UNIVERSITÀ DI PISA

Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea Specialistica in Informatica

Tesi di Laurea Specialistica

ADAPTIVE COMPUTER GAME LEVEL GENERATION

Candidato: Alessio Perrotti

Relatori:

Prof. Antonio Cisternino

Dr. Diego Colombo

Controrelatore:

Prof. Alessio Micheli

Anno accademico 2008-2009

Ringraziamenti

La consuetudine vuole che ogni tesi inizi dai ringraziamenti ma queste righe non vengono scritte per rispettare una prassi formale, bensì per esprimere un sincero grazie a chi mi ha permesso di arrivare a questo punto. Il lavoro che ho svolto come progetto di tesi è frutto di una passione che mi accompagna da molti anni, che mi ha fatto intraprendere questo corso di studi e che spero di ritrovare anche in futuro in ambiti lavorativi.

Il primo grazie è rivolto al Prof. Antonio Cisternino per avermi proposto un progetto così coerente con la mia passione e per avermi supportato lungo il suo svolgimento. Ringrazio anche il Dr. Diego Colombo per aver dato il suo contributo in una fase dello sviluppo del progetto.

Ringrazio infine sinceramente i miei genitori per la pazienza e la fiducia che hanno dimostrato nei miei confronti.

Indice

Capitolo 1
Introduzione

In questa tesi viene presentato lo sviluppo di un modello per la generazione dinamica di contenuti nell'ambito dei videogiochi.

L'impiego di sistemi dinamici è coinvolto in molti settori dell'informatica, con lo scopo principale di ridurre i costi di produzione e sviluppo. Prendendo in esame il campo delle interfacce grafiche si nota come i sistemi dinamici vengano impiegati per adattare i componenti grafici dell'applicazione ai diversi tipi di terminali. In particolare nel caso di contenuti web la topologia degli elementi che compongono una pagina viene modificata in base alla dimensione dello schermo che la visualizza, così come l'aspetto grafico e le funzionalità dell'elemento stesso possono essere riadattate in funzione delle risorse del dispositivo. Data l'enorme espansione e diversificazione dei terminali mobili in grado di accedere a contenuti web è evidente che un'applicazione in grado di erogare dinamicamente i contenuti ha implicato una drastica riduzione dei costi, rispetto ad uno sviluppo specifico per ogni tipo di macchina.

Ritornando nell'ambito dei videogiochi vengono di seguito considerati alcuni aspetti riguardanti lo sviluppo di un gioco privo di procedure per la generazione dinamica e le possibili limitazioni che ne derivano.

Attualmente non sono pochi i casi in cui team di sviluppo, anche molto affermati, devono sospendere la produzione di un titolo o addirittura terminare l'attività a causa di costi di produzione troppo elevati. L'impiego così ampio di risorse economiche dipende spesso dalla necessità di legare lo sviluppo del videogioco e di tutte le sue componenti alla stesura di una storia. Il fatto che attualmente le storie che vengono proposte in un videogioco risultino comparabili a quelle dell'ambito cinematografico, amplifica l'impiego di risorse atte alla creazione di contenuti necessari alla rappresentazione delle vicende narrate. La dipendenza dello sviluppo da eventi ben definiti che scandiscono l'avventura permette di curare in ogni dettaglio la rappresentazione della scena ma implica anche un'enorme elaborazione di contenuti specifici per ogni scena.

Un altro aspetto da considerare nel caso di uno sviluppo "story driven"riguarda la longevità del videogioco, non prettamente in termini di durata dell'avventura ma soprattutto in termini del fattore di riuso di cui dispone il prodotto. Le considerazioni riguardo questa caratteristica possono essere soggettive ma si può assumere che nella norma la linearità delle vicende è inversamente proporzionale alla probabilità che il giocatore inizi una nuova partita dopo la fine del gioco. Astraendo la struttura generale, spesso il giocatore deve procedere lungo un percorso prefissato, dove ad ogni attraversamento di uno stesso punto implica il dover riaffrontare la stessa situazione invariata.

Questo aspetto ci porta a considerare anche il concetto di adattabilità legata al grado di sfida che il videogioco propone. Tipicamente lungo il percorso sono posti oggetti o nemici controllati dal programma che ostacolano il giocatore. La progressione lungo il gioco è accompagnata da una progressione del livello di difficoltà, che però dipende unicamente dalla posizione del protagonista lungo la scalata verso la meta finale.

Queste considerazioni forniscono il presupposto per chiederci se l'adozione di un approccio dinamico nella generazione dei livelli di un videogioco possa risolvere le problematiche e le limitazioni esposte.

1.1 La Grammatica Generativa

L'idea alla base dello sviluppo di questo progetto è stata quella di implementare un sistema capace di rendere il gioco diverso ad ogni attraversamento del percorso. Partendo quindi da strutture atomiche il percorso che il giocatore affronta viene creato dinamicamente e gli elementi che lo compongono vengono adattati in base alle scelte ed alle capacità del giocatore stesso.

Uno degli obiettivi posti è stato anche quello di rendere la difficoltà elemento attivo e dinamico, dipendente dal tipo di approccio al gioco del giocatore stesso e non un parametro impostato a priori.

Partendo quindi dall'implementazione delle unità atomiche del gioco è stato costruito un sistema in grado di rielaborare ed assemblare tali unità in moduli strutturati ed adattabili al tipo di interazione dell'utente.

Concettualmente questa soluzione implementa, sotto molti aspetti, la teoria definita dalle grammatiche generative. In questo contesto la grammatica esprime di fatto il gioco stesso, nella forma di un percorso attraverso il quale il giocatore manovra il proprio avatar. A partire quindi da un livello iniziale ne viene delineata una successione, che varia in funzione dei vincoli definiti nel sistema, ovvero le regole di produzione applicabili. A sua volta un livello è una composizione di elementi atomici, come ad esempio nemici o trappole, che definiscono l’alfabeto della grammatica.

Già da questa premessa si può assumere che un approccio implementativo, improntato su procedure di generazione iterativa, può produrre ambienti di gioco dinamici e facilmente espandibili.

1.2 Contesto

La prima meta prefissata nella creazione di questo progetto ha riguardato lo studio e l'analisi di quale tipologia di videogioco sviluppare.

Il 1981 vede la nascita di nuovo genere videoludico : il platform game letteralmente, "gioco a piattaforme", così definito perché il protagonista controllato dal giocatore deve affrontare una serie di peripezie per raggiungere una meta specifica, spostandosi da una piattaforma all'altra sullo schermo. Spesso lungo il percorso sono posti oggetti o nemici controllati dal programma che ostacolano il protagonista nel raggiungere la meta. La prima generazione era caratterizzata dalla presenza di un singolo quadro visualizzato a schermo, all'interno del quale il personaggio si poteva muovere; ogni quadro definiva una sessione di gioco completa, che una volta completata si aggiornava in un nuovo quadro indipendente dal precedente. In seguito venne introdotta una struttura definita a scorrimento orizzontale, dove il personaggio principale si muoveva in una porzione prefissata dello schermo, mentre il percorso di gioco scorreva sotto i suoi piedi, dando così un senso di continuità. L'elemento comune a tutti i platform è sempre stato il salto, azione principale di tutta l'economia di gioco. Tuttavia nel corso degli anni sono stati integrati elementi caratteristici di altri generi : da giochi puramente platform, come Super Mario Bros, sono state introdotte meccaniche legate alla capacità di sparare come in Contra (1987, Konami), o elementi classici dei giochi di ruolo come in Castlevania : Symphony Of The Night (1986, Konami) o una struttura tipica delle avventure come in FlashBack (1992, Delphine Software).

Partendo dal presupposto di sviluppare un'applicazione bidimensionale per poter maggiormente concentrare il lavoro sulla logica del gioco è stato quindi deciso di progettare e definire l'ambiente di lavoro nel contesto di un Platform Game 2D.

Il design e le meccaniche di base su cui costruire l'intero progetto non sono state create partendo da un concept nuovo ma è stato deciso di riprendere e rielaborare la struttura di un gioco esistente. La creazione di una struttura totalmente nuova avrebbe implicato anche la

creazione di tutti gli elementi grafici dell'architettura di gioco (attività tipicamente svolte a livello di gamedesign) : personaggi, strutture di interazione, sfondi dei livelli etc.

Il gioco dal quale prendere spunto per avere quantomeno accesso a parte degli assets necessari alla rappresentazione visiva della logica sottostante è stato un gioco uscito nel 1997 su piattaforma Playstation e Windows, Oddworld: Abe's Oddysee. Sviluppato da un team californiano, gli Oddworld Inhabitants, fondato nel 1994 da Sherry McKenna and Lorne Lanning, veterani di effetti speciali ed animazioni.

Il gioco ebbe la capacità di ridare vita al concetto di videogioco 2D in un periodo dove nel mercato dei videogiochi si stava ormai affermando il 3D su tutti i generi.

La base che sta alla meccanica di gioco è, oltre al farsi strada lungo i vari livelli correndo e saltando su varie piattaforme, il portare in salvo altri personaggi guidandoli lungo il percorso. Altra caratteristica è quella che il personaggio principale non può contare sull'uso di armi per sconfiggere i nemici ma deve avvalersi di varie strategie come ad esempio sfruttare zone in ombra per non farsi individuare o ricorrere al potere di potersi impadronire del corpo del nemico di turno.

1.3 Struttura del Documento

Nel capitolo introduttivo sono stati presentati gli obiettivi principali definiti nella fase di progettazione dell'applicazione e il contesto generale nel quale sviluppare il lavoro.

Di seguito verranno presentati altri 4 capitoli :

- **Capitolo 2**. In questo capitolo viene fatta una panoramica riguardo le tecnologie e gli strumenti utilizzati durante tutte le fasi di sviluppo del progetto.
- **Capitolo 3**. In questa sezione viene descritto lo sviluppo dei componenti necessari per la creazione di un ambiente di lavoro su cui improntare la fase successiva di generazione dei livelli.
- **Capitolo 4**. Questo capitolo presenta il modello concettuale della generazione dei livelli di gioco e descrive le scelte implementative adottate.
- **Capitolo 5**. A conclusione del documento di tesi vengono esposte alcune considerazioni sui risultati ottenuti e vengono delineati i possibili sviluppi futuri dell'applicazione.

Nell'esposizione del documento sono stati inseriti alcuni riferimenti al codice sorgente, riportando solo le istruzioni maggiormente significative per il contesto. Nell'appendice sono state inserite le principali classi definite nel progetto.

Capitolo 2

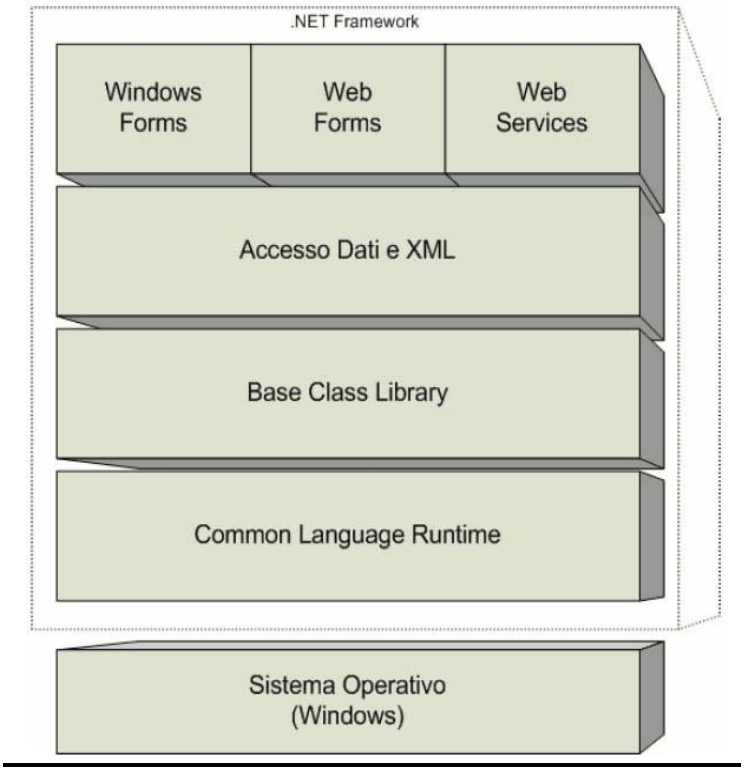
Strumenti e Tecnologie di riferimento

Prima di passare alla descrizione dello sviluppo del progetto viene esposta una panoramica sulle tecnologie alla base degli ambienti di sviluppo coinvolti nella realizzazione del progetto. In particolare vengono descritti i componenti principali dell'architettura del .Net Framework e le principali caratteristiche del nuovo ambiente di sviluppo XNA, iniziando con una considerazione di quali sono state le basi su cui questo nuovo strumento ha poggiato nel periodo del suo rilascio. Si prosegue dunque con una descrizione delle caratteristiche della sua architettura.

Vengono infine presentate altre due applicazioni, sviluppate appositamente come strumenti per il supporto in varie fasi della creazione del progetto principale

2.1 Architettura del .NET Framework

.NET è una piattaforma per lo sviluppo e l'esecuzione di varie tipologie di applicazioni, con il supporto a diversi linguaggi di programmazione. Il modello di questa architettura consente di utilizzare un paradigma di programmazione unificato, indipendente dal linguaggio utilizzato o dal tipo di applicazione sviluppata.



Il cuore del framework è il CLR (Common Language Runtime), o CLI (Common Language Infrastructure) secondo lo standard ECMA. Il CLR/CLI non fornisce solo supporto a runtime alle applicazioni *managed* (applicazioni direttamente gestite dal CLI) ma anche in fase di sviluppo, in particolare mettendo a disposizione una serie di servizi aggiuntivi.

- **A livello runtime** :
  - Gestisce la memoria (garbage collection) istanziando e deallocando oggetti dall'heap.
  - Gestisce il ciclo di vita di processi e threads.

- Interviene sulla sicurezza con un sistema dinamico di controllo degli overflow anche grazie al sistema di eccezioni e della tipizzazione forte.
- Interviene dinamicamente sulle dipendenze.

- **In fase di sviluppo** :
  - Gli automatismi introdotti, come ad esempio la gestione della memoria, permettono di scrivere meno codice ma più sicuro e in modo più semplice.
  - La tipizzazione forte, la gestione del ciclo di vita, l'introduzione del sistema ad eventi, il binding dinamico e i sistemi di reflection permettono di scrivere meno codice ed in modo più semplice.

L'accesso alle funzionalità del CLR/CLI avviene non in modo diretto ma tramite il BCL (Base Class Library), un sistema di librerie che espone operazioni di vario genere (I/O, gestione stringhe, connettività, etc.).

Nello strato successivo dell'architettura troviamo le classi per la gestione dell'accesso ai database e la manipolazione di file XML.

Il livello più alto dell'architettura mette a disposizione del programmatore strumenti per la realizzazione di componenti grafici in ambiente desktop e web.

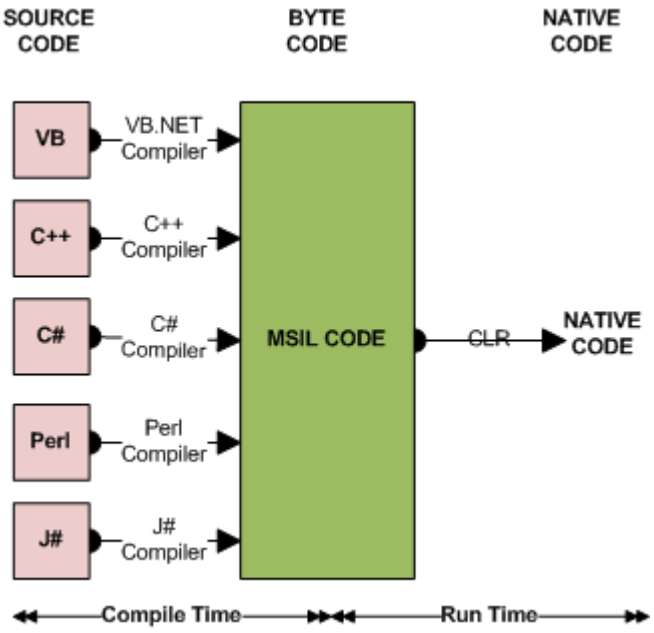
2.1.1 Unità del CLR

Il CLR/CLI coinvolge fondamentalmente 5 componenti :

1. **CIL/IL**
2. **JIT**
3. **CTS**
4. **CLS**
5. **VES**

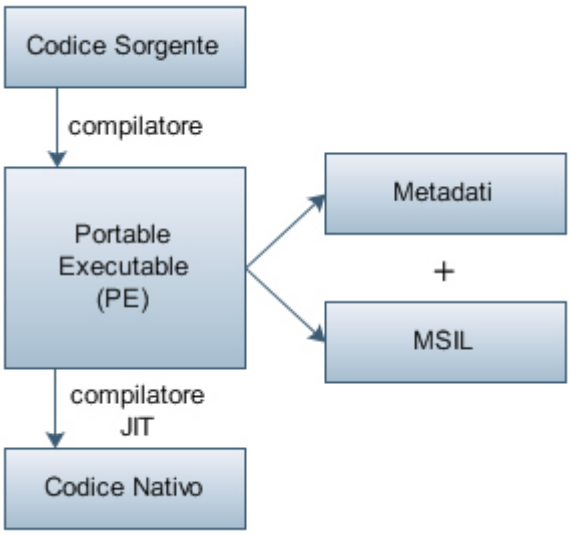
- **CIL/IL**

La caratteristica principale del CLR è quella di poter gestire in maniera unificata diversi linguaggi (C#, C++, J#, VB). La gestione non avviene infatti sul codice nativo ma su un codice intermedio CIL (Common Intermediate Language) o IL (Intermediate Language), generato dal compilatore di ogni linguaggio, che in seguito verrà tradotto ed assemblato in un eseguibile.



Nello specifico vengono prodotti Moduli Managed, definiti Portable Executable, che comprendono sia il codice IL generato, sia metadati. Il CLR accede e gestisce

direttamente gli Assembly, entità composte da un modulo e un insieme di risorse esterne (come ad esempio i file di configurazione).



- **JIT Compiler**

Come già accennato il codice IL non è direttamente eseguibile. Quindi è compito del compilatore JIT (Just In Time) tradurre i moduli managed in codice nativo eseguibile dalla macchina.

La traduzione del codice viene fatta *On Demand*, in modo così da garantire un risparmio di tempo e memoria in fase di avvio del programma. Il codice compilato viene mantenuto in memoria per eventuali chiamate successive. Il codice può quindi essere compilato indipendentemente dalla piattaforma sottostante, assumendo che venga implementato un CLR specifico per ogni piattaforma.

- **CTS e CLS**

Uno dei vantaggi introdotti dal .NET Framework risiede nella interoperabilità dei linguaggi. Il CLR definisce quindi un sistema per la definizione di tipi comuni, il CTS (Common Type System). In questo modo è possibile scrivere librerie in un solo linguaggio, evitando fasi di porting in altri linguaggi.

Una libreria o un nuovo linguaggio devono aderire alle specifiche delineate dal CLS (Common Language Specification) per poter essere interoperabili.

- **VES**

Il ruolo fondamentale del VES (Virtual Execution System) è quello di caricare, linkare ed eseguire i file in formato PE (Portable Executable), astrendo dal livello sottostante della macchina fisica. Può essere paragonato alla JVM (Java Virtual Machine) nell'ambito del linguaggio Java. Le attività che svolge sono legate alle informazioni contenute nei metadati e includono servizi di supporto a tempo di esecuzione, come ad esempio la gestione della memoria, il debugging o la gestione delle eccezioni.

2.2 Le origini di XNA

Microsoft entra nel mercato dei videogiochi per console nel Novembre del 2001 con il sistema Xbox. Riesce a ritagliarsi una propria porzione di utenti in un periodo dove i colossi Giapponesi quali Nintendo, Sony e Sega hanno ormai da tempo investito le proprie risorse nello sviluppo di sistemi hardware e software in ambito videoludico.

Il sistema Xbox è una console che a differenza delle concorrenti è fortemente predisposta alla fornitura di un servizio online che permetta ai giocatori di usufruire delle funzionalità multiplayer dei titoli sviluppati per questa piattaforma.

La rete virtuale che Microsoft lancia per espandere le funzionalità locali proprie di tutte le console dell'epoca si chiama "Live"; il Live è un servizio che permette ai giocatori, qualora il gioco ne sia provvisto, di usufruire anche delle modalità online del titolo, tipicamente sfidare altri giocatori in ambienti e con meccaniche di gioco derivanti dalla modalità single player.

Le potenzialità del servizio Live sono tante ma Microsoft decide di sfruttarle per tempi più maturi, ovvero nel 2005 con il lancio della nuova console Xbox360. La nuova console da accesso fin da subito alla nuova versione del servizio Live che oltre a mantenere tutte le caratteristiche di base della precedente versione diventa una vera e propria vetrina multimediale. Un utente ha quindi accesso a contenuti di vario genere, demo e video per provare in anteprima i giochi in uscita, film, musica e giochi interamente scaricabili sotto previo pagamento. Questi ultimi sono caratterizzati da un prezzo piuttosto contenuto anche se non completi come quelli acquistabili in negozio.

Il crescente successo della piattaforma Live che ogni anno incrementa considerevolmente il numero di utenti e il forte interesse della comunità verso i giochi Live arcade sono probabilmente una buona base di partenza per lanciare una piattaforma di sviluppo di videogiochi indirizzata anche a piccoli sviluppatori indipendenti.

2.3 **XNA is Not Acronyme**

Questo è il nome del nuovo ambiente di sviluppo orientato alla creazione di giochi. Il cuore di questa piattaforma è il .NET Framework che estende le proprie funzionalità di base a librerie e tool di sviluppo specifici per l'ambito di videogiochi. In particolare la prima versione espone le librerie grafiche DirectX 9.0c e il supporto al model shader nelle versioni 1.1 e 2.0.

XNA permette di sviluppare giochi per sistemi Windows e per la console Xbox360. Lo sviluppo e la pubblicazione dei giochi in ambiente Windows è totalmente gratuita, mentre lo sviluppo per xbox360 ha un costo di 99 $ all'anno.

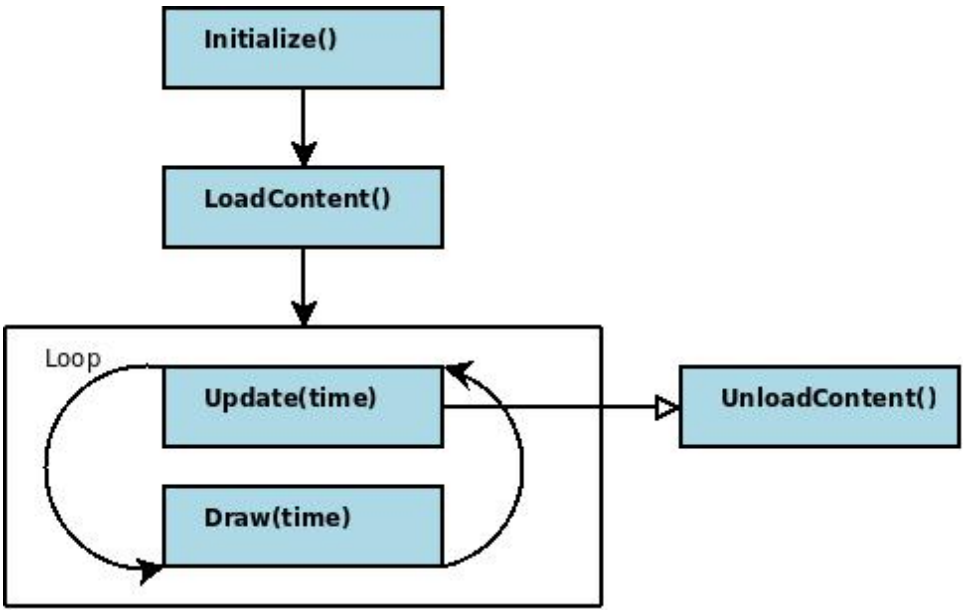
Anche se forse non si è ancora formato un mercato ampio come quello dei giochi sviluppati da professionisti, sicuramente con XNA tutti hanno avuto accesso ad uno strumento per cimentarsi nello sviluppo di applicazioni orientate al videogioco. Uno dei pregi di questo strumento è quello di essere facilmente accessibile e di poter usufruire di una comunità vasta di sviluppatori. La crescente popolarità di questa piattaforma ha contribuito di fatto a renderla uno standard nello sviluppo di giochi a livello amatoriale e accademico.

XNA è un insieme di tool sviluppati da Microsoft con lo scopo di semplificare il processo di creazione di Videogame (o applicazioni interattive). E' basato su .NET e per tanto sfrutta uno dei linguaggi compatibili (in particolare c#) e si appoggia a DirectX per la gestione della grafica. XNA implementa un framework molto completo, che consente di interagire facilmente con Audio, Input e Device grafico. Uno dei punti di forza di questo sistema è l'utilizzo di una content pipeline utilizzata per caricare con facilità materiali come modelli 3d e texture, e renderli raggiungibili all'interno del programma senza dover implementare layer o modelli di importazione complessi.

Un altro elemento molto interessante è l'utilizzo dei GameComponents, un ottimo modo per incapsulare informazioni implementando un'interfaccia software comune a tutti i componenti.

2.3.1 Workflow di XNA

Il workflow di XNA è basato su un loop continuo. In generale il workflow presentato da XNA è comunque "standard" per quello che concerne il game development.



- **Initialize.**

  In questo punto vengono istanziate le variabili e le classi principali utilizzate in molte altre funzioni.

- **LoadContent.**

  Questa funzione è dedicata al caricamento di tutti gli elementi che sfrutteranno la content pipeline. Quindi vengono caricati suoni, texture e modelli 3d per esempio.

- **UnloadContent.**

  Richiamata quando è necessario eliminare tutti i contenuti caricati tramite la LoadContent().

- **Update.**

  Qui entriamo nel game loop, questa funzione verrà chiamata ripetutamente per tutta la durata del programma ed è il punto dedicato alle operazioni cosidette di business, o meglio dove verranno inserite le modifiche ai dati del programma.

  Un semplice esempio: nel caso si voglia far avanzare di un passo il nostro personaggio verrà premuto il tasto corrispondente e proprio nell'Update verrà effettuato il controllo se il pulsante risulta effettivamente nello stato premuto e di conseguenza modificato il valore di una variabile corrispondente allo stato del pulsante.

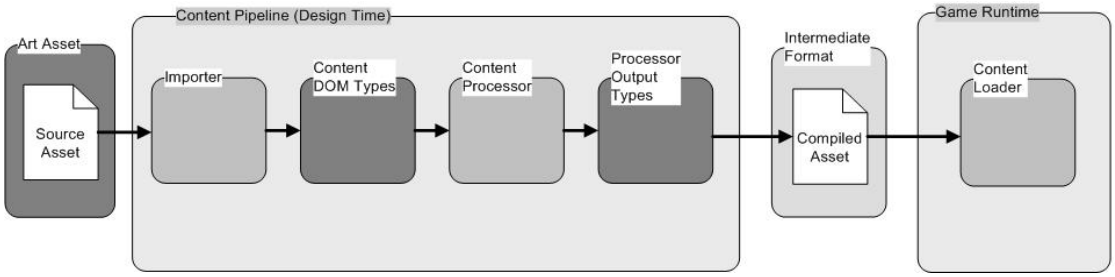
- **Draw.**

  Questa funzione si occupa di gestire l'output, ciò che deve essere disegnato e quando deve essere disegnato. Riprendendo l'esempio di prima qui viene effettivamente disegnato il personaggio aggiornando l'animazione relativa al passo in avanti.

2.3.2 La Content Pipeline

Attraverso questo strumento è possibile caricare e accedere a molti tipi diversi di risorse.

La **Content Pipeline** trasforma gli assets importati nelle risorse di progetto in istanze di oggetti fruibili nell'esecuzione dell'applicazione.



- **Importer - Content DOM Types**

  Ad ogni tipo di risorsa viene associato un Importer ed un Processor specifici per il tipo di formato della risorsa. Una risorsa viene convertita dall'Importer in un oggetto analizzando il rispettivo Content Document Object Model (DOM).

  Nel caso di assets con formato non elaborabile dalla Content Pipeline di Xna è possible sviluppare un Custom Importer o importarne uno di terze parti.

- **Content Processor**

  Come per l'Importer ad ogni assets è possible associare un Processor specifico. Un Content Processor compila l'oggetto passato dall'Importer in un codice che può essere usato all'interno del codice Xna.

- **Content Compiler**

  Il codice prodotto dal Processor viene serializzato in un formato binario compatto (formato intermedio) dal Content Compiler. A questo punto l'asset è stato processato dalla Content Pipeline in un formato direttamente gestibile a tempo di esecuzione.

- **Content Loader**

  Tramite la chiamata ContentManager.Load l'asset processato nel formato intermedio viene caricato nella memoria del gioco instanziandolo in una variabile di tipo opportuno, nel caso in esame una Texture2D.

```
Texture2D Sprite = Content.Load<Texture2D>("Sprite_Sheet");
```

2.3.3 XNA e Directx

Le Directx (DX) sono un'avanzata suite di APIs creata per lo sviluppo di applicazioni in ambito multimediale. Le DX forniscono un set di strumenti standardizzati per l'accesso e l'interfacciamento all'hardware sottostante, permettendo così allo sviluppatore di non dover scrivere codice di basso livello specifico per ogni componente HW. L'architettura alla base delle DX è stata disegnata per dare pieno supporto al linguaggio C++.

Per meglio integrare le funzionalità di questa suite nella piattaforma .NET, Microsoft ha pubblicato una nuova versione di API, denominate per l'appunto "Managed DirectX"(MDX), che nell'architettura del framework si posizionano ad un livello più alto delle DX. La completa integrazione nella piattaforma .NET ha fornito, oltre ai vantaggi derivanti dall'uso del CLR un tipo di approccio ancora a più alto livello di astrazione dall'hardware.

XNA ha sostituito completamente le MDX nella gestione dell'accesso alle funzionalità di basso livello fornite dall'hardware.

2.3.4 Le Release di XNA

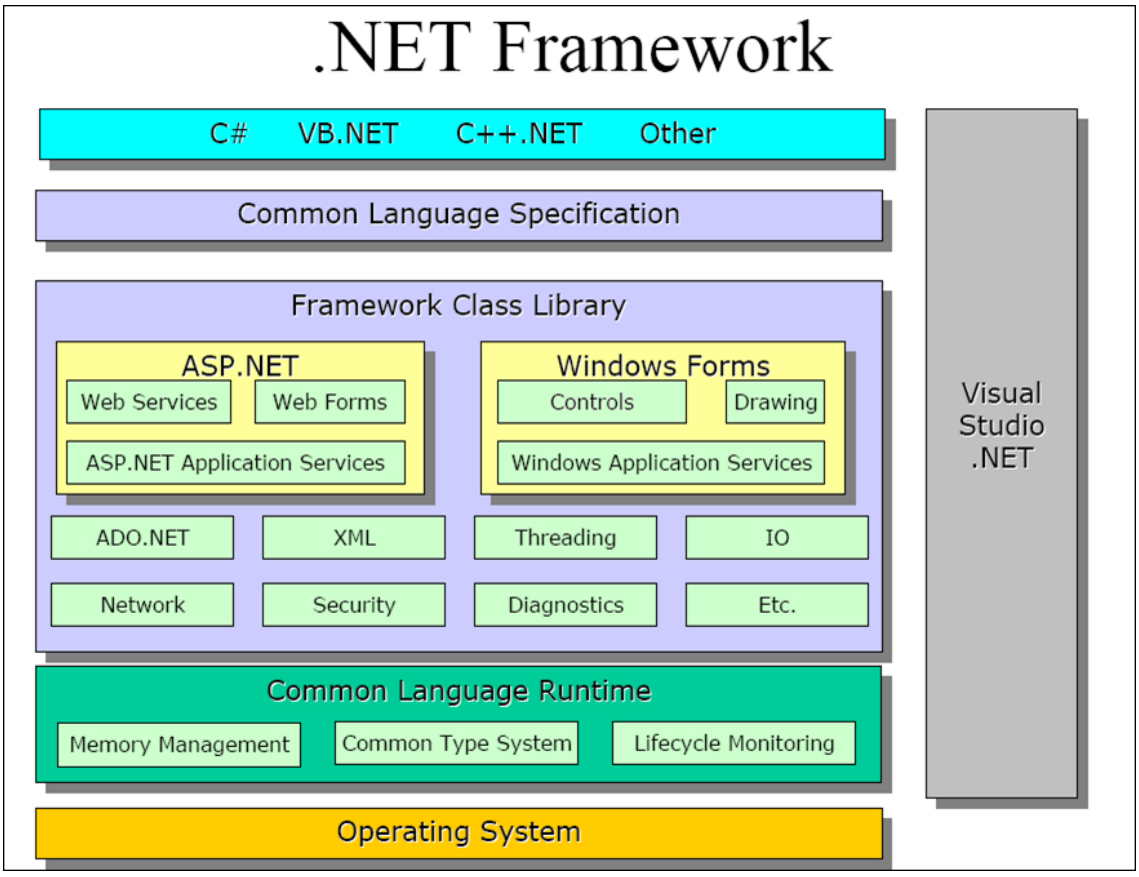
Dal suo debutto, XNA si è evoluto attraverso il rilascio di 4 versioni.

1. **XNA Game Studio Express.** Rilasciato l'11 Dicembre 2006, forniva il primo set di tool per lo sviluppo di videogiochi, supportato dalla piattaforma Microsoft Visual C# Express Edition e basato sulla versione 2.0 del .NET Framework.
2. **XNA Game Studio 2.0.** Rilasciato il 13 Dicembre 2007. Con questa versione viene introdotto il supporto a Visual Studio 2005. Vengono estese le funzionalità con l'introduzione di un set di API per l'accesso e la gestione di servizi legati a Xbox Live.
3. **XNA Game Studio 3.0.** Il rilascio di questa versione avviene il 30 Ottobre 2008. In questa versione troviamo il pieno supporto a C# 3.0 e la possibilità di sviluppare applicazioni per dispositivi Zune. Viene ampliato l'accesso alle funzionalità di Xbox Live, introducendo la possibilità di creare giochi multiplayer, comunicazione tra i giocatori tramite il sistema di inviti in tempo reale. Molto utile la possibilità di sviluppare giochi closs-platform eseguibili su tutte le piattaforme.
4. **XNA Game Studio 3.1.** Rilasciata l'11 Giugno del 2009, introduce la gestione di file video ed amplia le API dedicate alle risorse audio.

Il 9 Marzo 2010 è stata annunciata la versione 4.0 di XNA, che come principali aggiornamenti apporta il pieno supporto alle funzionalità 3D dei dispositivi Windows Phone 7 e l'integrazione in Visual Studio 2010. In questo progetto è stato usato XNA 3.0 in ambiente Visual Studio 2008. L'applicazione è stata sviluppata per piattaforma Windows.

2.4 WINDOWS FORMS

Windows Forms è il nome di un API per la gestione di interfacce grafiche, inclusa come parte del .NET Framework. Riferendoci all'architettura del framework, questo modulo si pone nel livello che definisce il sistema di librerie.

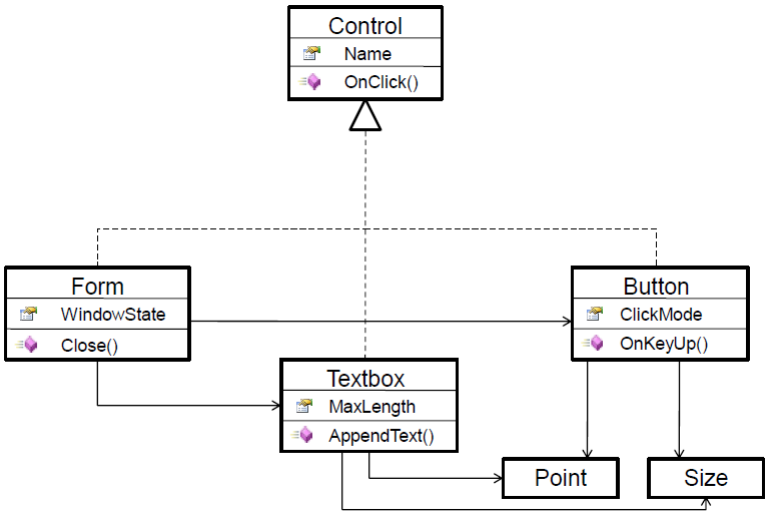


Con questa classe viene fornito l'accesso, la manipolazione e la gestione di componenti definiti dal sistema di interfaccia di Windows.

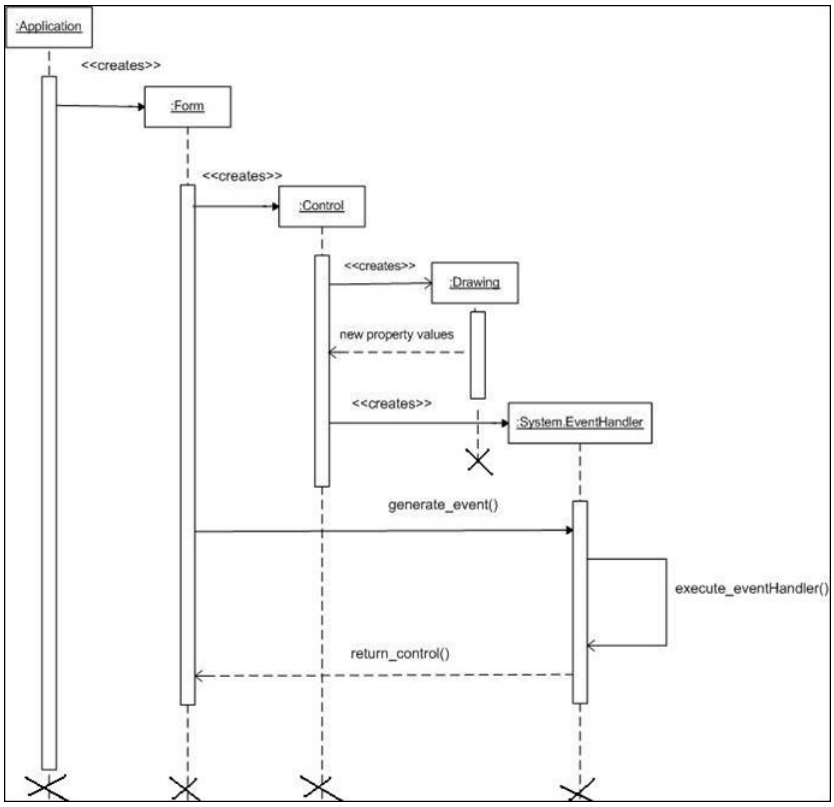
Le applicazioni sviluppate in questo ambito rispettano un parafigma di programmazione guidato da eventi (event-driven). In questo ambito il flusso del programma è determinato dal verificarsi di determinati eventi, spesso generati da azioni dell'utente.

Questo paradigma definisce programmi, che sotto il profilo architetturale, sono divisi in due segmenti, che determinano il ciclo di vita principale dell'applicazione : da una parte troviamo una fase di event-detection, ovvero viene determinato se un evento è stato generato. Il secondo segmento provvede alla gestione dell'evento in funzione della tipologia specifica.

Una Form rappresenta il canvas dell'applicazione, all'interno del quale vengono inseriti i Controlli che implementano gli elementi grafici definiti in Windows. Ogni controllo espone un'interfaccia per l'implementazione di un set di proprietà e metodi necessari alla gestione del tipo di azione che l'utente effettua su un elemento.



L'individuazione del tipo di evento generato è determinata da una subroutine, l'Event Handler, che quindi si occupa anche di chiamare il metodo delegato alla gestione dell'evento associato.



2.5 XML

L'XML, acronimo di *eXtensible Markup Language*, ovvero "Linguaggio di marcatura estensibile" è un metalinguaggio creato e gestito dal *World Wide Web Consortium* (W3C). Viene definito come meta-linguaggio di markup, un linguaggio che permette di definire altri linguaggi di markup. A differenza di HTML, XML non ha tag predefiniti e non serve per definire pagine Web.

Fondamentalmente un documento XML viene utilizzato in qualità di contenitore di informazioni.

Di seguito una breve descrizione dei principali strumenti coinvolti nell'utilizzo di XML :

- ***DTD* (*Document Type Definition*)**

  *U*n documento attraverso cui si specificano le caratteristiche strutturali di un documento XML attraverso una serie di "regole grammaticali". In particolare definisce l'insieme degli elementi del documento XML, le relazioni gerarchiche tra gli elementi, l'ordine di apparizione nel documento XML e quali elementi e quali attributi sono opzionali o meno.
- ***XML Schema***

  Come la DTD, serve a definire la struttura di un documento XML. Espone caratteristiche più nuove ed avanzate. La sua sigla è XSD, acronimo di *XML Schema Definition.*
- ***XLink***

  Serve a collegare in modo completo due documenti XML; al contrario dei classici collegamenti ipertestuali di HTML, XLink permette di creare link multidirezionali e semanticamente avanzati.
- ***XSL* (*eXtensible Stylesheet Language*)**

  Llinguaggio con cui si descrive il foglio di stile di un documento XML.
- ***XSLT* (XSL *Trasformations*)**

  Versione estesa dell'XSL. L'obiettivo principale per cui l'XSLT è stato creato è

rendere possibile la trasformazione di un documento XML in un altro documento. È possibile anche aggiungere al documento trasformato elementi completamente nuovi o non prendere in considerazione determinati elementi del documento origine, riordinare gli elementi, fare elaborazioni in base al risultato di determinate condizioni, ecc. In termini strutturali, XSLT specifica la trasformazione di un albero di nodi in un altro albero di nodi.

- ***XPath***

  Linguaggio con cui è possibile individuare porzioni di un documento XML e sta alla base di altri strumenti per l'XML come *XQuery*.

2.6 Progetti di supporto

XNA si è rivelato uno strumento molto versatile e accessibile, ma sono stati comunque sviluppati altri due programmi per il supporto in alcune fasi di sviluppo del gioco.

Il primo programma SpriteSheetMaker (SSM) è stato creato per le fasi di manipolazione di immagini da importare nel progetto XNA.

Il secondo programma LevelMaker (LM) sviluppato è sostanzialmente un editor dei livelli che compongono il gioco.

Entrambe le applicazioni sono state sviluppate parallelamente al progetto di tesi, introducendo nuove funzionalità ogni volta che il progetto principale ne richiedeva il supporto. SSM è stato inizialmente sviluppato per convertire una GIF in un'immagine contenente tutti i suoi frame; in seguito è stato ampliato con funzionalità in grado di effettuare operazione di “taglia e incolla”sulle immagini generate o di navigare la struttura dei pixel dell'immagine ed operare modifiche alla loro composizione.

LM è stato sviluppato al fine di permettere una modellazione dei livelli da importare nell'applicazione XNA. Ogni volta che un nuovo elemento di gioco viene sviluppato e testato in XNA, viene così ampliato LM per permettere di inserire il nuovo elemento nelle fasi di editing del livello.

SSM e LM sono stati sviluppati su piattaforma Visual Studio 2008 con il supporto della libreria Windows Forms.

Capitolo 3

Implementazione componenti di gioco

Lo sviluppo di un modello generativo capace di produrre ambienti di gioco durante l'esecuzione del gioco stesso necessita di una base di componenti atomici sulla quale poter applicare regole di produzione, in grado di definire ad ogni passo generativo diverse scelte di combinazione degli elementi.

In questo capitolo vengono quindi descritte le fasi di implementazione degli atomi di gioco.

Ogni elemento è stato sviluppato in un modulo indipendente al fine di garantire nella fase generativa una maggiore libertà di espressione e di conseguenza una più ampia scelta di combinazioni.

Partendo dal concept del gioco originale Oddworld, sono state ricercate le caratteristiche portanti della struttura globale. Ogni modulo è stato estrapolato dal contesto originale e ricostruito come componente atomico nell'ambiente del progetto di tesi. In funzione della

struttura o dell’elemento da ricostruire è stata attuata una specifica strategia di reverse engineering :

- Creazione del personaggio principale
- Implementazione del sistema di controllo del personaggio
- Creazione degli ambienti di gioco
- Gestione dell'interazione con le strutture dell'ambiente
- Creazione dei personaggi secondari non controllabili
- Sviluppo delle intelligenze artificiali da assegnare ai personaggi secondari
- Sviluppo degli oggetti interattivi

Sulla base di questi moduli è stato infine sviluppato il sistema di generazione dei livelli di gioco, descritto nel capitolo successivo.

3.1 Primi Passi

L'intento della prima fase del progetto era quello di prendere confidenza con la piattaforma XNA, quindi cercare di sviluppare il modulo più basilare di tutto il gioco: far muovere i primi passi al personaggio principale (Abe).

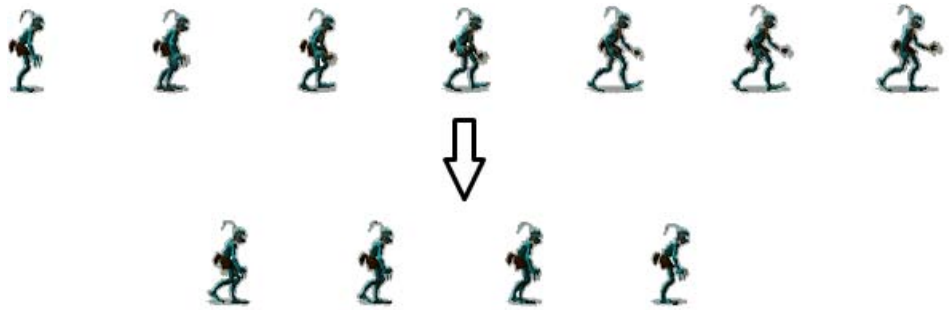
La risorsa necessaria per far camminare un personaggio su di un piano risiede unicamente nella successione dei frame che compongono l'animazione che definisce la camminata. Dato che la versione originale del gioco è composta da file con formato proprietario ".lvl" non è stato possibile attingere liberamente alle risorse del gioco. Tramite un sito di appassionati delle opere degli Oddworld Inhabithants è stato recuperato un programma amatoriale la cui funzione sembrava essere proprio quella di poter decodificare i file proprietari in formati conosciuti in modo così da avere accesso diretto a tutte le risorse artistiche del gioco. In realtà il programma si è rivelato efficace solo nel recuperare i background dei vari livelli di gioco e poterli sostituire con immagini arbitrarie. Sfruttando però la funzione di sostituzione degli sfondi si è intrapresa la strada di acquisire le movenze del personaggio su sfondo bianco, da gioco avviato.

Quindi nello specifico le operazioni eseguite sono state :

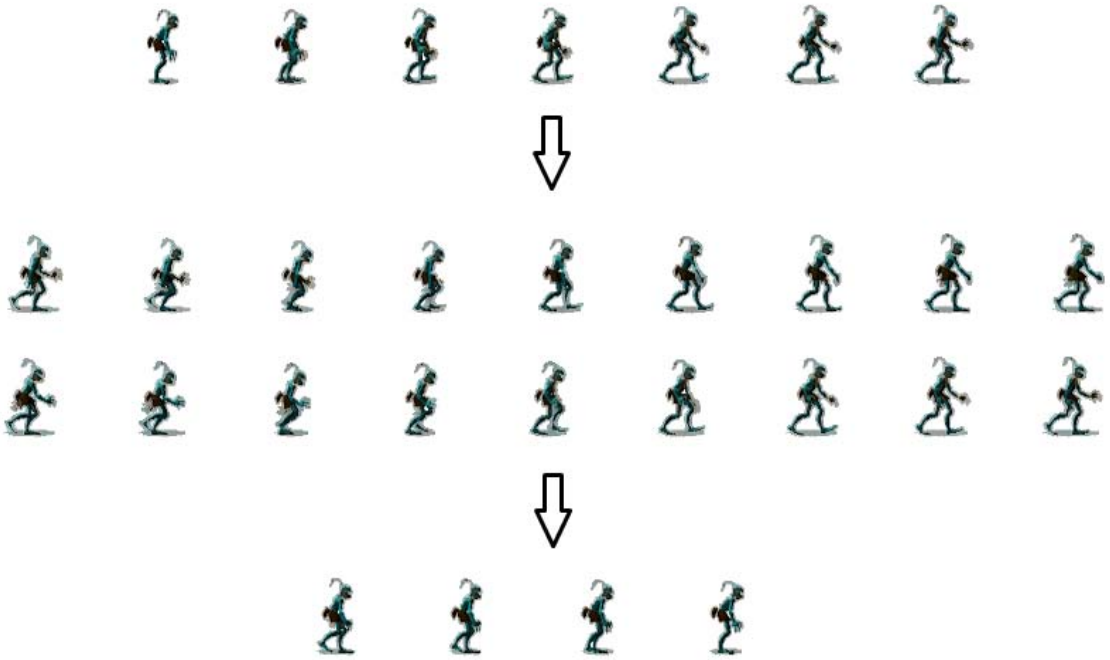
- Estrarre da un file .lvl tutti i livelli contenuti,anch'essi di formato proprietario .cam.
- Scegliere un livello dove potersi muovere liberamente per effettuare la registrazione e sostituire lo sfondo originale con una bitmap totalmente bianca di pari dimensione.
- Far eseguire al personaggio l'animazione necessaria mentre il l'applicazione Camtasia (sviluppata da TechSmith) acquisisce ogni frame.
- Tramite l'editor di Camtasia tagliare i frame in eccesso della registrazione.
- Esportare l'animazione finale come Gif.

Gli ultimi due punti richiedono un maggiore approfondimento, in particolare la fase di editing della registrazione grezza viene effettuata in due fasi. In una prima fase devono essere tagliate tutte le parti evidentemente in eccesso o i frame adiacenti e identici causati dalla non perfetta

sincronia tra il framerate del gioco e la frequenza di acquisizione di Camtasia. Una volta che l'animazione è stata ripulita nella seconda fase avviene l'analisi di come è composta l'animazione in esame. Ritornando al caso dell'animazione della camminata di Abe si nota che a seconda di quanto a lungo il pulsante per camminare viene premuto l'animazione si compone di diverse successioni di frames. Si nota che in questo caso ad esempio se il pulsante viene premuto per un tempo relativamente piccolo Abe effettua un singolo passo in avanti e si riporta in una posizione stazionaria.



Nel caso di pressione prolungata Abe esegue un'animazione che condivide la prima parte del passo singolo ma poi prosegue alternando altre due animazioni che si ripetono in ciclo dando il senso della camminata continua.



