Creare videogiochi DOS retrò
Multipiattaforma nel 2018

18-11-2018 Bertinetto Bartolomeo Davide

Indice

Ringraziamento
Premessa
Materiale necessario
Restare nei limiti hardware
Obbiettivi di questo Ebook
Come ho scritto questo manuale
Come compilare i listati
Creare eseguibili DOS Multipiattaforma grazie a DosBox
Nel vivo
Primo livello del fondale
Secondo livello di fondale, prende il via l'effetto del parallelasse
Aggiungiamo lo scorrimento di isole e nuvole
Scorrimento Mare
Compare lo sprite del protagonista
Movimenti autonomi del nemico
Collisione sprite
Libreria separata 'buff.h'
Comando aggiuntivo 'doppiobuffering()'
Primo comando per la gestione del fondale 'mappafondale()'
Comando di gestione sprite 'personaggio()'
Vediamo ora una breve rassegna dei comandi utilizzati nel listato
Opzioni aggiuntive, qui non contemplate
Listato completo del file 'bicico.c'
Audio nei nostri lavori retrò
Avvio dell'audio su una macchina reale con CPU 386 o superiore
Autoavvio da Floppy
Conclusione
Altri miei lavori dedicati ad Allegro.h

Ringraziamento

Ringrazio l'autore del videogioco del quale ignoro l'identità, che io conosco come Bicicotteri, con titolo originale ManCopter. Un videogioco stupendo che ai tempi dei miei 11-13 anni ha accompagnato molte partite a casa mia insieme a molti amici e naturalmente al mitico Commodore 64. Un ringraziamento a te Fabio che tanti anni fa eravamo inseparabili, quando eravamo piccoli e ci divertivamo con questo capolavoro. Ora non ci sei più ma ti penso spesso anche se sei lassù... Ciao

Premessa

Finalmente riesco ad iniziare questo libro, al quale pensavo da molti anni... E' stato un percorso particolare il mio nel settore della programmazione, fatto di alcune intersezioni significative che citerò piu' avanti. Si tratta di una passione che nasce già dall'adolescenza con quei primi Commodore 64, Amiga, e PC 386... Tutto allora, se ci ripenso ora, sembrava già prendere la piega del retrò gaming. Se guardo indietro ho come l'impressione che il videogioco di allora sia quello per antonomasia, a vita.
A quei tempi passavo le giornate a scuola alle superiori a dibattere col mio amico Luigi. Lui era un 'amighista' convinto ed io ancora un possessore dell'ormai già allora vetuso, Commodore 64 ma aspirante 'pciista'. Erano gli anni tra il 1990 ed il 1992. Interminabili parole e discussioni sulle potenzialità di quelle macchine primordiali, C64 a parte. Sulla bravura dei programmatori nello sfruttare appieno le risorse di Amiga ma ancora di piu' per il piccolo 64, erano veramente dei 'guru'

capaci di 'tirare fuori' ogni cosa da quegli insiemi di chip. Le cose erano già un pò diverse per il mondo PC, dove iniziava ad apparire la tendenza moderna fatta di programmazione basale supportata da hardware in continua crescita che rendeva vero il motto per il quale: 'Se un videogioco oggi non gira bene, tra 6 mesi uscirà un PC in grado di farlo funzionare al meglio!'
Bé in realtà anche se a quei tempi ero appassionato di PC ho sempre ammirato/invidiato la filosofia Amiga, senza per altro mai ammetterlo con Luigi. Così oggi sono qui che realizzo un manuale fatto di tendenze retrò atte a sfruttare hardware di allora, con l'obbiettivo di realizzare quei 'primi veri videogiochi' senza superare le specifiche della macchina a disposizione. Quindi il minimo hardware per fare praticamente ogni cosa in quel 1990 e poco oltre: CPU486, VGA 256 colori, 4MB di ram, un floppy, disco fisso opzionale'. Un pò come si faceva con Amiga, magari creando un autoboot da floppy, del quale tanto si vantavamo i Commodoristi di quei tempi e che mancava alle classi x86!
Per la cronaca ho testato tutto il materiale che qui vi propongo con un 'clone' del mio vecchio PC 386 a 40mhz, e funziona!

Materiale necessario
Gli strumenti che voglio qui proporre per realizzare 'veri' videogiochi in stile Dos retrò sono:

- La libreria C allegro.h versione 3.0, fantastica risorsa che permette di dare vita a qualsiasi lavoro videoludico di allora sfruttando al meglio le risorse di sistemi 386 o superiori audio sound blaster e grafica VGA, senza l'utilizzo del linguaggio Assembly.
- L'editor C RHIDE con il compilatore DJGPP, che mette a disposizione un interfaccia unica ed interamente Dos per realizzare qualsiasi software e non solo giochi. Questo editor gira solo su macchine 486 o superiori(I possessori di 386 possono usare il comando EDIT del dos o altri editor di testo ampiamente disponibili per DOS sul web).
- Il prodigioso emulatore che permette di far funzionare egregiamente i lavori proposti in questo manuale su piattaforme moderne è senza dubbio DOSBOX! Ciò nel caso che credo sarà scontato, decidiate di usare un PC moderno e non una vera macchina retrò.
- Sarà comunque dedicata una sezione appositamente studiata(alla fine dell'Ebook) per avviare i lavori qui descritti su una vera piattaforma 386 o 486, evitando l'emulazione. A tal proposito proporrò l'uso di una speciale procedura Basata sull'autoavvio con il sistema operativo freeware FREEDOS, che permetterà di utilizzare i videogiochi che creeremo direttamente su vecchie macchine con boot da floppy!
- Il tool di sviluppo scaricabile dal mio sito web contiene DosBox per Windows. Se qualcuno vuole produrre su altri sistemi sarà sufficiente sostituire l'eseguibile dell'emulatore:

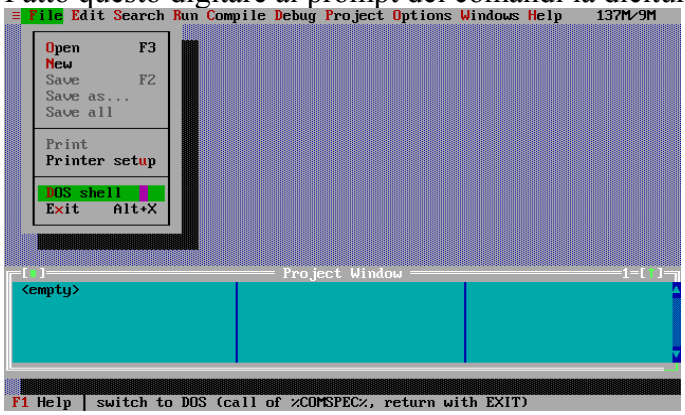
http://www.bertinettobartolomeodavide.it/programmazione/ALLEGROH/GiochiDosRetro/index.html

Restare nei limiti hardware
Voglio però proporre, al fine di rispettare alcuni requisiti di sviluppo in voga nel 1990, di non abusare delle richieste di sistema facendo sì che i lavori realizzati siamo effettivamente avviabili da quelle ormai obsolete quando bellissime macchine. Sicuramente il 90% dei programmatori retrò svilupperà il tutto attraverso DOSBOX, tuttavia è bene poter provare la propria 'fatica' con avvio funzionante su un sistema reale.
Pertanto, come già detto, deve essere possibile un boot su un sistema compreso tra 386 e Pentium I. Si deve possedere una grafica non superiore all'allora diffusissima VGA 320x200 pixel a 256. L'audio deve essere di tipo Sound Blaster. Tutto il lavoro deve poter essere contenuto su un singolo floppy disk da 1,44Mb. Proprio quest'ultimo punto è sempre stato fortemente criticato dalle vecchie riviste del tempo dato che per PC in lavori non erano certamente parsimoniosi in termini di spazio occupato(come del resto anche oggi), forti delle macchine sempre dotate di 'capienti' dischi fissi. Altri sistemi come Amiga riuscivano a fare miracoli occupando spesso un dischetto da 880kb! E' mia premura in questo Ebook azzerare tale differenza, imponendo il rispetto ai programmatori appassionati di retrò che leggono il presente lavoro di questa 'prima direttiva'.

Obbiettivi di questo Ebook
Attraverso passaggi graduali verrà proposto lo sviluppo di uno pseudo remake di un glorioso titolo C64, ovvero BICICOTTERI. Con uno stile di sviluppo a compartimenti, verranno analizzati alcuni temi dedicati ad una apposita libreria nella quale sono 'creati/contenuti' comandi ricorrenti, poi un successivo lavoro di listati sempre funzionanti dei vari elementi che compongono la scena di gioco, come ad esempio fondale, personaggi, comandi e regole varie...
Procedendo in questo modo credo che il lettore potrà maggiormente comprendere tutti i meccanismi. Vedere le cose un pezzo alla volta a mio avviso permette di assimilare meglio il modo di procedere per completare un videogame nel suo insieme.
Ho già tentato in passato la via, in alcuni vecchi tutorial, di spiegare tutto un listato attraverso un esempio unico. Nonostante ciò anche se ricco di commenti un listato di centinaia o migliaia di linee fa 'perdere il filo' all'aspirante programmatore! Un vero peccato. E' successo molte volte anche a me di visionare esempi che abbracciavano varie sfaccettature di un listato per arrivare tristemente a non capirci nulla. La soluzione della 'suddivisione' al contrario la ritengo davvero valida.

