Zombie Apocalypse in R - Pete Moore

Pete Moore - STUDENT ID: 170000983 - COURSE: AC52050 PROGRAMMING LANGUAGES

11/06/2018

## NULL

## Understanding

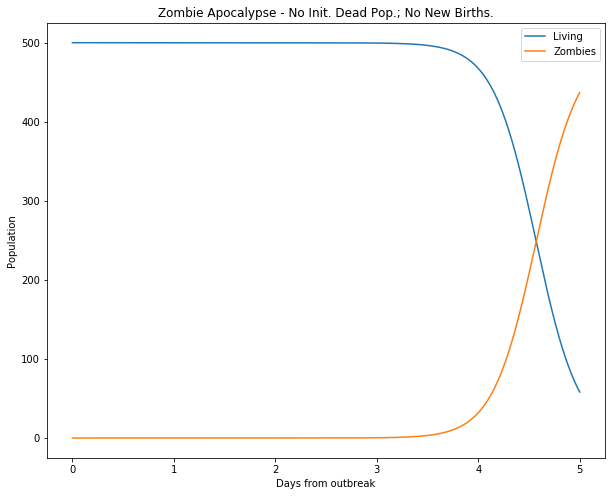
This is a study of epidemiology by analogy. The basis of the study is to model the interactions between zombies and humans. The interatction between humans and zombies has previously been modelled mathematically by the below ordinary differential equations (ODE’s), known as the SZR Model[[1]](#footnote-1):

S′ <- Π−βSZ−δS

Z′ <- βSZ+ζR−αSZ

R′ <- δS+αSZ−ζR

Where the following variables are used: Π - birth rate δ - natural death rate β - transmission rate ζ - resurrection rate α - zombie destruction rate And ′ signifies rate of change (e.g. S′ = dS/dt)

Solving these equations using the code in Appendix One [[2]](#footnote-2), produces the below curve: image:

##### Figure 1: Zombie conquest with (P=0,d=0.0001,B=0.0095,G=0.0001,a=0.0001)[[3]](#footnote-3)

This shows that even with nominal death and transmission rates, the zombies win. However this only tells half of the story because the SZR model does not allow for zombies or humans to be attributed a velocity. Once this happens these differential equations can no no longer be solved. This project investigates the effect of velocity on the SZR model.

### Goal

The goal of this project is to introduce velocity to the SZR model and, via a series of Monte Carlo Simulations (MCS); see how Zombies and humans interact as veloicty varies.

### Definitions

The following definitions form the essential components of the model. (Where subsequent definitions are made they will appear in **bold typeface**)

* Humanoid: a human or a zombie
* Susceptible: an uninfected Humanoid
* Zombification: a Susceptible Humanoid that is turned into a zombie
* Death: a Susceptible Humanoid who dies naturally
* Destruction: a zombie destroyed by a Humanoid
* Removed: a dead human, a destroyed zombie
* Resurrection: a Removed Humanoid that is turned into a zombie

## Exploratory Data Analysis

The original SZR model allows a solution because the equations are in balance, that is Susceptibles are added to by births and reduced by death and zombification, Zombies are added to by Zombification and Resurrection but depleted by Destruction. Finally, the Removed class are added to by Zombification and Death but reduced by resurrections.

All of these principles may be preserved for the MCS model, however, additional questions need to be asked about the ways in which Humanoids move. For example, what is the average speed of a Humanoid? How does it deviate from the mean? Is this normally distributed? Are the populations geographically distinct? Is the geography bounded?

The model outlined below takes these considerations into account under the following assumptions:

### Initial assumptions

* Different Species of Humanoids begin at geographically distinct locations
* Humanoids act as point particles. That is they may become arbitrarily near to one another (this allows the initial populations to start at the same grid co-ordinate per species)
* Humanoids move in **footsteps**, with one footstep taking place in one unit of time (or **timestep**)
* A **footpath** constitutes a set of footsteps over a series of time series, footpaths must be contiguous (i.e. no levitating!)
* The footstep lengths are normally distrubuted and share the same mean and standard deviation per Species (the **normal distribution assumption**)
* **Since each footstep length is mapped *per unit time* the mean footstep length is equivalent to a mean velocity**
* When Humanoids of different Species get within a critical distance of one another there is a consequence (a human gets zombified, a zombie gets destroyed). This is the **consequences assumption**.
* This is a zero-sum game: a human gets Zombified or a zombie gets destroyed. This is the **zero-sum assumption**.
* This **critical distance** is the same for both species
* Only two Humanoids can interact at any point in time (i.e. a zombie cannot kill two humans in a single timestep)
* Humanoids only have eyes for the nearest victim
* Zombies beat humans at the **Zombie Win Rate** a real number between 0 and 1[[4]](#footnote-4)
* Newborns move instantly as per the speed of the herd (e.g. they are carried)

## Data model

### Overview

As Humanoids take footsteps they tread out a footpath. The normal distribution assumption, means that Humanoids exhibit Brownian motion in the model. By adapting code for a single particle[[5]](#footnote-5), random walks in one- and two-dimensions were created (the code can be seen in appendix two[[6]](#footnote-6)). By sending each humanoid on a random walk through time, the different species of Humanoids would eventually get within critical distance of one another at which point either a Susceptible turns into a Zombie or the Zombie is destroyed. Aysconhrnously, Humans are dying, being born amd Zombies are bieng resurrected. Any or all of these could be happening at any given epoch, that is, as the model iterated through time the future path of a Humanoid needed constant addressing via reconfiguration routine. In order to aviod stochasitc interference, the process need to be ran over many repetitions. Finally, the model needed to be executed acorss scenarios, in particular, the scenario fo varying velocity was analysed in depth. The results were output to an OLAP cube in which the humanoid interacts could be analysed.

### Method for Interactions

#### Spatial plane

The humanoids movements are therefore centered around a mean and standard deviation and these have real world meanings. Namely, the standard deviation represents the footstep length of the Humanoid and the mean the direction of motion in a two-dimensional plane. This “Brownian motion code”[[7]](#footnote-7) allows the Humanoids to move in space and time.[[8]](#footnote-8) Zombies and Susceptibles start at geographically distinct locations (held in the variable Xt). That is, *all* of the Zombies start at the same grid co-ordinate, all of the Susceptibles at a different shared grid co-ordinate.

#### Time dimension

The model, then, places Humanoids in a two-dimensional vector space in the real numbers. The random walk takes place over a series of iterations called timesteps. At any given timestep the Zombies and Susceptibles will interact if they are near enough. By coalescing time as third dimension, the model now sits in a three-dimensional vector space.

#### Humanoid “dimension”

Technically Humanoids count as a fourth dimension, to the model however, since each Humanoid can only exist on one spatial co-ordinate at any given timestep, a natural principle component analysis presents itself. Each individual Humanoid is represented as a named row. The name of the row is subject to change as the populations interact.

#### Programtic vector space

Programatically, the three-dimensional vector-space constitutes an array and this is seen in the W variable output from the function rwalk2dhumanoids.

#### Laplacian determinism

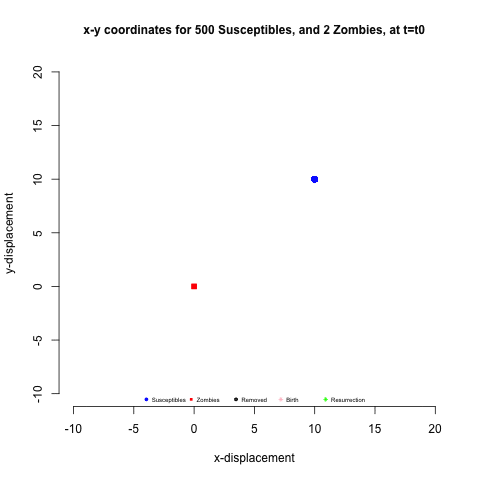
The random walks were programmed right at the beginning of our model. In the real world this has the effect of issuing each humanoid with a deterministic set of co-ordinates. Were there to be zero interactions, these would describe the position of the Humanoid at any point in eternity. (Fortunately they did indeed interact)

#### Interactions

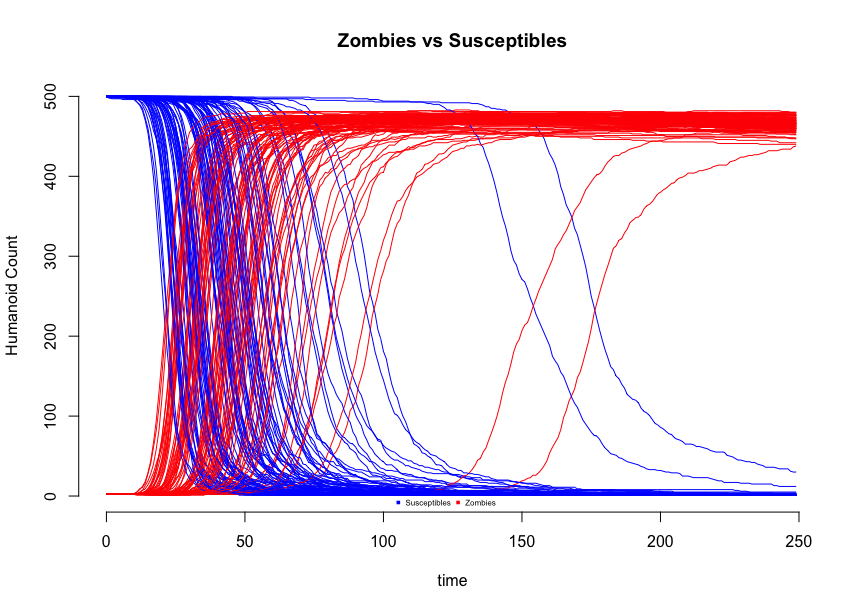
At each time step the routine *reconfigurepopulation* (see Appendix Four: population reconfiguration) was executed. The effect of this was to model the following: - Zombification of a Susceptible for Humanoids within critical distance of one another - Destruction of a Zombie for Humanoids within critical distance of one another - Natural deaths - Natural births - Resurrections Furthermore, this part of the model handles the “leftovers” from the interactions in the previous timestep. For example, if timestep t11 was a Zombification then that Zombie would become active on timestep t11. Each interaction necessitated a different functionality. For example, a natural death meant the Humanoid had to stop moving, programatically, this meant updating all their future x-y co-ordinates to thie last known postion as per the function *exhumeremoved*; conversely, a resurrection meant the opposite, a once stunned Humanoid needed a fresh set of co-ordinates with their initial position their last know, this was the function *reconfigure.revive*. (There are rather a lot of these functions and full glossary of functions can be found in: Appendix Five: All functions in Alphabetical order)