Ogni modulo dell'animazione viene salvato in un file di tipo Gif che dovrà essere srotolato in una PNG contenente la successione dei frame e applicata una maschera di trasparenza al colore di fondo bianco.

A questo scopo è stato sviluppato appositamente il programma SpriteSheetMaker. L'operazione di rendere trasparente lo sfondo bianco ha richiesto un'operazione aggiuntiva in quanto lo sfondo si è rivelato essere non totalmente bianco. Evidentemente il gioco originale applica un filtro grafico a tutta la scena perturbando la composizione cromatica di ogni pixel dello sfondo, che nel caso di sfondo bianco risulta un alternarsi di righe di varie tonalità di grigio chiaro. Quindi caricando una gif in SpriteSheetMaker ne vengono estratti i frames che la compongono, si procede quindi nell'individuare i pixel che appartengono allo sfondo e non al soggetto e nel settarli al colore bianco. Viene infine applicata la trasparenza e salvata l'immagine in un file PNG da poter importare nel progetto XNA.

3.2 Sistema di Controllo

Quindi la Texture2D è l'immagine che contiene tutti i frame dell'animazione da eseguire. Nel caso in esame l'animazione che viene trattata è quella relativa al singolo passo in avanti, composta da due moduli: PASS1 e STOP.


```
Texture2D PASS1 = Content.Load<Texture2D>("PASS1 ");

Texture2D STOP = Content.Load<Texture2D>("STOP ");
```

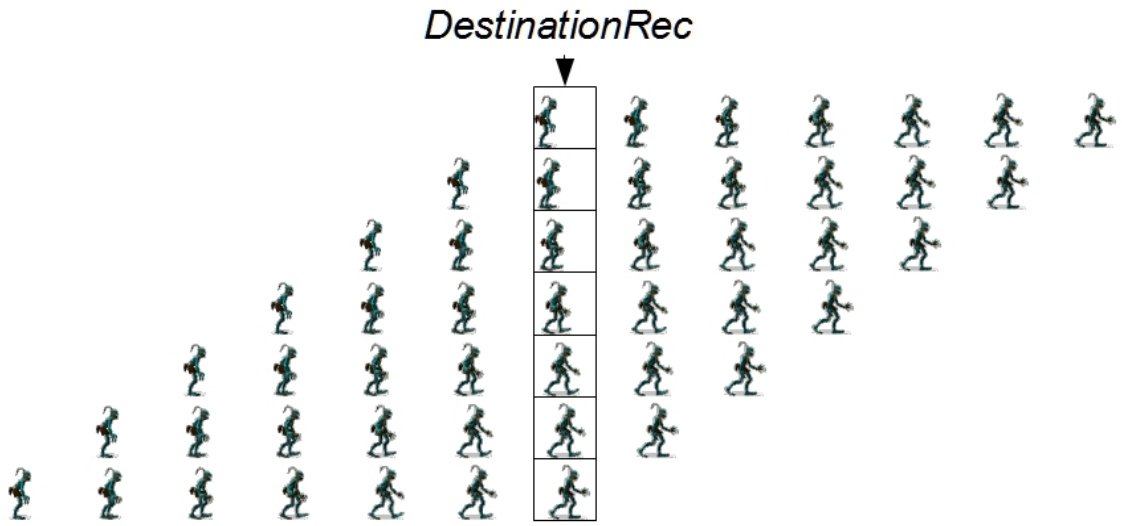
Solitamente ogni frame di una animazione contiene il soggetto da animare centrato rispetto al rettangolo che contiene il frame stesso e ad ogni aggiornamento di frame viene applicata una trasformazione spaziale (nel caso in esame una traslazione verso destra) al fine di rendere coerente l'animazione con uno spostamento nello spazio che essa rappresenta. Nello sviluppo del gioco è stato invece scelto di mantenere una dimensione, uguale per ogni frame della stessa animazione, tale da contenere lo spostamento del soggetto all'interno del frame stesso e di aggiornare la traslazione del rettangolo a fine animazione. In questo modo viene preservata la dinamica esatta dell'animazione originale del gioco.

Nel metodo Update viene stabilito quale sprite disegnare in base allo stato del tasto corrispondente e aggiornato il frame dello sprite.

```
protected override void Update(GameTime gameTime){
      //Right Arrow is Up
      if (Keyboard.GetState().IsKeyUp(Keys.Right) & rightDown)
      {
            rightDown = false;
            sprite = STOP;

      }
                  //Right Arrow is Down
      if (Keyboard.GetState().IsKeyDown(Keys.Right)){
            rightDown = true;
            sprite = PASS1;
      }
      if(gameTime >= framerate)
            SourceRect.X =+ 180; //frame width = 180
}
```

Lo sprite viene quindi disegnato in un rettangolo di output (DestinationRec) prendendo come input il frame sottostante l'area del rettangolo SourceRect.

```
protected override void Draw(GameTime gameTime){

      spriteBatch.Begin();
            spriteBatch.Draw(Sprite,DestinationRec,SourceRec);
      spriteBatch.End();

}
```



3.3 Problemi di Immagine

Come già detto nel gioco originale viene applicato un filtro grafico a tutta la scena. Nel caso del personaggio principale si evince che i pixel che compongono il bordo del corpo vengono interpolati con i pixel dello sfondo sottostante. Dato che nella fase di acquisizione delle animazioni del personaggio lo sfondo è un'immagine bianca il risultato finale presenta il soggetto circondato da un bordo di colore tendente al bianco, che male si adatta agli sfondi dei livelli del gioco che generalmente non hanno tinte tendenti al bianco.

In questa immagine lo sfondo è nero per rendere ancora più evidente il problema trattato ma tipicamente il risultato con gli sfondi del gioco è simile. Inizialmente è stato pensato di rendere trasparente il bordo chiaro ma ciò spesso implicava una perdita di dettaglio nella struttura del soggetto ed a volte di intere parti del corpo.

Come soluzione completa al problema è stato implementato nel programma SpriteSheetMaker un algoritmo capace di interpolare il colore del bordo esterno con il colore del bordo internamente adiacente. Nello specifico è stata implementata una sorta di scanline che individuasse tutti e soli i pixel del bordo esterno da correggere; sulla base delle coordinate di questi pixel l'algoritmoricava la topologia di tutti gli altri bordi sempre più interni.

Dalla composizione cromatica corretta del bordo più interno ne vengono interpolati i pixel per correggere il bordo adiacente sino ad arrivare al bordo più esterno.

3.4 Ampliamento Set Azioni

Il passo successivo ha riguardato il fornire al personaggio principale una gamma ampia di azioni da poter eseguire.

Sono state quindi acquisite tutte le movenze principali di Abe come descritto prima. Quindi le animazioni di base da implementare sono le seguenti :

- Attendere

- Camminare

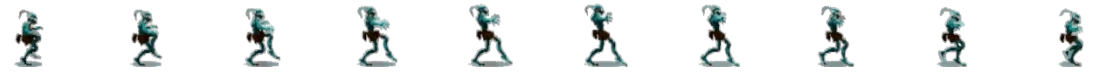
- Saltare in avanti

- Saltare in alto

- Correre

- Salto in corsa

- Camminare furtivamente



- Arrampicarsi

- Rotolare

- Accucciarsi

- Girarsi

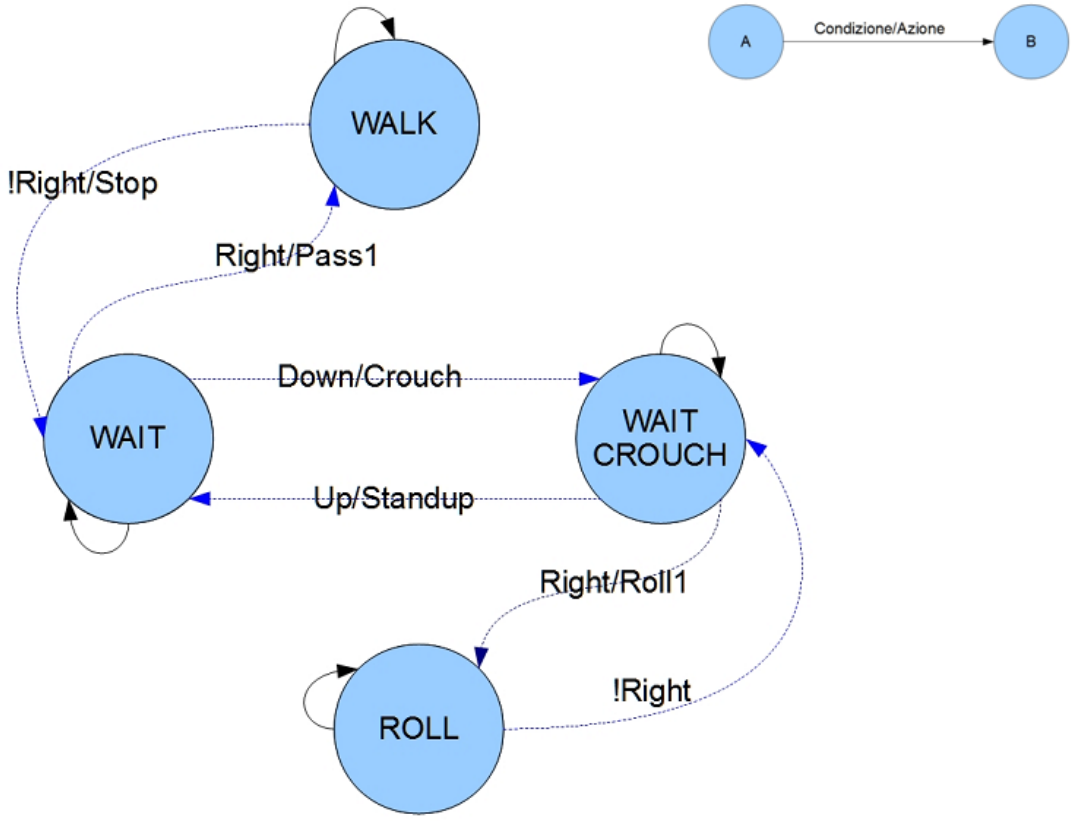
Inizialmente si è cercato di implementare tutta la gestione dei movimenti direttamente nei vari punti di controllo del metodo *Update*. Con l'ampliarsi dei movimenti di Abe il codice si è rivelato alquanto ingestibile. Al fine di rendere compatto e ben strutturato lo sviluppo della logica riguardante i movimenti e le azioni da eseguire è stato deciso di racchiuderlo in un modulo a parte. Questo modulo è stato sviluppato attraverso l'implementazione di un automa.

3.4.1 Automa a stati finiti

Sommariamente ogni stato dell'automa racchiude un tipo di azione composta da una o più animazioni. La transizione da uno stato all'altro avviene se una determinata condizione è soddisfatta, sulla base delle variabili di controllo settate nel metodo *Update*. Al verificarsi di una condizione può seguire una determinata azione. Le animazioni contenute nello stato corrente dell'automa vengono eseguite in un loop, interrotto solo quando la condizione posta su uno qualsiasi degli archi uscenti è soddisfatta. Le animazioni contenute nell'azione eventualmente associata alla condizione vengono eseguite una sola volta. Sono state quindi create quattro classi per implementare l'architettura dell'automa:

- ***STATEa***
- ***PRECONDITIONa***
- ***ACTIONa***
- ***ARCa***

Nella classe *AbeControl* è stato sviluppato l'algoritmo che esegue l'automa e la costruzione dello stesso. Ecco una porzione dell'automa prendendo in esame solo tre movimenti possibili.



3.4.2 Implementazione

Nel costruttore della classe ***STATEa*** ogni istanza di stato contiene una lista di animazioni da eseguire in loop.

```
if (name == "WAIT")
       AnimsTODO.Add(Control.Abe.WAIT);
if (name == "WAITCROUCH")
       AnimsTODO.Add(Control.Abe.WAITcROUCH);
if (name == "WALK"){
       AnimsTODO.Add(Control.Abe.PASS_2);
       AnimsTODO.Add(Control.Abe.PASS_3);
}
if (name == "ROLL"){
       AnimsTODO.Add(Control.Abe.ROLL2);
       AnimsTODO.Add(Control.Abe.ROLL3);
}
```

Similmente vengono definite le azioni possibili nella classe ***ACTIONa***.

```
if (name == "STOP")
       AnimsTODO.Add(Control.Abe.STOP);
if (name == "PASS1")
       AnimsTODO.Add(Control.Abe.PASS_1);
if (name == "CROUCH")
       AnimsTODO.Add(Control.Abe.CROUCH);
if (name == "STANDUP")
       AnimsTODO.Add(Control.Abe.STANDUP);
if (name == "ROLL1")
       AnimsTODO.Add(Control.Abe.ROLL1);
```

Nel costruttore della classe PRECONDITIONa ogni precondizione contiene una espressione booleana.

```
if (name == "WALK")
       if ((Control.game.rightDown & Control.Abe.Direction == 1) ||
       (Control.game.leftDown & Control.Abe.Direction == -1))
              return true;
if (name == "STOP")
       if ((!Control.game.rightDown & Control.Abe.Direction == 1) ||
       (!Control.game.leftDown & Control.Abe.Direction == -1))
              return true;
if (name == "TURN")
       if ((Control.Abe.Direction == -1 & Control.game.rightDown) ||
       (Control.Abe.Direction == 1 & Control.game.leftDown))
              return true;
if (name == "UP")
       if (Control.game.upDown)
              return true;
if (name == "DOWN")
       if (Control.game.downDown)
              return true;
```

In ***AbeControl*** viene così costruito l'automa. Il metodo ***LinkTO*** collega due stati e restituisce un arco. Il metodo ***addPRECONDITION*** lega la precondizione ad un arco e gli associa un'azione.

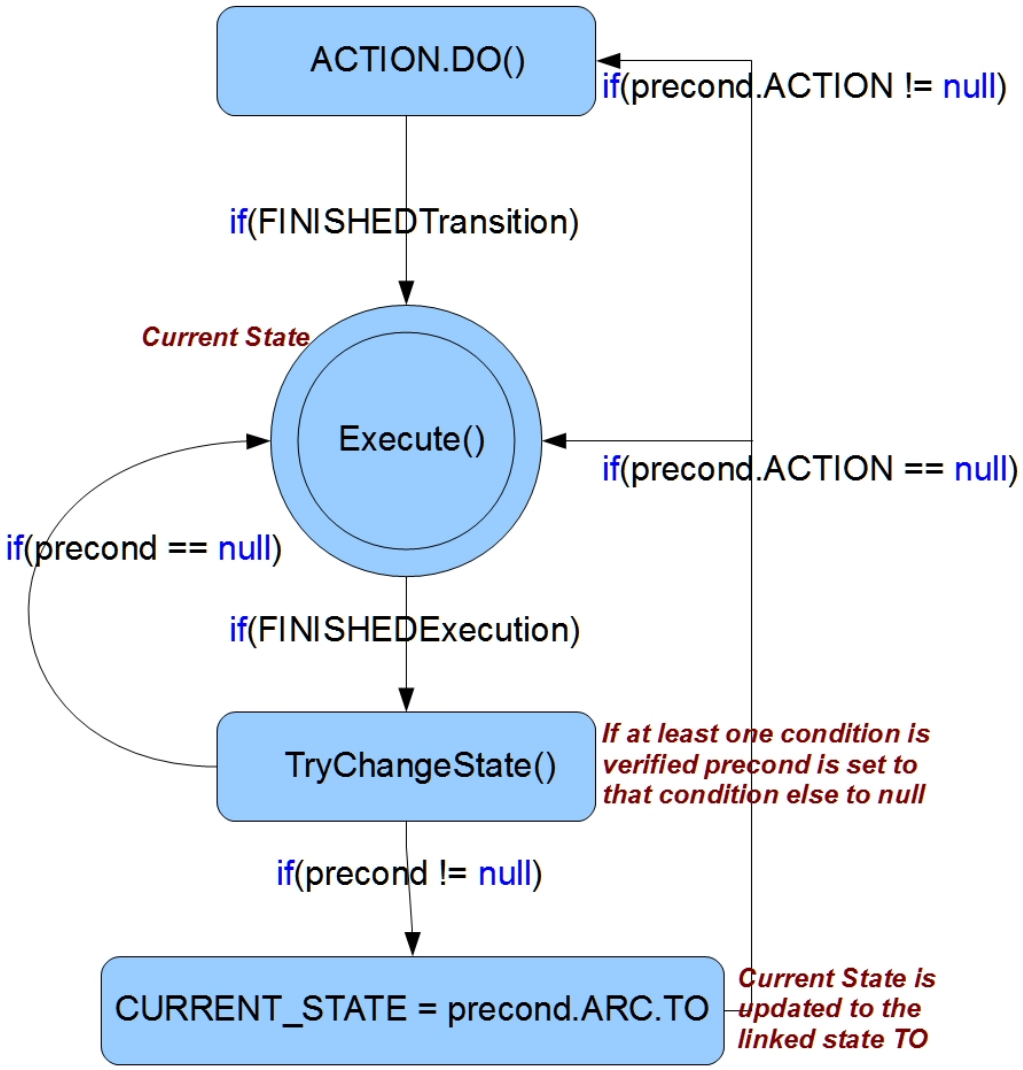
```
WAIT = new STATEa("WAIT", this);
WALK = new STATEa("WALK", this);
WAITCROUCH = new STATEa("WAITCROUCH", this);
ROLL = new STATEa("ROLL", this);

WAIT.LinkTO(WAITCROUCH).addPRECONDITION(PRECONDITIONS.DOWN,ACTIONS.CROUCH);
WAIT.LinkTO(WALK).addPRECONDITION(PRECONDITIONS.RIGHT, ACTIONS.PASS1);

WALK.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.STOP, ACTIONS.STOP);

WAITCROUCH.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.UP, ACTIONS.STANDUP);
WAITCROUCH.LinkTO(ROLL).addPRECONDITION(PRECONDITIONS.RIGHT,ACTIONS.ROLL1);

ROLL.LinkTO(WAITCROUCH).addPRECONDITION(PRECONDITIONS.STOP);

CURRENT_STATE = WAIT;
```

Viene anche definita la **routine** che si occupa di eseguire l'automa.

```
public void ROUTINE(){
	if (FINISHEDTransition)
	{
		if (FINISHEDExecution)
			precond = CURRENT_STATE.TryChangeState();
		if (precond != null)
			FINISHEDExecution = true;
		else
			FINISHEDExecution = CURRENT_STATE.Execute();
	}
	if (FINISHEDExecution)
	{
		CURRENT_STATE.AnimsTODO[CURRENT_STATE.CURRENT_ANIM].Reset();
		PREV_STATE = CURRENT_STATE;
		if (FINISHEDTransition & precond == null)
			precond = CURRENT_STATE.TryChangeState();
		if (precond != null)
		{
			if (precond.ACTION != null)
			    FINISHEDTransition = precond.ACTION.DO();
			 if (FINISHEDTransition)
			{
			    CURRENT_STATE = precond.ARC.TO;
			    CURRENT_STATE.ENDSTATE = false;
			    precond = null;
			 }
		}
	}
}
```

Di seguito viene mostrato uno schema del flusso di esecuzione dell'automa.

La routine che gestisce l'automa viene chiamata all'interno del metodo Update, dopo che le variabili di controllo che gestiscono lo stato dei tasti premuti sono state opportunamente aggiornate.



L'esecuzione delle animazioni che compongono uno stato può essere interrotto al termine di ogni singola animazione, se almeno una delle precondizioni poste sull'arco è verificata: la verifica delle condizioni poste sull'arco avviene al completamento di ogni animazione. Diversamente il set di istruzioni poste nel passo di transizione tra due stati è atomico, ovvero

lo stato successivo viene aggiornato a stato corrente solo quando le precondizioni vengono soddisfatte e tutte le azioni sono eseguite. Nel caso in cui ad una precondizione non è stata associata alcuna azione viene direttamente iniziata l'esecuzione del nuovo stato corrente.

Questa implementazione permette di ampliare in modo semplice e ben strutturato il set di movimenti di Abe.

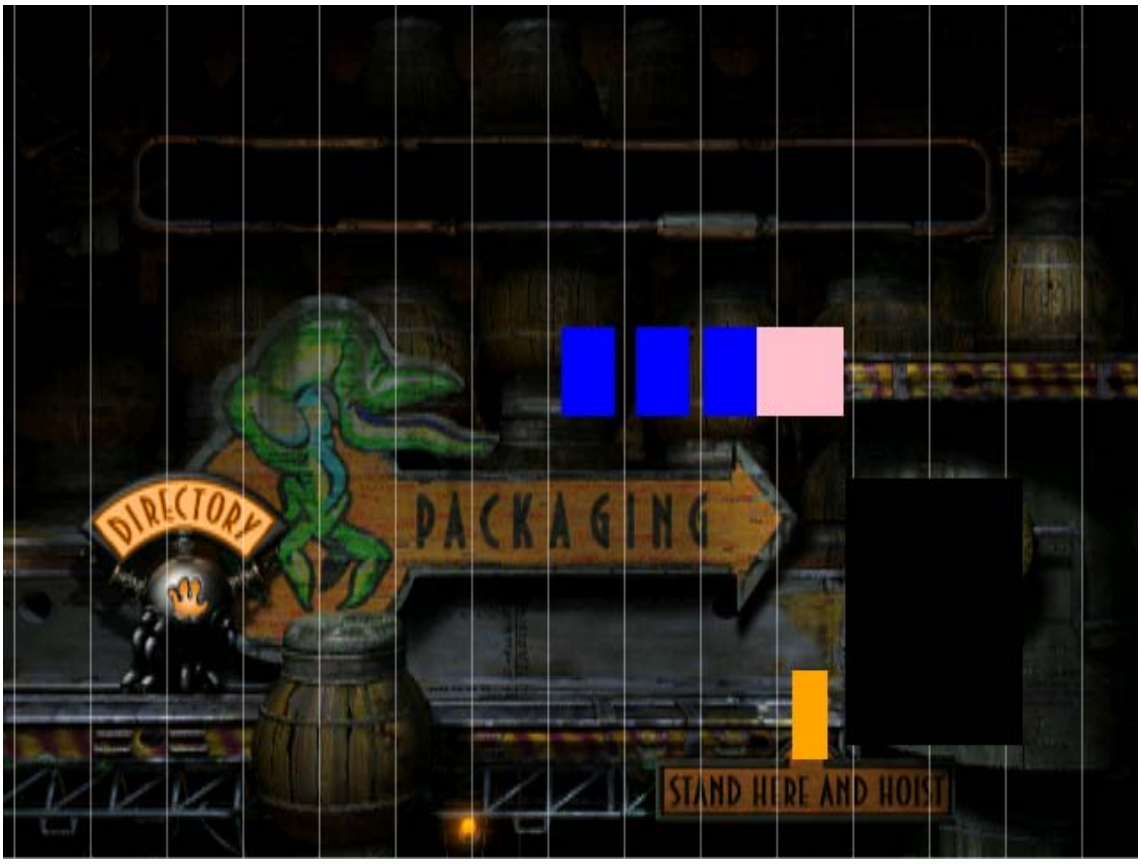
3.5 Interazione con l’ambiente

Le immagini che fanno da sfondo ai livelli del gioco presentano in realtà tre piani virtuali di profondità.

Il primo livello di profondità generalmente comprende il vero sfondo della scena. Nel terzo livello sono posti alcuni elementi scenici che aumentano il senso di profondità percepito. Nel

secondo livello si trovano tutti i personaggi e l'intero set di elementi con i quali il personaggio principale interagisce.

In questo esempio compare un pavimento su cui camminare, una struttura che blocca il cammino sottostante una piattaforma sulla quale Abe si può aggrappare e proseguire il cammino. Trattandosi di un'immagine non esiste ovviamente alcuna struttura logica che definisca esattamente questi elementi, che quindi devono essere mappati in opportune strutture gestibili all'interno del progetto Xna. A questo scopo è stato sviluppato il programma LevelMaker, che implementa un editor di livelli da poter importare nel progetto Xna. LevelMaker permette di caricare lo sfondo del livello da creare e di associare ad ogni tipo di elemento di interazione una descrizione che ne definisca fondamentalmente il tipo e la collocazione spaziale all'interno della scena. Nello specifico è stata creata la classe *Asset* per marcare ogni tipo di elemento. Ogni istanza di *Asset* contiene un nome ed un rettangolo specifici per l'elemento della scena che si desidera riportare nella definizione del livello. Ogni Asset viene quindi posizionato in corrispondenza della struttura che esso rappresenta.



In questo esempio l'asset nero definisce il muro, i due asset di colore bianco e arancione definiscono una zona nella quale si può salire e scendere. Gli asset blu definiscono una zona di vuoto (per comodità è stato scelto di gestire il comportamento del personaggio nei punti privi di pavimentazione).

Alla fine del flusso di lavoro viene generato un file xml contenente la lista di tutti gli asset che compongono la struttura del livello.

```
<?xml version="1.0" encoding="UTF-8"?>
<LEVEL>
  <ENVIOREMENT NAME="R1P15C02" />
  <GODOWNLIST>
    <GODOWN X="271" Y="185" WIDTH="30" HEIGHT="50" />
    <GODOWN X="314" Y="185" WIDTH="30" HEIGHT="50" />
    <GODOWN X="357" Y="185" WIDTH="30" HEIGHT="50" />
  </GODOWNLIST>
  <APPIGLIOLIST>
    <APPIGLIO X="443" Y="385" WIDTH="30" HEIGHT="50" />
  </APPIGLIOLIST>
  <HOISTLIST>
    <HOIST X="406" Y="185" WIDTH="60" HEIGHT="50" ADVISE="N" />
  </HOISTLIST>
  <BLOCCOLIST>
    <BLOCCO X="471" Y="300" WIDTH="60" HEIGHT="100" />
  </BLOCCOLIST>
</LEVEL>
```

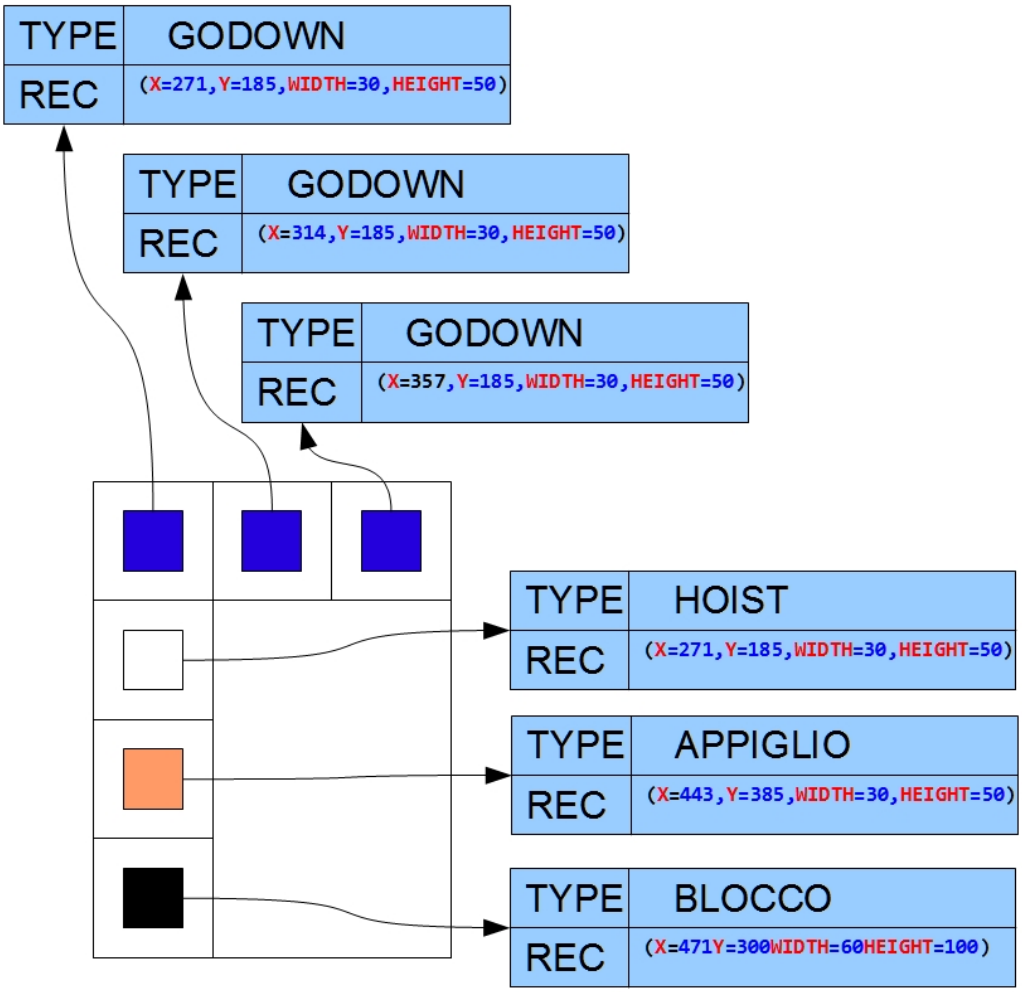
Il file xml viene quindi importato e caricato nel progetto Xna. Questa fase è stata sviluppata in modo custom senza l'ausilio della Content Pipeline standard di Xna, dato che il Content Processor per file generici xml non esiste.

A partire dal Document Object Model viene navigata la struttura del file e creata un'istanza della classe *Level* contenente tutti i riferimenti alle caratteristiche strutturali del livello generato. Questi riferimenti vengono generalmente acceduti per verificare se il personaggio (nello specifico la sua boundingbox) è in collisione e di conseguenza gestire opportunamente quale movimento può essere intrapreso.

La classe *Collision* descrive gli elementi con i quali i personaggi possono entrare in collisione, come avveniva con la classe *Asset* del programma *LevelMaker*.

La struttura dati fondamentale contenuta in un oggetto di tipo *Level* è una matrice (*collisiongroups*) contenente tutti i riferimenti agli oggetti di tipo *Collision* .

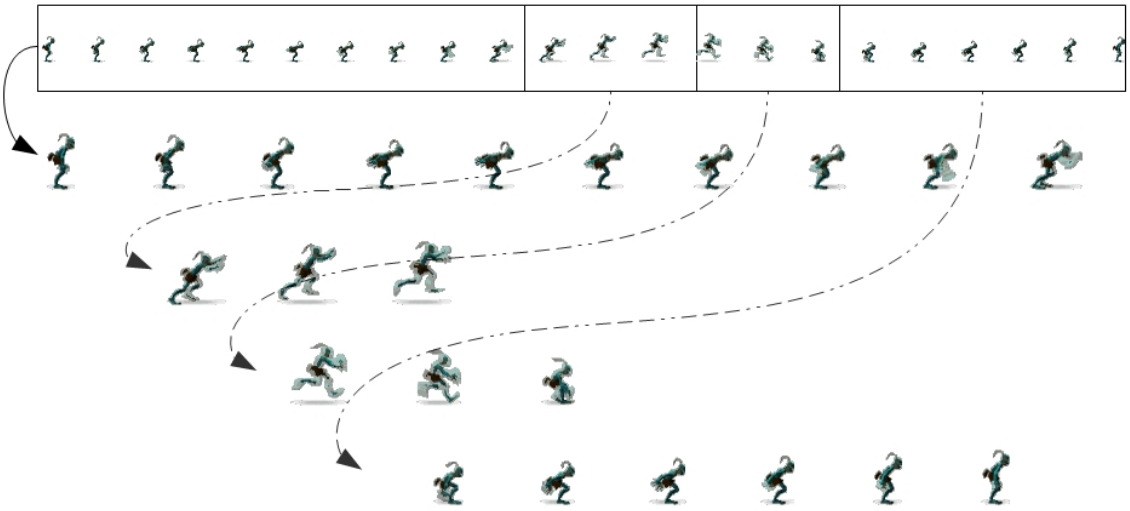
Uno schema della struttura dati COLLISIONGROUPS



Ogni schermata di gioco viene idealmente suddivisa in quindici aree adiacenti. Ogni area racchiude il minimo passo di avanzamento che ogni personaggio effettua lungo il cammino all'interno del livello. La logica che gestisce quindi lo spostamento di un soggetto all'interno di un ambiente e la loro interazione, discretizza ogni movimento nei moduli atomici che lo compongono. Al completamento di ogni modulo viene effettuato un controllo di collisione con l'ambiente circostante.

Per questo motivo le animazioni che fanno avanzare il personaggio in più aree vengono spezzate in più sequenze.

Ad esempio l'animazione del salto in avanti è composta da quattro sequenze. Le prime tre sequenze determinano lo spostamento nell'area successiva, mentre la quarta dispone Abe in una fase di atterraggio nell'area corrente.



In accordo con le possibili strutture architettoniche presenti nell’ambiente, le sequenze che compongono il salto vengono eseguite interamente dalla prima all’ultima solo nel caso in cui la porzione del quadro interessata risulti priva di muri o zone di vuoto.

In questo esempio, l’animazione può essere interrotta durante le prime tre sequenze in caso di collisione con un muro, oppure nell’ultima sequenza in caso di collisione con una zona di vuoto.

L'automa che regola il controllo di Abe effettua un controllo di collisione al termine di ogni sequenza.

Se al termine della seconda sequenza Abe collide con un muro l'automa interrompe l'esecuzione dello stato corrente, esegue le animazioni di impatto col muro e di riassestamento in posizione di riposo e quindi transita nello stato di WAIT.

```
JUMP.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.BLOCK,
                                  ACTIONS.AND(ACTIONS.KO, ACTIONS.STANDUP));
```

3.6 Personaggi e loro IA

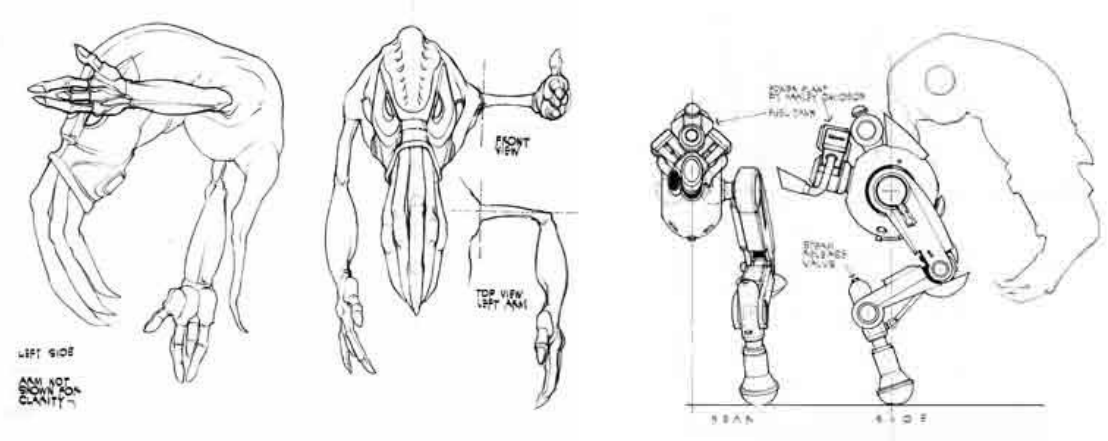
Una volta definito il sistema di controllo di Abe e la logica di interazione con l'ambiente sono stati sviluppati i personaggi con i quali il giocatore avrà modo di interagire.

Una volta definito il sistema di controllo di Abe e la logica di interazione con l'ambiente sono stati sviluppati i personaggi con i quali il giocatore avrà modo di interagire.

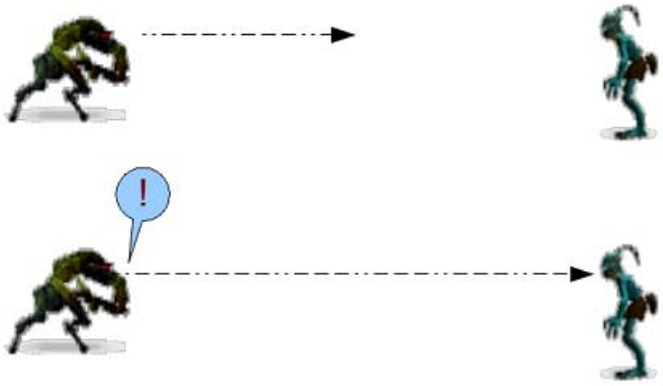
I personaggi che quindi popolano i livelli del gioco vengono suddivisi in due categorie : amici (i **Mudokon**) e nemici (gli **Slig**). I nemici rappresentano l'ostacolo principale che Abe deve aggirare per poter proseguire lungo il percorso. I personaggi non ostili devono essere guidati dal giocatore verso un punto di salvezza. Entrambe le categorie sono gestite da routine autonome che determinano la logica di comportamento del soggetto in base allo stato dell'ambiente circostante.

Come nel caso della gestione del sistema di controllo le IA di un personaggio vengono gestite da un automa, uno per ogni istanza di personaggio .Le risorse necessarie alla rappresentazione dei personaggi vengono acquisite dal gioco originale tramite Camtasia come descritto nel caso di Abe.

3.6.1 Slig



Il comportamento tipico di uno Slig è quello di ispezionare il livello di gioco e utilizzare l'arma in dotazione per eliminare Abe. La capacità di individuare il personaggio principale varia a seconda del grado della propria IA che viene determinata dinamicamente in base al livello di difficoltà del quadro corrente. Livelli bassi di IA permettono ad Abe di avvicinarsi al nemico senza essere visti, aumentando così il grado di libertà del giocatore.



Esistono aree del livello, rappresentate come zone d'ombra, entro le quali Abe si può nascondere, aspettare che il nemico sia passato e proseguire lungo la strada. Uno Slig è capace di rilevare i rumori prodotti dalla maggior parte delle movenze di Abe. In questo caso si rivela molto utile la capacità di poter camminare furtivamente, azionabile utilizzando la giusta combinazione di tasti. Queste prime meccaniche di gioco forniscono diverse situazioni in cui il giocatore deve attuare strategie opportune per poter avanzare nel livello.

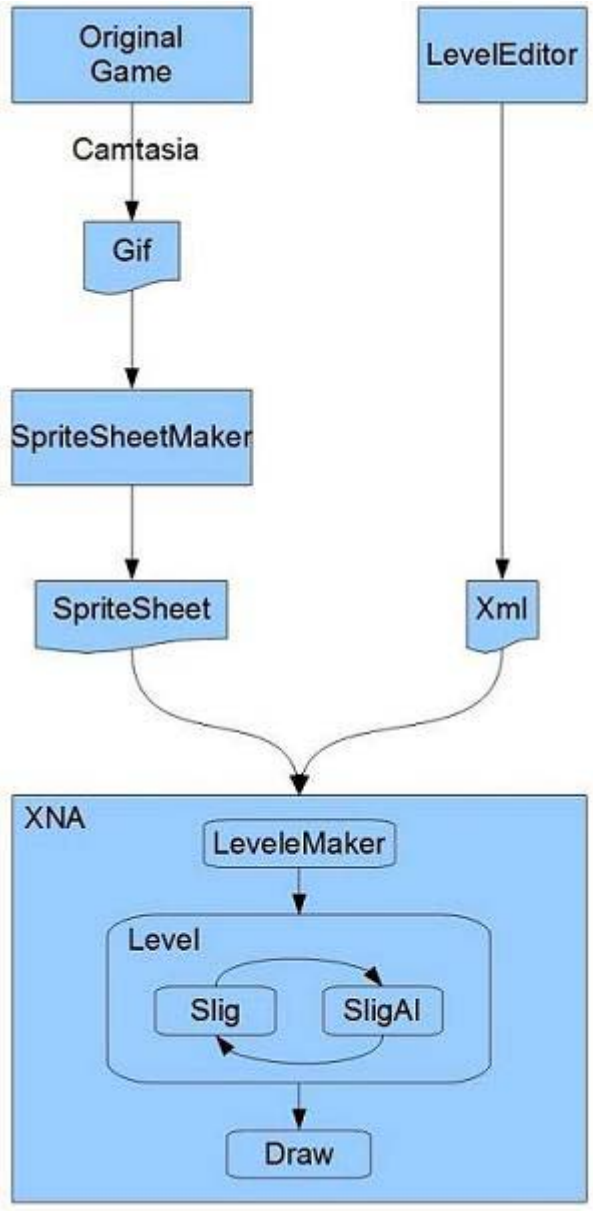
Un livello viene popolato di nemici con modalità simili a quanto descritto nella fase di editing con la mappatura degli elementi strutturali di cui è composto. Si procede quindi ad inserire nel quadro uno o più *SpawnPoint*, ovvero elementi che generano Slig durante l'esecuzione del gioco. Ogni *SpawnPoint* può essere editato permettendo così di determinare il numero di personaggi che genererà e di determinare il livello di IA ed il loro stato di partenza. In questo esempio l'elemento *SpawnPoint* è composto da due Slig, uno inizialmente dormente e con IA di secondo livello, un altro con stato iniziale SENTINELLA e con IA di terzo livello.

Il file xml generato avrà quindi questa forma :

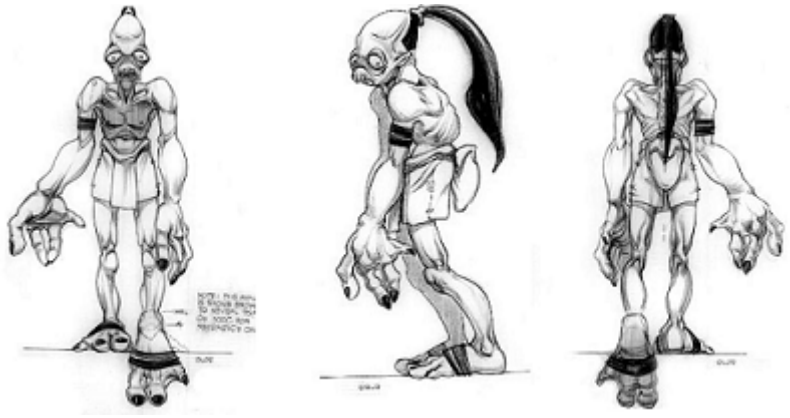
```
<?xml version="1.0" encoding="UTF-8"?>
<LEVEL>
  <ENVIOREMENT NAME="R1P15C02" />
  <SPAWNSLIGLIST>
    <SPAWNSLIG X="271" Y="185" WIDTH="44" HEIGHT="50" />
      <Slig STATE="SLEEP" AI="2"/>
      <Slig STATE="SENTINELLA" AI="3" />
    </SPAWNSLIG>
 </SPAWNSLIGLIST>
</LEVEL>
```

In Xna viene quindi creato il livello contenente due Slig. Ogni Slig è gestito da due classi, *Slig* e *SligAI*. La prima classe gestisce tutta la parte riguardante la visualizzazione a schermo delle animazioni, mentre la seconda implementa l'automa che regola il comportamento del soggetto. Le due classi comunicano continuamente in un loop definito dal metodo *Update* della classe principale del gioco.

Di seguito viene mostrato lo schema del flusso di lavoro che riguarda tutta la gestione di uno Slig:



3.6.2 Mudokon



Le modalità di creazione di un Mudokon e della sua gestione nel progetto Xna sono fondamentalmente simili a quelle dello Slig. Il comportamento standard di un Mudokon è quello di rimanere fermo in una zona del quadro. Tramite un comando il giocatore può interagire con un Mudokon e farsi seguire lungo i livelli. Ogni Mudokon che il giocatore incontra lungo il percorso può essere aggiunto al gruppo formando così una fila, dove ogni personaggio occupa un proprio posto ben distinto. Tutti i componenti della fila seguono Abe e si dispongono in modo da non sovrapporsi nello stesso punto della fila. In realtà solo il primo Mudokon segue Abe, mentre ognuno dei Mudokon successivi si riferisce a quello immediatamente precedente. L'IA che governa i loro movimenti non riconosce di proposito alcune zone di pericolo lungo il cammino, in quanto è compito del giocatore istruire il gruppo a fin che tutti superino determinati ostacoli. I Mudokon possono essere liberati tramite l'azionamento di appositi portali disposti in alcuni quadri.

3.7 Oggetti della Scena

Come già descritto una schermata di gioco è univocamente rappresentata da un'immagine contenente tre livelli di profondità. Nel secondo livello è dove avviene effettivamente l'interazione con le strutture dell'ambiente, i nemici e gli alleati. In questo livello vengono quindi posti tutti gli oggetti con i quali si può interagire, elementi attivi, la cui presenza contribuisce effettivamente nell'economia dell'azione di gioco.

Nel primo e nel terzo livello vengono solitamente posti oggetti di contorno, elementi passivi, con i quali il giocatore non interagisce ma che aumentano l'impatto visivo percepito.

3.7.1 Oggetti attivi

Gli oggetti attivi vengono acquisiti ed importati in XNA con le stesse modalità dei personaggi. La logica che gestisce il loro comportamento è fondamentalmente più semplice di quella di un nemico o di un alleato, per questo è stato deciso di implementare la gestione di tutte le tipologie in sole due classi (ObjectA per la gestione grafica e ObjectControl per la logica del funzionamento).

Tipicamente molti oggetti hanno la funzione di ostacolare l'avanzamento lungo il percorso. Come mine o campi elettrici, che causano la morte del personaggio in caso di collisione. Altri oggetti invece possono essere usati per eliminare i nemici, usandoli come trappole da azionare opportunamente.

Principalmente gli oggetti vengono attivati interagendo con una leva, la quale ha associato univocamente l'oggetto collegato da azionare.

3.7.2 Oggetti passivi

Gli oggetti posti sullo sfondo della scena tipicamente sono delle sequenze video che dinamizzano alcune porzioni dell'immagine. Ad esempio all'interno di una locazione che raffigura un sistema di trasporto di barili, sullo sfondo viene collocato un video in loop di un nastro trasportatore in funzione.

Per importare e gestire i file video è stata utilizzata una content pipeline di tipo custom, sviluppata da terze parti.

Nel terzo livello di profondità sono collocati tutti gli oggetti di contorno alla scena. Il passaggio di un personaggio in corrispondenza di un oggetto posto su questo livello deve dare l'illusione della profondità. Tutti gli oggetti dovranno quindi essere disegnati per ultimi. Per semplificare il lavoro ad ogni immagine di sfondo di una schermata è associata una texture di pari dimensioni contente solo gli oggetti posti in rilievo.

Capitolo 3. Implementazione componenti di gioco

Capitolo 4

Generazione dinamica dei livelli

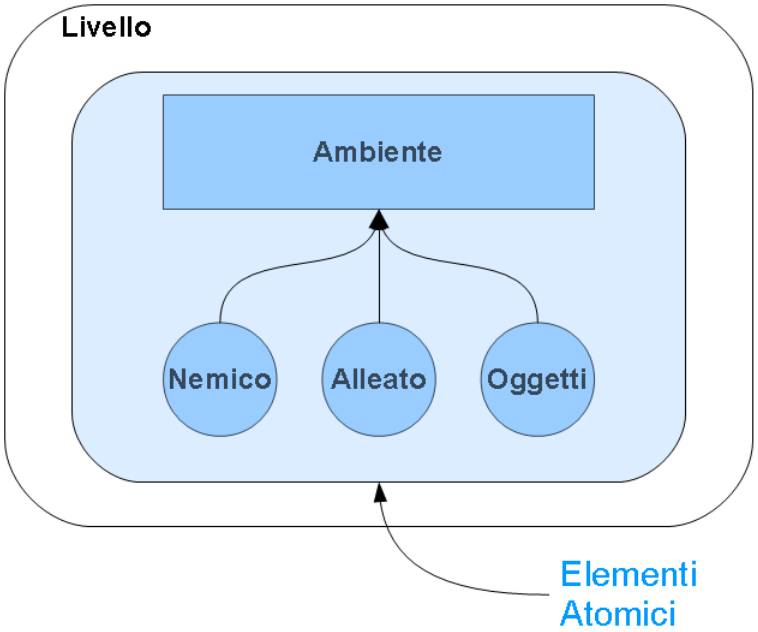
Conclusa la fase di ricerca e sviluppo dei componenti di base del videogioco, è stata definita una grammatica in grado di comporre i livelli di gioco a partire dagli elementi atomici.

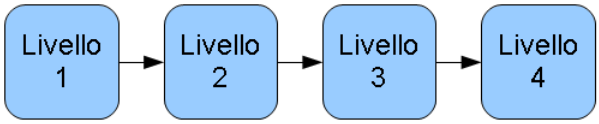
In questo capitolo viene quindi illustrato il passo generativo che determina lo sviluppo del percorso di gioco in tempo reale. Viene prima descritto il modello astratto del sistema che determina la componibilità dei livelli, sulla base del quale verranno definiti i vincoli della grammatica. In seguito vengono mostrate le procedure mostrate nel progetto, in riferimento ad un esempio concreto.

A conclusione del capitolo vengono mostrati alcuni casi d'uso dell'applicazione, confrontando i percorsi generati con un percorso definito staticamente nella versione originale del gioco.

4.1 Modello astratto

Nella generazione del gioco ogni passo generativo produce un modulo come composizione di elementi atomici.





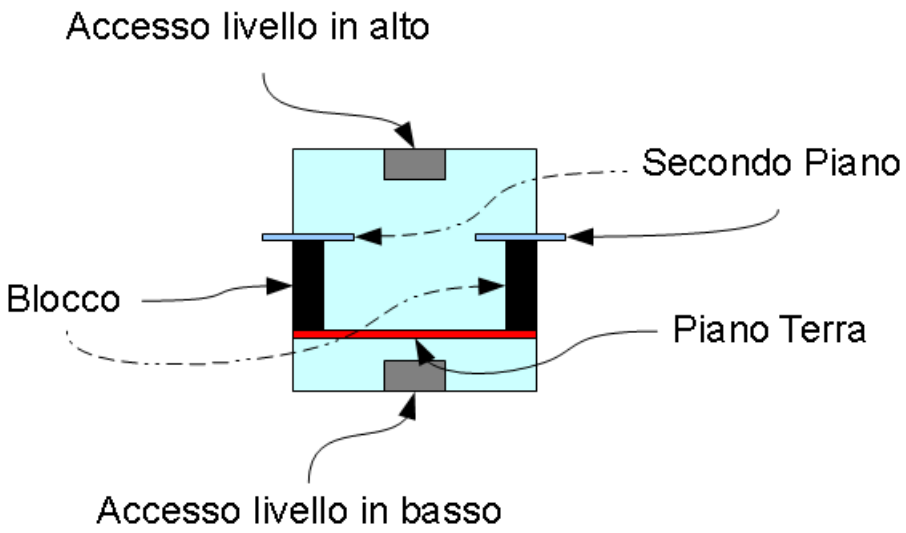
Percorso di gioco generato

La composizione deve rispettare determinati vincoli che definiscono la compatibilità degli atomi da combinare.

