

# Pokaiju

Luca Barattini     Samuele Carafassi     Andrea Castorina  
Jia Hao Guo     Michael Pierantoni

20 aprile 2022

## Sommario

Pokaiju, composizione di "Po" (dall'inglese pocket, ovvero "tascabile") e "Kaiju" (dal giapponese kaiju, ovvero "mostro") è un RPG (Role Playing Game) single player ispirato alla saga videoludica del colosso nipponico The Pokémon Company™. Il software ti permette d'impersonare un giocatore che sta partendo all'avventura con il suo primo pokaiju, in questo viaggio potrà scoprirne e catturarne altri e potrà confrontare il proprio team (composta da massimo sei pokaiju) con i teams degli NPC avversari attraverso dei combattimenti. Il giocatore inoltre potrà incontrare ed interagire con NPC amichevoli, tipo i mercanti, dove potrà comprare oggetti utili per i suoi pokaiju, o curatori per rimettere in piedi il proprio team dopo uno scontro impegnativo .

# Indice

<b>1</b>	<b>Analisi</b>	<b>2</b>
1.1	Requisiti . . . . .	2
1.2	Analisi e modello del dominio . . . . .	3
<b>2</b>	<b>Design</b>	<b>5</b>
2.1	Architettura . . . . .	5
2.2	Design dettagliato . . . . .	6
<b>3</b>	<b>Sviluppo</b>	<b>16</b>
3.1	Testing automatizzato . . . . .	16
3.2	Metodologia di lavoro . . . . .	17
3.3	Note di sviluppo . . . . .	19
<b>4</b>	<b>Commenti finali</b>	<b>21</b>
4.1	Autovalutazione e lavori futuri . . . . .	21
<b>A</b>	<b>Guida utente</b>	<b>23</b>
<b>B</b>	<b>Esercitazioni di laboratorio</b>	<b>30</b>
B.1	Link consegne . . . . .	30
B.1.1	Luca Barattini . . . . .	30
B.1.2	Samuele Carafassi . . . . .	30
B.1.3	Andrea Castorina . . . . .	31
B.1.4	Jia Hao Guo . . . . .	31

# Capitolo 1

## Analisi

Lo scopo del gioco è quello di sconfiggere avversari sempre più ardui e per far sì che succeda, bisogna migliorare il proprio team attraverso la cattura di pokaiju più forti, la meccanica di levelling (per ogni incontro vinto, il proprio pokaiju guadagna punti esperienza) e una meccanica di evoluzione che permette ai propri mostri di cambiare forma e di migliorarsi; infatti per far evolvere il proprio pokaiju ci sono due modalità: evoluzione tramite il raggiungimento di un preciso livello o usando un' oggetto in possesso al giocatore. Il giocatore sarà libero di muoversi attraverso la mappa, entrare negli edifici e interagire con tutti gli NPC presenti.

### 1.1 Requisiti

All'avvio del gioco, all'utente verrà presentato un menu' in cui sarà possibile creare il proprio personaggio ed iniziare una partita. Il giocatore comparirà in una mappa dove potrà muoversi tramite input da tastiera, potrà aprire il menù di gestione del proprio equipaggiamento/team ed interagire con l'ambiente circostante.

#### Requisiti funzionali

- Il giocatore deve essere in grado di muoversi liberamente in qualsiasi direzione nella mappa di gioco a meno che non sia presente un ostacolo o un' avversario da fronteggiare per avanzare nell'avventura.
- Deve essere possibile combattere contro i pokaiju sia selvatici (in questo caso, dev'essere possibile anche catturarli) che di NPC allenatori.

- La battaglia si può considerare vinta solo quando i pokaiju avversari sono stati tutti sconfitti (o catturati), in caso contrario il giocatore perderà un ingente somma di denaro ma i suoi pokaiju verranno curati totalmente.
- I soldi devono poter essere guadagnati dopo una battaglia contro un allenatore solo in caso di vittoria.
- Il giocatore deve poter catturare i pokaiju, a meno che non siano appartenenti ad un NPC.
- I pokaiju trasportabili dal giocatore sono al massimo sei, i restanti catturati verranno trasferiti in un box da cui il giocatore potrà scambiarli tra box e team.
- In seguito ad una lotta i mostri devono essere in grado acquisire punti esperienza, salire di livello ed eventualmente evolversi.

## Requisiti non funzionali

- Il gioco dovrà essere fluido e compatibile con tutte le piattaforme supportanti Java 11

## 1.2 Analisi e modello del dominio

L'entità principale è il player, il quale contiene una squadra di sei mostri e ad esso è associata un deposito diviso in box (delle specie di magazzini) in cui può scambiare i mostri dalla squadra al box e viceversa. I mostri cambiano in base alla specie e ne possono esistere di diversi tipi per una stessa specie. Ogni mostro ha una lista di mosse che può utilizzare in battaglia. Le battaglie saranno tra player e un allenatore o un mostro selvatico. Durante le battaglie il player può utilizzare degli item in suo possesso per curare i suoi mostri. Il player deve essere libero di muoversi lungo la mappa a meno che non siano presenti ostacoli oppure altre entità. Il giocatore dovrà essere in grado di interagire con gli Npc.

