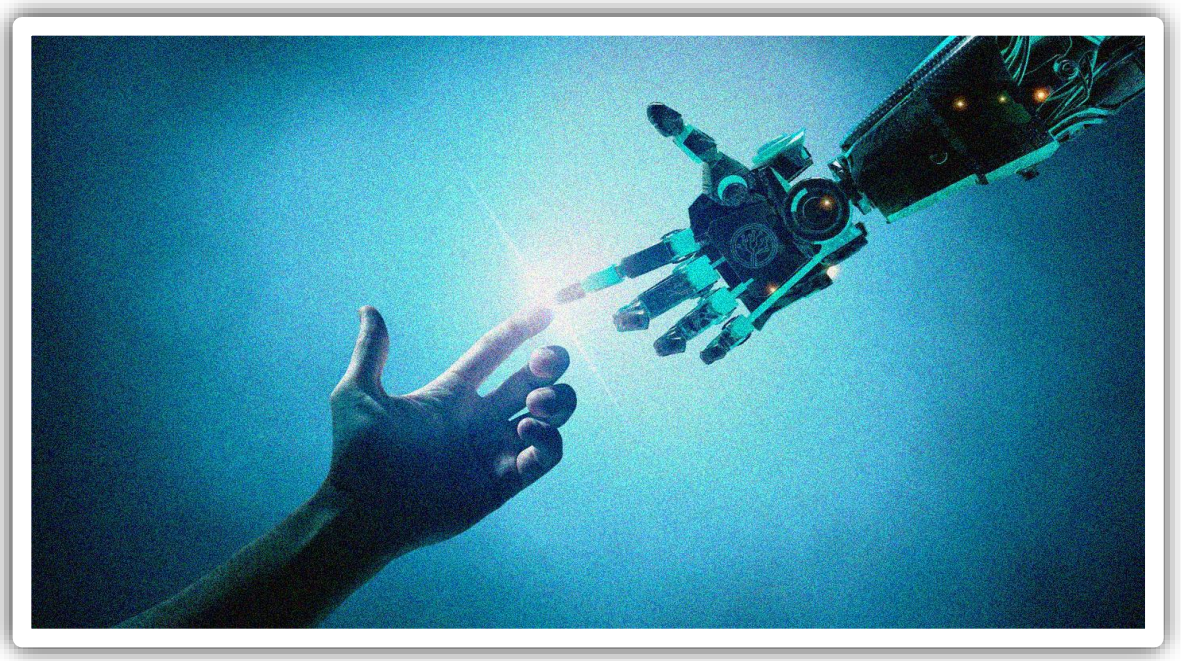
UNIVERSITÀ DELLA CALABRIA
DIPARTIMENTO DI
INGEGNERIA INFORMATICA,
MODELLISTICA, ELETTRONICA
E SISTEMISTICA
DIMES

Corso di Laurea in Ingegneria dell'Automazione

A.A. 2021/2022



WUMPUS WORLD GAME

Applicazione Android

Insegnamento di Programmazione dei dispositivi mobili

Relazione del progetto di un'applicazione Android che implementa una versione del gioco "Hunt the Wumpus".

IVONNE RIZZUTO
Matricola 196525

Indice

Indice 1
Introduzione 3
Specifiche progettuali 4
Informazioni sulla versione del gioco 4
Modalità di gioco 4
Descrizione dell'ambiente 5
Formulazione PEAS 6
Misurazione delle performance 6
Ambiente 6
Attuatori 7
Sensori 7
Struttura del back-end 9
Struttura delle classi 9
Caratteristiche della mappa e degli elementi di gioco 9
Struttura di una cella 11
Struttura del vettore dei sensori 11
Creazione del terreno di gioco 12
Gestione della sessione di gioco 15
Avvio della sessione di gioco 17
Controllo delle mosse di gioco 19
Giocatore automatico 21
Calcolo del punteggio 22
Generazione della libreria 23
Applicazione Android 24
Caratteristiche dell'applicazione 24
Casi d'uso 24
Testing 26
Emulazione 26
Struttura del front-end 27
Diagramma delle classi 27
Organizzazione delle risorse 28
Manifesto dell'applicazione 29
Integrazione dei moduli in un progetto Git 31
Sviluppi futuri 32

Strumenti utilizzati 33

Introduzione

In questo elaborato verrà presentato il progetto di un'applicazione Android ispirata al videogioco conosciuto con il nome di "**Hunt the Wumpus**", realizzato nel 1972 da *Gregory Yob*.

Si tratta di un gioco di avventura che si svolge in un labirinto, generato in maniera casuale e costituito da una serie di stanze, tra loro comunicanti se adiacenti.

Il giocatore interpreta il ruolo di un *cacciatore* che, durante l'esplorazione del dungeon, dovrà sopravvivere alle insidie disseminate lungo il percorso e cercare di individuare la tana del mostro, il *Wumpus*, senza rivelargli la propria presenza e rischiare di farsi uccidere.

Questo sarà possibile sfruttando le tracce disseminate per il labirinto, ovvero il cattivo odore emanato dal mostro, che giunge sino alle celle attigue alla sua tana. Questo indizio, nel gioco originale, viene fornito mostrando, nelle celle adiacenti, delle macchie di sangue.

La sola possibilità che si ha di sfuggire al mostro è quella di ucciderlo scoccando una freccia, da una qualsiasi stanza adiacente a quella in cui è nascosto.

Inoltre, bisogna evitare le stanze con i pozzi, per evitare di caderci dentro, attigue alle celle da cui giunge la brezza, che sostituisce, nel gioco originale, la presenza del muschio.

Un ulteriore rischio, nel gioco di Yob, è costituito dai super-pipistrelli, che possono catturare il giocatore e rilasciarlo in una stanza differente, scelta in maniera casuale.

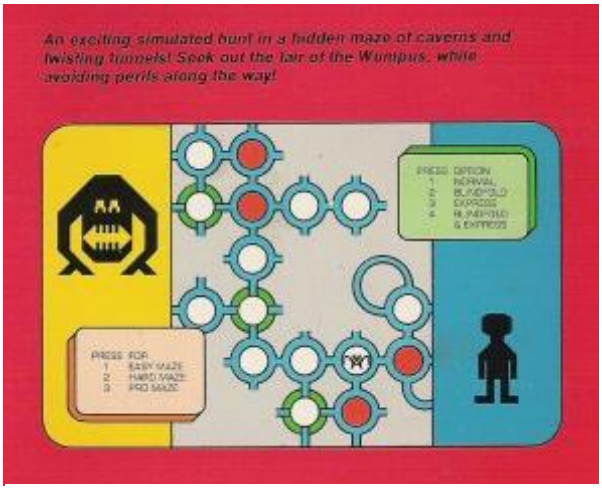


Figura 1. Gioco "Hunt the Wumpus" di Yob.

Si è deciso di implementare questo gioco, un esempio ricorrente nell'ambito dell'intelligenza artificiale, proprio per realizzare un agente che fosse in grado di calcolare una soluzione, ovvero un percorso sicuro, che conduca il personaggio giocabile alla vittoria.

Questa funzionalità può essere richiesta dall'utente selezionando la voce corrispondente dal menu di gioco, visualizzando così il percorso che l'agente dotato di intelligenza artificiale è riuscito a calcolare, a partire dalla posizione corrente del personaggio giocabile.

Specifiche progettuali

La versione del gioco realizzata in questo progetto, differisce, per alcuni aspetti, dal gioco ideato da Yob.

Infatti, mentre il gioco dello statunitense fu sviluppato, in linguaggio BASIC, inizialmente in modalità testuale e soltanto in seguito convertito in modalità grafica, la versione trattata in questo elaborato è stata ideata ed implementata per interezza in Java.

Dopo averla testata e resa giocabile dall'utente, ricevendo degli input testuali da console, è stata creata una libreria che ne racchiudesse le funzionalità, affinché andasse a costituire il back-end di quella che sarebbe stata l'implementazione dell'applicazione Android vera e propria, dotata di grafica e progettata per dispositivi mobili dotati di questo sistema operativo.

Il nome dell'applicazione è "Wumpus World Game", o "Il mondo del Wumpus", in base alla lingua del dispositivo.

Informazioni sulla versione del gioco

Per quanto riguarda la modalità di gioco, oltre all'assenza dei super-pipistrelli, che verranno integrati in futuro, un'ulteriore differenza con il gioco originale è costituita dalla possibilità, data al giocatore, di decidere che ruolo interpretare nella storia.

Nello specifico, l'utente potrà decidere se impersonare il cacciatore e quindi avere come obiettivo quello di trovare il tesoro nascosto del mostro, oppure se prendere le parti del Wumpus e cercare di sopravvivere a questa caccia spietata, trovando una via di fuga che lo conduca fuori dal labirinto.

Questo vuol dire che, se il cacciatore avrà a sua disposizione una sola freccia per tentare di colpire il Wumpus, nel caso in cui quest'ultimo sia il personaggio giocabile, allora, potrà lanciare un masso per cercare di uccidere il cacciatore, prima che lo stani.

Qualunque sia il ruolo scelto, la partita termina se il giocatore muore, sia che cada nel pozzo o che venga ucciso dal mostro, nel caso del cacciatore, sia che venga catturato con una trappola o che venga colpito dal cacciatore, nel caso del Wumpus.

Per quanto riguarda lo scenario in cui si può muovere il giocatore, nella versione originale questo era costituito da una serie di caverne misteriose collegate tra loro, mentre, nella versione di questo progetto l'ambientazione è rappresentata da una foresta fitta, dai percorsi intricati.

Modalità di gioco

Il mondo del gioco "Hunt the Wumpus" è strutturato come una matrice di dimensione (4 x 4).

Questa mappa andrà costituire il terreno di gioco in cui dovrà muoversi il giocatore, indipendentemente che sia l'utente dell'applicazione Android oppure il giocatore automatico, cioè l'agente basato su conoscenza.

A prescindere che si trovi ad impersonare il cacciatore oppure il mostro, il giocatore avrà come unico obiettivo quello di puntare alla vittoria, cercando di evitare i pericoli e di colpire il nemico se si trova nelle vicinanze.

Si precisa che, a seconda della modalità di gioco scelta ad inizio partita, il PG sarà quello che potrà muoversi sulla mappa di gioco, mentre il nemico rappresenterà un NPG, cioè un personaggio non giocabile, che manterrà fissa la sua posizione, per tutta la durata della sessione di gioco corrente.

Dunque, oltre alla possibilità di scegliere che ruolo avere nel gioco, l'utente potrà decidere se muovere direttamente il PG oppure affidare la risoluzione della sessione avviata ad un giocatore automatico, costituito da un agente software basato su conoscenza.

Come già detto per l'utente, anche l'agente avrà a disposizione dei suggerimenti, degli indizi sul contenuto delle celle della mappa che sono adiacenti alla sua posizione, in modo tale da acquisire una conoscenza parziale dell'ambiente che lo circonda.

Figura 2. Modalità di gioco.

