UNIVERSITY OF WATERLOO

Faculty of Engineering

Department of Software Engineering

Scribbler Project: myPetto

Prepared by:

November 29, 2013

Jiaxuan (Eric) Pei
20524193
jxpei

Andrew Zhou
20509626
a25zhou

Xi (Jesse) Jie
20516361
xjie

Zhaotian (Jason) Fang
20519284
z23fang

Zuqi Li
20512622
zq6li

Houze (Allen) Wang
20523903
h353wang

Yuyang (Jake) Si
20515947
y6si

November 28, 2013

Mr. Krzysztof Czarnecki, Ph.D., P.Eng.
Professor
University of Waterloo

SUBJECT: Scribbler Project: myPetto, TEAM 12

Dear Dr. Czarnecki,

Enclosed is the project report for our Scibbler Robot Project. This report has been prepared and written entirely by us and has not received any previous academic credit at this or any other institution. In addition, this report was prepared according to the guidelines and criteria of the “Software engineering Work-Term report Checklist”. The report focuses on the process our group took with designing the myPetto software.

The myPetto robot aims to emulate the behavior of a real pet such that prospective pet owners can experience what it’s like to own a pet without running the risk of buying one and then realizing that the commitment is too much. Throughout the project, a strict set of software engineering rules and principles set out by the waterfall development cycle were followed to ensure that the final product made (myPetto) satisfied the requirements, constraints, and design criteria set during the early stages of the project.

We would like to thank the teaching assistants for helping us solve the constant problems we had with our Bluetooth connection. In addition, we would like to thank the University of Waterloo, the Engineering Faculty and the Department of Software Engineering for providing us with this great opportunity.

Sincerely,

Jiaxuan (Eric) Pei
20524193

Andrew Zhou
20509626

Xi (Jesse) Jie
20516361

Zhaotian (Jason) Fang
20519284

Zuqi Li
20512622

Houze (Allen) Wang
20523903

Yuyang (Jake) Si
20515947

Executive Summary

This report aims show a breakdown of the many components/modules of the project and explain their significance to the project as a whole. In specific, it describes the initial context including the design constraints and criteria, decisions made related to the design of the software, as well as the solutions to problems.

The concept of the project is to create a software using Python that makes use of the scribbler robot. Two ideas are initially considered: the robot as an alarm clock and as a robot pet. Eventually, the former is deemed inapplicable due to constraints of the robot, while the latter seem more flexible and functional.

To simulate a domestic animal, the robot are given three main functions: eating, sleeping, and pooping, in which all of them require interaction from the user. When the robot wants to eat, the user has to “feed” it by putting cards of different colours in front of the robot. For the robot to sleep, it has to be put in a dark place. When the robot wants to poop, it has to be placed on a line so that the robot can follow the line, find the appropriate place to defecate, and return to where it began from. An extra function was added to allow the robot to play fetch with the user. When the robot wants to play fetch, the owner must place an object represented by a flash card in front of the pet to encourage it to move towards the card. The robot is also integrated with functions that control the chances of these events, simulating the seemingly random nature of animal behaviours. In addition, to keep a pet happy the user must interact with it and play a game of fetch with the robot.

Solutions to most of these functions rely on the sensors. For example, front sensors are used to detect objects and brightness, and the bottom sensor is used to detect line beneath the robot. The camera is used to detect different colors and the speakers are used to make signals for the users.

Table of Contents

List of Figures

1 Introduction

Created by Parallax Inc, the Scribbler robot is a low-cost, programmable robot armed with a variety of sensors. Capable of brightness detection, obstacle avoidance, and graphics manipulation, the Scribbler is well-suited to solve a variety of problems. Due to its accessibility and power the robot has attracted a diverse user base, finding uses in industry, education, and the homes of hobbyists.

Calico Myro is an interface for programming robots whose purpose is to assist in introductory computing courses. Developed by the Institute for Personal Robots in Education, the library allows programmers to take advantage of the Scribbler Robot's functionality. Calico Myro provides methods to control movement, I/O, object/light/colour detection, sound, and graphics manipulation. Based off of Python, Myro's functions are high-level and easy to use, so that even students are able to use the Scribbler robot to its full extent.

