Relazione del progetto di Programmazione ad Oggetti "Object One Piece"

De Tommaso Tommaso
Giorgini Matteo
Grimaldi Leonardo
Scorza Edoardo

18 febbraio 2024

Indice

Capitolo 1

Analisi

1.1 Requisiti

Il nostro progetto, richiesto per l'esame di Programmazione ad Oggetti, consiste in un videogioco di battaglie tra navi a tema One Piece.

Requisiti funzionali

- Il gioco è strutturato a turni, dove le mosse si alternano tra giocatore e nemici.
- Il giocatore controlla una nave con cui può navigare all'interno della mappa e interagire con essa attaccando i nemici, evitando le mine, raccogliendo i barili per guadagnare esperienza e attraccando alle isole per curarsi e salvare. Inoltre, l'esperienza raccolta dai barili potrà essere utilizzata per riparare la nave.
- Le navi nemiche hanno il compito di seguire ed attaccare la nave del giocatore.
- La mappa di gioco si genera in modo casuale e quindi il gioco prosegue all'infinito.

Requisiti non funzionali

- L'interfaccia grafica deve essere scalabile.
- Il turno dei nemici deve essere calcolato in tempi brevi.

1.2 Analisi e modello del dominio

Il mondo di gioco è diviso in sezioni che possono essere generate casualmente o caricate, se precedentemente salvate. Ogni sezione contiene il giocatore ed altre eventuali entità che possono essere: nemici, barili, mine ed isole. Il giocatore deve potersi muovere nella sezione corrente e deve poterla cambiare navigando nei bordi. Tutte le navi, che siano nemici o il giocatore, devono collidere con altre navi o entità. Devono inoltre poter sparare e danneggiare le altre navi. Il colpo sparato da una nave deve poter sorvolare i barili e le mine presenti lungo la traiettoria, ma non le altre navi e le isole. Una nave è composta da quattro componenti che sono: la chiglia, la prua, la vela e l'arma. Ognuno contribuisce ad una diversa abilità e la loro distruzione ne comporta quindi la perdità. Le isole, i barili e le mine devono eseguire determinate azioni alla collisione. Le mine dovranno procurare danno alle navi che collidono, i barili invece daranno dell'esperienza al giocatore che vi collide. Le isole, invece, dovranno riparare la nave del giocatore che vi ha colliso e salvare lo stato del gioco. Quando si parla di "salvare", si intende un checkpoint, ovvero l'abilità del giocatore di rinascere dove si è salvato. La possibilità di salvare lo stato del gioco per riprenderlo in un secondo momento dopo la sua chiusura non è prevista. I nemici devono vedere e seguire il giocatore, se questo entra nel loro raggio visivo. I nemici devono anche sparare al giocatore, se questo entra nel raggio dell'arma. I nemici non seguono il giocatore se cambia sezione.

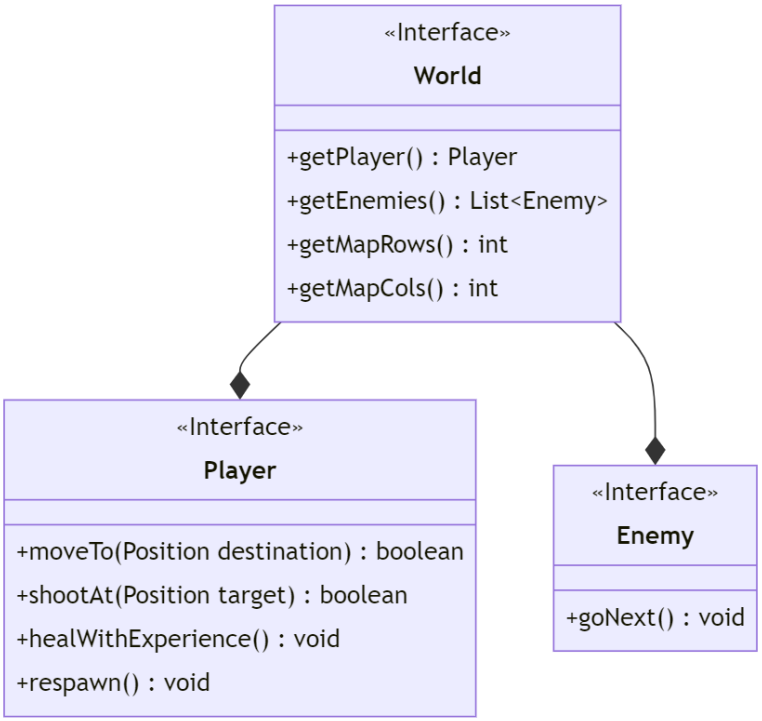

Figura 1.1: Schema UML dell'analisi del progetto

Capitolo 2

Design

2.1 Architettura

L'architettura del progetto segue il pattern architetturale MVC. Più nello specifico, la View si occupa di visualizzare a schermo lo stato corrente del gioco, il Controller si occupa di gestire gli input del giocatore ed agire di conseguenza, ed il Model si occupa unicamente della logica che definisce il gioco.

