04/10/2021

Progetti info3

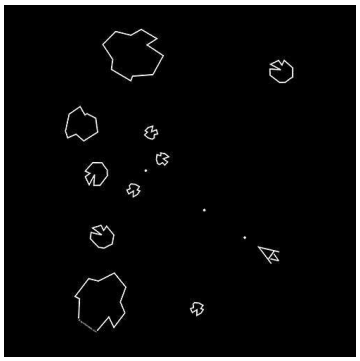
38068-1 MODULO DI PROGRAMMAZIONE

Dario Sardi
d.sardi@studenti.unibg.it

Sommario

Progetto C++ (Asteroids)

Introduzione.



È stato realizzato in grazie all'utilizzo delle librerie Qt il classico gioco asteroids. Lo scopo del gioco è sparare agli asteroidi con la propria navetta spaziale fino a distruggerli tutti. Ogni asteroide parte da una dimensione Grande, e si divide ogni volta che viene colpito in 2 medi che a loro volta si divideranno in 2 piccoli asteroidi.

Classe asteroid

La classe asteroid è la classe base da cui erediteranno le classi Large,Medium e Small.

```
class Asteroid: public QObject,public QGraphicsPixmapItem
{
    Q_OBJECT
public:
    Asteroid(MyGame *g);
    virtual ~Asteroid();
    virtual void kill()=0;
    void hit();
    QString toString();
public slots:
    void move();
protected:
    MyGame *game;
    int speed =10;
    QTimer *asteroidTimer;
};
```

La classe contiene come funzioni oltre a costruttore e distruttore:

- virtual *kill()* che si occuperà di generare asterodi a catena dopo la distruzione del primo da ridefinire nelle sottoclassi
- una funzione per il movimento move()
- una funzione hit() per la routine dopo che l'asteroide è stato colpito.
- un riferimento al gioco stesso **game* per l'interazione con l'interfaccia grafica.

Costruttore

```
Asteroid::Asteroid(MyGame* g) {
    game =g;
    asteroidTimer = new QTimer();
    connect(asteroidTimer,SIGNAL(timeout()),this,SLOT(move()));
    asteroidTimer->start(20);
    setRotation(QRandomGenerator::global()->bounded(360 * 16));

}
```

alla creazione di un asteroide viene avviato un timere (QTimer) che invierà un segnale ogni qualvolta scadrà. Questo segnale viene connesso a una funzione di tipo SLOT responsabili di attivarsi a seguito di un segnale, in questo caso la funzione è la move() che verrà chiamata ogni 20ms per aggiornare la posizione dell'asteroide in base a direzione e velocità.

move()

La funzione move si occupa di aggiornare la posizione dell' asteroide tenendo conto della velocità e della direzione dell'oggetto.

Poiché il sistema di riferimento degli assi è invertito per l'asse Y e il senso di rotazione positivo è inverso rispetto al tradizionale le equazioni trigonometriche sono state adattate di conseguenza.

nel caso in cui l'asteroide uscisse dalla schermata rientrerà nel gioco dal lato opposto a quello da cui è uscito.

```
void Asteroid::move(){
    double angle=-(rotation()+180)*M_PI/180;
    setPos(x()+speed*sin(angle),y()+speed*cos(angle));
    if(x()>800){
        setX(0);
    }
    else if (x()<-5){
        setX(800);
    }

    if(y()>600){
        setY(0);
    }
    else if(y()<-10){
        setY(600);
    }
//     qDebug() << "asteroid online at" << x() << ", "<< y() ;
}
```

hit()

La funzione hit() viene invocata quando l'asteroide è stato colpito e serve per chiamare la funzione di kill() che verrà poi ridefinita nelle sottoclassi a seguito della rimozione dell'oggetto grafico.

```
void Asteroid::hit(){
    scene()->removeItem(this);
    asteroidTimer->stop();
    this->kill();
}
```

Sottoclassi

Le sottoclassi ridefiniscono la texture dell'asteroide, determinano una velocità in base alla dimensione e effettuano l'override del metodo kill per ottenere l'effetto desiderato.

