# Seminar Report: Paxy

**GOKTUG CENGIZ**
**ANAIS CAROLINA MARTINEZ MANZANILLA**
**JONNATAN MENDOZA ESCOBAR**

May 20, 2019

## Introduction

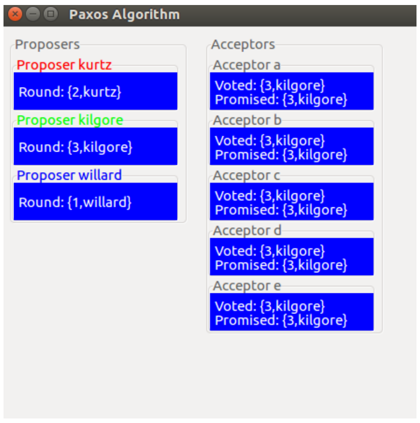
*This seminars purpose is to explore and explain the Paxos algorithm, which is useful to have consensus in a distributed system. Paxos is a fault tolerant method to solve the consensus problem where participant have to agree on a single chosen value.*

*The algorithm has three roles for processes: proposers, acceptors and learners; all three role can be contained in a single process. Learner is not necessary to reach a consensus, hence it is not implemented for this seminar.*

*Main objective for this seminar report is to investigate the mechanism of having a consensus in a distributed fashion with faulty processes.*

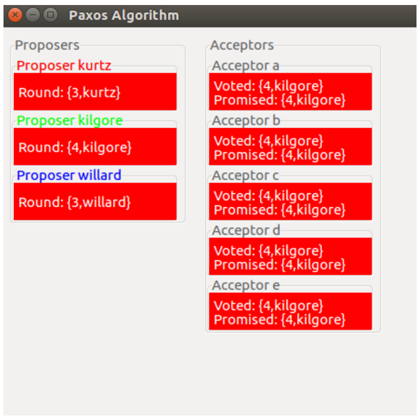
## Experiments & Open questions

**Experiment 1:** Try introducing different delays in the acceptor (for instance, just after receiving prepare and/or accept messages).



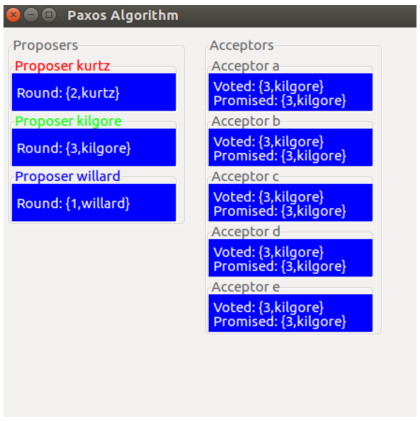
```
Paxos Algorithm
Proposers                 Acceptors
Proposer kurtz            Acceptor a
Round: {2,kurtz}          Voted: {3,kilgore}
                          Promised: {3,kilgore}
Proposer kilgore          Acceptor b
Round: {3,kilgore}        Voted: {3,kilgore}
                          Promised: {3,kilgore}
Proposer willard          Acceptor c
Round: {1,willard}        Voted: {3,kilgore}
                          Promised: {3,kilgore}
                          Acceptor d
                          Voted: {3,kilgore}
                          Promised: {3,kilgore}
                          Acceptor e
                          Voted: {3,kilgore}
                          Promised: {3,kilgore}
```

(a) Test: paxy:start([1000,1000,1000]).



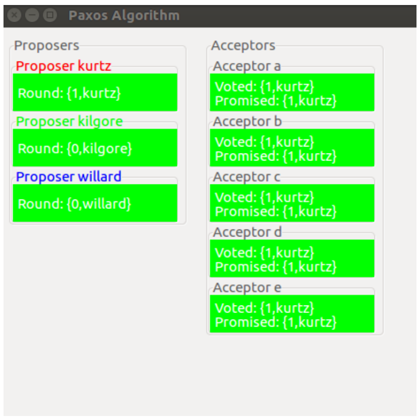
```
Paxos Algorithm
Proposers                 Acceptors
Proposer kurtz            Acceptor a
Round: {3,kurtz}          Voted: {4,kilgore}
                          Promised: {4,kilgore}
Proposer kilgore          Acceptor b
Round: {4,kilgore}        Voted: {4,kilgore}
                          Promised: {4,kilgore}
Proposer willard          Acceptor c
Round: {3,willard}        Voted: {4,kilgore}
                          Promised: {4,kilgore}
                          Acceptor d
                          Voted: {4,kilgore}
                          Promised: {4,kilgore}
                          Acceptor e
                          Voted: {4,kilgore}
                          Promised: {4,kilgore}
```

(b) Test: paxy:start([30,200,820]).



```
Paxos Algorithm
Proposers                 Acceptors
Proposer kurtz            Acceptor a
Round: {2,kurtz}          Voted: {3,kilgore}
                          Promised: {3,kilgore}
Proposer kilgore          Acceptor b
Round: {3,kilgore}        Voted: {3,kilgore}
                          Promised: {3,kilgore}
Proposer willard          Acceptor c
Round: {1,willard}        Voted: {3,kilgore}
                          Promised: {3,kilgore}
                          Acceptor d
                          Voted: {3,kilgore}
                          Promised: {3,kilgore}
                          Acceptor e
                          Voted: {3,kilgore}
                          Promised: {3,kilgore}
```

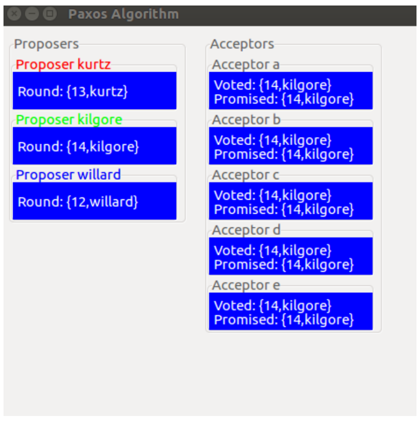
(c) Test: paxy:start([45,40,80]).



```
Paxos Algorithm
Proposers                 Acceptors
Proposer kurtz            Acceptor a
Round: {1,kurtz}          Voted: {1,kurtz}
                          Promised: {1,kurtz}
Proposer kilgore          Acceptor b
Round: {0,kilgore}        Voted: {1,kurtz}
                          Promised: {1,kurtz}
Proposer willard          Acceptor c
Round: {0,willard}        Voted: {1,kurtz}
                          Promised: {1,kurtz}
                          Acceptor d
                          Voted: {1,kurtz}
                          Promised: {1,kurtz}
                          Acceptor e
                          Voted: {1,kurtz}
                          Promised: {1,kurtz}
```

(d) Test: paxy:start([2250,1200,2500]).

Figure 1: Uniform Random Delay at 2000ms.



Proposers

Proposer kurtz

Round: {13,kurtz}

Proposer kilgore

Round: {14,kilgore}

Proposer willard

Round: {12,willard}

Acceptors

Acceptor a

Voted: {14,kilgore}
Promised: {14,kilgore}

Acceptor b

Voted: {14,kilgore}
Promised: {14,kilgore}

Acceptor c

Voted: {14,kilgore}
Promised: {14,kilgore}

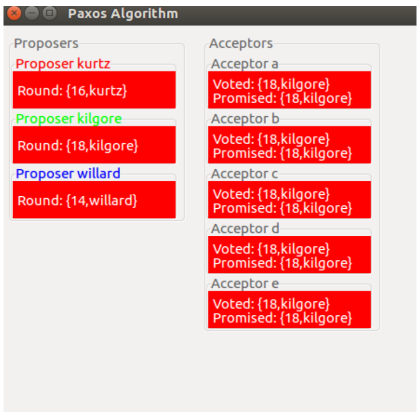
Acceptor d

Voted: {14,kilgore}
Promised: {14,kilgore}

Acceptor e

Voted: {14,kilgore}
Promised: {14,kilgore}

(a) Test: paxy:start([1000,1000,1000]).



Proposers

Proposer kurtz

Round: {16,kurtz}

Proposer kilgore

Round: {18,kilgore}

Proposer willard

Round: {14,willard}

Acceptors

Acceptor a

Voted: {18,kilgore}
Promised: {18,kilgore}

Acceptor b

Voted: {18,kilgore}
Promised: {18,kilgore}

Acceptor c

Voted: {18,kilgore}
Promised: {18,kilgore}

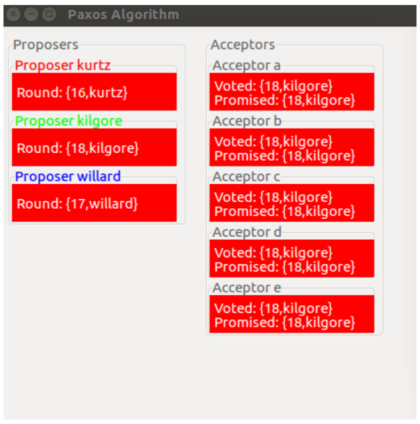
Acceptor d

Voted: {18,kilgore}
Promised: {18,kilgore}

Acceptor e

Voted: {18,kilgore}
Promised: {18,kilgore}

(b) Test: paxy:start([30,200,820]).



