

Relazione “Farming Simulator”

Linda Fabbri,
Federico Raffoni,
Simone Rega,
Giacomo Montali

20 Aprile 2021

Indice

1	Analisi	3
1.1	Requisiti	3
1.2	Analisi e modello del dominio	4
2	Design	6
2.1	Architettura	6
2.2	Design dettagliato	7
2.2.1	Simone Rega	7
2.2.2	Federico Raffoni	8
2.2.3	Linda Fabbri	9
2.2.4	Giacomo Montali	10
3	Sviluppo	13
3.1	Testing automatizzato	13
3.2	Metodologia di lavoro	14
3.2.1	Federico Raffoni	14
3.2.2	Linda Fabbri	15
3.2.3	Simone Rega	16
3.2.4	Giacomo Montali	17
3.3	Note di sviluppo	18
3.3.1	Federico Raffoni	18
3.3.2	Simone Rega	19
3.3.3	Linda Fabbri	19
3.3.4	Giacomo Montali	19
4	Commenti finali	20
4.1	Autovalutazione e lavori futuri	20
4.1.1	Simone Rega	20
4.1.2	Federico Raffoni	20
4.1.3	Linda Fabbri	21
4.1.4	Giacomo Montali	21

4.2	Difficoltà incontrate e commenti per i docenti	22
4.2.1	Simone Rega	22
4.2.2	Federico Raffoni	22
4.2.3	Linda Fabbri	22
4.2.4	Giacomo Montali	23
A	Guida utente	24
A.0.1	Schermata di Gioco	24
A.0.2	Mappa di Gioco	25
A.0.3	Animali	25
A.0.4	Negozi	25
B	Esercitazioni di laboratorio	26
B.0.1	Federico Raffoni	26
B.0.2	Simone Rega	26
B.0.3	Linda Fabbri	26

Capitolo 1

Analisi

1.1 Requisiti

Il gruppo si pone come obiettivo quello di realizzare "Farming Simulator", un software a scopo ludico basato sul videogioco "Hay Day"¹ sviluppato da Supercell. Il progetto mira alla realizzazione di un ambiente agricolo in 2D, in cui il giocatore, per mezzo del personaggio virtuale, interagisce con le colture e gestisce le proprie finanze. Lo scopo del gioco è arrivare all'apice della ricchezza, sbloccando tutti gli appezzamenti di terra e coltivando la maggior varietà di semi possibili.

Requisiti funzionali

- Questa applicazione deve permettere al giocatore di muoversi all'interno della mappa di gioco evitando di salire su zone di mappa non calpestabili.
- Il personaggio deve avere la possibilità di coltivare piantagioni di grano, patate, pomodori, carote, alberi di ciliege e altri frutti.
- Nella mappa deve comparire una zona per l'allevamento degli animali, grazie ai quali si possono raccogliere latte, uova e altri derivati.
- Farming Simulator deve mettere a disposizione un negozio in cui è possibile vedere il proprio inventario, vendere i prodotti agricoli raccolti in cambio di monete virtuali e comprare nuovi semi da coltivare.

¹<https://supercell.com/en/games/hayday/>

- Per aumentare la difficoltà di gioco devono essere messi a disposizione appezzamenti di terreno coltivabili inizialmente bloccati, che si possono sbloccare tramite l'utilizzo delle monete virtuali.
- Il gioco deve permettere il salvataggio della partita per poter dare la possibilità di essere ripreso in un secondo momento.

Requisiti non funzionali

- Farming Simulator deve poter essere eseguito su dispositivi con schermi dotati di risoluzioni differenti.
- Il programma deve poter essere eseguito sia su Windows sia su sistemi operativi Unix.

1.2 Analisi e modello del dominio

In Farming Simulator il soggetto principale è il personaggio virtuale, il quale si muove all'interno di una mappa, composta da diversi tipologie di blocchi.

All'interno della mappa possiamo trovare blocchi coltivabili (*Field Block*), blocchi non calpestabili dal personaggio (*Wall*), blocchi estetici (*Water*), il quale unico compito è quello di abbellire la mappa, blocchi che compongono la stalla in cui possono muoversi gli animali (*Stall*), ed infine blocchi "locked" (*Unlockable Block*), ovvero che inizialmente sono bloccati e possono essere sbloccati durante la partita in cambio di monete virtuali.

Il personaggio (*Player*), grazie all'inventario (*Inventory*), è in grado di mantenere risorse al suo interno. Tra le risorse collezionabili possiamo trovare: il cibo (*Food*) ottenuto dalla raccolta di coltivazioni o di animali oppure i semi (*Seed*). Questi ultimi possono essere comprati nel negozio (*Shop*) del videogioco, dentro il quale è anche possibile vendere il cibo che si è raccolto.

Il personaggio può interagire con blocchi e animali (*Animals*), in particolare può piantare e raccogliere i semi all'interno dei blocchi coltivabili (*FieldBlock*), sbloccare i blocchi non ancora accessibili (*UnlockableBlock*) e raccogliere i prodotti animali.

