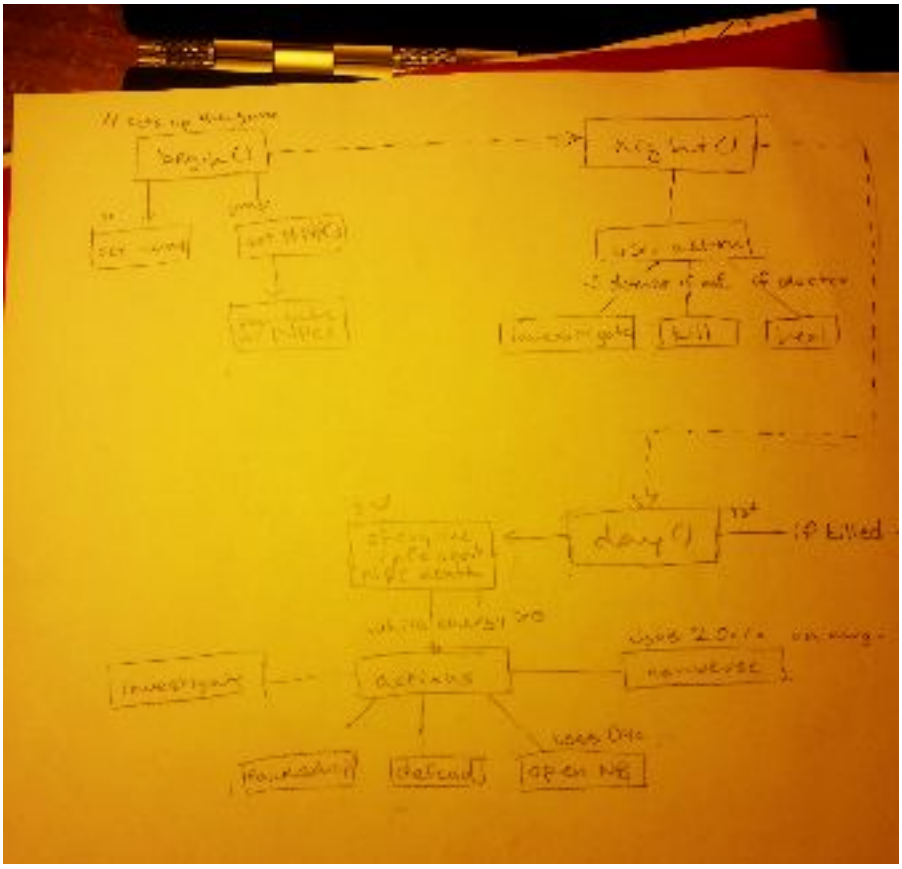
Programstinators
Cheryl Qian, Nadine Jackson, T. Fabiha
APCS1 pd2
2018-01-02

Town of Salem: Java Edition



Description: A text-based game of *Mafia* where the player is randomly assigned a role of Citizen, Doctor, Detective, or Mafia, competes against other pre-programmed players, and attempts to survive.

Rules: The user begins the game by selecting or being randomly assigned a role. When night falls, the Mafia kills off one player. The Doctor can select a player to heal, and if healed, the killed player survives the night. Roles of all players are revealed upon death.

When the night ends and day comes, the killed player's role is revealed. Evidence and hints are left at the scene, but the user gains SP (suspicious points) if they choose to

snoop. The players can then interrogate each other, as well as provide alibis. These interrogations and dialogues are shown to the townspeople and the user to determine who is guilty. At the end of the day, the townspeople cast their votes. The Detectives' vote are worth more than the Citizens', and the highest voted player is killed/burned at the stake. The players have a SP (suspicious points) stat, based on the cumulative number of votes they receive, as well as the value of their chosen alibis. The user can factor this into their decision when voting.

User Experience:

- User is initially prompted with a Start Menu, where rules and commands are displayed. They set their name, the number of players, and their role (or opt to have it randomized)
- After selection, a list of all players, their names, SP/votes, day roles, and a short quote from them is displayed. This can be called up at any time by typing C, the rules by typing R, and the notebook can be called up by typing N
- Day
    - Starts with a short description of the killed player and death scene.
    - User is prompted by a list of actions (Investigate, Eavesdrop, Converse, Defend)
        - **Converse (x Actions Left):** The user can choose who to converse with, and 3 questions to select from.
        - **Investigate (x Actions Left):** Check the death scene for evidence, at the risk of gaining SP.
        - **Eavesdrop (x Actions Left):** Listen to dialogue between two other players, at the risk of gaining SP.
        - **Defend (n Questions Left) :** The player wakes up to 3 questions/accusations every day. These are not in the same category as the above 3 actions, which are limited in number. Defend shows the question and 3 possible alibis to choose from.
        - **Open notebook** : The player has a notebook where they can store clues and other information that they have found. They have the option to read it, write in it, or erase entries.

- Once the x actions are used up, the user is prompted to vote on who is most likely part of the mafia. After voting, if there is no tie, then the person voted most likely is hanged. An updated stats list is displayed, and players go home for the night.

