CS112 Final Project: Sebastian Paredes, Lisa Phan, Alexander Womack

Title (Pending): *God's Judgment*

Brief Description:

This project is a society simulator. After being granted the power to affect a village by God himself, the user will have the ability to both observe how villagers interact with each other, their environment, and their nonhuman companions, and affect the livelihoods of the villagers themselves through events, such as instilling a plague on the village, or converting a villager into a murderer.

What will your program look like at the end?

The program will be seen from a top-down perspective, with the user looking down at a village with inhabitants constantly in motion, as well as the environment and the user-prompted special events having animations as well. We will use Swing to construct our graphics–a pixel art style similar to games like Stardew Valley. The program will also include two sidebars: the left sidebar indicates user-triggered events that the user can select to influence the actions or environment of the villagers, while the right side-bar will include an "event log" as well as the health and happiness of the village as a whole. The villagers will both interact with each other and their environment.

How will the user input work?

The user will be able to input the number of “villagers” that will appear on the map initially, as well as inputting the village name. They can influence the “development” of the village through the possible event buttons on the left side-bar, which have immediate effects on the lives of the villagers. They can also click on the villagers to view their characteristics, such as their occupation, health, and happiness.

How will the program respond?

After the user clicks the start button and inputs the number of villagers, the program will begin a short cutscene in which God explains to the user the duties they are assigned, and the power granted to them. The program will then create a map with a specified number of inhabitants (who can have 5 occupations) who are constantly in motion and interacting with their environment. They sleep, eat, and work. The program will include a calendar, in which the current hour, day and month are displayed, affecting which season the village is in, the look of the village (day vs. night) as well as the schedules of the villagers. When the user prompts an event, it will have an immediate effect on the lives of the villagers, with significant effects (like death, a murderer, or a plague infection) appearing on the event log. The user will have the choice to end the game at any point using a button, in which afterwards, a final screen showing the event log, as well as performance of the user (such as how many deaths occurred under his judgment) will be displayed.

What purpose does it serve? (e.g., is a game, a productivity tool, a screen saver?)

This experience will be a simulator-esque game, not a productivity tool or screen saver. It is a charming way for players to watch the development of a mini-society and slowly rot their brains.

CLASSES

- ***public abstract class Person*** (and the subclasses that extend it)

There are multiple instances of the subclasses that extend Person, since we cannot create instances of Person itself (as it is an abstract class). The subclasses of Person all represent a certain type of "Person" in the village, such as subclass Farmer, subclass Caretaker, and subclass Herbalist . These will be the inhabitants of our village. We have two constructors to ensure that when villagers are first generated, the gender distribution is a 50/50 split.

Fields:

Personal Characteristics:

- **public static final int CARETAKER=1, HERBALIST=2, CARPENTER=3, GUARD=4, MALE=0; FEMALE=1;**

These are all representing characteristics of the Person themselves, which is used to both check what type of Person each instance is, as well to ensure that gender is chosen randomly when villagers are spawned.

- **Pair homeLocation, workLocation, pathInfo, bedLocation, position, velocity:** The first four are used for movement collision among the villagers, as well as to generate walking paths between to points, while position and velocity are the person's current position and walking velocity.
- **Node Path**: used for path generation as well.
- **Boolean isThief, isMurderer, isInfected, isLocked:** isThief represents whether a villager will go around burning flower, isMurderer represents whether that specific person is a Murderer, isInfected represents whether that person is infected with the plague, causing a decrease in health and happiness.
- **String name**= their name;
- **int gender**= their gender;
- **double happiness**

This variable will make up the overall happiness level of the villager. Affected by events like the plague.

- **private double health (0-100)**

This variable indicates the health of the Person; if it is zero, they die.

Appearance:

- Color skintone

- Color haircolor
- Color shirt
- Color pants
- Color eyes
- BufferedImage[] appearance: the appearance of the villagers; they are pixel art.

These, as the names imply, randomly assign a certain skin tone and hair color to each individual villager upon their creation.

Physical Characteristics:

- Pair position;
- Pair velocity;

These describe the position and velocity of each person, to keep track of their movement.