Proposers

Proposer kurtz

Round: {16,kurtz}

Proposer kilgore

Round: {18,kilgore}

Proposer willard

Round: {17,willard}

Acceptors

Acceptor a

Voted: {18,kilgore}
Promised: {18,kilgore}

Acceptor b

Voted: {18,kilgore}
Promised: {18,kilgore}

Acceptor c

Voted: {18,kilgore}
Promised: {18,kilgore}

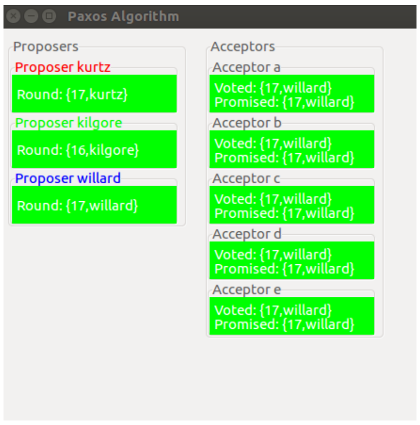
Acceptor d

Voted: {18,kilgore}
Promised: {18,kilgore}

Acceptor e

Voted: {18,kilgore}
Promised: {18,kilgore}

(c) Test: paxy:start([45,40,80]).



Proposers

Proposer kurtz

Round: {17,kurtz}

Proposer kilgore

Round: {16,kilgore}

Proposer willard

Round: {17,willard}

Acceptors

Acceptor a

Voted: {17,willard}
Promised: {17,willard}

Acceptor b

Voted: {17,willard}
Promised: {17,willard}

Acceptor c

Voted: {17,willard}
Promised: {17,willard}

Acceptor d

Voted: {17,willard}
Promised: {17,willard}

Acceptor e

Voted: {17,willard}
Promised: {17,willard}

(d) Test: paxy:start([2250,1200,2500]).

Figure 2: Uniform Random Delay at 4000ms.

## Questions

- Does the algorithm still terminate?

  Yes, it still terminate, but there are some interesting things to discuss, if the gap on the delay time between the timeout and the acceptor is to big, the convergence of the algorithm relays on 2 main things, 1)the probality in long term tends to the average which means that in many rounds the random delay of the acceptor will be short enogh to make possible the correct reception of a proposer that can complete his round, 2) The backoff time doubles every round which make the proposer sleeps more time, this is helpful when the delay in the reception is to big because at some point will be enogh time to the acceptor can recive all the messages. In a real aproach maybe is better to consider that in a very big gap of timeout and response time, the system will not end, but in a teorical point of view, in very long term you could always thing that is posible the algorithm will achive agremment.

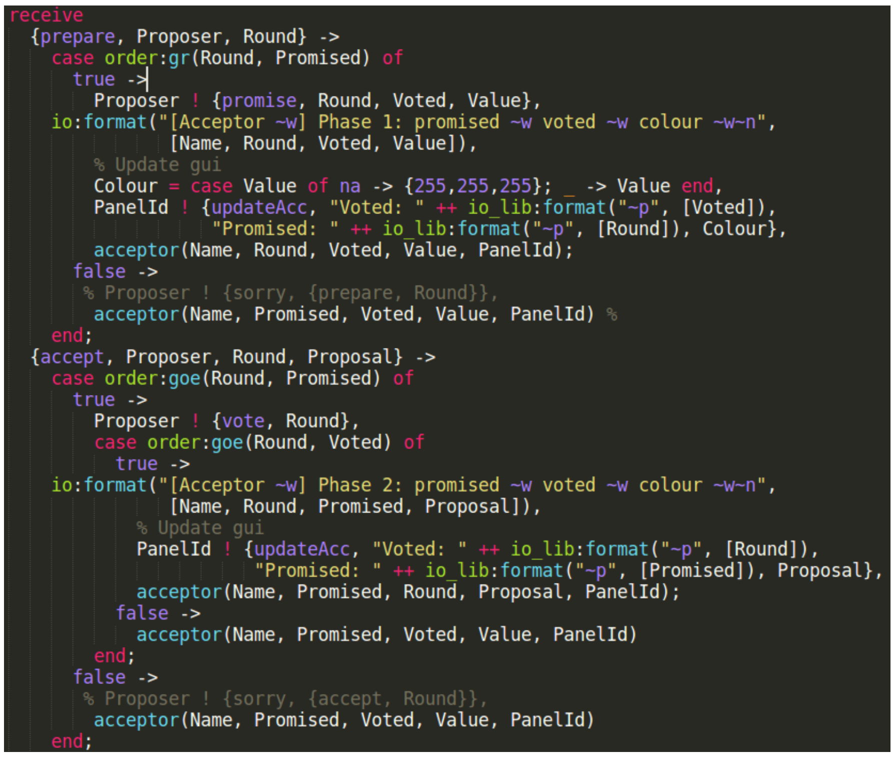
- Does it require more rounds?

In general yes but not necesarily, the behavior is also strong related with the delay time of proposers when they are initialize and also with the timeout of the round if the delay in the aceptor is above the time out achive the consensus definitely will take more rounds.

- How does the impact of adding delays depend on the value of the timeout at the proposer?

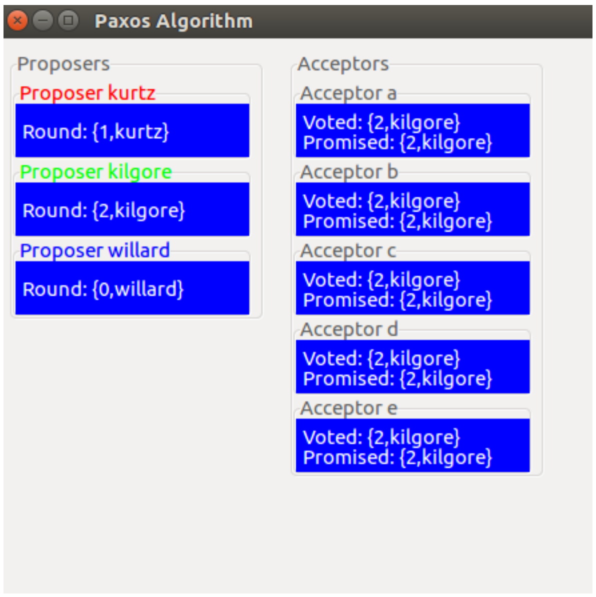
  As we comment in the last questions the timeout is strong related because if the delay in the reception of the acceptors is above the timeout this cause the algorithm take more rounds to finish, one future work idea to try to do the convergence more quick, is try to control the backoff time and the timeout of the proporser dynamicly, by measuring the reponse time of the acceptor and with this infomation change the backoff time and the timeout, now the algorithm always do a 2x growing in the backoff time and the timeout is constant.

**Experiment 2:** Avoid sending sorry messages by commenting the corresponding sen-tences in the acceptor.



```
receive
  {prepare, Proposer, Round} ->
    case order:gr(Round, Promised) of
      true ->
        Proposer ! {promise, Round, Voted, Value},
    io:format("[Acceptor ~w] Phase 1: promised ~w voted ~w colour ~w~n",
               [Name, Round, Voted, Value]),
        % Update gui
        Colour = case Value of na -> {255,255,255}; _ -> Value end,
        PanelId ! {updateAcc, "Voted: " ++ io_lib:format("~p", [Voted]),
                   "Promised: " ++ io_lib:format("~p", [Round]), Colour},
        acceptor(Name, Round, Voted, Value, PanelId);
      false ->
       % Proposer ! {sorry, {prepare, Round}},
        acceptor(Name, Promised, Voted, Value, PanelId) %
    end;
  {accept, Proposer, Round, Proposal} ->
    case order:goe(Round, Promised) of
      true ->
        Proposer ! {vote, Round},
        case order:goe(Round, Voted) of
          true ->
    io:format("[Acceptor ~w] Phase 2: promised ~w voted ~w colour ~w~n",
               [Name, Round, Promised, Proposal]),
          % Update gui
          PanelId ! {updateAcc, "Voted: " ++ io_lib:format("~p", [Round]),
                     "Promised: " ++ io_lib:format("~p", [Promised]), Proposal},
          acceptor(Name, Promised, Round, Proposal, PanelId);
        false ->
          acceptor(Name, Promised, Voted, Value, PanelId)
        end;
      false ->
       % Proposer ! {sorry, {accept, Round}},
        acceptor(Name, Promised, Voted, Value, PanelId)
    end;
```

Figure 3: Code modifications.



Proposer kurtz
Round: {1,kurtz}

Proposer kilgore
Round: {2,kilgore}

Proposer willard
Round: {0,willard}

Acceptor a
Voted: {2,kilgore}
Promised: {2,kilgore}

Acceptor b
Voted: {2,kilgore}
Promised: {2,kilgore}

Acceptor c
Voted: {2,kilgore}
Promised: {2,kilgore}

Acceptor d
Voted: {2,kilgore}
Promised: {2,kilgore}

Acceptor e
Voted: {2,kilgore}
Promised: {2,kilgore}

Figure 4: Test: paxy:start([1000,1000,1000]).

