áttervezése, billentyűzet és joystick lekérdezése, hangkeltés, fájlkezelés, és In, Out műveletek. Valamint néhány kiegészítő funkció is került bele, mint a négyzet rajzolás, vagy a kitöltött négyzet rajzolása, várakozás a cursor-megszakításra.

3. Game Engine alapok

Ahogy minden TRSE unit esetében, a *Game Engine*-nél is a unit nevét, a **GE** szót, prefixként használva lehet elérni a *Game Engine* eljárásokat, függvényeket, osztályokat és konstansokat.

Például így: **GE::Init();**

A *Game Engine* kétféle üzemmódban tud működni:

- direkt a képernyőre dolgozva - ez TVC 32K-n is működik
- háttérképernyő használatával - ehhez TVC 64k vagy 64K+ szükséges

Hogy melyik módot érdemes használni, az teljesen játékfüggő. Ha nincsenek maszkolt sprite-ok a játékban, csak direkt felülírós módon kitett alakzatok, vagy ha csak kevés maszkolt sprite mozog a képernyőn, vagy hardver scroll van a játékban, akkor a direkt képernyős módot érdemes választani. Ha viszont 4-5-nél több maszkolt sprite mozog a játékban, vagy a sprite-ok a képernyő bármelyik területén lehetnek egyidejűleg, akkor a háttérképernyős mód lesz a jobb választás. Ezért is támogatja mindkét lehetőséget a *Game Engine*.

1. Direkt képernyős mód *(direct to screen)*

Ekkor minden grafikus művelet közvetlenül a képernyőre, azaz a VIDEORAM-ba dolgozik.

Előnyei:

- gyorsabb
- nem igényel +16k háttérképernyőt
- a képernyőfrissítéssel szinkronizálva szép mozgást eredményez

Hátrányai:

- egy képernyőfrissítés alatt maximum 4-5-6 közepes méretű maszkolt sprite rakható ki villogás nélkül

2. Háttérképernyős mód *(use back screen)*

Ekkor minden grafikus művelet a legfelső 16k-ban történik (3. memórialap), ami háttérképernyőként funkcionál, és onnan csak a változások kerülnek kimásolásra a képernyőre.

Előnyei:

- több maszkolt sprite megjeleníthető
- nincs villogás

Hátrányai:

- 16k plusz memóriát igényel
- valamivel lassabb a megjelenítés a plusz képernyőre kimásolás miatt

4. Grafikus beállítások

4.1 SetGraphics

A TVC grafikus módjának beállítása.

Például:

GE::SetGraphics(GE::GRAPHICS2);

vagy

GE::SetGraphics(GE::GRAPHICS4);

vagy

GE::SetGraphics(GE::GRAPHICS16);

4.2 GetGraphics

Az aktuális grafikus mód lekérdezése.

GetGraphics() : byte

Például:

if (**GE::GetGraphics()** = **GE::GRAPHICS4**) then

begin

end;

4.3 SetPalette

A paletta egyik színének beállítása Graphics2 vagy Graphics4 módok esetén.

GE::SetPalette(Nr, Color : byte)

Első paraméter a palette indexe, ami Graphics2 mód esetében 0 vagy 1 lehet, míg Graphics4 mód esetében 0, 1, 2 vagy 3. A második paraméter a szín kódja.

Például:

GE::SetPalette(0, GE::PALETTE_BLACK);

GE::SetPalette(1, GE::PALETTE_WHITE);

GE::SetPalette(2, GE::PALETTE_DARK_CYAN);

GE::SetPalette(3, GE::PALETTE_RED);

Minden használható paletta színhez van egy szín konstans, amit megadható. Ezek az alábbiak:

- PALETTE_BLACK
- PALETTE_DARKBLUE
- PALETTE_DARKRED
- PALETTE_DARKMAGENTA
- PALETTE_DARKGREEN
- PALETTE_DARKCYAN
- PALETTE_DARKYELLOW
- PALETTE_GRAY
- PALETTE_BLACK2
- PALETTE_BLUE
- PALETTE_RED
- PALETTE_MAGENTA
- PALETTE_GREEN
- PALETTE_CYAN
- PALETTE_YELLOW
- PALETTE_WHITE

4.4 SetBorder

A keret színének beállítása.