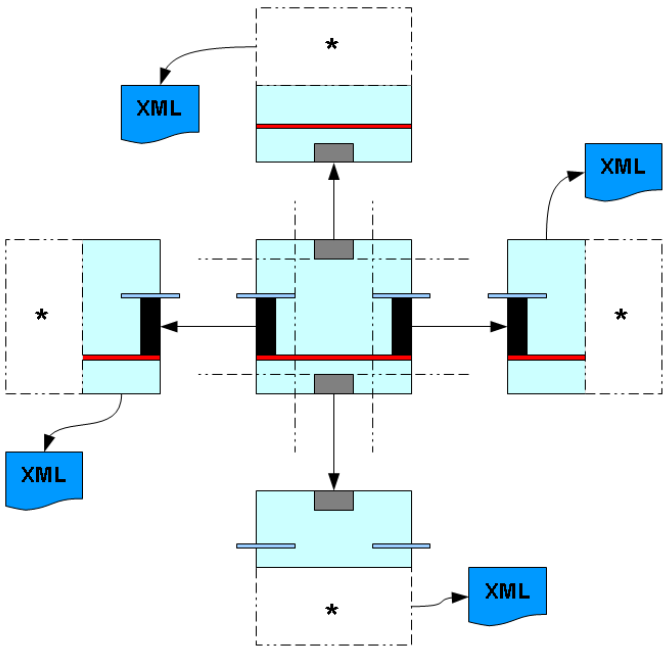
Il passo minimo di generazione che espande realmente la struttura di gioco viene definito dall'aggiunta di un nuovo livello. Il percorso viene quindi creato iterativamente e composto da tanti livelli quanti sono stati i passi di generazione.

La struttura architettonica di ogni livello pone dei vincoli a quanti e quali nuovi livelli si può legare. Partendo da un generico livello il numero minimo di nuovi livelli che possono essere generati direttamente da esso varia da 1 a 5, in base alla presenza o meno di strutture che

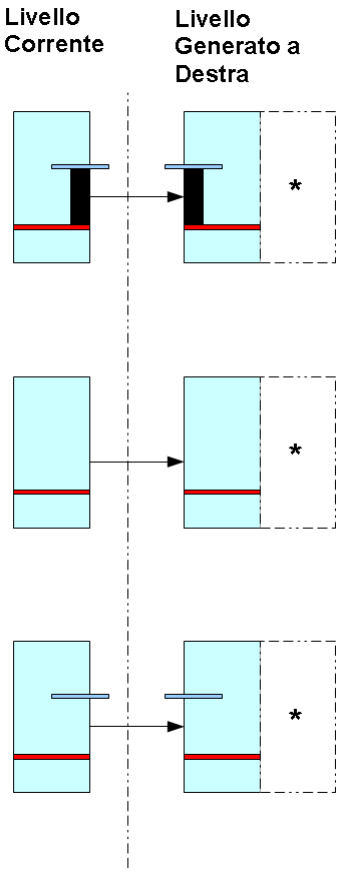
consentano l'avanzamento nelle quattro diverse direzioni cardinali e di una porta che conduca in un'altra locazione.



Per ognuna delle direzioni percorribili possono essere generate diverse istanze di un livello. Per ogni livello viene quindi definita una struttura architettonica, che rispetti un modello descrittivo standard per tutti i livelli, contenuto nel file XML prodotto nella fase di editing ed importato in XNA.



Data la direzione di avanzamento, la regola generale che stabilisce la compatibilità del livello successivo definisce che le porzioni relative ai lati adiacenti dei livelli manifestino una sorta di simmetria speculare.



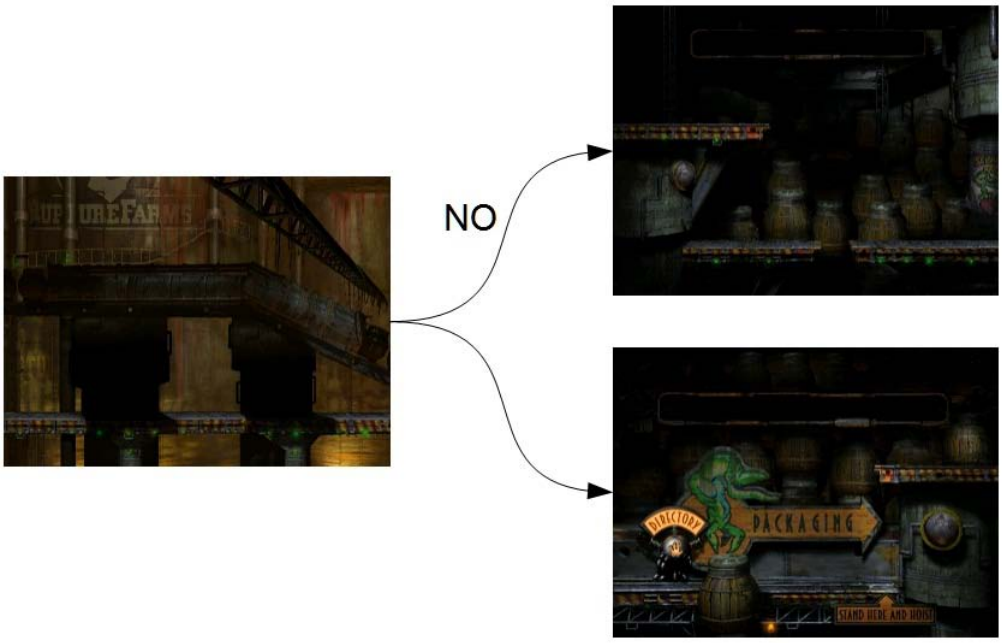
Gli elementi strutturali delle porzioni relative agli altri lati del livello possono avere qualsiasi forma. Viene quindi definita una prima forma della grammatica generativa, dove l'insieme degli elementi su cui applicare le regole di produzione è rappresentato da tutte le possibili istanze di livelli importate dall'editor e le produzioni vincolano le regole da applicare ad ogni passo, dove l'elemento sinistro è una possibile configurazione strutturale di un lato del livello e gli elementi a destra sono tutti i livelli compatibili.

4.2 Implementazione

Generalmente dal livello corrente si può passare al livello successivo proseguendo oltre i quattro bordi del canvas della scena oppure entrando in una porta.

Lo scopo principale di mutare quindi l'esperienza di gioco ad ogni sessione è quello di aumentare la longevità in termini di riuso del prodotto, mantenendo costanti le risorse necessarie alla creazione degli assets del gioco.

Dato il livello corrente la generazione casuale del livello successivo deve avvenire rispettando vincoli di compatibilità delle strutture architettoniche dei due livelli.



In questo esempio il livello corrente non presenta alcun ostacolo strutturale in corrispondenza del lato destro, quindi proseguendo oltre, il quadro successivo dovrà essere privo di qualsiasi struttura in corrispondenza del lato sinistro.

E' stata quindi implementata una funzione (*GetNextCompLevels*) che, dato il livello corrente e la direzione di spostamento, analizza la lista di tutte le istanze dei livelli disponibili e restituisce un lista contenente solo i livelli strutturalmente compatibili.

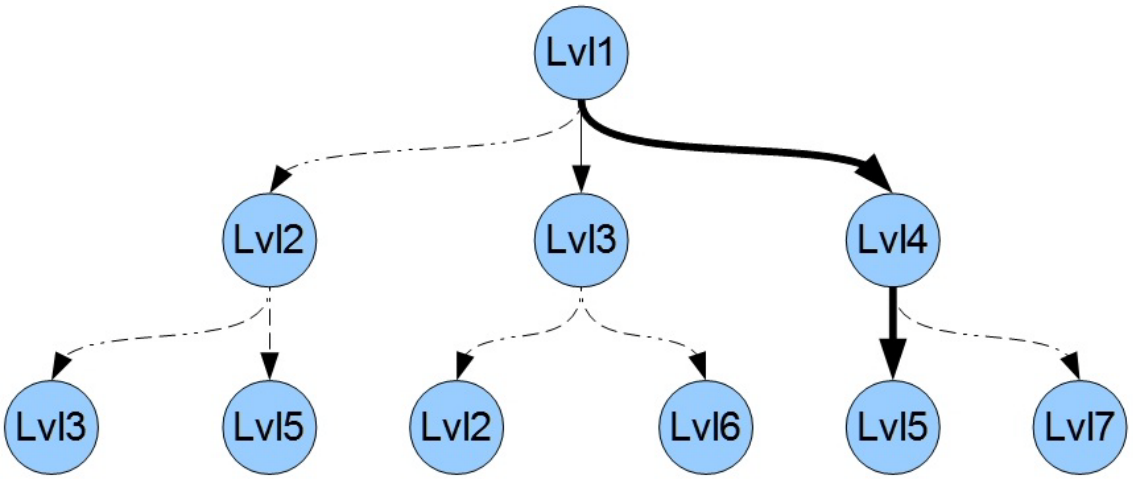
Di seguito viene mostrata una porzione del metodo, in riferimento al caso di generazione di un livello lungo la direzione destra.

```
List<Level> GetNextCompLevels(Level lvl, string Direction){
      foreach (Level comp in LevelMaker.lvlList){
            if (comp.Enviorement != lvl.Enviorement{
                  switch (Direction){
                        case "RIGHT":
                        if (lvl.BLOCCOUPRight == comp.BLOCCOUPLeft &&
                        lvl.FLOOR2NDRight == comp.FLOOR2NDLeft &&
                        lvl.BLOCCODOWNRight == comp.BLOCCODOWNLeft)
                        lvl.compatibleLevels.Add(new Level(comp));
                        break;
                         case "LEFT":
                        […]
                         break;
                         case "DOWN":
                         […]
                         break;
                         case "UP":
                         […]
                         break;
                         case "DOOR":
                         […]
                         break;
                     }
                }
            }
            return lvl.compatibleLevels;
        }
```

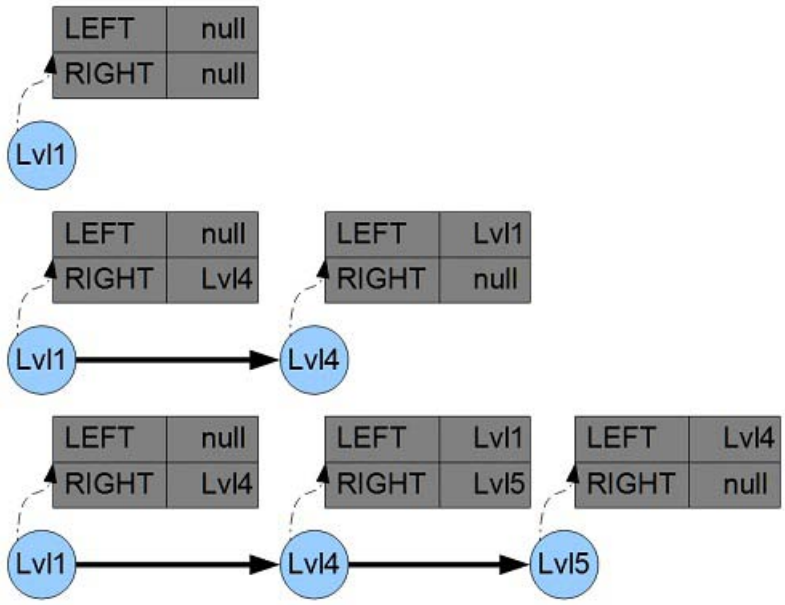
La lista viene poi passata ad un'altra funzione (*GetNextLvl*) che preleva un livello scelto casualmente la cui difficoltà rientri in un range consono con quella del livello appena lasciato.

```
Level getNextLvl(Level lvl,List<Level> compLvls){
      […]
      if (NUMLVLGEN < MAXLVLGEN){
            while (!found){
                  RangeDifficult++;
                  for (int i = 0; i < compLvls.Count; i++){
                        lvlsDifference = Math.Abs(lvl.difficult -
                        compLvls[i].difficult);

                        if (lvlsDifference <= Range){
                              lvlsNext.Add(compLvls[i]);
                              found = true;
                              […]
                        }
                  }
            }

            rand = random.Next(0, lvlsNext.Count);
            NextLevel = lvlsNext[rand];
            NUMLVLGEN++;
            REPORT.LEVELS.Add(NextLevel);
      }
      else
            SHOW_REPORT = true;
            […]
      return NextLevel;
}
```

A partire quindi dal livello iniziale possono essere generati diversi percorsi.



I nuovi livelli vengono quindi generati proseguendo lungo il percorso ma ogni livello mantiene il riferimento al suo predecessore.



4.3 Il sistema adattivo

Tipicamente i videogiochi permettono di preimpostare un determinato livello di difficoltà ad inizio partita. La difficoltà scelta determina i parametri di sfida offerti al giocatore lungo tutto lo svolgersi del gioco. In questo modo la difficoltà aumenta linearmente lungo il percorso; gli step dove la difficoltà subisce variazioni sono decisi a priori e non mutano ad ogni inizio di una nuova sessione di gioco.

Questo si traduce in un adattamento del giocatore al livello di sfida imposto dal gioco, che quindi in alcuni casi può risultare ottimale per lo stile e l'esperienza di un giocatore ma in altri risultare troppo facile o troppo frustrante.

Nello sviluppo di questo progetto si è scelto di implementare un sistema in grado di adattare il livello di difficoltà al giocatore. L'idea generale è stata quella di proporre variazioni continue ai parametri delle sfide, rendendo la curva di difficoltà non necessariamente lineare.

Al termine del percorso viene visualizzato un riassunto dei livelli esplorati. Per ogni livello il giocatore esprime un giudizio, in base al quale viene modificato il grado di difficoltà del livello in esame. In particolare il livello viene sostituito con una nuova istanza modificata.

La difficoltà viene modificata a partire da un parametro calcolato quando un livello viene istanziato nella fase di avvio del gioco. Ogni elemento atomico presente nel livello contribuisce al calcolo del grado di difficoltà. Viene quindi navigata la struttura dati del livello, discriminando le unità ostili da quelle fondamentalmente amichevoli. In base alla tipologia ogni elemento contribuisce positivamente o negativamente nel determinare la difficoltà del livello. Ogni elemento può influire in diversa misura in base al suo stato.

Nel caso in cui venga assegnato un giudizio positivo il sistema procede ad aumentare il grado di difficoltà. Ad esempio vengono aumentate le capacità delle intelligenze artificiali dei nemici o, nel caso in cui il livello in esame non contenga nemici, ne vengono aggiunti dinamicamente.

Se ad esempio il livello appena esaminato presenta un nemico con IA di livello base ed un set di due mine lungo il percorso, il giocatore può averlo superato attuando più di una strategia facilmente eseguibile : impadronendosi del corpo del nemico, poco scaltro per accorgersi della minaccia, oppure semplicemente correndo e oltrepassando le mine che essendo solo due risultano facilmente superabili.

Nel caso quindi che il giocatore trovi poco soddisfacenti le sfide proposte, il sistema aumenta la difficoltà del livello esaminando ogni elemento. Viene quindi aumentato il livello di IA del nemico, il set di mine viene ampliato e viene aggiunta una sentinella che impedisce al giocatore di ricorrere al potere di impadronirsi del nemico.

In questo nuovo scenario il giocatore dovrà quindi approcciarsi in modo sensibilmente diverso rispetto alla situazione precedente, rendendo di fatto la fruibilità del livello diversa dalla sua istanza in versione più facile.

4.4 Casi d’uso

In questo paragrafo vengono esaminati alcuni casi d’uso dell’applicazione.

Gli esempi riportati di seguito descrivono principalmente la componente adattiva e quella di generazione dinamica. In ogni esempio viene focalizzata l’attenzione su una componente, tralasciando altri particolari, al fine di rendere maggiormente chiare le caratteristiche specifiche di ognuna di esse.

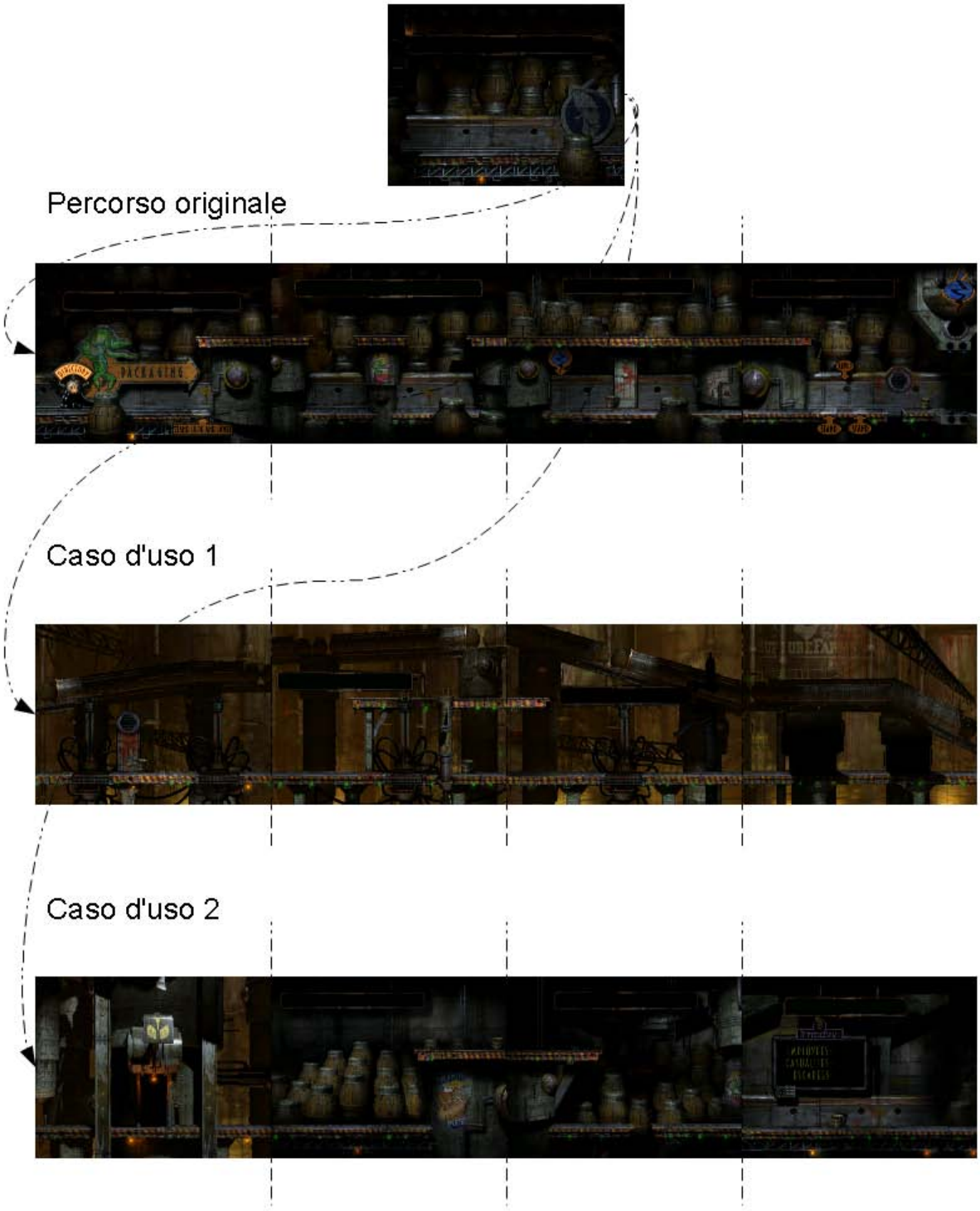
Nel caso di esempi incentrati sul sistema di generazione dinamica, per ogni livello del percorso sviluppato, vengono mostrati solo gli elementi architettonici di cui si compone, mettendo così in evidenza il sistema di valutazione dei vincoli strutturali.

Nel caso del sistema di adattamento del grado di difficoltà viene mostrato un breve percorso e la sessione del sondaggio proposto. Il successivo attraversamento genera di proposito i medesimi livelli incontrati nel percorso precedente, in modo da poter esaminare come tali livelli siano stati modificati sulla base delle valutazioni date dal giocatore.

Di seguito vengono esposti alcuni casi specifici:

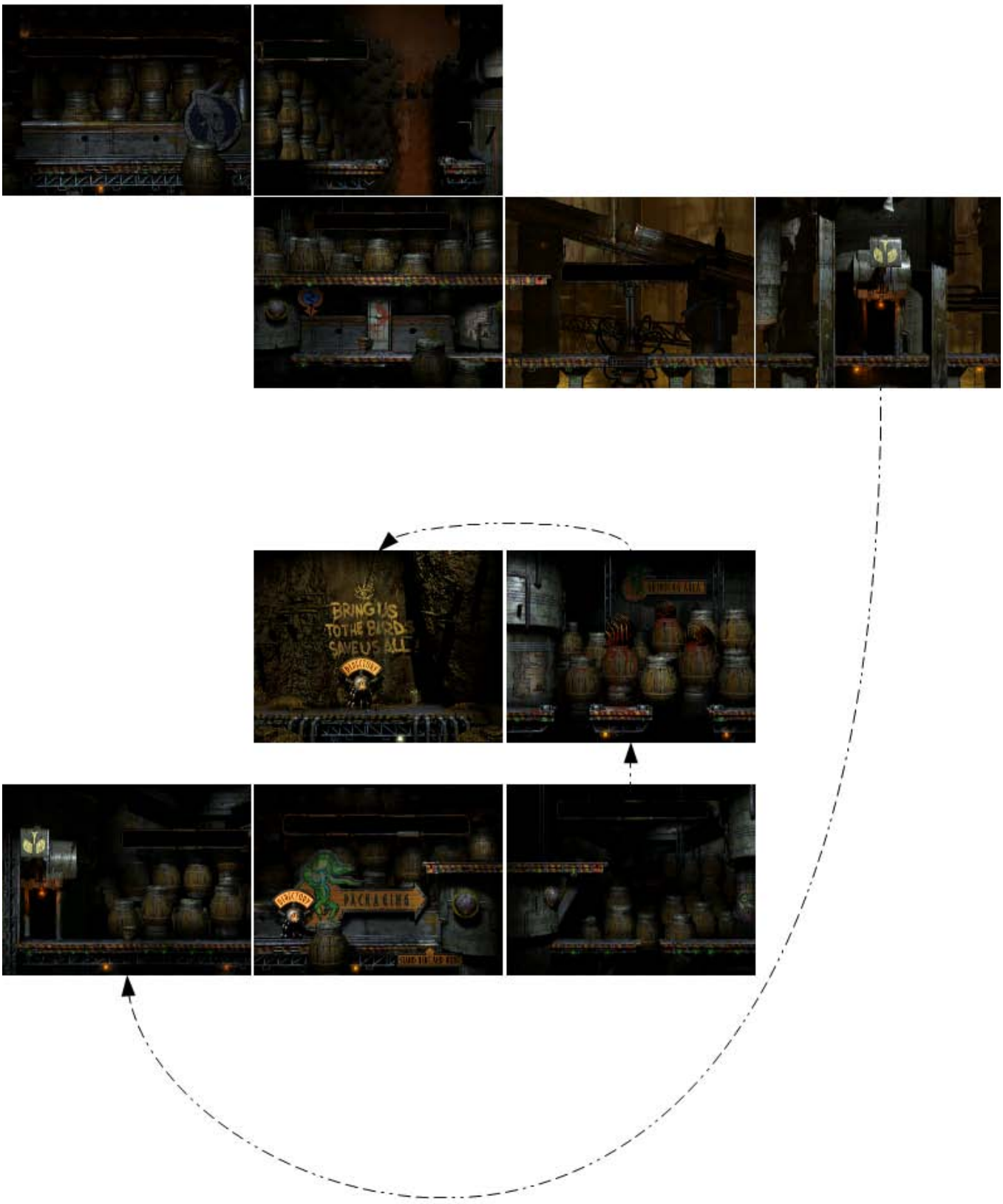
- **La generazione dinamica di 2 percorsi composti da 4 livelli**

A partire dal medesimo livello iniziale, vengono mostrate le generazioni di due percorsi distinti, composti entrambi da 4 livelli. Questi percorsi sono stati prodotti sulla base di circa 20 istanze di livelli disponibili, vincolando la generazione al solo caso di avanzamento verso destra. Le due produzioni sono messe a confronto del percorso proposto dal gioco originale.

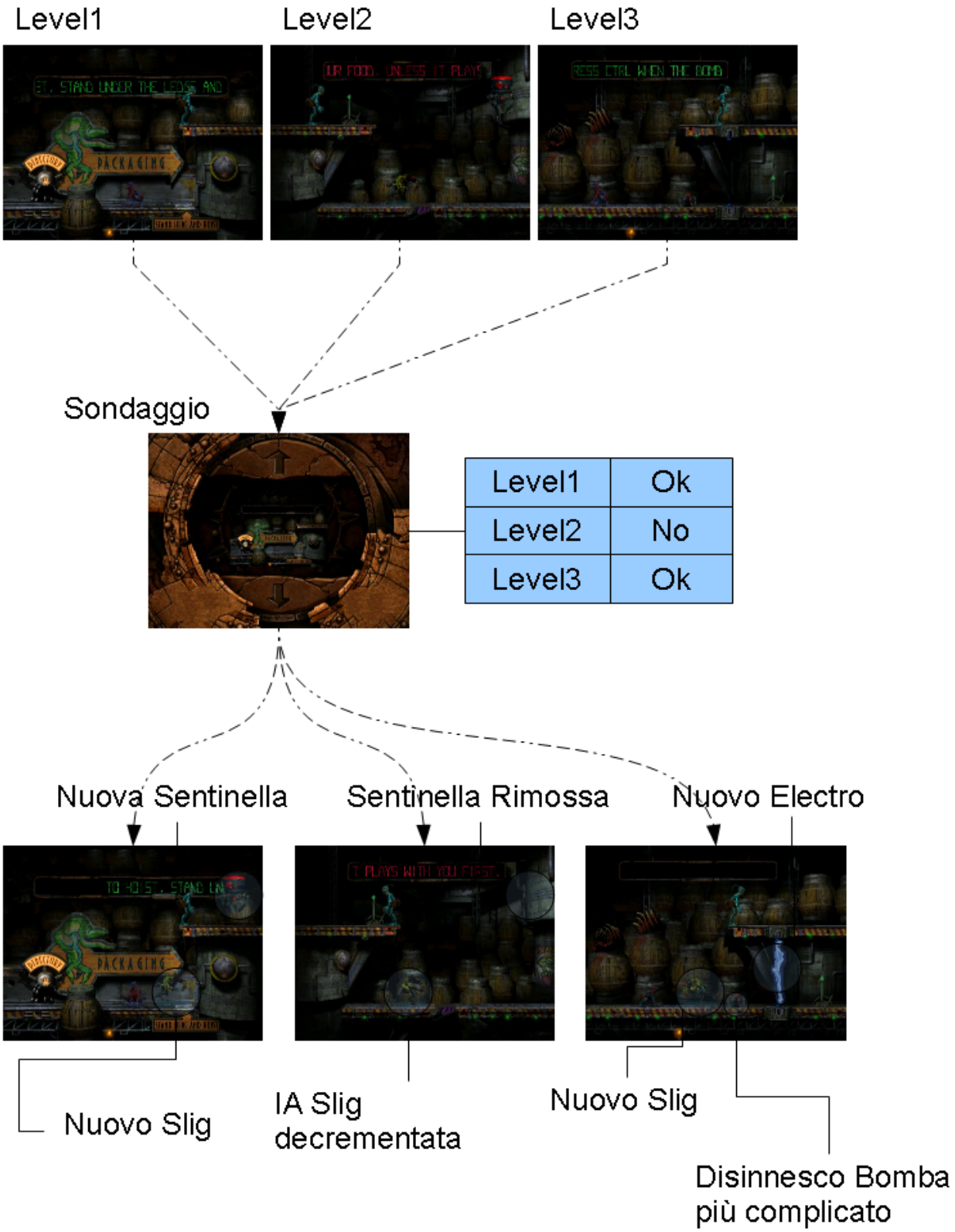


- **La generazione dinamica di 1 percorso completo**

Di seguito viene mostrato lo sviluppo di un percorso che coinvolge i passi generativi in tutte le 5 direzioni.



- **Adattamento del grado di difficoltà**



Capitolo 5
Conclusioni

In questo capitolo conclusivo vengono esaminati i risultati ottenuti dall'implementazione di un sistema a generazione dinamica di contenuti in ambito videoludico, riferendoci anche al caso di prodotti privi di comportamenti dinamici ed a possibili sue applicazioni in ambiti diversi da quello del progetto di tesi, come ad esempio quello 3D.

La prima considerazione viene fatta in riferimento proprio al videogioco Oddworld: Abe's Oddyse, rielaborato nella fase di reverse engineering per reperire gli elementi su cui modellare la grammatica generativa.

La struttura di gioco originale prevede come attività principale il condurre gli alleati (Mudokon) in salvo in determinati punti del percorso. I personaggi alleati sono in grado di percorrere solo i livelli che non siano composti da determinate "barriere architettoniche", fondamentalmente a causa dell'incapacità di eseguire salti. Quindi le zone di salvataggio vengono collocate in opportuni punti del percorso adatti alle capacità motorie dei Mudokon.

Nel caso invece del sistema sviluppato in un contesto dinamico non è possibile stabilire a priori in quali punti del percorso verrà generato il livello contenente il portale per liberare gli alleati incontrati nei livelli precedenti. Quindi è stato scelto di ampliare le capacità dei personaggi da salvare, rendendoli capaci di seguire il giocatore attraverso ogni livello. In quest'ottica si può delineare un meccanismo in grado di generare intere missioni di salvataggio, a partire dall'incontro di un personaggio, ponendo come meta finale della ramificazione del percorso un livello contenente un portale.

Il fatto di essere stati legati ai contenuti artistici del gioco originale, in particolare agli sfondi contenenti anche la rappresentazione grafica degli elementi ambientali, ha posto molti vincoli nella procedura generativa dei livelli.

La possibilità di scindere gli elementi strutturali dal contesto dello sfondo avrebbe permesso una maggiore capacità espressiva della grammatica generativa ma avrebbe anche implicato un lavoro di creazione di contenuti non prettamente attinenti alle tematiche trattate in ambito informatico.

Ipotizzando di sviluppare un sistema di generazione dinamica di contenuti adattabili al contesto di gioco corrente in un ambito 3D ci porta a considerare scenari meno vincolati rispetto al caso 2D.

Risulta evidente che fra tutte le tipologie di gioco il genere definito Free Roaming o Open World sia particolarmente adatto ad essere integrato in meccanismi di generazione dinamica. Come suggerisce il nome questa tipologia di gioco adotta un concept dove il giocatore è libero di muoversi attraverso ampi spazi, privi di barriere artificiali e dove il gioco si sviluppa fondamentalmente in modo non lineare.

Consideriamo ad esempio il caso di Grand Theft Auto 4, sviluppato da Rockstar Games e rilasciato nel 2008. Il giocatore si muove lungo le strade di una città virtuale costruita sulla base di New York.

L'obiettivo principale riguarda l'esecuzione di determinati incarichi, che tipicamente comprendono la seguente sequenza di eventi:

1. Il giocatore accetta l'incarico proposto
2. La missione principale viene attivata
3. Il giocatore si procura gli oggetti necessari a compiere la missione
4. Svolge la missione principale

In questo scenario il terzo evento potrebbe essere integrato in una sessione di gioco generata dinamicamente. Il sistema potrebbe generare una nuova porzione della città, come un quartiere o un isolato, combinando edifici ed elementi architettonici di diverse zone e generare gli oggetti da recuperare e i nemici, disponendoli all'interno della nuova area.

In questo esempio verrebbe generato un modulo di gioco completo, gestibile come una nuova missione. La generazione dinamica potrebbe risultare molto efficace anche per la gestione del traffico e dei personaggi che popolano le strade della città, entrambi fenomeni tipicamente a carattere dinamico e dettati da un comportamento privo di schemi preimpostati.

Il lavoro svolto in questo progetto di tesi è stato incentrato sullo sviluppo del modello generativo e sull'implementazione dei moduli atomici da integrare in tale modello. La componente adattiva inserita per modulare il grado di sfida offerto al giocatore è stata realizzata inserendo un sondaggio a conclusione del percorso di gioco. Ogni livello del percorso viene riassemblato in un nuovo livello, dove le componenti che incidono sul grado di difficoltà vengono editate in accordo al giudizio dato dal giocatore. Le procedure che regolano l'adattamento della difficoltà, in futuro potrebbero essere maggiormente integrate nel processo generativo, delineando così uno sviluppo dei livelli basato su un approccio ricorsivo : a partire dal livello iniziale, il livello successivo verrebbe generato adattando la difficoltà anticipatamente, sulla base degli eventi avvenuti nel livello precedente. Il sistema valuterebbe i due livelli come un unico modulo, dal quale generare un terzo livello. Sulla base di queste considerazioni, una possibile scelta implementativa potrebbe coinvolgere l'utilizzo di un sistema per la generazione e la compilazione dinamica di codice.

Bibliografia

1. *Aaron Reed, Learning XNA 3.0 - 2008*
2. *Mark DeLoura, Best of Game Programming Gems - 2009*
3. *http://imparandoxna.blogspot.com/*
4. *http://creators.xna.com/en-US/*
5. *http://www.oddworldforums.net/*
6. *http://www.oddworld.com/*
7. *http://www.codeplex.com/ScurvyMedia*
8. *http://www.techsmith.com/camtasia.asp*
9. *http://www.c-sharpcorner.com/*
10. *http://blogs.academicclub.org/xna/*
11. *Matteo Bittanti, L'innovazione tecnoludica - 1999*

Appendice

Codice sorgente

*Le porzioni di codice descritte nel documento di tesi riportano nel commento ad esse associato le pagine del documento a cui si riferiscono, con la dicitura **[p. numero pagina].***

*Le porzioni di codice descritte nel documento di tesi riportano nel commento ad esse associato le pagine del documento a cui si riferiscono, con la dicitura **[p. numero pagina].***