Gli elementi costitutivi del problema sono sintetizzati in fig. 1.1

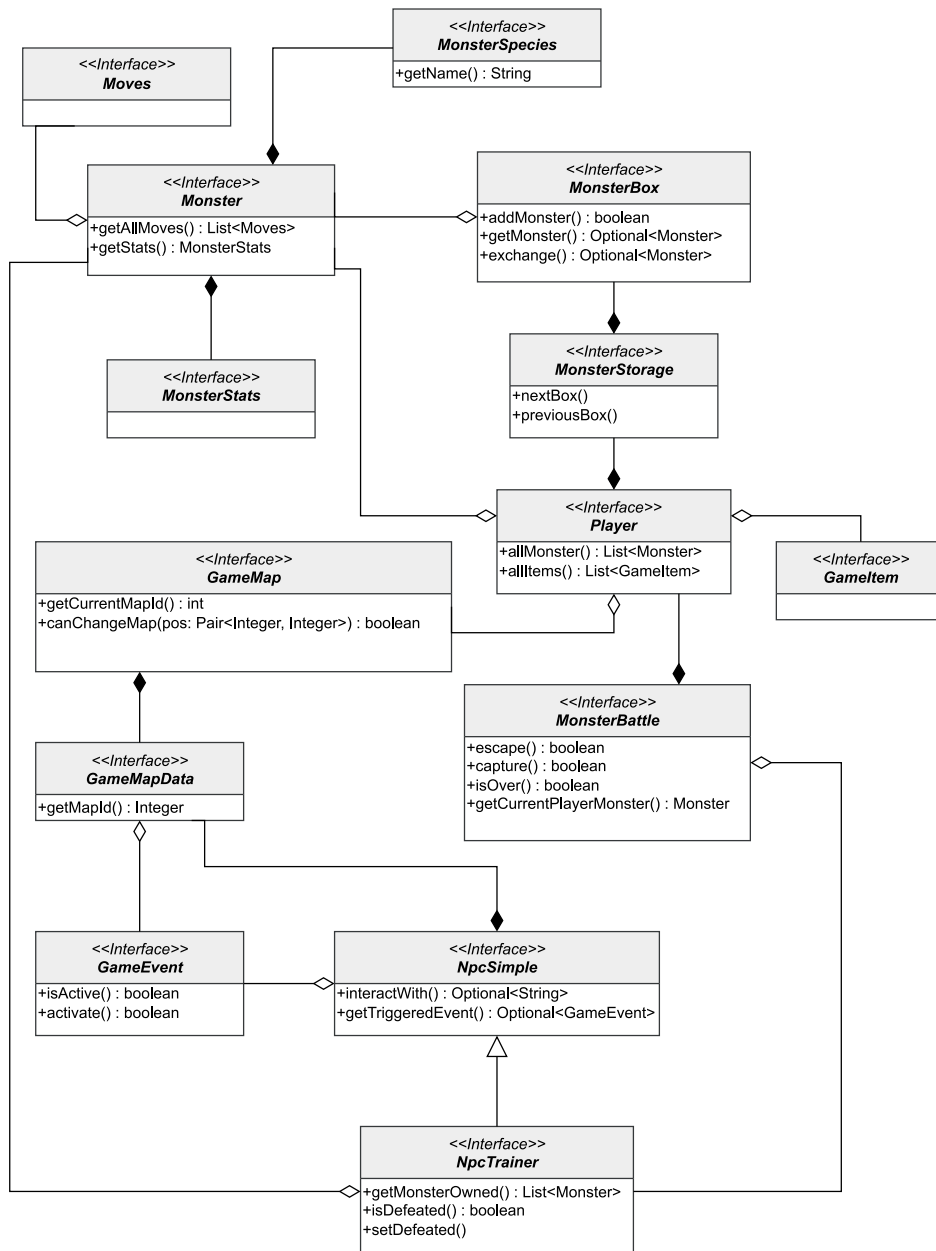


Figura 1.1: Schema UML dell'analisi del problema, con rappresentate le entità principali ed i rapporti fra loro

# Capitolo 2

## Design

### 2.1 Architettura

Per questo progetto è stato scelto il pattern architetturale MVC (model - view - controller) per far sì che la parte di logica (model) sia indipendente dalla parte di view e queste due parti possano comunicare attraverso gli appositi controller. La base del model è l'interfaccia Player attraverso la quale si può interagire con gli NPC oppure fare muovere il proprio sprite nella mappa. Il DataController si occupa della creazione dei dati, ovvero istanzia tutti gli elementi del dominio (incluso il player). Il PlayerController sfrutta i dati contenuti nel DataController e fornisce le informazioni necessarie alla view per poterle mostrare. Il GameFrame è la view che si occupa di mostrare i dati ottenuti dal controller e comandargli le azioni che il player deve intraprendere. Tramite le azioni del Player, il PlayerController ottiene tutti i dati riguardanti le lotte e gli avversari e fornisce un BattleController per la gestione della battaglia. A quel punto la view si aggiorna per rappresentare l'informazione della battaglia.

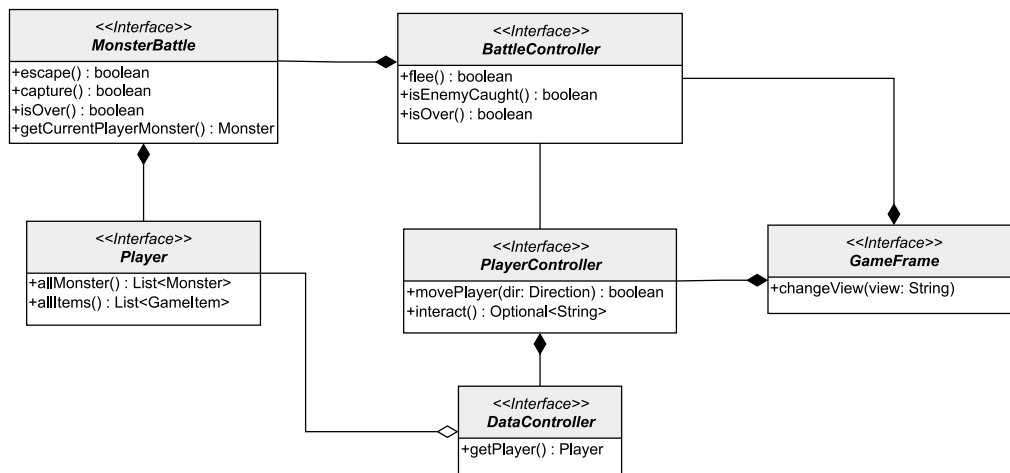


Figura 2.1: Schema UML dell'analisi del problema, con rappresentate le entità principali ed i rapporti fra loro

## 2.2 Design dettagliato

Luca Barattini

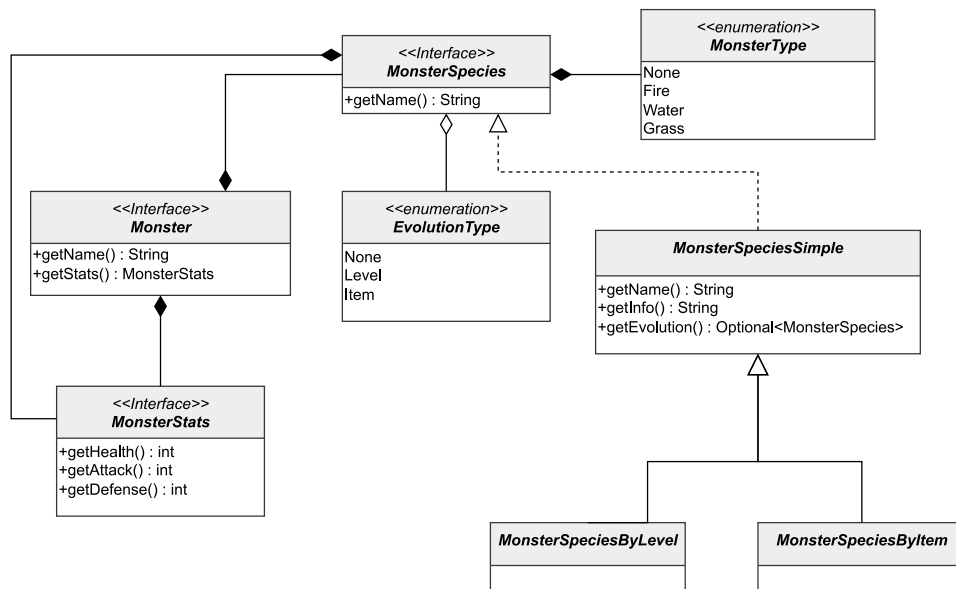


Figura 2.2: Schema UML della parte di model del Monster, con rappresentate le entità principali ed i rapporti fra loro



**Problema** Creazione della parte di model riguardante un Monster, le sue statistiche, la sua evoluzione ed il tipo di evoluzione che può essere tramite livello oppure GameItem.

**Soluzione** Visto che ci possono essere più mostri di una stessa specie ho deciso di scorporare la specie dal Monster. Le MonsterSpecies definiscono il metodo di evoluzione del Monster. La MonsterSpeciesSimple crea una specie che non può in alcun modo evolversi, MonsterSpeciesByLevel crea una specie che può evolversi solamente tramite aumento di livello ed infine MonsterSpeciesByItem crea una specie che si evolve solo tramite oggetto. È possibile, attraverso questa struttura, creare un mostro che si evolve prima tramite livello e successivamente tramite un oggetto. Tutte le varie MonsterSpecies sono create dal builder illustrato in fig. 2.4 I mostri hanno un'interfaccia a parte che gestisce le loro statistiche, ovvero MonsterStats, che permette di settare la vita ed altri parametri come l'attacco, la difesa e la velocità che verranno utilizzati solamente a livello di battaglia.

