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

Corso di Laurea Magistrale in Ingegneria dell'Automazione
A.A. 2021/2022

AGENTI SOFTWARE ED IMPLEMENTAZIONE IN ANDROID

DOCENTE: PROF. PUPO STUDENTE: IVONNE RIZZUTO

SOMMARIO

Agenti software

- Caratteristiche e proprietà di un agente
- Descrizione dei sistemi Multi-Agente

Teoria dei giochi

- Elementi di teoria dei giochi
- Strategie e paradossi

Android OS

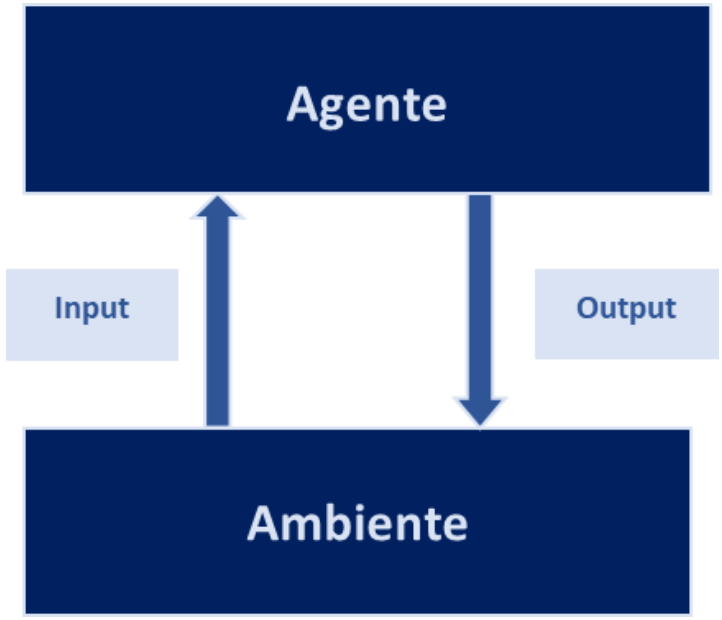
- Introduzione alla piattaforma
- Applicazione Wumpus World Game

DESCRIZIONE E MODELLAZIONE DEI SISTEMI MULTI-AGENTE

PROGRAMMAZIONE DEI SISTEMI TEMPO-REALE E DISTRIBUITI

DEFINIZIONE DI AGENTE

Con il termine "**Agente**" si intende un sottosistema, hardware o software, in grado di percepire l'ambiente in cui si trova attraverso dei sensori e di agire di conseguenza, sulla base delle informazioni raccolte, tramite degli attuatori, di cui è, eventualmente, provvisto.



Un agente, in quanto entità deliberativa, rappresenta l'evoluzione del concetto di oggetto, così com'è definito nella programmazione, poiché è capace di ricevere e reagire agli stimoli dell'ambiente circostante.

Questo concetto è stato introdotto da **Carl Hewitt** nel 1977, tramite la presentazione di un oggetto, denominato ***Actor***, che fosse autonomo, interattivo e capace esecuzione concorrente.

CARATTERISTICHE DI UN AGENTE

Un agente, in quanto sistema computazionale capace di agire, in maniera autonoma, nell'ambiente che lo circonda, al fine di raggiungere i propri obiettivi, deve essere dotato delle seguenti caratteristiche principali:

- **Reattività**, per cui deve essere in grado di mantenere una costante interazione con l'ambiente circostante, per rispondere prontamente ai cambiamenti a cui verrà sottoposto;
- **Proattività,** agendo non soltanto in base agli eventi che si verificheranno nell'ambiente, ma anche per il perseguimento dei propri obiettivi, creandosi delle opportunità per realizzarli;
- **Abilità sociale**, per cui dovrà essere dotato di capacità comunicative e di cooperazione che gli consentiranno di interagire, adeguatamente, con gli altri agenti dell'ambiente;

PROPRIETÀ DI UN AGENTE

Affinché un agente sia modellato in maniera corretta, dovrà soddisfare ulteriori proprietà, quali:

mobilità	ovvero la capacità di muoversi in un ambiente diverso da quello originario;
veridicità	garantire di non poter comunicare false informazioni;
benevolenza	per cui gli obiettivi prefissati non devono andare in conflitto con quelli degli agenti che si trovano nel suo stesso ambiente;
razionalità	agire mosso sempre dallo scopo di realizzare i propri obiettivi;
persistenza	per cui la vita a disposizione dell'agente dovrà essere certamente superiore al tempo richiesto per lo svolgimento delle sue attività;
autonomia	cioè la capacità, di un agente, di decidere se eseguire o meno una determinata azione, di cui gli è giunta richiesta;

MODELLAZIONE DI UN AGENTE

Il modo in cui gli agenti ragionano ed interagiscono con l'ambiente viene rappresentato attraverso delle architetture di modellazione, quali:

architettura reattiva

in cui il modo di comportarsi degli agenti è basato sulla messa in atto di ragionamenti non molto complessi e basati su priorità e viene descritto tramite l'utilizzo degli automi a stati finiti;

architettura BDI

basata su ciò che l'agente conosce e sulle intenzioni che animano le sue azioni;

architettura deliberativa

in cui l'agente elabora il suo comportamento sulla base delle caratteristiche dell'ambiente esterno;

architettura ibrida

una mediazione delle architetture precedenti, in modo da disporre di funzionalità che consentano di stabilire quale sia il comportamento più idoneo, sulla base della situazione che si sta valutando.

PROCESSI COGNITIVI IN UN AGENTE RAZIONALE

A prescindere dall'architettura che si vuole adottare, nell'ideazione di un agente si deve tenere conto di alcuni concetti che si riferiscono a degli stati mentali caratterizzanti dei processi cognitivi, quegli eventi che sono necessari alla formazione ed all'acquisizione di un qualsiasi contenuto di conoscenza.

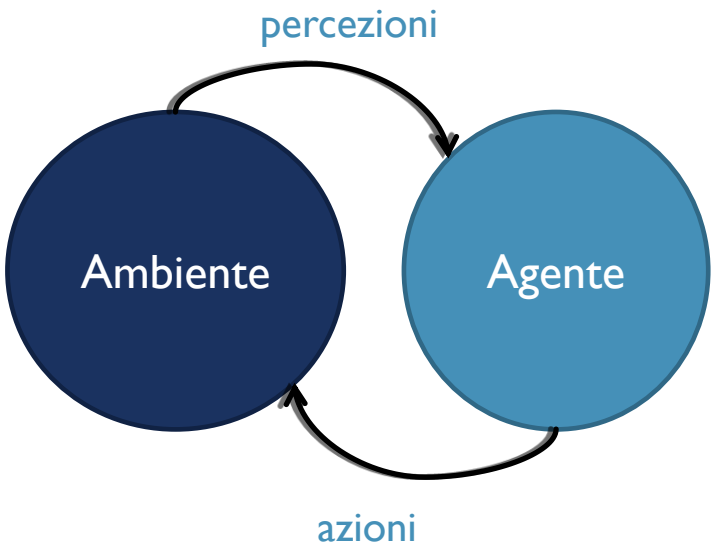
Questi concetti cognitivi sono identificati come:

- le credenze o **beliefs**, cioè le informazioni che l'agente assimila sull'ambiente circostanze tramite l'osservazione e l'esperienza;
- i desideri o **desires**, che costituiscono le situazioni che l'agente vorrebbe realizzare;
- le intenzioni o **intentions**, ovvero quegli obiettivi che l'agente si è prefissato e che vorrebbe raggiungere;
- la conoscenza o **knowledge**, che consiste nella capacità, dell'agente, di imparare ed apprendere, per migliorare le proprie prestazioni.

DEFINIZIONE DELL'AMBIENTE

L'ambiente o **environment** in cui le entità razionali si muovono ed interagiscono può essere definito:

- **accessibile**, se l'agente può acquisire delle informazioni complete, accurate ed aggiornate, riguardo lo stato dell'ambiente in cui si trova, altrimenti verrà definito **inaccessibile**;
- **deterministico**, se ogni azione effettuata genera uno specifico effetto, dal risultato non incerto; nel caso contrario, verrà definito **non-deterministico**, come il mondo reale;
- **statico**, se l'ambiente preso in considerazione rimarrà immutato, fatta eccezione per le azioni che, eventualmente, compirà l'agente; al contrario, verrà definito **dinamico**;
- **discreto**, se le percezioni e le azioni che caratterizzano l'ambiente si presentano in numero finito; se così non è, allora lo si definisce **continuo**.



FORMALIZZAZIONE DI UN AGENTE: AMBIENTE

L'ambiente può essere descritto come un insieme finito di stati istantanei e discreti, definito come:

$$E = \{ e_0 , e_1, \dots , e_n\}$$

in cui gli agenti potranno effettuare un numero finito di azioni, rappresentate dall'insieme seguente:

$$A_c = \{\propto_0, \propto_1, \dots , \propto_n\}$$

che andranno a modificare lo stato dell'ambiente, assunto come non deterministico e con memoria.

La sequenza di stati interfogliati dell'ambiente, risultata dalle azioni compiute dall'agente, è chiamata *run*.

Per specificare il modo in cui l'ambiente si modifica si utilizza la seguente ***trasformazione di stato***:

$$\tau: R^{A_c} \rightarrow \rho(E)$$

dove R è l'insieme di tutte le possibili sequenze finite di E e A_c , mentre R^{A_c} è il sottoinsieme delle sequenze che si sono concluse a seguito del compimento di un'azione.