Indice Classi

```
using ...

//Classe Principale del progetto
//Gestisce l'intero workflow del gioco
namespace TheGame
{
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        VARS

        public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
            colTypes = new List<string>();
            VideoContent = new VideoContentManager(Services);
            VideoContent.RootDirectory = "Content";
        }

        //Vengono settate alcune impostazioni del sistema
        //[p.21]
        protected override void Initialize()
        {
            this.graphics.PreferredBackBufferWidth = 640;
            this.graphics.PreferredBackBufferHeight = 480;
            graphics.ApplyChanges();
            screenWidth = graphics.GraphicsDevice.Viewport.Width;
            screenHeight = graphics.GraphicsDevice.Viewport.Height;
            base.Initialize();
        }

        //Vengono caricate le risorse del gioco (sprites, suoni, video)
        //[p.22]
        protected override void LoadContent()
        {
            REPORT = new Report(this);
            MudokonFactory = new MudokonFactory(this);
            SligFactory = new SligFactory(this);
            LevelMaker vlv = new LevelMaker(this);
            Abe = new Abe(this);
            abeCONTROL = new AbeControl(Abe, this);
            foreach(Video vid in currentLevel.VIDEOS)
                vid.Play();

            spriteBatch = new SpriteBatch(GraphicsDevice);
            spriteCurrent = Abe.CurrentAnimation.Sheet;

            backRec = new Rectangle(0, 0, 640, 480);
        }

        //Chiamato a fine sondaggio
        //I riferimenti ai livelli generati vengono eliminati
        public void ReInit()
        {
            CurrentLevel = LevelMaker.lvlList[0];
            currentLevel.compatibleLevels.Clear();
            currentLevel.NextRIGHT = null;
            currentLevel.NextDOWN = null;

            Abe = new Abe(this);
            abeCONTROL = new AbeControl(Abe, this);

            DEMO

            foreach (Video vid in currentLevel.VIDEOS)
                vid.Play();
        }

        //Chiamato quando il personaggio muore
        //Riavvia il gioco dal primo livello senza eliminare i riferimenti ai     ↙
```

```
livelli generati
    public void ReStart()
    {
        CurrentLevel = LevelMaker.lvlList[0];

        Abe = new Abe(this);
        abeCONTROL = new AbeControl(Abe, this);

        DEMO
        foreach (Level lvl in REPORT.LEVELS)
            lvl.RESET();

        foreach (Video vid in currentLevel.VIDEOS)
            vid.Play();
    }

    //La memoria viene liberate dalle risorse caricate
    //[p.22]
    protected override void UnloadContent()
    {
        // TODO: Unload any non ContentManager content here
    }


    //Aggiorna lo stato del sistema
    //Frame rate impostato a 30fps
    //[p.22]
    protected override void Update(GameTime gameTime)
    {
        //Aggiorna i frame delle animazioni
        FrameUpdate

        //Rileva comandi secondari
        SystemCommand

        if (timer >= FRAME_RATE)
        {
            timer = 0f;

            //Gestione uscita dal livello corrente (Abe)
            //Se Abe esce dal livello corrente ne viene generato uno nuovo.
            ABEChangeLevel

            //Gestione uscita dal livello corrente (Mudokon)
            MudokonChangeLevel

            //Gestione uscita dal livello corrente (SligControl)
            //Se uno Slig controllato dal giocatore esce dal livello corrente↙
 ne viene generato uno nuovo.
            SligControlChangeLevel

            //Gestione uscita dal livello corrente (Slig)
            SligChangeLevel

            //Gestione Input per sistema di controllo
            ManageInputControl

        }

        if (!PAUSE & !SHOW_REPORT)//Se il gioco non è in pausa e il percorso ↙
non è finito
        {
            abeCONTROL.ROUTINE();//Sistema di controllo[pp.48,49]

            //Esegue gli automi definiti nelle rispettive classi
            SligAI

            MudokonAI

            ObjectAControl
```

```
            ////////////////////////////////////////////////
        }

        if (SHOW_REPORT)//Se il percorso è finito
            REPORT.Control();//Gestione sondaggio finale[pp.76,77]

        WindowTitle

    }

    //Disegna tutti gli elementi della scena
    //[p.22]
    protected override void Draw(GameTime gameTime)
    {
        graphics.GraphicsDevice.Clear(Color.White);

        if (!SHOW_REPORT)//Se il percorso non è finito
        {
            DrawBackground

            DrawVideos

            DrawObjectA

            DrawMudokon

            DrawAbe

            DrawSlig

            DrawObject

            DebugView
        }
        else
        {
            DrawSurvey

        }
        base.Draw(gameTime);
    }

    //Dato il livello corrente e la direzione di uscita
    //restituisce una lista contenente i livelli strutturalmente compatibili
    //[p.73]
    List<Level> GetNextCompLevels(Level lvl, string Direction)
    {
        foreach (Level comp in LevelMaker.lvlList)
        {
            if (comp.Enviorement != lvl.Enviorement)
            {
                switch (Direction)
                {
                    case "RIGHT":
                        if (lvl.BLOCCOUPRight == comp.BLOCCOUPLeft && lvl.
FLOOR2NDRight == comp.FLOOR2NDLeft && lvl.BLOCCODOWNRight == comp.
BLOCCODOWNLeft)
                            lvl.compatibleLevels.Add(new Level(comp));
                        break;

                    case "LEFT":
                        if (lvl.BLOCCOUPLeft == comp.BLOCCOUPRight && lvl.
FLOOR2NDLeft == comp.FLOOR2NDRight && lvl.BLOCCODOWNLeft == comp.
BLOCCODOWNRight)
                            lvl.compatibleLevels.Add(new Level(comp));
                        break;

                    case "DOWN":
                        int HoistsLvlDown = 0;
                        int HoistsCompDown = 0;
```

```
                            foreach (List<Collision> collGroup in lvl.          ↙
    COLLISIONGROUPS)
                                foreach (Collision coll in collGroup)
                                    if ((coll.TYPE == COLLISIONS.HOISTDESTRO ||  ↙
    coll.TYPE == COLLISIONS.HOISTSINISTRO) & coll.REC.Y == 385)
                                    {
                                        HoistsLvlDown++;
                                        foreach (List<Collision> collGroupComp in↙
     comp.COLLISIONGROUPS)
                                            foreach (Collision collComp in       ↙
    collGroupComp)
                                                if ((collComp.TYPE == COLLISIONS.↙
    HOISTDESTRO || collComp.TYPE == COLLISIONS.HOISTSINISTRO) & collComp.REC.Y ==↙
     -15 & coll.REC.X == collComp.REC.X)
HoistsCompDown++;
                                    }

                            if (HoistsLvlDown == HoistsCompDown)
                                lvl.compatibleLevels.Add(new Level(comp));
                            break;

                        case "UP":
                            int HoistsLvlUp = 0;
                            int HoistsCompUp = 0;
                            foreach (List<Collision> collGroup in lvl.          ↙
    COLLISIONGROUPS)
                                foreach (Collision coll in collGroup)
                                    if ((coll.TYPE == COLLISIONS.HOISTDESTRO ||  ↙
    coll.TYPE == COLLISIONS.HOISTSINISTRO) & coll.REC.Y == -15)
                                    {
                                        HoistsLvlUp++;
                                        foreach (List<Collision> collGroupComp in↙
     comp.COLLISIONGROUPS)
                                            foreach (Collision collComp in       ↙
    collGroupComp)
                                                if ((collComp.TYPE == COLLISIONS.↙
    HOISTDESTRO || collComp.TYPE == COLLISIONS.HOISTSINISTRO) & collComp.REC.Y ==↙
     385 & coll.REC.X == collComp.REC.X)
HoistsCompUp++;
                                    }

                            if (HoistsCompUp == HoistsLvlUp)
                                lvl.compatibleLevels.Add(new Level(comp));
                            break;

                        case "DOOR":
                            foreach (List<Collision> collGroup in comp.         ↙
    COLLISIONGROUPS)
                            {
                                foreach (Collision coll in collGroup)
                                {
                                    if (coll.TYPE == COLLISIONS.DOOR)
                                    {
                                        Level newCopy = new Level(comp);
                                        lvl.compatibleLevels.Add(newCopy);
                                    }
                                }
                            }
                            break;
                    }
                }
            }
            return lvl.compatibleLevels;
        }

        GetCompLvls2

        //Dato il livello corrente e la lista dei livelli compatibili
        //restituisce un livello con grado di difficoltà opportuno
        //[p.74]
```

```
        Level getNextLvl(Level lvl,List<Level> compLvls)
        {
            Random random = new Random();
            List<Level> lvlsNext = new List<Level>();
            int rand;
            int Range = 1;
            bool found = false;
            int lvlsDifference;
            Level NextLevel = currentLevel;


            if (NUMLVLGEN < MAXlvls)
            {
                while (!found)
                {
                    Range++;
                    for (int i = 0; i < compLvls.Count; i++)
                    {
                        lvlsDifference = Math.Abs(lvl.difficult - compLvls[i].
    difficult);

                        if (lvlsDifference <= Range)
                        {
                            lvlsNext.Add(compLvls[i]);
                            found = true;

                            foreach (Level lvlVisited in REPORT.LEVELS)
                            {
                                if (i < compLvls.Count && lvlVisited.Enviorement
    == compLvls[i].Enviorement)
                                {
                                    lvlsNext.Remove(compLvls[i]);
                                    compLvls.Remove(compLvls[i]);
                                    found = false;
                                }
                            }
                        }
                    }
                }
                rand = random.Next(0, lvlsNext.Count);
                NextLevel = lvlsNext[rand];
                NUMLVLGEN++;
                REPORT.LEVELS.Add(NextLevel);
            }
            else
                SHOW_REPORT = true;

            compLvls.Clear();

            return NextLevel;
        }

        public void ResetVideos(Character currentChar, Level nextLvl)
        {
            foreach (Video vid in currentLevel.VIDEOS)
                vid.Stop();

            currentLevel = nextLvl;
            currentChar.CurrentLevel = currentLevel;

            foreach (Video vid in currentLevel.VIDEOS)
                vid.Play();
        }

        Attribute

    }
}
```

```
using ...

namespace TheGame
{
    //Gestisce grafica ed effetti sonori del personaggio principale
    public class Abe : Character
    {
        VARS

        public Abe(Game1 game)
        {
            this.game = game;
            MUDOKONS.Add(this);

            POSITION = 0;

            ////////////////////////////////////////////////////////////////////

            Sound_PASS = game.Content.Load<SoundEffect>("Sound\\PASS");
            Sound_TURN = game.Content.Load<SoundEffect>("Sound\\TURN");
            Sound_JUMP = game.Content.Load<SoundEffect>("Sound\\JUMP");
            Sound_RUNSTOP = game.Content.Load<SoundEffect>("Sound\\RUNSTOP");
            Sound_SNEAK = game.Content.Load<SoundEffect>("Sound\\SNEAK");
            Sound_IMPACT = game.Content.Load<SoundEffect>("Sound\\IMPACT");
            Sound_HELLO = game.Content.Load<SoundEffect>("Sound\\HELLO");
            Sound_FOLLOW = game.Content.Load<SoundEffect>("Sound\\FOLLOW");
            Sound_WAIT = game.Content.Load<SoundEffect>("Sound\\WAIT");
            Sound_ROLL = game.Content.Load<SoundEffect>("Sound\\ROLL");
            Sound_OK = game.Content.Load<SoundEffect>("Sound\\OK");
            Sound_COMMANDJUMP = game.MudokonFactory.Sound_COMMANDJUMP;
            Sound_COMMANDRUNJUMP = game.MudokonFactory.Sound_COMMANDRUNJUMP;
            Sound_OMM = game.Content.Load<SoundEffect>("Sound\\OMM");

            ////////////////////////////////////////////////////////////////////

            BOMBDEACTIVATE = new Animation(game.MudokonFactory.BombDeactivate_R, ↙
    game.MudokonFactory.BombDeactivate_L, AnimateRate, 26, "BOMBDEACTIVATE", 0, 0↙
    , -1, 10, 10);
            BOMBDEACTIVATEFAIL = new Animation(game.MudokonFactory.              ↙
    BombDeactivateFail_R, game.MudokonFactory.BombDeactivateFail_L, AnimateRate, ↙
    21, "BOMBDEACTIVATEFAIL", 0, 0, -1, 10, 10);

            BOMBEXPLODE = new Animation(game.MudokonFactory.BombExplode_C, game. ↙
    MudokonFactory.BombExplode_C, AnimateRate, 13, "BOMBEXPLODE", 0, 0, -1, 10,  ↙
    10);

            WAITcROUCH = new Animation(game.MudokonFactory.WaitCrouch_R, game.   ↙
    MudokonFactory.WaitCrouch_L, 1000 / 6, 6, "WAITcROUCH", 0, 0, -1, 10, 10);

            WAIT = new Animation(game.MudokonFactory.Wait_R, game.MudokonFactory.↙
    Wait_L, 1000 / 6, 6, "WAIT", 0, 0, -1, 15, 15);
            TURN = new Animation(game.MudokonFactory.Turn_L, game.MudokonFactory.↙
    Turn_R, AnimateRate, 12, "TURN", new Sound(Sound_TURN, 2, 0.05f), 0, 0, -1,  ↙
    10, 10);

            STOP = new Animation(game.MudokonFactory.Stop_R, game.MudokonFactory.↙
    Stop_L, AnimateRate, 4, "STOP", new Sound(Sound_PASS, 4), 0, 0, -1, 43, 43);
            PASS_1 = new Animation(game.MudokonFactory.Walk_1PASS_R, game.       ↙
    MudokonFactory.Walk_1PASS_L, AnimateRate, 7, "PASS_1", new Sound(Sound_PASS, ↙
    4), 1, 0, 6, 0, 43);
            PASS_2 = new Animation(game.MudokonFactory.Walk_2PASS_R, game.       ↙
    MudokonFactory.Walk_2PASS_L, AnimateRate, 9, "PASS_2", new Sound(Sound_PASS, ↙
    6), 1, 0, 8, 43, 86);
            PASS_3 = new Animation(game.MudokonFactory.Walk_3PASS_R, game.       ↙
    MudokonFactory.Walk_3PASS_L, AnimateRate, 9, "PASS_3", new Sound(Sound_PASS, ↙
    6), 1, 0, 8, 86, 129);

            BLOCK = new Animation(game.MudokonFactory.Block_R, game.             ↙
    MudokonFactory.Block_L, AnimateRate, 21, "BLOCK", new Sound(Sound_RUNSTOP, 9)↙
    , 0, 0, -1, 20, 20);
```

```
        SNEAKsTOP = new Animation(game.MudokonFactory.SneakStop_R, game.
MudokonFactory.SneakStop_L, AnimateRate, 4, "SNEAKsTOP", 0, 0, -1, 53, 53);
        SNEAK_1 = new Animation(game.MudokonFactory.Sneak1_R, game.
MudokonFactory.Sneak1_L, AnimateRate, 6, "SNEAK_1", new Sound(Sound_SNEAK, 3)
, 1, 0, 5, 10, 53);
        SNEAK_2 = new Animation(game.MudokonFactory.Sneak2_R, game.
MudokonFactory.Sneak2_L, AnimateRate, 10, "SNEAK_2", new Sound(Sound_SNEAK,
7), 1, 0, 9, 53, 96);
        SNEAK_3 = new Animation(game.MudokonFactory.Sneak3_R, game.
MudokonFactory.Sneak3_L, AnimateRate, 10, "SNEAK_3", new Sound(Sound_SNEAK,
7), 1, 0, 9, 96, 139);

        RUN_1 = new Animation(game.MudokonFactory.Run1_R, game.MudokonFactory
.Run1_L, AnimateRate, 4, "RUN_1", 2, 0, 3, 35, 35);
        RUN_2 = new Animation(game.MudokonFactory.Run2_R, game.MudokonFactory
.Run2_L, AnimateRate, 2, "RUN_2", new Sound(Sound_PASS, 2), 2, 0, 4, 35, 78);
        RUN_3 = new Animation(game.MudokonFactory.Run3_R, game.MudokonFactory
.Run3_L, AnimateRate, 3, "RUN_3", 2, 0, 9, 78, 121);
        RUN_4 = new Animation(game.MudokonFactory.Run4_R, game.MudokonFactory
.Run4_L, AnimateRate, 5, "RUN_4", new Sound(Sound_PASS, 5), 2, 0, 9, 121,
164);
        RUN_5 = new Animation(game.MudokonFactory.Run5_R, game.MudokonFactory
.Run5_L, AnimateRate, 6, "RUN_5", 2, 0, 9, 164, 207);
        RUNsTOP_1 = new Animation(game.MudokonFactory.RunStop1_R, game.
MudokonFactory.RunStop1_L, AnimateRate, 8, "RUNSTOP_1", new Sound
(Sound_RUNSTOP, 3), 1, 0, 7, 121, 164);
        RUNsTOP_2 = new Animation(game.MudokonFactory.RunStop2_R, game.
MudokonFactory.RunStop2_L, AnimateRate, 8, "RUNsTOP_S", 1, 0, 7, 164, 207);
        RUNJUMP1 = new Animation(game.MudokonFactory.RunJump1_R, game.
MudokonFactory.RunJump1_L, AnimateRate, 2, "RUNJUMP1", 1, 0, 1, 0, 43,0,0);
        RUNJUMP2 = new Animation(game.MudokonFactory.RunJump2_R, game.
MudokonFactory.RunJump2_L, AnimateRate, 5, "RUNJUMP2", 1, 0, 7, 43, 86,0,50);
        RUNJUMP3 = new Animation(game.MudokonFactory.RunJump3_R, game.
MudokonFactory.RunJump3_L, AnimateRate, 4, "RUNJUMP3", 1, 0, 7, 86, 129,50,
50);
        RUNJUMP4 = new Animation(game.MudokonFactory.RunJump4_R, game.
MudokonFactory.RunJump4_L, AnimateRate, 5, "RUNJUMP4", 1, 0, 7, 129, 172,50,
0);

        JUMP_1 = new Animation(game.MudokonFactory.Jump1_R, game.
MudokonFactory.Jump1_L, AnimateRate, 10, "JUMP_1", 1, 0, 3, 18, 61);
        JUMP_2 = new Animation(game.MudokonFactory.Jump2_R, game.
MudokonFactory.Jump2_L, AnimateRate, 3, "JUMP_2", 1, 0, 3, 61, 104);
        JUMP_3 = new Animation(game.MudokonFactory.Jump3_R, game.
MudokonFactory.Jump3_L, AnimateRate, 3, "JUMP_3", 1, 0, 3, 104, 147);
        JUMP_4 = new Animation(game.MudokonFactory.Jump4_R, game.
MudokonFactory.Jump4_L, AnimateRate, 6, "JUMP_4", new Sound(Sound_JUMP, 1), 0
, 0, -1, 147, 147);

        ROLL1 = new Animation(game.MudokonFactory.Roll1_R, game.
MudokonFactory.Roll1_L, AnimateRate, 5, "ROLL1", new Sound(Sound_ROLL, 2), 0,
 0, -1, 10, 53);
        ROLL2 = new Animation(game.MudokonFactory.Roll2_R, game.
MudokonFactory.Roll2_L, AnimateRate, 3, "ROLL2", new Sound(Sound_ROLL, 2), 0,
 0, -1, 53, 96);
        ROLL3 = new Animation(game.MudokonFactory.Roll3_R, game.
MudokonFactory.Roll3_L, AnimateRate, 3, "ROLL3", new Sound(Sound_ROLL, 2), 0,
 0, -1, 96, 139);
        WAITcROUCH = new Animation(game.MudokonFactory.WaitCrouch_R, game.
MudokonFactory.WaitCrouch_L, 1000 / 6, 6, "WAITcROUCH", 0, 0, -1, 10, 10);
        TURNdOWN = new Animation(game.MudokonFactory.TurnDown_L, game.
MudokonFactory.TurnDown_R, AnimateRate, 8, "TURNdOWN", new Sound(Sound_TURN,
2, 0.05f), 0, 0, -1, 10, 10);
        CROUCH = new Animation(game.MudokonFactory.Crouch_R, game.
MudokonFactory.Crouch_L, AnimateRate, 9, "CROUCH", 0, 0, -1, 10, 10);
        STANDUP = new Animation(game.MudokonFactory.StandUp_R, game.
MudokonFactory.StandUp_L, AnimateRate, 9, "STANDUP", 0, 0, -1, 10, 10);

        JUMPDOWN = new Animation(game.MudokonFactory.JumpDown_R, game.
```

```
MudokonFactory.JumpDown_L, AnimateRate, 18, "JUMOdOWN", new Sound(Sound_JUMP, 7), 0, -1, 10, 15, 15, 50, 0);
        JUMPUP = new Animation(game.MudokonFactory.JumpUp_R, game.MudokonFactory.JumpUp_L, AnimateRate, 14, "JUMPuP", new Sound(Sound_PASS, 13, true, game, this), 0, 1, 13, 15, 15, 0, 50);
        HUNG = new Animation(game.MudokonFactory.Hung_R, game.MudokonFactory.Hung_L, 1000 / 6, 6, "HUNG", 0, 0, -1, 15, 15, 60, 60);
        HUNGLONG = new Animation(game.MudokonFactory.HungLong_R, game.MudokonFactory.HungLong_L, AnimateRate, 15, "HUNGLONG", 0, 0, -1, 20, 20, 60, 60);

        HOISTUP = new Animation(game.MudokonFactory.HoistUp_R, game.MudokonFactory.HoistUp_L, AnimateRate, 23, "HOISTUP", 0, 0, -1, 15, 15, 0, 150);
        HOISTDOWN = new Animation(game.MudokonFactory.HoistDown_R, game.MudokonFactory.HoistDown_L, AnimateRate, 23, "HOISTDOWN", 0, 0, -1, 15, 15, 150, 0);

        //DOWNPLANE = new Animation(game.MudokonFactory.DownPlane_R, game.MudokonFactory.DownPlane_L, AnimateRate, 21, "DOWNPLANE", new Sound (Sound_IMPACT, 14), 0, 0, -1, 47, 90, 200, 0);
        GODOWN = new Animation(game.MudokonFactory.GoDown_R, game.MudokonFactory.GoDown_L, AnimateRate, 9, "GODOWN", 1, -2, 5, 12, 55, 200, 50);
        PLANE = new Animation(game.MudokonFactory.Plane_R, game.MudokonFactory.Plane_L, AnimateRate, 12, "PLANE", new Sound(Sound_IMPACT, 3), 1, 0, 2, 55, 98, 50, 0);

        HELLO = new Animation(game.MudokonFactory.Wait_R, game.MudokonFactory.Wait_L, 1000 / 6, 6, "HELLO", new Sound(Sound_HELLO, 1), 0, 0, -1, 15, 15);
        FOLLOW = new Animation(game.MudokonFactory.Wait_R, game.MudokonFactory.Wait_L, 1000 / 6, 6, "FOLLOW", new Sound(Sound_FOLLOW, 1), 0, 0, -1, 15, 15);
        WAITME = new Animation(game.MudokonFactory.Wait_R, game.MudokonFactory.Wait_L, 1000 / 6, 6, "WAITME", new Sound(Sound_WAIT, 1), 0, 0, -1, 15, 15);
        COMMANDJUMP = new Animation(game.MudokonFactory.Wait_R, game.MudokonFactory.Wait_L, 1000 / 6, 6, "COMMANDJUMP", new Sound (Sound_COMMANDJUMP, 1), 0, 0, -1, 15, 15);
        COMMANDRUNJUMP = new Animation(game.MudokonFactory.Wait_R, game.MudokonFactory.Wait_L, 1000 / 6, 6, "COMMANDRUNJUMP", new Sound (Sound_COMMANDRUNJUMP, 1), 0, 0, -1, 15, 15);


        KO = new Animation(game.MudokonFactory.Ko_R, game.MudokonFactory.Ko_L, AnimateRate, 13, "KO", new Sound(Sound_IMPACT, 3), 0, 0, -1, 46, 46);
        GETUP = new Animation(game.MudokonFactory.GetUp_R, game.MudokonFactory.GetUp_L, AnimateRate, 13, "GETUP", 0, 0, -1, 46, 46);

        DIE = new Animation(game.MudokonFactory.Die_R, game.MudokonFactory.Die_L, AnimateRate, 28, "DIE", 0, 0, -1, 50, 50);

        PULLEVER = new Animation(game.MudokonFactory.PulLever_R, game.MudokonFactory.PulLever_L, AnimateRate, 19, "PULLEVER", 0, 0, -1, 6, 6);

        OMM = new Animation(game.MudokonFactory.Omm_R, game.MudokonFactory.Omm_L, AnimateRate, 16, "OMM", new Sound(Sound_OMM, 1), 0, 0, -1, 10, 10);
        NOTCONTROL = new Animation(game.MudokonFactory.Omm_R, game.MudokonFactory.Omm_L, AnimateRate, 16, "NOTCONTROL", 0, 0, -1, 10, 10);

        ENTERDOOR = new Animation(game.MudokonFactory.EnterDoor_R, game.MudokonFactory.EnterDoor_L, 1000 / 15, 12, "ENTERDOOR", 0, 0, -1, 15, 15);
        EXITDOOR = new Animation(game.MudokonFactory.ExitDoor_R, game.MudokonFactory.ExitDoor_L, 1000 / 15, 11, "EXITDOOR", 0, 0, -1, 15, 15);


        CurrentLevel = game.CurrentLevel;
        NoNOISE = new Rectangle(-50, -50, 0, 0);
        NOISE = NoNOISE;
        CurrentAnimation = WAIT;
```

```
        canmove = true;
        Direction = 1;
        color = Color.White;
        aBE_BBOX_PNG = game.Content.Load<Texture2D>("Sprite_Sheet\\Abe\\
ABE_BBOX");
        aBE_NOISE = game.Content.Load<Texture2D>("Sprite_Sheet\\Abe\\NoiseAbe
");
        DrawingRec = new Rectangle(-18 + 43 * 4, 400 - CurrentAnimation.
SpriteHeight, CurrentAnimation.SpriteWidth, CurrentAnimation.SpriteHeight);
        boundingBox = new Rectangle(6 + 43*4, 285, 44, 150);//42
    }

    //Gestisce la spritesheet dell'animazione corrente.
    //Definisce quale frame della spritesheet deve essere disegnato.
    public void AnimateAbe(Animation anim)
    {
        if (anim.CurrentFrame == 0)
        {
            PrevAnimation = CurrentAnimation;
            PrevAnimation.Reset();
            CurrentAnimation = anim;

            if (Direction == 1)
            {
                DrawingRec.X = DrawingRec.X + (PrevAnimation.xEND -
CurrentAnimation.xSTART);
            }
            else
            {
                DrawingRec.X = DrawingRec.X + ((PrevAnimation.SpriteLWidth -
PrevAnimation.xEND) - (CurrentAnimation.SpriteLWidth - CurrentAnimation.
xSTART));
            }

            DrawingRec.Y = DrawingRec.Y - (PrevAnimation.yEND -
CurrentAnimation.ySTART) + (PrevAnimation.SpriteHeight - CurrentAnimation.
SpriteHeight);

            if (CurrentAnimation == TURN || CurrentAnimation == TURNdOWN)
            {
                Direction = Direction * -1;
            }

            if (Direction == 1)
            {
                anim.SpriteWidth = anim.SpriteRWidth;
                anim.SpriteHeight = anim.SpriteRHeight;
                anim.Sheet = anim.SheetR;
            }
            else
            {
                anim.SpriteWidth = anim.SpriteLWidth;
                anim.SpriteHeight = anim.SpriteLHeight;
                anim.Sheet = anim.SheetL;
            }

            DrawingRec.Width = anim.SpriteWidth;
            DrawingRec.Height = anim.SpriteHeight;

            this.game.spriteCurrent = anim.Sheet;
        }

        if (anim.Time > anim.AnimateRate)
        {
            if (anim.CurrentFrame == 1)
            {
                boundingBox.X += (anim.xEND - anim.xSTART) * Direction;
                boundingBox.Y = DrawingRec.Bottom - (anim.yEND + boundingBox.
Height);
            }
```

```
                anim.Time = 0f;
                if (anim.Effect != null && anim.CurrentFrame == anim.Effect.
    FrameToPlay)
                    anim.Effect.Play();

                if (anim.CurrentFrame == anim.FrameCount)
                {
                    anim.LastFrameDone = true;
                }
                else
                    anim.LastFrameDone = false;

                if (anim.CurrentFrame < anim.FrameCount)
                {
                    anim.SourceRect = new Rectangle(anim.CurrentFrame * anim.
    SpriteWidth, 0, anim.SpriteWidth, anim.SpriteHeight);
                    this.game.sourceRect = anim.SourceRect;
                    anim.Forward();
                }
            }
        }
        Attribute
    }
}
```

```
//AbeControl definisce l'automa per la gestione del sistema
//di controllo del personaggio principale
//[pp.43]
public class AbeControl
{
    VARS

    public AbeControl() { }
    public AbeControl(Abe Abe, Game1 game)
    {
        this.game = game;
        this.Abe = Abe;

        PRECONDITIONS = new PRECONDITIONSa(this);
        ACTIONS = new ACTIONSa(this);

        DIED = new STATEa("DIED", this);
        WAIT = new STATEa("WAIT", this);
        PULLEVER = new STATEa("PULLEVER", this);
        DEACTIVATEBOMB = new STATEa("DEACTIVATEBOMB", this);
        ACTIVATEBOMB = new STATEa("ACTIVATEBOMB", this);
        DEACTIVATEBOMBFAIL = new STATEa("DEACTIVATEBOMBFAIL", this);
        WALK = new STATEa("WALK", this);
        SNEAK = new STATEa("SNEAK", this);
        JUMP = new STATEa("JUMP", this);
        HUNG = new STATEa("HUNG", this);
        HUNGLONG = new STATEa("HUNGLONG", this);
        RUN = new STATEa("RUN", this);
        RUNSTART = new STATEa("RUNSTART", this);
        RUNSTOP = new STATEa("RUNSTOP", this);
        RUNJUMP = new STATEa("RUNJUMP", this);
        WAITCROUCH = new STATEa("WAITCROUCH", this);
        ROLL = new STATEa("ROLL", this);
        GODOWN = new STATEa("GODOWN", this);
        OMM = new STATEa("OMM", this);
        NOTCONTROL = new STATEa("NOTCONTROL", this);
        ENTERDOOR = new STATEa("ENTERDOOR", this);
        EXITDOOR = new STATEa("EXITDOOR", this);


        WAIT.LinkTO(GODOWN).addPRECONDITION(PRECONDITIONS.GODOWN);
        WAIT.LinkTO(PULLEVER).addPRECONDITION(PRECONDITIONS.LEVA);
        WAIT.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.HELLO, ACTIONS.HELLO)↙
;
        WAIT.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.FOLLOW, ACTIONS.     ↙
FOLLOW);
        WAIT.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.WAITME, ACTIONS.     ↙
WAITME);
        WAIT.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.COMMANDJUMP, ACTIONS.↙
COMMANDJUMP);
        WAIT.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.COMMANDRUNJUMP,      ↙
ACTIONS.COMMANDRUNJUMP);
        WAIT.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.AND(PRECONDITIONS.   ↙
BLOCK, PRECONDITIONS.RIGHT), ACTIONS.BLOCK);
        WAIT.LinkTO(HUNG).addPRECONDITION(PRECONDITIONS.AND(PRECONDITIONS.   ↙
DOWN,PRECONDITIONS.HOIST), ACTIONS.HOISTDOWN);
        WAIT.LinkTO(HUNG).addPRECONDITION(PRECONDITIONS.AND(PRECONDITIONS.   ↙
DOWN, PRECONDITIONS.HOISTTURN), ACTIONS.AND(ACTIONS.TURN,ACTIONS.HOISTDOWN));
        WAIT.LinkTO(WAITCROUCH).addPRECONDITION(PRECONDITIONS.DOWN, ACTIONS. ↙
CROUCH);
        WAIT.LinkTO(WALK).addPRECONDITION(PRECONDITIONS.RIGHT, ACTIONS.PASS1)↙
;
        WAIT.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.AND(PRECONDITIONS.   ↙
toSNEAK, PRECONDITIONS.BLOCK), ACTIONS.BLOCK);
        WAIT.LinkTO(SNEAK).addPRECONDITION(PRECONDITIONS.toSNEAK, ACTIONS.   ↙
SNEAK1);
        WAIT.LinkTO(RUNSTART).addPRECONDITION(PRECONDITIONS.toRUN, ACTIONS.  ↙
RUN1);
        WAIT.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.toTURN, ACTIONS.TURN)↙
;
```

```
        WAIT.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.AND(PRECONDITIONS.
toJUMP, PRECONDITIONS.BLOCK), ACTIONS.AND(ACTIONS.KO, ACTIONS.GETUP));
        WAIT.LinkTO(ENTERDOOR).addPRECONDITION(PRECONDITIONS.ENTERDOOR);
        WAIT.LinkTO(JUMP).addPRECONDITION(PRECONDITIONS.toJUMP, ACTIONS.
JUMP1);
        WAIT.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.AND(PRECONDITIONS.AND
(PRECONDITIONS.notAPPIGLIO,PRECONDITIONS.NOTDOOR), PRECONDITIONS.UP), ACTIONS
.AND(ACTIONS.JUMPUP, ACTIONS.JUMPDOWN));
        WAIT.LinkTO(HUNG).addPRECONDITION(PRECONDITIONS.AND(PRECONDITIONS.AND
(PRECONDITIONS.APPIGLIO, PRECONDITIONS.NOTDOOR), PRECONDITIONS.UP), ACTIONS.
JUMPUP);
        WAIT.LinkTO(HUNG).addPRECONDITION(PRECONDITIONS.AND(PRECONDITIONS.AND
(PRECONDITIONS.APPIGLIOTURN, PRECONDITIONS.NOTDOOR), PRECONDITIONS.UP),
ACTIONS.AND(ACTIONS.TURN, ACTIONS.JUMPUP));
        WAIT.LinkTO(OMM).addPRECONDITION(PRECONDITIONS.ZERO);

        ENTERDOOR.LinkTO(EXITDOOR).addPRECONDITION(PRECONDITIONS.DOORENTERED)
;
        EXITDOOR.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.ENDSTATE);

        PULLEVER.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.ENDLEVA);

        OMM.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.SHOCK,ACTIONS.AND
(ACTIONS.KO,ACTIONS.GETUP));
        OMM.LinkTO(NOTCONTROL).addPRECONDITION(PRECONDITIONS.AND
(PRECONDITIONS.ENDSTATE,PRECONDITIONS.SLIGFORCONTROL));
        OMM.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.ZEROUP);

        WALK.LinkTO(DIED).addPRECONDITION(PRECONDITIONS.BOMB,ACTIONS.EXPLODE)
;
        WALK.LinkTO(GODOWN).addPRECONDITION(PRECONDITIONS.GODOWN);
        WALK.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.STOP, ACTIONS.STOP);
        WALK.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.BLOCK, ACTIONS.BLOCK)
;
        WALK.LinkTO(SNEAK).addPRECONDITION(PRECONDITIONS.toSNEAK);
        WALK.LinkTO(RUNSTART).addPRECONDITION(PRECONDITIONS.toRUN);

        SNEAK.LinkTO(GODOWN).addPRECONDITION(PRECONDITIONS.GODOWN);
        SNEAK.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.toSTOPSNEAK, ACTIONS
.STOPSNEAK);
        SNEAK.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.BLOCK, ACTIONS.
BLOCK);
        SNEAK.LinkTO(WALK).addPRECONDITION(PRECONDITIONS.RIGHT);

        JUMP.LinkTO(GODOWN).addPRECONDITION(PRECONDITIONS.AND(PRECONDITIONS.
ENDSTATE,PRECONDITIONS.GODOWN));
        JUMP.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.ENDSTATE, ACTIONS.
JUMP4);
        JUMP.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.BLOCK, ACTIONS.AND
(ACTIONS.KO, ACTIONS.GETUP));

        GODOWN.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.AND(PRECONDITIONS.
NOTHOIST,PRECONDITIONS.ENDSTATE),ACTIONS.PLANE);
        GODOWN.LinkTO(HUNG).addPRECONDITION(PRECONDITIONS.AND(PRECONDITIONS.
HOIST, PRECONDITIONS.ENDSTATE), ACTIONS.HUNGLONG);

        HUNGLONG.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.UP, ACTIONS.
HOISTUP);
        HUNGLONG.LinkTO(HUNG).addPRECONDITION(PRECONDITIONS.ENDSTATE);

        HUNG.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.DOWN, ACTIONS.
JUMPDOWN);
        HUNG.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.UP, ACTIONS.HOISTUP);

        RUNSTART.LinkTO(GODOWN).addPRECONDITION(PRECONDITIONS.GODOWN);
        RUNSTART.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.BLOCK, ACTIONS.
AND(ACTIONS.KO, ACTIONS.GETUP));
        RUNSTART.LinkTO(RUN).addPRECONDITION(PRECONDITIONS.AND(PRECONDITIONS.
toRUN, PRECONDITIONS.ENDSTATE));
        RUNSTART.LinkTO(RUNSTOP).addPRECONDITION(PRECONDITIONS.AND
```

```
(PRECONDITIONS.ENDSTATE,PRECONDITIONS.toSTOPRUN));
        RUNSTART.LinkTO(RUNSTOP).addPRECONDITION(PRECONDITIONS.AND
(PRECONDITIONS.ENDSTATE, PRECONDITIONS.STOP));
        RUNSTART.LinkTO(RUNJUMP).addPRECONDITION(PRECONDITIONS.toJUMP);

        RUN.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.BLOCK, ACTIONS.AND
(ACTIONS.KO, ACTIONS.GETUP));
        RUN.LinkTO(GODOWN).addPRECONDITION(PRECONDITIONS.GODOWN);
        RUN.LinkTO(WALK).addPRECONDITION(PRECONDITIONS.AND(PRECONDITIONS.
RIGHT, PRECONDITIONS.RUNOK));
        RUN.LinkTO(RUNSTOP).addPRECONDITION(PRECONDITIONS.AND(PRECONDITIONS.
toSTOPRUN, PRECONDITIONS.RUNOK));
        RUN.LinkTO(RUNSTOP).addPRECONDITION(PRECONDITIONS.AND(PRECONDITIONS.
STOP, PRECONDITIONS.RUNOK));
        RUN.LinkTO(RUNJUMP).addPRECONDITION(PRECONDITIONS.toJUMP);

        RUNJUMP.LinkTO(HUNGLONG).addPRECONDITION(PRECONDITIONS.AND
(PRECONDITIONS.HOIST,PRECONDITIONS.NOTGODOWN),ACTIONS.HUNGLONG);
        RUNJUMP.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.BLOCK, ACTIONS.AND
(ACTIONS.KO, ACTIONS.GETUP));
        RUNJUMP.LinkTO(RUNSTOP).addPRECONDITION(PRECONDITIONS.AND
(PRECONDITIONS.ENDSTATE, PRECONDITIONS.toSTOPRUN),ACTIONS.STOPRUNJUMP);
        RUNJUMP.LinkTO(RUN).addPRECONDITION(PRECONDITIONS.AND(PRECONDITIONS.
ENDSTATE, PRECONDITIONS.toRUN), ACTIONS.STOPRUNJUMP);
        //RUNJUMP.LinkTO(RUNSTOP).addPRECONDITION(PRECONDITIONS.ENDSTATE,
ACTIONS.STOPRUNJUMP);

        RUNSTOP.LinkTO(GODOWN).addPRECONDITION(PRECONDITIONS.GODOWN);
        RUNSTOP.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.ENDSTATE);

        RUNSTOP.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.BLOCK, ACTIONS.AND
(ACTIONS.KO, ACTIONS.GETUP));

        WAITCROUCH.LinkTO(DEACTIVATEBOMB).addPRECONDITION(PRECONDITIONS.
BOMBOK);
        WAITCROUCH.LinkTO(DEACTIVATEBOMB).addPRECONDITION(PRECONDITIONS.
BOMBFAIL);
        WAITCROUCH.LinkTO(ACTIVATEBOMB).addPRECONDITION(PRECONDITIONS.
BOMBACTIVATE);
        WAITCROUCH.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.AND
(PRECONDITIONS.BLOCK, PRECONDITIONS.RIGHT), ACTIONS.AND(ACTIONS.KO, ACTIONS.
GETUP));
        WAITCROUCH.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.UP, ACTIONS.
STANDUP);
        WAITCROUCH.LinkTO(HUNG).addPRECONDITION(PRECONDITIONS.AND
(PRECONDITIONS.DOWN, PRECONDITIONS.HOIST), ACTIONS.HOISTDOWN);
        WAITCROUCH.LinkTO(ROLL).addPRECONDITION(PRECONDITIONS.RIGHT, ACTIONS.
ROLL1);
        WAITCROUCH.LinkTO(WAITCROUCH).addPRECONDITION(PRECONDITIONS.toTURN,
ACTIONS.TURNDOWN);

        ACTIVATEBOMB.LinkTO(WAITCROUCH).addPRECONDITION(PRECONDITIONS.
ENDBOMB);

        DEACTIVATEBOMB.LinkTO(DIED).addPRECONDITION(PRECONDITIONS.AND
(PRECONDITIONS.ENDSTATE,PRECONDITIONS.BOMBACTIVE));
        DEACTIVATEBOMB.LinkTO(WAITCROUCH).addPRECONDITION(PRECONDITIONS.
ENDSTATE);

        ROLL.LinkTO(GODOWN).addPRECONDITION(PRECONDITIONS.GODOWN);
        ROLL.LinkTO(WAITCROUCH).addPRECONDITION(PRECONDITIONS.STOP);
        ROLL.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.AND(PRECONDITIONS.
BLOCK,PRECONDITIONS.RIGHT), ACTIONS.AND(ACTIONS.KO, ACTIONS.GETUP));
        ROLL.LinkTO(SNEAK).addPRECONDITION(PRECONDITIONS.toSNEAK);
        ROLL.LinkTO(RUN).addPRECONDITION(PRECONDITIONS.toRUN);

        CURRENT_STATE = WAIT;
        PREV_STATE = CURRENT_STATE;

    }
```

```
    //Esegue l'automa definito nel costruttore di AbeControl
    //Viene chiamato nel metodo Update della classe Game1
    //[pp.48,49]
    public void ROUTINE()
    {
        //if (CURRENT_STATE.name != "DIED")
        //{
        ColorMask
            //La transizione da questi stati può avvenire in qualsiasi momento
            if (CURRENT_STATE == WAIT || CURRENT_STATE == HUNG || CURRENT_STATE == HUNGLONG || CURRENT_STATE == WAITCROUCH)
            {
                HandleStateTransition
            }
            else//Tutti gli altri stati esigono che tutte le animazione contenute siano state eseguite
            {
                //L'arco è stato percorso, esegue lo stato corrente
                if (FINISHEDTransition)
                {
                    if (FINISHEDExecution)
                        precond = CURRENT_STATE.TryChangeState();
                    if (precond != null)
                        FINISHEDExecution = true;
                    else
                        FINISHEDExecution = CURRENT_STATE.Execute();
                }
                //Tutte le animazione sono state seguite
                if (FINISHEDExecution)
                {
                    CURRENT_STATE.AnimsTODO[CURRENT_STATE.CURRENT_ANIM].Reset();
                    PREV_STATE = CURRENT_STATE;
                    if (FINISHEDTransition & precond == null)
                        precond = CURRENT_STATE.TryChangeState();//Controllo precondizioni

                    if (precond != null)//Almeno una precondizione è stata verificata
                    {
                        if (precond.ACTION != null)//Se una precondizione ha associata un'azione
                            FINISHEDTransition = precond.ACTION.DO();//L'azione viene eseguita
                        if (FINISHEDTransition)
                        {
                            CURRENT_STATE = precond.ARC.TO;//L'automa transita nello stato successivo
                            CURRENT_STATE.ENDSTATE = false;
                            precond = null;
                        }
                    }
                }
            }
        //}
    }
}
```

```
//Definisce gli stati dell'automa
    //Ogni stato contiene una o più animazioni da eseguire
    //[p.45]
    public class STATEa
    {
        public bool ENDSTATE;
        public List<ARCa> ARCS;
        public List<Animation> AnimsTODO;
        public int CURRENT_ANIM = 0;
        public string CURRENTACTION = "null";
        public string name;
        public AbeControl Control;
        public STATEa(string name, AbeControl Control)
        {
            this.Control = Control;
            this.name = name;
            ENDSTATE = false;
            AnimsTODO = new List<Animation>();
            ARCS = new List<ARCa>();

            if (name == "ENTERDOOR")
            {
                AnimsTODO.Add(Control.Abe.ENTERDOOR);
            }
            if (name == "EXITDOOR")
            {
                AnimsTODO.Add(Control.Abe.EXITDOOR);
                //AnimsTODO.Add(Control.Abe.WAIT);
            }
            if (name == "PULLEVER")
            {
                AnimsTODO.Add(Control.Abe.PULLEVER);
            }
            if (name == "ACTIVATEBOMB")
            {
                AnimsTODO.Add(Control.Abe.BOMBDEACTIVATE);
            }
            if (name == "DEACTIVATEBOMB")
            {
                AnimsTODO.Add(Control.Abe.BOMBDEACTIVATE);
            }
            if (name == "DEACTIVATEBOMBFAIL")
            {
                AnimsTODO.Add(Control.Abe.BOMBDEACTIVATEFAIL);
            }
            if (name == "WAIT")
            {
                AnimsTODO.Add(Control.Abe.WAIT);
            }
            if (name == "WAITCROUCH")
            {
                AnimsTODO.Add(Control.Abe.WAITcROUCH);
            }
            if (name == "HUNG")
            {
                AnimsTODO.Add(Control.Abe.HUNG);
            }
            if (name == "HUNGLONG")
            {
                AnimsTODO.Add(Control.Abe.HUNGLONG);
            }
            if (name == "GODOWN")
            {
                AnimsTODO.Add(Control.Abe.GODOWN);
            }

            if (name == "WALK")
            {
                AnimsTODO.Add(Control.Abe.PASS_2);
                AnimsTODO.Add(Control.Abe.PASS_3);
```

```
        }

        if (name == "SNEAK")
        {
            AnimsTODO.Add(Control.Abe.SNEAK_2);
            AnimsTODO.Add(Control.Abe.SNEAK_3);
        }

        if (name == "JUMP")
        {
            AnimsTODO.Add(Control.Abe.JUMP_2);
            AnimsTODO.Add(Control.Abe.JUMP_3);
        }
        if (name == "RUN")
        {
            AnimsTODO.Add(Control.Abe.RUN_4);
            AnimsTODO.Add(Control.Abe.RUN_5);
            AnimsTODO.Add(Control.Abe.RUN_2);
            AnimsTODO.Add(Control.Abe.RUN_3);
        }
        if (name == "RUNSTOP")
        {
            AnimsTODO.Add(Control.Abe.RUNsTOP_1);
            AnimsTODO.Add(Control.Abe.RUNsTOP_2);
        }
        if (name == "RUNJUMP")
        {
            AnimsTODO.Add(Control.Abe.RUNJUMP1);
            AnimsTODO.Add(Control.Abe.RUNJUMP2);
            AnimsTODO.Add(Control.Abe.RUNJUMP3);
            //AnimsTODO.Add(Control.Abe.RUNJUMP4);
        }
        if (name == "RUNSTART")
        {
            AnimsTODO.Add(Control.Abe.RUN_2);
            AnimsTODO.Add(Control.Abe.RUN_3);
        }
        if (name == "ROLL")
        {
            AnimsTODO.Add(Control.Abe.ROLL2);
            AnimsTODO.Add(Control.Abe.ROLL3);
        }
        if (name == "OMM")
        {
            AnimsTODO.Add(Control.Abe.OMM);
            AnimsTODO.Add(Control.Abe.OMM);
            AnimsTODO.Add(Control.Abe.OMM);
            AnimsTODO.Add(Control.Abe.OMM);
            AnimsTODO.Add(Control.Abe.OMM);
            AnimsTODO.Add(Control.Abe.OMM);
        }
        if (name == "NOTCONTROL")
        {
            AnimsTODO.Add(Control.Abe.NOTCONTROL);
        }
        if (name == "DIED")
        {
            AnimsTODO.Add(Control.Abe.DIE);
        }
        if (name != "DIED")
        {
            this.LinkTO(Control.DIED).addPRECONDITION(Control.PRECONDITIONS. ↙
DIED);
            //this.LinkTO(Control.DIED).addPRECONDITION(Control.PRECONDITIONS↙
.WARN, Control.ACTIONS.DIE);
        }
    }

    //Collega due stati e restituisce l'arco associato
    public ARCa LinkTO(STATEa to)
```

```
        {
            ARCa arc = new ARCa(this.name + "to" + to.name, this, to);

            foreach (ARCa arcNew in ARCS)
            {
                if (arcNew.NAME == arc.NAME)
                    return arcNew;
            }

            this.ARCS.Add(arc);
            return arc;
        }

        //Esegue tutte le animazioni contenute nello stato corrente
        public bool Execute()
        {
            if (!AnimsTODO[CURRENT_ANIM].LastFrameDone)
            {
                Control.Abe.AnimateAbe(AnimsTODO[CURRENT_ANIM]);
                return false;
            }
            else
            {
                if (CURRENT_ANIM == AnimsTODO.Count - 1)
                {
                    ENDSTATE = true;
                    CURRENT_ANIM = 0;
                }
                else
                {
                    CURRENT_ANIM++;
                }

                return true;
            }
        }

        //Controlla le precondizioni di tutti gli archi uscenti
        public PRECONDITIONa TryChangeState()
        {
            foreach (ARCa arc in Control.CURRENT_STATE.ARCS)
            {
                foreach (PRECONDITIONa precondition in arc.PRECONDITIONS)
                {
                    if (precondition.Verify())
                        return precondition;
                }
            }
            return null;
        }
    }
```

```
//Definisce gli archi dell'automa
//[p.47]
public class ARCa
{
    string name;
    STATEa from;
    STATEa to;

    List<PRECONDITIONa> preconditions;

    public ARCa(string name, STATEa from, STATEa to)
    {
        preconditions = new List<PRECONDITIONa>();
        this.name = name;
        this.from = from;
        this.to = to;
    }

    //Pone una precondizione sull'arco
    //[p.47]
    public void addPRECONDITION(PRECONDITIONa precondition)
    {
        PRECONDITIONa p = new PRECONDITIONa(from.Control, precondition.NAME);
        p.prec1 = precondition.prec1;
        p.prec2 = precondition.prec2;
        preconditions.Add(p);
        p.ARC = this;
    }

    //Pone una precondizione e un'azione sull'arco
    //Se la precondizione è verificata, viene eseguita l'azione
    //[p.47]
    public void addPRECONDITION(PRECONDITIONa precondition, ACTIONa action)
    {
        PRECONDITIONa p = new PRECONDITIONa(from.Control, precondition.NAME);
        p.prec1 = precondition.prec1;
        p.prec2 = precondition.prec2;
        preconditions.Add(p);
        p.addACTION(action);
        p.ARC = this;
    }

    Attribute

}
```

```
//Definisce le precondizioni che devono essere aggiunte ad un arco per transitare da uno stato all'altro.
//[p.46]
public class PRECONDITIONa
{
    string name;
    AbeControl Control;
    ACTIONa action;
    ARCa arc;
    public PRECONDITIONa prec1;
    public PRECONDITIONa prec2;
    public PRECONDITIONa(AbeControl Control, string name)
    {
        this.name = name;
        this.Control = Control;
    }

    public void addACTION(ACTIONa action)
    {
        this.action = action;
    }

    //Controlla se la precondizione è soddisfatta
    public bool Verify()
    {
        if (name == "RIGHT")
        {
            if (((Control.game.rightDown & Control.Abe.Direction == 1) || (Control.game.leftDown & Control.Abe.Direction == -1)) & !Control.game.altDown & !Control.game.shiftDown)
                return true;
        }

        if (name == "DOORENTERED")
        {
            if (Control.CURRENT_STATE ==  Control.ENTERDOOR && Control.CURRENT_STATE.ENDSTATE)
            {
                Control.DOORENTERED = true;
                return true;
            }
        }
        if (name == "ENTERDOOR")
        {
            if (Control.game.upDown)
            {
                List<Collision> colls = new List<Collision>();

                colls = Control.game.CurrentLevel.GetCollision(Control.Abe);

                foreach (Collision coll in colls)
                {
                    if (coll.TYPE == COLLISIONS.DOOR)
                    {
                        Control.DOOR = coll;
                        return true;
                    }
                }
            }
        }
        if (name == "NOTDOOR")
        {
            List<string> colTypes = new List<string>();

            colTypes = Control.game.CurrentLevel.collision(Control.Abe);
            if (!colTypes.Contains(COLLISIONS.DOOR))
                return true;
        }

        if (name == "LEVA")
```

```
        {
            List<string> colTypes = new List<string>();

            colTypes = Control.game.CurrentLevel.collision(Control.Abe);
            if (Control.Abe.CURRENTOBJECTA != null && (colTypes.Contains
(COLLISIONS.LEVA)))
            {
                if (Control.game.CTRLDown)
                {

                    Control.Abe.CURRENTOBJECTA.objControl.ACTIVE = true;
                    foreach (ObjectA slave in Control.Abe.CURRENTOBJECTA.
SLAVES)
                    {
                        slave.objControl.ACTIVE = true;
                        slave.objControl.ELECTRO_ON = !slave.objControl.
ELECTRO_ON;
                    }

                    return true;
                }
                else
                {
                    foreach (ObjectA slave in Control.Abe.CURRENTOBJECTA.
SLAVES)
                        slave.objControl.ACTIVE = false;
                }
            }
        }

        if (name == "BOMBACTIVE")
        {
            List<string> colTypes = new List<string>();

            colTypes = Control.game.CurrentLevel.collision(Control.Abe);
            if (Control.Abe.CURRENTOBJECTA != null && colTypes.Contains
(COLLISIONS.BOMBINNESCO) && Control.Abe.CURRENTOBJECTA.objControl.ACTIVE)
            {
                Control.Abe.DEAD = true;
                return true;
            }
        }

        if (name == "BOMBACTIVATE")
        {
            List<string> colTypes = new List<string>();

            colTypes = Control.game.CurrentLevel.collision(Control.Abe);
            if ((Control.Abe.CURRENTOBJECTA != null && colTypes.Contains
(COLLISIONS.BOMBINNESCO)))
            {
                if (Control.game.CTRLDown & !Control.Abe.CURRENTOBJECTA.
objControl.ACTIVE)
                {
                    return true;
                }
            }
        }
        if (name == "BOMBOK")
        {
            List<string> colTypes = new List<string>();

            colTypes = Control.game.CurrentLevel.collision(Control.Abe);
            if ((Control.Abe.CURRENTOBJECTA != null && Control.Abe.
CURRENTOBJECTA.objControl.ACTIVE && colTypes.Contains(COLLISIONS.
BOMBINNESCO)))
            {
                int fromBip = Math.Abs(9 - Control.Abe.CURRENTOBJECTA.
objControl.CURRENT_STATE.AnimsTODO[Control.Abe.CURRENTOBJECTA.objControl.
CURRENT_STATE.CURRENT_ANIM].CurrentFrame);
```

```
                if (Control.game.CTRLDown & fromBip <= Control.Abe.
CURRENTOBJECTA.RangeInnesco)
                {
                    Control.Abe.CURRENTOBJECTA.objControl.ACTIVE = false;
                    return true;
                }
