Adrian Cross

Student ID: 1223199

School of Physics and Astronomy

Introduction to C++

Year 3 Project

2015

Simulating a Zombie Outbreak

Abstract

Using the Quincy C++ environment entities are generated which have their own individual properties and interact with each other based on these properties according to an algorithm. Using FLTK these properties can be altered in a user-friendly GUI and are displayed to a grid with different colours representing different entity types.

Introduction

The aim of this project was to create a zombie simulation with flexible inputs. This was done by creating a movement algorithm which determined the movement and interaction of the entities and then mapping these actions onto a grid display. These actions are then looped until only one type of entity remains. Each entity also has several properties, for example a skill level, which also affect these actions.

In order to structure the project logically it was decided that the code would be developed in three stages. The first version was a simple program which runs a movement algorithm for the entities and an attack algorithm for just the zombies. This is achieved by using several multi-dimensional arrays, with x and y coordinates, where each array holds a property for each coordinate. This version was written without the use of classes or FLTK and simply outputs a grid to the console.

The second version had a similar output to the first version, but significant changed the internal structure of the code. A class containing all the properties of an entity, such as its x and y coordinate, is now used. A vector containing these entities is subsequently created. This greatly reduces processing time because only the vector needs to be looked through for the movement algorithm, instead of the whole grid. Several changes have also been made to the algorithm due to the inclusion of this class.

The third version was originally intended to take inputs from FLTK, run the program from the second version, and then output to an FLTK grid; significant improvements have also been made to the code to add more functionality and to reduce processing time.

Program

Using the program

As soon as the program is run the window, shown in figure 1, and the console will display.

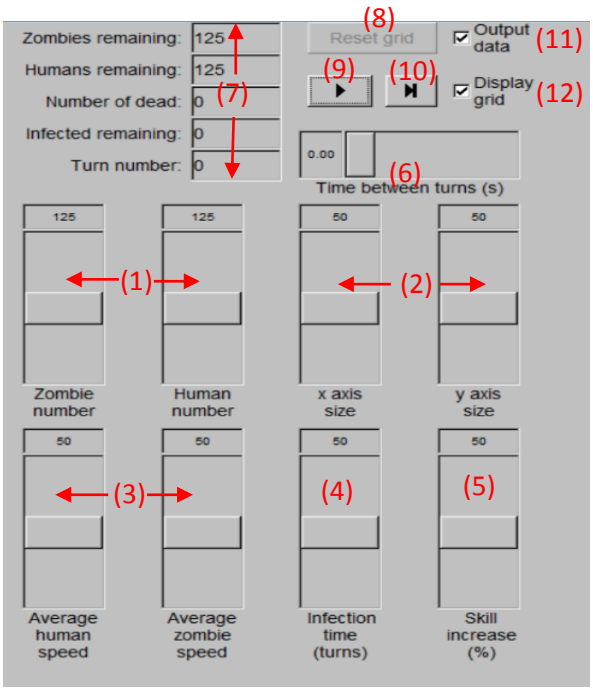

Figure 1: labelled initial display when the program is run

The console is included as it will display error messages from the code. An example of this is if the grid is too small for the number of entities, an error message will display asking you to reduce the number of zombies or humans. However if someone expands upon the program these error messages are useful for error checking.

The sliders in figure 1 are used to alter initial parameters. Altering the zombie or human number slider (1) will change the initial number of zombies or humans entered into the grid. The x-axis size and the y-axis size sliders (2) change the size of the grid axes. The size of the boxes in the grid scale with the axes sizes.

The average speed sliders (3) determine the speeds of the entities as they are input into the grid. The entities themselves have different speeds centred on the slider value which is determined in the characterchange function.

The infection time slider (4) determine the number of turns taken for the infected entities to turn into zombies.

The skill increase slider (5) determines the skill increase of a human when it kills a zombie.

The time between turns slider (6) sets the program to wait, for inputted amount of time, between each turn.

The output boxes (7) simply output the remaining number of entities as well as the turn number.

The play button (9) will run the simulation continuously whereas the step button (10) runs the simulation for one turn only and then pauses.

If the output data check button (11) is active, a text file containing the number of entities on each turn is outputted when the reset button is clicked, overwriting any previous text file. This text file is called “zombie data”. If the display grid button (12) is checked and the play or step button is clicked, the simulation grid will be displayed as shown in figure 2.

