# Progetto – Architettura del sistema Software

Possiamo osservare come la nostra applicazione sia composta da quattro moduli principali:

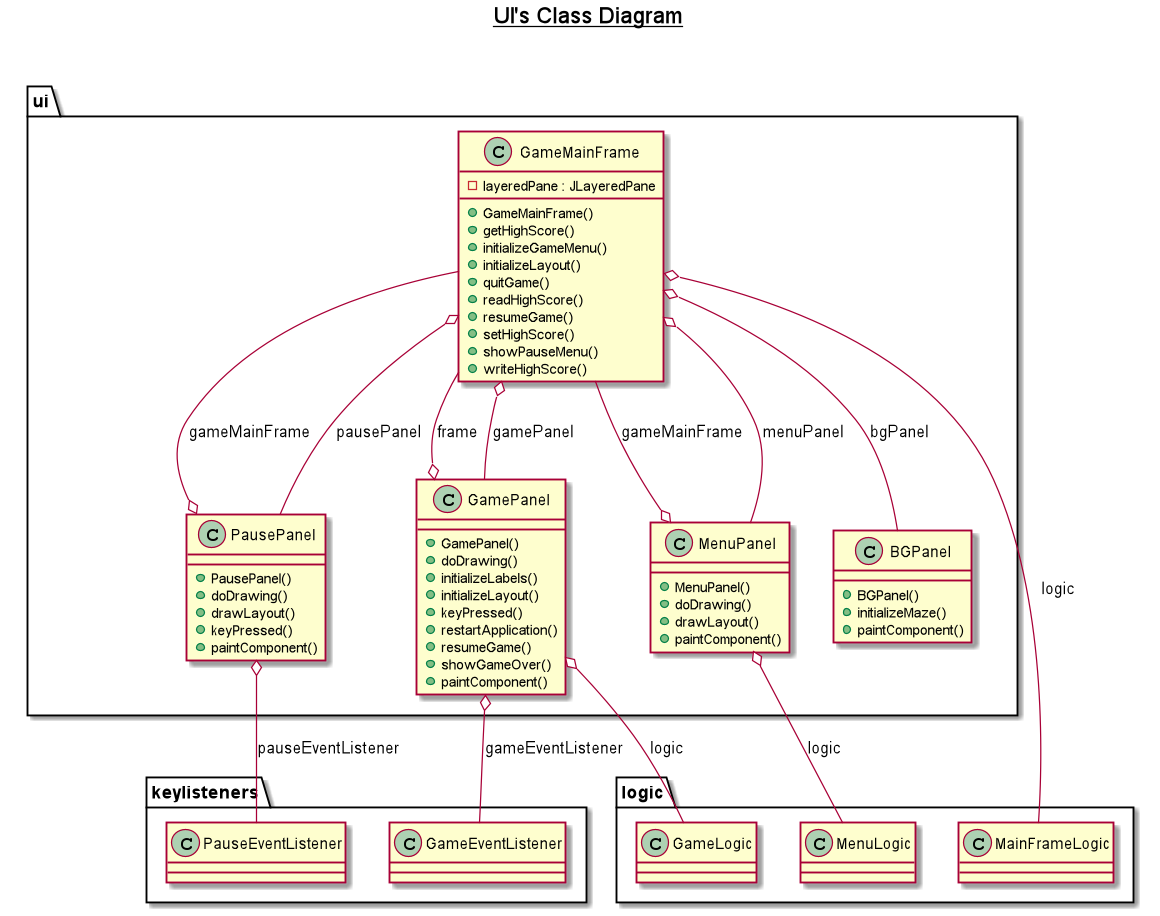
* UI (User Interface): Classi che si occupano di stampare su schermo le componenti grafiche e gli oggetti immagine che gli vengono forniti, le porzioni di codice comprese sono quasi esclusivamente finalizzate alla stampa e non comprendono logiche proprie di gioco;
* Logic: Classi controparti di altre classi dell’UI, contengono i metodi logici che scelgono cosa stampare e con quale criterio, definiscono quindi l’effettivo gameplay;
* Sprites: Classi che rappresentano entità tangibili su schermo, che siano personaggi o elementi di gioco, incapsulano tutte le informazioni che li riguardano (posizione, quantità di movimento progressivo, punto di nascita, frame della loro animazione, ecc…), che forniranno al Logic all’occorrenza,