Methods:

- public void update (World w, double time, long hour): updates the position and state of the villagers.
- Public void eat (): eating animation
- public void eat(); (will prompt the Person to eat, with a representing graphic)
- public void sleep(); (will prompt the Person to sleep, with a representing graphic)
- public void moveTo (): traverses through a path, updating the position, toward the goal location.

- public abstract void draw(Graphics g){} constantly called upon when updating and drawing.
- Public abstract void work (int type): changes animations according to occupations
- public void getPlayerImage(): further defines animations and sprites.
- public void toString(): returns the name of the villager.

Subclasses (all extend Person):

- Class Farmer: includes methods to navigate while at work, to work(World w) with the corresponding sprites, workInteraction (World w){} which causes the farmer to plant, water, and reap crops.
- Class Caretaker: includes methods to navigate while at work, to work(World w) with the corresponding sprites and at certain hours, which is seen through feeding, milking cows, and sweeping.
- Class Carpenter: includes methods to navigate while at work, to work(World w) with the corresponding sprites and at certain hours, which is seen through chopping wood, building, and holding gifts.
- Class Guard: includes a method workInteraction(World w) {} which, when the guard collides with a Murderer, the murderer is arrested and sent to Prison.
- Class Herbalist: holds the same work() method with corresponding work sprites that represent their work picking up herbs, brewing potions, and healing. Furthermore, infected Villagers will visit the Herbalist clinic, which will restore their health.

- ***class World***

  This class represents the villager itself, with key fields including a height and width of the world, a Person [] people, Plant[] plants, Cow[] cows, Pig[] pigs, Blood[] bloodstains, which represent all of the inhabitants/nonhuman inhabitants/miscellaneous objects that grant the village life. There is also a boolean raining that determines whether it is raining or not at the current moment, as well as a boolean clinicHasVisitor, which during certain times, infected people will come to the Herbalist's clinic to look for healing. Moreover, there are int counters for totalDeaths, totalInfected, totalMurders, totalMurderers, and totalRains, which will be used during the final screen to give the player an idea of what happened through the game. Finally, there is a crucial JTextField textArea inside a JScrollPane labelOfEvents that serves as an event log, giving the user full access to event history at any time during the playthrough. The class holds one constructor, as the creation of a world happens only once, and we felt no need to add additional constructors. Worl

Methods:

- public void drawWorld (): draws the entire village at a given frame, from the grass, structures, people, animals, and plants.
- updateWorld(): calls the update methods on the animal and human classes, as well as the bloodstains, elements, and plants. It also deals with infection and murder opportunities

when a Murderer/Infected collides with another villager. Moreover, the update method declares villagers dead when their health is less than zero.

- Events (see below for a list of the "event methods" that will be used to affect our world.
    - Public void Plague () : marks a random Person in the village as Infected. Herbalists are immune to being infected
    - Public void addCriminal (): marks a random Person in the Village as a murderer. They will murder people they collide with.
    - Public void addForeigner(): adds a random new Person into the village;
    - Public void vanish(): Makes the last Person element in the Person [] people disappear.
    - Public void rain() {}: sets raining as true, making it rain.
- All of these events are saved in our labelOfEvents/textField, which is constantly updated after every event/significant occurrences like deaths.

- ***public class Pair***

It will be very similar to the public class Pair we have used in the homework. This class will help us in terms of the position of each villager, their velocity, and graphics in general.

- ***public class GenericNodeStructure***

The generic node structure, as the name implies, is a linked list that is also generic which is created in order to save the progress of the player throughout the game, which are saved in multiple different variable types meaning generics allows for ease of appending: these are all displayed in the final screen after the user is finished with the program. The methods within are

what is to be expected from a linked list and generally simply: the GenericNodeStructure constructor, the append method to add an element to the end of the list, and the GenericNode class which has the standard node properties. Only the append method is needed for this data structure because within the context of the program, we are only accumulating data to be displayed at the end of the game, so methods like “remove” or “pop” are not necessary and therefore a waste of coding space.

