CS6310 – Software Architecture & Design
Assignment #6 [200 points]: OsMowSis – Team Implementation & Demo (v3)
Spring Term 2019 – Prof. Mark Moss

Submission

- This assignment must be completed as part of an approved group.
- One member of each team must submit:
  1. A VirtualBox (preferred) or VMware image that contains a pre-configured, ready-to-run version of your application named ***team_<team-number>.ova***, such as ***team_19.ova***;
  2. All source code (Java and otherwise) and links to external packages and other resources used to create your application in a file named ***source_code.zip***;
  3. The updated "as-is" UML Class and Sequence Diagrams, and any other accompanying documentation, in a file named ***design_docs.pdf***; and,
  4. An MP4 video (or other reasonably comparable format) with a ***viewing time of 10 minutes*** or less named ***mts_video.mp4*** that presents an overview of your application.
- For this project, you should select reasonably named files, and parts (2) and (3) must be submitted via Canvas. The virtual machines and videos for parts (1) and (4) can be very large, so a link to an external but accessible storage site (e.g. Google Drive) should be submitted via Canvas for those deliverables.
- Alternate (non-VM based) submissions for part (1) must be pre-approved by the Instructor or TAs. Also, submitting a link to an established code repository – for example, Georgia Tech's GitHub – for part (2) must be pre-approved by the evaluating TA.

- Submit your answers via Canvas.
- You must notify us via a private post on Piazza BEFORE the Due Date if you are encountering difficulty submitting your project. You will not be penalized for situations where Canvas is encountering significant technical problems. However, you must alert us before the Due Date – not well after the fact.

Scenario

The clients are excited about your design proposals as we enter the final phase of the project. During this phase, you will: (1) finalize, implement and deploy your team's design; (2) ensure your design documents are consistent with the finished application; and, (3) prepare a short video presentation for the clients and evaluators that demonstrates how your application functions.

Disclaimer

This scenario has been developed solely for this course. Any similarities or differences between this scenario and any of the programs at Georgia Tech programs are purely coincidental.

Deliverables

This assignment requires you to submit the following items:

1. **Functioning Application [75 points]:** You must submit a working application within a virtual machine image that satisfies the client's requirements as presented throughout all of the assignments, and as clarified further through all associated Office Hours and Piazza posts. Your

system must incorporate all of the design requests as described in all of the previous assignments and below. Your application must be submitted as a VirtualBox (strongly preferred) or VMware virtual machine image.

You have administrative control over the VM, and this submission method allows your team to have maximum flexibility for selecting, implementing and configuring different packages (e.g. GUIs, databases, etc.) in developing a solution for this problem, while minimizing the time needed by the graders to evaluate your (widely varying) solutions. You are allowed to use an alternate submission method if (and only if) you have received prior approval from your TA.

From a grading perspective, the points for implementation are distributed as follows:

- 25 points for providing a well-designed Graphical User Interface – the amount of points earned will be determined by the clarity and "ease-of-use" of your system, as well as the extent to which the interface supports the normal flow of operations during each simulation run.
- 15 points for the proper handling and operation of the mowers and their interaction with other aspects of the environment (e.g. craters, fences, puppies, other mowers, etc.), and their overall effectiveness in cutting the lawn.
- 10 points for the proper handling and operation of the puppies (e.g. distribution of movement and staying still, selection of new squares, etc.).
- 10 points for the overall behavior of your monitoring system, and the accuracy of your display of the state of the simulation.
- 15 points for the overall correctness of your simulation output log files.

2. **Source Code [50 points]:** You must also provide copies of the actual source code that your team has developed, along with references to all of the external packages, libraries, frameworks, services and systems used to develop your application. The overall architectural and design quality of your system will be evaluated, to include factors such as good use of abstraction of modularity, reasonable documentation and descriptive class, variable and method names, etc.

3. **Design Documentation [50 points]:** You have already developed numerous design documents: Class Diagrams, Sequence Diagrams, Object Diagrams, and possibly other forms such as State Charts and/or Collaboration Diagrams. For this assignment, you must select and provide the most appropriate UML-compliant Structural and Behavioral Diagrams that describe the final design of your application. There are no more pending or additional requirements after this assignment, so your final design documents should be as accurate and consistent with the final implementation of your application as possible. And especially since you have permission to use languages and system stacks other than Java, you need to ensure that your design documentation is consistent with your actual implementation.

