Relazione
“Objectmon”

Jiaqi Xu, Weijie Fu, Azael Garcia Rufer, Jiekai Sun

17 febbraio 2024

Indice

Capitolo 1

Analisi

Il team si propone di sviluppare il videogioco Objectmon, ispirandosi alla serie di giochi RPG Pokemon. Nella nostra versione l'obiettivo principale è sconfiggere tutti gli avversari, distribuiti in un mondo bidimensionale, in battaglie a turni che coinvolgono squadre di Objectmon, creature dotate di caratteristiche ed abilità differenti che li rendono più forti o più deboli rispetto ad altri.

1.1 Requisiti

Il software è realizzato come progetto per il corso di Programmazione ad Oggetti 2023-24 dell'Università di Bologna.

Requisiti funzionali

- Il gioco dovrà essere suddiviso in due modalità intercambiabili:

  - Modalità esplorazione: il giocatore dovrà essere libero di muoversi in uno spazio bidimensionale ed interagire con l'ambiente ed NPC che svolgono varie funzioni (allenatori, infermieri, venditori).

  - Modalità battaglia: il giocatore e avversario dovranno sfidarsi in battaglie a turni sfruttando le abilità dei propri Objectmon.

- Ogni Objectmon dovrà essere caratterizzato da statistiche (es. punti vita, attacco, difesa e velocità), tipo elementale (fuoco, acqua, erba, etc) e mosse che nel loro complesso determinano nel meglio o nel peggio l'esito della battaglia.

- Dovrà essere possibile migliorare il proprio gruppo di Objectmon attraverso level-up oppure catturandone in stato "selvatico". Per imbattersi in Objectmon selvatici dovrà necessario esplorare zone di "erba alta", riconoscibili visibilmente.

- Dovranno essere presenti strumenti per rigenerare i punti vita dei propri Objectmon e strumenti per catturare Objectmons selvatici.
  Tali strumenti, di varia efficacia dovranno essere comprati presso specifici NPC venditori.

Requisiti non funzionali

- La risoluzione video del gioco dovrà essere scalabile in modo tale da non alterare eccessivamente l'esperienza di gioco.

- Il gioco dovrà eseguito con performance accettabili su un macchina entry level del 2019 (es. Athlon 3000g / Intel i5-8250u (2 core/4 threads), 8gb RAM, Integrated Graphics).

- Il gioco dovrà essere compatibile su piattaforme Windows, Mac e Linux.

1.2 Analisi e modello del dominio

L'applicazione dovra permettere al giocatore di esplorare un mondo bidimensionale popolato da personaggi non giocanti (*NPC*) e creature chiamate *Objectmon*.

Gli *Objectmon* possiedono una serie di attributi unici che li distinguono l'un dall'altro. Questi attributi includono statistiche (*Stats*) quali punti vita, attacco, difesa, velocità, tipi elementali (*Aspect*) e un set di mosse utilizzabili (*Move*).

Ogni *Objectmon* è associato a uno o più *Aspect*, che influenzano le sue performance durante gli scontri.
Ad esempio, un *Objectmon* di *Aspect* Fuoco potrebbe essere debole contro attacchi di *Aspect* Acqua ma resistente verso l'*Aspect* Erba.

Le battaglie nel gioco sono a turni e offrono al giocatore una serie di opzioni tattiche. Queste includono l'utilizzo di *mosse d'attacco*, l'utilizzo di *strumenti* nell'inventario, lo *switch* dell'Objectmon e la *fuga* dalla battaglia. Il danno inflitto ad un Objectmon è influenzato da vari fattori, come le *Stat* degli attaccanti e dei difensori, l'efficacia (*Potency*), la *Base Power* delle mosse e gli *Aspect* degli Objectmon coinvolti.

Il giocatore può interagire con i *Vendor NPC* per acquistare strumenti utili alla progressione del gioco. Tali *Items* si suddividono in due categorie: *Heal* e *ObjectBall*. I primi rigenerano i punti vita di un *Objectmon* mentre le *ObjectBall* permettono di catturare *Objectmon* di stato selvatico (*Wild*).
Questi oggetti possono essere acquistati utilizzando *crediti* ottenuti dalle vittorie.

Il giocatore può catturare nuovi *Objectmon* in zone *Tiles* infestate e migliorare la propria squadra attraverso level-up, che causano un incremento delle *Stats* totali.

L'obiettivo finale è sconfiggere tutti gli NPC "allenatori", detti *Trainers*, sapendo che venir sconfitti anche una sola volta implica la fine del gioco.

Gli elementi costitutivi sono sintetizzati in Figura 1.1.

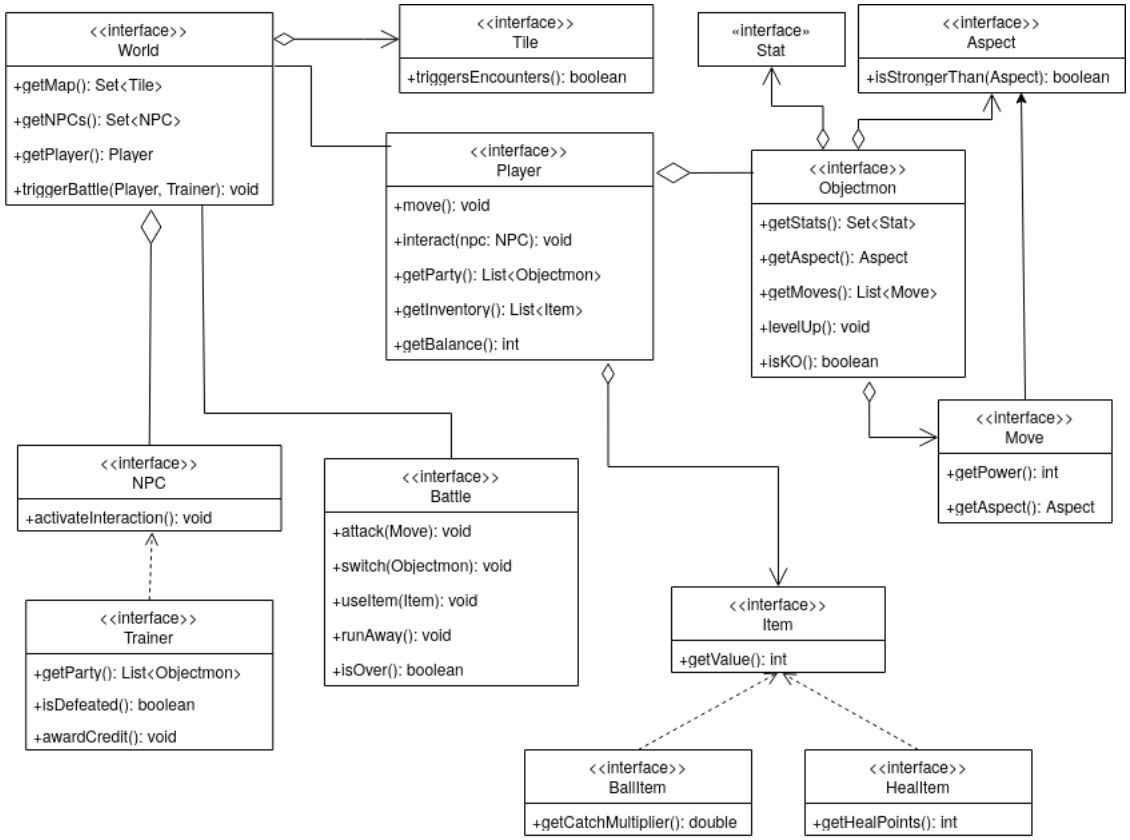


Figura 1.1: Schema UML dell'analisi del problema, con rappresentate le entità principali ed i rapporti fra loro

Capitolo 2

Design

2.1 Architettura

L'architettura dell'applicazione segue il pattern architetturale MVC.