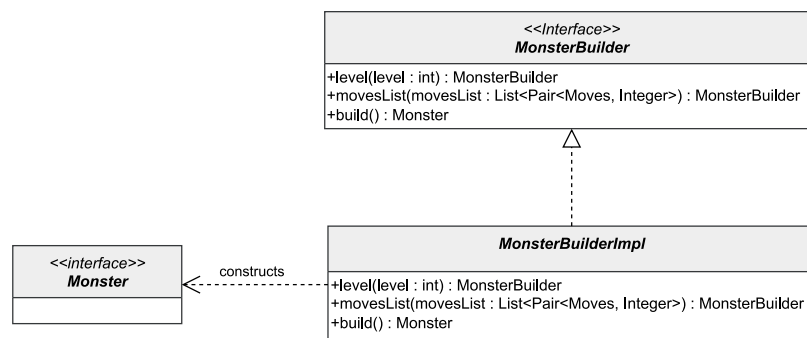


Figura 2.3: Schema UML del pattern builder per Monster, con rappresentate le entità principali ed i rapporti fra loro

**Problema** Creazione di un oggetto complesso con elevato numero di parametri.

**Soluzione** Per ovviare al problema della creazione di un oggetto complesso in modo semplice ho deciso di usare il pattern Builder che ci permette di creare un Monster facilmente step-by-step. L'utilizzo di questo pattern inoltre incrementa la leggibilità del codice, diminuisce i possibili errori tramite creazione con costruttore e separa la creazione del Monster e la sua implementazione. Il builder prende come parametri obbligatori la specie e la lista di mosse del mostro. È presente come parametro obbligatorio anche un

identificativo del mostro che viene gestito tramite lettura da file ed in modo incrementale. Se durante la creazione di un mostro il livello e l'esperienza (ovvero i punti che ti permettono di salire di livello) non sono settati allora vengono messi ad un valore di default; invece se non vengono specificate le statistiche allora verranno utilizzate quelle di base della specie.

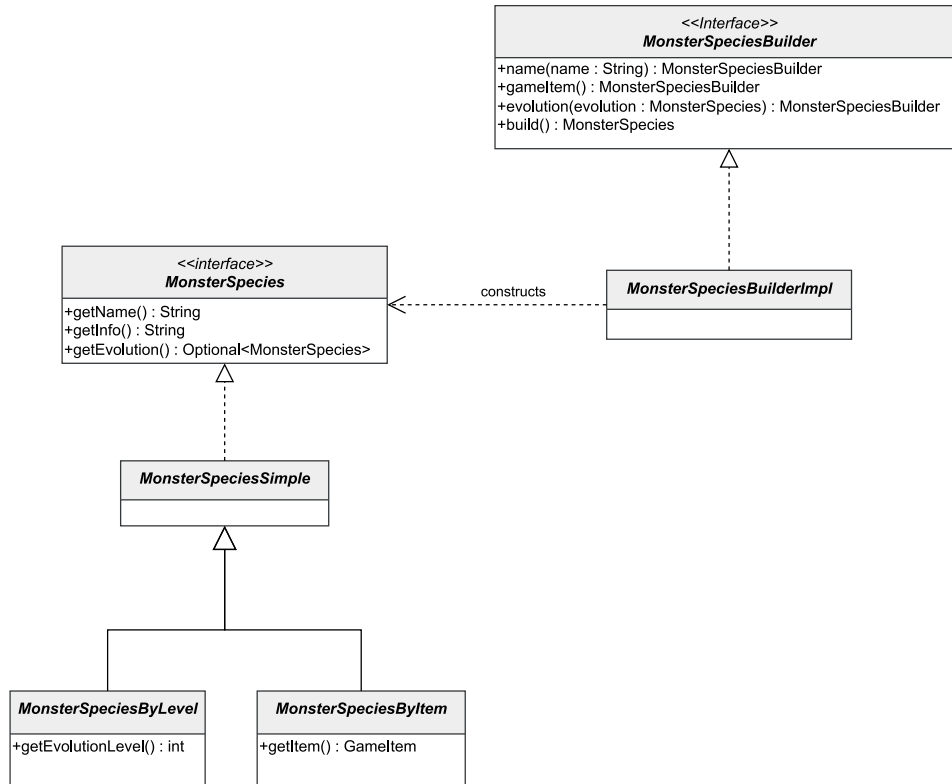


Figura 2.4: Schema UML del pattern builder per **MonsterSpecies**, con rappresentate le entità principali ed i rapporti fra loro

**Problema** Creazione di un oggetto di tipo **MonsterSpecies** che possa essere Simple, ByLevel o ByItem.

**Soluzione** Per risolvere il problema della creazione di una **MonsterSpecies** ho usato sempre il pattern Builder ma questa volta è strutturato in maniera lievemente più complessa. Visto che una **MonsterSpecies** poteva essere di più tipi, ovvero Simple, ByLevel e ByItem ho proceduto alla creazione di una builder che in base ai parametri passati (**evolutionLevel()**, **gameItem()**) controlla che tipo di **MonsterSpecies** deve creare.

## Samuele Carafassi

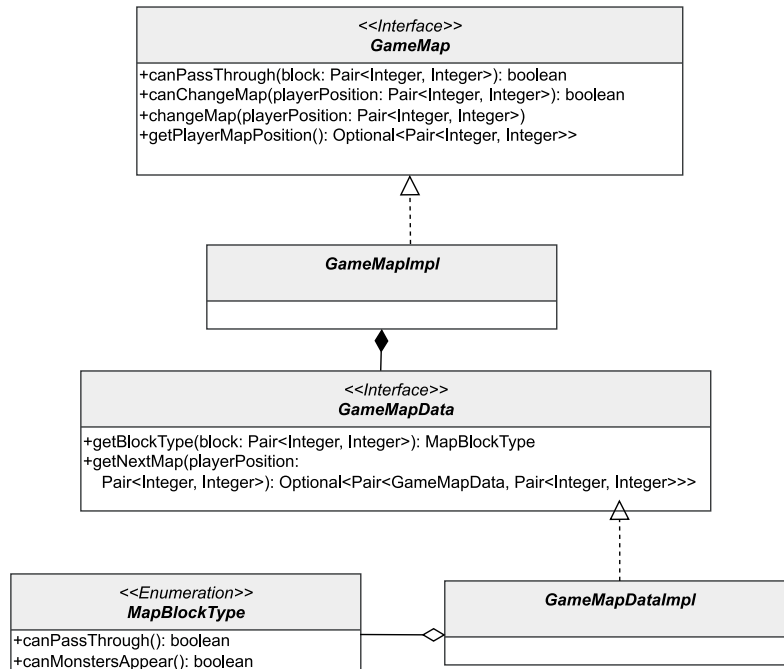


Figura 2.5: Schema UML della parte di model della mappa, con rappresentate le entità principali ed i rapporti fra loro

**Problema** Il giocatore deve essere libero di andare in giro per la mappa senza uscire dai limiti o camminare sugli ostacoli. Inoltre in presenza di collegamenti ad altre mappe deve essere riposizionato e deve essere aggiornata la mappa.

**Soluzione** Si è utilizzata l'interfaccia **GameMapData** per contenere i dati riguardanti una mappa, tra i quali: gli NPC, le specie di mostri selvatici che possono apparire, gli eventi che possono attivarsi in una certa posizione e le mappe ad essa collegata.