   Please clearly designate which version of UML you will be using – either 2.0 (preferable, and the latest OMG-accepted version: https://www.iso.org/standard/52854.html) or 1.4 (the latest ISO-accepted version). There are significant differences between the versions, so your diagrams must

be consistent with the standard you've designated. The design documents can be submitted as different documents but must be named clearly and accurately.

From a grading perspective, the points for documentation are distributed as follows:

- 20 points for your UML Structural Diagrams, which should include (at a minimum):
  - A Class Diagram that describes system as it is actually implemented
  - A Deployment (or similar) Diagram that shows how your system is actually implemented, especially in integrating external packages, services and/or systems
- 20 points for your UML Behavioral Diagrams which should include (at a minimum):
  - Sequence, State Machine (or similar) Diagrams the give a reasonable overview of how the system is implemented
  - Use Case Diagrams for all functionality added per your design proposals
- 10 points for the overall quality and presentation of your design documentation, including the addition of various diagrams that add significant value for the clients in understanding and maintaining your system into the future. *Do not simply create lots of diagrams to attempt to collect points – this is definitely a "quality over quantity" issue*. Ensure that new diagrams and/or documents are well formatted and add significant value.

4. **Demonstration Video [25 points]:** Your team must submit a video demonstration of the prototype that your all have developed. You can either submit a YouTube link (preferred) or an MP4 attachment in Canvas. If submitting a video attachment, make sure that the size doesn't exceed 360MB, which should be approximately 10 minutes for a 720p quality video. Also, we realize that not everyone wants to "make it to the big screen": faces are welcome, but there's no requirement for your faces to be displayed during the video. A screen capture of your system in action, along with a clear, audible English-language audio track, will be sufficient.

   Your video should be organized to give a clear demonstration of your system in action, including a well-considered test case/scenario that allows you to show how your interface and supporting components operate; and, how your overall system meets the client's requirements. And your video should highlight your design modifications. There's no need to spend significant amounts of time demonstrating the capabilities that already existed in the system – focus on the aspects of your system that have been changed, upgraded, etc.

Writing Style Guidelines

The style guidelines can be found on the course Udacity site, and at:
https://s3.amazonaws.com/content.udacity-data.com/courses/gt-cs6310/assignments/writing.html

The deliverables should be submitted in the appropriate formats (OVA, ZIP, JAR, PDF, etc.) with file names such as ***team_19.ova*** (if applicable), ***source_code.zip***, ***design_docs.pdf*** and ***mts_video.mp4***. You should also use a similarly clear and simple name structure if you need to submit a file that we haven't explicitly listed here. Ensure that all files are clear and legible; points will likely be deducted for unreadable submissions.

Final Problem Description

In earlier assignments, we discussed the multi-mower option of the OsMowSis application, including the addition of puppies as dynamic obstacles. There aren't any other significant changes to the requirements from earlier assignments – we're primarily just clarifying the details that have already been presented. In this assignment, you are tasked to: (1) coordinate with your team members to develop a consolidated design proposal; (2) implement your consolidated design; and, (3) demonstrate those changes to the clients. We provide extra details below which build on the most recent additional requirements.

User Interface and Error Checking

Your application must provide a reasonable Graphical User Interface (GUI) to support your application. You are welcome to develop a web-based or application-based GUI. An extremely complex GUI is not required (and not the intent for this course) – a simple GUI that provides the required functionality will be sufficient.

