Quizmaster

Assignment 5

Kai Arne S. Myklebust, Silvan Adrian

Handed in: October 19, 2018

Contents

1 Solution **2**
- 1.1 Files . . . 2
- 1.2 Running the programm . . . 2
- 1.3 Running the tests . . . 2

2 Implementation **3**
- 2.1 Gen-Statem . . . 3
- 2.2 Data Structure . . . 4
- 2.3 All states . . . 4
  - 2.3.1 get_questions . . . 4
- 2.4 Editable state . . . 4
  - 2.4.1 add_question . . . 4
  - 2.4.2 Play . . . 5
  - 2.4.3 Other Messages . . . 5
- 2.5 Between_questions state . . . 5
  - 2.5.1 join . . . 5
  - 2.5.2 leave . . . 5
  - 2.5.3 next . . . 5
  - 2.5.4 timesup . . . 5
- 2.6 Active_question state . . . 6
  - 2.6.1 join . . . 6
  - 2.6.2 leave . . . 6
  - 2.6.3 guess . . . 6
  - 2.6.4 timesup . . . 6

3 Assessment **6**
3.1 OnlineTA 6
3.2 Scope of Test Cases 6
3.3 Correctness 7
3.4 Code Quality 7

A Code Listing **7**

1 Solution

1.1 Files

All Files are situated in the **src/** folder:

- **src/quizmaster.erl** The quizmaster Server implementation
- **src/quizmaster_helpers.erl** The greetings module implementation
- **tests/quizmaster_conductor.erl** Conductor Implementation for testing
- **tests/quizmaster_player.erl** Player implementation for testing
- **tests/quizmaster_tests.erl** Eunit Tests for quizmaster

1.2 Running the programm

Out of convenience we used a Emakefile which compiles all the erlang files in one go then rather compile each file on it's own. This can be done by using the erlang shell and run:

```
1 make:all([load]).
```

1.3 Running the tests

The tests can be run this way:

```
1 eunit:test(quizmaster_tests, [verbose]).
```

2 Implementation

2.1 Gen-Statem

Since the Quizmaster can be seen as a simple State machine we chose gen_statem. The Quizmaster has overall 3 important states:

- editable
- between_questions
- active_question

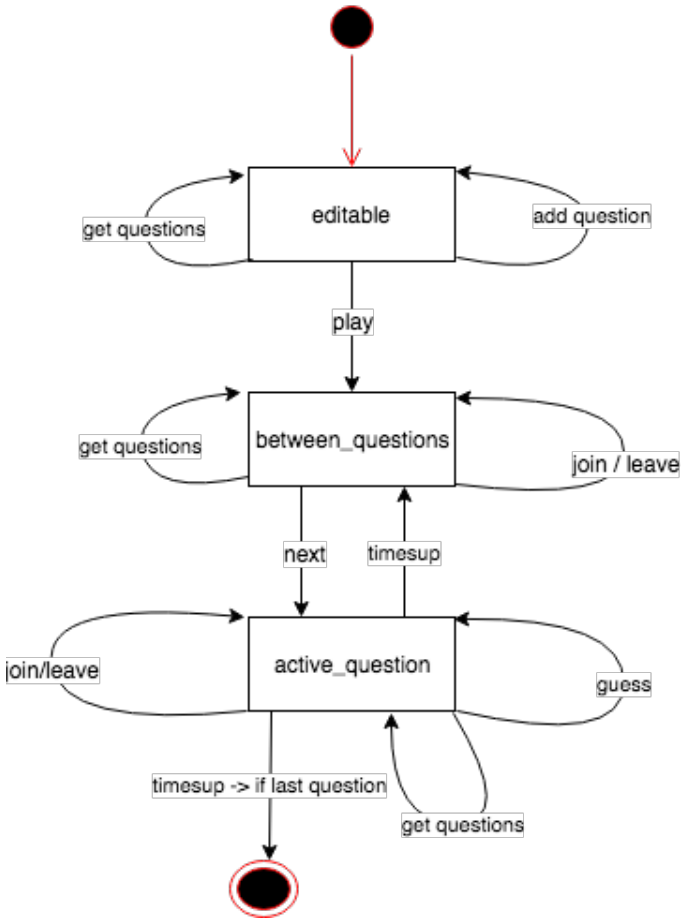

Figure 1: Simple drawing of the quizmaster state machine

2.2 Data Structure

Data with which we loop is a map with following entries:

- **conductor** The Pid of the Conductor (Gamemaker)
- **questions** all questions which belong to a quiz
- **players** all joined players (we chose to either have active or inactive players (left the game) to get through the tests of OnlineTA)
- **active_question** The index of the active question (in the questions list)
- **answered** Saving the first guess for each player, to be sure only one guess can be made for each question
- **distribution** the distribution of the current active question which shows which index has be chosen how many times

2.3 All states

Messages which get accepted in all states.