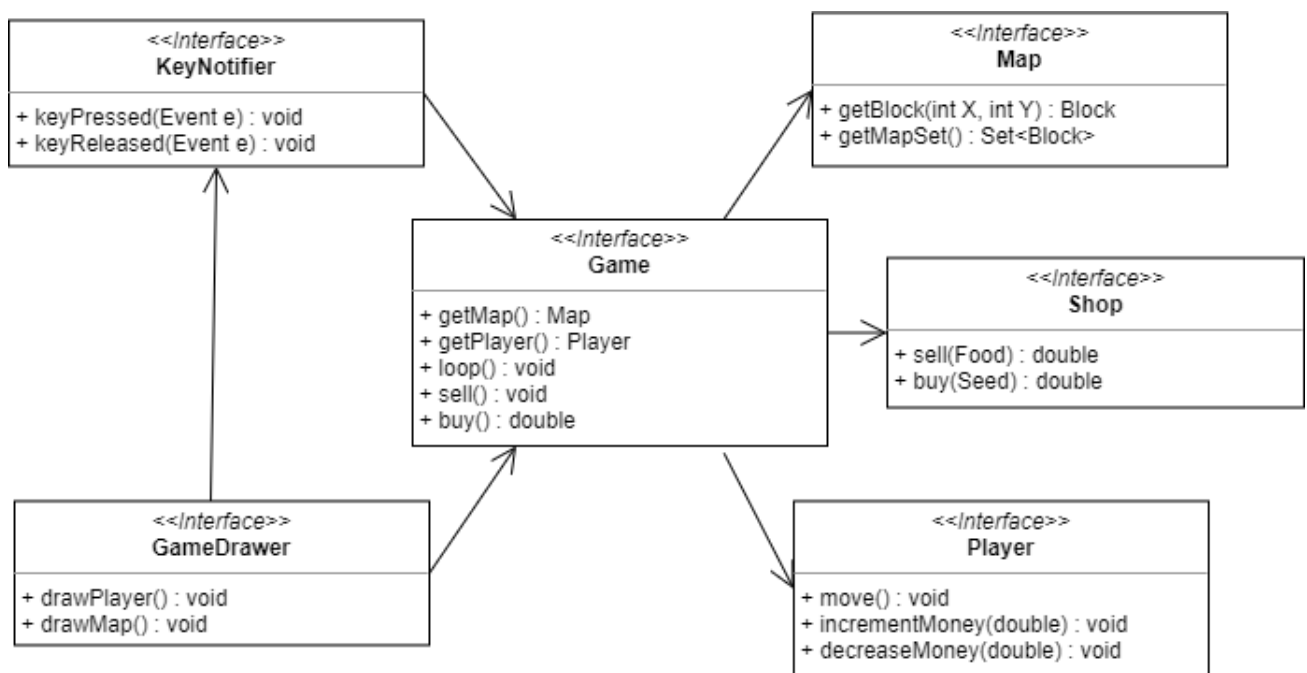


Figura 1.1: Schema UML dell'analisi del problema, con rappresentate le entità principali ed i rapporti fra loro.

Capitolo 2

Design

2.1 Architettura

L'architettura di Farming Simulator è basata su un pattern MVC. In particolare è stata sviluppata l'interfaccia Game che funge da Controller e si posiziona tra View e Model. Game gestisce inoltre eventuali interazioni tra componenti. All'arrivo di un input Game si occupa di ordinare al Model il cambio di stato.

Gli input da tastiera provenienti dall'utente passano attraverso la classe KeyNotifier, la quale si occupa di informare il Game dell'avvenuto input.

La parte di View è formata da diversi GameDrawer, i quali si occupano di interrogare il Game e mostrare a video il gioco.

Quando viene catturato un input da tastiera, il GameDrawer avvisa il KeyNotifier dell'avvenuta pressione. In questo modo il GameDrawer non si deve occupare della gestione degli input, spostando il carico di lavoro al KeyNotifier; questa astrazione permette al programma di non dipendere dall'implementazione e di poter cambiare il binding¹ o addirittura dispositivo di input (ad esempio un Joystick).

In Figura 2.1 è esemplificato il diagramma UML che descrive l'architettura MVC del progetto, mostrando le classi principali.

¹Binding = associazione di un tasto ad un determinato movimento e/o interazione visiva in gioco (ad esempio [WASD] per spostarsi nella mappa)

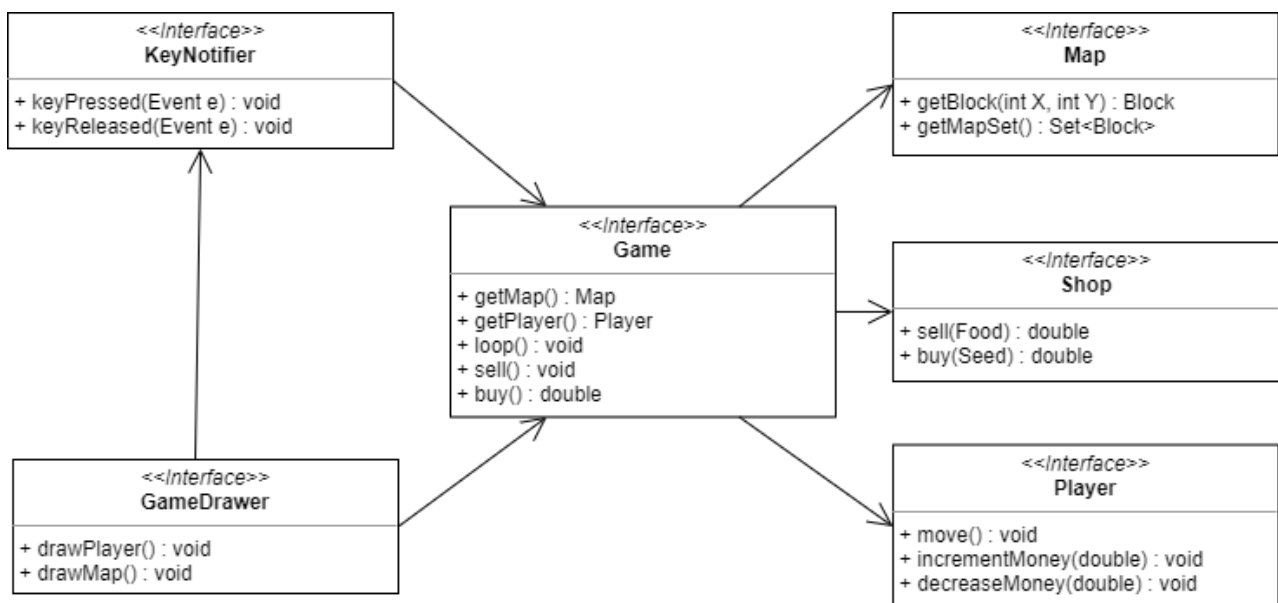


Figura 2.1: Schema UML architetturale di Farming Simulator, MVC.

2.2 Design dettagliato

2.2.1 Simone Rega

Io nel progetto mi sono occupato della parte riguardante l'aggiornamento dinamico della GUI nella sezione dello Shop, in particolare mi preoccupo di aggiornare il JPanel dello shop e la visualizzazione dell'inventario ogni qualvolta che si interagisce con i JButton di compra/vendita, la JComboBox e il JSpinner per comprare semi.

Ho ritenuto utile usare un pattern Observer in modo che alla ricevuta notifica di un "Observable", quindi un Evento sui componenti grafici descritti prima, gli altri "Observer" vengano notificati per poi essere aggiornati.