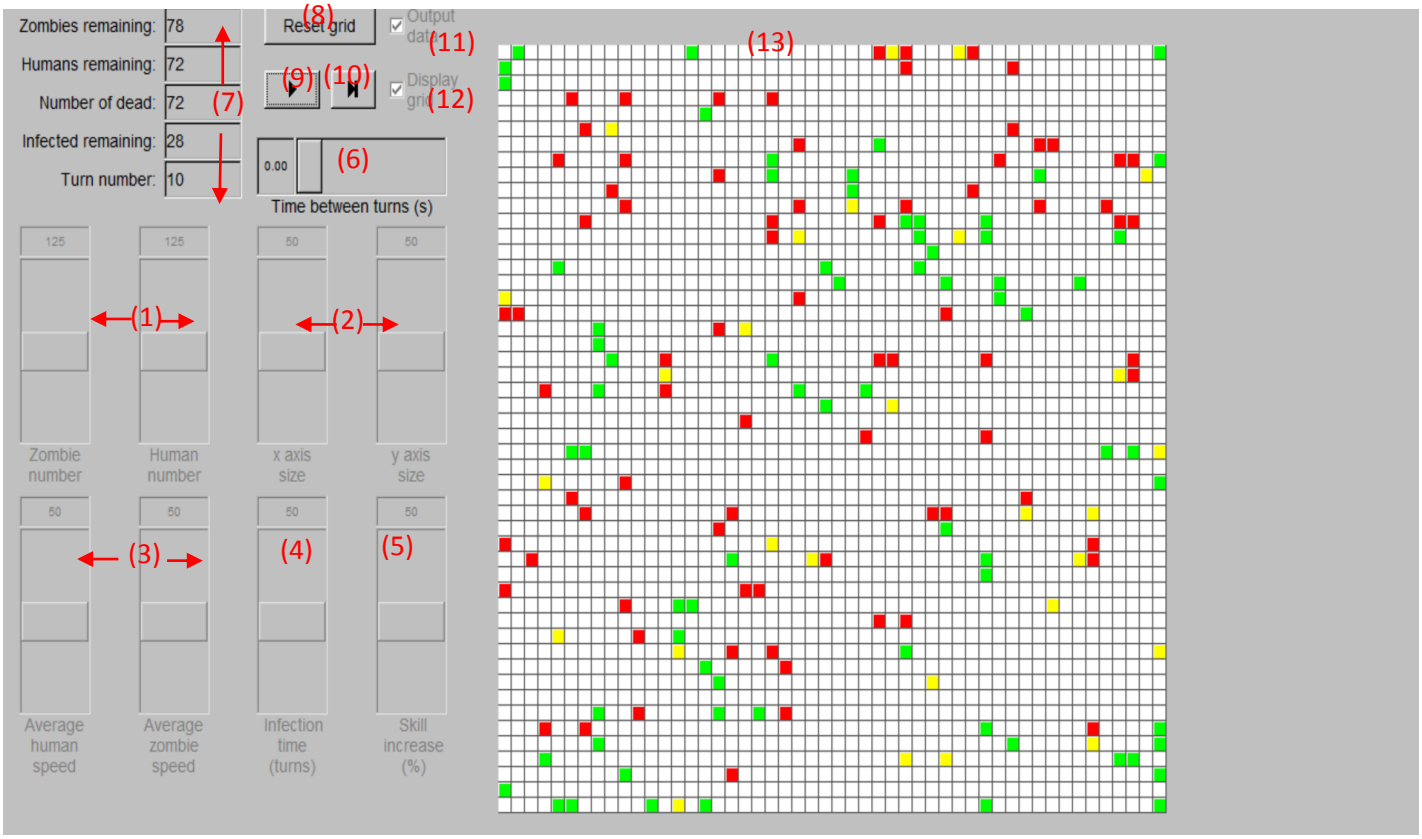

Figure 2: labelled display of program window while the simulation is running

Figure 2 shows the window when the simulation is running and the display grid button is activated. If this button is not activated the window will look the same but without the grid (13) being displayed. Each coloured square on the grid represents an entity. The red colour represents a zombie, yellow is an infected person and green is a human.

The sliders, (1, 2, 3, 4, 5), and display buttons, (11, 12), are deactivated while the simulation is running so that the program cannot receive these inputs. The reset button (8) has now been activated and when clicked returns the window to the display shown in figure 1.

How the program works

Inside the main the widgets are declared and the programloop function is set up. The program mainly runs within the programloop as it infinitely loops the program until the window is closed. This allows actions to continually take place dependant on what buttons have been pressed. If the simulation has not been started yet, the call back functions for the input sliders and check boxes will keep looping round allowing these values to be altered.

Once the simulation starts these sliders are deactivated a code is run for the initial placement of entities and creation of the grid is performed. Once this is completed it is not active again until the grid has been reset.

The algorithm for the movement of entities now occurs and is looped until the pause button is clicked, the reset button is clicked or the endgame parameters are met. Each time this happens the number of each entity type is counted and outputted to the text file. The endgame parameters are met when there is only one type of entity left and at this point only the reset button can be clicked which can be seen in figure 3.

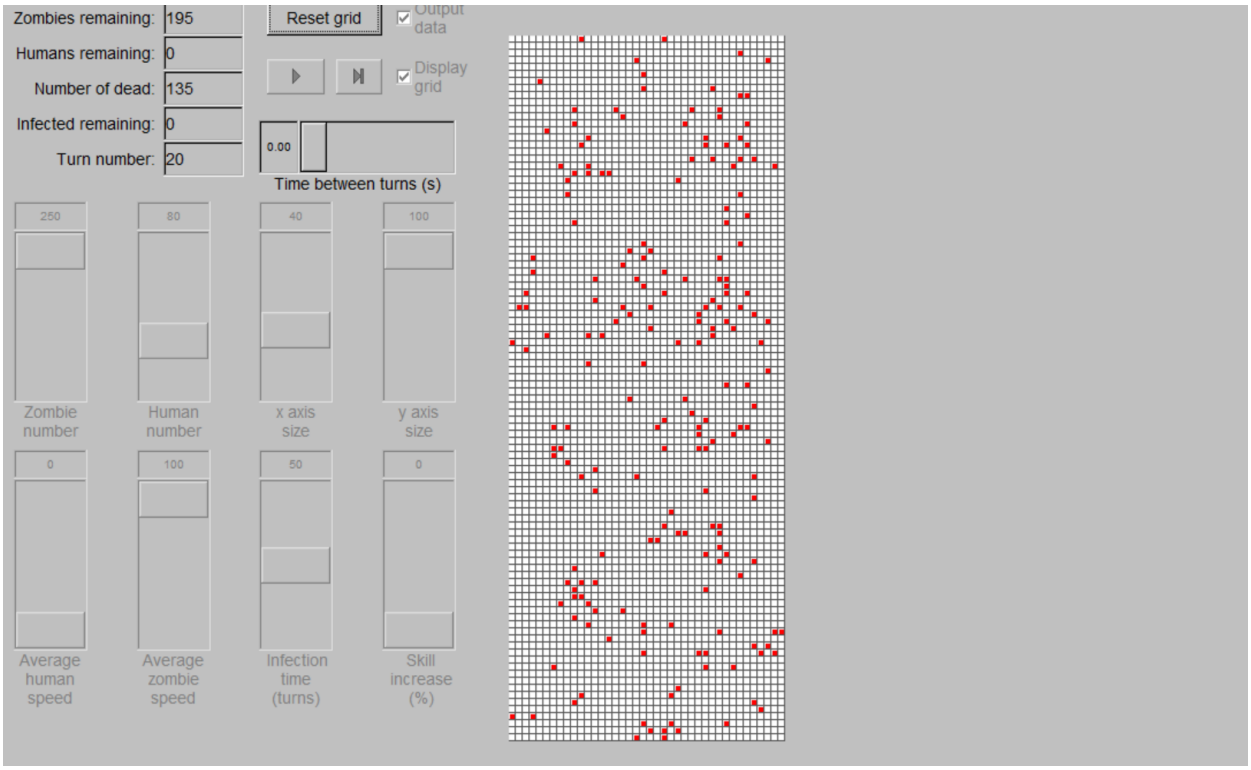

Figure 3: Endgame scenario of simulation

Project algorithm

Figure 4 shows a simplified version of the entity movement algorithm used in the code in a flowchart format. This algorithm is run each turn while the simulation is running and it performs the actions of all the entities during that turn.

