SPIEGAZIONE DEL PROGRAMMA

Strategie sviluppate dal gruppo:

Rintontino1, Rintontino2,

UnDueTreCibo

Storia ed evoluzione del programma:

Come prima idea abbiamo deciso di far girare il nostro batterio in cerchio in attesa di ricevere del cibo nella sua posizione, ma è risultata fallimentare poiché il cibo non viene creato sempre in quel punto; la seconda idea consisteva nel portare tutti i batteri allineati sull'asse delle y e contro il bordo di sinistra e farli muovere in linea retta, anche questa idea è risultata fallimentare; la terza idea che è stata poi scelta consiste nel cercare del cibo intorno a sé in un'area di dimensioni prestabilite e in caso di risultato positivo si sarebbe teletrasportato sulla posizione del cibo (questa soluzione però è risultata poco efficiente e molto costosa nel caso in cui si volesse ingrandire l'area oltre le 5 unità). In caso in cui non si fosse trovato cibo il batterio si sarebbe mosso randomicamente di un'unità; un'altra idea consisteva nel far muovere il batterio randomicamente per il terreno di gioco e cerca anche il cibo in posizioni randomiche, ci siamo accorti che rischiavamo però di farlo morire per lo sforzo e la ricerca non era affidabile, oltre alla difficoltà di scrittura di un codice con controlli randomici, il codice di quest'idea non è mai stato scritto.
Abbiamo deciso di chiamare il nostro batterio Rintontino (a.k.a Et2) e lavorando sul codice della terza strategia abbiamo reso la sua esecuzione più rapida e l'area più grande.
Tentando di migliorare la velocità di esecuzione abbiamo aggiunto due break (poi sostituiti da un return) per uscire dai due cicli for, in modo da evitare la completa esecuzione di tutti i giri dei cicli ed eventuali errori nell'aggiornamento della posizione del batterio, una variabile per decidere movimento da utilizzare poi eliminata e ripresa nella seconda strategia, un attributo che poi moltiplicherà l'indice presente nel for per il controllo del cibo in modo da aumentare la distanza della posizione controllata nella ricerca del cibo e lo spostamento del batterio in caso il cibo venga trovato, per evitare

che il costo computazionale salisse troppo abbiamo deciso di diminuire il numero di cicli effettuati di entrambi i for.

È stato sviluppato un secondo tipo di batterio. L'idea consisteva nel farlo rimanere fermo fino a quando non avrebbe trovato del cibo intorno a sé basandosi sulla ricerca utilizzata in Rintontino, in questo modo però finchè non si sarebbe mosso la sua riproduzione non sarebbe potuta avvenire, questo metodo si è rivelato molto deludente e poco preciso nella ricerca del cibo inoltre moriva prima di riprodursi a causa dell'età. Per rimediare alla scarsa efficacia della ricerca iniziale sono state utilizzate due aree di ricerca differenti, la prima, la ricerca MAXI viene attivata quando nella zona circostante non si trova del cibo ed esegue una ricerca in un'ampia area intorno al batterio; la seconda, la ricerca MINI, viene attivata quando viene trovato del cibo ed esegue un controllo in una piccola area intorno a sé. Per rimediare alla mancata riproduzione in caso non venga trovato cibo il batterio si sposterà di un'unità per volta in una direzione casuale scelta randomicamente al momento della sua creazione e rimbalzerà contro i bordi della schermata.

Nell'ultima versione del batterio la ricercaMini è stata sostituita da 4 ricerche che si differenziano tra di loro per il modo in cui mangiano, la prima mangia il blocco di cibo dal basso verso l'alto, la seconda mangia dall'alto verso il basso, la terza mangia da sinistra verso destra e l'ultima da destra verso sinistra; in questo modo i batteri hanno una velocità superiore a mangiare il cibo rispetto alla versione precedente. La modalità in cui il batterio mangia viene scelta randomicamente ogni volta che ricercaMaxi trova del cibo.

Per evitare di rendere il codice troppo costoso nelle ricercheMini sono stati tolti i moltiplicatori e sostituiti da un ciclo for più grande e un incremento del contatore aumentato da 1 a 2, in questo modo manteniamo una grande area di ricerca senza aumentare il costo.

SPIEGAZIONI DELLE CLASSI

CLASSE FOOD

La classe *food* gestisce la distribuzione del cibo in vari modi:

-randomDistribution:

Distribuisce il cibo secondo una distribuzione casuale.

-cornerDistribution:

Questo metodo distribuisce il cibo secondo una distribuzione che distribuisce solo negli angoli.

-squareDistribution:

Il metodo squareDistribution distribuisce il cibo secondo una distribuzione quadrata.

-isFood:

Questo metodo controlla se c'è cibo in posizione x,y.