### Results for Interactions

#### Results: Data visualisation of interactions

The animation in figure 3[[9]](#footnote-9) visualises the net effect of the algorithm. The Zombies (bottom left) walk randomly as do the humans in the top left. In this case the mean is set to zero so the random walk leads to each Humanoid drifting to the square root of t (for timestep t)[[10]](#footnote-10). Since the direction of this is random, this inevitably leads to the two populations interacting. As soon as they are within a **critical distance** of one another the **consequences assumption** means that something will happen and the **zero-sum assumption** means this is bad news for one of the Humanoids. The animation shows the humans (red) being attacked and turning into zombies. And vice-versa (although this is rare because we have set a high **Zombie Win Rate**). Every so often we see a Resurrection occur (green), a natural human death (black) or (rarely) a natural birth (pink). image: ##### Figure 2: Zombie conquest with (P=0.000001,d=0.01,G=0.01,rcritical=1,zombiewinratio=0.95)[[11]](#footnote-11)

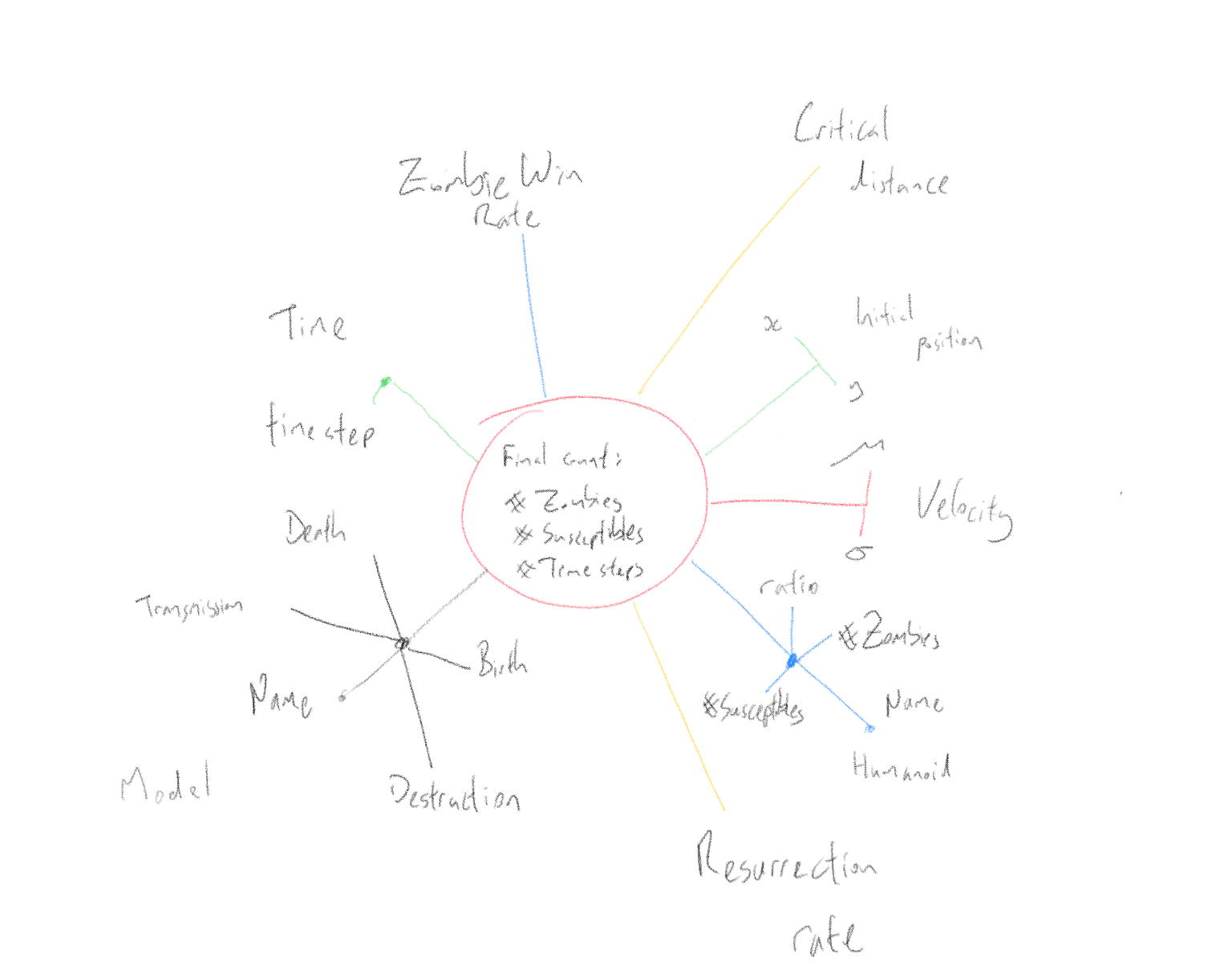
#### Results: comparison to Munz et al

As stated already, the original SZR model as presented by Munz et al [[12]](#footnote-12) was solvable. This, however is a stochastic model leading to different results over each iteration. It was, therefore, necessary to ascertain whether or not the models aligned in similar circumstances. Namely, if average velocity was set to zero and Munz’s original parameters (as per figure 1) were retained[[13]](#footnote-13), would the same results appear. The result of running this model over 100 iterations is shown in Figure 3: image: ##### Figure 3: Zombie conquest with (P=0,d=0.0001,B=0.0095,G=0.0001,a=0.0001,rcritical=1,zombiewinratio=0.95[[14]](#footnote-14)) ###Conclusion for Interactions This was considered a qualified success. The shapes of the red and blue curves clearly resemble those in figure 1 and there are no scenarios where the Zombies do not win. However the wide spread of the curves was informative as it hinted at a wide margin for error. Therefore it was concluded that subsequent iterations of the model would each be ran repeatedly with the mean output being recorded.

### Method for velocity investigation

The model as presented allowed for the varying of 23 different parameters. Whilst the full list is provided in Appendix Six, the crux of the model can be provided by eight variables: - The critical distance below which an interacton happens - The ratio of Susceptibles to Zombies - Starting position of a humanoid - Resurrection rate - Number of timesteps - Mean and standard deviation for a humanood - Zombie win rate - Number of iterations per parameter set The quality of the investigation depended on two critical factors: (1) The number of angles from which the model can be interrogated. This is described by the first seven parameters. (2) The number of times we could run each iteration (as per the findings in figure 2). This is described by the final parameter.

#### Dimensional Model

The interplay between the first seven dimensions was encapsulated by the below user model: image: ##### Figure 4: Zombie sun model This was realised on disk as the logical models shown in Appendix Seven parts one and two (relational and dimensional models, respectively. The relational model was pre-populated according to the parameters we wanted to test. (For example the population of the velocity parameter is seen in Appendix Seven part three: DimVelocity) The cartesian product of these iterations was then returned to the algortihm via a SQL View (See Appendix Seven part four: vModeliteration - *cross joins* provide cartesian product).

Each iteration was then excuted three times and represented by the mean values of following: - final Zombie count - final susceptible cound - number of time steps taken (until a winner was declared i.e. no Humans or no Zombies)

The R code for this is in Appendix Eight: The Engine Room. Each loop of the code returned the above three outputs to SQL Server whereupon the data was rolled into the OLAP cube fo querying. ###Results for velocity investigation After performing preliminary anlayses with MDX (See Appendix nine: Sample MDX) the OLPA cube was placed behind Power BI to investigate the effect of variation on the model.

## Evaluation

The initial phase of work was concentrated on producing the zombie animation. This felt like a good approach because it allowed one to get a “feel” of the interaction between Humanoids. Observing the zombies in this way allowed the initial assumptions to be questioned: for example, it was easy to see that the average of the distribution did indeed equate to a velocity; on the other hand, the idea that the speeds of Humanoids were normally distributed was challenged. The group of humans in particular just *looked* wrong as the “weakest of the herd” were just the unluckiest. With more time the model would have been tweaked to account for this because, as is, the model effectively ignores age and genetics! I think that in reality humans would run away more quickly and this would need revisiting.

One (late discovered) limitation of the model was that there always needed to be at least two zombies (nz=2). This wasn’t deemed too much of an impediment because any models with zero zombies would either wait for a resurrection to happen (nz=1!) or be benign. This left nz=2 as an outstanding issue, in this particular modelling style, this was not too problematic because of the reliance on the initial ratios of Zombies:Susceptibles rather than an absolute number (although this did lead to more iterations!).

I would like to have played around with the concept of interface distance because having it the same for both species would seem incorrect. In general humans are more capable of acting at a greater distance (for example with stones, arrows, guns) so again this would need revisiting.

Indeed, computation was a big problem for this project. There were more parameters than it was possible to vary. The focus on assessing velocity led to other parameters being either ignored totally (for example death rate) or merely tweaked (standard deviation). This was pure combinatorics, a “fully realised” model would have produced in excess of a trillion iterations. This was imposible with the hardware available. Mitigating for this was only possible in some places (for example using a Zombie Win Rate as opposed to transmission and destruction rates reduced dimensionality by one).

A further impediment on this front was the use of Euclidean distance which led to (zombies x susceptibles) inline calculations per timesetp thus inhibiting larger species sets, timespans and therefore spatial areas.