Grazie a questa architettura è molto più semplice modificare o cambiare completamente la View senza conoscere, né modificare, l'architettura interna del Model. Inoltre, il nostro gioco è strutturato per essere a turni ed è quindi "passivo", modificando il suo stato solo dopo un'input, tuttavia la logica dei turni è stata implementata nel Controller e questo potenzialmente ci permette di usare lo stesso Model anche per una versione del gioco in tempo reale che sarebbe quindi "attiva", dove il gioco non aspetta l'input dell'utente.

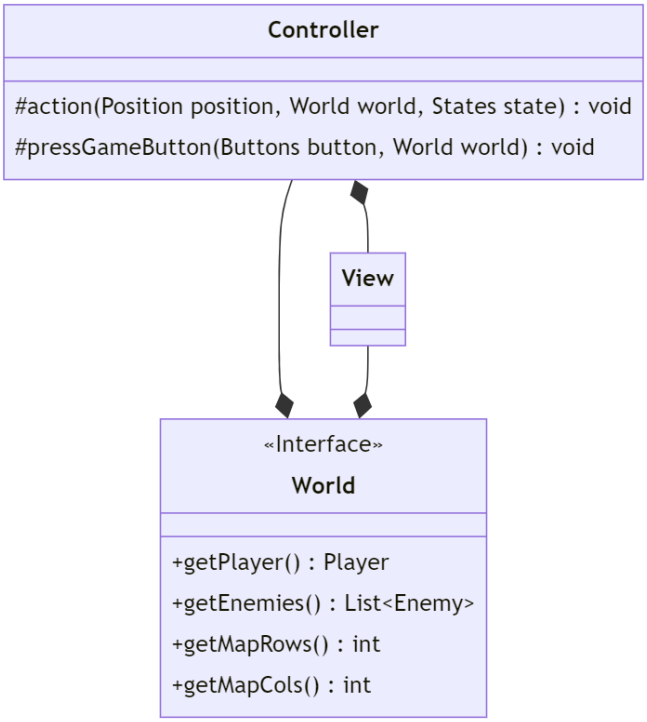

Figura 2.1: Schema UML dell'architettura del progetto

2.2 Design dettagliato

Tommaso De Tommaso

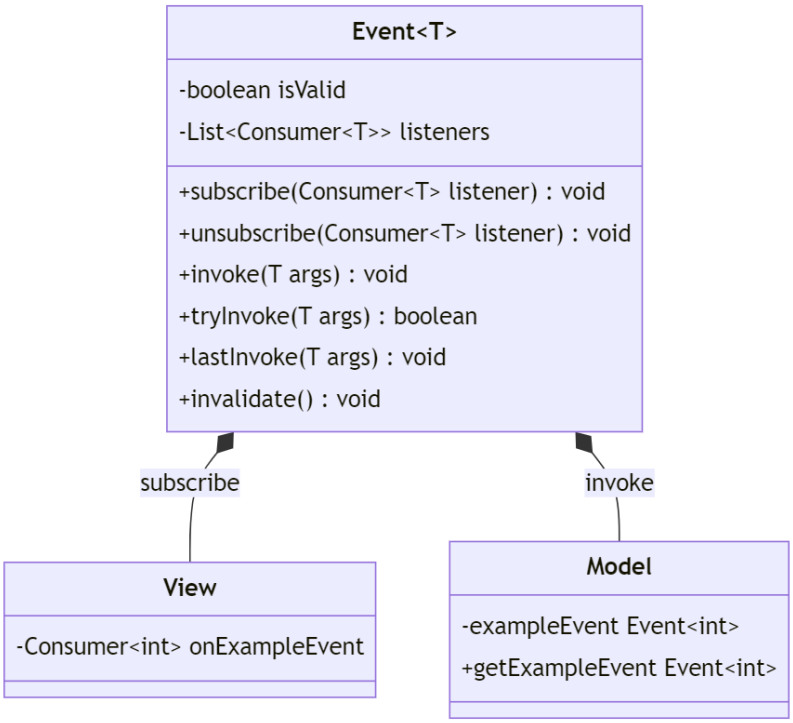

Figura 2.2: Rappresentazione UML della classe Event con una View ed un Model ad-hoc ed un esempio di utilizzo

Aggiornamento della View

Problema Il Model deve poter comunicare alla View quando e come aggiornarsi senza dover esporre la sua struttura interna.

Soluzione Usare l'*Observer pattern* per creare un oggetto generico e riutilizzabile che si occupi, nel nostro caso, di informare la View. Questo approccio garantisce anche che View e Model siano "loosely coupled".