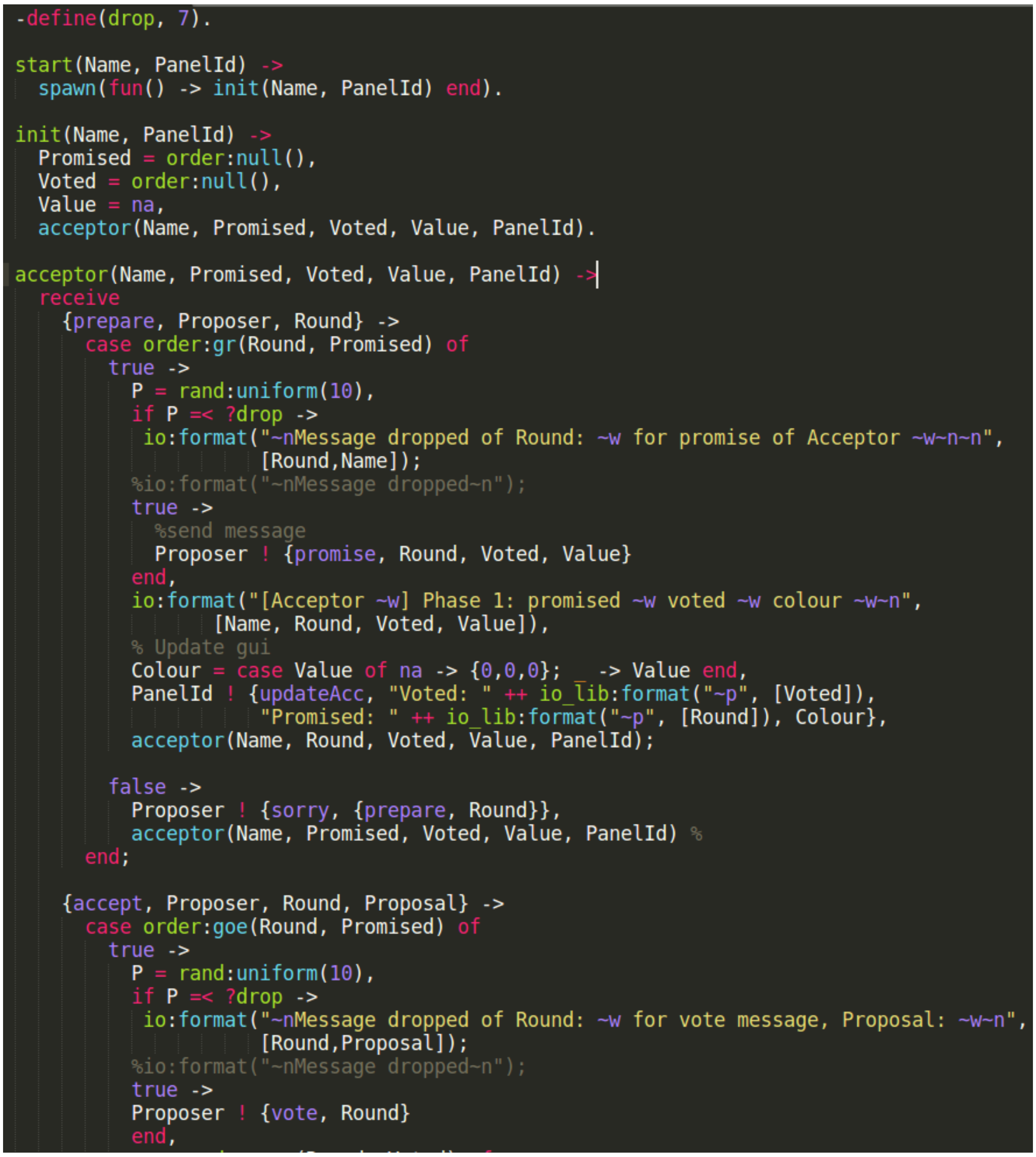
**Questions**

- Could you even come to an agreement when sorry messages are not sent?

  Yes, because the system is synchronus and the missing of sorry messages does not afect the convergence of the algorithm, the proposer is specting to recive promise or vote message, if not recives enogh to have a majority simply will abort so, in this implementation sending sorry message is precindable.

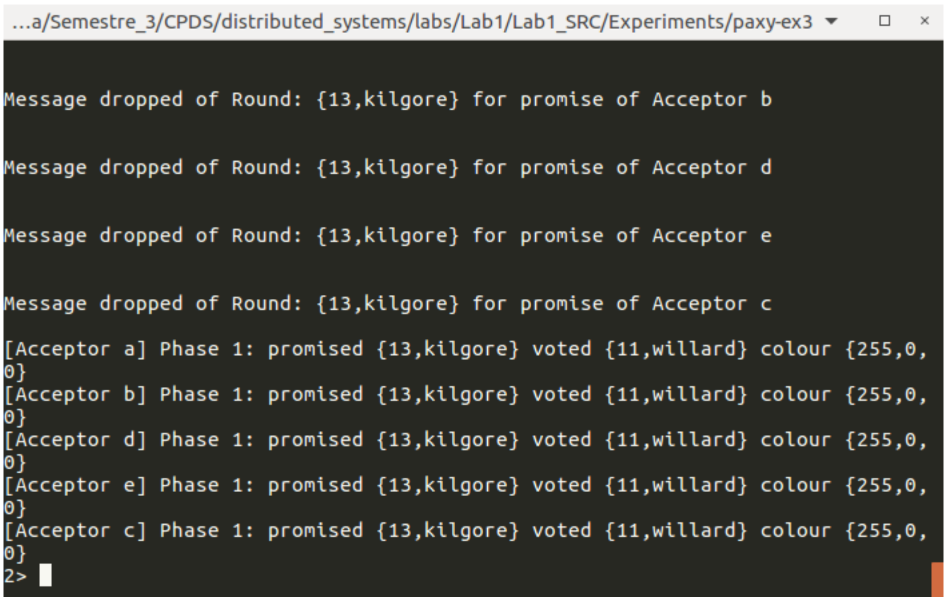
**Experiment 3:** Try randomly dropping promise and/or vote messages in the acceptor.

We use the drop code provided for interrup the vote message and the promise message.



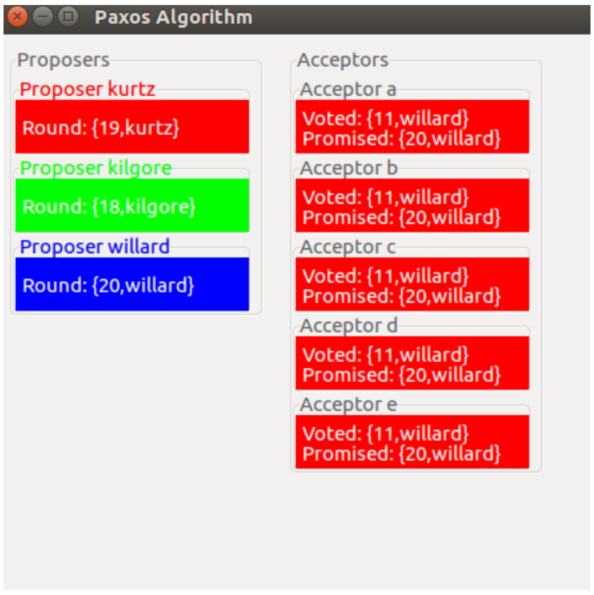
```
-define(drop, 7).

start(Name, PanelId) ->
  spawn(fun() -> init(Name, PanelId) end).

init(Name, PanelId) ->
  Promised = order:null(),
  Voted = order:null(),
  Value = na,
  acceptor(Name, Promised, Voted, Value, PanelId).

acceptor(Name, Promised, Voted, Value, PanelId) ->
  receive
    {prepare, Proposer, Round} ->
      case order:gr(Round, Promised) of
        true ->
          P = rand:uniform(10),
          if P =< ?drop ->
           io:format("~nMessage dropped of Round: ~w for promise of Acceptor ~w~n~n",
                     [Round,Name]);
          %io:format("~nMessage dropped~n");
          true ->
            %send message
            Proposer ! {promise, Round, Voted, Value}
          end,
          io:format("[Acceptor ~w] Phase 1: promised ~w voted ~w colour ~w~n",
                 [Name, Round, Voted, Value]),
          % Update gui
          Colour = case Value of na -> {0,0,0}; _ -> Value end,
          PanelId ! {updateAcc, "Voted: " ++ io_lib:format("~p", [Voted]),
                     "Promised: " ++ io_lib:format("~p", [Round]), Colour},
          acceptor(Name, Round, Voted, Value, PanelId);

        false ->
          Proposer ! {sorry, {prepare, Round}},
          acceptor(Name, Promised, Voted, Value, PanelId) %
      end;

    {accept, Proposer, Round, Proposal} ->
      case order:goe(Round, Promised) of
        true ->
          P = rand:uniform(10),
          if P =< ?drop ->
           io:format("~nMessage dropped of Round: ~w for vote message, Proposal: ~w~n",
                     [Round,Proposal]);
          %io:format("~nMessage dropped~n");
          true ->
          Proposer ! {vote, Round}
          end,
```

Figure 5: Code modifications.



```
...a/Semestre_3/CPDS/distributed_systems/labs/Lab1/Lab1_SRC/Experiments/paxy-ex3

Message dropped of Round: {13,kilgore} for promise of Acceptor b

Message dropped of Round: {13,kilgore} for promise of Acceptor d

Message dropped of Round: {13,kilgore} for promise of Acceptor e

Message dropped of Round: {13,kilgore} for promise of Acceptor c
[Acceptor a] Phase 1: promised {13,kilgore} voted {11,willard} colour {255,0,
0}
[Acceptor b] Phase 1: promised {13,kilgore} voted {11,willard} colour {255,0,
0}
[Acceptor d] Phase 1: promised {13,kilgore} voted {11,willard} colour {255,0,
0}
[Acceptor e] Phase 1: promised {13,kilgore} voted {11,willard} colour {255,0,
0}
[Acceptor c] Phase 1: promised {13,kilgore} voted {11,willard} colour {255,0,
0}
2>
```