GE::SetBorder(Color : global byte)

Például:

GE::SetBorder(GE::BORDER_DARKBLUE);

Minden használható keretszínhez van egy konstans, amit megadható. Ezek az alábbiak:

- BORDER_BLACK
- BORDER_DARKBLUE
- BORDER_DARKRED
- BORDER_DARKMAGENTA
- BORDER_DARKGREEN
- BORDER_DARKCYAN
- BORDER_DARKYELLOW
- BORDER_GRAY
- BORDER_BLACK2
- BORDER_BLUE
- BORDER_RED
- BORDER_MAGENTA
- BORDER_GREEN
- BORDER_CYAN
- BORDER_YELLOW
- BORDER_WHITE

5. Init

GE::Init(UseBackScreen : boolean);

Ezzel az utasítással lehet inicializálni a *Game Engine*-t. A paraméterben azt kell megadni, hogy direkt képernyőre dolgozzon, vagy a háttérképernyőre.

Direkt képernyő esetén:

GE::Init(GE::DIRECT_TO_SCREEN);

Háttérképernyő esetén:

GE::Init(GE::USE_BACK_SCREEN);

Ennek az utasításnak meg kell előznie minden sprite kirakó, vagy képernyőre rajzoló utasítást. De a sprite-ok inicializálása, a grafikus mód, paletta, és keret beállító eljárások, valamint a billentyűzettel és hanggal kapcsolatos eljárások hívása is megelőzhetik az **Init** hívását.

6. Memória lapozás

TV-Computer egyszerre 64 kbyte memóriát tud megcímezni a 16 bites regisztereivel. A számítógép memóriája 16 kbyte-os lapokra van felosztva, így 4 lap használatos a maximum 64 kbyte memória kiosztására, amik 0, 1, 2, 3 sorszámokon vannak említve a szakirodalomban.

A felhasználó által használható RAM-okat **U**-val jelöli a szakirodalom, ami talán a **U**ser RAM-ra utal, így ezekre az **U0, U1, U2, U3** jelölésekkel hivatkozunk. A VIDEORAM jelölésére a **VID** rövidítést szokás használni, míg a rendszer ROM-hoz a **SYS**-t, az oldalról bedugható cartridge-khez a **CART**-ot, a felső bővítőhelyekre behelyezhető kártyák esetében pedig az **EXT** jelölést.

A gép indulásakor a memória a következőképpen kerül belapozásra a rendszer által: a 0-2. lapokon a felhasználó által szabadon használható RAM van belapozva (TVC 32K-n csak a 0-1 lapokon, ami így 2x16K = 32K), míg a 3., legfelső lapon a rendszer ROM helyezkedik el a BASIC-kel. Így a gép indulásakor a következő memória belapozás van érvénybe: **U0, U1, U2, SYS**.

A képernyőre íráshoz be kell lapozni a VIDEORAM-ot (**VID**) a 2-es lapra (csak 2-es lapra lehet), hogy az elérhető legyen a futó program számára is. Ha pedig használni szeretnénk a teljes 64K memóriát, azt is külön be kell lapozni. A memórialapozásra van egy rendszerfunkció, ami nagyon gyors, így egy játék futása közben sem okoz lassulást, vagy akadást, ezért a játék főciklusban is ki-belapozható a VIDEORAM, vagy a teljes 64K RAM és egy másik memóriakiosztás, ha szükséges.

Ha az **Init** procedúra hívásakor háttérképernyős működés kerül megadásra (**USE_BACK_SCREEN**), akkor az **Init** automatikusan belapozza a teljes 64k-t RAM-nak (**U0, U1, U2, U3**). Ebben az esetben nincs szükség memória lapozásra, mert a háttérképernyőről a képernyőre másoló *Game Engine* eljárások automatikusan belapozzák maguknak a VIDEORAM-ot, amikor szükséges, majd visszaállítják a teljes 64k RAM lapozást. Kivéve persze, ha „manuálisan” is történik képernyőre írás a programból, például pálya kirajzoláskor, pontszám kiírásakor, mert ekkor a VIDEORAM belapozást is a programban kell elvégezni a szükséges részen.

