Classe Giocatore (Model)

La classe contiene una serie di attributi utili per memorizzare le informazioni del giocatore:

Nome: Stringa
Id: Stringa
Colore: Int
Un giocatore è caratterizzato da un colore, per memorizzarlo si usano indici da 0 a 4 (5 colori), memorizzato nella variabile.
Attivo: Bool
Variabile a 1 se il giocatore è attivo, zero se è in stato di attesa.
Turno: Int
Variabile che considera che turno di gioco è.
Danni: ArrayList<Integer>(4)
Lista di interi che rappresenta i danni subiti da ogni giocatore, per identificare chi è il giocatore che ha fatto il danno si usa l'indice della lista (esempio il colore blu corrisponde all'intero '0', vedi nella variabile *colore*, quindi in posizione [0] dell'array ci sono i danni che sono stati dati dal giocatore con il colore blu). Non ci sono limiti sui danni, se non che la somma non può superare il valore 12. Alcuni danni hanno delle caratteristiche speciali perché corrispondono a punti extra, questi saranno considerati nella valutazione del Control, e sono:

- Danno 1 (1 punto) – Primo sangue
- Danno 11 (6 punti) – Uccisione
- Danno 12 (1 punto) – Infierire (eventualmente solo a chi ha fatto il danno dell'uccisione)

Marchi: ArrayList<Integer>(4)
Lista di interi con analogo significato alla lista *danni*. Ha il vincolo che si possono avere al massimo tre marchi da un dato giocatore nemico, e massimo tre marchi da ogni altro giocatore (significa nel complesso 12 marchi massimo, tre per ogni giocatore avversario).
Munizioni: : ArrayList<Integer>(3)
Lista di interi che rappresenta per ogni indice le munizioni restanti di un dato colore. Anche in questo caso si rappresenta il colore con un intero che è indice della lista (blu, rosso, giallo)->(0,1,2). Valore massimo di ogni cella è 3 (massimo tre munizioni per colore), minimo zero ovviamente.
Gun: ArrayList<Arma>(3)
Lista che tiene conto delle armi possedute dal giocatore, possono essere massimo tre.
Pow: ArrayList<Potenziamento>(3)
Lista che tiene conto dei potenziamenti posseduti dal giocatore, possono essere massimo tre.
Cel: int[2]
Vettore a due dimensioni per le coordinate del giocatore.
Death: int
Variabile che indica il numero di volte che il giocatore è morto e serve per sapere quanti sono i massimi punti attribuibili in caso di nuova morte ai giocatori che hanno fatto danno.
Action: int
Primosangue: int
Variabile che tiene conto di chi è il giocatore che ha fatto il danno 1 (primo sangue).
Score: int

Per questi attributi sono necessari dei metodi interni alla classe per descrivere il modello (pensa come ad un'analisi statica). Oltre ai metodi utili e ai significati, dettaglio anche le possibili evoluzioni, e le modifiche che possono essere apportate ai parametri. Questi metodi andranno nel Controller, qui faccio un'analisi di insieme per pensare a tutto quello che può essere necessario.

Costruttore. Nel costruttore ho passato come parametri il nome scelto dal giocatore e la miniatura scelta. In base al colore della miniatura si inizializza la variabile intera *colore*. Importante in questo caso fare il controllo che non si passi il colore di una miniatura già scelta. Per ogni giocatore della partita ci possono essere massimo cinque colori distinti, e nessun colore aggiuntivo oltre a quelli considerati ovviamente, che corrispondono agli interi da 0 a 4. La variabile attivo viene inizializzata a 1 perché il giocatore parte in quel momento, una volta che ha fatto due azioni viene posta a zero e il giocatore va in wait. La variabile turno viene inizializzata a 1 perché si parte dal primo turno e così è un indice più leggibile (che non partire da zero). Il vettore che registra i danni e quello che registra i marchi vengono inizializzati a tutti zeri, mentre il vettore delle munizioni a tutti 1 (valore iniziale). Punteggio, azione, morte sono inizializzate a zero. Cel è inizializzato a -1, -1.

public String getname(). Ritorna il nome del giocatore.

public String getid(). Ritorna l'ID del giocatore.

public int getnumberdamage(int i). Ritorna il numero dei danni inflitti al giocatore da uno specifico nemico, passo come parametro il colore (indice numero del nemico), ricavato dal chiamante con la getcolor() fatta sul player.

public int gettotaldamage(). Ritorna il numero totale di danni subiti dal giocatore.

public int setdamage(int n, int c). Riceve come parametro il numero di danni da aggiungere, e il colore del giocatore che li ha inflitti. Se il colore passato ha lo stesso indice del player che subisce danno ritorna -1 (impossibilità di danni autoinflitti), altrimenti verifica quanto è la somma complessiva dei danni. Se la somma è pari a 11 il nemico ha solo il punto dell'uccisione e si restituisce 1 e in corrispondenza dell'indice del nemico si aggiungono tutti i danni passati come parametro, se la somma è superiore a 12 si restituisce 2 (punto extra dell'infierire) e si aggiungono solo i danni leciti residui (somma non superiore a 12). Altrimenti si restituisce 0 e si aggiungono normalmente tutti i danni.

