P = Patient
R = radio
Sc = Scream
S = static
M = Medic (Agent)
C = Cure
CP = Cured patient

¬Z1,1 #There is no zombie on the starting square
¬R1,1 #There is no radio on the starting square

Sc1,1 ⇔ (P1,2 ∨P2,1), Sc1,2... #If there is a scream on 1,1, there can be a zombie on 1,2 or 2,1, etc
S1,1 ⇔ (R1,2 ∨ R2,1), S1,2... #If there is a static on 1,1, there can be a radio on 1,2 or 2,1, etc
(P1,1 ^ P1,2) ∨ (P1,1 ^ P1,3) v ⋯ ∨ (P4,4 ^ P4,3) #There is at least 2 zombies

{(¬(P1,1 ^ P1,2) ∨ ¬(P1,1 ^ Z1,3)) , (¬(P1,1 ^ Z1,2) ∨ ¬(P1,1 ^ Z1,4)), ⋯ ¬(P4,4 ^ Z4,3) ∨ ¬(P1,1 ^ P1,3)) } #There at most 2 zombies

#If there are 2 cured zombies, the game is over and won
((CP1,1 ^ CP1,2) ∨ (CP1,1 ^ CP1,3)) , (CP1,1 ^ Z1,2) ∨ (CP1,1 ^ CP1,4), ⋯ ¬(CP4,4 ^ CP4,3) ∨ ¬(CP1,1 ^ CP1,3)): Endgame(Win)
((P1,1 ^ M1,1) v ((P1,2 ^ M1,2) v ... v ((P4,4 ^ M4,4)): Endgame(Lose) #If the agent player and zombie are on the same location, the game is over and lost

Als de agent op een radio bevind, krijgt het te horen waar een cure zich bevind.
(R2,3 ^ M2,3): KB(Cx,y)

Wanneer de agent op een vak loopt krijgt het ook natuurlijk meteen alle informatie van de locatie waar het zich nu bevindt.
M2,2: KB(R2,3) v KB(¬R2,3) v KB(C2,3) v KB(¬P2,3)...

Nu moeten we de agent informatie laten verkrijgen via informatie dat hij al heeft. Zoals dat als er op genoeg plekken een growl of static gehoord kan worden dat hij kan uitmaken dat er zeker op een plek radio's of zombies zijn.

(Sc1,2 ^ Sc2,1) ⇔ (P1,1 v P2,2) #als er een growl op 1,2 en 2,1 is dan is er een zombie op 1,1 of 2,2 of beide.
((Sc1,2 ^ Sc2,1) ⇔ (P1,1 v P2,2)) ^ (¬Sc3,2 v ¬P2,2): KB(P1,1) #Als er dus een zombie op 1,1 of 2,2 is maar hoort geen growl op 3,2 of weet al dat er geen zombie op 2,2 is dan moet er dus een zombie op 1,1 zitten.

Deze zelfde regels gaan voor radios en static events.
(S1,2 ^ S2,1) ⇔ (R1,1 v R2,2)
((S1,2 ^ S2,1) ⇔ (R1,1 v R2,2)) ^ (¬S3,2 v ¬R2,2): KB(R1,1)

Gezien er meerder zombies zijn moeten we werken met kansberekeningen. Als er bijvoorbeeld een growl word gehoord op 2,2 en 4,2 kan er 1 zombie zijn op 3,2, of 2 zombies op 2,1 en 4,1. Zodra dit het geval is moeten we kijken naar alle mogelijkheden.

Sc2,2 ⇔ (P2,1 v P3,2 v P2,3 v P1,2 v (P2,1 ^ P3,2) v (P2,3 ^ P1,2)...)

Hier word dan bekeken waar hoevaak een locatie voorkomt in de mogelijkheden, voor 1 growl is alles nog 25%

:KB(P2,1-25%, P3,2-25%, P2,3-25%, P1,2-25%)

Maar met een tweede kan er verschil komen in hoevaak een locatie voorkomt, hier wordt dan de kansen berekent met de aantal locaties gedeeld door de aantal mogelijkheden.

Als we de situatie nemen op het volgende bord:

X	G		G

Op dit bord zien we dat we twee growls in de agents knowledgebase hebben (G2,2; G4,2), weten dat er 2 zombies op het veld zijn, maar voor de rest niks over de situatie weten.

