Relazione del progetto di “Programmazione ad oggetti”

Christian Remschi, Giorgia Patera, Jago Camoni.

03/06/2024

christian.remshi@studio.unibo.it
giorgia.patera@studio.unibo.it
jago.camoni@studio.unibo.it

INDICE:

Capitolo 1: Analisi

1.1 Requisiti

Il gruppo si pone come obiettivo la creazione di un semplice gioco platform a turni, ispirato dalla celebre serie di giochi dal titolo “Worms”. E’ un gioco sviluppato a turni e in modalità multigiocatore con relative armi e entità influenzate dalla gravità. La particolarità del gioco è la possibilità di costruire la mappa in cui affrontare la partita. L'obiettivo del gioco è rimanere l’unico giocatore in vita, è possibile utilizzare il razzo o la pala per danneggiare i giocatori o eliminare terreno, la differenza tra le due armi è il danno ad aria del razzo a differenza della pala. All’inizio della partita verranno spawnate delle entità casuali, esistono 3 diverse tipologie di entità trappole, che danneggiano il giocatore, medikit , che medicano il giocatore dandogli più vita, e infine una scatola di munizioni.

Requisiti funzionali :

- Il software dovrà gestire correttamente i turni dei giocatori, le relazioni tra loro e la possibilità di vittoria di un giocatore o nessun giocatore.
- Il software dovrà gestire correttamente tutte le entità presenti in gioco, quindi la loro gravità e le loro interazioni con i vari giocatori.
- Il software dovrà gestire le 2 armi presenti nel gioco: razzo e pala e la possibilità di danneggiare il giocatore e di distruggere la mappa.

Requisiti non funzionali:

- Il software dovrà gestire la possibilità di costruire la mappa e tutto ciò che ne comporta.

1.2 Analisi e modello del dominio

Il gioco inizialmente si apre con un piccolo menu, che permette al giocatore di selezionare il numero di giocatori e premere il pulsante play per continuare. Una volta fatto ciò si apre una schermata in cui vengono illustrati i comandi, le regole e dei piccoli consigli per usufruire al meglio del gioco. Premendo il bottone play si potrà iniziare a costruire la mappa, inizialmente viene visualizzata la mappa standard di base ma cliccando sui vari pannelli si potrà costruire una mappa personalizzata. Attenzione che cliccando su un pannello inizierà la modalità creativa e qualunque pannello passato sopra con il mouse verrà selezionato fino al prossimo click con il mouse. Una volta costruita la propria mappa personalizzata bisognerà scegliere dove spawnare i giocatori, e infine premendo finish inizierà definitivamente il gioco. GrubClash è un gioco a turni quindi avrai 12 secondi per muovere un giocatore alla volta. In questa finestra temporale è possibile interagire con gli oggetti e utilizzare, solo una volta a turno, due tipologie di armi: il lanciarazzi e la pala. Il primo esaurisce le munizioni e danneggia le strutture circostanti mentre la seconda funge da vero e proprio attacco corpo a corpo, penetrando anche il terreno. Nella mappa si generano casualmente 3 tipologie di item, tutte soggette a gravità: la trappola (danneggia), il medikit (cura) e la scatola di munizioni (ripristina le munizioni). Infine è stato aggiunto un 4 item, il MobSpawner che passandoci sopra genererà un Mob casuale, tra 3 tipologie di mob diversi con caratteristiche diverse. L’obiettivo del gioco è rimanere in vita più degli altri giocatori.

