Advanced Programming

Exam 2018

Exam Number: 95, Username: zlp432

November 9, 2018

Contents

1 Utility functions

The Code for this task is attached in the appendix A.1.

1.1 Version

The Implementation of Version is relatively straight forward and throughly tested by unit tests, which include the examples from the exam text. I did ended up with a not working implementation before, so I ended up reimplementing the function which is now working as it should.

1.2 Merge

Merge is implemented as described in the exam text and tested with many different examples in the unit tests, which all run through. I had some problems with matching the constraints together, since I kind of lost overview of the function. Especially ending up when merging only with same package and the different ones (not matching) where not added to the resulting list but in the end just forgot to append the rest to the result.

1.3 Assessment

The Utility functions seems to work as intended, as least I was able to reuse them in the parser, and thanks to lots of unit tests to both functions I do believe they work as they should.

2 Parsing appm databases

2.1 Choice of parser library

I implemented the Parser for appm in parsec, mostly out of this reason:

- Better Error handling compared to ReadP
- I do have more experience with Parsec then ReadP (Assignments)

I did end up using **try** quite a lot, which wasn't my intention at all but with the presented Grammar I haven't found a better solution and overall the parser works more or less.

2.2 Grammar

I decided to make a more strict choice about the Clauses, by parsing them in a fixed ordering (name first etc.), I didn't find much of a better solution for that grammar.

2.3 Assessment

I did quite a few unit tests for the parser (including failing ones), since not everything ended up to be working or there was just not enough time left to fixing all the bugs which showed up.

3 Solver

4 Earls of Ravnica

The code for this task can be found in Appendix

4.1 Solution

4.2 Implementation

The earls of Ravnica can be seen as a state machine for which I chose to use gen_statem. The following states exist:

- Under Configuration
- Under Activation
- Active
- Shutting down

4.3 Data Structure

The Data structure I used to implement Ravnica consists of a map with following entries:

- **description** Saves the description which gets saved when starting a server
- **connections** Map for Handling the connections from one District to an other
- **creatures** Map for handling all the entered/active creatures on a Server
- **trigger** Set a trigger for a district

4.4 All states

Messages which get accepted in all states.

4.4.1 get_description

Gets the description `Desc` which gets set on create of a District.

4.5 Under configuration

As soon as a Server started it is in the under_configuration state.

4.5.1 connect

Connects 2 District with a Action, by saving it in the `connections` map, connects can only be made while district is under configuration in other states an error gets returned.

4.5.2 trigger

Under configuration also a trigger can be added to the server, here always the last one gets taken (overwritting whit the newest one). Trigger gets rung whenever a creature enters or leaves a district.

4.6 Under activation

When `activate` gets called the district and it's neighbors need to get activated, under_activation is a intermediate state until all neighbors and the district itself are activated. In case the neighbors can't be activated (for example when a neighbor got shutdown), then the server goes back to the state of under_configuration.

4.6.1 activate

Activate tries to activate all it's neighbors and changes the state of the server to `active` or back to `under_configuration`.

4.7 Active

In the active state, no more new connections can be added, also no triggers. So as soon as a district and it's neighbors is activated, it should only be possible to either run `get_description`, `enter` or `take_action` and of course shutting down.

4.8 Shutting down

When shutting down is called all neighbors of a district will be shut down as well and this can be propagated until all districts and it's neighbors are shutdown.

4.8.1 shutdown

4.9 Territories with cycle

4.10 Assassment

A Code Listing

A.1 Question 1.1: handin/appm/src/Utils.hs

```haskell
module Utils where

-- Any auxiliary code to be shared by Parser, Solver, or tests
-- should be placed here.

import Defs

instance Ord Version where
    (<=) (V []) (V []) = True
    (<=) (V ((VN _ _):_)) (V []) = False
    (<=) (V []) (V ((VN _ _):_)) = True
    (<=) (V ((VN v1int v1str) : vnmbr1)) (V ((VN v2int v2str) : vnmbr2))
        | v1int < v2int = True
        | v1int > v2int = False
        | length(v1str) < length(v2str) = True
        | length(v1str) > length(v2str) = False
        | v1str < v2str = True
        | v1str > v2str = False
        | otherwise = (V vnmbr1) <= (V vnmbr2)

merge :: Constrs -> Constrs -> Maybe Constrs
merge [] [] = Just []
merge c1 [] = Just c1
merge [] c2 = Just c2
merge (const:c1) (c2) = case constInC2 const c2 [] of
                            Just x -> merge c1 (x)
                            Nothing -> Nothing

-- Check if Constraint from c1 is in the Constraint list C2
constInC2 :: (PName, PConstr) -> Constrs -> Constrs -> Maybe Constrs
constInC2 const [] x = Just (x ++ [const])
constInC2 const (c2const:c2tail) x =
```

```
33                  case fst const == fst c2const of
34                      True -> case mergeConst (snd const) (snd c2const) of
35                                  Nothing -> Nothing
36                                  Just mconst -> Just (x ++ [(fst const,
                                    ↪  mconst)] ++ c2tail)
37                      False -> constInC2 const c2tail (x ++ [c2const])
38
39  -- Compare the 2 Constraints with
40  mergeConst :: PConstr -> PConstr -> Maybe PConstr
41  mergeConst (b1,c1v1,c1v2) (b2,c2v1,c2v2)
42          | c1v2 <= c2v1 = Nothing
43          | c2v2 <= c1v1 = Nothing
44          | b1 == True && b2 == True = Just (b1, (largest c1v1 c2v1),
            ↪  (smallest c1v2 c2v2))
45          | b1 == False && b2 == False = Just (b1, (largest c1v1 c2v1),
            ↪  (smallest c1v2 c2v2))
46          | b1 == True && b2 == False = Just (b1, (largest c1v1 c2v1),
            ↪  (smallest c1v2 c2v2))
47          | b1 == False && b2 == True = Just (b2, (largest c1v1 c2v1),
            ↪  (smallest c1v2 c2v2))
48  mergeConst _ _ = Nothing
49
50  -- Return the smaller of 2 Versions
51  smallest :: Version -> Version -> Version
52  smallest v1 v2 =
53      case v1 <= v2 of
54          True -> v1
55          False -> v2
56
57  -- Returns the bigger of 2 Versions
58  largest :: Version -> Version -> Version
59  largest v1 v2 =
60      case v1 >= v2 of
61          True -> v1
62          False -> v2
```

