So You Think You Can Survive - User's and Developer's Guide

By: Eric Su, Victoria Yuan, and Maggie Han

USER'S GUIDE

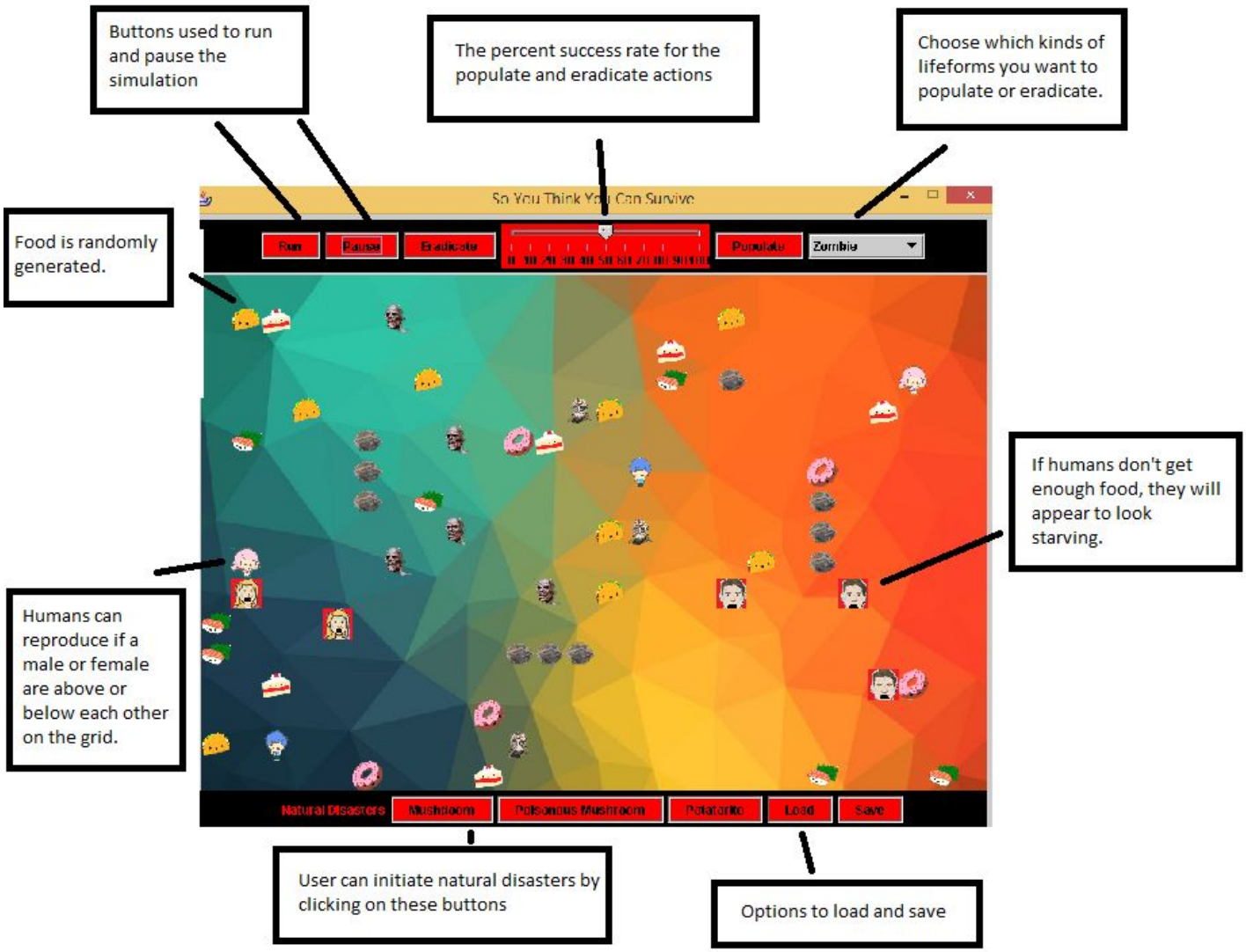
Background

So You Think You Can Survive is an innovative and creative game which incorporates unique life forms. It includes an equilibrium as well as an extinction mode of gaming. This fun game has the user controlling almost every aspect of it while they are able to observe reproduction and death patterns of each lifeform.