-eatFood:

Mangia il cibo in posizione x,y.

CLASSE BATTERIO

La classe *batterio* è una classe astratta genitore della gerarchia dei batteri. Ogni tipo diverso di batterio eredita da questa classe.

-Sposta:

Questo metodo sposta il batterio nel terreno. Deve essere ridefinita nelle classi ereditate per dar loro un comportamento diverso.

-ControllaCibo:

Il metodo ControllaCibo controlla se c'è del cibo nella posizione x,y.

-ControllaCibo

Controlla se c'è del cibo nella posizione occupata dal batterio.

-Mangia:

Il metodo mangia controlla se nella posizione occupata dal batterio c'è del cibo lo mangia e incrementa la sua salute di DELTA.

-Fecondo:

Questo metodo controlla se un batterio è fecondo.

-Morto:
Controlla se un batterio è morto o perché troppo vecchio o perché non ha abbastanza salute.
-Run:
Run esegue le mosse del batterio, inoltre viene diminuita la sua età e gli viene fatto mangiare il cibo.
In più viene calcolato lo sforzo e sottratto alla salute del batterio.
Successivamente viene diminuito il tempo di attesa per poi essere clonato solo se il batterio è in movimento.
-Clona:
Clona il batterio in senso biologico.
Un nuovo batterio creato con la stessa posizione di quello originale.

CLASSE TERRAIN

Questa classe serve a mostrare sul campo di gioco i batteri e tutto ciò che viene mostrato sul campo.
-toggle food
In toggle food viene creato nuovo cibo usando la squareDistribution.
-paintComponent
paint component colora gli elementi che sono presenti nello schermo quindi tutti i batteri, tutto il cibo e rimuove o aggiunge i batteri quando nascono e quando muoiono sempre per colorarli e clonareli quando sono fecondi e aggiunge i batteri clonati alla lista di batteri.
-getPreferredSize
La getPreferredSize ritorna la grandezza del campo di gioco.

CLASSE UnDueTreCibo

La classe *UnDueTreCibo* è la classe da noi sviluppata la quale contiene le seguenti classi che si occupano della gestione del movimento dei batteri.
-ricercaMiniSx:
Metodo che ricerca del cibo in una piccola area intorno al batterio partendo da sinistra e modifica la sua posizione in caso positivo.
-ricercaMiniDx:
Questo metodo che ricerca del cibo in una piccola area intorno al batterio partendo da destra e modifica la sua posizione in caso positivo.

-ricercaMiniDown

Metodo che ricerca del cibo in una piccola area intorno al batterio partendo dal basso e modifica la sua posizione in caso positivo.

-ricercaMiniUp

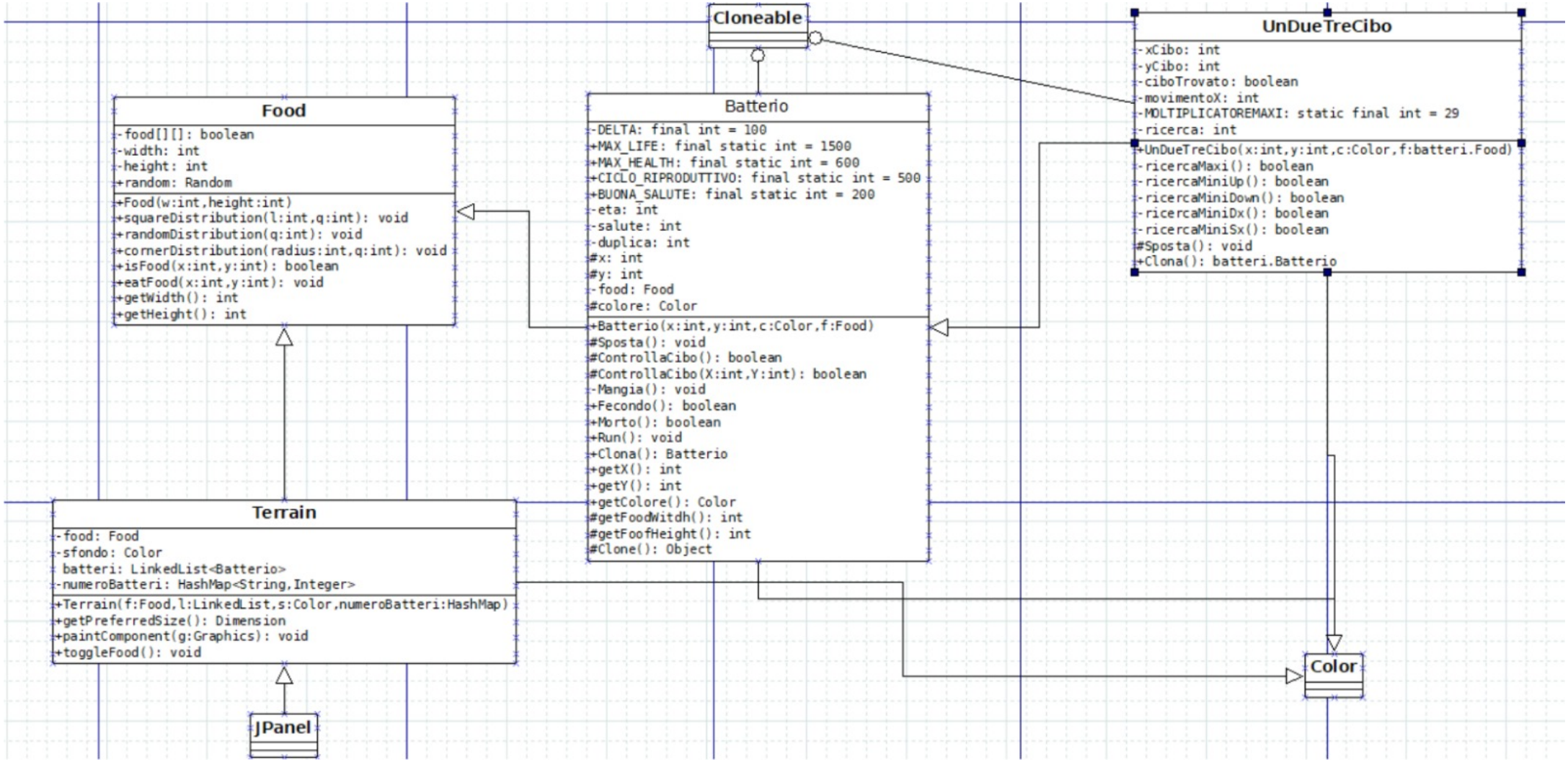
Metodo che ricerca del cibo in una piccola area intorno partendo al batterio dall'alto e modifica la sua posizione in caso positivo.