A.2 Question 1.2: handin/appm/src/ParserImpl.hs

```
1  module ParserImpl where
2
3  -- put your parser in this file. Do not change the types of the following
4  -- exported functions
5  import Data.Char
6  import Defs
7  import Text.Parsec.Char
8  import Text.Parsec.Combinator
9  import Text.Parsec.Prim
```

```haskell
import Text.Parsec.String
import Utils
import Control.Monad (guard)

parseVersion :: String -> Either ErrMsg Version
parseVersion str =
  case parse
         (do res <- (many parseVersionN)
             return res)
         "Parse Error"
         str of
    Left a -> Left (show a)
    Right b -> Right ((V b))

parseVersionN :: Parser VNum
parseVersionN = do
  number <- read <$> (many1 (satisfy isDigit))
  guard (number < 1000000)
  string <- many lower
  guard (length(string) <= 4)
  _ <- optional (char '.')
  return (VN number string)

parseDatabase :: String -> Either ErrMsg Database
parseDatabase db =
  case parse
         (do res <- (many parsePackage)
             eof
             return res)
         "Parse Error"
         db of
    Left a -> Left (show a)
    Right b -> Right (DB b)

-- Parse Packages
parsePackage :: Parser Pkg
parsePackage = do
  _ <- parseWhitespace (caseString "package")
  _ <- parseWhitespace (string "{")
  pname <- parseName
  version <- try parseStringVersion <|> return (V [VN 1 ""])
  description <- try parseDescription <|> return ""
  deps <- many (choice [try parseRequires, try parseConflicts])
  _ <- parseWhitespace (string "}")
  return
    Pkg
      { name = pname
      , ver = version
      , desc = description
```

```
59          -- filter self referential Constraints
60        , deps = filter (\(name, _) -> name /= pname) (cleanConst (concat
          ↪  (deps)))
61        }
62
63
64  -- Accept 2 "" return "
65  parseHighComma :: Parser Char
66  parseHighComma = do
67                      _ <- char '"'
68                      _ <- char '"'
69                      return '"'
70
71  parseHighComma2 :: Parser [Char]
72  parseHighComma2 = do
73                      _ <- char '"'
74                      _ <- char '"'
75                      return ['"']
76
77  -- Parse Package name
78  parseName :: Parser PName
79  parseName = do
80    _ <- parseWhitespace (caseString "name")
81    _ <- parseWhitespace (optional (char '"'))
82    name <- many1 (letter <|> digit <|> char '-' <|> try parseHighComma)
83    guard((last name) /= '-')
84    _ <- optional (char '"')
85    _ <- optional (string ";")
86    return (P name)
87
88  parseStringVersion :: Parser Version
89  parseStringVersion = do
90    _ <- parseWhitespace (caseString "version")
91    version <- parseWhitespace (many1 (digit <|> letter <|> char '.'))
92    optional (string ";")
93    case parseVersion version of
94      Right a -> return a
95      _ -> fail "Version wasn't possible to parse"
96
97  escape :: Parser String
98  escape = do
99    d <- char '\\'
100   c <- oneOf "\\\"0nrvtbf" -- all the characters which can be escaped
101   return [d, c]
102
103 nonEscape :: Parser Char
104 nonEscape = noneOf "\\\"\0\n\r\v\t\b\f"
105
106 character :: Parser String
```

```
107 character = fmap return nonEscape <|> escape
108
109 parseDescription :: Parser String
110 parseDescription = do
111   _ <- parseWhitespace (caseString "description")
112   parseWhitespace (char '"')
113   description <- many (character <|> ( try parseHighComma2))
114   char '"'
115   _ <- optional (string ";")
116   return $ concat(description)
117
118 parseRequires :: Parser Constrs
119 parseRequires = do
120   _ <- parseWhitespace (caseString "requires")
121   pconsts <-
122     parseWhitespace
123       (many
124          (choice
125             [ try (parsePConstrH (True))
126             , try (parseSConstrL (True))
127             , try (parseSConstrH (True))
128             ]))
129   _ <- optional (string ";")
130   return (concat (pconsts))
131
132 parseConflicts :: Parser Constrs
133 parseConflicts = do
134   _ <- parseWhitespace (caseString "conflicts")
135   pconsts <-
136     parseWhitespace
137       (many
138          (choice
139             [ try (parsePConstrH (False))
140             , try (parseSConstrL (False))
141             , try (parseSConstrH (False))
142             ]))
143   _ <- (optional (string ";"))
144   return (concat (pconsts))
145
146 parseSConstrL :: Bool -> Parser Constrs
147 parseSConstrL req = do
148   name <- many1 letter
149   version <- parseWhitespace (parseVersionLow)
150   return [((P name), (req, minV, version))]
151
152 parseSConstrH :: Bool -> Parser Constrs
153 parseSConstrH req = do
154   name <- many1 letter
155   version <- parseWhitespace (parseVersionHigh)
```

```
156     return [((P name), (req, version, maxV))]
157
158 parsePConstrH :: Bool -> Parser Constrs
159 parsePConstrH req = do
160     name <- many1 letter
161     lower <- parseWhitespace (parseVersionLow)
162     _ <- parseWhitespace (string ",")
163     name2 <- parseWhitespace (many1 letter)
164     max <- parseWhitespace (parseVersionHigh)
165     case lower <= max of
166       True ->
167         return [((P name), (req, lower, maxV)), ((P name2), (req, minV, max))]
168       False -> fail "Error"
169
170 parseVersionLow :: Parser Version
171 parseVersionLow = do
172     _ <- string "<"
173     version <- parseWhitespace (many1 (digit <|> letter <|> char '.'))
174     case parseVersion version of
175       Right a -> return a
176       _ -> fail "Version wasn't possible to parse"
177
178 parseVersionHigh :: Parser Version
179 parseVersionHigh = do
180     _ <- string ">="
181     version <- parseWhitespace (many1 (digit <|> letter <|> char '.'))
182     case parseVersion version of
183       Right a -> return a
184       _ -> fail "Version wasn't possible to parse"
185
186 -- Merges parsed constraints to remove duplicates etc.
187 cleanConst :: Constrs -> Constrs
188 cleanConst [] = []
189 cleanConst (x:xs) =
190     case merge xs [x] of
191       Nothing -> []
192       Just a -> a
193
194 isPrintChar :: Char -> Bool
195 isPrintChar c
196     | ord c >= 32 && ord c <= 126 = True
197     | otherwise = False
198
199 parseComment :: Parser ()
200 parseComment = do
201     _ <- string "--"
202     _ <- manyTill anyChar (newLine <|> eof)
203     return ()
204
```

```haskell
--makes newline be of type ()
newLine :: Parser ()
newLine = do
  _ <- newline
  return ()

parseWhitespace :: Parser a -> Parser a
parseWhitespace par = do
  spaces
  optional parseComment
  spaces
  par

caseChar :: Char -> Parser Char
caseChar c = char (toLower c) <|> char (toUpper c)

-- Match any case of the characters
caseString :: String -> Parser String
caseString s = try (mapM caseChar s) <?> "\"" ++ s ++ "\""
```