Getting Started

So You Think You Can Survive has been programmed using Ready to Program and it is fully compatible with this Java IDE. Ready to Program can be downloaded online for free. Upon opening the .zip file, there should be a folder full of images and a java file labelled Main. Open the Main.java file and then run the program in Ready to Program to begin the simulation. The user will be greeted with the main screen of the program, allowing the user to choose between three different modes: normal mode, extinction mode, and survival mode. The last two modes are two presets settings of the program that display especially interesting interactions between the lifeforms. Within the simulation itself, the user will see a bar of buttons on top that allow them to pause or run the simulation and populate the environment with as many of whichever lifeform they desire. There is also a bottom bar of buttons that offer ways of initiating natural disasters within the environment. These natural disasters will kill off lifeforms.

Features of the Program



In the extinction mode of gaming, users will be randomly generated a 5% density board where they may begin their game. After clicking “Run”, the user will be to observe the lifeforms of Humans, Zombies, and [Zombie] Hunters moving around the screen. The user will also be able to “Pause” the game at any given moment to freeze the movements of the game.

The user is able to watch as the humans reproduce and eventually die from a zombie takeover or starvation. The zombies are able to attack humans and change them into a zombie if they land on consecutive squares. Zombie hunters will eradicate zombies if they come into contact.

The program also contains unique natural disasters that include potatorites, the derplitz, as well as the mushdoom. Each natural disaster is able to be randomly generated by the user and will eradicated life forms if they are in the area of attack.

The game also includes the features of populate and eradicate which will be able to perform selectively (of one life form) or generally (of all life forms). Save and load functions will help the user save an existing game into a file and load it when the user wishes to continue the game.

The equilibrium mode features a preset starting game where the end behaviour will be one of relative equilibrium and co-existence. Thus the user will not be able to have control of other variables such as populate, eradicate, as well as creating randomized natural disasters.

Limitations of the Program

Some limitations of the program include no user control of a specific piece. This game is meant to be an observation game of the environment and not one where the user is able to be one of the life forms.

In some situations the game is also laggy as there are many images that have to move and load simultaneously. The movements do not look extremely smooth in these cases and such graphic components can be improved in the future.

DEVELOPER'S GUIDE

Overview

The game "So You Think You Can Survive" is made up of over 10 different classes. These classes work together to provide functions, graphics, and different life forms in order for the game to be operated. The main purpose of the game is so that users of the game are able to observe reproduction and death cycles of specific life forms in the environment of a zombies and humans. The game has 2 different modes which include the freestyle mode in which users are able to control almost every aspect of the game and a preset starting version of the game which will yield specific results, such as equilibrium.

Features of the Program and Classes

Classes Involving Lifeforms

The LifeForms Class

This class is the heart of the programming section of the game. All of the lifeforms including zombie, human, and zombie hunter all extend this class. The lifeform class moves lifeform typed objects randomly to an adjacent up, down, left, or right square. The class also checks if each step will go out of bounds on the game board. It keeps track of the number of moves of each life form as well as it's respective image to represent the life form in the game.

Human Class

This is the most fundamental type of life form and also the one with the most functions. It extends the life form class and the human is essentially a life form that is at the bottom of the food chain. This class creates an object that shows either an image of a male or female human. The humans are able to reproduce if a male and a female human are on consecutive squares with a given reproduction success rate. At this point, a new Human object is created with a baby image and after 10 moves of the game, the baby is randomly generated to be either a male or female human.

Furthermore, the humans are able to move 60 times without eating any food but after that, the image is changed to a warning red picture of starvation. After the warning, the human will promptly disappear.

Food Class

This class does not extend life form but rather is an object which has the image of different types of foods. The Human class depends on this class because if a food object is acquired within 60 moves, the move count will be set back to 0 and the human will be able to move another 60 steps without eating any food. The food is randomly generated and placed onto the board in intervals. They are able to be eradicated by the user, however, they are not able to be repopulated.

Zombie Class

The zombie class extends the life form class and mainly sets the image of the zombie. The zombie is able to turn a human into a zombie when they come into contact and is essentially in the middle of the food chain. Thus, the game generally uses very little initial zombies because zombie sweeps are generally inevitable if there are too many zombies.

Hunter Class

The hunter class extends the life form class and sets the image of the hunter. The hunter is able to completely eradicate a zombie when they come into contact and is essentially at the top of the food chain. They will protect the humans from the zombies but it is generally not recommended to have too many hunters or else all the zombies will be eradicated.