Il Model della nostra applicazione, una volta inizializzato dal Controller, rende disponibili i Managers che definiscono la business logic del gioco. Questi Managers gestiscono vari aspetti del gioco, come le battaglie, le transazioni e le interazioni con il mondo di gioco.

Il Controller agisce da intermediario tra Model e View, più nello specifico limita l'accesso al Model all'esterno esponendo solo istanze read-only. Ciò significa che gli altri componenti dell'applicazione, come la View, possono accedere solo a una rappresentazione non modificabile dei dati nel Model, garantendo un maggiore controllo sull'integrità dei dati.

Per alterare lo stato interno del Model in modo controllato, è necessario inviare comandi predefiniti al Controller.

Questo approccio alla gestione del flusso di dati assicura una corretta separazione tra Model e View e facilita la manutenzione e l'estensione del codice.

Rispettando questi vincoli, è possibile sostituire in modo semplice la View, ad esempio passando da un'implementazione Swing a JavaFX.

Gli elementi architetturali sono sintetizzati in Figura 2.1.

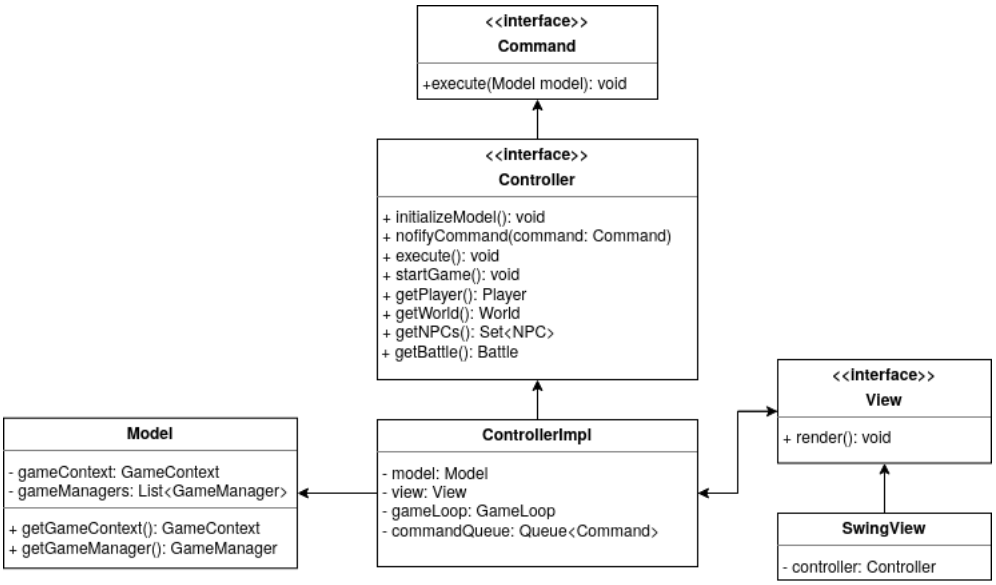
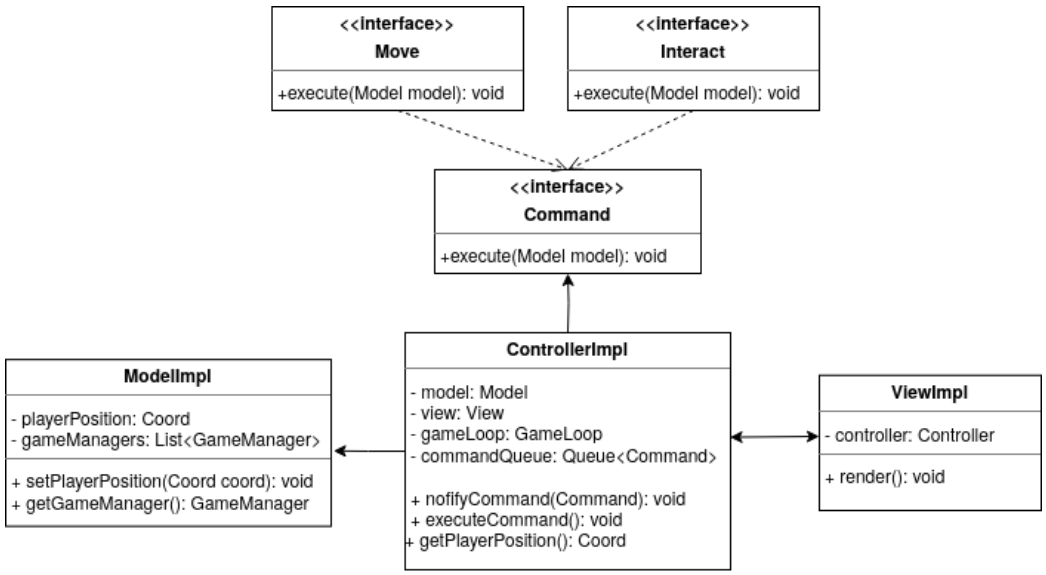


Figura 2.1: Architettura MVC

2.2 Design dettagliato

Jiekai Sun

Gestione degli input provenienti dall View



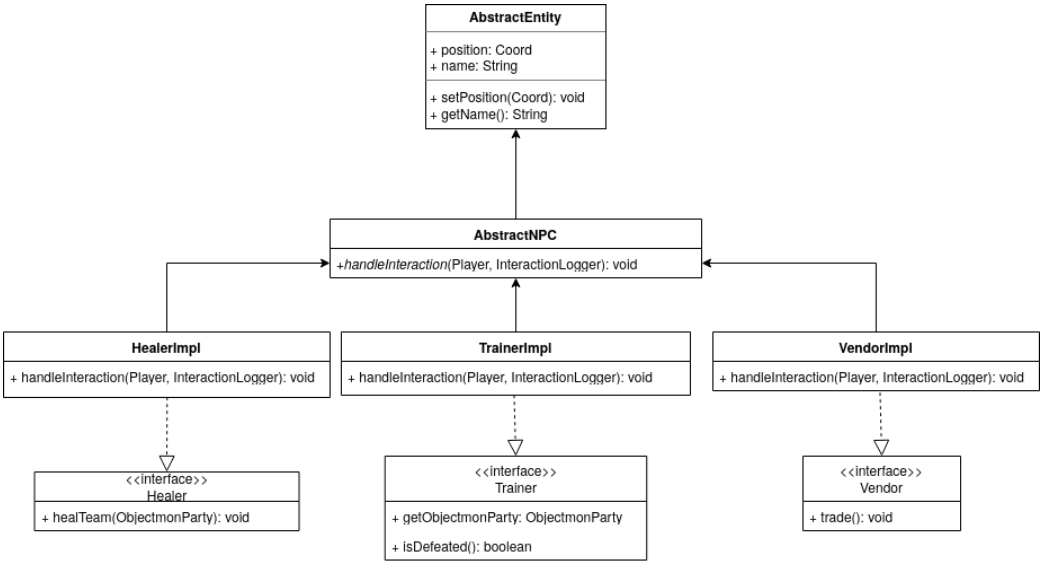
Problema: Nelle fasi iniziali di sviluppo mi ero reso conto che c'era bisogno di fornire alla View un modo per effettuare operazioni sul Model come ad esempio muovere il Player alla pressione di alcuni tasti.
Inizialmente effettuavo un accesso diretto al Player via Model, ma questo approccio violava i principi dell'architettura MVC bypassando completamente il layer del Controller.

Soluzione: Ho risolto il problema utilizzando il Command pattern.
Sapendo che il controller non fornisce metodi per operare direttamente sul Model, ho creato un'interfaccia Command che astrae e incapsula l'azione da eseguire sul Model. In questo modo, ho anche raggiunto un buon livello di disaccoppiamento tra l'invoker e il receiver, consentendo una maggiore flessibilità del sistema, oltre a fornire una sistema di comandi utilizzabile anche da altri.