Come ho scritto questo manuale
Non posso fare a meno di menzionare il software che ho utilizzato per stilare le pagine che state leggendo. Infatti, come ho menzionato sopra, sono appassionato di alcune piattaforme informatiche retrò ed in particolare del buon vecchio Commodore Amiga, che ho posseduto hai tempi nella sua versione A1200, della quale conservo ancora un paio di modelli accelerati... Bene, il wordprocessor che sto utilizzando è il portentoso, quanto attuale, AmigaWriter! Un software ancora in grado di regalare importanti soddisfazioni ad ogni tipo di scrittore che ha ben poche differenze con altri programmi piu' moderni che girano su Windows o Linux. Si è vero non l'ho faccio girare su un Amiga classic salvo che in rare occasioni, anche se ogni tanto mi prendo questa libertà. Mi spiace utilizzare troppo quei vecchi 'macinini' perché sento che ad ogni accensione ne sottraggo una alla loro già lunga vita! Infatti funzionano ancora benissimo dopo circa trent'anni(2018) e questo la dice lunga su come lavorava bene la grande C! Sfido ogni computer attuale a raggiungere una longevità simile...
Per Amiga classic esiste una versione di DosBox ma non è stabile e per giunta incompleta visto che ad ora manca la parte audio(nell'ultima versione sembra essere tutto presente ma lentissima!), nonostante tutto è possibile avviare i lavori basati sulla libreria allegro.h ma ahimè invece il compilatore manda in crash l'emulatore. Peccato perché sarebbe stato bellissimo realizzare tutto quanto su AmigaOS 3.9. Se qualcuno vuole però può procurarsi un recente modello Amiga basato su PowerPC e fare tutto su quella piattaforma, bypassando Windows e company... Funzionerà benissimo!
PS. Ho voluto comunque forzare la mano rilasciando sul mio sito web una versione di questo tool avviabile da un vero amiga. Perdonatimi se però non è perfetto!
Tenendo ancora conto che i nostri lavori sono compilati per allegro.h 3.0, quindi in versione Dos e girano su Dosbox in ogni sua forma: Java(web e sistema), su tante console(Xbox, WII, Playstation...), su tutte le forme esistenti di cellulari recenti o passati! Quindi i nostri lavori sono veramente multipiattaforma dato che se funziona l'emulatore Dosbox funzionano anche i software che realizziamo.

Come compilare i listati
Bè per compilare i listati in linguaggio C che troverete tra poco potrete usare due modi:
- Uno è quello più comodo di configurare l'editor RHIDE per gestire il compilatore DJGPP con la libreria allegro.h e quindi prediligere questa soluzione automatizzata(già standarizzata nella versione in download con questo lavoro). Per cui andate col mouse su 'Compile', quindi 'build all'. Fatto ciò spostatevi sul menù 'Run'. Naturalmente prima di tutto ciò dovrete aver caricato/creato un file '.c' da compilare...
- un altro sistema è quello di uscire in 'task' Dos dall' editor, cliccando 'FILES' e poi 'DOS SHELL'. Fatto questo digitare al prompt dei comandi la dicitura: gcc bicico.c -o bicico.exe -lalleg -lppcol



Questa seconda soluzione è quella che ho utilizzato per creare ogni eseguibile necessario a questa guida. L'ho travata più ricca di opzioni dato che la finestra Dos ti permette di agire immediatamente sui files se necessario...
Un software fondamentale per creare files DAT, archivi dove sono condensati tutti i materiali multimediali dei nostri programmi, è di certo **grabber.exe** che trovate nella cartella principale del tool di sviluppo. Sarà necessario avviarlo all'interno dell'emulatore DOSBOX o da DOS SHELL.

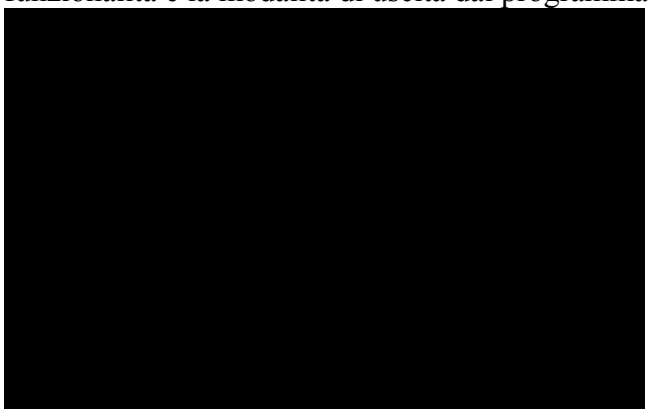
Creare eseguibili DOS Multipiattaforma grazie a DosBox
Ho inserito tre esempi concreti dove si dimostra che è possibile realizzare lavori che si avviano su altre piattaforrme differenti da quella nativa di sviluppo. Qui trovate sotto la cartella 'eseguibile multipiattaforma dosbox' nel tool di sviluppo versioni avviabili da: Windows, Amiga68k, Java. Se ne possono creare molte altre...
Nella cartella DosBox trovate l'eseguibile dell'emulatore, trovate il file dosbox.conf e proprio in questo viene aggiunta la dicitura:

```
mount c bicico
c:
CWSDPMI.EXE
bicico.exe
exit
```

Tale sequenza di comandi permette di montare come 'C' la cartella 'bicico', spostare il lavoro in 'C:', poi attivare il software per il file di swap ed in fine avviare l'eseguibile del nostro lavoro. Si chiude il tutto con il comando 'exit' che all'uscita del nostro lavoro permette anche la fine dell'emulatore.
In questo manuale non espongo la tecnica ma è possibile, anzi consigliabile, modificare l'icona dell'emulatore per renderla consona al tema del nostro lavoro DOS. Un buon programma PC è EasyIco, mentre su amiga esiste un app(Icon Edit) nella cartella Tools di AmigaOS 3.9.
Qui negli esempi per praticità ho voluto racchiudere il videogioco all'interno della cartella 'bicico', consiglio di mantenere attiva tale schematizzazione, visto che a prescindere della posizione della cartella sul disco fisso, l'emulatore in questo modo sarà in grado di operare e trovare tutti i files!
Non aggiungo altro, con la speranza di essere stato abbastanza esaustivo in questo titoletto sull'effettiva praticità di sviluppare giochi DOS multipiattaforma. Anche se possono apparire in conflitto queste due ultime parole, nella presente sede si abbracciano!

Nel vivo

Tra poco entrerò nel vivo dell'inizio lavori. Partirò così dal primo listato contenuto in questo titoletto con un sorgete 'vuoto'. Ovvero il termine 'vuoto' non è del tutto esatto visto che le linee che verranno esposte tra poco attiveranno la libreria allegro.h in Dos, daranno il via alla pagina grafica e alle funzionalità primarie per gestire i nostri programmi. Un ciclo 'while', dove saranno inserite le funzionalità e la modalità di uscita dal programma... Vediamo:



```
// Crea i presupposti per l'avvio della modalità grafica
// e molte altre funzionalità, il ciclo 'while' e l'uscita

#include "allegro.h" // Libreria allegro.h

// Inizia la routine principale
int main ()

allegro_init(); // Inizializzo allegro.h
install_keyboard(); // Inizializzo tastiera
install_mouse(); // Inizializzo il mouse

set_gfx_mode(GFX_VGA, 320, 200, 0, 0); // attivo la modalità grafica VGA(320x200
pixel a 256 colori)

// ciclo del gioco
while (!key[KEY_ESC]) // CREA UN CICLO INFINO FINO ALLA PRESSIONE DEL
TASTO

allegro_exit(); // termina allegro.h ed il programma
```

Il primo listato sopraesposto possiede i presupposti per dare il via a tutto. Tengo a precisare che nel file sorgente ho incluso ogni listato, in modo che sia immediatamente compilabile e testabile dal compilatore DJGPP per Dos. File: 00vuoto.c

Primo livello del fondale

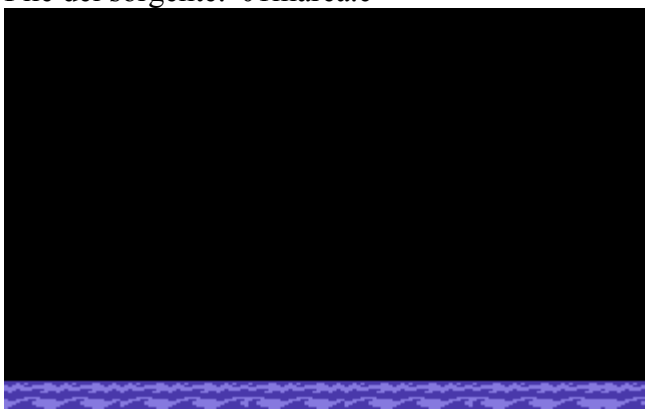
Con questa occasione si inizia a caricare il primo file 'dat' contenente i files grafici del primo livello di scorrimento del mare. Anche le variabili vengono dichiarate, almeno quelle necessarie per l'avvio di queste poche istruzioni...

E' importante notare il caricamento della palette per la corretta gestione dei 256 colori su schermo. Il contenuto però viene visualizzato dalla routine del doppio buffering che regola l'aggiornamento dello schermo senza il fastidioso sfarfallio, tanto conosciuto nelle 'cattive' realizzazioni degli anni ottanta e novanta!

Proseguendo con il listato si arriva all'istruzione while che dà il via al ciclo di gioco, già proposta nel listato zero. Qui si inseriscono le varie funzioni tra cui quella inserita nella libreria (separata) 'buff.h' denominata 'mappafondale' che sarà spiegata nel dettaglio in un titoletto apposito. Ancora, nel ciclo deve per forza essere collocato il meccanismo del 'doppiobuffering', il quale ad ogni aggiornamento ridisegna lo schermo.

Per concludere sono riportate le variabili di conteggio dello scorrimento del fondale, con il limiti per il 'loop infinito' grazie al comando 'if'.