- ***public class Main***

The function of Main is self-explanatory: it is here that the implementation of the world occurs and the program actually runs, meaning that this is also where the interactive events that the player can cause are on display here. After declaring a number of basic and specific variables that become relevant later (e.g. World world, String Name (of the village), but also a list of JButtons), the Runner implements Runnable is added, which updates all the local variables with each passing moment, including the time and the aggregate data that will displayed at the final screen such as totalMurders. Then, the event buttons and the drawings of the actual buttons are drawn onto the screen in the form of JButtons and gdrawings, before the consequences of each of the events: adding a Plague, adding a Murderer, killing a Person, adding a Villager, and toggling Rain. After the running Thread making sure that the game is running, the paintcomponent method draws all the shapes on the screen, including labeling the village by its name and displaying certain updating stats on the screen, such as average village happiness which, as the name implies, takes the average of every villager’s happiness. The Event Log is also implemented, as each time an event occurs, the log is added to the event log. Certain contingencies to keep the code robust is also included here, such as the button to add villagers

disappearing when the maximum number of villagers have been reached. Finally, at the end of the class is the mandatory main method that executes upon running.

- ***public class Plant***

  The Plant class includes all the buffered images that are used for fire animations, for trees, and flowers, as well as a counter that keeps track of the frames for the purposes of animation as well as a boolean deciding whether the plant is a tree or not. The constructor for the plant makes it a tree for every 30 plants created, and decides where the plant's Position Pair is located based on whether it is a tree or not (trees are in a specified clearing, flowers are randomly anywhere on screen). The method getFlowerImage() causes all the buffered image files in the class to be read, the public void draw(g) method returns a flower if the flower is not on fire, a fire animation on the flower decided by flowerNum if the flower is on fire, or one of the four tree sprites (depending on the season) if the plant is a tree. Finally, the update method establishes the frame counter for the burning animation.

- ***public class InitialScreen***

The InitialScreen class is the introduction the user has to the game. The key fields are three JButtons villageEnterButton, villagerNumberEnterButton, and beginGameButton; as well as three JTextFields textOfVillage, numberOfVillagers, inputAgain (inputAgain is non-editable). The user types in the name of the village in textOfVillage, and the initial numberOfVillagers (with an error message if the numberOfVillagers>36, which is inputAgain). He then confirms his

choices through villagerNumberEnterButton and beginGameButton, which when both are confirmed and numberOfVillager is valid, beginGameButton becomes visible, which then allows the user to begin the game. There are Clip fields and BufferedImage fields that also serve to grant a more artistic and medieval look to our game introduction.

- ***public class InitialStoryScreen***

  InitialScreen is followed by InitialStoryScreen, which is a cutscene-like JPanel subclass that grants a little bit of story to the screen. The key fields in this class are JButton continueButton and a int numOfClicks, as well as multiple Clip objects in order to add music and voice acting. The main method of this class is our paintComponent() override, which, according to how many times the user has clicked the continueButton, draws and plays different sounds to give the illusion of a conversation with God. We divided the work by creating four draw methods: draw(Graphics g) which draws the background, drawFirstText (Graphics g, Font font) which draws and plays the first cutscene, drawSecondText (Graphics g, Font font) which draws and plays the second cutscene, and drawThirdText (Graphics g, Font font) which draws and plays the third cutscene. When numOfClicks==3, the main method (which is purposely stuck in a while loop) is able to continue, and our Main mainInstance can be initialized.

- ***public class Cow***

  Although the class is called Cow, this class determines all the animal behaviors, which are a simplified version of the human behaviors that occur inside the village barn. The class Livestock is found within the Cow file, where the classes Chicken, Sheep, Pig, and Cow all extend Livestock. Each of these subclasses have their own getImage() method where, just like usual, it reads in all the buffered image files needed for their animation. Since the image names for the same action are the same in each subclass, the public draw method dictates what sprite should be drawn depending on the velocity of the animal, as the animals are constantly moving around inside the barn. The navigate method in Livestock keeps the animals confined to the barn, flipping their velocity when they hit the walls of their enclosure. As in all other classes, the integers animalCounter and animalNum are frame counters to be used to periodically update the frames of the animals with time, successfully creating a walking animation in both directions, which are updated in the update method.

