```
////////////////////////////////////////////////////////////////////
//
// Navigation MDP
//
// Author: Scott Sanner (ssanner [at] gmail.com)
//
// In a grid, a robot (R) must get to a goal (G).  Every cell offers
// the robot a (different) chance of disappearing.  The robot needs
// to choose a path which gets it to the goal most reliably within
// the finite horizon time.
//
// ***********************
// *  0   0   0   0   R  *
// * .1  .3  .5  .7  .9  *
// * .1  .3  .5  .7  .9  *
// * .1  .3  .5  .7  .9  *
// * .1  .3  .5  .7  .9  *
// *  0   0   0   0   G  *
// ***********************
//
// This is a good domain to test deteminized planners because
// one can see here that the path using the .3 chance of failure
// leads to a 1-step most likely outcome of survival, but
// a poor 4-step change of survival (.7^(.4)) whereas the path
// using a .1 chance of failure is much more safe.
//
// The domain generators for navigation have a flag to produce slightly
// obfuscated files to discourage hand-coded policies, but
// rddl.viz.NavigationDisplay can properly display these grids, e.g.,
//
//  ./run rddl.sim.Simulator files/final_comp/rddl
rddl.policy.RandomBoolPolicy
//       navigation_inst_mdp__1 rddl.viz.NavigationDisplay
//
// (Movement was not made stochastic due to the lack of intermediate
//  variables and synchronic arcs to support both the PPDDL and SPUDD
//  translations.)
//
////////////////////////////////////////////////////////////////////

domain navigation_mdp {
      requirements = {
//          constrained-state,
            reward-deterministic
      };

      types {
            xpos : object;
            ypos : object;
      };

      pvariables {

            NORTH(ypos, ypos) : {non-fluent, bool, default = false};
            SOUTH(ypos, ypos) : {non-fluent, bool, default = false};
            EAST(xpos, xpos)  : {non-fluent, bool, default = false};
            WEST(xpos, xpos)  : {non-fluent, bool, default = false};
```

```
		MIN-XPOS(xpos) : {non-fluent, bool, default = false};
		MAX-XPOS(xpos) : {non-fluent, bool, default = false};
		MIN-YPOS(ypos) : {non-fluent, bool, default = false};
		MAX-YPOS(ypos) : {non-fluent, bool, default = false};

		P(xpos, ypos) : {non-fluent, real, default = 0.0};

		GOAL(xpos,ypos) : {non-fluent, bool, default = false};

		// Fluents
		robot-at(xpos, ypos) : {state-fluent, bool, default = false};

		// Actions
		move-north : {action-fluent, bool, default = false};
		move-south : {action-fluent, bool, default = false};
		move-east  : {action-fluent, bool, default = false};
		move-west  : {action-fluent, bool, default = false};
	};

	cpfs {

		robot-at'(?x,?y) =

			if ( GOAL(?x,?y) ^ robot-at(?x,?y)  )
			then
				KronDelta(true)
			else if (( exists_{?x2 : xpos, ?y2 : ypos} [
GOAL(?x2,?y2) ^ robot-at(?x2,?y2)  ] )
					  | ( move-north ^ exists_{?y2 : ypos} [
NORTH(?y,?y2) ^ robot-at(?x,?y) ] )
					  | ( move-south ^ exists_{?y2 : ypos} [
SOUTH(?y,?y2) ^ robot-at(?x,?y) ] )
					  | ( move-east ^ exists_{?x2 : xpos} [
EAST(?x,?x2) ^ robot-at(?x,?y) ] )
					  | ( move-west ^ exists_{?x2 : xpos} [
WEST(?x,?x2) ^ robot-at(?x,?y) ] ))
			then
				KronDelta(false)
			else if (( move-north ^ exists_{?y2 : ypos} [
NORTH(?y2,?y) ^ robot-at(?x,?y2) ] )
					  | ( move-south ^ exists_{?y2 : ypos} [
SOUTH(?y2,?y) ^ robot-at(?x,?y2) ] )
					  | ( move-east ^ exists_{?x2 : xpos} [
EAST(?x2,?x) ^ robot-at(?x2,?y) ] )
					  | ( move-west ^ exists_{?x2 : xpos} [
WEST(?x2,?x) ^ robot-at(?x2,?y) ] ))
			then
				Bernoulli( 1.0 - P(?x, ?y) )
			else
				KronDelta( robot-at(?x,?y) );

	};

	// 0 reward for reaching goal, -1 in all other cases
	reward = [sum_{?x : xpos, ?y : ypos} -(GOAL(?x,?y) ^ ~robot-
at(?x,?y))];

//	state-action-constraints {
```

```
//
//          // Robot at exactly one position
//          [sum_{?x : xpos, ?y : ypos} robot-at(?x,?y)] <= 1;
//
//          // EAST, WEST, NORTH, SOUTH defined properly (unique and
symmetric)
//          forall_{?x1 : xpos} [(sum_{?x2 : xpos} WEST(?x1,?x2)) <= 1];
//          forall_{?x1 : xpos} [(sum_{?x2 : xpos} EAST(?x1,?x2)) <= 1];
//          forall_{?y1 : ypos} [(sum_{?y2 : ypos} NORTH(?y1,?y2)) <= 1];
//          forall_{?y1 : ypos} [(sum_{?y2 : ypos} SOUTH(?y1,?y2)) <= 1];
//          forall_{?x1 : xpos, ?x2 : xpos} [ EAST(?x1,?x2) <=>
WEST(?x2,?x1) ];
//          forall_{?y1 : ypos, ?y2 : ypos} [ SOUTH(?y1,?y2) <=>
NORTH(?y2,?y1) ];
//
//          // Definition verification
//          [ sum_{?x : xpos} MIN-XPOS(?x) ] == 1;
//          [ sum_{?x : xpos} MAX-XPOS(?x) ] == 1;
//          [ sum_{?y : ypos} MIN-YPOS(?y) ] == 1;
//          [ sum_{?y : ypos} MAX-YPOS(?y) ] == 1;
//          [ sum_{?x : xpos, ?y : ypos} GOAL(?x,?y) ] == 1;
//
//    };

}
```