UNIVERSITÀ DEGLI STUDI DI URBINO CARLO BO

Anno Accademico 2023/2024

Progetto d'esame di
Programmazione e Modellazione a Oggetti:
PROGETTO FARFALLA

Prof.ssa Sara Montagna

Studente: Dimitri Cioppi matr. 292909

Specifica

Scopo

Lo scopo del progetto consiste nel creare un semplice videogame arcade bidimensionale con una farfalla come personaggio giocante che ha lo scopo di raccogliere vari pickups presenti nel mondo di gioco prima che di essere uccisa dai nemici che compariranno proseguendo la partita.

Oggetti di gioco

Il mondo di gioco può contenere al suo interno vari tipi di oggetti divisi in due categorie principali: **entità, pickup** e alcuni oggetti speciali.
Le **entità** sono tutti quegli oggetti che si muovono all'interno del mondo di gioco. Essi sono:

1. **Farfalla:** la protagonista del gioco, pilotata dal giocatore è l'unica entità in grado di raccogliere i pickups presenti nel mondo di gioco.
2. **Nemici**: sono quelle entità che a contatto con la farfalla le fanno perdere una vita e poi scompaiono; ci sono tre tipi nemici con tre movimenti diversi:
    a. **Pipistrelli neri**: si muovono in linea retta e rimbalzano quando toccano i confini del mondo di gioco.
    b. **Libellule:** vengono generate da un pickup speciale, si muovono come i pipistrelli neri ma cambiano direzione costantemente in modo simile ad un insetto.
    c. **Pipistrelli rossi**: il nemico più pericoloso, segue lentamente la farfalla.
3. **Killer**: il killer è un'entità generata da un pickup con lo scopo di uccidere un nemico presente nel gioco: esso, quando viene generato, bersaglierà il nemico più vicino alla farfalla e lo seguirà finché non lo avrà raggiunto; se il nemico in questione dovesse morire prima di essere raggiunto, il killer continuerà a muoversi come un pipistrello nero finchè non toccherà un nemico.
   Se il killer tocca un nemico, sia il killer che il nemico spariranno dal mondo di gioco.

I **Pickups** sono tutti quegli oggetti che possono essere raccolti dalla farfalla; essi sono:

1. **Pickup base**: il pickup base del gioco è un fiore blu che quando raccolto dà il punteggio base.
2. **Extra:** è un fiore rosso che garantisce più punti.
3. **Bomba:** genera un killer quando viene raccolta.
4. **Vita extra**: è un cuore che dà una vita extra.
5. **Stella:** oggetto raro che dà molti punti se raccolto.
6. **Maledizione**: garantisce molti punti ma evoca una libellula nella posizione in cui è stata raccolta.

Infine ci sono 2 oggetti speciali che non sono né entità ne pickups; essi sono:

1. **Evocazioni:** sono un effetto visivo che serve ad avvertire il giocatore alcuni secondi prima che in quel punto del mondo di gioco sta per essere generato un nemico.
2. **Bottoni**: non vengono usati nel mondo di gioco ma nelle schermate del **menù**, **aiuto** e **game over**.

Partita

La partita comincia con la farfalla al centro del mondo di gioco e quattro pickups basilari generati in posizioni casuali all'interno del mondo.
Ogniqualvolta un pickup basilare viene raccolto, una nuova istanza di esso verrà generata in un’altra coordinata casuale: inoltre, se un pickup di una qualunque tipologia viene raccolto, un contatore viene incrementato di 1.
In base all'incremento del contatore vengono creati tutti gli altri pickups e i nemici.
La farfalla inizia il gioco con 3 vite; il gioco finisce quando le vite finiscono.

Requisiti

Requisiti Funzionali

- Il programma deve essere in grado di gestire un mondo di gioco bidimensionale dotato di più oggetti al suo interno.

- Il programma deve essere in grado di gestire il movimento di diverse entità all’interno del mondo di gioco che avranno tra di loro movimenti molto diversi.

- Il programma avrà anche diversi eventi che serviranno per gestire l’interazione che gli oggetti nel mondo di gioco avranno tra di loro e anche il cambio delle varie interfacce della GUI del programma.

- Il programma avrà 4 interfacce: menu, help, playing e game over.

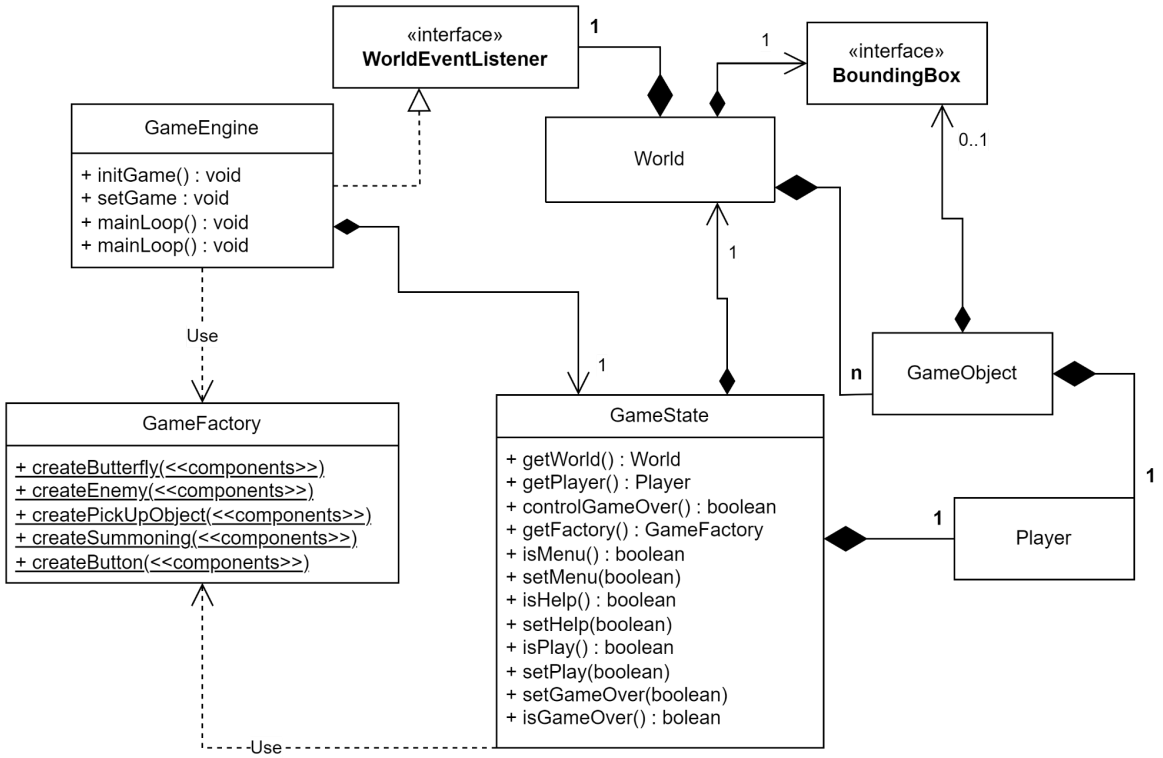
- L’utente potrà navigare queste interfacce cliccando su appositi bottoni.

- Il programma deve essere in grado di gestire i vari input dell'utente, sia attraverso il mouse, quando si clicca sui bottoni, sia attraverso la tastiera quando viene pilotata la farfalla.

- Tutti gli oggetti all’interno del mondo di gioco avranno uno sprite e una componente grafica che sarà responsabile dell’animazione di tale sprite.