Quindi, sia nella modalità di gioco *interattiva*, ovvero quella in cui viene coinvolto l'utente del dispositivo su cui gira l'applicazione Android, sia nella modalità di risoluzione *automatica*, in cui sarà, invece, l'agente a condurre la partita, saranno fornite queste informazioni:

- Le celle adiacenti al nemico (tramite la variabile *ENEMY_SENSE*), rappresentato dal Wumpus oppure dal cacciatore, dipendentemente dalla scelta di giocare con l'uno o l'altro come PG;
- Le celle adiacenti ad un pericolo (per mezzo della variabile *DANGER_SENSE*), che può essere un pozzo o una trappola, a seconda del PG che si sta utilizzando; allora, queste conterranno, rispettivamente, la brezza che giunge sin dalla prossimità del pozzo oppure un mucchio di foglie, volte a coprire la trappola;

Descrizione dell'ambiente

Per quanto riguarda le caratteristiche del gioco che si è voluto implementare, si può affermare, dal punto di vista formale, che l'ambiente che costituisce il mondo del Wumpus sia *completamente osservabile* (cioè *accessibile*), in quanto l'agente può percepire e quindi trarre delle informazioni dall'ambiente circostante che, benché siano relative soltanto alle celle adiacenti a quella in cui si trova, sono sufficienti ai fini della determinazione dell'azione da compiere.

Nonostante questa limitazione, l'ambiente in cui l'agente si trova ad interagire può essere definito:

- *deterministico*, poiché l'esito di ogni azione che verrà eseguita avrà un determinato effetto, noto a priori;
- di tipo *statico*, poiché rimane immutato, per tutta la sessione di gioco, se non si prendono in considerazione le azioni che, eventualmente, verranno compiute dall'agente.

Infatti, a parte il giocatore, tutti gli elementi di gioco, quali i pericoli o lo stesso nemico, non potranno muoversi e quindi variare la propria posizione sulla mappa.

Si può, inoltre, specificare che l'ambiente sia:

- *discreto*, perché le percezioni che l'agente può acquisire e le azioni che può eseguire sono limitate, ovvero si presentano in numero finito;
- di tipo *sequenziale*, in quanto ogni decisione presa dall'agente andrà ad influenzare quelle successive.

In ogni partita del gioco, si avrà sempre un *singolo agente*, rappresentato proprio dall'algoritmo di risoluzione della mappa di gioco utilizzato per implementare il giocatore automatico.

Questo potrà ricoprire il ruolo del Wumpus o del cacciatore, a seconda della modalità scelta dall'utente dell'applicazione e di conseguenza, verrà assegnato, come nemico, il corrispettivo antagonista, che non potendo cambiare né la propria posizione né il proprio stato, non sarà considerato un agente in nessuna circostanza, ma solamente un personaggio non giocabile.

Formulazione PEAS

Per meglio comprendere il funzionamento dell'applicazione, soprattutto per quanto concerne il meccanismo della modalità di gioco automatica e di conseguenza, il comportamento dell'agente razionale, bisogna effettuare una descrizione PEAS.

Con il termine PEAS ci si riferisce all'acronimo inglese di "**Performance Environment Actuators Sensors**", ovvero la definizione di un problema, di cui un agente razionale dovrà determinare la soluzione, comprensiva della descrizione dell'*ambiente operativo* in cui l'agente stesso si trova ad agire.

Misurazione delle performance

Per il problema trattato, ovvero il gioco del Wumpus, la definizione delle prestazioni riguarda la determinazione, da parte dell'agente, di un percorso sicuro ed il più breve possibile, che gli consenta di evitare i pericoli, abbattere il nemico ed ottenere il premio, conseguendo, così, la vittoria della partita.

Infatti, il massimo profitto si ottiene acquisendo il punteggio massimo, che potrà essere raggiunto riuscendo a terminare la partita dopo aver compiuto le seguenti azioni:

- evitare tutti i pericoli disseminati sulla mappa;
- abbattere il nemico con la freccia;
- trovare la cella contenente il premio, che per il cacciatore è rappresentato dal tesoro del Wumpus, mentre, nella modalità in cui è il mostro ad essere il PG scelto dall'utente, è costituito da un passaggio segreto che lo condurrà in salvo;
- effettuare il minor numero possibile di mosse, poiché ogni cella in più che viene visitata comporta, nel punteggio, comporta un punto di penalità;

La valutazione di tutti questi elementi sarà quella che consentirà di effettuare una stima delle prestazioni raggiunte dall'agente razionale in questione.

A questo proposito, si riportano i valori dei punteggi, stabiliti per ogni azione:

- +100, in caso di vittoria del personaggio giocabile, che evidentemente ha trovato il premio;
- -50, nel caso in cui il giocatore si posizioni in una cella contenente un pericolo;
- -100, se il personaggio giocabile viene ucciso dal nemico;

- +50, se il giocatore è riuscito ad individuare ed abbattere il nemico;
- -1, per ogni cella esplorata, ovvero per ogni mossa effettuata dal personaggio giocabile;

Ambiente

L'ambiente è costituito da una griglia, costituita da quattro righe e quattro colonne, per un totale di sedici celle.

Sarà questa matrice, di dimensione prefissata e non modificabile, che andrà a costituire il terreno su cui saranno posizionati tutti gli elementi di gioco.

Nel dettaglio, il contenuto di ogni cella verrà scelto in maniera casuale, secondo una determinata funzione di probabilità e verrà specificato nella fase di creazione del terreno di gioco.

Gli elementi che andranno a riempire la mappa sono:

- il personaggio giocabile, scelto dall'utente;
- il nemico del personaggio giocabile;
- gli alberi che, eventualmente, potrebbero ostruire il passaggio al giocatore;
- il premio che permetterà di ottenere, al giocatore, dei punti in più;
- dei pericoli, ovvero dei pozzi o delle trappole, correlati alla modalità di gioco;

Si precisa che alcune celle potranno essere vuote, perché non conterranno nessuno degli elementi sopraelencati, pertanto dovranno essere considerate come sicure.

Attuatori

Gli attuatori rappresentano il modo effettivo in cui l'agente può interagire nel mondo circostante; infatti, costituiscono dei dispositivi di output tramite cui potrà compiere delle azioni che andranno ad apportare delle modifiche allo stato dell'ambiente in cui si trova.

Nel caso trattato, come attuatore, si dovrà considerare l'azione stessa di muoversi, compiuta dall'agente.

Questa consiste nello spostarsi, casella dopo casella e sempre tra celle della matrice che siano adiacenti tra loro, per proseguire l'esplorazione della mappa di gioco.

I movimenti consentiti, quindi, sono:

- lo spostamento a destra;
- lo spostamento a sinistra;
- lo spostamento verso il basso;
- lo spostamento verso l'alto;

Un ulteriore attuatore è rappresentato dall'azione di sparo che, nello specifico, consiste nello scoccare una freccia, oppure nel lancio di un masso.

All'agente, infatti, viene fornita la possibilità di tentare di colpire il nemico, se crede di aver individuato la sua posizione, specificandone la direzione sempre tramite degli input direzionali, come quelli sopradescritti.

Il tentativo di sparo può essere ripetuto, idealmente, senza limitazioni, però la freccia (o il masso) che l'agente ha a disposizione è una soltanto, quindi dopo averla utilizzata, se il tiro non è andato a buon fine, l'agente ha sprecato la possibilità di abbattere il nemico per tutta la durata della sessione di gioco corrente.

Sensori

Per percepire gli stati che caratterizzano l'ambiente operativo, ogni agente è dotato di sensori, cioè dei dispositivi di input che gli consentano di rilevare proprio le informazioni che utilizzerà nei suoi processi decisionali.

Nel gioco del Wumpus, i sensori messi a disposizione all'agente lo aiuteranno a determinare la posizione dei pericoli e del nemico, sul terreno di gioco.

Nel dettaglio, si avrà l'accensione del sensore ENEMY_SENSE, se il nemico si trova in una cella adiacente e/o del sensore DANGER_SENSE, se il personaggio giocabile si trova vicino ad un pericolo.

Nell'implementazione del gioco trattata in questo elaborato non è stato previsto un sensore che indichi, all'agente, la sua vicinanza al premio.

Altre indicazioni che l'agente riceve, durante la sua esplorazione, sono:

- la percezione della fine della mappa, se tenta di spostarsi oltre le celle della cornice;
- l'informazione che la cella in cui si trova è sicura, perché nessuna delle celle adiacenti contiene un elemento che è stato segnalato dai sensori;

Si riporta la tabella associata alla descrizione PEAS del problema trattato:

Problema	P	E	A	S
Wumpus World Game	o -1, ad ogni mossa; o +50, colpendo il nemico; o +100, trovando il premio; o -50, trovando il pericolo; o -100, venendo uccisi;	o matrice 4 x 4; o posizione PG; o posizione nemico; o posizione pericoli; o posizione premio;	o muovi a destra; o muovi a sinistra; o muovi verso l'alto; o muovi verso il basso; o spara;	o enemy sense; o danger sense; o border; o safe;

Struttura del back-end

Per l'implementazione dell'applicazione Android "Wumpus World Game", dal punto di vista progettuale, si è deciso di scrivere prima il codice che andasse a costituirne il back-end e poi procedere con la sua integrazione, dopo averne verificato il corretto funzionamento, con la parte grafica, realizzata con le librerie di Android.

Il back-end, quindi, costituisce un progetto a sé stante, denominato "**Hunt the Wumpus or Not**", implementato interamente in Java 8, che realizza una versione del gioco "Hunt the Wumpus" anch'essa testuale.

Infatti, già da questa versione del progetto, si hanno a disposizione tutte le funzionalità che poi verranno utilizzate e richiamate nell'interfaccia utente dell'applicazione Android.

Ad esempio, per quanto riguarda la modalità di gioco, è stata prevista l'introduzione di due tipi differenti di giocatore, ovvero:

- il giocatore vero e proprio, *Human Player*, pilotato dall'utente;
- il giocatore automatico, *Automatic Player*, che consiste nell'implementazione di un agente dotato di una semplice intelligenza artificiale;

Quindi, al momento dell'avvio del gioco, l'utente potrà scegliere in che modalità giocare, ovvero se nella versione classica, **Hero Side**, in cui l'avventuriero deve uccidere il Wumpus oppure nella versione **Wumpus Side**, in cui dovrà impersonare proprio il mostro del gioco e riuscire a sopravvivere alla battuta di caccia.

Inoltre, l'utente sarà nelle condizioni di poter decidere se vuole essere lui direttamente a controllare le mosse del suo personaggio oppure lasciare che la risoluzione del gioco e, quindi, l'esplorazione del labirinto, sia affidata al giocatore automatico, ovvero un agente provvisto di intelligenza artificiale.

Struttura delle classi

Di seguito verrà riportata una descrizione delle funzionalità realizzate per ciascuna delle classi che costituiscono il gioco, evidenziando il ruolo che ognuna di queste ricopre ed il modo in cui si interfaccia agli altri componenti del codice, anche tramite i diagrammi UML.

Caratteristiche della mappa e degli elementi di gioco

Si riporta il diagramma UML di tutte le classi contenute nel package *game.structure*, dedite alla costruzione delle celle che andranno a comporre il terreno di gioco ed al loro riempimento, tramite il posizionamento del PG, del nemico e degli altri elementi di gioco.