The purchase of a pet is a large commitment for any family, and can be a major challenge for both children and their parents to handle. Not only is a pet a financial burden, oftentimes the child will leave responsibility of the pet over to the parents as soon as the child realizes the chores associated with owning a pet, leading to additional, unnecessary work for their parents. For families with children, and especially those with children who want pets, it is advisable to purchase an animal only if the children are responsible enough to take care of it. But there is currently no easy way of determining if a child is capable of owning a pet.

2 Problem Specification

2.1 Problem description

In summary, the problem at hand is that it is very difficult for parents to judge whether or not their children will have the responsibility, commitment, and interest required to properly own and take care of their own pet. To address this issue, the Scribbler Robot can be used to mimic a real life pet to give parents the opportunity to see how their children handles caring for a pet without the hassle and expense of buying a real one.

Therefore, armed with a sufficient knowledge of Python 2.4.4, the Calico Myro Library, and ambitious minds, the team is given the challenge to program realistic pet-like behavior and functionality to the Scribbler S2 Robot.

2.2 Proposed Solution

To achieve the objective of preparing young pet owners, the robot must simulate the behavior of a real pet. There are four main actions that the robot will exhibit: eating, sleeping, excretion, and playing fetch with the user. These actions are all independent of one another such that none of the actions when executed should interfere with the success of other actions happening.

To emulate eating, the robot will give out an audio signal and demand to be fed one of the three types of food available: apples, blueberries, and grapes. To simulate the feeding the robot, the user will show the robot one of three colored cards, with each color corresponding to a distinct food type. By placing the colored card over the camera of the pet, it should recognize the food being fed to it and output an appropriate sound to mark the end of feeding.

Furthermore, to simulate the event of putting the pet to sleep, the robot must be moved to an appropriate environment that it can sleep comfortably. In this case, the location is defined as a place with little to no light. The light sensor of the robot will aid in determining if the location is appropriate for the robot to sleep.

In addition, for the event that the pet needs to defecate, the user is given the task of potty training it. More specifically, the user must place the robot on a track laid out by a dark line that will lead to the “bathroom”. The robot’s line sensor will be used to follow the track to the “bathroom.” After, with no additional user interaction, the robot will finish its business, turn around, and proceed back the way it came to the beginning of the track.

Finally, when the pet wants to play fetch with the owner, the user has to bring a specific flash card up to the robot that the robot will be looking for. After sensing the flash card, the robot will move toward the flash card, simulating the event of throwing a ball for the pet to fetch.

One of the biggest hurdles to overcome to become a successful pet owner is to realize that the pet is not going to take care of itself. More specifically, the pet will rarely ask for to be taken care of, let only tell the pet owner exactly what it wants. The pet owner is the person that needs to take the initiative to proactively care for the pet without encouragement from others. In order to test the users for the same quality while playing with myPetto, the pet will never ask to be fed, put to sleep, etc. Instead, the user will be shown the pets health for each of the four actions the pet can do so that the user must decide from his/her choice what he/she needs to do to keep his/her pet healthy.

2.3 Alternative Solution

Before the group decided on creating a domestic animal simulation device out of the Scribbler S2 Robot, the group considered many different ideas ranging from the most basic of the basic to the outrageous. One of the ideas that the team contemplated for a very long time was to make an alarm clock out of the Scribbler S2 Robot. This was the original idea however it was to a halt after assessing the realistic viability of the project. Needless to say, the idea had several issues which did not have realistic solutions and thus the potential project was not used.

The idea was to make an interactive and highly customizable alarm clock for the heavy sleepers who just sometimes cannot wake up. Like all alarm clocks, the alarm would go off at the time which the user specifies. However, the goal of the project was to make a more interactive alarm clock. Thus using a Bluetooth connection with one's Android device, the team was prepared to program the robot such that it would play a song from the Android device at the time when the user specifies. Furthermore, to prevent the user from using a "Snooze" button, there would not have been a "Snooze" button on the Android device but rather it would be located on the Scribbler robot. Ultimately, the goal is to get the user out of bed and therefore it would be logical that the "Snooze" button would not be within arm's reach. The robot would have been programmed such that at the user-specified time, it would start to move outside the room and start playing the song. At first, the song would start at a low volume but as the robot gets further and further away, the volume of the song increases. The user would then have to get up, find the robot outside the room, and shut it off. This makes it so that the user must get out of bed to stop a very loud sound, thus achieving the goal of waking up the user. The potential project did not go according to the team's plan as several shortcomings were found in the team as well as the equipment.

