Conway's Hive : relazione progetto

Studente: Stojkovic Danilo 1222399

Prefazione

Lo scopo del progetto consiste nella creazione di un'applicazione a scelta libera con alcuni vincoli generali utilizzando la libreria Qt e il linguaggio c++. La principale fonte di ispirazione per lo sviluppo di questo programma è stato l'automa cellulare Conway's game of life, referenziato anche nel titolo.

Descrizione Generale

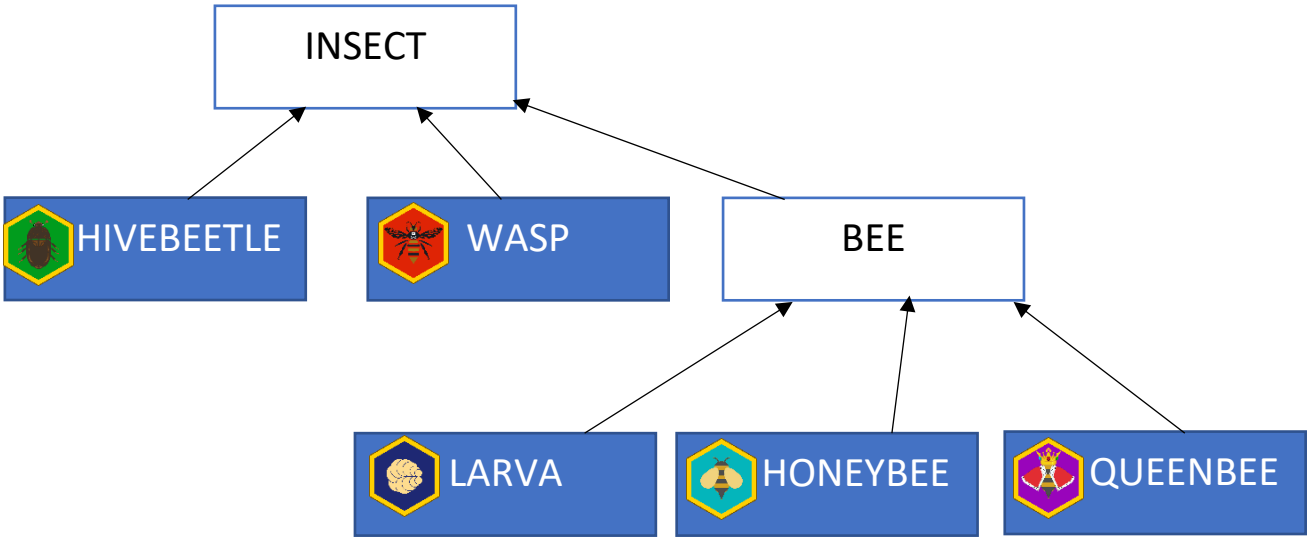
Un automa cellulare è un modello matematico che delinea l'evoluzione di sistemi discreti, più semplicemente è un insieme di regole che definiscono come si trasformerà lo stato di una determinata griglia basandosi sullo stato attuale. Lo stato è semplicemente dato dal contenuto di ogni cella (ad esempio in Conway's game of life ogni cella può essere viva o morta). I giochi che si basano su questo tipo di modelli consistono nell'istanziare uno stato iniziale della griglia ed attivare la successione di turni che ne varierà la configurazione seguendo regole precise e mai casuali.

Questo progetto in particolare è un automa cellulare che prende spunto dal mondo delle api; le celle della griglia 16x11 saranno infatti esagonali e potranno contenere una varietà di insetti appartenenti a questo ambiente nonché essere riempite di miele. L'obiettivo principale del gioco è infatti accumulare la maggior quantità di miele possibile, prodotto dalle api. Ad ogni turno infatti ogni insetto farà una mossa:

- Le api si sposteranno nella prima cella adiacente senza miele per produrne ancora;
- Le larve attenderanno 3 turni prima di diventare delle api;
- L'ape regina (una sola) produrrà nella prima cella adiacente libera una larva ogni 5 turni;
- Il coleottero degli alveari si sposterà in una delle celle raggiungibili con miele per consumarne, altrimenti si muoverà semplicemente in qualche cella libera;
- La vespa velatina si sposterà con l'obiettivo di mangiare una delle api vicine o in una cella vuota in assenza di api.

