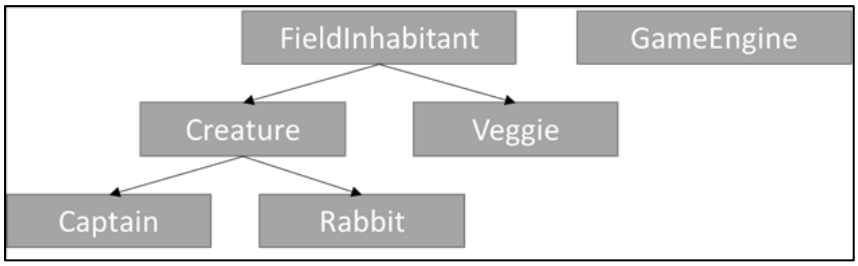
Project 2: Captain Veggie

Program Description:

For this project you will be creating the game Captain Veggie, in which rabbits have invaded the good captain's field and so the captain must harvest as many of their vegetables as possible before they are consumed by the leporine menace. In the game, Captain Veggie should be able to move about the field and attempt to harvest vegetables by moving on top of them. The captain scores points for each vegetable they harvest. At the same time, the rabbits should also be randomly hopping about consuming any vegetables they should land on. The game will continue until all of the vegetables have been removed from the field, after which the player's score will be displayed to the screen.

Program Requirements

- This a team-based project, as such, only one member of your group should create a private repository on GitHub to store the code and add the other group members to it
    - Each group member should make at least 3 substantial commits to the group repository during development
    - Consider making a commit to the repository after completing a one of the super or subclasses or after completing a major function
- The veggie file contains the comma delimited information regarding the initial configuration of the game:
    - The first line will always specify the field size, with the height first and the width second
    - All other lines will have the name of a vegetable, the letter representing that vegetable, and then the number of points that vegetable is worth
    - The vegetables will be in no particular order
    - See example input files
- Your program will need several classes that may utilize inheritance and polymorphism to properly function in the program:
    - `FieldInhabitant`
    - `Veggie`
    - `Creature`
    - `Captain`
    - `Rabbit`
    - `GameEngine`
- The class hierarchy should look like the following:



- All class member variables must be private or protected depending on how they are or are not used across the different classes in the program
- In a filed named **FieldInhabitant.py**, define a class named `FieldInhabitant`, which should contain:
    - A constructor that takes in a parameter representing a text symbol for the field inhabitant (vegetable, rabbit, captain, etc) and which assigns that value to a new member variable
    - Appropriate getter/setter functions
- In a file named **Veggie.py**, define a class named `Veggie`, that is a subclass of `FieldInhabitant`, and which should contain:
    - A constructor that takes in two parameters representing the name and symbol of the vegetable and a value representing the number of points the vegetable is worth
        - The superclass's constructor should be called and the symbol should be passed to it
        - The name and points values should be assigned to new member variables
    - An overridden `__str__()` function that outputs the symbol, name, and points for the vegetable in an easy to read format (See Example Output)
    - Appropriate getter/setter functions
- In a file named **Creature.py**, define a class named `Creature`, that is a subclass of `FieldInhabitant`, and which should contain:
    - A constructor that takes in two parameters representing x and y coordinates and one representing the symbol for that creature
        - The superclass's constructor should be called and the symbol should be passed to it
        - The x and y coordinate values should be assigned to new member variables
    - Appropriate getter/setter functions
- In a file named **Captain.py,** define a class named `Captain`, that is a subclass of `Creature`, and which should contain:
    - A constructor that takes in two integer values representing x and y coordinates
        - The superclass's constructor should be called and the x and y coordinates and the string "V" should be passed to it
        - Declare a new member variable to store an empty List containing all of the `Veggie` objects the captain has collected should be declared
    - A function named `addVeggie()` that takes in a `Veggie` object as a parameter, returns nothing, and adds the object to the List of `Veggie` objects
    - Appropriate getter/setter functions
- In a file named **Rabbit.py,** define a class named `Rabbit`, that is a subclass of `Creature`, and which should contain:
    - A constructor that takes in two values representing x and y coordinates
        - The superclass's constructor should be called and the x and y coordinates and the string "R" should be passed to it
    - Appropriate getter/setter functions