A.3 Question 2.1: handin/ravnica/district.erl

```erlang
-module(district).
-behaviour(gen_statem).
-export([create/1,
  get_description/1,
  connect/3,
  activate/1,
  options/1,
  enter/2,
  take_action/3,
  shutdown/2,
  trigger/2]).
%% Gen_statem callbacks
-export([terminate/3, code_change/4, init/1, callback_mode/0]).
%State Functions
-export([under_configuration/3, active/3, shutting_down/3,
↪  under_activation/3]).
-type passage() :: pid().
-type creature_ref() :: reference().
-type creature_stats() :: map().
-type creature() :: {creature_ref(), creature_stats()}.
-type trigger() :: fun((entering | leaving, creature(), [creature()])
  -> {creature(), [creature()]}).


-spec create(string()) -> {ok, passage()} | {error, any()}.
```

```
25  create(Desc) ->
26    gen_statem:start(?MODULE, Desc, []).
27
28  -spec get_description(passage()) -> {ok, string()} | {error, any()}.
29  get_description(District) ->
30    gen_statem:call(District, get_description).
31
32  -spec connect(passage(), atom(), passage()) -> ok | {error, any()}.
33  connect(From, Action, To) ->
34    gen_statem:call(From, {connect, Action, To}).
35
36  -spec activate(passage()) -> active | under_activation | impossible.
37  activate(District) ->
38    gen_statem:call(District, activate).
39
40  -spec options(passage()) -> {ok, [atom()]} | none.
41  options(District) ->
42    gen_statem:call(District, options).
43
44  -spec enter(passage(), creature()) -> ok | {error, any()}.
45  enter(District, Creature) ->
46    gen_statem:call(District, {enter, Creature}).
47
48  -spec take_action(passage(), creature_ref(), atom()) -> {ok, passage()} |
    ↪  {error, any()}.
49  take_action(From, CRef, Action) ->
50    gen_statem:call(From, {take_action, CRef, Action}).
51
52  -spec shutdown(passage(), pid()) -> ok.
53  shutdown(District, NextPlane) ->
54    gen_statem:call(District, {shutdown, NextPlane}).
55
56  -spec trigger(passage(), trigger()) -> ok | {error, any()} | not_supported.
57  trigger(District, Trigger) ->
58    gen_statem:call(District, {trigger, Trigger}).
59
60
61  %% States
62  handle_event({call, From}, get_description, Data) ->
63    case maps:is_key(description, Data) of
64      true -> {keep_state, Data, {reply, From, {ok, maps:get(description,
      ↪  Data)}}};
65      false -> {error, "No Description"}
66    end;
67
68  handle_event({call, From}, options, Data) ->
69    {keep_state, Data, {reply, From, {ok, maps:keys(maps:get(connections,
    ↪  Data))}}};
70
```

```
71  % ignore all other unhandled events
72  handle_event({call, From}, activate, Data) ->
73    {next_state, active, Data, {reply, From, ok}};
74
75  handle_event({call, From}, {run_action, CRef, Stats}, Data) ->
76    case maps:is_key(CRef, maps:get(creatures, Data)) of
77      true -> {keep_state, Data, {reply, From, {error, "Creature is already in
      ↪  this District"}}};
78      false -> NewCreatures = maps:put(CRef, Stats, maps:get(creatures,
      ↪  Data)),
79        NewData = maps:update(creatures, NewCreatures, Data),
80        {keep_state, NewData, {reply, From, ok}}
81    end;
82
83  % Handle Enter on other states
84  handle_event({call, From}, {enter, _}, Data) ->
85    {keep_state, Data, {reply, From, {error, "Can't enter in this state"}}};
86
87  % Shutdown can be called in any state
88  handle_event({call, From}, {shutdown, NextPlane}, Data) ->
89    NextPlane ! {shutting_down, From, maps:to_list(maps:get(creatures,
      ↪  Data))},
90    {next_state, shutting_down, Data, {next_event, internal, {From,
      ↪  NextPlane}}};
91
92  handle_event({call, From}, {trigger, _Trigger}, Data) ->
93    {keep_state, Data, {reply, From, {error, "Can't set a trigger in this
      ↪  state"}}};
94
95  handle_event({call, From}, {connect, _Action, _To}, Data) ->
96    {keep_state, Data, {reply, From, {error, "Can't connect in this state"}}};
97
98  % ignore all other unhandled events
99  handle_event(_EventType, _EventContent, Data) ->
100   {keep_state, Data}.
101
102 under_configuration({call, From}, {connect, Action, To}, Data) ->
103   case is_process_alive(To) of
104     true -> case maps:is_key(Action, maps:get(connections, Data)) of
105               false -> Connections = maps:put(Action, To,
                  ↪  maps:get(connections, Data)),
106                 NewData = maps:update(connections, Connections, Data),
107                 {keep_state, NewData, {reply, From, ok}};
108               true -> {keep_state, Data, {reply, From, {error, "Action
                  ↪  already exists"}}}
109             end;
110     false -> {keep_state, Data, {reply, From, {error, "Process not alive
      ↪  anymore"}}}
111   end;
```

```
112
113 under_configuration({call, From}, activate, Data) ->
114   {next_state, under_activation, Data, {next_event, internal, From}};
115
116
117 under_configuration({call, From}, {trigger, Trigger}, Data) ->
118   NewData = maps:update(trigger, Trigger, Data),
119   {keep_state, NewData, {reply, From, ok}};
120
121 %% General Event Handling for state under_configuration
122 under_configuration(EventType, EventContent, Data) ->
123   handle_event(EventType, EventContent, Data).
124
125 under_activation(internal, From, Data) ->
126   Result = broadcast_connection(maps:to_list(maps:get(connections, Data)),
    ↪  From, active),
127   case Result of
128     impossible -> {next_state, under_configuration, Data, {reply, From,
      ↪  Result}};
129     active -> {next_state, active, Data, {reply, From, Result}}
130   end;
131
132 under_activation({call, From}, activate, Data) ->
133   {keep_state, Data, {reply, From, under_activation}};
134
135 under_activation({call, From}, options, Data) ->
136   {keep_state, Data, {reply, From, {ok, maps:keys(maps:get(connections,
    ↪  Data))}}};
137
138 %% General Event Handling for state under_activation
139 under_activation(EventType, EventContent, Data) ->
140   handle_event(EventType, EventContent, Data).
141
142 active({call, From}, {enter, {Ref, Stats}}, Data) ->
143   case maps:is_key(Ref, maps:get(creatures, Data)) of
144     true -> {keep_state, Data, {reply, From, {error, "Creture is already in
      ↪  this District"}}};
145     false -> Creatures = maps:get(creatures, Data),
146       case maps:get(trigger, Data) of
147         none -> Creature1 = none, Creatures1 = none;
148         Trigger -> case run_trigger(Trigger, entering, {Ref, Stats},
          ↪  Creatures) of
149                      {error, _} -> Creature1 = none, Creatures1 = none;