6.1 SetMemory

SetMemory(Mode : byte) - a memória lapozását végző utasítás

Négy előre definiált konstans paraméter segíti a kívánt memória lapozási mód beállítását. Ezek a következők:

MEM_DEFAULT: alapértelmezett memória lapozás, ahol a 0-2 lapokon RAM (32k/48k), a 3. lapon pedig a ROM lesz *(**U0, U1, U2, SYS**)*

MEM_VID_ON: az 0-1 lapokon RAM (32k), a 2. lapon a VIDEORAM, a legfelső lapon pedig a ROM lesz *(**U0, U1, VID, SYS**)*

MEM_FULL_64K: a teljes 64 kbyte memória belapozása, ahol mind a négy lapon szabadon használható RAM lesz *(**U0, U1, U2, U3**)*

MEM_48K_VID: 0-1 lapokon RAM, a 2. lapon a VIDEORAM, a 3. lapon pedig szintén RAM lesz *(**U0, U1, VID, U3**)*

További memória lapozások is megadhatók, ekkor a paraméter értékét az alábbi táblázat alapján lehet meghatározni:

3. lap	0 - 1 - 2 lap					
	SYS U1 VID	SYS U1 U2	CART U1 VID	CART U1 U2	U0 U1 VID	U0 U1 U2
CART	00	20	08	28	10	30
SYS	40	60	48	68	50	70
U3	80	A0	88	A8	90	B0
EXT	C0	E0	C8	E8	D0	F0

FONTOS! *Direkt képernyő használata esetén gondoskodni kell arról, hogy minden képernyőre történő művelet meghívása előtt a VIDEORAM be legyen lapozva. Ha a program mérete nem haladja meg a 25 kbyte-ot, akkor elegendő valamikor a program elején egyszer belapozni a VIDEORAM-ot a **MEM_VID_ON** vagy a **MEM_48K_VID** konstansok használatával, és többet nem kell foglalkozni a memória lapozással. Ha viszont ennél nagyobb méretű a program, akkor az rálóg a 2. memórialapra is, ahová a VIDEORAM kerül belapozásra, ezért a képernyőműveletek idejére érdemes csak belapozni a VIDEORAM-ot, majd visszalapozni, mondjuk a **MEM_DEFAULT** vagy **MEM_FULL_64K** memória kiosztásra, hogy a program másik fele se legyen kilapozódva futás közben, mert az csúnya hibákat fog okozni. Emellett célszerű a képernyőműveleteket egy olyan eljárásba tenni, ami a forráskód elején helyezkedik el, hogy az mindenképp a 0. vagy maximum az 1. lapon levő memóriacímre forduljon, és semmiképp ne a 2. lapra, ahová a VIDEORAM lesz majd belapozni a képernyőműveletek alatt.*

7. A Sprite osztály

A *Game Engine*-ben egy külön osztály van a sprite-ok kezelésére, ez a **GE::Sprite** osztály.

Kétféle sprite-ot tud megjeleníteni a **Sprite** osztály:

1. a nem átlátszó / nem maszkolt sprite, ami gyakorlatilag egy négyzet alakú képdarab, amit egyszínű háttereken praktikus mozgatni, vagy egy helyben levő animációkhoz is alkalmas, esetleg hátterek, pontszámok, címfeliratok kirakására is használható.

2. az átlátszó vagy maszkolt sprite, ami esetében a megadott háttérszínnel rajzolt részek átlátszóak lesznek, így az háttér előtt is mozgatható, amire kitakarás nélkül rakható ki.

A **BMPToTVCSprite** utility program képes 24 bites *.bmp fájlból a *Game Engine* formátumában kimenteni a kívánt sprite-okat, maszkkal, vagy maszk nélkül. A kimentett bináris fájl tartalmazza a sprite típusát, méretét és magát a sprite image-et, valamint a 2-es típus esetében a maszkot is.

A **BMPToTVCSprite** programból kimentett bináris fájlt a következőképpen lehet beilleszteni a Pascal forrásba:

```
var
  PlayerSpriteBin : incbin("sprites/PlayerSpr.bin");
```

A sprite-okat következő adatstruktúra írja le:

Byte	Nem maszkolt sprite	Maszkolt sprite
0.	Típus: **1**	Típus: **2**
1.	Szélesség byte-okban	Szélesség byte-okban
2.	Magasság pixelekben	Magasság pixelekben
További byte-ok	A képernyő byte-ok egymás után TVC formátumban (szélesség * magasság számú byte)	1 byte maszk + 1 byte képernyő (TVC formátumban) váltakozva egymás után (szélesség * magasság számú * 2 byte)