**GameMap** si occupa di elaborare i dati contenuti in **GameMapData** per capire dove il player può andare o quando deve cambiare mappa. Al cambio di mappa, **GameMap** sostituisce la sua istanza di **GameMapData** con un'altra **GameMapData** e fornisce al giocatore la posizione nella nuova mappa. Si è preferito non inserire un player nella **GameMap** per non creare una dipendenza tra **GameMap** e **Player**, in modo da poter interrogare la **GameMap** senza avere un player che si muove a disposizione.

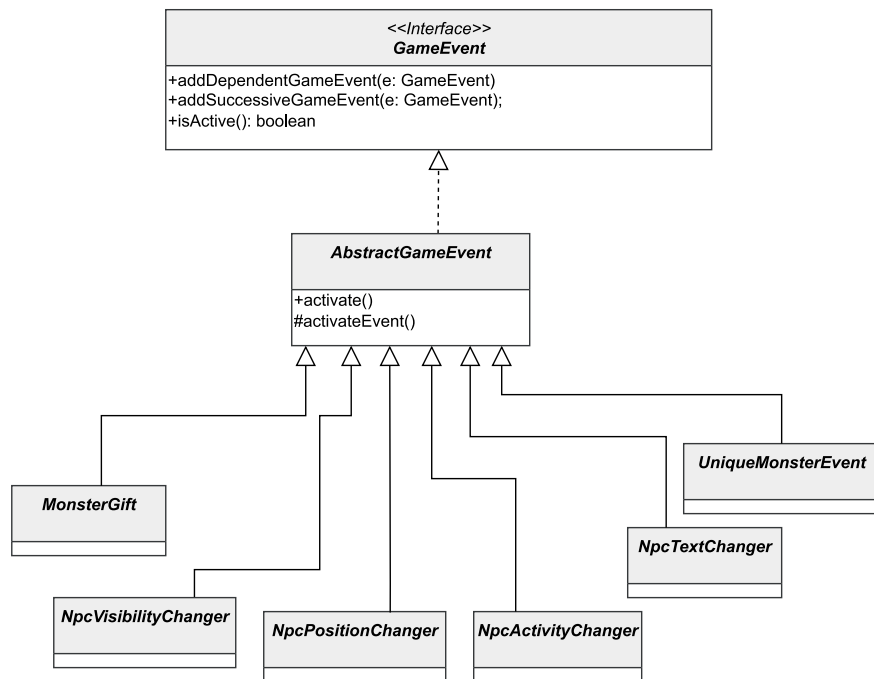


Figura 2.6: Schema UML dell'applicazione del pattern Method Template agli eventi di gioco

**Problema** Nel gioco sono presenti diversi tipi di eventi che possono essere attivi o inattivi e possono attivare o disattivare gli altri eventi. Tutti gli eventi differiscono per la funzione che devono svolgere (ad esempio: cambiare la posizione di un Npc o regalare un Pokaiju al giocatore). In fase di progettazione ci si è accorti che creare classi separate avrebbe portato a ripetizione di codice.

**Soluzione** Visto che gli eventi cambiano solo per il tipo di azione che devono svolgere, si è utilizzato il *pattern template method*. Il metodo template è **activate()** che chiama il metodo abstract e protected **activateEvent()**. Questo pattern ha minimizzato la ripetizione di codice permettendo di creare tante classi, come da fig. 2.6, riusando molto del codice già prodotto.

## Andrea Castorina

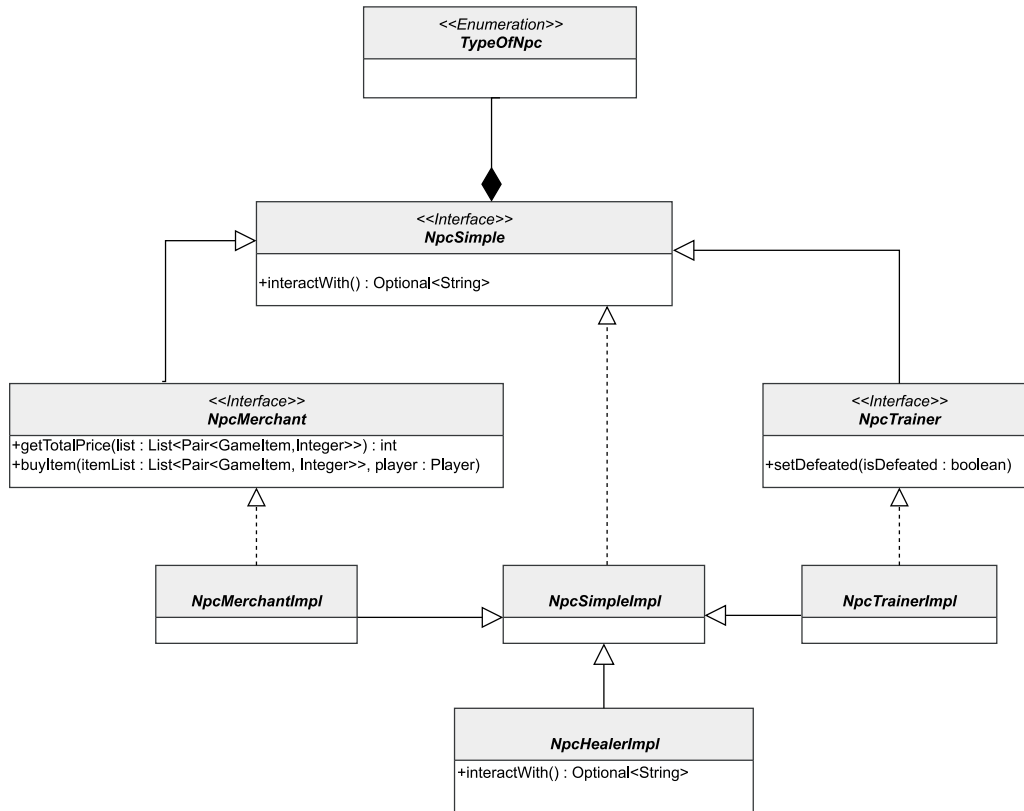


Figura 2.7: Schema UML della parte di model degli npc, con rappresentate le entità principali ed i rapporti fra loro

**Problema** Realizzazione della parte di model riguardante gli NPC, gestione delle interazioni con gli eventi di gioco e delle diverse funzionalità per ogni tipo di NPC.

**Soluzione** Di base ogni tipo di NPC presenta simili caratteristiche ed a differire sono le funzionalità che ne definiscono il tipo, ad eccezione del SimpleNPC che non presenta alcuna peculiarità (versione base di ogni tipo di NPC). SimpleNPC delinea i dati e le funzioni di base di un npc. gestisce l'interazione con il player attraverso **interactWith()**. Questa funzione esegue gli eventi attivi che scaturiscono dall'interazione del player con npc, eventualmente restituendo la frase correlata all'evento. Tutti i tipi di npc estendono NpcSimpleImpl così da riutilizzarne il codice. NpcTrainerImpl e NpcMerchantImpl implementano le rispettive interfacce (NpcTrainer, NpcMerchant)

assumendone le funzioni. NpcHealer sovrascrivendo **interactWith()** si occupa di ristabilire al massimo la vita dei mostri del player ed i PP delle relative mosse.