Salvataggio del gioco

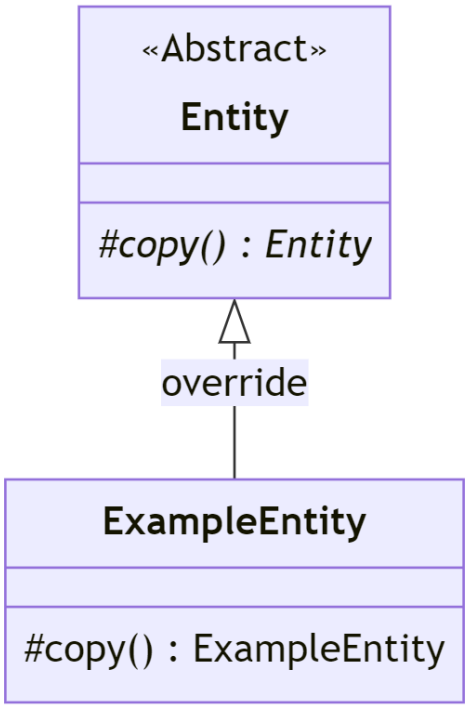

Figura 2.3: Rappresentazione UML della classe Entity con un ExampleEntity che eredita da questa e implementa il metodo astratto "copy"

Problema Il Model deve poter salvare il suo stato, questo implica la copia dello stato di ogni entità piuttosto che la copia del suo riferimento.

Soluzione Usare il *Prototype pattern* per delegare il processo di copia alle entità stesse, con un metodo astratto "copy" nella classe Entity e la sua implementazione in ogni classe che la estende.

Matteo Giorgini

Composizione della nave

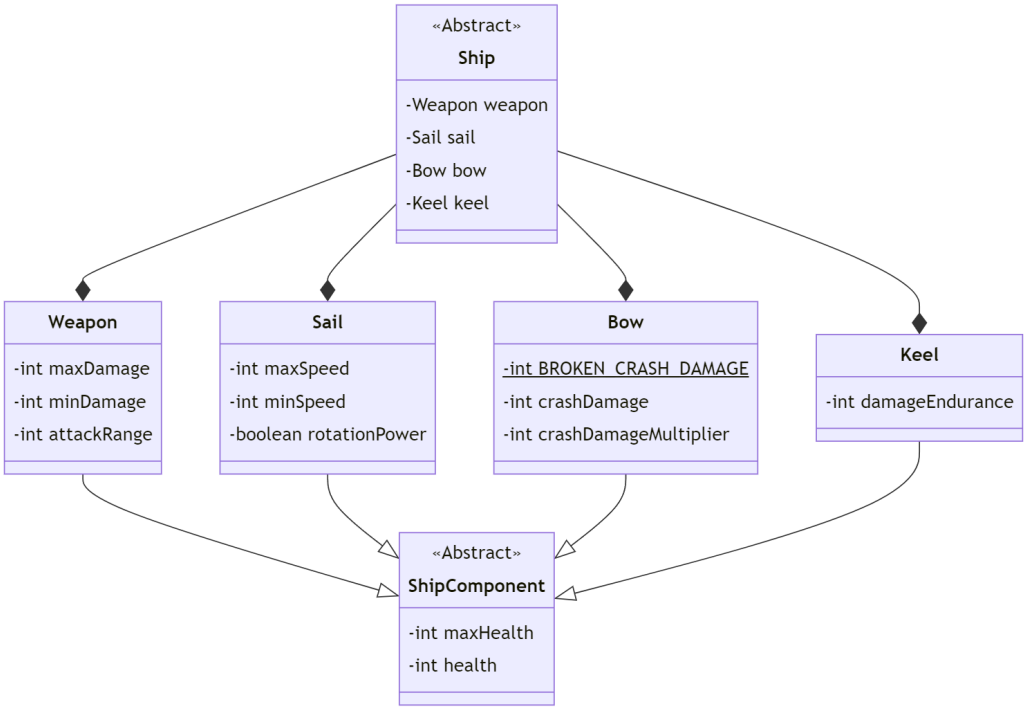

Figura 2.4: Rappresentazione UML del Composite pattern per la composizione della nave

Problema Una nave è composta da varie parti che si occupano di compiti differenti.

Soluzione Usare il *Composite pattern* per creare un componente composito che contiene i riferimenti ad i vari componenti della nave (Weapon, Sail, Bow, Ship). Ogni componente svolge una funzione differente e alla sua rottura causa un malus alla nave. Questo sistema permette di avere una nave eventualmente estendibile.