One of the reasons the group chose the myPetto project over the alarm clock idea was the limited amount of possibilities that an alarm clock holds compared to that of a simulated domestic animal. The alarm clock could only function as an alarm clock whereas a pet, for example, could do an infinite amount of things, including waking the user up in the morning. This limitation did not seem appealing, making it one of the reasons that the alarm clock was not used.

Moreover, as the functions in the Myro Library were tested at the start of the project, the team discovered that some of the key functions for the alarm clock idea did not work as expected. Firstly, the speed of robot is incredibly slow compared to what was imagined. Because the robot is supposed to escape from the user, it would need to be at least as fast as a semi-conscious human being. In reality, the robot was just too slow to be effective. Furthermore, the robot's volume was very quiet compared to what was expected. The point of an alarm clock is to wake up the user, however, the group felt that the robot's volume was far too small to wake someone up, especially the heavy sleepers that the project was aimed towards. Simply put, the limitations of the robot kept the team from pursuing the idea of an alarm clock because the robot could not adhere to our expectations and the requirements of the project, leading us to think about abandoning the project. What really led us to abandon the project had to do with the technical issues that arose once the group looked further into the feasibility of the project.

In conclusion, project Alarm Clock was not chosen due to various factors which, in the team's opinion, made the Scribbler S2 robot unappealing to both make and use as an alarm clock. This included technical issues with the software, limitations of the robot's hardware and the limited amount of possibilities that an alarm clock represents. After a discussion in which the team came to this conclusion, project Alarm Clock was deemed not usable and was therefore scrapped.

3 Design Constraints

One of the major design constraints is time - the project has to be complete within about two months. Combined with the busy schedule of the members, the time constraint means that the functionality of the robot must be simple enough to implement during the limited amount of time. Another constraint is the fact that the programming has to be done in Python. Like all other programming languages, Python has its own specific set of syntax and conventions that need to be followed which members may not be familiar with. Therefore, time normally used for project design needs to be allocated instead towards learning Python in order to become familiar enough to utilize the Myro library. In this project, the Myro library is used for basic robot functions. Since there is only so many different kinds of functions available to use, it limits the potential functionality that can be made using the library functions only. The hardware of the robot is the most pressing constraint when designing the project. The reality of it is that the code and functions created for any project is done assuming that the hardware of the robot will function correctly. As a result, any hardware failure or a Bluetooth connection failure will cause sever damage in functionality since the assumption was made that the hardware should work perfectly. In addition, the quality and performance of the hardware is another constraint for the functionality of the robot. Drops in performance of hardware like the Infrared sensors or the camera can severly impact the functionality potential in object and color detection, for example.

4 Criteria Description

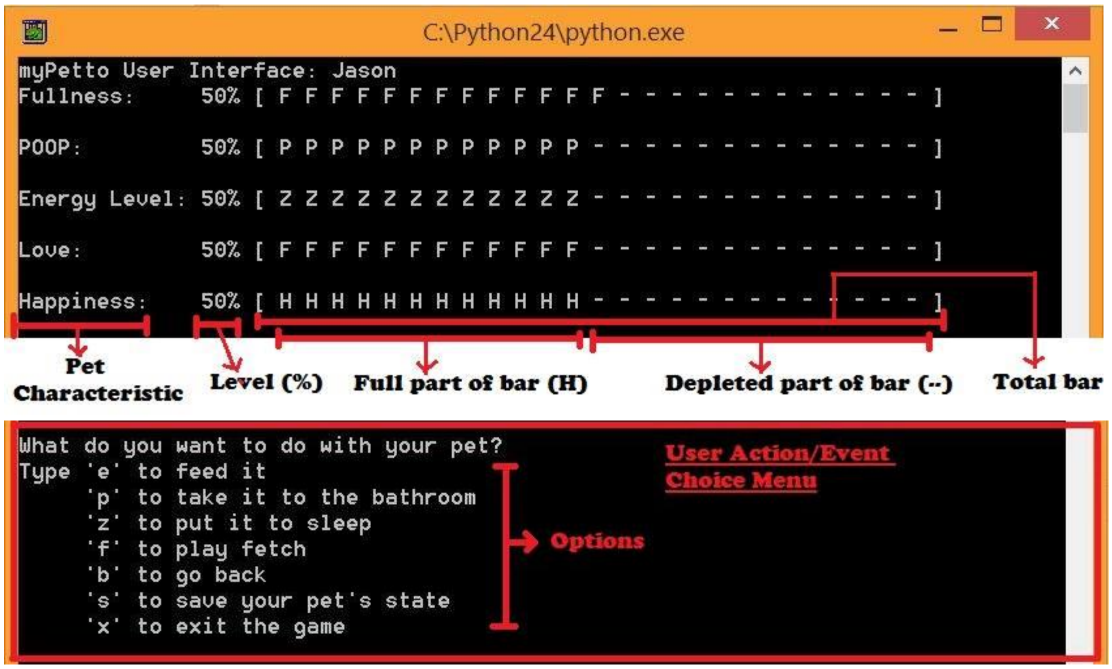
Since the software is aimed to be utilized by all-perspective pet owners, from young children to grown adults, ease of use is one the most important criteria. There must be a simple yet thorough user interface to effectively allow straightforward human-pet interaction between the robot and the user. Additionally, there are certain main criterion regarding the functionality that must be satisfied. The first of which is that the software should be able to mimic a domestic pet as realistically as possible and to simulate some of the major events associated with pets, such as eating, sleeping, playing, and potty training. In order to make the robot more realistic, some of these events have to be proportionally randomized to generate a realistic pattern of a pet's lifestyle. Finally, a pet's existence should not be restricted by the amount of time the program runs. A real pet would continue to be a loyal companion to its owner day after day. In other words, the program should be able to remember the pet's state after the program is finished.