- Your GUI must provide a display that shows the current state of the simulation run, including:
  - The state of the lawn to include the location of mowers, puppies, craters and the fence line;
  - Related information about the current state of all objects, to include which object will execute the next action (i.e. which one will be polled next), whether mowers are stalled or permanently crashed, and the remaining number of turns a mower will be stalled (if caused by a collision);
  - Summary information such as the number of grass squares that have been cut so far, the number of grass squares remaining, and the number of turns taken so far.
- Also, your GUI must provide (at a minimum) the following functionality:
  - A "Next" button that allows the user to control the pace of the simulation. When the user clicks the Next button, then the simulation monitor must poll the next object, validate that the proposed action to determine the proper response, and then update the state of the simulation run appropriately.
  - A "Fast-Forward" button that allows the user to execute the current simulation run to completion. When the user clicks the Fast-Forward button, then the simulation monitor must continue to poll the objects in sequence until the simulation run has been completed.
  - A "Stop" button that terminates the current simulation run. When the user clicks the Stop button, then the simulation monitor must terminate the simulation run at the current state of the simulation and ensure the that output log includes the final report.
- The clients will greatly appreciate any efforts to make the GUI user-friendly, and to help prevent errors when entering data, configuring any relevant simulation settings, or executing a simulation run. For example, robust warnings and error messages will be very helpful.
- You are still permitted to have Command-Line Interface (CLI) capabilities with your application to support development, maintenance, troubleshooting and execution functionality. Your GUI, however, must be able support the major functionality required for a user to conduct a simulation. Your CLI is allowed to provide "behind the scenes" support for the GUI as needed.
- We are giving your team fairly wide latitude in selecting a framework to support the development of your user interface, given that you are responsible for installing and configuring that framework on the course VM (or an alternate pre-approved platform) for submission.
- Your application must support the requirements previously described, in addition to any updates listed below. This is absolutely critical to our ability to evaluate your application thoroughly.

Updated Input File Format

Your system must be able to read in an input file to begin the simulation run. Given the addition of puppies and other changes to the simulation environment, we are providing an updated input file format:

1. <the width (horizontal/X-direction) of the lawn>
2. <the height (vertical/Y-direction) of the lawn>
3. <the number of lawnmowers being used>
4. <the mower "collision delay": the number of turns stalled if it collides with another mower>
5. <the initial location and direction of each lawnmower> [one line per lawnmower]
6. <the number of craters on the lawn>
7. <the location of each crater> [one line per crater]
8. <the number of puppies on the lawn>
9. <the puppy "stay percentage": the probability that a puppy stays at its current location>
10. <the initial location of each puppy> [one line per puppy]
11. <the maximum number of turns for the simulation>

Simulation and Input File Value Constraints

- The width of the lawn will be a value between 1 and 15 inclusive, and the height of the lawn will be a value between 1 and 10 inclusive. Hopefully this limitation will help as you plan your GUI design.
- There will be at most 10 mowers and at most 6 puppies for any single simulation run or test scenario input file.
- The mower "collision delay" will be a value between 0 and 4 inclusive.
- The puppy "stay percentage" will be a value between 0 and 100 inclusive, where 0 would cause the puppies to move every turn, and 100 would cause the puppies to stay at the same location for the duration of the simulation run.
- The maximum number of turns for any simulation run will be 300.

Input File Format Example

Consider the following example input file:

```
10
8
3
2
1,1,north
5,2,north
2,4,west
5
4,0
1,2
1,3
8,6
8,7
2
60
0,7
5,4
75
```

How to interpret this file:

- The lawn has a width of 10 squares and a height of 8 squares for a total size of 80 squares. The lower left (southwestern-most) corner of the lawn is (0, 0), and the upper right (northeastern-most) corner of the lawn is (9, 7).
- There are three mowers on the lawn, starting at locations (1, 1), (5, 2) and (2, 4) with the given initial orientations (i.e. direction of travel) as shown in the file. If a mower collides with another mower, then the mower that attempted the **`move`** action will be "stalled" and prevented from performing any action – **`move`**, **`scan`** or **`turn_off`** – for the next two (2) turns. After this time, the mower may resume normal operations. Also, the mowers must be polled during the simulation run in the same order that they are listed in the input file.
- There are five craters at locations (4, 0), (1, 2), (1, 3), (8, 6) and (8, 7). This means that there are actually only 80 – 5 = 75 grass squares that can be cut. And don't forget that three of these squares are cut by the lawnmowers simply being on their initial squares, so then the challenge is for the three mowers to cut the remaining 75 – 3 = 72 grass squares in as few turns as possible.
- There are two puppies starting at locations (0, 7) and (5, 4). Each time a puppy is polled, it has a 60% chance of staying at its current location – otherwise, it moves to one of the surrounding safe squares with a uniformly distributed probability. And like the mowers, the puppies must be polled during the simulation run in the same order that they are listed in the input file.
- Finally, the simulation run must be terminated after 75 turns.