È stato deciso insieme a *Federico Raffoni* di creare le interfacce Observer e Observable di tipo generico, in modo da poter implementare le classi in più possibili situazioni eterogenee. Il ruolo di Observable è ricoperto dalla classe ObservableShopGUI mentre il ruolo di Observer è ricoperto da ObserverShop (classe innestata all'interno di ShopDrawer).

In Figura 2.2 è esemplificato il diagramma UML che descrive l'architettura del pattern Observer usato da Simone Rega, mostrando le classi principali.

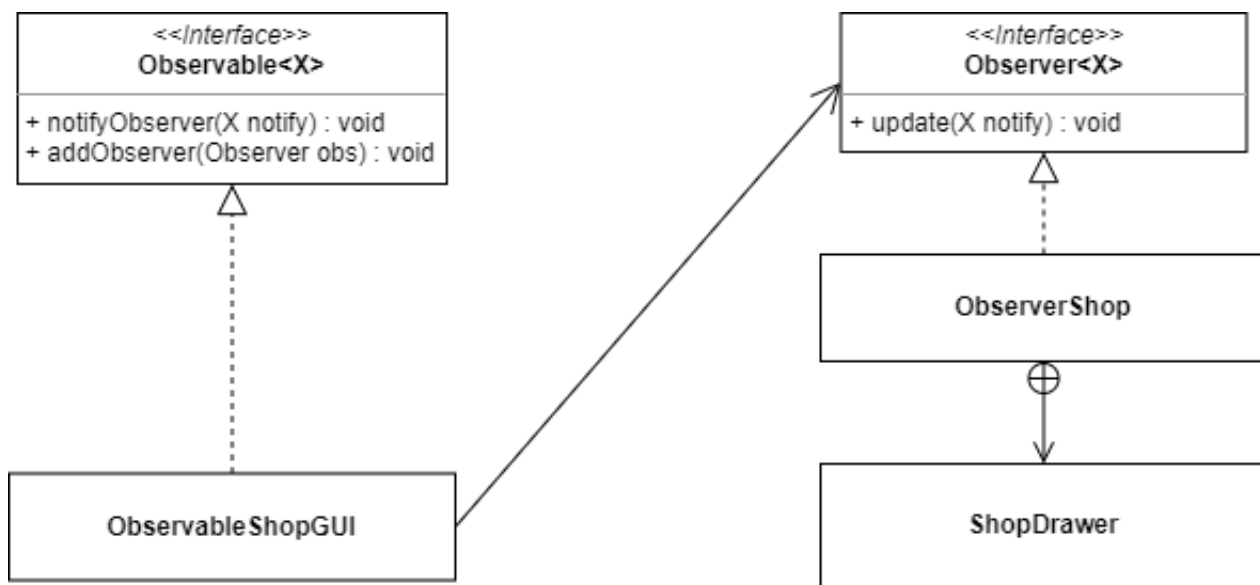


Figura 2.2: Schema UML architetturale del pattern Observer (Simone Rega).

2.2.2 Federico Raffoni

In questa sezione ci occuperemo principalmente del caricamento delle risorse grafiche e non solo. In particolare è stata creata la classe Resources, la quale si occupa di caricare e distribuire tutte le risorse.

È stato usato il pattern Singleton per rendere unico nell'applicazione l'accesso alla classe Resources. In questo modo si evita che diversi GameDrawer carichino più volte le risorse spreco di memoria.

Le risorse verranno caricate solo alla prima chiamata, mentre le chiamate successive riceveranno una copia statica della classe risparmiando tempo e risorse.

In Figura 2.3 è esemplificato il diagramma UML che descrive l'architettura singleton, mostrando le classi principali.

Per quanto riguarda il caricamento di una partita salvata, è stato utilizzato il pattern Observer per consentire all'Engine di rimanere in attesa mentre la classe GamePreloader si occupa di chiedere all'utente se continuare una partita salvata oppure iniziare una partita da zero.

In particolare Engine è "Observer", mentre la classe GamePreloader è "Observable". Sono state realizzate inoltre le interfacce Observer e Observable generiche, in modo da poter essere riutilizzate anche in altri campi.

In Figura 2.4 è esemplificato il diagramma UML che descrive l'architettura Observer, mostrando le classi principali.

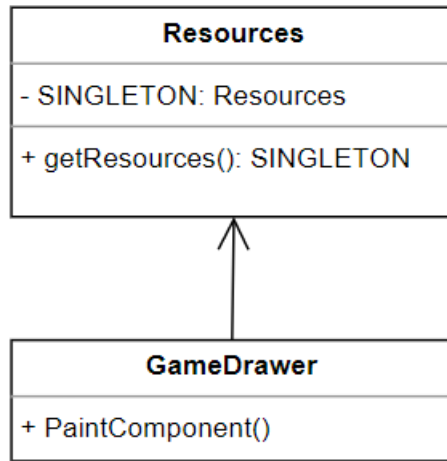


Figura 2.3: Schema UML architetturale del pattern Singleton (Federico Raffoni).

2.2.3 Linda Fabbri

La parte di progetto riguardo al design di cui mi sono occupata personalmente è quella sui blocchi.

Questa parte è composta da una classe enum (BlockType) per la classificazione delle tipologie dei vari blocchi, in base alla tipologia ho avuto la necessità di implementare metodi diversi, dunque ho creato tre interfacce: Block per tutti i generici blocchi, FieldBlock (estensione di Block) a cui ho aggiunto i metodi per i blocchi coltivabili (FIELD in BlockType), UnlockableBlock (estensione di FieldBlock) a cui ho aggiunto i metodi per i blocchi coltivabili bloccati (LOCKED in BlockType).

L'implementazione di queste interfacce si trova in FactoryBlock come classi innestate. Inoltre, per ogni tipologia di blocco si ha un differente costruttore al fine di determinare i vari campi.

