SENG201 – Software Engineering I
2017
Assignment: Virtual Pets

For assignment administration queries:

Matthias Galster, Jin Hong
matthias.galster@canterbury.ac.nz, jin.hong@canterbury.ac.nz

For assignment help, hints and questions:

Matthew Ruffell, Josh Nimmo
msr50@uclive.ac.nz, josh.nimmo@gmail.com

1 Introduction

1.1 Administration

This assignment is a part of the SENG201 assessment process and requires you to **design, implement, test and document** a software product. It is worth **25%** of the final marks. This assignment will be done in pairs, and you must find your own partners. Please register your partnership on Learn before two weeks from the assignment being handed out. Submissions from individuals will not be accepted. Pairs are not allowed to collaborate with other pairs, and you may not use material from other sources without appropriate attribution. Plagiarism detection systems will be used on your report and over your code, please do not copy others work. You may discuss your assignment *in general terms* with others. Please submit your deliverables on learn no later than **5pm, Friday the 26th of May**. Students will be asked to **demo their assignment during labs and lectures** in the week of the **29th** (week 12). The drop dead date is the **2nd of June at 5:00pm**, with a standard 15% late penalty.

1.2 Outline

This assignment is to give you a brief idea of how a software engineer would go about creating a complete application from scratch that has a nice graphical interface. In this assignment, you will be creating a game that lets you manage and play with virtual pets, along the lines of Tamagotchi, NeoPets and the like. You will need to be able to create pets, buy toys for them, feed them and generally play with them, in a game like environment. The idea is slightly open to allow some room for creativity, but please ensure you implement the main assignment tasks as that is what will be graded.

1.3 Help

This assignment can get confusing and frustrating at times when your code does not work. This will likely be the largest program you have written thus far, so it is very important to break larger tasks into small achievable parts, and then implement small parts at a time, testing as you go. Having a nice tight modular application will help with debugging, so having appropriate classes is a must. If you are having problems, try not to get too much help from your classmates, and instead ask for help from your tutors, Matthew and Josh. You can email them or ask them questions in labs. Always save your work and have backups, do not assume that the COSC department will be able to recover any lost data.

2 Requirements

This section describes what your game must do, and is written as a set of requirements. Try thinking of each requirement as a separate ticket that needs to be closed before others started, and it will help you have code which works and can be built upon, instead of a lot of broken spaghetti code. Hint: Functionality in each subsection can be placed in its own module or class. Modularisation is the key, especially when you begin GUI programming.

2.1 Setting Up the Game

When your game first starts it should:

1. Ask how many players would like to play the game. You should allow between 1 and 3 players.
2. Ask for the amount of "days" the game will span. Each "day" is a round in the game, and is the amount of rounds each player will play.
3. Have a help section that explains how the game is played.

2.2 Creating Players and Selecting Pets

Now, for each player, the game should ask:

1. What the player's name is. Make sure no two players have the same name.
2. How many pets the player would like to play with. Each player should be able to select 1, 2 or 3 pets.
3. For each pet, the game should ask:
   (a) What species the pet is. Each species have different traits, and the player should be able to have different species of pets.
   (b) What the name of the pet is. Make sure that no two pets have the same name. True across all players and pets.
4. There should be some way of viewing attributes of each pet, so that the players can compare pets and make an informed decision about the characteristics of each species.

2.3 Playing the Main Game

Once all players have selected pets, the main game can begin. There will be a series of options displayed to the player. Some of these options constitute an "action", and each pet may only perform a maximum of two actions per day.

For each player, and each pet, the player should be able to:

1. View the status of the pet. This includes viewing its hunger levels, tiredness, playfulness or toilet levels. Pet attributes will be better explained in the "Design and Architecture" section (Section 3).
2. Visit the store, and:
   (a) View objects, such as food and toys, that are for sale.
   (b) Show what objects the player currently owns, their amounts, and the amount of money the player has.
   (c) See the prices of each object.
   (d) See the attributes of the object, attributes will be better explained in the "Design and Architecture" section (Section 3).
   (e) Enable the player to purchase objects such as food and toys, and

    (f) Be able to purchase multiple objects at a time without leaving the store.