All of that said, the project *did* model the interactions between Humanoids effectively and it would have been fun to introduce concepts like quarantining (as discussed in more detail by Alemi et al[[15]](#footnote-15)) or latent infection (Brooks et al[[16]](#footnote-16)). In my opinion a very good virtual petri dish was built in which wider epedimiological questions could have been asked.

Pete Moore, 2018

## References

Munz, P. et al. (2009) ‘Zombies!’, Science. Alemi, A. A., Bierbaum, M., Myers, C. R., & Sethna, J. P. (2015). You can run, you can hide: The epidemiology and statistical mechanics of zombies. Physical Review E - Statistical, Nonlinear, and Soft Matter Physics. <https://doi.org/10.1103/PhysRevE.92.052801> Brooks, M. (2003). The zombie survival guide : complete protection from the living dead. The Hand. ## Appendix Zero: Coding Environment The model was built using R-base version 3.4.r ran on RStudio 1.1.432 launched from Anaconda Navigator 1.8.7.  
The following named additional libraries were used: matrixStats, animation, pdist, RODBC for the purposes of: linear algebra, building the gif file, calculating the euclidean distance between two matrices and writing to the SQL Server respectively. For the building of the OLAP cube SQL Server 2017 was used with SQL Server Management Studio v17.6 and SQL Server Analysis Services 2017 run from Visual Studio 2015. Where R code was ran from SQL it was done as an external script; these external scripts were built in RStudio originally and where they appear herein do not need reading. The code was executed on a virtualised Windows environment with 12GB of RAM and 2 x 3 GHz Intel Core i7 vCPUs ## Appendix One: Original Python ODE Script ##### zombie apocalypse modeling ##### from <http://scipy-cookbook.readthedocs.io/items/Zombie_Apocalypse_ODEINT.html>

import numpy as np import matplotlib.pyplot as plt from scipy.integrate import odeint plt.ion() plt.rcParams[‘figure.figsize’] = 10, 8

P = 0 # birth rate d = 0.0001 # natural death percent (per day) B = 0.0095 # transmission percent (per day) G = 0.0001 # resurect percent (per day) A = 0.0001 # destroy percent (per day)

##### solve the system dy/dt = f(y, t)

def f(y, t): Si = y[0] Zi = y[1] Ri = y[2] # the model equations (see Munz et al. 2009) dSdt = P - B*Si*Zi - d*Si dZdt = B*Si*Zi + G*Ri - A*Si*Zi dRdt = d*Si + A*Si*Zi - G*Ri return [dSdt, dZdt, dRdt]

##### initial conditions

S0 = 500. # initial population Z0 = 0 # initial zombie population R0 = 0 # initial death population y0 = [S0, Z0, R0] # initial condition vector t = np.linspace(0, 5., 1000) # time grid

##### solve the DEs

soln = odeint(f, y0, t) S = soln[:, 0] Z = soln[:, 1] R = soln[:, 2]

##### plot results

plt.figure()

plt.plot(t, S, label=‘Living’) plt.plot(t, Z, label=‘Zombies’) plt.xlabel(‘Days from outbreak’) plt.ylabel(‘Population’) plt.title(‘Zombie Apocalypse - No Init. Dead Pop.; No New Births.’) plt.legend(loc=0)

## Appendix Two: Random walks

### Appendix Two part one: Random walk in one dimension

## function(Ni, mu, sigma, Xt,humanoidtype="Humanoid",startind=1,humanoidnames=c(),offset=0){  
## #builds a one-d random walk along nominal time axis  
## #where mu=0 this is akin to Brownian motion  
## #inspired by ideas at http://www.phytools.org/eqg/Exercise\_4.1/  
## #time.ntimesteps-number of timesteps, Ni-size of initial Humaoid population,  
## #mu,sigma-mean and standard deviation for footstep length  
## #xinit-initial x-position  
## #time.vtimesteps=number of timestepos  
## #humanoidtype: typically human or zombie  
## #startind-starting ordinal for zombies  
## Nlarge<-time.ntimesteps\*Ni # number of time steps multiplied by initial number of humanoids  
## #Xt<-rep(xinit,Ni) #set of start points (one for each humanoid)  
## footsteps<-rnorm(n=Nlarge,mean=mu,sd=sigma) #lets now get a super long set of footsteps  
## X<-matrix(data=footsteps,nrow=Ni,ncol=time.ntimesteps) #now get a matrix where we get time.ntimesteps columns for each of our Ni humanoids  
## X<-cbind(Xt,X) #now place our initial co-ordinate against x (literally x vs t)  
## X<-apply(X=X,MARGIN=1,FUN=cumsum)  
## X<-t(X) #SUBTLETY: transpose  
## if(length(humanoidnames)==0){  
## humanoidnames<-paste(humanoidtype,c(startind:(Ni+startind-1)))  
## }  
## dimnames(X)=list(humanoidnames,paste0("t", time.vtimesteps+offset))  
## return(X)  
## }

### Appendix Two part two: Random walk in two dimensions

## function(x,Ni, Xt, Yt, humanoidtype="Humanoid",startind=1,humanoidnames=c(),offset=0){  
## mu<-x["mu"]  
## sigma<-x["sigma"]  
## # xinit<-x["xinit"]  
## # yinit<-x["yinit"]  
## # Xt<-rep(xinit,Ni) #set of start points (one for each humanoid)  
## # Yt<-rep(yinit,Ni) #set of start points (one for each humanoid)  
## X<-rwalk1dhumanoids( Ni=Ni, mu=mu, sigma=sigma, Xt=Xt,humanoidtype=humanoidtype,startind=startind,humanoidnames=humanoidnames,offset=offset) # get Ni rows of humanoids X co-ordinates  
## Y<-rwalk1dhumanoids(Ni=Ni, mu=mu, sigma=sigma, Xt=Yt,humanoidtype=humanoidtype,startind=startind,humanoidnames=humanoidnames,offset=offset) # get Ni rows of humanoids Y co-ordinates  
## Z<-cbind(X,Y)  
## Zdim<-c(Ni,time.ntimesteps+1,2)#humanoid,time,x-y  
## humanoidnames<-rownames(X)#paste(humanoidtype,c(1:Ni))  
## W<-array(data= Z,  
## dim = Zdim,  
## dimnames=list(humanoidnames,paste0("t", time.vtimesteps+offset),c("x-displacement","y-displacement"))  
## )  
## return(W)  
## }

## Appendix Three: Humanoid reconfiguration

## function(Z,  
## S.params,  
## Z.params,  
## timesteplocal){  
##   
## #anything with the name RemovedX was removed last time  
## #now it must join the official removed class  
## #this means that the name is changed to remove and the co-ordinates are locked in time  
## Z<-exhumeremoved(Z,timesteplocal,prefix="RemovedX[ZD]",replacement="Removed ")  
## Z<-exhumeremoved(Z,timesteplocal,prefix="BackFromDeadXR",replacement="Zombie ")  
## Z<-exhumeremoved(Z,timesteplocal,prefix="RemovedX[S]",replacement="Zombie ", mode="revive")  
## Z<-exhumeremoved(Z,timesteplocal,prefix=prefix.Newborn,replacement=prefix.Susceptible, mode="start")  
##   
## #start with the resurrections  
## introducednames.Zombies.Resurrected<-reconfigurepopulation.MCSnames(Z=Z,prefix=paste(prefix.Removed,""),r=config.model["G"])  
## if(length(introducednames.Zombies.Resurrected)>0){  
## Ni<-length(introducednames.Zombies.Resurrected)  
## Xt<-Z[introducednames.Zombies.Resurrected,timesteplocal,1]  
## Yt<-Z[introducednames.Zombies.Resurrected,timesteplocal,2]  
## startind<-1  
## humanoidtype=prefix.Resurrected  
## humanoidnames<-introducednames.Zombies.Resurrected #build a new random walk for these guys but update dont insert  
## offset<-max(timesteplocal-1,0)  
## Z<-reconfigurepopulation.revive(Z=Z,Zparams=Z.params,Ni=Ni,Xt=Xt,Yt=Yt,startind=startind,humanoidtype=humanoidtype,humanoidnames=humanoidnames,offset=offset,timesteplocal=timesteplocal)  
## Z<-reconfigurepopulation.changenames(Z=Z,namelist=introducednames.Zombies.Resurrected,prefixfind=prefix.Removed,prefixreplace="BackFromDeadXR")  
## }  
##   
## #death by contact (nearest neighbour deaths)  
## deathmatrixlocal<-deathmatrix(Z,timestep=timesteplocal)  
## s<-sum(deathmatrixlocal,na.rm =TRUE)  
## if(s > 0){  
## #rename the removed class with a placeholder name of RemovedX  
## deathnames.Susceptibles<-deatmatrixisolatehumanoids(deathmatrixlocal,prime.ZombieWins)  
## # print(deathnames.Susceptibles)  
## if (length(deathnames.Susceptibles)>0){  
## # print(deathnames.Susceptibles)  
## # Ni<-length(deathnames.Susceptibles)  
## # Xt<-Z[deathnames.Susceptibles,timesteplocal,1]  
## # Yt<-Z[deathnames.Susceptibles,timesteplocal,2]  
## # startind<-1  
## # humanoidtype="Removed"  
## # humanoidnames<-deathnames.Susceptibles #build a new random walk for these guys but update dont insert  
## # offset<-max(timesteplocal-1,0)  
## # Z<-reconfigurepopulation.revive(Z=Z,Zparams=Z.params,Ni=Ni,Xt=Xt,Yt=Yt,startind=startind,humanoidtype=humanoidtype,humanoidnames=deathnames.Susceptibles,offset=offset,timesteplocal=timesteplocal)  
## Z<-reconfigurepopulation.changenames(Z=Z,namelist=deathnames.Susceptibles,prefixfind=prefix.Susceptible,prefixreplace="RemovedXS")  
## }  
##   
## deathnames.Zombies<-deatmatrixisolatehumanoids(t(deathmatrixlocal),prime.SusceptibleWins)  
## if(length(deathnames.Zombies)>0){  
## Z<-reconfigurepopulation.changenames(Z=Z,namelist=deathnames.Zombies,prefixfind=prefix.Zombie,prefixreplace="RemovedXZ")  
## }  
## }  
##   
## #nautral deaths  
## deathnames.Susceptibles.Natural<-reconfigurepopulation.MCSnames(Z=Z,prefix=prefix.Susceptible,r=config.model["d"])  
## if(length(deathnames.Susceptibles.Natural)>0){  
## Z<-reconfigurepopulation.changenames(Z=Z,namelist=deathnames.Susceptibles.Natural,prefixfind=prefix.Susceptible,prefixreplace="RemovedXD")  
## }  
##   
## #natural births  
## introducednames.Susceptibles.Natural<-reconfigurepopulation.MCSnames(Z=Z,prefix=prefix.Susceptible,r=config.model["P"],mode="numbers")  
## introducednames.Susceptibles.Natural.Names<-reconfigurepopulation.MCSnames(Z=Z,prefix=prefix.Susceptible,r=config.model["P"])  
## if(introducednames.Susceptibles.Natural>0){#SUBTLETY: length function omitted  
## Ni<-introducednames.Susceptibles.Natural  
## xinit<-S.params["xinit"]  
## yinit<-S.params["yinit"]  
## Xt<-rep(xinit,Ni) #set of start points (one for each humanoid)  
## Yt<-rep(yinit,Ni) #set of start points (one for each humanoid)  
## startind<-length(Z[,1,1])+1  
## humanoidtype<-prefix.Newborn  
## humanoidnames<-introducednames.Susceptibles.Natural.Names  
## Z<-reconfigurepopulation.introduce(Z,S.params,Ni,Xt,Yt,startind,humanoidtype, humanoidnames)  
## }  
##   
##   
##   
## return(Z)  
## }