- In a file named **GameEngine.py,** define a class named `GameEngine`, which should contain:
    - A private constant to store the initial number of vegetables in the game named `NUMBEROFVEGGIES`, initialized to 30
    - A private constant to store the number of rabbits in the game named `NUMBEROFRABBITS`, initialized to 5
    - A private constant to store the name of the high score file named HIGHSCOREFILE, initialized to "highscore.data"
    - A constructor function that takes in no parameters
        - Declare a new, member variable to store an empty List representing the field
        - Declare a new, member variable to store an empty List representing the rabbits in the field
        - Declare a new, member variable to store the captain object, initialized to `None`
        - Declare a new, member variable to store an empty List representing all of the possible vegetables in the game
        - Declare a new, member variable to store the score, initialized to 0
    - A function named `initVeggies()` in which:
        - The user is prompted for the name of the veggie file, and if the user's file name doesn't exist, repeatedly prompts for a new file name until a file that does exist is provided
        - The first line in the veggie file should be used to initialize the field to a 2D List of the size specified in the file
            - All slots should be initialized to `None`
        - The remaining lines in the files should be used to create new `Veggie` objects that are added to the List of possible vegetables
        - The field 2D List should be populated with `NUMBEROFVEGGIES` number of new `Veggie` objects, located at random locations in the field
            - If a chosen random location is occupied by another `Veggie` object, repeatedly choose a new location until an empty location is found
        - Do not forget to close your files after you are done reading from them!
    - A function named `initCaptain()` in which:
        - A random location is chosen for the `Captain` object
            - If a chosen random location is occupied by another object, repeatedly choose a new location until an empty location is found
        - A new `Captain` object is created using the random location and the object is stored in the appropriate member variable and to the random location the field
    - A function named `initRabbits()` in which:
        - For `NUMBEROFRABBITS`, a random location is chosen for a `Rabbit` object
            - If a chosen random location is occupied by another object, repeatedly choose a new location until an empty location is found
            - A new `Rabbit` object is created using the random location and the object is added to the member variable List of rabbits and assigned to the random location in the field

- A function named `initializeGame()` in which:
  - The `initVeggies()` method is called
  - The `initCaptain()` method is called
  - The `initRabbits()` method is called
- A function named `remainingVeggies()` that takes in no parameters, examines the field and returns the number of vegetables still in the game
- A function named `intro()` that takes in no parameters, returns nothing, and in which:
  - The player is welcomed to the game
  - The premise and goal of the game are explained
  - The list of possible vegetables is output including each vegetable's symbol, name, and point value
    - Be sure to use the appropriate `Veggie` function for the printing
  - Captain Veggie and the rabbit's symbols are output
  - Remember that you are informing the user about the game, so be sure to include appropriate messages and descriptions
- A function named `printField()` that takes in no parameters, returns nothing, and in which:
  - The contents of the field are output in a pleasing 2D grid format with a border around the entire grid
- A function named `getScore()` that takes in no parameters and returns the current score
- A function named `moveRabbits()` that takes in no parameters, returns nothing, and in which:
  - Each `Rabbit` object in the List of rabbits is moved up to 1 space a random x,y direction
    - Thus, the rabbit could move 1 space up, down, left, right, any diagonal direction, or possibly not move at all
    - If a `Rabbit` object attempts to move outside the boundaries of `field` it will forfeit its move
    - If a `Rabbit` object attempts to move into a space occupied by another `Rabbit` object or a `Captain` object it will forfeit its move
    - If a `Rabbit` object moves into a space occupied by a `Veggie` object, that `Veggie` object is removed from `field`, and the `Rabbit` object will take its place in `field`
    - Note that `Rabbit` object's appropriate member variables should be updated with the new location as well
    - Make sure you set the `Rabbit` object's previous location in the field to `None` if it has moved to a new location
- A function named `moveCptVertical()`, that takes in a value representing the vertical movement of the `Captain` object, returns nothing, and in which:
  - If the `Captain` object's current position plus the movement would move them into an empty slot in the field
    - Update their appropriate member variable
    - Assign them to the new location in the field