File del sorgente: '01marea.c'

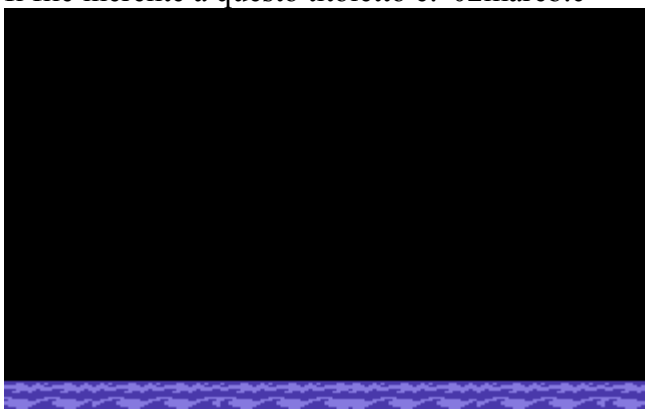


```
// Visualizza primo scorrimento del fontale per la creazione del parallelasse
#include "allegro.h" // Libreria allegro.h
#include"buff.h" // Libreria dei comandi

#include"mappa1.h" // fondale mare basso

int scorrx; // scorrimento mare basso
// float perché valore di scorrimento di mezzo punto
int animare, animare2, rallenty, rallenty2; // scorrimento mare alto


// Inizia la routine principale

int main ()

allegro_init(); // Inizializzo allegro.h

install_keyboard(); // Inizializzo tastiera

install_mouse(); // Inizializzo il mouse

set_gfx_mode(GFX_VGA, 320, 200, 0, 0); // attivo la modalit grafica VGA(320x200 pixel a 256 colori)

// carico la mappa del mare basso contenuta nel file dat - frame 1
mappa_dat = load_datafile ("mappa.dat");

set_pallete(mappa_dat[2].dat); // assegno la pallete


// Creo il puntatore per il doppio buffering necessario alla visualizzazione
```

```
buffer = create_bitmap (320,200);
clear (buffer);

// inizio del punto di scrolling
scorrx=-2432;

// ciclo del gioco
while (!key[KEY_ESC]) // CREA UN CICLO INFINO FINO ALLA PRESSIONE DEL TASTO
// scorrimento fondale mare basso
mappafondale(304,14,10,0,1,scorrx,186);

scorrx++; // scorrimento di un pixel alla volta

if (scorrx>=0) scorrx=-2432; // Raggiunto il limite di scorrimento ricomincia da capo

doppiobuffering(); // Visualizza il contenuto del buffer su schermo

allegro_exit(); // termina allegro.h ed il programma
```

Secondo livello di fondale, prende il via l'effetto del parallelasse

Gettate le basi per la visualizzazione del primo livello di scorrimento vediamo come è possibile aggiungere altri due 'strati di mare' con relativa animazione. Non è così complicato come sembra se si realizzano apposite funzioni già pronte all'uso(vedi 'mappafondale'), con tale accorgimento sarà piu' facile abbreviare il listato, rendendolo maggiormente comprensibile.
Si caricano quindi gli 'include' per i files mappa2.h e mappa3.h, con di seguito i relativi files '.dat'.
Scorrendo in basso con il codice si trovano i comandi condizionali 'if' per l'animazione e l'uso delle funzioni 'mappafondale1 e 2'...
Interagendo nel ciclo ciclo di gioco con le variabile 'rallenty' e 'animare', si vedono altre condizioni 'if' che determinano il procedere dell'animazione e la relativa velocità.
Il file inerente a questo titoletto è: '02mareb.c'



```
// Visualizza primo scorrimento del fondale per la creazione del parallelasse
// viene qui aggiunta l'animazione del primo livello di mare

#include "allegro.h" // Libreria allegro.h
#include"buff.h" // Libreria dei comandi
#include"mappa.h" // fondale mare basso
#include"mappa1.h" // fondale mare basso
#include"mappa2.h" // fondale mare basso

int scorrx; // scorrimento mare basso
// float perch valore di scorrimento di mezzo punto
int animare, animare2, rallenty, rallenty2; // scorrimento mare alto

// Inizia la routine principale

int main ()

allegro_init(); // Inizializzo allegro.h

install_keyboard(); // Inizializzo tastiera

install_mouse(); // Inizializzo il mouse

set_gfx_mode(GFX_VGA, 320, 200, 0, 0); // attivo la modalit grafica VGA(320x200 pixel a 256 colori)

// carico la mappa del mare basso contenuta nel file dat - frame 1
mappa_dat = load_datafile ("mappa.dat");
// carico la mappa del mare basso contenuta nel file dat - frame 2
m1mappa_dat = load_datafile ("mappa1.dat");
// carico la mappa del mare basso contenuta nel file dat - frame 3
m2mappa_dat = load_datafile ("mappa2.dat");

set_pallete(mappa_dat[2].dat); // assegno la pallete

// Creo il puntatore per il doppio buffering necessario alla visualizzazione
buffer = create_bitmap (320,200);
clear (buffer);
```

```
// inizio del punto di scrolling
scorrx=-2432;

// ciclo del gioco
while (!key[KEY_ESC]) // CREA UN CICLO INFINO FINO ALLA PRESSIONE DEL TASTO
// scorrimento fondale mare basso
if (animare==0) mappafondale(304,14,10,0,1,scorrx,186);
// fotogrammi di animazione del primo livello di mare
if (animare==1) mappafondale1(304,14,10,0,1,scorrx,186);
if (animare==2) mappafondale2(304,14,10,0,1,scorrx,186);

// regolo la velocit dell'animazione dei fotogrammi mare basso
if (rallenty>=30)
animare++;

rallenty++;
if (rallenty>=31) rallenty=0;
if (animare>=3) animare=0;

scorrx++; // scorrimento di un pixel alla volta
if (scorrx>=0) scorrx=-2432; // Raggiunto il limite di scorrimento ricomincia da capo
doppiobuffering(); // Visualizza il contenuto del buffer su schermo

allegro_exit(); // termina allegro.h ed il programma
```

Aggiungiamo lo scorrimento di isole e nuvole

Ogni videogioco degno di questo nome, specialmente se si tratta di una tipologia con scorrimento dello schermo, deve avere almeno un paio di livelli di parallelasse sul fondale.

Ricordo ancora le lunghe chiaccherate con un mio amico di adolescenza, quando si discuteva sulla bravura del programmatori degli anni '80/'90 e loro capacità di aggiungere numerosi strati di scorrimento. All'epoca l'Amiga la faceva da padrone!

Bé qui in questo breve corso voglio abbozzare il discorso creando in modo semplice e comprensibile qualche strato di parallelasse.

Ora espongo un nuovo codice con tutti i commenti su praticamente ogni riga per ottenere il risultato tanto ambito. Ovviamente i listati qui esposti, come per altro ho già spiegato all'inizio del testo sono tutti indipendenti per rendere piu' facile la comprensione da parte dell'utente. In realtà però nel codice finale che esporrò al termine(bicicotteri per l'appunto), saranno da immaginare fusi insieme al ciclo di gioco.

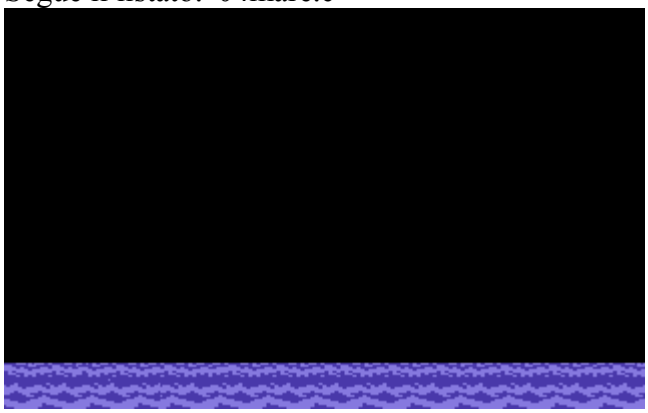
Segue ora il codice '03isloe.c', vorrei ricordare che ho nuovamente riportato la libreria 'mare.h' perché contiene il file della palette cromatica:

```
// Visualizza un nuovo scorrimento del fondale per la creazione del parallelasse
// viene qui aggiunto il livello di isole e nuvole

#include "allegro.h" // Libreria allegro.h
#include"buff.h" // Libreria dei comandi

#include"mappa.h" // fondale mare basso(contiene palette)
#include"sfondo.h" // fondale terra e nuvole

float scorrx3; // float perch valore di scorrimento di mezzo punto

// Inizia la routine principale

int main ()

allegro_init(); // Inizializzo allegro.h
install_keyboard(); // Inizializzo tastiera
install_mouse(); // Inizializzo il mouse

set_gfx_mode(GFX_VGA, 320, 200, 0, 0); // attivo la modalit grafica VGA(320x200 pixel a 256 colori)

// carico la mappa del mare basso contenuta nel file dat - frame 1
mappa_dat = load_datafile ("mappa.dat");

// carico la mappa di isole e nuvole
fomappa_dat = load_datafile ("sfondo.dat");

set_pallete(mappa_dat[2].dat); // assegno la pallete

// Creo il puntatore per il doppio buffering necessario alla visualizzazione
buffer = create_bitmap (320,200);
clear (buffer);
```

```
// ciclo del gioco
while (!key[KEY_ESC]) // CREA UN CICLO INFINO FINO ALLA PRESSIONE DEL TASTO
mappafondale6(80,160,9,0,3,scorrx3,17);
if (scorrx3>=-80) scorrx3=-560; // Ripete il fondale
scorrx3=scorrx3+0.5; // valori float per unica variabile float
doppiobuffering(); // Visualizza il contenuto del buffer su schermo

allegro_exit(); // termina allegro.h ed il programma
```