## Appendix Four: population reconfiguration

## function(Z,  
## S.params,  
## Z.params,  
## timesteplocal){  
##   
## #anything with the name RemovedX was removed last time  
## #now it must join the official removed class  
## #this means that the name is changed to remove and the co-ordinates are locked in time  
## Z<-exhumeremoved(Z,timesteplocal,prefix="RemovedX[ZD]",replacement="Removed ")  
## Z<-exhumeremoved(Z,timesteplocal,prefix="BackFromDeadXR",replacement="Zombie ")  
## Z<-exhumeremoved(Z,timesteplocal,prefix="RemovedX[S]",replacement="Zombie ", mode="revive")  
## Z<-exhumeremoved(Z,timesteplocal,prefix=prefix.Newborn,replacement=prefix.Susceptible, mode="start")  
##   
## #start with the resurrections  
## introducednames.Zombies.Resurrected<-reconfigurepopulation.MCSnames(Z=Z,prefix=paste(prefix.Removed,""),r=config.model["G"])  
## if(length(introducednames.Zombies.Resurrected)>0){  
## Ni<-length(introducednames.Zombies.Resurrected)  
## Xt<-Z[introducednames.Zombies.Resurrected,timesteplocal,1]  
## Yt<-Z[introducednames.Zombies.Resurrected,timesteplocal,2]  
## startind<-1  
## humanoidtype=prefix.Resurrected  
## humanoidnames<-introducednames.Zombies.Resurrected #build a new random walk for these guys but update dont insert  
## offset<-max(timesteplocal-1,0)  
## Z<-reconfigurepopulation.revive(Z=Z,Zparams=Z.params,Ni=Ni,Xt=Xt,Yt=Yt,startind=startind,humanoidtype=humanoidtype,humanoidnames=humanoidnames,offset=offset,timesteplocal=timesteplocal)  
## Z<-reconfigurepopulation.changenames(Z=Z,namelist=introducednames.Zombies.Resurrected,prefixfind=prefix.Removed,prefixreplace="BackFromDeadXR")  
## }  
##   
## #death by contact (nearest neighbour deaths)  
## deathmatrixlocal<-deathmatrix(Z,timestep=timesteplocal)  
## s<-sum(deathmatrixlocal,na.rm =TRUE)  
## if(s > 0){  
## #rename the removed class with a placeholder name of RemovedX  
## deathnames.Susceptibles<-deatmatrixisolatehumanoids(deathmatrixlocal,prime.ZombieWins)  
## # print(deathnames.Susceptibles)  
## if (length(deathnames.Susceptibles)>0){  
## # print(deathnames.Susceptibles)  
## # Ni<-length(deathnames.Susceptibles)  
## # Xt<-Z[deathnames.Susceptibles,timesteplocal,1]  
## # Yt<-Z[deathnames.Susceptibles,timesteplocal,2]  
## # startind<-1  
## # humanoidtype="Removed"  
## # humanoidnames<-deathnames.Susceptibles #build a new random walk for these guys but update dont insert  
## # offset<-max(timesteplocal-1,0)  
## # Z<-reconfigurepopulation.revive(Z=Z,Zparams=Z.params,Ni=Ni,Xt=Xt,Yt=Yt,startind=startind,humanoidtype=humanoidtype,humanoidnames=deathnames.Susceptibles,offset=offset,timesteplocal=timesteplocal)  
## Z<-reconfigurepopulation.changenames(Z=Z,namelist=deathnames.Susceptibles,prefixfind=prefix.Susceptible,prefixreplace="RemovedXS")  
## }  
##   
## deathnames.Zombies<-deatmatrixisolatehumanoids(t(deathmatrixlocal),prime.SusceptibleWins)  
## if(length(deathnames.Zombies)>0){  
## Z<-reconfigurepopulation.changenames(Z=Z,namelist=deathnames.Zombies,prefixfind=prefix.Zombie,prefixreplace="RemovedXZ")  
## }  
## }  
##   
## #nautral deaths  
## deathnames.Susceptibles.Natural<-reconfigurepopulation.MCSnames(Z=Z,prefix=prefix.Susceptible,r=config.model["d"])  
## if(length(deathnames.Susceptibles.Natural)>0){  
## Z<-reconfigurepopulation.changenames(Z=Z,namelist=deathnames.Susceptibles.Natural,prefixfind=prefix.Susceptible,prefixreplace="RemovedXD")  
## }  
##   
## #natural births  
## introducednames.Susceptibles.Natural<-reconfigurepopulation.MCSnames(Z=Z,prefix=prefix.Susceptible,r=config.model["P"],mode="numbers")  
## introducednames.Susceptibles.Natural.Names<-reconfigurepopulation.MCSnames(Z=Z,prefix=prefix.Susceptible,r=config.model["P"])  
## if(introducednames.Susceptibles.Natural>0){#SUBTLETY: length function omitted  
## Ni<-introducednames.Susceptibles.Natural  
## xinit<-S.params["xinit"]  
## yinit<-S.params["yinit"]  
## Xt<-rep(xinit,Ni) #set of start points (one for each humanoid)  
## Yt<-rep(yinit,Ni) #set of start points (one for each humanoid)  
## startind<-length(Z[,1,1])+1  
## humanoidtype<-prefix.Newborn  
## humanoidnames<-introducednames.Susceptibles.Natural.Names  
## Z<-reconfigurepopulation.introduce(Z,S.params,Ni,Xt,Yt,startind,humanoidtype, humanoidnames)  
## }  
##   
##   
##   
## return(Z)  
## }

## Appendix Five: All functions in alphapbetical order

## function(Z,oldnames,newnames){  
## #takes an array, Z, and replaces a vector of oldnames with a vector of new ones  
## myrownames<-rownames(Z)  
## for (i in 1:length(oldnames)){ #blimey, there has to be a more efficient way than this  
## myrownames<-replace(myrownames, myrownames == oldnames[i], newnames[i])  
## }  
## rownames(Z)<-myrownames  
## return(Z)  
## }

## function(Z,prefix){  
## nm<-rownames(Z)  
## startwithlocal<-paste("^",prefix,sep="")  
## vstartwith<- nm %in% grep(startwithlocal, nm, value = TRUE)  
## nm<-subset(nm,vstartwith)  
## Z<-Z[nm,1:(time.ntimesteps+1),1:2,drop=FALSE]  
##   
## return(Z)  
## }

## function(H,J, topology="NTXY"){  
## #takes two arrays where expected dimensions are planar co-ordinates in time (for humanoids)  
## #these are then deconstructed into the X and Y co-ordinates  
## #and reconstructed  
## #the result is a union in the n-dimensional vector space  
##   
## Hx<-H[,,1]  
## Hy<-H[,,2]  
## Jx<-J[,,1]  
## Jy<-J[,,2]  
##   
## xcoords<-rbind(Hx,Jx)  
## ycoords<-rbind(Hy,Jy)  
## allcoords<-cbind(xcoords,ycoords)  
##   
## Nih<-length(H[,1,1])  
## Nij<-length(J[,1,1])  
## Zdim<-c(Nih+Nij,time.ntimesteps+1,2)#humanoid,time,x-y  
## Z<-array(data= allcoords,  
## dim = Zdim,  
## dimnames=list(rownames(allcoords),paste0("t", time.vtimesteps),c("x-displacement","y-displacement"))  
## )  
## return(Z)  
## }

## function(outputfilename="/Users/peterfmoore/Documents/GitHub/ZombieEpidemic/zombieanimation.gif"){  
## #define axes limits on the whole data set  
## xmin<-min(config.Susceptible["xinit"], config.Zombie["xinit"])  
## ymin<-min(config.Susceptible["yinit"], config.Zombie["yinit"])  
## xmax<-max(config.Susceptible["xinit"], config.Zombie["xinit"])  
## ymax<-max(config.Susceptible["yinit"], config.Zombie["yinit"])  
## xlimlocal<-c(xmin-1\*floor(sqrt(time.ntimesteps+1)),xmax+ceiling(sqrt(time.ntimesteps+1))) #using square root of n as arough approximatei  
## ylimlocal<-c(ymin-1\*floor(sqrt(time.ntimesteps+1)),ymax+ceiling(sqrt(time.ntimesteps+1)))  
## x<-list(xlimlocal,ylimlocal,outputfilename)  
## names(x)<-c("xlimlocal","ylimlocal","outputfilename")  
## return(x)  
## }