Obstacle Class

The obstacle class does not extend the life form class and sets the image of an obstacle typed object to a stone. The obstacle is made so that no life form is able to bypass the obstacle. However, the user is able to eliminate obstacle objects but cannot populate any.

Natural Disaster Classes

These classes create natural disasters, each of which have specific functions to systematically eradicate random life forms on the board. The user is able to activate these natural disaster functions through JButtons in the game. If the button is pressed, a randomly generated area is promptly affected by the natural disaster.

Mushdoom Class

The mushdoom is a innocent looking but deadly mushroom. A randomly generated mushdoom has the ability to explode and kill all lifeforms who are adjacent to it. The Mushdoom class mainly sets the image of the mushdoom type object. The image of the mushdoom changes after 4 moves into an explosive fire image and disappears all together after it explodes.

Potatorite Class

The potatorite is a natural disaster which has small potatoes falling out of the sky like meteorites. The potatorites that land on squares which are already occupied by a life form will eradicate the life form, then disappears after 5 moves. The Potatorite class sets the image of the potatorites to a randomly generated image of a potato (out of the 3 that are available) so that not all the potatoes have the same image.

Poisonous Derp Class

The poisonous derp is accompanied by 4 squares of poison adjacent to it. These 4 squares of poison will eradicate any life form which lands on them at any time. The poisonous derp then disappears after 20 moves. The class is mostly used for setting the image of the poisonous derp. It also has a boolean variable for recording whether or not poison has been shot out of the derp. This is used in the Colony class to restore the poison if it is eradicated by the user or another life form.

Classes Involving the Actual Function of the Program

The Colony Class

The Colony class holds the core of the graphics and function of the game. It contains a show method which displays each life form onto the board as well as an advance method which the game uses to move each life form forward during each step. It also contains the functions of each life form including how the humans reproduce, how the zombies infect the humans, and how the hunters eradicate the zombies.

The Colony class also contains the populate and the eradicate methods which are dragged using user input and a MouseListener. Each method has the parameters of the coordinates of each corner as well as the success rate which is determined through the JSlider

as well as what type of life form it is which is determined through the JComboBox. Then each method will generate a random number from 1-100 inclusive and check if it has a lesser value than that of the success rate from the JSlider. This will ensure that the success rate is met because the actions are only performed if a random number is less than the success rate.

If the random number if less than the success, the methods will perform their respective functions. The eradicate method will check if the life form that occupies each space is that of the type passed in through the parameters. Then it will set the value of the grid space to null to eradicate the life form. The populate method will create new objects of the type that is passed in through the parameters to populate the grid.

The advance method is really the heart of the entire program. In this method, the program will move all the lifeforms and then check afterwards to see if any actions need to be performed. First the advance method will move all the lifeforms randomly by calling the move method within the move method within the Lifeform class. Next, it will check each zombie and whether there are any humans in the squares adjacent to it; the ones that are will turn into zombies. For reproduction, the program will search for a male and determine whether there is a female on the square above or below him. If this is the case, there is a 50% chance that a baby will be born. The program will search for zombiehunters next and will kill of any zombies which are in the squares adjacent to the zombie hunter.

Classes Involving the Flow of the Program

The Main Class

The purpose of the main class is to put the game together into one class so that other programmers and developers are able to see the flow of the program in a clearer fashion. This class is where the components, JPanels, and JFrames are all declared and set values. JButtons are declared for “Run” and “Pause” as well as each natural disaster and the populate and eradicate functions. A JSlider is declared to enable users to decide what the success rate of either eradicate or populate are.

The Main class also contains Listeners for each component including ActionListeners for each JButton, a MouseListener for the eradicate and populate, as well as a ChangeListener for the JSlider. The Main class then calls methods from other classes depending on which Listener’s action performed method was called.

Finally, it also contains a menu so that users are able to navigate the modes of the game easily. By clicking on the preset buttons, the program will load the normal game, but it will also load the preset containing all the positions for each of the lifeforms in order to create the interesting simulation.

Suggestions for Improvement

Areas of the program in which we could have improved include incorporating an option where the user is able to navigate through the environment as a lifeform. We attempted this, however, but faced many difficulties in getting the user input to be incorporated simultaneously

with the timer that runs the simulation. As well, we could have tried to make the movements of the lifeforms appear more fluid.