FORMALIZZAZIONE DI UN AGENTE: PERCEZIONE

La capacità, di un agente, di osservare l'ambiente in cui si trova ed effettuare un'azione sulla base delle informazioni acquisite, è definita ***percezione*** e viene descritto tramite la funzione:

$$see : E \rightarrow Percep$$

che collegherà gli stati dell'ambiente alle percezioni dell'agente che si trova al suo interno.

Un'***azione*** può essere descritta dalla funzione:

$$action : I \rightarrow A$$

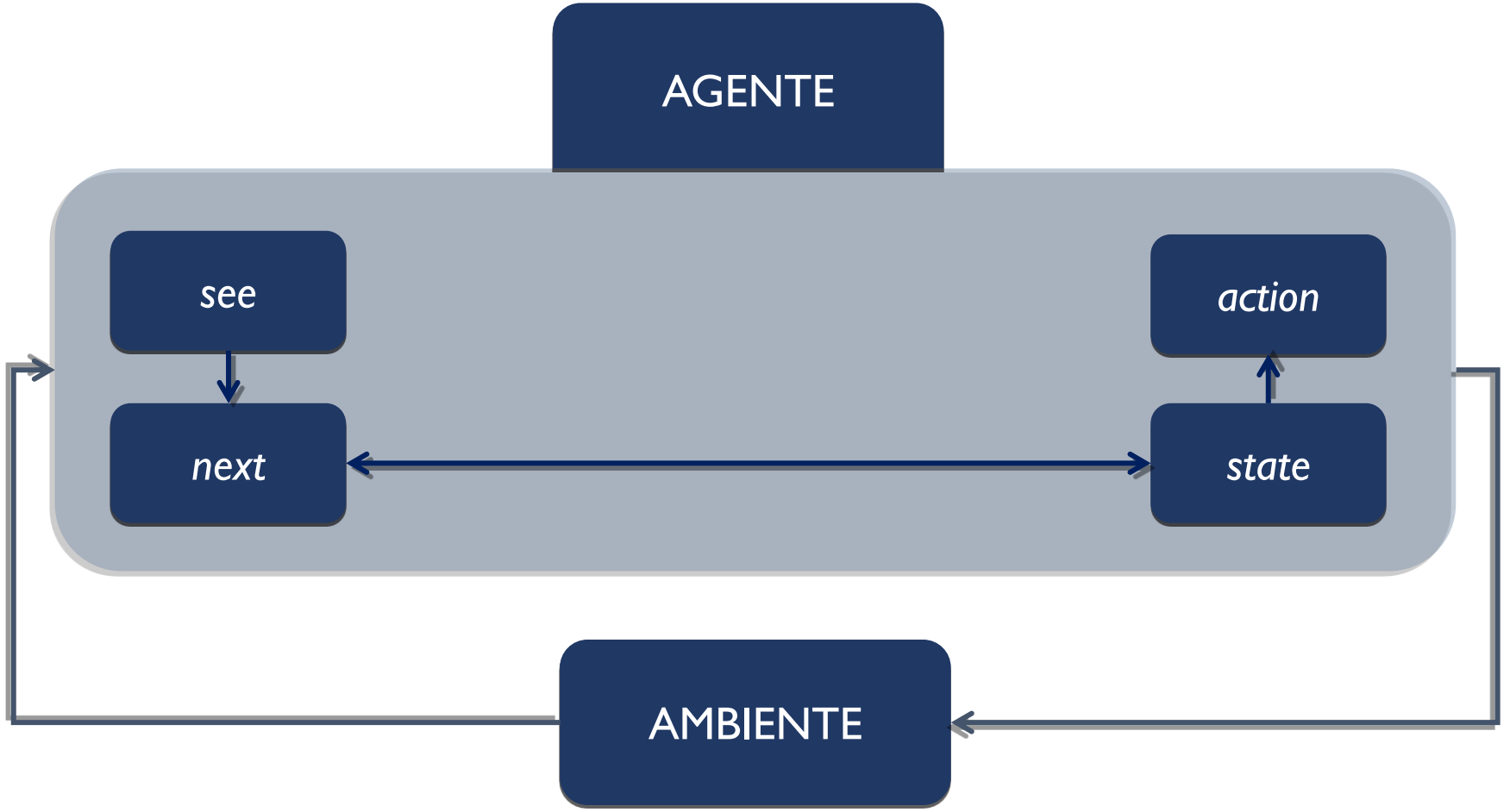
che associa gli stati interni I dello stesso agente a delle azioni.

Per l'aggiornamento dello stato interno dell'agente si utilizza la funzione seguente:

$$next : I \; x \; Percep \rightarrow I$$

che associa uno stato interno dell'agente ed una sua percezione, ad un ulteriore stato interno.

FORMALIZZAZIONE DI UN AGENTE



FORMALIZZAZIONE DI UN AGENTE: FUNZIONE DI UTILITÀ

Il mezzo tramite cui è possibile assegnare ad un agente un determinato task, senza esplicitare la metodologia con cui dovrà svolgerlo, consiste nella definizione di una ***funzione di utilità***:

$$u : E \rightarrow R$$

dove R è un numero reale ed u è una funzione che associa, ad ogni stato interno, un valore di utilità, che si dovrà cercare di massimizzare.

Tra i tipi di task che possono essere assegnati ad un agente si distinguono:

- i task di realizzazione o *achievement tasks*, che impartiscono, all'agente, il compito di raggiungere uno stato specifico, detto *goal state*;
- i task di mantenimento o *maintenance tasks* , che assegnano, all'agente, il compito di mantenere invariato un determinato stato, detto *bad state*.

SINTESI DI UN AGENTE

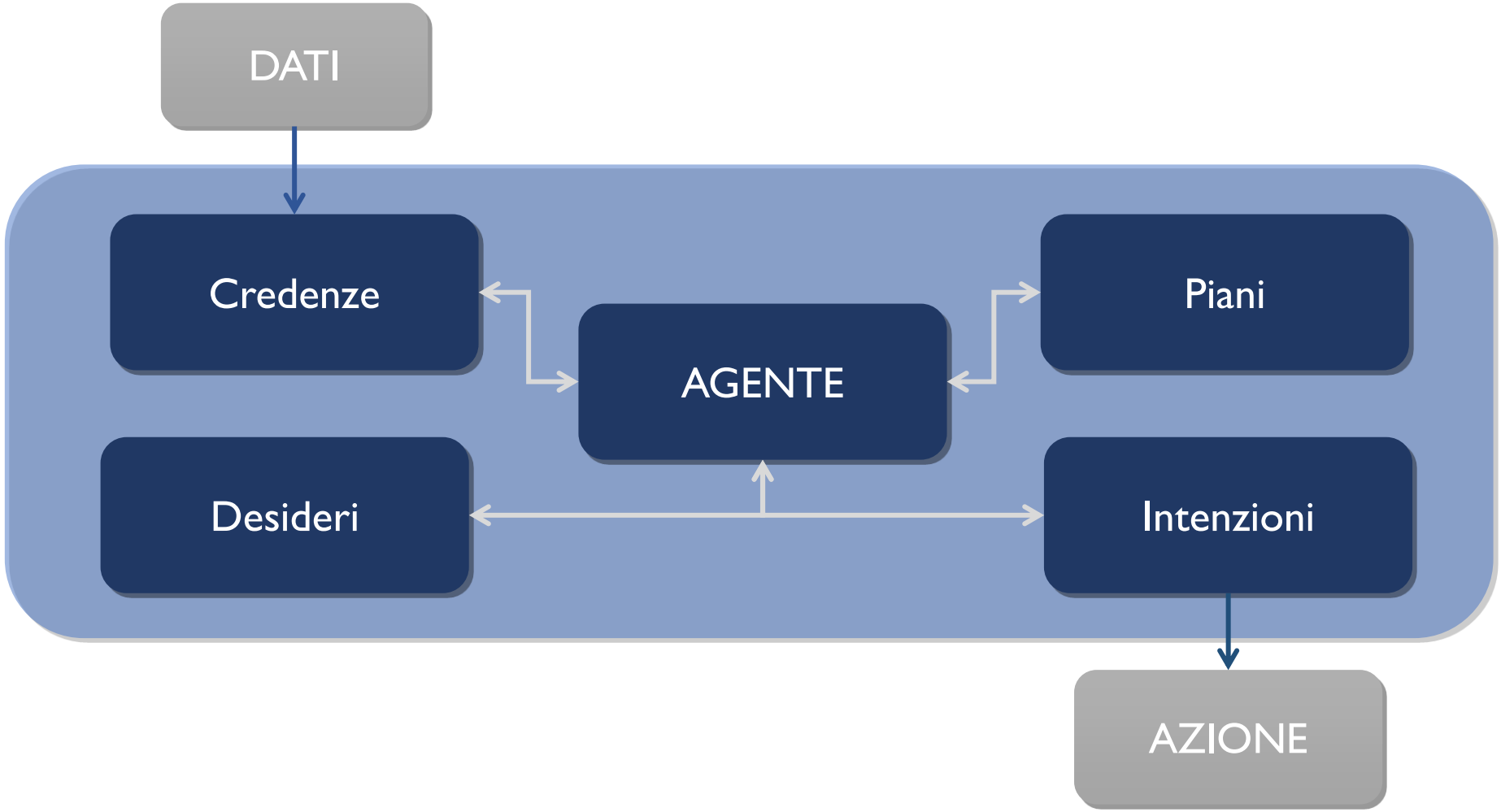
Il fulcro della sintesi di un agente consiste nell'implementazione di un meccanismo di programmazione automatica, che realizzi un processo di ragionamento volto decidere a quale obiettivo perseguire, tra quelli raggiungibili, stabilendo quali azioni siano necessarie per l'effettiva realizzazione.