5 Design/Analysis

This section of the report will include design and analysis for each of the main functions of the software.

5.1. main.py

The script "main.py" is the script that is executed to start the program and is responsible for utilizing all the functions available to it from the other scripts to make the program work. The main script is like a control class in the sense that it has little functionality to add to the program, but is responsible for using all the functions supplied by other scripts in a way that gives meaning to the functions.



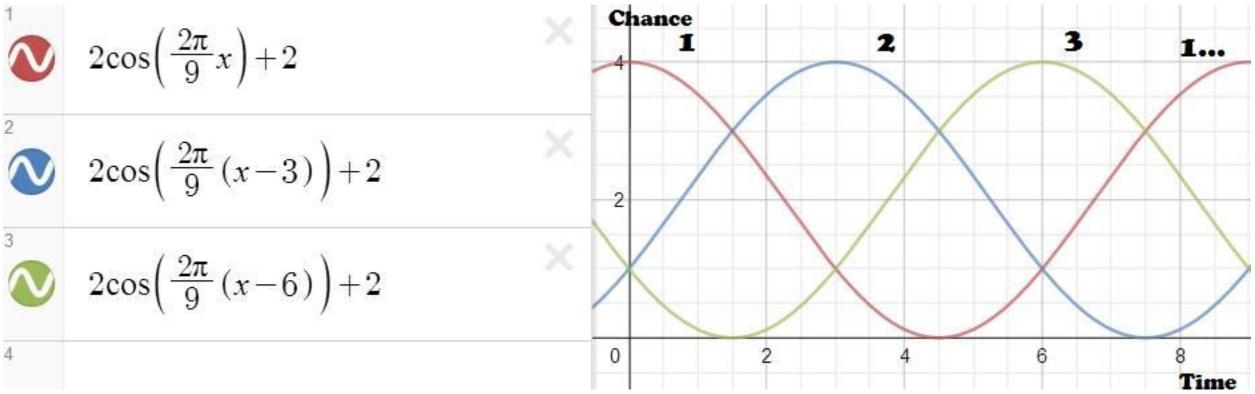
```
myPetto User Interface: Jason
Fullness:     50% [ F F F F F F F F F F F F F - - - - - - - - - - - - - ]

POOP:         50% [ P P P P P P P P P P P P - - - - - - - - - - - - - ]

Energy Level: 50% [ Z Z Z Z Z Z Z Z Z Z Z Z - - - - - - - - - - - - - ]

Love:         50% [ F F F F F F F F F F F F - - - - - - - - - - - - - ]

Happiness:    50% [ H H H H H H H H H H H H - - - - - - - - - - - - - ]

What do you want to do with your pet?
Type 'e' to feed it
     'p' to take it to the bathroom
     'z' to put it to sleep
     'f' to play fetch
     'b' to go back
     's' to save your pet's state
     'x' to exit the game
```