Scorrimento Mare

Qui è esposta l'intera parte dedicata allo scorrimento del mare che ci permette di vedere come muovere in parallelasse gli strati acquosi per simulare la prospettiva.
Ho potuto scomporre quest'altra parte dal 'videogioco demo' che ho realizzato sui bicicotteri per illustrare al meglio come è possibile arrivare per parti al risultato finale.
Si trovano le componenti a 'strisce' di mare che verranno ripetute all'infinito per l'intero svolgimento del programma. Ovviamente è possibile variare i contenuti a piacere in modo da rendere maggiormente diversificato un ipotetico videogioco. Molti strati dovranno scorrere a velocità differenti in modo da simulare la distanza verso l'orizzonte. Quindi i piu' vicini allo spettatore saranno molto velici, per rallentarli mano a mano che ci si allontana!
Quando ero poco piu' che bambino e vedevo con il miei compagni di gioco Daniele e Claudio, poi con Luigi davanti agli occhi, su macchine tipo C64 od il portentoso Amiga; dei veri miracoli di programmazione che spesso misuravano la bravura dei programmatori con queste 'prodezze' grafiche. Piu' c'erano livelli di scorrimento e piu' il gioco era graficamente di alto livello per i canoni di allora! Qui ne possiamo ricreare una piccola parte...
Segue il listato: '04mare.c'



```
// Visualizza lo scorrimento completo del mare per la creazione del parallelasse
// Viene qui aggiunta l'animazione del mare nel suo insieme

#include "allegro.h" // Libreria allegro.h
#include"buff.h" // Libreria dei comandi

#include"mappa.h" // fondale mare basso
#include"mappa1.h" // fondale mare basso
#include"mappa2.h" // fondale mare basso
#include"mappa3.h" // fondale mare basso
#include"mappa4.h" // fondale mare basso
#include"mappa5.h" // fondale mare basso

int scorrx, scorry, scorrx2, stage; // scorrimento mare basso
int animare, animare2, rallenty, rallenty2; // scorrimento mare alto

// Inizia la routine principale

int main ()

allegro_init(); // Inizializzo allegro.h

install_keyboard(); // Inizializzo tastiera

install_mouse(); // Inizializzo il mouse

set_gfx_mode(GFX_VGA, 320, 200, 0, 0); // attivo la modalit grafica VGA(320x200 pixel a 256 colori)

// carico la mappa del mare basso contenuta nel file dat - frame 1
mappa_dat = load_datafile ("mappa.dat");
// carico la mappa del mare basso contenuta nel file dat - frame 2
```

```
m1mappa_dat = load_datafile ("mappa1.dat");
// carico la mappa del mare basso contenuta nel file dat - frame 3
m2mappa_dat = load_datafile ("mappa2.dat");
// carico la mappa del mare alto contenuta nel file dat - frame 1
m3mappa_dat = load_datafile ("mappa3.dat");
// carico la mappa del mare alto contenuta nel file dat - frame 2
m4mappa_dat = load_datafile ("mappa4.dat");
// carico la mappa del mare alto contenuta nel file dat - frame 3
m5mappa_dat = load_datafile ("mappa5.dat");

set_pallete(mappa_dat[2].dat); // assegno la pallete

// Creo il puntatore per il doppio buffering necessario alla visualizzazione
buffer = create_bitmap (320,200);
clear (buffer);

// inizio del punto di scrolling
scorrx=-2432; scorrx2=-1216;

// ciclo del gioco

while (!key[KEY_ESC]) // CREA UN CICLO INFINO FINO ALLA PRESSIONE DEL TASTO
// scorrimento fondale mare basso
if (animare==0) mappafondale(304,14,10,0,1,scorrx,186);
// fotogrammi di animazione del primo livello di mare
if (animare==1) mappafondale1(304,14,10,0,1,scorrx,186);
if (animare==2) mappafondale2(304,14,10,0,1,scorrx,186);
// scorrimento fondale mare alto - tre fotogrammi -
if (animare2==0) mappafondale3(304,9,5,0,1,scorrx2,177);
if (animare2==1) mappafondale4(304,9,5,0,1,scorrx2,177);
if (animare2==2) mappafondale5(304,9,5,0,1,scorrx2,177);

// regolo la velocit dell'animazione dei fotogrammi mare basso
if (rallenty>=30)
animare++;

rallenty++;
if (rallenty>=31) rallenty=0;
if (animare>=3) animare=0;

// regolo la velocit dell'animazione dei fotogrammi mare alto
if (rallenty2>=30)
animare2++;

rallenty2++;
if (rallenty2>=31) rallenty2=0;
if (animare2>=3) animare2=0;

// scorrimento mare alto
scorrx2++;
// scorrimento mare basso
scorrx=scorrx+2;

// ad un certo punto "0" riprende il ciclo di scorrimento del mare ripetendolo all'infinito
if (scorrx>=0)
scorrx=-2432; scorrx2=-1216;

doppiobuffering(); // Visualizza il contenuto del buffer su schermo

allegro_exit(); // termina allegro.h ed il programma
```

Compare lo sprite del protagonista

Ora finalmente si può iniziare ad interagire con il programma e per riuscirci possiamo farlo grazie all'aggiunta di un giocatore. Con il listato di seguito, si vedrà comparire un buffo personaggio a bordo del suo bicicottero... Premendo ripetutamente il tasto di spazio lo faremo pedalare e di conseguenza decollare e volare verso l'alto dello schermo. Usando i tasti cursore 'destra' e 'sinistra' lo sposteremo avanti ed indietro. Attenzione però che se smetteremo di premere il tasto di spazio a ripetizione i pedali del bicicottero si arresteranno ed il veicolo precipiterà!!!

Non vedremo alcun fondale e quindi niente mare nuvole, isole... In questo frangente ci occupiamo solamente del personaggio. Ricordo che la routine di pedalata mi aveva dato parecchio filo da torcere ma alla fine ero arrivato all'obbiettivo, così pedalata e gravità rispecchiavano abbastanza quelle presentate nell'originale per C64.

Gustatevi il listato e prestate molta attenzione a variabili, matematica e le numerose note presenti.

Ricordo che qui non è inserita la routine di collisione ma la presenterò in seguito dopo aver scritto anche dell'antagonista nemico...

Il titolo del listato è: '05prot.c'

```
// Avvia la routine le protagonista senza collisioni

#include "allegro.h" // Libreria allegro.h
#include"buff.h" // Libreria dei comandi
#include"eli.h" // animazione Protagonista

int premo, premo2, elx, ely, ely1; // variabili protagonista
BITMAP *protagonista; // Puntatore del protagonista

// Inizia la routine principale

int main ()

allegro_init(); // Inizializzo allegro.h

install_keyboard(); // Inizializzo tastiera

install_mouse(); // Inizializzo il mouse

set_gfx_mode(GFX_VGA, 320, 200, 0, 0); // attivo la modalit grafica VGA(320x200 pixel a 256 colori)

//file dat contenete la palette del protagonista
mappa_dat = load_datafile ("mappa.dat");

// carico i fotogrammi del protagonista
eli_dat = load_datafile ("eli.dat");

set_pallete(mappa_dat[2].dat); // assegno la pallete

// Creo i buffer vuoti che conterranno i vari bitmap
// Creo il puntatore per il doppio buffering necessario alla visualizzazione
buffer = create_bitmap (320,200);
```

```
clear (buffer);

// Mappa protagonista
protagonista = create_bitmap (24,21);
clear (protagonista);

// posizione del protagonista
elx=160; ely=150;

// ciclo del gioco
while (!key[KEY_ESC]) // CREA UN CICLO INFINO FINO ALLA PRESSIONE DEL TASTO

// Azioni del protagonista

// attivazione animazione della pedalata sul protagonista
// pressione del tasto di spazio ad intermittenza per attuarla
if (key[KEY_SPACE]) // --> se premo
if (premo<=100) premo2=1;
if (ely1!=0) ely1=0;
if(ely<=0) ely=0;
if ((premo>=0) && (premo<=100)) ely--;
if (premo>100)
premo2=0;
ely++;
if (ely>=150) ely=150;

premo++;

if (!key[KEY_SPACE]) // --> se non premo
if (ely>=150) ely=150;
ely1++;
if (ely1>30) ely++;
if (premo>100)
premo2=0; premo=0;

premo++;

// movimento a sinistra
if (key[KEY_LEFT]) elx--;
if (elx<=0) elx=0;
// movimento a destra
if (key[KEY_RIGHT]) elx++;
if (elx>=300) elx=300;

// Visualizza il bicicottero del protagonista
personaggio(protagonista, 0, 0, premo2, 1, 0, 0, 0, 7, 10);
draw_sprite(buffer, protagonista, elx,ely);

// trasferisce il contenuto di gioco su schermo
doppiobuffering();

allegro_exit(); // termina allegro.h ed il programma
```

Movimenti autonomi del nemico

Ogni videogioco che si rispetti deve avere almeno un avversario da sconfiggere! Altrimenti tutto perde di scopo. Nel listato che segue illustro come inserire in un ciclo alcuni movimenti autonomi da parte di un ipotetico nemico... Si potrà agire per compartimenti stagni, assegnando vari step di movimento e rinnovarli attraverso un preciso indicatore che in questo caso è 'flagnemico1'. Tale variabile tende a salire di valore ed a ogni numero possiamo far coincidere un'azione differente. Solitamente il passaggio da un'azione di movimento all'altra è determinato dalla scomparsa dietro lo schermo dell'antagonista o dalla sua distruzione, per poi farne ricomparire un'altra dal bordo dello schermo opposto.

Tutta la macchinosità del listato di movimento autonomo è invisibile all'utente, potete però scommettere che per il programmatore è una componente di non di poca difficoltà da realizzare!

Nel listato sotto ho impartito sequenze di movimento totalmente preimpostate, ciò però non toglie che alcuni volenterosi possano realizzare routine di 'intelligenza artificiale' per ogni avversario, così da rendere sempre differente l'azione di gioco e del tutto imprevedibile.