Questo meccanismo, detto ragionamento pratico o ***practical reasoning***, è costituito da due processi:

- un processo decisionale, detto *deliberation*, che esamina un insieme di prospettive differenti, per poi stabilire quale stato raggiungere, elaborando, come risultato, un'intenzione;
- un processo denominato *means-ends reasoning*, per il raggiungimento della condizione desiderata, che è volto a delineare il modo in cui si può raggiungere lo stato desiderato.

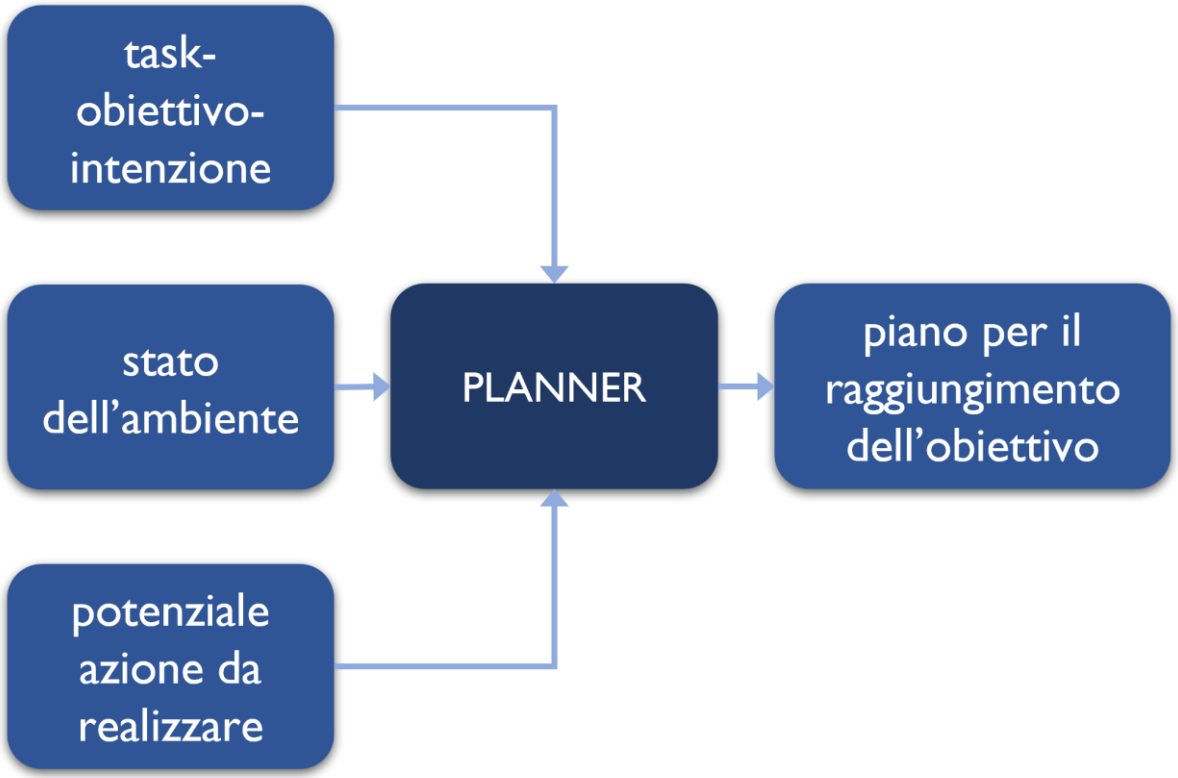
Questo metodo, quindi, si basa sulla valutazione dei vantaggi e degli svantaggi delle possibilità che si hanno a disposizione, tenendo sempre conto di quello in cui l'agente crede e di quali siano i suoi desideri ed è per questo che si contrappone al *theoretical reasoning* o ragionamento teorico, che è, invece, orientato alle credenze, piuttosto che alle azioni.

SINTESI DI UN AGENTE: PRACTICAL REASONING



SINTESI DI UN AGENTE: PLANNING

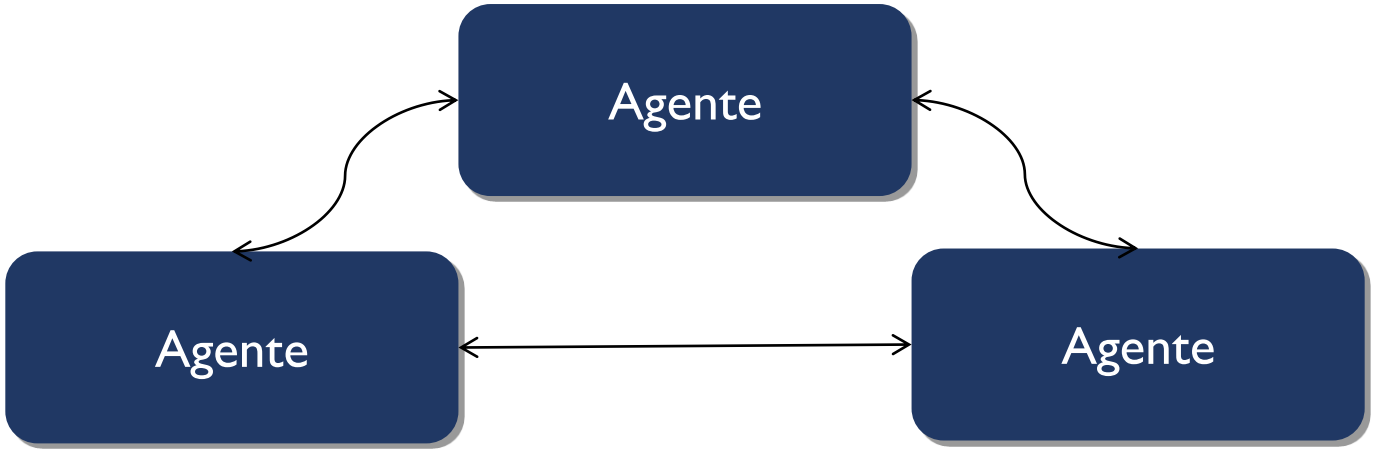
La progettazione di un piano d'azione volto a condurre l'agente al soddisfacimento del proprio obiettivo costituisce un problema di *planning* o pianificazione.



La programmazione automatica di un agente riguarderà, allora, la creazione di un oggetto di tipo *planner* che avrà il compito di fornire all'agente stesso una rappresentazione dell'obiettivo che dovrà raggiungere, un elenco delle azioni che dovrà compiere per la sua effettiva realizzazione ed una descrizione accurata dell'ambiente in cui si dovrà muovere.

DEFINIZIONE DI UN SISTEMA MULTI-AGENTE

Un sistema *multi-agente* è costituito da un determinato numero di agenti che si trovano nello stesso ambiente e che interagiscono tra loro, secondo una specifica organizzazione.



Ogni agente deve essere in grado di cooperare e coordinarsi con tutte le altre entità facenti parte del sistema, in modo tale che possano essere raggiunti tutti gli obiettivi prefissati.

Il meccanismo di comunicazione adottato dagli agenti di uno stesso sistema consiste nello scambio di messaggi, che forniscono informazioni sullo stato attuale di ciascuno e sulle proprie intenzioni.

ELEMENTI DI UN SISTEMA MULTI-AGENTE

Formalmente, un agente costituisce una funzione che associa un'esecuzione ad una determinata azione, come segue:

$$A_g: R^E \rightarrow A_c$$

dove per A_c si intende l'insieme che racchiude le azioni che l'agente deve eseguire, con R^E l'insieme degli stati dell'ambiente, ottenuto a seguito delle azioni che sono state effettuate e con A_g l'insieme di tutti gli agenti che compongono il sistema.

Il modo in cui l'agente seleziona, tra le potenziali azioni, quella da eseguire, è condizionato dalla storia del sistema di cui fa parte, sul quale ha avuto modo di acquisire della conoscenza.

Quindi, un sistema si può definire come una coppia di tipo agente-ambiente, a cui verrà associata una serie di possibili esecuzioni.

CARATTERISTICHE DI UN SISTEMA MULTI-AGENTE

Un sistema multi-agente dovrà:

- fornire una rappresentazione dell'ambiente in cui gli agenti agiscono, stabilendo il modo in cui questi possono, effettivamente, collaborare tra loro, tramite un'accurata pianificazione delle attività che ogni agente deve portare a compimento;
- dotare l'agente di una rappresentazione del mondo esterno che sia conforme al suo modello cognitivo e quindi provvista di tutte le informazioni di cui dovrà tenere conto nel suo processo decisionale;
- elaborare dei metodi di interazione che consentano agli agenti di collaborare tra loro, effettuando un coordinamento delle proprie azioni e risolvendo eventuali conflitti di interesse;
- definire il modo in cui dall'apprendimento del singolo agente si possa passare ad una forma di apprendimento che porti ad una crescita della conoscenza collettiva.