Costruzione di componenti con statistiche diverse

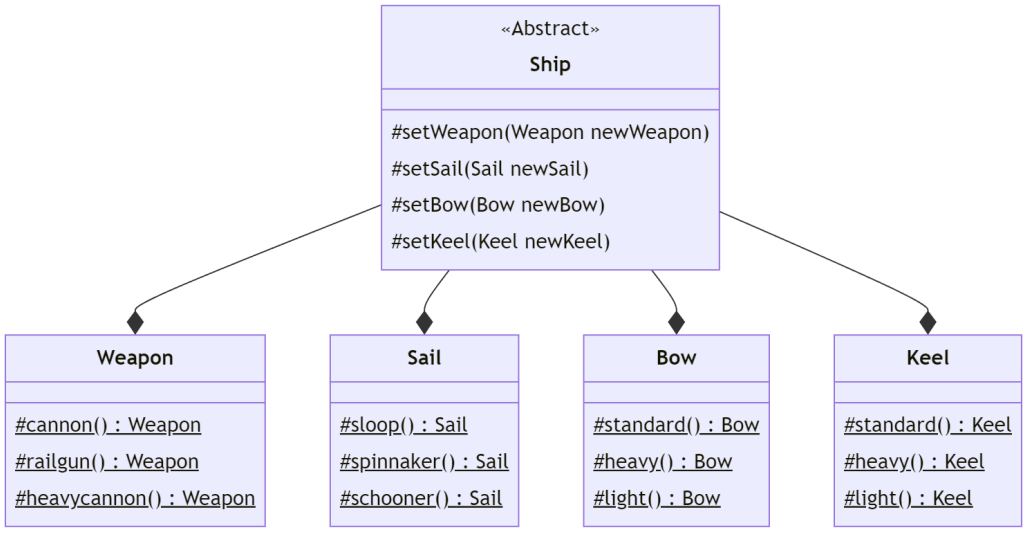

Figura 2.5: Rappresentazione UML del Factory pattern per la composizione della nave

Problema I componenti di una nave possono avere statistiche differenti.

Soluzione Usare il *Factory pattern* per creare configurazioni diverse di componenti con statistiche diverse. Ogni componente possiede al suo interno dei metodi che costruiscono il suddetto componente con valori differenti e incapsulano il costruttore.

Leonardo Grimaldi

Barra di progresso

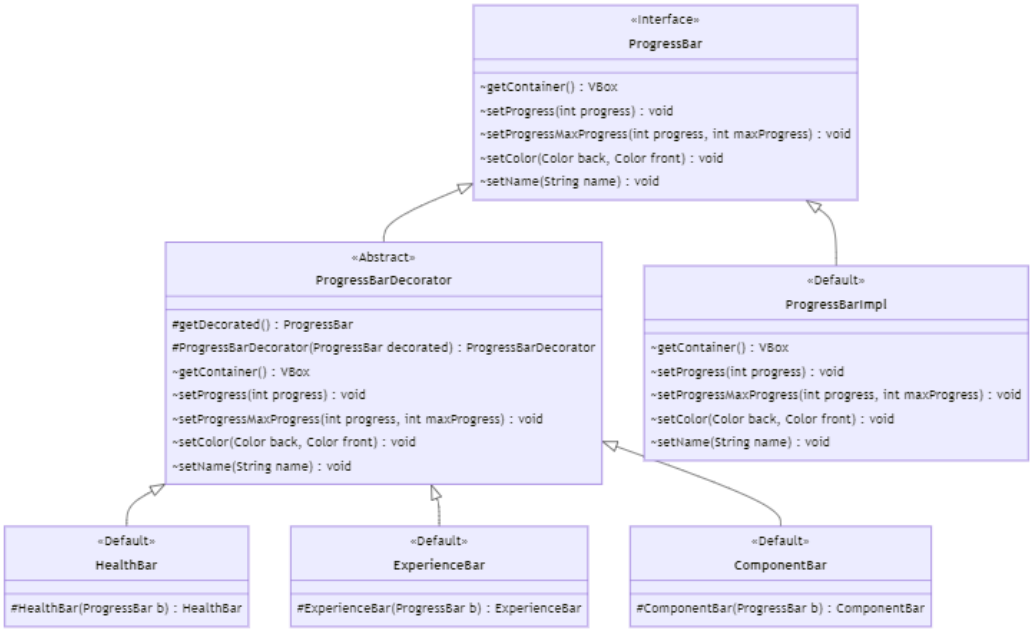

Figura 2.6: Rappresentazione UML del Decorator pattern per la creazione di vari tipi di barre di progresso

Problema La view deve rappresentare barre di vita con la stessa struttura, ma colori o proprietà diverse.

Soluzione Usare il *Decorator pattern* per creare una barra di progresso basilare che può essere assegnata colori diversi, associato un nome, aggiornata con il progresso e che restituisce il componente pronto per essere rappresentato. La classe astratta che fa da Decorator è `ProgressBarDecorator`, mentre le sue implementazioni sono