What the Mowers Are Allowed to "Know" at the Beginning of the Simulation Run

- Mowers are allowed to know their direction and their absolute location on the lawn.
- Mowers are allowed to know the lawnmower collision delay and puppy stay percentage factors.
- Mowers are NOT allowed to know the size of the lawn, nor are they allowed to know the initial locations of the craters and puppies – this knowledge must be gained by scanning the lawn.
- Each mower can share any knowledge learned with the other mowers as needed.

Output Log File

Your system is required to produce an output log file that we can use to analyze the details of the simulation run. We will give some flexibility on the way you produce the results – for example, outputting them directly to the terminal, or into a clearly named output file – but they must be easily accessible to the TA's evaluating the work. The format of the output log results will be very similar to the output from all of the earlier work, with a few small changes to account for puppy actions and mowers that are delayed.

Each time a mower or puppy is polled, then the output log file must include three distinct lines as:

(1) The identifier for that particular mower or puppy – for example, **`mower,1`** or **`puppy,2`**;
(2) The action that the mower or puppy will execute: {**`move`**, **`scan`**, **`turn_off`**} for mowers, or {**`stay`**, **`move`**} for puppies; and,
(3) The result of the selected action.

When a mower executes a **`move`** action, it can move 0, 1 or 2 squares on a given turn.

- If the move causes a collision with a crater or fence, then the result of the action must be **`crash`**.

- If the move causes a collision with another mower, then the mower that attempted the move stops on the square immediately before the other mower-occupied square, and the result of the action must be **`stall,k`** where ***k*** is equal to the actual number of squares traveled.
- If the move causes a collision with a puppy, then the mower that attempted the move stops on the same square as the puppy, and the result of the action must be **`stall,k`** where ***k*** is equal to the actual number of squares that the mower traveled.
- Finally, if there were no collisions, then the mower completes its move successfully, and the result of the action must be **`ok`**.

- When a mower executes a **`scan`** action, then the result must be the list of contents for the eight squares surrounding the mower, starting with the northern square and proceeding in a clockwise direction. The value for each square must be one of the following: {**`grass`**, **`empty`**, **`crater`**, **`fence`**, **`mower`**, **`puppy_grass`**, **`puppy_empty`**, **`puppy_mower`**}.

- Finally, when a mower executes a **`turn_off`** action, then the result must be **`ok`**. Of course, that mower is not allowed to execute any more actions for the durations of the simulation run.

When a puppy is polled, it will either stay at its current location, or move to a new square:
- When a puppy executes a **`stay`** action, then the result must be **`ok`**, and the puppy simply remains on its current square.
- When a puppy decides to move instead of simply staying on its current square, then it executes a **`move,X,Y`** action and changes its location to square (***X***, ***Y***), and the result must be **`ok`**. Remember that puppies only move to surrounding "safe" (i.e. non-crater and non-fence) squares.

This is an example of one possible turn and the corresponding output log sequence:

turn #1
from the example scenario

mower #1:
initial location (1,1)
```
mower,1
move,2,northeast
crash
```

mower #2:
initial location (5,2)
```
mower,2
move,1,east
ok
```

mower #3:
initial location (2,4)
```
mower,3
scan
grass,grass,grass,grass,grass,crater,grass,grass
```

puppy #1:
initial location (0,7)
```
puppy,1
stay
ok
```

puppy #2:
initial location (5,4)
```
puppy,2
move,5,3
ok
```

UPDATE: All directions will be in lowercase to simplify the format (e.g. **`north`** instead of **`North`**).

At the end of the given sequence, mower #1 has crashed, mower #2 has just been jumped on and temporarily stalled by puppy #2, and mower #3 is operating well. The simulation will continue with turn #2, where mower #1 is not polled because it has crashed, mower #2 is not polled on this turn because it is temporarily stalled, and so only mower #3 is polled to request and validate its action, and then update the simulation state. Of course, the puppies are also polled for their actions as well – puppy #1 stays on its given square, which puppy #2 moves to the destination square (5, 3).