3. Feed the pet:
    (a) This counts as an "action", and contributes to one of the two actions per day.
    (b) The player should be able to select what food to feed the pet.
    (c) Food is consumed and removed from the inventory.
4. Play with the pet:
    (a) This counts as an "action", and contributes to one of the two actions per day.
    (b) The player should be able to select what toy to let the pet play with.
    (c) Toys may break if pets are too rough with them, or have been used past its durability levels.
    (d) Broken toys should be removed from the inventory.
5. Put the pet to bed to sleep:
    (a) This counts as an "action", and contributes to one of the two actions per day.
    (b) The pet should be able to rest.
6. Let the pet go to the toilet:
    (a) This counts as an "action", and contributes to one of the two actions per day.
    (b) The pet should feel relieved afterwards.
7. Move to the next day.
    (a) The player should be able to move to the next day at any time, even when there are still actions remaining for each pet. That is, one pet could be fed, while another pet has yet to complete any actions.

There will also be some random events which you will need to implement. Each event only effects one of the pets, and it is possible for two events to happen to two different pets at the same time.

1. Your pet may begin to misbehave:
    (a) The player needs to be alerted to this fact.
    (b) The player needs to have the option of correcting the behaviour with punishment.
    (c) The pet should feel unhappy afterwards.
    (d) If the player does not correct the behaviour, the pet remains misbehaving.
2. Your pet may get sick:
    (a) The player needs to be alerted to this fact.
    (b) The player needs to have the option of paying for treatment, there will be a cost involved.
    (c) The pet should feel happier afterwards.
    (d) If the player does not treat the pet, the pet remains sick.
3. Your pet may die:
    (a) If the pets stats fall too low, such as being too hungry, too sleepy or needing the toilet then the pet should die.
    (b) The player can have the option of reviving the pet once.
    (c) After being revived, if the pet dies again it is dead for the remainder of the game.
    (d) If the player does not revive the pet, the pet remains dead.

2.4 Finishing the Game

Once the players have completed all of their days, a final score will be displayed. How you score is up to you, but we recommend computing a daily score of status attributes.

2.5 Extensions

If you have completed the system as described above then you should be able to get a very good mark. however, once you have the basic system working, you can add some additional features for extra credit. If you decide to try some of these then please discuss your plans with us in the lab, and make sure that you do not break any of the essential functionality described above. Any additional features should be accompanied by unit tests and any other relevant artifacts. If it is not obvious how to use them then please include brief instructions or examples. Here are some suggestions:

- A player may want to save the current status of the game to continue the game later.
- A player may want to save the results of a game. This player or other players should be able to view the results later. Players may even want to keep a list of high scores.
- A player may want to create their own types of pets or toys. These should also be available to other players and players may "trade" toys and pets.

3 Design and Architecture

This section provides some ideas with how your program should be structured. The following are some class candidates, and should be represented in your program. Important things to think about here are interactions between classes, what classes should be instantiated, and what roles inheritance should play.

3.1 Players

All players have a name, and will have a list of pets and toys available to the player. The player will initially have no food and no toys, and must purchase them from the store. The player also has an account balance of how much money they have to spend at the store, and should be initialised when the player is created to a suitable amount.

3.2 Pets

As we know, there are different species of pets and each species have different values for their attributes. Six species of pets will be enough. Each pet has a name and a species. All pets should be able to get hungry, get tired, have a mood like happy or sad or in between, be healthy or sick, be alive or dead, or needing to visit the toilet. Some pets will get hungrier than others as each day passes and will need more food. Some pets will require more sleep than others, and some will like playing with toys more than others. All pets should get heavier when they eat and want to visit the bathroom after eating, which will decrease their weight. Rigorous exercise might make them tired and hungry, depending on the toy.

3.3 Toys

There should be different types of toys, six will be enough. All toys have a price, and prices will vary depending on the toy. Toys will make pets happier when they play with them, and each pet has a favourite toy, which will increase the happinesses even more. Toys will break once they have been played with too much or if a pet is particularly rough with the toy, so you should keep track of a toys durability and how much damage pets do.

3.4 Food

There should be different types of food, six will be enough. All food has a price, and the prices will vary depending on the food. Food will also differ in nutrition and tastiness. By feeding food to a pet, the nutrition value could perhaps reduce the hunger of a pet by a certain amount. Foods with poor nutrition could leave pets still hungry, or good food will completely fill a pet up. A pet will want to visit the toilet with more urgency after a big meal then after a small meal. Tasty food will make a pet happier, and it will depend on how much the pet likes that food. Pets have favourite foods, and their favourite will provide a boost to how tasty the food is.