## function(xinit,#set a starting co-ordinate for humanoids on the x-dimension  
## yinit,#set a starting co-ordinate for humanoids on the y-dimension  
## n,#initial number of humanoids  
## mu,#mean footstep length for humanoids IMPORTANT: because we have unit time this is average humanoids speed  
## sigma#standard deviation of footstep lengths for humanoids  
## ){  
## x<-c(xinit,  
## yinit,  
## n,  
## mu,  
## sigma)  
## names(x)<-c("xinit",  
## "yinit",  
## "n",  
## "mu",  
## "sigma")  
## return(x)  
## }

## function(P=0,# birth rate  
## d=0.0001,# natural death percent (per unit time)  
## B=0.0095,# transmission percent (per unit time)  
## G=0.0001,# resurrect percent (per unit time)  
## a=0.0001,# destroy  
## rcritical=2,# critical Euclidean distance below which a zombie and human will go into battle  
## zombiewinratio=0.95# ratio at which Zombie wins  
## ){  
## x<-c(P,d,B,G,a,rcritical,zombiewinratio)  
## names(x)<-c("P","d","B","G","a","rcritical","zombiewinratio")  
## return(x)  
## }

## function(Z,timestep){  
## # a matrix of zombie winners (2s), zombie losers (3s), and live to fight another day-ers (NAs)  
## # NB it is important that these numbers are prime.SusceptibleWins  
## interfacedistance = config.model["rcritical"]  
## zombiewinrate = config.model["zombiewinratio"]  
## # naturaldeathrate=config.model["d"]  
##   
## H<-arrayStartWith(Z,prefix.Susceptible)#Susceptibles  
## J<-arrayStartWith(Z,prefix.Zombie)#Zombies  
## xyfootprintH<-as.matrix(H[,timestep,])  
## xyfootprintJ<-as.matrix(J[,timestep,])  
##   
## #nearest neighbour killings  
## pdistHJ<-euclideandistancebetweentwomatrices(xyfootprintH, xyfootprintJ)  
## L<-isolateminima(pdistHJ) #isolate minima (ASSUMPTION only nearest neighbours do anything)  
## x<-ifelse(runif(length(L))>zombiewinrate, prime.SusceptibleWins, prime.ZombieWins) #CONVENTION 2 means zombie wins, 3 means human wins (zombie doesnt), NA means do nothing  
## K<-ifelse(L<interfacedistance,x,NA) #create a matrix of winners, losers, and live to fight another day-ers  
##   
##   
## # #natural causes killings  
## # y<-ifelse(runif(length(K))<naturaldeathrate,prime.NaturalDeath,NA)  
## # K<-ifelse(is.na(K),y,K)  
## return(K)  
## }

## function(deathmatrixlocal,deathprime){  
## dmatrowsums<-rowSums(deathmatrixlocal, na.rm=TRUE)  
## namelist<-(subset(dmatrowsums,dmatrowsums==deathprime))  
## namelist<-names(namelist)  
##   
## if (length(subset(namelist,namelist=="x-displacement"))+length(subset(namelist,namelist=="y-displacement"))>0){  
## namelist<-c()  
## }  
##   
## return(namelist)  
## }

## function(H,J){  
## #function that takes in two matrices EXPECTING col 1 and col2 two be pairwise co-ordinates in Euclidean space  
## #note also the importance of the naming here, the function itself is humanoid agnostic, this information is contained in  
## #the meta data of the matrix (i.e. the row and column names)  
## #these are then returned and this is non-trivial  
## library(pdist)  
## dists<-as.matrix(pdist(H,J))  
## distsnames<-list(rownames(H), rownames(J))  
## dimnames(dists)<-distsnames  
## return(dists)  
## }

## function(Z,timesteplocal,prefix,replacement,mode="stun"){  
## #when humanoids are removed something happens to their position  
## #the mode handles this:  
## #mode="stun"=>dead humanoids stay still so they get stuck with the x-y of their last known whereabouts for eternity  
## #mode="start"=>newborns already have their co-ordinates so we dont need to do anything  
## #mode="revive"=>resurrected were previously stunned so they need a new set of coords  
## K<-arrayStartWith(Z,prefix)  
## if(matrixnn(K)){  
## Knames<-rownames(K)  
## if (mode=="stun"){  
## for (i in (1:2)){  
## Z[Knames,,i]<-K[,timesteplocal-1,i]#SUBTLETY: timestep-1  
## }  
## }  
## prefix<-paste(prefix,".",sep="")  
## pm<-sub(prefix,replacement,Knames)  
## Z<-arrayChangeRowNames(Z,Knames,pm)  
## }  
## return(Z)  
## }

## function(M, removeNA = TRUE){  
## #takes a matrix and returns the values that match the minima for that column and leaves the rest as NA  
## return(t(isolaterowminima(t(M),removeNA))) #note that we transpose twice  
## }

## function(M, removeNA = TRUE){  
## #takes a matrix and returns the values that match the minima for that row and column and leaves the rest as NA  
## return(isolatecolminima(isolaterowminima(M, removeNA), removeNA))  
## }

## function(M, removeNA = TRUE){  
## #takes a matrix and returns the values that match the minima for that row and leaves the rest as NA  
## library(matrixStats)  
## x<-rowMins(M,na.rm = removeNA)  
## L<-ifelse(M==x,M,NA)  
## return(L)  
## }

## function(K){  
## booGood<-FALSE  
## if (!is.null(rownames(K))){  
## booGood<-TRUE  
## }  
## return(booGood)  
## }

## function(  
## S.params,  
## Z.params,  
## outputfilename=config.plot$outputfilename,  
## mode="matrix",  
## plot=FALSE,  
## iterations=c(1:1)  
## ){  
## #runs a model of zombies atacking humans and vice versa, has two modes:  
## #1. matrix: the default, produces a matrix of Zombie and Susceptible ubers at each timestep  
## #2. gif: maps the same information in a gif animation  
## if (mode=="matrix"){  
## if (plot){  
## p<-plot(x=c(),y=c(),type="l", col="blue",xlab="time", ylab="Humanoid Count", main="Zombies vs Susceptibles",  
## frame.plot = FALSE,xlim=c(0,length(time.vtimesteps)),ylim=c(0,(S.params["n"]+Z.params["n"])))  
## p<-legend("bottom", c("Susceptibles", "Zombies"), xpd = TRUE, horiz = TRUE,   
## inset = c(0, 0), pch=15,bty = "n", col = c("blue","red"), cex = 0.5)  
## }  
## }  
## for(mi in iterations){   
## #####setup####  
## Ni=S.params["n"]  
## xinit<-S.params["xinit"]  
## yinit<-S.params["yinit"]  
## Xt<-rep(xinit,Ni) #set of start points (one for each humanoid)  
## Yt<-rep(yinit,Ni) #set of start points (one for each humanoid)  
## Z<-rwalk2dhumanoids(  
## x=S.params,  
## Ni,  
## Xt,  
## Yt,  
## humanoidtype=prefix.Susceptible,  
## startind = 1)  
##   
## Ni=Z.params["n"]  
## xinit<-Z.params["xinit"]  
## yinit<-Z.params["yinit"]  
## Xt<-rep(xinit,Ni) #set of start points (one for each humanoid)  
## Yt<-rep(yinit,Ni) #set of start points (one for each humanoid)  
## startind<-length(Z[,1,1])+1  
## Z1<-rwalk2dhumanoids(  
## x=Z.params,  
## Ni,  
## Xt,  
## Yt,  
## humanoidtype=prefix.Zombie,  
## startind=startind)  
## #combine them to get a humanoid array  
## Z<-arrayunion(Z,Z1)  
## #####iterate#####  
## m.M<- matrix(data=NA,nrow = length(time.vtimesteps),ncol=2,dimnames = list(paste0("t",time.vtimesteps),c("SusceptibleCount", "ZombieCount")))  
## m.M["t0",]<-c(S.params["n"],Z.params["n"])  
## tstep<-0  
## if (mode=="matrix"){  
## for (i in time.vtimesteps){  
## tstep<-i+1  
##   
## #for each timestep we need to remove any humanoids that get too near, too born, or too tired (i.e. they die)  
## Z<-reconfigurepopulation(Z=Z,  
## S.params = S.params,  
## Z.params = Z.params,  
## timesteplocal=tstep)  
## #look at tstepust the suceptibles  
## m.Susceptible<-arrayStartWith(Z,prefix.Susceptible)  
## #look at tstepust the zombies  
## m.Zombie<-arrayStartWith(Z,prefix.Zombie)  
## m.Susceptible.count<-length(rownames(m.Susceptible))  
## m.Zombie.count<-length(rownames(m.Zombie))  
## m.M[tstep,]<-c(m.Susceptible.count,m.Zombie.count)  
## x<-(paste0("Final timestep recorded at t=t",tstep))  
## if(model.timestep.assessment(m.Susceptible.count,m.Zombie.count)){  
## break #if there are no zombies or suscpetibles left then leave  
## }  
## }  
## print(x)  
## time.axis<-time.vtimesteps[1:length(time.vtimesteps)]  
## m.M<-cbind(subset(m.M,rowSums(m.M)>=0),tstep)  
## if (plot){  
## p<-lines(x=time.axis,y=m.M[,"SusceptibleCount"],type="l", col="blue")  
## p<-lines(x=time.axis,y=m.M[,"ZombieCount"],type="l", col="red")  
## }  
## } else if (mode=="gif"){  
## Z<-plotoutbreakXYTasgif(Z,  
## S.params,  
## Z.params,  
## unleashplot = TRUE)  
## }  
## }  
## return(m.M)  
## }

## function(Scount,Zcount){  
## breakme<-FALSE  
## if (Scount==0){  
## breakme<-TRUE  
## } else if (Zcount==0){  
## breakme<-TRUE  
## }  
## return(breakme)  
## }