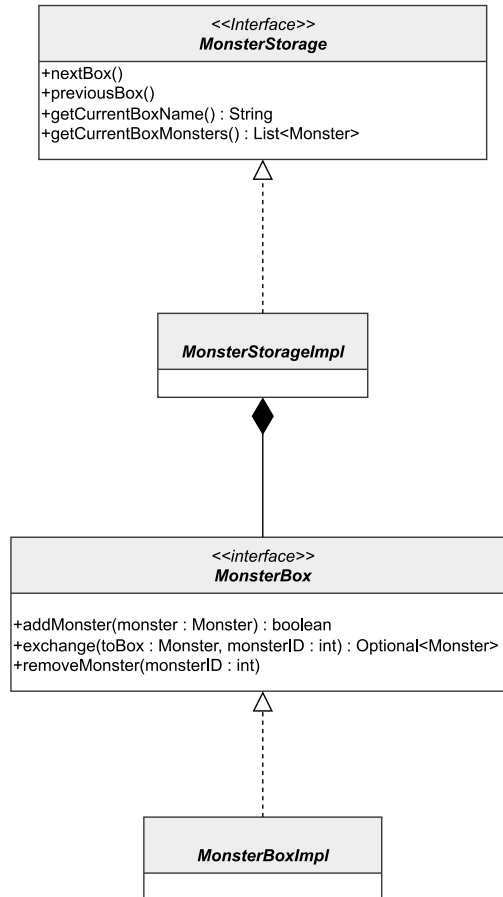


Figura 2.8: Schema UML della parte di model del monster storage e delle monster box

**Problema** Creazione di uno spazio dove poter depositare i mostri non contenibili in squadra e gestione delle funzionalità applicabili.

**Soluzione** Il player ha a sua disposizione uno storage, contenente al suo interno dei box con capacità di contenimento dei mostri limitata. Uno storage ha al suo interno un indice del box corrente e una lista di box, ognuno ha al suo interno un campo che ne indica il nome e la propria lista di mostri. In uno storage, il Player può depositare un mostro, effettuare uno scambio ed aggiungerne uno nel proprio team. Tali funzioni sono implementate in

MonsterBox così rendendone MonsterStorage indipendente dalla struttura interna.

**Jia Hao Guo**

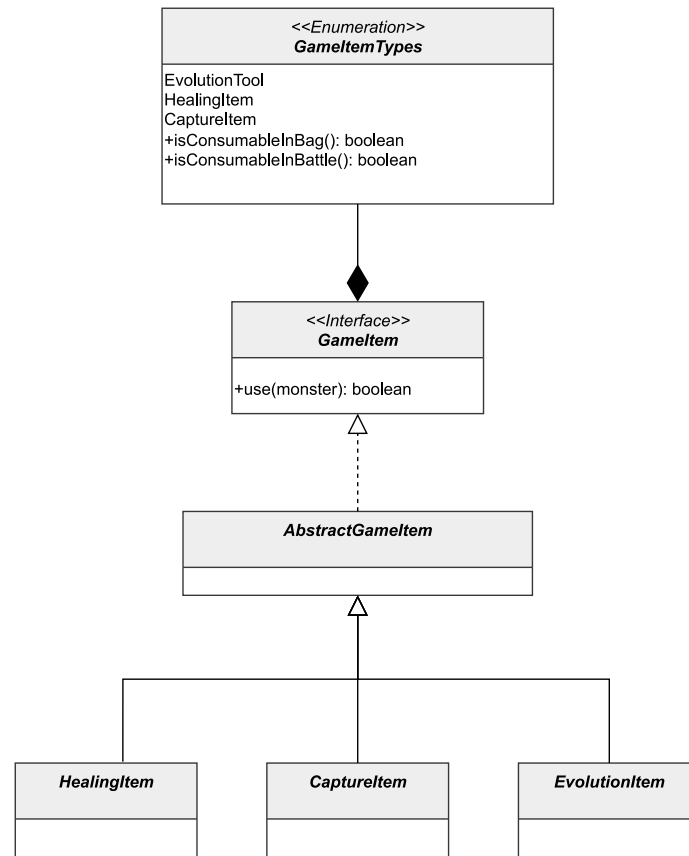


Figura 2.9: Schema UML dell'applicazione del pattern Method Template agli oggetti di gioco

**Problema** Gestire diversi tipi di oggetti di gioco che svolgono funzioni in base al loro tipo ed evitare la ridondanza di codice.

**Soluzione** Si è utilizzato il *pattern template method* per ridurre la ripetizione di codice su funzioni comuni. La funzione **public abstract use()** è la funzione incaricata di svolgere il compito dell'oggetto di gioco, che in quanto diversa in base al tipo di oggetto deve essere implementata dalle classi che estendono la classe astratta.

Inoltre ciascuna classe è legata ad un `GameItemTypes`, che rappresenta la

sua categoria. In questo modo è possibile agevolare la comprensione di quando è possibile usare un oggetto (in battaglia o fuori) e intuire qual è la sua funzione.

## Michael Pierantoni

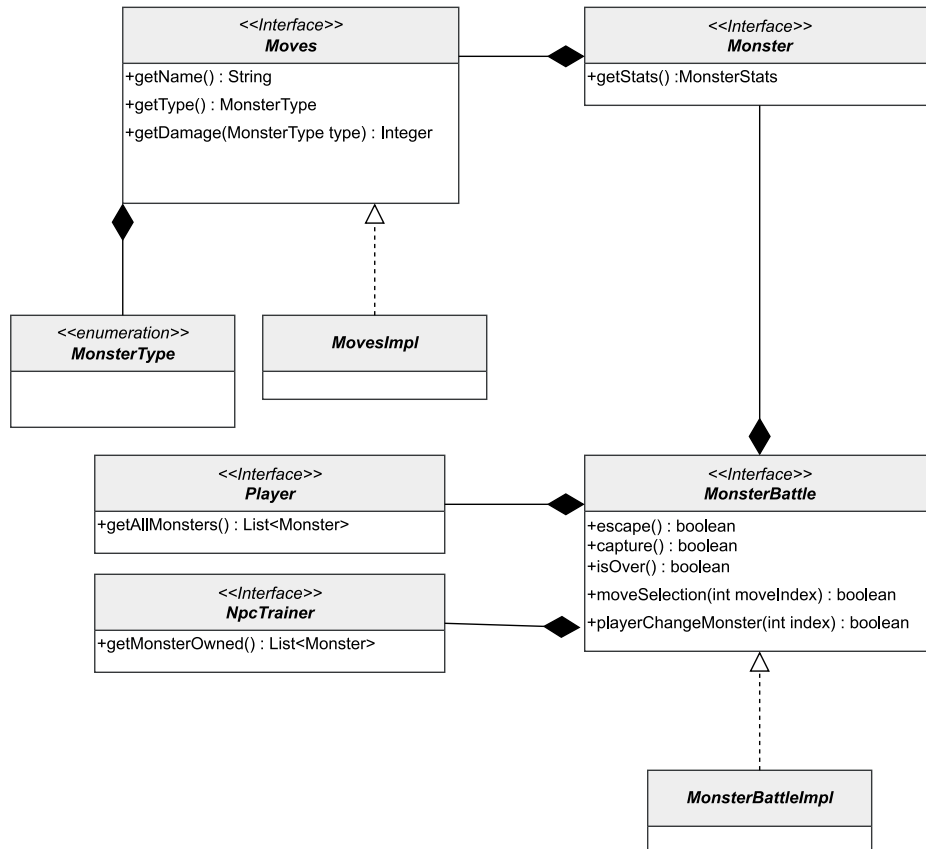


Figura 2.10: Schema UML della parte di model della battaglia, con rappresentate le entità principali ed i rapporti fra loro