Grazie al Command pattern, se nella View viene rilevata la pressione di un tasto di movimento del Player posso chiedere al controller di eseguire il comando nel seguente modo: *controller.notifyCommand(new MoveLeft())*.

Dove MoveLeft() incapsula il cambio di posizione del Player reperito dal Model.

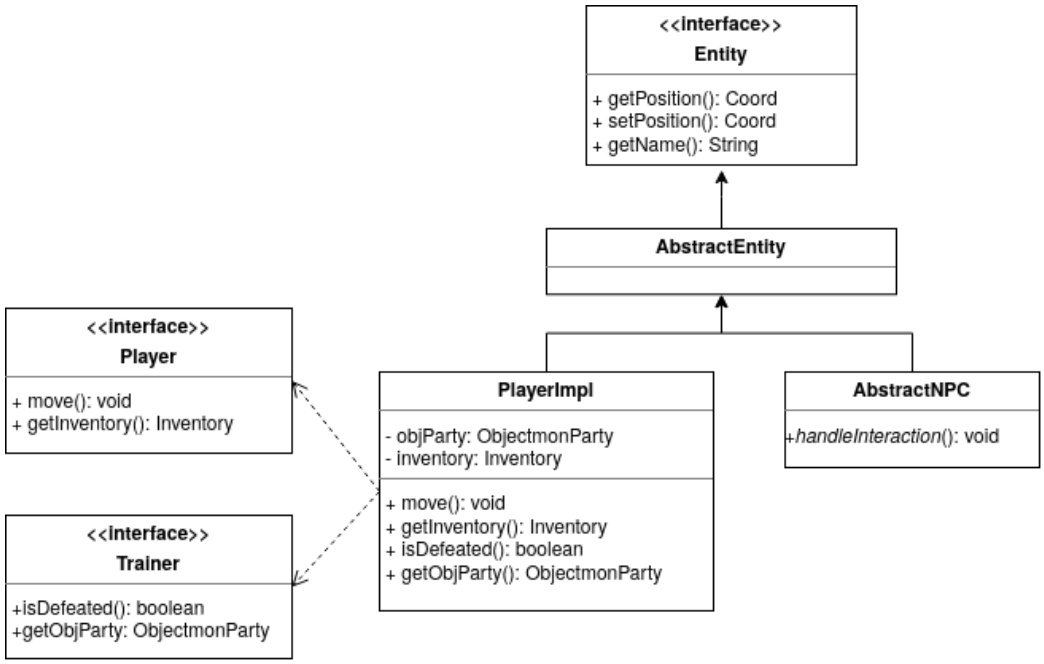
Riutilizzo del codice per la creazione di NPC diversi



Problema: Durante lo sviluppo del gioco si è considerata l'idea di introdurre diversi tipi di NPC, come Vendors, Sellers e Healers. Tutti questi condividono la caratteristica di rispondere alle interazioni del giocatore, ma l'esito specifico dell'interazione dipende da ciò che si intende implementare.

Soluzione: Per massimizzare il riutilizzo del codice esistente, ho deciso di usare handleInteraction() come Template Method. Questa strategia mi consente di implementare facilmente nuovi tipi di NPC con reazioni diverse, eventualmente integrando comportamenti provenienti da interfacce aggiuntive per ulteriori personalizzazioni.

Riutilizzo del codice comune al Player e agli NPC

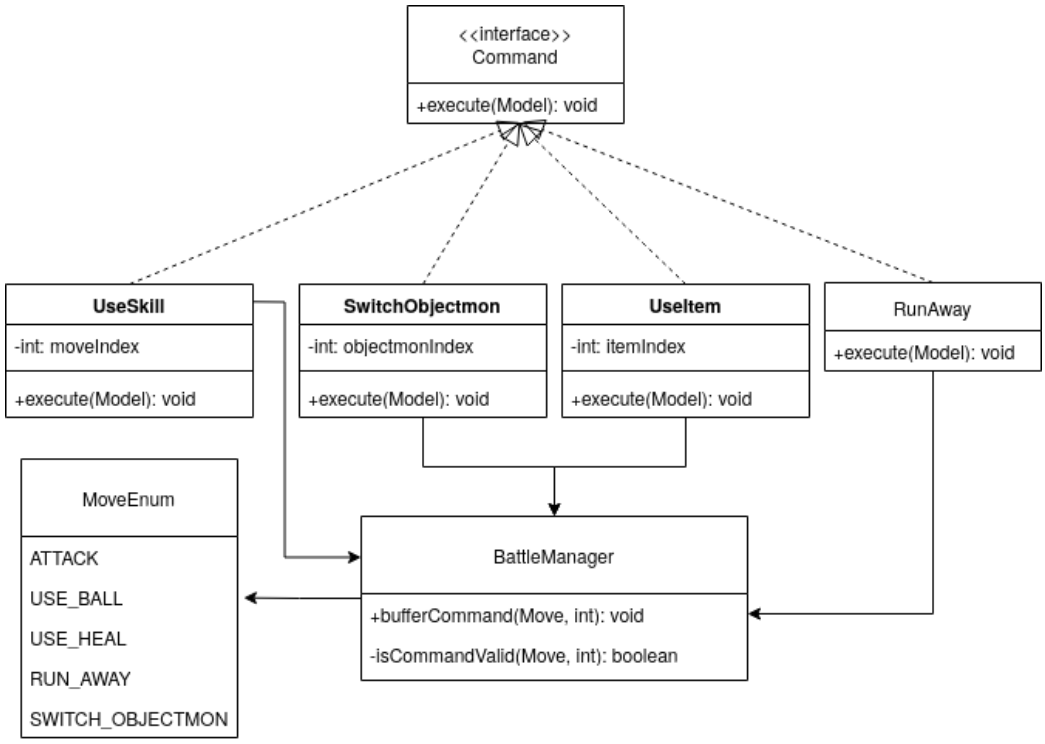


Problema: In fase di progettazione mi ero posto il problema di come strutturare gerarchicamente le entità nel gioco, cercando di riutilizzare più codice possibile.

Soluzione: Ho individuato nell'interfaccia Entity le funzionalità di base di che sarebbero il nome e la posizione corrente. Li ho in seguito implementati in una classe AbstractEntity che funge da bivio per l'implementazione del Player e degli NPC. In questo modo riesco a definire chiaramente che un Player oltre alla funzionalità base di un entità ha aspetti caratterizzanti quali la possibilità di sfidare altri, avere un inventario di oggetti e muoversi. Dall'altro lato invece abbiamo AbstractNPC che funge da base per NPC interagibili.

Weijie Fu

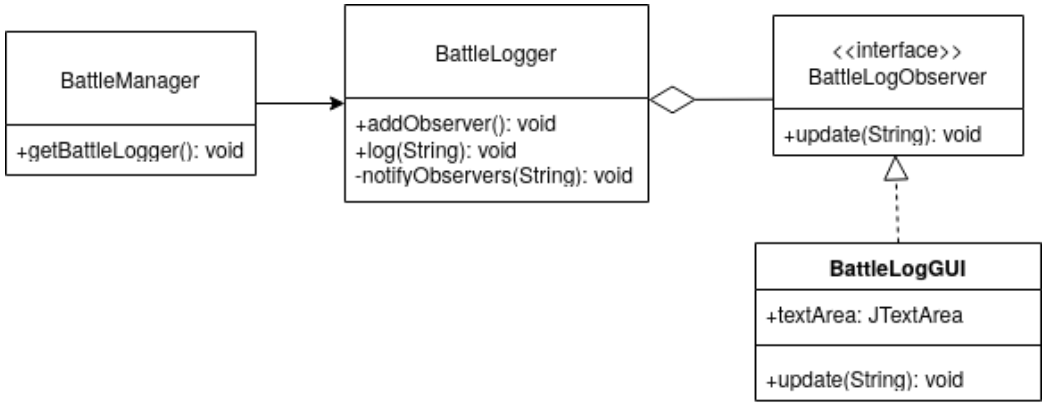
Passaggio delle mosse scelte del giocatore al BattleManager



Problema: Un problema che mi sono posto è stata la ricezione delle mosse del giocatore dalla GUI per quanto riguarda il BattleManager. Serviva un modo generalizzato che non facesse coupling con la controparte.