Tracciamento del player e dei nemici

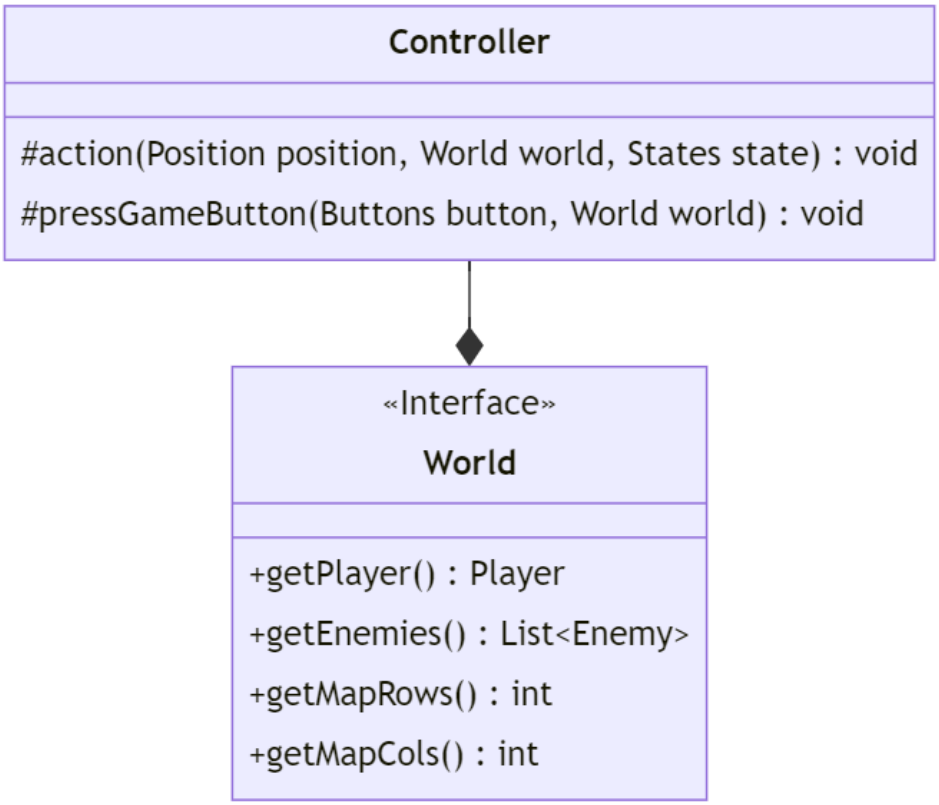

Figura 2.7: Rappresentazione UML del Facade pattern per l'esposizione di metodi

Problema Consentire al Controller di accedere al `Player` e alla lista di `Enemy` senza smascherare la struttura complessa interna del `World`

Soluzione Si è deciso di usare il *Facade pattern* per creare una interfaccia `World` che contenesse un metodo `getPlayer()` che ritornasse il `Player` e un metodo `getEnemies()` che ritornasse una lista di `Enemy`. Il Controller in questo modo non dovrà preoccuparsi di come vengono generate le sezioni oppure quale sia quella corrente: riceve subito le entità in quella presente per comodamente gestirne i turni. Alternativamente, avremmo potuto consentire al Controller di accedere direttamente al `WorldImpl` e le `Section`, ma avremmo violato principi di incapsulazione permettendo comportamenti indesiderati.

Edoardo Scorza

AI delle Entità Enemy

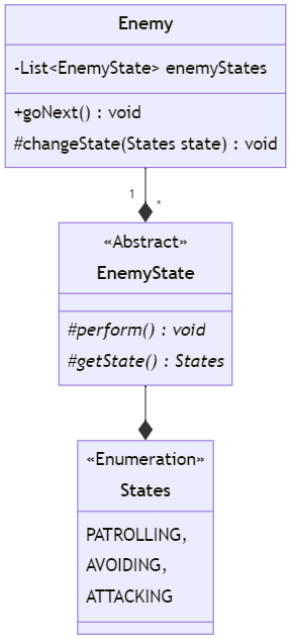

Figura 2.8: Rappresentazione UML dello State pattern dell'Enemy

Problema realizzare un sistema di algoritmi in grado di manovrare una Ship, al fine di navigare nella mappa ed attaccare il Giocatore in prossimità di essa.

Soluzione Ho deciso di optare per uno State pattern, ciò consente di separare le diverse logiche dall'Enemy e dalle logiche stesse, consentendo sia di creare algoritmi volti a una singola situazione ma anche facili da intercambiare.

Controller

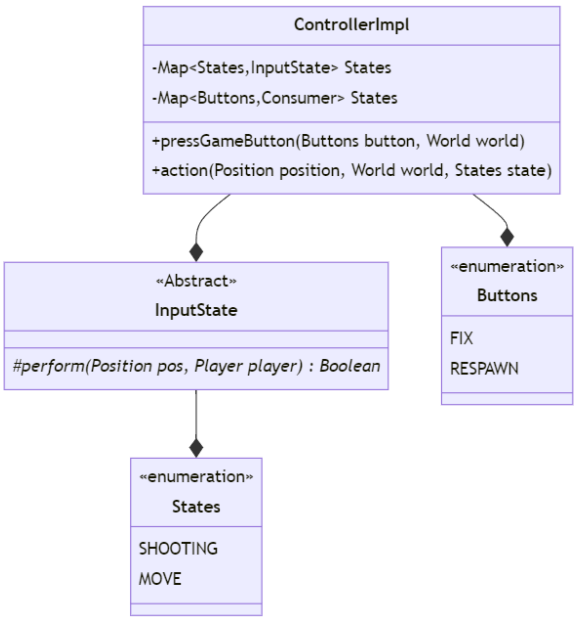

Figura 2.9: Rappresentazione UML del Strategy pattern usato dal controller per gestire gli input

Problema realizzare un controller che oltre a scandire il tempo del model, esponesse in modo chiaro le funzionalità del model in poche, semplici funzioni.

Soluzione Per l'input di gioco ho utilizzato uno Strategy pattern, atto alla separazione delle due varianti di input richieste dal nostro gioco: muoversi o sparare, in modo simile anche per la gestione di funzionalità del model collegabili a pulsanti esterni, attraverso l'uso di una mappa.

Capitolo 3

Sviluppo

3.1 Testing automatizzato

Abbiamo eseguito dei testing utilizzando la suite di JUnit.