**Problema** Realizzazione della parte di model riguardante la battaglia includendo nella progettazione la possibilità di un'input di un soggetto esterno (esempio: giocatore) che scatena determinate conseguenze nello scontro. Infine calcolare l'esito della battaglia

**Soluzione** Durante il proprio turno, il giocatore ha a disposizione varie possibilità d'azione tra cui: l'attacco, sostituire il Pokaiju in campo con un altro del proprio team, utilizzo di un Item e scappare; tutte queste azioni



vengono realizzate attraverso delle funzioni che all'interno contengono altre funzioni (evitando ridondanza): l'attacco è gestito tramite la funzione **public boolean movesSelection(int moveIndex)** che controlla se è possibile effettuare la mossa e procede con il turno. Ogni Monster ha a disposizione fino a quattro Move e può utilizzare ogni mossa per un numero limitato di volte (PP), nel caso in cui un Monster abbia finito i PP di ogni sua Move, esso avrà sempre a disposizione una mossa extra che potrà usare in modo illimitato, questa scelta di progettazione deriva dal voler evitare una situazione di stallo dove sia il Monster del giocatore che l'avversario abbiano finito le Move utilizzabili e non possano più fare nulla. Indipendentemente dalla tipologia d'avversario, i due esiti principali di una battaglia sono: la vittoria del player (nel caso dell'allenatore quest'esito porterà all'impossibilità di combatterle quest'avversario un'altra volta), dove si guadagnerà una somma di denaro e il Monster in campo guadagnerà una quantità di punti esperienza variabile; nel caso della battaglia contro un Monster selvaggio, la battaglia potrà avere anche altri due esiti: cattura, dove usando un oggetto di cattura si potrà provare a catturare l'avversario (possibilità casuale) ed aggiungerlo alla propria squadra (se la propria squadra contiene già sei Pokaiju, il nuovo verrà aggiunto al proprio box) e la fuga dove si potrà fermare la battaglia senza nessuna ripercussione.

# Capitolo 3

## Sviluppo

### 3.1 Testing automatizzato

Sfruttando JUnit sono stati realizzati i seguenti test:

- TestMonster: testing della creazione di un mostro, il leveling e l'evoluzione
- TestBattle: testing dello svolgimento di una battaglia contro un mostro selvaggio o un NPC avversario e i veri esiti della battaglia (vittoria, cattura, fuga)
- GameEventsTest: testing eventi della battaglia con un mostro unico (esempio: mostro leggendario), eventi che cambiano lo stato di un npc, scelta dello starter (ovvero si sceglie un mostro fra i tre possibili Pokaiju e quelli non selezionati spariscono, mentre quello selezionato viene aggiunto al giocatore come primo Pokaiju)
- MonsterStorageTest: testing della creazione di un monster storage, dei relativi box, navigazioni fra box e funzioni di aggiunta, scambio e deposito mostro
- NpcTest: testing della creazione di npc, l'interazione con il player, il corretto funzionamento di npcTrainer dopo una battaglia, la cura dei mostri per NpcHealer ed acquisti di item per npcMerchant
- PlayerTest: testing della creazione del player, il corretto funzionamento dell'aggiunta ed uso di GameItem, il movimento nella mappa e la lista di Monster del giocatore.

## 3.2 Metodologia di lavoro

Per l'esecuzione di questo progetto è stato creato un repository su github con il branch master (principale) ed un branch per ogni componente del gruppo in cui poter lavorare in totale autonomia; per condividere le proprie modifiche si procedeva ad un merge dal proprio branch verso master e per ottenerle invece ad un pull dal repository remoto.

La suddivisione dei lavori è stata gestita nel seguente modo:

### Luca Barattini

- Creazione di interfaccia Monster ed implementazione.
- Creazione di builder per Monster e MonsterSpecies, più le relative implementazioni.
- Creazione di MonsterSpeciesSimple, ByLevel e ByItem.
- Creazione dell'interfaccia MonsterStats ed implementazione.
- Creazione di enum EvolutionType.
- Creazione dell'interfaccia per BattleController e relativa implementazione.
- Creazione di alcune funzioni dentro a PlayerController.
- Creazione di test JUnit TestMonster per controllo di funzionamento del Monster e le evoluzioni.

### Samuele Carafassi

- Strutturazione mappa ed eventi di gioco a livello di model.
- Gestione a livello di view il movimento del personaggio e l'interazione con gli NPC.
- Gestione della view del mercante.
- Implementazione di qualche funzione nel PlayerController.
- Implementazione di qualche funzione in DataController e definito ordine di caricamento.
- Creazione di ImagesLoader per la bufferizzazione delle immagini.

- Creazione test JUnit per gli eventi di gioco.
- Creazione della possibilità di attivare un evento tramite un NPC (In NpcSimpleImpl aggiunti visibilità e possibilità di non interazione).

## **Andrea Castorina**

- Creazione dell'interfaccia NPC e implementazione.
- Creazione dell'interfaccia dei tipi di NPC e implementazione.
- Creazione dell'interfaccia MonsterBox e dell'implementazione.
- Creazione dell'interfaccia MonsterStorage e dell'implementazione.
- Creazione dell'interfaccia di PlayerController e implementazione di alcune funzioni.
- Creazione test Junit per gli NPC.
- Creazione test JUnit per monsterStorage e monsterBox.

## **Jia Hao Guo**

- Creazione dell'interfaccia Player e della sua implementazione a livello di model.
- Creazione di interfaccia GameItem ed implementazione
- Creazione di enum GameItemTypes.
- Gestione view del menu di gioco.
- Realizzazione ed istanziamento dei dati in DataController.
- Gestione in parte del GameFrame.
- Creazione view di Login e di New Game.
- Creazione test JUnit per il Player.

## Michael Pierantoni

- Creazione di interfaccia `MonsterBattle` ed implementazione.
- Creazione di interfaccia `Moves` ed implementazione.
- Creazione test JUnit per la battaglia.
- Creazione enum `MonsterType`.
- Gestione lettura da file delle weakness.
- Creazione view della battaglia.
- implementazione funzione `getMonster` in `ImageLoadersImpl`.

## 3.3 Note di sviluppo

### Luca Barattini

- Utilizzo di `Optional`, ad esempio per `MonsterSpecies` in cui la funzione per ottenere l'evoluzione del mostro aveva come tipo di ritorno un `Optional<MonsterSpecies>` poiché poteva anche non essere presente.
- Utilizzo in `BattleControllerImpl` di `Stream` e `Lambda Expressions` per operazioni su liste, come per esempio delle `filter`.

### Samuele Carafassi

- Utilizzo di `lambda expression` in `AbstractGameEvent` (funzione **`void activate()`**)
- Utilizzo di `Stream` in `DataControllerImpl` (funzione **`List<Move> getMovesByType(MonsterType type)`**)
- Utilizzo di `Optional` nelle mappe

In `ImagesLoaderImpl` la funzione per il ridimensionamento dell'immagine (`private resizeImage(BufferedImage, int, int)`) è stata presa da `StackOverflow`.

### Andrea Castorina

- Utilizzo di `Optional` per la gestione dei mostri nello storage.

## **Jia Hao Guo**

- Utilizzo di Lambda Expression, soprattutto per ActionListener dei pulsanti
- Utilizzo di Optional
- Utilizzo di Stream in DataController

In BoxPanel la funzione per attivare e disattivare tutti componenti contenuti nel pannello è stata presa da StackOverflow.