Analisi del Modello

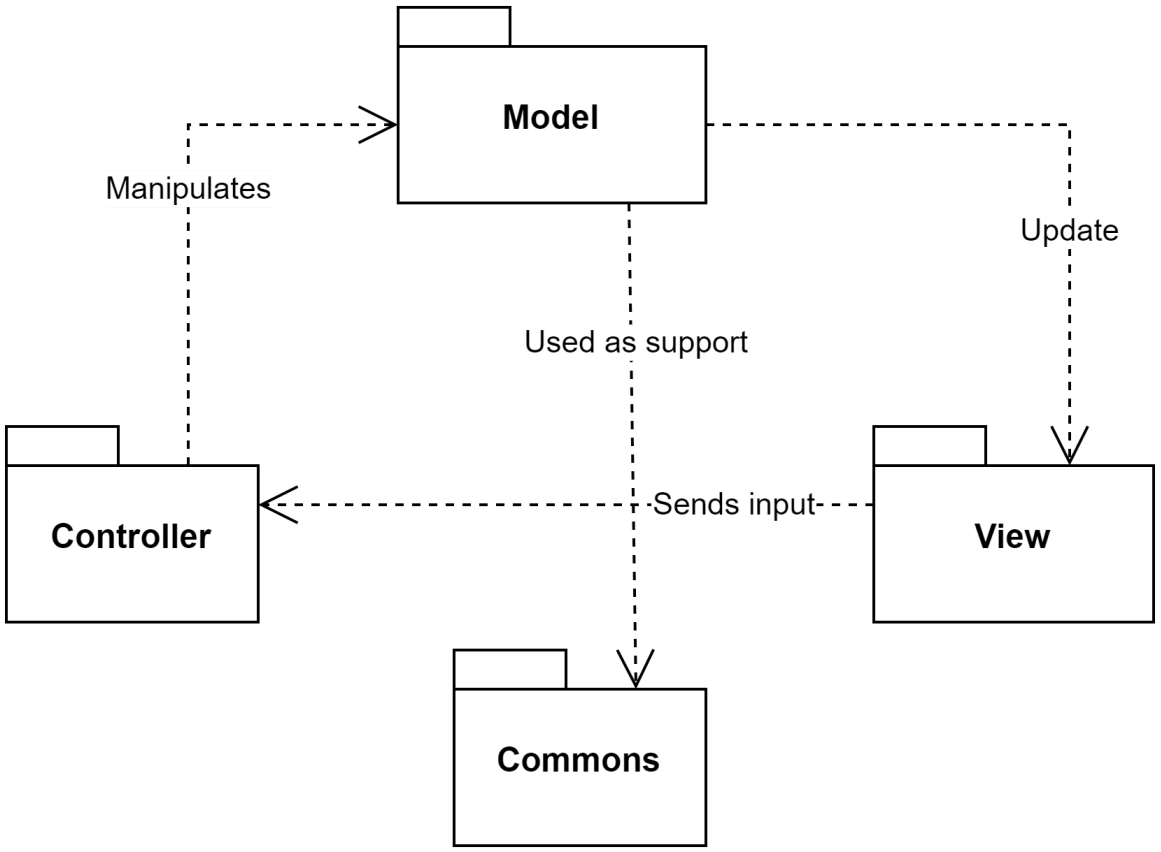
Il programma è un videogame dinamico che presenta vari eventi quindi bisognerà creare un motore di gioco che mandi avanti il gioco e che gestisca i vari comandi ed eventi nel gioco.
Inoltre sarà necessario avere un mondo dove gli oggetti possano esistere: questo mondo conterrà tutti gli oggetti di gioco necessari nei vari momenti.
Il gioco dovrà avere anche uno stato per ognuna delle 4 interfacce sopra menzionate e quindi sarà necessaria anche una classe che gestisca i vari stati del gioco e che comunichi con il motore di gioco.
Infine i vari oggetti di gioco sono piuttosto complessi e dovranno dunque essere composti da vari componenti che rappresenteranno le varie proprietà dell'oggetto in questione.
Sarà inoltre necessaria una classe che generi gli oggetti di gioco quando necessario.



Questo é uno schema UML dell'analisi del problema: si nota che in esso vi sono rappresentate solo le entità principali del progetto in maniera sintetica.
Si andrà più nel dettaglio in seguito quando si descriveranno le singole entità e le loro interazioni.

Design

Analizzate le varie necessità del programma, si è giunti alla conclusione che i design pattern **STRATEGY, STATIC FACTORY, GAME LOOP, COMMAND, OBSERVER, EVENT QUEUE** e **COMPONENT** risultano particolarmente adeguati per lo sviluppo di questo progetto e verranno quindi implementati nella scrittura del programma.
Inoltre, per motivi di ordine e chiarezza, il programma verrà progettato seguendo il pattern **MVC** che garantirà un design più chiaro ed elegante.



Come si può vedere nello schema il progetto è programmato seguendo una versione molto semplice del pattern MVC.
Il programma è infatti suddiviso principalmente nelle tre macro sezioni (qui rappresentate come package perché sono package anche nel programma) del pattern MVC che interagiscono in modo ciclico fra di loro: la View invia input al Controller, quest'ultimo li elabora e in base ai risultati comanda il Model che a sua volta aggiorna la View affinché mostri a schermo gli effetti di tali input.
La quarta parte infine, sono invece i Commons che sono chiamati per rappresentare le posizioni e velocità dei vari oggetti di gioco presenti nel mondo di gioco che è a sua volta contenuto nel Model.
Le macroparti Model, View, Controller sono a loro volta divise in package in base alla funzione delle sottoparti di ognuna di esse; questa suddivisione è stata fatta per motivi di leggibilità e struttura in modo tale da semplificare la lettura e quindi la modifica dell'intero progetto.

Architettura

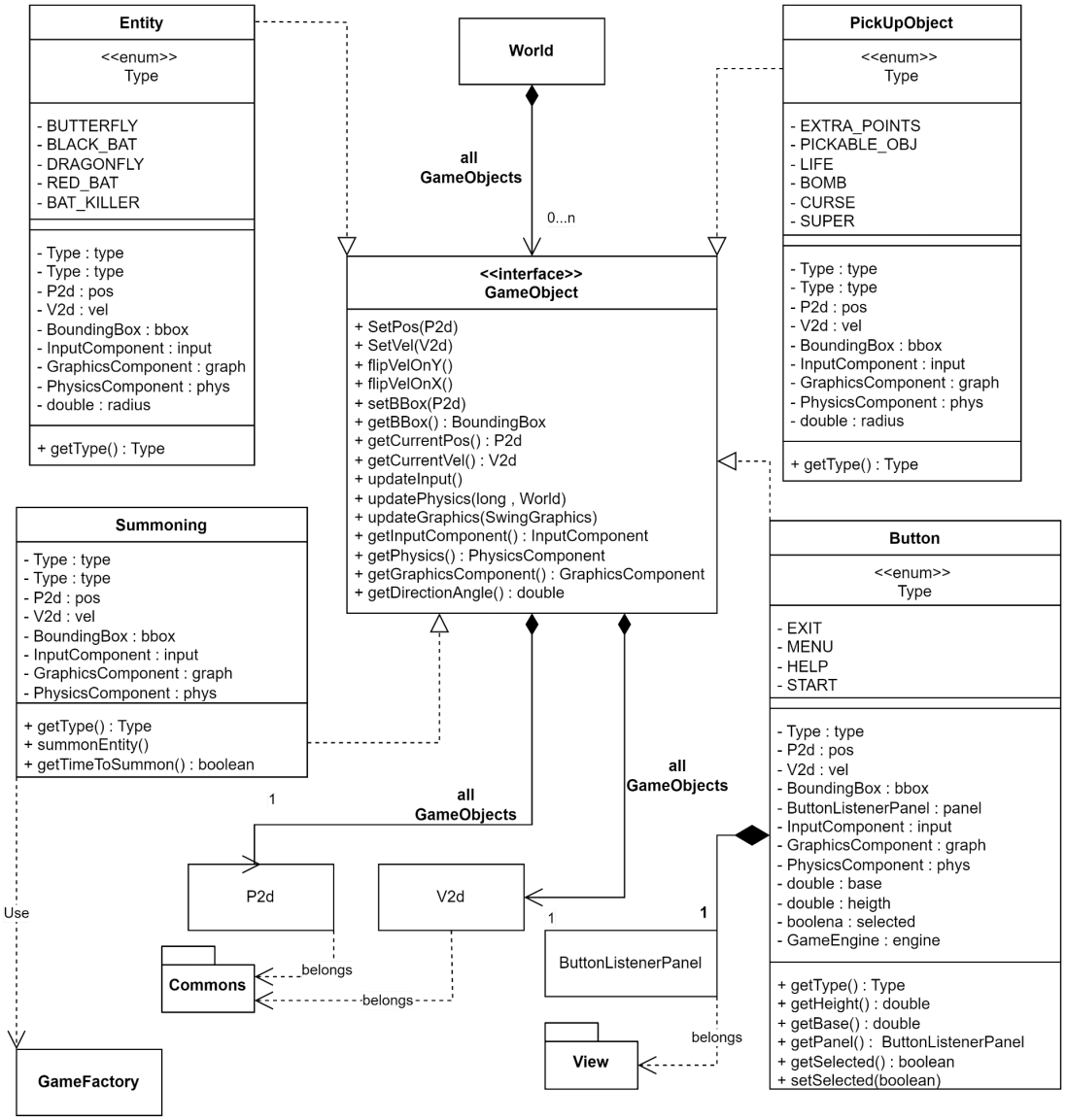
Model

Il Model del progetto è suddiviso in tre package: Game, Inputs e World.
La prima parte contiene il GameEngine, il GameState e tutti i GameObject nonché la GameFactory e la classe Player: praticamente contiene tutte quelle entità che interagiscono con il gioco oppure che lo costituiscono e ne dettano lo stato.
Gli Inputs o InputComponent sono invece coloro che dicono ai GameObject come comportarsi nel mondo di gioco e possono rappresentare un comportamento automatico oppure conseguente ad un input dell'utente ed in tal caso lavorano in tandem con i Controller.
Infine World contiene la classe World che rappresenta il mondo di gioco; inoltre contiene gli Events, le Physics e i Bounds che dettano come e quando gli oggetti di gioco interagiscono fra di loro.
in seguito sarà esaminata ognuna di queste singole classi e parti.