150                      {Creature1, Creatures1} -> {Creature1, Creatures1}
151                    end
152       end,
153       case {Creature1, Creatures1} of
154         {none, none} -> NewCreatures = maps:put(Ref, Stats,
          ↪  maps:get(creatures, Data)),
```

```
155             NewData = maps:update(creatures, NewCreatures, Data);
156           {{Ref1, Stats1}, NewCreatures1} -> NewCreatures = maps:put(Ref1,
              ↪  Stats1, maps:from_list(NewCreatures1)),
157             NewData = maps:update(creatures, NewCreatures, Data)
158         end,
159         {keep_state, NewData, {reply, From, ok}}
160     end;
161
162 active({call, From}, {take_action, CRef, Action}, Data) ->
163     case maps:is_key(Action, maps:get(connections, Data)) of
164       true ->
165         case maps:is_key(CRef, maps:get(creatures, Data)) of
166           false -> {keep_state, Data, {reply, From, {error, "Creature doesn't
              ↪  exist in this district"}}};
167           true -> case maps:get(trigger, Data) of
168                     none -> Creature1 = none, Creatures1 = none;
169                     Trigger ->
170                       RemoveCreature = maps:remove(CRef, maps:get(creatures,
                          ↪  Data)),
171                       RemovedData = maps:update(creatures, RemoveCreature,
                          ↪  Data),
172                       case run_trigger(Trigger, leaving, {CRef, maps:get(CRef,
                          ↪  maps:get(creatures, Data))},
173                         maps:get(creatures, RemovedData)) of
174                         {error, _} -> Creature1 = none, Creatures1 = none;
175                         {Creature1, Creatures1} -> {Creature1, Creatures1}
176                       end
177                   end,
178             case {Creature1, Creatures1} of
179               {none, none} -> NewDataCreatures = Data;
180               {{Ref, Stats}, _} -> NewCreatures = maps:put(Ref, Stats,
                  ↪  maps:get(creatures, Data)),
181                 NewDataCreatures = maps:update(creatures, NewCreatures, Data)
182             end,
183             {NewData, To} = creature_leave(CRef, Action, From,
                ↪  NewDataCreatures),
184             case NewData of
185               error -> {keep_state, Data, {reply, From, {error, To}}};
186               _ -> {keep_state, NewData, {reply, From, {ok, To}}}
187             end
188         end;
189       false -> {keep_state, Data, {reply, From, {error, "Action doesn't
          ↪  exist"}}}
190     end;
191
192 active({call, From}, activate, Data) ->
193     {keep_state, Data, {reply, From, active}};
194
195 %% Handle Calls to active
```

```
196 active(EventType, EventContent, Data) ->
197   handle_event(EventType, EventContent, Data).
198
199 shutting_down(internal, {From, NextPlane}, Data) ->
200   Result = broadcast_shutdown(maps:to_list(maps:get(connections, Data)),
      ↪  From, NextPlane),
201   {stop_and_reply, normal, {reply, From, Result}};
202
203 shutting_down({call, From}, activate, Data) ->
204   {keep_state, Data, {reply, From, impossible}};
205
206 shutting_down({call, From}, options, Data) ->
207   {keep_state, Data, {reply, From, none}};
208
209 shutting_down({call, From}, shutdown, Data) ->
210   {keep_state, Data, {reply, From, ok}};
211
212 %% Handle Calls to shutting_down
213 shutting_down(EventType, EventContent, Data) ->
214   handle_event(EventType, EventContent, Data).
215
216 %% Mandatory callback functions
217 terminate(_Reason, _State, _Data) ->
218   void.
219
220 code_change(_Vsn, State, Data, _Extra) ->
221   {ok, State, Data}.
222
223 % initial State under_configuration
224 init(Desc) ->
225   %% Set the initial state + data
226   State = under_configuration, Data = #{description => Desc, connections =>
      ↪  #{}, creatures => #{}, trigger => none},
227   {ok, State, Data}.
228
229 callback_mode() -> state_functions.
230
231 %% Synchronous Call which should wait until each response
232 broadcast_shutdown([], _, _NextPlane) -> ok;
233 broadcast_shutdown([{_Action, To} | Actions], {Pid, Ref}, NextPlane) ->
234   case is_process_alive(To) of
235     true ->
236       case term_to_binary(To) == term_to_binary(Pid) of
237         true -> void;
238         false -> case term_to_binary(To) == term_to_binary(self()) of
239                    true -> void;
240                    false -> gen_statem:call(To, {shutdown, NextPlane})
241                  end
242       end;
```

```
243       false -> void
244     end,
245     broadcast_shutdown(Actions, {Pid, Ref}, NextPlane).
246
247   %% Synchronous Call which should wait until each response
248   broadcast_connection([], _, Result) -> Result;
249   broadcast_connection([{_Action, To} | Actions], {Pid, Ref}, _) ->
250     case is_process_alive(To) of
251       false -> Result1 = impossible;
252       true -> Result1 = active,
253         case term_to_binary(To) == term_to_binary(Pid) of
254           false -> case term_to_binary(To) == term_to_binary(self()) of
255                      true -> void;
256                      false -> gen_statem:call(To, activate)
257                    end;
258           true -> void
259         end
260     end,
261     broadcast_connection(Actions, {Pid, Ref}, Result1).
262
263   creature_leave(CRef, Action, {_Pid, _}, Data) ->
264     To = maps:get(Action, maps:get(connections, Data)),
265     Stats = maps:get(CRef, maps:get(creatures, Data)),
266     case is_process_alive(To) of
267       true -> case term_to_binary(self()) == term_to_binary(To) of
268                 true -> {Data, To};
269                 false -> case gen_statem:call(To, {run_action, CRef, Stats})
                  ↪  of
270                            ok -> NewCreatures = maps:remove(CRef,
                           ↪  maps:get(creatures, Data)),
271                              NewData = maps:update(creatures, NewCreatures,
                             ↪  Data),
272                              {NewData, To};
273                            {error, Reason} -> {error, Reason}
274                          end
275               end;
276       false -> {error, "District is shutdown"}
277     end.
278
279   run_trigger(Trigger, Event, Creature, Creatures) ->
280     Self = self(),
281     spawn(fun() -> Self ! {self(), Trigger(Event, Creature,
      ↪  maps:to_list(Creatures))} end),
282     receive
283       {_Pid, {Creature1, Creatures1}} -> {Creature1, Creatures1}
284     after
285       2000 -> {error, "didnt't run function"}
286     end.
```