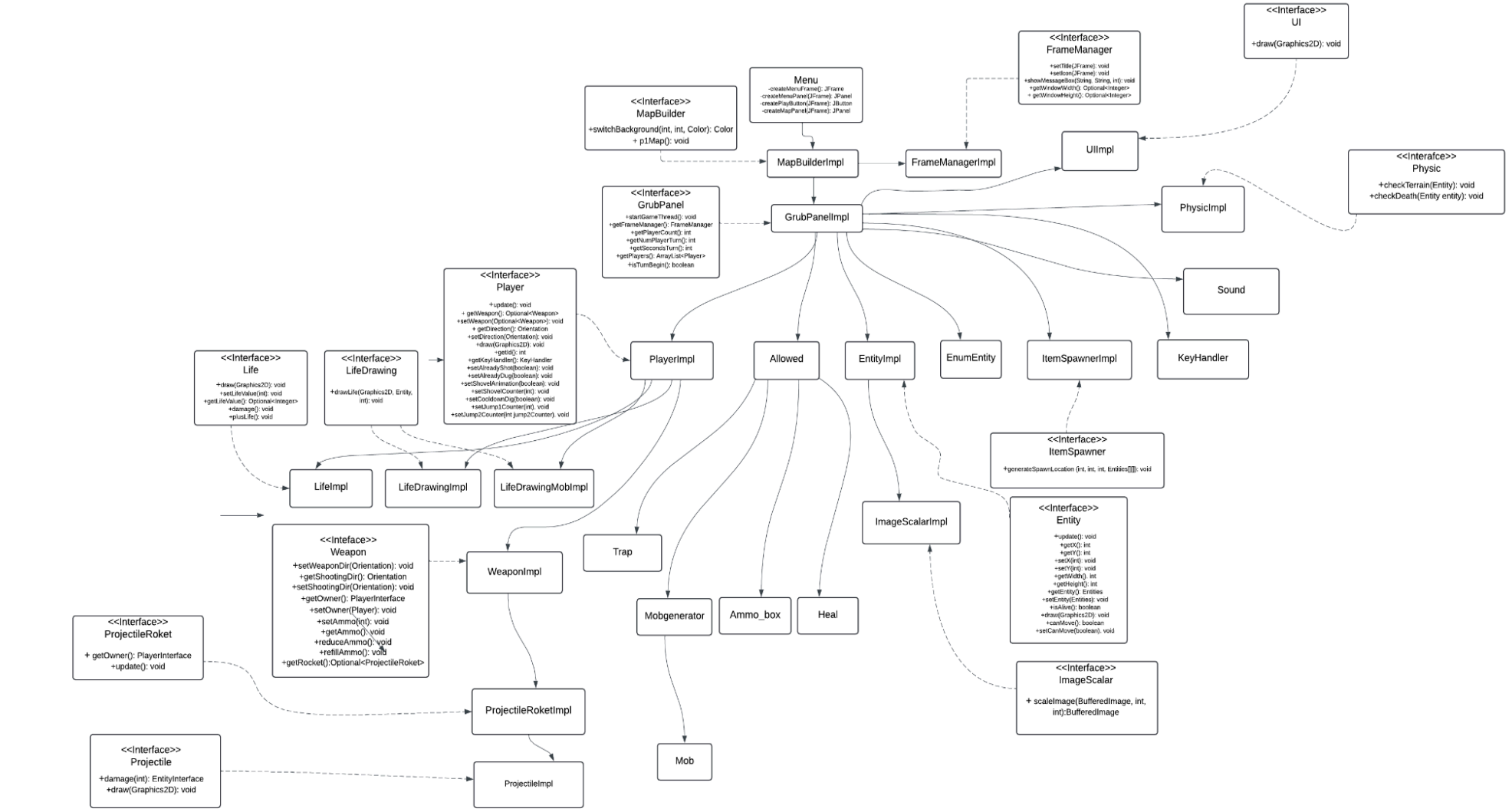
Capitolo 2

Design

2.1 Architettura

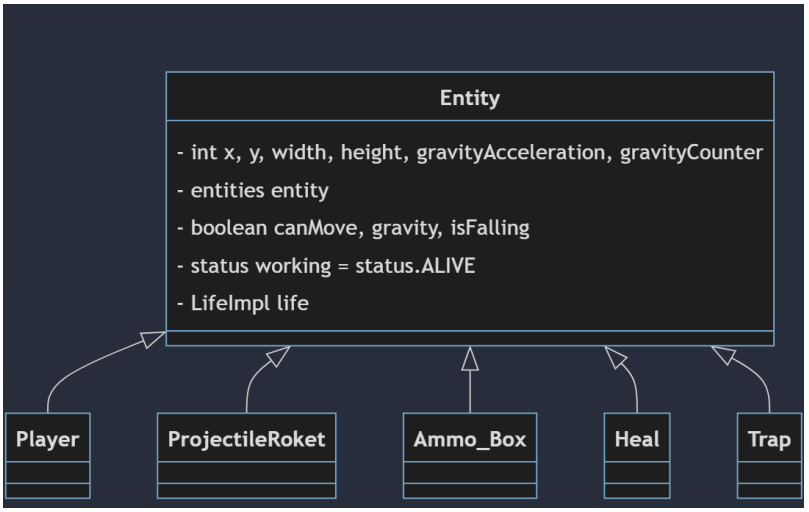
L'architettura del progetto adottata è MVC. Nel Model è presente la logica di gioco, nella View tutto quello che concerne la GUI e la grafica a schermo, mentre il Controller fa comunicare le prime due parti.
Il design non permette la completa separazione di Model, View e Controller.