STRUTTURA DI UN SISTEMA MULTI-AGENTE

Nell'ambiente, ogni agente facente parte del sistema potrà scegliere quale azione compiere, tuttavia, siccome vi coesistono più entità deliberative, il modo in cui questo verrà modificato dipenderà dalla combinazione delle azioni svolte da ciascuna di queste e potrà essere descritto tramite la seguente funzione di stato:

$$\tau : A_c \, x \, A_c \rightarrow \Omega$$

dove il primo termine A_c si riferisce all'azione dell'agente i-esimo, il secondo termine A_c all'azione dell'agente j-esimo, mentre Ω rappresenta l'insieme dei risultati delle azioni compiute dagli agenti appartenenti al sistema.

Ciascun agente del sistema, per interagire con gli altri, potrà scegliere tra due soli tipi di azioni, quali:

- **cooperate**, cioè un'azione che consiste nella collaborazione con un altro agente;
- **defect**, ovvero un'azione che vada ad ostacolare l'altro agente, un atteggiamento di non collaborazione.

ELEMENTI DI TEORIA DEI GIOCHI

PROGRAMMAZIONE DEI SISTEMI TEMPO-REALE E DISTRIBUITI

INTRODUZIONE ALLA TEORIA DEI GIOCHI

La ***teoria dei giochi*** è una disciplina che si occupa dei modelli matematici che formalizzano i comportamenti adottati dai decisori, cioè i giocatori, in differenti situazioni, in cui si troveranno a perseguire obiettivi comuni, diversi o conflittuali.

Generalmente, la categoria di giochi che viene presa maggiormente in considerazione è quella rappresentata dai *giochi a somma zero*, in cui il guadagno (o la perdita) di un giocatore è totalmente bilanciato dalla perdita (o dal guadagno) di un altro partecipante, in somma uguale ed opposta.

MATRICE DI PAYOFF

Per analizzare le strategie dei giocatori e il modo in cui effettivamente si concretizza l'interazione strategica si utilizza la matrice di **Payoff**, proprio per descrivere i guadagni associati ad ogni azione che può essere compiuta.
La matrice avrà una struttura del tipo:

		i	
		defect	coop
j	defect	1 1	4 1
	coop	1 4	4 4

in cui ogni riga rappresenta un'azione giocatore *j* ed ogni colonna un'azione del giocatore *i*.

STRATEGIA DOMINANTE

Una strategia di gioco si definisce ***dominante*** se garantisce, al giocatore che la adotta, di ottenere il payoff più alto, cioè il risultato migliore tra tutte le possibili alternative, indipendentemente dalle mosse effettuate dagli altri giocatori.

Un agente razionale, quindi, cercherà sempre di adottare una strategia di questo tipo, che non sia, dunque, dominata da nessun'altra.

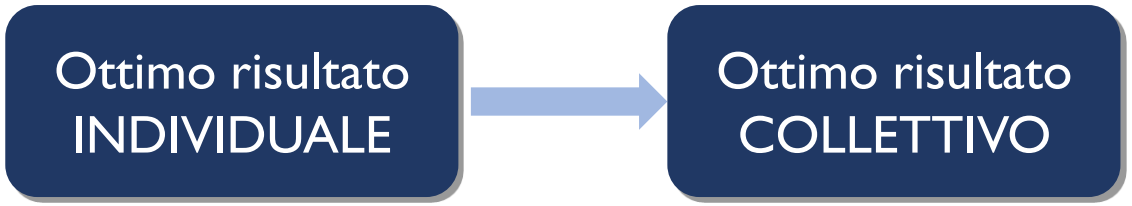
Se ciascun giocatore dispone di una strategia dominante, allora il gioco avrà una ***soluzione*** dominante, ottenendo, di conseguenza, una condizione di equilibrio dominante.

Per giocatore si intende un ***agente razionale***, cioè un individuo che sia in grado di ordinare le proprie preferenze in un insieme di risultati e che sia intelligente, ovvero dotato di capacità logica, in modo tale da poter riconoscere le azioni necessarie per massimizzare la propria utilità.

EQUILIBRIO DI NASH

L'equilibrio, nella teoria dei giochi non cooperativi, è un tipo di soluzione che descrive una situazione in cui nessun giocatore ha interesse a migliorare, in maniera unilaterale, il proprio comportamento.

Questo scenario non si fonda un accordo tra i giocatori, ma è dovuto al fatto che ognuno di essi ha deciso di adottare una strategia di tipo dominante, garantendo, in questo modo, il migliore risultato possibile a sé stesso ed a tutti gli altri partecipanti del gioco.



L'***equilibrio di Nash*** costituisce, quindi, una situazione in cui nessun agente razionale manifesterà alcun interesse nel cambiare la strategia adottata, generando una condizione di equilibrio stabile ed ottimale, poiché ciascun giocatore ricaverà, dalle proprie scelte, la massima utilità possibile, pur tenendo conto che un altro giocatore possa effettuare una scelta migliore della propria.

IL DILEMMA DEL PRIGIONIERO

Il dilemma del prigioniero è un gioco ad informazione completa, proposto da *Tucker* proprio come problema della teoria dei giochi, negli anni '50, descritto come segue:

Due criminali vengono accusati di aver commesso un reato e perciò vengono arrestati e rinchiusi in due celle diverse, in maniera tale che non possano comunicare tra loro.

Ad entrambi vengono presentate due scelte, cioè di collaborare o meno con le forze dell'ordine, perciò si delineano tre possibili scenari, così caratterizzati:

- se solamente uno dei due prigionieri scegliesse di collaborare, accusando l'altro, allora quest'ultimo verrebbe condannato a quattro anni di carcere, mentre il primo ne sconterà uno soltanto;
- se entrambi scegliessero di collaborare, accusandosi a vicenda di aver commesso il crimine, allora verrebbero condannati a due anni di carcere ciascuno;
- se nessuno dei due prigionieri scegliesse di collaborare questi verrebbero condannati a tre anni di carcere, con l'accusa di porto d'armi abusivo.

IL DILEMMA DEL PRIGIONIERO: UN APPARENTE PARADOSSO

Siccome lo scopo di ogni prigioniero è ridurre la propria condanna, la strategia migliore sarà di tipo "non cooperativo". Infatti, non essendo a conoscenza della strategia adottata dall'altro, nessuno dei due prigionieri proteggerà il complice, per cercare di ottenere una pena ridotta, tramite una confessione.

prigioniero p_2 \ prigioniero p_1	Defect	Cooperate
Defect	2 2	1 4
Cooperate	4 1	3 3

I giocatori, infatti, potranno compiere due sole azioni:

- *defect*, ovvero l'azione di "collaborare" con le forze dell'ordine;
- *cooperate*, che rappresenta la volontà di "non collaborare";

Per far sì che i due prigionieri cooperino tra loro bisogna reiterare il gioco più volte, incentivandoli alla collaborazione, nell'eventualità che poi possano rincontrarsi. È difficile, quindi, realizzare un meccanismo di tipo cooperativo in una società costituita da agenti *self-interested*, impegnati nella ricerca del proprio utile.

IL TORNEO DI AXELROD

Nel 1984 Axelrod, a partire dal dilemma del prigioniero iterato, propose uno studio che evidenziò come, nei problemi della vita sociale, gli individui manifestassero la tendenza a cooperare tra loro, anziché scegliere la defezione, in contrapposizione alla teoria dei giochi classica.
Dallo studio, validato tramite delle simulazioni in cui dei programmi si affrontarono in una serie di tornei, basata sul gioco iterativo del prigioniero, risultò vincente, tra le 15 strategie proposte, di cui una randomizzata, quella denominata "Tit for tat". Si riportano le più significative:

- ***Always Defect*** (o strategia del falco), in cui si attacca, senza manifestare le proprie intenzioni, avventandosi contro l'avversario;
- ***Tit for tat***, in cui al primo round si effettua un'azione di cooperate e successivamente si adotta lo stesso comportamento dell'avversario;
- ***Tester***, secondo cui, al primo round si effettua un'azione di defect e poi, se l'avversario si vendica si adotta l'approccio tit-for-tat, mentre, se non si vendica, si alternano azioni di cooperate e defect;
- ***Joss***, simile a tit-for-tat, con la differenza che, periodicamente, si effettua un'azione di defect.

IL GIOCO DEL POLLO

Il gioco del pollo, o del pavido, è una situazione in cui due giocatori devono indurre la controparte ad adottare un determinato comportamento, senza eseguire la stessa azione.
Un classico esempio di gioco a somma non nulla ed informazione completa è ispirato alla sfida automobilistica (*chicken run*) descritta nel film «Gioventù bruciata», del 1955.

La prova di coraggio che i due ragazzi si trovano ad affrontare consiste nel dirigersi, con le proprie automobili, verso un dirupo e sterzare il più tardi possibile.

Si presentano i seguenti scenari:

- entrambi sterzeranno prima di giungere in prossimità del dirupo e falliranno la prova;
- uno dei due sterzerà prima dell'altro, facendo la figura del codardo, mentre l'altro guadagnerà il rispetto degli spettatori perché avrà percorso un tratto maggiore verso il dirupo;
- entrambi non si fermeranno in prossimità del dirupo e moriranno;

IL GIOCO DEL POLLO: MATRICE DI PAYOFF

La cooperazione tra i due avversari, come nel dilemma del prigioniero, conduce ad una condizione di equilibrio instabile, che non si mantiene neanche per un breve periodo, nel caso in cui il gioco venga reiterato una volta soltanto.

		i	
		Cooperate	**Defect**
j	**Cooperate**	**0** **0**	**-1** **1**
	Defect	**1** **-1**	**-10** **-10**

In questo caso l'azione di *defect* corrisponde al "guidare dritto verso il dirupo", mentre *cooperate* rappresenta l'azione di "sterzare". Si noti come risultato più temuto sia l'azione di defezione reciproca, perciò ad entrambi i giocatori conviene adottare la strategia opposta a quella dell'avversario, anziché una di tipo dominante.