- ***public class Structures***

The Structures class includes the methods for the implementation and drawing of the crops and buildings on the map. The class Coordinates and grid inside the method were part of an attempt to implement collision into these structures, but the idea has not come to fruition with the code instead used for the drawing of each of the buildings on the map through the Coordinate class. The getImage() method, as always, reads in all the buffered image files, the drawStructures file draws every element of the structures array onto the world, and the drawCrops method does

the same thing for the crops, keeping the position to the crop area and choosing which image of the crops to draw depending on the state of the crops.

- ***public class Thing***

The Thing class refers to the replacement of a villager with a blood sprite when they have been murdered. The constructor invokes a position Pair and the getBloodImage() method, which reads the files for the blood animation. The integers bloodNum and bloodCounter serve as frame counters to switch between the sprites for a consistent animation, and the draw method chooses which of the two animation frames to draw depending on these integers, which are updated in the update method. The Thing class also includes seasonal effects to the game, where objects fall out of the sky depending on the season (snow for winter, petals for spring, sunrays for summer, and leaves for autumn). This is accomplished through another draw method, an update method that allows for the objects to fall with realistic physics from the sky, and a changeImage method that checks the season and changes the image according to the season.

- ***public class Time***

The class Time controls the calendar that constantly updates at the top of the screen while the program is running, which equates the elapsed time in-game (represented by the variable long elapsedTime and long start) to a calendar day and hour within the program. A number of longs within the class take the elapsedTime and condenses it through division and modular arithmetic to the amount of hours, days, and months that have passed. The public int monthNumber() method takes one such variable day and returns the number of the month based on the range the number of days passed falls into: later, in the public String calendar() method,

this variable is mapped to the appropriate string that labels the current month. The result of this, when day and hour is also applied to this method, is that the month, day, and hour are drawn second by second out on the screen while the program is running, done through the drawTime method that draws the String calendar().

- ***public class Appearance***

  An instance of the appearance class is created in each instance of Person, which uses the methods and variables inside the class in order to assign the person's sprites as arrays of layered BufferedImages which are used later. The arrays stand, walkleft, walkright, etc. initially start as null arrays of buffered images which are filled with layers of sprites that correspond to the person's unique animation for that action. For Work[][], we have a 2D array instead of a 1D array because there are multiple possible work sprites, as each subclass has more than one work action which may contain an animation. The class also contains static arrays and arraylists which have a list of male names, female names, and bed locations, so that in the methods each villager can be assigned a name and bed in the village.

  Methods:

  - All of the methods inside of the Appearance class are run at the construction of each villager:
    - getImages(): reads through the list of possible file images that might be used in later methods to decide the villager's appearance

- decideAppearance(): randomly assigns characteristics to the villager such as eye color, hairstyle, etc. and assigns sets of sprites based on these random characteristics as well as their gender and occupation
- decideName(): matches the villager's gender to the list of either male or female names, and picks one to assign to the villager
- decideBed(): matches the villager's bed location to a Pair from the static array of bed locations, and assigns it to the villager

- ***public class Main***

  Self-explanatory: is the main. Responsible for making the program run.

  We will also import Runner to make the program execute.

EVENTS

General Description/Features:

- All implemented in the class World
- The only way the user can influence the development of the program's society
- These events can only happen with the user's input.

Natural:

1) Plague (makes one random villager get the plague)
2) Drought (crops have less yield than normal, decreasing happiness and health)
3) Supplies (given from overseas) (supplies arrive at the village, increasing happiness and health)

Social:

1) Add criminal (makes a random villager a criminal, either a Thief or a Murderer)
2) Add Foreigner(s) (adds new inhabitants to the village)
3) Vanish (thanos snap a villager of choice) (villager of choice disappears into thin air - changes their position to a point that is out of the screen and makes velocity zero).
4) Thanos Snap (half the villagers disappear) (half the villagers disappear into thin air - change their position to a point that is out of the screen and make their velocity zero).

ADDITIONAL FEATURES

- Background music that the player will be able to mute
    - Song playing might change depending on certain external factors (e.g. average village happiness, season, etc.)
- Could add potential start menu or ending screen