UML completo pagina successiva.

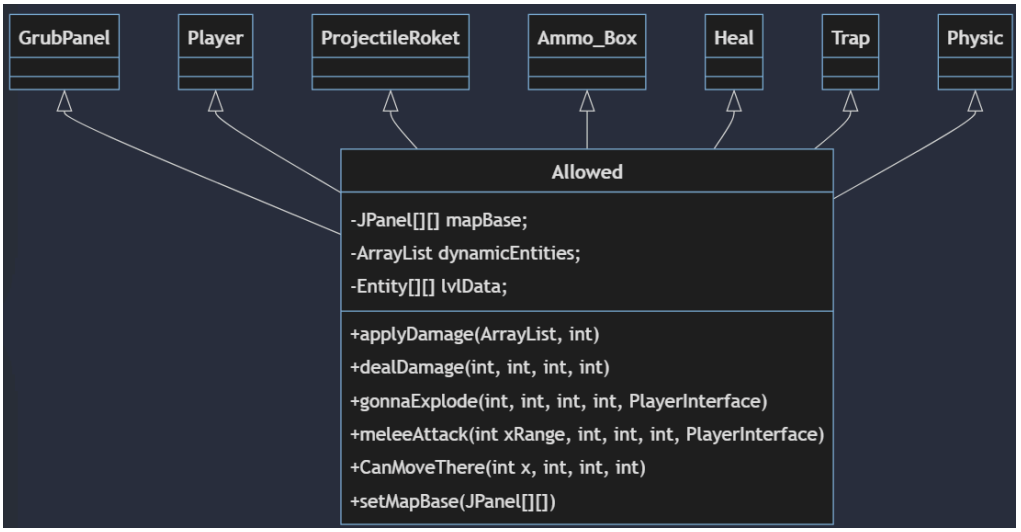

<<Interface>>
UI
+draw(Graphics2D): void
<<Interface>>
FrameManager
+setTitle(JFrame): void
+setIcon(JFrame): void
+showMessageBox(String, String, int): void
+getWindowWidth(): Optional<Integer>
+ getWindowHeight(): Optional<Integer>
Menu
-createMenuFrame(): JFrame
-createMenuPanel(JFrame): JPanel
-createPlayButton(JFrame): JButton
-createMapPanel(JFrame): JPanel
<<Interface>>
MapBuilder
+switchBackground(int, int, Color): Color
+ p1Map(): void
UIImpl
MapBuilderImpl
FrameManagerImpl
<<Interafce>>
Physic
+checkTerrain(Entity): void
+checkDeath(Entity entity): void
<<Interface>>
GrubPanel
+startGameThread(): void
+getFrameManager(): FrameManager
+getPlayerCount(): int
+getNumPlayerTurn(): int
+getSecondsTurn(): int
+getPlayers(): ArrayList<Player>
+isTurnBegin(): boolean
GrubPanelImpl
PhysicImpl
Sound
KeyHandler
<<Interface>>
Player
+update(): void
+ getWeapon(): Optional<Weapon>
+setWeapon(Optional<Weapon>): void
+ getDirection(): Orientation
+setDirection(Orientation): void
+draw(Graphics2D): void
+getId(): int
+getKeyHandler(): KeyHandler
+setAlreadyShot(boolean): void
+setAlreadyDug(boolean): void
+setShovelAnimation(boolean): void
+setShovelCounter(int): void
+setCooldownDig(boolean): void
+setJump1Counter(int): void
+setJump2Counter(int jump2Counter): void
<<Interface>>
Life
+draw(Graphics2D): void
+setLifeValue(int): void
+getLifeValue(): Optional<Integer>
+damage(): void
+plusLife(): void
<<Interface>>
LifeDrawing
+drawLife(Graphics2D, Entity, int): void
PlayerImpl
Allowed
EntityImpl
EnumEntity
ItemSpawnerImpl
<<Interface>>
ItemSpawner
+generateSpawnLocation (int, int, int, Entities[][]): void
LifeImpl
LifeDrawingImpl
LifeDrawingMobImpl
<<Inteface>>
Weapon
+setWeaponDir(Orientation): void
+getShootingDir(): Orientation
+setShootingDir(Orientation): void
+getOwner(): PlayerInterface
+setOwner(Player): void
+setAmmo(int): void
+getAmmo(): void
+reduceAmmo(): void
+refillAmmo(): void
+getRocket():Optional<ProjectileRoket>
WeaponImpl
Trap
ImageScalarImpl
<<Interface>>
Entity
+update(): void
+getX(): int
+getY(): int
+setX(int): void
+setY(int): void
+getWidth(): int
+getHeight(): int
+getEntity(): Entities
+setEntity(Entities): void
+isAlive(): boolean
+draw(Graphics2D): void
+canMove(): boolean
+setCanMove(boolean): void
<<Interface>>
ProjectileRoket
+ getOwner(): PlayerInterface
+update(): void
Mobgenerator
Ammo_box
Heal
<<Interface>>
ImageScalar
+ scaleImage(BufferedImage, int, int):BufferedImage
ProjectileRoketImpl
<<Interface>>
Projectile
+damage(int): EntityInterface
+draw(Graphics2D): void
Mob
ProjectileImpl