Figure 5.1: Example console screen showing pet stats and commands

The main functionality this script adds is the pet customization at the beginning of the program, the saving/loading feature, and the user interface that is shown while the pet is alive. In order for the user to feel more connected to the pet he/she is going to create, the user must be able to customize the pet in some way to make it his or her pet. This is why the user should at least be able to give the pet a custom name, or a random name if he/she can't think of a name. In addition, the script is responsible for saving the pet's state at the exact moment to a file and also for loading that pet file as needed. Finally, a user interface of some kind must be implemented in the program in order to give the user some indication as to how his/her pet is doing in terms of health and happiness. The interface must be laid out in an appealing manner such that the user can quickly identify which of the pet's characteristics are good and which needs to be higher. In this case, a bar was used to indicate the health level of each of the pet's characteristics based on how full the bar is and a percentage accompanied with each bar.

For the robot, "main.py" will be used primarily to handle the task of calling the appropriate functions to execute any event the user will want to do with the pet at any time the user wishes during the pet's life. In order to accomplish this there must a loop that runs for an infinitely long amount of time as long as the pet hasn't died yet, which is decided by whether one of the four bars depletes to 0. This loop is crucial to making the robot act as realistically as possible. In this loop, the program must handle many tasks such as making the robot idle, changing the amount in the bars with time, changing the pet's food preference with time, and deciding whether the user wants to do something with the pet. After the program determines that the user indeed wants to execute an action, the program must get the user's input to correctly execute the action specified by the user.

A design goal for this project was to make the robot mimic the behavior of a real pet as closely as possible. One of the ways this is achieved in the main script is by the idle function. This function guarantees that the pet will move in a completely random manner so that its direction, speed, and time will all be randomized, similar to the movements of a pet in real life. The function also makes sure the pet only moves when it idles 25%, since a pet in real life is not constantly moving randomly when it has nothing to do. In addition, the script implements a different sinusoidal function to represent the chance of the pet wanting to eat of the three types of food as shown below:



1 $2\cos\left(\frac{2\pi}{9}x\right)+2$

2 $2\cos\left(\frac{2\pi}{9}(x-3)\right)+2$

3 $2\cos\left(\frac{2\pi}{9}(x-6)\right)+2$

4

Figure 5.2: Sinusoidal function showing the pet's eating habits

The goal of this functionality is to give the impression that the pet has varying food preferences that change as the pet eats more and more food given by a counter variable "x". The function are defined the way they are above (same function except for phase shift) is to make sure that each type of food has their own point in time where their chance/preference is much higher than the chance of the other food. Since these functions repeats periodically, it models how a pet will constantly change its food preference periodically as well as it gets sick of one food and starts to prefer the other types.

In order to teach the user the responsibilities that come with owning a pet, the program makes sure the pet's health levels (eat, sleep, poop, etc.) deteriorate at a rate much higher than in real life. This teaches the users a valuable lesson about how a pet is a real animal that needs to be taken care of, and not something that can be ignored/disregarded whenever the user chooses. It is programmed that the pet will die after one of the pet's four bars completely depletes. This is realistic as it shows how consistent disregard and lack of care for the pet can cause its health/happiness to deteriorate to the point where it dies.

The saving and loading functionalities are key in helping the user maintain a long relationship with the pet, even after closing the program. This mimics the scenario in real life since pets are often under the user's care for several years from birth to death of the pet. The reason saving/loading feature is included is to create a lasting connection between the pet owner and the myPetto robot.

However, when creating the function for idle movement, a dilemma arose where if the function makes the robot move completely random, then it would be almost impossible for the robot to move in such a way that it will avoid obstacles, since the IR sensors on the robot can only detect object in front of it, which only covers about 25% of where the obstacle could be. Therefore, to solve it and to preserve the realistic idle movements of the robot, an assumption must be stated in which the robot must be placed in an obstacle-free environment to prevent these events from happening.

5.2. foodinput.py

To mimic a real pet eating food, this project uses different coloured flash cards that represent various foods. In order for the pet to determine which flash card corresponds to which food, it is important to store all of the flash card's RGB values, along with their associated foods, into a text file. To assist the "hungry()" function, it is necessary to have a method to manage this text file and add new foods to the database.

The "Foodinput.py" script adds new foods and their RGB values into a text file named "food.txt". The text file follows the format: [Food_name] [R] [G] [B]. It begins by taking in user input to determine the food name. The script then takes a picture with the robot's camera and loops through its individual pixels to determine the average RGB value of the image. Finally, it appends this data into the "food.txt" file and saves it.