Soluzione: Utilizzando il Command Pattern possiamo creare dei comandi modellati in modo tale da bufferizzare la scelta della mossa nel BattleManager. In questo modo viene fornito un protocollo per la ricezione dei comandi, naturalmente vi sono comunque controlli fatti all'interno di BattleManager per validare la scelta.
Il comando in sè è strutturato in modo tale che venga reperito il BattleManager e a questo viene passato il MoveEnum corrispondente più l'indice di quel contesto.

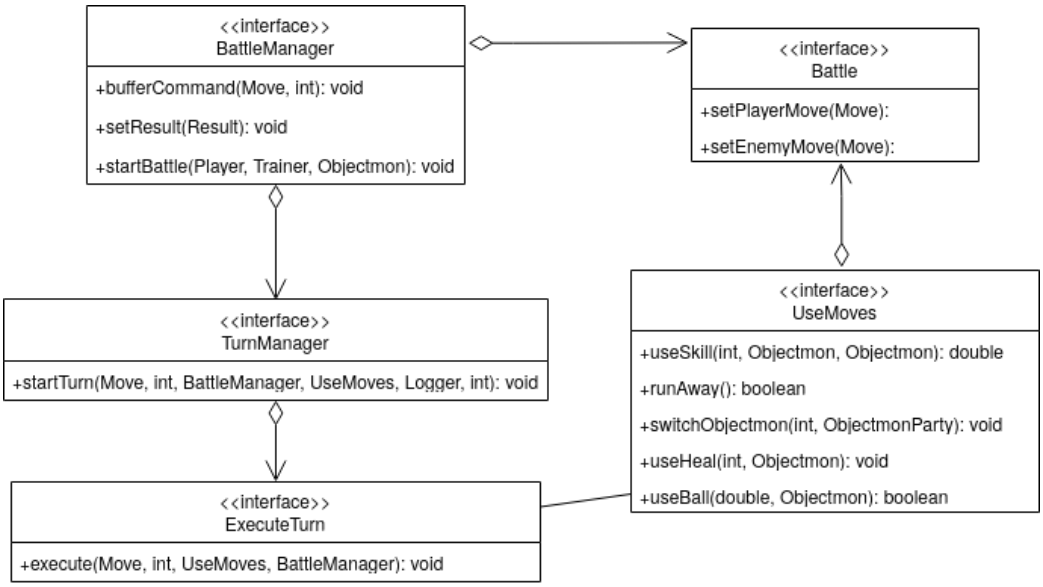
Mandare alla GUI l'output testuale dello svolgimento della battaglia



Problema: Avevo necessità di mandare alla BattleLogGUI le informazioni relative allo svolgimento della battaglia. In modalità esplorazione il reperimento delle informazioni è effettuato ad ogni refresh. Nel mio caso avevo a che fare con un una JTextArea che non veniva aggiornata attraverso paintComponent(Graphics g) di Swing.

Soluzione: Ho applicato l'Observer pattern e fatto in modo che appena la BattleLogGUI viene inizializzata questa si registri come Observer a BattleLog nel Model, il quale agisce da Subject esponenendo il metodo addObserver(Observer observer). In questo modo ogni qualvolta all'interno del BattleManager viene loggato un messaggio la GUI viene aggiornata subito dopo. Questa soluzione è inoltre inoltre applicabile a più View nel caso si voglia ottenere lo stesso output in posizioni differenti.

Architettura generale del BattleManager



Problema: Durante lo sviluppo della gestione delle battaglie, mi sono reso conto che molti compiti devono essere eseguiti all'interno di una singola classe che contiene tutte le informazioni relative alla battaglia. Questo approccio potrebbe portare a una situazione in cui la gestione della battaglia è affidata a una sola classe al fine di ridurre le dipendenze. Tuttavia, questa scelta può rendere il codice difficile da modificare e violare il principio di single responsibility (SRP).

Soluzione: Ho deciso di assegnare i compiti di gestione dei turni alla classe TurnManager e di consentire a UseMoves di incapsulare la logica della battaglia. Dopo che i turni sono stati gestiti dal TurnManager, viene chiamato UseMoves per eseguire le azioni effettive della battaglia.