2.2 Design dettagliato

Camoni Jago

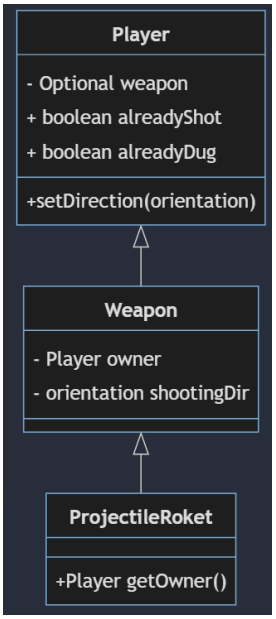


Problema: Abbondanza di oggetti interagibili che hanno un comportamento simile
Soluzione: Usando il Prototype pattern la classe “Entity” si comporta anche come classe astratta. Rappresenta un’entità che può anche fungere da scheletro per altre, che si serviranno di attributi diversi e/o aggiuntivi. I muri vengono considerati Entity allo stesso modo dei player o i proiettili, ma questi hanno qualità aggiuntive.

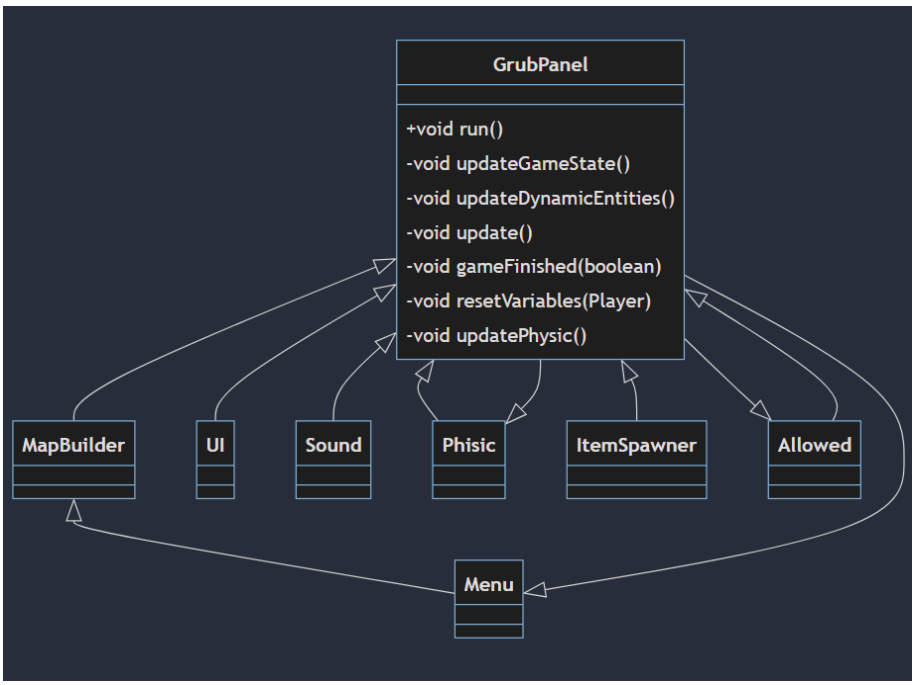


Problema: Una classe è unica e molti dei suoi metodi sono statici
Soluzione: Grazie al Singleton pattern rispecchio le necessità della classe “Allowed”, che contiene le informazioni relative al livello e tutte le strutture contenenti le entità. Si avvale di molti metodi statici.