Ho scelto l'utilizzo del pattern "Factory method" allo scopo di creare e ritornare blocchi in base al loro tipo mediante l'uso di costruttori differenti,

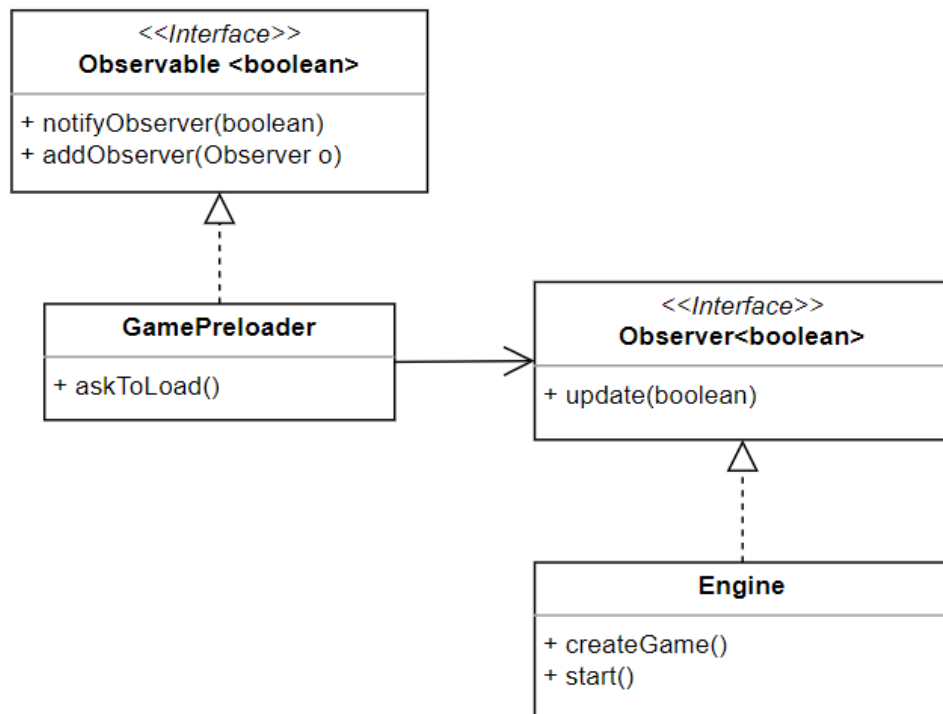


Figura 2.4: Schema UML architetturale del pattern Observer (Federico Raf-foni).

delegando la responsabilità della scelta della creazione di ogni oggetto solo alla classe che implementa tale pattern.

In Figura 2.5 è esemplificato il diagramma UML che descrive l’architettura Factory Method, mostrando le classi principali.

2.2.4 Giacomo Montali

I principali punti dei quali mi sono occupato in questo progetto sono le varie entità presenti in essa, cioè il Player e i vari animali.

In particolare per questi ultimi ho utilizzato il pattern “Simple Factory” nella classe `FactoryAnimal`, con l’obiettivo di delegare la creazione degli animali a quest’ultima, separandola così dalla classe `AnimalImpl` per rendere

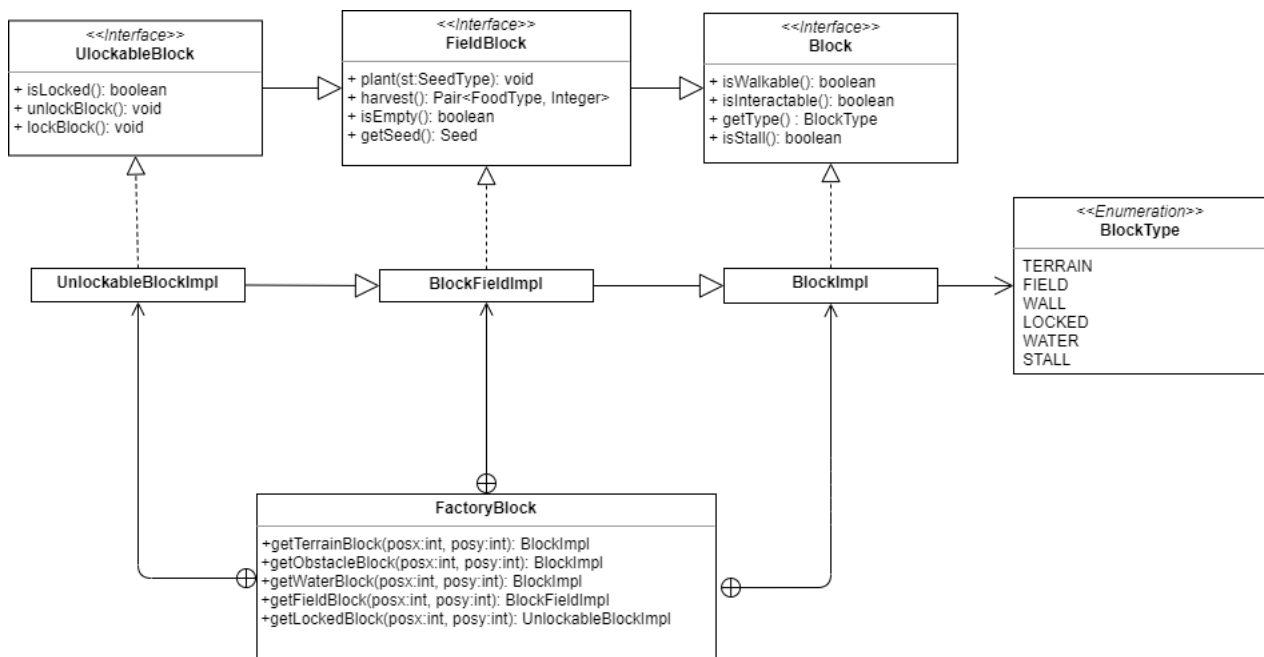


Figura 2.5: Schema UML architetturale del pattern Factory Method (Linda Fabbri).

più facile l'aggiunta di eventuali nuove classi che implementano l'interfaccia `Animal`.

La suddivisione degli animali è stata effettuata grazie ad un Enum (`AnimalType`).

In Figura 2.6 è esemplificato il diagramma UML che descrive l'architettura del Pattern Simple Factory, mostrando le classi principali.

Inoltre ho creato la classe astratta `Entity` al fine di renderla classe padre di `PlayerImpl` e `AnimalImpl`, evitando il riutilizzo del codice derivato dai metodi in comune, cioè quelli che riguardano il movimento e la posizione sulla mappa.

In Figura 2.7 è esemplificato il diagramma UML che descrive l'architettura delle classe `Abstract Entity`, mostrando le classi principali.