B Tests Listing

B.1 Question 1.1 and Question 1.2 handin/appm/tests/BB/Main.hs

```haskell
module Main where

-- Put your black-box tests in this file

import Defs
import Utils
import Parser (parseDatabase)
import Solver (install, normalize)

import Test.Tasty
import Test.Tasty.HUnit


tests = testGroup "Unit Tests"
    [
        utilities,
        parser,
        example,
        normalizeTests
        --predefined
    ]

utilities = testGroup "Utilities tests"
    [
        -- Versions
        testCase "Version 1 <= 1" $ V [VN 1 ""] <= V [VN 1 ""] @?= True,
        testCase "Version 1 <= 2" $
            V [VN 1 ""] <= V [VN 2 ""] @?= True,
        testCase "Version 2 <= 1" $
            V [VN 2 ""] <= V [VN 1 ""] @?= False,
        testCase "Version 1a <= Verion1z" $
            V [VN 1 "a"] <= V [VN 1 "z"] @?= True,
        testCase "Version 1.1 <= 1.2" $
            V [VN 1 "", VN 1 ""] <= V [VN 1 "", VN 2 ""] @?= True,
        testCase "Version 1.2 <= 1.1" $
            V [VN 1 "", VN 2 ""] <= V [VN 1 "", VN 1 ""] @?= False,
        testCase "Version 1.1a <= 1.1b" $
            V [VN 1 "", VN 1 "a"] <= V [VN 1 "", VN 1 "b"] @?= True,
        testCase "Version 4.0.1 <= 04.00.001" $
            V [VN 4 "", VN 0 "", VN 1 ""] <= V [VN 04 "", VN 00 "", VN 001
            ↪  ""] @?= True,
        testCase "Version 4.0.1.3 <= 4.1.2" $
            V [VN 4 "", VN 0 "", VN 1 "", VN 3 ""] <= V [VN 4 "", VN 1 "",
            ↪  VN 2 ""] @?= True,
```

```
43          testCase "802.11 <= 802.11n" $ V [VN 802 "", VN 11 ""] <= V [VN 802
            ↪  "", VN 11 "n"] @?= True,
44          testCase "802.11n <= 802.11ax" $ V [VN 802 "", VN 11 "n"] <= V [VN
            ↪  802 "", VN 11 "ax"] @?= True,
45          testCase "802.11ax <= 802.11bb" $ V [VN 802 "", VN 11 "ax"] <= V [VN
            ↪  802 "", VN 11 "bb"] @?= True,
46          -- Merge Constraints
47          testCase "Merge 2 Empty Lists" $ merge [] [] @?= Just [],
48          testCase "Merge non empty and empty List" $ merge
49              [(P "Test", (False, V [VN 0 ""] , V [VN 1 ""] ))] [] @?=
50              Just [(P "Test", (False, V [VN 0 ""] , V [VN 1 ""] ))],
51          testCase "Merge 2 non empty Lists" $ merge
52              [(P "Test", (False, V [VN 0 ""] , V [VN 1 ""] ))] [(P "Test",
                ↪  (False, V [VN 0 ""] , V [VN 1 ""] ))] @?=
53              Just [(P "Test",(False,V [VN 0 ""],V [VN 1 ""]))],
54          testCase "Merge True and False" $ merge
55              [(P "Test", (True, V [VN 2 ""] , V [VN 8 ""] ))] [(P "Test",
                ↪  (False, V [VN 4 ""] , V [VN 6 ""] ))] @?=
56              Just [(P "Test",(True,V [VN 4 ""],V [VN 6 ""]))],
57          testCase "Merge True and False 2nd example" $ merge
58              [(P "Test", (True, V [VN 4 ""] , V [VN 6 ""] ))] [(P "Test",
                ↪  (False, V [VN 3 ""] , V [VN 8 ""] ))] @?=
59              Just [(P "Test",(True,V [VN 4 ""],V [VN 6 ""]))],
60          testCase "Merge False and False" $ merge
61              [(P "Test", (False, V [VN 4 ""] , V [VN 6 ""] ))] [(P "Test",
                ↪  (False, V [VN 3 ""] , V [VN 8 ""] ))] @?=
62              Just [(P "Test",(False,V [VN 4 ""],V [VN 6 ""]))],
63          testCase "Merge False and True" $ merge
64              [(P "Test", (False, V [VN 4 ""] , V [VN 6 ""] ))] [(P "Test",
                ↪  (True, V [VN 3 ""] , V [VN 8 ""] ))] @?=
65              Just [(P "Test",(True,V [VN 4 ""],V [VN 6 ""]))],
66          testCase "Merge Many Constrints example" $ merge
67              [(P "Test", (True, V [VN 4 ""] , V [VN 6 ""] )),  (P "Test2",
                ↪  (False, V [VN 3 "a"], V [VN 9 ""])),
68               (P "Test3", (False, V [VN 1 ""], V [VN 10 ""]))]
69              [(P "Test", (False, V [VN 3 ""] , V [VN 8 ""] )), (P "Test2",
                ↪  (False, V [VN 3 "z"], V [VN 7 ""]))] @?=
70              Just [(P "Test",(True,V [VN 4 ""],V [VN 6 ""])),(P
                ↪  "Test2",(False,V [VN 3 "z"],V [VN 7 ""])),(P "Test3",
                ↪  (False, V [VN 1 ""], V [VN 10 ""]))],
71          testCase "Merge same Version" $ merge
72                      [(P "Test", (False, V [VN 1 ""] , V [VN 1 ""] ))] [] @?=
73                      Just [(P "Test", (False, V [VN 1 ""] , V [VN 1 ""] ))]
74      ]
75
76
77  parser = testGroup "parser"
78      [
79          testCase "parse 3 packages with names" $
```

```
80              parseDatabase "package {name foo}package {name foo}package
                ↪  {name foo}" @?=
81              Right db1,
82          testCase "parse package with name and description" $
83              parseDatabase "package {name foo;description \"test\"}" @?=
84              Right db2,
85          testCase "parse package with name and description" $
86              parseDatabase "package {name foo;description \"test\"}" @?=
87              Right db2,
88          testCase "parse package with name, description, version" $
89              parseDatabase "package {name foo; version 1.2; description
                ↪  \"test\"}" @?=
90              Right db3,
91          testCase "parse package with name, description, version and string"
            ↪  $
92              parseDatabase "package {name foo; version 1.2a; description
                ↪  \"test\"}" @?=
93              Right db4,
94          testCase "longer Version" $
95              parseDatabase "package {name foo; version 1a.2a.45;
                ↪  description \"test\"}" @?=
96              Right db5,
97          -- pName hyphen, end hyphen also allowed
98          testCase "Package name hypens" $
99              parseDatabase "package {name 123-wewe-RR-}" @?=
100             Left "\"Parse Error\" (line 1, column 27):\nunexpected
                ↪  \"}\"\nexpecting letter, digit, \"-\" or \"\\\"\"",
101         testCase "Package name strings" $
102             parseDatabase "package {name \"123-wewe-RR\"}" @?=
103             Right (DB [Pkg (P "123-wewe-RR") (V [VN 1 ""])  "" []]),
104         testCase "Double High comma equals 1 highcomma" $
105             parseDatabase "package {name \"123\"\"\"}" @?=
106             Right (DB [Pkg {name = P "123\"", ver = V [VN 1 ""], desc =
                ↪  "", deps = []}]),
107         testCase "Double High comma equals 1 highcomma desc" $
108             parseDatabase "package {name \"123\"; description \"\"\"\"}"
                ↪  @?=
109             Right (DB [Pkg {name = P "123", ver = V [VN 1 ""], desc =
                ↪  "\"", deps = []}]),
110         -- Case doesn't matter for keywords
111         testCase "Case insensitiveness" $
112             parseDatabase "pAckAgE {nAmE foo; vErSiOn 1a.2a.45;
                ↪  deSCripTion \"test\"}" @?=
113             Right db5,
114         -- Dependencies Tests
115         testCase "Deps conflicts and requires" $
116             parseDatabase "package {name foo2; version 1a.2a.45;
                ↪  description \"test\"; requires foo < 2}" @?= --requires
                ↪  foo < 1.2 , foo >= 3;
```

```
117                 Right (DB [Pkg {name = P "foo2", ver = V [VN 1 "a",VN 2
                    ↪  "a",VN 45 ""],
118                 desc = "test", deps = [(P "foo",(True,V [VN 0 ""],V [VN 2
                    ↪  ""]))]}]),
119             testCase "Deps requires range overwrite" $
120                 parseDatabase "package {name foo2; requires foo < 3 , foo >=
                    ↪  8.0.0}" @?=
121                 Right (DB [Pkg {name = P "foo2", ver = V [VN 1 ""], desc =
                    ↪  "",
122                 deps = [(P "foo",(True,V [VN 3 ""],V [VN 8 "",VN 0 "",VN 0