Figure 6: Consol Output.



```
Paxos Algorithm
Proposers
Proposer kurtz
Round: {19,kurtz}
Proposer kilgore
Round: {18,kilgore}
Proposer willard
Round: {20,willard}
Acceptors
Acceptor a
Voted: {11,willard}
Promised: {20,willard}
Acceptor b
Voted: {11,willard}
Promised: {20,willard}
Acceptor c
Voted: {11,willard}
Promised: {20,willard}
Acceptor d
Voted: {11,willard}
Promised: {20,willard}
Acceptor e
Voted: {11,willard}
Promised: {20,willard}
```

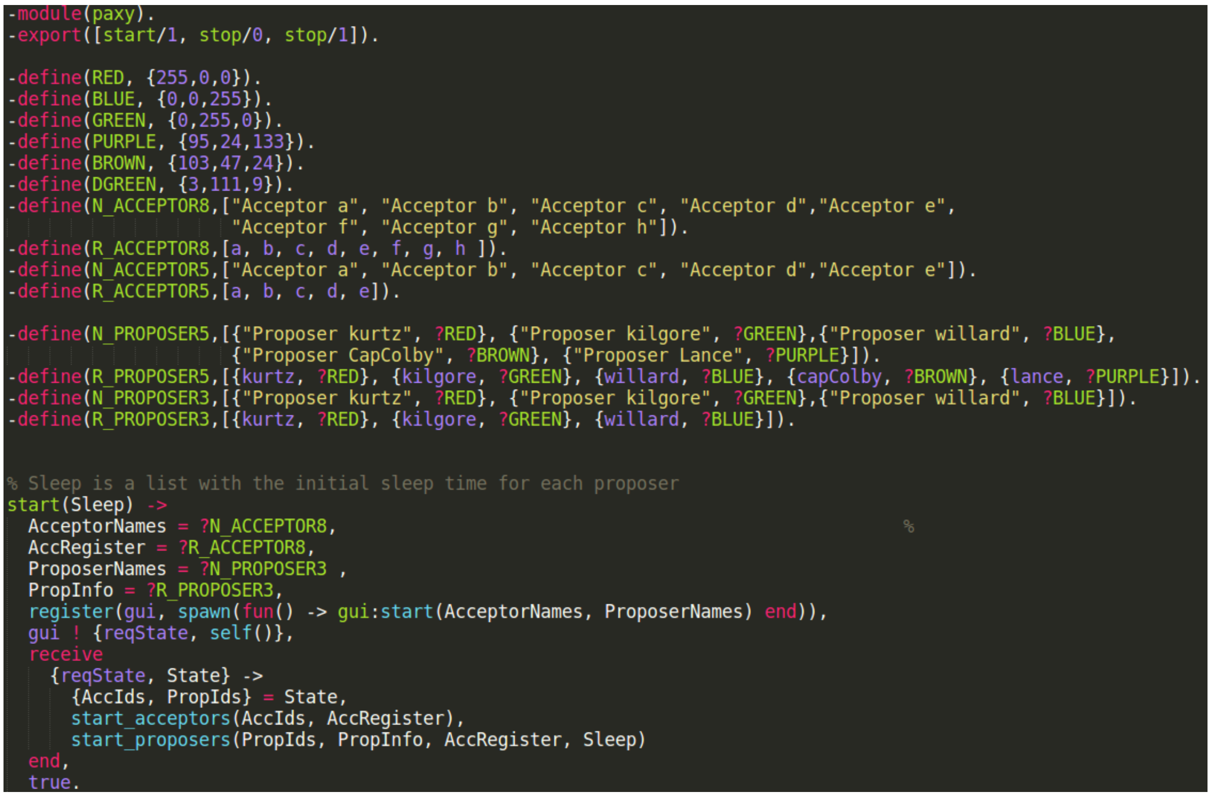
Figure 7: Test: paxy:start([1000,1000,1000]).

## Questions

- What percentage of messages can we drop until consensus is not longer possible?

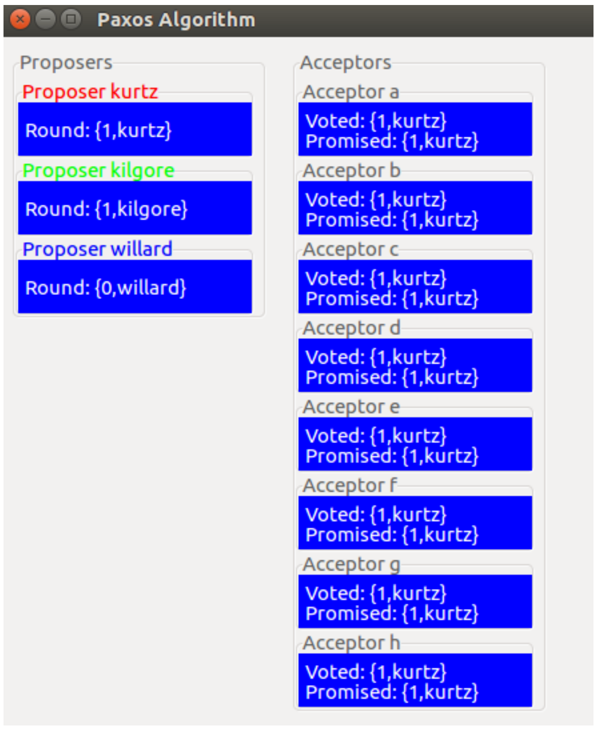
  Despite probality tends to average, we can not say that the percentage is precise, because the average will be visible in so many tests, nevertheless what we observe in the behavior is ok with the teory, when the drop constant in the code is greter or equal to 6 it seams that the algorithm have problems to end as is shown in the test picture.

**Experiment 4:** Try increasing the number of acceptors and proposers.



```erlang
-module(paxy).
-export([start/1, stop/0, stop/1]).

-define(RED, {255,0,0}).
-define(BLUE, {0,0,255}).
-define(GREEN, {0,255,0}).
-define(PURPLE, {95,24,133}).
-define(BROWN, {103,47,24}).
-define(DGREEN, {3,111,9}).
-define(N_ACCEPTOR8,["Acceptor a", "Acceptor b", "Acceptor c", "Acceptor d","Acceptor e",
                     "Acceptor f", "Acceptor g", "Acceptor h"]).
-define(R_ACCEPTOR8,[a, b, c, d, e, f, g, h ]).
-define(N_ACCEPTOR5,["Acceptor a", "Acceptor b", "Acceptor c", "Acceptor d","Acceptor e"]).
-define(R_ACCEPTOR5,[a, b, c, d, e]).

-define(N_PROPOSER5,[{"Proposer kurtz", ?RED}, {"Proposer kilgore", ?GREEN},{"Proposer willard", ?BLUE},
                     {"Proposer CapColby", ?BROWN}, {"Proposer Lance", ?PURPLE}]).
-define(R_PROPOSER5,[{kurtz, ?RED}, {kilgore, ?GREEN}, {willard, ?BLUE}, {capColby, ?BROWN}, {lance, ?PURPLE}]).
-define(N_PROPOSER3,[{"Proposer kurtz", ?RED}, {"Proposer kilgore", ?GREEN},{"Proposer willard", ?BLUE}]).
-define(R_PROPOSER3,[{kurtz, ?RED}, {kilgore, ?GREEN}, {willard, ?BLUE}]).


% Sleep is a list with the initial sleep time for each proposer
start(Sleep) ->
  AcceptorNames = ?N_ACCEPTOR8,                                                  %
  AccRegister = ?R_ACCEPTOR8,
  ProposerNames = ?N_PROPOSER3 ,
  PropInfo = ?R_PROPOSER3,
  register(gui, spawn(fun() -> gui:start(AcceptorNames, ProposerNames) end)),
  gui ! {reqState, self()},
  receive
    {reqState, State} ->
      {AccIds, PropIds} = State,
      start_acceptors(AccIds, AccRegister),
      start_proposers(PropIds, PropInfo, AccRegister, Sleep)
  end,
  true.
```

Figure 8: Code modifications.



Paxos Algorithm

Proposers

Proposer kurtz
Round: {1,kurtz}

Proposer kilgore
Round: {1,kilgore}

Proposer willard
Round: {0,willard}

Acceptors

Acceptor a
Voted: {1,kurtz}
Promised: {1,kurtz}

Acceptor b
Voted: {1,kurtz}
Promised: {1,kurtz}

Acceptor c
Voted: {1,kurtz}
Promised: {1,kurtz}

Acceptor d
Voted: {1,kurtz}
Promised: {1,kurtz}

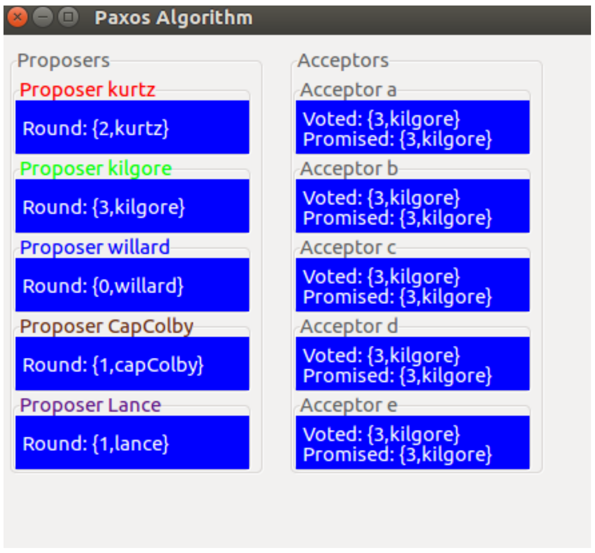
Acceptor e
Voted: {1,kurtz}
Promised: {1,kurtz}

Acceptor f
Voted: {1,kurtz}
Promised: {1,kurtz}

Acceptor g
Voted: {1,kurtz}
Promised: {1,kurtz}

Acceptor h
Voted: {1,kurtz}
Promised: {1,kurtz}

Figure 9: Test: paxy:start([1000,1000,1000]).