E cosi:

- La classe Large creerà due asteroidi medi e rimuoverà se stessa
- La classe Medium creerà due asteroidi piccoli e rimuoverà se stesssa
- La classe Small rimuoverà se stessa e chiederà alla classe gioco di controllare se fosse l'ultimo asteroide

```
Large::Large(MyGame* g):Asteroid(g){
    setPixmap(QPixmap(":/sprites/deathstar.png").scaled(QSize(60,60),Qt::KeepAspectRatio));
}

void Large::kill(){
    game->spawnMedium(x(),y());
    game->spawnMedium(x(),y());
    delete this;
}

Medium::Medium(MyGame* g):Asteroid(g){
    setPixmap(QPixmap(":/sprites/tie.png").scaled(QSize(40,40),Qt::KeepAspectRatio));
    this->speed=5;
}
void Medium::kill(){
    game->spawnSmall(x(),y());
    game->spawnSmall(x(),y());
    delete this;
}

Small::Small(MyGame* g):Asteroid(g){
    setPixmap(QPixmap(":/sprites/tie.png").scaled(QSize(25,25),Qt::KeepAspectRatio));
    this->speed=5;
}
void Small::kill(){
    delete this;
    this->game->victory();                                    Use of mem
}
```

Spaceship

la classe spaceship rappresenta la navicella spaziale pilotata dal giocatore.

Puo muoversi tramite frecce direzionali e sparare con la barra spaziatrice.

```
class Spaceship: public QObject,public QGraphicsPixmapItem{
    Q_OBJECT
private:
    QTransform trans;
    int speed;
    QSet<Qt::Key> keysPressed;


public:
    void keyPressEvent(QKeyEvent * event);
    Spaceship(MyGame *g);
    MyGame *game;
    void keyReleaseEvent(QKeyEvent *event);
public slots:
    void move();
};
```

la classe possiede una lista di tasti premuti poiché la pressione simultanea di piu tasti non era gestita. I tasti premuti vengono dunque aggiunti alla lista alla pressione e rimossi al rilascio del tasto grazie alle funzioni ereditate da QObject keypressEvent e keyReleaseEvent.

```
void Spaceship::keyPressEvent(QKeyEvent *ev){
  keysPressed += (Qt::Key)ev->key();
}
void Spaceship::keyReleaseEvent(QKeyEvent *ev){
  keysPressed -= (Qt::Key)ev->key();
}
```

move()

La funzione move si occupa di muovere la navicella e creare i proiettili

```
void Spaceship::move(){
    foreach(Qt::Key k, keysPressed){
      switch(k){
      case Qt::Key_Left:setRotation(rotation()-15);
                        break;
      case Qt::Key_Right:setRotation(rotation()+15);
                        break;
      case Qt::Key_Up: if(speed<=30){speed+=2;};
                        break;
      case Qt::Key_Space:{
                    double angle=-(rotation()+180)*M_PI/180;
                    Projectile *p = new Projectile(this);
                    p->setPos(x()+30+30*sin(angle),y()+30+30*cos(angle));
                    p->setRotation(rotation());
                    scene()->addItem(p);};
                    break;
      default:break;
      }
    }
    double angle=-(rotation()+180)*M_PI/180;
    setPos(x()+speed*sin(angle),y()+speed*cos(angle));
    if(x()>800){
        setX(0);
    }
    else if (x()<-5){
        setX(800);
    }

    if(y()>600){
        setY(0);
    }
    else if(y()<-10){
        setY(600);
    }

    if(speed>0){
    speed-=0.1;}
}
```

Se la navicella esce dai limiti della finestra rientrerà dal lato opposto.

La velocità della navicella cresce con una accelerazione lineare fino a raggiungere le 30 unità in caso di pressione della freccia UP e decresce costantemente a ogni ciclo in modo da creare un effetto di “rallentamento” naturale.

Nel caso si prema la barra spaziatrice viene generato un proiettile con la stessa direzione della navicella.

Proiettile

Il proiettile viene generato da una navicella, ne possiede un riferimento per l’interazione con il resto del gioco ed è dotato di una funzione per il movimento proprio come gli altri elementi del gioco.

```
class Projectile: public QObject,public QGraphicsPixmapItem{
    Q_OBJECT //macro needed for sign-slots
public:
    Projectile(Spaceship *s);
    int speed =40;
    ~Projectile();
public slots:
    void move();
private:
    QTimer *bulletTimer;
    void check();
    Spaceship *mother;
};
```

Possiede però una funzione di check con cui controlla a ogni ciclo di non essersi scontrato con uno degli asteroidi presenti nel gioco.