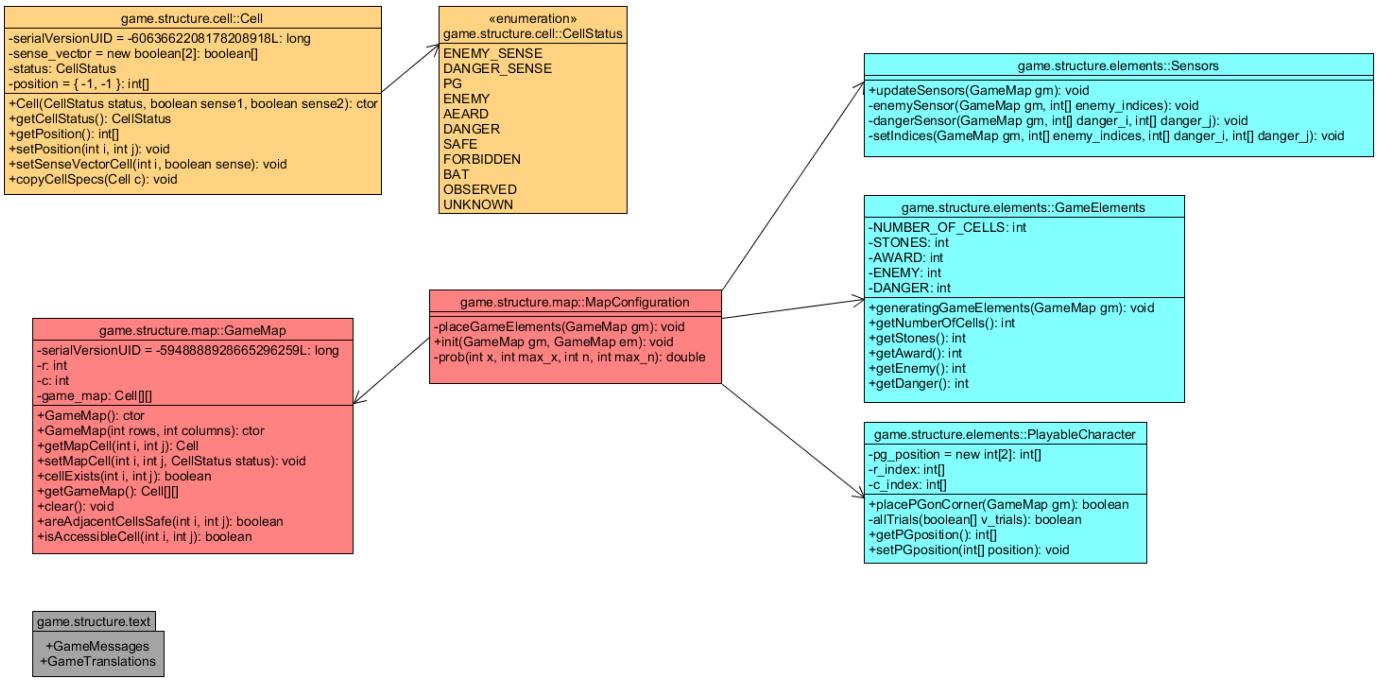


Figura 3. Diagramma UML delle classi del package game.structure.

Segue una descrizione delle funzionalità delle classi riportate sopra, quali:

- la classe *Cell*, che implementa le caratteristiche di ogni singola casella della matrice, ovvero della mappa di gioco;
- C*ellStatus*, un'enumerazione che si occupa di definire le tipologie che può assumere l'oggetto Cell, nella mappa di gioco;
- la classe *Sensors*, che assegnerà il valore dei sensori per le celle che circoscrivono gli elementi di gioco di interesse per il PG, quali il nemico ed i pericoli, sulla base degli elementi con cui è stata riempita la mappa;
- la classe *GameElements*, che restituirà il numero degli elementi di gioco che dovranno essere posizionati sulla mappa, specificandone il valore per ogni tipologia; ad esempio, il numero di celle non accessibili (STONES), generato in maniera casuale;
- la classe *PlayableCharacter*, che dovrà tenere traccia della posizione del PG sul terreno di gioco, dopo aver provveduto al suo inserimento, in seguito al riempimento della matrice, con tutti gli altri elementi di gioco; il metodo dedito a questa funzione, denominato *placePGOnCorner(GameMap)*, inserirà il personaggio giocabile in una cella posta sulla cornice della matrice, verificando che il passaggio, a partire da questa cella, sia libero almeno verso una delle celle adiacenti;
- la classe *GameMap*, che definisce la struttura basilare del gioco, ovvero la mappa su cui si potrà muovere il personaggio giocabile;
- la classe *MapConfiguration*, dedita alla generazione della mappa di gioco, popolandola di tutti i suoi elementi;
- *GameMessages*, una classe che conterrà i messaggi visualizzati durante la sessione di gioco per informare il giocatore dello stato della partita e dell'ambiente circostante;

- la classe di utilità *GameTranslations*, che metterà a disposizione i nomi assegnati agli elementi di gioco, che differiscono tra le due modalità previste, cioè eroe e mostro.

Struttura di una cella

Ogni casella della mappa di gioco, cioè della matrice di dimensione (4x4), sarà descritta e caratterizzata in base al suo contenuto, ovvero in dipendenza dall'elemento di gioco che conterrà, dopo la generazione della mappa.

Questa specifica verrà esplicitata dal valore dell'attributo *status*, di tipo *CellStatus*, poiché costituito da uno soltanto dei valori dichiarati dalla classe che definisce questa enumerazione, quali:

- PG, cioè il personaggio giocabile, che sarà:
  - l'Avventuriero, se ci si trova nella modalità eroe;
  - il Wumpus, se ci si trova nella modalità mostro;

- ENEMY, che è l'avversario del PG, ovvero il Wumpus, se è stata scelta la modalità eroe, oppure l'Avventuriero, nella modalità mostro;

- AWARD, che rappresenta un premio che può essere trovato dal PG, ovvero:
  - un tesoro, nella modalità eroe;
  - una via di fuga dal dungeon, in modo da consentire al Wumpus di scappare dal cacciatore, nella modalità mostro;

- DANGER, è il pericolo in cui può incorrere il PG che, sempre in correlazione alla modalità di gioco, può essere:
  - PIT, la fossa, cioè il pozzo in cui rischierà di cadere l'avventuriero;
  - TRAP, la trappola, preparata dal cacciatore in cui potrebbe cadere il Wumpus;

- SAFE, che indica la presenza di una cella sicura, accessibile e priva di pericoli;

- FORBIDDEN, che rappresenta una cella non accessibile, raffigurata come un sasso o un albero, poiché costituisce un punto della mappa in cui il PG non può essere posizionato;

Struttura del vettore dei sensori

Ogni casella della mappa di gioco è caratterizzata da un attributo, denominato *sense_vector*, cioè un vettore di due celle di tipo booleano, che costituisce il vettore dei sensori.

Questo significa che ognuna delle celle di questo vettore conterrà un'informazione riguardo le caselle adiacenti a quella attuale, in modo da indicare se nelle caselle che circoscrivono quella presa in considerazione, si possa trovare un pericolo oppure il nemico.

Dunque, questo vettore è composto da due celle, che sono associate, rispettivamente, a due dei campi definiti dall'enumerazione *CellStatus*, ovvero:

- ENEMY_SENSE, *sense_vector[0]*, il campo che indica se il nemico si trova nelle vicinanze; questo valore, in base alla modalità di gioco scelta, può segnalare:
    - la presenza dell'odore del Wumpus (STINK);
    - lo scricchiolio del terreno della foresta, dovuto al passo dell'eroe (CREAK);
- DANGER_SENSE, *sense_vector[1]*, il cui valore indica che nelle vicinanze, ovvero le celle adiacenti a quella di cui si sta analizzando il vettore dei sensori, si potrebbe incorrere in una situazione di pericolo, dovuta alla presenza:
    - di una fossa, come si evince dalla brezza (BREEZE) che, dalla prossimità del pozzo, giunge fino al PG;
    - del cacciatore in agguato, di cui si sente il fruscio delle foglie che vengono spostate al suo passaggio (SWISH);

Creazione del terreno di gioco

La classe più rilevante è **MapConfiguration**, poiché, utilizzando tutte le altre classi del package considerato, è dedicata alla messa in atto di tutte le configurazioni necessarie alla preparazione del campo di gioco.

Infatti, è tramite i metodi di questa classe che verranno inizializzati tutti gli elementi di gioco, sulla base di una funzione di probabilità che ne stabilisce il posizionamento in maniera totalmente casuale.

La struttura della mappa che si otterrà alla fine di questo processo sarà del tipo:

Mappa			
S	A	S	S
D	S	S	S
S	S	S	E
P	S	S	D

Legenda	
S = SPAZIO VUOTO	D = PERICOLO
E = NEMICO	F = SPAZIO VIETATO
P = PG	A = PREMIO

La funzione che genera una configurazione valida per il terreno di gioco, ovvero tramite cui verranno posizionati tutti gli elementi di gioco, è la seguente:

```
public static void init(GameMap gm, GameMap em) {
	//variabile asiliaria
	boolean done = false;
```

```
        //generazione degli elementi di gioco
        GameElements.generatingGameElements(gm);
        //ciclo di posizionamento degli elementi di gioco e del pg
        do {
                //posizionamento degli elementi di gioco nella mappa
                placeGameElements(gm);
                //si cerca di posizionare il personaggio giocabile
                done = PlayableCharacter.placePGonCorner(gm);
                //si esce dal ciclo se il pg e' stato posizionato
        }while(!done);
        //si aggiorna il vettore dei sensori per ogni cella
        Sensors.updateSensors(gm);
        //si preleva la posizione del pg
        int [] pg_pos = PlayableCharacter.getPGposition();
        //si aggiorna la mappa di esplorazione
        em.getMapCell(pg_pos[0], pg_pos[1]).
        copyCellSpecs(gm.getMapCell(pg_pos[0],pg_pos[1]));
}//init()
```

Quindi, il posizionamento effettivo degli elementi di gioco viene realizzato, rispettivamente, dalla funzione *placeGameElements(GameMap)*, per quanto riguarda tutti gli elementi ad eccezione del personaggio giocabile e dalla funzione *placePGonCorner(GameMap)*, per il posizionamento di quest'ultimo, dopo che è stata determinata una configurazione valida della matrice di gioco.

Una configurazione risulta idonea se garantisce al PG la possibilità di raggiungere tutti gli elementi di gioco e la sua piena libertà di movimento, durante l'esplorazione, per tutte le caselle della mappa di gioco, ad eccezione di quelle vietate, se presenti.