Paxos Algorithm

Proposers

Proposer kurtz
Round: {2,kurtz}

Proposer kilgore
Round: {3,kilgore}

Proposer willard
Round: {0,willard}

Proposer CapColby
Round: {1,capColby}

Proposer Lance
Round: {1,lance}

Acceptors

Acceptor a
Voted: {3,kilgore}
Promised: {3,kilgore}

Acceptor b
Voted: {3,kilgore}
Promised: {3,kilgore}

Acceptor c
Voted: {3,kilgore}
Promised: {3,kilgore}

Acceptor d
Voted: {3,kilgore}
Promised: {3,kilgore}

Acceptor e
Voted: {3,kilgore}
Promised: {3,kilgore}

Figure 10: Test: paxy:start([1000,1000,1000,1000,1000]).

### Questions

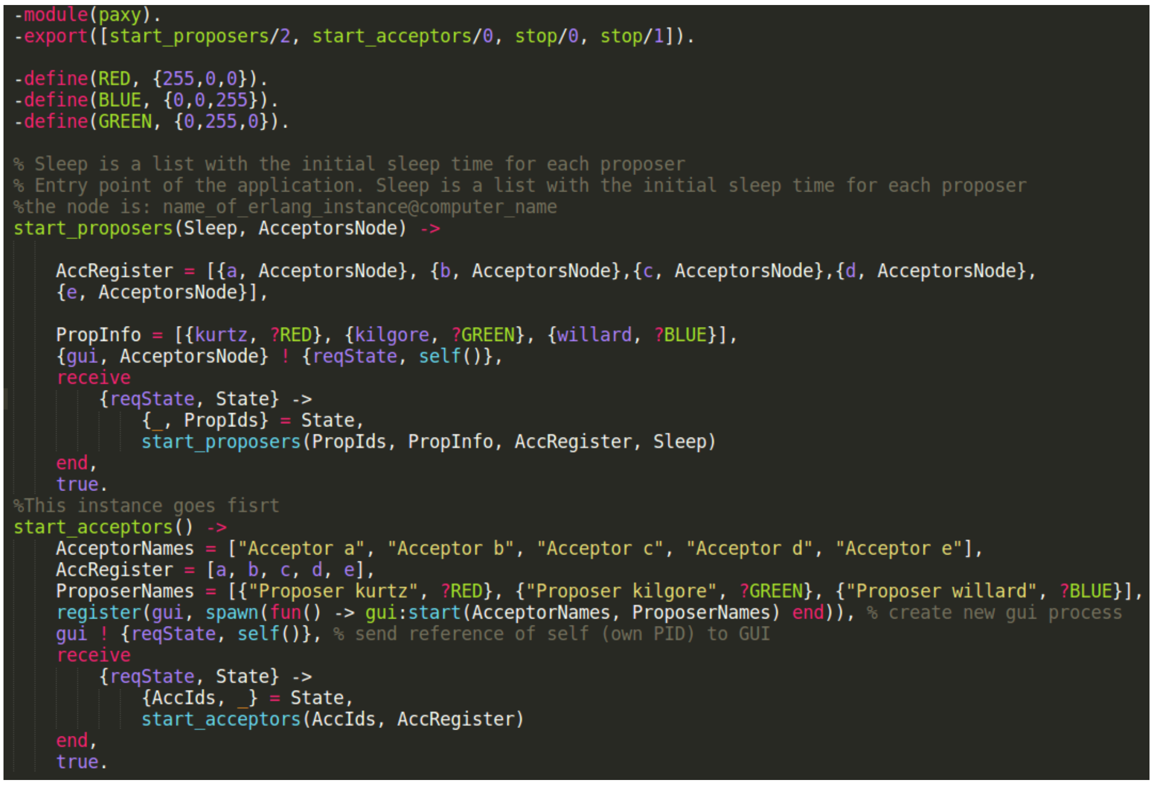
- What is the impact of having more acceptors while keeping the same number of proposers?

There is not a strong impact with more acceptor the algorithm finish in the same numer of rounds.

- What if we have more proposers while keeping the same number of acceptors?

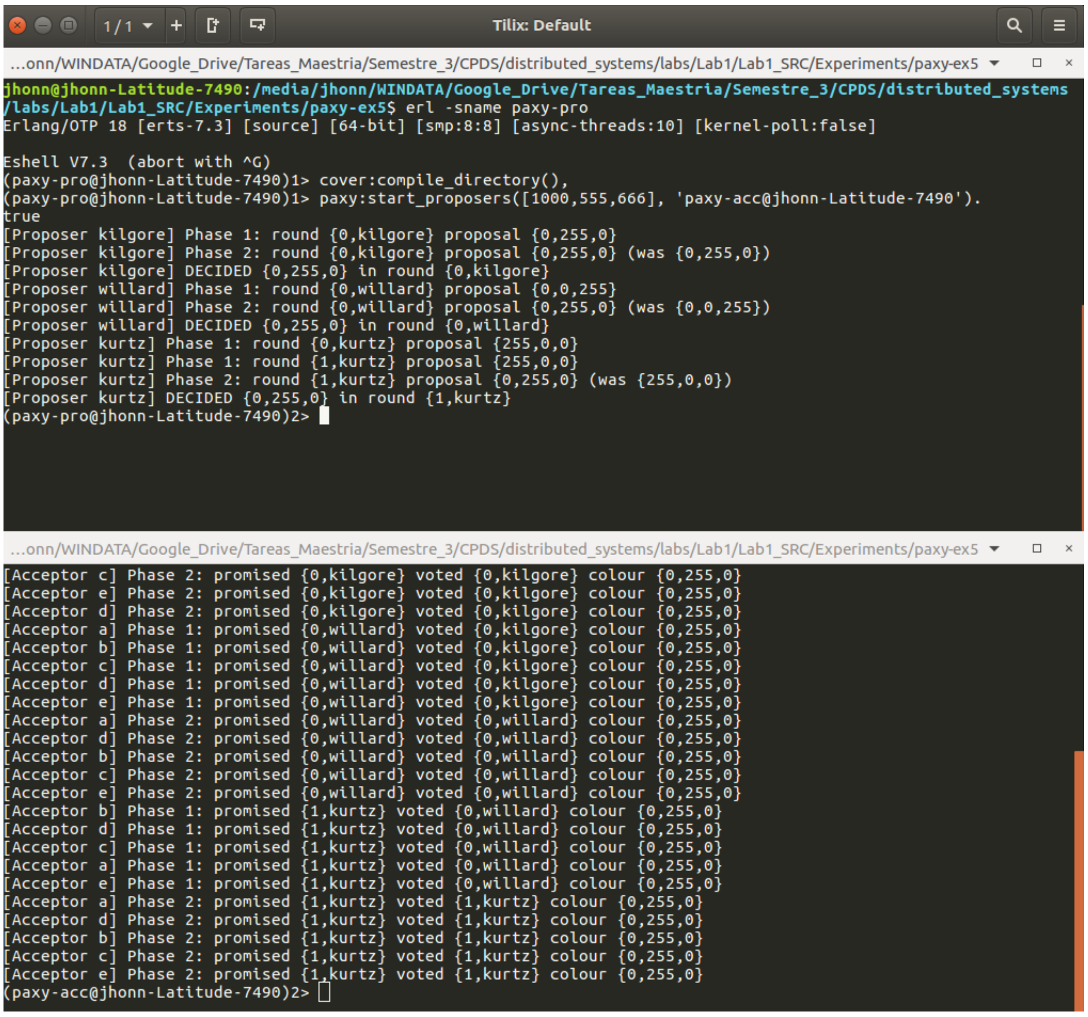
  When there are more proposers, takes more rounds to achieve consensus, this is because every of the proposer indirectly obstruct each other.

**Experiment 5:** Adapt the paxy module to create the proposers in a remote Erlang instance (named paxy-pro) and to ensure that they can connect correctly to the acceptors, which must be created in a different remote Erlang instance (named paxy-acc).



```erlang
-module(paxy).
-export([start_proposers/2, start_acceptors/0, stop/0, stop/1]).

-define(RED, {255,0,0}).
-define(BLUE, {0,0,255}).
-define(GREEN, {0,255,0}).

% Sleep is a list with the initial sleep time for each proposer
% Entry point of the application. Sleep is a list with the initial sleep time for each proposer
%the node is: name_of_erlang_instance@computer_name
start_proposers(Sleep, AcceptorsNode) ->

    AccRegister = [{a, AcceptorsNode}, {b, AcceptorsNode},{c, AcceptorsNode},{d, AcceptorsNode},
    {e, AcceptorsNode}],

    PropInfo = [{kurtz, ?RED}, {kilgore, ?GREEN}, {willard, ?BLUE}],
    {gui, AcceptorsNode} ! {reqState, self()},
    receive
        {reqState, State} ->
            {_, PropIds} = State,
            start_proposers(PropIds, PropInfo, AccRegister, Sleep)
    end,
    true.
%This instance goes fisrt
start_acceptors() ->
    AcceptorNames = ["Acceptor a", "Acceptor b", "Acceptor c", "Acceptor d", "Acceptor e"],
    AccRegister = [a, b, c, d, e],
    ProposerNames = [{"Proposer kurtz", ?RED}, {"Proposer kilgore", ?GREEN}, {"Proposer willard", ?BLUE}],
    register(gui, spawn(fun() -> gui:start(AcceptorNames, ProposerNames) end)), % create new gui process
    gui ! {reqState, self()}, % send reference of self (own PID) to GUI
    receive
        {reqState, State} ->
            {AccIds, _} = State,
            start_acceptors(AccIds, AccRegister)
    end,
    true.
```

Figure 11: Code modifications.