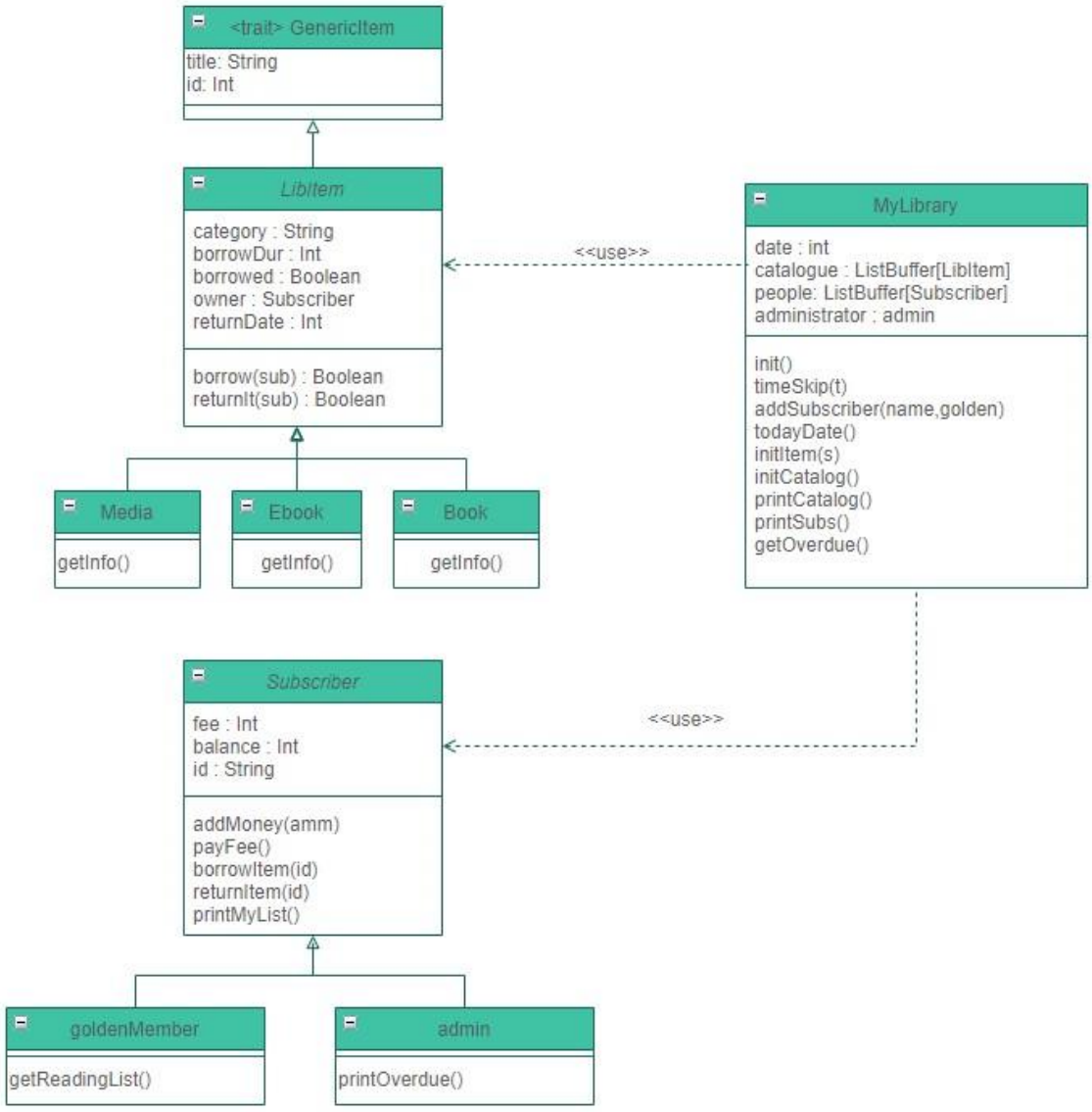
```
void Projectile::check(){
    for(Asteroid *ast: mother->game->asteroidsList){
        bool xCollision= (x()>ast->x()-30) && (x()<ast->x()+30);
        bool yCollision= (y()>ast->y()-30) && (y()<ast->y()+30);
        if(xCollision && yCollision){
            qDebug() << "HIT! remaining:" << mother->game->asteroidsList.length();
            ast->hit();
            mother->game->asteroidsList.removeAll(ast);
//            qDebug() << "removed! remaining:" << mother->game->asteroidsList.length();
            if(this->mother->game->asteroidsList.length()==0){
                mother->game->victory();
            }
            delete this;
            break;
        }
    }
}
```