            }
        }

        if (name == "BOMBFAIL")
        {
            List<string> colTypes = new List<string>();

            colTypes = Control.game.CurrentLevel.collision(Control.Abe);
            if ((Control.Abe.CURRENTOBJECTA != null && Control.Abe.
CURRENTOBJECTA.objControl.ACTIVE && colTypes.Contains(COLLISIONS.
BOMBINNESCO)))
            {
                int fromBip = Math.Abs(9 - Control.Abe.CURRENTOBJECTA.
objControl.CURRENT_STATE.AnimsTODO[Control.Abe.CURRENTOBJECTA.objControl.
CURRENT_STATE.CURRENT_ANIM].CurrentFrame);
                if (Control.game.CTRLDown & fromBip > Control.Abe.
CURRENTOBJECTA.RangeInnesco)
                {
                    Control.Abe.CURRENTOBJECTA.objControl.ACTIVE = true;
                    //Control.Abe.DEAD = true;
                    return true;
                }
            }

        }
        if (name == "BOMB")
        {
            List<string> colTypes = new List<string>();

            colTypes = Control.game.CurrentLevel.collision(Control.Abe);
            if ((colTypes.Contains(COLLISIONS.BOMB)))
            {
                Control.Abe.DEAD = true;
                return true;
            }

        }

        if (name == "ENDLEVA")
        {
            if (Control.CURRENT_STATE == Control.PULLEVER && Control.PULLEVER
.ENDSTATE)
            {
                Control.Abe.CURRENTOBJECTA.objControl.ACTIVE = false;

                return true;
            }
        }

        if (name == "ENDBOMB")
        {
            if (Control.CURRENT_STATE == Control.ACTIVATEBOMB && Control.
ACTIVATEBOMB.ENDSTATE)
            {
                Control.Abe.CURRENTOBJECTA.objControl.ACTIVE = true;
                return true;
            }
        }

        if (name == "ZERO")
        {
            if (Control.game.ZeroDown)
            {
                foreach (ObjectA obj in Control.game.CurrentLevel.ObjectsA)
```

```
                    if (obj.NAME == "OMM")
                        obj.objControl.ACTIVE = true;

                return true;
            }
        }

        if (name == "ZEROUP")
        {
            if (!Control.game.ZeroDown)
            {
                foreach (ObjectA obj in Control.game.CurrentLevel.ObjectsA)
                    if (obj.NAME == "OMM")
                        obj.objControl.ACTIVE = false;

                Control.OMM.CURRENT_ANIM = 0;
                return true;
            }
        }

        if (name == "STOP")
        {
            if ((((!Control.game.rightDown & Control.Abe.Direction == 1) || (!↙
Control.game.leftDown & Control.Abe.Direction == -1)))
                return true;
        }

        if (name == "toSNEAK")
        {
            if (((Control.game.rightDown & Control.Abe.Direction == 1) ||     ↙
(Control.game.leftDown & Control.Abe.Direction == -1)) & Control.game.        ↙
altDown)
                return true;
        }

        if (name == "toRUN")
        {
            if (((Control.game.rightDown & Control.Abe.Direction == 1) ||     ↙
(Control.game.leftDown & Control.Abe.Direction == -1)) & Control.game.        ↙
shiftDown)
                return true;
        }

        if (name == "toSTOPSNEAK")
        {
            if ((((!Control.game.rightDown & Control.Abe.Direction == 1) || (!↙
Control.game.leftDown & Control.Abe.Direction == -1)))
                return true;
        }

        if (name == "toSTOPRUN")
        {
            //if ((((!Control.game.rightDown & Control.Abe.Direction == 1) || ↙
(!Control.game.leftDown & Control.Abe.Direction == -1)))
            if(!Control.game.shiftDown)
                return true;
        }

        if (name == "toTURN")
        {
            if ((Control.Abe.Direction == -1 & Control.game.rightDown) ||     ↙
(Control.Abe.Direction == 1 & Control.game.leftDown))
                return true;
        }

        if (name == "toJUMP")
        {
            if (Control.game.spaceDown)
                return true;
        }
```

```
        if (name == "UP")
        {
            if (Control.game.upDown)
                return true;
        }
        if (name == "DOWN")
        {
            if (Control.game.downDown)
                return true;

        }

        if (name == "APPIGLIO")
        {
            List<string> colTypes = new List<string>();

            colTypes = Control.game.CurrentLevel.collision(Control.Abe);
            if (colTypes.Contains(COLLISIONS.APPIGLIODESTRO) & Control.Abe.
Direction == 1 || colTypes.Contains(COLLISIONS.APPIGLIOSINISTRO) & Control.
Abe.Direction == -1)
                return true;
        }
        if (name == "APPIGLIOTURN")
        {
            List<string> colTypes = new List<string>();

            colTypes = Control.game.CurrentLevel.collision(Control.Abe);
            if (colTypes.Contains(COLLISIONS.APPIGLIODESTRO) & Control.Abe.
Direction == -1 || colTypes.Contains(COLLISIONS.APPIGLIOSINISTRO) & Control.
Abe.Direction == 1)
                return true;
        }

        if (name == "notAPPIGLIO")
        {
            List<string> colTypes = new List<string>();

            colTypes = Control.game.CurrentLevel.collision(Control.Abe);
            if (!colTypes.Contains(COLLISIONS.APPIGLIODESTRO) & !colTypes.
Contains(COLLISIONS.APPIGLIOSINISTRO))
                return true;
        }

        if (name == "HOIST")
        {
            List<string> colTypes = new List<string>();

            colTypes = Control.game.CurrentLevel.collision(Control.Abe);
            if (colTypes.Contains(COLLISIONS.HOISTDESTRO) & Control.Abe.
Direction == 1 || colTypes.Contains(COLLISIONS.HOISTSINISTRO) & Control.Abe.
Direction == -1)
                return true;
        }

        if (name == "NOTHOIST")
        {
            List<string> colTypes = new List<string>();

            colTypes = Control.game.CurrentLevel.collision(Control.Abe);
            if (!colTypes.Contains(COLLISIONS.HOISTDESTRO) & Control.Abe.
Direction == 1 || !colTypes.Contains(COLLISIONS.HOISTSINISTRO) & Control.Abe.
Direction == -1)
                return true;
        }
        if (name == "HOISTTURN")
        {
            List<string> colTypes = new List<string>();

            colTypes = Control.game.CurrentLevel.collision(Control.Abe);
            if (colTypes.Contains(COLLISIONS.HOISTDESTRO) & Control.Abe.
```

```
Direction == -1 || colTypes.Contains(COLLISIONS.HOISTSINISTRO) & Control.Abe.↙
Direction == 1)
                return true;
        }

        if (name == "BLOCK")
        {
            List<string> colTypes = new List<string>();

            colTypes = Control.game.CurrentLevel.collision(Control.Abe);
            if (colTypes.Contains(COLLISIONS.BLOCCO))
                return true;
        }
        if (name == "GODOWN")
        {
            List<string> colTypes = new List<string>();

            colTypes = Control.game.CurrentLevel.collision(Control.Abe);
            if ((colTypes.Contains(COLLISIONS.GODOWN)))
                return true;
        }

        if (name == "NOTGODOWN")
        {
            List<string> colTypes = new List<string>();

            colTypes = Control.game.CurrentLevel.collision(Control.Abe);
            if ((!colTypes.Contains(COLLISIONS.GODOWN)))
                return true;
        }
        if (name == "MATTATOIO")
        {
            List<string> colTypes = new List<string>();

            colTypes = Control.game.CurrentLevel.collision(Control.Abe);
            if ((colTypes.Contains(COLLISIONS.MATTATOIO)))
                return true;
        }

        if (name == "SHOCK")
        {
            if (Control.CURRENT_STATE.CURRENT_ANIM == 1 & Control.game.      ↙
CurrentLevel.Sentinella)
            {
                Control.OMM.CURRENT_ANIM = 0;
                List<string> colTypes = new List<string>();

                colTypes = Control.game.CurrentLevel.collision(Control.Abe);
                if ((colTypes.Contains(COLLISIONS.SHOCK)))
                    return true;
            }
        }

        if (name == "ENDSTATE")
        {
            if (Control.CURRENT_STATE.ENDSTATE)
                return true;
        }

        if (name == "SLIGFORCONTROL")
        {
            if (Control.game.CurrentLevel.Sligs.Count > 0 & Control.         ↙
sligToControl == null & Control.CURRENT_STATE.ENDSTATE)
            {
                Control.sligToControl = Control.game.CurrentLevel.Sligs[0];
                Control.sligToControl.AI.CURRENT_STATE = Control.            ↙
sligToControl.AI.POSSESSED;
                return true;
            }
        }
```

```
            if (name == "HELLO")
            {
                if (Control.game.OneDown)
                    return true;
            }

            if (name == "FOLLOW")
            {
                if (Control.game.TwoDown)
                    return true;
            }

            if (name == "WAITME")
            {
                if (Control.game.ThreeDown)
                    return true;
            }

            if (name == "COMMANDJUMP")
            {
                if (Control.game.FourDown & !Control.game.shiftDown)
                    return true;
            }
            if (name == "COMMANDRUNJUMP")
            {
                if (Control.game.FourDown & Control.game.shiftDown)
                    return true;
            }

            if (name == "RUNOK")
            {
                if (Control.CURRENT_STATE.AnimsTODO[Control.CURRENT_STATE.        ↙
CURRENT_ANIM] == Control.Abe.RUN_3 || Control.CURRENT_STATE.AnimsTODO[Control↙
.CURRENT_STATE.CURRENT_ANIM] == Control.Abe.RUN_5)
                    return true;
            }

            if (name == "DIED")
            {
                if (Control.Abe.DEAD)
                {
                    Control.Abe.boundingBox = new Microsoft.Xna.Framework.        ↙
Rectangle(Control.Abe.boundingBox.X, Control.Abe.boundingBox.Y, 0, 0);
                    return true;
                }
            }

            if (name == "AND")
            {
                return this.prec1.Verify() & this.prec2.Verify();
            }

            return false;
        }

        Attribute
    }

    //Contiene i riferimenti a tutte le istanze della classe PRECONDITIONa
    public class PRECONDITIONSa...
```

```
//Definisce le azioni che possono essere associate ad una precondizione
//Ogni azione può contenere una o più animazioni.
//[p.45]
public class ACTIONa
{
    AbeControl Control;
    string name;
    public List<Animation> AnimsTODO;
    int CURRENT_ANIM = 0;
    public ACTIONa action1;
    public ACTIONa action2;

    public ACTIONa(AbeControl Control, string name)
    {
        AnimsTODO = new List<Animation>();
        this.Control = Control;
        this.name = name;

        if (name == "STOP")
        {
            AnimsTODO.Add(Control.Abe.STOP);
        }
        if (name == "STOPSNEAK")
        {
            AnimsTODO.Add(Control.Abe.SNEAKsTOP);
        }
        if (name == "STOPRUNJUMP")
        {
            AnimsTODO.Add(Control.Abe.RUNJUMP4);
        }
        if (name == "HUNGLONG")
        {
            AnimsTODO.Add(Control.Abe.HUNGLONG);
        }
        if (name == "PASS1")
        {
            AnimsTODO.Add(Control.Abe.PASS_1);
        }
        if (name == "JUMP1")
        {
            AnimsTODO.Add(Control.Abe.JUMP_1);
        }
        if (name == "JUMP4")
        {
            AnimsTODO.Add(Control.Abe.JUMP_4);
        }
        if (name == "RUN1")
        {
            AnimsTODO.Add(Control.Abe.RUN_1);
        }
        if (name == "JUMPUP")
        {
            AnimsTODO.Add(Control.Abe.JUMPUP);
        }
        if (name == "JUMPDOWN")
        {
            AnimsTODO.Add(Control.Abe.JUMPDOWN);
        }
        if (name == "TURN")
        {
            AnimsTODO.Add(Control.Abe.TURN);
        }

        if (name == "TURNDOWN")
        {
            AnimsTODO.Add(Control.Abe.TURNdOWN);
        }
        if (name == "SNEAK1")
        {
            AnimsTODO.Add(Control.Abe.SNEAK_1);
```

```
}
if (name == "CROUCH")
{
    AnimsTODO.Add(Control.Abe.CROUCH);
}
if (name == "STANDUP")
{
    AnimsTODO.Add(Control.Abe.STANDUP);
}
if (name == "ROLL1")
{
    AnimsTODO.Add(Control.Abe.ROLL1);
}
if (name == "KO")
{
    AnimsTODO.Add(Control.Abe.KO);
}
if (name == "GETUP")
{
    AnimsTODO.Add(Control.Abe.GETUP);
}
if (name == "BLOCK")
{
    AnimsTODO.Add(Control.Abe.BLOCK);
}
if (name == "HOISTUP")
{
    AnimsTODO.Add(Control.Abe.HOISTUP);
}
if (name == "HOISTDOWN")
{
    AnimsTODO.Add(Control.Abe.HOISTDOWN);
}
if (name == "PLANE")
{
    AnimsTODO.Add(Control.Abe.PLANE);
}
if (name == "HELLO")
{
    AnimsTODO.Add(Control.Abe.HELLO);
}
if (name == "FOLLOW")
{
    AnimsTODO.Add(Control.Abe.FOLLOW);
}
if (name == "WAITME")
{
    AnimsTODO.Add(Control.Abe.WAITME);
}
if (name == "COMMANDJUMP")
{
    AnimsTODO.Add(Control.Abe.COMMANDJUMP);
}
if (name == "COMMANDRUNJUMP")
{
    AnimsTODO.Add(Control.Abe.COMMANDRUNJUMP);
}
if (name == "EXPLODE")
{
    AnimsTODO.Add(Control.Abe.BOMBEXPLODE);
}
if (name == "BOMBACTIVATE")
{
    AnimsTODO.Add(Control.Abe.BOMBDEACTIVATE);
}
if (name == "DIE")
{
    AnimsTODO.Add(Control.Abe.DIE);
}
if (name == "PULLEVER")
```

```
            {
                AnimsTODO.Add(Control.Abe.PULLEVER);
            }
        }

        //Metodo chiamato per eseguire tutte le animazioni contenute in un'azione
        public bool DO()
        {
            if (!AnimsTODO[CURRENT_ANIM].LastFrameDone)
            {
                Control.Abe.AnimateAbe(AnimsTODO[CURRENT_ANIM]);
                return false;
            }
            else
            {
                if (CURRENT_ANIM == AnimsTODO.Count - 1)
                {
                    CURRENT_ANIM = 0;
                    Control.CURRENT_STATE.CURRENT_ANIM = 0;
                    return true;
                }
                else
                {
                    CURRENT_ANIM++;
                }
                return false;

            }
        }
        Attribute
}

//Contiene i riferimenti a tutte le istanze della classe ACTIONa
public class ACTIONSa...
```

```
using ...

namespace TheGame
{
    //Questa classe serve per caricare solo una volta le risorse (sprites, audio)
    //I riferiemnti alle risorse vengono acceduti da ogni istanza delle classi
    Mudokon/MudokonAI e Abe/AbeControl
    public class MudokonFactory
    {
        VARS

        public MudokonFactory(Game1 game)
        {
            this.game = game;

            BombDeactivate_R = game.Content.Load<Texture2D>("Sprite_Sheet\\
    Mudokon\\BOMBINNESCODEACTIVATE_R_120");
            BombDeactivate_L = game.Content.Load<Texture2D>("Sprite_Sheet\\
    Mudokon\\BOMBINNESCODEACTIVATE_L_120");

            BombDeactivateFail_R = game.Content.Load<Texture2D>("Sprite_Sheet\\
    Mudokon\\BOMBINNESCOEXPLODE1_R_120");
            BombDeactivateFail_L = game.Content.Load<Texture2D>("Sprite_Sheet\\
    Mudokon\\BOMBINNESCOEXPLODE1_L_120");

            BombExplode_C = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
    BOMBINNESCOEXPLODE2_C_120");

            Walk_1PASS_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
    PASS1_R_180");
            Walk_2PASS_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
    PASS2_R_180");
            Walk_3PASS_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
    PASS3_R_180");

            Walk_1PASS_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
    PASS1_L_180");
            Walk_2PASS_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
    PASS2_L_180");
            Walk_3PASS_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
    PASS3_L_180");

            Stop_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
    STOP_R_180");
            Stop_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
    STOP_L_180");

            Wait_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
    WAIT_R_70");
            Wait_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
    WAIT_L_70");

            WaitCrouch_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
    WAITcROUCH_R_70");
            WaitCrouch_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
    WAITcROUCH_L_70");

            Turn_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
    TURN_R_70");
            Turn_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
    TURN_L_70");

            TurnDown_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
    TURNdOWN_R_70");
            TurnDown_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
    TURNdOWN_L_70");

            Roll1_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
    ROLL1_R_180");
            Roll1_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
```

```
ROLL1_L_180");
        Roll2_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
ROLL2_R_180");
        Roll2_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
ROLL2_L_180");
        Roll3_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
ROLL3_R_180");
        Roll3_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
ROLL3_L_180");

        Jump1_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
JUMP1_R_200");
        Jump2_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
JUMP2_R_200");
        Jump3_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
JUMP3_R_200");
        Jump4_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
JUMP4_R_200");
        Jump1_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
JUMP1_L_200");
        Jump2_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
JUMP2_L_200");
        Jump3_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
JUMP3_L_200");
        Jump4_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
JUMP4_L_200");

        JumpUp_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
JUMPUP_R_80");
        JumpUp_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
JUMPUP_L_80");

        JumpDown_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
JUMPDOWN_R_80");
        JumpDown_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
JUMPDOWN_L_80");

        Hung_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
HUNG_R_70");
        Hung_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
HUNG_L_70");
        HungLong_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
HUNGLONG_R_120");
        HungLong_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
HUNGLONG_L_120");

        Crouch_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
CROUCH_R_180");
        Crouch_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
CROUCH_L_180");
        StandUp_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
STANDUP_R_180");
        StandUp_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
STANDUP_L_180");

        Block_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
BLOCK_R_80");
        Block_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
BLOCK_L_80");

        HoistUp_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
HOISTUP_R_80");
        HoistUp_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
HOISTUP_L_80");
        HoistDown_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
HOISTDOWN_R_80");
        HoistDown_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
HOISTDOWN_L_80");

        Block1_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Abe\\
```

```
BLOCK1_R_200");
        Block2_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Abe\\
BLOCK2_R_200");
        Block1_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Abe\\
BLOCK1_L_200");
        Block2_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Abe\\
BLOCK2_L_200");

        HoistUp1_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Abe\\
HOISTuP1_R_210");
        HoistUp2_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Abe\\
HOISTuP2_R_210");
        HoistUp1_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Abe\\
HOISTuP1_L_210");
        HoistUp2_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Abe\\
HOISTuP2_L_210");

        HoistDown1_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Abe\\
HOISTdOWN1_R_210");
        HoistDown2_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Abe\\
HOISTdOWN2_R_210");
        HoistDown1_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Abe\\
HOISTdOWN1_L_210");
        HoistDown2_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Abe\\
HOISTdOWN2_L_210");

        GoDown_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
GODOWN_L_160");
        GoDown_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
GODOWN_R_160");
        Plane_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
PLANE_L_160");
        Plane_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
PLANE_R_160");

        Ko_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\KO_R_130"
);
        Ko_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\KO_L_130"
);
        GetUp_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
GETUP_R_130");
        GetUp_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
GETUP_L_130");

        Run1_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
RUN1_R_300");
        Run1_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
RUN1_L_300");
        Run2_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
RUN2_R_300");
        Run2_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
RUN2_L_300");
        Run3_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
RUN3_R_300");
        Run3_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
RUN3_L_300");
        Run4_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
RUN4_R_300");
        Run4_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
RUN4_L_300");
        Run5_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
RUN5_R_300");
        Run5_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
RUN5_L_300");
        RunStop1_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
RUNSTOP1_R_300");
        RunStop2_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
RUNSTOP2_R_300");
        RunStop1_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
RUNSTOP1_L_300");
```

```
        RunStop2_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
RUNSTOP2_L_300");
        RunJump1_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
RUNJUMP1_R_260");
        RunJump1_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
RUNJUMP1_L_260");
        RunJump2_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
RUNJUMP2_R_260");
        RunJump2_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
RUNJUMP2_L_260");
        RunJump3_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
RUNJUMP3_R_260");
        RunJump3_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
RUNJUMP3_L_260");
        RunJump4_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
RUNJUMP4_R_260");
        RunJump4_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
RUNJUMP4_L_260");

        Sneak1_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
SNEAK1_R_200");
        Sneak2_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
SNEAK2_R_200");
        Sneak3_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
SNEAK3_R_200");
        Sneak1_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
SNEAK1_L_200");
        Sneak2_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
SNEAK2_L_200");
        Sneak3_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
SNEAK3_L_200");
        SneakStop_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
SNEAKsTOP_R_200");
        SneakStop_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
SNEAKsTOP_L_200");

        Die_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
SHOOTSPLAT_R_130");
        Die_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
SHOOTSPLAT_L_130");

        PulLever_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
PULLEVER_R_110");
        PulLever_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
PULLEVER_L_110");

        Omm_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\OMM_R_80
");
        Omm_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\OMM_L_80
");

        EnterDoor_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
ENTERDOOR_R_80");
        EnterDoor_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
ENTERDOOR_L_80");
        ExitDoor_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
EXITDOOR_R_80");
        ExitDoor_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
EXITDOOR_L_80");

        CleanUp = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
CLEANUP_C_80");
        RestUp = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
RESTUP_C_80");
        CleanUpStop_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
STOPCLEANUP_R_90");
        CleanUpStop_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
STOPCLEANUP_L_90");
        CleanDown_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
CLEANDOWN_R_80");
```

```
            CleanDown_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Mudokon\\
    CLEANDOWN_L_80");

            ////////////////////////////////////////////////////////

            Sound_PASS = game.Content.Load<SoundEffect>("Sound\\PASS");
            Sound_TURN = game.Content.Load<SoundEffect>("Sound\\TURN");
            Sound_JUMP = game.Content.Load<SoundEffect>("Sound\\JUMP");
            Sound_RUNSTOP = game.Content.Load<SoundEffect>("Sound\\RUNSTOP");
            Sound_SNEAK = game.Content.Load<SoundEffect>("Sound\\SNEAK");
            Sound_IMPACT = game.Content.Load<SoundEffect>("Sound\\IMPACT");
            Sound_HELLO = game.Content.Load<SoundEffect>("Sound\\HELLO");
            Sound_FOLLOW = game.Content.Load<SoundEffect>("Sound\\FOLLOW");
            Sound_WAIT = game.Content.Load<SoundEffect>("Sound\\WAIT");
            Sound_ROLL = game.Content.Load<SoundEffect>("Sound\\ROLL");
            Sound_OK = game.Content.Load<SoundEffect>("Sound\\OK");
            Sound_OMM = game.Content.Load<SoundEffect>("Sound\\OMM");
            Sound_ENTERPORTAL = game.Content.Load<SoundEffect>("Sound\\
    PORTALOPENING");
            Sound_COMMANDJUMP = game.Content.Load<SoundEffect>("Sound\\
    COMMANDJUMP");
            Sound_COMMANDRUNJUMP = game.Content.Load<SoundEffect>("Sound\\
    COMMANDRUNJUMP");
        }
    }
}
```

```
using ...

namespace TheGame
{
    //Gestisce grafica ed effetti sonori di un Mudokon
    public class Mudokon : Character
    {
        VARS

        public Mudokon(Game1 game,string state,Point position)
        {
            this.game = game;

            ////////////////////////////////////////////////

            Sound_PASS = game.MudokonFactory.Sound_PASS;
            Sound_TURN = game.MudokonFactory.Sound_TURN;
            Sound_JUMP = game.MudokonFactory.Sound_JUMP;
            Sound_RUNSTOP = game.MudokonFactory.Sound_RUNSTOP;
            Sound_SNEAK = game.MudokonFactory.Sound_SNEAK;
            Sound_IMPACT = game.MudokonFactory.Sound_IMPACT;
            Sound_HELLO = game.MudokonFactory.Sound_HELLO;
            Sound_OK = game.MudokonFactory.Sound_OK;
            Sound_ENTERPORTAL = game.MudokonFactory.Sound_ENTERPORTAL;

            ////////////////////////////////////////////////


            WAIT = new Animation(game.MudokonFactory.Wait_R, game.MudokonFactory.↙
    Wait_L, 1000 / 6, 6, "WAIT", 0, 0, -1, 15, 15);
            TURN = new Animation(game.MudokonFactory.Turn_L, game.MudokonFactory.↙
    Turn_R, AnimateRate, 12, "TURN", new Sound(Sound_TURN, 2, 0.05f), 0, 0, -1,  ↙
    10, 10);

            STOP = new Animation(game.MudokonFactory.Stop_R, game.MudokonFactory.↙
    Stop_L, AnimateRate, 4, "STOP", new Sound(Sound_PASS, 4), 0, 0, -1, 43, 43);
            PASS_1 = new Animation(game.MudokonFactory.Walk_1PASS_R, game.       ↙
    MudokonFactory.Walk_1PASS_L, AnimateRate, 7, "PASS_1", new Sound(Sound_PASS, ↙
    4), 1, 0, 6, 0, 43);
            PASS_2 = new Animation(game.MudokonFactory.Walk_2PASS_R, game.       ↙
    MudokonFactory.Walk_2PASS_L, AnimateRate, 9, "PASS_2", new Sound(Sound_PASS, ↙
    6), 1, 0, 8, 43, 86);
            PASS_3 = new Animation(game.MudokonFactory.Walk_3PASS_R, game.       ↙
    MudokonFactory.Walk_3PASS_L, AnimateRate, 9, "PASS_3", new Sound(Sound_PASS, ↙
    6), 1, 0, 8, 86, 129);

            JUMP_1 = new Animation(game.MudokonFactory.Jump1_R, game.            ↙
    MudokonFactory.Jump1_L, AnimateRate, 10, "JUMP_1", 1, 0, 3, 18, 61);
            JUMP_2 = new Animation(game.MudokonFactory.Jump2_R, game.            ↙
    MudokonFactory.Jump2_L, AnimateRate, 3, "JUMP_2", 1, 0, 3, 61, 104);
            JUMP_3 = new Animation(game.MudokonFactory.Jump3_R, game.            ↙
    MudokonFactory.Jump3_L, AnimateRate, 3, "JUMP_3", 1, 0, 3, 104, 147);
            JUMP_4 = new Animation(game.MudokonFactory.Jump4_R, game.            ↙
    MudokonFactory.Jump4_L, AnimateRate, 6, "JUMP_4", new Sound(Sound_JUMP, 1), 0↙
    , 0, -1, 147, 147);

            SNEAKsTOP = new Animation(game.MudokonFactory.SneakStop_R, game.     ↙
    MudokonFactory.SneakStop_L, AnimateRate, 4, "SNEAKsTOP", 0, 0, -1, 53, 53);
            SNEAK_1 = new Animation(game.MudokonFactory.Sneak1_R, game.          ↙
    MudokonFactory.Sneak1_L, AnimateRate, 6, "SNEAK_1", new Sound(Sound_SNEAK, 3)↙
    , 1, 0, 5, 10, 53);
            SNEAK_2 = new Animation(game.MudokonFactory.Sneak2_R, game.          ↙
    MudokonFactory.Sneak2_L, AnimateRate, 10, "SNEAK_2", new Sound(Sound_SNEAK,  ↙
    7), 1, 0, 9, 53, 96);
            SNEAK_3 = new Animation(game.MudokonFactory.Sneak3_R, game.          ↙
    MudokonFactory.Sneak3_L, AnimateRate, 10, "SNEAK_3", new Sound(Sound_SNEAK,  ↙
    7), 1, 0, 9, 96, 139);

            RUN_1 = new Animation(game.MudokonFactory.Run1_R, game.MudokonFactory↙
    .Run1_L, AnimateRate, 4, "RUN_1", 2, 0, 3, 35, 35);
```

```
        RUN_2 = new Animation(game.MudokonFactory.Run2_R, game.MudokonFactory
.Run2_L, AnimateRate, 2, "RUN_2", new Sound(Sound_PASS, 2), 2, 0, 4, 35, 78);
        RUN_3 = new Animation(game.MudokonFactory.Run3_R, game.MudokonFactory
.Run3_L, AnimateRate, 3, "RUN_3", 2, 0, 9, 78, 121);
        RUN_4 = new Animation(game.MudokonFactory.Run4_R, game.MudokonFactory
.Run4_L, AnimateRate, 5, "RUN_4", new Sound(Sound_PASS, 2), 2, 0, 9, 121,
164);
        RUN_5 = new Animation(game.MudokonFactory.Run5_R, game.MudokonFactory
.Run5_L, AnimateRate, 6, "RUN_5", 2, 0, 9, 164, 207);

        RUNsTOP_1 = new Animation(game.MudokonFactory.RunStop1_R, game.
MudokonFactory.RunStop1_L, AnimateRate, 8, "RUNSTOP_1", new Sound
(Sound_RUNSTOP, 3), 1, 0, 7, 121, 164);
        RUNsTOP_2 = new Animation(game.MudokonFactory.RunStop2_R, game.
MudokonFactory.RunStop2_L, AnimateRate, 8, "RUNSTOP_2", 1, 0, 7, 164, 207);

        RUNJUMP1 = new Animation(game.MudokonFactory.RunJump1_R, game.
MudokonFactory.RunJump1_L, AnimateRate, 2, "RUNJUMP1", 1, 0, 1, 0, 43, 0, 0);
        RUNJUMP2 = new Animation(game.MudokonFactory.RunJump2_R, game.
MudokonFactory.RunJump2_L, AnimateRate, 5, "RUNJUMP2", 1, 0, 7, 43, 86, 0,
50);
        RUNJUMP3 = new Animation(game.MudokonFactory.RunJump3_R, game.
MudokonFactory.RunJump3_L, AnimateRate, 4, "RUNJUMP3",new Sound
(Sound_ENTERPORTAL, 3), 1, 0, 7, 86, 129, 50, 50);
        RUNJUMP4 = new Animation(game.MudokonFactory.RunJump4_R, game.
MudokonFactory.RunJump4_L, AnimateRate, 5, "RUNJUMP4", 1, 0, 7, 129, 172, 50,
 0);

        WAITcROUCH = new Animation(game.MudokonFactory.WaitCrouch_R, game.
MudokonFactory.WaitCrouch_L, 1000 / 6, 6, "WAITcROUCH", 0, 0, -1, 10, 10);
        TURNdOWN = new Animation(game.MudokonFactory.TurnDown_L, game.
MudokonFactory.TurnDown_R, AnimateRate, 8, "TURNdOWN", new Sound(Sound_TURN,
2, 0.05f), 0, 0, -1, 10, 10);
        CROUCH = new Animation(game.MudokonFactory.Crouch_R, game.
MudokonFactory.Crouch_L, AnimateRate, 9, "CROUCH", 0, 0, -1, 10, 10);
        STANDUP = new Animation(game.MudokonFactory.StandUp_R, game.
MudokonFactory.StandUp_L, AnimateRate, 9, "STANDUP", 0, 0, -1, 10, 10);

        HELLO = new Animation(game.MudokonFactory.Wait_R, game.MudokonFactory
.Wait_L, 1000 / 6, 6, "HELLO", new Sound(Sound_HELLO, 1), 0, 0, -1, 15, 15);
        OK = new Animation(game.MudokonFactory.Wait_R, game.MudokonFactory.
Wait_L, 1000 / 6, 6, "OK", new Sound(Sound_OK, 1), 0, 0, -1, 15, 15);

        DIE = new Animation(game.MudokonFactory.Die_R, game.MudokonFactory.
Die_L, AnimateRate, 28, "DIE", 0, 0, -1, 50, 50);
        //DOWNPLANE = new Animation(game.MudokonFactory.DownPlane_R, game.
MudokonFactory.DownPlane_L, AnimateRate, 21, "DOWNPLANE", new Sound
(Sound_IMPACT, 14), 0, 0, -1, 47, 90, 200, 0);
        GODOWN = new Animation(game.MudokonFactory.GoDown_R, game.
MudokonFactory.GoDown_L, AnimateRate, 9, "GODOWN", 1, -2, 5, 12, 55, 200, 50)
;
        PLANE = new Animation(game.MudokonFactory.Plane_R, game.
MudokonFactory.Plane_L, AnimateRate, 12, "PLANE", new Sound(Sound_IMPACT, 3),
 1, 0, 2, 55, 98, 50, 0);

        CLEANUP = new Animation(game.MudokonFactory.CleanUp, game.
MudokonFactory.CleanUp, 1000 / 10, 10, "CLEANUP", 0, 0, -1, 15, 15);
        RESTUP = new Animation(game.MudokonFactory.RestUp, game.
MudokonFactory.RestUp, 1000 / 6, 6, "RESTUP", 0, 0, -1, 15, 15);
        CLEANUPSTOPLEFT = new Animation(game.MudokonFactory.CleanUpStop_L,
game.MudokonFactory.CleanUpStop_L, 1000 / 30, 17, "CLEANUPSTOPLEFT", 0, 0, -1
, 15, 15);
        CLEANUPSTOPRIGHT= new Animation(game.MudokonFactory.CleanUpStop_R,
game.MudokonFactory.CleanUpStop_R, 1000 / 30, 17, "CLEANUPSTOPRIGHT", 0, 0, -
1, 15, 15);
        CLEANDOWN = new Animation(game.MudokonFactory.CleanDown_R, game.
MudokonFactory.CleanDown_L, 1000 / 30, 11, "CLEANDOWN", 0, 0, -1, 10, 10);


        HOISTUP = new Animation(game.MudokonFactory.HoistUp_R, game.
```

```
MudokonFactory.HoistUp_L, AnimateRate, 23, "HOISTUP", 0, 0, -1, 15, 15, 0,
150);
        HOISTDOWN = new Animation(game.MudokonFactory.HoistDown_R, game.
MudokonFactory.HoistDown_L, AnimateRate, 23, "HOISTDOWN", 0, 0, -1, 15, 15,
150, 0);

        JUMPDOWN = new Animation(game.MudokonFactory.JumpDown_R, game.
MudokonFactory.JumpDown_L, AnimateRate, 18, "JUMOdOWN", new Sound(Sound_JUMP,
 7), 0, -1, 10, 15, 15, 50, 0);
        JUMPUP = new Animation(game.MudokonFactory.JumpUp_R, game.
MudokonFactory.JumpUp_L, AnimateRate, 14, "JUMPuP", new Sound(Sound_PASS, 13)
, 0, 1, 13, 15, 15, 0, 50);

        KO = new Animation(game.MudokonFactory.Ko_R, game.MudokonFactory.Ko_L
, AnimateRate, 13, "KO", new Sound(Sound_IMPACT, 3), 0, 0, -1, 46, 46);
        GETUP = new Animation(game.MudokonFactory.GetUp_R, game.
MudokonFactory.GetUp_L, AnimateRate, 13, "GETUP", 0, 0, -1, 46, 46);

        Direction = 1;
        color = Color.Yellow;
        AI = new MudokonAI(this, game, state);
        CurrentAnimation = AI.CURRENT_STATE.AnimsTODO[AI.CURRENT_STATE.
CURRENT_ANIM];
        animTODO = CurrentAnimation;
        PrevAnimation = CurrentAnimation;
        DrawingRec = new Rectangle(position.X - 18, position.Y -
CurrentAnimation.Sheet.Height + 5, CurrentAnimation.SpriteWidth,
CurrentAnimation.SpriteHeight);//-18 + 215, 285
        boundingBox = new Rectangle(position.X, position.Y - boundingBox.
Height, 44, 150);//42

        Mudokon_BBOX_PNG = game.Content.Load<Texture2D>("Sprite_Sheet\\Abe\\
ABE_BBOX");
        this.state = state;


    }

    //Gestisce la spritesheet dell'animazione corrente.
    //Definisce quale frame della spritesheet deve essere disegnato.
    public void AnimateMudokon(Animation anim)
    {
        if (anim.CurrentFrame == 0)
        {
            PrevAnimation = CurrentAnimation;
            PrevAnimation.Reset();
            CurrentAnimation = anim;

            if (Direction == 1)
            {
                DrawingRec.X = DrawingRec.X + (PrevAnimation.xEND -
CurrentAnimation.xSTART);
            }
            else
            {
                DrawingRec.X = DrawingRec.X + ((PrevAnimation.SpriteLWidth -
PrevAnimation.xEND) - (CurrentAnimation.SpriteLWidth - CurrentAnimation.
xSTART));
            }

            DrawingRec.Y = DrawingRec.Y - (PrevAnimation.yEND -
CurrentAnimation.ySTART) + (PrevAnimation.SpriteHeight - CurrentAnimation.
SpriteHeight);

            if (CurrentAnimation == TURN || CurrentAnimation == TURNdOWN)
                Direction = Direction * -1;
            if (CurrentAnimation == CLEANUPSTOPLEFT)
                Direction =  -1;
            if (CurrentAnimation == CLEANUPSTOPRIGHT)
                Direction = 1;
```

```
                if (Direction == 1)
                {
                    anim.SpriteWidth = anim.SpriteRWidth;
                    anim.SpriteHeight = anim.SpriteRHeight;
                    anim.Sheet = anim.SheetR;
                }
                else
                {
                    anim.SpriteWidth = anim.SpriteLWidth;
                    anim.SpriteHeight = anim.SpriteLHeight;
                    anim.Sheet = anim.SheetL;
                }

                DrawingRec.Width = anim.SpriteWidth;
                DrawingRec.Height = anim.SpriteHeight;

                this.game.spriteCurrent = anim.Sheet;
            }

            if (anim.Time > anim.AnimateRate)
            {
                if (anim.CurrentFrame == 1)
                {
                    boundingBox.X += (anim.xEND - anim.xSTART) * Direction;
                    boundingBox.Y = DrawingRec.Bottom - (anim.yEND + boundingBox.↙
    Height);
                }
                anim.Time = 0f;
                if (anim.Effect != null && anim.CurrentFrame == anim.Effect.     ↙
    FrameToPlay)
                    anim.Effect.Play();

                if (anim.CurrentFrame == anim.FrameCount)
                {
                    anim.LastFrameDone = true;

                }
                else
                    anim.LastFrameDone = false;

                if (anim.CurrentFrame < anim.FrameCount)
                {
                    anim.SourceRect = new Rectangle(anim.CurrentFrame * anim.    ↙
    SpriteWidth, 0, anim.SpriteWidth, anim.SpriteHeight);
                    this.game.sourceRect = anim.SourceRect;
                    anim.Forward();
                }
            }
        }

        Attribute

    }
}
```

```
//MudokonAI definisce l'automa per la gestione del comportamento di un Mudokon
//[pp.43]
public class MudokonAI
{
    VARS

    public MudokonAI() { }
    public MudokonAI(Mudokon Mudokon, Game1 game,string state)
    {
        this.game = game;
        this.Mudokon = Mudokon;


        PRECONDITIONS = new PRECONDITIONSm(this);
        ACTIONS = new ACTIONSm(this);

        DIED = new STATEm("DIED", this);
        CLEANUP = new STATEm("CLEANUP", this);
        CLEANDOWN = new STATEm("CLEANDOWN", this);
        WAIT = new STATEm("WAIT", this);
        WALK = new STATEm("WALK", this);
        JUMP = new STATEm("JUMP", this);
        SNEAK = new STATEm("SNEAK", this);
        WAITfOLLOW = new STATEm("WAITfOLLOW", this);
        WAITCOMMAND = new STATEm("WAITCOMMAND", this);
        JUMPCOMMAND = new STATEm("JUMPCOMMAND", this);
        RUNJUMPCOMMAND1 = new STATEm("RUNJUMPCOMMAND1", this);
        RUNJUMPCOMMAND2 = new STATEm("RUNJUMPCOMMAND2", this);
        RUNJUMPCOMMAND3 = new STATEm("RUNJUMPCOMMAND3", this);
        RUNJUMPCOMMAND4 = new STATEm("RUNJUMPCOMMAND4", this);
        GOTOABE = new STATEm("GOTOABE", this);
        RUN = new STATEm("RUN", this);
        RUNSTART = new STATEm("RUNSTART", this);
        RUNSTOP = new STATEm("RUNSTOP", this);
        RUNJUMP = new STATEm("RUNJUMP", this);
        RUNJUMPESCAPE = new STATEm("RUNJUMPESCAPE", this);
        GODOWN = new STATEm("GODOWN", this);
        SHOOTED = new STATEm("SHOOTED", this);

        CLEANUP.LinkTO(RUNSTART).addPRECONDITION(PRECONDITIONS.PORTALLEFT, ACTIONS.CLEANUPSTOPLEFT);
        CLEANUP.LinkTO(RUNSTART).addPRECONDITION(PRECONDITIONS.PORTALRIGHT, ACTIONS.CLEANUPSTOPRIGHT);
        CLEANUP.LinkTO(GODOWN).addPRECONDITION(PRECONDITIONS.GODOWN);
        CLEANUP.LinkTO(WAITfOLLOW).addPRECONDITION(PRECONDITIONS.ABeHELLOLEFT, ACTIONS.AND(ACTIONS.CLEANUPSTOPLEFT,ACTIONS.HELLO));
        CLEANUP.LinkTO(WAITfOLLOW).addPRECONDITION(PRECONDITIONS.ABeHELLORIGHT, ACTIONS.AND(ACTIONS.CLEANUPSTOPRIGHT, ACTIONS.HELLO));

        CLEANDOWN.LinkTO(RUNSTART).addPRECONDITION(PRECONDITIONS.PORTALOPENAHEAD, ACTIONS.STANDUP);
        CLEANDOWN.LinkTO(RUNSTART).addPRECONDITION(PRECONDITIONS.PORTALOPENBEHIND,ACTIONS.AND(ACTIONS.TURNDOWN,ACTIONS.STANDUP));
        CLEANDOWN.LinkTO(GODOWN).addPRECONDITION(PRECONDITIONS.GODOWN);
        CLEANDOWN.LinkTO(WAITfOLLOW).addPRECONDITION(PRECONDITIONS.ABeHELLOA, ACTIONS.AND(ACTIONS.STANDUP, ACTIONS.HELLO));
        //CLEANDOWN.LinkTO(WAITfOLLOW).addPRECONDITION(PRECONDITIONS.ABeHELLOB, ACTIONS.AND(ACTIONS.TURNDOWN, ACTIONS.AND(ACTIONS.STANDUP, ACTIONS.HELLO)));

        WAITfOLLOW.LinkTO(RUNSTART).addPRECONDITION(PRECONDITIONS.PORTALOPENAHEAD);
        WAITfOLLOW.LinkTO(RUNSTART).addPRECONDITION(PRECONDITIONS.PORTALOPENBEHIND, ACTIONS.TURN);
        WAITfOLLOW.LinkTO(GODOWN).addPRECONDITION(PRECONDITIONS.GODOWN);
        WAITfOLLOW.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.ABeFOLLOW, ACTIONS.OK);
        WAITfOLLOW.LinkTO(CLEANDOWN).addPRECONDITION(PRECONDITIONS.ABEWAITME, ACTIONS.AND(ACTIONS.OK,ACTIONS.CROUCH));
```

```
        GOTOABE.LinkTO(WAITCOMMAND).addPRECONDITION(PRECONDITIONS.
ABESAMEPOSITION,ACTIONS.STOP);

        WAIT.LinkTO(RUNJUMPCOMMAND1).addPRECONDITION(PRECONDITIONS.
ABECOMMANDRUNJUMP);
        WAIT.LinkTO(JUMPCOMMAND).addPRECONDITION(PRECONDITIONS.ABECOMMANDJUMP
,ACTIONS.JUMP1);
        WAIT.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.AND(PRECONDITIONS.
ABECOMMANDJUMP, PRECONDITIONS.BLOCK), ACTIONS.AND(ACTIONS.KO, ACTIONS.GETUP))
;

        WAIT.LinkTO(GOTOABE).addPRECONDITION(PRECONDITIONS.ABECALLA, ACTIONS.
PASS1);
        WAIT.LinkTO(GOTOABE).addPRECONDITION(PRECONDITIONS.ABECALLB, ACTIONS.
AND(ACTIONS.TURN, ACTIONS.PASS1));

        WAIT.LinkTO(RUNSTART).addPRECONDITION(PRECONDITIONS.PORTALOPENAHEAD);
        WAIT.LinkTO(RUNSTART).addPRECONDITION(PRECONDITIONS.PORTALOPENBEHIND,
ACTIONS.TURN);
        WAIT.LinkTO(GODOWN).addPRECONDITION(PRECONDITIONS.GODOWN);
        WAIT.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.BLOCK,ACTIONS.TURN);
        WAIT.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.ABEBEHIND, ACTIONS.
TURN);
        WAIT.LinkTO(CLEANDOWN).addPRECONDITION(PRECONDITIONS.ABEWAITME,
ACTIONS.AND(ACTIONS.OK, ACTIONS.CROUCH));
        WAIT.LinkTO(SNEAK).addPRECONDITION(PRECONDITIONS.AND(PRECONDITIONS.
ABESNEAK, PRECONDITIONS.ABEBEHIND), ACTIONS.AND(ACTIONS.TURN, ACTIONS.
SNEAK1));
        WAIT.LinkTO(SNEAK).addPRECONDITION(PRECONDITIONS.AND(PRECONDITIONS.
ABESNEAK, PRECONDITIONS.NOTABEOPPOSITE), ACTIONS.SNEAK1);
        WAIT.LinkTO(WALK).addPRECONDITION(PRECONDITIONS.AND(PRECONDITIONS.
ABEwALK, PRECONDITIONS.AND(PRECONDITIONS.ABEBEHIND,PRECONDITIONS.ABENOTNEAR))
, ACTIONS.AND(ACTIONS.TURN, ACTIONS.PASS1));
        WAIT.LinkTO(WALK).addPRECONDITION(PRECONDITIONS.AND(PRECONDITIONS.
ABEwALK, PRECONDITIONS.AND(PRECONDITIONS.NOTABEOPPOSITE,PRECONDITIONS.
ABENOTNEAR)), ACTIONS.PASS1);

        //WAIT.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.AND(PRECONDITIONS.
ABEOPPOSITE, PRECONDITIONS.ABERUNSTART)
        WAIT.LinkTO(RUNSTART).addPRECONDITION(PRECONDITIONS.AND(PRECONDITIONS
.ABERUNSTART, PRECONDITIONS.AND(PRECONDITIONS.ABEBEHIND, PRECONDITIONS.
ABENOTNEAR)), ACTIONS.TURN);
        WAIT.LinkTO(RUNSTART).addPRECONDITION(PRECONDITIONS.AND(PRECONDITIONS
.ABERUNSTART, PRECONDITIONS.AND(PRECONDITIONS.NOTABEOPPOSITE, PRECONDITIONS.
ABENOTNEAR)));
        WAIT.LinkTO(RUNSTART).addPRECONDITION(PRECONDITIONS.AND(PRECONDITIONS
.ABERUN, PRECONDITIONS.AND(PRECONDITIONS.NOTABEOPPOSITE, PRECONDITIONS.
ABENOTNEAR)));
        WAIT.LinkTO(WALK).addPRECONDITION(PRECONDITIONS.ABENOTNEAR, ACTIONS.
PASS1);

        WAIT.LinkTO(WAITCOMMAND).addPRECONDITION(PRECONDITIONS.HOISTTURN,
ACTIONS.STOP);
        WAIT.LinkTO(WAITCOMMAND).addPRECONDITION(PRECONDITIONS.HOIST, ACTIONS
.STOP);
        WAIT.LinkTO(WAITCOMMAND).addPRECONDITION(PRECONDITIONS.AND
(PRECONDITIONS.ABEUP, PRECONDITIONS.APPIGLIOTURN), ACTIONS.STOP);
        WAIT.LinkTO(WAITCOMMAND).addPRECONDITION(PRECONDITIONS.AND
(PRECONDITIONS.ABEUP, PRECONDITIONS.APPIGLIO), ACTIONS.STOP);

        //WALK.LinkTO(CLEANDOWN).addPRECONDITION(PRECONDITIONS.ENSsCREEN);
        WALK.LinkTO(WAITCOMMAND).addPRECONDITION(PRECONDITIONS.HOISTTURN,
ACTIONS.STOP);
        WALK.LinkTO(WAITCOMMAND).addPRECONDITION(PRECONDITIONS.HOIST, ACTIONS
.STOP);
        WALK.LinkTO(WAITCOMMAND).addPRECONDITION(PRECONDITIONS.AND
(PRECONDITIONS.ABEUP,PRECONDITIONS.APPIGLIOTURN), ACTIONS.STOP);
        WALK.LinkTO(WAITCOMMAND).addPRECONDITION(PRECONDITIONS.AND
(PRECONDITIONS.ABEUP, PRECONDITIONS.APPIGLIO), ACTIONS.STOP);
```

```
        WAITCOMMAND.LinkTO(RUNJUMPCOMMAND1).addPRECONDITION(PRECONDITIONS.
ABECOMMANDRUNJUMP);
        WAITCOMMAND.LinkTO(JUMPCOMMAND).addPRECONDITION(PRECONDITIONS.
ABECOMMANDJUMP, ACTIONS.JUMP1);
        WAITCOMMAND.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.AND
(PRECONDITIONS.HOISTTURN, PRECONDITIONS.ABEDOWN), ACTIONS.AND(ACTIONS.AND
(ACTIONS.TURN, ACTIONS.HOISTDOWN), ACTIONS.JUMPDOWN));
        WAITCOMMAND.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.AND
(PRECONDITIONS.HOIST, PRECONDITIONS.ABEDOWN), ACTIONS.AND(ACTIONS.HOISTDOWN,
ACTIONS.JUMPDOWN));
        WAITCOMMAND.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.AND
(PRECONDITIONS.APPIGLIO, PRECONDITIONS.ABEUP), ACTIONS.AND(ACTIONS.JUMPUP,
ACTIONS.HOISTUP));
        WAITCOMMAND.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.AND
(PRECONDITIONS.APPIGLIOTURN, PRECONDITIONS.ABEUP), ACTIONS.AND(ACTIONS.AND
(ACTIONS.TURN,ACTIONS.JUMPUP), ACTIONS.HOISTUP));
        WAITCOMMAND.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.AND
(PRECONDITIONS.ABESAMEFLOOR,PRECONDITIONS.AND(PRECONDITIONS.HOIST,
PRECONDITIONS.ABEAHEAD)));
        WAITCOMMAND.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.AND
(PRECONDITIONS.ABESAMEFLOOR, PRECONDITIONS.AND(PRECONDITIONS.HOISTTURN,
PRECONDITIONS.ABEBEHIND)), ACTIONS.TURN);
        WAITCOMMAND.LinkTO(WAITCOMMAND).addPRECONDITION(PRECONDITIONS.AND
(PRECONDITIONS.ABESAMEFLOOR, PRECONDITIONS.AND(PRECONDITIONS.HOIST,
PRECONDITIONS.ABEBEHIND)), ACTIONS.TURN);
        WAITCOMMAND.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.NOTHOIST);
        WAITCOMMAND.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.AND
(PRECONDITIONS.ABECOMMANDJUMP,PRECONDITIONS.BLOCK), ACTIONS.AND(ACTIONS.KO,
ACTIONS.GETUP));

        JUMPCOMMAND.LinkTO(WAITCOMMAND).addPRECONDITION(PRECONDITIONS.BLOCK,
ACTIONS.AND(ACTIONS.KO,ACTIONS.GETUP));
        JUMPCOMMAND.LinkTO(GODOWN).addPRECONDITION(PRECONDITIONS.AND
(PRECONDITIONS.ENDSTATE,PRECONDITIONS.GODOWN));
        JUMPCOMMAND.LinkTO(WAITCOMMAND).addPRECONDITION(PRECONDITIONS.
ENDSTATE, ACTIONS.JUMP4);

        RUNJUMPCOMMAND1.LinkTO(WAITCOMMAND).addPRECONDITION(PRECONDITIONS.
BLOCK, ACTIONS.STOP);
        //RUNJUMPCOMMAND1.LinkTO(GODOWN).addPRECONDITION(PRECONDITIONS.
GODOWN);
        RUNJUMPCOMMAND1.LinkTO(RUNJUMPCOMMAND2).addPRECONDITION(PRECONDITIONS
.ENDSTATE);

        RUNJUMPCOMMAND2.LinkTO(WAITCOMMAND).addPRECONDITION(PRECONDITIONS.
BLOCK, ACTIONS.AND(ACTIONS.KO, ACTIONS.GETUP));
        //RUNJUMPCOMMAND2.LinkTO(GODOWN).addPRECONDITION(PRECONDITIONS.
GODOWN);
        RUNJUMPCOMMAND2.LinkTO(RUNJUMPCOMMAND3).addPRECONDITION(PRECONDITIONS
.ENDSTATE,ACTIONS.RUNJUMP1);

        RUNJUMPCOMMAND3.LinkTO(WAITCOMMAND).addPRECONDITION(PRECONDITIONS.
BLOCK, ACTIONS.AND(ACTIONS.KO, ACTIONS.GETUP));
        //RUNJUMPCOMMAND3.LinkTO(GODOWN).addPRECONDITION(PRECONDITIONS.
GODOWN);
        RUNJUMPCOMMAND3.LinkTO(RUNJUMPCOMMAND4).addPRECONDITION(PRECONDITIONS
.ENDSTATE);

        RUNJUMPCOMMAND4.LinkTO(WAITCOMMAND).addPRECONDITION(PRECONDITIONS.
BLOCK, ACTIONS.AND(ACTIONS.KO, ACTIONS.GETUP));
        RUNJUMPCOMMAND4.LinkTO(GODOWN).addPRECONDITION(PRECONDITIONS.GODOWN);
        RUNJUMPCOMMAND4.LinkTO(WAITCOMMAND).addPRECONDITION(PRECONDITIONS.
ENDSTATE);

        GOTOABE.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.ABESAMEPOSITION,
ACTIONS.STOP);

        WAIT.LinkTO(GOTOABE).addPRECONDITION(PRECONDITIONS.ABECALLA, ACTIONS.
PASS1);
```

```
        WAIT.LinkTO(GOTOABE).addPRECONDITION(PRECONDITIONS.ABECALLB, ACTIONS.↙
AND(ACTIONS.TURN, ACTIONS.PASS1));

        WALK.LinkTO(GODOWN).addPRECONDITION(PRECONDITIONS.GODOWN);
        WALK.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.BLOCK, ACTIONS.STOP);
        WALK.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.ABENEAR, ACTIONS.    ↙
STOP);
        WALK.LinkTO(SNEAK).addPRECONDITION(PRECONDITIONS.ABESNEAK, ACTIONS.  ↙