File in cui è contenuto il codice: '06nemic.c'

```
// dimostrazione di movimento programmato per un nemico.


#include "allegro.h" // Libreria allegro.h
#include"buff.h" // Libreria dei comandi

#include"mappa.h" // fondale mare basso
#include"elicat1.h" // animazione cattivo


int n1premo, n1premo2, n1elx, n1ely, n1ely1, flagnemico1; // variabili nemico


BITMAP *primonemico; // Puntatori contenenti i bitmap del protagonista e nemico


// Inizia la routine principale

int main ()

allegro_init(); // Inizializzo allegro.h

install_keyboard(); // Inizializzo tastiera

install_mouse(); // Inizializzo il mouse

set_gfx_mode(GFX_VGA, 320, 200, 0, 0); // attivo la modalit grafica VGA(320x200 pixel a 256 colori)

// carico la mappa del mare basso contenuta nel file dat - frame 1
```

```
mappa_dat = load_datafile ("mappa.dat");

// carico i fotogrammi del nemico
elicat1_dat = load_datafile ("elicat1.dat");

set_pallete(mappa_dat[2].dat); // assegno la pallete

// Creo i buffer vuoti che conterranno i vari bitmap
// Creo il puntatore per il doppio buffering necessario alla visualizzazione
buffer = create_bitmap (320,200);
clear (buffer);

// Mappa primonemico
primonemico = create_bitmap (24,21);
clear (primonemico);

// parametri di partenza del nemico
flagnemico1=0;
n1elx=-30; n1ely=50;
// ciclo del gioco

while (!key[KEY_ESC]) // CREA UN CICLO INFINO FINO ALLA PRESSIONE DEL TASTO

// Visualizza il bicicottero del cattivo1
personaggio2(primonemico, 0, 0, 1, 1, 0, 0, 0, 7, 10);
draw_sprite(buffer, primonemico, n1elx,n1ely);

// movimento nemico1
n1ely1++; // scandisce la velocit
if (n1ely1>=5)
// Primo ciclo nemico1
// scende fino a met schermo
if ((flagnemico1==0) && (n1elx>0) && (n1elx<99)) n1ely++;
// spostamento orizzontale
if(flagnemico1==0) n1elx++;
// sale oltre la met dell schermo
if ((flagnemico1==0) && (n1elx>159) && (n1elx<250)) n1ely--;
// Secondo ciclo nemico1
if (flagnemico1==1) n1elx++;

// controllo velocit per nuovo ciclo
if (n1ely1>5) n1ely1=0;
// oltre ad un certo punto orizzontale ripete
if (n1elx>=320)
// torna alle coordinate di partenza
n1elx=-30;
n1ely=50;
n1ely1=0;
flagnemico1++;

// se non ci sono altre azioni di movimento il nemico riparte da capo...
if (flagnemico1==2) flagnemico1=0;

// trasferisce il contenuto di gioco su schermo
doppiobuffering();

allegro_exit(); // termina allegro.h ed il programma
```

Collisione sprite

Non posso in questo titoletto riportare l'intera parte di collisione fra gli spirites del giocatore e del nemico... Sarebbe necessario esporre entrambe le parti del programma e risulterebbero moltissime linee di codice! Per altro già esposti in questo libro nei titoletti precedenti. Trovate qui sotto solo un estratto. Il comando cardine per la collisione fa capo alla libreria "ppcol.h" e si tratta di: "check_pp_collsion()". Questo comando deve essere composto dalle due mappe bitmap degli sprites piu' le coordinate di posizione di entrambi. E' molto semplice, se restituisce un valore diverso da 0 esiste la collisione.
Consiglio di esaminare il file di esempio e d'istruzioni contenuti nella cartella:
allegro\ADDONS\PPCOL

La libreria "ppcol.h" è un elemento aggiuntivo di "Allegro.h" che prevede comandi preimpostati di contatto e sovrapposizione di alta precisione sui contorni.
Il listato estratto NON è compilabile: 07contat.c

```
// verifica la collisione con il primo nemico
contatto=check_pp_collision(primonemico, protagonista, n1elx, n1ely, elx, ely);

// se c' contatto il protagonista precipita (Game Over)
if (contatto==1) contattodown=1;
if (contattodown==1)
premo2=0;
ely++;
if (ely>=165) ely=165;
// visualizza la scritta di game over se c' stato contatto col nemico
textout(buffer, font, "Game Over",130,95,2);
```

Libreria separata 'buff.h'

Comando aggiuntivo 'doppiobuffering()'

Questo comando in realtà è costituito da un codice molto semplice che ho racchiuso nella funzione 'doppiobuffering()', utile per visualizzare il contenuto del puntatore 'buffer' su schermo nella modalità di doppio buffering. Ovvero con la pulizia completa della variabile schermo prima che venga visualizzata ad ogni ciclo. Accorgimento che evita lo sfarfallio dello schermo tipico dei giochi che ridisegnano le azioni direttamente su video senza prima applicarle ad un buffer, per l'appunto. Tale stratagemma era molto usato con le macchine della scena videogames fino alla metà degli anni novanta, dove, vista la loro 'limitata' velocità di calcolo, attraverso questo espediente potevano mostrare un'azione di gioco fluida.
Ora vi espongo il listato di questo comando, che per altro troverete in ogni codice del libro al fine di visualizzare il tutto su video.
Per comodità, pur essendo breve, ho racchiuso tutto nella funzione 'doppiobuffering()'. Il listato che segue non è compilabile perché rappresenta solo un estratto:

```
// Routine per la visualizzazione su schermo
BITMAP *buffer;
void doppiobuffering() // INIZIO PROCEDURA DI DOPPIO BUFFERING //

vsync(); // SINCRONIZZAZIONE AGGIORNAMENTO SCHERMO //
blit(buffer, screen, 0, 0, 0, 0, 320, 200); // VISUALIZZA IL CONTENUTO DI BUFFER SU SCHERMO //
clear(buffer); // PULISCE IL BUFFER //
```

Primo comando per la gestione del fondale 'mappafondale()':

Costruire un comando con il linguaggio C è abbastanza facile e soprattutto comodo. Infatti con tale espediente diventa possibile comprimere molto il codice del programma, riutilizzandolo all'occorrenza. Per far ottenere una migliore comprensione all'utente ho scelto di separare i vari comandi simili in modo da raggruppare a scopo didattico la 'distribuzione' di tutti gli elementi. Nonostante ciò è possibile, organizzando bene i comandi che compongono le varie funzioni, scrivere una sola volta il necessario per averlo sempre pronto all'uso... Per come vi propongo il codice, se provate a ripetere lo stesso comando in circostanze diverse potreste trovare delle anomalie dato che il listato deve essere opportunamente ottimizzato!
Ogni gioco degli anni 90 ma anche prima e poi dopo, prevedeva un fondale caratteristico alla scena di sviluppo per dare un senso grafico di ambientazione. E' un po' come la scenografica di un film.
Vediamo però come deve essere gestito nel codice il comando 'mappafondale()'.
Le variabili 'immx e immy' determinano la dimensione del singolo tassello da ripetere. Mentre 'cellx e celly' indicano quante volte deve essere ripetuto il tassello sui due assi. La voce 'tasselli' fa sapere al programma il numero di tasselli(appunto!) differenti da ripetere, spiegherò il perché: il comando si lega in modo indissolubile al DATAFILE '*mappa_dat' che non è altro che un bitmap creato a priori con un semplice programma di fotoritocco, in seguito immagazzinato nel puntatore '*mapp'.
Sono ancora da considerare le variabili di scrolling 'scrollx e scrolly', dove si impone la posizione e l'eventuale spostamento dell'intera mappa di sfondo.
Lo stratagemma di aumentare il numero di funzioni identiche a questo comando e a tutte le variabili che lo compongono, è stato quello di aggiungere un numero ad ogni fattore. Potrete constatarlo esaminando il file 'buff.h'.
Segue il codice della funzione 'mappafondale()':

```
// Mappa del mare basso
DATAFILE *mappa_dat;

BITMAP *mapp;
int collis, x, y; // risultato colore e posizione pixel letto sulla mappa
int immx,immy, risx, risy; // dimensione tasselli e risoluzione globale mappa
int cellx, celly; // numero tasselli
int colnum, tasselli; // colore da assegnare ad ogni tassello differente
char colore[80]; // variabile per caratteri stringhe testo
```

```
int sfondook, dimmap; // terminatori dalla creazione della mappa
int scrollx, scrolly; // attiva lo scrolling del fondale
/*
Valori ipotetici delle variabili come esempio:
immx=30; immy=22; // dimensione tassello
cellx=5; celly=5; // numero tasselli x ed y
tasselli=3; // Numero tasselli differenti
*/
void mappafondale(int immx, int immy, int cellx, int celly, int tasselli, int scrollx, int scrolly)

if (sfondook==0) // attende la creazione del fondale fino a 1
if (dimmap==0) // crea la mappa del fondale
risx=(immx*cellx)+immy;risy=(immy*celly)+immy;
mapp = create_bitmap (risx,risy);
clear (mapp);
dimmap=1;
for (x=cellx;x>=0;x--) //tasselli X
risx=risx-immx; // abbina i tassalli X
for (y=celly;y>=0;y--) // tasselli Y
draw_sprite(mapp, mappa_dat[1].dat,0,0); // Carica la mappa dei tasselli
risy=risy-immy; // abbina i tasselli Y
collis= getpixel(mapp,x,y); // legge i pixel sulla mappa per associarli ai tasselli

for(colnum=0;colnum<=tasselli;colnum++) // controlla che il colore dei pixel sia abbinato al tasselo giusto
if (collis==colnum) draw_sprite(mapp, mappa_dat[colnum].dat,risx,risy); // posiziona i tasselli sulla mappa nella posizione esatta

risy=(immy*celly)+immy; // riparte dall'inizio sull'asse Y

sfondook=1; // assegna uno se lo sfondo e completo
//blit(mapp, buffer, 0, 0, 0, 0, 320, 200);
draw_sprite(buffer, mapp, scrollx, scrolly); // carica la mappa nel buffer video

// sprintf(colore,"colore: %ld %ld %ld", (long)risx,(long)risy,(long)collis);
// textout(buffer, font, colore,50,50,255);
```

Comando di gestione sprite 'personaggio()':

Oltre a creare materiale rivolto alla gestione dell'ambientazione bisogna assolutamente pensare a manipolare personaggi animati che interagiscono con l'azione di gioco. Non ho nulla contro la semplice attività di programmazione tramite comandi diretti come 'draw_sprite()' di allegro.h. Secondo me però se si usa questo 'basilare' poi bisogna ogni volta ricreare tutto il contorno necessario alle attività del personaggio. Mentre se si crea un comando predefinito che poi attraverso alcune azioni di base è possibile riciclare all'infinito all'interno del listato, allora il programmatore potrà concentrarsi meglio su altri fattori del gioco senza perdere nulla di vista. Ripeto, come nel comando sopra, ottenere un listato 'snello' è una priorità se il gioco diventa ampio, od addirittura mastodontico. Non pensate che se si utilizza la vecchia capacità del floppy per amore retrò il codice debba essere per forza piccolo, perché non è così. Spesso nei giochi odierni le grandi dimensioni di CDROM, DVDROM, addirittura BLUERAY, o giochi web, gli infiniti megabytes o gigabites sono dati da suono/grafica/animazioni di tipo cinematografico ma non dal codice in se! Si i codici di oggi sono imponenti ma non perché il programmatore abbia creato chissà quale routine, perché i compilatori odierni non ottimizzano il codice così bene come facevano quelli di una 'volta', tanto le CPU attuali sono così potenti da gestire ogni spreco! Inoltre i tools di sviluppo racchiudono in se enormi listati preimpostanti che il piu' delle volte non sono minimamente necessari al gioco... Bè non divago oltre! Aggiungo comunque che il buon vecchio compilatore DJGPP ha la capacità ottimizzare molto il codice.