Just as with the "hungry()" function, the major barrier that this script faces is the poor quality of the Scribbler robot's camera. This results in a distortion of the flash cards' colours from the robot's perspective, and the error is so large that a change in lighting often results in the robot thinking that the card is a different colour altogether. The use of "autoCamera()" and turning off the LED lights on the robot helps to combat this, but nevertheless it is important to operate the robot only in areas with good lighting to avoid any problems.

5.3. Read.py

To simulate a real pet successfully, the product must be able to emulate an animal's eating habits. The pet should have a diverse diet and a variety of foods that it could want to eat. Just like with a real pet, the owner should not be able to determine the exact food that the pet wants to eat at any given point in time. However, they should be able to know what food the pet most likely craves at the moment.

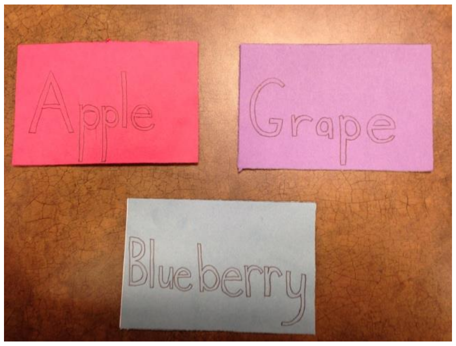


Figure 5.3: Flash card for food

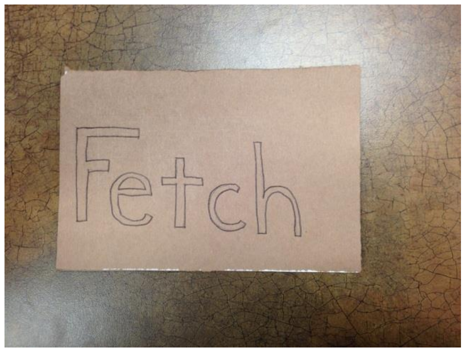


Figure 5.4: Flash card for fetch

The "hungry()" function describes the food-eating sub process for the pet. It uses coloured flash cards to represent different foods, and takes advantage of the Scribbler robot's camera and graphic manipulation library to distinguish between them.

The method begins by taking in a parameter called “foodName” that contains the name of the food which the pet wants to eat. It then searches through the “food.txt” file; a database of different foods and the RGB values of their associated flash cards (see “Foodinput.py” for more details). The function then executes its main loop, where it plays a distinct sound to tell the user that the pet is hungry, and then tries to determine if the pet is being fed. To do this, it takes a picture with its camera and then calculates the average RGB value across the pixels of the image. The method then compares this value to the one it found from the text file, and if they are within a certain range the function plays a “success” sound, telling the user that he/she has fed the pet, before exiting from the function.

The main problem that inhibits this function is the poor quality of the hardware. The quality of the camera’s pictures is so low that the colours of the flash cards are easily distorted by a change in lighting or a slight adjustment in the distance between the card and the camera. Turning off all of the LEDs on the fluke card and repeatedly calling “autoCamera()” helps lessen this problem however it is still advised to operate the pet in a well-lit location.

5.4. Sleep.py

One of the essential abilities of any animal is to sleep. Naturally, our project, which is supposed to emulate the behaviour of a domestic animal, has a sleeping function. It works using an internal clock which, using random chance, tells the robot when it wants to “sleep”. The user must comply with the animal’s “demand” and put it to sleep much like in real life. To put the robot to sleep, the user must put the robot in an area with little light within a certain time frame. The robot senses the level of light and if the robot has been put to “sleep” then its “happiness” will go up. On the other hand, the robot’s “happiness” will go down if it is not put to sleep within the specified time frame.

While writing the code for the sleep function, there were not that many problems. The main problems that occurred were the various Bluetooth issues as well as the testing. Once the team got the Bluetooth technology to work consistently, finding a threshold and then testing were the only things that needed to be done. Although this seemed like a simple task, the function which was used, getLight() (a function from the Myro Library), did not seem to give a consistent result. At some points, the function would return a level of brightness higher in a dark box than in a fully lit room. It was a truly bizarre problem and it baffled the team for a long time. The issue has, evidently, been resolved and it was because of two main solutions.