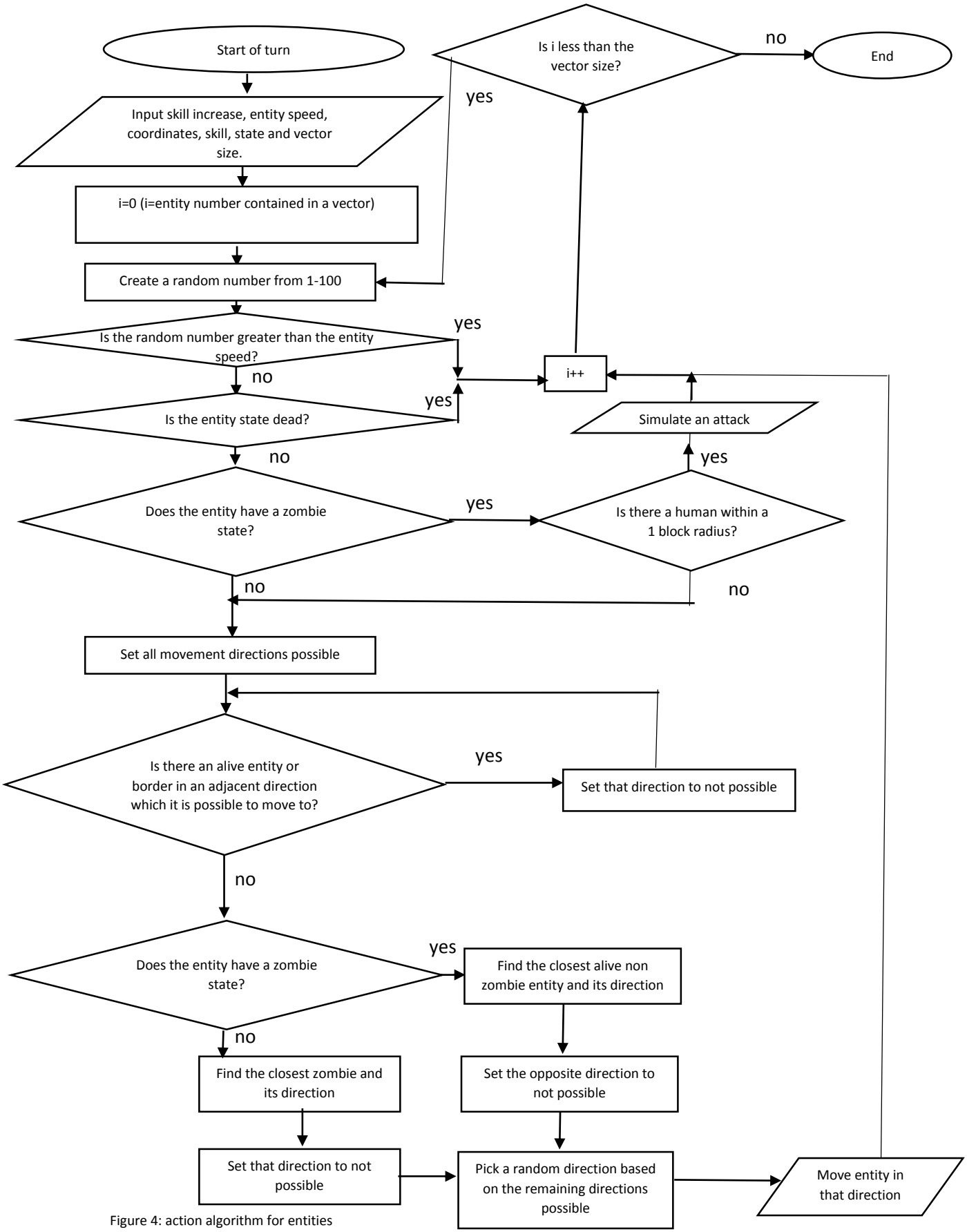

Figure 4: action algorithm for entities

Running this algorithm means that entities can now act in an algorithmic manner based on their properties as well as interact with other entities.

Limitations and improvements

The program has been designed in such a way to facilitate the expansion and alteration of the code. This is done by declaring certain input values as variables, such as the upper and lower bounds on the sliders, allowing for easy alteration. By keeping error messages included within some of the functions, mistakes can be easily tracked down if they are made during expansion. Finally by clearly structuring the code it allows it to be easily read and altered by others.

There are many ways in which this program could be improved. One such way is to process the output data into a different format. Currently only a text document containing the number of entities at each turn is outputted. By using other programs, such as OpenGl or Koolplot, a graph could be displayed instead or as well as this text document.

The way in which the entities interact is also limited in that the humans or infected can only interact with zombies and not with each other. This could be easily build upon in the code by adding to the search function used for the zombie human interactions but changing the output. For example if two humans are adjacent they could combine their skills in such a way as to improve their chances of survival.

The different types of entities could also be increased. For example having specific humans which can kill zombies from a distance could be easily implemented in the code by altering the fightorflee function in a way that causes the humans to kill the zombies from a distance instead of simply running away. Another possible type of entity is a solid block which other entities could not enter. This would simulate a building or a wall which is impassable.

As well as increasing user input the way in which the inputs occur could also be improved. One idea is that by clicking on the grid with a placement button selected would allow for entities to be manually placed.

The user input could be improved by allowing more conditions to be altered in the GUI. Variables such as the sense range of the entities could be input using a slider however the layout of the window would have to be altered to accommodate this.

There are no known bugs contained in this program.

Borrowed Code

No code was explicitly taken from other sources, however some C++ and FLTK manuals were used to assist with writing the code. These have been included in the references section.

Conclusion

By adding extras to the program, such as the text file output and the fightorflee function, this program has exceeded its original aims which were to create a zombie simulation with flexible inputs. However there is much scope for expansion which could be done in the future.

References

C++ website-//http://www.cplusplus.com/ This website was used to assist in the creation of the output file

FLTK grid creation example-//http://epweb2.ph.bham.ac.uk/user/hillier/course3cpp/fltkegs/grid/grid.cpp was used to assist in the creation of the grid.

FLTK manual- http://fltk.org/doc-1.3/fltk.pdf was used in order to create and use the widgets effectively.

FLTK website-http://www.fltk.org/ was also used in order to creation and use the widgets.