Allora, la soluzione di questo gioco di non-coordinamento, dipende dalla credibilità, di uno dei due giocatori, nel dichiarare la propria intenzione di non sterzare, per poter vincere la sfida. In questo modo, l'altro giocatore si vedrà costretto a sterzare per primo, per evitare di cadere nel dirupo.

REALIZZAZIONE DI UN'APPLICAZIONE ANDROID

PROGRAMMAZIONE DEI DISPOSITIVI MOBILI

ANDROID OS

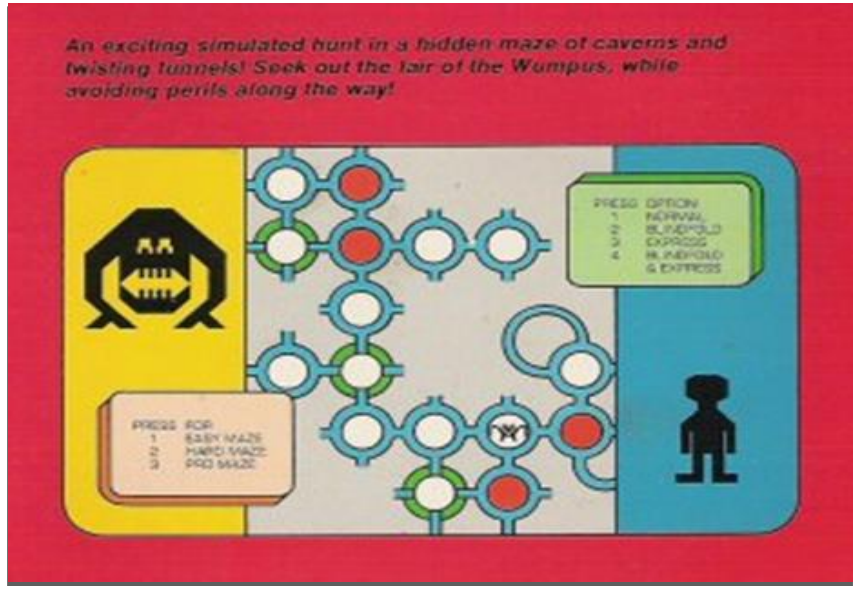
Android è un sistema operativo open-source, progettato da Google e basato su un kernel **Linux** (versione 2.6 e 3.0), in cui sono inseriti i driver per il controllo dell'hardware dei dispositivi mobili su cui andrà a girare.

L'architettura, la cui base è costituita dal kernel e da una libreria fondamentale, prevede la presenza di una macchina virtuale, la **Dalvik Virtual Machine** (simile alla JVM) che esegue del codice scritto in DEX, ovvero Dalvik Executable.

Per sviluppare un'applicazione Android si deve installare il kit di sviluppo o **SDK**, completo di emulatore AVD, librerie e documentazione ed utilizzare un ambiente di sviluppo, come Android Studio.

INTRODUZIONE AL GIOCO

Il progetto di questa applicazione Android è ispirato al videogioco "**Hunt the Wumpus**", realizzato da Gregory **Yob**, nel 1972.



È un gioco di avventura che si svolge in un labirinto, generato in maniera casuale e costituito da una serie di stanze, 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.
L'obiettivo del cacciatore, è infatti, quello di cercare di uccidere il mostro con l'unica freccia di cui dispone.

REGOLE DEL GIOCO

Rispetto al gioco originale, la versione che è stata realizzata, oltre all'assenza dei super-pipistrelli, prevede delle differenze per quanto riguarda la modalità di gioco.

Il giocatore, infatti, può decidere che ruolo interpretare nella storia, scegliendo 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.

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 da 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 del progetto trattato in questo elaborato l'ambientazione è rappresentata da una foresta fitta dai percorsi intricati.

SPECIFICHE PROGETTUALI

La versione del gioco di Yob realizzata come progetto è stata ideata ed implementata per interezza in Java, utilizzando la versione 8 della JDK.

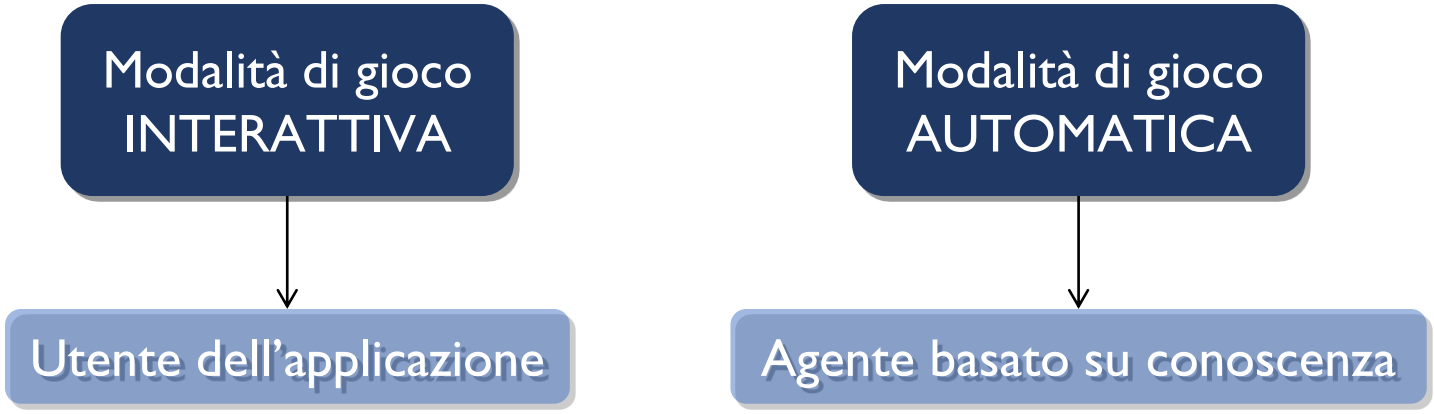
Dopo averla testata e resa giocabile dall'utente ricevendo degli input testuali da console, è stata creata una libreria che ne racchiudesse le funzionalità, affinché andasse a costituire il back-end di quella che sarebbe stata l'implementazione dell'applicazione Android vera e propria, provvista di grafica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

Il kit di sviluppo utilizzato per il front-end è la versione12 di Android.

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

MODALITÀ DI GIOCO

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.



A prescindere dalla scelta del ruolo e della modalità di gioco, 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.

FORMULAZIONE PEAS

Il termine PEAS si riferisce all'acronimo "**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.

Quindi la tabella riporta un'esamina dell'applicazione in termini di misurazione delle performance e dell'analisi di ambiente, attuatori e sensori.

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;

CARATTERISTICHE DELL'AMBIENTE

L'ambiente che costituisce il mondo del Wumpus si può definire:

- *completamente osservabile (accessibile)*, perché l'agente acquisisce delle informazioni dall'ambiente circostante, che, benché siano relative soltanto alle celle adiacenti a quella in cui si trova, sono sufficienti per la determinazione dell'azione che deve compiere;
- *deterministico*, poiché l'esito di ogni azione che verrà eseguita avrà un effetto noto a priori;
- *statico*, perché rimane immutato, per tutta la sessione di gioco, se si escludono le azioni che verranno compiute dall'agente;
- *discreto*, infatti le percezioni che l'agente può acquisire e le azioni che può eseguire sono limitate, ovvero si presentano in numero finito;
- *sequenziale*, in quanto ogni decisione presa dall'agente andrà ad influenzare quelle successive;
- a *singolo agente*, che è costituito proprio dall'algoritmo di risoluzione della mappa di gioco utilizzato per implementare il giocatore automatico;

ELEMENTI DI GIOCO

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

Su questa matrice, di dimensione prefissata e non modificabile, verranno posizionati, in maniera casuale, tramite una funzione di probabilità, tutti gli elementi di gioco, ovvero:

- il personaggio giocabile scelto dall'utente;
- il nemico del personaggio giocabile;
- gli alberi che 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, in base alla modalità di gioco;

Inoltre, alcune celle potranno essere vuote, cioè non conterranno nessuno degli elementi sopraelencati e pertanto potranno essere considerate come sicure.

ORGANIZZAZIONE DEL PROGETTO

I differenti progetti di back-end e front-end nell'applicazione Android confluiscono in un unico macro-progetto, denominato "**Hunt The Wumpus App**".

In questo modo si è ottenuta una maggiore flessibilità durante il processo di sviluppo, perché è stato possibile lavorare su parti differenti dell'applicazione, che sono state integrate dopo la fase di testing.

Inoltre, si è realizzato un back-end stabile e funzionante, in quanto indipendente dalla GUI, utilizzabile all'interno del progetto Android, come fonte esterna delle funzionalità fornite all'utente.

La separazione tra back-end e front-end ha impedito, quindi, di compromettere il funzionamento dell'intero progetto, nel caso di modifiche errate, sull'una o l'altra parte ed ha semplificato, l'aggiornamento del codice, perché l'integrazione dei cambiamenti, di uno dei due moduli, nel progetto principale, viene eseguita automaticamente, mantenendo, però, le commit separate.