Ha másik programból történik a sprite-ok előállítása, ahol nem kerül a kimentett bináris fájl elejére a sprite típusa és a méretei, akkor a sprite adata elé definiálni kell egy 3 elemű byte tömböt, amiben a sprite típusa, szélessége és magassága kerül megadásra. Például így:

```
var
  PlayerSpriteData : array[3] of byte = (1, 4, 10)     /* típus, szélesség, magasság */
  PlayerSpriteBin : incbin("sprites/PlayerSpr.bin");
```

Egy sprite deklarálása pedig így néz ki, ahol a későbbi hivatkozási nevet érdemes változó névnek adni, majd megadni, hogy ez egy **GE::Sprite** osztályú változó:

```
var
  PlayerSprite : GE::Sprite;
```

7.1 A Sprite osztály változói

Megnevezés	Típus	Leírás
Visible	boolean	**true**: a sprite ki lesz rakva; **false**: nem [1]
X	byte	A sprite vízszintes pozíciója. Értékei: **0-63**
Y	byte	A sprite függőleges pozíciója: Értékei: **0-239**
State	byte	Állapot (pl.: 0: meghalt; 1: él) [2]
Direction	byte	Irány (pl.: Left, Right, Up, Down) [2]
Timer	byte	Időzítő (pl.: két lövés közötti szünet időzítésére) [2]
Counter	byte	Számláló (pl.: mennyit lépjen az ellenség egy irányba) [2]
LastX	byte	A sprite előző X pozíciója [3]
LastY	byte	A sprite előző Y pozíciója [3]
Position	integer	A sprite pozíciója a képernyőn vagy a háttérképernyőn [3]
LastPosition	integer	A sprite előző pozíciója [3]
SpriteType	byte	A sprite típusa [3] - **1**: szimpla; **2**: maszkolt
Width	byte	A sprite szélessége byte-okban [3]
Height	byte	A sprite magassága pixelekben [3]
Size	integer	A sprite mérete byte-okban [3] (Width * Height)
Data	pointer	A sprite képét és maszkját tartalmazó adatra mutat [3]
Background	pointer	A sprite hátterének elmentésére szolgáló pufferre mutat
ToNextRow	byte	A sprite kirakásához szükséges előre kiszámolt érték [3]
AnimSet	pointer	Az animáció fázisaira mutat [3]
AnimType	byte	Az animáció típusa *(jelenleg nem használt)* [3]
AnimPhaseCount	byte	Animációs fázisok száma [3]
AnimPhaseNr	byte	Az aktuális animációs fázis sorszáma [3]
AnimDelayValue	byte	Szünet mértéke két animációs fázis között [3]
AnimDelay	byte	Két animációs fázis közötti szünet számlálója [3]

[1] Szabadon állítható az értéke, ezzel a sprite-ok megjelenítése ki-be kapcsolható. A csoportos sprite-kirakó eljárások estében van hatása, mint a **RenderAllSprites**, **CopyAllSpritesToScreen**, **DrawAllSprites.** De ha a programban egyesével történik sprite-ok kirakása, a változó akkor is lekérdezhető, és értékének függvényében történhet a sprite-ok megjelenítése.

[2] tetszőlegesen felhasználható változók, a *Game Engine* ezeket nem használja semmire.

[3] ezek a *Game Engine* belső használatú változói, így semmiképp nem ajánlatos módosítani az értékeiket.

7.2 Sprite.Init

Ezzel az utasítással kell inicializálni egy sprite-ot. Az **Init**-nek az első megjelenítés előtt meg kell történnie.

Init(GE::ImagePtr : pointer; GE::X, GE::Y : byte; GE::GetBackgroundBuffer : boolean);

Paraméterek:

- **ImagePtr**: a sprite bináris adatainak memóriacíme
- **X**: a sprite vízszintes kezdőpozíciója a képernyőn (0-63)
- **Y**: a sprite függőleges kezdőpozíciója a képernyőn (0-239)
- **GetBackgroundBuffer**: lefoglalja-e automatikusan a sprite hátterének elmentésére a helyet

Például:

```
var
  PlayerSpriteBin : incbin("sprites/PlayerSpr.bin");
  PlayerSprite : GE::Sprite;

begin
   PlayerSprite.Init(#PlayerSpriteBin, 0, 100, True);
end.
```