        - Otherwise, if the `Captain` object's current position plus the movement would move them into a space occupied by a `Veggie` object
            - Update the `Captain` object's appropriate member variable
            - Output that a delicious vegetable, using the `Veggie` object's name, has been found
            - Add the `Veggie` object to the `Captain` object's List of Veggies using the appropriate function
            - Increment the score using the `Veggie` object's point value
            - Assign the `Captain` object to the new location in the field
        - Otherwise, if the `Captain` object's current position plus the movement would move them into a space occupied by a `Rabbit` object
            - Inform the player that they should not step on the rabbits
            - Do not move the `Captain` object
        - Make sure you set the `Captain` object's previous location in the field to `None` if it has moved to a new location
    - A function named `moveCptHorizontal()`, that takes in a value representing the horizontal movement of the `Captain` object, returns nothing, and in which:
        - If the `Captain` object's current position plus the movement would move them into an empty slot in `field`
            - Update their appropriate member variable
            - Assign them to the new location in `field`
        - Otherwise, if the `Captain` object's current position plus the movement would move them into a space occupied by a `Veggie` object
            - Update the `Captain` object's appropriate member variable
            - Output that a delicious vegetable, using the `Veggie` object's name, has been found
            - Add the `Veggie` object to the `Captain` object's List of Veggies using the appropriate function
            - Increment the score by the `Veggie` object's point value
            - Assign the `Captain` object to the new location in `field`
        - Otherwise, if the `Captain` object's current position plus the movement would move them into a space occupied by a `Rabbit` object
            - Inform the player that they should not step on the rabbits
            - Do not move the `Captain` object
        - Make sure you set the `Captain` object's previous location in `field` to `null` if it has moved to a new location

- A function named `moveCaptain()`, that takes in no parameters, returns nothing, and in which:
  - The user is prompted for which direction to move the `Captain` object in, Up(W), Down(S), Left(A), or Right(D)
    - You should accept both uppercase and lowercase W,A,S,D for the movement
  - Check the player's input:
    - In the case of W or w:
      - If moving the `Captain` object one slot up would not put it outside the boundaries of field, call the `moveCptVertical()` function and pass it the appropriate value
      - Otherwise, inform the player that they cannot move that way and do not move the `Captain` object
    - In the case of S or s:
      - If moving the `Captain` object one slot up would not put it outside the boundaries of field, call the `moveCptVertical()` function and pass it the appropriate value
      - Otherwise, inform the player that they cannot move that way and do not move the `Captain` object
    - In the case of A or a:
      - If moving the `Captain` object one slot up would not put it outside the boundaries of field, call the `moveCptHorizontal()` function and pass it the appropriate value
      - Otherwise, inform the player that they cannot move that way and do not move the `Captain` object
    - In the case of D or d:
      - If moving the `Captain` object one slot up would not put it outside the boundaries of field, call the `moveCptHorizontal()` function and pass it the appropriate value
      - Otherwise, inform the player that they cannot move that way and do not move the `Captain` object
    - Otherwise, inform the user that their input is not a valid option and do not move the `Captain` object
- A function named `gameOver()`, that takes in no parameters, returns nothing, and in which:
  - The player is informed the game is over
  - The names of all of the vegetables the `Captain` object harvested are output
  - The player's score is output
  - Remember that you are informing the user about the game, so be sure to include appropriate messages and descriptions
- A function named `highScore()`, that takes in no parameters, returns nothing, and in which:
  - You declare an empty List to store Tuples representing player initials and their score
  - Check if the **highscore.data** file exists, and if it does
    - Open it, using the appropriate constant, for binary reading

        - Unpickle the file into the List of high scores
        - Close the file
    - Prompt the user for their initials and extract the first 3 characters
    - If there are no high scores yet recorded, create a Tuple with the player's initials and score and add it to the List of high scores
    - Otherwise, create a Tuple with the player's initials and score and add it to the correct position in the List so the List is in descending order of scores
    - Output all of the high scores in the List, with an appropriate header
    - Open the **highscore.data** file, using the appropriate constant, for binary writing
    - Pickle the List of high scores to the file
    - Close the file
- In a file named **main.py,** you should have:
    - Your main function in which:
        - You instantiate and store a `GameEngine` object
        - Initialize the game using the appropriate `GameEngine` function
        - Display the game's introduction using the appropriate `GameEngine` function
        - Create an variable to store the number of remaining vegetables in the game, initialized using the appropriate `GameEngine` function
        - While there are still vegetables left in the game
            - Output the number of remaining vegetables and the player's score
            - Print out the field using the appropriate `GameEngine` function
            - Move the rabbits using the appropriate `GameEngine` function
            - Move the captain using the appropriate `GameEngine` function
            - Determine the new number of remaining vegetables using the appropriate `GameEngine` function
        - Display the Game Over information using the appropriate `GameEngine` function
        - Handle the High Score functionality using the appropriate `GameEngine` function
- No global scoped or class variables, other than those specified are allowed for this project
- Your code should be well documented in terms of comments. For example, good comments in general consist of a header (with your name, date, and brief description), documentation comments for function, comments for each variable, and commented blocks of code