SNEAK1);
        WALK.LinkTO(WALK).addPRECONDITION(PRECONDITIONS.ABEBEHIND, ACTIONS.  ↙
AND(ACTIONS.STOP, ACTIONS.AND(ACTIONS.TURN,ACTIONS.PASS1)));
        WALK.LinkTO(RUN).addPRECONDITION(PRECONDITIONS.ABERUN, ACTIONS.RUN1);↙


        SNEAK.LinkTO(GODOWN).addPRECONDITION(PRECONDITIONS.GODOWN);
        SNEAK.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.BLOCK, ACTIONS.     ↙
STOPSNEAK);
        SNEAK.LinkTO(WALK).addPRECONDITION(PRECONDITIONS.ABEwALK, ACTIONS.   ↙
PASS1);
        SNEAK.LinkTO(RUN).addPRECONDITION(PRECONDITIONS.ABERUN, ACTIONS.RUN1)↙
;
        SNEAK.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.ABESAMEPOSITION,    ↙
ACTIONS.STOPSNEAK);

        RUNSTART.LinkTO(RUN).addPRECONDITION(PRECONDITIONS.PORTALOPEN);
        RUNSTART.LinkTO(GODOWN).addPRECONDITION(PRECONDITIONS.GODOWN);
        RUNSTART.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.BLOCK, ACTIONS.  ↙
AND(ACTIONS.KO, ACTIONS.STANDUP));
        RUNSTART.LinkTO(RUNSTOP).addPRECONDITION(PRECONDITIONS.ABEBEHIND);
        RUNSTART.LinkTO(RUN).addPRECONDITION(PRECONDITIONS.ABERUN);
        RUNSTART.LinkTO(RUNSTOP).addPRECONDITION(PRECONDITIONS.ABERUNSTOP);

        RUNSTOP.LinkTO(GODOWN).addPRECONDITION(PRECONDITIONS.GODOWN);
        RUNSTOP.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.BLOCK, ACTIONS.AND↙
(ACTIONS.KO, ACTIONS.STANDUP));
        RUNSTOP.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.AND(PRECONDITIONS.↙
ENDSTATE,PRECONDITIONS.ABEBEHIND),ACTIONS.TURN);
        RUNSTOP.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.ENDSTATE);

        RUN.LinkTO(RUNJUMPESCAPE).addPRECONDITION(PRECONDITIONS.PORTAL);
        RUN.LinkTO(GODOWN).addPRECONDITION(PRECONDITIONS.GODOWN);
        RUN.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.BLOCK,ACTIONS.AND     ↙
(ACTIONS.KO,ACTIONS.STANDUP));
       //RUN.LinkTO(RUN).addPRECONDITION(PRECONDITIONS.ABEBEHIND, ACTIONS.AND↙
(ACTIONS.RUNSTOP, ACTIONS.AND(ACTIONS.TURN, ACTIONS.RUN1)));
        RUN.LinkTO(RUNSTOP).addPRECONDITION(PRECONDITIONS.AND(PRECONDITIONS. ↙
ABEBEHIND,PRECONDITIONS.PORTALNOTOPEN));
        RUN.LinkTO(RUNSTOP).addPRECONDITION(PRECONDITIONS.ABERUNSTOP);
        RUN.LinkTO(WALK).addPRECONDITION(PRECONDITIONS.ABEwALK, ACTIONS.     ↙
PASS1);
        RUN.LinkTO(SNEAK).addPRECONDITION(PRECONDITIONS.ABESNEAK, ACTIONS.   ↙
SNEAK1);

        SHOOTED.LinkTO(DIED).addPRECONDITION(PRECONDITIONS.ENDSTATE);

        RUNJUMPESCAPE.LinkTO(DIED).addPRECONDITION(PRECONDITIONS.ENDSTATE);

        GODOWN.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.ENDSTATE, ACTIONS. ↙
PLANE);
        //GODOWN.LinkTO(HUNG).addPRECONDITION(PRECONDITIONS.AND(PRECONDITIONS↙
.HOIST, PRECONDITIONS.ENDSTATE), ACTIONS.HUNGLONG);

        if(state == "CLEANDOWN")
            CURRENT_STATE = CLEANDOWN;
        if (state == "CLEANUP")
            CURRENT_STATE = CLEANUP;
        PREV_STATE = CURRENT_STATE;

    }
```

```
    //Esegue l'automa definito nel costruttore di MudokonAI
    //Viene chiamato nel metodo Update della classe Game1
    //[pp.48,49]
    public void ROUTINE()
    {
        if (CURRENT_STATE.name != "DIED")
        {
            ColorMask
            //La transizione da questi stati può avvenire in qualsiasi momento
            if (CURRENT_STATE == CLEANUP || CURRENT_STATE == WAITfOLLOW || CURRENT_STATE == WAIT || CURRENT_STATE == WAITCOMMAND)
            {
                HandleStateTransition
            }
            else//Tutti gli altri stati esigono che tutte le animazione contenute siano state eseguite
            {
                //L'arco è stato percorso, esegue lo stato corrente
                if (FINISHEDTransition)
                {
                    if (FINISHEDExecution)
                        precond = CURRENT_STATE.TryChangeState();
                    if (precond != null)
                        FINISHEDExecution = true;
                    else
                        FINISHEDExecution = CURRENT_STATE.Execute();
                }
                //Tutte le animazione sono state seguite
                if (FINISHEDExecution)
                {
                    CURRENT_STATE.AnimsTODO[CURRENT_STATE.CURRENT_ANIM].Reset();
                    PREV_STATE = CURRENT_STATE;
                    if (FINISHEDTransition)
                        precond = CURRENT_STATE.TryChangeState();//Controllo precondizioni

                    if (precond != null)//Almeno una precondizione è stata verificata
                    {
                        if (precond.ACTION != null)//Se una precondizione ha associata un'azione
                            FINISHEDTransition = precond.ACTION.DO();//L'azione viene eseguita
                        if (FINISHEDTransition)
                        {
                            CURRENT_STATE = precond.ARC.TO;//L'automa transita nello stato successivo
                            CURRENT_STATE.ENDSTATE = false;
                        }
                    }
                }
            }
        }
        else
        {
            //Gestione morte personaggio
            game.CurrentLevel.Mudokons.Remove(Mudokon);
            game.Abe.MUDOKONS.Remove(Mudokon);
        }
    }
}
```

```
//Definisce gli stati dell'automa
//Ogni stato contiene una o più animazioni da eseguire
//[p.45]
public class STATEm
{
    public bool ENDSTATE;
    public List<ARCm> ARCS;
    public List<Animation> AnimsTODO;
    public int CURRENT_ANIM = 0;
    public string name;
    public MudokonAI AI;
    public STATEm(string name, MudokonAI AI)
    {
        this.AI = AI;
        this.name = name;
        ENDSTATE = false;

        AnimsTODO = new List<Animation>();
        ARCS = new List<ARCm>();

        if (name == "WAIT")
        {
            AnimsTODO.Add(AI.Mudokon.WAIT);
        }
        if (name == "CLEANUP")
        {
            AnimsTODO.Add(AI.Mudokon.CLEANUP);
            AnimsTODO.Add(AI.Mudokon.CLEANUP);
            AnimsTODO.Add(AI.Mudokon.CLEANUP);
            AnimsTODO.Add(AI.Mudokon.RESTUP);
            AnimsTODO.Add(AI.Mudokon.RESTUP);
            AnimsTODO.Add(AI.Mudokon.RESTUP);
        }
        if (name == "CLEANDOWN")
        {
            AnimsTODO.Add(AI.Mudokon.CLEANDOWN);
            AnimsTODO.Add(AI.Mudokon.CLEANDOWN);
            AnimsTODO.Add(AI.Mudokon.CLEANDOWN);
            AnimsTODO.Add(AI.Mudokon.TURNdOWN);
            AnimsTODO.Add(AI.Mudokon.WAITcROUCH);
        }

        if (name == "WAITCOMMAND")
        {
            AnimsTODO.Add(AI.Mudokon.WAIT);
        }
        if (name == "JUMPCOMMAND")
        {
            AnimsTODO.Add(AI.Mudokon.JUMP_2);
            AnimsTODO.Add(AI.Mudokon.JUMP_3);
        }
        if (name == "RUNJUMPCOMMAND1")//PRENDE SPAZIO
        {
            AnimsTODO.Add(AI.Mudokon.TURN);
            AnimsTODO.Add(AI.Mudokon.PASS_1);
            AnimsTODO.Add(AI.Mudokon.PASS_2);
            //AnimsTODO.Add(AI.Mudokon.PASS_3);
            AnimsTODO.Add(AI.Mudokon.STOP);
            AnimsTODO.Add(AI.Mudokon.TURN);
        }
        if (name == "RUNJUMPCOMMAND2")//CORRE
        {
            AnimsTODO.Add(AI.Mudokon.RUN_1);
            AnimsTODO.Add(AI.Mudokon.RUN_2);
            AnimsTODO.Add(AI.Mudokon.RUN_3);
        }
        if (name == "RUNJUMPCOMMAND3")//SALTA
        {
            AnimsTODO.Add(AI.Mudokon.RUNJUMP2);
            AnimsTODO.Add(AI.Mudokon.RUNJUMP3);
```

```
}
if (name == "RUNJUMPCOMMAND4")//ATTERRA e FRENA
{
    AnimsTODO.Add(AI.Mudokon.RUNJUMP4);
    AnimsTODO.Add(AI.Mudokon.RUNsTOP_1);
    AnimsTODO.Add(AI.Mudokon.RUNsTOP_2);
}
if (name == "GOTOABE")
{
    AnimsTODO.Add(AI.Mudokon.PASS_2);
    AnimsTODO.Add(AI.Mudokon.PASS_3);
}

if (name == "WAITfOLLOW")
{
    AnimsTODO.Add(AI.Mudokon.WAIT);
}
if (name == "WALK")
{
    AnimsTODO.Add(AI.Mudokon.PASS_2);
    AnimsTODO.Add(AI.Mudokon.PASS_3);
}

if (name == "JUMP")
{
    AnimsTODO.Add(AI.Mudokon.JUMP_1);
    AnimsTODO.Add(AI.Mudokon.JUMP_2);
    AnimsTODO.Add(AI.Mudokon.JUMP_3);
    AnimsTODO.Add(AI.Mudokon.JUMP_4);
}
if (name == "RUN")
{
    AnimsTODO.Add(AI.Mudokon.RUN_4);
    AnimsTODO.Add(AI.Mudokon.RUN_5);
    AnimsTODO.Add(AI.Mudokon.RUN_2);
    AnimsTODO.Add(AI.Mudokon.RUN_3);
}
if (name == "RUNSTOP")
{
    AnimsTODO.Add(AI.Mudokon.RUNsTOP_1);
    AnimsTODO.Add(AI.Mudokon.RUNsTOP_2);
}
if (name == "RUNSTART")
{
    AnimsTODO.Add(AI.Mudokon.RUN_2);
    AnimsTODO.Add(AI.Mudokon.RUN_3);
}
if (name == "RUNJUMP")
{
    AnimsTODO.Add(AI.Mudokon.RUNJUMP1);
    AnimsTODO.Add(AI.Mudokon.RUNJUMP2);
    AnimsTODO.Add(AI.Mudokon.RUNJUMP3);
    AnimsTODO.Add(AI.Mudokon.RUNJUMP4);
}
if (name == "RUNJUMPESCAPE")
{
    AnimsTODO.Add(AI.Mudokon.RUNJUMP1);
    AnimsTODO.Add(AI.Mudokon.RUNJUMP2);
    AnimsTODO.Add(AI.Mudokon.RUNJUMP3);
}
if(name == "SNEAK")
{
    AnimsTODO.Add(AI.Mudokon.SNEAK_2);
    AnimsTODO.Add(AI.Mudokon.SNEAK_3);
}
if (name == "GODOWN")
{
    AnimsTODO.Add(AI.Mudokon.GODOWN);
}
```

```
        if (name == "SHOOTED")
        {
            AnimsTODO.Add(AI.Mudokon.DIE);
        }
        if (name == "DIED")
        {

        }
        if (name != "DIED")
        {
            this.LinkTO(AI.DIED).addPRECONDITION(AI.PRECONDITIONS.BOTOLA,AI. ↙
ACTIONS.PLANE);
            this.LinkTO(AI.DIED).addPRECONDITION(AI.PRECONDITIONS.MATTATOIO, ↙
AI.ACTIONS.DIE);
        }
    }

    //Collega due stati e restituisce l'arco associato
    public ARCm LinkTO(STATEm to)
    {
        ARCm arc = new ARCm(this.name + "to" + to.name, this, to);

        foreach (ARCm arcNew in ARCS)
        {
            if (arcNew.NAME == arc.NAME)
                return arcNew;
        }
        this.ARCS.Add(arc);
        return arc;
    }

    //Esegue tutte le animazioni contenute nello stato corrente
    public bool Execute()
    {
        if (!AnimsTODO[CURRENT_ANIM].LastFrameDone)
        {
            AI.Mudokon.AnimateMudokon(AnimsTODO[CURRENT_ANIM]);
            return false;
        }
        else
        {
            if (CURRENT_ANIM == AnimsTODO.Count - 1)
            {
                ENDSTATE = true;
                CURRENT_ANIM = 0;
            }
            else
            {
                CURRENT_ANIM++;
            }
            return true;
        }
    }

    //Controlla le precondizioni di tutti gli archi uscenti
    public PRECONDITIONm TryChangeState()
    {
        foreach (ARCm arc in AI.CURRENT_STATE.ARCS)
        {
            foreach (PRECONDITIONm precondition in arc.PRECONDITIONS)
            {
                if (precondition.Verify())
                    return precondition;
            }
        }
        return null;
    }
}
```

```
//Definisce gli archi dell'automa
//[p.47]
public class ARCm
{
    string name;
    STATEm from;
    STATEm to;

    List<PRECONDITIONm> preconditions;

    public ARCm(string name, STATEm from, STATEm to)
    {
        preconditions = new List<PRECONDITIONm>();
        this.name = name;
        this.from = from;
        this.to = to;
    }

    //Pone una precondizione sull'arco
    public void addPRECONDITION(PRECONDITIONm precondition)
    {
        PRECONDITIONm p = new PRECONDITIONm(from.AI, precondition.NAME);
        p.prec1 = precondition.prec1;
        p.prec2 = precondition.prec2;
        preconditions.Add(p);
        p.ARC = this;
    }

    //Pone una precondizione e un'azione sull'arco
    //Se la precondizione è verificata, viene eseguita l'azione
    public void addPRECONDITION(PRECONDITIONm precondition, ACTIONm action)
    {
        PRECONDITIONm p = new PRECONDITIONm(from.AI, precondition.NAME);
        p.prec1 = precondition.prec1;
        p.prec2 = precondition.prec2;
        preconditions.Add(p);
        p.addACTION(action);
        p.ARC = this;
    }
    Attribute
}
```

```
//Definisce le precondizioni che devono essere aggiunte ad un arco per
transitare da uno stato all'altro.
//[p.46]
public class PRECONDITIONm
{
    string name;
    MudokonAI AI;
    ACTIONm action;
    ARCm arc;
    public PRECONDITIONm prec1;
    public PRECONDITIONm prec2;
    public PRECONDITIONm(MudokonAI AI, string name)
    {
        this.name = name;
        this.AI = AI;
    }

    public void addACTION(ACTIONm action)
    {
        this.action = action;
    }

    //Controlla se la precondizione è soddisfatta
    public bool Verify()
    {
        if (name == "ABEWAIT")
        {
            if (AI.game.abeCONTROL.CURRENT_STATE.name == "WAIT")
                return true;
        }

        if (name == "ABEwALK")
        {
            if (AI.game.abeCONTROL.CURRENT_STATE.name == "WALK")
                return true;
        }
        if (name == "ABESNEAK")
        {
            if (AI.game.abeCONTROL.CURRENT_STATE.name == "SNEAK" || AI.game.
abeCONTROL.CURRENT_STATE.CURRENTACTION == "SNEAK1")
                return true;
        }
        if (name == "ABERUN")
        {
            if (AI.game.abeCONTROL.CURRENT_STATE.name == "RUN")
                return true;
        }
        if (name == "ABERUNSTART")
        {
            if (AI.game.abeCONTROL.CURRENT_STATE.name == "RUNSTART")
                return true;
        }
        if (name == "ABERUNSTOP")
        {
            if (AI.game.abeCONTROL.CURRENT_STATE.name == "RUNSTOP")
                return true;
        }
        if (name == "ABESAMEPOSITION")
        {
            if (AI.game.Abe.boundingBox.X == AI.Mudokon.boundingBox.X)
                return true;
        }
        if (name == "ENSsCREEN")
        {
            if ((AI.Mudokon.boundingBox.X <= 50 & AI.Mudokon.Direction == -1
& AI.Mudokon.CurrentLevel.NextLEFT == null) ||
                (AI.Mudokon.boundingBox.Right >= 500 & AI.Mudokon.Direction =
= 1 & AI.Mudokon.CurrentLevel.NextRIGHT == null))
            {
                return true;
```

```
            }
        }
        if (name == "ABENEAR")
        {
            if (AI.Mudokon.boundingBox.Intersects(AI.game.Abe.MUDOKONS[AI.
Mudokon.POSITION - 1].boundingBox))
                return true;
        }
        if (name == "ABENOTNEAR")
        {
            if (!AI.Mudokon.boundingBox.Intersects(AI.game.Abe.MUDOKONS[AI.
Mudokon.POSITION-1].boundingBox))
                return true;
        }
        if (name == "ABeHELLOB" & AI.game.CurrentLevel == AI.Mudokon.
CurrentLevel)
        {
            if (AI.game.Abe.SAYS == "HELLO" & ((AI.game.Abe.boundingBox.X >
AI.Mudokon.boundingBox.X & AI.Mudokon.Direction == -1 & AI.game.Abe.Direction
 == -1) || (AI.game.Abe.boundingBox.X < AI.Mudokon.boundingBox.X & AI.Mudokon
.Direction == 1 & AI.game.Abe.Direction == 1)))
            {
                AI.Mudokon.TURNdOWN.Reset();
                AI.game.Abe.SAYS = "";
                return true;
            }
        }
        if (name == "ABeHELLOA" & AI.game.CurrentLevel == AI.Mudokon.
CurrentLevel)
        {
            if (AI.game.Abe.SAYS == "HELLO" & ((AI.game.Abe.boundingBox.X >
AI.Mudokon.boundingBox.X & AI.Mudokon.Direction == 1 & AI.game.Abe.Direction
== -1) || (AI.game.Abe.boundingBox.X < AI.Mudokon.boundingBox.X & AI.Mudokon.
Direction == -1 & AI.game.Abe.Direction == 1)))
            {
                AI.game.Abe.SAYS = "";
                return true;
            }
        }
        if (name == "ABeHELLOLEFT" & AI.game.CurrentLevel == AI.Mudokon.
CurrentLevel)
        {
            if (AI.game.Abe.SAYS == "HELLO" & (AI.game.Abe.boundingBox.X < AI
.Mudokon.boundingBox.X))
            {
                AI.game.Abe.SAYS = "";
                return true;
            }
        }
        if (name == "ABeHELLORIGHT" & AI.game.CurrentLevel == AI.Mudokon.
CurrentLevel)
        {
            if (AI.game.Abe.SAYS == "HELLO" & (AI.game.Abe.boundingBox.X > AI
.Mudokon.boundingBox.X))
            {
                AI.game.Abe.SAYS = "";
                return true;
            }
        }
        if (name == "ABeFOLLOW" & AI.game.CurrentLevel == AI.Mudokon.
CurrentLevel)
        {
            if (AI.game.Abe.SAYS == "FOLLOW")
            {
                AI.game.Abe.SAYS = "";
                if (!AI.game.Abe.MUDOKONS.Contains(AI.Mudokon))
                {
                    AI.game.Abe.MUDOKONS.Add(AI.Mudokon);
                    AI.Mudokon.POSITION = AI.game.Abe.MUDOKONS.Count-1;
                }
```

```
                return true;
            }
        }
        if (name == "ABECOMMANDJUMP" & AI.game.CurrentLevel == AI.Mudokon.
CurrentLevel)
        {
            if (AI.game.Abe.SAYS == "COMMANDJUMP")
            {
                AI.game.Abe.SAYS = "";
                return true;
            }
        }
        if (name == "ABECOMMANDRUNJUMP" & AI.game.CurrentLevel == AI.Mudokon.
CurrentLevel)
        {
            if (AI.game.Abe.SAYS == "COMMANDRUNJUMP")
            {
                AI.game.Abe.SAYS = "";
                return true;
            }
        }
        if (name == "ABECALLB" & AI.game.CurrentLevel == AI.Mudokon.
CurrentLevel)
        {
            if (AI.game.Abe.SAYS == "FOLLOW" & ((AI.game.Abe.boundingBox.X >
AI.Mudokon.boundingBox.X & AI.Mudokon.Direction == -1 & AI.game.Abe.Direction
 == -1) || (AI.game.Abe.boundingBox.X < AI.Mudokon.boundingBox.X & AI.Mudokon
.Direction == 1 & AI.game.Abe.Direction == 1)))
            {
                if(AI.Mudokon == AI.game.Abe.MUDOKONS.Last())
                    AI.game.Abe.SAYS = "";
                return true;
            }
        }
        if (name == "ABECALLA" & AI.game.CurrentLevel == AI.Mudokon.
CurrentLevel)
        {
            if (AI.game.Abe.SAYS == "FOLLOW" & ((AI.game.Abe.boundingBox.X >
AI.Mudokon.boundingBox.X & AI.Mudokon.Direction == 1 & AI.game.Abe.Direction
== -1) || (AI.game.Abe.boundingBox.X < AI.Mudokon.boundingBox.X & AI.Mudokon.
Direction == -1 & AI.game.Abe.Direction == 1)))
            {
                if (AI.Mudokon == AI.game.Abe.MUDOKONS.Last())
                    AI.game.Abe.SAYS = "";
                return true;
            }
        }
        if (name == "ABEWAITME")
        {
            if (AI.game.Abe.SAYS == "WAITME" & AI.game.CurrentLevel == AI.
Mudokon.CurrentLevel)
            {
                if (AI.Mudokon == AI.game.Abe.MUDOKONS.Last())
                    AI.game.Abe.SAYS = "";

                if (AI.game.Abe.MUDOKONS.Contains(AI.Mudokon))
                {
                    AI.game.Abe.MUDOKONS.RemoveAt(AI.Mudokon.POSITION);

                    for (int i = 0; i < AI.game.Abe.MUDOKONS.Count; i++)
                        AI.game.Abe.MUDOKONS[i].POSITION = i;
                }
                return true;
            }
        }
        if (name == "NOTABEOPPOSITE")
        {
            if (AI.Mudokon.Direction == AI.game.Abe.Direction)
                return true;
        }
```

```
        if (name == "ABEBEHIND")
        {
            if ((AI.game.Abe.boundingBox.X < AI.Mudokon.boundingBox.X & AI.
Mudokon.Direction == 1 || AI.game.Abe.boundingBox.X > AI.Mudokon.boundingBox.
X & AI.Mudokon.Direction == -1) & AI.game.CurrentLevel == AI.Mudokon.
CurrentLevel)
                return true;
            if (AI.game.CurrentLevel == AI.Mudokon.CurrentLevel.NextLEFT & AI
.Mudokon.Direction == 1 || AI.game.CurrentLevel == AI.Mudokon.CurrentLevel.
NextRIGHT & AI.Mudokon.Direction == -1)
                return true;
        }
        if (name == "ABEAHEAD")
        {
            int dist = AI.Mudokon.boundingBox.X - AI.game.Abe.boundingBox.X;
            if (!AI.Mudokon.boundingBox.Intersects(AI.game.Abe.boundingBox) &
 ((dist < 0 & AI.Mudokon.Direction == 1) || (dist > 0 & AI.Mudokon.Direction
== -1)))
                return true;
        }

        if (name == "ENDSTATE")
        {
            if (AI.CURRENT_STATE.ENDSTATE)
            {
                if (AI.CURRENT_STATE == AI.RUNJUMP)
                {
                    AI.game.Abe.MUDOKONS.Remove(AI.Mudokon);
                    AI.game.CurrentLevel.Mudokons.Remove(AI.Mudokon);
                }
                return true;
            }
        }
        if (name == "ABEDOWN")
        {
            if (AI.game.Abe.CurrentLevel == AI.Mudokon.CurrentLevel)
            {
                if (AI.game.Abe.MUDOKONS[AI.Mudokon.POSITION - 1].boundingBox
.Y > AI.Mudokon.boundingBox.Bottom)
                    return true;
            }
            else
            {
                if (AI.game.Abe.MUDOKONS[AI.Mudokon.POSITION - 1].boundingBox
.Bottom < AI.Mudokon.boundingBox.Y)
                    return true;
            }
        }
        if (name == "ABEUP")
        {
            if (AI.game.Abe.CurrentLevel == AI.Mudokon.CurrentLevel)
            {
                if (AI.game.Abe.MUDOKONS[AI.Mudokon.POSITION - 1].boundingBox
.Bottom < AI.Mudokon.boundingBox.Y)
                    return true;
            }
            else
            {
                if (AI.game.Abe.MUDOKONS[AI.Mudokon.POSITION - 1].boundingBox
.Y > AI.Mudokon.boundingBox.Bottom)
                    return true;
            }
        }
        if (name == "ABESAMEFLOOR")
        {
            if (AI.game.Abe.CurrentLevel == AI.Mudokon.CurrentLevel)
            {
                if (AI.game.Abe.MUDOKONS[AI.Mudokon.POSITION - 1].boundingBox
.Y == AI.Mudokon.boundingBox.Y)
                    return true;
```

```
            }
        }
        if (name == "HOIST")
        {
            List<string> colTypes = new List<string>();
            colTypes = AI.Mudokon.CurrentLevel.collision(AI.Mudokon);
            if (colTypes.Contains(COLLISIONS.HOISTDESTRO) & AI.Mudokon.
Direction == 1 || colTypes.Contains(COLLISIONS.HOISTSINISTRO) & AI.Mudokon.
Direction == -1)
                return true;
        }
        if (name == "NOTHOIST")
        {
            List<string> colTypes = new List<string>();
            colTypes = AI.Mudokon.CurrentLevel.collision(AI.Mudokon);
            if (!colTypes.Contains(COLLISIONS.HOISTDESTRO) & !colTypes.
Contains(COLLISIONS.HOISTSINISTRO))
                return true;
        }
        if (name == "HOISTTURN")
        {
            List<string> colTypes = new List<string>();
            colTypes = AI.Mudokon.CurrentLevel.collision(AI.Mudokon);
            if (colTypes.Contains(COLLISIONS.HOISTDESTRO) & AI.Mudokon.
Direction == -1 || colTypes.Contains(COLLISIONS.HOISTSINISTRO) & AI.Mudokon.
Direction == 1)
                return true;
        }
        if (name == "APPIGLIO")
        {
            List<string> colTypes = new List<string>();
            colTypes = AI.Mudokon.CurrentLevel.collision(AI.Mudokon);
            if (colTypes.Contains(COLLISIONS.APPIGLIODESTRO) & AI.Mudokon.
Direction == 1 || colTypes.Contains(COLLISIONS.APPIGLIOSINISTRO) & AI.Mudokon
.Direction == -1)
                return true;
        }
        if (name == "APPIGLIOTURN")
        {
            List<string> colTypes = new List<string>();
            colTypes = AI.Mudokon.CurrentLevel.collision(AI.Mudokon);
            if (colTypes.Contains(COLLISIONS.APPIGLIODESTRO) & AI.Mudokon.
Direction == -1 || colTypes.Contains(COLLISIONS.APPIGLIOSINISTRO) & AI.
Mudokon.Direction == 1)
                return true;
        }

        if (name == "BLOCK")
        {
            List<string> colTypes = new List<string>();

            colTypes = AI.Mudokon.CurrentLevel.collision(AI.Mudokon);
            if (colTypes.Contains(COLLISIONS.BLOCCO))
                return true;
        }

        if (name == "GODOWN")
        {
            List<string> colTypes = new List<string>();

            colTypes = AI.Mudokon.CurrentLevel.collision(AI.Mudokon);
            if (colTypes.Contains(COLLISIONS.GODOWN))
                return true;
        }
        if (name == "MATTATOIO")
        {
            List<string> colTypes = new List<string>();

            colTypes = AI.game.CurrentLevel.collision(AI.Mudokon);
            if (colTypes.Contains(COLLISIONS.MATTATOIO))
```

```
                return true;
            }

            if (name == "BOTOLA")
            {
                List<string> colTypes = new List<string>();

                colTypes = AI.game.CurrentLevel.collision(AI.Mudokon);
                if (colTypes.Contains(COLLISIONS.BOTOLA))
                    return true;
            }
            if (name == "DIED")
            {
                if (AI.Mudokon.DEAD)
                {
                    AI.Mudokon.boundingBox = new Microsoft.Xna.Framework.
Rectangle(AI.Mudokon.boundingBox.X, AI.Mudokon.boundingBox.Y, 0, 0);
                    return true;
                }
            }
            if (name == "PORTAL")
            {
                List<string> colTypes = new List<string>();

                colTypes = AI.game.CurrentLevel.collision(AI.Mudokon);
                if (colTypes.Contains(COLLISIONS.PORTAL))
                    return true;
            }
            if (name == "PORTALOPEN")
            {
                if (AI.game.IsPORTALOPEN)
                    return true;
            }
            if (name == "PORTALOPENAHEAD")
            {
                if (AI.game.IsPORTALOPEN)
                {
                    ObjectA PORTAL = null;
                    foreach (ObjectA OBJ in AI.game.CurrentLevel.ObjectsA)
                    {
                        if (OBJ.NAME == "PORTAL")
                            PORTAL = OBJ;
                    }
                    if (PORTAL != null)
                    {
                        if (PORTAL.COLLISION.REC.X > AI.Mudokon.boundingBox.X &&
AI.Mudokon.Direction == 1 || PORTAL.COLLISION.REC.Right < AI.Mudokon.
boundingBox.X && AI.Mudokon.Direction == -1)
                            return true;
                    }
                }
            }
            if (name == "PORTALOPENBEHIND")
            {
                if (AI.game.IsPORTALOPEN)
                {
                    ObjectA PORTAL = null;
                    foreach (ObjectA OBJ in AI.game.CurrentLevel.ObjectsA)
                    {
                        if (OBJ.NAME == "PORTAL")
                            PORTAL = OBJ;
                    }
                    if (PORTAL != null)
                    {
                        if (PORTAL.COLLISION.REC.X > AI.Mudokon.boundingBox.X &&
AI.Mudokon.Direction == -1 || PORTAL.COLLISION.REC.Right < AI.Mudokon.
boundingBox.X && AI.Mudokon.Direction == 1)
                            return true;
                    }
                }
```

```
            }
            if (name == "PORTALRIGHT")
            {
                if (AI.game.IsPORTALOPEN)
                {
                    ObjectA PORTAL = null;
                    foreach (ObjectA OBJ in AI.game.CurrentLevel.ObjectsA)
                    {
                        if (OBJ.NAME == "PORTAL")
                            PORTAL = OBJ;
                    }
                    if (PORTAL != null)
                    {
                        if (PORTAL.COLLISION.REC.X > AI.Mudokon.boundingBox.X)
                            return true;
                    }
                }
            }
            if (name == "PORTALLEFT")
            {
                if (AI.game.IsPORTALOPEN)
                {
                    ObjectA PORTAL = null;
                    foreach (ObjectA OBJ in AI.game.CurrentLevel.ObjectsA)
                    {
                        if (OBJ.NAME == "PORTAL")
                            PORTAL = OBJ;
                    }
                    if (PORTAL != null)
                    {
                        if (PORTAL.COLLISION.REC.X < AI.Mudokon.boundingBox.X)
                            return true;
                    }
                }
            }
            if (name == "AND")
            {
                return this.prec1.Verify() & this.prec2.Verify();
            }

            return false;
        }

        Attribute
    }

    //Contiene i riferimenti a tutte le istanze della classe PRECONDITIONm
    public class PRECONDITIONSm...
```

```
//Definisce le azioni che possono essere associate ad una precondizione
//Ogni azione può contenere una o più precondizioni.
//[p.45]
public class ACTIONm
{
    MudokonAI AI;
    string name;
    public List<Animation> AnimsTODO;
    int CURRENT_ANIM = 0;
    public ACTIONm action1;
    public ACTIONm action2;

    public ACTIONm(MudokonAI AI, string name)
    {
        AnimsTODO = new List<Animation>();
        this.AI = AI;
        this.name = name;

        if (name == "STOP_TURN_PASS1")
        {
            AnimsTODO.Add(AI.Mudokon.STOP);
            AnimsTODO.Add(AI.Mudokon.TURN);
            AnimsTODO.Add(AI.Mudokon.PASS_1);
        }
        if (name == "TURN")
        {
            AnimsTODO.Add(AI.Mudokon.TURN);
        }
        if (name == "TURNDOWN")
        {
            AnimsTODO.Add(AI.Mudokon.TURNdOWN);
        }
        if (name == "STANDUP")
        {
            AnimsTODO.Add(AI.Mudokon.STANDUP);
        }
        if (name == "CLEANUPSTOPLEFT")
        {
            AnimsTODO.Add(AI.Mudokon.CLEANUPSTOPLEFT);
        }
        if (name == "CLEANUPSTOPRIGHT")
        {
            AnimsTODO.Add(AI.Mudokon.CLEANUPSTOPRIGHT);
        }
        if (name == "CROUCH")
        {
            AnimsTODO.Add(AI.Mudokon.CROUCH);
        }
        if (name == "STOP")
        {
            AnimsTODO.Add(AI.Mudokon.STOP);
        }
        if (name == "STOPRUNJUMP")
        {
            AnimsTODO.Add(AI.Mudokon.RUNJUMP4);
        }
        if (name == "PASS1")
        {
            AnimsTODO.Add(AI.Mudokon.PASS_1);
        }
        if (name == "RUNJUMP1")
        {
            AnimsTODO.Add(AI.Mudokon.RUNJUMP1);
        }
        if (name == "STOPSNEAK_TURN_SNEAK1")
        {
            AnimsTODO.Add(AI.Mudokon.SNEAKsTOP);
            AnimsTODO.Add(AI.Mudokon.TURN);
            AnimsTODO.Add(AI.Mudokon.SNEAK_1);
        }
```

```
if (name == "STOPSNEAK")
{
    AnimsTODO.Add(AI.Mudokon.SNEAKsTOP);
}

if (name == "SNEAK1")
{
    AnimsTODO.Add(AI.Mudokon.SNEAK_1);
}
if (name == "RUN1")
{
    AnimsTODO.Add(AI.Mudokon.RUN_1);
    AnimsTODO.Add(AI.Mudokon.RUN_2);
}
if (name == "RUNSTOP")
{
    AnimsTODO.Add(AI.Mudokon.RUNsTOP_1);
    AnimsTODO.Add(AI.Mudokon.RUNsTOP_2);
}
if (name == "TURN_HELLO")
{
    AnimsTODO.Add(AI.Mudokon.TURN);
    AnimsTODO.Add(AI.Mudokon.HELLO);
}
if (name == "HELLO")
{
    AnimsTODO.Add(AI.Mudokon.HELLO);
}
if (name == "OK")
{
    AnimsTODO.Add(AI.Mudokon.OK);
}
if (name == "PLANE")
{
    AnimsTODO.Add(AI.Mudokon.PLANE);
}
if (name == "HOISTUP")
{
    AnimsTODO.Add(AI.Mudokon.HOISTUP);
}
if (name == "HOISTDOWN")
{
    AnimsTODO.Add(AI.Mudokon.HOISTDOWN);
}
if (name == "JUMPUP")
{
    AnimsTODO.Add(AI.Mudokon.JUMPUP);
}
if (name == "JUMPDOWN")
{
    AnimsTODO.Add(AI.Mudokon.JUMPDOWN);
}
if (name == "JUMP1")
{
    AnimsTODO.Add(AI.Mudokon.JUMP_1);
}
if (name == "JUMP4")
{
    AnimsTODO.Add(AI.Mudokon.JUMP_4);
}
if (name == "KO")
{
    AnimsTODO.Add(AI.Mudokon.KO);
}
if (name == "GETUP")
{
    AnimsTODO.Add(AI.Mudokon.GETUP);
}
if (name == "DIE")
{
```

```
                AnimsTODO.Add(AI.Mudokon.DIE);
            }
        }

        //Metodo chiamato per eseguire tutte le animazioni contenute in un'azione
        public bool DO()
        {
            if (!AnimsTODO[CURRENT_ANIM].LastFrameDone)
            {
                AI.Mudokon.AnimateMudokon(AnimsTODO[CURRENT_ANIM]);
                return false;
            }
            else
            {
                if (CURRENT_ANIM == AnimsTODO.Count - 1)
                {
                    CURRENT_ANIM = 0;
                    AI.CURRENT_STATE.CURRENT_ANIM = 0;
                    return true;
                }
                else
                {
                    CURRENT_ANIM++;
                }
                return false;
            }
        }
        Attribute
    }

    //Contiene i riferimenti a tutte le istanze della classe ACTIONm
    public class ACTIONSm...
```

```
using ...

namespace TheGame
{
    public class SligFactory
    {
        VARS

        //Questa classe serve per caricare solo una volta le risorse (sprites, audio)
        //I riferiemnti alle risorse vengono acceduti da ogni istanza delle classi Slig e SligAI
        public SligFactory(Game1 game)
        {
            this.game = game;

            Pass1_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\PASS1_R_220");
            Pass1_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\PASS1_L_220");
            Pass2_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\PASS2_R_220");
            Pass2_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\PASS2_L_220");
            Pass3_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\PASS3_R_220");
            Pass3_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\PASS3_L_220");
            PassStop_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\PASSsTOP_R_220");
            PassStop_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\PASSsTOP_L_220");
            OnePass_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\ONEPASS_R_220");
            OnePass_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\ONEPASS_L_220");
            Run1_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\RUN1_R_300");
            Run1_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\RUN1_L_300");
            Run2_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\RUN2_R_300");
            Run2_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\RUN2_L_300");
            Run3_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\RUN3_R_300");
            Run3_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\RUN3_L_300");
            Run4_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\RUN4_R_300");
            Run4_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\RUN4_L_300");
            RunStart_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\RUNsTART_R_300");
            RunStart_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\RUNsTART_L_300");
            RunStop1_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\RUNsTOP1_R_300");
            RunStop1_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\RUNsTOP1_L_300");
            RunStop2_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\RUNsTOP2_R_300");
            RunStop2_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\RUNsTOP2_L_300");
            Shoot_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\SHOOT_R_220");
            Shoot_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\SHOOT_L_220");
            ShootStart_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\SHOOTsTART_R_220");
```

```
            ShootStart_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\
    SHOOTsTART_L_220");
            ShootStop_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\
    SHOOTsTOP_R_220");
            ShootStop_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\
    SHOOTsTOP_L_220");
            Start_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\
    START_R_220");
            Start_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\
    START_L_220");
            Turn_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\TURN_R_100
    ");
            Turn_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\TURN_L_100
    ");
            Wait_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\WAIT_R_220
    ");
            Wait_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\WAIT_L_220
    ");

            Wake_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\WAKE_R_220
    ");
            Wake_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\WAKE_L_220
    ");
            Sleep_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\
    SLEEP_R_220");
            Sleep_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\
    SLEEP_L_220");

            DownPlane_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\
    DOWNPLANE_R_150");
            DownPlane_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\
    DOWNPLANE_L_150");
            Splat_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\
    SPLAT_R_130");
            Splat_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\
    SPLAT_L_130");
            Possessed_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\
    POSSESSED_R_130");
            Possessed_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\
    POSSESSED_L_130");
            Ouch_R = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\OUCH_R_180
    ");
            Ouch_L = game.Content.Load<Texture2D>("Sprite_Sheet\\Slig\\OUCH_L_180
    ");


            Sound_PASS = game.Content.Load<SoundEffect>("Sound\\SLIG_PASS");
            Sound_TURN = game.Content.Load<SoundEffect>("Sound\\SLIG_TURN");
            Sound_WHAAT = game.Content.Load<SoundEffect>("Sound\\SLIG_WHAAT");
            Sound_FREEZE = game.Content.Load<SoundEffect>("Sound\\SLIG_FREEZE");
            Sound_SPLAT = game.Content.Load<SoundEffect>("Sound\\SPLAT");
            Sound_HELP = game.Content.Load<SoundEffect>("Sound\\HEEELP");
            Sound_SHOOT = game.Content.Load<SoundEffect>("Sound\\SHOOT2");
            Sound_POSSESSED = game.Content.Load<SoundEffect>("Sound\\
    JUSTPOSSESSED");

            sLIG_BBOX_PNG = game.Content.Load<Texture2D>("Sprite_Sheet\\Abe\\
    ABE_BBOX");
        }
    }
}
```

```
using ...

namespace TheGame
{
    //Gestisce grafica ed effetti sonori di uno Slig
    public class Slig : Character
    {
        VARS

        public Slig(Game1 game, string state,int AI_Level, Point position)
        {
            this.game = game;
            this.state = state;
            this.AI_LEVEL = AI_Level;
            this.position = position;

            Sound_PASS = game.SligFactory.Sound_PASS;
            Sound_TURN = game.SligFactory.Sound_TURN;
            Sound_WHAAT = game.SligFactory.Sound_WHAAT;
            Sound_FREEZE = game.SligFactory.Sound_FREEZE;
            Sound_SPLAT = game.SligFactory.Sound_SPLAT;
            Sound_HELP = game.SligFactory.Sound_HELP;
            Sound_SHOOT = game.SligFactory.Sound_SHOOT;
            Sound_POSSESSED = game.SligFactory.Sound_POSSESSED;

            START = new Animation(game.SligFactory.Start_R, game.SligFactory.
    Start_L, AnimateRate, 4, "START", 0, 0, -1);
            PASS1 = new Animation(game.SligFactory.Pass1_R, game.SligFactory.
    Pass1_L, AnimateRate, 6, "PASS1", new Sound(Sound_PASS, 6), 2, 0, 1, 0, 43);
            PASS2 = new Animation(game.SligFactory.Pass2_R, game.SligFactory.
    Pass2_L, AnimateRate, 6, "PASS2", 2, 0, 1, 43, 86);
            PASS3 = new Animation(game.SligFactory.Pass3_R, game.SligFactory.
    Pass3_L, AnimateRate, 6, "PASS3", new Sound(Sound_PASS, 4), 1, 0, -1, 86, 86)
    ;
            PASSsTOP = new Animation(game.SligFactory.PassStop_R, game.
    SligFactory.PassStop_L, AnimateRate, 5, "PASSsTOP", 0, 0, -1, 86, 86);
            ONEPASS = new Animation(game.SligFactory.OnePass_R, game.SligFactory.
    OnePass_L, AnimateRate, 8, "ONEPASS", 0, 0, -1, 0, 43);
            RUN1 = new Animation(game.SligFactory.Run1_R, game.SligFactory.Run1_L
    , AnimateRate, 4, "RUN1", new Sound(Sound_PASS, 4), 4, 0, 4, 0, 43);
            RUN2 = new Animation(game.SligFactory.Run2_R, game.SligFactory.Run2_L
    , AnimateRate, 4, "RUN2", 2, 0, -1, 43, 86);
            RUN3 = new Animation(game.SligFactory.Run3_R, game.SligFactory.Run3_L
    , AnimateRate, 4, "RUN3", new Sound(Sound_PASS, 4), 4, 0, 4, 86, 129);
            RUN4 = new Animation(game.SligFactory.Run4_R, game.SligFactory.Run4_L
    , AnimateRate, 4, "RUN4", 2, 0, -1, 129, 172);
            RUNsTART = new Animation(game.SligFactory.RunStart_R, game.
    SligFactory.RunStart_L, AnimateRate, 4, "RUNsTART", 0, 0, -1, 0, 0);
            RUNsTOP1 = new Animation(game.SligFactory.RunStop1_R, game.
    SligFactory.RunStop1_L, AnimateRate, 6, "RUNsTOP1", 0, 0, -1, 86, 129);
            HEELP = new Animation(game.SligFactory.RunStop1_R, game.SligFactory.
    RunStop1_L, AnimateRate, 6, "HEELP", new Sound(Sound_HELP, 4), 0, 0, -1, 86,
    129);
            RUNsTOP2 = new Animation(game.SligFactory.RunStop2_R, game.
    SligFactory.RunStop2_L, AnimateRate, 7, "RUNsTOP2", 0, 0, -1, 129, 172);
            SHOOT = new Animation(game.SligFactory.Shoot_R, game.SligFactory.
    Shoot_L, AnimateRate, 4, "SHOOT", new Sound(Sound_SHOOT, 1), 0, 0, -1, 0, 0);
            SHOOTsTART = new Animation(game.SligFactory.ShootStart_R, game.
    SligFactory.ShootStart_L, AnimateRate, 3, "SHOOtsTART", 0, 0, -1);
            SHOOTsTOP = new Animation(game.SligFactory.ShootStop_R, game.
    SligFactory.ShootStop_L, AnimateRate, 6, "SHOOTsTOP", 0, 0, -10);
            WAIT = new Animation(game.SligFactory.Wait_R, game.SligFactory.Wait_L
    , 1000 / 7, 7, "WAIT", 0, 0, -1, 0, 0);
            JUSTPOSSESSED = new Animation(game.SligFactory.Wait_R, game.
    SligFactory.Wait_L, 1000 / 7, 7, "JUSTPOSSESSED",new Sound(Sound_POSSESSED,
    1), 0, 0, -1, 0, 0);
            WHAAT = new Animation(game.SligFactory.Wait_R, game.SligFactory.
    Wait_L, AnimateRate, 7, "WHAAT", new Sound(Sound_WHAAT, 1), 0, 0, -1, 0, 0);
            FREEZE = new Animation(game.SligFactory.Wait_R, game.SligFactory.
    Wait_L, 1000 / 7, 7, "FREEZE", new Sound(Sound_FREEZE, 1), 0, 0, -1, 0, 0);
```

```
        TURN = new Animation(game.SligFactory.Turn_L, game.SligFactory.Turn_R
, AnimateRate, 11, "TURN", new Sound(Sound_TURN, 6), 0, 0, -1, 0, 0);
        SLEEP = new Animation(game.SligFactory.Sleep_R, game.SligFactory.
Sleep_L, 1000 / 8, 8, "SLEEP", 0, 0, -1, 70, 70);
        WAKE = new Animation(game.SligFactory.Wake_R, game.SligFactory.Wake_L
, AnimateRate, 18, "WAKE", 0, 0, -1, 70, 70);
        DOWNPLANE = new Animation(game.SligFactory.DownPlane_R, game.
SligFactory.DownPlane_L, AnimateRate, 17, "DOWNPLANE", 0, 0, -1, 0, 86,200,0)
;
        SPLAT = new Animation(game.SligFactory.Splat_R, game.SligFactory.
Splat_R, AnimateRate, 7, "DOWNPLANE",new Sound(Sound_SPLAT, 1), 0, 0, -1, 40,
 40);
        POSSESSED = new Animation(game.SligFactory.Possessed_R, game.
SligFactory.Possessed_L, AnimateRate, 8, "POSSESSED", 0, 0, -1, 40, 40);
        SUICIDE = new Animation(game.SligFactory.Possessed_R, game.
SligFactory.Possessed_L, AnimateRate, 8, "SUICIDE", 0, 0, -1, 40, 40);
        OUCH = new Animation(game.SligFactory.Ouch_R, game.SligFactory.Ouch_L
, AnimateRate, 22, "OUCH", 0, 0, -1, 43, 43);

        AI = new SligAI(this, game, state);
        Direction = 1;
        CurrentAnimation = AI.CURRENT_STATE.AnimsTODO[AI.CURRENT_STATE.
CURRENT_ANIM];
        PrevAnimation = START;
        color = Color.White;
        if(state == "SLEEP")
            DrawingRec = new Rectangle(position.X - 100, position.Y -
CurrentAnimation.SpriteHeight, CurrentAnimation.SpriteWidth, CurrentAnimation
.SpriteHeight);//-18+43, 295,
        if (state == "SENTINELLA" || state == "WAIT")
            DrawingRec = new Rectangle(position.X - 30, position.Y -
CurrentAnimation.SpriteHeight, CurrentAnimation.SpriteWidth, CurrentAnimation
.SpriteHeight);

        boundingBox = new Rectangle(position.X, position.Y - 110, 44, 110);//
 + 43*2

        defaultDrawing = new Point(DrawingRec.Location.X, DrawingRec.Location
.Y);
        defaultBounding = new Point(boundingBox.Location.X, boundingBox.
Location.Y);

        if(AI_LEVEL == 1)
            EYE_CAPACITY = 100;
        if (AI_LEVEL == 2)
            EYE_CAPACITY = 200;
        if (AI_LEVEL == 3)
            EYE_CAPACITY = 300;
        if (AI_LEVEL == 4)
            EYE_CAPACITY = 640;
        SligEye = new Rectangle(boundingBox.X + boundingBox.Width / 2 + (320
* (Direction - 1)), boundingBox.Y, EYE_CAPACITY, 10);
        SligEar = new Rectangle(0,0,640,480);
        sLIG_BBOX_PNG = game.SligFactory.sLIG_BBOX_PNG;
    }

    public void UpdateEyeAndEar()...

    //Gestisce la spritesheet dell'animazione corrente.
    //Definisce quale frame della spritesheet deve essere disegnato.
    public void AnimateSLIG(Animation anim)
    {
        UpdateEyeAndEar();
        if (anim.CurrentFrame == 0)
        {
            PrevAnimation = CurrentAnimation;
            PrevAnimation.Reset();
            CurrentAnimation = anim;

            if (Direction == 1)
```

```
                    DrawingRec.X = DrawingRec.X + (PrevAnimation.xEND -
    CurrentAnimation.xSTART);
                else
                    DrawingRec.X = DrawingRec.X + ((PrevAnimation.SpriteLWidth -
    PrevAnimation.xEND) - (CurrentAnimation.SpriteLWidth - CurrentAnimation.
    xSTART));

                DrawingRec.Y = DrawingRec.Y - (PrevAnimation.yEND -
    CurrentAnimation.ySTART) + (PrevAnimation.SpriteHeight - CurrentAnimation.
    SpriteHeight);

                if (CurrentAnimation == TURN)
                {
                    Direction = Direction * -1;
                }

                if (Direction == 1)
                {
                    anim.SpriteWidth = anim.SpriteRWidth;
                    anim.SpriteHeight = anim.SpriteRHeight;
                    anim.Sheet = anim.SheetR;
                }
                else
                {
                    anim.SpriteWidth = anim.SpriteLWidth;
                    anim.SpriteHeight = anim.SpriteLHeight;
                    anim.Sheet = anim.SheetL;
                }
                DrawingRec.Width = anim.SpriteWidth;
                DrawingRec.Height = anim.SpriteHeight;

                this.game.spriteCurrent = anim.Sheet;
            }
            if (anim.Time > anim.AnimateRate)
            {
                if (anim.CurrentFrame == 1)
                {
                    boundingBox.X += (anim.xEND - anim.xSTART) * Direction;
                    boundingBox.Y = DrawingRec.Bottom - (anim.yEND + boundingBox.
    Height);
                }
                anim.Time = 0f;
                if (anim.Effect != null && anim.CurrentFrame == anim.Effect.
    FrameToPlay)
                    anim.Effect.Play();

                if (anim.CurrentFrame == anim.FrameCount)
                {
                    anim.LastFrameDone = true;

                }
                else
                    anim.LastFrameDone = false;

                if (anim.CurrentFrame < anim.FrameCount)
                {
                    anim.SourceRect = new Rectangle(anim.CurrentFrame * anim.
    SpriteWidth, 0, anim.SpriteWidth, anim.SpriteHeight);
                    this.game.sourceRect = anim.SourceRect;
                    anim.Forward();
                }
            }
        }
        Attribute
    }
}
```

```
//SligAI definisce l'automa per la gestione del comportamento
//di uno Slig
//[pp.43]
public class SligAI
{
    Vars

    public SligAI(Slig Slig, Game1 game,string state)
    {
        this.game = game;
        this.Slig = Slig;

        PRECONDITIONS = new PRECONDITIONS(this);
        ACTIONS = new ACTIONS(this);

        WAIT = new STATE("WAIT",this);
        SENTINELLA = new STATE("SENTINELLA", this);
        GOBACKHOME = new STATE("GOBACKHOME", this);
        GOBACKHOME2 = new STATE("GOBACKHOME2", this);
        ONEPASS = new STATE("ONEPASS", this);
        VEDETTA = new STATE("VEDETTA", this);
        VEDETTA2 = new STATE("VEDETTA2", this);
        SHOOT = new STATE("SHOOT", this);
        RUNNER = new STATE("RUNNER", this);
        RUNSTOP = new STATE("RUNSTOP", this);
        SLEEP = new STATE("SLEEP", this);
        DEAD = new STATE("DEAD", this);
        POSSESSED = new STATE("POSSESSED", this);
        SUICIDE = new STATE("SUICIDE", this);
        SHOOTED = new STATE("SHOOTED", this);

        JUSTPOSSESSED =  new STATE("JUSTPOSSESSED", this);
        WAITCONTROL = new STATE("WAITCONTROL", this);
        SENTINELLACONTROL = new STATE("SENTINELLACONTROL", this);
        RUNNERCONTROL = new STATE("RUNNERCONTROL", this);
        RUNSTOPCONTROL = new STATE("RUNSTOPCONTROL", this);
        SHOOTCONTROL = new STATE("SHOOTCONTROL", this);

        CRAZY1 = new STATE("CRAZY1", this);
        CRAZY2 = new STATE("CRAZY2", this);

        //////////////SLIG  AI///////////
        SLEEP.LinkTO(SENTINELLA).addPRECONDITION(PRECONDITIONS.AND
(PRECONDITIONS.ABENOISE,PRECONDITIONS.ABEAHEAD), ACTIONS.AND(ACTIONS.WAKE,
ACTIONS.WHAAT));
        SLEEP.LinkTO(SENTINELLA).addPRECONDITION(PRECONDITIONS.AND
(PRECONDITIONS.ABENOISE, PRECONDITIONS.ABEBEHIND), ACTIONS.AND(ACTIONS.WAKE,
ACTIONS.AND(ACTIONS.WHAAT, ACTIONS.AND(ACTIONS.TURN,ACTIONS.START))));
        SLEEP.LinkTO(CRAZY1).addPRECONDITION(PRECONDITIONS.ABEOMM,ACTIONS.
WAKE);
        WAIT.LinkTO(SENTINELLA).addPRECONDITION(PRECONDITIONS.ABENOISE,
ACTIONS.TURN);
        WAIT.LinkTO(SENTINELLA).addPRECONDITION(PRECONDITIONS.ENDSTATE,
ACTIONS.START);

        SENTINELLA.LinkTO(SHOOT).addPRECONDITION(PRECONDITIONS.LOOKATMUDOKON,
 ACTIONS.AND(ACTIONS.STOP,ACTIONS.FREEZE));
        SENTINELLA.LinkTO(SENTINELLA).addPRECONDITION(PRECONDITIONS.AND
(PRECONDITIONS.ABENOISE, PRECONDITIONS.ABEBEHIND), ACTIONS.AND(ACTIONS.AND