Remschi Christian

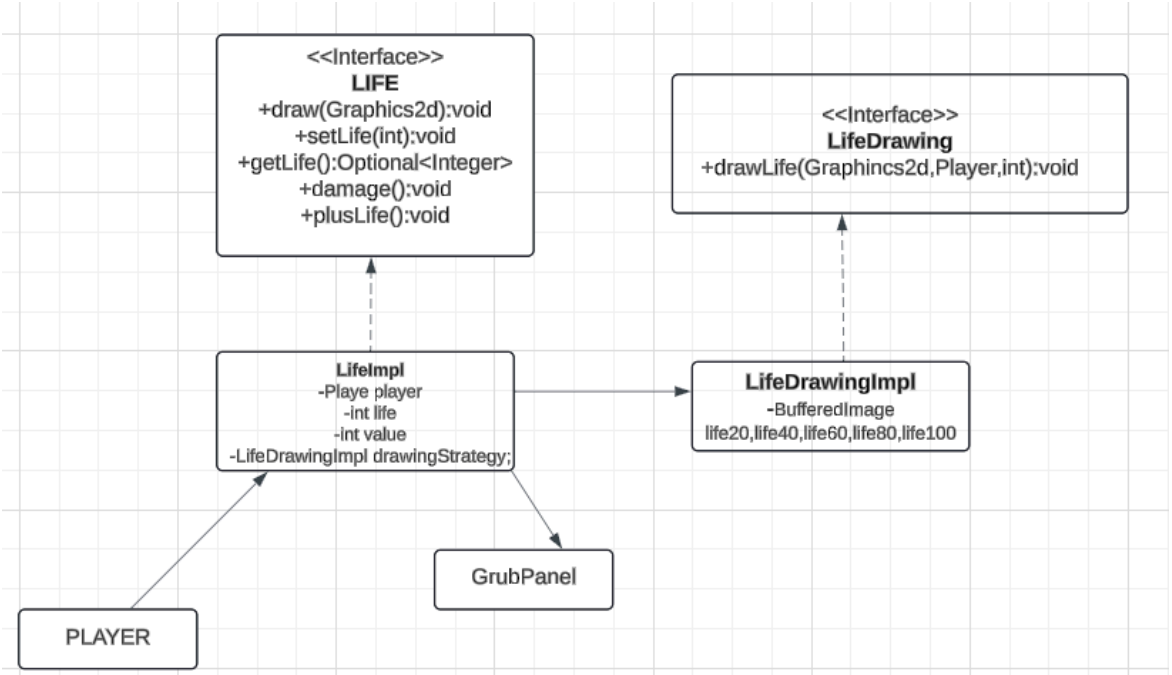


Problema: Necessità di creare un oggetto speciale attraverso il quale è possibile accedere ad altri oggetti unici.
Soluzione: utilizzo la classe Player come wrapper di Entity e al suo interno gestisco le funzionalità necessarie a gestire le interazioni con altre entità e la possibilità di crearne altre (in questo caso i proiettili). Questo con l'utilizzo del pattern decorator.



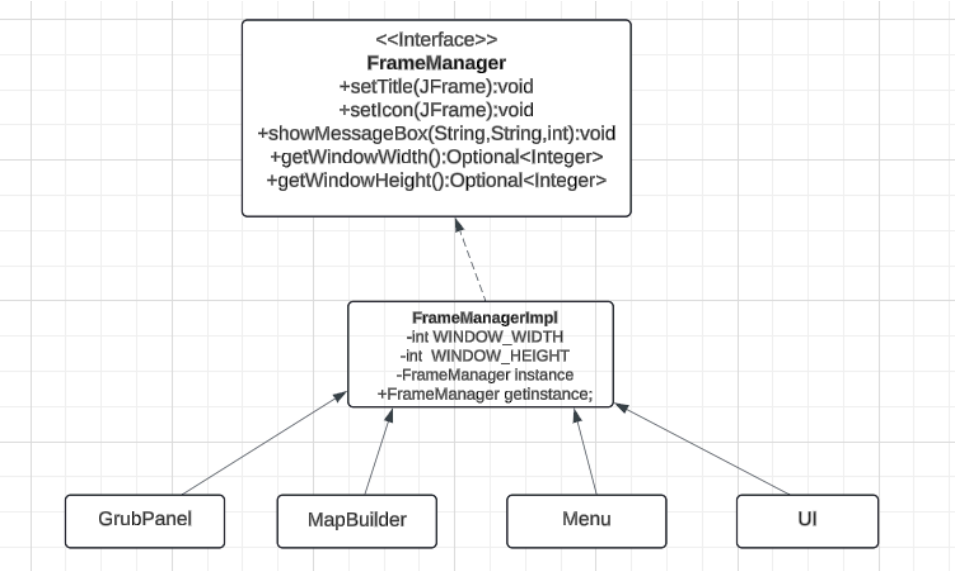
Problema: Dover unire le diverse classi e farle lavorare in modo coeso.
Soluzione: Utilizzo il GrubPanel come mediatore tra le parti, il quale si occupa di mettere al proprio posto i vari pezzi e gestire le loro intersezioni. In questo caso utilizzo il pattern Mediator.