Submission

- Your program will be graded largely upon whether it works correctly
- Your program will also be graded based upon your program style. This means that you should use comments (as directed) and meaningful variable names
- You must submit the **FieldInhabitant.py, Creature.py, Captain.py, Rabbit.py, Veggie.py, GameEngine.py,** and **main.py** files
- You must submit the URL link to your repository and set the repository from private to public three days after the due date (i.e. the day after the second late day)
- You must work in teams of 2 or 3 students for this project. You are not allowed to work with individuals outside of your team, other than the instructor and TA. Any discovered instances of this will be considered cheating and appropriate actions will the taken according to the course syllabus
- Additionally, you are not allowed to download code off of the internet or use generative AI for this project. Any discovered instances of this will be considered cheating and appropriate actions will the taken according to the course syllabus
- Be sure that you have tested the version of the program you wish to submit to make sure it works correctly. You will not be allowed to resubmit work after the deadline

Rubric

The entire assignment is worth 100 points and partial credit is possible. No credit will be given for portions of the program that cannot be tested due to the program crashing.

- **Program Executes Successfully**
  - If your program crashes due to syntax errors, you will lose 10 points, but we will attempt to fix minor issues (incorrect indentations, stray character, missing import) so that we can execute and test the program. We will not fix major issues that would require functionality to be further implemented, or a reorganization of logic in your code.
- **Data Storage (10 points)**
  - Each of the data storage classes `FieldInhabitant`, `Creature`, `Captain`, `Rabbit`, and `Veggie` are setup as specified and use inheritance as appropriate
- **Game Initialization (25 points)**
  - Requested functions are used to initialize the game (5 points)
  - Vegetable data is read in and stored (5 points)
  - `Veggie` objects are stored at random locations in the field (5 points)
  - A `Captain` object is stored in a variable and at a random location in the field (5 points)
  - `Rabbit` objects are stored both in the List and at random locations in the field (5 points)
- **Game Play (35 points)**
  - Program correctly moves all `Rabbit` objects adhering to requirements (10 points)
  - Program correctly prompts the player for which direction the captain should move and handles all inputs as requested (5 points)
  - Program correctly moves the `Captain` object vertically, adhering to requirements (10 points)
  - Program correctly moves the `Captain` object horizontally, adhering to requirements (10 points)
- **Program Output (20 points)**
  - Program correctly outputs all required welcome information at the beginning of the game (5 points)
  - Program correctly outputs the number of remaining vegetable and player's score before asking where the captain wants to move (5 points)
  - Program correctly prints the field before asking where the captain wants to move (5 points)
  - Program correctly outputs all required game over information (5 points)
- **Each group member made at least three substantial commits to GitHub (5 points)**
- **Program contains sufficient comments (5 points)**

Bonus

For 10 bonus points, implement a new class named `Snake` in a **Snake.py** file. The `Snake` class should inherit from the `Creature` class, and should contains a constructor that takes in `x` and `y` parameter variables and provides them to the superclass's constructor along with the letter `"S"` for the symbol.

In your `GameEngine` class, you should:

- In your `GameEngine` constructor, define a new private member variable to store a `Snake` object and assign it the value `None`
- Define a function named `initSnake()` that instantiates a new `Snake` object in a random, unoccupied slot in the field. Be sure to store the object in the appropriate `GameEngine` member variable This function should be called by your `initializeGame()` function after you have initialized the rabbits
- Define a function named `moveSnake()` that attempts to move the snake on the file. The snake can only move up, down, left, or right (not diagonally), cannot move out of the field, and cannot move into a space occupied by a vegetable or a rabbit. When the snake moves, it must always try to move closer to the captain object's position. If the snake ever attempts to move into the same position as the captain, the captain loses the last five vegetables that were added to their basket and the snake is reset to a new random, unoccupied position on the field. `moveSnake()` should be called in your **main.py** file after the captain has moved