In dit geval staat de agent op 2,2 en wil hij weten of het veilig is om te bewegen. Dit wordt gedaan met een kansberekening, en deze wordt uitgevoerd voor alle mogelijke richtingen die de agent op kan (In dit voorbeeld doen wij dit alleen voor 1,2).

In deze kansberekening gebruiken we alle data die wij in ons knowledgebase hebben, wat de locatie van 2 growls, en het weten dat er 2 zombies zijn is.

Om de kans uit te rekenen moeten we eerst weten wat de kans is dat er zombie op de locatie is, en wat de kans is dat deze er niet is. Dit wordt gedaan door elke mogelijkheid van de frontier (het onderdeel van het grid dat wordt gebruikt voor alleen het berekenen) uit te rekenen voor elke mogelijkheid van de query (het onderdeel van de grid waar we de kans voor willen hebben). Voor onze berekening zijn er 6 verbonden vakken in het frontier en de query, die elk een 1/8 kans hebben om een zombie te zijn, met een max van 2 zombies.

Voor de berekening zelf moet de agent eerst weten wat alle mogelijkheden zijn voor elke query. Dit is 64 (2^6) in totaal en 32 per query. Na het filteren van geldige mogelijkheden

(minstens 1 zombie per growl, max 2 zombies) kom je uit op: 9 voor de query met zombie en 3 voor die zonder.

De volgende stap is het vermenigvuldigen van de kans van elk vak in de frontier (geen zombie 7/8, wel een zombie 1/8), deze op te tellen van elke mogelijkheid, en deze ten slot te vermenigvuldigen met de query. Dit geeft in ons geval:

7/8 * (0.07 + 0.01 + 0.01 + 0.01 + 0.01 + 0.01 + 0.01 + 0.01 + 0.01) = 0.13125

1/8 * (0.01 + 0.01 + 0.01) = 0.00375

We hebben nu bijna het antwoord, alleen we moeten de getallen eerst met elkaar normaliseren zodat ze samen 1 worden. Dit geeft een kans van 0,03 voor een zombie op vak 1,2; en 0,97 voor geen zombie op vak 1,2.

In dit geval is dit nummer best hoog, en dit komt omdat onze situatie heel gunstig is maar eigenlijk nooit voorkomt. De agent zal hier deze kans vergelijken met de kansen van de andere vakken die hij naartoe kan, en dan naar het vak bewegen met de grootste kans op geen zombie.

Hierna moeten we gaan kijken naar alle mogelijke actions die de agent kan ondernemen. Eerst kijken we naar alleen bewegen. De agent moet natuurlijk alleen niet op een vak bewegen als er zekerweten een zombie op bevindt

P2,2: ¬Move(M2,2)

Daarna moet het ook een reden hebben om ergens wel op te bewegen. Als er meerder keuzes net zo goed zijn kiest het randomly uit de opties. Als er een radio zekerweten naast de agent is kan het altijd eerst het beste daar naartoe gaan.

R2,2: Move(M2,2)

Daarna kijkt het waar een cure is, als een cure boven en rechts van de agent bevind kan het het beste die kant op gaan.

C4,4 ^ (M[x] < C[x]): Move(right)
C4,4 ^ (M[y] < C[y]): Move(up)

Daarna kan het nog het geval zijn dat de agent zich verplaatst naar een regio waar het niet voorbij kan. Hier maken we dan gebruik van timesteps om te zien waar het eerder is geweest. Als het op een locatie inziet dat het terug moet kan de agent onthouden dat daar naartoe bewegen geen nut heeft.

M2,2 ^ Move(right)t ^ Move(left)t+1: KB(M2,2: ¬Move(right))

Dit kan herhaalt worden totdat de agent ver genoeg terug is om een correcte pad te kiezen, deze logische pad kan ook

berekent worden voordat agent ook daadwerkelijk zich beweegt door in zijn eigen hoofd het hele pad eerst te
simuleren. Dit kunnen we dan "Calculate_path(destination)" noemen.

C[x,y]: Calculate_path(C[x,y])
P[x,y] ^ M has cure: Calculate_path(P[x,y])
R[x,y]: Calculate_path(R[x,y])