Game

Il package Game è suddiviso in due parti: gli oggetti di gioco contenuti nel package GameObjects e le classi GameEngine, GameState, GameFactory e Player.
I GameObjects, come il nome implica, sono semplicemente tutte le entità che possono essere contenute nel mondo di gioco rappresentato dalla classe World.
Essi sono suddivisi in 4 classi: Entity, PickUp, Summoning e Button che a loro volta implementano l'interfaccia GameObject.
Entity sono tutte quelle entità che possono muoversi nel mondo di gioco: la farfalla, i 3 tipi di nemici e i killer che uccidono i nemici.
Gli oggetti PickUp sono invece tutti quegli oggetti che possono essere presi dalla farfalla quali pickups normali, punti extra, bombe, vite, maledizioni e stelle.
Le Summoning o evocazioni hanno invece il compito di evocare i nemici dopo un delay di 2 secondi; hanno un effetto puramente visivo e non interagiscono con niente; servono principalmente ad avvisare il giocatore dove verrà evocato il nemico per evitare lo spiacevole inconveniente di essere improvvisamente colpiti da un nemico che viene creato troppo vicino alla farfalla.
I Buttons infine sono oggetti speciali che interagiscono con il mouse: essi sono i bottoni che bisogna cliccare per navigare le schermate di Menù, Help e Game Over.
Nella pagina successiva vi è uno schema UML per illustrare la struttura dei GameObjects.

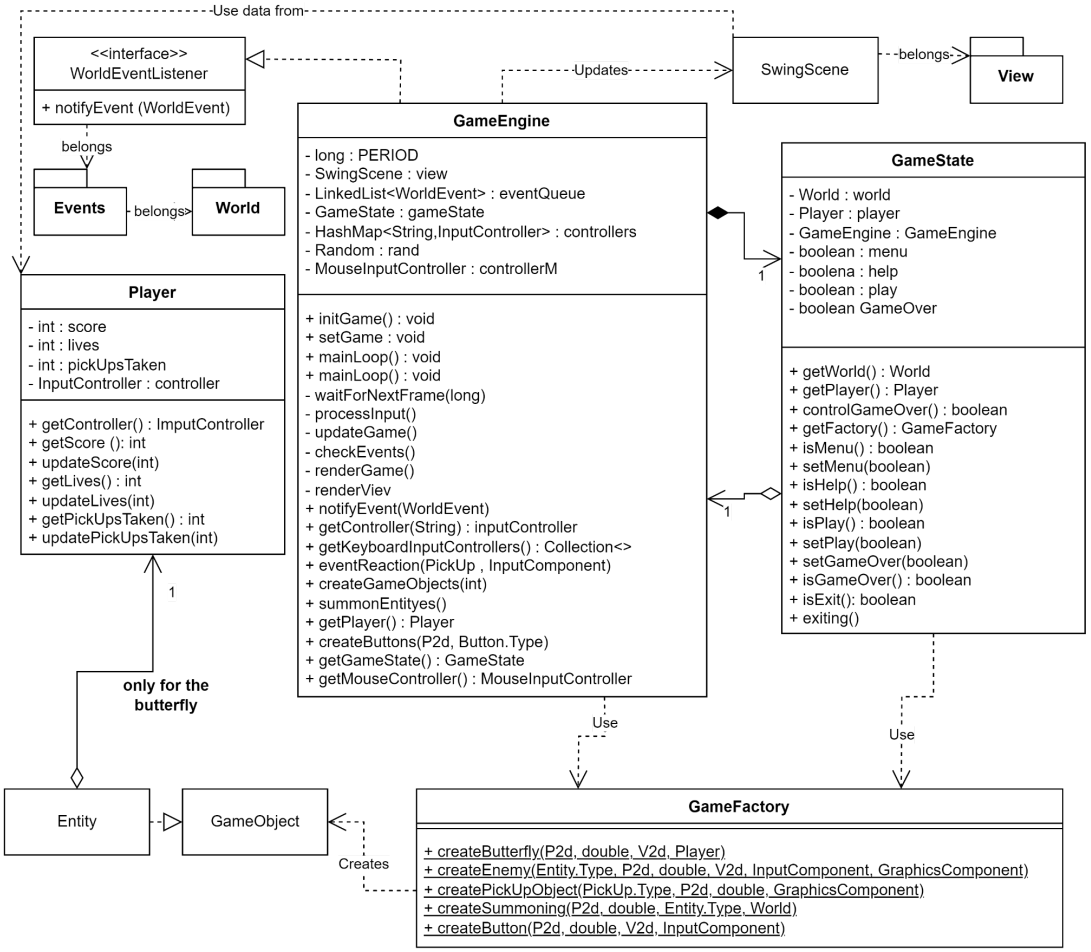
GameObjects



Come si può vedere tutti gli oggetti implementano l'interfaccia GameObject e di conseguenza tutti i metodi presenti nell'interfaccia sono implementati anche nelle classi che la implementano (sottinteso nel diagramma per motivi di leggibilità): naturalmente tali implementazioni differiscono da classe a classe e si vedrà spesso questa struttura basata sul pattern Strategy in quanto è molto utile per l'intercambiabilità dei vari componenti e oggetti.
Inoltre tutti gli oggetti, eccetto Summoning, hanno anche un Enum pubblico interno: questo Enum rappresenta i tipi che possono esistere di quel particolare oggetto; a seconda del tipo, un oggetto può interagire e comportarsi in maniera diversa con il mondo di gioco e con gli altri oggetti.
Si è scelto di differenziare i tipi con un Enum in quanto dal punto di vista della classe non c'è alcuna differenza nel codice fra due tipi dello stesso oggetto dal momento che le interazioni e comportamenti dei vari oggetti sono gestiti dal GameEngine rendendo la creazione di eventuali classi derivate completamente ridondante.
Altro dettaglio da notare è il fatto che tutti i GameObjects sono composti da un P2d e un V2d: queste due classi, appartenenti ai Commons, rappresentano rispettivamente la posizione e il vettore velocità di una particolare entità presente nel mondo di gioco; naturalmente gli oggetti che non dovrebbero muoversi presentano una velocità nulla.
Infine la classe ButtonListenerPanel serve per le interazioni con il mouse: sarà descritta quando si tratterà la View.

Game Engine

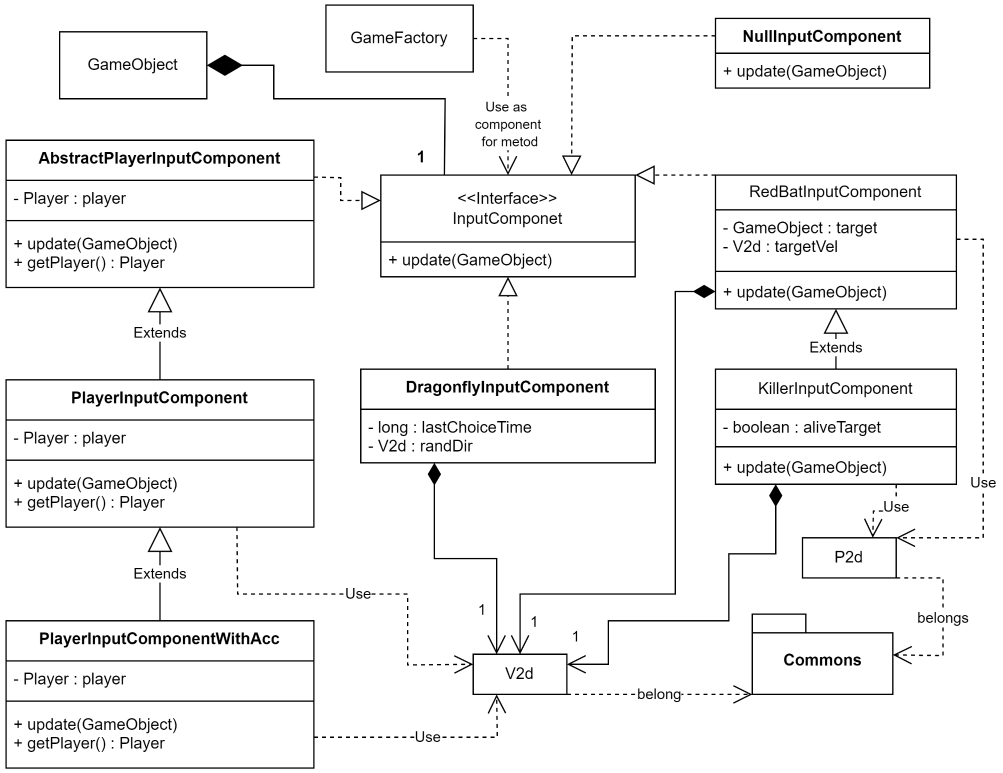
Il GameEngine è letteralmente il motore di gioco con il compito di gestire tutte le interazioni fra gli oggetti nel mondo di gioco e aggiornare la View dei vari cambiamenti che occorrono durante la partita, facendo avanzare il gioco. È infatti la classe responsabile per il Game Loop e per questo motivo ne esiste una sola istanza per avvio di programma; questa è anche la classe incaricata di inizializzare tutto all'avvio del programma.
Il GameState, invece, è la classe che rappresenta i vari stati di gioco e decide se il programma è in stato di Menu, Playing, Help, Game Over e Exit; è anche la classe che inizializza il mondo di gioco all'avvio di ogni partita.
La GameFactory si occupa di creare tutti gli oggetti di gioco; non ha istanza e ogniqualvolta si vuole creare un GameObject si chiama staticamente questa classe secondo il design Static Factory.
La classe Player infine, è una classe speciale che è legata solo alle Entity di tipo BUTTERFLY, la farfalla; questa classe tiene conto del punteggio e del numero di vite associati a quella farfalla. La ragione per cui tali dati non sono associati direttamente alla farfalla è perchè, ad ogni cambio di stato, tutti gli oggetti del mondo di gioco vengono distrutti e servono quei dati per visualizzare le vite e il punteggio nelle schermate di Playing e Game Over.
Qui sotto vi è un diagramma UML per visualizzare le interazioni fra classi.