Jiaqi Xu

Generazione di Objectmon e Skill

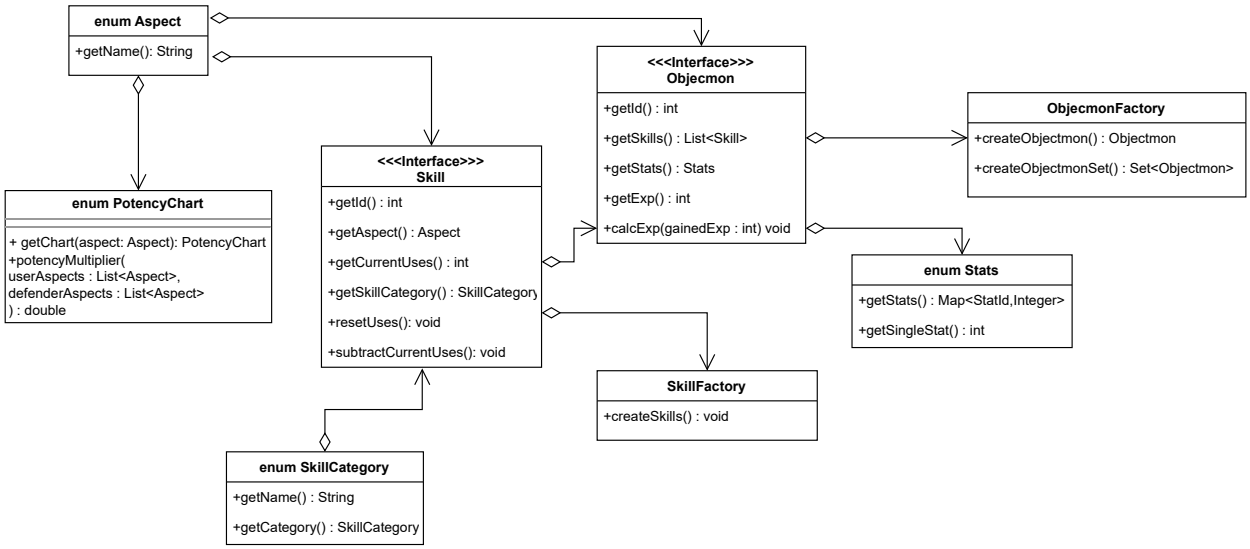


Figura 2.2: Rappresentazione UML di un Objectmon

Potency e PotencyChart

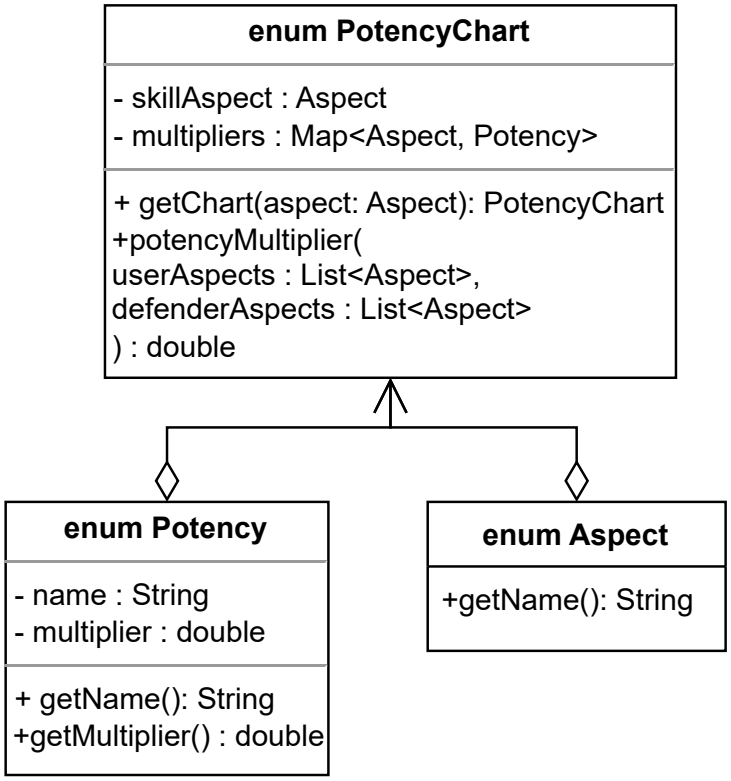


Figura 2.3: Rappresentazione UML dell'uso di Potency e PotencyChart

Problema: Quando un Objectmon utilizza una Skill, si deve calcolare il danno di quella Skill. Per il calcolo del danno devono essere controllati:

- gli Aspect del Objectmon che usa la Skill, per controllare se almeno uno condivide lo stesso Aspect per un moltiplicatore bonus;
- gli Aspect del Objectmon che è il target della Skill, per sapere qual'è il moltiplicatore del danno.

Sebbene il primo problema prima sia risolvibile, nel secondo è stato trovato difficoltoso sapere il moltiplicatore soltanto attraverso metodi.

Soluzione: Per risolvere entrambi i problemi, è stato implementato gii enum Potency e PotencyChart. PotencyChart contiene l'Aspect di una Skill e una Map di tutti gli Aspect (key) e il Potency (value) a loro associato. Attraverso l'utility method potencyMultiplier(), usando come paramentri gli Aspect dell'ObjectMon user e target, si ottiene il moltiplicatore.

Factory di Objectmon e Skill

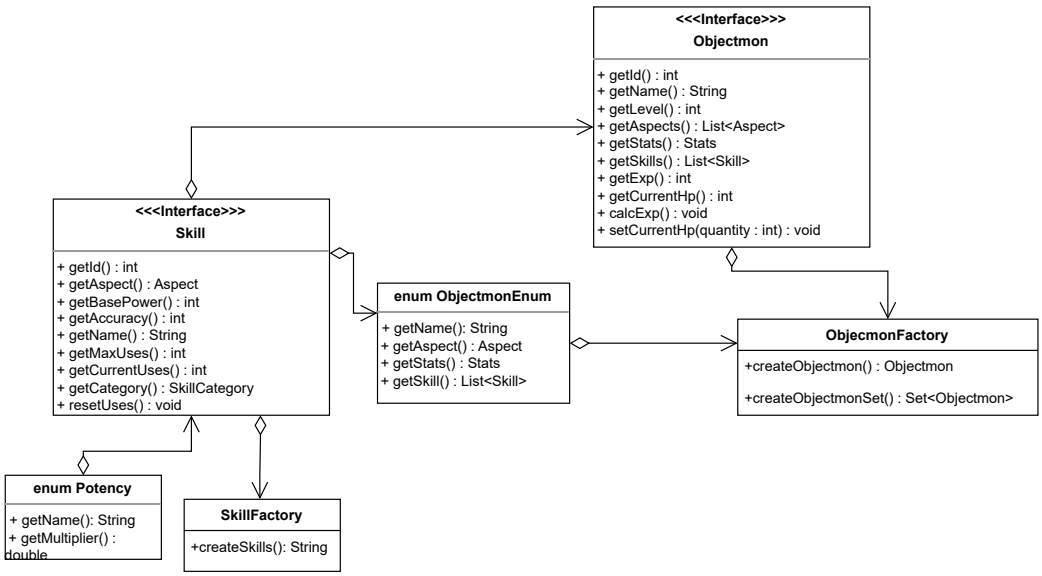


Figura 2.4: Rappresentazione UML dell'applicazione di Factory Pattern su Skill e Objectmon

Problema: Ci siamo ritrovati in una situazione dove la creazione di nuovi Objectmon e Skill diventava molto laborioso. Questo problema si metteva in evidenza soprattutto durante la popolazione di ObjectmonParty per i Trainer.

Soluzione: Per risolvere a questo problema si è ricorso al *Factory Pattern*. Attraverso l'implementazione di ObjectmonEnum, ObjectmonFactory e SkillFactory, invece di dover creare una nuova istanza di Objectmon da aggiungere al ObjectmonParty, adesso serve soltanto una Lista di ObjectmonEnum.

END GameState

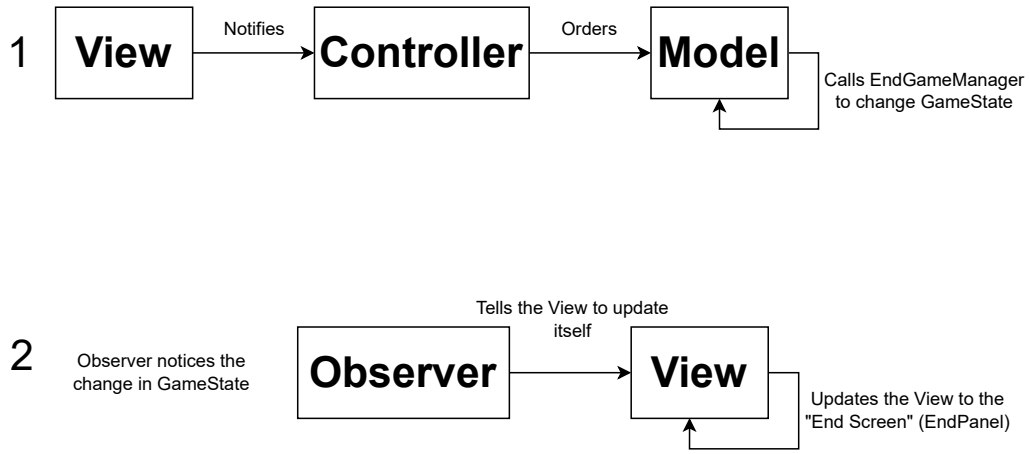


Figura 2.5: Rappresentazione UML dell'applicazione di Command Pattern per arrivare alla "End Screen"

Problema: Il gioco deve andare in una "End Screen" appena il Player vince o perde. Quando il giocatore arriva alla "End Screen" ha la possibilità di ricominciare la partita, facendo un reset del gioco.

Soluzione: Per risolvere a questo problema si è ricorso al *Command Pattern*. La View notifica il Controller che ordina al Model di aggiornare il GameState allo stato di END. L'Observer del GameState vede il cambio di stato e quindi aggiorna la View alla "End Screen". Quest'ultima ha un KeyListener che attiva il restart, creando una nuova istanza di Model e collegandola al Controller, facendo così un reset del gioco.