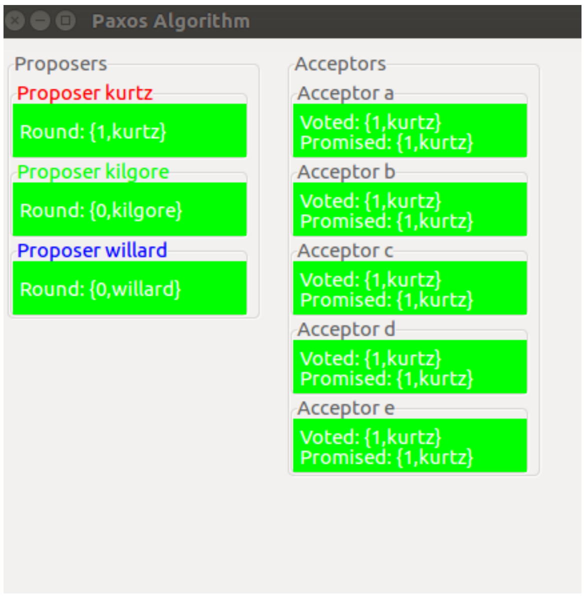

Tilix: Default
1 / 1
...onn/WINDATA/Google_Drive/Tareas_Maestria/Semestre_3/CPDS/distributed_systems/labs/Lab1/Lab1_SRC/Experiments/paxy-ex5

```
jhonn@jhonn-Latitude-7490:/media/jhonn/WINDATA/Google_Drive/Tareas_Maestria/Semestre_3/CPDS/distributed_systems
/labs/Lab1/Lab1_SRC/Experiments/paxy-ex5$ erl -sname paxy-pro
Erlang/OTP 18 [erts-7.3] [source] [64-bit] [smp:8:8] [async-threads:10] [kernel-poll:false]

Eshell V7.3  (abort with ^G)
(paxy-pro@jhonn-Latitude-7490)1> cover:compile_directory(),
(paxy-pro@jhonn-Latitude-7490)1> paxy:start_proposers([1000,555,666], 'paxy-acc@jhonn-Latitude-7490').
true
[Proposer kilgore] Phase 1: round {0,kilgore} proposal {0,255,0}
[Proposer kilgore] Phase 2: round {0,kilgore} proposal {0,255,0} (was {0,255,0})
[Proposer kilgore] DECIDED {0,255,0} in round {0,kilgore}
[Proposer willard] Phase 1: round {0,willard} proposal {0,0,255}
[Proposer willard] Phase 2: round {0,willard} proposal {0,255,0} (was {0,0,255})
[Proposer willard] DECIDED {0,255,0} in round {0,willard}
[Proposer kurtz] Phase 1: round {0,kurtz} proposal {255,0,0}
[Proposer kurtz] Phase 1: round {1,kurtz} proposal {255,0,0}
[Proposer kurtz] Phase 2: round {1,kurtz} proposal {0,255,0} (was {255,0,0})
[Proposer kurtz] DECIDED {0,255,0} in round {1,kurtz}
(paxy-pro@jhonn-Latitude-7490)2> 
```

...onn/WINDATA/Google_Drive/Tareas_Maestria/Semestre_3/CPDS/distributed_systems/labs/Lab1/Lab1_SRC/Experiments/paxy-ex5

```
[Acceptor c] Phase 2: promised {0,kilgore} voted {0,kilgore} colour {0,255,0}
[Acceptor e] Phase 2: promised {0,kilgore} voted {0,kilgore} colour {0,255,0}
[Acceptor d] Phase 2: promised {0,kilgore} voted {0,kilgore} colour {0,255,0}
[Acceptor a] Phase 1: promised {0,willard} voted {0,kilgore} colour {0,255,0}
[Acceptor b] Phase 1: promised {0,willard} voted {0,kilgore} colour {0,255,0}
[Acceptor c] Phase 1: promised {0,willard} voted {0,kilgore} colour {0,255,0}
[Acceptor d] Phase 1: promised {0,willard} voted {0,kilgore} colour {0,255,0}
[Acceptor e] Phase 1: promised {0,willard} voted {0,kilgore} colour {0,255,0}
[Acceptor a] Phase 2: promised {0,willard} voted {0,willard} colour {0,255,0}
[Acceptor d] Phase 2: promised {0,willard} voted {0,willard} colour {0,255,0}
[Acceptor b] Phase 2: promised {0,willard} voted {0,willard} colour {0,255,0}
[Acceptor c] Phase 2: promised {0,willard} voted {0,willard} colour {0,255,0}
[Acceptor e] Phase 2: promised {0,willard} voted {0,willard} colour {0,255,0}
[Acceptor b] Phase 1: promised {1,kurtz} voted {0,willard} colour {0,255,0}
[Acceptor d] Phase 1: promised {1,kurtz} voted {0,willard} colour {0,255,0}
[Acceptor c] Phase 1: promised {1,kurtz} voted {0,willard} colour {0,255,0}
[Acceptor a] Phase 1: promised {1,kurtz} voted {0,willard} colour {0,255,0}
[Acceptor e] Phase 1: promised {1,kurtz} voted {0,willard} colour {0,255,0}
[Acceptor a] Phase 2: promised {1,kurtz} voted {1,kurtz} colour {0,255,0}
[Acceptor d] Phase 2: promised {1,kurtz} voted {1,kurtz} colour {0,255,0}
[Acceptor b] Phase 2: promised {1,kurtz} voted {1,kurtz} colour {0,255,0}
[Acceptor c] Phase 2: promised {1,kurtz} voted {1,kurtz} colour {0,255,0}
[Acceptor e] Phase 2: promised {1,kurtz} voted {1,kurtz} colour {0,255,0}
(paxy-acc@jhonn-Latitude-7490)2> 
```

Figure 12: Both terminals working together.


Paxos Algorithm

Proposers

Proposer kurtz

Round: {1,kurtz}

Proposer kilgore

Round: {0,kilgore}

Proposer willard

Round: {0,willard}

Acceptors

Acceptor a

Voted: {1,kurtz}
Promised: {1,kurtz}

Acceptor b

Voted: {1,kurtz}
Promised: {1,kurtz}

Acceptor c

Voted: {1,kurtz}
Promised: {1,kurtz}

Acceptor d

Voted: {1,kurtz}
Promised: {1,kurtz}

Acceptor e

Voted: {1,kurtz}
Promised: {1,kurtz}

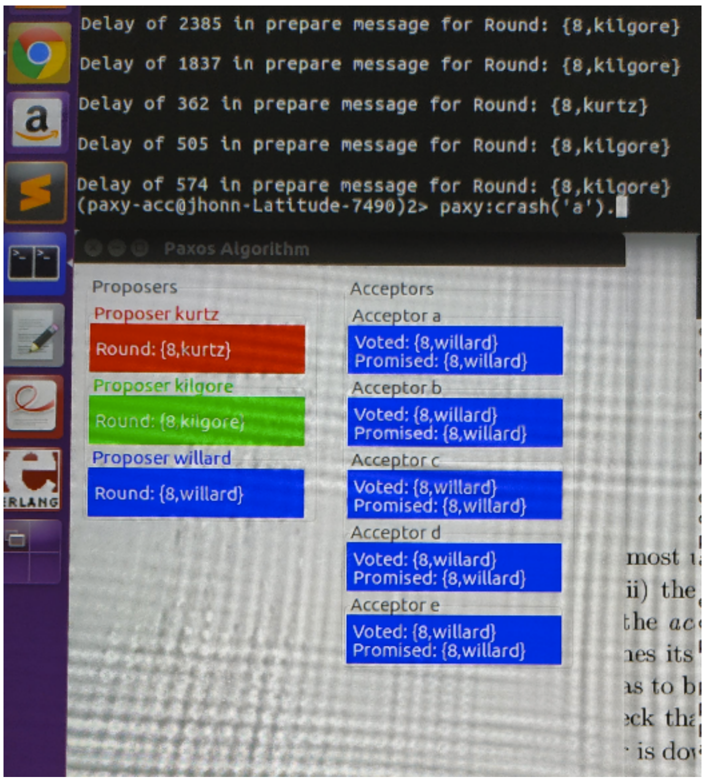
Figure 13: Test
paxy:start_proposers([1000,555,666], $'paxy-acc@jhonnLatitude-7490'$).

**Fault tolerance:** In order to make the implementation fault tolerant the acceptor needs toremember what it promises and what it votes for. If we use the module pers, that can be found in 'Appendix B', we can store state changes as we make promises and vote and we can recover our state to the state we had when we crashed.
The fault tolerance works fine we can recover the acceptor succesfully. In the code of the aceptors we use the delay code of the part one in order to guive time enogh to manualy insert a crash and see if recovers. In the following image we show how that goes.

Listing 1: Code Modfications for recover the acceptor.