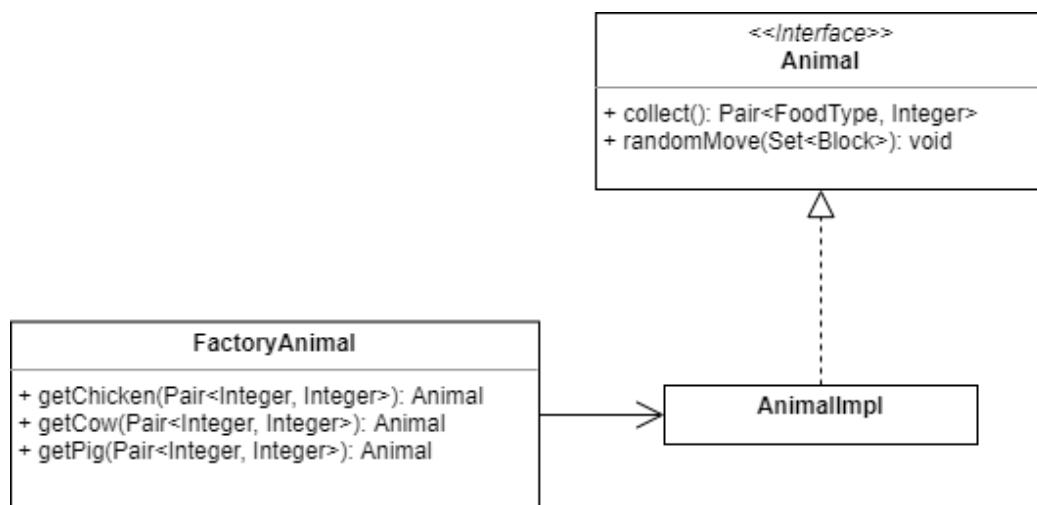


Figura 2.6: Schema UML architetturale del pattern Simple Factory (Giacomo Montali).

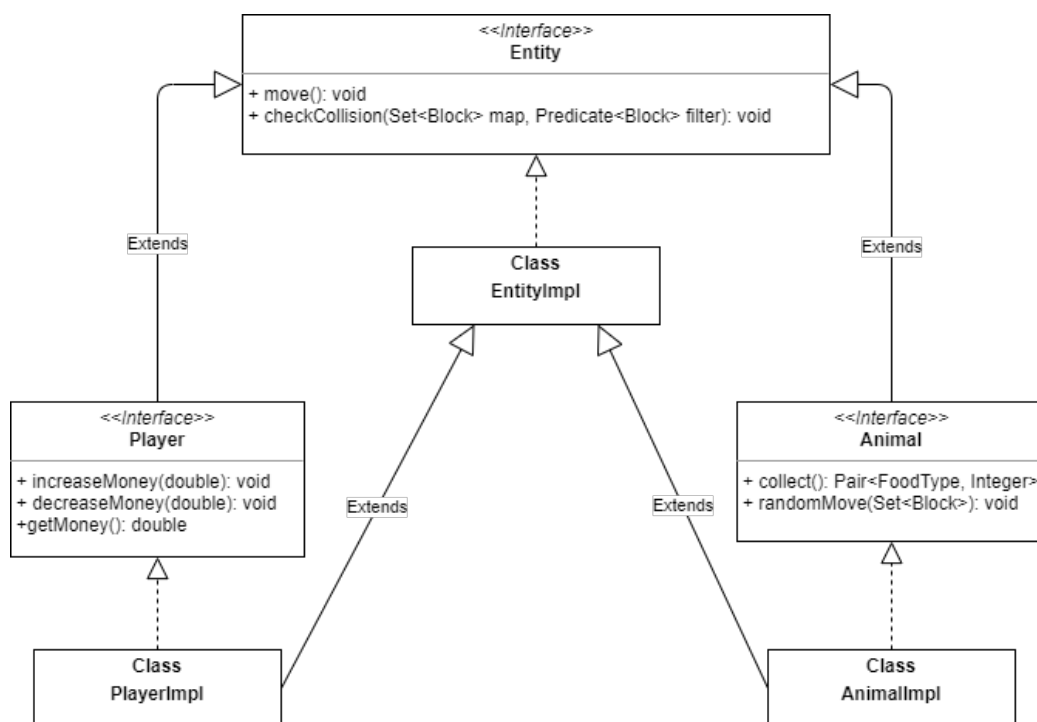


Figura 2.7: Schema UML architetturale della classe Entity (Giacomo Montali).

Capitolo 3

Sviluppo

3.1 Testing automatizzato

In questo progetto abbiamo deciso di testare prevalentemente la parte di Model poiché riteniamo sia la più propensa ad avere bug; tuttavia non mancano test riguardanti la View e salvataggio del gioco. Per svolgere test completamente automatizzati sul progetto è stato usato JUnit5.

Le classi che sono state testate sono:

- **testPlayerMovement:** con questo test muovo il personaggio simulando la pressione dei tasti;
- **testPlayerCollision:** con questo test controllo che non avvenga il movimento su blocchi o in direzioni in cui non mi è permesso;
- **testPlayerPlant:** con questo test simulo l'interazione del personaggio con un blocco di tipo `FieldBlock`, piantando un seme e poi controllando che quel blocco non sia vuoto;
- **testUnlockBlock:** con questo test simulo l'interazione con un blocco di tipo `UnlockableBlock`;
- **testUnlockBlockWithoutMoney:** con questo test simulo l'interazione con un blocco di tipo `UnlockableBlock` senza soldi;
- **testPlayerAnimalInteraction:** con questo test mi sposto nella zona "Stalla" della mappa e aspetto che un `Animal` qualunque sia pronto per poi poter raccogliere il suo *Food* simulando l'interazione tra *Player* e *Animal*;
- **testBuy:** con questo test provo a comprare un *Seed* dallo *Shop*, controllando poi che sia effettivamente presente nell'*Inventory*;
- **testBuyWithoutMoney:** questo test è simile al precedente, ma non ho monete, verificando quindi che non avvenga l'acquisto;

- **testSell:** in questo test simulo una normale interazione con un Field-Block, ricavandone poi il raccolto che proverò a vendere, verificandone l'esistenza nell'inventario e l'ottenimento dei soldi ricavati dalla vendita;
- **testSaver:** con questo test simulo il salvataggio del gioco su file con relativa apertura da esso in una nuova esecuzione del gioco, verificando che i nuovi e vecchi *Game* e *Player* siano gli stessi e abbiano le stesse caratteristiche;
- **testSaverPlayer:** questo test è simile al precedente, differisce semplicemente per il fatto che il Player viene mosso nella nuova istanza di gioco, così da far risultare falso il fatto che l'istanza salvata e l'istanza nuova creata siano uguali.