Here's a second possible turn for demonstration purposes:

turn #2
from the example scenario

mower #3:
current location (2,4)
```
mower,3
move,1,northwest
ok
```

puppy #1:
current location (0,7)
```
puppy,1
move,1,6
ok
```

puppy #2:
current location (5,3)
```
puppy,2
move,6,3
ok
```

Note here that mowers #1 and #2 are not polled: mower #1 has crashed and is permanently done for this simulation run, and mower #2 is covered by puppy #2. Puppy #2 moves off of mower #2 by the end of this turn, though, so mower #2 will be eligible to move again during turn #3. Also note that mower #2 did get to reorient itself to point East at the end of the previous turn, since it completed its earlier turn #1 movement action successfully before puppy #2 hopped on top.

And finally, a third possible turn:

turn #3
from the example scenario

mower #2:
current location (5,3)
```
mower,2
move,2,northeast
stall,1
```

mower #3:
current location (1,4)
```
mower,3
scan
grass,grass,empty,grass,crater,grass,grass,grass
```

puppy #1:
current location (1,6)
```
puppy,1
move,1,5
ok
```

puppy #2:
current location (6,3)
```
puppy,2
stay
ok
```

Of course, mower #2 is eligible to move again, and attempts to move "forward" (heading East) two squares. This causes it to collide with puppy #2, which means that it actually only travels one square – hence the **`stall,1`** response. Also, mower #2 does not get to reorient itself to the Northeast direction because of the collision, so it's still headed East. And even though it collided with the puppy, it did cut the square of grass on which it stopped.

The output log must include all of the mower and puppy actions that are executed. Once the simulation run is ended, the simulation monitor must also include the final report at the end of the output log file, which includes: (1) the total number of lawn squares; (2) the total number of grass squares (no craters); (3) the number of grass squares that were cut by all of the mowers combined; and, (4) the total number of turns.

The simulation run must end when all mowers have been turned off or crashed. If this happens, then the simulation monitor must still ensure that the final report is included at the end of the output log file, however displaying the final puppy actions is optional. The simulation run might also be ended prematurely by the evaluator using the "Stop" button – and, in this case, no more actions should be published to the output log, and the final report must be included at the end of the file.

As an example, if the evaluator were to click the "Stop" button at the this point during the simulation, then the output log file (including the final report) should be:

```
mower,1
move,2,northeast
crash
mower,2
move,1,east
ok
mower,3
scan
grass,grass,grass,grass,grass,crater,grass,grass
puppy,1
stay
ok
puppy,2
move,5,3
ok
mower,3
move,1,northwest
ok
puppy,1
move,1,6
ok
puppy,2
move,6,3
ok
mower,2
move,2,northeast
stall,1
mower,3
scan
grass,grass,empty,grass,crater,grass,grass,grass
puppy,1
move,1,5
ok
puppy,2
stay
ok
80,75,6,3
```

The lawn has a total of 80 squares, with 5 craters and 75 grass squares to be cut. The mowers cut a total of 6 of the grass squares during the three (3) turns of the simulation run.

Closing Comments & Suggestions

We (the OMSCS 6310 Team) will conduct Office Hours where you will be permitted to ask us questions in order to further clarify the client's intent, etc. Also, the TA who will evaluate your final submission will be pre-assigned to your team. You can communicate with them if you have questions related to application design, implementation, deployment, etc.

Quick Reminder on Collaborating with Others

Since this is a group project, you may (and should) communicate freely with all of your group members. However, your group is not allowed to communicate with any other groups while working on this project, including outside personnel or consultants. Please use Piazza for your questions and/or comments, and post publicly whenever it is appropriate. If your questions or comments contain information that specifically provides an answer for some part of the assignment, then please make your post private (your group members and OMSCS6310 TAs/Instructors only) first, and we (the OMSCS 6310 Team) will review it and decide if it is suitable to be shared with the larger class.

Best of luck on to you this assignment, and please contact us if you have questions or concerns.
Mark