Patera Giorgia



Problema:
Un giocatore ha bisogno di una vita che in base allo scorrimento di gioco si modifica fino ad arrivare anche a una morte.

Soluzione :
Ho gestito il problema della vita creando una classe LifeImpl in cui all'interno ha una variabile intera Life che varia da 0 a 10 che rappresenta la vita del giocatore e in base a tale valore rappresenta un'immagine diversa della vita. Ho pensato che per questo tipo di implementazione sarebbe stato utile utilizzare un Pattern Strategy, che permette di definire una famiglia di algoritmi, incapsularli e renderli intercambiabili. Questo poi permette di poter variare per esempio il disegno della vita indipendentemente dal giocatore. LifeImpl utilizza lo Strategy Pattern per disegnare la vita del giocatore, è possibile creare nuove implementazioni di LifeDrawingImpl per gestire diversi modi di disegnare la vita.



Problema:
Visto l'utilizzo di molti frame in questo progetto e il fatto che la mappa cambia sempre da game a game, c'è l'utilizzo in varie occasioni delle medesime variabili e funzioni che rendono ripetitivo il codice.

Soluzione:
Ho creato una classe FrameManagerImpl con tutte funzioni statiche che risolve il problema del codice ripetitivo. Il Singleton Pattern gestisce una classe che abbia una sola istanza e fornisce un punto di accesso globale a tale istanza.

I vantaggi su questo Pattern sono che fornisce un controllo stretto sull'unica istanza della classe, riduce l'uso della memoria avendo una sola istanza, però con il metodo getInstance() viene chiamato spesso e crea ogni volta una nuova classe di FrameManagerImpl.

Capitolo 3

Sviluppo

3.1 Testing automatizzato

GrubClash verifica attraverso dei test automatizzati la corretta implementazione delle varie componenti di gioco. Utilizzando la libreria JUnit vengono testati alcuni elementi di dominio del sistema:

- EntityTest:
  Testa la creazione delle trappole, heal e ammo box

- RoketTest:
  Testa la creazione, lo sparo e il rifornimento delle armi

- LifeTest:
  Testa la salute dei giocatori quando vengono danneggiati o curati

- PlayerTest:
  Testa la creazione dei player

3.2 Note di sviluppo

Esempio:

Uso di parti della libreria java.awt
https://github.com/Jagocamino/OOP23-GrubClash/blob/7dd5605594452d76584cb8b89e4aa25103d28ff2/src/main/java/it/unibo/grubclash/view/Implementation/LifeImpl.java#L30C5-L36C6
Uso di Optional
https://github.com/Jagocamino/OOP23-GrubClash/blob/bee2e5c721df3654fa17cf1a8959ddc989acab18/src/main/java/it/unibo/grubclash/model/Implementation/Weapon.java#L24C5-L30C6
Uso di ArrayList
https://github.com/Jagocamino/OOP23-GrubClash/blob/bee2e5c721df3654fa17cf1a8959ddc989acab18/src/main/java/it/unibo/grubclash/model/Implementation/Allowed.java#L158C5-L160C56
Uso di Java.sound
https://github.com/Jagocamino/OOP23-GrubClash/blob/bee2e5c721df3654fa17cf1a8959ddc989acab18/src/main/java/it/unibo/grubclash/model/Implementation/Sound.java#L26C6-L35C2
Uso Java.io
https://github.com/Jagocamino/OOP23-GrubClash/blob/bee2e5c721df3654fa17cf1a8959ddc989acab18/src/main/java/it/unibo/grubclash/model/Implementation/Sound.java#L13
Utilizzo delle Lambda expression
https://github.com/Jagocamino/OOP23-GrubClash/blob/bee2e5c721df3654fa17cf1a8959ddc989acab18/src/main/java/it/unibo/grubclash/controller/Implementation/GrubPanel.java#L183C9-L222C20