Nel comando 'personaggio()' che qui vi propongo il contenuto in 'buff.h', per altro ampiamente personalizzabile, trovate semplicemente una routine di animazione 'avanti/indietro' oltre alla gestione posizionale e parametri di provenienza del bitmap:

BITMAP USCITA: Puntatore dove viene caricato lo sprite
PERSX e PERSY: Posizione dello sprite
TASTO: Se viene premuto un tasto svolge un azione
ATTIVA e DISATTIVA: funzioni di attività
PARTENZA: inizia il conteggio dei fotogrammi

RIPETIZIONE: Ripete i fotogrammi all'infinito
VELOCITA: Velocità dell'animazione
Nel DATAFILE '*eli_dat' trovate i bitmaps che costituiscono l'animazione dello sprite...

PS. Come nel comando sopra ho ripetuto la funzione in altre circostanze del codice con denominazioni differenti e quindi variabili diversificate per separare i contenuti a scopi didattici... Se vi trovate bene così potrete anche voi usare questa tecnica ma se preferite potrete creare un comando indipendente completamente ripetibile all'occorrenza:

```
// routine primo personaggio
DATAFILE *eli_dat;

int frame;
int rallenta;
int mantieni;

char colore[80];
// bitmap uscita posizio x posizio y on/off n. attivazi n. disattivaz frame inizial frame avanz frame rip. azion veloc esecuz
personaggio(BITMAP* uscita, int persx, int persy, int tasto, int attiva, int disattiva, int partenza, int andata, int ripetizione, int velocita)

// ripristina il ritorno amenoche' non sia il premuto un tasto
if (tasto==disattiva) mantieni=0;

// Animazione di andata
if ((mantieni==0) && (tasto==attiva))
if (rallenta>=velocita)
frame++;
rallenta=0;
if (frame>=andata)
mantieni=1;
frame=andata;
draw_sprite(uscita, eli_dat[frame].dat, persx,persy); // sprite fermo

// Mantiene l'animazione
if ((mantieni==1) && (tasto==attiva))
draw_sprite(uscita, eli_dat[frame].dat, persx,persy); // sprite fermo
if (rallenta>=velocita)
frame++;
rallenta=0;
if (frame>=ripetizione) frame=andata;

// Riporta l'animazione al punto iniziale
if ((mantieni==0) && (tasto==disattiva))
if (rallenta>=velocita)
frame--;
rallenta=0;
mantieni=0;
if (frame<=partenza) frame=partenza;
draw_sprite(uscita, eli_dat[frame].dat, persx,persy); // sprite fermo

// sprintf(colore,"colore: %ld %ld %ld", (long)frame,(long)tasto,(long)mantieni);
// textout(buffer, font, colore,50,50,255);

rallenta++; // conteggio per la velocit dell'Animazione
tasto=disattiva; // Nessuna pressione del tasto
```

Vediamo ora una breve rassegna dei comandi utilizzati nel listato:
#include"" → fa sapere al programma quali librerie sono da includere nel listato

int e float → dichiarano il tipo di variabili che saranno usate nel programma

BITMAP *XXXX → dichiara i puntatori bitmap che conterranno i dati grafici

int main() → da inizio alla routine principale

allegro_init() → inizializza la libreria allegro.h

install_keyboard() → installa le funzionalità della tastiera

install_mouse() → installa le funzionalità del mouse

set_gfx_mode() → Fa sapere al computer quale risoluzione grafica usare

XXX = load_datafile() → carica la un file dati con all'interno elementi multimediali

set_pallette() → indica al programma come utilizzare i colori

XXX = create_bitmap() → crea un area bitmap in cui sarà caricato un puntatore

clear() → pulisce il puntatore di un area bitmap

variabile = numero → assegna un valore numerico ad una variabile già dichiarata

while → crea un ciclo di gioco infinito fino alla pressione di un tasto o un evento

if → Soddisfa una determinata condizione per determinare un evento

draw_sprite() → disegna un bitmap su scremo o su un buffer in posizione (x,y)

numero = check_pp_collision() → restituisce un valore al contatto di due sprites

textout() → visualizza un testo su un buffer o su schermo in posizione (x,y) e colore

allegro_exit() → esce dal programma disattivando la libreria allegro

Questa che ho esposto è la lista dei comandi parte delle librerie allegro.h e ppcol.h utilizzati nel programma. Vorrei però indicare il file presente nella cartella di documentazione della libreria allegro.h, denominato allegro.txt con la spiegazione dettagliata di ogni comando contenuto nella libreria. Esiste un file di testo similare per la libreria ppcol.h nella sua cartella di documentazione... Raccomando di esaminarli accuratamente.

Opzioni aggiuntive, qui non contemplate

La grandissima opportunità di rendere un videogioco creato in stile retrò è quello di arricchirlo di una presentazione accattivante, che magari sia contenuta all'interno dello stesso dischetto del videogame creato, proprio come i vecchi e bravi programmatori riuscivano a fare in quei miracoli degli anni 80 e 90. Inoltre anche la musica ha la sua parte ed è perfettamente inseribile all'interno di un videogames minimizzando l'utilizzo di memoria. In quegli anni ormai andati, riuscivano ad inserire nel floppy dedicato, musica strabiliante e magari perfino dell'audio a volte cantato...

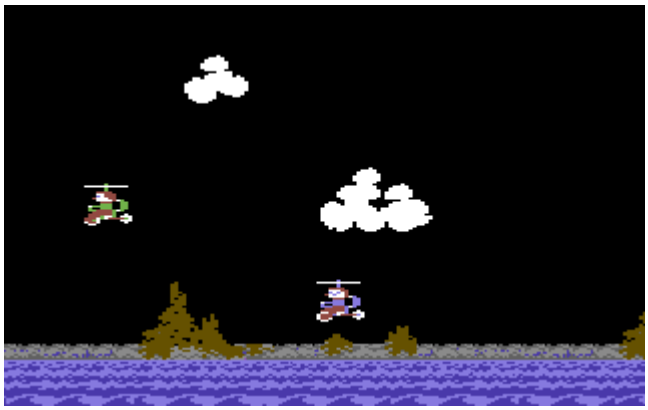
Allegro.h dispone di comandi specifici per la gestione dell'audio, sia in formato audio 'voc' che musicale 'mid'. Per di più questa libreria contiene specifiche routine per l'utilizzo di poderose animazioni Filc, sia 'fli' che 'flc'. Con questi comandi che uniti a quelli audio offerti, è possibile dedicare poche centinaia di chilobyte per una o due intro e perfino una sequenza finale.

Tuttavia però sconsiglio di utilizzare files di animazione per una presentazione o per un finale. Consiglio però la soluzione della tipica programmazione in stile 'videogame' per creare la intro. Infatti organizzando una parte del listato per una sequenza introduttiva è possibile riciclare parte dei files multimediali che già erano compresi nel gioco, inoltre sprites e suoni già presenti sul floppy saranno pronti all'uso grazie ad una mirata sequenza di comandi che potrà occupare solamente pochi bytes per arrivare al massimo ad una decina di kilobytes. Capite che con l'opzione della programmazione, che forse potrà sembrare leggermente più laboriosa, sarà possibile ottenere accattivanti 'animazioni', anche in diversi punti del videogame! Non solo ad inizio e fine.

Dopotutto allegro.h offre una vastissima gamma di comandi dedicati ad effetti di zoom, rotazione, inversione degli sprites. Oltre che dissolvenze, rotazione della palette e quant'altro. Ora non posso elencare al completo questi comandi ma vi rimando di certo al file di istruzioni di allegro.h: allegro.txt. E' tutto quanto in inglese ma senz'altro facilmente comprensibile.

Listato completo del file 'bicico.c'

