KU Leuven

Departement Computerwetenschappen

H0S01a , Comparative Porgramming Languages, prof. F. Piessens

Comparative Programming Languages

Erlang assignment: 2048

Master in de Ingenieurswetenschappen

Computerwetenschappen

Nele Rober, r0262954

Academiejaar 2014-2015

1 Tile Behaviour

The game of 2048 is represented by 16 tiles. At each moment in time, a tile has an id, a value and a boolean that says whether the tile has merged that turn or not.

A tile can be ordered to move (up, down, left or right). The tile will then do following steps:

- check whether it is possible to move in that direction;
- check whether it is possible to merge with some next tile;
- perform the right action;
- propagate the order to some next tile.

2 Failure

To represent good tile behaviour, tiles have to contact each other and ask for information. Tiles also have to be contacted by the manager so that it can learn their values or that it can order them to move. Communication with the tiles is thus crucial for this implementation of the game.

If a tile dies, it cannot communicate any longer and the game will crash.

The `Blaster` function kills tiles once in a while. To make the game work dispite of this `Blaster`, two solutions have been implemented: one that creates new empty tiles and one that duplicates the tile that just died.

2.1 Creating new empty tiles

Originally, communication with the tiles happend directly. However, this kind of communication can go wrong if the tile is no longer there.

Therefore, a method was created inside `Manager` to account for the communication with tiles (`sendmessage(N,Mess)`). This method will send a given message to the given tile. If this tile is unreachable because it has died, an error is thrown. When this happens, a new tile with a value of zero is created and the message is sent again to this new tile.

When all communication with the tiles goes via this method, the game will no longer crash when tiles are killed by the `Blaster`. However, tiles will randomly seem to disappear to the player as they are recreated with a value of zero.

2.2 Duplicating existing tiles

A better solution is to make the tiles notify the `Manager` right before they die:

```
die ->
    debug:debug("I, ~p, die.~n",[Id]),
    manager ! {tileDies, Id,Value,Merged},
    exit(killed);
```

When the `Manager` receives this message, he will instantely create a new tile with the same value and merge state:

```
{tileDies, Id,Value,Merged} ->
  registerTile(Id,Value,Merged)
```

This solution makes sure that tiles are always replaced by clones of the dying tiles and that the game can proceed without tiles dissapearing.

3 Tile Concurrency

When the `Manager` has given the tiles the order to move, it takes some time before the order is propagated through the field. It may happen that the `GUI` requires information about the new values of the tiles in that waiting period. The `Manager` should therefore know when it is safe to anwser the request of the `GUI`.

This is done by letting the tiles send a message to the `Manager` when propagation is over (when the next tile they should give the order to, lies outside of the playing field).

```
propagate(Dir,Id,Value,Merged) ->
  PropInBounds = inBounds(nextTileToPropagate(Dir,Id)),
  if
    PropInBounds -> manager:sendmessage(nextTileToPropagate(Dir,Id),
        Dir);
    not PropInBounds -> manager ! endOfPropagation
  end,
  tilelife(Id,Value,Merged).
```

The `Manager` should receive four of these propagation messages before he can be sure that propagation has truly ended. Therefore, the `Manager` starts a propagation loop when he sends an order to the tiles:

```
propagationloop(Num, AskedToSend) ->
  receive
    endOfPropagation ->
      case Num of
        3 ->
          if
            AskedToSend -> manager ! sendData, manageloop();
            not AskedToSend -> manageloop()
          end;
        _ ->
          propagationloop(Num+1,AskedToSend)
      end;
    {tileDies, Id,Value,Merged} ->
      registerTile(Id,Value,Merged),
      propagationloop(Num, AskedToSend);
    sendData ->
      propagationloop(Num, true)
  end.
```

This loop keeps track of the number of propagation messages the `Manager` has received so far (Num). When this number reaches four, the loop is ended and the `Manager` returns to its normal manage loop.

However, two other kind of messages should not be ignored when the `Manager` is in this loop: the message that a tile has died - because this is important and should be fixed immediately to avoid strange behaviour of the game - and the message that the `GUI` requires information about the tile - because otherwise the display will not get updated. In the case of a request from the `GUI`, the `Manager` remembers that the `GUI` has asked for this (AskedToSend) and answers this request as soon as propagation has fully ended. This way, the `GUI` will not receive incorrect information.

It may also happen that this `GUI` request comes in when propagation has already finished, in that case there is no problem and the `Manager` can answer the request immediately.

Total workload for this project: 15u00.