Scala (gestore prestiti libreria)

Il programma

È stato realizzato un sistema di gestione dei prestiti per una libreria. Composto da oggetti che possono esser prenotati, degli abbonati che possono richiedere prestiti e una singola libreria contenente tutti i dati necessari

Struttura



LibItem

Gli oggetti da richiedere in prestito ereditano da una classe astratta LibItem inizializzata con un titolo e un id unico e sequenziale. Per puro esercizio la classe estende un trait che definisce un generico oggetto GenericItem grazie a un id e a un titolo.

```
trait GenericItem{
  val title: String;
  val id:Int;
}

abstract class LibItem(val title:String, val id:Int) extends GenericItem{
  import scala.Subscriber
  def category    :  String
  def borrowDur   :  Int
  var borrowed    :  Boolean  = false
  var owner:Subscriber        = null
  var returnDate  :  Int      = 0
```

un LibItem è caratterizzato da una durata del prestito, da una flag per lo stato di prestito del libro, da un eventuale subscriber che ha preso in prestito il libro e da una data di restituzione espressa come int (di default a 0 se l'elemento non è in prestito.

classi figlio

Book, Ebook e Media ereditano da LibItem definendo category e borrowDur diverse.

```
class Book(val t:String,val i:Int) extends  LibItem(t,i){
    def category = "Book"
    def borrowDur = 90
    def getInfo() = "Book title: "+title+ " ,max borrow: "+borrowDur
}

class EBook(val t:String,val i:Int) extends LibItem(t,i){
  def category = "EBook"
  def borrowDur = 120
  def getInfo() = "EBook title: "+title+ " ,max borrow: "+borrowDur
}

class Media(val t:String,val i:Int) extends  LibItem(t,i){
  def category = "Media"
  def borrowDur = 30
  def getInfo() = "Media title: "+title+ " ,max borrow: "+borrowDur
```

Sistema di prestito

Il prestito viene gestito dall'oggetto stesso. L'oggetto riceve una richiesta da un Subscriber, verifica se è possibile essere preso in prestito e che il richiedente non abbia multe arretrate.

```
/* check if free
 * set as borrowed
 * set owner
 * set returnDate
 */
def borrow(sub : Subscriber)   :  Boolean  ={
  if(!borrowed && sub.fee==0){
    borrowed=true
    owner=sub
    returnDate = MyLibrary.todayDate()+borrowDur*(if (sub.isInstanceOf[goldenMember]) 2 else 1)
    return true
  }
  else if(borrowed){
    println("the book is already borrowed")
    return false
  }
  else if(sub.fee>0) {
    println("you cannot borrow until you pay the fee")
    return false
  }
  else{
    println("borr. error")
    return false
  }

}
```

Sistema di restituzione

La restituzione viene anch'essa gestita dall'oggetto stesso che controlla che il libro appartenga effettivamente a colui che sta dichiarando di aver restituito il libro, calcola eventuali multe e aggiorna il suo stato.

```
  /* check if you are the owner
   * check if you are not late
   * set not borrowed
   * unset owner
   * reset returnDate
   */
  def returnIt(sub : Subscriber) :  Boolean  ={
    if(owner.id == sub.id){
      if(MyLibrary.todayDate()>returnDate){
        if( sub.isInstanceOf[goldenMember]){
          sub.fee+=((MyLibrary.todayDate()-returnDate)*0.5).toInt
        }
        sub.fee+=MyLibrary.todayDate()-returnDate
      }
      borrowed = false
      owner = null
      returnDate = 0
      return true
    }
    return false
  }
```

toString

un metodo getInfo() viene definito nella classe astratta per poi esser implementato nelle classi figlio.