Il listato che vedete qui sotto è quello principale di gioco. Non bisogna però dimenticare che molte altre parti di codice sono contenute nei files libreria '.H'. Nei titoletti precedenti sono riportate alcuni segmenti, qui tuttavia per ragioni di compattezza non li espongo nel PDF che state leggendo. E' possibile trovare tutto il contenuto dei listati nell'archivio allegato, seguendo il link per il download: http://www.bertinettobartolomeodavide.it/programmazione/ALLEGROH/GiochiDosRetro/index.html



```
// Pseudo rivisitazione didattica del vecchio videogame
// per Commdorore 64 "Bicicotteri", con approssimative
// quanto efficaci routine di programmazione C.
// Realizzato da Bertinetto Bartolomeo Davide - 18/11/2018

// Pseudo Bicicotteri

#include "allegro.h" // Libreria allegro.h
#include"buff.h" // Libreria dei comandi
#include"ppcol.h" // Libreria delle collisioni

#include"mappa.h" // fondale mare basso
#include"mappa1.h" // fondale mare basso
#include"mappa2.h" // fondale mare basso
#include"mappa3.h" // fondale mare basso
#include"mappa4.h" // fondale mare basso
#include"mappa5.h" // fondale mare basso
#include"eli.h" // animazione Protagonista
#include"elicat1.h" // animazione cattivo
#include"sfondo.h" // fondale terra e nuvole

// Dichiaro le variabili

int scorrx, scorry, scorrx2, stage; // scorrimento mare basso
float scorrx3; // float perch, valore di scorrimento di mezzo punto
int animare, animare2, rallenty, rallenty2; // scorrimento mare alto
int premo, premo2, elx, ely, ely1; // variabili protagonista
int n1premo, n1premo2, n1elx, n1ely, n1ely1, flagnemico1; // variabili nemico
int contatto, contattodown; // variabili di contatto

BITMAP *protagonista, *primonemico; // Puntatori contenenti i bitmap del protagonista e nemico

// Inizia la routine principale
int main ()
{
allegro_init(); // Inizializzo allegro.h
install_keyboard(); // Inizializzo tastiera
install_mouse(); // Inizializzo il mouse

set_gfx_mode(GFX_VGA, 320, 200, 0, 0); // attivo la modalit… grafica VGA(320x200 pixel a 256 colori)

// Il mare viene caricato a lotti di fotogrammi perch, animato

// carico la mappa del mare basso contenuta nel file dat - frame 1
```

```
mappa_dat = load_datafile ("mappa.dat");
// carico la mappa del mare basso contenuta nel file dat - frame 2
m1mappa_dat = load_datafile ("mappa1.dat");
// carico la mappa del mare basso contenuta nel file dat - frame 3
m2mappa_dat = load_datafile ("mappa2.dat");
// carico la mappa del mare alto contenuta nel file dat - frame 1
m3mappa_dat = load_datafile ("mappa3.dat");
// carico la mappa del mare alto contenuta nel file dat - frame 2
m4mappa_dat = load_datafile ("mappa4.dat");
// carico la mappa del mare alto contenuta nel file dat - frame 3
m5mappa_dat = load_datafile ("mappa5.dat");
// carico i fotogrammi del protagonista
eli_dat = load_datafile ("eli.dat");
// carico i fotogrammi del nemico
elicat1_dat = load_datafile ("elicat1.dat");
// carico i fotogrammi del fondale
fomappa_dat = load_datafile ("sfondo.dat");
set_pallete(mappa_dat[2].dat); // assegno la pallete


// Creo i buffer vuoti che conterranno i vari bitmap
// Creo il puntatore per il doppio buffering necessario alla visualizzazione
buffer = create_bitmap (320,200);
clear (buffer);


// Mappa protagonista
protagonista = create_bitmap (24,21);
clear (protagonista);


// Mappa primonemico
primonemico = create_bitmap (24,21);
clear (primonemico);


// inizio del punto di scrolling
scorrx=-2432; scorrx2=-1216; scorrx3=-560;
// posizione del protagonista
elx=160; ely=150;
n1elx=-30; n1ely=50;


// ciclo del gioco
while (!key[KEY_ESC]) { // CREA UN CICLO INFINO FINO ALLA PRESSIONE DEL TASTO


// scorrimento fondale mare basso - tre fotogrammi -
if (animare==0) mappafondale(304,14,10,0,1,scorrx,186);
if (animare==1) mappafondale1(304,14,10,0,1,scorrx,186);
if (animare==2) mappafondale2(304,14,10,0,1,scorrx,186);
// scorrimento fondale mare alto - tre fotogrammi -
if (animare2==0) mappafondale3(304,9,5,0,1,scorrx2,177);
if (animare2==1) mappafondale4(304,9,5,0,1,scorrx2,177);
if (animare2==2) mappafondale5(304,9,5,0,1,scorrx2,177);
// determina il fondale terra/nuvole con una sua velocit… di scorrimento
mappafondale6(80,160,9,0,3,scorrx3,17);
if (scorrx3>=-80) scorrx3=-560; // Ripete il fondale


// regolo la velocit… dell'animazione dei fotogrammi mare basso
if (rallenty>=30) {
animare++;
}
rallenty++;
if (rallenty>=31) rallenty=0;
if (animare>=3) animare=0;


// regolo la velocit… dell'animazione dei fotogrammi mare alto
if (rallenty2>=30) {
animare2++;
}
rallenty2++;
if (rallenty2>=31) rallenty2=0;
if (animare2>=3) animare2=0;



if (premo2==1) { // se si pedala allora scrolling
// scorrimento mare alto
scorrx2++;
// scorrimento mare basso
```

```
scorrx=scorrx+2;
// velocit… di scorrimento fondale
scorrx3=scorrx3+0.5; // valori float per unica variabile float
}
// ad un certo punto "0" riprende il ciclo di scorrimento del mare ripetendolo all'infinito
if (scorrx>=0) {
scorrx=-2432; scorrx2=-1216; stage++; // con 'stage' si indica il numero di volte in cui inizia il ciclo per aventuali altre azioni...
}
// Azioni del protagonista
if (contattodown==0) { // se c', contatto blocca i comandi
// attivazione animazione della pedalata sul protagonista
// pressione del tasto di spazio ad intermittenza per attuarla
if (key[KEY_SPACE]) { // --> se premo
if (premo<=100) premo2=1;
if (ely1!=0) ely1=0;
if(ely<=0) ely=0;
if ((premo>=0) && (premo<=100)) ely--;
if (premo>100) {
premo2=0;
ely++;
if (ely>=150) ely=150;
}
premo++;
}

if (!key[KEY_SPACE]) { // --> se non premo
if (ely>=150) ely=150;
ely1++;
if (ely1>30) ely++;
if (premo>100) {
premo2=0; premo=0;
}
premo++;
}

// movimento a sinistra
if (key[KEY_LEFT]) elx--;
if (elx<=0) elx=0;
// movimento a destra
if (key[KEY_RIGHT]) elx++;
if (elx>=300) elx=300;
}

// Visualizza il bicicottero del protagonista
personaggio(protagonista, 0, 0, premo2, 1, 0, 0, 0, 7, 10);
draw_sprite(buffer, protagonista, elx,ely);
if (stage>1) stage=0;
if (flagnemico1>1) flagnemico1=0;

// Avvia il ciclo di azione del nemico
if ((stage==0) || (stage==1)) {
// Visualizza il bicicottero del cattivo1
personaggio2(primonemico, 0, 0, 1, 1, 0, 0, 0, 7, 10);
draw_sprite(buffer, primonemico, n1elx,n1ely);

// movimento nemico1
n1ely1++; // scandisce la velocit…
if (n1ely1>=5) {
// Primo ciclo nemico1
// scende fino a met… schermo
if ((flagnemico1==0) && (n1elx>0) && (n1elx<99)) n1ely++;
// spostamento orizzontale
if(flagnemico1==0) n1elx++;
// sale oltre la met… dell schermo
if ((flagnemico1==0) && (n1elx>159) && (n1elx<250)) n1ely--;
// Secondo ciclo nemico1
if (flagnemico1==1) n1elx++;
}
// controllo velocit… per nuovo ciclo
if (n1ely1>5) n1ely1=0;
// oltre ad un certo punto orizzontale ripete
if (n1elx>=320) {
// torna alle coordinate di partenza
n1elx=-30;
n1ely=50;
n1ely1=0;
```

```
flagnemico1++; // valore relativo all'avvio di altri movimenti del nemico
}

// verifica la collisione con il primo nemico
contatto=check_pp_collision(primonemico, protagonista, n1elx, n1ely, elx, ely);

// se c', contatto il protagonista precipita (Game Over)
if (contatto==1) contattodown=1;
if (contattodown==1) {
premo2=0;
ely++;
if (ely>=165) ely=165;
// visualizza la scritta di game over se c', stato contatto col nemico
textout(buffer, font, "Game Over",130,95,2);
}
}

/*
// controlla le variabile, preenza superflua
sprintf(colore,"colore: %ld %ld %ld", n1elx, stage,flagnemico1);
textout(buffer, font, colore,50,50,2);
*/

// trasferisce il contenuto di gioco su schermo
doppiobuffering();
}
allegro_exit(); // termina allegro.h ed il programma
```

In totale sono esposte 224 linee di codice, anche se possono apparire molte non lo sono, dato che le altre parti del programma sono racchiuse nei files libreria e quindi riciclabili con i dovuti accorgimenti!

Audio nei nostri lavori retrò

Aggiungo questo titoletto in ultima battuta circa ad un anno di distanza dalla pubblicazione di questo lavoro. L'ultima volta che ho messo mano ai listati qui spiegati ero in vacanza ed ora la situazione si ripete. Purtroppo il tempo libero a disposizione scarseggia sempre così mi diletto nella programmazione solo quando sono completamente rilassato.

Con l'aggiunta dell'audio questa mia 'proposta' sulla realizzazione di giochi in stile retrò con DOS arriva al suo compimento, cosa che mi permette di chiudere una lunghissima parentesi di oramai decenni sulla realizzazione di software con la libreria allegro.h. Infatti come ho già menzionato in molte parti del presente libro, è da molto tempo che la passione Amiga cresce dentro me, ritrovando sfaccettature che oramai avevo perso ... E' diventato un vero obbiettivo per me aiutare questa grande comunità a valorizzare la grande piattaforma informatica realizzata da Commodore, che ogni giorno perde sempre più meriti in modo davvero ingiusto.

Tornando all'audio per giochi DOS, espongo qui due listati in chiave separata e completamente eseguibili, utili a capire come possono essere caricati file MIDI e WAV all'interno di un videogame. Seguendo i vari passaggi del sorgente e le note inserite non sarà difficile apprendere la modalità di gestione audio con la libreria allegro.h.

File MIDI ne esistono 'già pronti' sulla rete a milioni, se però vorrete realizzarne di vostri esistono moltissimi software(Ideal è MidiSoft Studio 4 per windows) che ve lo permetteranno, per non parlare delle numerosissime Pianole Elettroniche che possono creare files MIDI con un semplice collegamento al computer... Si tratta di files molto piccoli che non impediranno all'utente di avviare musiche molto complesse nella capacità di un vecchio floppy disk.

Inoltre l'altro listato proposto permette l'esecuzione di un file WAV. Attenzione però a non alzare troppo la qualità sonora di questi files dato che possono facilmente arrivare a decine di megabyte e se sul vostro floppy, se volete metterci un gioco completo, dovrete ridimensionarne la qualità in HZ, limitandovi in aggiunta ad usarli per brevi 'comparizioni' come possono essere ad esempio gli spari di una pistola, un urlo, una breve sigla... Tenendo conto che allegro.h non supporta alcun codec di compressione, amenoché non scegliate di comprimerlo all'interno di un file DAT.