                    ↪  ""]))]}]),
123             testCase "Deps self referential" $
124                 parseDatabase "package {name foo; requires foo < 3 , foo >=
                    ↪  8.0.0}" @?=
125                 Right (DB [Pkg {name = P "foo", ver = V [VN 1 ""], desc =
                    ↪  "", deps = []}]),
126             testCase "Deps requires fixed range" $
127                     parseDatabase "package {name foo2; requires foo < 3,
                        ↪  foo >= 8.0.0a}" @?=
128                     Right (DB [Pkg {name = P "foo2", ver = V [VN 1 ""],
                        ↪  desc = "",
129                     deps = [(P "foo",(True,V [VN 3 ""],V [VN 8 "",VN 0
                        ↪  "",VN 0 "a"]))]}]),
130             testCase "Deps requires fixed range requires and conflicts" $
131                      parseDatabase "package {name foo2; requires foo < 3
                         ↪  , foo >= 8.0.0a; conflicts bar < 3 , bar >= 8}"
                         ↪  @?=
132                      Right (DB [Pkg {name = P "foo2", ver = V [VN 1 ""],
                         ↪  desc = "",
133                      deps = [(P "foo",(True,V [VN 3 ""],V [VN 8 "",VN 0
                         ↪  "",VN 0 "a"])),
134                      (P "bar",(False,V [VN 3 ""],V [VN 8 ""]))]}]),
135             testCase "Deps different package names" $
136                      parseDatabase "package {name foo2; requires foo <
                         ↪  3, bar >= 8.0.0a; conflicts bar < 3 , foo >=
                         ↪  8}" @?=
137                      Right (DB [Pkg {name = P "foo2", ver = V [VN 1 ""],
                         ↪  desc = "",
138                      deps = [(P "foo",(True,V [VN 3 ""],V [VN 8 ""])),
139                      (P "bar",(True,V [VN 3 ""],V [VN 8 "",VN 0 "",VN 0
                         ↪  "a"]))]}]),
140             -- doesn't work to change the lower, greater equal
141             testCase "Low/High changed" $
142                      parseDatabase "package {name foo2; requires foo >=3
                         ↪  , bar < 8.0.0;}" @?=
143                      Left "\"Parse Error\" (line 1, column
                         ↪  38):\nunexpected \",\"\nexpecting space,
                         ↪  \"--\", white space or \"}\"",
144              -- Whitespace and other more special things
```

```
145         testCase "whitespaces pkg and name" $
146         parseDatabase "package     {name        foo2;       requires
            ↪  foo     <      3    ,      foo    >=      8.0.0a; conflicts bar
            ↪  < 3 , bar >= 8   }" @?=
147             Right (DB [Pkg {name = P "foo2", ver = V [VN 1 ""], desc = "",
148                                deps = [(P "foo",(True,V [VN 3 ""],V
                                   ↪  [VN 8 "",VN 0 "",VN 0 "a"])),
149                                (P "bar",(False,V [VN 3 ""],V [VN 8
                                   ↪  ""]))]}]),
150         -- Comment parsing
151         testCase "Comment parsing" $
152         parseDatabase "   --comment\npackage {name --comment\n foo;
            ↪  --comment\ndescription \"test\" --comment\n}" @?=
153         Right (DB [Pkg {name = P "foo", ver = V [VN 1 ""], desc = "test",
            ↪  deps = []}]),
154         testCase "Comment name" $
155         parseDatabase "package {name \"fo--o\"; description \"te--st\"}" @?=
156         Right (DB [Pkg {name = P "fo--o", ver = V [VN 1 ""], desc =
            ↪  "te--st", deps = []}]),
157         -- failing to parse, comment after package
158         testCase "Comment after package" $
159         parseDatabase "package {name \"foo\"; description \"test\"}
            ↪  --comment" @?=
160         Left "\"Parse Error\" (line 1, column 51):\nunexpected end of
            ↪  input\nexpecting lf new-line, end of input, white space or
            ↪  \"package\"",
161         -- Wrong Version Number
162         testCase "parse too large Version" $
163         parseDatabase "package {name \"foo\"; version 1000000}" @?=
164         Left "\"Parse Error\" (line 1, column 22):\nunexpected
            ↪  \"v\"\nexpecting space, \"--\", white space or \"}\"",
165         testCase "Edge parsable" $
166         parseDatabase "package {name \"foo\"; version 999999aaaa}" @?=
167         Right (DB [Pkg {name = P "foo", ver = V [VN 999999 "aaaa"], desc =
            ↪  "", deps = []}]),
168         -- Z gets ignored, since not lowercase
169         testCase "Edge parsable" $
170         parseDatabase "package {name \"foo\"; version 999999Z}" @?=
171         Right (DB [Pkg {name = P "foo", ver = V [VN 999999 ""], desc = "",
            ↪  deps = []}]),
172         testCase "Version String too long" $
173         parseDatabase "package {name \"foo\"; version 9999aaaaa}" @?=
174         Left "\"Parse Error\" (line 1, column 22):\nunexpected
            ↪  \"v\"\nexpecting space, \"--\", white space or \"}\""
175       ]
176        where
177         ver = V [VN 1 ""]
178         pname = P "foo"
179         pname2 = P "foo2"
```

```
180         pkg = Pkg pname ver  "" []
181         db1 = DB [pkg,pkg,pkg]
182         pkg2 = Pkg pname ver  "test" []
183         db2 = DB [pkg2]
184         ver2 = V [VN 1 "", VN 2 ""]
185         pkg3 = Pkg pname ver2 "test" []
186         db3 = DB [pkg3]
187         ver3 = V [VN 1 "", VN 2 "a"]
188         pkg4 = Pkg pname ver3 "test" []
189         db4 = DB [pkg4]
190         ver4 = V [VN 1 "a", VN 2 "a", VN 45 ""]
191         pkg5 = Pkg pname ver4 "test" []
192         db5 = DB [pkg5]
193
194 -- Parser Example
195 example = testGroup "Example DB" [
196     testCase "Parse Example DB" $ parseDatabase "package { name foo; version
        ↪  2.3; description \"The foo application\"; requires bar >= 1.0}
        ↪  package { name bar; version 1.0; description \"The bar library\"}
        ↪  package { name bar; version 2.1; description \"The bar library, new
        ↪  API\";  conflicts baz < 3.4, baz >= 5.0.3} package { name baz;
        ↪  version 6.1.2;}"
197     @?=  Right (DB [Pkg {name = P "foo", ver = V [VN 2 "",VN 3 ""],
198             desc = "The foo application",
199             deps = [(P "bar",(True,V [VN 1 "",VN 0 ""],V [VN 1000000
                ↪  ""]))]},
200         Pkg {name = P "bar", ver = V [VN 1 "",VN 0 ""],
201             desc = "The bar library", deps = []},
202         Pkg {name = P "bar", ver = V [VN 2 "",VN 1 ""],
203             desc = "The bar library, new API",
204             deps = [(P "baz",(False,V [VN 3 "",VN 4 ""],V [VN 5 "",VN 0
                ↪  "",VN 3 ""]))]},
205         Pkg {name = P "baz", ver = V [VN 6 "",VN 1 "",VN 2 ""], desc = "",
            ↪  deps = []}])
206     ]
207
208 normalizeTests = testGroup "Normalize" [
209         testCase "empty Database" $ normalize (DB []) @?= Right (DB [])
210
211 -- DB [Pkg {name = P "foo", ver = V [VN 2 "",VN 3 ""],
212 -- desc = "The foo application",
213 -- deps = [(P "bar",(True,V [VN 1 "",VN 0 ""],V [VN 1000000 ""]))]}]
214     ]
215
216 -- just a sample; feel free to replace with your own structure
217 predefined = testGroup "predefined"
218   [testGroup "Parser tests"
219      [testCase "tiny" $
```

```
220         parseDatabase "package {name foo}package {name foo}package {name
            ↪  foo}" @?= Right db],
221     testGroup "Solver tests"
222       [testCase "tiny" $
223         install db pname @?= Just [(pname, ver)] ] ]
224   where
225     pname = P "foo"
226     ver = V [VN 1 ""]
227     db = DB [Pkg pname ver "" []]
228
229 main = defaultMain tests
```