(ACTIONS.STOP, ACTIONS.AND(ACTIONS.WHAAT, ACTIONS.TURN)), ACTIONS.START));
        SENTINELLA.LinkTO(SENTINELLA).addPRECONDITION(PRECONDITIONS.AND
(PRECONDITIONS.ABENOISE, PRECONDITIONS.ABEAHEAD), ACTIONS.AND(ACTIONS.AND
(ACTIONS.STOP,ACTIONS.WHAAT), ACTIONS.AND(ACTIONS.WAIT,ACTIONS.START)));
        SENTINELLA.LinkTO(SENTINELLA).addPRECONDITION(PRECONDITIONS.BLOCK,
ACTIONS.AND(ACTIONS.AND(ACTIONS.STOP, ACTIONS.WAIT.REPEAT(1)), ACTIONS.TURN))
;
        SENTINELLA.LinkTO(DEAD).addPRECONDITION(PRECONDITIONS.GODOWND,
ACTIONS.DOWNPLANE);
        SENTINELLA.LinkTO(SENTINELLA).addPRECONDITION(PRECONDITIONS.ENDSCREEN
, ACTIONS.AND(ACTIONS.AND(ACTIONS.STOP, ACTIONS.WAIT.REPEAT(1)), ACTIONS.
```

```
TURN));
        SENTINELLA.LinkTO(DEAD).addPRECONDITION(PRECONDITIONS.MATTATOIO,
ACTIONS.SPLAT);
        SENTINELLA.LinkTO(CRAZY1).addPRECONDITION(PRECONDITIONS.ABEOMM);

        //SHOOT.LinkTO(RUNNER).addPRECONDITION(PRECONDITIONS.ENDSTATE,
ACTIONS.AND(ACTIONS.SHOOTSTOP, ACTIONS.RUNSTART));
        SHOOT.LinkTO(RUNNER).addPRECONDITION(PRECONDITIONS.NOTLOOKATABE,
ACTIONS.AND(ACTIONS.SHOOTSTOP,ACTIONS.RUNSTART));//Aggiungere SligWhAaaat
        SHOOT.LinkTO(RUNNER).addPRECONDITION(PRECONDITIONS.ABENOTSAMELVL,
ACTIONS.AND(ACTIONS.SHOOTSTOP, ACTIONS.RUNSTART));//Aggiungere SligWhAaaat
        SHOOT.LinkTO(SENTINELLA).addPRECONDITION(PRECONDITIONS.ABEKILLED,
ACTIONS.START);
        //SHOOT.LinkTO(WAIT).addPRECONDITION(PRECONDITIONS.ENDSTATE);
        //SHOOT.LinkTO(SHOOT).addPRECONDITION(PRECONDITIONS.ENDSTATE, ACTIONS
.AND(ACTIONS.SHOOTSTOP, ACTIONS.SHOOTSTART));

        RUNNER.LinkTO(SENTINELLA).addPRECONDITION(PRECONDITIONS.BLOCK,
ACTIONS.OUCH);
        RUNNER.LinkTO(RUNNER).addPRECONDITION(PRECONDITIONS.AND(PRECONDITIONS
.ABENOTSAMELVL,PRECONDITIONS.ABEBEHIND),ACTIONS.AND(ACTIONS.AND(ACTIONS.
RUNSTOP,ACTIONS.TURN),ACTIONS.RUNSTART));
        RUNNER.LinkTO(RUNSTOP).addPRECONDITION(PRECONDITIONS.AND
(PRECONDITIONS.ENDSTATE, PRECONDITIONS.ABESAMELVL));

        //RUNNER.LinkTO(RUNSTOP).addPRECONDITION(PRECONDITIONS.AND
(PRECONDITIONS.LVLCHANGED,PRECONDITIONS.AND(PRECONDITIONS.ENDSTATE,
PRECONDITIONS.ABESAMELVL)));
        //RUNNER.LinkTO(RUNSTOP).addPRECONDITION(PRECONDITIONS.LOOKATMUDOKON)
;
        //RUNNER.LinkTO(RUNSTOP).addPRECONDITION(PRECONDITIONS.BLOCK);
        //RUNNER.LinkTO(RUNSTOP).addPRECONDITION(PRECONDITIONS.ENDSCREEN);
        //RUNNER.LinkTO(RUNNER).addPRECONDITION(PRECONDITIONS.GODOWN, ACTIONS
.AND(ACTIONS.RUNSTOP, ACTIONS.AND(ACTIONS.TURN, ACTIONS.RUNSTART)));
        RUNSTOP.LinkTO(SENTINELLA).addPRECONDITION(PRECONDITIONS.BLOCK,
ACTIONS.OUCH);
        RUNSTOP.LinkTO(VEDETTA2).addPRECONDITION(PRECONDITIONS.AND
(PRECONDITIONS.ENDSTATE,PRECONDITIONS.NOTATHOME));
        RUNSTOP.LinkTO(VEDETTA).addPRECONDITION(PRECONDITIONS.AND
(PRECONDITIONS.ENDSTATE, PRECONDITIONS.ATHOME));

        VEDETTA2.LinkTO(GOBACKHOME).addPRECONDITION(PRECONDITIONS.AND
(PRECONDITIONS.ENDSTATE,PRECONDITIONS.ABESAMELVL), ACTIONS.TURN);
        VEDETTA2.LinkTO(GOBACKHOME2).addPRECONDITION(PRECONDITIONS.AND
(PRECONDITIONS.ENDSTATE, PRECONDITIONS.ABENOTSAMELVL));
        VEDETTA2.LinkTO(SHOOT).addPRECONDITION(PRECONDITIONS.LOOKATMUDOKON,
ACTIONS.SHOOTSTART);

        VEDETTA.LinkTO(SHOOT).addPRECONDITION(PRECONDITIONS.LOOKATMUDOKON,
ACTIONS.SHOOTSTART);
        VEDETTA.LinkTO(SENTINELLA).addPRECONDITION(PRECONDITIONS.AND
(PRECONDITIONS.AND(PRECONDITIONS.ATHOME, PRECONDITIONS.ENDSTATE),
PRECONDITIONS.NOTLOOKATABE), ACTIONS.START);
        VEDETTA.LinkTO(GOBACKHOME).addPRECONDITION(PRECONDITIONS.AND
(PRECONDITIONS.NOTATHOME,PRECONDITIONS.ENDSTATE), ACTIONS.TURN);
        VEDETTA.LinkTO(SENTINELLA).addPRECONDITION(PRECONDITIONS.
ABENOTSAMELVL);

        CRAZY2.LinkTO(VEDETTA).addPRECONDITION(PRECONDITIONS.AND
(PRECONDITIONS.ENDSTATE, PRECONDITIONS.ABENOTOMM));
        CRAZY1.LinkTO(SHOOT).addPRECONDITION(PRECONDITIONS.LOOKATMUDOKON,
ACTIONS.AND(ACTIONS.STOP, ACTIONS.FREEZE));
        CRAZY1.LinkTO(CRAZY1).addPRECONDITION(PRECONDITIONS.BLOCK,ACTIONS.AND
(ACTIONS.OUCH,ACTIONS.AND(ACTIONS.TURN,ACTIONS.RUNSTART)));
        CRAZY1.LinkTO(CRAZY2).addPRECONDITION(PRECONDITIONS.ENDSCREEN);
        CRAZY1.LinkTO(CRAZY2).addPRECONDITION(PRECONDITIONS.ENDSTATE);
        CRAZY2.LinkTO(POSSESSED).addPRECONDITION(PRECONDITIONS.AND
(PRECONDITIONS.ENDSTATE, PRECONDITIONS.ABECONTROL));
        CRAZY2.LinkTO(CRAZY1).addPRECONDITION(PRECONDITIONS.ENDSTATE, ACTIONS
.AND(ACTIONS.TURN, ACTIONS.RUNSTART));
```

```
        SHOOTED.LinkTO(DEAD).addPRECONDITION(PRECONDITIONS.ENDSTATE, ACTIONS.↙
SPLAT);

        ///////////////SLIG  CONTROL///////////////
        POSSESSED.LinkTO(JUSTPOSSESSED).addPRECONDITION(PRECONDITIONS.       ↙
ENDSTATE);
        JUSTPOSSESSED.LinkTO(WAITCONTROL).addPRECONDITION(PRECONDITIONS.     ↙
ENDSTATE);

        WAITCONTROL.LinkTO(WAITCONTROL).addPRECONDITION(PRECONDITIONS.TURN,  ↙
ACTIONS.TURN);
        WAITCONTROL.LinkTO(WAITCONTROL).addPRECONDITION(PRECONDITIONS.BLOCK);
        WAITCONTROL.LinkTO(RUNNERCONTROL).addPRECONDITION(PRECONDITIONS.RUN);
        WAITCONTROL.LinkTO(SENTINELLACONTROL).addPRECONDITION(PRECONDITIONS. ↙
WALK,ACTIONS.START);
        WAITCONTROL.LinkTO(ONEPASS).addPRECONDITION(PRECONDITIONS.ONEPASS);
        WAITCONTROL.LinkTO(SUICIDE).addPRECONDITION(PRECONDITIONS.ZERO);
        WAITCONTROL.LinkTO(SHOOTCONTROL).addPRECONDITION(PRECONDITIONS.ZETA, ↙
ACTIONS.SHOOTSTART);

        SHOOTCONTROL.LinkTO(WAITCONTROL).addPRECONDITION(PRECONDITIONS.ZETAUP↙
,ACTIONS.SHOOTSTOP);
        SHOOTCONTROL.LinkTO(SHOOTCONTROL).addPRECONDITION(PRECONDITIONS.     ↙
KILLER);

        SUICIDE.LinkTO(WAITCONTROL).addPRECONDITION(PRECONDITIONS.ZEROUP);
        SUICIDE.LinkTO(DEAD).addPRECONDITION(PRECONDITIONS.ENDSTATE,ACTIONS. ↙
SPLAT);

        ONEPASS.LinkTO(WAITCONTROL).addPRECONDITION(PRECONDITIONS.BLOCK);
        ONEPASS.LinkTO(WAITCONTROL).addPRECONDITION(PRECONDITIONS.WALKSTOP);

        SENTINELLACONTROL.LinkTO(WAITCONTROL).addPRECONDITION(PRECONDITIONS. ↙
BLOCK);
        SENTINELLACONTROL.LinkTO(WAITCONTROL).addPRECONDITION(PRECONDITIONS. ↙
AND(PRECONDITIONS.ENDSTATE,PRECONDITIONS.WALKSTOP),ACTIONS.STOP);
        SENTINELLACONTROL.LinkTO(SENTINELLACONTROL).addPRECONDITION          ↙
(PRECONDITIONS.AND(PRECONDITIONS.ENDSTATE, PRECONDITIONS.WALK), ACTIONS.     ↙
PASS3);

        RUNNERCONTROL.LinkTO(RUNSTOPCONTROL).addPRECONDITION(PRECONDITIONS.  ↙
WALKSTOP);
        RUNNERCONTROL.LinkTO(WAITCONTROL).addPRECONDITION(PRECONDITIONS.BLOCK↙
,ACTIONS.OUCH);

        RUNSTOPCONTROL.LinkTO(WAITCONTROL).addPRECONDITION(PRECONDITIONS.    ↙
ENDSTATE);
        RUNSTOPCONTROL.LinkTO(WAITCONTROL).addPRECONDITION(PRECONDITIONS.    ↙
BLOCK, ACTIONS.OUCH);

        if(state == "SLEEP")
            CURRENT_STATE = SLEEP;
        if (state == "SENTINELLA")
            CURRENT_STATE = WAIT;
        if(state == "WAIT")
            CURRENT_STATE = WAIT;
        PREV_STATE = CURRENT_STATE;
    }

    //Esegue l'automa definito nel costruttore di SLigAI
    //Viene chiamato nel metodo Update della classe Game1
    //[pp.48,49]
    public void ROUTINE()
    {
        if (CURRENT_STATE.name != "DEAD")
        {
            ColorMask

            //La transizione da questi stati può avvenire in qualsiasi       ↙
```

```
momento
            if (CURRENT_STATE == WAIT || CURRENT_STATE == SLEEP ||
CURRENT_STATE == WAITCONTROL || CURRENT_STATE == JUSTPOSSESSED)
            HandleStateTransition

            else//Tutti gli altri stati esigono che tutte le animazione
contenute siano state eseguite
            {
                //L'arco è stato percorso, esegue lo stato corrente
                if (FINISHEDTransition)
                {
                    if (FINISHEDExecution)
                        precond = CURRENT_STATE.TryChangeState();
                    if (precond != null)
                        FINISHEDExecution = true;
                    else
                        FINISHEDExecution = CURRENT_STATE.Execute();
                }
                //Tutte le animazione sono state seguite
                if (FINISHEDExecution)
                {
                    CURRENT_STATE.AnimsTODO[CURRENT_STATE.CURRENT_ANIM].Reset
();
                    PREV_STATE = CURRENT_STATE;
                    if (FINISHEDTransition & precond == null)
                        precond = CURRENT_STATE.TryChangeState();//Controllo
precondizioni

                    if (precond != null)//Almeno una precondizione è stata
verificata
                    {
                        if (precond.ACTION != null)//Se una precondizione ha
associata un'azione
                            FINISHEDTransition = precond.ACTION.DO();//L
'azione viene eseguita
                        if (FINISHEDTransition)
                        {
                            CURRENT_STATE = precond.ARC.TO;//L'automa
transita nello stato successivo
                            CURRENT_STATE.ENDSTATE = false;
                            precond = null;
                        }
                    }
                }
            }
        }
        else
        {
            //Gestione morte personaggio
            if (game.abeCONTROL.sligToControl == Slig)
            {
                game.abeCONTROL.sligToControl = null;
                game.abeCONTROL.CURRENT_STATE = game.abeCONTROL.WAIT;
                game.CurrentLevel.Sligs.Remove(Slig);
                game.CurrentLevel = game.Abe.CurrentLevel;
                game.ResetVideos(Slig, game.Abe.CurrentLevel);
            }
            Slig.CurrentLevel.Sligs.Remove(Slig);
            game.CurrentLevel.Sligs.Remove(Slig);
            ////////////////////////////////////
        }
    }
}
```

```
//Definisce gli stati dell'automa
//Ogni stato contiene una o più animazioni da eseguire
//[p.45]
public class STATE
{
    public bool ENDSTATE;
    public List<ARC> ARCS;
    public List<Animation> AnimsTODO;

    public int CURRENT_ANIM = 0;
    public string name;
    public SligAI AI;
    public STATE(string name, SligAI AI)
    {
        ENDSTATE = false;
        this.AI = AI;
        this.name = name;
        AnimsTODO = new List<Animation>();
        ARCS = new List<ARC>();

        if (name == "WAIT")
        {
            AnimsTODO.Add(AI.Slig.WAIT);
        }
        if (name == "JUSTPOSSESSED")
        {
            AnimsTODO.Add(AI.Slig.JUSTPOSSESSED);
        }
        if (name == "WAITCONTROL")
        {
            AnimsTODO.Add(AI.Slig.WAIT);
        }
        if (name == "SHOOTCONTROL")
        {
            AnimsTODO.Add(AI.Slig.SHOOT);
            AnimsTODO.Add(AI.Slig.SHOOT);
        }
        if (name == "SENTINELLA")
        {
            AnimsTODO.Add(AI.Slig.PASS1);
            AnimsTODO.Add(AI.Slig.PASS2);
            AnimsTODO.Add(AI.Slig.PASS3);
        }
        if (name == "GOBACKHOME")
        {
            AnimsTODO.Add(AI.Slig.PASS1);
            AnimsTODO.Add(AI.Slig.PASS2);
            AnimsTODO.Add(AI.Slig.PASS3);
        }
        if (name == "GOBACKHOME2")
        {
            AnimsTODO.Add(AI.Slig.PASS1);
            AnimsTODO.Add(AI.Slig.PASS2);
            AnimsTODO.Add(AI.Slig.PASS3);
        }
        if (name == "SENTINELLACONTROL")
        {
            AnimsTODO.Add(AI.Slig.PASS1);
            AnimsTODO.Add(AI.Slig.PASS2);
        }
        if (name == "VEDETTA")
        {
            AnimsTODO.Add(AI.Slig.WAIT);
            AnimsTODO.Add(AI.Slig.TURN);
            AnimsTODO.Add(AI.Slig.WAIT);
            AnimsTODO.Add(AI.Slig.TURN);
            AnimsTODO.Add(AI.Slig.WAIT);
            AnimsTODO.Add(AI.Slig.TURN);

        }
```

```
if (name == "VEDETTA2")
{
    AnimsTODO.Add(AI.Slig.WAIT);
    AnimsTODO.Add(AI.Slig.TURN);
    AnimsTODO.Add(AI.Slig.WAIT);
    AnimsTODO.Add(AI.Slig.TURN);
    AnimsTODO.Add(AI.Slig.WAIT);
    AnimsTODO.Add(AI.Slig.TURN);

}
if (name == "SHOOT")
{
    AnimsTODO.Add(AI.Slig.SHOOT);
    AnimsTODO.Add(AI.Slig.SHOOT);
}
if (name == "SHOOTED")
{
    AnimsTODO.Add(AI.Slig.POSSESSED);
}
if (name == "RUNNER")
{
    AnimsTODO.Add(AI.Slig.RUN1);
    AnimsTODO.Add(AI.Slig.RUN2);
    AnimsTODO.Add(AI.Slig.RUN3);
    AnimsTODO.Add(AI.Slig.RUN4);
}
if (name == "RUNNERCONTROL")
{
    AnimsTODO.Add(AI.Slig.RUN1);
    AnimsTODO.Add(AI.Slig.RUN2);
    AnimsTODO.Add(AI.Slig.RUN3);
    AnimsTODO.Add(AI.Slig.RUN4);
}
if (name == "RUNSTOP")
{
    AnimsTODO.Add(AI.Slig.RUNsTOP1);
    AnimsTODO.Add(AI.Slig.RUNsTOP2);
}
if (name == "RUNSTOPCONTROL")
{
    AnimsTODO.Add(AI.Slig.RUNsTOP1);
    AnimsTODO.Add(AI.Slig.RUNsTOP2);
}
if (name == "SLEEP")
{
    AnimsTODO.Add(AI.Slig.SLEEP);
}
if (name == "POSSESSED")
{
    AnimsTODO.Add(AI.Slig.POSSESSED);
}
if (name == "SUICIDE")
{
    AnimsTODO.Add(AI.Slig.SUICIDE);
    AnimsTODO.Add(AI.Slig.SUICIDE);
    AnimsTODO.Add(AI.Slig.SUICIDE);
    AnimsTODO.Add(AI.Slig.SUICIDE);
}
if (name == "ONEPASS")
{
    AnimsTODO.Add(AI.Slig.ONEPASS);
    AnimsTODO.Add(AI.Slig.WAIT);
}
if (name == "CRAZY1")
{
    AnimsTODO.Add(AI.Slig.RUN1);
    AnimsTODO.Add(AI.Slig.RUN2);
    AnimsTODO.Add(AI.Slig.RUN3);
    AnimsTODO.Add(AI.Slig.RUN4);
}
```

```
            if (name == "CRAZY2")
            {
                AnimsTODO.Add(AI.Slig.HEELP);
                AnimsTODO.Add(AI.Slig.RUNsTOP2);
            }
            if (name == "DEAD")
            {

            }
        }
        STATEDEFAULT

        //Riporta uno Slig allo stato iniziale
        public void RESET()...

        //Collega due stati e restituisce l'arco associato
        public ARC LinkTO(STATE to)
        {
            ARC arc = new ARC(this.name + "to" + to.name, this, to);

            foreach (ARC arcNew in ARCS)
            {
                if (arcNew.NAME == arc.NAME)
                    return arcNew;
            }
            this.ARCS.Add(arc);
            return arc;
        }

        //Esegue tutte le animazioni contenute nello stato corrente
        public bool Execute()
        {
            if (!AnimsTODO[CURRENT_ANIM].LastFrameDone)
            {
                AI.Slig.AnimateSLIG(AnimsTODO[CURRENT_ANIM]);
                return false;
            }
            else
            {
                if (CURRENT_ANIM == AnimsTODO.Count - 1)
                {
                    ENDSTATE = true;
                    CURRENT_ANIM = 0;
                }
                else
                {
                    CURRENT_ANIM++;
                    AnimsTODO[CURRENT_ANIM].Reset();
                }
                return true;
            }
        }

        //Controlla le precondizioni di tutti gli archi uscenti
        public PRECONDITION TryChangeState()
        {
            foreach (ARC arc in AI.CURRENT_STATE.ARCS)
            {
                foreach (PRECONDITION precondition in arc.PRECONDITIONS)
                {
                    if (precondition.Verify())
                    {
                        CURRENT_ANIM = 0;
                        return precondition;
                    }
                }
            }
            return null;
        }
    }
```

```
//Definisce gli archi dell'automa
//[p.47]
public class ARC
{
    string name;
    STATE from;
    STATE to;

    List<PRECONDITION> preconditions;


    public ARC(string name, STATE from, STATE to)
    {
        preconditions = new List<PRECONDITION>();
        this.name = name;
        this.from = from;
        this.to = to;
    }

    //Pone una precondizione sull'arco
    public void addPRECONDITION(PRECONDITION precondition)
    {
        PRECONDITION p = new PRECONDITION(from.AI, precondition.NAME);
        p.prec1 = precondition.prec1;
        p.prec2 = precondition.prec2;
        preconditions.Add(p);
        p.ARC = this;
    }

    //Pone una precondizione e un'azione sull'arco
    //Se la precondizione è verificata, viene eseguita l'azione
    public void addPRECONDITION(PRECONDITION precondition, ACTION action)
    {
        PRECONDITION p = new PRECONDITION(from.AI, precondition.NAME);
        p.prec1 = precondition.prec1;
        p.prec2 = precondition.prec2;
        preconditions.Add(p);
        p.addACTION(action);
        p.ARC = this;
    }

    Attribute

}
```

```
//Definisce le precondizioni che devono essere aggiunte ad un arco per
transitare da uno stato all'altro.
//[p.46]
public class PRECONDITION
{
    string name;
    SligAI AI;
    ACTION action;
    ARC arc;
    public PRECONDITION prec1;
    public PRECONDITION prec2;
    public PRECONDITION(SligAI AI, string name)
    {
        this.name = name;
        this.AI = AI;
    }

    public void addACTION(ACTION action)
    {
        this.action = action;
    }

    //Controlla se la precondizione è soddisfatta
    public bool Verify()
    {
        if (name == "WALK")
        {
            if (((AI.game.rightDownSlig & AI.Slig.Direction == 1) || (AI.game
.leftDownSlig & AI.Slig.Direction == -1)) & !AI.game.shiftDown)
                return true;
        }
        if (name == "ONEPASS")
        {
            if (((AI.game.OnerightDownSlig & AI.Slig.Direction == 1) || (AI.
game.OneleftDownSlig & AI.Slig.Direction == -1)) & !AI.game.shiftDown)
            {
                AI.game.OnerightDownSlig = false;
                AI.game.OneleftDownSlig = false;
                return true;
            }
        }
        if (name == "WALKSTOP")
        {
            if (((!AI.game.rightDownSlig & AI.Slig.Direction == 1) || (!AI.
game.leftDownSlig & AI.Slig.Direction == -1)))
                return true;
        }
        if (name == "TURN")
        {
            if ((AI.Slig.Direction == -1 & AI.game.rightDown) || (AI.Slig.
Direction == 1 & AI.game.leftDown))
                return true;
        }
        if (name == "RUN")
        {
            if (((AI.game.rightDown & AI.Slig.Direction == 1) || (AI.game.
leftDown & AI.Slig.Direction == -1)) & AI.game.shiftDown)
                return true;
        }
        if (name == "ZERO")
        {
            if (AI.game.ZeroDown)
            {
                if( AI.CURRENT_STATE.name == "SUICIDE" & AI.CURRENT_STATE.
ENDSTATE)
                    AI.Slig.DEAD = true;
                return true;
            }
        }
        if (name == "ZEROUP")
```

```
        {
            if (!AI.game.ZeroDown)
                return true;
        }
        if (name == "ZETA")
        {
            if (AI.game.zetaDown)
                return true;
        }
        if (name == "ZETAUP")
        {
            if (!AI.game.zetaDown)
                return true;
        }
        if (name == "ENDSTATE")
        {
            if (AI.CURRENT_STATE.ENDSTATE)
            {
                return true;
            }
        }
        if (name == "ENDSCREEN")
        {
            if ((AI.Slig.boundingBox.Right >= 500 & AI.Slig.Direction == 1) |
| (AI.Slig.boundingBox.X <= 50 & AI.Slig.Direction == -1))//(AI.Slig.
boundingBox.X <= 50 & AI.Slig.Direction == -1) ||
                return true;
        }
        if (name == "BLOCK")
        {
            List<string> ColTypes = AI.Slig.CurrentLevel.collision(AI.Slig);
            if (ColTypes.Contains(COLLISIONS.BLOCCO))
                return true;
        }
        if (name == "GODOWN")
        {
            List<string> ColTypes = AI.Slig.CurrentLevel.collision(AI.Slig);
            if (ColTypes.Contains(COLLISIONS.GODOWN))
            {
                AI.Slig.DEAD = true;
                return true;
            }
        }
        if (name == "GODOWND")
        {
            List<string> ColTypes = AI.Slig.CurrentLevel.collision(AI.Slig);
            if (ColTypes.Contains(COLLISIONS.BOTOLA))// || Godownd
            {
                AI.Slig.DEAD = true;
                return true;
            }
        }
        if (name == "MATTATOIO")
        {
            List<string> ColTypes = AI.Slig.CurrentLevel.collision(AI.Slig);
            if (ColTypes.Contains(COLLISIONS.MATTATOIO))
            {
                AI.Slig.DEAD = true;
                return true;
            }
        }
        if (name == "ABENOISE")
        {
            if (AI.game.Abe.AbeNoise & AI.Slig.CurrentLevel == AI.game.
CurrentLevel)
            {
                return true;
            }
        }
        if (name == "ABEAHEAD")
```

```
        {
            if (AI.Slig.CurrentLevel.NAME == AI.game.Abe.CurrentLevel.NAME)
            {
                if (((AI.game.Abe.boundingBox.X >= AI.Slig.boundingBox.X & AI
.Slig.Direction == 1) || (AI.game.Abe.boundingBox.X < AI.Slig.boundingBox.X &
 AI.Slig.Direction == -1)) & AI.Slig.CurrentLevel == AI.game.Abe.
CurrentLevel)
                {
                    return true;
                }
            }
            else
            {
                if ((AI.Slig.CurrentLevel.NextRIGHT != null && AI.Slig.
CurrentLevel.NextRIGHT.NAME == AI.game.Abe.CurrentLevel.NAME && AI.Slig.
Direction == 1)
                    ||(AI.Slig.CurrentLevel.NextLEFT != null && AI.Slig.
CurrentLevel.NextLEFT.NAME == AI.game.Abe.CurrentLevel.NAME && AI.Slig.
Direction == -1))
                    return true;
            }
        }
        if (name == "ABEBEHIND")
        {
            if (AI.Slig.CurrentLevel.NAME == AI.game.Abe.CurrentLevel.NAME)
            {
                if (((AI.game.Abe.boundingBox.X > AI.Slig.boundingBox.X & AI.
Slig.Direction == -1) || (AI.game.Abe.boundingBox.X < AI.Slig.boundingBox.X &
 AI.Slig.Direction == 1)) & AI.Slig.CurrentLevel == AI.game.Abe.CurrentLevel)
                    return true;
            }
            else
            {
                if ((AI.Slig.CurrentLevel.NextRIGHT != null && AI.Slig.
CurrentLevel.NextRIGHT.NAME == AI.game.Abe.CurrentLevel.NAME && AI.Slig.
Direction == -1)
                    || (AI.Slig.CurrentLevel.NextLEFT != null && AI.Slig.
CurrentLevel.NextLEFT.NAME == AI.game.Abe.CurrentLevel.NAME && AI.Slig.
Direction == 1))
                    return true;
            }
        }
        if (name == "LOOKATMUDOKON")
        {
            if (AI.Slig.SligEye.Intersects(AI.game.Abe.boundingBox) & AI.Slig
.CurrentLevel.NAME == AI.game.Abe.CurrentLevel.NAME)
            {
                AI.ABE_DETECTED = true;

                if (!AI.game.Abe.SLIGS.Contains(AI.Slig))
                    AI.game.Abe.SLIGS.Add(AI.Slig);
                return true;
            }
        }
        if (name == "ABEDETECTED")
        {
            if (AI.ABE_DETECTED)
                return true;
        }
        if (name == "NOTLOOKATABE")
        {
            if (!AI.Slig.SligEye.Intersects(AI.game.Abe.boundingBox) & AI.
Slig.CurrentLevel.NAME == AI.game.Abe.CurrentLevel.NAME)
            {
                return true;
            }
        }
        if (name == "ABEKILLED")
        {
            bool OK = false;
```

```
            if (AI.CURRENT_STATE.name == "SHOOT" & AI.CURRENT_STATE.ENDSTATE)
            {
                if (AI.Slig.SligEye.Intersects(AI.game.Abe.boundingBox))
                {
                    OK = true;
                    AI.game.Abe.DEAD = true;
                }
                foreach (Mudokon MUD in AI.Slig.CurrentLevel.Mudokons)
                {
                    if (AI.Slig.SligEye.Intersects(MUD.boundingBox))
                    {
                        MUD.AI.CURRENT_STATE = MUD.AI.SHOOTED;
                        MUD.DEAD = true;
                    }

                }
                if(OK)
                    return true;
            }
        }
        if (name == "KILLER")
        {
            if (AI.CURRENT_STATE == AI.SHOOTCONTROL & AI.CURRENT_STATE.
ENDSTATE)
            {
                foreach (Mudokon MUD in AI.Slig.CurrentLevel.Mudokons)
                {
                    if (AI.Slig.SligEye.Intersects(MUD.boundingBox))
                    {
                        MUD.AI.CURRENT_STATE = MUD.AI.SHOOTED;
                        MUD.DEAD = true;
                    }

                }

                foreach (Slig slig in AI.Slig.CurrentLevel.Sligs)
                {
                    if (slig != AI.Slig)
                    {
                        if (AI.Slig.SligEar.Intersects(slig.boundingBox))
                        {
                            slig.AI.CURRENT_STATE = slig.AI.SHOOTED;
                            slig.DEAD = true;
                        }
                    }
                }
            }
        }
        if (name == "ABESAMELVL")
        {
            if (AI.Slig.CurrentLevel.NAME == AI.game.Abe.CurrentLevel.NAME)
                return true;
        }
        if (name == "ABENOTSAMELVL")
        {
            if (AI.Slig.CurrentLevel.NAME != AI.game.Abe.CurrentLevel.NAME &&
AI.game.abeCONTROL.CURRENT_STATE != AI.game.abeCONTROL.NOTCONTROL)
            {
                AI.ABE_DETECTED = false;
                if ((AI.CURRENT_STATE == AI.SENTINELLA || AI.CURRENT_STATE ==
AI.VEDETTA) & AI.Slig.levelDefault.NAME == AI.Slig.CurrentLevel.NAME)
                {
                    if (AI.game.Abe.SLIGS.Contains(AI.Slig))
                        AI.game.Abe.SLIGS.Remove(AI.Slig);
                }
                return true;
            }
        }
        if (name == "LVLCHANGED")
        {
```

```
            if (AI.Slig.levelDefault.NAME != AI.Slig.CurrentLevel.NAME)
            {
                //AI.Slig.levelDefault = AI.Slig.CurrentLevel;
                if(!AI.ABE_DETECTED)
                    AI.game.Abe.SLIGS.Remove(AI.Slig);
                return true;
            }
        }
        if (name == "NOTATHOME")
        {
            if (AI.Slig.levelDefault.NAME != AI.Slig.CurrentLevel.NAME)
                return true;
        }
        if (name == "ATHOME")
        {
            if (AI.Slig.levelDefault.NAME == AI.Slig.CurrentLevel.NAME)
                return true;
        }
        if (name == "AND")
        {
            return this.prec1.Verify() & this.prec2.Verify();
        }
        if (name == "ABECONTROL")
        {
            if (AI.game.abeCONTROL.CURRENT_STATE.name == "NOTCONTROL")
            {
                if (AI.game.abeCONTROL.sligToControl == AI.Slig)
                    return true;
                else
                    AI.CURRENT_STATE = AI.SENTINELLA;
            }
        }
        if (name == "ABEOMM")
        {
            if (AI.game.abeCONTROL.CURRENT_STATE.name == "OMM")
                return true;
        }
        if (name == "ABENOTOMM")
        {
            if (AI.game.abeCONTROL.CURRENT_STATE.name != "OMM" && AI.game.
abeCONTROL.CURRENT_STATE != AI.game.abeCONTROL.NOTCONTROL)
                return true;
        }
        return false;
    }

    Attribute
}

//Contiene i riferimenti a tutte le istanze della classe PRECONDITION
public class PRECONDITIONS...
```

```
//Definisce le azioni che possono essere associate ad una precondizione
//Ogni azione può contenere una o più precondizioni.
//[p.45]
public class ACTION
{
    SligAI AI;
    string name;
    public List<Animation> AnimsTODO;
    int CURRENT_ANIM = 0;


    public ACTION(SligAI AI, string name)
    {
        AnimsTODO = new List<Animation>();
        this.AI = AI;
        this.name = name;

        if (name == "WAIT")
        {
            AnimsTODO.Add(AI.Slig.WAIT);
        }
        if (name == "SUICIDE")
        {
            AnimsTODO.Add(AI.Slig.POSSESSED);
        }
        if (name == "WHAAT")
        {
            AnimsTODO.Add(AI.Slig.WHAAT);
        }
        if (name == "OUCH")
        {
            AnimsTODO.Add(AI.Slig.OUCH);
        }
        if (name == "FREEZE")
        {
            AnimsTODO.Add(AI.Slig.FREEZE);
        }
        if (name == "STOP")
        {
            AnimsTODO.Add(AI.Slig.PASSsTOP);
        }
        if (name == "PASS3")
        {
            AnimsTODO.Add(AI.Slig.PASS3);
        }
        if (name == "START")
        {
            AnimsTODO.Add(AI.Slig.START);
        }
        if (name == "TURN")
        {
            AnimsTODO.Add(AI.Slig.TURN);
        }

        if (name == "DOWNPLANE")
        {
            AnimsTODO.Add(AI.Slig.DOWNPLANE);
        }

        if (name == "SPLAT")
        {
            AnimsTODO.Add(AI.Slig.SPLAT);
        }
        if (name == "SHOOTSTART")
        {
            AnimsTODO.Add(AI.Slig.SHOOTsTART);
        }
        if (name == "SHOOTSTOP")
        {
            AnimsTODO.Add(AI.Slig.SHOOTsTOP);
```

```
        }
        if (name == "RUNSTART")
        {
            AnimsTODO.Add(AI.Slig.RUNsTART);
        }
        if (name == "RUNSTOP")
        {
            AnimsTODO.Add(AI.Slig.RUNsTOP1);
            AnimsTODO.Add(AI.Slig.RUNsTOP2);
        }
        if (name == "WAKE")
        {
            AnimsTODO.Add(AI.Slig.WAKE);
        }
        if (name == "KILLABE")
        {
            AnimsTODO.Add(AI.Slig.SHOOT);
            AnimsTODO.Add(AI.Slig.SHOOTsTOP);
            AnimsTODO.Add(AI.Slig.START);
        }
    }

    Repeat

    //Metodo chiamato per eseguire tutte le animazioni contenute in un'azione
    public bool DO()
    {
        if (!AnimsTODO[CURRENT_ANIM].LastFrameDone)
        {
            AI.Slig.AnimateSLIG(AnimsTODO[CURRENT_ANIM]);
            return false;
        }
        else
        {
            if (CURRENT_ANIM == AnimsTODO.Count - 1)
            {
                CURRENT_ANIM = 0;
                AI.CURRENT_STATE.CURRENT_ANIM = 0;
                return true;
            }
            else
            {
                CURRENT_ANIM++;
                AnimsTODO[CURRENT_ANIM].Reset();
            }
            return false;

        }
    }

    Attribute
}

//Contiene i riferimenti a tutte le istanze della classe ACTION
public class ACTIONS...
```

```
using ...

namespace TheGame
{

    //Classe per la gestione di un livello
    //[p.54,55]
    public class Level
    {
        Game1 game;
        public string NAME;
        public int difficult = 0;
        public string Enviorement = "";

        public Texture2D BackGround;
        public Texture2D BackGround_DEBUG;

        public Level NextRIGHT;
        public Level NextLEFT;
        public Level NextUP;
        public Level NextDOWN;

        public bool Sentinella = false;
        public List<Mudokon> Mudokons;
        public List<Slig> Sligs;
        public List<ObjectA> ObjectsA;
        public List<Video> VIDEOS;
        public List<Rectangle> VIDEOSREC;
        public Texture2D OBJECTS;

        public bool FLOOR2NDLeft = false;
        public bool FLOOR2NDRight = false;
        public bool BLOCCOUPRight = false;
        public bool BLOCCOUPLeft = false;
        public bool BLOCCODOWNRight = false;
        public bool BLOCCODOWNLeft = false;

        //Livelli strutturalmente compatibili
        public List<Level> compatibleLevels = new List<Level>();

        //Struttura architettonica[p.55]
        public List<List<Collision>> COLLISIONGROUPS = new List<List<Collision>> ↙
    ();

        public Level(Level levelCopy)...

        public Level(string Env, Game1 game)
        {
            this.Enviorement = Env;
            this.game = game;
            Mudokons = new List<Mudokon>();
            Sligs = new List<Slig>();

            ObjectsA = new List<ObjectA>();
            VIDEOS = new List<Video>();
            VIDEOSREC = new List<Rectangle>();

            BackGround = game.Content.Load<Texture2D>("LVL\\BackGround\\" +       ↙
    Enviorement);

            try{
                //BackGround_DEBUG = game.Content.Load<Texture2D>("LVL\\Debug\\" ↙
    + Enviorement + "_DEBUG");
                OBJECTS = game.Content.Load<Texture2D>("LVL\\OBJECTS\\" +        ↙
    Enviorement);
            }catch(Exception e){}
        }
```

```
    public List<string> collision(Rectangle recToTest)...

    //Dato un qualsiasi personaggio (Abe,Mudokon o Slig) restituisce una lista contenente
    //tutti gli elementi che collidono con la sua BoundingBox
    public List<string> collision(Character chracter)
    {
        List<string> collisions = new List<string>();
        chracter.CURRENTOBJECTA = null;

        for (int i = 0; i < COLLISIONGROUPS.Count; i++)
        {
            switch (COLLISIONGROUPS[i][0].TYPE)
            {
                case COLLISIONS.BLOCCO:
                    foreach (Collision coll in COLLISIONGROUPS[i])
                    {
                        if (chracter.boundingBox.Intersects(coll.REC) & IsNearFrontBlocco(chracter.boundingBox,coll.REC, chracter.Direction))
                        {
                            collisions.Add(COLLISIONS.BLOCCO);
                        }
                    }
                    break;

                case COLLISIONS.LEVA:
                    foreach (Collision coll in COLLISIONGROUPS[i])
                    {
                        if (chracter.boundingBox.Intersects(coll.REC) & IsNearObj(chracter.boundingBox, coll.REC, chracter.Direction))
                        {
                            collisions.Add(coll.TYPE);
                            chracter.CURRENTOBJECTA = coll.OBJA;
                        }
                    }
                    break;

                case COLLISIONS.BOMBINNESCO:
                    foreach (Collision coll in COLLISIONGROUPS[i])
                    {
                        if (chracter.boundingBox.Intersects(coll.REC) && IsNearObj(chracter.boundingBox, coll.REC, chracter.Direction))
                        {
                            collisions.Add(COLLISIONS.BOMBINNESCO);
                            chracter.CURRENTOBJECTA = coll.OBJA;

                        }
                        if (chracter.boundingBox.Intersects(coll.REC) && IsOnObj(chracter.boundingBox, coll.REC))
                        {
                            chracter.CURRENTOBJECTA = coll.OBJA;
                            if(chracter.CURRENTOBJECTA != null && chracter.CURRENTOBJECTA.objControl.ACTIVE)
                                collisions.Add(COLLISIONS.BOMB);
                        }
                    }
                    break;

                case COLLISIONS.PORTAL:
                    foreach (Collision coll in COLLISIONGROUPS[i])
                    {
                        if (chracter.boundingBox.Intersects(coll.REC))
                        {
                            chracter.CURRENTOBJECTA = coll.OBJA;
                            collisions.Add(coll.TYPE);
                        }
                    }
                    break;

                default:
```

```
                    foreach (Collision coll in COLLISIONGROUPS[i])
                    {
                        if (chracter.boundingBox.Intersects(coll.REC))
                            collisions.Add(coll.TYPE);
                    }
                    break;
                }
            }
            return collisions;
        }

        public List<Collision> GetCollision(Character chracter)...

        bool IsOnObj(Rectangle Chara, Rectangle Obj)...

        bool IsNearObj(Rectangle Chara, Rectangle Obj, int Direction)...

        bool IsNearFrontBlocco(Rectangle Chara, Rectangle Blocco, int Direction) ↙
...

        public void AddObject(Texture2D obj)...

        public void AddObjectA(ObjectA objA)...

        public void RemoveObjectA(ObjectA objA)...

        public void AddVideo(Video vid, Rectangle vidRec)...

        public void AddMudokon(Mudokon MUD)...

        public void RemoveMudokon(Mudokon MUD)...

        public void AddSlig(Slig SLIG)...

        public void RemoveSlig(Slig SLIG)...

        public void AddCOLLISION(Collision CollToAdd)...

        public void RemoveCOLLISION(Collision CollToRemove)...

        public void RESET()
        {
            for (int i = 0; i < ObjectsA.Count; i++)
                ObjectsA[i].objControl.RESET();

            for (int i = 0; i < Sligs.Count; i++)
            {
                Sligs[i].AI.CURRENT_STATE.RESET();
                if (this != Sligs[i].levelDefault)
                    this.RemoveSlig(Sligs[i]);
            }
        }

        //Metodo che riadatta la difficoltà del livello
        //Chiamato nella classe Report
        public void Modify(string Code)
        {
            if (Code == "DIFF_UP")
            {
                SetUPSlig();
                SetUPBomb();
                SetUPBombInnesco();
            }
            if (Code == "DIFF_DOWN")
            {
                SetDOWNSLIG();
                SetDOWNBomb();
            }
            NextRIGHT = null;
            NextLEFT = null;
```

```
            NextUP = null;
            NextDOWN = null;
        }

        //Incrementa la difficoltà del livello agendo sui nemici
        public void SetUPSlig()
        {
            bool bombs = false;
            foreach (ObjectA obj in ObjectsA)
            {
                if (obj.NAME == "BOMBS" & obj.COLLISION.REC.Y == 385)
                    bombs = true;
            }


            //Se il livello continene nemici
            //Ai nemici già presenti nel livello viene aumentato il fattore IA
            //Viene aggiunto un nemico(Slig) con IA minima
            //Viene aggiunta una Sentinella(classe ObjectA)
            if (Sligs.Count > 0)
            {
                bool AllAiMax = false;
                if ((FLOOR2NDLeft || FLOOR2NDRight) & !Sentinella)
                {
                    Sentinella = true;
                    ObjectA OMM = new ObjectA(game, "OMM", new Rectangle(500, 50,↙
 0, 0), new Collision(COLLISIONS.SHOCK, new Rectangle()), this);
                    this.AddObjectA(OMM);
                }

                foreach (Slig slig in Sligs)
                {
                    if (slig.AI_LEVEL == 4)
                        AllAiMax = true;
                    else
                        AllAiMax = false;
                }
                if (AllAiMax & Sligs.Count < 2)
                {
                    if (!bombs)
                    {
                        Rectangle Spawn = new Rectangle(49 + 43 * 3, 395, 44, 50)↙
;
                        Slig slig;
                        if (Sligs[0].state == "SLEEP")
                            slig = new Slig(game, "SENTINELLA", 2, Spawn.        ↙
Location);
                        else
                            slig = new Slig(game, "SLEEP", 2, Spawn.Location);

                        List<string> ColTypes = this.collision(slig);
                        int blocchi = 0;
                        for (int i = Spawn.Location.X; i <= 522; i += 43)
                        {
                            ColTypes = this.collision(slig);
                            if (ColTypes.Count == 0)
                            {
                                slig.levelDefault = this;
                                this.AddSlig(slig);
                                i = 1000;
                            }
                            if (ColTypes.Contains(COLLISIONS.BLOCCO))
                                blocchi++;
                            if (!ColTypes.Contains(COLLISIONS.BLOCCO) & blocchi >↙
 0)
                                blocchi--;

                            if (blocchi > 3)
                                i = 1000;
                        }
```

```
                }
            }
            foreach (Slig slig in Sligs)
            {
                if (slig.AI_LEVEL < 4)
                    slig.AI_LEVEL++;
            }
            foreach (Slig slig in Sligs)
            {
                if (slig.AI_LEVEL < 4)
                    slig.AI_LEVEL++;
            }
        }

        //Se il livello non continene nemici
        //Viene aggiunto un nemico(Slig) con IA media
        else
        {
            if (!bombs)
            {
                Rectangle Spawn = new Rectangle(49, 395, 44, 50);
                Slig slig = new Slig(game, "SENTINELLA", 2, Spawn.Location);
                List<string> ColTypes = this.collision(slig);
                int blocchi = 0;

                for (int i = Spawn.Location.X; i <= 522; i += 43)
                {
                    ColTypes = this.collision(slig);
                    if (ColTypes.Count == 0)
                    {
                        slig.levelDefault = this;
                        this.AddSlig(slig);
                        i = 1000;
                    }
                    if (ColTypes.Contains(COLLISIONS.BLOCCO))
                        blocchi++;
                    if (!ColTypes.Contains(COLLISIONS.BLOCCO) & blocchi > 0)
                        blocchi--;

                    if (blocchi > 3)
                        i = 1000;
                }
            }
        }
    }

    //Decrementa la difficoltà del livello agendo sui nemici
    public void SetDOWNSLIG()
    {
        int totalAI = 0;
        if (Sligs.Count > 0)
        {
            Sentinella = false;
            for (int i = 0; i < ObjectsA.Count; i++)
                if (ObjectsA[i].NAME == "OMM")
                    RemoveObjectA(ObjectsA[i]);
            foreach (Slig slig in Sligs)
                totalAI += slig.AI_LEVEL;

            if (totalAI == Sligs.Count)//tutti gli slig hanno AI di basso livello
            {
                Sligs.RemoveAt(0);
            }
            else
            {
                foreach (Slig slig in Sligs)
                    if (slig.AI_LEVEL > 1)
                        slig.AI_LEVEL--;
            }
```

```
        }
    }

    //Incrementa la difficoltà del livello agendo sulle trappole(ObjectA: BOMBINNESCO)
    public void SetUPBombInnesco()
    {
        int bombInnNum = 0;
        int AllRange = 0;
        foreach (ObjectA bombInn in ObjectsA)
        {
            if (bombInn.NAME == "BOMBINNESCO")
            {
                if (bombInn.RangeInnesco >= 3)
                {
                    bombInn.RangeInnesco -= 2;
                }
            }
        }
        foreach (ObjectA bombInn in ObjectsA)
        {
            if (bombInn.NAME == "BOMBINNESCO")
            {
                bombInnNum++;
                AllRange += bombInn.RangeInnesco;
            }
        }

        if (AllRange == bombInnNum)
        {
            int bombs = 0;
            Rectangle NewBomb = new Rectangle(19, 385, 19, 50);
            List<string> Colls = this.collision(NewBomb);


            for (int i = 19; i <= 572; i += 43)
            {
                NewBomb.X = i;
                Colls = this.collision(NewBomb);
                if (Colls.Contains(COLLISIONS.BOMBINNESCO))
                {
                    bombs++;
                }
                if (!Colls.Contains(COLLISIONS.BOMBINNESCO) & bombs > 0 & bombs < 3)
                {
                    if (Colls.Count == 0 || Colls.Contains(COLLISIONS.HOISTSINISTRO) || Colls.Contains(COLLISIONS.HOISTDESTRO) || Colls.Contains(COLLISIONS.APPIGLIOSINISTRO) || Colls.Contains(COLLISIONS.APPIGLIODESTRO))
                    {
                        NewBomb = new Rectangle(NewBomb.X - 20, NewBomb.Y, 60, 50);
                        Collision bombColl = new Collision(COLLISIONS.BOMBINNESCO, NewBomb);
                        ObjectA Bomb = new ObjectA(game, "BOMBINNESCO", new Rectangle(NewBomb.X - 9, NewBomb.Y - 55, 70, 70), bombColl, this);
                        bombColl.OBJA = Bomb;

                        this.AddObjectA(Bomb);
                        i = 1000;
                    }
                    else
                    {
                        NewBomb.X = i - 43 * (bombs + 1);
                        Colls = this.collision(NewBomb);
                        if (Colls.Count == 0)
                        {
                            NewBomb = new Rectangle(NewBomb.X - 20, NewBomb.Y, 60, 50);
                            Collision bombColl = new Collision(COLLISIONS.
```

```
BOMBINNESCO, NewBomb);
                                ObjectA Bomb = new ObjectA(game, "BOMBINNESCO",
new Rectangle(NewBomb.X - 9, NewBomb.Y - 55, 70, 70), bombColl, this);
                                bombColl.OBJA = Bomb;

                                this.AddObjectA(Bomb);
                                i = 1000;
                            }
                        }
                    }
                }

                bombs = 0;
                NewBomb = new Rectangle(19, 185, 19, 50);
                Colls = this.collision(NewBomb);


                for (int i = 19; i <= 572; i += 43)
                {
                    NewBomb.X = i;
                    Colls = this.collision(NewBomb);
                    if (Colls.Contains(COLLISIONS.BOMBINNESCO))
                    {
                        bombs++;
                    }
                    if (!Colls.Contains(COLLISIONS.BOMBINNESCO) & bombs > 0 &
bombs < 3)
                    {
                        if (Colls.Count == 0 || Colls.Contains(COLLISIONS.
HOISTSINISTRO) || Colls.Contains(COLLISIONS.HOISTDESTRO) || Colls.Contains
(COLLISIONS.APPIGLIOSINISTRO) || Colls.Contains(COLLISIONS.APPIGLIODESTRO))
                        {
                            NewBomb = new Rectangle(NewBomb.X - 20, NewBomb.Y, 60
, 50);
                            Collision bombColl = new Collision(COLLISIONS.
BOMBINNESCO, NewBomb);
                            ObjectA Bomb = new ObjectA(game, "BOMBINNESCO", new
Rectangle(NewBomb.X - 9, NewBomb.Y - 55, 70, 70), bombColl, this);
                            bombColl.OBJA = Bomb;

                            this.AddObjectA(Bomb);
                            i = 1000;
                        }
                        else
                        {
                            NewBomb.X = i - 43 * (bombs + 1);
                            Colls = this.collision(NewBomb);
                            if (Colls.Count == 0)
                            {
                                NewBomb = new Rectangle(NewBomb.X - 20, NewBomb.Y
, 60, 50);
                                Collision bombColl = new Collision(COLLISIONS.
BOMBINNESCO, NewBomb);
                                ObjectA Bomb = new ObjectA(game, "BOMBINNESCO",
new Rectangle(NewBomb.X - 9, NewBomb.Y - 55, 70, 70), bombColl, this);
                                bombColl.OBJA = Bomb;

                                this.AddObjectA(Bomb);
                                i = 1000;
                            }
                        }
                    }
                }
            }
        }

        //Incrementa la difficoltà del livello agendo sulle trappole(ObjectA:
BOMB)
        public void SetUPBomb()
        {
```

```
        int bombs = 0;
        Rectangle NewBomb = new Rectangle(13, 385, 30, 50);
        List<string> Colls = this.collision(NewBomb);

        for (int i = 13; i <= 572; i += 43)
        {
            NewBomb.X = i;
            Colls = this.collision(NewBomb);
            if (Colls.Contains(COLLISIONS.BOMB))
            {
                bombs++;
            }
            if (!Colls.Contains(COLLISIONS.BOMB) & bombs > 0 & bombs < 3)
            {
                if (Colls.Count == 0)
                {
                    Collision bombColl = new Collision(COLLISIONS.BOMB,
NewBomb);
                    ObjectA Bomb = new ObjectA(game, "BOMB", new Rectangle
(NewBomb.X - 6, NewBomb.Y - 40, 50, 50), bombColl, this);
                    bombColl.OBJA = Bomb;

                    this.AddObjectA(Bomb);
                    i = 1000;
                }
                else
                {
                    NewBomb.X = i - 43 * (bombs + 1);
                    Colls = this.collision(NewBomb);
                    if (Colls.Count == 0)
                    {
                        Collision bombColl = new Collision(COLLISIONS.BOMB,
NewBomb);
                        ObjectA Bomb = new ObjectA(game, "BOMB", new
Rectangle(NewBomb.X - 6, NewBomb.Y - 40, 50, 50), bombColl, this);
                        bombColl.OBJA = Bomb;

                        this.AddObjectA(Bomb);
                        i = 1000;
                    }
                }
            }
        }
        bombs = 0;
        NewBomb = new Rectangle(13, 185, 30, 50);
        Colls = this.collision(NewBomb);

        for (int i = 13; i <= 572; i += 43)
        {
            NewBomb.X = i;
            Colls = this.collision(NewBomb);
            if (Colls.Contains(COLLISIONS.BOMB))
            {
                bombs++;
            }
            if (!Colls.Contains(COLLISIONS.BOMB) & bombs > 0 & bombs < 3)
            {
                if (Colls.Count == 0)
                {
                    Collision bombColl = new Collision(COLLISIONS.BOMB,
NewBomb);
                    ObjectA Bomb = new ObjectA(game, "BOMB", new Rectangle
(NewBomb.X - 6, NewBomb.Y - 40, 50, 50), bombColl, this);
                    bombColl.OBJA = Bomb;
                    this.AddObjectA(Bomb);
                    i = 1000;
                }
                else
                {
                    NewBomb.X = i - 43 * (bombs + 1);
```

```
                    Colls = this.collision(NewBomb);