3.2 Metodologia di lavoro

Metodologia riguardante il progetto in generale

Lo sviluppo del progetto è stato preceduto da una fase preliminare di analisi e progettazione, durante la quale tutti insieme abbiamo lavorato per definire le basi dell'applicazione e, successivamente, per specificare i vari dettagli implementativi tramite UML sulla piattaforma Miro.

In queste fasi la collaborazione e il confronto sono state molto intense.

La suddivisione delle parti di Farming Simulator è avvenuta in modo abbastanza equo assegnando a ciascuno una parte significativa e importante del progetto, anche in base alle nostre preferenze.

Distributed Version Control System

Come DVCS è stato usato Git, creando un repository su GitHub. Abbiamo lavorato sullo scheletro del progetto su un unico branch main, per poi creare diversi branch per implementare e/o mettere a punto la realizzazione di feature nuove o più sostanziose.

3.2.1 Federico Raffoni

Sviluppo del motore grafico, in particolare l'Engine, la classe WindowManager e le varie interazioni tra questi due elementi. Ideazione e creazione della classe abstract GameDrawer che ha portato alla realizzazione delle varie scene di gioco. Sviluppo della classe KeyNotifier che permette l'interazione tra la view e il controller (Game).

Implementazione della classe Resources.

Sviluppo della classe che permette il salvataggio e il caricamento della partita (GameSaver).

Ho inoltre contribuito all'implementazione della classe Game, allo sviluppo della parte grafica nella classe MainScreenDrawer e allo sviluppo dei vari Test di gioco.

Classi sviluppate singolarmente:

- KeyNotifier
- Engine
- GameSaver
- GameDrawer
- GamePreloader
- Resources
- WindowManager
- MusicPlayer

3.2.2 Linda Fabbri

Si è occupata principalmente della parte riguardante la logica di gioco legata alle interazioni fra le entità e la parte che si occupa della creazione dei blocchi che compongono la mappa di gioco, in particolare le classi Interaction, InteractionImpl nel package engine e tutte le classi del package block.

Legato alle interazioni ha implementato i metodi riguardanti la compravendita di prodotti che si trovano nella classe ShopImpl, in particolare i metodi sellAll e buy.

In collaborazione con Simone Rega è stata implementata la classe ShopDrawer per la parte che si occupa della grafica dei pannelli legati alla compravendita di semi e prodotti.

Ha collaborato all'implementazione della classe Game, in particolare per chiamare i metodi implementati in InteractionImpl.

In collaborazione con altri ha eseguito il refactoring per correggere errori evidenziati dal checkstyle.

Classi sviluppate singolarmente:

- Block
- BlockType
- FactoryBlock

- FieldBlock
- Interaction
- InteractionImpl

Classi sviluppate insieme ad altre persone:

- mainScreenDrawer: implementazione del metodo drawMap per associare ai diversi tipi di blocco l'immagine corrispondente e in particolare l'immagine relativa ad ogni seme piantato.
- ShopDrawer: la logica legata alla pressione dei tasti
- FarmingSimulatorTestClass
- GameImpl: la parte che chiama i metodi relativi alle interazioni
- Shop
- ShopImpl
- UnlockableBlock

●Problemi

In fase di progettazione nella classe Inventory si era pensato di creare solo i metodi gotFoods e gotSeeds, per verificare l'esistenza di un certo numero di Food o Seed nell'inventario tuttavia ,per i metodi delle classi che sono state citate precedentemente, si è trovata la necessità di un metodo che ritornasse la mappa di tutte le tipologie di Food e Seed.

3.2.3 Simone Rega

Sviluppo delle classi all'interno del package Item che riguardano principalmente la realizzazione degli elementi Seed e Food del gioco.

Sviluppo della parte di grafica riguardante lo shop (ShopDrawer); le informazioni del gioco (InfoDrawer e implementazione di vari JPanelHUD nel mainScreenDrawer).

Ideazione e creazione delle classi nel package gameShop riguardante la logica base dello Shop.

Ideazione e sviluppo della classe test in JUnit5 a cui hanno collaborato allo sviluppo Linda Fabbri e Federico Raffoni.

Classi sviluppate singolarmente:

- FoodType
- SeedType
- Seed
- SeedImpl
- ItemConstants
- InfoDrawer
- QuitDrawer
- JPanelINFO
- JPanelHUD
- ObservableShopGUI

Classi sviluppate insieme ad altre persone

- MainScreenDrawer : implementazione di generateHUD per generare la barra di gioco superiore
- ShopDrawer : quasi tutta la parte grafica
- FarmingSimulatorTestClass
- Observer
- Observable
- Shop
- ShopImpl
- UnlockableBlock

3.2.4 Giacomo Montali

Sviluppo del package entity, che riguarda principalmente Player, Animal e le relative classi ad esse connesse.

Sviluppo della mappa di gioco dentro la quale interagiranno le entità, creazione del file della mappa e caricamento della mappa di gioco da file.

Collaborazione nella classe Game per la realizzazione grafica a runtime degli animali.

Classi sviluppate singolarmente:

- Entity

- EntityImpl
- Animal
- AnimalImpl
- AnimalType
- FactoryAnimal
- Direction
- Player
- PlayerImpl
- Inventory
- InventoryImpl
- Map
- MapImpl

3.3 Note di sviluppo

3.3.1 Federico Raffoni

- **Lambda:** ho usato le lambda all'interno del metodo loop della classe GameImpl per muovere tutte le entità e per impostare i blocchi camminabili dal player
- **Reflection:** All'interno della classe GameSaver

Ho utilizzato la libreria Gson per serializzare e deserializzare i componenti di gioco per permettere il salvataggio e il caricamento della partita. Nell'implementazione della classe GameSaver è stato necessario sviluppare un serializzatore di interfacce il quale è stato scritto prendendo spunto da questo thread (<https://stackoverflow.com/questions/62358189/gson-stackoverflowerror-during-serialization>)

Per la creazione della classe MusicPlayer, non essendo pratico delle funzioni di libreria di javax.sound, ho fatto riferimento a snippet presenti sul web.

3.3.2 Simone Rega

Per quanto riguarda le Feature Avanzate del linguaggio:

- **Lambda:** utilizzate nella classe FarmingSimulatorTestClass per i test sul Player e sui Blocchi;
- **IntStream:** utilizzate nella classe FarmingSimulatorTestClass.