Subscriber

Gli iscritti alla libreria definiti Subscriber in questo esercizio son caratterizzati da un nome, un id univoco e da un saldo per pagare eventuali multe.

```
class Subscriber(val name: String) {

  var fee = 0
  var balance = 0
  var id: String = Random.nextInt().toString()
  var myItems:ListBuffer[LibItem] = new ListBuffer()
```

GoldenMember

Gli abbonati golden sono abbonati speciali che hanno accesso alle liste di lettura degli altri utenti e che ricevono uno sconto del 50% nel caso di multa.

```
class goldenMember(val n:String) extends Subscriber(n){
  def getReadingList(s : Subscriber)={
    s.printMyList()
  }
}
```

Admin

È stato creato un amministratore di sistema come singleton estendendo la classe abbonato.
L'amministratore è in grado di vedere la lista degli abbonati con multe.

```
//single obj
object admin extends Subscriber("ADMIN"){
  def printOverdue()={
    println("-"*40+"\noverdue list \n"+"-"*40)
    MyLibrary.getOverdue()
  }
}
```

Fee system

Gli abbonati ricevono una unità di multa per ogni giorno di ritardo nella restituzione.
La multa puo essere pagata tramite i crediti dell'abbonato.

```
def addMoney(amm : Int ):{
  balance+=amm
}
def payFee():Boolean={
  if (balance>=fee){
    fee=0
    balance-=fee
    println("payment completed")
    return true
  }
  else{
    println("not enough money to pay your fee.\ncurrent fee: "+fee)
    return false
    }
}
```

Prestito e restituzione

Nel caso di richiesta di prestito l'elemento con l'id corrispondente a quello richiesto viene cercato nel catalogo della biblioteca, viene estratto il primo risultato disponibile e nel caso di successo l'elemento viene aggiunto alla propria lista.

```
def borrowItem(id : Int):{
  if(MyLibrary.catalogue.filter(p => p.id == id).head.borrow(this)){
    myItems+=MyLibrary.catalogue.filter(p => p.id == id).head;
  }
}
```

Per la restituzione, l'oggetto con id corrispondente viene cercato nella propria lista di lettura e restituito se possibile.

```
def returnItem(id : Int):{
  if(!myItems.filter(i => i.id ==id).isEmpty){
    myItems.filter(i => i.id ==id).head.returnIt(this)
    myItems = myItems.filterNot(i=> i.id == id)
  }
}
```

toString

È stato implementato un sistema per stampare i propri libri con una notazione compatta tramite foreach.

```
def printMyList(){
  println("-"*40+"\nreading list of subscriber "+this.name+"\n"+"-"*40)
  myItems.foreach(i => println(">"+i.id+"_"*(4-i.id.toString.length())+i.title))
}
```

Libreria

La libreria contiene il catalogo degli oggetti, la lista degli abbonati ed è stata incaricata di tener traccia del tempo in questo esercizio.

```
object MyLibrary {
  import scala.LibItem
  import scala.Subscriber
  var date = 0;
  var catalogue:ListBuffer[LibItem] = new ListBuffer()
  var people:ListBuffer[Subscriber] = new ListBuffer()
  val administrator = admin
```

Inizializzazione tramite CSV

nell'inizializzazione viene creato il catalogo e aggiunto un amministratore.

Vengono inoltre aggiunti degli abbonati.

```
def init()={
  initCatalog()
  initSubscribers()
  people+= administrator
}
```

Il catalogo viene inizializzato da un file csv contenente nome e tipologia di oggetto (0-Book , 1-Ebook , 2-Media). Per ogni linea verrà chiamato un inizializzatore che creerà l'oggetto corrispondente aggiungendolo al

```
1 The Some Storms;0
2 Time of Wave;2
3 The Servant's Bridge;0
4 The Legend of the Words;1
5 Tears in the Spirits;1
6 Deep Husband;1
7 The Forgotten Weeping;2
8 Soaring of Girl;2
9 The Thought's Nobody;1
10 The Door of the Streams;0
11 Flight in the Bridge;1
12 Hot Wave;0
```

```
def initItem(s:String)={
  var res=s.split(";")
  res(1).toInt match{
  case 0 =>  catalogue+=new Book(res(0),catalogue.size)
  case 1 =>  catalogue+=new EBook(res(0),catalogue.size)
  case 2 =>  catalogue+=new Media(res(0),catalogue.size)
  }
}

def initCatalog()={
  val filepath = Paths.get("src\\catalogue.csv").toAbsolutePath
  val csvLines = Source.fromFile(filepath.toString).getLines.toList
  csvLines.foreach(initItem(_))
  //printCatalog()
}
```