2.3.1 get_questions

Get all added questions.

2.4 Editable state

In the editable state the quiz can be modified, meaning new questions can be added, join is not allowed.

2.4.1 add_question

So the message add_question adds a new question to the server state, with whom we loop further. The question does have to in the format:

```
1 [Description, [Answers]]
```

Figure 2: Question Format

In case the question doesn't fit the format we will get an error back that we tried to add a question in the wrong format, but we don't check if there is a correct answer added (so technically can add a question without any correct answers).

```
{error, "Question is in wrong format"}
{error, "Wrong Format"}
```

Figure 3: Error Messages

2.4.2 Play

When all Questions have been added a Quiz can be played, which means it changes it's state to between_questions and no more questions can be added. Additionally the Conductor gets set (any process which calls play first).

2.4.3 Other Messages

All other messages get ignored in the editable state.

2.5 Between_questions state

In between_questions new Players can join/leave and the next question can be set to active.

2.5.1 join

When joining we check if the nickname already exists, is this is the case then we set the existing player to active again otherwise add it as a new player to the players map in data. And send a message to the conductor by announcing a new player has joined.

2.5.2 leave

When a player leaves the nickname will still exist in the players map, only his status will be set to inactive so that the player still shows in the Report after timesup (this way the distribution still makes sense).

2.5.3 next

Select the next active question (according to the index in data), the state also changes to active_question. Next can be only called by the conductor, which gets checked against the one saved in data. Additionally the next question get broadcasted to all joined players, with the Description and the Answers (removing the correct).

2.5.4 timesup

Return a error in between_questions since there is no active question.

2.6 Active_question state

In Active question a player can join or leave, and the players can make guesses. The conductor then can run timesup to finish up the round and change back to between_questions state.

2.6.1 join

Same as in between_questions.

2.6.2 leave

Same as in between_questions.

2.6.3 guess

A guess can be made on a specific index which then either gives a point (if correct answer) or just gets counted into the distribution of the guesses. If the guess is on a wrong index then the guess is going to be ignored.

2.6.4 timesup

Timesup get called by the conductor and only by him, by doing that the state gets changed back to in between_questions and a reports gets sent out (with distribution, score of all players etc.). In case it was the last question then all players get a Message with quiz_over.

3 Assessment

3.1 OnlineTA

OnlineTA gave ok results back but only the last test wasn't able to fully run through and the end we ended up with guessing what exactly is wished what we implement to get through all test cases, so we gave up in the end.

3.2 Scope of Test Cases

We tried to test all possible good and error cases, but to make it a little easier we ignored some of the messages that get sent (see quizmaster_conductor, quizmaster_player). So overall our tests should tests most cases but not all of them.

3.3 Correctness

In our opinion, we tested possible edge cases but nonetheless the onlineTA end up in a error in the last test, which we weren't able to find out why. So the Solution is not fully correct and still seems to miss a few things, but since it would need too much time to fix those we finished it earlier.

3.4 Code Quality

There are lots of things which could be made easier in the code, especially lots of duplicated code which could be moved into own helper methods to keep quizmaster.erl more clean. Especially in this assignment the quizmaster file got quite big and we just tried to make it a little better by moving some functions into the quizmaster_helpers.erl file.