Azael Garcia Rufer

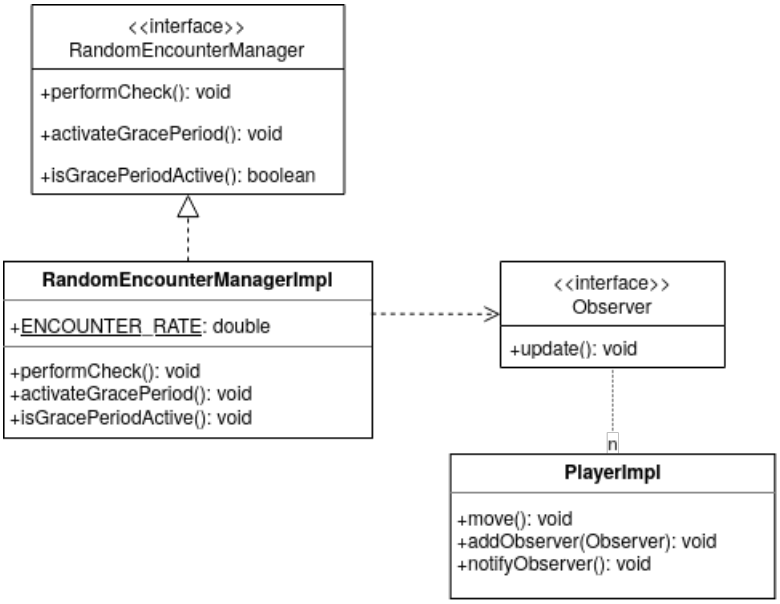


Figura 2.6: Rappresentazione UML dell'utilizzo dell'Observer Pattern

Problema: Volevamo implementare un sistema di random encounter che dia la possibilità di avviare battaglie contro wild Objectmon, questo sistema richiede che la posizione di Player corrisponda con l'erba alta, perciò sarebbe poco efficiente controllare la posizione di Player in continuazione.

Soluzione: Ho optato per l'utilizzo dell'Observer Pattern applicato al RandomEncounterManager che diventa un Observer di Player, a sua volta il Subject.
Questo permette di di notificare quando il Player effettua la mossa di nostro interesse: move(). Invoca quindi isTriggersEncounters() del Tile corrispondente alla posizione del Player e tramite random.nextDouble() ha un 20% di probabilità di fare iniziare la battle tra il Player e il wild Objectmon.

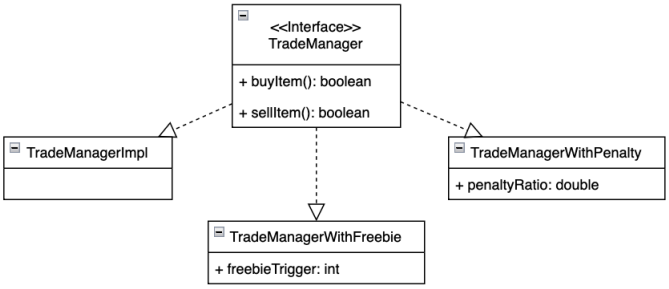
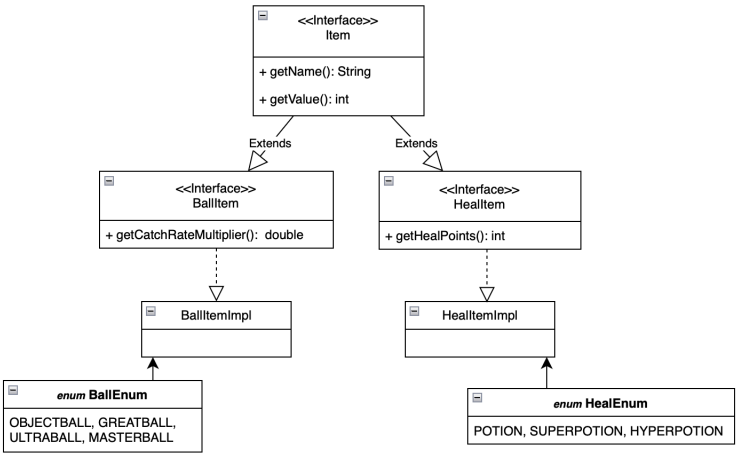


Figura 2.7: Rappresentazione UML dell'utilizzo del Decorator Pattern

Problema: Il nostro gioco prevede la presenza di un Vendor NPC, quando si interagisce con questo NPC c'è la possibilità di comprare e di vendere gli Item dell'inventario, il tutto è gestito da TradeManager. Per il nostro gioco volevamo che la vendita di un oggetto non restituisse il valore totale dell'oggetto ma il valore dimezzato e che se comprati tre oggetti dello stesso tipo se ne riceva il quarto in omaggio.

Soluzione: Ho optato per una soluzione tramite il Pattern Decorator che consente di arricchire una classe con nuove funzionalità, in particolare le due riportate sopra. Prendendo come Component l'interfaccia TradeManager, come ConcreteComponent l'implementazione TradeManagerImpl e come ConcreteDecorator TradeManagerWithFreebie e TradeManagerWithPenalty.

Per l'implementazione del Decorator Freebie durante l'acquisto di un item ho controllato che il modulo tra il numero di item comprati e il freebieTrigger(3) fosse 0, se sì aggiungo un item di quel tipo come regalo; per il Decorator Penalty durante la vendita di un item poichè la vendita di un item restituisce il value dell'item io ne prelevo il suo value moltiplicato per il penaltyRatio(0.5);



Problema: Per il nostro gioco abbiamo pensato a due tipi distinti di item: ObjectBall e ObjectHeal, rispettivamente 4 e 3 (nomi riportati in figura).

Soluzione: Ho pensato ad un interfaccia progettata per consentire ad entrambi tipi di item di estenderla, perciò abbiamo un interfaccia generica e per ognuno dei due tipi di item una ulteriore interfaccia (BallIItem, HealItem), una enum (BallEnum, HealEnum) e la conseguente implementazione (BallItemImpl, HealItemImpl).
Permettendo così il suo utilizzo per la creazione di un Inventory e del successivo TradeManager;

Capitolo 3

Sviluppo

3.1 Testing automatizzato

Per la verifica del corretto funzionamento dell'applicazione sono stati creati test automatizzati usando JUnit e Mockito.
I test realizzati mirano a garantire il corretto funzionamento del Model e del Controller.

- `TestGameLoop`: Verifica lo stato del gameloop e il funzionamento del framerate cap.
- `TestObjectmon`: Verifica il funzionamento di un Objectmon e la factory di essi.
- `TestPotency`: Verifica il funzionamento delle efficacie tra Aspects.
- `TestSkill`: Verifica la creazione delle Skill.
- `TestStats`: Verifica il funzionamento dei level up.
- `TestNPC`: Verifica il comportamento degli NPC.
- `TestPlayer`: Verifica le funzioni di movimento e sconfitta del Player.
- `TestGameState`: Verifica l'aggiornamento degli elementi Observer.
- `TestInventory`: Verifica il funzionamento di transazioni crediti e Items.
- `TestCollisionManager`: Verifica il funzionamento delle collisioni.
- `TestInteractionLogger`: Verifica il funzionamento di un logger per Overworld.

- `TestRandomEncounterManager`: Verifica il funzionamento della generazioni di Objectmon selvatici.
- `TestTradeManager`: Verifica il funzionamento delle sessioni di trading.
- `TestWorld`: Verifica il corretto caricamento del mondo.
- `TestAiTrainer`: Verifica la mossa randomica e lo switch.
- `TestBattleManager`: Verifica la corretta esecuzione delle mosse e la terminazione della battaglia.
- `TestCatchSystem`: Verifica il corretto funzionamento della cattura.
- `TestDamage`: Verifica la correttezza della formula dei danni.
- `TestObjectmonParty`: Verifica la capacità max e lo scambio posizioni.