Il progetto finale è caricato nel repository https://github.com/ivochan/HuntTheWumpus_App" di *Github* e contiene i link ai due progetti che utilizza.

STRUTTURA DEL BACK-END

Il back-end costituisce un progetto a sé stante, implementato in Java 8, denominato "Hunt the Wumpus or Not", che realizzata una versione del gioco "Hunt the Wumpus", anch'essa testuale, le cui funzionalità verranno utilizzate e richiamate nell'interfaccia utente dell'applicazione Android.

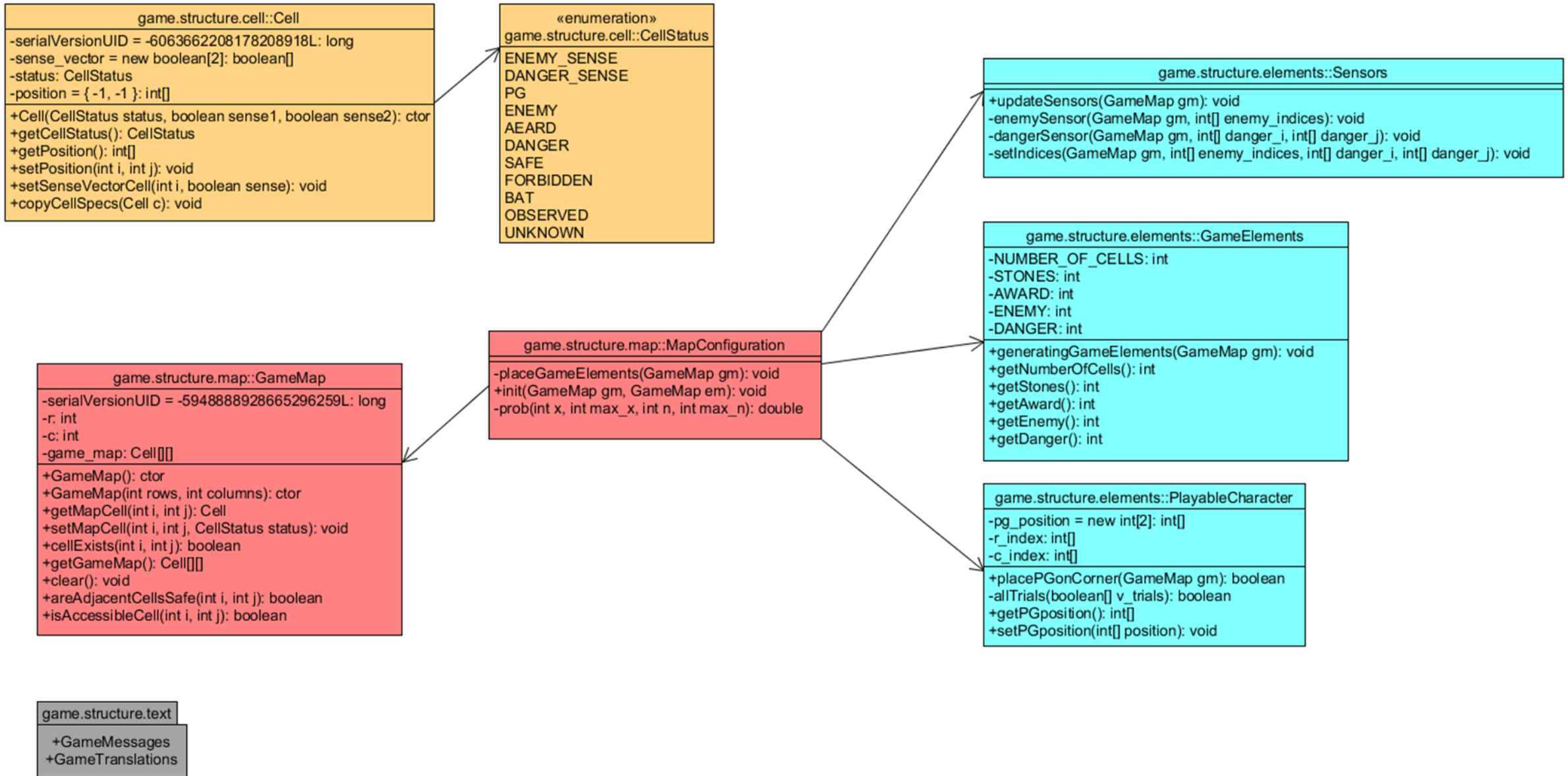
La modalità di gioco prevede la possibilità di scegliere tra:

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

Quindi, l'utente potrà decidere se vuole essere lui a controllare le mosse del suo personaggio oppure lasciare che la risoluzione e quindi l'esplorazione del labirinto venga eseguita dal giocatore automatico.

Inoltre, all'avvio della partita, potrà scegliere se giocare alla versione classica, *Hero Side*, in cui l'avventuriero deve uccidere il Wumpus oppure alla versione *Wumpus Side*, in cui dovrà impersonare proprio il mostro del gioco e riuscire a sopravvivere alla battuta di caccia.

DIAGRAMMA UML DELLE CLASSI DEL PACKAGE game.structure



TERRENO DI GIOCO

La classe più rilevante è **MapConfiguration**, poiché, utilizzando tutte le altre classi del package **game.structure**, svolge tutte le configurazioni necessarie alla preparazione del campo di gioco.

Infatti, tramite i relativi metodi verranno inizializzati tutti gli elementi di gioco, sulla base della seguente funzione di probabilità, che ne stabilisce il posizionamento in maniera totalmente casuale:

La struttura della mappa che si ottiene 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

RIEMPIMENTO DELLA MAPPA

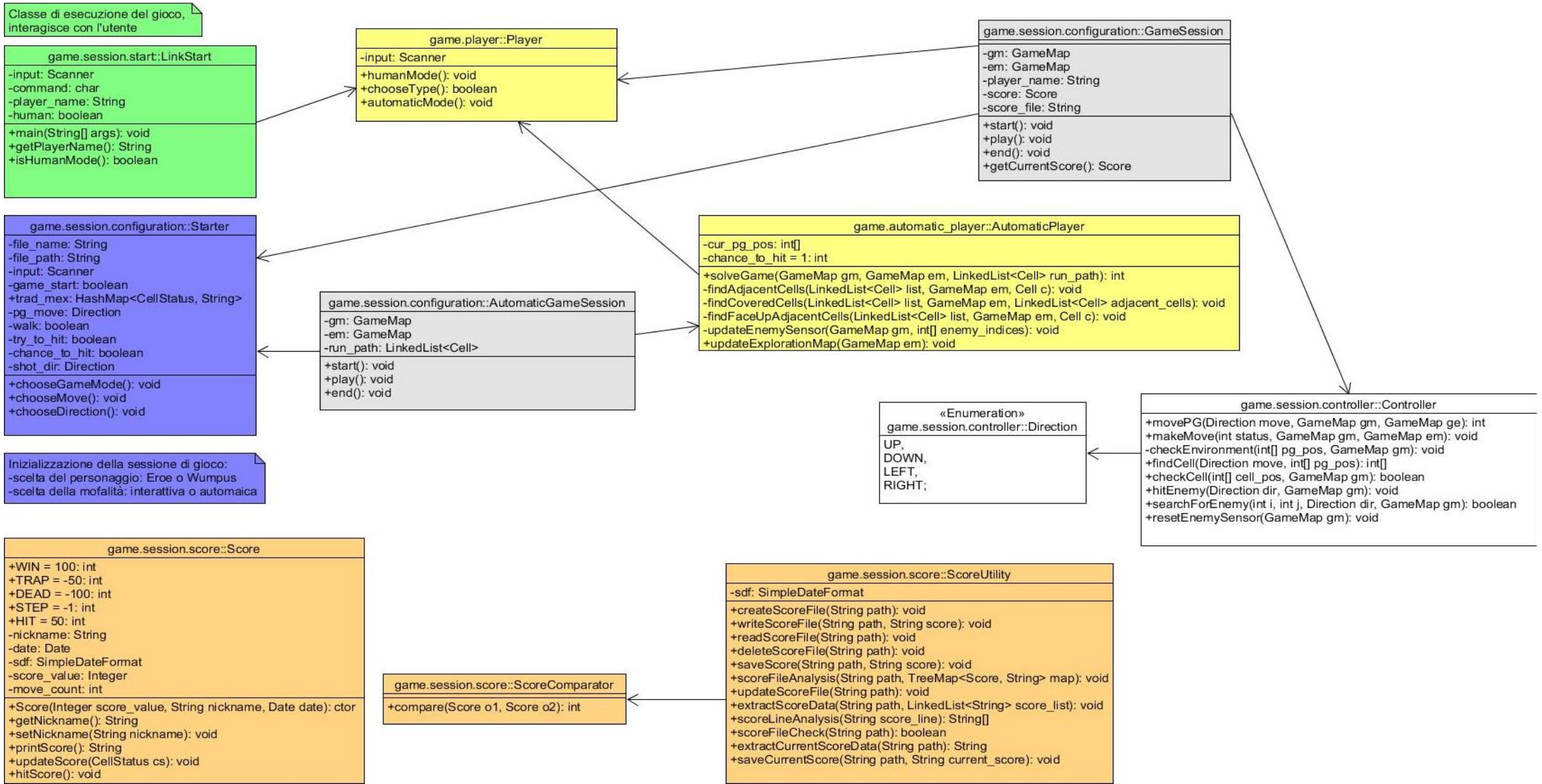
La determinazione del contenuto di ogni cella della matrice avviene calcolando la probabilità del valore che si sta considerando come possibile tipologia, verificando se questo sia maggiore della soglia ed etichettando, nel caso di esito positivo, la casella in esame. Segue la funzione di probabilità:

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

Dove:

- x è il numero di oggetti 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;
- $random$ è un numero casuale inserito per dare un minimo di varianza alla funzione;

DIAGRAMMA UML DELLE CLASSI DEL PACKAGE game.session



CONTROLLORE 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**, collocata nel package **game.session**.