- NavigationSystemTest.java testa la capacità del navigation system implementato (Compass) di dare la direzione corretta per raggiungere un obiettivo.
- UtilsTest.java esegue i test di tutti i metodi di Position e Bound in Utils.

3.2 Note di sviluppo

Tommaso De Tommaso

Utilizzo di una "custom" `Annotation`

Permalink: `https://github.com/thethunderingmarmot/OOP23-objectonepiece/blob/f1c77c3f045ecd5f6e5b50d53eaae92590d2c32e/src/main/java/it/unibo/object_onepiece/model/PlayerImpl.java#L14`

Utilizzo di `Stream`, `Function` e lambda expressions

Permalink: `https://github.com/thethunderingmarmot/OOP23-objectonepiece/blob/f1c77c3f045ecd5f6e5b50d53eaae92590d2c32e/src/main/java/it/unibo/object_onepiece/model/PlayerImpl.java#L111-L138`

Matteo Giorgini

Utilizzo di `Stream` e lambda expressions

Usate pervasivamente. Il seguente è un singolo esempio. Permalink: `https://github.com/thethunderingmarmot/OOP23-objectonepiece/blob/b935812786b9f4b2bf504b8380732d7609243f66/src/main/java/it/unibo/object_onepiece/model/Ship.java#L350C1-L353C123`

Utilizzo di `BiPredicate` dalla libreria java.util.function

Permalink: `https://github.com/thethunderingmarmot/OOP23-objectonepiece/blob/b935812786b9f4b2bf504b8380732d7609243f66/src/main/java/it/unibo/object_onepiece/model/Ship.java#L391C1-L396C11`

Utilizzo di `URL` dalla libreria java.net

Permalink: `https://github.com/thethunderingmarmot/OOP23-objectonepiece/blob/c3fa6ef0fc6adadd4db29c2110c899c068c49ea8/src/main/java/it/unibo/object_onepiece/view/Sound.java#L32C1-L38C7`

Utilizzo della libreria javax.sound

Utilizzata in vari punti. Un esempio è il seguente. Permalink: `https://github.com/thethunderingmarmot/OOP23-objectonepiece/blob/c3fa6ef0fc6adadd4db29c2src/main/java/it/unibo/object_onepiece/view/Sound.java#L52C1-L54C45`

Utilizzo di `Logger` dalla libreria java.util.logging

Permalink: `https://github.com/thethunderingmarmot/OOP23-objectonepiece/blob/c3fa6ef0fc6adadd4db29c2110c899c068c49ea8/src/main/java/it/unibo/object_onepiece/view/Sound.java#L56C1-L56C56`

Leonardo Grimaldi

Utilizzo dell'algoritmo di *White noise* dalla libreria JNoise

Permalink: `https://github.com/thethunderingmarmot/OOP23-objectonepiece/blob/7a0137595ea7d9762cb942ab84609625c92ad2cc/src/main/java/it/unibo/object_onepiece/model/Section.java#L85C9-L86C22`

Utilizzo di `GridView` e `GridModel` dalla libreria Grid

Permalink: https://github.com/thethunderingmarmot/OOP23-objectonepiece/blob/6e21ea8b2bcbecf144d98e7dac5f3d1b23f450d8/src/main/java/it/unibo/object_onepiece/view/ObjectOnePiece.java#L156C5-L176C6

Utilizzo della libreria JavaFX

Permalink: https://github.com/thethunderingmarmot/OOP23-objectonepiece/blob/6e21ea8b2bcbecf144d98e7dac5f3d1b23f450d8/src/main/java/it/unibo/object_onepiece/view/ProgressBarImpl.java#L27C5-L59C6

Utilizzo di `BiFunction` dentro una `Map`

Permalink: https://github.com/thethunderingmarmot/OOP23-objectonepiece/blob/6ae9d20e3d3e9858b3098309406cf93128f788fd/src/main/java/it/unibo/object_onepiece/model/Section.java#L35C5-L41C8

Utilizzo di `Stream`, `Optional` e lambda expressions

Permalink: https://github.com/thethunderingmarmot/OOP23-objectonepiece/blob/6ae9d20e3d3e9858b3098309406cf93128f788fd/src/main/java/it/unibo/object_onepiece/model/Section.java#L131C1-L142C6

Edoardo Scorza

Lambda e supplier per la costruzione di una Mappa

Permalink: https://github.com/thethunderingmarmot/OOP23-objectonepiece/blob/c3fa6ef0fc6adadd4db29c2110c899c068c49ea8/src/main/java/it/unibo/object_onepiece/model/Compass.java#L16C1-L26C7

Consumer

Permalink: https://github.com/thethunderingmarmot/OOP23-objectonepiece/blob/805803a2ce3b2c9844d0cecb63fac15f8a5d2780/src/main/java/it/unibo/object_onepiece/controller/ControllerImpl.java#L20C4-L22C7

Stream

Permalink: https://github.com/thethunderingmarmot/OOP23-objectonepiece/blob/805803a2ce3b2c9844d0cecb63fac15f8a5d2780/src/main/java/it/unibo/object_onepiece/controller/ControllerImpl.java#L26C9-L28C10