## function(Z,  
## timestep  
## ){  
##   
## #humanoids outbreak for x and y displacements for two matrices of humanoids  
## #this is effectively a four dimensional vector space contained in Z  
## #this function uses that to plot the X-Y displacement for humans and zombies in x-y x-coordinate for this epoch  
## #by calling it repeatedly we can build an animation per plotourbreakXYTasGIf  
##   
## #Susceptibles  
## m.Susceptible<-arrayStartWith(Z,prefix.Susceptible)  
## #Zombies  
## m.Zombie<-arrayStartWith(Z,prefix.Zombie)  
## #Those killed on this iteration  
## m.Death.Susceptibles.Killed<-arrayStartWith(Z,"RemovedXS")  
## m.Death.Susceptibles.NaturalDeath<-arrayStartWith(Z,"RemovedXD")  
## m.Death.Zombies.Killed<-arrayStartWith(Z,"RemovedXZ")  
## #Mark those already dead  
## m.Dead<-arrayStartWith(Z,"Removed[^X]")  
## #and those just born  
## m.Born<-arrayStartWith(Z,prefix.Newborn)  
##   
## #and those back from the dead  
## m.Back<-arrayStartWith(Z,prefix.Resurrected)  
##   
## susplot<-plotsub(m.Susceptible,timestep = timestep,singlecolour = "blue")  
## zomplot<-plotsub(m.Zombie,timestep = timestep,singlecolour = "red")  
##   
##   
## #generic  
## mlocal<-paste("x-y coordinates for ", susplot$nlocal, " ", prefix.Susceptible, "s,", sep="")  
## mlocal<-paste(mlocal, " and ", zomplot$nlocal, " ", prefix.Zombie, "s,", sep="")  
## mlocal<-paste(mlocal, " at t=t", (timestep-1), sep="")  
## plot(x=c(), xlab="x-displacement", ylab="y-displacement", main=mlocal, cex.main = 1.0, frame.plot = FALSE, xlim = config.plot$xlimlocal, ylim = config.plot$ylimlocal)  
##   
##   
## if(matrixnn(m.Susceptible)){  
## points(x=susplot$xfootprint,y=susplot$yfootprint,type="p",pch=19, col=susplot$collocal)  
## }  
##   
## if(matrixnn(m.Zombie)){  
## points(x=zomplot$xfootprint,y=zomplot$yfootprint,type="p", pch=15, col=zomplot$collocal)  
## }  
##   
## # if(matrixnn(m.Death)){  
## # justplot<-plotsub(m.Death, timestep = timestep,singlecolour = "red")#plotsub(plotmatrix=m.Death,timestep = timestep,singlecolour = "red")  
## # points(x=justplot$xfootprint,y=justplot$yfootprint,type="p", col=justplot$collocal, pch=8, cex=2)  
## # }  
##   
## if(matrixnn(m.Death.Susceptibles.Killed)){  
## justplot<-plotsub(m.Death.Susceptibles.Killed, timestep = timestep,singlecolour = "blue")#plotsub(plotmatrix=m.Death,timestep = timestep,singlecolour = "red")  
## points(x=justplot$xfootprint,y=justplot$yfootprint,type="p", col=justplot$collocal, pch=8, cex=2)  
## }  
##   
## if(matrixnn(m.Death.Susceptibles.NaturalDeath)){  
## justplot<-plotsub(m.Death.Susceptibles.NaturalDeath, timestep = timestep,singlecolour = "black")#plotsub(plotmatrix=m.Death,timestep = timestep,singlecolour = "red")  
## points(x=justplot$xfootprint,y=justplot$yfootprint,type="p", col=justplot$collocal, pch=8, cex=2)  
## }  
##   
## if(matrixnn(m.Death.Zombies.Killed)){  
## justplot<-plotsub(m.Death.Zombies.Killed, timestep = timestep,singlecolour = "red")#plotsub(plotmatrix=m.Death,timestep = timestep,singlecolour = "red")  
## points(x=justplot$xfootprint,y=justplot$yfootprint,type="p", col=justplot$collocal, pch=8, cex=2)  
## }  
##   
## if(matrixnn(m.Dead)){  
## deadplot<-plotsub(m.Dead,timestep = timestep,singlecolour = "black")#plotsub(plotmatrix=m.Dead,timestep = timestep,singlecolour = "black")  
## points(x=deadplot$xfootprint,y=deadplot$yfootprint,type="p", col=deadplot$collocal, pch=13, cex=1)  
## }  
##   
## if(matrixnn(m.Born)){  
## bornplot<-plotsub(m.Born,timestep = timestep,singlecolour = "pink")#plotsub(plotmatrix=m.Born,timestep = timestep,singlecolour = "black")  
## points(x=bornplot$xfootprint,y=bornplot$yfootprint,type="p", col=bornplot$collocal, pch=11, cex=1)  
## }  
##   
## if(matrixnn(m.Back)){  
## backplot<-plotsub(m.Back,timestep = timestep,singlecolour = "green")#plotsub(plotmatrix=m.Back,timestep = timestep,singlecolour = "black")  
## points(x=backplot$xfootprint,y=backplot$yfootprint,type="p", col=backplot$collocal, pch=6, cex=1)  
## }  
##   
## legend("bottom", c("Susceptibles", "Zombies", "Removed", "Birth", "Resurrection"), xpd = TRUE, horiz = TRUE,   
## inset = c(0, 0),bty = "n", col = c("blue","red", "black", "pink", "green"), cex = 0.5,pch=c(19,15,13,8,8))  
## }

## function(Z,  
## S.params,  
## Z.params,  
## outputfilename=config.plot$outputfilename,  
## unleashplot=TRUE){  
## #builds a gif of the zombie vs susceptible action and outputs it to the specified filename (Downloads folder)  
## if(unleashplot){  
## library(animation)  
##   
## saveGIF({  
## for (i in time.vtimesteps){  
## j<-i+1  
##   
## #for each timestep we need to remove any humanoids that get too near, too born, or too tired (i.e. they die)  
## Z<-reconfigurepopulation(Z=Z,  
## S.params,  
## Z.params,  
## timesteplocal=j)  
## #look at just the suceptibles  
## m.Susceptible<-arrayStartWith(Z,prefix.Susceptible)  
## #look at just the zombies  
## m.Zombie<-arrayStartWith(Z,prefix.Zombie)  
## m.Susceptible.count<-length(rownames(m.Susceptible))  
## m.Zombie.count<-length(rownames(m.Zombie))  
## if(model.timestep.assessment(m.Susceptible.count,m.Zombie.count)){  
## print(paste("Game Over at t=t",j))  
## break #if there are no zombies or suscpetibles left then leave  
## }  
## plotoutbreakXYall(Z=Z,  
## timestep=j)  
## }  
## }, movie.name=outputfilename)  
## }  
##   
## return(Z)  
## }

## function(plotmatrix, timestep, singlecolour){  
## nameextent<-1:length(rownames(plotmatrix))  
## xyfootprint<-(plotmatrix[nameextent,timestep,1:2,drop=FALSE])  
## #get ourselves a nice two-D matrix--,1,,drop=FALSE  
## xfootprint<-xyfootprint[,,1] #a nice x-displacement vector  
## yfootprint<-xyfootprint[,,2] #and a nice y-displacement vector  
## nlocal<-length(xfootprint)  
## collocal<-rep(singlecolour,nlocal)  
## #danger CONVENTION: axes labels are asumed the same because of the way H and J are created (means it wonl;t work for any old matrix)  
## xlocal<-colnames(xyfootprint)[1]  
## ylocal<-colnames(xyfootprint)[2]  
## l1<-list(xyfootprint,xfootprint,yfootprint,nlocal,collocal,xlocal,ylocal)  
## names(l1)<-c("xyfootprint","xfootprint","yfootprint","nlocal","collocal","xlocal","ylocal")  
##   
## return(l1)  
## }