Amennyiben háttér előtt történik a sprite mozgatása, akkor el kell menteni a sprite hátterét a mozgatás előtt, hogy legyen honnan visszaállítani azt, ha a sprite elmozdul. Ehhez le kell foglalni egy memóriaterületet, ahová ez elmentésre fog kerülni. Amennyiben a **GetBackgroundBuffer** paraméterben **True** érték kerül átadásra, akkor a *Game Engine* automatikusan lefoglalja ezt a memóriaterületet. Ehhez kezdetben a BASIC rendszer által használt memóriaterületet fogja felhasználni a **$100 - $73F** tartományban, majd ha ez esetleg elfogyna, akkor a memória legfelső címétől visszafelé kezd memóriát foglalni ehhez. Háttérképernyős megjelenítés esetén a $BFFF memóriacímtől, direkt képernyős megjelenítés esetén pedig 64k-s gép esetében az $FFFF memóriacímtől, 32k-s gép esetében pedig a $7FFF memóriacímtől.

Ha egyszínű háttér előtt történik a sprite mozgatása, akkor a **GetBackgroundBuffer** paraméterben **False** értéket kell átadni.

Nem muszáj a *Game Engine*-re bízni a sprite hátterének elmentésére szolgáló memóriaterület lefoglalását, manuálisan is meg lehet adni azt, például így:

```
var
  PlayerSpriteBin : incbin("sprites/PlayerSpr.bin");
  PlayerSprite : GE::Sprite;
  PlayerBackground : array[40] of byte;

begin
   PlayerSprite.Init(#PlayerSpriteBin, 0, 100, False);
   PlayerSprite.Background = # PlayerBackground;
```

vagy így:

```
   PlayerSprite.Background = $100; /* a hexadecimális $100 memóriacímtől */
```

7.3 Sprite.Put

Nem maszkolt, azaz 1-es típusú sprite esetén, ahol a sprite gyakorlatilag egy négyzet alakú képdarab, ezzel az utasítással lehet megjeleníteni a sprite-ot.

PlayerSprite.Put;

Amennyiben háttérképernyős megjelenítéssel lett inicializálva a *Game Engine*, akkor ez az utasítás a háttérképernyőre fogja kitenni a sprite-ot, és egy további utasítással, a **CopyToScreen**-nel, lehet a képernyőre kimásolni. Ha direkt a képernyőre dolgozik a *Game Engine*, akkor ezzel az utasítással a sprite azonnal megjeleníthető a képernyőn.

7.4 Sprite.Draw

A maszkolt, azaz **2**-es típusú sprite esetén, ahol a sprite háttere átlátszó, ezzel az utasítással lehet megjeleníteni a sprite-ot.

PlayerSprite.Draw;

Amennyiben háttérképernyős megjelenítéssel lett inicializálva a *Game Engine*, akkor ez az utasítás a háttérképernyőre fogja kitenni a sprite-ot, és egy további utasítással, a **CopyToScreen**-nel, lehet a képernyőre kimásolni. Ha direkt a képernyőre dolgozik a *Game Engine*, akkor ezzel az utasítással a sprite azonnal megjeleníthető a képernyőn.

FONTOS! *Vannak később ismertetésre kerülő eljárások, amik egy lépésben kirakják az összes inicializált sprite-ot. Érdemes ezeket használni az egyenkénti **Put** és **Draw** helyett, mert automatikusan mentik és visszaállítják a sprite-ok háttereit, és frissítik a sprite-ok utolsó pozícióit is. A **Put** és **Draw** eljárások egyéni képfrissítések esetén használhatóak, például ha az a cél, hogy csak azok a sprite-ok legyenek újra kirakva, amik elmozdultak, és a játék egy ciklusán belül nem kerül mozgatásra az összes sprite. Ezzel gyorsítható a játék sok sprite esetén.*

7.5 Sprite.FastDraw

A maszkolt, azaz **2**-es típusú sprite esetén, ahol a sprite háttere átlátszó, ezzel az utasítással is meg lehet jeleníteni a sprite-ot.

PlayerSprite.FastDraw;

Működése megegyezik a **Draw** eljárással, de annál gyorsabb. Hátránya, hogy letiltja a megszakításokat, mert az **SP** regisztert is felhasználja a megjelenítéshez, emiatt tud gyorsabb lenni.

Ezért csak akkor érdemes használni, ha nem probléma a megszakítások letiltása. Például, ha nincs digitális hanglejátszás a játékban, vagy a megjelenítés nem várja meg a *cursor-megszakítás*t, vagy megvárja, de egy képernyőfrissítés alatt biztosan le is fut az összes sprite kirakása **FastDraw** eljárással.