Nello sviluppo dell’applicazione siamo partiti con l’idea di sviluppare un gioco altamente modulare e che ci offrisse oggetti da poter riutilizzare senza troppe modifiche in altri ambiti e con funzionalità che potessero essere rimosse ed aggiunte all’occorrenza. Anziché seguire pattern come quello del Model-View-Controller abbiamo cercato di inglobare il comportamento di ogni entità all’interno di singole classi a sé stanti: ogni personaggio ed elemento dell’applicativo contiene infatti non solo le informazioni sul suo stato, la sua posizione e il frame dell’animazione (ciò non dovrebbe comunque creare problemi di portabilità in quanto si tratta di un semplice riferimento ad un ImageIcon, sostituibile a seconda delle necessità), calcolato con un apposito oggetto, ma anche un Timer che gli permette di implementare un comportamento ben definito a seconda dei casi che gli vengono notificati. Le classi che si occupano di utilizzare effettivamente componenti grafiche e di stamparle su schermo sono affiancate da classi logiche che implementano l’effettivo controllo su cosa stampare e quando. Nella creazione di questa applicazione si è perciò arrivati allo sviluppo di un effettivo motore di gioco per videogame che implementano labirinti: per inserire una nuova entità o personaggio basterà infatti aggiungerlo al package sprite e inserire il suo metodo di gestione nel package logic accppiato con il metodo per disegnarlo su schermo nel package ui. Sono poi presenti classi di struttura, come il gestore dei labirinti che mette a disposizione metodi e strumenti per generare e potersi muovere dentro un qualsiasi scenario e fornisce alla logica del gioco un labirinto popolato di entità statiche, e di supporto come il gestore dei suoni e quello delle immagini memorizzate nel File System.

# Descrizione dei moduli

Andiamo ad analizzare ogni package partendo dai più importanti:

## UI



Le classi facenti parte di questo package sono le uniche che, escludendo il riferimento alla loro immagine attuale degli sprite, operano con delle componenti grafiche e si occupano di stamparle su schermo. Eccetto GameMainFrame che estende JFrame, tutte estendono JPanel: Nel GameMainFrame infatti ci occuperemo di impilare questi quattro pannelli a seconda delle necessità: mentre l’unico presente all’avvio sarà MenuPanel che mostrerà appunto il menù principale, appena il gioco verrà avviato questo verrà sostituito da BGPanel con la struttura del labirinto con sopra GamePanel per mostrare gli sprite presenti in esso e le JLabel di stato. Alla pressione del tasto Invio sopra dei due pannelli verrà posto PausePanel che mostrerà il menù di pausa con le sue opzioni. Le funzionalità logiche delle classi sono affidate alle loro controparti nel pacchetto “logic” in quanto in “ui” tutte le operazioni sono funzionali al mostrare su schermo.

## Logic

Le classi del “logic” si occupano dell’effetiva logica con cui le classi del “ui” mostrano le componenti e le immagini, mentre MenuLogic, che popola la ComboBox per la scleta del labirinto nel menù principale, e MainFrameLogic, che registra il font nell’ambiente grafico e registra l’HighScore, sono praticamente più un supporto alle loro controparti UI, GameLogic è effettivamente una delle classi più importanti dell’applicazione, in quanto si occupa di scandire i cicli effettivi update-repaint del gioco e di inserire gli sprite all’interno dell’ambiente e di stabilire ciò che accade se avviene una collisione tra di essi. Mostriamo ora alcuni dei suoi metodi che vengono chiamati quando si inizia una partita:

1. GameLogic(): Inizializza tutte le variabili necessarie al corretto funzionamento del gioco e avvia il timer principale che richiamerà periodicamente, grazie alla classe GameLoop, il prossimo metodo;
2. doOneLoop(): Richiede il focus della tastiera su GamePanel, richiama update() e repaint sempre di GamePanel;
3. update(): Se passato il breve periodo di pausa, ordina al Pac-Man e ai fantasmi di muoversi chiamando il loro metodo move(), controlla le collisioni con checkCollision(), controlla se il livello deve terminare e sceglie quale effetto in loop riprodurre;
4. endGame(): Chiamato se Pac-Man mangia tutte le pill, resetta il labirinto e passa al livello successivo chiamando restartLevel();
5. restartLevel(): Chiamato nel caso precedente o nel caso Pac-Man perda una vita, riporta tutti i personaggi nelle loro posizioni iniziali e riavvia il timer;
6. restartApplication(): Chiamato nel caso di un Game Over o se l’utente sceglie di abbandonae la partita, salva un eventuale HighScore e fa tornare il controllo al GameMainFrame.

## Sprites

Nel package “sprites” ogni classe estende la classe astratta Sprite e rappresenta un’entità all’interno del labirinto con tutti i valori che la rappresentano (posizione, incremento di movimento, dimensioni, immagine, punti, visibilità), le classi non astratte che estendo Sprite in maniera diretta rappresentano tutte entità fisse, sono infatti Pill, PowerPill, Fruit e Portal, non viene definito nelle classi un vero e proprio comportamento, sarà il GameLogic a definire ciò che succede quando un entità mobile verrà a contatto con esse tramite il suo metodo checkCollision() (es. Pac-Man entra a contatto con una PowerPill, questa sparisce e ai fantasmi viene notificata l’entrata in stato di FRIGHTENED).

Mentre gli oggetti delle classi successive, in quanto personaggi, verranno inseriti nel GameLogic direttamente, gli sprite sopracitati saranno prima inseriti nell’oggetto Maze che andrà poi in GameLogic, ciò semplificherà sia la logica (quegli oggetti sono infatti propri del labirinto e fissi a differenza dei personaggi) che il ripristino alla fine del livello.

Character estende Sprite ma è una classe astratta: fornisce metodi comuni per i personaggi del labirinto come returnToSpawnPoint() per ritornare alla posizione iniziale o i metodi astratti move() e addFrameManager().

La prima classe ad estendere Character è Pacman: il keyListener di GamePanel gli notifica i tasti premuti dal giocatore, se sono frecce direzionali Pacman salva il codice dell’ultima premuta in keyPressed e, controllando periodicamente attraverso il suo timer collegato a PacmanLoop, con il metodo changeLoop() cambierà direzione non appena tale direzione sarà disponibile. La sua animazione è regolata attraverso PacmanFrameManager che gli permette di impostare l’immagine da restituire a GamePanel in quel preciso istante.

La seconda classe, seppur astratta, è Ghost: la classe implementa la logica del movimento dei fantasmi in tutti i loro stati grazie ad un sistema di coordinate target per cui sceglieranno la direzione più breve in linea d’aria senza avere la possibilità di fare dietrofront. I target sono diversi a seconda dello stato dei fantasmi:

* Scatter: un lato del labiritno a seconda del fantasma;
* Frightened: ignora il target, il movimento è randomico;
* Eaten: lo Spwan Point di Blinky;
* Chase: dinamico durante la partita, si calcola in modo diverso per ogni fantasma con un Overrdie del metodo setChaseTarget().

Ogni fantasma modifica dinamicamente il proprio stato grazie al suo timer con GhostLoop che regola il cambio Chase-Scatter e decide quando terminare lo stato di Frightened, quest’ultimo si attiva col metodo di ghost becomeFrightened(), invocato dal GameLoop nel caso di una collisione di Pac-Man con una PowerPill, in maniera analoga viene impostato lo stato di Eaten.

Tra i quattro fantasmi l’unica differenza considerevole è la presenza di un riferimento a Blinky in Inky, il quale ha bisogno di ottenere dinamicamente sia le coordinate di Pacman (come gli altri) che di Blinky per poter calcolare il suo Chase Target.

Tutti i Character controllano la possibilità di muoversi avanti o di cambiare direzione tramite i metodi messi a disposizione dalla classe statica MazeManager che vedremo in seguito.