3.5 Game Environment

The game environment contains your game, and will implement functions to provide the options mentioned above, and will call methods of the above classes to make that option happen. The game environment keeps track of a list of players, and a list of pets to select from. The game environment instantiates players, pets, food and toys, and places the objects where they belong. All of the game logic will be placed in the game environment, such as the `feed()` method may call `pet.feed(food);` or similar. This class will get large, please try and keep it modular.

4 Assignment Tasks

4.1 Writing UML

Before you start writing code, sketch out a UML class diagram of how you think your program will look like. It will help you get an idea of what classes there are, what class attributes are required, and will get you thinking of what classes communicate with other classes (call methods of another class).

4.2 Implementing a Command Line Application

Begin implementing classes, starting with Player, Food and Toys, moving onto Pets and the Game Environment. Make the Game Environment a simple command line application which works in a simple runtime loop which:

1. Prints out a list of options the player may choose, with numbers next to the options.
2. Prompt the player to enter a number to complete an option.
3. Read the number, parse it and select the correct option.
4. Call the method relating to the option and if necessary:
    (a) Print out any information for that option, such as select toy to play with.
    (b) Read in the number for the information offered.
    (c) Parse the number and complete the action.
5. Go back to step 1.

This will enable you to slowly build up features, and we recommend to only implement one feature at a time. Make sure you test your feature before moving onto implementing more features. Once you are feature complete and have a working game, you may move onto implementing a graphical application. The command line application will only be assessed if there is no graphical application, or if there are fatal bugs in the graphical application. In this case, partial marks will be awarded for correct command line functionality.

4.3 Implementing a Graphical Application

You will be implementing a graphical application for your game using Swing, which will be explained in lectures. For the purposes of this assignment, we do not recommend writing the Swing code by hand, and instead using the interface builder offered by the Eclipse IDE. This lets you build graphical interfaces in Swing by dragging and dropping components onto a canvas onscreen, and it will automatically generate the code to create the graphical application you built. Please note, you are required to ensure that any automatically generated code complies with the rest of your code style, so you will need to change variable / method names and code layout.

Once you have built your interface, the next task is to simply wire up the graphical components to the methods your command line application supplies, and to update the onscreen text fields with the new values of your class attributes / member variables. Most of these functions are triggered on `onClick()` methods from buttons. Start small, and complete Section 2.1 "Setting up the Game" first to get used to GUI programming. You might need to slightly adjust your methods to achieve this. Then move onto the slightly more complicated Section 2.2 "Creating Players and Selecting Pets", and then finally start creating Section 2.3 "Playing the Main Game" using what you have learned in the previous sections. Section 2.4 "Finishing the Game" should be straightforward after your previous efforts.

4.4 Writing Javadoc

Throughout your application, you need to be documenting what you implement. Each attribute of a class should have Javadoc explaining what its purpose is. Each method needs to explain what it does, what variables it takes as parameters, and what types those variables are. You should be building your Javadoc regularly, as it integrates into the IDE very nicely, and will aid you in writing good code.

4.5 Writing Unit Tests

You should design JUnit tests for your smaller, basic classes, such as Player, Pet, Food and Toys and their descendants if you think necessary. Try and design useful tests, not just ones that mindlessly verify that getters and setters are working as intended.

4.6 Report

Write a short two-page report describing your work. Include on the first page:

- Student names and ID numbers
- The structure of your application and any design choices you had to make. We are particularity interested in communication between classes, patterns, choice of collection types and how inheritance was used. You might want to reference your UML class diagram.
- Explanation of unit test coverage, its meaning and why you got a high / low coverage.

Include on the second page:

- Your thoughts and feedback on the assignment.
- A brief retrospective of what went well, what did not go well, and what improvements you could make for your next project.
- A signed statement of agreed % contribution from both partners, including a brief paragraph per partner that describes key contributions.

4.7 How to Get Started

We suggest that you carefully read this document, the description of requirements, initial design consideration and then work through the following steps.

- Find a partner and register on Learn.
- Agree with partner on a work schedule and availability. Create a plan with your partner for how to tackle this project. Discuss expectations (e.g., do you just want to pass the course or are you aiming for a high grade grade).
- Set up an Eclipse project.
- Create a high-level class diagram based on the information provided in Section 3.
- Implement basic model classes (e.g., Pet, Toy). Test each class to make sure they work properly.
- Enhance your initial class diagram based on the information provided in Section 2.
- Implement a command line application as outlined in Section 4.2.
- Implement a Graphical application as outlined in Section 4.3.