## function(Z,  
## S.params,  
## Z.params,  
## timesteplocal){  
##   
## #anything with the name RemovedX was removed last time  
## #now it must join the official removed class  
## #this means that the name is changed to remove and the co-ordinates are locked in time  
## Z<-exhumeremoved(Z,timesteplocal,prefix="RemovedX[ZD]",replacement="Removed ")  
## Z<-exhumeremoved(Z,timesteplocal,prefix="BackFromDeadXR",replacement="Zombie ")  
## Z<-exhumeremoved(Z,timesteplocal,prefix="RemovedX[S]",replacement="Zombie ", mode="revive")  
## Z<-exhumeremoved(Z,timesteplocal,prefix=prefix.Newborn,replacement=prefix.Susceptible, mode="start")  
##   
## #start with the resurrections  
## introducednames.Zombies.Resurrected<-reconfigurepopulation.MCSnames(Z=Z,prefix=paste(prefix.Removed,""),r=config.model["G"])  
## if(length(introducednames.Zombies.Resurrected)>0){  
## Ni<-length(introducednames.Zombies.Resurrected)  
## Xt<-Z[introducednames.Zombies.Resurrected,timesteplocal,1]  
## Yt<-Z[introducednames.Zombies.Resurrected,timesteplocal,2]  
## startind<-1  
## humanoidtype=prefix.Resurrected  
## humanoidnames<-introducednames.Zombies.Resurrected #build a new random walk for these guys but update dont insert  
## offset<-max(timesteplocal-1,0)  
## Z<-reconfigurepopulation.revive(Z=Z,Zparams=Z.params,Ni=Ni,Xt=Xt,Yt=Yt,startind=startind,humanoidtype=humanoidtype,humanoidnames=humanoidnames,offset=offset,timesteplocal=timesteplocal)  
## Z<-reconfigurepopulation.changenames(Z=Z,namelist=introducednames.Zombies.Resurrected,prefixfind=prefix.Removed,prefixreplace="BackFromDeadXR")  
## }  
##   
## #death by contact (nearest neighbour deaths)  
## deathmatrixlocal<-deathmatrix(Z,timestep=timesteplocal)  
## s<-sum(deathmatrixlocal,na.rm =TRUE)  
## if(s > 0){  
## #rename the removed class with a placeholder name of RemovedX  
## deathnames.Susceptibles<-deatmatrixisolatehumanoids(deathmatrixlocal,prime.ZombieWins)  
## # print(deathnames.Susceptibles)  
## if (length(deathnames.Susceptibles)>0){  
## # print(deathnames.Susceptibles)  
## # Ni<-length(deathnames.Susceptibles)  
## # Xt<-Z[deathnames.Susceptibles,timesteplocal,1]  
## # Yt<-Z[deathnames.Susceptibles,timesteplocal,2]  
## # startind<-1  
## # humanoidtype="Removed"  
## # humanoidnames<-deathnames.Susceptibles #build a new random walk for these guys but update dont insert  
## # offset<-max(timesteplocal-1,0)  
## # Z<-reconfigurepopulation.revive(Z=Z,Zparams=Z.params,Ni=Ni,Xt=Xt,Yt=Yt,startind=startind,humanoidtype=humanoidtype,humanoidnames=deathnames.Susceptibles,offset=offset,timesteplocal=timesteplocal)  
## Z<-reconfigurepopulation.changenames(Z=Z,namelist=deathnames.Susceptibles,prefixfind=prefix.Susceptible,prefixreplace="RemovedXS")  
## }  
##   
## deathnames.Zombies<-deatmatrixisolatehumanoids(t(deathmatrixlocal),prime.SusceptibleWins)  
## if(length(deathnames.Zombies)>0){  
## Z<-reconfigurepopulation.changenames(Z=Z,namelist=deathnames.Zombies,prefixfind=prefix.Zombie,prefixreplace="RemovedXZ")  
## }  
## }  
##   
## #nautral deaths  
## deathnames.Susceptibles.Natural<-reconfigurepopulation.MCSnames(Z=Z,prefix=prefix.Susceptible,r=config.model["d"])  
## if(length(deathnames.Susceptibles.Natural)>0){  
## Z<-reconfigurepopulation.changenames(Z=Z,namelist=deathnames.Susceptibles.Natural,prefixfind=prefix.Susceptible,prefixreplace="RemovedXD")  
## }  
##   
## #natural births  
## introducednames.Susceptibles.Natural<-reconfigurepopulation.MCSnames(Z=Z,prefix=prefix.Susceptible,r=config.model["P"],mode="numbers")  
## introducednames.Susceptibles.Natural.Names<-reconfigurepopulation.MCSnames(Z=Z,prefix=prefix.Susceptible,r=config.model["P"])  
## if(introducednames.Susceptibles.Natural>0){#SUBTLETY: length function omitted  
## Ni<-introducednames.Susceptibles.Natural  
## xinit<-S.params["xinit"]  
## yinit<-S.params["yinit"]  
## Xt<-rep(xinit,Ni) #set of start points (one for each humanoid)  
## Yt<-rep(yinit,Ni) #set of start points (one for each humanoid)  
## startind<-length(Z[,1,1])+1  
## humanoidtype<-prefix.Newborn  
## humanoidnames<-introducednames.Susceptibles.Natural.Names  
## Z<-reconfigurepopulation.introduce(Z,S.params,Ni,Xt,Yt,startind,humanoidtype, humanoidnames)  
## }  
##   
##   
##   
## return(Z)  
## }

## function(Z,namelist,prefixfind,prefixreplace){  
##   
## deaths<-sort(unique(namelist))  
## oldnames<-sort(unique(deaths))  
## prefix<-paste("^", prefixfind,sep="")  
## newnames<-sort(sub(prefix, prefixreplace, oldnames))  
## Z<-arrayChangeRowNames(Z=Z,oldnames=oldnames,newnames=newnames)  
## return(Z)  
## }

## function(Z,S.params,Ni,Xt,Yt,startind,humanoidtype,humanoidnames){  
## Z1<-rwalk2dhumanoids(  
## x=S.params,  
## Ni,  
## Xt,  
## Yt,  
## humanoidtype,  
## startind,  
## humanoidnames)  
## unames<-union(rownames(Z),rownames(Z1))  
## Z<-arrayunion(Z,Z1)  
## rownames(Z)<-unames  
##   
## return(Z)  
## }

## function(Z,prefix,r,mode="names"){  
## #returns the names of humanoids whose status is aboutto change (e.g. Susceptible->Dead)  
## #mode="names" means return a set of names, "numbers" means return lenght  
## K<-arrayStartWith(Z,prefix=prefix)  
## Knames<-rownames(K)  
## u<-runif(1:length(Knames))  
## u<-ifelse(u<r,1,0)  
## names(u)<-Knames  
## x<-subset(u,u==1)  
## x<-names(x)  
## if(mode=="numbers"){  
## x<-length(x)  
## }  
## return(x)  
## }

## function(Z,Zparams,Ni,Xt,Yt,startind,humanoidtype,humanoidnames,offset,timesteplocal){  
## K<-rwalk2dhumanoids(  
## x=Zparams,  
## Ni=Ni,  
## Xt=Xt,  
## Yt=Yt,  
## humanoidtype=humanoidtype,  
## startind,  
## humanoidnames=humanoidnames,  
## offset=offset)  
##   
## rownames(K)<-humanoidnames  
## Knames<-rownames(K)  
## vt<-paste0("t",(timesteplocal-1):(length(time.vtimesteps)-1))  
##   
## for (i in (1:2)){  
## Z[Knames,vt,i]<-K[Knames,vt,i]  
## }  
## return(Z)  
## }

## function(Ni, mu, sigma, Xt,humanoidtype="Humanoid",startind=1,humanoidnames=c(),offset=0){  
## #builds a one-d random walk along nominal time axis  
## #where mu=0 this is akin to Brownian motion  
## #inspired by ideas at http://www.phytools.org/eqg/Exercise\_4.1/  
## #time.ntimesteps-number of timesteps, Ni-size of initial Humaoid population,  
## #mu,sigma-mean and standard deviation for footstep length  
## #xinit-initial x-position  
## #time.vtimesteps=number of timestepos  
## #humanoidtype: typically human or zombie  
## #startind-starting ordinal for zombies  
## Nlarge<-time.ntimesteps\*Ni # number of time steps multiplied by initial number of humanoids  
## #Xt<-rep(xinit,Ni) #set of start points (one for each humanoid)  
## footsteps<-rnorm(n=Nlarge,mean=mu,sd=sigma) #lets now get a super long set of footsteps  
## X<-matrix(data=footsteps,nrow=Ni,ncol=time.ntimesteps) #now get a matrix where we get time.ntimesteps columns for each of our Ni humanoids  
## X<-cbind(Xt,X) #now place our initial co-ordinate against x (literally x vs t)  
## X<-apply(X=X,MARGIN=1,FUN=cumsum)  
## X<-t(X) #SUBTLETY: transpose  
## if(length(humanoidnames)==0){  
## humanoidnames<-paste(humanoidtype,c(startind:(Ni+startind-1)))  
## }  
## dimnames(X)=list(humanoidnames,paste0("t", time.vtimesteps+offset))  
## return(X)  
## }

## function(x,Ni, Xt, Yt, humanoidtype="Humanoid",startind=1,humanoidnames=c(),offset=0){  
## mu<-x["mu"]  
## sigma<-x["sigma"]  
## # xinit<-x["xinit"]  
## # yinit<-x["yinit"]  
## # Xt<-rep(xinit,Ni) #set of start points (one for each humanoid)  
## # Yt<-rep(yinit,Ni) #set of start points (one for each humanoid)  
## X<-rwalk1dhumanoids( Ni=Ni, mu=mu, sigma=sigma, Xt=Xt,humanoidtype=humanoidtype,startind=startind,humanoidnames=humanoidnames,offset=offset) # get Ni rows of humanoids X co-ordinates  
## Y<-rwalk1dhumanoids(Ni=Ni, mu=mu, sigma=sigma, Xt=Yt,humanoidtype=humanoidtype,startind=startind,humanoidnames=humanoidnames,offset=offset) # get Ni rows of humanoids Y co-ordinates  
## Z<-cbind(X,Y)  
## Zdim<-c(Ni,time.ntimesteps+1,2)#humanoid,time,x-y  
## humanoidnames<-rownames(X)#paste(humanoidtype,c(1:Ni))  
## W<-array(data= Z,  
## dim = Zdim,  
## dimnames=list(humanoidnames,paste0("t", time.vtimesteps+offset),c("x-displacement","y-displacement"))  
## )  
## return(W)  
## }