                    if (Colls.Count == 0)
                    {
                        Collision bombColl = new Collision(COLLISIONS.BOMB,
NewBomb);
                        ObjectA Bomb = new ObjectA(game, "BOMB", new
Rectangle(NewBomb.X - 6, NewBomb.Y - 40, 50, 50), bombColl, this);
                        bombColl.OBJA = Bomb;

                        this.AddObjectA(Bomb);
                        i = 1000;
                    }
                }
            }
        }
    }

    //Decrementa la difficoltà del livello agendo sulle trappole(ObjectA:
BOMB)
    public void SetDOWNBomb()
    {
        int bombs = 0;
        Rectangle NewBomb = new Rectangle(13, 385, 30, 50);
        List<string> Colls = this.collision(NewBomb);

        for (int i = 13; i <= 572; i += 43)
        {
            NewBomb.X = i;
            Colls = this.collision(NewBomb);
            if (Colls.Contains(COLLISIONS.BOMB))
            {
                bombs++;
            }
            if (!Colls.Contains(COLLISIONS.BOMB) && bombs >= 2 && bombs <= 3)
            {
                NewBomb.X = i - 43;
                for (int o = 0; o < ObjectsA.Count; o++)
                {
                    if (ObjectsA[o].COLLISION.REC.Intersects(NewBomb))
                    {
                        RemoveObjectA(ObjectsA[o]);
                    }
                }
            }
        }
        bombs = 0;
        NewBomb = new Rectangle(13, 185, 30, 50);
        Colls = this.collision(NewBomb);
        for (int i = 13; i <= 572; i += 43)
        {
            NewBomb.X = i;
            Colls = this.collision(NewBomb);
            if (Colls.Contains(COLLISIONS.BOMB))
            {
                bombs++;
            }
            if (!Colls.Contains(COLLISIONS.BOMB) && bombs >= 2 && bombs <= 3)
            {
                NewBomb.X = i - 43;
                for (int o = 0; o < ObjectsA.Count; o++)
                {
                    if (ObjectsA[o].COLLISION.REC.Intersects(NewBomb))
                    {
                        RemoveObjectA(ObjectsA[o]);
                    }
                }
            }
        }
    }
```

```
    //Assegna il grado di difficoltà al livello in base agli elementi presenti
    public void SetDifficult()
    {
        int diff = 0;

        foreach (Slig slig in this.Sligs)
        {
            if (slig.AI.CURRENT_STATE.name == "SLEEP")
                diff += 2;
            if (slig.AI.CURRENT_STATE.name == "WAIT")
                diff += 3;

            diff += slig.AI_LEVEL;
        }

        foreach (Mudokon mudokon in this.Mudokons)
            diff += 1;

        foreach (ObjectA objA in this.ObjectsA)
        {
            if (objA.NAME == "MATTATOIO")
            {
                if (objA.objControl.ACTIVE)//& loop
                    diff++;
                else
                    diff--;
            }

            if (objA.NAME == "BOTOLA")
                diff--;

            if (objA.NAME == "BOMBINNESCO")
            {
                List<string> Colls = this.collision(objA.COLLISION.REC);

                if (Colls.Contains(COLLISIONS.HOISTSINISTRO) || Colls.Contains(COLLISIONS.HOISTDESTRO) || Colls.Contains(COLLISIONS.APPIGLIOSINISTRO) || Colls.Contains(COLLISIONS.APPIGLIODESTRO))
                {
                    if (objA.RangeInnesco == 9)
                        diff += 1;
                    if (objA.RangeInnesco == 7)
                        diff += 2;
                    if (objA.RangeInnesco == 5)
                        diff += 3;
                    if (objA.RangeInnesco == 5)
                        diff += 4;
                    if (objA.RangeInnesco == 1)
                        diff += 5;
                }
                else
                {
                    diff += 1;
                }
            }

            if (objA.NAME == "BOMB")
                diff++;
            if (objA.NAME == "ELECTRO")
                diff++;
        }
        if (this.Sentinella)
            diff++;

        this.difficult = diff;
    }
}
```

```
    //Definisce un elemento srutturale del livello, secondo il tipo, dimensione e posizione
    //[p.54,55]
    public class Collision
    {
        public string TYPE;
        public Rectangle REC;
        public ObjectA OBJA;
        public Level LINKEDLEVEL;

        public Collision(string Type, Rectangle Rec)
        {
            this.TYPE = Type;
            this.REC = Rec;

            if (TYPE == COLLISIONS.DOOR)
                LINKEDLEVEL = null;
        }
    }
    //Contiene i riferimenti a tutte le tipologie di elemento strutturale
    public class COLLISIONS
    {
        public const string GRAYAREA = "GRAYAREA";
        public const string BLACKAREA = "BLACKAREA";
        public const string APPIGLIODESTRO = "APPIGLIODESTRO";
        public const string APPIGLIOSINISTRO = "APPIGLIOSINISTRO";
        public const string BLOCCO = "BLOCCO";
        public const string HOISTDESTRO = "HOISTDESTRO";
        public const string HOISTSINISTRO = "HOISTSINISTRO";
        public const string GODOWN = "GODOWN";
        public const string LEVA = "LEVA";
        public const string BOMBINNESCO = "BOMBINNESCO";
        public const string BOMB = "BOMB";
        public const string ELECTRO = "ELECTRO";
        public const string MATTATOIO = "MATTATOIO";
        public const string BOTOLA = "BOTOLA";
        public const string GODOWND = "GODOWND";
        public const string SHOCK = "SHOCK";
        public const string PORTAL = "PORTAL";
        public const string DOOR = "DOOR";

        public COLLISIONS(){}
    }

}
```

```
using ...

namespace TheGame
{
    //Per ogni file XML generato dall'editor crea un'istanza della classe Level
    //Tutti i livelli sono contenuti in lvlList
    //[p.54]
    class LevelMaker
    {
        VARS

        public LevelMaker(Game1 game)
        {
            VideoContent = new VideoContentManager(game.Services);
            VideoContent.RootDirectory = "Content";
            //recList = new List<Rectangle>();
            this.game = game;
            di = new DirectoryInfo(@"..\..\..\Content\LVL\XML\");
            fi = di.GetFiles();
            lvlNameList = new List<string>();
            lvlList = new List<Level>();

            diVid = new DirectoryInfo(@"..\..\..\Content\VIDEOS\");
            fiVid = diVid.GetFiles();

            string Nametmp;

            foreach (FileInfo f in fiVid)
            {
                if (f.Extension == ".avi")
                {
                    Nametmp = f.Name.Remove(f.Name.Length - 4, 4);
                    if (Nametmp.Split('_').Count() == 5)
                    {
                        videos.Add(VideoContent.Load<Video>("VIDEOS\\" +          ↙
    Nametmp));
                        videosName.Add(Nametmp.Split('_')[0]);
                        videosRec.Add(new Rectangle(int.Parse(Nametmp.Split('_') ↙
    [1]), int.Parse(Nametmp.Split('_')[2]), int.Parse(Nametmp.Split('_')[3]), int↙
    .Parse(Nametmp.Split('_')[4])));
                    }
                }
            }
            Level level;

            foreach (FileInfo f in fi)
            {
                level = makeLVL(f.Name.Remove(f.Name.Length - 4));
                if(level != null)
                    lvlList.Add(level);
            }

            game.CurrentLevel = lvlList[0];

            DEMO

        }

        //Viene navigata la struttura di ogni file XML e creato il livello       ↙
    corrispondente
        public Level makeLVL(string lvlName)
        {
            doc = new XmlDocument();
            doc.Load(@"..\..\..\Content\LVL\XML\" + lvlName + ".xml");
            root = doc.DocumentElement;
            nodeList = root.ChildNodes;

            Level lvl = null;

            for (int i = 0; i < nodeList.Count; i++)
```

```
{
    if (nodeList[i].Name == "ENVIOREMENT")
    {
        attrList = nodeList[i].Attributes;
        string ENV = attrList["NAME"].Value;
        lvl = new Level(ENV, game);
    }

    if (nodeList[i].Name == "NAME")
    {
        attrList = nodeList[i].Attributes;
        string Name = attrList["NAME"].Value;
        lvl.NAME = Name;
    }
    if (nodeList[i].Name == "FLOOR2NDLEFT")
    {
        attrList = nodeList[i].Attributes;
        string Val = attrList["SET"].Value;
        if (Val == "Y")
            lvl.FLOOR2NDLeft = true;
        else
            lvl.FLOOR2NDLeft = false;
    }
    if (nodeList[i].Name == "FLOOR2NDRIGHT")
    {
        attrList = nodeList[i].Attributes;
        string Val = attrList["SET"].Value;
        if (Val == "Y")
            lvl.FLOOR2NDRight = true;
        else
            lvl.FLOOR2NDRight = false;
    }
    if (nodeList[i].Name == "BLOCCOUPRIGHT")
    {
        attrList = nodeList[i].Attributes;
        string Val = attrList["SET"].Value;
        if (Val == "Y")
            lvl.BLOCCOUPRight = true;
        else
            lvl.BLOCCOUPRight = false;
    }
    if (nodeList[i].Name == "BLOCCOUPLEFT")
    {
        attrList = nodeList[i].Attributes;
        string Val = attrList["SET"].Value;
        if (Val == "Y")
            lvl.BLOCCOUPLeft = true;
        else
            lvl.BLOCCOUPLeft = false;
    }
    if (nodeList[i].Name == "BLOCCODOWNLEFT")
    {
        attrList = nodeList[i].Attributes;
        string Val = attrList["SET"].Value;
        if (Val == "Y")
            lvl.BLOCCODOWNLeft = true;
        else
            lvl.BLOCCODOWNLeft = false;
    }
    if (nodeList[i].Name == "BLOCCODOWNRIGHT")
    {
        attrList = nodeList[i].Attributes;
        string Val = attrList["SET"].Value;
        if (Val == "Y")
            lvl.BLOCCODOWNRight = true;
        else
            lvl.BLOCCODOWNRight = false;
    }
    if (nodeList[i].Name == "GODOWNLIST")
    {
```

```
                Godowns = nodeList[i].ChildNodes;
                for (int k = 0; k < Godowns.Count; k++)
                {
                    if (Godowns[k].Name == "GODOWN")
                    {
                        attrList = Godowns[k].Attributes;
                        Rectangle rec = new Rectangle();
                        rec.X = toInt(attrList["X"].Value);
                        rec.Y = toInt(attrList["Y"].Value);
                        rec.Width = toInt(attrList["WIDTH"].Value);
                        rec.Height = toInt(attrList["HEIGHT"].Value);

                        lvl.AddCOLLISION(new Collision(COLLISIONS.GODOWN,
rec));
                    }
                }
            }
            if (nodeList[i].Name == "HOISTLIST")
            {
                HoistDestri = nodeList[i].ChildNodes;
                for (int k = 0; k < HoistDestri.Count; k++)
                {
                    if (HoistDestri[k].Name == "HOIST")
                    {
                        attrList = HoistDestri[k].Attributes;
                        Rectangle rec = new Rectangle();
                        rec.X = toInt(attrList["X"].Value);
                        rec.Y = toInt(attrList["Y"].Value);
                        rec.Width = toInt(attrList["WIDTH"].Value);
                        rec.Height = toInt(attrList["HEIGHT"].Value);

                        string isAdvise = attrList["ADVISE"].Value;

                        if (isAdvise == "Y")
                        {
                            ObjectA Advise = new ObjectA(game, "HOISTADVISE",
 new Rectangle(rec.X, rec.Y, rec.Width, 480), new Collision("", new Rectangle
()), lvl);
                            lvl.AddObjectA(Advise);
                        }
                        if (IntersectsGodown(new Rectangle(rec.X - 40, rec.Y,
 rec.Width, rec.Height), lvl))
                            lvl.AddCOLLISION(new Collision(COLLISIONS.
HOISTDESTRO, rec));
                        else
                            lvl.AddCOLLISION(new Collision(COLLISIONS.
HOISTSINISTRO, rec));
                    }
                }
            }
            if (nodeList[i].Name == "APPIGLIOLIST")
            {
                AppigliDestri = nodeList[i].ChildNodes;
                for (int k = 0; k < AppigliDestri.Count; k++)
                {
                    if (AppigliDestri[k].Name == "APPIGLIO")
                    {
                        attrList = AppigliDestri[k].Attributes;
                        Rectangle rec = new Rectangle();
                        rec.X = toInt(attrList["X"].Value);
                        rec.Y = toInt(attrList["Y"].Value);
                        rec.Width = toInt(attrList["WIDTH"].Value);
                        rec.Height = toInt(attrList["HEIGHT"].Value);

                        if (IntersectsGodown(new Rectangle(rec.X - 40, rec.Y
- 200, rec.Width, rec.Height), lvl))
                            lvl.AddCOLLISION(new Collision(COLLISIONS.
APPIGLIODESTRO, rec));
                        else
                            lvl.AddCOLLISION(new Collision(COLLISIONS.
```

```
APPIGLIOSINISTRO, rec));
                    }
                }
            }
            if (nodeList[i].Name == "BLOCCOLIST")
            {
                BlocchiDestri = nodeList[i].ChildNodes;
                for (int k = 0; k < BlocchiDestri.Count; k++)
                {
                    if (BlocchiDestri[k].Name == "BLOCCO")
                    {
                        attrList = BlocchiDestri[k].Attributes;
                        Rectangle rec = new Rectangle();
                        rec.X = toInt(attrList["X"].Value);
                        rec.Y = toInt(attrList["Y"].Value);
                        rec.Width = toInt(attrList["WIDTH"].Value);
                        rec.Height = toInt(attrList["HEIGHT"].Value);

                        lvl.AddCOLLISION(new Collision(COLLISIONS.BLOCCO,
rec));
                    }
                }
            }
            if (nodeList[i].Name == "DOORLIST")
            {
                DOORList = nodeList[i].ChildNodes;
                for (int k = 0; k < DOORList.Count; k++)
                {
                    if (DOORList[k].Name == "DOOR")
                    {
                        attrList = DOORList[k].Attributes;
                        Rectangle rec = new Rectangle();
                        rec.X = toInt(attrList["X"].Value);
                        rec.Y = toInt(attrList["Y"].Value);
                        rec.Width = toInt(attrList["WIDTH"].Value);
                        rec.Height = toInt(attrList["HEIGHT"].Value);

                        lvl.AddCOLLISION(new Collision(COLLISIONS.DOOR, rec))
;
                    }
                }
            }

            if (nodeList[i].Name == "PORTALLIST")
            {
                PORTALList = nodeList[i].ChildNodes;

                for (int k = 0; k < PORTALList.Count; k++)
                {
                    if (PORTALList[k].Name == "PORTAL")
                    {
                        attrList = PORTALList[k].Attributes;
                        Rectangle PortalRec = new Rectangle();
                        PortalRec.X = toInt(attrList["X"].Value);
                        PortalRec.Y = toInt(attrList["Y"].Value);
                        PortalRec.Width = toInt(attrList["WIDTH"].Value);
                        PortalRec.Height = toInt(attrList["HEIGHT"].Value);

                        Collision PortaColl = new Collision(COLLISIONS.PORTAL
, PortalRec);
                        ObjectA portal = new ObjectA(game, "PORTAL",
PortalRec, PortaColl, lvl);
                        PortaColl.OBJA = portal;
                        lvl.AddObjectA(portal);
                        lvl.AddCOLLISION(PortaColl);
                    }
                }
            }
            if (nodeList[i].Name == "ELECTROLIST")
            {
```

```
                    XmlNodeList ElectroChilds;
                    XmlNodeList AzionatoreChilds;
                    ObjectA Leva = null;
                    ELECTROList = nodeList[i].ChildNodes;

                    for (int k = 0; k < ELECTROList.Count; k++)
                    {
                        if (ELECTROList[k].Name == "ELECTRO")
                        {
                            attrList = ELECTROList[k].Attributes;
                            Rectangle CollisionRecE = new Rectangle();
                            CollisionRecE.X = toInt(attrList["X"].Value);
                            CollisionRecE.Y = toInt(attrList["Y"].Value);
                            CollisionRecE.Width = toInt(attrList["WIDTH"].Value);
                            CollisionRecE.Height = toInt(attrList["HEIGHT"].
Value);

                            ElectroChilds = ELECTROList[k].ChildNodes;

                            attrList = ElectroChilds[0].Attributes;
                            Rectangle DrawingRecE = new Rectangle();
                            DrawingRecE.X = toInt(attrList["X"].Value);
                            DrawingRecE.Y = toInt(attrList["Y"].Value);
                            DrawingRecE.Width = toInt(attrList["WIDTH"].Value);
                            DrawingRecE.Height = toInt(attrList["HEIGHT"].Value);

                            ObjectA Electro = new ObjectA(game, "ELECTRO",
DrawingRecE, new Collision(COLLISIONS.ELECTRO, CollisionRecE), lvl);

                            if (ELECTROList[k].ChildNodes.Count > 1)
                            {
                                AzionatoreChilds = ELECTROList[k].ChildNodes[1].
ChildNodes;

                                attrList = ELECTROList[k].ChildNodes[1].
Attributes;
                                Rectangle CollisionRecA = new Rectangle();
                                CollisionRecA.X = toInt(attrList["X"].Value);
                                CollisionRecA.Y = toInt(attrList["Y"].Value);
                                CollisionRecA.Width = toInt(attrList["WIDTH"].
Value);
                                CollisionRecA.Height = toInt(attrList["HEIGHT"].
Value);

                                attrList = AzionatoreChilds[0].Attributes;
                                Rectangle DrawingRecA = new Rectangle();
                                DrawingRecA.X = toInt(attrList["X"].Value);
                                DrawingRecA.Y = toInt(attrList["Y"].Value);
                                DrawingRecA.Width = toInt(attrList["WIDTH"].
Value);
                                DrawingRecA.Height = toInt(attrList["HEIGHT"].
Value);

                                Collision levaColl = new Collision(COLLISIONS.
LEVA, CollisionRecA);
                                Leva = new ObjectA(game, "LEVA", DrawingRecA,
levaColl, lvl);
                                levaColl.OBJA = Leva;

                                lvl.AddObjectA(Leva);
                                lvl.AddCOLLISION(Leva.COLLISION);
                            }

                            Leva.AddSlave(Electro);
                            lvl.AddObjectA(Electro);
                        }
                    }
                }
                if (nodeList[i].Name == "BOMBLIST")
                {
```

```
                BOMBList = nodeList[i].ChildNodes;

                for (int k = 0; k < BOMBList.Count; k++)
                {
                    if (BOMBList[k].Name == "BOMB")
                    {
                        attrList = BOMBList[k].Attributes;
                        Rectangle CollisionRecB = new Rectangle();
                        CollisionRecB.X = toInt(attrList["X"].Value);
                        CollisionRecB.Y = toInt(attrList["Y"].Value);
                        CollisionRecB.Width = toInt(attrList["WIDTH"].Value);
                        CollisionRecB.Height = toInt(attrList["HEIGHT"].
Value);


                        attrList = BOMBList[k].FirstChild.Attributes;
                        Rectangle DrawingRecB = new Rectangle();
                        DrawingRecB.X = toInt(attrList["X"].Value);
                        DrawingRecB.Y = toInt(attrList["Y"].Value);
                        DrawingRecB.Width = toInt(attrList["WIDTH"].Value);
                        DrawingRecB.Height = toInt(attrList["HEIGHT"].Value);

                        Collision bombColl = new Collision(COLLISIONS.BOMB,
CollisionRecB);
                        ObjectA Bomb = new ObjectA(game, "BOMB", DrawingRecB,
 bombColl, lvl);
                        bombColl.OBJA = Bomb;

                        lvl.AddObjectA(Bomb);
                    }
                }
            }

            if (nodeList[i].Name == "BOMBINNESCOLIST")
            {
                BOMBINNESCOList = nodeList[i].ChildNodes;

                for (int k = 0; k < BOMBINNESCOList.Count; k++)
                {
                    if (BOMBINNESCOList[k].Name == "BOMBINNESCO")
                    {
                        attrList = BOMBINNESCOList[k].Attributes;
                        Rectangle CollisionRecB = new Rectangle();
                        CollisionRecB.X = toInt(attrList["X"].Value);
                        CollisionRecB.Y = toInt(attrList["Y"].Value);
                        CollisionRecB.Width = toInt(attrList["WIDTH"].Value);
                        CollisionRecB.Height = toInt(attrList["HEIGHT"].
Value);

                        attrList = BOMBINNESCOList[k].FirstChild.Attributes;
                        Rectangle DrawingRecB = new Rectangle();
                        DrawingRecB.X = toInt(attrList["X"].Value);
                        DrawingRecB.Y = toInt(attrList["Y"].Value);
                        DrawingRecB.Width = toInt(attrList["WIDTH"].Value);
                        DrawingRecB.Height = toInt(attrList["HEIGHT"].Value);

                        Collision bombInnColl = new Collision(COLLISIONS.
BOMBINNESCO, CollisionRecB);
                        ObjectA BombInnesco = new ObjectA(game, "BOMBINNESCO"
, DrawingRecB, bombInnColl, lvl);
                        bombInnColl.OBJA = BombInnesco;

                        lvl.AddObjectA(BombInnesco);
                    }
                }
            }

            if (nodeList[i].Name == "MATTATOIOLIST")
            {
                XmlNodeList MattatoioChilds;
```

```
                XmlNodeList AzionatoreChilds;

                MATTATOIOList = nodeList[i].ChildNodes;
                for (int k = 0; k < MATTATOIOList.Count; k++)
                {
                    if (MATTATOIOList[k].Name == "MATTATOIO")
                    {
                        attrList = MATTATOIOList[k].Attributes;
                        Rectangle CollisionRecM = new Rectangle();
                        CollisionRecM.X = toInt(attrList["X"].Value);
                        CollisionRecM.Y = toInt(attrList["Y"].Value);
                        CollisionRecM.Width = toInt(attrList["WIDTH"].Value);
                        CollisionRecM.Height = toInt(attrList["HEIGHT"].
Value);

                        MattatoioChilds = MATTATOIOList[k].ChildNodes;

                        attrList = MattatoioChilds[0].Attributes;
                        Rectangle DrawingRecM = new Rectangle();
                        DrawingRecM.X = toInt(attrList["X"].Value);
                        DrawingRecM.Y = toInt(attrList["Y"].Value);
                        DrawingRecM.Width = toInt(attrList["WIDTH"].Value);
                        DrawingRecM.Height = toInt(attrList["HEIGHT"].Value);

                        ObjectA Mattatoio = new ObjectA(game, "MATTATOIO",
DrawingRecM, new Collision(COLLISIONS.MATTATOIO, CollisionRecM), lvl);

                        AzionatoreChilds = MATTATOIOList[k].ChildNodes[1].
ChildNodes;

                        attrList = MATTATOIOList[k].ChildNodes[1].Attributes;
                        Rectangle CollisionRecA = new Rectangle();
                        CollisionRecA.X = toInt(attrList["X"].Value);
                        CollisionRecA.Y = toInt(attrList["Y"].Value);
                        CollisionRecA.Width = toInt(attrList["WIDTH"].Value);
                        CollisionRecA.Height = toInt(attrList["HEIGHT"].
Value);

                        attrList = AzionatoreChilds[0].Attributes;
                        Rectangle DrawingRecA = new Rectangle();
                        DrawingRecA.X = toInt(attrList["X"].Value);
                        DrawingRecA.Y = toInt(attrList["Y"].Value);
                        DrawingRecA.Width = toInt(attrList["WIDTH"].Value);
                        DrawingRecA.Height = toInt(attrList["HEIGHT"].Value);

                        Collision levaColl = new Collision(COLLISIONS.LEVA,
CollisionRecA);
                        ObjectA Leva = new ObjectA(game, "LEVA", DrawingRecA,
 levaColl, lvl);
                        levaColl.OBJA = Leva;

                        Leva.AddSlave(Mattatoio);
                        lvl.AddObjectA(Mattatoio);
                        lvl.AddObjectA(Leva);
                        lvl.AddCOLLISION(Leva.COLLISION);
                    }
                }
            }

            if (nodeList[i].Name == "BOTOLALIST")
            {
                XmlNodeList BotolaChilds;
                XmlNodeList AzionatoreChilds;

                MATTATOIOList = nodeList[i].ChildNodes;
                BOTOLAList = nodeList[i].ChildNodes;
                for (int k = 0; k < BOTOLAList.Count; k++)
                {
                    if (BOTOLAList[k].Name == "BOTOLA")
                    {
```

```
                        attrList = BOTOLAList[k].Attributes;
                        Rectangle CollisionRecM = new Rectangle();
                        CollisionRecM.X = toInt(attrList["X"].Value);
                        CollisionRecM.Y = toInt(attrList["Y"].Value);
                        CollisionRecM.Width = toInt(attrList["WIDTH"].Value);
                        CollisionRecM.Height = toInt(attrList["HEIGHT"].
Value);

                        BotolaChilds = MATTATOIOList[k].ChildNodes;

                        attrList = BotolaChilds[0].Attributes;
                        Rectangle DrawingRecM = new Rectangle();
                        DrawingRecM.X = toInt(attrList["X"].Value);
                        DrawingRecM.Y = toInt(attrList["Y"].Value);
                        DrawingRecM.Width = toInt(attrList["WIDTH"].Value);
                        DrawingRecM.Height = toInt(attrList["HEIGHT"].Value);

                        ObjectA Botola = new ObjectA(game, "BOTOLA",
DrawingRecM, new Collision(COLLISIONS.BOTOLA, CollisionRecM), lvl);

                        AzionatoreChilds = BOTOLAList[k].ChildNodes[1].
ChildNodes;

                        attrList = BOTOLAList[k].ChildNodes[1].Attributes;
                        Rectangle CollisionRecA = new Rectangle();
                        CollisionRecA.X = toInt(attrList["X"].Value);
                        CollisionRecA.Y = toInt(attrList["Y"].Value);
                        CollisionRecA.Width = toInt(attrList["WIDTH"].Value);
                        CollisionRecA.Height = toInt(attrList["HEIGHT"].
Value);

                        attrList = AzionatoreChilds[0].Attributes;
                        Rectangle DrawingRecA = new Rectangle();
                        DrawingRecA.X = toInt(attrList["X"].Value);
                        DrawingRecA.Y = toInt(attrList["Y"].Value);
                        DrawingRecA.Width = toInt(attrList["WIDTH"].Value);
                        DrawingRecA.Height = toInt(attrList["HEIGHT"].Value);

                        Collision levaColl = new Collision(COLLISIONS.LEVA,
CollisionRecA);
                        ObjectA Leva = new ObjectA(game, "LEVA", DrawingRecA,
 levaColl, lvl);
                        levaColl.OBJA = Leva;

                        Leva.AddSlave(Botola);
                        lvl.AddObjectA(Botola);
                        lvl.AddObjectA(Leva);
                        lvl.AddCOLLISION(Leva.COLLISION);
                    }
                }
            }

            if (nodeList[i].Name == "GODOWNDLIST")
            {
                GODOWNDList = nodeList[i].ChildNodes;
                for (int k = 0; k < GODOWNDList.Count; k++)
                {
                    if (GODOWNDList[k].Name == "GODOWND")
                    {
                        attrList = GODOWNDList[k].Attributes;
                        Rectangle rec = new Rectangle();
                        rec.X = toInt(attrList["X"].Value);
                        rec.Y = toInt(attrList["Y"].Value);
                        rec.Width = toInt(attrList["WIDTH"].Value);
                        rec.Height = toInt(attrList["HEIGHT"].Value);

                        lvl.AddCOLLISION(new Collision(COLLISIONS.GODOWND,
rec));
                    }
                }
```

```
            }

            if (nodeList[i].Name == "LEVALIST")
            {
                LEVE = nodeList[i].ChildNodes;
                for (int k = 0; k < LEVE.Count; k++)
                {
                    if (LEVE[k].Name == "LEVA")
                    {
                        attrList = LEVE[k].Attributes;
                        Rectangle rec = new Rectangle();
                        rec.X = toInt(attrList["X"].Value);
                        rec.Y = toInt(attrList["Y"].Value);
                        rec.Width = toInt(attrList["WIDTH"].Value);
                        rec.Height = toInt(attrList["HEIGHT"].Value);

                        lvl.AddCOLLISION(new Collision(COLLISIONS.LEVA, rec))↙
;
                    }
                }
            }

            if (nodeList[i].Name == "GRAYLIST")
            {
                Grayareas = nodeList[i].ChildNodes;
                for (int k = 0; k < Grayareas.Count; k++)
                {
                    if (Grayareas[k].Name == "GRAY")
                    {
                        attrList = Grayareas[k].Attributes;
                        Rectangle rec = new Rectangle();
                        rec.X = toInt(attrList["X"].Value);
                        rec.Y = toInt(attrList["Y"].Value);
                        rec.Width = toInt(attrList["WIDTH"].Value);
                        rec.Height = toInt(attrList["HEIGHT"].Value);

                        lvl.AddCOLLISION(new Collision(COLLISIONS.GRAYAREA,  ↙
rec));
                    }
                }
            }

            if (nodeList[i].Name == "BLACKLIST")
            {
                Blackareas = nodeList[i].ChildNodes;
                for (int k = 0; k < Blackareas.Count; k++)
                {
                    if (Blackareas[k].Name == "BLACK")
                    {
                        attrList = Blackareas[k].Attributes;
                        Rectangle rec = new Rectangle();
                        rec.X = toInt(attrList["X"].Value);
                        rec.Y = toInt(attrList["Y"].Value);
                        rec.Width = toInt(attrList["WIDTH"].Value);
                        rec.Height = toInt(attrList["HEIGHT"].Value);

                        lvl.AddCOLLISION(new Collision(COLLISIONS.BLACKAREA, ↙
rec));
                    }
                }
            }

            if (nodeList[i].Name == "SENTINELLA")
            {

                string sentinella = nodeList[i].Attributes["SET"].Value;

                if (sentinella == "Y")
                {
                    lvl.Sentinella = true;
```

```
                    ObjectA OMM = new ObjectA(game, "OMM", new Rectangle(500,
 50, 0, 0), new Collision(COLLISIONS.SHOCK, new Rectangle()), lvl);
                    lvl.AddObjectA(OMM);
                }
                else
                    lvl.Sentinella = false;
            }

            if (nodeList[i].Name == "SPAWNSLIGLIST")
            {
                SpawSligs = nodeList[i].ChildNodes;
                XmlNodeList Sligs;
                for (int k = 0; k < SpawSligs.Count; k++)
                {
                    if (SpawSligs[k].Name == "SPAWNSLIG")
                    {
                        attrList = SpawSligs[k].Attributes;
                        Rectangle SpawnRec = new Rectangle();
                        SpawnRec.X = toInt(attrList["X"].Value);
                        SpawnRec.Y = toInt(attrList["Y"].Value);
                        SpawnRec.Width = toInt(attrList["WIDTH"].Value);
                        SpawnRec.Height = toInt(attrList["HEIGHT"].Value);

                        Sligs = SpawSligs[k].ChildNodes;

                        for (int s = 0; s < Sligs.Count; s++)
                        {

                            attrList = Sligs[s].Attributes;
                            Slig slig = new Slig(game, attrList["STATE"].
Value, toInt(attrList["AI"].Value), SpawnRec.Location);
                            slig.levelDefault = lvl;
                            lvl.AddSlig(slig);
                        }
                    }
                }
            }

            if (nodeList[i].Name == "SPAWNMUDOKONLIST")
            {
                SpawMudokons = nodeList[i].ChildNodes;
                XmlNodeList Mudokons;
                for (int k = 0; k < SpawMudokons.Count; k++)
                {
                    if (SpawMudokons[k].Name == "SPAWNMUDOKON")
                    {
                        attrList = SpawMudokons[k].Attributes;
                        Rectangle SpawnRec = new Rectangle();
                        SpawnRec.X = toInt(attrList["X"].Value);
                        SpawnRec.Y = toInt(attrList["Y"].Value);
                        SpawnRec.Width = toInt(attrList["WIDTH"].Value);
                        SpawnRec.Height = toInt(attrList["HEIGHT"].Value);

                        Mudokons = SpawMudokons[k].ChildNodes;

                        for (int m = 0; m < Mudokons.Count; m++)
                        {
                            attrList = Mudokons[m].Attributes;
                            Mudokon mudokon = new Mudokon(game, attrList[
"STATE"].Value, SpawnRec.Location);
                            lvl.AddMudokon(mudokon);
                        }
                    }
                }
            }
        }

        for (int i = 0; i < videosName.Count; i++)
        {
            if (videosName[i] == lvl.Enviorement)
```

```
                {
                    lvl.AddVideo(videos[i], videosRec[i]);
                }
            }
            lvlNameList.Add(lvlName);

            lvl.SetDifficult();
            return lvl;
        }


        bool IntersectsGodown(Rectangle recToTest, Level lvl)...

        public static Level getLevel(string enviorement)...

        public static Level getLevelByName(string name)...

        int toInt(String val)...
    }
}
```

```
using ...

namespace TheGame
{
    //Gestisce il sondaggio proposto al giocatore a fine percorso
    //[p.76,77]
    public class Report
    {
        Game1 game;
        public Texture2D BackGround;
        public List<Level> LEVELS;
        public int LVL_INDEX = 0;

        public Rectangle DrawRecLvl;

        public bool FDown = true;
        public bool FUp = true;
        public bool FOne = true;
        public bool FTwo = true;

        public SoundEffect UPDOWN;
        public SoundEffect OK;
        public SoundEffect NO;


        public Report(Game1 game)
        {
            this.game = game;
            LEVELS = new List<Level>();
            BackGround = game.Content.Load<Texture2D>("REPORT\\Background");

            DrawRecLvl = new Rectangle(200, 150, 240, 180);

            UPDOWN = game.Content.Load<SoundEffect>("Sound\\PORTALENTER");
            OK = game.Content.Load<SoundEffect>("Sound\\OK");
            NO = game.Content.Load<SoundEffect>("Sound\\PETO");
        }

        public void Control()
        {
            if (game.downDown & FDown & LVL_INDEX < LEVELS.Count - 1)
            {
                FDown = false;
                UPDOWN.Play(0.1f);
                LVL_INDEX++;
            }
            if (!game.downDown)
                FDown = true;

            if (game.upDown & FUp & LVL_INDEX > 0)
            {
                FUp = false;
                UPDOWN.Play(0.1f);
                LVL_INDEX--;
            }
            if (!game.upDown)
                FUp = true;

            if (game.OneDown & FOne)//Giudizio positivo(livello superato con
    facilità)
            {
                FOne = false;
                OK.Play(0.1f);
                Level lvlMod = LevelMaker.getLevelByName(LEVELS[LVL_INDEX].NAME);
                lvlMod.Modify("DIFF_UP");
                lvlMod.SetDifficult();
                LEVELS.Remove(LEVELS[LVL_INDEX]);
                LVL_INDEX = 0;
                if (LEVELS.Count == 0)
                {
```

```
                    game.ReInit();
                    game.SHOW_REPORT = false;
                    game.NUMLVLGEN = 0;
                }
            }
            if (!game.OneDown)
                FOne = true;

            if (game.TwoDown & FTwo)///Giudizio negativo(livello superato con
    difficoltà)
            {
                FTwo = false;
                NO.Play(0.1f);
                Level lvlMod = LevelMaker.getLevelByName(LEVELS[LVL_INDEX].NAME);
                lvlMod.Modify("DIFF_DOWN");
                lvlMod.SetDifficult();

                LEVELS.Remove(LEVELS[LVL_INDEX]);
                LVL_INDEX = 0;
                if (LEVELS.Count == 0)
                {
                    game.ReInit();
                    game.SHOW_REPORT = false;
                    game.NUMLVLGEN = 0;
                }
            }
            if (!game.TwoDown)
                FTwo = true;
        }
    }
}
```

```
//Gestisce grafica ed effetti sonori degli oggetti attivi
//[pp.65]
public class ObjectA : Character
{
    VARS

    //Definisce varie tipologie di ObjectA
    public ObjectA(Game1 game, string NAME,Rectangle DrawingRec,Collision COLLISION,Level LEVEL)
    {
        this.game = game;
        this.NAME = NAME;
        this.DrawingRec = DrawingRec;
        this.COLLISION = COLLISION;

        if (NAME == "HOISTADVISE")
        {
            SHEETD = game.Content.Load<Texture2D>("Sprite_Sheet\\ObjectA\\HOISTADVISE");

            DEFAULT = new Animation(SHEETD, SHEETD, 1000 / 20, 62, "DEFAULT", 0, 0, -1, 0, 0);
            ACTIVATED = new Animation(null, null, 1000 / 15, 20, "ACTIVATED", 0, 0, -1, 0, 0);

            this.LEVEL = LEVEL;
            CurrentAnimation = DEFAULT;
            PrevAnimation = CurrentAnimation;
            objControl = new ObjectControl(this, game);

            objControl.CURRENT_STATE = objControl.DEFAULT;
        }


        if (NAME == "BOMBINNESCO")
        {
            Sound_BOMBINNESCO = game.Content.Load<SoundEffect>("Sound\\BOMBINNESCO");

            SHEETD = game.Content.Load<Texture2D>("Sprite_Sheet\\ObjectA\\BOMBINNESCOD");
            SHEETA = game.Content.Load<Texture2D>("Sprite_Sheet\\ObjectA\\BOMBINNESCOA");

            DEFAULT = new Animation(SHEETD, SHEETD, 1000 / 15, 1, "DEFAULT", 0, 0, -1, 0, 0);
            ACTIVATED = new Animation(SHEETA, SHEETA, 1000 / 15, 19, "ACTIVATED",new Sound(Sound_BOMBINNESCO, 9), 0, 0, -1, 0, 0);

            this.LEVEL = LEVEL;
            CurrentAnimation = DEFAULT;
            PrevAnimation = CurrentAnimation;
            objControl = new ObjectControl(this, game);

            objControl.ACTIVE = true;
            objControl.CURRENT_STATE = objControl.ACTIVATED;

            LEVEL.AddCOLLISION(COLLISION);
        }

        if (NAME == "BOMB")
        {
            Sound_BOMBINNESCO = game.Content.Load<SoundEffect>("Sound\\BOMBINNESCO");

            SHEETD = game.Content.Load<Texture2D>("Sprite_Sheet\\ObjectA\\BOMBD");

            DEFAULT = new Animation(SHEETD, SHEETD, 1000 / 15, 3, "DEFAULT", 0, 0, -1, 0, 0);
```

```
            ACTIVATED = new Animation(null, null, 1000 / 15, 0, "ACTIVATED", ↙
0, 0, -1, 0, 0);

            this.LEVEL = LEVEL;
            CurrentAnimation = DEFAULT;
            PrevAnimation = CurrentAnimation;
            objControl = new ObjectControl(this, game);

            objControl.CURRENT_STATE = objControl.DEFAULT;

            LEVEL.AddCOLLISION(COLLISION);

        }

        if (NAME == "ELECTRO")
        {
            Sound_ELECTRIC = new Sound(game.Content.Load<SoundEffect>("Sound\↙
\ELECTRIC"), 2);
            Sound_SHOCKED = game.Content.Load<SoundEffect>("Sound\\SHOCKED");

            SHEETD = game.Content.Load<Texture2D>("Sprite_Sheet\\ObjectA\\   ↙
ELECTRO");

            DEFAULT = new Animation(SHEETD, SHEETD, 1000 / 30, 50, "DEFAULT",↙
 Sound_ELECTRIC, 0, 0, -1, 0, 0);
            ACTIVATED = new Animation(SHEETD, SHEETD, 1000 / 30, 50,         ↙
"ACTIVATED",new Sound(Sound_SHOCKED, 2), 0, 0, -1, 0, 0);
            INACTIVE = new Animation(null, null, 1000 / 15, 0, "ACTIVATED", 0↙
, 0, -1, 0, 0);

            this.LEVEL = LEVEL;
            CurrentAnimation = DEFAULT;
            PrevAnimation = CurrentAnimation;
            objControl = new ObjectControl(this, game);

            objControl.CURRENT_STATE = objControl.ELECTRODEFAULT;

            LEVEL.AddCOLLISION(COLLISION);

        }

        if (NAME == "LEVA")
        {
            SHEETD = game.Content.Load<Texture2D>("LVL\\Object\\LEVA");

            DEFAULT = new Animation(SHEETD, SHEETD, 1000 / 15, 1, "DEFAULT", ↙
0, 0, -1, 0, 0);
            ACTIVATED = new Animation(null, null, 1000 / 15, 0, "ACTIVATED", ↙
0, 0, -1, 0, 0);

            this.LEVEL = LEVEL;
            CurrentAnimation = DEFAULT;
            PrevAnimation = CurrentAnimation;
            objControl = new ObjectControl(this, game);

            objControl.CURRENT_STATE = objControl.DEFAULT;
            objControl.ACTIVE = false;
        }

        if (NAME == "OMM")
        {
            Sound_SHOCKED = game.Content.Load<SoundEffect>("Sound\\SHOCKED");
            SHEETA = game.Content.Load<Texture2D>("Sprite_Sheet\\ObjectA\\   ↙
SENTINELLAA");
            SHEETD = game.Content.Load<Texture2D>("Sprite_Sheet\\ObjectA\\   ↙
SENTINELLAD");

            DEFAULT = new Animation(SHEETD, SHEETD, 1000 / 15, 26, "DEFAULT",↙
 0, 0, -1, 0, 0);
            ACTIVATED = new Animation(SHEETA, SHEETA, 1000 / 15, 25,         ↙
```

```
"ACTIVATED",new Sound(Sound_SHOCKED, 10), 0, 0, -1, 0, 0);

            this.LEVEL = LEVEL;
            CurrentAnimation = DEFAULT;
            PrevAnimation = CurrentAnimation;
            objControl = new ObjectControl(this, game);

            objControl.CURRENT_STATE = objControl.DEFAULT;
            objControl.ACTIVE = false;
        }

        if (NAME == "BOTOLA")
        {
            Sound_BOTOLA = game.Content.Load<SoundEffect>("Sound\\BOTOLA");

            SHEETA = game.Content.Load<Texture2D>("Sprite_Sheet\\ObjectA\\
BOTOLA");
            SHEETD = game.Content.Load<Texture2D>("LVL\\Object\\BOTOLA");

            DEFAULT = new Animation(SHEETD, SHEETD, 1000 / 15, 1, "DEFAULT",
0, 0, -1, 0, 0);
            ACTIVATED = new Animation(SHEETA, SHEETA, 1000 / 15, 9,
"ACTIVATED", new Sound(Sound_BOTOLA, 1), 0, 0, -1, 0, 0);

            this.LEVEL = LEVEL;
            CurrentAnimation = DEFAULT;
            PrevAnimation = CurrentAnimation;
            objControl = new ObjectControl(this, game);

            objControl.CURRENT_STATE = objControl.DEFAULT;
            objControl.ACTIVE = false;
        }

        if (NAME == "MATTATOIO")
        {
            Sound_MATTATOIOA = game.Content.Load<SoundEffect>("Sound\\
MATTATOIOA");

            SHEETA = game.Content.Load<Texture2D>("Sprite_Sheet\\ObjectA\\
MATTATOIOA");
            SHEETD = game.Content.Load<Texture2D>("Sprite_Sheet\\ObjectA\\
MATTATOIOD");

            DEFAULT = new Animation(SHEETD, SHEETD, 1000 / 20, 8, "DEFAULT",
0, 0, -1, 0, 0);
            ACTIVATED = new Animation(SHEETA, SHEETA, 1000 / 20, 16,
"ACTIVATED", new Sound(Sound_MATTATOIOA, 8), 0, 0, -1, 0, 0);

            this.LEVEL = LEVEL;
            CurrentAnimation = DEFAULT;
            PrevAnimation = CurrentAnimation;
            objControl = new ObjectControl(this, game);

            objControl.CURRENT_STATE = objControl.DEFAULT;
            objControl.ACTIVE = false;
        }

        if (NAME == "PORTAL")
        {
            Sound_PORTALDEFAULT = game.Content.Load<SoundEffect>("Sound\\
PORTALDEFAULT");
            Sound_PORTALFLYAWAY = game.Content.Load<SoundEffect>("Sound\\
PORTALFLYAWAY");
            Sound_PORTALOPENING = game.Content.Load<SoundEffect>("Sound\\
PORTALOPENING");
            Sound_PORTALOPEN = game.Content.Load<SoundEffect>("Sound\\
PORTALOPEN");

            PortalDefault = game.Content.Load<Texture2D>("Sprite_Sheet\\
ObjectA\\PORTALD1");
```

```
            PortalFlyAway = game.Content.Load<Texture2D>("Sprite_Sheet\\
ObjectA\\PORTALFLYAWAY");
            PortalOpening = game.Content.Load<Texture2D>("Sprite_Sheet\\
ObjectA\\PORTALOPENING");
            PortalOpen = game.Content.Load<Texture2D>("Sprite_Sheet\\ObjectA\
\PORTALOPEN");

            PORTALDEFAULT = new Animation(PortalDefault, PortalDefault, 1000
/ 20, 25, "PORTALDEFAULT", new Sound(Sound_PORTALDEFAULT, 8), 0, 0, -1, 0, 0,
 0, 0);
            PORTALFLYAWAY = new Animation(PortalFlyAway, PortalFlyAway, 1000
/ 20, 8, "PORTALFLYAWAY", new Sound(Sound_PORTALFLYAWAY, 1), 0, 0, -1, 0, 0,
0, 0);
            PORTALOPENING = new Animation(PortalOpening, PortalOpening, 1000
/ 20, 24, "PORTALOPENING", new Sound(Sound_PORTALOPENING, 1), 0, 0, -1, 0, 0,
 60, 60);
            PORTALOPEN = new Animation(PortalOpen, PortalOpen, 1000 / 20, 25,
 "PORTALOPEN", new Sound(Sound_PORTALOPEN, 1), 0, 0, -1, 0, 0, 60, 60);

            this.LEVEL = LEVEL;
            CurrentAnimation = PORTALDEFAULT;
            PrevAnimation = CurrentAnimation;
            objControl = new ObjectControl(this, game);

            objControl.CURRENT_STATE = objControl.PORTALDEFAULT;
            objControl.ACTIVE = false;
        }
    }


    public void AddSlave(ObjectA slave)...

    public void DEACTIVATE()...

    public void ACTIVATE()...

    //Gestisce la spritesheet dell'animazione corrente.
    //Definisce quale frame della spritesheet deve essere disegnato.
    public void AnimateObj(Animation anim)...
}
```

```
//ObjectControl definisce l'automa per la gestione del sistema
 //di controllo del personaggio principale
 //[pp.43]
 public class ObjectControl
 {
     VARS

     public ObjectControl(ObjectA Obj, Game1 game)
     {
         this.game = game;
         this.Obj = Obj;

         ACTIVE = false;

         PRECONDITIONS = new PRECONDITIONSo(this);
         ACTIONS = new ACTIONSo(this);

         DEFAULT = new STATEo("DEFAULT", this);
         ACTIVATED = new STATEo("ACTIVATED", this);

         PORTALDEFAULT = new STATEo("PORTALDEFAULT", this);
         PORTALOPEN = new STATEo("PORTALOPEN", this);
         INVISIBLE = new STATEo("INVISIBLE", this);

         ELECTRODEFAULT = new STATEo("ELECTRODEFAULT", this);
         ELECTROSHOCK = new STATEo("ELECTROSHOCK", this);
         ELECTROINACTIVE = new STATEo("ELECTROINACTIVE", this);

         PORTALDEFAULT.LinkTO(INVISIBLE).addPRECONDITION(PRECONDITIONS.       ↙
 PORTADEFAULTLCOLLISION, ACTIONS.PORTALFLYAWAY);
         PORTALDEFAULT.LinkTO(PORTALOPEN).addPRECONDITION(PRECONDITIONS.ABEOMM↙
 , ACTIONS.PORTALOPENING);
         PORTALOPEN.LinkTO(INVISIBLE).addPRECONDITION(PRECONDITIONS.ABEENDOMM)↙
 ;

         ELECTRODEFAULT.LinkTO(ELECTROSHOCK).addPRECONDITION(PRECONDITIONS.   ↙
 ELECTROTOUCH);
         ELECTROSHOCK.LinkTO(ELECTRODEFAULT).addPRECONDITION(PRECONDITIONS.   ↙
 ENDSTATE);
         ELECTRODEFAULT.LinkTO(ELECTROINACTIVE).addPRECONDITION(PRECONDITIONS.↙
 ELECTROD);
         ELECTROINACTIVE.LinkTO(ELECTRODEFAULT).addPRECONDITION(PRECONDITIONS.↙
 ELECTROA);

         DEFAULT.LinkTO(ACTIVATED).addPRECONDITION(PRECONDITIONS.ACTIVATED);
         ACTIVATED.LinkTO(DEFAULT).addPRECONDITION(PRECONDITIONS.DEACTIVATED);
         ACTIVATED.LinkTO(ACTIVATED).addPRECONDITION(PRECONDITIONS.ENDSTATE);

         PREV_STATE = CURRENT_STATE;

     }

     public void RESET()...

     //Esegue l'automa definito nel costruttore di ObjctControl
     //Viene chiamato nel metodo Update della classe Game1
     //Simile al caso delle classi SLigAi e MudokonAI
     //[pp.48,49]
     public void ROUTINE()...
 }


 //Definisce gli stati dell'automa
 //Ogni stato contiene una o più animazioni da eseguire
 //Simile al caso delle classi SLigAi e MudokonAI
 //[p.45]
 public class STATEo ...

 //Definisce gli archi dell'automa
 //[p.47]
```

```
public class ARCo ...

//Definisce le azioni che possono essere associate ad una precondizione
//Ogni azione può contenere una o più precondizioni.
//[p.45]
public class ACTIONo ...

//Contiene i riferimenti a tutte le istanze della classe ACTIONo
public class ACTIONSo ...

//Definisce le precondizioni che devono essere aggiunte ad un arco per
transitare da uno stato all'altro.
//[p.46]
public class PRECONDITIONo ...

//Contiene i riferimenti a tutte le istanze della classe PRECONDITIONo
public class PRECONDITIONSo ...
```