5 Deliverables

5.1 Submission

Please create a ZIP archive with the following:

- Your source code and unit tests (exported from your Eclipse project using File - Export - General - File System); make sure the exported project contains all relevant files.
- Javadoc (already compiled and ready to view).
- UML class diagrams (as a PDF, or PNG. Do not submit Umbrello or Dia files).
- Your report (as a PDF. Do not submit MS Word or LibreOffice documents).
- A README.txt file describing how to build your source code and run your program.
- A packaged version of your program as a JAR. We must be able to run your program along the lines of: `java -jar usercode1_usercode2_virtual_pet.jar`.

Submit your ZIP archive to Learn before the due date mentioned on the first page. Only one partner of the pair is required to submit the ZIP archive. Before you submit your assignment, please check that your Eclipse project

- compiles,
- can be imported into Eclipse,
- can be run from Eclipse, and
- can be run from outside Eclipse (command line).

5.2 Lab Demos

During the last week of term, you will be asked to demo your program during lab and lecture time. Each team member must be prepared to talk about any aspect of your application, we will be asking questions about any and all functionality. There will be a form on Learn in which you can book a time slot, please ensure you are both available, as you must come as a pair. There are no demos after the drop dead date, but all demos happen in week 12.

6 Marking Scheme

Marking will take into account the professionalism of your approach, the quality of the work you have done, the degree of success you have achieved and the extent to which you have recorded your results and communicated them to the marker.

6.1 Overall Assignment [100 Marks]

6.1.1 Functionality [40 Marks]

We will be testing the extent to which your code meets the requirements using the graphical interface. If your graphical application is broken or faulty, partial marks will be awarded for your command line application.

6.1.2 Code Quality and Design [20 Marks]

We will be examining technical quality (e.g., use of properties, methods, algorithms), your naming conventions, layout and architecture, and use of object oriented features. Quality of your in-line comments is also very important. You may wish to run `checkstyle` over your code before you hand it in.

6.1.3 Javadoc [15 Marks]

We will be looking at your use of Javadoc, and how well it describes the attribute or method you are commenting on, and how well it covers your code.

6.1.4 Unit Tests [15 Marks]

We will run your unit tests to see how well they cover your code, and we will examine the quality of those tests. Try to make your tests do something other than verifying that getters and setters work.

6.1.5 Report [10 Marks]

Your report will be marked based on how well written it is and the information it conveys about your program. Employers place a lot of value on written communication skills, and your ability to reflect on a project, especially in agile programming environments.

6.2 Lab Demos [30% Penalty on Failure to Show Up]

During the lab demos, you and your partner will be showing the examiners how your program works. If you do not turn up to demo in your time slot, you will be penalised 30% of your marks for the assignment. During the demos, you may be asked to:

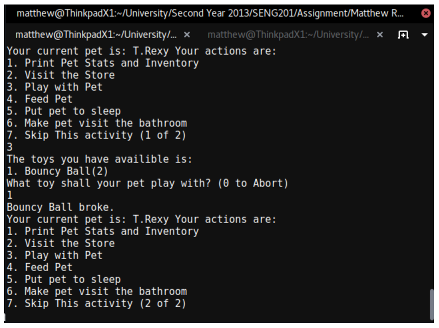
- Construct a new game, and add players and pets.
- Purchase objects from the store.
- Perform actions such as feeding, playing, sleeping or toileting.
- Show that random events occur, such as pets misbehaving, being sick or eventually dying.
- Show that the game can be completed and scores shown.
- Show that the game runs without errors, obvious bugs or crashes.
- Fulfills any or all of the requirements set.
- You may be asked to explain how your graphical interface is written, and point to specific code.

- You may be asked about how inheritance works in your program, again pointing to specific code.
- Anything else that the examiner wishes to ask about.

7 Examples

These examples are from Matthew's reference implementation. DO NOT COPY THEM, they are only to be used as inspiration.