3.2 Note di sviluppo

Jiaqi Xu

Utilizzo di Stream:
Permalink: https://github.com/jiekai-sun/OOP23-ObjectMon/blob/7a86f73d76cadd5584c
src/main/java/it/unibo/objectmon/model/data/objectmon/ObjectmonEnum.
java#L430C8-L432C19

Weijie Fu

Utilizzo pervasivo di Optional:
Permalink: https://github.com/jiekai-sun/OOP23-ObjectMon/blob/03d4c1530474ccbcfdf
src/main/java/it/unibo/objectmon/model/battle/impl/BattleManagerImpl.
java#L56-L73

Utilizzo di Lambda e Streams:
Permalink: https://github.com/jiekai-sun/OOP23-ObjectMon/blob/03d4c1530474ccbcfdf
src/main/java/it/unibo/objectmon/model/battle/impl/BattleManagerImpl.
java#L111-L113

Permalink: https://github.com/jiekai-sun/OOP23-ObjectMon/blob/03d4c1530474ccbcfdf
src/main/java/it/unibo/objectmon/model/battle/impl/RewardImpl.java#
L13-L15

Utilizzo di ImmutablePair della libreria ApacheCommons
Permalink: https://github.com/jiekai-sun/OOP23-ObjectMon/blob/03d4c1530474ccbcfdf
src/main/java/it/unibo/objectmon/model/ai/ChooseMoveImpl.java#L23-L33

Azael Garcia Rufer

Utilizzo di stream:
Permalink: https://github.com/jiekai-sun/OOP23-ObjectMon/blob/7fb83568eaafe702b88
src/main/java/it/unibo/objectmon/model/encounters/impl/RandomEncounterManagerImp
java#L77-L80

Utilizzo di Lambda e stream:
Permalink: https://github.com/jiekai-sun/OOP23-ObjectMon/blob/7fb83568eaafe702b88
src/main/java/it/unibo/objectmon/model/item/inventory/impl/InventoryImpl.
java#L45-L83

Utilizzo di Reflection ai fini di testing:

Permalink: https://github.com/jiekai-sun/OOP23-ObjectMon/blob/7fb83568eaafe702b88
src/test/java/it/unibo/objectmon/model/randomencountermanager/TestRandomEncounte
java#L66-L128

Utilizzo di generici:
Permalink: https://github.com/jiekai-sun/OOP23-ObjectMon/blob/7fb83568eaafe702b88
src/main/java/it/unibo/objectmon/model/item/inventory/impl/InventoryImpl.
java#L35-L43

Jiekai Sun

Utilizzo di Gson:
Permalink: https://github.com/jiekai-sun/OOP23-ObjectMon/blob/443fc19978695190387
src/main/java/it/unibo/objectmon/model/world/TileSet.java#L59-L73
Permalink: https://github.com/jiekai-sun/OOP23-ObjectMon/blob/443fc19978695190387
src/main/java/it/unibo/objectmon/model/world/MapData.java#L98-L111

Utilizzo di Lambda:
Permalink: https://github.com/jiekai-sun/OOP23-ObjectMon/blob/443fc19978695190387
src/main/java/it/unibo/objectmon/controller/ControllerImpl.java#
L79-L84
Utilizzo generale nella seguente classe: https://github.com/jiekai-sun/
OOP23-ObjectMon/blob/443fc19978695190387768c78c0209510ceffd7/src/
main/java/it/unibo/objectmon/view/battleview/CommandPanel.java#L52-L72

Per quanto riguarda il GameLoop ho tratto ispirazione dall'implementazione
del professor A. Ricci, mostratoci durante il seminario GameAsALab.

Capitolo 4

Commenti finali

4.1 Autovalutazione e lavori futuri

Jiaqi Xu

Contribuendo principalmente al Model sono soddisfatto dalle mie parti svolte, ma penso che avrei potuto implementare di più con il tempo concesso.
Ci sono stati problemi generali di pianificazione, principalmente dal fatto che l'idea iniziale era troppo complessa (colpa mia) per il tempo limite
e quindi abbiamo dovuto ridimensionare il progetto un certo numero di volte.
Questo non è il primo "grande" progetto in cui ho partecipato, ma l'esperienza è stata diversa, in quanto in quello precedente era un progetto in singolo, mentre il progetto corrente è stato piacevole, in quanto ho avuto la possibilità di collaborare con i miei colleghi.

Weijie Fu

Durante lo sviluppo del progetto, ho individuato alcuni punti di debolezza, soprattutto nella fase di progettazione. Quando non mi immergo direttamente nel codice, riscontro difficoltà nel trovare una soluzione ottimale. In altre parole, pur avendo un'idea su come affrontare il problema, mi accorgo che emergono nuove sfide che mi costringono a ripensare il design. Inoltre, ho difficoltà nel trovare il pattern più adatto per la situazione. Tuttavia, tra i miei punti di forza, posso gestire efficacemente la parte logica del progetto e utilizzare in modo efficiente i componenti disponibili. Nel contesto del team, mi sono impegnato a contribuire in modo significativo al raggiungimento degli obiettivi comuni. Ho assunto responsabilità specifiche assegnate e ho lavorato diligentemente per completare i compiti assegnati in modo tempestivo e accurato.

Azael Garcia Rufer

Il mio ruolo nel gruppo è stato quello della creazione di Item di due tipi: 4 ObjectBall(oggetti ball) e 3 ObjectHeal(oggetti curativi), l'implementazione dell'Inventario che ha fornito le basi solide per poi implementare la vendita e l'acquisto di Item(Trade) interagendo con NPC e infine la gestione di Incontri casuali con Objectmon selvatici. Considero questo progetto come un ottima esperienza, istruttiva e concreta. Attraverso l'implementazione dei concetti appresi durante il corso ho avuto l'opportunità di approfondire la mia comprensione della programmazione orientata agli oggetti. Durante lo sviluppo mi sono trovato di fronte a difficoltà e dubbi e sicuramente è stato molto utile anche il comunicare e il collaborare in squadra oltre che l'impegno personale. Sono soddisfatto del risultato finale ma soprattuto del coordinamento all'interno del team che ha favorito un ambiente stimolante e produttivo, le competenze acquisite grazie a questo progetto sono sicuro prima o poi si ripresenteranno e saprò come sfruttarle al meglio.

Jiekai Sun

Avendo avuto l'opportunità di contribuire a tutti e tre gli aspetti dell'architettura MVC, posso affermare di essere molto soddisfatto delle parti svolte relative al Model e al Controller.
Ho investito tempo ed energia nell'implementazione di queste parti, cercando di fornire una solida base per la logica di business e l'efficacia del flusso di controllo.
Tuttavia avverto delle lacune nella progettazione complessiva, che avrebbero potuto essere colmate con una pianificazione più approfondita e una maggiore attenzione ai dettagli e bisogni degli altri membri del team.
Essere alla guida del gruppo, soprattutto considerando che questo è anche il mio primo progetto, mi ha posto di fronte a sfide non semplici.
Ho dovuto assumere il ruolo di figura di riferimento per risolvere dubbi e chiarire incomprensioni, ma anche di quello che sprona e incoraggia il resto del team. Nel complesso, questa esperienza è stata un'ottima opportunità di crescita e apprendimento.