(Camoni Jago) Codice online come ispirazione, per ricreare la funzione di “touch any key to continue”:
https://stackoverflow.com/questions/19870467/how-do-i-get-press-any-key-to-continue-to-work-in-my-java-code
(Remschi Christian) Delta method per il game loop:
https://stephendoddtech.com/blog/game-design/variable-delta-time-javascript-game-loop#:~:text=const%20currentTime%20%3D%20Date.now()%0Aconst%20deltaTimeMillis%20%3D%20currentTime%20%2D%20gameState.engine1.timePreviousLoop%0Aconst%20deltaTimeSecs%20%3D%20deltaTimeMillis%20/%201000.0%0AGameRuntime.gameState.engine1.timePreviousLoop%20%3D%20currentTime%3B
preso ispirazione per fare
https://github.com/Jagocamino/OOP23-GrubClash/blob/bee2e5c721df3654fa17cf1a8959ddc989acab18/src/main/java/it/unibo/grubclash/controller/Implementation/GrubPanel.java#L100C4-L103C43

Capitolo 4

Commenti finali

4.1 Autovalutazione e lavori futuri

Camoni Jago:
Questo è stato il mio primo progetto su Java. All'inizio ho sottovalutato le ore di lavoro e soprattutto la quantità di nozioni e micro aspetti da considerare (usare git, fare test, la logica a oggetti, il modello..). Pensavo che avremmo avuto tutto il tempo del mondo ma la notizia dell'abbandono di un componente del gruppo ci ha portato a riscrivere i nostri obiettivi dovendo distribuire in tre il lavoro che avremmo dovuto svolgere in 4. Il prodotto finale è incompleto ma pronto alla consegna, volendo saremmo riusciti ad aggiungere anche altre armi, proiettili o item con poche righe di codice dato che la sua struttura ce lo permetteva. Credo di aver superato, di poco, le 70 ore di lavoro. Ho utilizzato abbondantemente quel tempo, aggiungendo poche feature opzionali. Non mi pento di aver scritto, prima di ogni altra implementazione, la parte della costruzione della mappa, inizialmente segnata come funzione aggiuntiva. Probabilmente non avrei saputo come aggiungerla se l'avessi fatto dopo. Ho avuto pochi battibecchi con il gruppo, mi sono trovato bene. Sono stato indeciso riguardo i comportamenti di alcune classi, cosa che ha rallentato il nostro lavoro perché cambiavo spesso idea creando lunghe discussioni. Siamo andati avanti con l'idea "Committo solo se funziona ogni cosa" che ci ha portato a lavorare su un unico branch. Pur dovendo risolvere alcuni merge conflict nessuno ha sentito la necessità di creare branch aggiuntivi. Ho provato a lavorare su "dev" (diverso dal master), ma essendo abituato a lanciare pochi commit quel branch non è stato praticamente utilizzato. Considero questo progetto una grande esperienza. Ho imparato a utilizzare meglio i forum e so con più chiarezza dove trovare le risposte ai miei problemi.

Patera Giorgia:
Guardando oggettivamente il mio lavoro ci sono molti aspetti che potrebbero essere migliorati e in cui si potrebbero applicate soluzioni migliori, perciò non posso ritenermi soddisfatta di me stessa e del codice che ho scritto. Essendo il mio primo progetto in java per quanto riguarda la correttezza del codice ma soprattutto la generalità del codice e l'utilizzo dei relativi pattern, il codice sarebbe potuto essere scritto in maniera migliore.

Per quanto riguarda il lavoro di gruppo e la relazione con i miei compagni mi ritengo molto soddisfatta, perchè in caso di blocco ci siamo sempre aiutati a vicenda e molte classi ci siamo trovati per realizzarle insieme.
Questo progetto di lavoro di gruppo utilizzando il linguaggio java, mi ha dato senza dubbio un'esperienza significativa e molte nuove conoscenze. Ogni problema nel gioco e nell'esperienza in generale mi ha permesso di approfondire e migliorare le mie conoscenze.
In fin dei conti questo progetto mi ha preso più tempo di quanto mi aspettassi, un po' dato dalla inesperienza un po' dato dal fatto che l'abbandono di un componente del gruppo ha complicato la consegna del progetto. Ho lavorato a questo progetto più o meno 70 ore, il risultato finale non è completo ma secondo me adatto alla consegna, in futuro sarà possibile volendo implementare nuove armi o aggiustare il codice senza problemi.