Nello schema si vede il Game Engine implementare l'interfaccia WorldEventListenener: questo perché il GameEngine è la classe responsabile del controllo di tutte le interazioni tra oggetti ed è quindi il listener di tutti gli eventi possibili che verranno lanciati dall'interazione di due GameObject che, come specificato nel design EventQueue, continueranno la loro esecuzione indipendentemente.

Inputs

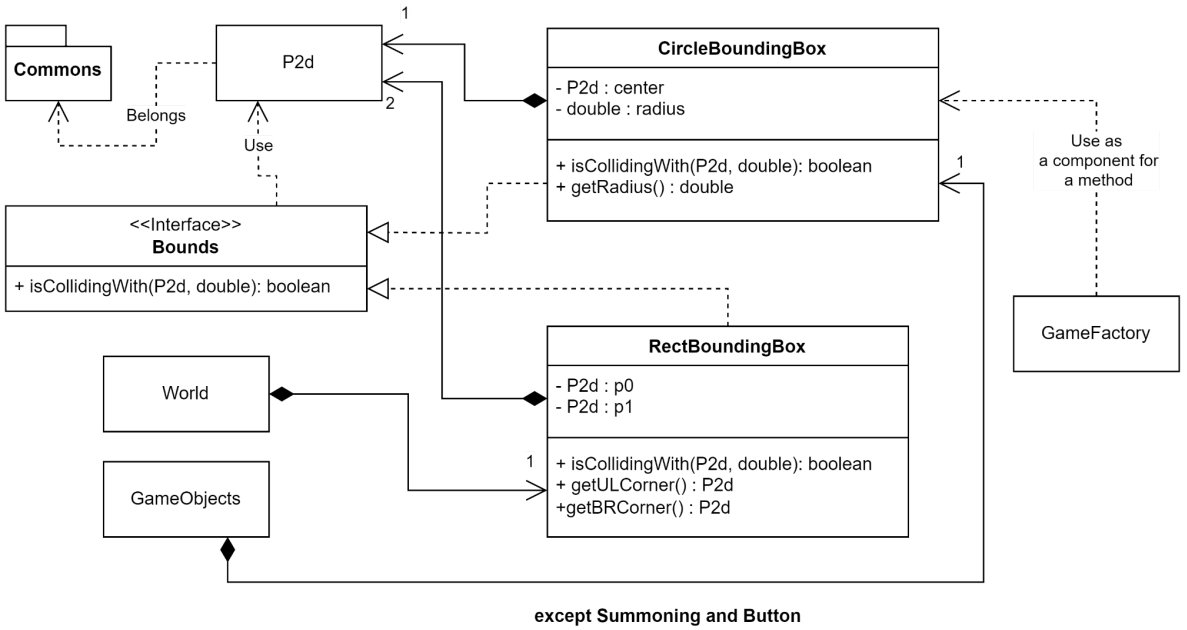
Questo package è chiamato così poiché contiene tutti gli InputComponent necessari per il funzionamento dei GameObjects.
Un input component determina il comportamento di un oggetto nel mondo di gioco per la precisione decide se l'oggetto deve poter muoversi nel mondo di gioco e, se sì, come e perchè.
Lo schema sotto riportato aiuterà a spiegare la struttura e la natura di ognuno di questi componenti.



Come si può vedere, tutti gli input component implementano l'interfaccia InputComponent poiché i GameObjects hanno componenti diversi in base al loro tipo e quindi i loro metodi costruttore hanno bisogno di prendere un oggetto generico di tipo InputComponent che cambierà in base al tipo di oggetto che la GameFactory deve costruire, perché la scelta dell'input component viene fatta nel momento della creazione dell'oggetto.
Questa struttura segue il principio del pattern Strategy perché tale pattern è necessario all'implementazione del pattern Component, necessario a sua volta per la creazione degli oggetti.
In generale tutti gli oggetti che non devono muoversi e le entità di tipo BLACK_BAT, che non hanno bisogno di cambiare direzione autonomamente, avranno un NullInputComponent che semplicemente non farà nulla quando il metodo update verrà chiamato.
Invece ogni oggetto che deve muoversi ha un component personale con un nome che coincide con il nome dell'oggetto a eccezione della farfalla che utilizzerà il PlayerInputComponentWithAcc che prenderà gli input passati dal controller e li trasformerà in una accelerazione della farfalla verso la direzione desiderata: questa classe estende PlayerInputComponent che fa la stessa cosa ma setta la velocità della farfalla verso la direzione selezionata; in questo caso PlayerInputComponent viene semplicemente usata come base per la classe figlia esattamente come AbstractPlayerInputComponent (astratta).
Tutti gli InputComponent non nulli usano e sono composti da un elemento V2d e nel caso del RedBatInputComponent e del KillerInputComponent che lo estende viene utilizzata anche la P2d del bersaglio (la farfalla per il primo e un nemico per il secondo) in quanto necessaria per permettere alle due Entity di inseguire i loro bersagli.

World

Il package World contiene la classe World e altri tre sottopackage Bounds, Events e Physics.
Bounds contiene le classi CircleBoundingBox e RectBoundingBox: la prima rappresenta un confine circolare e viene usata da tutti i GameObject che devono interagire con il mondo mentre la seconda è un confine quadrato e rappresenta il confine del mondo di gioco (appartenente alla classe World); queste due classi implementano l'interfaccia BoundingBox come mostrato nello schema sottostante.

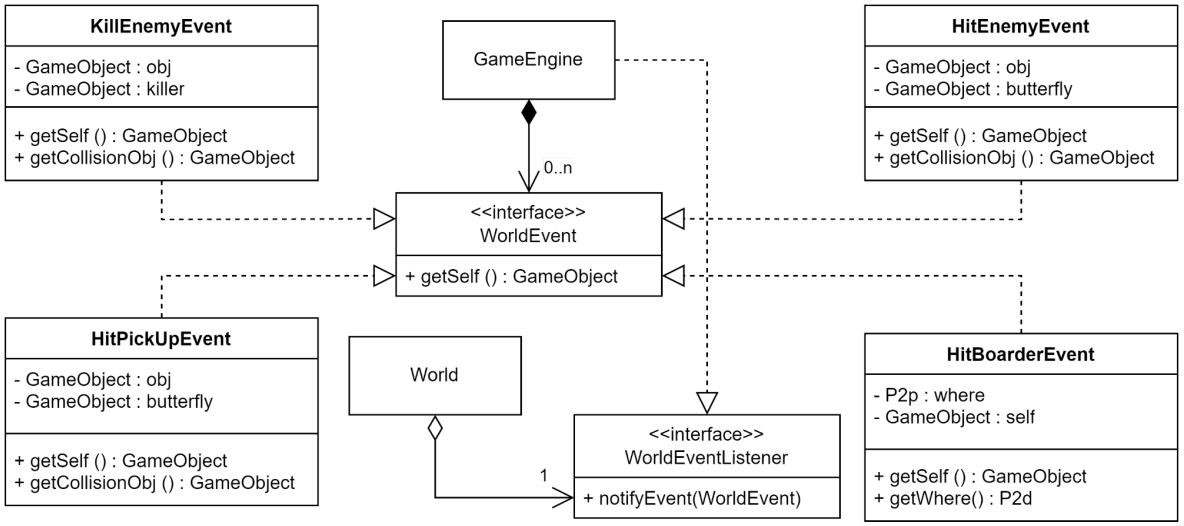


except Summoning and Button

Queste due classi servono per permettere agli oggetti di interagire tra di loro e con il mondo di gioco. Infatti queste classi rappresentano l'area o l'insieme di coordinate entro cui è possibile interagire: se due BoundingBox condividono almeno in parte le stesse coordinate allora vuol dire che che gli oggetti composti da quei due BoundingBox si stanno toccando e quindi stanno interagendo fra di loro.