Questa si occupa di:

- verificare se la direzione in cui si vuole fare muovere il PG è valida;
- 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;
- visualizzare il risultato ottenuto dopo l'esecuzione della mossa richiesta ed aggiornare il punteggio del giocatore;

GIOCATORE AUTOMATICO

Il giocatore automatico, rappresentato dalla classe ***AutomaticPlayer***, è stato progettato per risolvere la partita in corso.

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

Questo giocatore adotterà una strategia che consta di due differenti approcci, in modo che il suo comportamento muti e si adatti al variare della conoscenza che ha dell'ambiente.

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

L'*algoritmo*, di tipo *backtracking*, che ne implementa il comportamento, allora, 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.

DESCRIZIONE DELLA STRATEGIA

Il giocatore automatico sceglie di muoversi sempre nella direzione più sicura, ma nel momento in cui questo non sarà più possibile, perché si troverà sempre circondato da celle non sicure, per evitare una situazione di stallo, compirà comunque una mossa.

Se è acceso il sensore che indica la presenza del nemico, allora l'agente tenterà di colpirlo, scegliendo in maniera casuale, una delle caselle che non ha ancora esplorato.

Se il colpo è andato a segno, l'agente si è creato un nuovo percorso sicuro, perciò adotterà nuovamente la ***strategia iniziale***, scegliendo casualmente una cella, tra quelle non ancora visitate, che sia sicura e nel caso in cui non esista, una cella a caso, tra quelle già visitate.

Se non esiste una cella adiacente a quella in cui si trova il PG oppure tutte le celle adiacenti sono potenzialmente pericolose, il giocatore automatico adotterà una **strategia di emergenza**, decidendo casualmente la direzione verso cui muoversi e tenendo in conto, così, il rischio di incorrere in una situazione di pericolo e perdere la partita.

WUMPUS WORLD GAME

Questo progetto, realizzato per dispositivi mobili su cui gira il sistema operativo Android, ha come fulcro la libreria **huntthewumpusornot** scritta in Java, che è stata ottenuta esportando il codice del back-end dell'applicazione mobile, in un file .jar.

Questa libreria è stata integrata, importandola come modulo dell'applicazione **Wumpus's World Game,** con un'interfaccia utente realizzata sfruttando le funzionalità messe a disposizione dal kit di sviluppo Android.

Volendo esportare e distribuire il file apk, cioè il file eseguibile dell'applicazione, si è provveduto a firmarlo digitalmente e poi ad installarlo e testarlo su diversi dispositivi mobili, sia fisici che virtuali.

Ad esempio, utilizzando l'emulatore Android Virtual Device, è stato simulato il comportamento dell'app su un dispositivo con sistema operativo Android Oreo 8.1, che dunque utilizza le Android API 27, cioè il Pixel 3a XL. Per quanto il testing su dispositivo reale, invece, è stato maggiormente utilizzato un Huawei Mate 20 Lite, con sistema operativo Android 10 e dallo schermo di sei pollici.

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: MENU PRINCIPALE

1a. L'utente può accedere, tramite il menu principale, a differenti schermate:

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 la storia ed i 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 della 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:
    a. Dalla sezione "Record", l'utente può condividere il suo miglior punteggio;
    b. Dalla sezione "Ultimo punteggio", può condividere il suo ultimo punteggio;
    c. L'utente ritorna alla schermata precedente tramite il pulsante di navigazione "Indietro";

ESTENSIONI DEI CASI D'USO: IMPOSTAZIONI

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 il file contenente dei punteggi precedenti,
        ii. Esportare i dati di gioco scegliendo dove memorizzarli;
        iii. Eliminare i dati di gioco cancellando il file dei salvataggi;
    c. Nella scheda "About" l'utente può inviare un feedback allo sviluppatore tramite email o visualizzare le informazioni ed i crediti dell'applicazioni;
    d. L'utente ritorna alla schermata precedente tramite il pulsante di navigazione "Indietro".

ESTENSIONI DEI CASI D'USO: MENU DI GIOCO

2a. L'utente, durante la sessione di gioco, può accedere al menu 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;

Per creare un componente Menu, oltre a definirne lo stile, bisogna renderlo visibile tramite il metodo *onCreateOptions*(*Menu*) e gestire le azioni da svolgere alla selezione di ognuna delle voci con *onOptionsItemSelected*(*MenuItem*).

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, lasciando che sia l'agente software a concludere la sessione di gioco.

Si è cercato di rispettare le convenzioni adottate nello sviluppo di applicazioni Android, suddividendo nei rispettivi package i componenti dello stesso tipo.

Per quanto riguarda le relazioni tra questi e le classi scritte in Java si avranno le seguenti dipendenze:

- le classi di tipo Activity estendono *AppCompatActivity*;
- le classi che definiscono un componente Adapter estendono la classe *BaseAdapter*;
- la classe GameSettingsFragment, richiamata dall'activity GameSettings, estende la classe astratta *PreferenceFragmentCompat*;
- la classe TypeWriter, scritta per realizzare un testo animato, estende la classe *AppCompatTextView*;

CICLO DI VITA DI UN'ACTIVITY

Una generica **Activity**, ovvero il componente che definisce una schermata visualizzata dall'utente, il quale interagisce con l'applicazione Android che sta utilizzando tramite il display del dispositivo, segue un determinato ciclo di vita, che inizia con la sua creazione e termina solamente al compimento del task che gli è stato assegnato oppure quando la sua esecuzione viene interrotta.