7.6 Sprite.SetPosition

Ha a sprite mozgatása az **X** és **Y** pozíciói alapján történik, akkor a sprite kirakása előtt meg kell hívni a **SetPosition** eljárást fizikai képernyő pozíció meghatározásához.

Például:

```
PlayerSprite.X = PlayerSprite.X + 1;
PlayerSprite.SetPosition;
```

FONTOS! *Vannak beépített utasítások is a sprite mozgatására (például: **MoveLeft, MoveRight, MoveUp, MoveDown**, stb.), ezek használata esetén nincs szükség a **SetPosition** meghívására.*

7.7 Sprite.UpdateLastPos

A sprite utolsó pozíciójának frissítése. Az itt frissített pozíciót fogja használni az elmentett háttér visszaállításához, ezért ezt a sprite kirakása után, vagy a háttérképernyőről a képernyőre másolása után kell beállítani.

FONTOS! *Csak egyedi sprite kirakás esetén használatos, a később ismertetésre kerülő, az összes sprite-ot egy lépésben kirakó eljárások automatikusan beállítják a sprite utolsó pozícióját.*

7.8 Sprite.MoveLeft

A sprite mozgatása egy byte-tal balra.

Például:

```
PlayerSprite.MoveLeft;
```

7.9 Sprite.MoveRight

A sprite mozgatása egy byte-tal jobbra.

Például:

```
PlayerSprite.MoveRight;
```

7.10 Sprite.MoveUp

A sprite mozgatása egy pixellel felfelé.

Például:

PlayerSprite.MoveUp;

Van két további utasítás is, ami felfelé mozgatja a sprite-ot:

MoveUpBy2 – 2 pixellel mozgatja fentebb a sprite-ot.
MoveUpBy4 – 4 pixellel mozgatja fentebb a sprite-ot.

7.11 Sprite.MoveDown

A sprite mozgatása egy pixellel lefelé.

Például:

PlayerSprite.MoveDown;

Van két további utasítás is, ami felfelé mozgatja a sprite-ot:

MoveDownBy2 – 2 pixellel mozgatja lentebb a sprite-ot.
MoveDownBy4 – 4 pixellel mozgatja lentebb a sprite-ot.

A a fenti mozgató utasításoknál nagyobb lépésben szükséges mozgatni a sprite-okat, vagy egy adott képernyő pozícióba kell mozgatni azokat, akkor az **X** és **Y** változókon keresztül lehet ezt megtenni. A változók beállítása után meg kell hívni a **SetPosition** eljárást.

Például így:

```
PlayerSprite.X = 30;
PlayerSprite.Y = 120;
PlayerSprite.SetPosition;
```

7.12 Sprite.SaveBackground

Ha a sprite nem egyszínű háttér előtt mozog, akkor szükség van a hátterének elmentésére, ami ezzel az utasítással végezhető el.

PlayerSprite.SaveBackground;

Ezt az eljárást legalább egyszer használni kell, amikor a játék megkezdése előtt a hátterek már kirajzolásra kerültek a képernyőre (vagy a háttérképernyőre), és a sprite kezdőpozíciója is beállításra került az **Init** utasítással, vagy az **X** és **Y** változókon keresztül, majd a **SetPosition** eljárást használva.

Például:

```
PlayerSprite.Init(#PlayerSpriteBin, 0, 100, True);
PlayerSprite.SaveBackground;
```

FONTOS! *Az összes sprite-ot egyetlen utasítással, a* ***GE::RenderAllSprites****-al is ki lehet rakni a képernyőre. Ebben az esetben nem szükséges minden kirakás előtt elvégezni a sprite-ok hátterének elmentését, mert ezt a később ismertetésre kerülő utasítások automatikusan elvégzik. De a játék főciklusa előtt egyszer meg kell ezt tenni.*

7.13 Sprite.RestoreBackground

A sprite elmentett hátterének visszaállítása.

PlayerSprite.RestoreBackground;

Ezt a sprite kirakása előtt kell meghívni, ha manuálisan, egyesével kerülnek kirakásra a sprite-ok.

FONTOS! *Az összes sprite-ot egyetlen utasítással, a* ***GE::RenderAllSprites****-al is ki lehet rakni a képernyőre. Ebben az esetben nem szükséges minden kirakás előtt elvégezni a sprite-ok hátterének visszaállítását, mert ezt a később ismertetésre kerülő utasítások automatikusan elvégzik.*

7.14 Sprite.CopyToScreen

Sprite kimásolása háttérképernyőről a képernyőre.