Remschi Christian:
Come primo progetto in linguaggio Java posso ritenermi soddisfatto, di sicuro ci sono migliorie possibili in ogni punto del codice ma allo stesso tempo ritengo sia accettabile in quanto, ogni riga che abbiamo scritto del codice, sia uscita quasi del tutto dalla nostra testa, provando, testando, riflettendoci e uscendo fuori con la migliore soluzione che ci veniva in mente. Però, non avendo seguito inizialmente uno schema ben fissato, abbiamo iniziato ad abbozzare classi che poi hanno trovato difficoltà a interagire l'una con le altre, ma poi con un pò di impegno tutto ha cominciato a funzionare. Questa esperienza mi ha sicuramente insegnato molto, tra lo prendere più confidenza con il linguaggio, allo localizzare più velocemente i problemi e le rispettive soluzioni, oltre che avermi donato una vista, seppur limitata, del lavoro di gruppo in questo ambito.
Per quanto riguarda la relazione con il gruppo non ho nulla di cui lamentarmi, ci si è sempre venuti in contro o comunque trovato compromessi, i problemi venivano risolti insieme se particolarmente impegnativi e la disponibilità c'è sempre stata, peccato solo per l'abbandono del quarto componente che ha un attimo complicato la situazione, ma che poi si è risolta. Il quantitativo di ore dato a questo lavoro di gruppo è stato leggermente superiore alle 70 ore per far sì che completassimo le implementazioni da noi prestabilite, anche se inizialmente ideate per 4 membri. In futuro non ci saranno problemi nell'aggiungere nuove armi o item, per come abbiamo gestito queste ultime.

4.2 Difficoltà incontrate e commenti per i docenti

Camoni Jago:
Ho iniziato il progetto senza una struttura precisa in testa, questo mi ha portato a rivedere molte volte il mio codice. La parte “abbandonata” di più a sé stessa è quella del MapBuilder. Funge a tutti gli effetti da God class per le prime sequenze di gioco, ignorando l’architettura MVC. Scrivere una classe in grado di gestire la costruzione della mappa e il piazzamento dei personaggi, pur non essendo molto comprensibile da osservatori esterni, è stata un pretesto per dare una struttura al resto del progetto. MapBuilder si occupa di colorare la tabella, necessaria unicamente all’inizio del gioco. Una disorganizzazione simile si può anche trovare in Allowed, anche se presenta elementi più chiari grazie all’ausilio del pattern Singleton.

Patera Giorgia:
La realizzazione di questo progetto è stata un esperienza in cui ho imparato tanto come l’utilizzo di Git Hub e diversi meccanismi della programmazione ad oggetti in particolare Java. Durante la realizzazione di questo progetto ho incontrato diverse difficoltà che mi sembravano impossibili da superare ma anche con l’aiuto dei miei compagni siamo riusciti a risolvere tutti gli imprevisti senza troppi problemi. Essendo il mio primo software realizzato da zero in java purtroppo la mia programmazione non sfrutta a pieno le caratteristiche di java.

Appendice A

Guida utente :

Avviata, l'applicazione mostra un menu principale contenente:

- La selezione a tendina dove verrà impostato il numero dei giocatori
- Un pulsante arancione di avvio

Al click apparirà una guida riassuntiva a schermo con un pulsante uguale a quello cliccato in precedenza.

Premuto, un'altra finestra si sostituirà a quella principale. Siamo nella Fase 1, quella del disegno della mappa. Il giocatore che disegna la mappa sarà anche l'ultimo a svolgere il turno. La tabella 20x20, inizialmente vuota, può essere colorata al passaggio del mouse. Con il tasto sinistro si attiva/disattiva la modalità disegno. Disegnata la mappa, lo stesso giocatore preme il pulsante "Finish" in alto a destra.

Arriviamo alla Fase 2, dove il primo giocatore piazza lo spawnpoint del proprio personaggio e aziona il bottone "Finish". Quando l'ultimo giocatore (quello che ha disegnato la mappa) finisce di piazzare il suo personaggio, la mappa e le immagini si caricano. Inizia la Fase 3.

Ora ogni player (partendo dal primo) ha 12 secondi per muovere la propria larva. In questa finestra temporale è possibile interagire con gli oggetti e utilizzare, solo una volta a turno, due tipologie di armi: il lanciarazzi e la pala. Il primo esaurisce le munizioni e danneggia le strutture circostanti mentre la seconda funge da vero e proprio attacco corpo a corpo, penetrando anche il terreno. Nella mappa si generano casualmente 3 tipologie di item, tutte soggette a gravità: la trappola (danneggia), il medikit (cura) e la scatola di munizioni (ripristina le munizioni).

Le uniche direzioni di mira sono sopra, sotto, destra e sinistra. Se il player è fermo mira sotto, l'azione di salto mira sopra. Ci si può danneggiare da soli. Per azionare il razzo premere "Enter", per la pala "F". Ci si muove con "A" e "D", il salto è "Spacebar"

L'ultimo che sopravvive vince