Capitolo 4

Commenti finali

4.1 Autovalutazione e lavori futuri

Tommaso De Tommaso

Questo progetto è stata un'ottima occasione più per "sviluppare" me stesso che il gioco in sé. La mia parte consisteva nello sviluppo del Player, del Barrel e dell'Island. Inoltre, dovevo sviluppare anche il sistema di salvataggio del gioco, siccome collegato alle Island. Tutto ciò è stato terminato prima del previsto ed ho quindi pensato di aver sovrastimato la mia parte. Temendo di non raggiungere la quota di 70 ore, ho proseguito nell'aggiunta di feature opzionali, rimanendo nei limiti del progetto, invece di fare testing. Fino a quel momento, infatti, nessun test era stato fatto a causa dell'erroneo sentimento maggioritario, nonché il mio, di testare il gioco giocandolo, non appena la View fosse stata completata. Inevitabilmente, quando ciò è accaduto, ci siamo ritrovati sommersi da bug e comportamenti inaspettati. Questo ha provocato diversi aggiustamenti "dell'ultimo minuto" che non hanno ricevuto la cura del resto del progetto. Oltre a questo, nella foga di aggiungere feature opzionali, si è deragliati molto dalla progettazione iniziale, causando diverse discordie. Queste cose, da perfezionista quale sono, mi hanno dato non poco fastidio, ma, dopotutto, la colpa è per lo più mia. Non penso di portare avanti questo progetto poiché, citando la meta-relazione, a mio avviso non è affatto "sufficientemente ben riuscito da poter esser usato come dimostrazione di esser capaci progettisti", ma anzi è un esempio da non imitare.

Matteo Giorgini

La realizzazione di questo progetto è stata un grande insegnamento sul lavoro di gruppo. Sono soddisfatto della parte che ho svolto nonostante ci siano

stati molti cambiamenti in corso d'opera, probabilmente a causa di una progettazione non troppo dettagliata. Il mio ruolo consisteva nella realizzazione dell'entità come elemento generale e della nave (movimento, combattimento, collisioni, etc.). Inizialmente ho sottostimato i tempi per la realizzazione della mia parte, ed ho quindi implementato successivamente nuove funzionalità come le mine ed i suoni. Successivamente però si sono rivelate più complicate del previsto sempre a causa della progettazione e degli scarsi test e hanno necessitato di vari adattamenti, che non hanno permesso la totale implementazione di alcune funzionalità opzionali. Nonostante questo la mia parte è risultata flessibile ed estendibile, soprattutto la nave ed i suoi componenti sono modificabili e sono contento di come ho realizzato le meccaniche di movimento, sparo e collisione della nave. Concludo dicendo che questo progetto è stato molto formativo su vari aspetti come confrontarsi per risolvere problemi comuni e risolvere errori in compilazione e eccezioni durante l'esecuzione. In futuro vorrei ultimare l'implementazione dei suoni e rendere possibile ottenere in gioco i componenti con statistiche differenti che ho creato per la nave.

Leonardo Grimaldi

La mia mansione in questo progetto era disegnare la grafica del gioco e la generazione casuale delle sezione nonché la gestione delle stesse. Il mio punto di forza è stato sicuramente la ricerca e l'uso di librerie esterne che mi hanno facilitato parte dello sviluppo e comprensione di modi per ricrearne lo stesso comportamento. Altresì, si sono dimostrate arme a doppio taglio per il tempo impiegato nel capirne il modo di uso e addattabilità al progetto, ma resto contento del conoscerne l'esistenza e impiego per futuri lavori. Il mio punto debole è stato sicuramente la poca progettazione. Molte parti del mio codice penso che potrebbero essere migliorate come ad esempio l'aggiornamento della View. Parlando dell'UI, sono soddisfatto della minimalità del gioco e come l'utente si trova tutte le informazioni necessarie visibili facilmente sullo schermo, ma riconosco che l'utente si trova senza dubbio in difficoltà nel capire la modalità di gioco e quali comandi sono consentiti e che avrei dovuto fare una guida visiva migliore. Penso di aver svolto un buon lavoro di teamwork e organizzazione, ricordando dead-line interne e cercando di aiutare il possibile da fuori. La ricerca di risorse in rete è stata indubbiamente un punto forte nel mio sviluppo dell'applicazione.

Edoardo Scorza

Il mio ruolo nel progetto consisteva nella realizzazione del controller(MVC) e dei nemici, In generale ho seguito il progetto nella fase di analisi e strutturazione, essendo parte attiva del progetto, ritengo di dover conoscerlo interamente per contribuire nel modo migliore possibile e, anche perchè la mia parte si basava su quella struttura. Il problema principale che abbiamo riscontrato è stata la coordinazione di gruppo, essendo il modello MVC, senza una variante di tutti i componenti non eravamo in grado di eseguire prove a mano, ne ci siamo preoccupati troppo di eseguire testing automatizzato, ciò ha seguitato a una serie di riscritture e adattamenti, sicuramente una cosa su cui riflettere. Fortunatamente ciò non ha precluso il lavoro finale, mi ritengo fortunato, abbiamo sempre discusso assieme a fronte di problemi e formulato strategie assieme. Sono abbastanza soddisfatto del mio lavoro, l'uso di pattern e il pensiero alla semplicità, mi hanno permesso di realizzare un enemy semplice modulabile ed efficace, in minima parte anche il controller, che soprattutto beneficiava delle scelte del model. Se dovessi pensare a un futuro per questo gioco direi che ci sono tante possibilità, non solo è strutturato come un sandbox, ovvero che estendendo le classi astratte è possibile aggiungere nuove entità, ma anche la natura stessa del gioco si può cambiare, potrebbe facilmente diventare Real-Time (invece che a turni)