Appendix: Source code

```
//Adrian Cross
//23/01/2015
//year 3 intro to C++ project
//zombie outbreak simulation v3(final)
//This code simulates a zombie outbreak by taking inputs in an FLTK window and outputting to an FLTK grid
//this code has been written in quincy and is compiled by simply compiling the project file (included in zip file)


#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Value_Slider.H>
#include <FL/Fl_Hor_Value_Slider.H>
#include <Fl_Value_Output.H>
#include <FL/Fl_Button.H>
#include <Fl_Toggle_Button.H>
#include <FL/Fl_Box.H>//these includes are taken from the FLTK manual
#include <FL/Fl_Check_Button.H>
#include <fstream>
#include <iostream>
#include <string>
#include <algorithm>
#include <vector>
#include <windows.h>
#include <math.h>

using namespace std;

enum character {human, zombie, infected, dead, blank};


//global pointers for FLTK widgets----------------------------------------------------------------------------------------------------
----------
const int maxgridsizex=100;//need to be global to use in grid[][]
const int maxgridsizey=100;

//global pointers for FLTK widgets
Fl_Window* window;

Fl_Value_Output* zombienobox;
Fl_Value_Output* humannobox;
Fl_Value_Output* deadbox;
Fl_Value_Output* infectedbox;
Fl_Value_Output* countbox;

Fl_Toggle_Button* pausebutton;
Fl_Toggle_Button* resetbutton;

Fl_Check_Button* gridbutton;
Fl_Check_Button* textfilebutton;

Fl_Button* stepbutton;

Fl_Box* grid[maxgridsizex][maxgridsizey];

Fl_Value_Slider* zombienoslider;
```

```
Fl_Value_Slider* humannoslider;
Fl_Value_Slider* gridsizexslider;
Fl_Value_Slider* gridsizeyslider;
Fl_Value_Slider* skillincreaseslider;
Fl_Value_Slider* incubationslider;
Fl_Value_Slider* zombiespeedslider;
Fl_Value_Slider* humanspeedslider;
Fl_Hor_Value_Slider* simulationspeedslider;



//entity class which contains the information for each entity in the simulation----------------------------------------
----------------------------
class entity {
   private:
                        character state;
                        bool checked;      //temporary state to determine if the character has moved that turn
                        int skill;
                        float speed;
                        int x;
                        int y;
                        int turnsinfected;


   public:
                        // Constructor, with defaults
                        entity(character statein=blank, bool checkedin=false,int skillin=0,
                        int speedin=0, int xin=0, int yin=0, int turnsinfectedin=0);


                        //access functions prototypes
                        character getstate();
                        bool getchecked();
                        int getskill();
                        float getspeed();
                        int getx();
                        int gety();
                        int getturnsinfected();


                        // member function prototypes
                        void charactergen(character, int, int, vector<entity>&, int, int);
                        void characterchange(int, int, character, int,int);
                        void checkedreset();
                        void movementchance();
                        void action(vector<entity>&, int, int, int);
                        void attacksearch(bool&, bool&,bool&, bool&, vector<entity>&, int);
                        void simulateattack(vector<entity>&, int, int);
                        void increaseskill(int);
                        void adjacentcheck(bool&, bool&,bool&, bool&, vector<entity>&);
                        void bordercheck(int,int, bool&, bool&, bool&, bool&);
                        void fightorflee(bool& ,bool& ,bool& ,bool& , vector<entity>&);
                        int directionreturn(bool, bool,bool, bool);
                        void movement(int);
                        void turntozombie(int,int);
                        void counter (int&, int&, int&, int&);
};
```

```
//callback function prototypes-------------------------------------------------------------------------------------------------------------
---------------
void outputbox_cb          (Fl_Value_Output* w, int data);
void pausebutton_cb        (Fl_Button* w, bool& buttonstate);
void resetbutton_cb        (Fl_Button* w, bool& buttonstate);
void checkbutton_cb        (Fl_Check_Button* w, bool& buttonstate);
void stepbutton_cb         (Fl_Button* w, bool& buttonstate);
void slider_cb             (Fl_Value_Slider* w, int& data);
void sliderspeed_cb        (Fl_Value_Slider* w, float& data);
void grid_cb               (int gridsizex,int  gridsizey, vector<entity>& id);



//general function prototypes---------------------------------------------------------------------------------------------------------------
---------------
void programloop(void* data);

void onestep(int& humanno, int& zombieno,int& infectedno, int& deadno, int gridsizex,
                              int gridsizey,vector<entity>& id,int skillincrease,int incubation, int zombiespeed);

void reinitialise(bool& play, int& count, bool& reset, vector<entity>& id, int& deadno,
                                        int& infectedno, float& timeresolution, int gridsizex, int gridsizey);

void charactergenloop(int initialhno,int initialzno, vector<entity>& id, int gridsizex,
                                                int gridsizey, int humanspeed, int zombiespeed);

bool blankcheck(int x, int y, vector<entity> id);

void generategrid(int gridsizex,int gridsizey);

void hidegrid(int gridsizex, int gridsizey);

bool endgame(int humanno,int  zombieno, int infectedno);



//start of main-----------------------------------------------------------------------------------------------------------------------------
int main() {

   srand(time(0));

         //creation of FLTK window
   window                  =          new       Fl_Window                          (Fl::w(),Fl::h()-30,"Zombie
pandemic simulation");

         //creation of FLTK output boxes
   zombienobox =           new Fl_Value_Output              (150,0,   70,       30,"Zombies remaining: ");
   humannobox   =          new Fl_Value_Output              (150,30,70,         30,"Humans remaining: ");

   deadbox         =       new Fl_Value_Output              (150,60,70,         30,"Number of dead: ");
   infectedbox =   new Fl_Value_Output              (150,90,70,         30,"Infected remaining: ");
   countbox        =       new Fl_Value_Output              (150,120,70, 30,"Turn number: ");

         //creation of FLTK buttons
         pausebutton =      new Fl_Toggle_Button      (240,50,50, 30,"@||");
```

```
        stepbutton      =       new Fl_Button                   (300,50,40, 30,"@>|");
resetbutton =   new Fl_Toggle_Button    (240,0, 100,30, "Reset grid");
        textfilebutton =  new Fl_Check_Button       (350,0, 80,30,"Output\ndata");
        gridbutton      =       new Fl_Check_Button     (350,50,80, 30,"Display\ngrid");

        textfilebutton->value(1);
        gridbutton->value(1);
        resetbutton->deactivate();

//creation of FLTK sliders
        int maxzombienumber=250;