Gli abbonati vengono aggiunti tramite una lista di nomi e randomicamente vengono generati golden e non membri.

```
def initSubscribers()={
  val filepath = Paths.get("src\\names.csv").toAbsolutePath
  val csvLines = Source.fromFile(filepath.toString).getLines.toList
  csvLines.foreach(n => addSubscriber(n, Random.nextBoolean()))
}
```

il metodo invocato consente di aggiungere un iscritto specificando la tipologia

```
def addSubscriber(name:String,golden:Boolean)={
  if(golden){
    val sb = new goldenMember(name)
    people+=sb
  }
  else{
    val sb = new Subscriber(name)
    people+=sb
  }

}
```

toSring

Il catalogo, la lista degli iscritti e la lista dei morosi possono essere stampati tramite le corrispondenti funzioni

```
def printCatalog(){
  catalogue.foreach(i => println(">"+i.id+"__"*(4-i.id.toString.length())+i.title+"__"*(35-(i.title.length()))+"status:"+(
}

def printSubs(){
  people.foreach(p => println(">"+p.name+"__"*(35-p.name.length())+"fee: "+p.fee )  )
}

def getOverdue(){
  people.filter(p => p.fee>0 ).foreach(p => println(">"+p.name+"__"*(35-p.name.length())+"fee: "+p.fee )  )
}
```

```
>0   The Some Storms               status:free  more:Book title
>1   Time of Wave                  status:free  more:Media titl
>2   The Servant's Bridge          status:free  more:Book title
>3   The Legend of the Words       status:free  more:EBook titl
>4   Tears in the Spirits          status:free  more:EBook titl
>5   Deep Husband                  status:free  more:EBook titl
>6   The Forgotten Weeping         status:free  more:Media titl
>7   Soaring of Girl               status:free  more:Media titl
>8   The Thought's Nobody          status:free  more:EBook titl
>9   The Door of the Streams       status:free  more:Book title
>10  Flight in the Bridge          status:free  more:EBook titl
>11  Hot Wave                      status:free  more:Book title
>12  The Luscious Thief            status:free  more:Book title
>13  Hustler of Search             status:free  more:Book title
>14  The Force's Ice               status:free  more:EBook titl
>15  The Body of the Boy           status:free  more:EBook titl
>16  Servant in the Year           status:free  more:Media titl
>17  Ragged Spirit                 status:free  more:Book title
>18  The Splintered Mage           status:free  more:EBook titl
>19  Dreaming of Servant           status:free  more:Book title
>20  The Emperor's Ship            status:free  more:EBook titl
>21  The Something of the Petals   status:free  more:Book title
>22  Slaves in the Voyager         status:free  more:EBook titl
>23  Next Spirits                  status:free  more:Media titl
>24  The Purple Healing            status:free  more:Media titl
>25  Woman of Soaring              status:free  more:Book title
>26  The Streams's Child           status:free  more:Book title
>27  The Nothing of the Words      status:free  more:EBook titl
>28  Visions in the Shard          status:free  more:Book title
```