public int getmarks(int c). Dato il colore c (indice giocatore avversario), ritorna il numero di marchi inflitti da quel giocatore. Ho messo valore di ritorno -1 se viene richiesto il numero di danni inflitti dal giocatore a se stesso, mas sarebbe meglio fare un test su questa cosa.

public int setmarks(int n, int c). Dato il colore c (indice giocatore avversario), e il numero di marchi da aggiungere ritorna -1 se si chiede il proprio numero di marchi inflitti (quelli con il proprio indice), ma vorrei mettere questa cosa come test. Altrimenti in caso il numero complessivo superi 3, aggiorna i marchi all'indice c con 3 (valore massimo), in caso contrario somma il numero di marchi nella cella indicata da c. Ritorna zero.

public int get_ammo(int c). Ritorna il numero di munizioni residue di colore con indice c, -1 se mando un colore che non è previsto (c non è compreso tra zero e 3).

public int add_ammo(int n, int c). Aggiunge il numero n di munizioni di colore c, ritornando -1 se c non è nel range dei colori disponibili, altrimenti aggiunge le munizioni all'indice del colore c, settando di default a 3 se la somma di munizioni già presenti e quelle da aggiungere supera il valore 3.

public int remove_ammo(int n, int c). Rimuove il numero di munizioni di colore c, ritornando -1 se c non è nel range dei colori disponibili, altrimenti controllando che l'operazione sia lecita, ovvero che ci siano abbastanza munizioni da prelevare (n<ammo[c]). Se l'operazione non è lecita ritorna -2, 0 altrimenti.

public int weaponIspresent(Weapon weapon). Riceve come parametro l'ID dell'arma e restituisce 1 se trova tra le armi del giocatore quella indicata, 0 altrimenti.

public int add_weapon(Weapon weapon). Aggiunge l'arma alla lista gun alla prima posizione libera, controllando prima che il giocatore non abbia già 3 armi (numero massimo), in quel caso ritorna -1, altrimenti aggiunge alla lista e ritorna zero.

public int remove_weapon(Weapon weapon). Rimuove l'arma passata come parametro dalla lista gun, controllando sia che la lista non sia nulla (in tal caso si ritorna direttamente -1 perché è impossibile rimuovere un'arma), sia che nella lista gun ci sia effettivamente l'arma da rimuovere.

public int pow_ispresent(Pow p). Riceve come parametro il potenziamento e restituisce 1 se trova tra i potenziamenti del giocatore quello indicato, 0 altrimenti.

public int add_pow(Pow p). Aggiunge il potenziamento alla lista pow alla prima posizione libera, controllando prima che il giocatore non abbia già 3 potenziamenti (numero massimo), in quel caso ritorna -1, altrimenti aggiunge alla lista e ritorna zero.

public int remove_pow(Pow p). Rimuove il potenziamento passato come parametro dalla lista pow, controllando sia che la lista non sia nulla, in tal caso ritorna -1, sia che il giocatore possieda il potenziamento che vuole rimuovere (se non ce l'ha ritorna -2). Altrimenti rimuove e ritorna zero.

public int get_cel(). Ritorna la cella in cui si trova il giocatore, le sue coordinate.

public void set_cel(int x, int y). Aggiorna le coordinate del giocatore sulla mappa.

public int get_death(). Ritorna la variabile death, il numero di volte che il giocatore è morto per l'attribuzione dei punteggi.

public void set_death(). Aggiorna la variabile che tiene conto del numero di volte che il giocatore è morto.

public int get_action(). Ritorna il numero di azioni che il giocatore ha compiuto fino a quel momento. Serve per capire quando il turno è terminato (se ritorna 2, si sono compiute due azioni e non se ne possono fare altre).

public void set_action(). Aggiorna la variabile che tiene conto del numero di azioni che il giocatore ha fatto,

public int get_score(). Ritorna il punteggio del giocatore.

public void set_score(int s). Aggiorna la variabile che conta il punteggio.

Controlli sulle variabili:

- Il colore è univoco, non si possono dare stringhe diverse dai colori predefiniti, e due giocatori non possono avere lo stesso colore.
- La stringa del nome è univoca, non ci possono essere giocatori con lo stesso nome. Stessa cosa per l'id.
- Il primo giocatore, quello che avrà la tessera di giocatore iniziale, andrà nella prima posizione dell'arraylist che tiene traccia dei giocatori.
- La lista che registra i danni e quella che registra i marchi hanno 5 elementi ciascuno, tuttavia al giocatore considerato possono fare danno i quattro giocatori avversari quindi si dovrà controllare ogni volta che si aggiornano i valori della lista che in corrispondenza dell'indice pari al valore di ritorno della getcolor() chiamata sul giocatore ci sia sempre il valore zero. Ovvero in parole povere che per nessun giocatore si registrino marchi o danni autoinflitti, nella cella corrispondente all'indice del proprio colore si abbia sempre valore nullo.
- La lista che registra i danni ha come somma massima dei propri elementi il valore 12.
- La somma degli elementi della lista danni è significativa per stabilire le mosse lecite ed eventuali miglioramenti.