In GameFrameImpl l'impostazione del font è stato realizzato mediante UIManager preso da StackOverflow

## **Michael Pierantoni**

- Utilizzo di Lambda Expression e di Stream in MonsterBattleImpl per scorrere e filtrare delle liste
- Utilizzo di lambda expression livello di view per l'ActionListener dei pulsanti non statici

# Capitolo 4

## Commenti finali

### 4.1 Autovalutazione e lavori futuri

#### Luca Barattini

Penso che la mia parte di model sia ben svolta e facilmente manipolabile, ma il meccanismo di evoluzione è un po' troppo rigido poiché svolto con una enum.

Mi sarebbe piaciuto poter ampliare il progetto inserendo delle mosse del mostro imparabili tramite aumento di livello oppure tramite evoluzione e mosse dimenticabili in caso di set delle mosse pieno.

#### Samuele Carafassi

Ritengo che la parte di model che ho svolto sia abbastanza flessibile e riutilizzabile da terzi che abbiano in mente un progetto simile.

Non mi convince molto la parte di grafica non avendola approfondita molto nella mia esperienza da programmatore.

Per quanto riguarda il progetto nel suo complesso mi sembra un buon lavoro, le uniche cose che mi infastidiscono sono la mancanza di salvataggi e il non aver legato un database o un sistema di file al DataController, in maniera da rendere il progetto ancora più usufruibile da terzi. A tempo perso mi piacerebbe implementare quest'ultima parte e creare appositi programmi per l'inserimento dei dati anche da parte di non programmatori o programmatori poco esperti.

## **Andrea Castorina**

Personalmente mi ritengo soddisfatto della mia parte di model, a mio avviso il codice è scalabile ed efficiente. Mi sarebbe però piaciuto ampliare maggiormente il codice prodotto, aggiungendo ulteriori funzionalità agli NPC ed avere altri tipi; inoltre mi sarebbe piaciuto gestire i dati di gioco caricandoli da file. Complessivamente mi ritengo soddisfatto del progetto e del lavoro di gruppo svolto da tutti.

## **Jia Hao Guo**

Sono contento di essere d'aiuto nonostante le difficoltà avute durante lo svolgimento, e direi che sono più che soddisfatto per la progettazione e l'implementazione della parte model che ho realizzato dopo numerose ottimizzazioni e miglioramenti.

Invece per la parte view, non sono totalmente d'accordo di ciò che ho sviluppato, sia a lato tecnico che la chiarezza del codice, in quanto ho usato diverse scorciatoie e "imbrogli" per mantenere aggiornati i dati; sarebbe stato un piacere usare JavaFx rispetto a Swing per avere un funzionamento migliore, ma non avendo troppa dimestichezza, ho optato l'obsoleto.

Riassumendo tutto, sono molto soddisfatto ma alquanto sorpreso anche dall'ottimo contributo del team, della forte collaborazione e serietà tra i membri, in quanto nella fase di analisi e di progettazione iniziale non mi sarei mai aspettato di riuscire a compiere questo progetto corposo così come è venuto.

## **Michael Pierantoni**

Mi ritengo soddisfatto della mia sezione di model, credo d'aver scritto del codice efficiente, scalabile e che riesce a far interagire ogni elemento necessario senza creare ridondanza di codice. Non sono pienamente soddisfatto della parte di view a cui ho lavorato, perché non avendo molta esperienza in campo grafico, ho usato degli escamotage per aggiornare i vari JButton e le immagini non troppo eccellenti. Cercando d'avere una visione completa del progetto, mi ritengo molto soddisfatto del risultato finale e del livello di collaborazione che c'è stato nel team, questo mi porta ad avere un piccolo desiderio di portare avanti il progetto in futuro cercando di migliorarlo il più possibile.



# Appendice A

## Guida utente

In questo gioco non sono presenti salvataggi, tornando al menù principale i progressi verranno persi.

Appena cliccato "New Game" scegliere l'aspetto del personaggio e il nome e premere "Create" per creare la nuova partita (fig. A.1).

- "W" Movimento del giocatore verso l'alto.
- "A" Movimento del giocatore verso sinistra.
- "S" Movimento del giocatore verso il basso.
- "D" Movimento del giocatore verso destra.
- "Z" Interazione del giocatore con NPC nella mappa.
- "X" Menù giocatore da cui è possibile controllare la propria squadra, depositare i propri mostri nel box, consultare la propria borsa oggetti, uscire dal menù oppure tornare a quello principale. (fig. A.3)

Camminando nell'erba alta sarà possibile incontrare mostri selvatici, che possono essere combattuti o catturati tramite il menù battaglia (fig. A.5)

Ci sono svariati NPC con cui interagire che attivano diversi eventi ad esempio:

- in basso a destra nella mappa iniziale è possibile trovare un ostacolo invisibile, interagendo si renderà visibile un NPC
- a sinistra della casetta troviamo un allenatore, parlare con lui farà partire una battaglia

- dentro la casetta è presente un personaggio dalle sembianze femminili di nome "MOM" che cura la squadra del giocatore

Durante una battaglia è possibile attaccare l'avversario con il mostro corrente scegliendo una mossa, se il mostro che si usa ha esaurito le energie di tutte le mosse viene utilizzato un attacco corpo a corpo di default. È possibile mandare in campo un altro Pokaiju a patto che non sia quello già in campo e che non sia esausto. È possibile catturare Pokaiju selvatici, ma è vietato rubare i Pokaiju di altri allenatori. Una volta partita la sfida con un allenatore non è possibile tirarsi indietro. L'esito dello scontro deve essere una sconfitta o una vittoria. Finita una battaglia basta attendere qualche secondo per essere riportati alla mappa di gioco. Nella prima mappa è presente una casella speciale che farà scaturire una battaglia con un mostro che si può affrontare un'unica volta a prescindere dall'esito dello scontro.

Tramite il menù consultabile tramite il tasto "X" è possibile, nella sezione "Box", depositare o riprendere i propri mostri. Nel caso in cui i mostri in squadra siano già sei, tutti i mostri catturati successivamente verranno mandati automaticamente nel box. (fig. A.4) Per poter trasferire un Pokaiju dalla propria squadra al box bisogna cliccare sul mostro che si desidera depositare, poi cliccare sul pulsante "Deposit"; invece per metterlo in squadra bisogna cliccare sul mostro nel box e cliccare il pulsante "Take". In entrambi i casi se non ci sono posti per il mostro nel box o in squadra, i mostri non vengono spostati. È possibile unire le due operazioni selezionando prima il mostro in squadra, poi il mostro del box e cliccando il tasto "Exchange". Queste tre operazioni sono annullabili cliccando il tasto "Cancel".