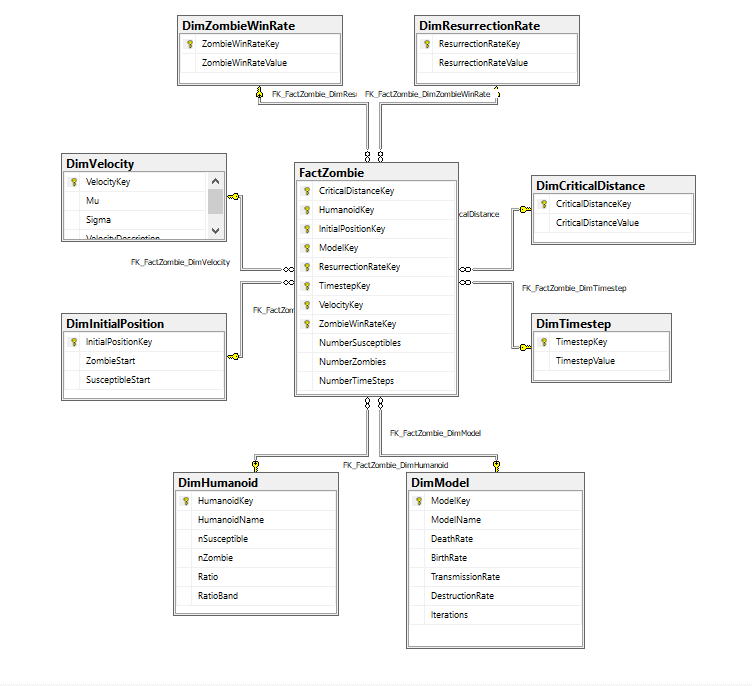
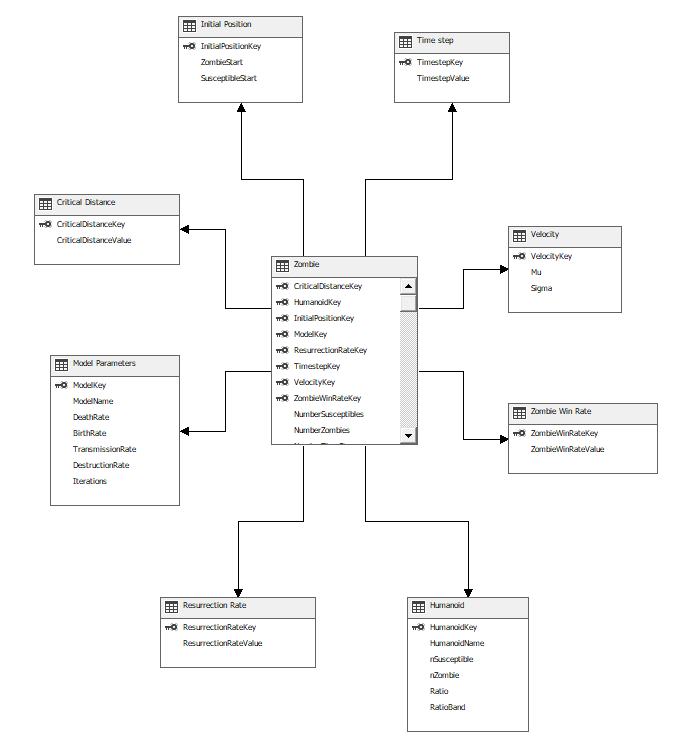
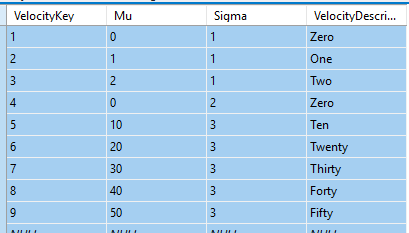
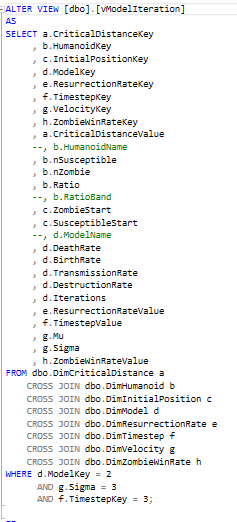
## function(cn, m.R, CriticalDistanceKey,  
## HumanoidKey,  
## InitialPositionKey,  
## ModelKey,  
## ResurrectionRateKey,  
## TimestepKey,  
## VelocityKey,  
## ZombieWinRateKey,  
## tablename="dbo.FactZombie"){  
## NumberSusceptibles<-m.R["SusceptibleCount"]  
## NumberZombies<-m.R["ZombieCount"]  
## NumberTimeSteps<-m.R["tstep"]  
## tablecols<-sqlColumns(cn, tablename)  
## varTypes <- as.character(tablecols$TYPE\_NAME)   
## names(varTypes) <- as.character(tablecols$COLUMN\_NAME)   
##   
## m.A<-c(CriticalDistanceKey,   
## HumanoidKey,   
## InitialPositionKey,   
## ModelKey,   
## ResurrectionRateKey,   
## TimestepKey,   
## VelocityKey,   
## ZombieWinRateKey,  
## NumberSusceptibles,  
## NumberZombies,  
## NumberTimeSteps)  
## m.AT<-matrix(m.A,nrow=1,ncol=length(m.A),byrow=TRUE)  
## colnames(m.AT)<- as.character(tablecols$COLUMN\_NAME)  
## m.df<-data.frame(m.AT)  
## x<-sqlSave(channel=cn,dat=m.df,tablename=tablename,fast=TRUE,append=TRUE,rownames=FALSE,varTypes = varTypes,verbose = TRUE)  
## return(x)  
##   
## }

## Appendix Six: Variables for model

CriticalDistanceKey HumanoidKey InitialPositionKey ModelKey ResurrectionRateKey TimestepKey VelocityKey ZombieWinRateKey CriticalDistanceValue nSusceptible nZombie ZombieStart SusceptibleStart DeathRate BirthRate TransmissionRate DestructionRate TimestepValue ResurrectionRateValue Mu Sigma ZombieWinRateValue Iterations

## Appendix Seven OLAP Cube for Interactions

### Appendix Seven part one: relational model

image: ##### Figure A7.1: Relational Model ###Appendix Seven part two: dimensional model image: ##### Figure A7.2: OLAP Cube ###Appendix Seven part three: DimVelocity image: ##### Figure A7.3: DimVelocity ###Appendix Seven part four: vModeliteration- *cross joins* provide cartesian product image: ##### Figure A7.4: vModeliteration (note that in this case the *where* clause is empowering different parts of the model to be isolated)

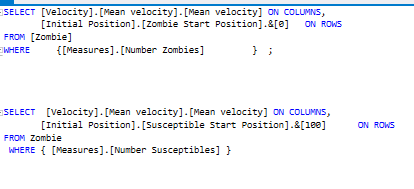
### Appendix Eight: the Engine Room

library(RODBC) cn <- odbcConnect(“LAYDSQL\_Zombie”) tabledata <- sqlFetch(cn, “vModelIteration”) m.R<-c() for (i in 1:length(rownames(tabledata))){ CriticalDistanceKey<-tabledataHumanoidKey[i] InitialPositionKey<-tabledataModelKey[i] ResurrectionRateKey<-tabledataTimestepKey[i] VelocityKey<-tabledataZombieWinRateKey[i] rcritical<-tabledatanSusceptible[i] nZ<-tabledataZombieStart[i] initS<-tabledataDeathRate[i] P<-tabledataTransmissionRate[i] a<-tabledataTimestepValue[i] G<-tabledataMu[i] sigma<-tabledataZombieWinRateValue[i] Iterations<-tabledata$Iterations[i]

time.ntimesteps<-nT #time steps time.vtimesteps<-c(0:time.ntimesteps) #set up a zero-based vector, t, that will represent time, NB unit time in steps of one # epidemiology parameters (note that these have been anglicised from Munz et al, e.g. Π is represented as P) config.model<-configuration.SRZModel(P=P,d=d,B=B,G=G,a=a,rcritical=rcritical,zombiewinratio=ZombieWinRate) config.Susceptible<-configuration.Humanoid(xinit=initS,yinit=initS,n=nS,mu=mu,sigma=sigma)# Susceptible configuration config.Zombie<-configuration.Humanoid(xinit=initZ,yinit=initZ,n=nZ,mu=mu,sigma=sigma)# Zombie configuration config.plot<-config.plot.global()#axes configuration for gif

#zombie playground :-)# m.M<-model.run.params(S.params = config.Susceptible, Z.params = config.Zombie, mode=“matrix”, iterations = c(1:1)) # m.M<-model.run.params(S.params = config.Susceptible, Z.params = config.Zombie, mode=“gif”) m.R<-colMeans(m.M) sql.addtoSQLServer(cn, m.R, CriticalDistanceKey, HumanoidKey, InitialPositionKey, ModelKey, ResurrectionRateKey, TimestepKey, VelocityKey, ZombieWinRateKey) } odbcClose(cn)

## See Appendix nine: Sample MDX

image: ##### Figure A9: MDX Queries - examples of how preliminary investigation was performed

warnings()

## NULL

1. Munz, P. et al. (2009) ‘Zombies!’, Science. [↑](#footnote-ref-1)
2. Original code as provided in assignment and avaialable at <http://scipy-cookbook.readthedocs.io/items/Zombie_Apocalypse_ODEINT.html> [↑](#footnote-ref-2)
3. Using Anglicised equivalents to Greek letters (e.g. P=Π) [↑](#footnote-ref-3)
4. A corollary to this is that the transmission rates and destruction rates are independent [↑](#footnote-ref-4)
5. This code is not identical but was inspired by the example here: <http://www.phytools.org/eqg/Exercise_4.1/> [↑](#footnote-ref-5)
6. All code samples can be found inline with the text in the attached Rmarkdown project [↑](#footnote-ref-6)
7. The function rwalk1dhumanoids [↑](#footnote-ref-7)
8. A note on space and time in this model: since all incremental steps are relative to one another, this means that, just like the Ordnance Survey, we never have to specify a unit of time or space. This however would be challenged were we to bring the real world into it, for example by including latency of infection, or arrival of weaponry. [↑](#footnote-ref-8)
9. The animation is also provided as a gif in the attached documentation and I will cry if you don’t watch it. [↑](#footnote-ref-9)
10. TODO Find reference [↑](#footnote-ref-10)
11. Figures used here slightly higher than rest of model for illustrative purposes [↑](#footnote-ref-11)
12. Munz, P. et al. (2009) ‘Zombies!’, Science. [↑](#footnote-ref-12)
13. We do not use identical parameters here, for example the β and α parameters are superceded by the **Zombie Win Rate** [↑](#footnote-ref-13)
14. One limitation of the model was the need to have at least two Zombies, hence the difference in inital amounts [↑](#footnote-ref-14)
15. Alemi, A. A., Bierbaum, M., Myers, C. R., & Sethna, J. P. (2015). You can run, you can hide: The epidemiology and statistical mechanics of zombies. Physical Review E - Statistical, Nonlinear, and Soft Matter Physics. <https://doi.org/10.1103/PhysRevE.92.052801> [↑](#footnote-ref-15)
16. Brooks, M. (2003). The zombie survival guide : complete protection from the living dead. The Hand. [↑](#footnote-ref-16)