A Code Listing

```
1  -module(quizmaster).
2  -behaviour(gen_statem).
3  %% API exports.
4  -export([start/0, add_question/2, get_questions/1,
5    play/1, next/1, timesup/1,
6    join/2, leave/2, guess/3]).
7  %% Gen_statem callbacks
8  -export([terminate/3, code_change/4, init/1, callback_mode/0]).
9  -export([editable/3, between_questions/3, active_question/3]).
10
11 %%%-------------------------------------------------------------------
12 %%%
13 %%% Quizmaster API
14 %%%
15 %%%-------------------------------------------------------------------
16 start() ->
17   gen_statem:start(?MODULE, [], []).
18
```

```
19  add_question(Server, {Description, Answers}) ->
20    gen_statem:call(Server, {add_question, {Description,
      ↪  Answers}}).
21
22  get_questions(Server) ->
23    gen_statem:call(Server, get_questions).
24
25  play(Server) ->
26    gen_statem:call(Server, play).
27
28  next(Server) ->
29    gen_statem:call(Server, next).
30
31  timesup(Server) ->
32    gen_statem:call(Server, timesup).
33
34  join(Server, Nickname) ->
35    gen_statem:call(Server, {join, Nickname}).
36
37  leave(Server, Ref) ->
38    gen_statem:call(Server, {leave, Ref}).
39
40  guess(Server, Ref, Index) ->
41    gen_statem:cast(Server, {guess, Ref, Index}).
42
43
44  % get_questions Call
45  handle_event({call, From}, get_questions, Data) ->
46    Questions = maps:get(questions, Data),
47    {keep_state, Data, {reply, From, Questions}};
48
49  handle_event({call, From}, {join, Nickname}, Data) ->
50    PlayersValues = maps:values(maps:get(players, Data)),
51    case quizmaster_helpers:check_if_player_exists(Nickname,
      ↪  PlayersValues) of
52      true -> {keep_state, Data, {reply, From, {error, is_taken}}};
53      false ->  {Pid, _} = From,
54        PlayersMap = maps:get(players, Data),
55        Ref = make_ref(),
56        NewData = Data#{players => PlayersMap#{Ref => {Nickname,
          ↪  Pid, 0, 0}}},
57        maps:get(conductor, NewData) ! {player_joined, Nickname,
          ↪  maps:size(maps:get(players, NewData))},
58        {keep_state, NewData, {reply, From, {ok, Ref}}}
```

```
59    end;
60
61  handle_event({call, From}, {leave, Ref}, Data) ->
62    case maps:is_key(Ref, maps:get(players, Data)) of
63      true ->
64        {Nickname, _, _, _} = maps:get(Ref, maps:get(players,
          ↪  Data)),
65        UpdatedPlayers = maps:remove(Ref, maps:get(players, Data)),
66        NewData = maps:update(players, UpdatedPlayers, Data),
67        maps:get(conductor, NewData) ! {player_left, Nickname,
          ↪  maps:size(maps:get(players, NewData))},
68        {keep_state, NewData, {reply, From, ok}};
69      false ->  {keep_state, Data, {reply, From, {error,
          ↪  who_are_you}}}
70    end;
71
72  % ignore all other unhandled events
73  handle_event(_EventType, _EventContent, Data) ->
74    {keep_state, Data}.
75
76  editable({call, From}, {add_question, Question}, Data) ->
77    case Question of
78      {_, [_ | _]} -> OldQuestions = maps:get(questions, Data),
79        UpdatedQuestions = maps:update(questions,
          ↪  lists:append(OldQuestions, [Question]), Data),
80        {keep_state, UpdatedQuestions, {reply, From, ok}};
81      {_, []} -> {keep_state, Data, {reply, From, {error, "Question
          ↪  is in wrong format"}}}
82    end;
83
84  % start playing a quiz -> change state to between_questions
85  editable({call, From}, play, Data) ->
86    case maps:get(questions, Data) of
87      [] -> {keep_state, Data, {reply, From, {error,
          ↪  no_questions}}};
88      _ -> {Pid, _} = From,
89        Conductor = maps:update(conductor, Pid, Data),
90        {next_state, between_questions, Conductor, {reply, From,
          ↪  ok}}
91    end;
92
93  % catch join message while editable
94  editable({call, From}, {join, _Name}, Data) ->
95    {keep_state, Data, {reply, From, {error, "Can't join while
      ↪  editable"}}};
```

```
96
97  editable(EventType, EventContent, Data) ->
98    handle_event(EventType, EventContent, Data).
99
100 between_questions({call, From}, next, Data) ->
101   case quizmaster_helpers:is_conductor(From, Data) of
102     true -> {Description, Answers} =
        ↪  lists:nth(maps:get(active_question, Data),
        ↪  maps:get(questions, Data)),
103       NewData = Data#{distribution =>
          ↪  quizmaster_helpers:init_distribution(length(Answers),
          ↪  #{})},