Per quanto riguarda le librerie utilizzate

- **Java AWT**
- **Java Swing**

Sono consapevole che queste due librerie sono state introdotte a lezione spiegandone i principali elementi di maggior rilevanza ma ci tengo comunque a precisare lo studio e l'approfondimento per poter realizzare elementi di grafica più complessi o che purtroppo non abbiamo avuto tempo e modo di trattate a lezione.

3.3.3 Linda Fabbri

Per quanto riguarda le Feature Avanzate del linguaggio:

- **Optional:** nella classe riguardanti i blocchi per verificare se il terreno è stato seminato o se può essere seminato;
- **Lambda:** utilizzate nella classe FarmingSimulatorTestClass per i test sullo Shop.

3.3.4 Giacomo Montali

Per quanto riguarda le Feature Avanzate del linguaggio:

- **Filter:** utilizzato nella classe MapImpl e nella classe EntityImpl;
- **Lambda:** utilizzata per la creazione di animali nella classe Game, nelle classi InventoryImpl e AnimalImpl;
- **Stream:** utilizzato nell'Inventory;
- **Optional:** utilizzato nell'Inventory.

Classi riutilizzate:

- Per comodità ho riutilizzato la classe Pair fornita dal professore Viroli durante le sue lezioni, apportandole qualche piccola modifica.

Capitolo 4

Commenti finali

4.1 Autovalutazione e lavori futuri

4.1.1 Simone Rega

Sin dall'inizio sono stato partecipe e disponibile per la progettazione e l'analisi del progetto.

Devo dire che sono leggermente dispiaciuto perché nella fase di sviluppo avrei voluto essere più partecipe o comunque dedicarmi di più allo sviluppo del programma. Avrei potuto migliorare l'ottimizzazione del mio codice e le sue funzionalità, o magari osare di più con elementi di programmazione più avanzati introdotti a lezione.

Sono contento del Team con cui ho avuto modo di lavorare a questo progetto. La comunicazione è stata di fondamentale importanza. È stato il primo vero progetto di Programmazione ad Oggetti a cui ho preso parte e, se devo essere sincero, l'inesperienza si è fatta sentire: non avevamo punti di riferimento riguardo a mole e tempo di lavoro necessario, i quali ci hanno forse portato a non sfruttare al meglio le nostre capacità.

Sono abbastanza contento del risultato finale riguardante il lavoro da me svolto e della modesta qualità del mio codice, che è stata equilibrata e trasparente, permettendone un utilizzo corretto e utile; ma sono contento anche del risultato finale generale che ha portato alla realizzazione di un gioco molto fedele alla nostra idea in fase di progettazione.

4.1.2 Federico Raffoni

Sono molto soddisfatto del gruppo e di quello che abbiamo prodotto. Per me è stato il primo lavoro di sviluppo con altre persone, è stato quindi fondamentale per capire le dinamiche del lavoro di gruppo.

Il mio compito è stato quello di creare il collante tra parte di grafica e la parte logica, costruendo un semplice Engine di gioco.

Probabilmente a causa di inesperienza e per mancanza di competenze (principalmente grafiche) abbiamo sovrastimato la quantità di lavoro necessaria per svolgere questo progetto, scartando altri progetti perché credevamo troppo grandi.

Ad inizio progetto non avevo idea di come poter creare un engine che legasse la logica del gioco ad una buona interfaccia reattiva che mostrasse in tempo reale la posizione del personaggio. Seppur avessimo già utilizzato Java Swing a lezione non avevo ancora le basi necessarie a costruire un semplice gioco in 2d. Questo mi ha spinto subito a cercare una soluzione a questo problema impegnandomi per parecchio tempo ad inizio progetto.

4.1.3 Linda Fabbri

Sono soddisfatta del mio operato in quanto è il mio primo approccio alla programmazione ad oggetti e al lavoro di gruppo. Ho riscontrato qualche difficoltà nella coordinazione con gli altri membri, specialmente all'inizio del progetto a causa delle tante idee differenti su come strutturare lo scheletro del gioco, tuttavia, una volta stabilite le linee guida sono riuscita a lavorare alla mia parte con tranquillità.

Ho riscontrato qualche problema nella scelta dei nomi di metodi e di campi in inglese, in quanto non padroneggio al pieno la lingua.

In conclusione, sono soddisfatta del progetto ultimato, il quale rispecchia l'idea da cui eravamo partiti.

4.1.4 Giacomo Montali

Lo sviluppo di questo progetto mi ha permesso di crescere notevolmente dal punto di vista organizzativo, in quanto questo è stato il primo software prodotto lavorando in team. Questo ha infatti portato a delle iniziali difficoltà, dovute alla mancata conoscenza nella gestione di progetti di questa scala.

Nonostante tutto, al termine di questo, grazie ad una ottima collaborazione all'interno del team e alle conoscenze apprese a lezione, mi trovo soddisfatto del lavoro svolto.

Ritengo di aver acquisito abilità nello scrivere ed apprezzare codice di qualità; inoltre ho trovato molto utile Git, software che inizialmente sottovalutavo, il quale si è invece rivelato fondamentale per la realizzazione del progetto e la gestione di esso.

4.2 Difficoltà incontrate e commenti per i docenti

4.2.1 Simone Rega

In conclusione sono soddisfatto del mio percorso in questa materia e ritengo che il corso sia stato utile e funzionale al mio percorso di studi e formazione professionale. Ritengo che il carico del lavoro sia tarato abbastanza bene, senza inutili e superflue ripetizioni basilari o concetti magari già trattati da altri corsi come Programmazione e Algoritmi, ma anche senza lasciare troppe lacune o parti spiegate e non affrontate.

I professori sono stati a mio avviso capaci di spiegare e motivare al proseguimento attivo del corso.

Purtroppo il mio percorso di sviluppo del progetto di OOP non è stato lineare nel tempo; nonostante fosse stimolante per me avere per la prima volta un progetto di programmazione così bello e complesso da portare avanti, questo non è stato sempre possibile poiché non sempre sono riuscito a trovare il tempo per dedicarmi ad esso dopo le lezioni o lo studio. È effettivamente il primo progetto di sviluppo software da zero a cui ho partecipato e mi ha fatto crescere dal punto di vista organizzativo, rendendomi cosciente di ciò che mi aspetta una volta uscito dall'università.