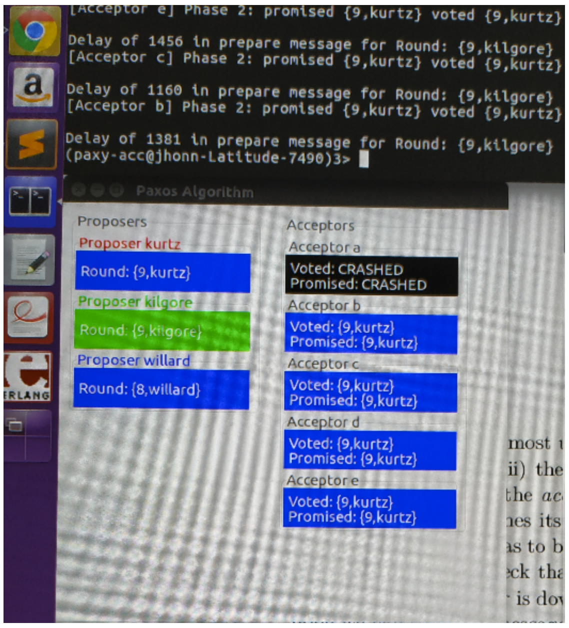
```
-module(acceptor).
-export([start/2,startR/2]).
-define(delay,2800). %A Rand Range for delay

start(Name, PanelId) ->
  spawn(fun() -> init(Name, PanelId) end).

startR(Name, PanelId) ->
  spawn(fun() -> initR(Name, PanelId) end).

init(Name, PanelId) ->
  Promised = order:null(),
  Voted = order:null(),
  Value = na,
  acceptor(Name, Promised, Voted, Value, PanelId).

initR(Name, PanelId) ->
```

```
  pers:open(Name),
  {Promised,Voted,Value,PanelIdR}=pers:read(Name),
  pers:close(Name),
  io:format("Recover␣Info␣:[Acceptor␣~w]␣promised␣~w␣voted␣~w␣colour␣~w~n",
            [Name, Promised, Voted, Value]),
  if PanelIdR==na ->
      acceptor(Name, Promised, Voted, Value, PanelId);
    true ->
      acceptor(Name, Promised, Voted, Value, PanelIdR)
  end.

acceptor(Name, Promised, Voted, Value, PanelId) ->
  receive
    {prepare, Proposer, Round} ->
      R = rand:uniform(?delay),
      io:format("~nDelay␣of␣~w␣in␣prepare␣message␣for␣Round:␣~w~n",
                [R, Round]),
      timer:sleep(R),
      case order:gr(Round, Promised) of
        true ->
          pers:open(Name),
          pers:store(Name,Round,Voted,Value,PanelId),
          pers:close(Name),
          Proposer ! {promise, Round, Voted, Value},
      io:format("[Acceptor␣~w]␣Phase␣1:␣promised␣~w␣voted␣~w␣colour␣~w~n",
                [Name, Round, Voted, Value]),
          % Update gui
          Colour = case Value of na -> {0,0,0}; _ -> Value end,
          PanelId ! {updateAcc, "Voted:␣" ++ io_lib:format("~p", [Voted]),
                    "Promised:␣" ++ io_lib:format("~p", [Round]), Colour},
          acceptor(Name, Round, Voted, Value, PanelId);
        false ->
          Proposer ! {sorry, {prepare, Round}},
          acceptor(Name, Promised, Voted, Value, PanelId) %
      end;
    {accept, Proposer, Round, Proposal} ->
      R = rand:uniform(?delay),
      io:format("~nDelay␣of␣~w␣in␣vote␣message␣for␣Round:␣~w~n",
                [R, Round]),
      timer:sleep(R),
      case order:goe(Round, Promised) of
        true ->
          Proposer ! {vote, Round},
          case order:goe(Round, Voted) of
            true ->
              pers:open(Name),
              pers:store(Name,Promised,Round,Proposal,PanelId),
              pers:close(Name),
      io:format("[Acceptor␣~w]␣Phase␣2:␣promised␣~w␣voted␣~w␣colour␣~w~n",
                [Name, Round, Promised, Proposal]),
            % Update gui
            PanelId ! {updateAcc, "Voted: " ++ io_lib:format("~p", [Round]),
                "Promised: " ++ io_lib:format("~p", [Promised]), Proposal},
            acceptor(Name, Promised, Round, Proposal, PanelId);
          false ->
            acceptor(Name, Promised, Voted, Value, PanelId)
        end;
      false ->
        Proposer ! {sorry, {accept, Round}},
        acceptor(Name, Promised, Voted, Value, PanelId)
    end;
  stop ->
    PanelId ! stop,
    ok
end.
```



```
Delay of 2385 in prepare message for Round: {8,kilgore}
Delay of 1837 in prepare message for Round: {8,kilgore}
Delay of 362 in prepare message for Round: {8,kurtz}
Delay of 505 in prepare message for Round: {8,kilgore}
Delay of 574 in prepare message for Round: {8,kilgore}
(paxy-acc@jhonn-Latitude-7490)2> paxy:crash('a').
```

Figure 14: Test going.



```
[Acceptor e] Phase 2: promised {9,kurtz} voted {9,kurtz}

Delay of 1456 in prepare message for Round: {9,kilgore}
[Acceptor c] Phase 2: promised {9,kurtz} voted {9,kurtz}

Delay of 1160 in prepare message for Round: {9,kilgore}
[Acceptor b] Phase 2: promised {9,kurtz} voted {9,kurtz}

Delay of 1381 in prepare message for Round: {9,kilgore}
(paxy-acc@jhonn-Latitude-7490)3>
```

Paxos Algorithm

Proposers

Proposer kurtz
Round: {9,kurtz}

Proposer kilgore
Round: {9,kilgore}

Proposer willard
Round: {8,willard}

Acceptors

Acceptor a
Voted: CRASHED
Promised: CRASHED

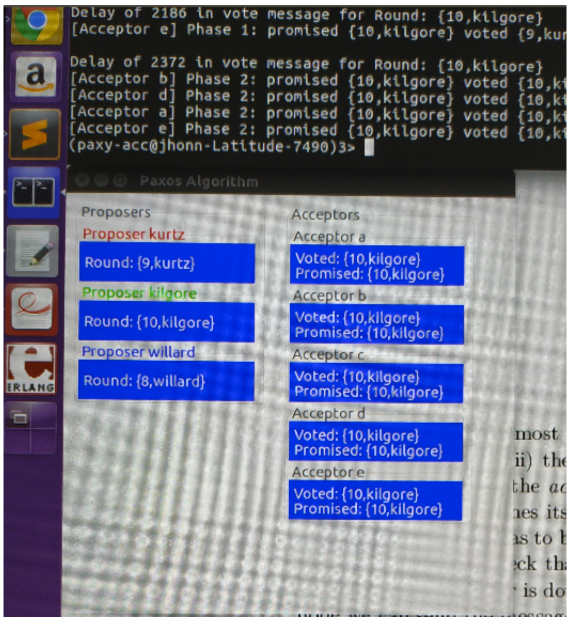
Acceptor b
Voted: {9,kurtz}
Promised: {9,kurtz}

Acceptor c
Voted: {9,kurtz}
Promised: {9,kurtz}

Acceptor d
Voted: {9,kurtz}
Promised: {9,kurtz}

Acceptor e
Voted: {9,kurtz}
Promised: {9,kurtz}

Figure 15: Introducing Crash.



```
Delay of 2186 in vote message for Round: {10,kilgore}
[Acceptor e] Phase 1: promised {10,kilgore} voted {9,kur

Delay of 2372 in vote message for Round: {10,kilgore}
[Acceptor b] Phase 2: promised {10,kilgore} voted {10,ki
[Acceptor d] Phase 2: promised {10,kilgore} voted {10,ki
[Acceptor a] Phase 2: promised {10,kilgore} voted {10,ki
[Acceptor e] Phase 2: promised {10,kilgore} voted {10,ki
(paxy-acc@jhonn-Latitude-7490)3>
```

Paxos Algorithm

Proposers

Proposer kurtz
Round: {9,kurtz}

Proposer kilgore
Round: {10,kilgore}

Proposer willard
Round: {8,willard}

Acceptors

Acceptor a
Voted: {10,kilgore}
Promised: {10,kilgore}

Acceptor b
Voted: {10,kilgore}
Promised: {10,kilgore}

Acceptor c
Voted: {10,kilgore}
Promised: {10,kilgore}

Acceptor d
Voted: {10,kilgore}
Promised: {10,kilgore}

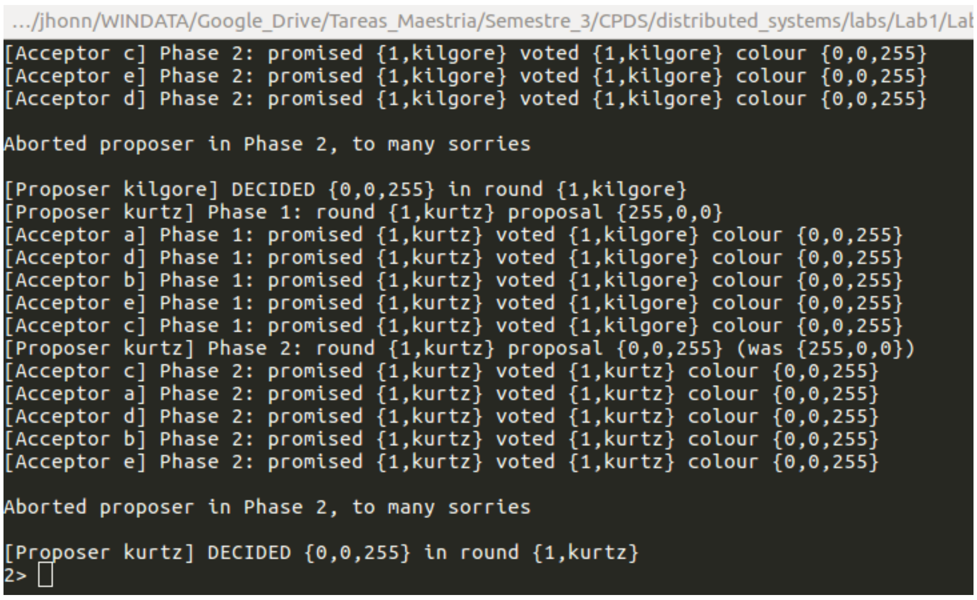
Acceptor e
Voted: {10,kilgore}
Promised: {10,kilgore}

Figure 16: Acceptor Recovers.