Events

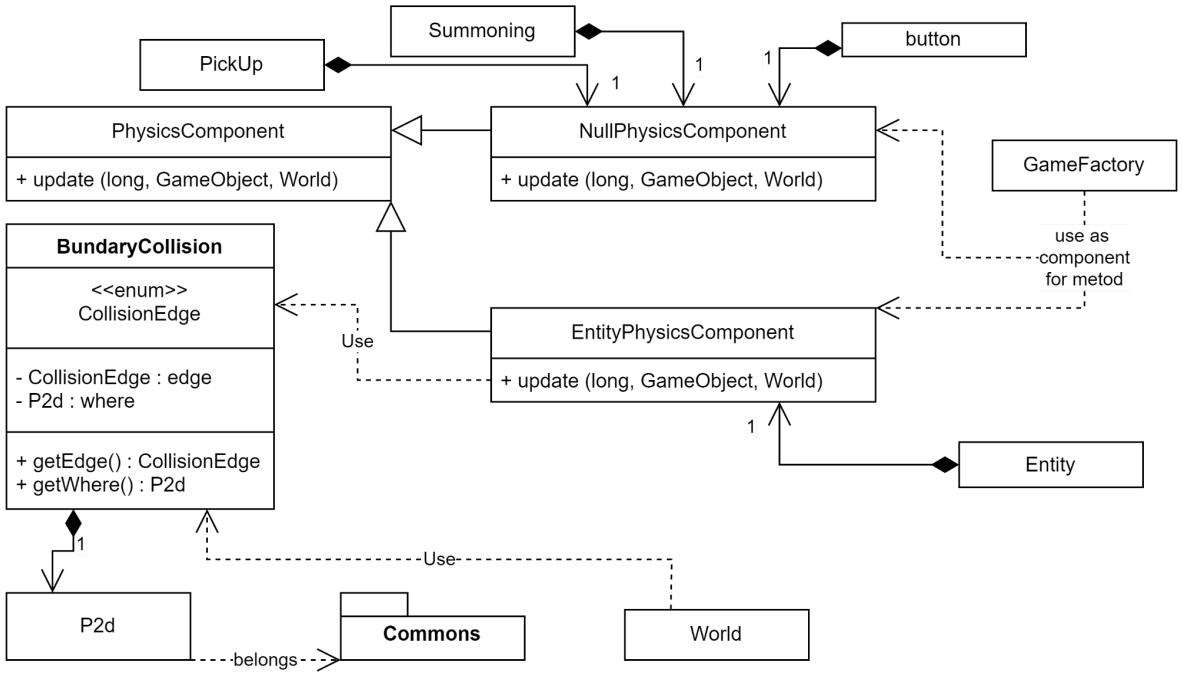
Il package Events contiene tutte quelle classi che rappresentano una specifica interazione che i GameObjects possono avere tra di loro all'interno del mondo di gioco e con il mondo di gioco stesso. Esse hanno il ruolo di mediatore fra il GameEngine e le altre componenti di gioco e sono un esempio di implementazione del design pattern Command.
Lo schema sottostante mostra i nomi e le relazioni delle classi:



Come si può vedere ci sono quattro tipi di eventi che implementano WorldEvent: KillEnemyEvent che avviene quando un Killer tocca un nemico, HitEnemyEvent che avviene se è la farfalla a toccare un nemico, HitPickUpEvent che si ha quando la farfalla tocca un pickup e infine HitBoarderEvent che avviene quando una qualunque Entity in movimento raggiunge i confini del mondo di gioco.
Cosa fanno questi eventi è deducibile dalle regole del gioco descritte in precedenza.
Si nota che la classe GameEngine implementa l'interfaccia WorldEventListener che a sua volta ha una relazione di aggregazione con World; come già indicato, il GameEngine è responsabile della gestione dei WorldEvents (infatti ne è composto da un numero variabile) e viene però aggregato a World in quanto a differenza del GameEngine, che rimane lo stesso per tutta la durata del programma, il mondo di gioco cambia ogni qualvolta si incomincia una nuova partita e quindi è necessario ricollegare i due.

Physics

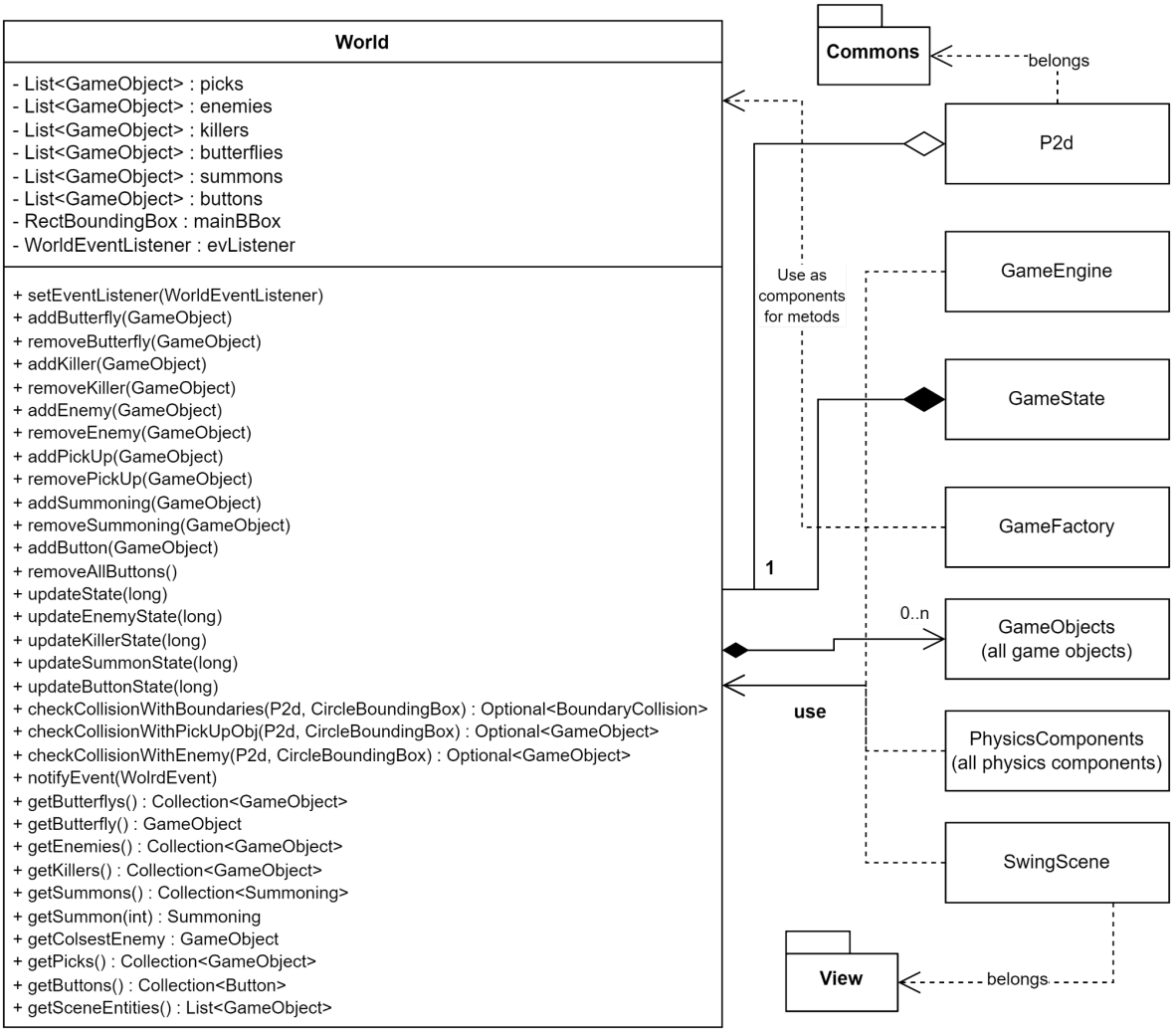
Il package Physics contiene i componenti della fisica dei GameObject; questi componenti dettano le interazioni che un oggetto ha quando tocca un BoundingBox.
Nello schema sottostante si vedrà più nel dettaglio.