B.2 Question 2.1

```
1  -module(district_tests).
2  -author("silvan").
3  -include_lib("eunit/include/eunit.hrl").
4
5  district_create_test() ->
6    ?assertMatch({ok, _}, district:create("Panem")).
7
8  district_get_description_test() ->
9    {ok, P} = district:create("Panem"),
10   ?assertEqual({ok, "Panem"}, district:get_description(P)),
11   district:activate(P),
12   ?assertEqual({ok, "Panem"}, district:get_description(P)).
13
14 district_connect_districts_test() ->
15   {A, B, C} = create_districts(),
16
17   ?assertEqual(ok, district:connect(A, b, B)),
18   district:connect(A, c, C),
19   % Action c already exists in A
20   ?assertEqual(active, district:activate(A)),
21   ?assertMatch({error, _}, district:connect(A, c, C)).
22
23 district_connect2_districts_test() ->
24   {A, B, C} = create_districts(),
25
26   ?assertEqual(ok, district:connect(A, b, B)),
27   district:shutdown(C, self()),
28   %trying to connect to a terminated district
29   ?assertMatch({error, _}, district:connect(A, c, C)).
30
31 district_active_test() ->
32   {A, B, C} = create_districts(),
33
```

```
34    district:connect(A, c, C),
35    district:shutdown(C, self()),
36    % Process C not alive anymore, so A can't be activated
37    ?assertEqual(false, is_process_alive(C)),
38    ?assertEqual(impossible, district:activate(A)),
39    % B doesn't have any neighbors, so easily to be activated
40    ?assertEqual(active, district:activate(B)).
41
42  district_active2_test() ->
43    {A, _, C} = create_districts(),
44
45    district:connect(A, c, C),
46    % Activate C already, activate A later
47    ?assertEqual(active, district:activate(C)),
48    ?assertEqual(active, district:activate(A)).
49
50  district_options_test() ->
51    {A, B, C} = create_districts(),
52
53    district:connect(A, b, B),
54    district:connect(A, c, C),
55
56    ?assertEqual({ok, [b, c]}, district:options(A)),
57    ?assertEqual({ok, []}, district:options(B)),
58    ?assertEqual({ok, []}, district:options(C)).
59
60  district_enter_test() ->
61    {A, B, C} = create_districts(),
62
63    district:connect(A, b, B),
64    district:connect(A, c, C),
65
66    Bob = {make_ref(), #{}},
67    % only can enter if district active
68    ?assertMatch({error, _}, district:enter(A, Bob)),
69    district:activate(A),
70    ?assertEqual(ok, district:enter(A, Bob)).
71
72  dsitrict_take_action_test() ->
73    {A, B, C} = create_districts(),
74
75    district:connect(A, b, B),
76    district:connect(A, c, C),
77
78    {KatnissRef, _} = Katniss = {make_ref(), #{}},
79    {PeetaRef, _} = {make_ref(), #{}},
80    district:activate(A),
81    ?assertEqual(ok, district:enter(A, Katniss)),
82    %Action doesn't exist
```

```
83    ?assertMatch({error, _}, district:take_action(A, KatnissRef, d)),
84    % Katniss stays in A
85    ?assertMatch({error, _}, district:enter(A, Katniss)),
86    %Creature hasn't joined A District
87    ?assertMatch({error, _}, district:take_action(A, PeetaRef, b)),
88    ?assertMatch({ok, _}, district:take_action(A, KatnissRef, b)),
89    % Katniss now not in District A anymore
90    ?assertEqual(ok, district:enter(A, Katniss)),
91    % But now in district B
92    ?assertMatch({error, _}, district:enter(B, Katniss)),
93    %try to move Katniss by action again to district begin
94    ?assertMatch({error, _}, district:take_action(A, KatnissRef, b)),
95    district:shutdown(B, self()),
96    ?assertMatch({error, _}, district:take_action(A, KatnissRef, b)),
97    %therefore Katniss is still in A
98    ?assertMatch({error, _}, district:enter(A, Katniss)).
99
100 district_shutdown_test() ->
101   {A, B, C} = create_districts(),
102
103   % Process is available
104   ?assertEqual(true, is_process_alive(A)),
105   ?assertEqual(true, is_process_alive(B)),
106   ?assertEqual(true, is_process_alive(C)),
107   district:connect(A, b, B),
108   district:connect(A, c, C),
109
110   ?assertEqual(ok, district:shutdown(A, self())),
111   % after shutdown undefined
112   ?assertEqual(false, is_process_alive(A)),
113   ?assertEqual(false, is_process_alive(B)),
114   ?assertEqual(false, is_process_alive(C)).
115
116 district_shutdown2_test() ->
117   {A, B, C} = create_districts(),
118
119   % Process is available
120   ?assertEqual(true, is_process_alive(A)),
121   ?assertEqual(true, is_process_alive(B)),
122   ?assertEqual(true, is_process_alive(C)),
123   district:connect(A, b, B),
124   district:connect(A, c, C),
125
126   ?assertEqual(ok, district:shutdown(B, self())),
127   % after shutdown undefined
128   ?assertEqual(true, is_process_alive(A)),
129   ?assertEqual(false, is_process_alive(B)),
130   ?assertEqual(true, is_process_alive(C)),
131   %since B already shutdown, no need to send it a shutdown message anymore
```

```
132     ?assertEqual(ok, district:shutdown(A, self())),
133     %every district should be shutdown now
134     ?assertEqual(false, is_process_alive(A)),
135     ?assertEqual(false, is_process_alive(B)),
136     ?assertEqual(false, is_process_alive(C)).
137
138   district_shutdown_cycle_test() ->
139     {A, B, _} = create_districts(),
140
141     district:connect(A, b, B),
142     district:connect(B, a, A),
143     district:connect(A,a,A),
144     %times out since cycle exists
145     district:shutdown(A,self()),
146     ?assertEqual(false, is_process_alive(A)),
147     ?assertEqual(false, is_process_alive(B)).
148
149   district_active_cycle_test() ->
150     {A, B, C} = create_districts(),
151
152     district:connect(A, b, B),
153     district:connect(B, a, A),
154     district:connect(B, c, C),
155     district:connect(C, c, C),
156     district:activate(A),
157     {Ref, _} = Katniss = {make_ref(), #{}},
158     % all connected districts get active
159     ?assertMatch(ok, district:enter(C, Katniss)),
160     district:take_action(C,Ref,c).
161
162   increment_grade(_, {CreatureRef, Stats}, Creatures) ->
163     #{grade := CurGrade} = Stats,
164     NewGrade = CurGrade + 4,
165     case NewGrade of
166       12 -> get_grade(CreatureRef, Stats, 12, happy, Creatures);
167       7 -> get_grade(CreatureRef, Stats, 7, okay, Creatures);
168       2 ->  get_grade(CreatureRef, Stats, 2, okay, Creatures);
169       Grade -> get_grade(CreatureRef, Stats, Grade, sad, Creatures)
170     end.
171
172   get_grade(Ref, Stats, Grade, Mood, Creatures) ->
173     {{Ref, Stats#{grade := Grade,mood:= Mood}}, Creatures}.
174
175   district_trigger_test() ->
176     {A, B, C} = create_districts(),
177
178     district:connect(A, b, B),
179     district:connect(A, c, C),
180     district:connect(C, a, A),
```

```
181     district:connect(B, a, A),
182
183     district:trigger(A, fun increment_grade/3),
184     district:activate(A),
185     {Ref, _Stats} = Silvan = {make_ref(), #{grade => 0, mood => sad}},
186     district:enter(A, Silvan),
187     ?assertMatch({ok, _}, district:take_action(A, Ref, b)),
188     ?assertMatch({ok, _},district:take_action(B, Ref, a)),
189     ?assertMatch({ok, _}, district:take_action(A, Ref, b)),
190     ?assertMatch({ok, _}, district:take_action(B, Ref, a)),
191     ?assertMatch({ok, _}, district:get_description(B)),
192     %Moved Silvan 4 times between A and B
193     ?assertMatch({error,_},district:enter(A,Silvan)).
194
195
196 create_districts() ->
197     {ok, A} = district:create("A"),
198     {ok, B} = district:create("B"),
199     {ok, C} = district:create("C"),
200     {A, B, C}.
```