La funzione che si occupa del posizionamento degli elementi di gioco è la seguente:

```
private static void placeGameElements(GameMap gm) {
        //numero di celle della mappa di gioco
        int n_cells=GameElements.getNumberOfCells();
        int cells=n_cells;
        //sassi
        int n_stones=GameElements.getStones();
        int stones=n_stones;
        //nemico
        int n_enemy=GameElements.getEnemy();
        int enemy=n_enemy;
        //pericolo
        int n_danger=GameElements.getDanger();
        int danger=n_danger;
        //premio
        int n_award=GameElements.getAward();
        int award=n_award;
        //variabili che conterranno la probabilita'
        double pdanger=0;
        double penemy=0;
        double pstones=0;
        double paward=0;
        //variabile ausiliaria per il random
        double random=0;
        //variabile ausiliaria per la corretta posizione del premio
        boolean correct_award = false;
        //variabili ausiliarie per gli indici della cella in cui e' stato posizionato il
        premio
        int i_award = -1;
        int j_award = -1;
        //ciclo di riempimento della mappa
```

```
	do {
		//si riassegnano alle variabili i valori di default
		//in modo da resettare la situazione se la configurazione ottenuta
		//per la mappa di gioco non e' idonea
		danger=n_danger;
		enemy=n_enemy;
		cells=n_cells;
		stones=n_stones;
		award=n_award;
		//svuotare la matrice per ripristinarla alla situazione iniziale
		gm.clear();
		//for righe
		for(int i=0;i<gm.getRows();i++) {
			//for colonne
			for(int j=0;j<gm.getColumns();j++) {
				//si genera un numero casuale (da 0 a 1) come soglia
				random = Math.random();
				//calcolo delle probabilita' per ogni tipologia di cella
				pdanger = prob(danger, n_danger, cells, n_cells);
				penemy = prob(enemy, n_enemy, cells, n_cells);
				pstones = prob(stones, n_stones, cells, n_cells);
				paward= prob(award, n_award, cells, n_cells);
				//si preleva la cella attuale
				Cell c = gm.getMapCell(i, j);
				//confronto delle probabilita' con la soglia random
				if(random < pdanger) {
					//la cella e' un pozzo/trappola
					c.setCellStatus(CellStatus.DANGER);
					//un elemento e' stato posizionato
					danger=danger-1;
				}//fi pericolo
				else if(random < penemy) {
					//la cella conterra' l'avversario del pg
					c.setCellStatus(CellStatus.ENEMY);
					//un elemento e' stato posizionato
					enemy=enemy-1;
				}//fi nemico
				else if(random < pstones) {
					//la cella non sara' giocabile
					c.setCellStatus(CellStatus.FORBIDDEN);
					//un elemento e' stato posizionato
					stones=stones-1;
				}//fi sasso
				else if(random < paward) {
					//la cella conterra' il premio
					c.setCellStatus(CellStatus.AWARD);
					// l'elemento e' stato posizionato
					award=award-1;
					//si memorizzano gli indici
					i_award = i;
					j_award = j;
				}//fi premio
				else {
					//la cella e' etichettata come sicura, libera
					c.setCellStatus(CellStatus.SAFE);
				}//esle libera
				//si impostano gli indici della posizione della cella
				c.setPosition(i,j);
				//si decrementa il numero di celle rimaste da riempire
				cells=cells-1;
			}//for colonne
		}//for righe
		//si controlla se il premio e' stato posizionato in una cella accessibile
		correct_award = gm.isAccessibleCell(i_award,j_award);
		//controllo dei flag del ciclo
	}while( enemy>0 || danger>0 || stones>0 || award>0 || !correct_award);
}//placeGameElements(GameMap, int[])
```

Come si evince dal codice, in questo metodo, dopo aver assegnato ai parametri i valori degli elementi da posizionare, viene effettuato un ciclo che verrà ripetuto fino a quando non si riuscirà a determinare una configurazione del terreno di gioco che sia in grado di soddisfare tutte le condizioni.

Allora, la scelta della tipologia da assegnare ad ogni cella della matrice avviene calcolando la probabilità per cui una data cella possa essere etichettata con il valore che si sta prendendo in considerazione.

Nello specifico, se il valore ottenuto tramite la funzione *prob(int, int, int, int)* risulta maggiore del valore della soglia, allora l'esito di questa valutazione sarà positivo e la cella in esame verrà caratterizzata con quella determinata tipologia, modificando il valore del suo attributo *cellStatus*.

La funzione che implementa il calcolo della probabilità è:

```
private static double prob(int x,int max_x, int n, int max_n) {
	//controllo sui parametri
	if(x==0)return 0;
	//numero casuale
	double random = Math.random();
	//funzione di probabilita'
	double prob = ((x/max_x) - (n/max_n) + random*0.3) /3;
	//si restituisce il valore calcolato
	return prob;
}//prob(int, int, int, int)
```

Si precisa che nella funzione di probabilità definita come:

$$\frac{(\frac{x}{x_{max}} - \frac{n}{n_{max}} + random * 0.3)}{3}$$

la variabile *random* rappresenta un numero casuale, generato utilizzando la funzione *Math.random()*, che è stato inserito per dare un minimo di varianza alla funzione stessa.

Per quanto riguarda le altre variabili, il significato è il seguente:

- x è il numero di oggetti di tipo X rimanenti, che devono ancora essere posizionati nella mappa;
- x_{max} è il numero massimo di oggetti di tipo X che possono essere posizionati;
- n è il numero di celle della mappa che devono ancora essere riempite;
- n_{max} è il numero delle celle che compongono la mappa di gioco;

Gestione della sessione di gioco

Dopo aver configurato il terreno di gioco, in base alla scelta effettuata dall'utente, in merito all'intenzione di eseguire la modalità interattiva oppure automatica dell'applicazione, sarà necessario far partire la sessione di gioco.

Dunque, sono state previste due differenti tipologie di gioco, ovvero quella in cui l'utente muove il PG sulla mappa, inserendo degli input da console (oppure da periferica di input touch, sul dispositivo Android) oppure la modalità di gioco automatica, in cui l'esplorazione della matrice verrà eseguita da un agente software.

Il concetto di giocatore è stato concretizzato tramite l'implementazione della classe *Player*, che dispone, oltre del metodo che consente di scegliere in che ruolo giocare, se impersonando il Wumpus oppure l'avventuriero, di due metodi che inglobano quelli che sono stati definiti, rispettivamente, per il giocatore controllato dall'utente, nella classe *GameSession* e per il giocatore automatico, nella classe denominata *AutomaticGameSession*.

La struttura dei due metodi in questione è analoga, poiché vanno a richiamare, entrambi, i metodi *start()*, *play()* ed *end()*. Tuttavia, differiscono nell'implementazione, soprattutto per quanto riguarda il metodo che si occupa della gestione vera e propria della partita.

Si riporta, a riprova di quanto detto, il codice del metodo *play*() della classe GameSession:

```
public static void play() {
	//variabile ausiliaria per il colpo
	Direction dir;
	//variabile asusiliaria per la mossa
	Direction move;
	//variabile ausiliaria per lo stato della mossa
	int status = 0;
	//avvio della partita
	while(Starter.getGameStart()){
		//acquisizione dell'azione del giocatore
		Starter.chooseMove();
		//si verifica se si vuole sparare
		if(Starter.getTryToHit()) {
			//si verifica se si ha un colpo a disposizione
			if(Starter.getChanceToHit()) {
				//si chiede la direzione in cui sparare
				Starter.chooseDirection();
				//si preleva la direzione
				dir = Starter.getShotDir();
				//si tenta il colpo
				Controller.hitEnemy(dir, gm);
				//il colpo a disposizione e' stato utilizzato
				Starter.setChanceToHit(false);
			}//fi
			//non si ja piu' la possibilita' di colpire
			else {
				System.out.println(GameMessages.no_hit);
			}//esle
			//reset flag relativo al tentare il colpo
			Starter.setTryToHit(false);
		}//fi
		//si verifica se si deve spostare il personaggio
		if(Starter.getWalk()) {
			//si preleva la direzione in cui muovere il pg
			move = Starter.getPGmove();
			//verifica dell'azione
			status = Controller.movePG(move, gm, em);
			//si effettua la mossa
			Controller.makeMove(status, gm, em);
			//si stampa la mappa con la legenda
			if(status==0)System.out.println(em.mapToStringAndLegend());
			//reset flag
			Starter.setWalk(false);
		}//fi
		//punteggio parziale
		System.out.println("Punteggio attuale: "+score.printScore());
	}//end while sessione di gioco
}//play()
```

Le chiamate che in questo codice vengono fatte alla classe *Starter* vengono utilizzate per gestire alcune configurazioni della sessione di gioco, sia prima che questa venga avviata, sia mentre l'utente sta giocando.

Questa classe cura l'aggiornamento della posizione del personaggio giocabile, il reset dei flag che indicano l'inizio della partita e la disponibilità del tentativo di tiro, cioè il possesso o meno dell'arma con cui provare a colpire il nemico.

Il metodo corrispettivo a quello descritto sopra, che è contenuto, però, nella classe *AutomaticGameSession*, è il seguente:

```
public static void play() {
	//risoluzione
	int status = AutomaticPlayer.solveGame(gm, em, run_path);
	//messaggio di fine partita
	System.out.println(AutomaticPlayer.printStatusMessage(status));
	//si stampa la mappa di esplorazione
	System.out.println("Ecco cosa ho visto:\n"+em.gameMapToString());
	//si mostra il percorso compiuto
	System.out.println("Path "+AutomaticPlayer.runPathToString(run_path));
}//play()
```

Questo metodo effettuerà, per risolvere la partita e quindi terminare la sessione di gioco dell'agente intelligente, delle chiamate ai metodi della classe in cui è stato implementato l'algoritmo che costituisce il fulcro del funzionamento del giocatore automatico.

Si riporta, quindi, uno schema delle classi coinvolte in queste operazioni, includendo quelle del calcolo e del salvataggio del punteggio ottenuto dall'utente e dell'effettiva esecuzione della mossa stabilita, il cui controllo e realizzazione viene gestito dalla classe *Controller*.

Ecco il diagramma UML delle classi contenute nel package *game.session*:

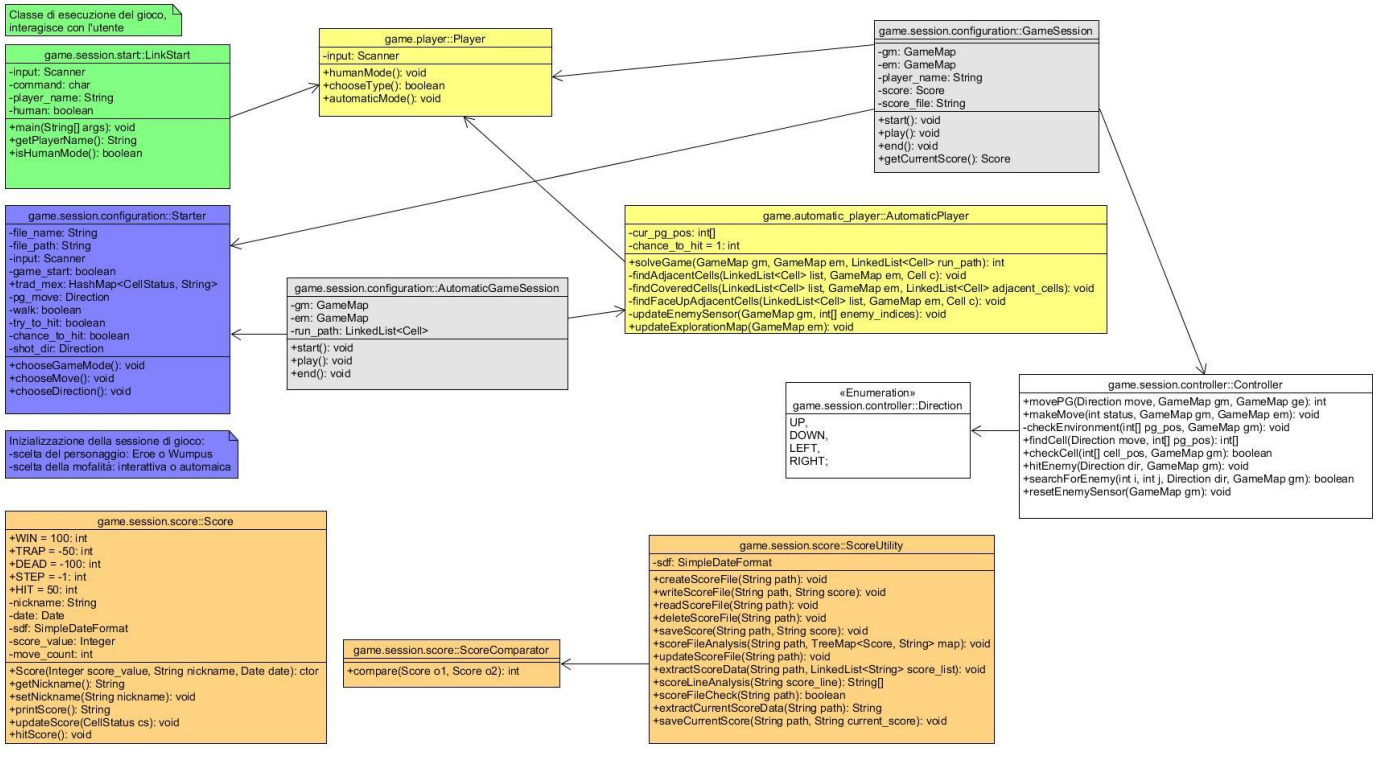


Figura 4. Diagramma UML delle classi contenute nel package game.session.

Avvio della sessione di gioco

L'applicazione verrà avviata mandando in esecuzione la classe *LinkStart*, che mostrerà all'utente i comandi che può utilizzare, tramite una schermata introduttiva del tipo:

```
Loading the G4M3.......
Link... Start-o!

Ecco la lista dei comandi:

[q - quit] [g - game start] [s - score] [c - credits] [n - set player name]

Cosa vuoi fare?
```

Dunque, l'utente potrà chiudere l'applicazione o visualizzarne i crediti, avviare una nuova partita oppure inserire il nome che vuole utilizzare come giocatore, all'interno del gioco.

Dopo di che, nel caso in cui abbia scelto di iniziare a giocare, dovrà inserire alcune specifiche, richieste in questo modo:

```
Vuoi metterti alla prova? [y - yes] [n – no]
```

Se la risposta sarà affermativa, allora verrà creata una mappa ed avviata una sessione di gioco in cui l'utente potrà muovere il personaggio che ha scelto.

La decisione del personaggio avverrà tramite la seguente finestra:

```
Uh uh, coraggio allora!
Link... Start-o!
Ciao :3
Dimmi di te...Sarai il cacciatore oppure il Wumpus?
[h - cacciatore] [w - wumpus]
```

Dopo aver effettuato la scelta del PG, verranno, infine, spiegati i comandi di gioco, mostrando a schermo delle informazioni di questo tipo:

```
      MAPPA                                  LEGENDA

 ||X| |X| |X| |X||          ------------------------------------------
 ||X| |X| |X| |X||         | X = LUOGO DA VISITARE | O = LUOGO VISITATO |
 ||X| |X| |X| |X||         | P = GIOCATORE         | F = LUOGO VIETATO  |
 ||X| |P| |X| |X||          ------------------------------------------

 Comandi:

 w = sopra
 a = sinistra
 s = sotto
 d = destra
 i = interrompi partita
 l = prova a colpire il nemico
```

Inoltre, verranno riportati, in basso, la posizione attuale del giocatore e lo stato dei sensori, in modo da fornire, all'utente, delle informazioni sull'ambiente circostante.

Ad esempio, relativamente alla mappa sopradescritta, verrà visualizzato:

```
Ti trovi nella cella (3,1)
Tutto a posto all'orizzonte
```

Seguito dal path in cui è stato memorizzato, di default, il file di testo che conterrà i punteggi ottenuti.

Se, invece, l'utente non vorrà giocare in prima persona, allora la mappa che è stata creata per la sessione di gioco corrente verrà esplorata e risolta dal giocatore automatico.

Controllo delle mosse di gioco

Il controllo e la messa in atto dei comandi, inseriti dall'utente, utilizzati per controllare il personaggio giocabile sono stati affidati alla classe *Controller*.

Nello specifico, ci si dovrà occupare di:

- verificare se la direzione in cui si vuole fare muovere il PG sia valida (ad esempio, potrebbe non esistere alcuna cella per la direzione ricevuta come input);
- effettuare la mossa, spostando effettivamente il PG nella cella corrispondente alla direzione specificata;
- aggiornare le informazioni dei sensori, in merito alla nuova posizione del giocatore;
- visualizzare il contenuto del vettore dei sensori in modo che possa essere reso noto al giocatore;
- aggiornare la mappa di esplorazione, cioè la mappa nota al giocatore che, ovviamente, non è a conoscenza dalla mappa di gioco completa;
- visualizzare il risultato ottenuto dopo l'esecuzione della mossa richiesta ed aggiornare il punteggio del giocatore;

Il controllo della validità della mossa, quindi, sarà eseguito nel metodo *movePG(Direction, GameMap, GameMap*) che riceverà, come parametri, oltre la direzione da esaminare, la mappa nota all'utente e quella completa di tutti gli elementi di gioco.

Si specifica che per "direzione" si intende un parametro di tipo *Direction*, definito dall'enumerazione riportata di seguito:

```
public enum Direction {
	UP,
	DOWN,
	LEFT,
	RIGHT;
}//end Direction
```

Quindi, i movimenti consentiti sono quelli attuabili nelle quattro direzioni, cioè sopra, sotto, destra e sinistra, ricevuti tramite una periferica di input con disposizione WASD.

Per quanto riguarda, invece, l'esecuzione vera e propria del comando di movimento del PG, questo verrà realizzato dal metodo *makeMove(int, GameMap, GameMap)* che, inoltre, informerà l'utente del risultato della mossa.

La classe in questione gestisce, inoltre, anche il tentativo di sparo, ovvero la possibilità, messa a disposizione del giocatore, di provare a colpire in nemico, se si pensa di averne individuato la posizione.

Segue il codice del metodo principale, ovvero:

```java
public static int movePG(Direction move, GameMap gm, GameMap ge) {
	//vettore della posizione successiva
	int [] cell_pos= new int[2];
	//variabile da restituire
	int status = 0;
	//si preleva la posizione del pg
	int [] pg_pos = PlayableCharacter.getPGposition();
	//si controlla il risultato della direzione scelta
	cell_pos = findCell(move, pg_pos);
	//la cella indicata da next_pos esiste
	if(checkCell(cell_pos, gm)) {
		//si controlla il contenuto della cella in questione
		CellStatus cs = gm.getMapCell(cell_pos[0], cell_pos[1]).getCellStatus();
		//controllo sullo stato
	if(cs.equals(CellStatus.ENEMY)) {
		//il pg e' stato ucciso dal nemico
		status = 1;
		//si deve aggiornare la posizione del PG
		PlayableCharacter.setPGposition(cell_pos);
	}//fi
	else if(cs.equals(CellStatus.FORBIDDEN)) {
		//questa cella e' vietata perche' e' un sasso
		status = -1;
		//si aggiunge alla mappa di esplorazione
		ge.getMapCell(cell_pos[0], cell_pos[1]).
			copyCellSpecs(gm.getMapCell(cell_pos[0], cell_pos[1]));
		//il pg rimane dove si trova
	}//fi
	else if(cs.equals(CellStatus.AWARD)) {
		//il pg vince
		status = 2;
		//si aggiorna la posizione
		PlayableCharacter.setPGposition(cell_pos);
	}//fi
	else if(cs.equals(CellStatus.DANGER)) {
		//il pg e' caduto nella trappola
		status = 1;
		//si aggiorna la posizione
		PlayableCharacter.setPGposition(cell_pos);
	}//fi
	else {//CellStatus.SAFE
		//il pg si trova in una cella libera
		status = 0;
		//la cella in cui si trovava prima il pg si segna come visitata
		ge.getMapCell(pg_pos[0], pg_pos[1]).
			setCellStatus(CellStatus.OBSERVED);
		//si aggiorna la posizione del pg
		PlayableCharacter.setPGposition(cell_pos);
		//contenuto della cella in cui si trova attualmente il pg
		Cell c = gm.getMapCell(cell_pos[0], cell_pos[1]);
		//si copia questa cella nella matrice di esplorazione
		ge.getMapCell(cell_pos[0], cell_pos[1]).copyCellSpecs(c);
		//nella mappa di esplorazione il pg e' in questa cella
		ge.getMapCell(cell_pos[0], cell_pos[1]).
			setCellStatus(CellStatus.PG);
	}//esle
	//variabile punteggio
	Score s = new Score();
	//si aggiorna il punteggio nella modalita' interattiva
	if(LinkStart.isHumanMode()) {
		//aggiornamento del punteggio
		s = GameSession.getCurrentScore();
	}//fi
	//aggiornamento del punteggio
	s.updateScore(cs);
	}//fi indici di mossa corretti
	else {
		//comando non valido, oppure la cella non esiste
		status = -2;
	}//esle
```

```
        //si restituisce il codice associato al tipo di mossa
        return status;
}//movePG(Direction, GameMap, GameMap)
```

Si può notare come, dopo la verifica dell'esistenza della cella di interesse, che corrisponde, cioè, alla direzione in cui il giocatore ha deciso di spostarsi, verrà controllato il contenuto della cella obiettivo. Questo perché bisogna determinare l'esito della mossa, in modo da aggiornare lo stato della partita e quindi il punteggio del giocatore. Il risultato di questa valutazione, allora, sarà rappresentato dalla variabile *status*, il cui valore potrà essere pari ad uno di questi interi, ovvero:

- 1, se il giocatore è finito nella cella del nemico o in cui è posizionato un pericolo, perdendo la partita;
- 2, se il PG si trova nella cella con il premio, ottenendo la vittoria;
- -1, se il comando ricevuto non è valido oppure ha condotto ad una cella non accessibile;
- 0, se la mossa da effettuare è valida, indicativa di una cella sicura, che costituirà la nuova posizione del giocatore sulla mappa;

Giocatore automatico

Il giocatore automatico, rappresentato dalla classe *AutomaticPlayer*, è stato progettato per risolvere la partita attuale, dopo che l'utente ha specificato il ruolo che dovrà avere il PG.

Non è stata resa disponibile la funzionalità di memorizzare il punteggio ottenuto durante una sessione di gioco automatica, perché attualmente ritenuta superflua, anche se il codice è stato predisposto a tale fine.

Il meccanismo di funzionamento dell'agente intelligente, pur non essendo ottimale, porta al conseguimento della vittoria nella maggior parte dei casi.

L'algoritmo che ne implementa il comportamento, infatti, tenta di procedere all'esplorazione della matrice di gioco sulla base di statistiche che gli consentano di prevedere quale sia il contenuto delle celle adiacenti, tenendo conto delle informazioni acquisite fino a quel momento.