Trovate qui esposti i due listati:

File: MIDI.C

```
/* LISTATO DI ESEMPIO PER CARICAMENTO FILE MIDI
BARTOLOMEO DAVIDE BERTINETTO
www.bertinettobartolomeodavide.it
Sezione programmazione allegro.h giochi dos retr•
*/

#include <allegro.h> // Carica la libreria allegro.h //


int main() // Inizia la procedura //
{
MIDI *music; // Dichiarazione della variabile del MIDI //

allegro_init(); // Inizializza la libreria allegro.h //
install_keyboard(); // Installa la tastiera //
install_timer(); // Installa il timer per temporizzare i suoni //

install_sound(DIGI_AUTODETECT, MIDI_AUTODETECT, 0); // Seleziona la scheda audio e il midi, il numerino e necessario solo per le vecchie versioni di allegro.h e
se da 0 e tutto ok e se da -1 indica errore//

music = load_midi("midi.mid"); // Carica il file midi .mid //

set_volume(255,255); // Setta il volume dei canali stereo destro e sinistro // // Setta il volume dei canali stereo destro e sinistro //

play_midi(music, TRUE); // manda in esecuzione il il contenuto della variabile musica, TRUE o FALSE indica il loop //

readkey(); // Attende la pressione di un tasto //

destroy_midi(music); // Distrugge il midi //

allegro_exit();
}
```

File: SAMPLE.C

```
/* LISTATO CHE CARICA UN FILE WAV
BARTOLOMEO DAVIDE BERTINETTO
www.bertinettobartolomeodavide.it
Sezione programmazione allegro.h giochi dos retr•
*/
```

```
#include <allegro.h> // Carica la libreria allegro.h //

int main() // Inizia la procedura //
{
SAMPLE *sound; // Dichiarazione della variabile del sample //

allegro_init(); // Inizializza la libreria allegro.h //
install_keyboard(); // Installa la tastiera //
install_timer(); // Installa il timer per temporizzare i suoni //

install_sound(DIGI_AUTODETECT, MIDI_AUTODETECT, 0); // Seleziona la scheda audio e il midi, il numerino e necessario solo per le vecchie versioni di allegro.h e
se da 0 e tutto ok e se da -1 indica errore//

sound = load_sample("sample.wav"); // Carica il file audio .wav //

set_volume(255,255); // Setta il volume dei canali stereo destro e sinistro //

play_sample(sound, 255,128,1000, FALSE); // manda in esecuzione il il contenuto della variabile sample, volume, posizione, frequenza, TRUE o FALSE indica il loop //

readkey(); // Attende la pressione di un tasto //

destroy_sample(sound); // Distrugge il sample
}
```

Avvio dell'audio su una macchina reale con CPU 386 o superiore
E' di estrema importanza, per far si che l'audio possa essere avviato su una macchina reale che sia presente l'hardware giusto. Se su DosBox è sufficiente lanciare l'eseguibile con i files di gioco veri e propri, in una macchina reale è necessario lanciare i drivers per la scheda audio. Qui si parla di Sound Blaster, diciamo su tutti i modelli prodotti fino alla SB16. Ho notato però che con i vari test dei componenti che ho avuto la possibilità di possedere, il primo requisito sia quello di averne una su slot ISA o EISA, su slot PCI non sono riuscito ad avviare l'audio in modalità DOS per usarlo con la libreria allegro.h.
Dopo aver assemblato un PC con i componenti richiesti non sarà difficile reperire sul web alcuni drivers di base realizzati dalla mitica casa Creative su DOS...
Per l'esattezza serve un solo file Creative per far si che tutto l'audio venga riconosciuto automaticamente: CTCM.EXE
Sarà possibile mandare tutto in esecuzione automatica dal file: AUTOEXEC.BAT
Previo però, e lo ricordo ossessivamente, che deve essere lanciato il file CWSDPMI.EXE prima del nostro eseguibile di gioco compilato con allegro.h.
Tutto si avvierà da floppy e se sarete ben organizzati con i files audio e grafici, potrete assistere alla 'magia' di vedere tutto il vostro contenuto su un 'antichissimo' floppy di solo 720KByte a doppia densità!
Puntualizzo che non posso includere il file Creative all'interno dell'archivio con i vari esempi legati a questo manuale, visto che con ogni probabilità si tratta di un file ancora protetto da diritti d'autore. Credo però che non sarà difficile per voi reperirlo sul web!

Autoavvio da Floppy
Ho inserito all'interno dell'archivio rivolto a questo manuale un file immagine di floppy denominato:
boot freedos.flp

Questo contenuto vi permette di avviare il demo qui sviluppato direttamente all'inserimento del dischetto, all'accensione del computer tutto si avvierà automaticamente. Vedrete la schermata di gioco dove potrete interagire con la tastiera direttamente sul monitor del vostro vecchio PC! Ho testato il tutto sia su VirtualBox, su un vecchio 386 e su un Pentium. Funziona bene direttamente da floppy e senza la presenza del disco fisso.
Sono circa 697Kb e l'immagine è nel formato classico di 1,44Mb, pertanto avete molto spazio in avanzo per aggiungere audio, musica, presentazioni, immagini... Ogni cosa che volete! Proprio come si faceva una volta.
All'interno trovate alcuni file obbligati, di sistema ma anche necessari all'eseguibile e gli immancabili file multimediali... Vi divertirete.

Conclusione

Ho voluto realizzare questo manuale come mio lavoro finale nei confronti della libreria Allegro.h. E' mia intenzione da tempo proporre questa idea in soluzione DOS per creare lavori multipiattaforma fusi con DOSBOX, se usati da Windows, Linux, AmigaOS o altri... Dopotutto dove esisteste un emulatore DOSBOX diventa possibile realizzare un videogame DOS! Credo sia fantastico tutto ciò e può essere stupendo utilizzare Allegro.h nella sua versione 3.0, che con pochissime modifiche permette di riciclare il gioco sui moderni sistemi... Inoltre non è da escludere la soluzione Web grazie a DOSBOX Java.

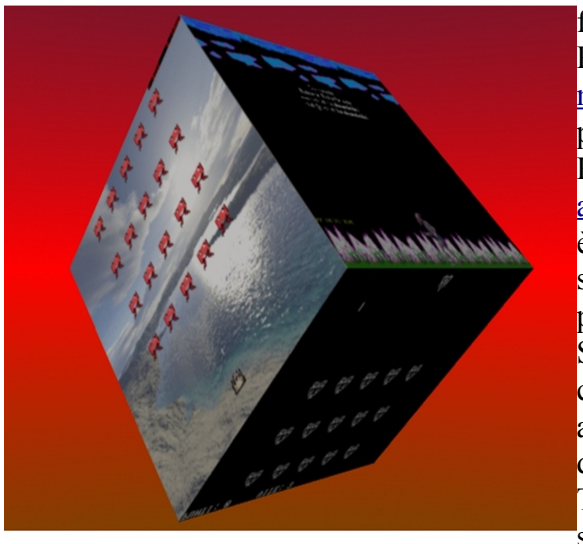
Il fulcro però di tutto ciò è una rivincita che parte da una antica diatriba tra Pciisti ed Amighisti, i quali quest'ultimi si sono sempre 'vantati' della possibilità di avviare un videogame direttamente con la pura macchina alla sua accensione, da floppy. Grazie ad un piccolo espediente, per chi possiede ancora un vecchio PC(386, 486 o Pentium I) con lettore floppy, potrà vedere il proprio lavoro retrò in modo reale!

Non realizzerò altro per Allegro.h, ma non perché sono stufo o non amo più questa libreria. Ho semplicemente voglia di dedicarmi alla piattaforma Amiga anche per la sua programmazione. A tal fine ho individuato due linguaggi che trovo strepitosi. Uno è Amibliz che permette un utilizzo completo della macchina fin dai primordi con l'antenato Blitzbasic arrivando fino ad oggi con AGA, true color, opengl e quant'altro... Con lo svantaggio di non essere multipiattaforma però. Oppure con Hollywood, altro grandioso tool di programmazione multisfaccettato per presentazioni multimediali e programmazione di ogni genere, soprattutto multipiattaforma, senza la minima variazione di codice! Windows, Mac, ogni Amiga OS e perfino Android...

Volevo senz'altro perfezionare di più questo ebook ma il tempo come sempre mi è tiranno e metto tutto online così com'è. Mi scuso per i molti errori presenti, spero comunque che questa mia altra fatica sia utile a qualcuno...

ALLEGRO.H dalla A alla Z...

(Programmare videogiochi in linguaggio C è facile)



Dott. Bartolomeo Davide Bertinetto

Altri miei lavori dedicati ad Allegro.h

Ho scritto altri due libri dedicati a questa fenomenale libreria C, oggi nel 2018 possiede ancora un certo fascino retrò. Uno dei libri è disponibile in formato PDF(http://www.bertinettobartolomeodavide.it/programmazione/ALLEGROH/LIBRO/libro.html) e l'altro più completo potete reperirlo su Amazon come Ebook Kindle(http://www.bertinettobartolomeodavide.it/programmazione/ALLEGROH/LIBRO2/libro2.htm). Inoltre è disponibile un abbozzo di un videogame che mi ha sempre appassionato e che volevo riscrivere per la piattaforma Windows ma che poi ho abbandonato: Shadow of the Beast, che nel mio caso si sarebbe chiamato BeastPC. Inoltre esiste una libreria aggiuntiva per allegro.h con comandi 'avanzati' denominata AllegroBBD.h.

Tutto questo materiale ed ancora altro potete trovarlo sulla mia pagina web appositamente dedicata a questo argomento:http://www.bertinettobartolomeodavide.it/programmazione/ALLEGROH/

Un caro Saluto a tutti quelli che hanno seguito i miei lavori per allegro.h negli ultimi 18 anni.

Bertinetto Bartolomeo Davide