Come si può vedere tutte le tipologie di PhysicsComponent implementano l'interfaccia PhysicsComponent ed, esattamente come la loro controparte InputComponent, i PickUp, le Summoning e i Button hanno delle componenti nulle; questo perché nel caso dei Button e delle Summoning questi oggetti non interagiscono con altri oggetti mentre nel caso dei PickUp è la farfalla che interagisce con loro e non il contrario.
EntityPhysicsComponent è invece responsabile delle interazioni che le varie Entity hanno con il mondo e con altri oggetti.
La classe BoundaryCollision invece serve a identificare con quale lato del bordo del mondo di gioco si sta collidendo: questo è necessario poiché collidere con il bordo del mondo di gioco fa rimbalzare la Entity come se quest'ultima fosse una pallina di gomma che rimbalza dentro una scatola chiusa; la direzione che la pallina prende se colpisce il soffitto è naturalmente diversa di quella che avrebbe se colpisse una delle pareti o il fondo della scatola.

World

La classe World è una delle classi più importanti del progetto: essa è il contenitore di tutti i GameObjects e deve essere chiamata ogniqualvolta si desidera interagire con un qualunque oggetto.
World è anche la classe attraverso la quale si deve passare se si vuole aggiornare lo stato (la fisica) di un oggetto nonché controllare se tale oggetto sta collidendo con qualcosa o meno.
World è in generale il publisher nel design Observer che ha il compito di notificare ai vari oggetti di gioco la risoluzione dei vari eventi a loro associati.
Lo schema sottostante mostra con quante classi World interagisce.



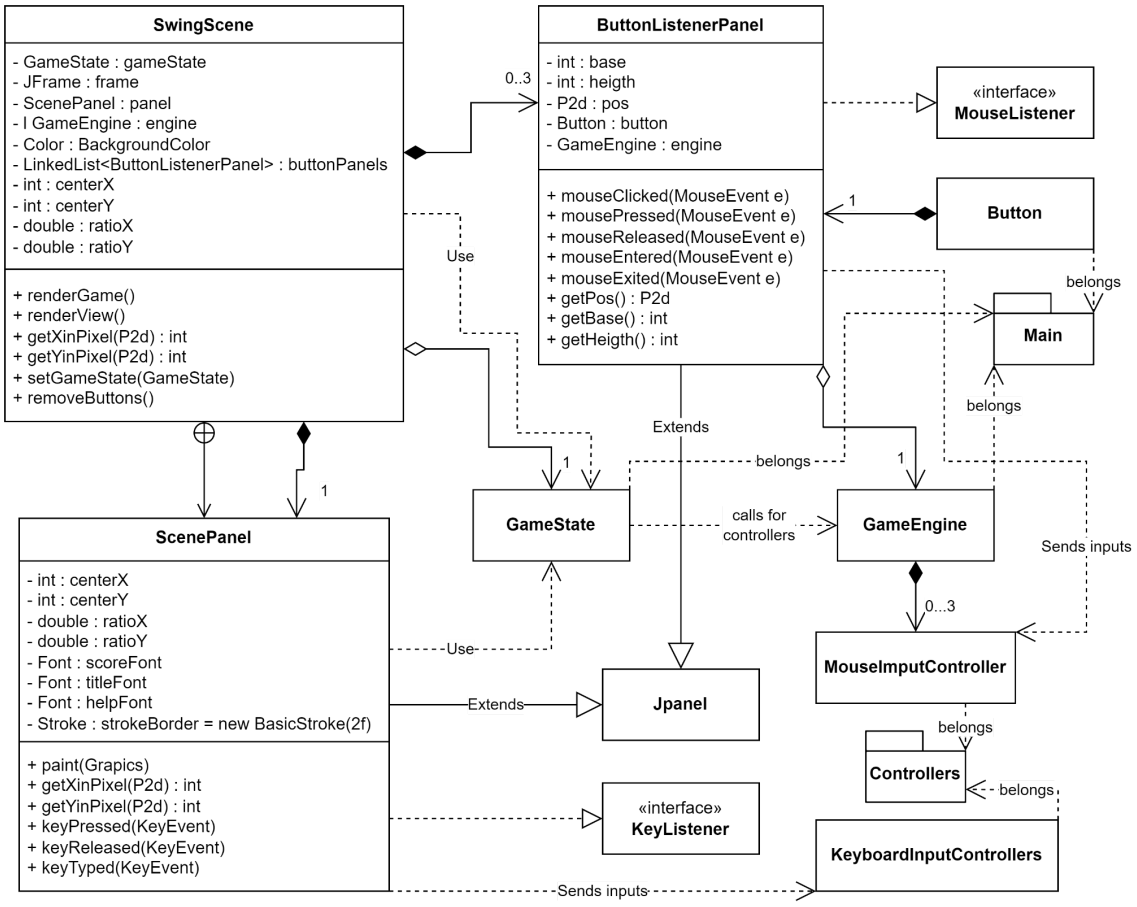
Come si può vedere World è una classe con un gran quantitativo di metodi che sono per lo più metodi setter e getter per i vari GameObjects che compongono World.
World aggrega P2d: questo è normale in quanto P2d rappresenta delle coordinate che non potrebbero esistere senza un'area di riferimento.
Infine World compone anche GameState: questo perché per accedere a World si passa attraverso il GameState che è anche responsabile dell'inizializzazione e del reset del mondo di gioco.

View

La View contiene due package e la classe SwingScene che controlla tutta la parte grafica.
Il primo package si chiama Graphics e contiene tutti i componenti grafici degli oggetti di gioco; contiene inoltre la classe SwingGraphics responsabile dell'aggiornamento di tali componenti.
Il secondo package, chiamato Listeners, contiene solo la classe ButtonListenerPanel che è responsabile dell'assunzione degli input da mouse per i bottoni; la si è divisa in questo modo per dare un package per possibili classi future dal funzionamento simile.

Scene

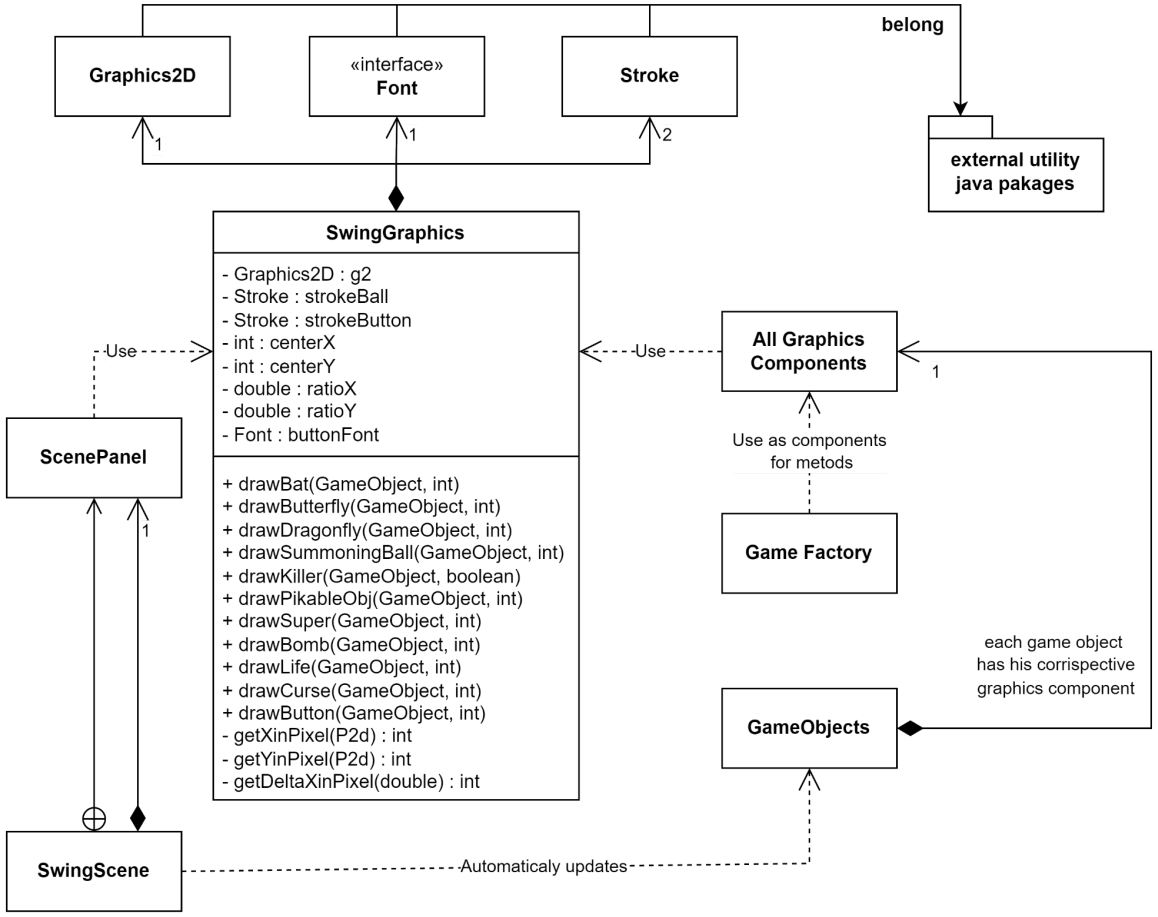
La classe SwingScene è responsabile dell'inizializzazione e del settaggio della finestra del programma e delle schermate di gioco ed è anche la classe che invia gli input ai controllers.
Lo schema sottostante sarà utile per fare chiarezza sulla struttura e sul comportamento di questa importante classe.