104       quizmaster_helpers:broadcast_next_question({Description,
          ↪  Answers}, maps:to_list(maps:get(players, Data))),
105       {next_state, active_question, NewData, {reply, From, {ok,
          ↪  {Description, Answers}}}};
106     false -> {keep_state, Data, {reply, From, {error,
        ↪  who_are_you}}}
107   end;
108
109 between_questions({call, From}, timesup, Data) ->
110   case quizmaster_helpers:is_conductor(From, Data) of
111     true -> {keep_state, Data, {reply, From, {error,
        ↪  no_question_asked}}};
112     false -> {keep_state, Data, {reply, From, {error, nice_try}}}
113   end;
114 between_questions({call, From}, {join, Name}, Data) ->
115   handle_event({call, From}, {join, Name}, Data);
116
117 between_questions({call, From}, {leave, Ref}, Data) ->
118   handle_event({call, From}, {leave, Ref}, Data);
119
120 between_questions(EventType, EventContent, Data) ->
121   handle_event(EventType, EventContent, Data).
122
123 active_question({call, From}, next, Data) ->
124   case quizmaster_helpers:is_conductor(From, Data) of
125     true -> {keep_state, Data, {reply, From, {error,
        ↪  has_active_question}}};
126     false -> {keep_state, Data, {reply, From, {error,
        ↪  who_are_you}}}
127   end;
128
129 active_question({call, From}, timesup, Data) ->
```

```
130     case quizmaster_helpers:is_conductor(From, Data) of
131       true ->
132         case length(maps:get(questions, Data)) ==
            ↪  maps:get(active_question, Data) of
133           true ->
134
                ↪  quizmaster_helpers:broadcast_quiz_over(From,maps:to_list(maps:get(pla
                ↪  Data))),
135             {stop_and_reply, normal, {reply, From,
                ↪  quizmaster_helpers:get_report(Data, true)}};
136           false -> NewData = maps:update(active_question,
              ↪  map_get(active_question, Data) + 1, Data),
137             NewData2 =
                ↪  quizmaster_helpers:reset_last_points(NewData),
138             {next_state, between_questions, NewData2#{answered =>
                ↪  []}, {reply, From,
                ↪  quizmaster_helpers:get_report(Data, false)}}
139         end;
140       false -> {keep_state, Data, {reply, From, {error, nice_try}}}
141     end;
142
143 active_question(cast, {guess, Ref, Index}, Data) ->
144     case quizmaster_helpers:check_index_in_range(Index, Data, Ref)
      ↪  of
145       true -> NewData = quizmaster_helpers:check_guess(Ref, Index,
          ↪  Data),
146         {keep_state, NewData};
147       false -> {keep_state, Data} % ignore guess if Index out of
          ↪  range or from a not in players map ref
148     end;
149
150 active_question({call, From}, {join, Name}, Data) ->
151     handle_event({call, From}, {join, Name}, Data);
152
153 active_question({call, From}, {leave, Name}, Data) ->
154     handle_event({call, From}, {leave, Name}, Data).
155
156 %% Mandatory callback functions
157 terminate(_Reason, _State, _Data) ->
158     void.
159
160 code_change(_Vsn, State, Data, _Extra) ->
161     {ok, State, Data}.
162
```

```
163 init([]) ->
164   %% Set the initial state + data
165   State = editable, Data = #{conductor => none, questions => [],
      ↪  players => #{}, active_question => 1, answered => [],
      ↪  distribution => #{}},
166   {ok, State, Data}.
167
168 callback_mode() -> state_functions.
```

```
1  -module(quizmaster_helpers).
2  -export([check_index_in_range/3,
3    get_active_question/1,
4    is_conductor/2,
5    check_if_player_exists/2,
6    check_guess/3,
7    init_distribution/2,
8    broadcast_next_question/2,
9    broadcast_quiz_over/2,
10   get_report/2,
11   reset_last_points/1,
12   reset_points/1
13   ]).
14
15 % check index of guess
16 check_index_in_range(Index, _, _) when Index < 1 -> false;
17 check_index_in_range(Index, Data, Ref) when Index > 0 ->
18   index_is_in_range(Index, get_active_question(Data)) and
      ↪  maps:is_key(Ref, maps:get(players, Data)).
19
20 index_is_in_range(Index, {_, Answers}) ->
21   Index =< length(Answers).
22
23 % get the at the moment active question
24 get_active_question(Data) ->
25   lists:nth(maps:get(active_question, Data), maps:get(questions,
      ↪  Data)).
26
27 % check if pid in from is the same as in Data
28 is_conductor({Pid, _}, Data) ->
29   Pid == maps:get(conductor, Data).
30
31 % check if player exists in players map
32 check_if_player_exists(_, []) -> false;
```

```