Appendice A

Guida utente

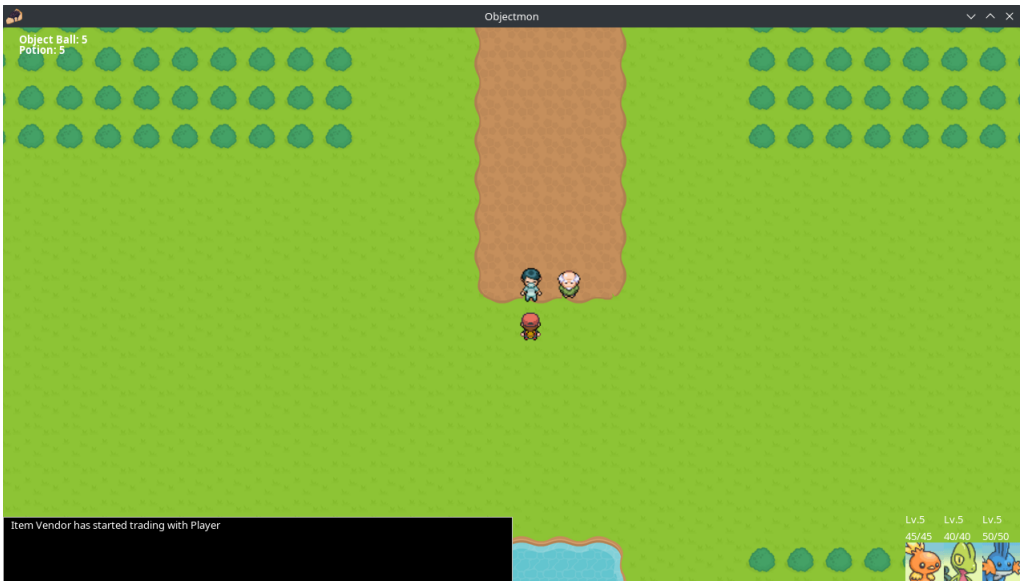


Figura A.1: Schermata overworld

Per muovere il personaggio i controlli sono WASD, per interagire con un NPC bisogna posizionarsi di fronte ad esso e premere il tasto J.
Ci sono tre tipi di NPC: Trainers, infermiere e mercante.
Interagire con gli Trainers scatena una battaglia a turni.
Parlare con l'infermiere rigenera la vita di tutti gli Objectmon posseduti.
Interagire con il mercante permette di comprare e vendere oggetti usando i crediti ottenuti dalle vittorie contro gli allenatori.

Muoversi tra i cespugli ha una certa probabilità di scatenare un incontro casuale con un Objectmon selvatico, quest'ultimo catturabile utilizzando le Objectballs.

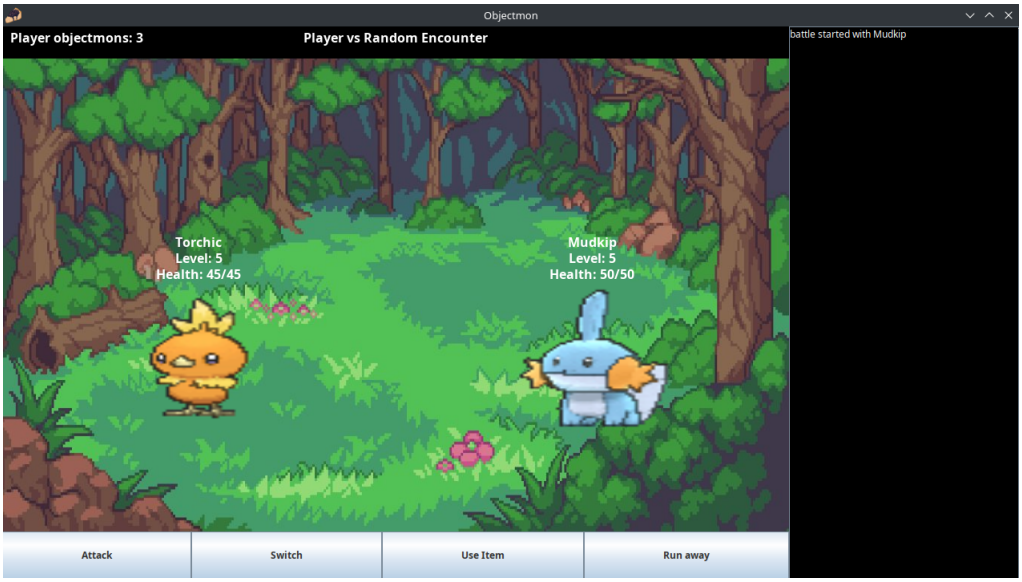


Figura A.2: Schermata combattimento

Una volta coinvolti in una battaglia saranno disponibili 4 tipi di azioni: *attacchi*, *switch*, *uso di item* e *fuga*.
Solo una mossa è concessa per turno, nel caso non fosse valida il turno non viene risolto (es. cercare di catturare un Objectmon non selvatico, fuggire da una battaglia contro un Trainer).
E' possibile ottenere maggiori informazioni sulle azioni disponibili librando il cursore sopra le singole mosse.
L'Objectmon che esegue per primo la mossa è quello che ha la statistica velocità più altra fra i due. Usare mosse non di attacco ha una priorità maggiore rispetto a quelle di attacco.

Se i punti vita di un Objectmon raggiungono zero esso viene rimosso permanentemente dal party. Il giocatore potrà comunque procedere nella battaglia ma nel caso in cui perde tutti gli Objectmon, sarà sconfitto e sarà necessario ricominciare la partita. E' perciò importante cercare una strategia per arrivare a fine gioco con un party di Objectmon solido.
Molte informazioni di gioco che potrebbero essere utili per effettuare la mossa migliore non sono mostrate. Questa è una scelta di design in quanto desideriamo che sia il giocatore ad inferire se usare una certa mossa sia ragionevole meno, attraverso successi e fallimenti.

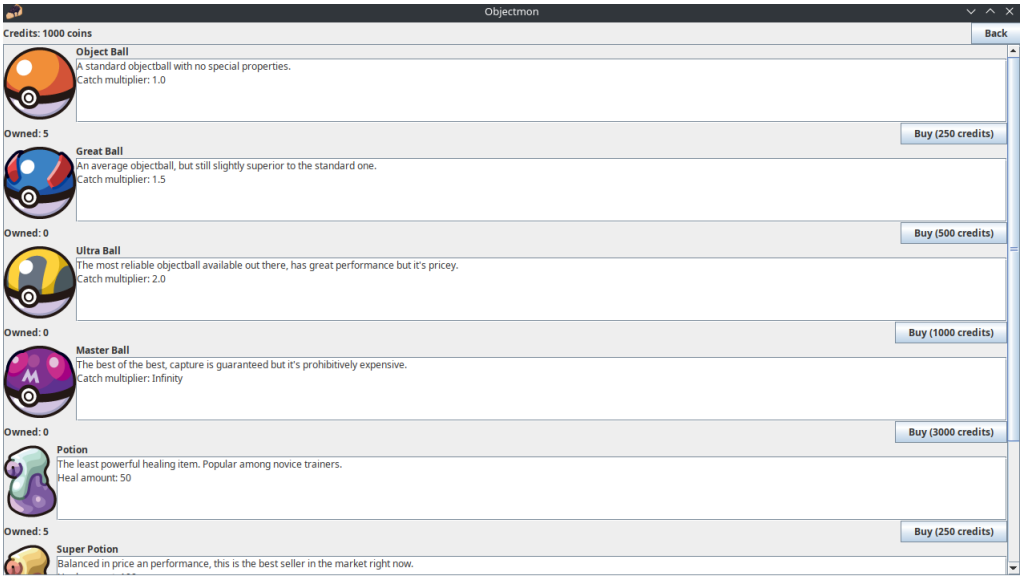


Figura A.3: Schermata shop

Sconfiggere i Trainers permette di aggiudicarsi crediti, spendibili in strumenti utili alla cattura o alla cura.
Nello shop è possibile anche vendere gli oggetti posseduti anche se a prezzo ridotto del 50% per evitare speculazioni.

Per offrire supporto al giocatore gli sviluppatori si sono sentiti generosi e ogni tre oggetti comprati dello stesso tipo al giocatore ne viene offerto uno in omaggio.

Appendice B

Esercitazioni di laboratorio

B.1 jiekai.sun@studio.unibo.it

- Laboratorio 09: `https://virtuale.unibo.it/mod/forum/discuss.php?d=149231#p211497`
- Laboratorio 10: `https://virtuale.unibo.it/mod/forum/discuss.php?d=150252#p212797`
- Laboratorio 11: `https://virtuale.unibo.it/mod/forum/discuss.php?d=151542#p213957`