La prima importante caratteristica che si nota è la classe privata interna ScenePanel che è responsabile del paint di tutte le schermate di gioco e dell'ascolto e dell'invio di input da tastiera ai controllers.
Il secondo grande dettaglio è ButtonListenerPanel: questa classe è un pannello legato ad ogni bottone che ha come scopo quello di inviare a MouseInputController gli input del mouse che entra, esce o clicca all'interno di ButtonListenerPanel.
Per il resto SwingScene ha il compito di inizializzare e settare ScenePanel e ButtonListenerPanel: questo perché il primo viene resettato all'inizio di ogni partita e il secondo può variare in numero in base alla schermata di gioco.

Graphics

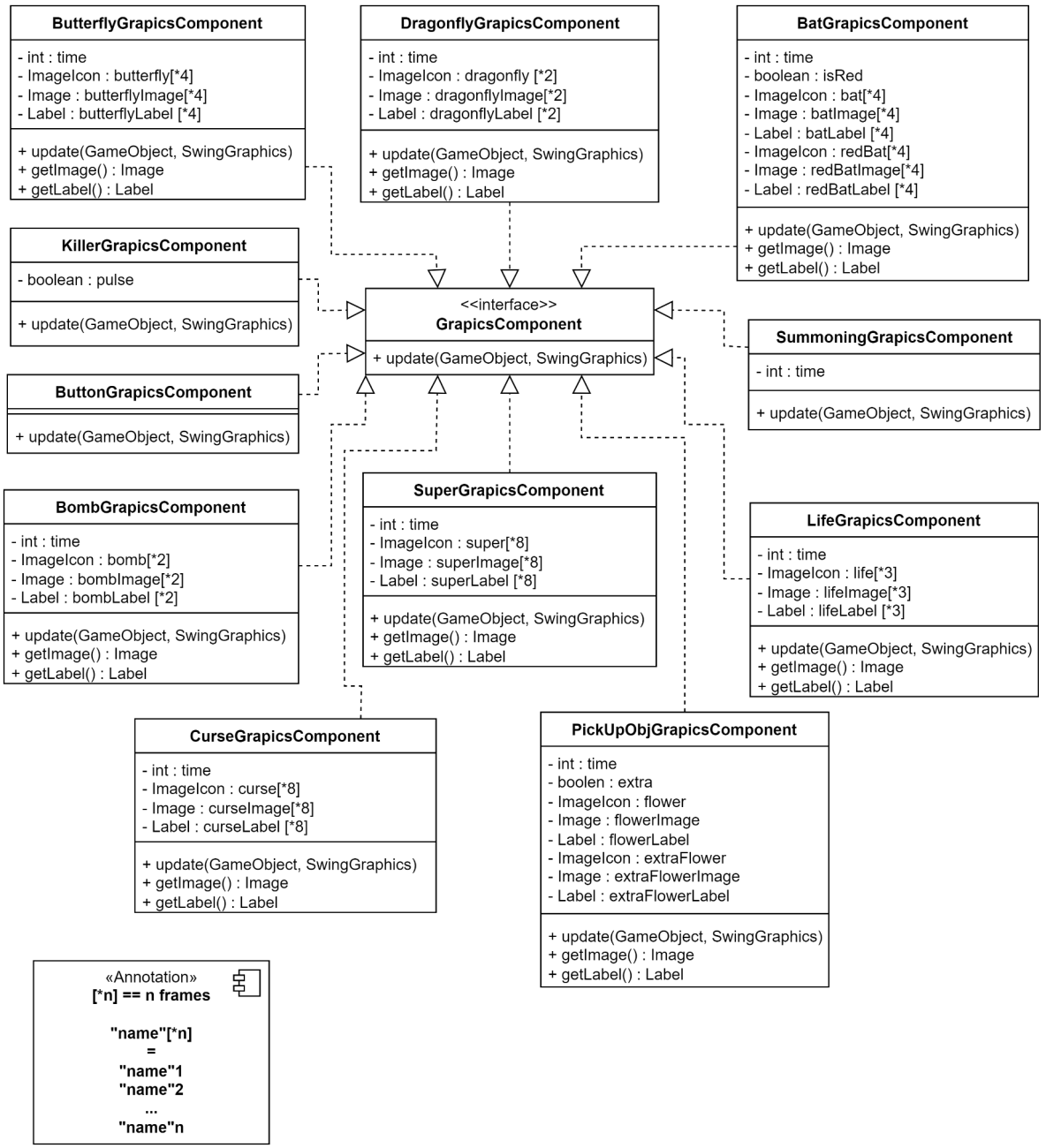
Il package Graphics, come già indicato, è suddiviso in due parti: GraphicsComponents, che contiene tutti i componenti grafici dei GameObjects, e la classe SwingGraphics che ha il compito di disegnare gli oggetti di gioco e di aggiornare le loro grafiche.
Partendo da SwingGraphics lo schema sottostante fa comprendere meglio il funzionamento di questa classe.



Come si può vedere, per disegnare un oggetto, SwingScene aggiorna automaticamente gli oggetti di gioco che a loro volta chiamano i loro componenti grafici che infine, in base al frame in cui si trova l'oggetto di gioco, chiamano il metodo draw "OggettoInQuestione" di SwingGraphics per disegnare l'oggetto nel mondo di gioco.
Inoltre anche lo ScenePanel usa SwingGraphics: questo perché lo ScenePanel può essere considerato come la componente grafica del mondo di gioco in quanto è responsabile del disegno delle varie schermate di gioco.

Graphics components

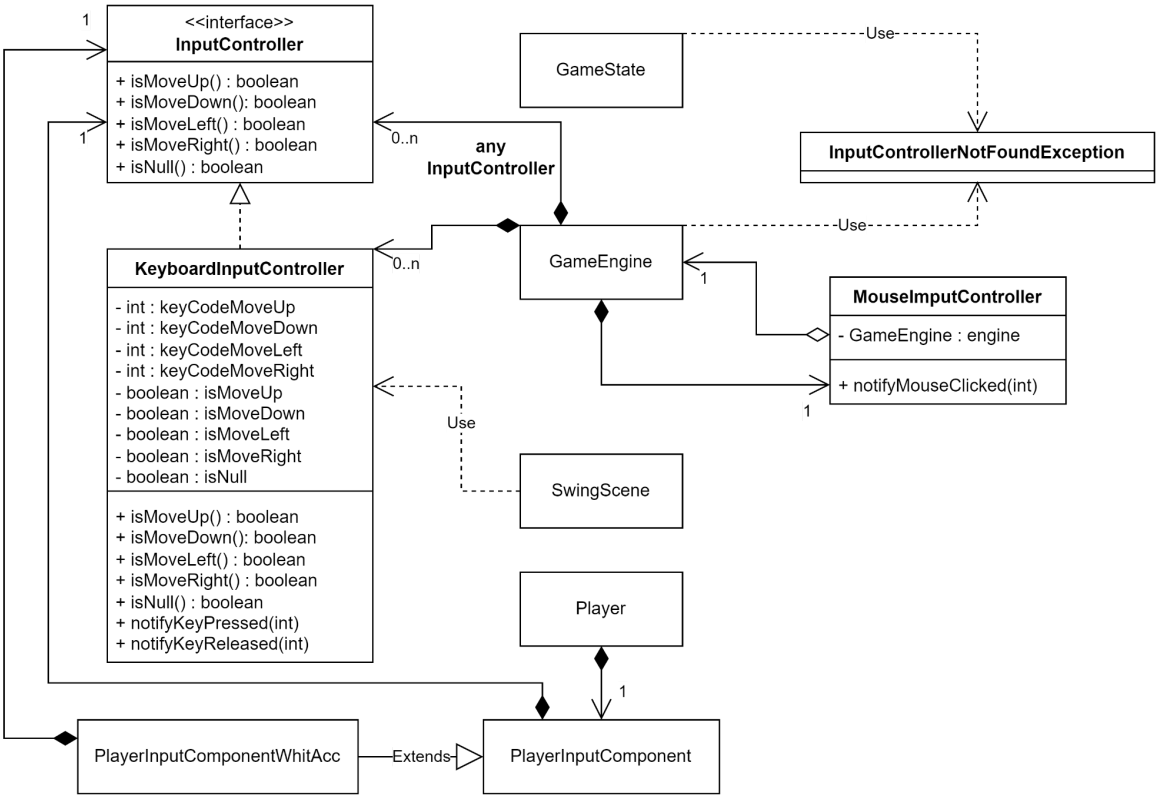
Le GraphicsComponent sono la parte grafica di tutti gli oggetti di gioco; ogni oggetto ha una sua componente ed esiste una componente grafica per ogni tipologia di oggetto come mostrato nello schema sottostante.