Nello specifico, se si stima, per la cella che si vuole visitare, un'elevata probabilità di incorrere in una situazione di rischio, costituita dai vari pericoli disseminati sulla mappa, allora il giocatore automatico preferirà muoversi in una cella che ha visitato in precedenza, piuttosto che scegliere di posizionarsi in una casella coperta, cioè una zona della mappa che è ancora inesplorata.

Per effettuare queste valutazioni, l'algoritmo dell'agente software terrà conto di tutte le informazioni di cui dispone, ampliando la sua conoscenza relativa all'ambiente circostante.

In questo processo il ruolo chiave, quindi, è costituito dai sensori, che forniranno all'agente una panoramica sui pericoli che potrebbe incontrare, intraprendendo una determinata direzione.

Dunque, la strategia adottata consiste nel muoversi sempre nella direzione più sicura, in modo da garantire il proseguimento della partita.

Nel momento in cui questo non sarà più possibile perché, qualunque sia il percorso che tenta di seguire, il giocatore automatico si troverà sempre circondato da celle non sicure, allora, per evitare una situazione di stallo, questo compirà comunque una mossa.

Nel dettaglio, considerando la cella in cui si vuole fare lo spostamento successivo, se ad essere acceso è il sensore che indica la presenza del nemico, allora l'agente tenterà di colpirlo, scegliendo una direzione casuale, che non includa, però, nessuna delle caselle che ha già visitato.

In questo modo, se il colpo è andato a segno ed è dunque riuscito a liberarsi la via verso un nuovo percorso sicuro, l'agente adotterà nuovamente la *strategia iniziale*, ovvero quella di scegliere casualmente una cella, tra quelle non ancora visitate, che risulti sicura e nel caso in cui non ci sia alcuna direzione che soddisfi questa condizione, di scegliere una cella a caso, tra quelle già visitate e quindi contenute nella matrice di esplorazione.

Se, invece, non fosse possibile adottare questo comportamento, perché non esiste una sola cella adiacente a quella in cui si trova il PG (è il caso di una mossa non valida) oppure tutte le celle adiacenti sono potenzialmente pericolose, il giocatore automatico si troverà costretto ad adottare una *strategia di emergenza*, decidendo comunque di proseguire la sua esplorazione del terreno di gioco, scegliendo casualmente la direzione verso cui muoversi e tenendo in conto, così, il rischio di incorrere in una situazione di pericolo e di perdere la partita.

Il metodo dedito alla risoluzione è riportato sotto:

```
public static int solveGame(GameMap gm, GameMap em, LinkedList<Cell> run_path) {
	System.out.println("Risolvo!");
	//si inserisce la cella di partenza
	run_path.add(gm.getMapCell(cur_pg_pos[0], cur_pg_pos[1]));
	//la posizione precedente e' uguale a quella corrente nella prima mossa
	//si richiama il metodo ricorsivo
	return solvingGameFirstStrategy(cur_pg_pos, cur_pg_pos, gm, em, run_path);
}//end solveGame
```

Come si evince dal codice, è stato implementato un algoritmo di back-tracking.

Il metodo *solvingGameFirstStrategy(int[], int[], GameMap, GameMap, LonkedList<Cell>)*, inoltre, in base alla situazione in cui si andrà a trovare il giocatore, potrà fare delle chiamate al metodo *solvingGameEmergencyStrategy(int[], int[], GameMap, GameMap, LinkedList<Cell>)*, effettuando il cambio di strategia descritto in precedenza.

Calcolo del punteggio

La gestione del punteggio, in termini di operazioni che riguardano il calcolo effettivo e la sua memorizzazione in un file di testo, in cui verrà tenuta nota anche del nome del giocatore e dalla data in cui è stato conseguito il suo risultato, è compito delle classi contenute nel package *game.session.score*.

Nello specifico, se ad occuparsi della definizione dell'oggetto punteggio sarà la classe *Score*, a gestire il salvataggio di questi dati su file sarà la classe di utilità denominata *ScoreUtility*.

In questa classe, infatti, sono stati definiti ed implementati tutti i metodi fondamentali per la scrittura, lettura e cancellazione di un file di testo, oltre che i metodi necessari alla gestione dei dati da memorizzare, riguardanti, ad esempio, l'ordinamento dei punteggi in maniera decrescente e l'aggiornamento di questi, all'inserimento di un nuovo record.

Questo confronto tra punteggi differenti è reso possibile dall'utilizzo della classe *ScoreComparator* che, implementando l'interfaccia *Comparator*, ha permesso di definire un criterio di comparazione sulla base del punteggio effettivo ed a parità di questo, della data in cui è stato conseguito; a parità di punteggio, infatti, verrà preferito quello più recente.

Si riporta proprio il metodo che costituisce lo strumento di comparazione:

```
@Override
public int compare(Score o1, Score o2) {
	//controllo sui parametri
	if(o1==null && o2==null)return 1;
	if(o1==null)return -1;
	if(o2==null)return 1;
	//confronto per integer
	if(o1.getScore()>o2.getScore())return 1;
	if(o1.getScore()<o2.getScore())return -1;
	//se i punteggi sono uguali si confronta per data
	return o1.getDate().compareTo(o2.getDate());
}//compare()
```

Si precisa che un oggetto di tipo *Score* avrà come campi, cioè come attributi di classe, una stringa che rappresenta il nickname scelto dal giocatore, la data in cui ha ottenuto il punteggio, il valore vero e proprio del punteggio, memorizzato come *Integer* ed il numero di mosse che ha compiuto durante la sessione di gioco.

Quest'ultimo parametro verrà direttamente utilizzato nel calcolo del punteggio effettuato, tenendo conto delle azioni compiute dal personaggio giocabile durante la partita e dalle conseguenti penalità o ricompense ottenute, quali:

- **+100**, è il valore che viene aggiunto al punteggio, se si trova la posizione del premio **AWARD**, qualunque sia la modalità di gioco scelta (WIN = 100);
- **-50**, è la penalità subita se il personaggio giocabile si posiziona in corrispondenza di un pericolo **DANGER**, sia questo il pozzo oppure la trappola, a seconda della modalità in cui si sta giocando, (TRAP = -50);
- **-100**, è il valore che viene sottratto al punteggio complessivo se il giocatore incontra il nemico **ENEMY**, ovvero se viene ferito dal Wumpus, se sta impersonando l'Avventuriero, oppure se viene ucciso dal cacciatore, se si sta giocando nel ruolo del Wumpus (DEAD = -100);
- **-1**, è la penalità che viene assegnata per ogni mossa compiuta, cioè per ogni cella in cui verrà scelto di posizionare il personaggio giocabile (STEP = -1);

Generazione della libreria

Il progetto Java "**Hunt The Wumpus Or Not**", descritto in questo elaborato, è stato necessario per fornire una struttura di base all'applicazione Android che si voleva realizzare.

Infatti, avendo realizzato il gioco in maniera testuale ed avendo così avuto modo di testarne le meccaniche ed il funzionamento, si è potuto dotare l'applicazione Android di un back-end che fosse sufficientemente solido.

Il ruolo di questo progetto, all'interno dell'applicazione Android, sarà, infatti, quello di una libreria vera propria, di cui utilizzare metodi e servizi all'interno delle varie attività che andranno a costituire l'interfaccia utente.

Per rendere il progetto esaminato finora una libreria è stato necessario esportarne il codice Java in un file .jar, denominato "**huntthewumpusornot**" e poi importarlo nel progetto dell'applicazione mobile come dipendenza, inserendolo come modulo Android.

Applicazione Android

Questo progetto, realizzato per dispositivi mobili su cui gira il sistema operativo Android, costituisce, quindi, un'implementazione personale del noto gioco testuale "Hunt the Wumpus".

La versione del gioco che è stata realizzata, le cui caratteristiche sono state descritte in precedenza, ha come fulcro proprio la libreria scritta in Java.

Tuttavia, questa struttura, che rappresenta il back-end dell'applicazione mobile, è stata integrata con un'interfaccia utente realizzata sfruttando le librerie e le funzionalità messe a disposizione dal kit di sviluppo Android.

Caratteristiche dell'applicazione

Il nome dell'applicazione Android è "Wumpus World Game", o "Il mondo del Wumpus", a seconda della lingua di sistema che è stata impostata sul dispositivo.

Volendo esportare e distribuire il file apk, cioè il file eseguibile dell'applicazione, si è provveduto a firmarlo digitalmente.

Il procedimento è stato effettuato tramite l'ambiente di sviluppo Android Studio, scegliendo la voce "Generate Signed Bundle/ APK" dal menu "Build".

Dopo di che è stato creato un nuovo keystore, ovvero un file binario dedito all'archiviazione delle chiavi, specificando nome e percorso di memorizzazione in locale. A questo punto bisognerà impostare una password, che valga sia per lo stesso keystore e sia per la chiave che dovrà essere generata.

Quindi, l'applicativo "Wumpus World Game" sarà dotato di un certificato digitale contenente la chiave pubblica che costituisce la coppia di chiavi, rispettivamente pubblica e privata, impostata prima, oltre alle informazioni del proprietario della chiave, in questo caso lo sviluppatore, che custodisce la chiave privata.

In questo modo è stata garantita l'autenticità dei successivi aggiornamenti del file apk, certificando che questi sono stati forniti dallo sviluppatore, che ha apposto, con questo procedimento, la sua firma sull'applicazione.

Questa firma avrà una validità pari alla durata che è stata specificata nel processo di generazione della chiave e consentirà agli utenti di installare ogni nuova versione dell'applicazione pubblicata sullo store Google Play, come se si trattasse di un aggiornamento.

Ovviamente, affinché lo sviluppatore venga riconosciuto come fonte attendibile da cui installare applicativi ed aggiornamenti, dovrà essere registrato come tale, effettuando una vera e propria iscrizione sul sito di Google riservato agli sviluppatori, cioè *Developer Console*.

Casi d'uso

Per casi d'uso si intende la descrizione dell'insieme di interazioni che si possono verificare tra un utente ed un sistema, per consentire all'utilizzatore del software di eseguire delle azioni tramite le funzionalità di cui questo dispone.