PlayerSprite.CopyToScreen;

FONTOS! *Az összes sprite-ot egyetlen utasítással, a* ***GE::CopyAllSpritesToScreen****-el is ki lehet másolni a háttérképernyőről a képernyőre. Ebben az esetben nem szükséges ezt az eljárást meghívni.*

7.15 Sprite.InitAnimation

Sprite animáció inicializálása.

InitAnimation(AnimPhaseCount, AnimDelay : byte);

A sprite-ot inicializálni kell az **Init** eljárással, ahol a sprite bináris adatának tartalmaznia kell a sprite összes animációs fázisát. Ezt követően lehet meghívni az **InitAnimation** eljárást.

Paraméterek:

- **AnimPhaseCount**: animációs fázisok száma
- **AnimDelay**: hányadik hívásra lépjen a következő animációs fázisra

7.16 Sprite.Animation

Sprite animálása.

Az eljárás nem jeleníti meg a sprite-ot, csak beállítja, hogy melyik animációs fázist kell majd kiraknia.

Egyelőre csak egyféle animáció típust támogat a *Game Engie*, ez pedig a körkörös animáció, ami azt jelenti, hogy az első fázistól az utolsóig tart az animálás, majd az utolsó fázis után ismét az első fázisra ugrik.

Amennyiben az **InitAnimation** eljárásban az **AnimDelay** értéke **1**, akkor minden hívásra a sprite a következő animációs fázisra lép. Ha az **AnimDelay** értéke **2**, akkor csak minden második hívásra lép a következő animációs fázisra.

Példa:

```
var
  FanAnim : incbin("ssprites/FanAnim.bin");
  FanAnimSprite : GE::Sprite;
begin
  FanAnimSprite.Init(#FanAnim, 48, 170, false);
  FanAnimSprite.InitAnimation(4, 3);
  /* játék főciklus */
  while (true) do
  begin
    FanAnimSprite.Animation;
    FanAnimSprite.Put;
   end;
end.
```

8. Sprite-ok csoportos megjelenítése

Ha minden sprite a játék minden ciklusában megjelenítésre kerül, akkor ebben az esetben érdemes az alábbi eljárásokat használni, amik nagyban leegyszerűsítik a sprite-ok kezelését.

Egyes sprite-ok megjelenítését a **Visible** változójukon keresztül külön is ki-be lehet kapcsolni a játék futása közben csoportos megjelenítés esetén, például, ha kilövésre került egy ellenség.

8.1 RenderAllSprites

GE::RenderAllSprites;

Ez az eljárás az összes sprite-nak automatikusan menti és visszaállítja a hátterét, valamint a típusának megfelelően a **Put** vagy a **Draw** eljárással kirakja azokat a háttérképernyőre, vagy a képernyőre, attól függően, hogy a *Game Engine* **Init**-ben háttérképernyős, vagy direkt képernyős működés lett-e beállítva, majd frissíti az utolsó pozícióikat is (**LastX**, **LastY**, **LastPosition**).

Az eljárás meghívása előtt csak sprite-ok mozgatásait kell elvégezni az **X** és **Y** koordinátáik beállításával, majd a **SetPosition** eljárás meghívásával, vagy a **MoveLeft, MoveRight, MoveUp, MoveDown** stb. eljárásokkal.

Az eljárás figyelembe veszi a sprite-ok **Visible** változóját, és ha az **false** értéken áll, akkor az adott sprite nem lesz megjelenítve.

A sprite-ok az inicializálásuk (sprite.Init) sorrendjében lesznek kirakva, azaz, ha átfedésbe kerülnek, akkor a később inicializált kerül „fentebb”.

8.2 CopyAllSpritesToScreen

GE::CopyAllSpritesToScreen;

Ha háttérképernyős működés kerül beállításra, akkor a **RenderAllSprites** eljárás után a **CopyAllSpritesToScreen** eljárást is meg kell hívni, ami kimásolja az összes sprite-ot a háttérképernyőről a képernyőre.

Az eljárás figyelembe veszi a sprite-ok **Visible** változóját, és ha az **false** értéken áll, akkor az adott sprite nem lesz kimásolva a képernyőre.