BACK TO MENU

Insert name :

Select your gender :  ▼

Trainer number is generated randomly :

CREATE

Figura A.1: Menù della creazione di un nuovo personaggio

Player's balance: 60000\$				
Healing potion	This item heal a monster for 50 HP	200\$	0	
KracezStone	This item evolves krados	500\$	0	
Ultra healing potion	This item heal a monster for 400 HP	400\$	0	
Monster Ball	This item allows the player to capture a monster	100\$	0	
Super healing potion	This item heal a monster for 250 HP	300\$	0	
0\$		buy	exit	

Figura A.2: Schermata della vendita degli oggetti







MONSTER		BOX		BAG		PLAYERINFO		QUIT MENU		BACK TO MAIN MENU	
name : bibol Level : 100 Hp : 2531/2531				name : greyfish Level : 5 Hp : 212/212				name : yepicon Level : 1 Hp : 70/70			
											
STATS				STATS				STATS			
name : kratres Level : 10 Hp : 277/277				name : ponix Level : 50 Hp : 1300/1300				name : puppin Level : 10 Hp : 238/238			
											
STATS				STATS				STATS			

Figura A.3: Squadra del giocatore

<div>MONSTER</div> <div>BOX</div> <div>BAG</div> <div>PLAYERINFO</div> <div>QUIT MENU</div> <div>BACK TO MAIN MENU</div>				
Exchange		Deposit	Take	Cancel
BOX1		Team		
Name : kracez Lv : 7 Hp : 210/210	<input type="checkbox"/> 3024	Name : bibol Lv : 100 Hp : 2531/2531	<input type="checkbox"/> 3002	
Name : bibol Lv : 5 Hp : 149/149	<input type="checkbox"/> 3025	Name : greyfish Lv : 5 Hp : 212/212	<input type="checkbox"/> 3005	
Name : bibol Lv : 7 Hp : 196/196	<input type="checkbox"/> 3026		<input type="checkbox"/> 3007	
Name : greyfish Lv : 3 Hp : 170/170	<input type="checkbox"/> 3027	Name : yepicon Lv : 1 Hp : 70/70	<input type="checkbox"/> 3014	
		Name : krados Lv : 10 Hp : 277/277	<input type="checkbox"/> 3004	
		Name : ponix Lv : 50 Hp : 1300/1300	<input type="checkbox"/> 3003	
		Name : puppin Lv : 10 Hp : 238/238		
Previous box		Next box		

Figura A.4: Schermata dei box

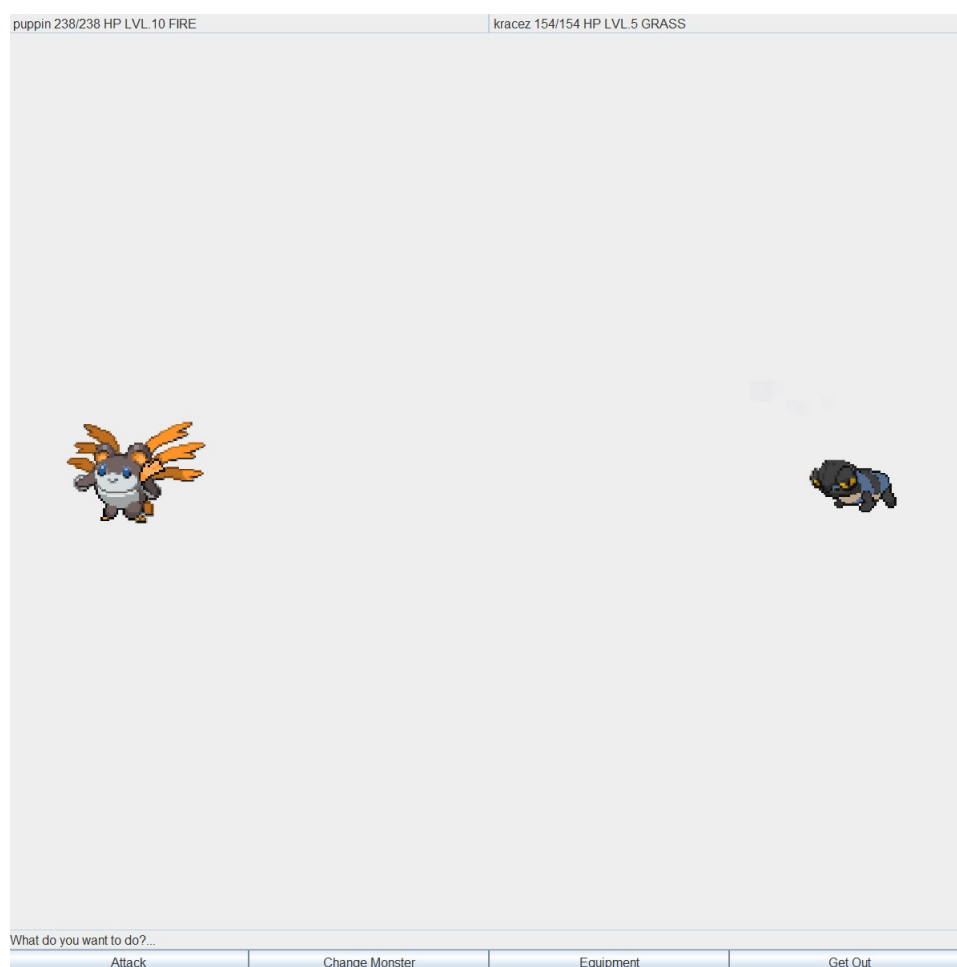


Figura A.5: Schermata della battaglia

# Appendice B

## Esercitazioni di laboratorio

### B.1 Link consegne

#### B.1.1 Luca Barattini

- Laboratorio 05: <https://virtuale.unibo.it/mod/forum/discuss.php?d=87881#p155101>
- Laboratorio 06: <https://virtuale.unibo.it/mod/forum/discuss.php?d=87880#p155103>
- Laboratorio 07: <https://virtuale.unibo.it/mod/forum/discuss.php?d=88829#p155104>
- Laboratorio 08: <https://virtuale.unibo.it/mod/forum/discuss.php?d=89272#p155105>
- Laboratorio 09: <https://virtuale.unibo.it/mod/forum/discuss.php?d=90125#p155107>
- Laboratorio 10: <https://virtuale.unibo.it/mod/forum/discuss.php?d=91128#p155150>

#### B.1.2 Samuele Carafassi

- Laboratorio 05: <https://virtuale.unibo.it/mod/forum/discuss.php?d=87881#p138869>
- Laboratorio 06: <https://virtuale.unibo.it/mod/forum/discuss.php?d=87880#p138889>



- Laboratorio 07: <https://virtuale.unibo.it/mod/forum/discuss.php?d=88829#p138892>
- Laboratorio 08: <https://virtuale.unibo.it/mod/forum/discuss.php?d=89272#p138923>
- Laboratorio 09: <https://virtuale.unibo.it/mod/forum/discuss.php?d=90125#p139285>
- Laboratorio 10: <https://virtuale.unibo.it/mod/forum/discuss.php?d=91128#p139701>

### **B.1.3 Andrea Castorina**

- Laboratorio 05: <https://virtuale.unibo.it/mod/forum/discuss.php?d=87881#p155748>
- Laboratorio 06: <https://virtuale.unibo.it/mod/forum/discuss.php?d=87880#p155749>
- Laboratorio 07: <https://virtuale.unibo.it/mod/forum/discuss.php?d=88829#p155750>
- Laboratorio 08: <https://virtuale.unibo.it/mod/forum/discuss.php?d=89272#p155751>
- Laboratorio 09: <https://virtuale.unibo.it/mod/forum/discuss.php?d=90125#p155752>

### **B.1.4 Jia Hao Guo**

- Laboratorio 05: <https://virtuale.unibo.it/mod/forum/discuss.php?d=87881#p155769>
- Laboratorio 06: <https://virtuale.unibo.it/mod/forum/discuss.php?d=87880#p155770>
- Laboratorio 07: <https://virtuale.unibo.it/mod/forum/discuss.php?d=88829#p155771>
- Laboratorio 08: <https://virtuale.unibo.it/mod/forum/discuss.php?d=89272#p155775>