        int minzombienumber=1;
zombienoslider = new Fl_Value_Slider (20,171,64,171,"Zombie\nnumber");
zombienoslider->Fl_Slider::scrollvalue(1,1,100,100);//this sets the slider to go up in integer steps
zombienoslider->maximum(minzombienumber);
zombienoslider->minimum(maxzombienumber);
        zombienoslider->value((maxzombienumber+minzombienumber)/2);//sets default value

        int maxhumannumber=250;
        int minhumannumber=1;
humannoslider = new Fl_Value_Slider (127,171,64,171,"Human\nnumber");
humannoslider->Fl_Slider::scrollvalue(1,1,100,100);
humannoslider->maximum(minhumannumber);
humannoslider->minimum(maxhumannumber);
humannoslider->value((maxhumannumber+minhumannumber)/2);

        int mingridsizex=1;
gridsizexslider = new Fl_Value_Slider (233, 171,64,171,"x axis\nsize");
gridsizexslider->Fl_Slider::scrollvalue(1,1,100,100);
gridsizexslider->maximum(mingridsizex);
gridsizexslider->minimum(maxgridsizex);
gridsizexslider->value((mingridsizex+maxgridsizex)/2);

        int mingridsizey=1;
gridsizeyslider = new Fl_Value_Slider (340,171,64,171,"y axis\nsize");
gridsizeyslider->Fl_Slider::scrollvalue(1,1,100,100);
gridsizeyslider->maximum(mingridsizey);
gridsizeyslider->minimum(maxgridsizey);
gridsizeyslider->value((mingridsizey+maxgridsizey)/2);

        int maxskillincrease=100;
        int minskillincrease=0;
skillincreaseslider = new Fl_Value_Slider(340, 382,64,171," Skill\n increase\n (%)");
skillincreaseslider->Fl_Slider::scrollvalue(1,1,100,100);
skillincreaseslider->maximum(minskillincrease);
skillincreaseslider->minimum(maxskillincrease);
        skillincreaseslider->value((minskillincrease+maxskillincrease)/2);

        int maxincubation=100;
        int minincubation=0;
incubationslider = new Fl_Value_Slider (233, 382,64,171," Infection\ntime\n(turns)");
incubationslider->Fl_Slider::scrollvalue(1,1,100,100);
incubationslider->maximum(minincubation);
incubationslider->minimum(maxincubation);
        incubationslider->value((minincubation+maxincubation)/2);

        float maxzombiespeed=100;
```

```
        float minzombiespeed=0;
   zombiespeedslider = new Fl_Value_Slider (127, 382,64,171,"Average\nzombie\nspeed");
   zombiespeedslider->Fl_Slider::scrollvalue(1,1,100,100);
        zombiespeedslider->maximum(minzombiespeed);
   zombiespeedslider->minimum(maxzombiespeed);
        zombiespeedslider->value((minzombiespeed+maxzombiespeed)/2);

   float maxhumanspeed=100;
        float minhumanspeed=0;
   humanspeedslider = new Fl_Value_Slider (20, 382,64,171,"Average\nhuman\nspeed");
        humanspeedslider->Fl_Slider::scrollvalue(1,1,100,100);
   humanspeedslider->maximum(minhumanspeed);
   humanspeedslider->minimum(maxhumanspeed);
        humanspeedslider->value((minhumanspeed+maxhumanspeed)/2);

        float maxsimulationspeed=1;
        float minsimulationspeed=0;
   simulationspeedslider = new Fl_Hor_Value_Slider (233, 102,171,46,"Time between turns (s)");
   simulationspeedslider->maximum(maxsimulationspeed);
   simulationspeedslider->minimum(minsimulationspeed);
        simulationspeedslider->value(minsimulationspeed);

   Fl::add_timeout( 0, programloop); //runs the main timer loop for the program

   window->show();
   window->end();
   return Fl::run();

}


//functions of entity class-----------------------------------------------------------------------------------------------------------------
-------------
//input initial into entity class
entity::entity(character statein, bool checkedin, int skillin, int speedin, int xin, int yin, int turnsinfectedin){
   state=statein;
   checked=checkedin;
   skill=skillin;
   speed=speedin;
   x=xin;
   y=yin;
   turnsinfected=turnsinfectedin;
}


//access functions of entity class
character entity::getstate()    {return state;}
bool entity::getchecked()       {return checked;}
int entity::getskill()          {return skill;}
float entity::getspeed()        {return speed;}
int entity::getx()              {return x;}
int entity::gety()              {return y;}
int entity::getturnsinfected()    {return turnsinfected;}


//member functions of entity class
//Charactergen function checks if a coordinate is empty and will run character change function if it is
```

```
void entity::charactergen(character state, int gridsizex, int gridsizey,vector<entity>& id, int humanspeed, int
zombiespeed){
   int w=0;
   while (w==0){
      int xdummy=rand() % gridsizex;
      int ydummy=rand() % gridsizey;
      if (blankcheck(xdummy, ydummy, id)==true){
                                 characterchange(xdummy, ydummy, state,humanspeed,zombiespeed);

                                 return;
      }
   }
}


//characterchange function will generate a character
//with inputted x coord, y coord, state and a speed based on the average speed
void entity::characterchange(int i, int j, character a, int humanspeed, int zombiespeed){

   checked=true;

   int maxskillgenerated=40;

            //set speedspread to zero if you want all entities of the same type to have the same speed
            int speedspread=20;

            //will produce a random number between -speedspread and speedspread
            int randomspeedspread=(rand()% (speedspread*2)+1)-20;

            state=a;
   x=i;
   y=j;

            switch (a){
      case human:
         skill=(rand() % maxskillgenerated)+1;
         speed=humanspeed+randomspeedspread;
         break;
      case zombie:
         speed=zombiespeed+randomspeedspread;
         break;
      case infected:
         break;
      default:
         cout<<"error in characterchange function"<<endl;
   }
}


//checkedreset function sets checked=false for all entities except dead
void entity::checkedreset(){
            if (state!=dead){
                        checked=false;
            }
}
```

```
//movementchance function will stop entity from acting if its speed is lower than a random number from 1 to
100
void entity::movementchance(){
   int a= (rand() % 100)+1;
   if (speed<=a){
      checked=true;
   }
}


//action function will make the entity move or attack depending on where the entity is and what surrounds it
void entity::action(vector<entity>& id, int skillincrease, int gridsizex, int gridsizey){

   bool up=true, right=true, down=true, left=true;//sets all directions possible which will change in the called
functions

   attacksearch(up, right, down, left, id, skillincrease);

   if (checked==true){
      return;
   }
         adjacentcheck(up, right, down, left, id);

   bordercheck(gridsizex,gridsizey,up,right,down,left);

         fightorflee(up, right, down, left, id);

   movement(directionreturn(up, right, down, left));

   return;
}


//attacksearch will search surrounding spots for entities and attack if it's possible
void entity::attacksearch(bool &up, bool &right,bool &down, bool &left, vector<entity>& id, int skillincrease){

   for (int i=0; i<id.size(); i++){
      int xdiff=x-id[i].x, ydiff=y-id[i].y;
      switch (state){
         case zombie:
                              //will simulate an attack if within a 1 block radius
            if (xdiff<2 && xdiff>-2 && ydiff<2 && ydiff>-2 && (id[i].state==(human) or id[i].state==infected)){