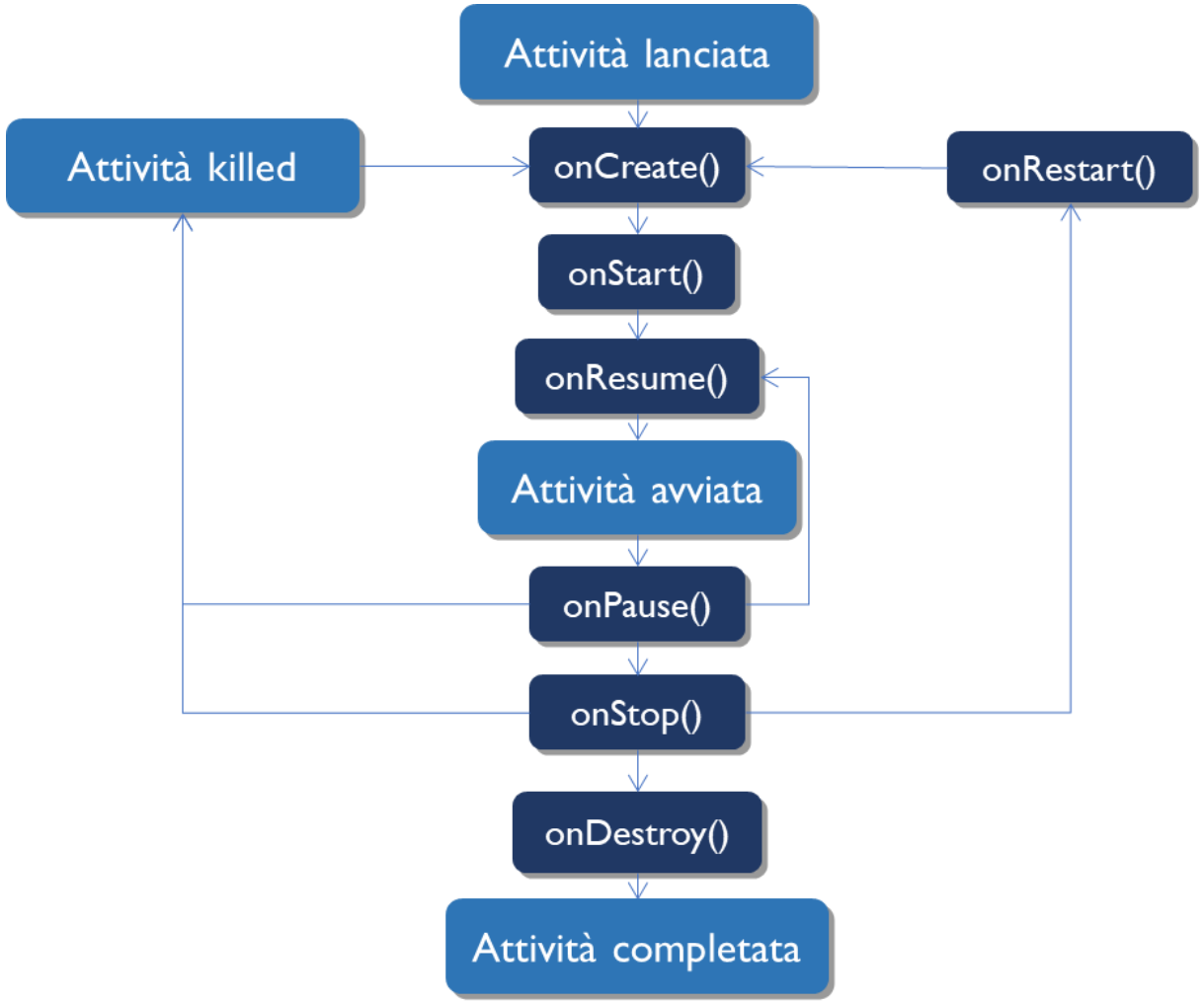
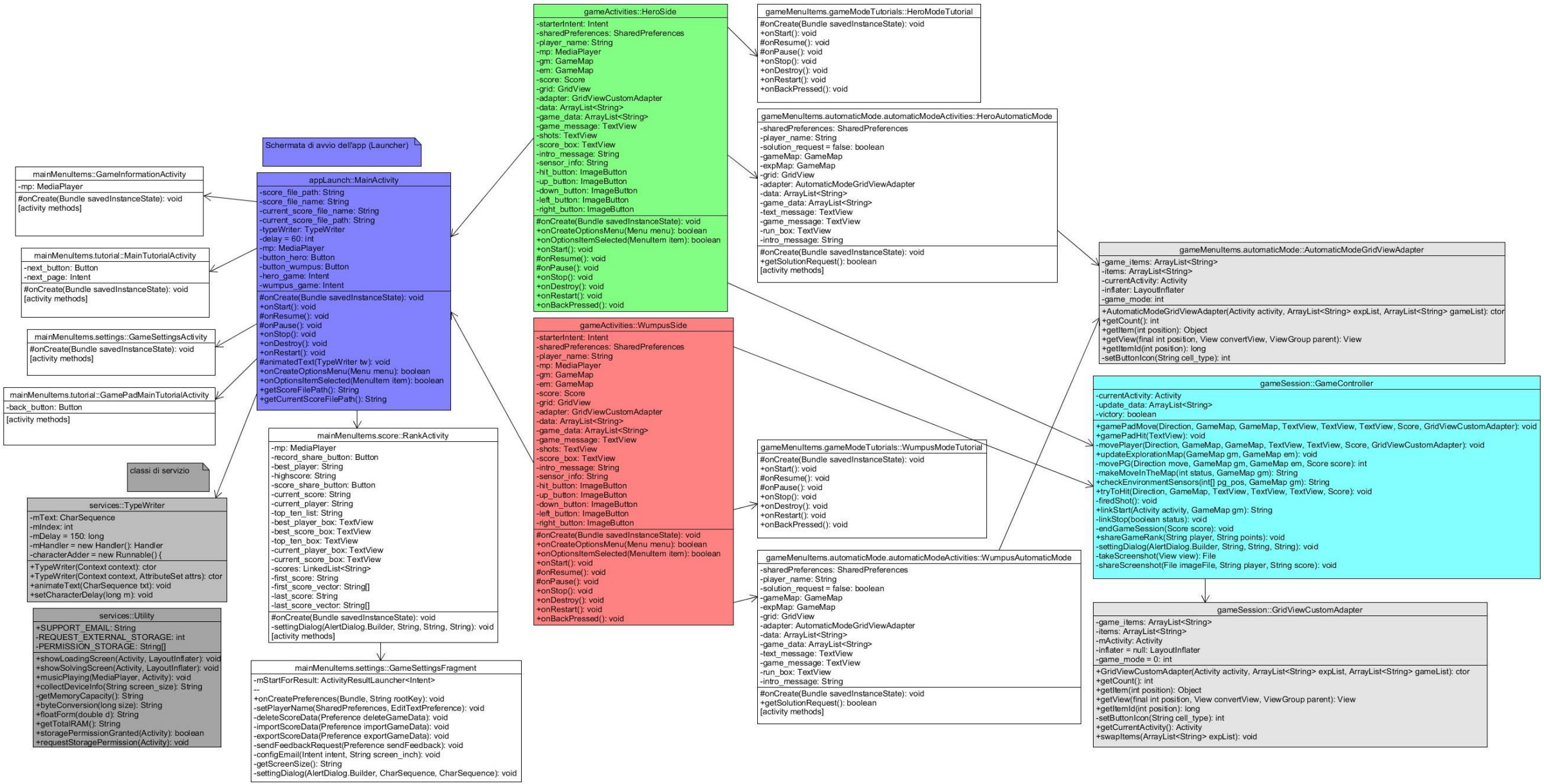


DIAGRAMMA UML DELLE CLASSI



ORGANIZZAZIONE DELLE RISORSE

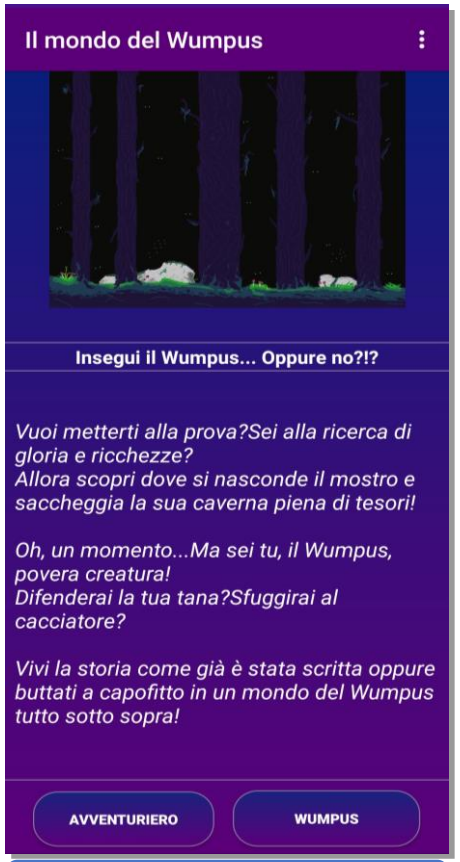
Le risorse sono raggruppate in cartelle specifiche, quali:

- *Src*, per quanto riguarda i package e le classi che implementano l'applicazione;
- *Res*, che racchiude le risorse esterne, come immagini e clip audio, utilizzate dall'applicazione;

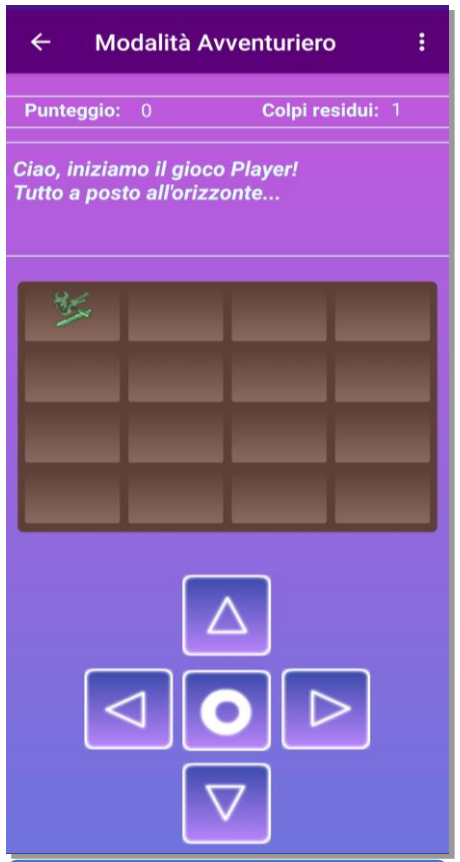
Quest'ultima è costituita da alcune sottodirectory, che sono:

- *Drawable*, che contiene le immagini ed 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 dei componenti dell'interfaccia utente, i file strings.xml di ogni lingua prevista e la cartella "themes", in cui sono definiti i temi light mode e dark mode;
- *Raw*, la cartella dove sono memorizzate le clip audio;
- *Menu*, in cui si trovano i file .xml che definiscono gli stili del menu principale e del menu di gioco;
- *Xml*, che contiene il file "PreferenceScreen" utilizzato per la schermata delle impostazioni;

INTERFACCIA UTENTE



App Launcher



Schermata di gioco

Risoluzione

Classifica

SVILUPPI FUTURI

Per migliorare l'applicazione, oltre ad eventuali ottimizzazioni del codice, si è pensato di inserire alcune funzionalità aggiuntive, come:

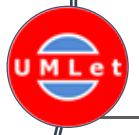
- la presenza dei super pipistrelli, così come previsto nella versione originale del gioco; questi, se incontrati lungo il percorso, trasportano il PG in un punto casuale della mappa;
- Differenti algoritmi di risoluzione, tra cui l'utente potrà scegliere la strategia che dovrà adottare il giocatore automatico;
- La definizione di un insieme di regole per la configurazione del terreno di gioco, come la dimensione della mappa e la difficoltà della partita;
- Inserimento di diversi livelli, diversi per abilità richiesta e scenario di gioco;

STRUMENTI UTILIZZATI

L'ambiente di sviluppo integrato multipiattaforma *Eclipse*, per l'implementazione del codice Java.

L'ambiente di sviluppo integrato *Android Studio* per applicazioni Android.

Il sistema di controllo di versione distribuito *Git*.



Il tool di modellazione *UMLet*, per creare i diagrammi UML rappresentativi delle classi.

Il linguaggio di programmazione Java, con il software di sviluppo *Java Development Kit* (Java 8).

Il kit di sviluppo *SDK* per la piattaforma Android (Android API 32).

Email: ivonne.rizzuto@gmail.com

Github: ivochan

CONTATTI

IVONNE RIZZUTO

Grazie per l’attenzione.