33  check_if_player_exists(Nickname, [{Playername, _, _, _} |
    ↪  Players]) ->
34    case Nickname == Playername of
35      true -> true;
36      false -> check_if_player_exists(Nickname, Players)
37    end.
38
39  % check if guess is correct or not
40  check_guess(Ref, Index, Data) ->
41    CurrentQuestion = get_active_question(Data),
42    case is_first_guess(Ref, Data) of
43      true -> UpdatedData = maps:update(answered,
        ↪  lists:append([Ref], maps:get(answered, Data)), Data),
44        case is_correct(Index, CurrentQuestion) of
45          true -> NewData = update_players_score(Ref, UpdatedData,
            ↪  correct), NewData2 = update_distribution(Index,
            ↪  NewData), NewData2;
46          false -> NewData = update_players_score(Ref, UpdatedData,
            ↪  incorrect), NewData2 = update_distribution(Index,
            ↪  NewData), NewData2
47        end;
48      false -> Data %send back old Data, so only first guess counts
49    end.
50
51  % only accept the first guess by remembering Ref of players who
    ↪  answered
52  is_first_guess(Ref, Data) ->
53    case lists:member(Ref, maps:get(answered, Data)) of
54      true -> false;
55      false -> true
56    end.
57
58  % check if index is the right answer
59  is_correct(Index, {_, Answers}) ->
60    Answer = lists:nth(Index, Answers),
61    case Answer of
62      {correct, _} -> true;
63      _ -> false
64    end.
65
66  update_players_score(Ref, UpdatedData, Correct) ->
67    case Correct of
68      correct ->
69        {_Nickname, _Pid, Total, _LastScore} = maps:get(Ref,
          ↪  maps:get(players, UpdatedData)),
```

```
70          maps:update(players, maps:update(Ref, {_Nickname, _Pid,
            ↪  Total + 1, 1}, maps:get(players, UpdatedData)),
            ↪  UpdatedData);
71        incorrect ->
72          {_Nickname, _Pid, Total, _LastScore} = maps:get(Ref,
            ↪  maps:get(players, UpdatedData)),
73          maps:update(players, maps:update(Ref, {_Nickname, _Pid,
            ↪  Total, 0}, maps:get(players, UpdatedData)),
            ↪  UpdatedData)
74      end.
75
76  init_distribution(1, Map) -> Map#{1 => 0};
77  init_distribution(AnswerIndex, Map) ->
    ↪  init_distribution(AnswerIndex - 1, Map#{AnswerIndex => 0}).
78
79  % update distribution between index and how many times answered
80  update_distribution(Index, NewData) ->
81    Dist = maps:get(distribution, NewData),
82    Count = maps:get(Index, Dist),
83    UpdatedDist = Dist#{Index => Count + 1},
84    NewData#{distribution => UpdatedDist}.
85
86  % broadcast next_question to all players
87  broadcast_next_question({_Description, _Answers}, []) -> void;
88  broadcast_next_question({Description, Answers}, [{Ref, {_, Pid,
    ↪  _, _}} | Players]) ->
89    NewAnswers = remove_correct(Answers),
90    Pid ! {next_question, Ref, {Description, NewAnswers}},
91    broadcast_next_question({Description, Answers}, Players).
92
93  % broadcast next_question to all players
94  broadcast_quiz_over({_,_}, []) -> void;
95  broadcast_quiz_over({Q,O}, [{_, {_, Pid, _, _}} | Players]) ->
96    Pid ! {Q, quiz_over},
97    broadcast_quiz_over({Q,O}, Players).
98
99  get_report(Data, LastQ) ->
100   LastPoints =
      ↪  get_points_last_question(maps:to_list(maps:get(players,
      ↪  Data))),
101   TotalPoints = get_points_total(maps:to_list(maps:get(players,
      ↪  Data))),
102   {ok, maps:values(maps:get(distribution, Data)),
      ↪  maps:from_list(LastPoints), maps:from_list(TotalPoints),
      ↪  LastQ}.
```

```
103
104 get_points_last_question([]) -> [];
105 get_points_last_question([{_Ref, {Name, _Pid, _Total, LastScore}}
    ↪  | T]) ->
106   [{Name, LastScore} | get_points_last_question(T)].
107
108 get_points_total([]) -> [];
109 get_points_total([{_Ref, {Name, _Pid, Total, _LastScore}} | T])
    ↪  ->
110   [{Name, Total} | get_points_total(T)].
111
112 remove_correct([]) -> [];
113 remove_correct([Answer | Answers]) ->
114   case Answer of
115     {_, Text} -> [Text | remove_correct(Answers)];
116     Text -> [Text | remove_correct(Answers)]
117   end.
118
119
120 reset_last_points(Data) ->
121   PlayerList = reset_points(maps:to_list(maps:get(players,
    ↪  Data))),
122   maps:update(players, maps:from_list(PlayerList), Data).
123
124 reset_points([]) -> [];
125 reset_points([{Ref, {Nickname, Pid, Total, _}} | Players]) ->
126   [{Ref, {Nickname, Pid, Total, 0}} | reset_points(Players)].
```