La Gerarchia

La specifica del progetto richiedeva una gerarchia di almeno 3 diversi tipi istanziabili, una classe astratta e un altezza >=2. La gerarchia utilizzata nell'applicativo è qui rappresentata (le classi non colorate al proprio interno sono astratte).



- INSECT: classe base astratta che permetterà di collezionare i vari insetti insieme e di fare chiamate polimorfe. Contiene un distruttore virtuale ed un metodo virtuale puro (play). Contiene inoltre tutti gli attributi in comune di ogni insetto, in particolare il cell Index.
- HIVEBEETLE: classe derivata da INSECT, rispetto a questa classe aggiunge un attributo che conta la quantità di miele consumata e due attributi statici che avrà ogni classe istanziabile: numberOf (che conta il numero di questi oggetti attualmente sulla griglia) e ID (che semplicemente contiene l'ID assegnato a questa classe in particolare per render il codice più leggibile). Infine è presente un override della funzione virtuale play() che detta le regole che ogni oggetto di questo tipo deve rispettare al passaggio di turno.
- WASP: classe derivata da INSECT, l'espansione è molto simile a quella di HIVEBEETLE, con la differenza che questa divora le api.
- BEE: classe astratta derivata da Insect, sarà supertipo delle rimanenti 3 classi.
- HONEYBEE: derivata da BEE, insetto principale del gioco. Sono presenti l'overriding, numeberOf e ID e questa classe contiene anche una funzione getHoneyMade() che permette di aggiornare il contatore totale del miele prodotto nelle statistiche.
- LARVA: derivata da BEE, l'overriding di play prevede il conteggio dei turni di esistenza per poi creare una nuova ape ed eliminare la larva presente nella griglia.
- QUEENBEE: derivata da BEE, il suo overriding di play prevede la ricerca della prima cella disponibile e la creazione di una nuova larva al suo interno.

CHIAMATE POLIMORFE

- Come già detto, la chiamata polimorfa principale del programma consiste nell'esecuzione del metodo play di ognuno degli insetti che si trova sulla griglia di gioco.

```
void Game::playTurn()
{
    turn++;
    for(auto i=s.begin(); i!=s.end(); i++)
        (*i)->info->play(grid, &s);
}
```

  La variabile **s** è un'istanza del contenitore swarm di cui si parlerà più avanti. Il ciclo che si trova in questo metodo, invocato all'inizio di ogni turno, scorre semplicemente il contenitore di puntatori deep a oggetti insect (tipo statico) e chiama il metodo play di ognuno di loro. Il fatto che play è una funzione virtuale di insect permette chiamate polimorfe e perciò ad ogni insetto di agire secondo le proprie regole.

- Quando il gioco inizia premere su una cella non cambierà più la configurazione bensì aprirà un messageBox contenente una statistica particolare dell'insetto cliccato. La funzione che si occupa di questo nella classe Game contiene una chiamata polimorfa, essendo getStatOf() una funzione virtuale di Insect.

```
std::string Game::getStatOf(int index)
{
    for(auto i=s.begin(); i!=s.end(); i++)
        if(i->info->getIndex()==index)
            return (*i)->info->getMainStat();
    return "";
}
```

CONTENITORE

Una delle richieste della specifica era la definizione di un template di classe C<T> che fungesse da contenitore del tipo generico T (nel programma questo tipo corrisponde sempre a DeepPtr<Insect>). Il template-contenitore di questo progetto è Swarm<T> e si basa su una lista di nodi (classe node definita all'interno della sua parte privata) ognuno con campo T* info e node* next. Swarm contiene anche una classe iterator che permette di scorrere tra i suoi oggetti ed infine alcuni metodi base come begin(), end(), insert(T*), remove(T*), deepCopy(node*) e destroy(node*). Gli ultimi due in particolare assicurano che l'assegnazione di un swarm ne causi la clonazione e non una shallow copy e che la sua distruzione porti alla distruzione di ogni suo oggetto.