4.2.2 Federico Raffoni

In conclusione mi ritengo molto soddisfatto di questo corso. Ritengo che sia stato estremamente utile non solo per apprendere nuove skills di programmazione ad oggetti, ma anche per comprendere le dinamiche del lavoro di gruppo.

Avrei preferito ricevere qualche nozione in più riguardo a Gradle, data la sua importanza all'interno dell'ambiente Java, in particolare nel contesto di un progetto grande come può essere questo.

Tuttavia sono molto contento delle competenze ottenute durante questo corso e coltivate durante la costruzione e lo sviluppo di questo progetto.

4.2.3 Linda Fabbri

Il corso, secondo la mia opinione, è stato chiaro ed esaustivo per quanto riguarda le basi della programmazione ad oggetti. Ho riscontrato alcune difficoltà a seguire al pieno le lezioni a causa delle scarse pause. Un problema che ho riscontrato durante l'esame è stata la mancanza di concentrazione e

il poco tempo a disposizione, forse dovuti principalmente alla modalità "in remoto" dell'esame e alla scarsa efficienza del mio pc.

4.2.4 Giacomo Montali

In conclusione ritengo questo un ottimo corso, grazie al quale ho portato la mia capacità nello scrivere il codice ad un livello superiore, che ha già portato risultati come per esempio in altri miei progetti. Valuto quindi questo come uno dei corsi più interessanti sostenuti fin'ora.

Appendice A

Guida utente

A.0.1 Schermata di Gioco

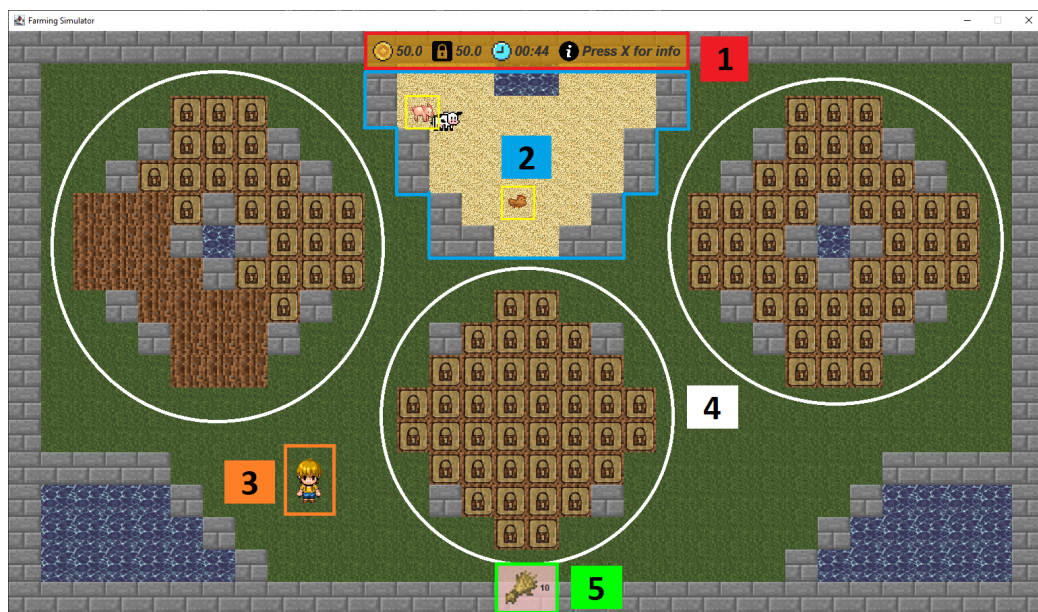


Figura A.1: Schermata iniziale di gioco: 1)Pannello Informativo; 2)Stalla; 3)Personaggio; 4)Aree coltivabili; 5)Selezione seme.

- 1) **Pannello informativo:** in questa parte della schermata puoi vedere le monete che hai a disposizione, il costo per sbloccare la prossima area coltivabile bloccata (vedere appendice A.0.2), il tempo trascorso dall'inizio della partita e, premendo X in gioco, compaiono le informazioni basilari sui controlli.

- 2) **Stalla:** luogo in cui vengono allevati gli animali(vedere appendice A.0.3).
- 3) **Personaggio:** avatar virtuale che si muove nella mappa tramite la pressione dei tasti WASD. Può interagire con alcune zone della mappa e con gli animali.
- 4) **Aree Coltivabili:** aree in cui è possibile coltivare i prodotti.
- 5) **Selezione seme:** girando la rotellina del mouse è possibile scegliere il seme da piantare.

A.0.2 Mappa di Gioco

La mappa di gioco è divisa in varie zone:



Premendo F, su queste zone puoi seminare e prendere il raccolto.



Se hai a disposizione abbastanza monete virtuali per sbloccare la zona di terra, premendo F su di essa potrai sbloccarla facendola diventare una zona coltivabile.

A.0.3 Animali

Il personaggio può interagire con gli animali che si trovano nella stalla solo quando quest'ultimi sono "pronti", cioè contornati di giallo.

Per interagire con questi bisogna premere la F.

A.0.4 Negozio

Premendo E si entra nel negozio, in cui si possono comprare semi da piantare nella parte in basso a sinistra e vendere i prodotti raccolti nella parte in basso a destra.

Nella parte in alto a destra puoi trovare l'inventario.

Appendice B

Esercitazioni di laboratorio

B.0.1 Federico Raffoni

- Laboratorio 05: <https://virtuale.unibo.it/mod/forum/discuss.php?d=62684#p102596>
- Laboratorio 06: <https://virtuale.unibo.it/mod/forum/discuss.php?d=62579#p103324>
- Laboratorio 09: <https://virtuale.unibo.it/mod/forum/discuss.php?d=64639#p104599>
- Laboratorio 10: <https://virtuale.unibo.it/mod/forum/discuss.php?d=66753#p106746>

B.0.2 Simone Rega

- Laboratorio 08: <https://virtuale.unibo.it/mod/forum/discuss.php?d=63865#p121222>
- Laboratorio 09: <https://virtuale.unibo.it/mod/forum/discuss.php?d=64639#p121220>

B.0.3 Linda Fabbri

- Laboratorio 06: <https://virtuale.unibo.it/mod/forum/discuss.php?d=62579#p103325>
- Laboratorio 08: <https://virtuale.unibo.it/mod/forum/discuss.php?d=63865#p121223>

- Laboratorio 09: <https://virtuale.unibo.it/mod/forum/discuss.php?d=64639#p121226>