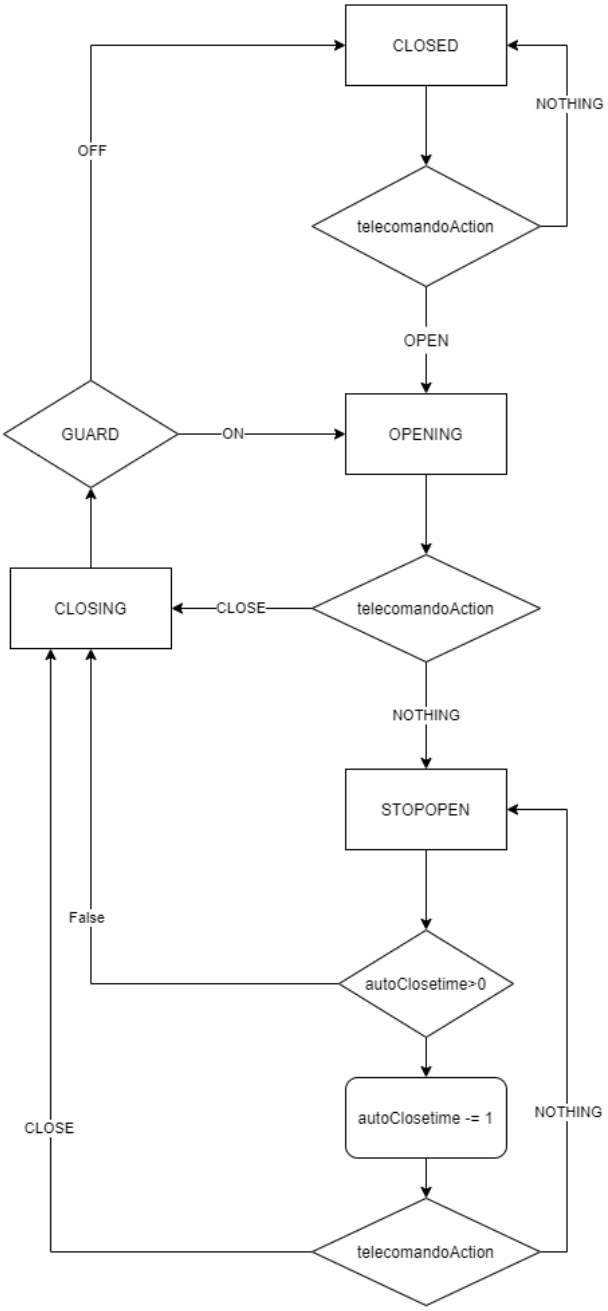
```
>Liam                              fee: 0
>Noah                              fee: 0
>Oliver                            fee: 0
>Elijah                            fee: 0
>William                           fee: 0
>James                             fee: 0
>Benjamin                          fee: 0
>Lucas                             fee: 0
>Henry                             fee: 0
>Alexander                         fee: 0
>Olivia                            fee: 0
>Emma                              fee: 0
>Ava                               fee: 0
>Charlotte                         fee: 0
>Sophia                            fee: 0
>Amelia                            fee: 0
>Isabella                          fee: 0
>Mia                               fee: 0
>Evelyn                            fee: 0
>Harper                            fee: 0
>ADMIN                             fee: 0
```

Progetto ASM (Cancello automatico)

È stato implementato un cancello automatico controllato tramite telecomando.

Il telecomando è in grado di aprire e chiudere il cancello e la chiusura può interrompere l'apertura.

È presente, inoltre, una banda di sicurezza che se attivata durante la chiusura forza la riapertura immediata.



Domini e funzioni

```
signature:

    //DOMINI
    enum domain GateStatus = {OPENING | STOPOPEN |CLOSING | STOPCLOSED}
    enum domain SecurityGuard = {ON | OFF}
    enum domain Pulsanti = {OPEN | CLOSE | NOTHING }

    //CONTROLLED
    dynamic controlled gate : GateStatus
    dynamic controlled autoClosetime : Integer
    dynamic controlled state : String

    //MONITORED
    dynamic monitored telecomandoAction: Pulsanti
    dynamic monitored guard : SecurityGuard

    static closeTime : Integer
```

vengono definiti gli stati del cancello ovvero:

- STOPOPEN : il cancello è aperto e può essere richiuso manualmente o automaticamente grazie a un timer
- STOPCLOSED: il cancello è chiuso e può solo essere riaperto tramite telecomando
- OPENING: il cancello si sta aprendo e può essere richiuso tramite comando
- CLOSING: il cancello si sta chiudendo e può essere riaperto tramite apertura di emergenza o comando

vengono definiti gli stati della banda di guardia:

- ON : la banda viene premuta e in caso di fase di chiusura attiva l'immediata riapertura
- OFF: il sistema procede normalmente

si definisce anche un tempo di chiusura *closeTime* per determinare quanto tempo passi nella fase di richiusura automatica.