Segue la definizione dei casi d'uso principali dell'applicazione mobile descritta in questo elaborato:

1. L'utente sceglie la modalità in cui avviare la partita, se Avventuriero o Wumpus;
2. Dopo aver scelto la modalità di gioco, l'utente visualizza la schermata di gioco, in cui potrà muovere il suo PG sulla mappa tramite i tasti direzionali e potrà scagliare una freccia tramite il pulsante centrale;
3. Al termine della partita l'utente potrà decidere se condividere o meno il punteggio ottenuto, selezionando la sua preferenza nella finestra che verrà visualizzata;

Estensioni dei casi d'uso riportati sopra:

1a. L'utente può accedere, tramite il menu, a differenti schermate, indicate dalle voci: "Informazioni", "Tutorial", "Punteggi" ed "Impostazioni";
   1. L'utente seleziona la voce "*Informazioni*" e visualizza una descrizione del gioco;
      a. L'utente ritorna alla schermata precedente tramite il pulsante di navigazione "Indietro".
   2. L'utente seleziona la voce "*Tutorial*" e visualizza una descrizione della storia e dei personaggi del gioco;
      a. L'utente seleziona il tasto "Avanti" e visualizza una spiegazione dei comandi generali del gioco;
         i. L'utente seleziona il tasto "Indietro" e ritorna alla schermata in cui è illustrata la storia del gioco;
      b. L'utente ritorna alla schermata precedente tramite il pulsante di navigazione "Indietro".
   3. L'utente seleziona la voce "*Punteggi*" e visualizza la classifica dei punteggi che ha ottenuto fino a quel momento.
      a. Cliccando il pulsante di condivisione nella sezione "Record", l'utente può condividere il suo miglior punteggio;
      b. Cliccando il pulsante di condivisione nella sezione "Ultimo punteggio", l'utente può condividere il suo ultimo punteggio;
      c. L'utente ritorna alla schermata precedente tramite il pulsante di navigazione "Indietro".
   4. L'utente seleziona la voce "*Impostazioni*" ed accede alle impostazioni dell'applicazione.
      a. Nella scheda "Generali" l'utente può:
         i. abilitare oppure disattivare gli effetti sonori dell'applicazione;
         ii. modificare il suo nome da giocatore;
      b. Nella scheda "Dati e Sincronizzazione" l'utente può:
         i. Importare i dati di gioco selezionando dall'archivio del dispositivo un file .txt contenente i punteggi precedenti;
         ii. Esportare i dati di gioco scegliendo la locazione in cui memorizzare il file contenente i punteggi ottenuti fino a quel momento;
         iii. Eliminare i dati di gioco cancellando il file dei salvataggi;
      c. Nella scheda "*About*" l'utente può:
         i. Inviare un feedback allo sviluppatore dell'applicazione tramite posta elettronica;
         ii. Visualizzare le informazioni dello sviluppatore ed i crediti dell'applicazione;

d. L’utente ritorna alla schermata precedente tramite il pulsante di navigazione “Indietro”.

2a. L’utente, durante la sessione di gioco, può accedere al menu, costituito dalle voci “Nuova partita”, “Giocatore Automatico” e “Comandi di gioco”.

1. L’utente seleziona la voce “*Nuova Partita*” per avviare una nuova sessione di gioco; perciò, la mappa viene generata da capo;
2. L’utente seleziona la voce “*Giocatore Automatico*”, avviando la risoluzione automatica della mappa di gioco, sfruttando l’agente;
3. L’utente seleziona la voce “*Comandi di gioco*”, visualizzando, così, i comandi di gioco specifici per la modalità che ha scelto;

Testing

Per quanto riguarda la fase di testing dell’applicazione, si precisa che questa sia stata installata ed eseguita su differenti dispositivi mobili fisici.

Uno di questi è un cellulare con sistema operativo Android di versione 10, ovvero un Huawei Mate 20 Lite, dotato di uno schermo di sei pollici.

Ecco le schermate delle attività più significative dell’applicazione:

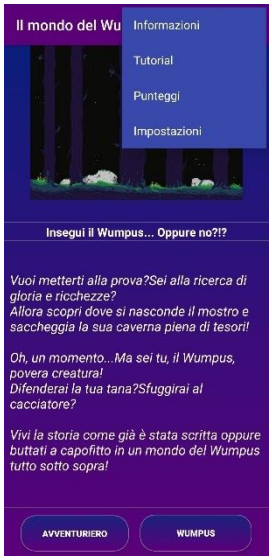


Figura 5. Main Activity (Launcher) con Menu.

Figura 6. Schermata di gioco della modalità Eroe con Menu.

Figura 7. Schermata di risoluzione automatica nella modalità Wumpus.

Figura 8. Attività che visualizza i punteggi ottenuti dal giocatore.

Emulazione

Avendo sviluppato l’applicazione su Android Studio, è stato possibile testarla tramite l’emulatore messo a disposizione da questo IDE.

Questo componente, denominato Android Virtual Device, consente di creare vari dispositivi virtuali, differenti per caratteristiche hardware e software, su cui testare l’applicazione sviluppata, senza necessità di avere a disposizione un dispositivo fisico.

In questo caso, pur avendo utilizzato le Android API 32, si è voluta simulare l'esecuzione dell'app su un dispositivo con sistema operativo Android Oreo 8.1 che, dunque, utilizza le Android API 27, come il Pixel 3a XL.

Struttura del front-end

L'applicazione è stata progettata in maniera tale che fosse semplice ed intuitiva per l'utente, implementando due differenti modalità di gioco e mettendo a disposizione la possibilità di vedere la soluzione di ogni partita avviata, lasciando che sia un agente software a concludere la sessione di gioco corrente.

Per quanto riguarda la struttura del progetto, si è cercato di rispettare quelle che sono le convenzioni adottate nello sviluppo di applicazioni Android, suddividendo, nei rispettivi package, i componenti dello stesso tipo, come si è fatto, ad esempio, per la definizione dell'aspetto di ogni activity, descritto in un file .xml contenuto in *layout* o per la memorizzazione delle immagini utilizzate, contenute nella cartella *drawable*, insieme ai file .xml che realizzano gli sfondi delle attività e dei loro componenti.

Diagramma delle classi

Per meglio descrivere l'organizzazione dei file di progetto si riporta il diagramma UML delle classi principali dell'applicazione, scritte in Java e che implementano le attività visualizzate dall'utente.

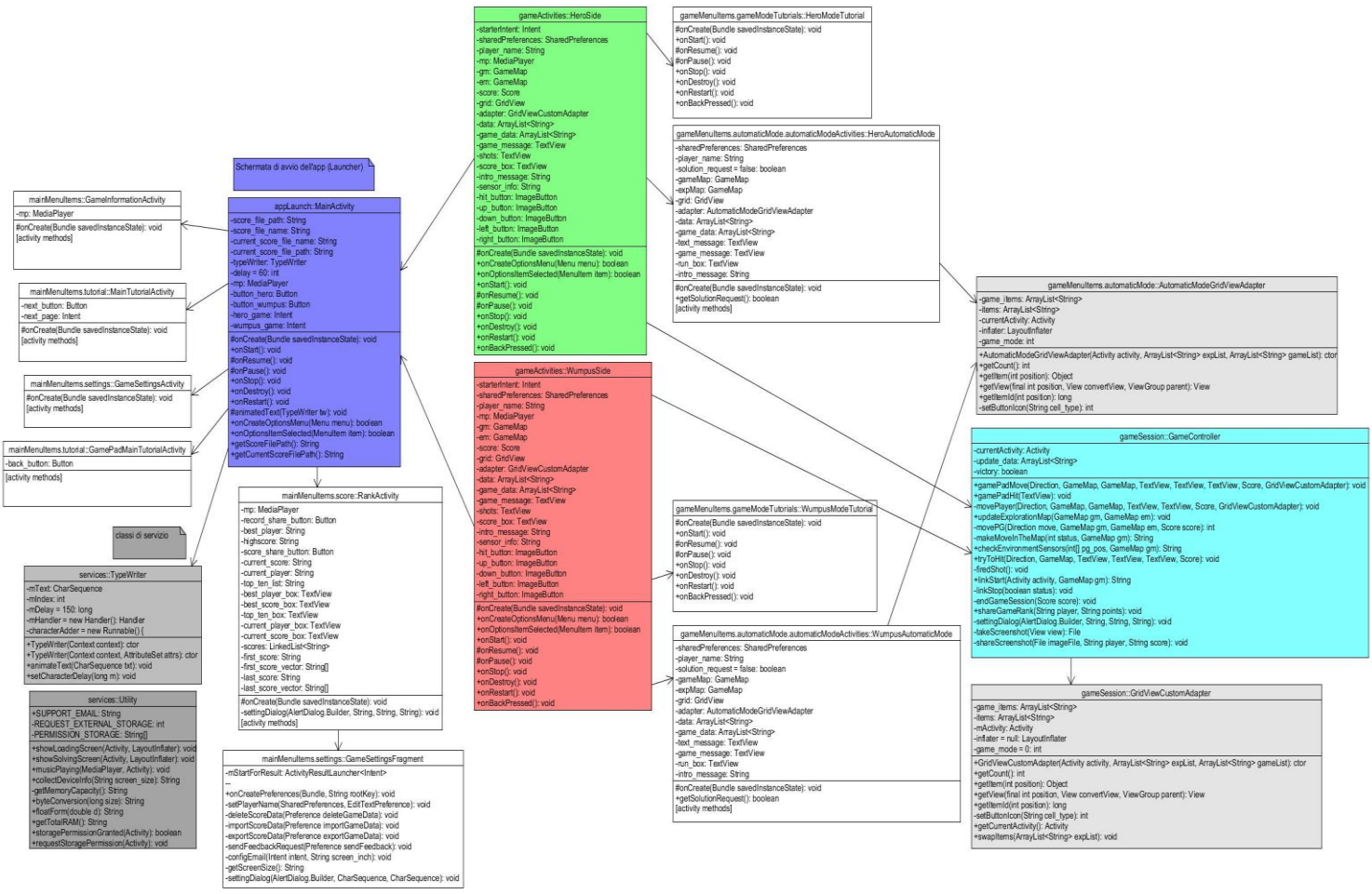


Figura 9. Diagramma UML delle classi del progetto Android.

Per quanto riguarda le relazioni tra le classi scritte in Java e quelle fornite dalle librerie Android, si avranno le seguenti dipendenze:

- Le classi di tipo *Activity*, ovvero quelle che rappresentano le schermate visualizzate dall'utente, estendono **AppCompatActivity**;
- Le classi che definiscono un componente di tipo *Adapter*, come AutomaticGridViewAdapter e GridViewCustomAdapter, utilizzate per creare degli adapter, cioè dei widget che consentono di visualizzare le informazioni che contengono, sullo schermo del dispositivo; per fare in modo che siano personalizzati per l'applicazione sviluppata, dovranno estendere la classe **BaseAdapter**;
- La classe GameSettingsFragment, utilizzata per definire la schermata delle impostazioni richiamata dall'activity GameSetting, essendo implementata adottando le convezioni stabilite dalla maggior parte degli sviluppatori Android, estende la classe astratta **PreferenceFragmentCompat**;
- La classe TypeWriter, scritta per realizzare un testo animato, che viene visualizzato, carattere per carattere, nell'attività principale, estende la classe **AppCompatTextView**, del package widget;

Per quanto riguarda l'**Activity**, ovvero il tipo di componente che, in una generica applicazione Android, utilizza il display del dispositivo per interagire con l'utente, si può dire che sia caratterizzata dal ciclo di vita riportato di seguito:

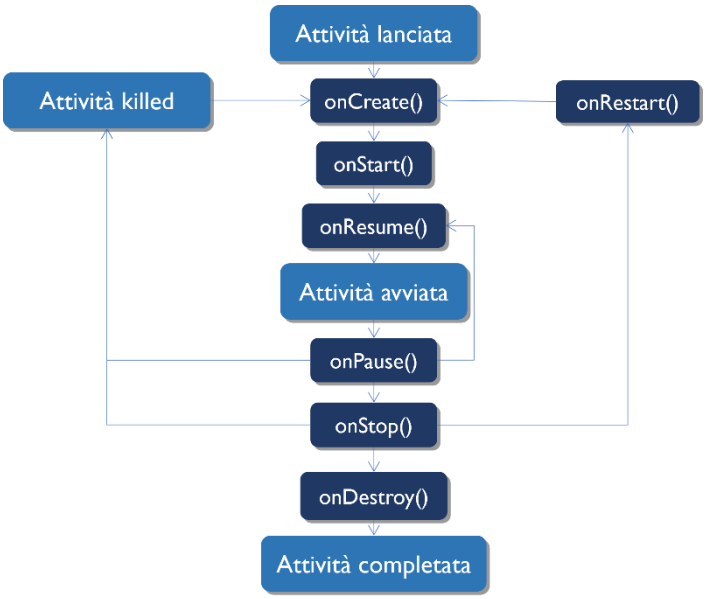


Figura 10. Ciclo di vita di un'attività.