Tutte le componenti grafiche implementano GraphicsComponent per permettere alla fabbrica di inizializzarle tutte in base alla tipologia di oggetto.
Inoltre come riportato nella notazione dello schema, le varie componenti grafiche sono un composto di triplette di Image, ImageIcon e Label, una per ogni frame diverso che l'animazione della tipologia di oggetto richiede; ciò a eccezione di ButtonGraphicsComponent, KillerGraphicsComponent e SummonigGraphicsComponent che. non avendo un'immagine complessa. non necessitano di queste componenti extra per essere disegnate.

Controllers

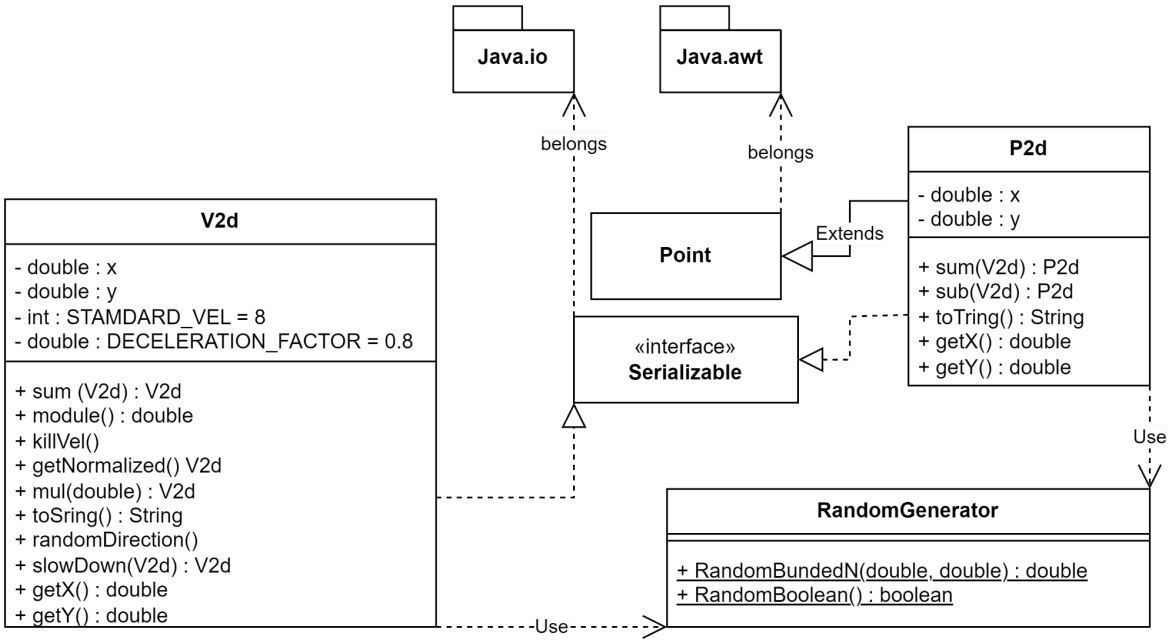
Il programma necessita di un numero molto piccolo di controllers in quanto la varietà di input che può prendere è limitata.
Ci sono principalmente due controllers: MouseInputController e KeyboardInputController.
Lo schema sottostante mostra come e da cosa vengono usati.



La prima cosa da notare è che KeyboardInputController implementa un'interfaccia madre chiamata InputController; questo è dovuto al fatto che se in futuro si volesse creare un altro controller per tastiera diverso da KeyboardInputController lo si può fare tranquillamente senza dover cambiare il GameEngine o gli InputComponent legati al Player (è lo stesso motivo per cui ci sono due PlayerInputComponent); inoltre il game engine è composto da un numero variabile n di InputController (in questa versione o 1 o 0) per permettere a versioni future del gioco di implementare un eventuale multiplayer con più facilità.
MouseInputComponent invece non necessita di tipologie diverse in quanto la sua funzione molto difficilmente può avere una funzione diversa dal cliccare un bottone alla volta: se aumentassero le tipologie di bottoni basterebbe semplicemente aggiornare MouseInputController e ButtonListenerPanel con cui comunica.
InputControllerNotFoundException è esattamente quello che dice il nome, cioè un'eccezione che serve a bloccare il programma in caso non si sia trovato un controller nella costruzione della farfalla.

Commons

I commons P2d e V2d sono la rappresentazione della posizione ed eventuale velocità di una qualunque cosa che necessita di una posizione e di un'eventuale velocità all'interno della schermata di gioco.
Lo schema sottostante mostra la struttura di Commons.



Come si può notare entrambe le classi usano staticamente RandomGenerator; questo viene fatto per motivi di leggibilità e per permettere una migliore modifica del programma in caso si necessitino soluzioni random a particolari funzioni.

Testing e Planning

Il progetto è stato fatto da una singola persona (me medesimo) e quindi non sono stati necessari particolari tools per agevolare il coordinamento di un team.
È stata utilizzata una metodologia di progettazione simile alla metodologia a spirale: pianificazione, analisi dei rischi, programmazione e verifica.
Nel caso di questo progetto ci si è potuti permettere una pianificazione del modello di dominio più breve del normale in quanto il progetto stesso altro non è che la riprogettazione ed evoluzione di un progetto preesistente già realizzato da me in Scratch; non è quindi stato necessario chiedersi cosa fare ma piuttosto come farlo e la pianificazione iniziale ha comunque richiesto alcuni giorni.
Nella successiva programmazione del progetto si è partiti costruendo dapprima le parti più basilari in grado di creare un prototipo funzionante; una volta testato tale prototipo si è cominciato ad aggiungerci funzionalità.
Tali funzionalità sono state dapprima pianificate, poi implementate e testate per assicurarsi che svolgessero correttamente quanto progettato e non si è passati avanti a progettare un'altra funzionalità finché l'intero progetto scritto fino al momento non funzionasse perfettamente.

Note ed Idee

In questa sezione vi sono elencate note per aggiustamenti ed eventuali implementazioni extra possibili in versioni future:

- Implementazione applet: al momento il progetto è semplicemente un programma scritto in Java tramite l'IDE NetBeans ma è possibile renderlo una piccola applicazione autonoma in grado di girare ovunque.
- Implementazione di suoni e musiche: il progetto al momento è muto e totalmente privo di suoni o musiche e mentre i primi sarebbero relativamente semplici da implementare le seconde richiederebbero uno sforzo maggiore, con il supporto di competenze musicali.
- Maggior varietà di entità e oggetti: sono state implementate solo 4 tipi di entità e 6 tipologie di oggetti ma è immaginabile di incrementarle arricchendo il programma.
    - Miglior implementazione di entità e oggetti: sempre in questo punto è possibile revisionare la struttura dei GameObjects per una miglioria nell'implementazione.
- Miglioramento e aggiunta di componenti grafiche: la grafica del programma è abbastanza basilare ma certamente migliorabile anche con il supporto di competenze grafiche.
- Aggiunta di un multiplayer: il progetto è un single player ma è possibile convertirlo ad un possibile multiplayer.
- Aggiunta delle Option: siccome questa è una versione basilare non è stato necessario aggiungere un menù opzioni (suono, colore, sfondi, comandi, livello di difficoltà, modalità di gioco), che sarebbe però stata una piacevole aggiunta.
- Modalità extra di gioco: questo punto è puramente teorico in quanto a differenza dei precedenti richiederebbe molto tempo nell'implementazione.

Conclusione

In conclusione nonostante la difficoltà dovuta alla scelta di lavorare da solo, creare questo progetto è stata un'impresa molto divertente e stimolante.
In questo progetto ho imparato molte cose prima fra tutte come si utilizza una GUI con cui avevo qualche problema a livello concettuale.
Il progetto mi anche aiutato ad acquisire una migliore pianificazione sia di studio che generica.
Infine per quanto genuinamente divertente sia stato progettare un piccolo videogame passo dopo passo ho anche compreso il perché certi progetti necessitino di essere svolti in gruppo: con l'aiuto di un'altra persona e una ripartizione dei compiti tutto il progetto sarebbe potuto essere progettato molto più velocemente oltre che con una notevole riduzione della difficoltà.