Apertura e chiusura

Le due regole base determinano l'apertura e la chiusura del cancello

```
rule r_close =
    if (gate=OPENING or gate=STOPOPEN) then
        seq
        gate:=CLOSING
        state:="closing gate"
        endseq
    endif
rule r_open =
    if (gate=CLOSING or gate=STOPCLOSED) then
        seq
            gate:=OPENING
            state:="opening gate"
        endseq
    endif
```

Il cancello puo essere chiuso se è aperto o se si sta aprendo mentre puo esser aperto se è chiuso o si sta chiudendo.

Apertura di emergenza

Nel caso in cui la guardia sia attivata e il cancello sia in fase di chiusura questo deve essere riaperto immediatamente.

```
rule r_emergencyOpen =
    if(gate=CLOSING and guard=ON) then
        seq
        gate := OPENING
        state:="emergency open!"
        endseq
    endif
```

Stop rule

La seguente regola gestisce la transizione di finecorsa del cancello e il timer automatico per la chiusura.

```
rule r_stop =
    seq
        if(gate=OPENING) then
            seq
            gate:= STOPOPEN
            autoClosetime := closeTime
            state:="Gate stopped open"
            endseq
        endif
        if (gate=CLOSING and guard=OFF) then
            seq
            gate:= STOPCLOSED
            state:="Gate stopped closed"
            endseq
        endif
        if (gate=STOPOPEN and autoClosetime=0) then r_close[]
        endif
    endseq
```

Se il cancello si sta aprendo lo stadio successivo sarà il bloccaggio in apertura, se si sta chiudendo sarà quello in chiusura.

A seguito di una apertura il timer del cancello viene impostato a *closeTime* per essere utilizzato dal timer.

Nel caso in cui il cancello sia aperto e il timer raggiunga lo zero il cancello verrà chiuso in automatico come descritto nella linea:

```
if (gate=STOPOPEN and autoClosetime=0) then r_close[]
```

Autoclose

La regola di autoclose è stata definita tramite due regole:

- r_countdown incaricata solamente di decrementare *autoClosetime* qualora fosse un numero maggiore di zero
- r_autoClose incaricata di verificare lo stato del cancello e di attivare il countdown solo se questo è fermo e aperto.

Main rule

La main rule si occupa di gestire gli input forniti dal telecomando mediante il segnale inviato dai pulsanti.

Se il telecomando invia il segnale di OPEN solo la regola r_open verrà invocata.

Se il telecomando invia il segnale CLOSE verranno eseguite la regola di close in parallelo a quella di emergency open in caso di problemi durante la chiusura.

Se il telecomando non invierà segnali il sistema resterà chiuso o continuerà la sua esecuzione in caso di fasi di apertura/chiusura/timer. Le regole chiamate in sequenza sono:

- autoclose: se il cancello fosse stato aperto per abbastanza tempo si dovrebbe richiudere
- stop: se il cancello fosse in movimento
- emergencyOpen: se a seguito di una chiusura dettata dalle regole precedenti si dovesse verificare un problema

Inizializzazione

Il sistema parte da chiuso con il sistema di guardia non attivato.

```
default init s0:
    function guard = OFF
    function gate=STOPCLOSED
```

Avalla

La simulazione viene effettuata con il seguente scenario:

- 0: inviato il comando di apertura
- 1: si attende la fine dell'apertura senza inviare segnali
- 2: il cancello è fermo aperto
- 2/3/4: terminato il countdown il cancello comincia la chiusura automatica
- 5: la banda di sicurezza viene attivata e il cancello viene riaperto
- 6: inviato il comando close al cancello aperto
- 7/8: il cancello si chiude completamente

Di seguito il risultato della simulazione manuale

	Type	Functions	State 0	State 1	State 2	State 3	State 4	State 5	State 6	State 7	State 8
☐	M	telecomandoAction	OPEN	NOTHING	NOTHING	NOTHING	NOTHING	NOTHING	CLOSE	NOTHING	
☐	C	gate	STOPCLOSED	OPENING	STOPOPEN	STOPOPEN	CLOSING	OPENING	STOPOPEN	CLOSING	STOPCLOSED
☐	C	state		opening gate	Gate stopped open	counting down	closing gate	emergency open!	Gate stopped open	closing gate	Gate stopped closed
☐	M	guard		OFF	OFF	OFF	ON	OFF	OFF	OFF	
☐	C	autoClosetime			2	1	0	0	2	2	2