Organizzazione delle risorse

Le risorse esterne, come i messaggi visualizzati nell'interfaccia utente o il layout definito per ciascuna activity, sono gestite tramite una determinata struttura, che ne prevede l'organizzazione ed il raggruppamento in specifiche cartelle, quali:

- *Src*, per quanto riguarda i package e le classi che realizzano l'applicazione vera e propria, dal punto di vista implementativo;
- *Res*, che racchiude le risorse esterne, come immagini e clip audio, utilizzate dall'applicazione.

Per quanto riguarda la cartella *res*, questa è composta, a sua volta, da alcune sottodirectory, ovvero:

- *Drawable*, che contiene le immagini e i file .xml che descrivono gli sfondi e l'aspetto dei componenti;
- *Layout*, in cui si trovano i file .xml utilizzati per definire i layout delle attività e dei widget;
- *Values*, che contiene gli stili ed i colori che sono stati impiegati nella definizione dei componenti dell'interfaccia utente, i messaggi di testo riportati nel file strings.xml per ogni lingua prevista e la cartella "themes", in cui sono raccolti il tema chiaro (light mode) e quello scuro (night mode);
- *Raw*, la cartella dove sono memorizzate le clip audio;
- *Menu*, in cui si trovano i file .xml che definiscono gli stili, rispettivamente, del menu principale e del menu della sessione di gioco;
- *Xml,* che contiene il file "PreferenceScreen" dedito alla creazione della schermata delle impostazioni;

Manifesto dell'applicazione

Considerato che un'applicazione Android viene distribuita tramite un file APK (Android Package), al suo interno verranno raccolti:

- l'eseguibile, in formato DEX, cioè di tipo Dalvik Executable, scritto, quindi, nel codice della Dalvik Virtual Machine;
- le eventuali risorse associate all'applicazione;
- dei descrittori che delineano il contenuto di questo pacchetto;

Uno di questi descrittori è il manifesto, il file "AndroidManifest.xml", in cui vengono dichiarate le attività, i servizi, i provider ed i receiver inclusi nel pacchetto, in modo tale che il sistema su cui verrà installato l'applicativo possa identificarli ed utilizzarli correttamente.

Si riporta il manifesto dell'applicazione "Wumpus World Game":

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.wumpusworldgame">
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <application
        android:requestLegacyExternalStorage="true"
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher_red_monster"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_red_monster_round"
        android:supportsRtl="true"
```

```
        android:theme="@style/Theme.AppCompat.DayNight">
        <!-- activity principale, launcher dell'app -->
        <activity android:name=".appLaunch.MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <!-- Game Activity: Hero Side and Wumpus Side -->
        <activity
            android:name=".gameActivities.HeroSide"
            android:label="@string/hero_side_title"
            android:launchMode="singleTop"
            android:parentActivityName=".appLaunch.MainActivity" />
        <activity
            android:name=".gameActivities.WumpusSide"
            android:label="@string/wumpus_side_title"
            android:launchMode="singleTop"
            android:parentActivityName=".appLaunch.MainActivity" />
        <!-- Settings -->
        <activity
            android:name=".mainMenuItems.settings.GameSettingsActivity"
            android:label="@string/game_settings_title"
            android:parentActivityName=".appLaunch.MainActivity" />
        <!-- Menu Activity -->
        <activity
            android:name=".mainMenuItems.info.GameInformationActivity"
            android:label="@string/game_info_title"
            android:parentActivityName=".appLaunch.MainActivity" />
        <activity
            android:name=".mainMenuItems.score.RankActivity"
            android:label="@string/score_title"
            android:parentActivityName=".appLaunch.MainActivity" />
        <!-- Automatic Player -->
        <activity android:name=".gameMenuItems.automaticMode.automaticModeActivities.He-
roAutomaticMode"
            android:label="@string/hero_automatic_mode"
            android:parentActivityName=".gameActivities.HeroSide"/>
        <activity android:name=".gameMenuItems.automaticMode.automaticModeActivities.Wum-
pusAutomaticMode"
            android:label="@string/wumpus_automatic_mode"
            android:parentActivityName=".gameActivities.WumpusSide"/>
        <!-- Game Tutorial -->
        <activity
            android:name=".mainMenuItems.tutorial.MainTutorialActivity"
            android:label="@string/main_tutorial_title"
            android:parentActivityName=".appLaunch.MainActivity" />
        <activity
            android:name=".mainMenuItems.tutorial.GamePadMainTutorialActivity"
            android:label="@string/main_tutorial_controls_title"
            android:parentActivityName=".mainMenuItems.tutorial.MainTutorialActivity" />
        <!-- Mode Tutorail -->
        <activity
            android:name=".gameMenuItems.gameModeTutorials.HeroModeTutorial"
            android:label="@string/hero_mode_tutorial_title"
            android:parentActivityName=".gameActivities.HeroSide" />
        <activity
            android:name=".gameMenuItems.gameModeTutorials.WumpusModeTutorial"
            android:label="@string/wumpus_mode_tutorial_title"
            android:parentActivityName=".gameActivities.WumpusSide" />
        <!-- File sharing provider -->
        <provider
            android:name="androidx.core.content.FileProvider"
            android:authorities="${applicationId}.provider"
            android:exported="false"
            android:grantUriPermissions="true">
            <meta-data
                android:name="android.support.FILE_PROVIDER_PATHS"
                android:resource="@xml/provider_paths" />
```

```
        </provider>
    </application>
</manifest>
```

Integrazione dei moduli in un progetto Git

Per realizzare questa integrazione di back-end e front-end, nell'applicazione Android vera e propria, sono stati fatti convergere i due rispettivi progetti in un unico macro-progetto, denominato "Hunt The Wumpus App", memorizzato su **Github**, un servizio di hosting per il software.

Questa distinzione ha permesso una maggiore flessibilità durante il processo di sviluppo, perché è stato possibile lavorare su parti differenti dell'applicazione, che poi sono state integrate dopo l'ultimazione della fase di testing.

In questo modo, non solo si è stati sicuri di realizzare un back-end che fosse stabile e funzionante, in quanto un progetto a sé stante ed indipendente dalla GUI, ma è stato possibile renderlo, esportandolo come file .jar, una libreria utilizzabile all'interno del progetto Android, come fonte esterna delle funzionalità fornite all'utente.

Questa separazione dei due livelli, cioè del differenziamento tra back-end e front-end, ha impedito di compromettere il funzionamento dell'intero progetto, nell'eventualità che si volessero testare delle nuove modifiche, sull'una o l'altra parte, oppure nel caso in cui fossero stati apportati dei cambiamenti non corretti.

È anche il processo di aggiornamento del codice ad essere semplificato tramite questo approccio, perché l'integrazione delle modifiche di una delle due sotto-parti con il progetto principale verrà eseguita automaticamente, incorporandone le variazioni, pur mantenendo le commit distinte.

Allora, si può lavorare su ognuno dei due moduli separatamente e convogliare le funzionalità di ciascuno in un unico progetto, ovvero quello che costituisce effettivamente l'applicazione "Wumpus' World Game".

I due moduli sono, quindi, il progetto implementato in Java, come versione testuale del gioco, locato al repository https://github.com/ivochan/HuntTheWumpusOrNot ed il progetto in cui è stata strutturata l'interfaccia utente, salvata nel repository https://github.com/ivochan/WumpusWorldGame.

Dunque, se si volessero, eventualmente, eseguire dei cambiamenti, per quanto riguarda la grafica e le funzionalità che essa offre all'utente, questi non andrebbero a compromettere il progetto finale, caricato in un repository differente, cioè https://github.com/ivochan/HuntTheWumpus_App, contenente i link ai due progetti che utilizza.

Sviluppi futuri

Per migliorare l'applicazione, oltre ad eventuali ottimizzazioni del codice, si è pensato di inserire alcune funzionalità aggiuntive, come la presenza dei super pipistrelli, così come previsto nella versione originale del gioco. In questo modo, durante la sessione di gioco, l'utente avrà una variabile in più di cui tenere conto, che potrebbe portarlo a dover cambiare strategia, visto che il PG, incontrando questo tipo di elemento sul terreno di gioco, verrebbe trasportato in un punto casuale della mappa.

Inoltre, si cercherà di implementare differenti algoritmi di risoluzione, in modo da dare la possibilità, allo stesso utente, di scegliere il tipo di intelligenza artificiale di cui verrà dotato l'agente software, cioè il giocatore automatico.

Per permettere una maggiore personalizzazione della sessione di gioco, in aggiunta, si vorrebbe definire un insieme di regole da adottare per la creazione e per la configurazione del terreno di gioco, in modo da lasciare che l'utente sia libero di impostare la dimensione della mappa, la difficoltà della partita, che verrà incrementata aumentando il numero di pericoli da posizionare sulla stessa e prevedere, di conseguenza, una serie di livelli, diversi per abilità richiesta e scenario.

Infine, si vorrebbero disegnare, in formato vettoriale, l'icona identificativa dell'applicazione e tutte le immagini rappresentative degli elementi di gioco, in maniera tale che siano originali, al contrario di quelle attualmente in uso che, invece, sono state prese, così come è stato fatto per le clip audio, dal sito https://opengameart.org.

Strumenti utilizzati

Si riportano, di seguito, le tecnologie utilizzate per la realizzazione del progetto:

- l'ambiente di sviluppo integrato, multi-linguaggio e multipiattaforma *Eclipse IDE* (versione 2021-03), per l'implementazione del codice del back-end, sviluppato in Java;

- l'ambiente di sviluppo integrato *Android Studio* (versione Bumblebee 2021.1.1) per applicazioni Android;

- il sistema di controllo di versione distribuito *Git*, per lavorare in remoto senza avere conflitti a causa di modifiche concorrenziali sul codice sorgente e per ripristinare la versione precedente in caso di cambiamenti che ne abbiano alternato la stabilità;

- il tool di modellazione *UMLet*, per creare i diagrammi UML rappresentativi delle classi principali, sia lato back-end che lato front-end;

- il linguaggio di programmazione Java, che si appoggia sull'omonima piattaforma software di esecuzione, con il relativo software di sviluppo, chiamato *Java Development Kit* (versione 8);

- il kit di sviluppo *SDK* completo di emulatore, librerie e documentazione per la piattaforma Android, il sistema operativo basato su kernel *Linux* (Android API 32);