**Improvement based on sorry messages:** There are some improvements that could be made in the implementation of the proposer. If we need three promises for a quorum and we have received three sorry messages from the in total five acceptors then we can abort the ballot.
The implementation works very good with this improve, it finish more fast, below is the code of the proposer implementation.

Listing 2: Code Modfications for improve the proposer.

```
-module(proposer).
-export([start/5]).

-define(timeout, 2000).
-define(backoff, 10).

start(Name, Proposal, Acceptors, Sleep, PanelId) ->
  spawn(fun() -> init(Name, Proposal, Acceptors, Sleep, PanelId) end).

init(Name, Proposal, Acceptors, Sleep, PanelId) ->
  timer:sleep(Sleep),
  Round = order:first(Name),
  round(Name, ?backoff, Round, Proposal, Acceptors, PanelId).

round(Name, Backoff, Round, Proposal, Acceptors, PanelId) ->
  io:format("[Proposer␣~w]␣Phase␣1:␣round␣~w␣proposal␣~w~n",
            [Name, Round, Proposal]),
  % Update gui
  PanelId ! {updateProp, "Round:␣" ++ io_lib:format("~p", [Round]), Proposal},
  case ballot(Name, Round, Proposal, Acceptors, PanelId) of
    {ok, Decision} ->
      io:format("[Proposer␣~w]␣DECIDED␣~w␣in␣round␣~w~n",
                [Name, Decision, Round]),
      PanelId ! stop,
      {ok, Decision};
    abort ->
      timer:sleep(rand:uniform(Backoff)),
      Next = order:inc(Round),
      round(Name, (2*Backoff), Next, Proposal, Acceptors, PanelId)
  end.

ballot(Name, Round, Proposal, Acceptors, PanelId) ->
  prepare(Round, Acceptors),
  Quorum = (length(Acceptors) div 2) + 1,        %The majority, half + 1
  N_Sorry = Quorum, % ???
  MaxVoted = order:null(),
  case collect(Quorum, N_Sorry, Round, MaxVoted, Proposal) of
    {accepted, Value} ->
      io:format("[Proposer␣~w]␣Phase␣2:␣round␣~w␣proposal␣~w␣(was␣~w)~n",
                [Name, Round, Value, Proposal]),
      % update gui
      PanelId ! {updateProp, "Round:␣" ++ io_lib:format("~p", [Round]), Value},
      accept(Round, Value, Acceptors),
      case vote(Quorum, N_Sorry, Round) of
        ok ->
          {ok, Value};
        abort ->
          abort
      end;
    abort ->
      abort
  end.

collect(0, _, _, _, Proposal) ->
  {accepted, Proposal};

collect(_, 0, _, _, Proposal) ->
   io:format("~nAborted proposer in Phase 1, to many sorries ~n~n"),
   abort;                                  % Impruve of sorry counting

collect(N, N_Sorry, Round, MaxVoted, Proposal) ->
  receive
    {promise, Round, _, na} ->
      collect(N-1, N_Sorry, Round, MaxVoted, Proposal);
    {promise, Round, Voted, Value} ->
      case order:gr(Voted, MaxVoted) of
        true ->
          collect(N-1, N_Sorry, Round, Voted, Value);
        false ->
          collect(N-1, N_Sorry, Round, MaxVoted, Proposal)
      end;
    {promise, _, _,  _} ->
      collect(N, N_Sorry, Round, MaxVoted, Proposal);
    {sorry, {prepare, Round}} ->
      collect(N, N_Sorry-1, Round, MaxVoted, Proposal);
    {sorry, _} ->
      collect(N, N_Sorry-1,Round, MaxVoted, Proposal)
  after ?timeout ->
    abort
  end.

vote(0, _, _) ->
  io:format("~nAborted proposer in Phase 2, to many sorries ~n~n"),
  ok;
vote(_, 0, _) ->
  abort;
vote(N, N_Sorry, Round) ->
  receive
    {vote, Round} ->
      vote(N-1, N_Sorry, Round);
    {vote, _} ->
      vote(N, N_Sorry, Round);
    {sorry, {accept, Round}} ->
      vote(N, N_Sorry-1, Round);
    {sorry, _} ->
      vote(N, N_Sorry-1, Round)
  after ?timeout ->
    abort
  end.

prepare(Round, Acceptors) ->
  Fun = fun(Acceptor) ->
    send(Acceptor, {prepare, self(), Round})
  end,
  lists:foreach(Fun, Acceptors).

accept(Round, Proposal, Acceptors) ->
  Fun = fun(Acceptor) ->
    send(Acceptor, {accept, self(), Round, Proposal})
  end,
  lists:foreach(Fun, Acceptors).

send(Name, Message) ->
  Name ! Message.
```



```
.../jhonn/WINDATA/Google_Drive/Tareas_Maestria/Semestre_3/CPDS/distributed_systems/labs/Lab1/Lab
[Acceptor c] Phase 2: promised {1,kilgore} voted {1,kilgore} colour {0,0,255}
[Acceptor e] Phase 2: promised {1,kilgore} voted {1,kilgore} colour {0,0,255}
[Acceptor d] Phase 2: promised {1,kilgore} voted {1,kilgore} colour {0,0,255}

Aborted proposer in Phase 2, to many sorries

[Proposer kilgore] DECIDED {0,0,255} in round {1,kilgore}
[Proposer kurtz] Phase 1: round {1,kurtz} proposal {255,0,0}
[Acceptor a] Phase 1: promised {1,kurtz} voted {1,kilgore} colour {0,0,255}
[Acceptor d] Phase 1: promised {1,kurtz} voted {1,kilgore} colour {0,0,255}
[Acceptor b] Phase 1: promised {1,kurtz} voted {1,kilgore} colour {0,0,255}
[Acceptor e] Phase 1: promised {1,kurtz} voted {1,kilgore} colour {0,0,255}
[Acceptor c] Phase 1: promised {1,kurtz} voted {1,kilgore} colour {0,0,255}
[Proposer kurtz] Phase 2: round {1,kurtz} proposal {0,0,255} (was {255,0,0})
[Acceptor c] Phase 2: promised {1,kurtz} voted {1,kurtz} colour {0,0,255}
[Acceptor a] Phase 2: promised {1,kurtz} voted {1,kurtz} colour {0,0,255}
[Acceptor d] Phase 2: promised {1,kurtz} voted {1,kurtz} colour {0,0,255}
[Acceptor b] Phase 2: promised {1,kurtz} voted {1,kurtz} colour {0,0,255}
[Acceptor e] Phase 2: promised {1,kurtz} voted {1,kurtz} colour {0,0,255}

Aborted proposer in Phase 2, to many sorries

[Proposer kurtz] DECIDED {0,0,255} in round {1,kurtz}
2>
```

Figure 17: Terminal output of test.

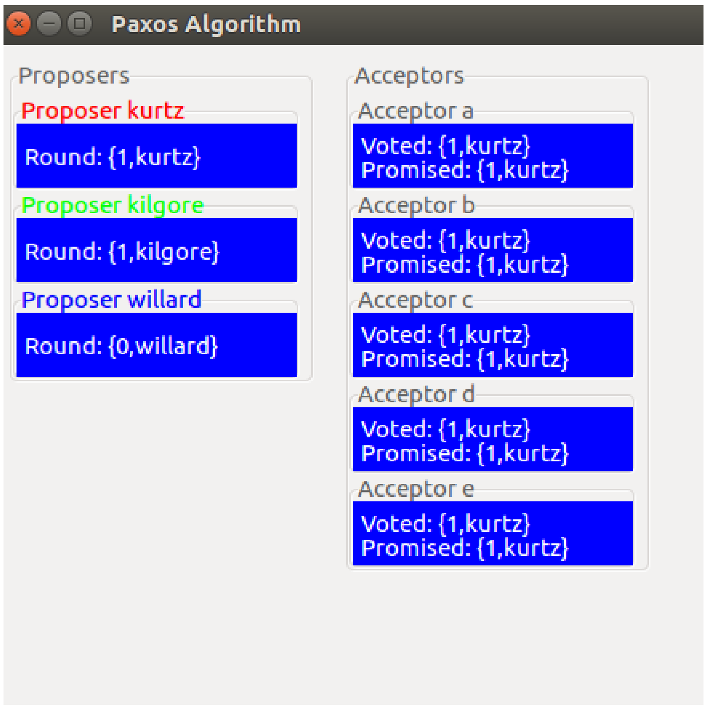


Figure 18: Test: paxy:start([1,1,1]).

## Personal opinion

*Jonnatan: The seminar is nice and complete, maybe a little bit more of talk for real applications using this algorithm.*