DEEP PTR

DeepPtr<T> è un template di classe che rappresenta dei puntatori polimorfi smart per la gestione profonda della memoria, ogni insetto della griglia è un oggetto INSECT (o un suo sottotipo) che viene puntato da un DeepPtr<Insect> e Swarm è una loro collezione. Tra i suoi attributi questo template di classe contiene un puntatore regolare al tipo generico T; tra i metodi la rule of three è stata rispettata in quanto costruttore di copia, assegnazione e distruttore sono stati ridefiniti in modo da non essere shallow. Infine gli operatori uguaglianza e dereferenziazione si assicurano che questi metodi siano applicati agli oggetti puntati e non ai puntatori.

ALTRE CLASSI: Cell & Game

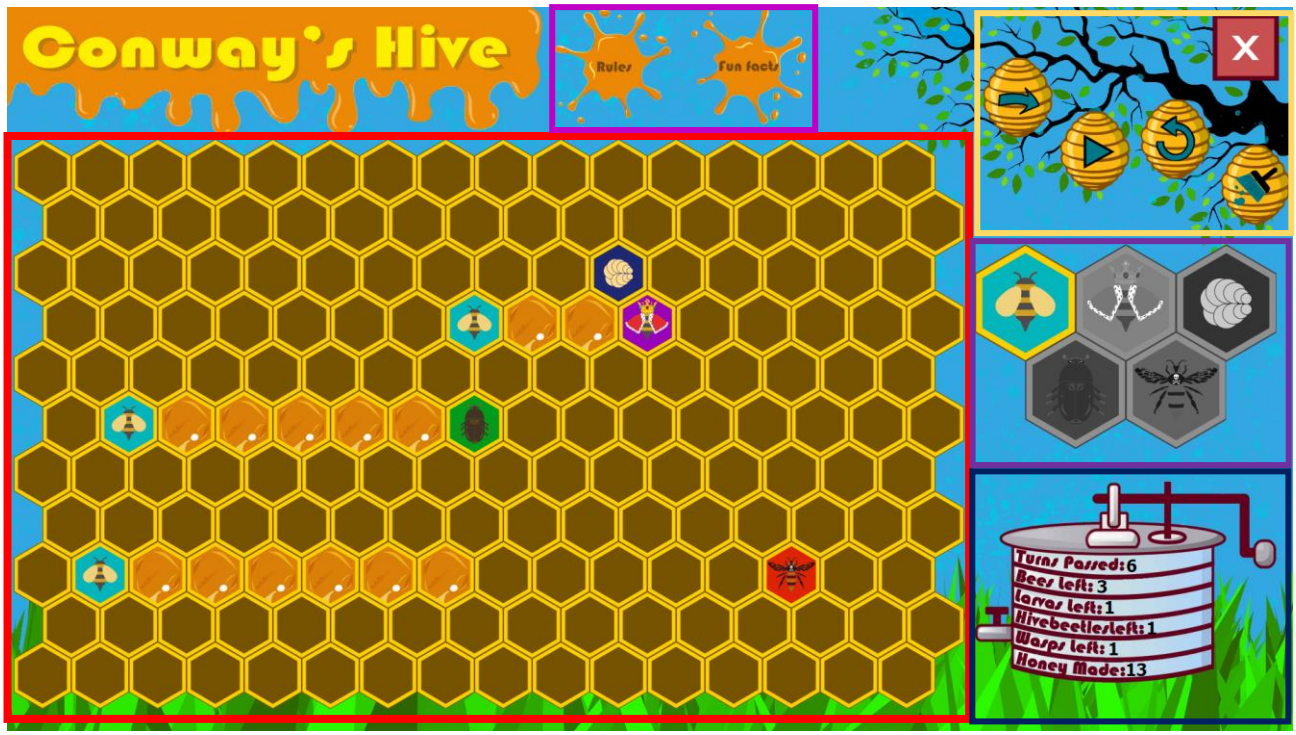
Cell è la classe le cui istanze rappresentano le varie posizioni della griglia, contiene un intero che indica il suo stato attuale (vuota, con miele o quale insetto la occupa) e il metodo statico fondamentale getAdjecentCells(int) che restituisce un vector delle posizioni raggiungibili dalla cella il cui indice è passato come parametro, la natura esagonale della griglia utilizzata nel gioco ha richiesto una particolare attenzione nella scrittura di queste righe di codice.