- Night
  - If the user has extra abilities, they are prompted to choose which player perform those abilities on.
    - **Kill:** Displays success or failure based on Doctor's actions
    - **Heal:** Displays success if Mafia killed the same player, else failure
    - **Investigate:** Displays the real role of the player

- Woo
  - Attributes:
    - ArrayList<Character> livingChars
    - ArrayList<String> notebook
  - Actions:
    - begin()
      - Displays rules if prompted
      - Asks for name
      - Asks for number of players
    - day()
    - night()
    - end()
      - Displays the types of all the characters
    - void converse(Character)
    - void investigate()
    - void eavesdrop(Character)
    - void defend()
    - void open()
    - void write()
    - void erase()
- Character (all subclasses other than civilian are playable)
  - Attributes:

        - int energy : Contains the number of moves the player has left for the day.
        - int susVal : measures how suspicious the User seems to the town
        - String type : ie. civilian, mafia, doctor, detective
        - String name : name of the character
        - boolean player : if it is not a player, randomization will be needed for certain actions
        - int charNum : gives a unique number to each character in the game
    - Actions:
        - boolean vote()
        - Character accuse(Character)
    - Subclasses:
        - Mafia
            - Actions:
                - kill()
        - Doctor
            - Actions:
                - save()
        - Detective
            - Actions:
                - investigate()

Implementation/Technical Details (From most core to least core):

- User can vote on whether someone is the mafia every day.
- User can complete actions at night (if relevant for their type of character).
- Players are sorted into different classes and subclasses and have different actions such as killing, healing, or discovering sensitive information. (Opportunity for interfaces and inheritance.)
- User can set their own role or have it randomly generated, and set the max number of players (estimated playtime shown).
- When voting, players are sorted by SP. Higher SP == higher chance of being voted for. (Usage of sorting algorithms)

- User gets clues/evidence during the day to the status of other players. (Somewhat automated conversations.)
- Interrogation questions have three hard coded responses to choose from, with different SP values. (Answering a question well will make you less suspicious in the eyes of the town, but you will lose the chance to gain valuable information by snooping.)
- "Day roles" and names for all players (butcher, mailman, etc.) through implementation of an interface.

Prioritized To-Do List:

1. Make class Woo request basic user input, and output dialogue indicating day and night cycles. (The computer is God.)
    a. Attributes:
        i. ArrayList<Character> livingChars
2. Make class Character with:
    a. Attributes:
        i. int susVal
        ii. boolean evil
        iii. String name
    b. Actions:
        i. boolean vote()
        ii. Character accuse()
3. Incorporate Character into Woo by having the player vote() every day and initializing the susVal randomly.
4. Sort by susVal at the beginning of every day before voting
5. Incorporate ArrayList<Character> livingChars into process and dialogue with people dying as appropriate
6. Make story for how a game ends (Mafia won or lost)
7. Create rule list and allow user to access it at any time by typing R
8. Make two subclasses of Character:
    a. Mafia:
        i. Actions:
            1. Character kill()

    b. Doctor:
        i. Actions:
            1. Character save()
9. Update how game ending story works
10. Allow player to choose whether they are Mafia or Doctor, and allow them to kill() and save() as appropriate.
11. Woo outputs description of killed and death scene
12. Ask player for name, type of character, and number of players
13. Allow randomization of above
14. Add more characters:
    a. Detective
15. Update rule list
16. Allow the player to have simple dialogue with other characters (hardcoded) during the day phase
17. Make random susVal for each player
18. Weight the susVal depending on whether the player is Mafia
19. Make the susVal more accurate over time
20. Increase susVal based on how much the player speaks
21. Warn the player that susVal may increase based on dialogue
22. Construct and implement questions that other players will ask to gauge the player's suspiciousness.
    a. Allow the user to defend him/herself
    b. 3 happen per day
23. Create list of characters to be displayed w/ numbers
24. Update rule list
25. Allow user to investigate the scene
26. Create energy instance variable so when the player completes an action, it depletes their energy
27. Create notebook for player to write suppositions, connections, etc.
28. Allow user to enter info in notebook at end of day
29. Allow user to eavesdrop on conversations
30. Update rule list
31. Debug more

Timeline (Jan 4th - Jan 16th, highlit days are weekends/no school):

- Jan 4th:
    - UML Diagrams
    - To Do List
    - Timeline
- Jan 8th:
    - Steps 1-2
    - Make Readme
- Jan 9th:
    - Flowchart of Woo.java
    - Revise Readme
- Jan 10th:
    - Steps 3-7
    - Debug
- Jan 11th:
    - Steps 8-12
- Jan 12th:
    - Steps 13-17
- Jan 13th:
    - Steps 18-22
    - Debug
- Jan 14th:
    - Steps 23-27
- Jan 15th:
    - Steps 28-31
    - Rigorous checking
    - Improvements of dialogue
- Jan 16th:
    - Finalization of everything:
        - Tweaks in dialogue
        - More testing
        - More testing