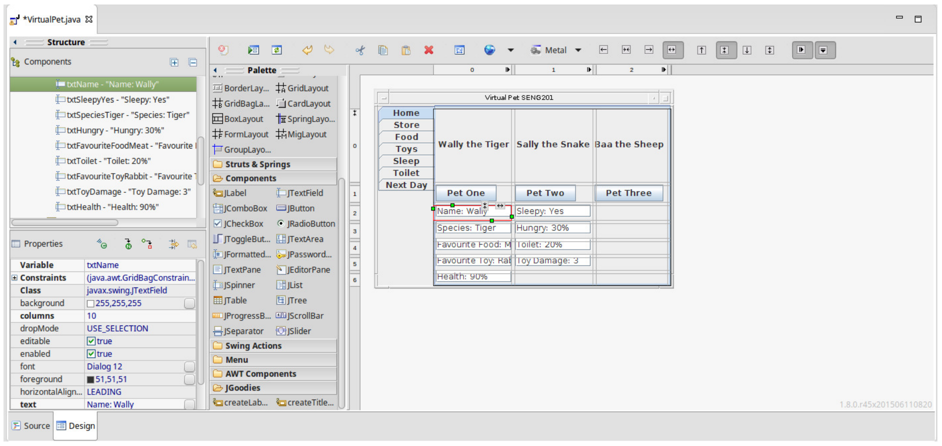
7.1 Command Line Application


matthew@ThinkpadX1:~/University/Second Year 2013/SENG201/Assignment/Matthew R...
matthew@ThinkpadX1:~/University/...
matthew@ThinkpadX1:~/University/...

```
Your current pet is: T.Rexy Your actions are:
1. Print Pet Stats and Inventory
2. Visit the Store
3. Play with Pet
4. Feed Pet
5. Put pet to sleep
6. Make pet visit the bathroom
7. Skip This activity (1 of 2)
3
The toys you have availible is:
1. Bouncy Ball(2)
What toy shall your pet play with? (0 to Abort)
1
Bouncy Ball broke.
Your current pet is: T.Rexy Your actions are:
1. Print Pet Stats and Inventory
2. Visit the Store
3. Play with Pet
4. Feed Pet
5. Put pet to sleep
6. Make pet visit the bathroom
7. Skip This activity (2 of 2)
```

Figure 1: Command Line Application

7.2 Graphical Application

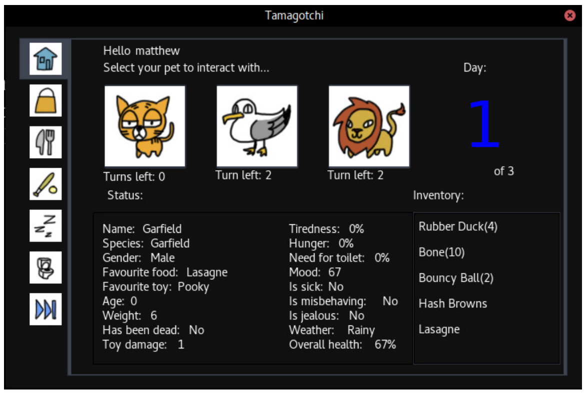


txtName - "Name: Wally"
txtSleepyYes - "Sleepy: Yes"
txtSpeciesTiger - "Species: Tiger"
txtHungry - "Hungry: 30%"
txtFavouriteFoodMeat - "Favourite
txtToilet - "Toilet: 20%"
txtFavouriteToyRabbit - "Favourite
txtToyDamage - "Toy Damage: 3"
txtHealth - "Health: 90%"

Variable	txtName
Constraints	(java.awt.GridBagConstrain...
Class	javax.swing.JTextField
background	255,255,255
columns	10
dropMode	USE_SELECTION
editable	✓ true
enabled	✓ true
font	Dialog 12
foreground	51,51,51
horizontalAlign...	LEADING
text	Name: Wally

Name: Wally
Sleepy: Yes
Species: Tiger
Hungry: 30%
Favourite Food: M
Toilet: 20%
Favourite Toy: Rat
Toy Damage: 3
Health: 90%

Figure 2: Window Builder in Eclipse



Hello matthew
Select your pet to interact with...

Turns left: 0
Turn left: 2
Turn left: 2

Name: Garfield
Species: Garfield
Gender: Male
Favourite food: Lasagne
Favourite toy: Pooky
Age: 0
Weight: 6
Has been dead: No
Toy damage: 1

Tiredness: 0%
Hunger: 0%
Need for toilet: 0%
Mood: 67
Is sick: No
Is misbehaving: No
Is jealous: No
Weather: Rainy
Overall health: 67%

Rubber Duck(4)
Bone(10)
Bouncy Ball(2)
Hash Browns
Lasagne

Figure 3: Matthew's Finished Product