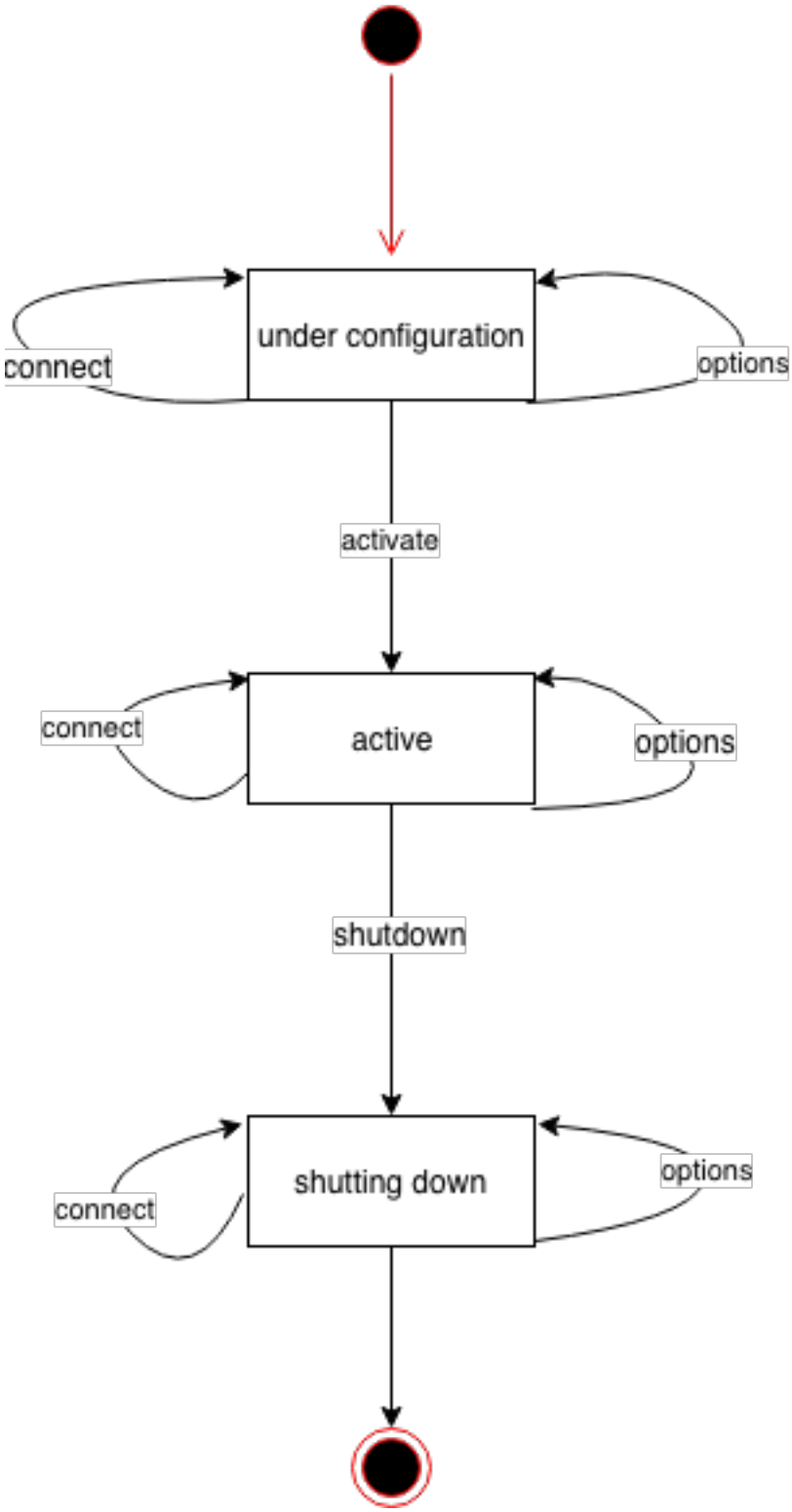

Figure 1: Simple State machine diagramm