               simulateattack(id, i, skillincrease);
               checked=true;
               id[i].checked=true;
               return;
            }
            break;
      }
   }
}


//simulate attack function will alter entity values depending on which entity wins the fight
void entity::simulateattack(vector<entity>& id, int i, int skillincrease){

   int a=((rand() % 100)+1);
```

```
        int zdeathlowerbound=a+10;
        int zdeathinfectionlowerbound=a-5;
        int hinfectionlowerbound=a-40;

        if (checked==true){//checks to see if the zombie has attacked already
        return;
  }

        switch(id[i].state){
                case infected:
                case human:
                        if (id[i].skill>zdeathlowerbound){
                state=dead;//just zombie dies, skill of human increases
        id[i].increaseskill(skillincrease);
                        }

                        else if (id[i].skill<=zdeathlowerbound && id[i].skill>=zdeathinfectionlowerbound){
                        state=dead;//zombie dies, human infected
                id[i].state=infected;
                id[i].increaseskill(skillincrease);
                }

                        else if (id[i].skill<zdeathinfectionlowerbound && id[i].skill>hinfectionlowerbound){
                id[i].state=infected;//human is infected, zombie lives
                id[i].increaseskill(skillincrease);
                }

                        else if (id[i].skill<=hinfectionlowerbound){//human dies, zombie lives
        id[i].state=dead;
                }

        else cout<<"error in simulateattack function";
        }
}


//skillincrease increases the skill of a character based on an upper limit and the skillincrease input
void entity::increaseskill(int skillincrease){
        int maxskill=99;
  if (skill+skillincrease>maxskill){
     skill=maxskill;
     }
  else skill=skill+skillincrease;
}


//adjacentcheck checks adjacent spots for entities and stops movements in directions where the spot is
occupied
void entity::adjacentcheck(bool &up, bool &right,bool &down, bool &left, vector<entity>& id){

  for (int i=0; i<id.size(); i++){

     int xdiff=x-id[i].x, ydiff=y-id[i].y;

     if (xdiff==0 && ydiff==1 && id[i].state!=dead){
        up=false;
```

```
      }

      if (xdiff==-1 && ydiff==0 && id[i].state!=dead){
         right=false;
      }

      if (xdiff==0 && ydiff==-1 && id[i].state!=dead){
         down=false;
      }

      if (xdiff==1 && ydiff==0 && id[i].state!=dead){
         left=false;
      }
   }
}


//bordercheck function sets directions to false if entities are on a border
void entity::bordercheck(int gridsizex,int gridsizey,bool &up, bool &right, bool &down, bool &left){
   if (y==0){
      up=false;
   }
   if (x==0){
      left=false;
   }
   if (x==gridsizex-1){
      right=false;
   }
   if (y==gridsizey-1){
      down=false;
   }
}


//fightorflee function forces zombies to run to humans
//within a specific range and humans to run away from zombies in a specific range
void entity::fightorflee(bool& up,bool& right,bool& down,bool& left, vector<entity>& id){


         if (up+right+down+left<=1){//ends the function if the entity can only move in 1 direction
                  return;
         }
         int zombiesenserange=10;
         int humansenserange=10;
         float senserange;

         //picks the correct sense range for the entity
         switch (state){
                  case zombie:
                           senserange=zombiesenserange;
                           break;
                  case infected:
                  case human: senserange=humansenserange;
         }

         float mindistance=senserange;
         int closestentity=0;
```

```
	int minxdistance=0;
	int minydistance=0;

	//this loop finds the closest entity to itself with a different state
	for (int i=0; i<id.size(); i++){

		float xdiff=x-id[i].x, ydiff=y-id[i].y;

		float distance=sqrt(pow(xdiff,2)+pow(ydiff,2));
		bool differentstate=false;


		if ((state==zombie && (id[i].state==human or id[i].state==infected))
			or (state==human or state==infected) && (id[i].state==zombie)){
			differentstate=true;//sets differentstate to true if one entity is a zombie and the
other is human/infected
		}

		if (distance!=0 && distance<mindistance && differentstate==true){ //records the closest
entity
			mindistance=distance;
			minxdistance=xdiff;
			minydistance=ydiff;
			closestentity=i;
		}
	}

	if (minxdistance==0 && minydistance==0){//returns if no entity is less than the senserange
	return;
	}

	//sets the entity to run away or run towards the closest entity
	switch (state){
		case zombie:
			if (minxdistance<0 && (right+down+up)!=0){
			left=false;
			}
			if (minxdistance>0 && (left+down+up)!=0){
				right=false;
			}
			if (minydistance>0 && (left+up+right)!=0){
				down=false;
			}
			if (minydistance<0 && (left+down+right)!=0){
				up=false;
			}
			if (minxdistance==0 && (up+down)!=0){
				right=false;
				left=false;
			}
			if (minydistance==0 && (right+left)!=0){
				up=false;
				down=false;
			}
			break;
		case infected:
		case human:		//does the same as the zombie case but in the opposite direction
```

```
                        if (minxdistance<0 && (left+down+up)!=0){
                                right=false;
                        }
                        if (minxdistance>0 && (right+down+up)!=0){
                                left=false;
                        }
                        if (minydistance>0 && (left+down+right)!=0){
                                up=false;
                        }
                        if (minydistance<0 && (left+up+right)!=0){
                                down=false;
                        }
                        if (minxdistance==0 && (up+down)!=0){
                                right=false;
                                left=false;
                        }
                        if (minydistance==0 && (right+left)!=0){
                                up=false;
                                down=false;
                        }
        }
}

// directionreturn function will return a random number based upon the state of up,right,down,left
int entity::directionreturn(bool up, bool right,bool down, bool left){

        if (up==false and right==false and down==false and left==false){//returns 0 if entity cannot move
                return 0;
        }
        int w=0;
        while (w==0){

                int randomno=(rand() % 4)+1;

                switch((rand() % 4)+1){
                        case 1:
                                if(up==true){
                                        return 1;
                                }
                                break;

                        case 2:
                                if(right==true){
                                        return 2;
                                }
                                break;

                        case 3:
                                if(down==true){
                                        return 3;
                                }