4.2 Difficoltà incontrate e commenti per i docenti

Tommaso De Tommaso

Grazie a questo corso ho potuto scoprire Java che, nonostante conoscessi già abbastanza bene il C#, desideravo studiare in quanto molto usato in diversi settori. La cosa che mi ha colpito di più di questo corso è il fatto che ci si è concentrati anche sulla programmazione funzionale, soprattutto grazie all'utilizzo delle Stream di Java, su cui ero totalmente ignorante. Questo mi ha permesso di imparare qualcosa di nuovo, nonostante conoscessi già la programmazione ad oggetti. Il mio unico grande problema è stato adattarmi alla presenza del Type Erasure e, per questo motivo, col C# mi continuo a sentire più a casa.

Leonardo Grimaldi

Nonostante la complessità, apprezzo molto la varietà che il corso offre, insegnando non solo Java, ma anche progettazione, design, lavoro di gruppo, versioning e programmazione funzionale, lasciando agli studenti fondamenti molto importanti per il futuro proseguimento degli studi e mondo del lavoro. Sugli esercizi di laboratorio volevo dire che avvolte richiedono forse troppo tempo per essere completati e lasciano molto spazio allo studente per farli erroneamente. Preferirei invece che gli esercizi direzionassero meglio lo studente verso la soluzione ottimale, ovviamente non rivelandola. Personalmente, aprendo la prima volta un esame vecchio per prepararmi mi sono trovato in molta difficoltà a scrivere anche una sola riga di codice, non capendo che dovessi scrivere una classe anonima anche se avrei dovuto assimilare il concetto in laboratorio. Inoltre, molte volte mi è stato difficile reperire il docente o i tutor per aiutarmi a proseguire gli esercizi durante il laboratorio.

Appendice A

Guida utente

Avviando il gioco, l'utente si troverà sullo schermo una finestra con in centro una griglia azzurra (mappa di gioco) e un HUD (denotato dal testo "Player info") sulla parte destra. Nella griglia di gioco il player è identificato dalla nave più chiara con il teschio sulla vela. Esso si può muovere nelle quattro direzioni premendo con il click sinistro del mouse sulla casella adiacente. Se la nave non è direzionata verso la casella in cui ci si vuole recare, la nave impiegerà un turno per girarsi. Quando il giocatore cerca di muoversi sulle caselle del bordo, si sposta su una sezione diversa del mondo e non potrà più tornare alla precedente ammenoché non l'abbia salvata e sia in seguito "morto". Per sparare ad una nave nemica il giocatore deve premere con il click destro nelle celle in linea con i fianchi della nave. Il range di sparo è di 3 caselle sia per il player che per i nemici e il cannone fa un danno AoE (Area of Effect) di 3x3 caselle con il danno maggiore nel centro. È possibile anche speronare una nave nemica mentre si è accanto ad essa cliccandoci sopra con il tasto sinistro del mouse. Se il Keel di una nave è danneggiato, speronandola riceve un maggior danno rispetto a quello di default. Il giocatore può curarsi e salvare su una isola premendo il tasto sinistro mentre è a fianco ad essa. Se il giocatore viene affondato (Keel della nave a zero) per continuare il gioco dovrà premere su una casella e a quel punto respawnerà dove ha salvato oppure in una nuova mappa.
Nel HUD vengono mostrate quattro barre di vita che riguardano la nave:

- Weapon: vita dell'arma; raggiunto lo zero il giocatore non potrà più sparare.
- Bow: vita della prua; raggiunto lo zero non potrà più speronare.
- Sail: vita della vela; raggiunto lo zero non potrà muoversi.

- Keel: vita della chiglia; raggiunto lo zero il gioco si fermerà e l'utente dovrà cliccare su una casella per respawnare.

Inoltre, sotto alle barre di vita sono presenti tre pulsanti. Il primo con l'icona di una tavola di legno consente al giocatore di riparare la nave (curarsi) utilizzando l'esperienza raccolta dai barili nella mappa. Sotto ad esso si trova un numero che indica quanta esperienza l'utente ha a disposizione per ripararsi. Il costo di cura è 100. Il secondo con la scritta "Respawn" consente al player di rigenerare una nuova mappa oppure tornare all'isola salvata. Il terzo bottone spegne/riaccende la musica di sottofondo del gioco.

Appendice B

Esercitazioni di laboratorio

B.0.1 leonardo.grimaldi2@studio.unibo.it

- Laboratorio 08: `https://virtuale.unibo.it/mod/forum/discuss.php?d=148025#p209751`