La classe Game si occupa infine di controllare la partita corrente, essa ha infatti lo `Swarm<DeepPtr<Insect>>` e un `std::vector<Cell*>` come attributi e il suo compito principale è chiamare i metodi corretti delle altre classi in base a cosa è stato richiesto dell'utente o dalle regole del gioco.

DESIGN PATTERN

Per questo progetto è stato scelto il design pattern Model-View-Controller che permette una decisa separazione tra il modello (dove non si fa uso di alcuna libreria Qt) e la View (dove è indispensabile usare metodi e classi di Qt per creare la GUI). Il modello infatti, costituito dalle classi Insect e sottotipi, Game e Cell può tranquillamente essere riutilizzato in ambienti di sviluppo diversi da Qt creator. La classe controller si occupa del collegamento tra le classi View e quelle del Model, i suoi metodi sono infatti sempre molto semplici e si limitano a trasferire al model la richiesta effetuata dalla view.

GUI



In questo screenshot della schermata principale del gioco è possibile vedere la configurazione della GUI.

- Griglia Principale: 11x16 celle nelle quali è rappresentato lo stato attuale del gioco;
- Menu Selezione: selezionando uno degli insetti con un click è possibile aggiungerne quando la partita è in pausa;
- Bottoni di Controllo: in ordine da sinistra a destra: next fa procedere il turno, play/pause attiva un timer che ogni tot secondi svolge il turno automaticamente, restart riporta la griglia alla disposizione iniziale e clear svuota la griglia. Infine il bottone X chiude il programma;
- Statistiche: questa grafica contiene tutte le statistiche principali del gioco come il numero di un certo tipo di insetto, i turni passati o il miele prodotto;
- Rules & Fun Facts: questi due bottoni aprono due nuovi form nei quali sono spiegate le regole del gioco e sono elencati alcuni fatti curiosi sulle api rispettivamente.

DISTRUBUZIONE delle ORE di LAVORO

- Apprendimento libreria Qt: 6h
- Analisi preliminare del problema, ideazione regole di gioco: 6h
- Creazione immagini da utilizzare nella grafica: 5h
- Progettazione GUI: 2h
- Codifica View: 8h
- Scrittura UML: 2h
- Codifica Model: 12h
- Codifica Controller: 1h
- Debugging: 5h
- Testing: 3h

Totale: 50h

SPECIFICHE

- Sistema Operativo di sviluppo: Windows 10
- Versione Qt: 5.15.2
- Versione Compilatore: g++ (MinGW.org GCC-6.3.0-1) 6.3.0

ISTRUZIONI COMPILAZIONE

File .pro fornito perché è stato modificato quello originale.

MODIFICHE RISPETTO A CONSEGNA PRECEDENTE

Un nuovo progetto è stato creato e saranno inclusi i due file UI di FunFactsDialog e RulesDialog che sono stati comunque lasciati com'erano quando sono stati creati. Il campo NumberOf è stato sostituito da un metodo che conta le occorrenze di ogni tipo di insetto nello swarm.