                                break;

                        case 4:
                                if(left==true){
                                        return 4;
```

```
                                        }
                                        break;
                        }
                }
}


//movement function physically moves the entity depending on input
void entity:: movement(int a){
   checked=true;
   switch(a){
      case 0:
         return;
         break;
      case 1://up
         y--;
         return;
         break;
      case 2://right
         x++;
         return;
         break;
      case 3://down
         y++;
         return;
         break;
      case 4://left
         x--;
         return;
         break;
      default:
         cout<<"error in movement function"<<endl;

   }
}


//turntozombie function turn infected into zombie
void entity::turntozombie(int incubation, int zombiespeed){
   if (state!=infected){
            return;
   }
            if (turnsinfected>=incubation){
            characterchange(x, y, zombie,0.0,zombiespeed);
            return;
            }
            else{
                        turnsinfected++;
            return;
   }
}


//counter adds 1 to the total number of entities in its state
void entity::counter(int& humanno, int& zombieno, int& infectedno, int& deadno){
   switch(state){
      case human:
```

```
            humanno++;
            break;
        case zombie:
            zombieno++;
            break;
        case infected:
            infectedno++;
            break;
        case dead:
            deadno++;
    }
}


//callback functions-----------------------------------------------------------------------------------------------------------------
//outputbox_cb sets the output box value to the inputted value
void outputbox_cb(Fl_Value_Output* w, int data){
    w->value(int(data));
}


//pausebutton_cb returns the state of the button as a bool type and changes the image on the button
void pausebutton_cb(Fl_Button* w, bool& buttonstate){

    if (w->value()==1){
                    stepbutton->deactivate();
        buttonstate=true;
            w->label("@||");
    }
            else if (w->value()==0){
                    stepbutton->activate();
        buttonstate=false;
            w->label("@>");
    }
}


//resetbutton_cb returns the state of the button as a bool type and also turns the
//pausebutton off if the resetbutton is on
void resetbutton_cb(Fl_Button* w, bool& buttonstate){
    if (w->value()==1){
                    pausebutton->value(0);
        buttonstate=true;
    }
            else if (w->value()==0){
        buttonstate=false;
    }
}


//checkbutton_cb returns the state of the checkbutton as a bool type
void checkbutton_cb(Fl_Check_Button* w, bool& buttonstate){
            if (w->value()==1){
                    buttonstate=true;
            }
            else if (w->value()==0){
        buttonstate=false;
```

```
  }
}


//checkbutton_cb returns the state of the button as a bool type
void stepbutton_cb(Fl_Button* w,bool& buttonstate){
        if (w->value()==1){
                int sleeptime=100;
                buttonstate=true;
                Sleep(sleeptime); //sleep stops user from accidently stepping more that 1 step
        }
        else if (w->value()==0){
    buttonstate=false;
  }
}


//returns the value of the slider as an integer value
void slider_cb(Fl_Value_Slider* w, int& data){
  data=w->value();
}


//returns the value of the slider as a float value
void sliderspeed_cb(Fl_Value_Slider* w, float& data){
  data=w->value();
}


//grid_cb function will cycle through all coordinates and will change
//the button colour depending on the state of the entity occupying that spot
void grid_cb(int gridsizex, int gridsizey, vector<entity>& id){

        //sets all boxes white
        for (int j=0; j<gridsizey; j++){
                for (int i=0; i<gridsizex; i++){
                        grid[i][j]->color(FL_WHITE);
                        grid[i][j]->redraw();
                }
        }
        //sets the colour of box dependant on the character
        for (int i=0; i<id.size(); i++){
                switch (id[i].getstate()){
                                case human:
                                        grid[id[i].getx()][id[i].gety()]->color(FL_GREEN);
                                        grid[id[i].getx()][id[i].gety()]->redraw();
                                        break;

                                case zombie:
                                        grid[id[i].getx()][id[i].gety()]->color(FL_RED);
                                        grid[id[i].getx()][id[i].gety()]->redraw();
                                        break;

                                case infected:
                                        grid[id[i].getx()][id[i].gety()]->color(FL_YELLOW);
                                        grid[id[i].getx()][id[i].gety()]->redraw();
                                        break;
```

```
                }
        }
}


//normal functions-----------------------------------------------------------------------------------------------------------------------
----------

//The programloop function infinitely loops round and allows the program to run
void programloop(void* data){

        static ofstream myfile;

        static bool reset=true; //need to be static to stop variables being reinitialised on every loop.
        static vector<entity> id(humannoslider->minimum()+zombienoslider->minimum());
   static bool play, textfileon, gridon, step;
   static int gridsizex,gridsizey, incubation, skillincrease,
                                humanno, zombieno, infectedno, deadno, count,humanspeed, zombiespeed;
   static float timeresolution;

        //these callback functions are always active
        pausebutton_cb(pausebutton, play);
   outputbox_cb(countbox, count);
   outputbox_cb(zombienobox, zombieno);
   outputbox_cb(humannobox, humanno);
   outputbox_cb(infectedbox, infectedno);
   outputbox_cb(deadbox, deadno);
        stepbutton_cb(stepbutton, step);
   resetbutton_cb(resetbutton, reset);
        checkbutton_cb(gridbutton, gridon);
        checkbutton_cb(textfilebutton,textfileon);
        stepbutton_cb(stepbutton, step);


        //this loop is only active before the simulation is run
        if (play==false && count==0){
                slider_cb(gridsizexslider, gridsizex);
                slider_cb(gridsizeyslider, gridsizey);
                slider_cb(humannoslider, humanno);
        slider_cb(zombienoslider, zombieno);
                slider_cb(zombiespeedslider, zombiespeed);
        slider_cb(humanspeedslider,humanspeed);
                slider_cb(skillincreaseslider, skillincrease);
        slider_cb(incubationslider, incubation);
        }


        //stops the simulation if endgame parameters are met
        if (endgame(humanno, zombieno, infectedno)==true){
                pausebutton->value(0);
                pausebutton->deactivate();
                stepbutton->deactivate();

        }


        //reintialises the grid if the reset button is pressed
```

```