-ricercaMaxi

Metodo che ricerca del cibo in una grande area intorno al batterio e modifica la sua posizione in caso positivo.

-Sposta:

Il metodo sposta permette al batterio di muoversi nel campo di gioco.
Comprende 5 tipi diversi di ricerca una per il cibo vicino e una per il cibo lontano. La ricerca lontana la prima ad essere eseguita.
Se viene trovato del cibo verrà scelta casualmente una delle ricerca mini.
Ogni volta che il batterio trova del cibo si teletrasporterá su di esso per mangiarlo. Non vengono mai eseguite tutte le ricerche insieme.
Quando non viene trovato del cibo si muoverà in direzioni casuali e rimbalzerà sui bordi.


Cloneable
UnDueTreCibo
-xCibo: int
-yCibo: int
-ciboTrovato: boolean
-movimentoX: int
-MOLTIPLICATOREMAXI: static final int = 29
-ricerca: int
+UnDueTreCibo(x:int,y:int,c:Color,f:batteri.Food)
-ricercaMaxi(): boolean
-ricercaMiniUp(): boolean
-ricercaMiniDown(): boolean
-ricercaMiniDx(): boolean
-ricercaMiniSx(): boolean
#Sposta(): void
+Clona(): batteri.Batterio
Food
-food[][]: boolean
-width: int
-height: int
+random: Random
+Food(w:int,height:int)
+squareDistribution(l:int,q:int): void
+randomDistribution(q:int): void
+cornerDistribution(radius:int,q:int): void
+isFood(x:int,y:int): boolean
+eatFood(x:int,y:int): void
+getWidth(): int
+getHeight(): int
Batterio
-DELTA: final int = 100
+MAX_LIFE: final static int = 1500
+MAX_HEALTH: final static int = 600
+CICLO_RIPRODUTTIVO: final static int = 500
+BUONA_SALUTE: final static int = 200
-eta: int
-salute: int
-duplica: int
#x: int
#y: int
-food: Food
#colore: Color
+Batterio(x:int,y:int,c:Color,f:Food)
#Sposta(): void
#ControllaCibo(): boolean
#ControllaCibo(X:int,Y:int): boolean
-Mangia(): void
+Fecondo(): boolean
+Morto(): boolean
+Run(): void
+Clona(): Batterio
+getX(): int
+getY(): int
+getColore(): Color
#getFoodWitdh(): int
#getFoofHeight(): int
#Clone(): Object
Terrain
-food: Food
-sfondo: Color
batteri: LinkedList<Batterio>
-numeroBatteri: HashMap<String,Integer>
+Terrain(f:Food,l:LinkedList,s:Color,numeroBatteri:HashMap)
+getPreferredSize(): Dimension
+paintComponent(g:Graphics): void
+toggleFood(): void
Color
JPanel