        if (reset==true){
    reinitialise( play, count, reset, id, deadno, infectedno, timeresolution,gridsizex, gridsizey);
                myfile.close();// text file is outputted when the reset button is pressed
  }


        //performs an error check on parameters
        if (play==true && (gridsizey*gridsizex)<=(humanno+zombieno)){
                cout<<"There are too many entities to fit into the grid."<<endl<<
                        "Please reduce the zombie or human number."<<endl;
                play=false;
                pausebutton->value(0);
        }


        //allows the use of the step button
        if (step==true){
                play=true;
                timeresolution=0;
        }


        //this will only run on the first loop (just after play/pause button is clicked)
        if (play==true && count==0){

                myfile.open ("zombiedata.txt");//opens the text file
                myfile<<"Turn"<<"\t";
                myfile<<"zombie number"<<"\t";
                myfile<<"human number"<<"\t";
                myfile<<"infected number"<<"\t";
                myfile<<"dead number"<<"\t"<<endl;

                zombienoslider->deactivate();//deactivates initial parameters widgets as they are no longer
required
    humannoslider->     deactivate();
    gridsizexslider->deactivate();
    gridsizeyslider->deactivate();
    zombiespeedslider->deactivate();
    humanspeedslider->deactivate();
                skillincreaseslider->deactivate();
                incubationslider->deactivate();
                gridbutton->deactivate();
                textfilebutton->deactivate();

                resetbutton->activate();

                charactergenloop(humanno, zombieno, id,gridsizex,gridsizey,humanspeed,zombiespeed);

                if (gridon==true){
                        generategrid(gridsizex, gridsizey);
                        grid_cb(gridsizex, gridsizey, id);
                }
                if (step==true){
                        play=false;
                }
                else{
                        sliderspeed_cb(simulationspeedslider,timeresolution);
```

```
		}
	}


		//will repeat as long as the play button is pressed
  if (play==true && count!=0){

    onestep(humanno,zombieno,infectedno,deadno,gridsizex, gridsizey, id,skillincrease,incubation,
zombiespeed);


		if (step==false){
			sliderspeed_cb(simulationspeedslider,timeresolution);
		}

		if (step==true){//this stops the program from looping if the step button is pressed
			play=false;
		}

		if(gridon==true){
			grid_cb(gridsizex, gridsizey, id);
		}
  }


	//increases count and records information to an output file
	if (step==true or play==true){
		count++;
		myfile<<count<<"\t";
		myfile<<zombieno<<"\t"<<"\t";
		myfile<<humanno<<"\t"<<"\t";
		myfile<<infectedno<<"\t"<<"\t";
		myfile<<deadno<<"\t"<<endl;
	}


	Fl::repeat_timeout(timeresolution, programloop);//repeats the function until the program is closed

}


//onestep function performs 1 count in the simulation by looping round the vector and perform different tasks
void onestep(int& humanno, int& zombieno,int& infectedno, int& deadno, int gridsizex,int
gridsizey,vector<entity>& id,
			int skillincrease,int incubation, int zombiespeed){

  humanno=0;
  zombieno=0;
  infectedno=0;
	deadno=0;

	//loop to run the movement and interaction of entities
  for (int i=0; i<id.size(); i++){
    id[i].checkedreset();
    id[i].movementchance();
    if (id[i].getchecked()==false){
      id[i].action(id, skillincrease, gridsizex, gridsizey);
```

```
    }
  }

        //loop to turn infected into zombies
        for (int i=0; i<id.size(); i++){
                id[i].turntozombie(incubation, zombiespeed);
        }

        //loop to count entities and delete dead entities
        for (int i=0; i<id.size(); i++){
                id[i].counter(humanno, zombieno,infectedno,deadno);

        }

}


//reinitialise function resets values and reactivates certain widgets when the reset button is pressed
void reinitialise(bool& play, int& count, bool& reset, vector<entity>& id, int& deadno, int& infectedno,
                                        float& timeresolution, int gridsizex, int gridsizey){

        hidegrid(gridsizex, gridsizey);
        timeresolution=0;
        id.clear();
  id.resize(humannoslider->minimum()+zombienoslider->minimum()); //resets the vector size
  reset=false;
        play=false;
  count=0;
  deadno=0;
        infectedno=0;

        zombienoslider->activate();
        humannoslider->activate();
        gridsizexslider->activate();
        gridsizeyslider->activate();
        zombiespeedslider->activate();
        humanspeedslider->activate();
        skillincreaseslider->activate();
        incubationslider->activate();
        textfilebutton->activate();
        gridbutton->activate();
        pausebutton->activate();
        stepbutton->activate();

        resetbutton->value(0); //resets the reset button
        resetbutton->deactivate();
}


//charactergenloop function loops the vector values and generates a character in each one
void charactergenloop(int humanno,int zombieno, vector<entity>& id, int gridsizex,
                                                int gridsizey, int humanspeed, int zombiespeed){
  for (int i=0; i<humanno; i++){
    id[i].charactergen(human, gridsizex, gridsizey, id, humanspeed, zombiespeed);
  }
   for (int i=humanno; i<humanno+zombieno; i++){
    id[i].charactergen(zombie, gridsizex, gridsizey, id,humanspeed, zombiespeed);
```

```
   }
}


//blankcheck function will return if a coordinate is occupied or not
bool blankcheck(int x, int y, vector<entity> id){
   for (int i=0; i<id.size(); i++){
      if (id[i].getx()==x && id[i].gety()==y){
         return false;
      }
   }
   return true;
}


//generategrid function generates and draws a grid of many small boxes dependant on the gridsizes
void generategrid(int gridsizex, int gridsizey){
            window->begin();

            for (int j=0; j<gridsizey; j++){
                        for (int i=0; i<gridsizex; i++){

                                    int scalefactorx=(860)/gridsizex;//used to scale box size dependant on the size of
the grid
                                    int scalefactory=(623)/gridsizey;
                                    int scalefactor=min(scalefactory,scalefactorx);
                                    grid[i][j]= new Fl_Box(FL_THIN_UP_BOX,
450+i*scalefactor,30+j*scalefactor,scalefactor,scalefactor,"");
                        }
            }

            window->end();
            window->redraw();
}


//hidegrid function hides the grid
void hidegrid(int gridsizex, int gridsizey){

            for (int j=0; j<gridsizey; j++){
                        for (int i=0; i<gridsizex; i++){
                                    grid[i][j]->hide();
                        }
            }
}


//endgame function will return true when the simulation should end
bool endgame(int humanno,int zombieno,int infectedno){
            if ((humanno+infectedno)==0 or (zombieno+infectedno)==0){
                        return true;
            }
            else return false;
}
```