The first solution was the battery level of the robot. It seems that once the battery levels drop below a certain threshold, all the sensors become inconsistent. This affected all the functions but after the team came to realize this fact, the fix was easy; the team simply charged the robot’s batteries. Once this finished, many of the function values for our various functions seemed to stabilize however the getLight() function still seemed to give inconsistent answers. The real solution was discovered as the online manual was scrutinized for details about the getLight() function’s inconsistent answers.

The solution that fixed everything for the program was the getBright() function (also from the Myro Library). This function ended up giving the consistent readings which were needed for the sleep function. After running several tests, it did not falter a single time. This was the function which the group ended up using to get the light intensity from the surroundings. Using that information it is possible to determine if the robot was put to “sleep” or not, meaning that the whole sleep function revolves around the getBright() function.

5.5. Fetch.py

The Fetch() function is designed to simulate playing fetch with the Scribbler robot with a flash card. The function should be able to play a distinct song that tells the user the robot wants to play fetch. While playing fetch, the robot should continuously take pictures and calculate the average RGB values of their central portions. It should then take the calculated RGB values and compare them to those of the RGB values of a set constant based on the flash card used for fetch. If the values are within a certain range, the robot moves forward. If the robot has moved forward successfully 5 times, it plays a success sound and the function returns 1 indicating success.

While the function is running, it uses makeSong() to create a tune that tells the user the robot wants to play fetch. Then, it uses takePicture() and nested for loops to calculate the average RGB values of the central half of the picture. After comparing the calculated RGB values to the constants, the function uses an if statement to check if the values are within a certain range signified by a global variable VARIANCE. If the statement returns true, the robot moves forward for 1 second and maximum speed and the fetch counter increments by 1. Once the counter reachs 5, the robot plays a success sound, returns 1 and returns to idling.

One problem encountered while testing the function was that the camera settings would distort the colours of the image. To fix this problem, all the frontal LEDs were disabled and the autoCamera() function was called every time the robot took a picture. This solved most of the problems regarding the camera.

5.6. Poop.py

The defecation script is a set of functions that simulates a pet going to the bathroom. It is used to decrease myPetto's "POOP" level and thus increase the pet's overall happiness. The script involves myPetto to follow a premade track to the "toilet" and to return back to its original starting point. The idea of this script is to mimic the "potty training" process of a real pet with its owner to teach children the responsibility of caring for a real pet. This script mainly utilizes the Scribbler S2's left and right line sensors located on the bottom of the unit to detect the track. The line sensors return a value of 1 if they sense dark colour and they return a value of 0 if they sense a light colour. The track is made using a blank piece of paper with the track drawn using a thick tip sharpie.

The "potty training" process begins when the user calls the defecation script through the console user-interface. myPetto then starts to try to detect for a track. A timer will start to give the user twenty seconds to put myPetto on the track. If the user fails to put myPetto on the track in twenty seconds, the user will fail the "potty training". Once both of its line sensors sense a track, myPetto will move forward slowly and steadily. Once a line sensor goes out of the track, myPetto will adjust itself by rotating the other way. When both light sensors go off the track, it means that myPetto has reached the end of the track. myPetto will then play a beeping sound to simulating pooping, rotate until it faces the other way, and finally proceed to move back to the original spot using the same track following algorithm.

The team has encountered two major obstacles during the development of the defecation script. One of which was the odd placement of the line sensors. The line sensors were placed on the rear end of the unit. Since the Scribbler's axis of rotation is in the centre of the unit, the line sensors often went off the track unexpectedly. To correct this issue, the team has decided to design the scribbler so that it moves backwards into the track instead. This allowed the line-sensors to work as intended as they are now in the front of the unit. The other hindrance the team came across was the accuracy of the line sensors. The line sensors often sensed the carpet as a track and thus returned a false value. The team accommodated this limitation of hardware by making the robot only start to detect track when it is on a white surface.

The defecation script is composed of a main function and four side functions. On one hand, there are two status-checking functions for updating the status of the line-sensors: 'checkTrack()', 'getTurnDir()'. On the other hand, there are two movement functions to carry out the specific movement commands for myPetto to carry out the "potty training" procedure: 'trackMove()', and 'startMovement()'. Finally the main function acts as a link between these four side-functions and organizes them together to fully carry out the script.

'checkTrack()' is used to check whether both sensors are sensing dark colour on the ground. If this function returns a Boolean value of 1, it shows that myPetto is perfectly on track, while a Boolean value of 0 means that myPetto is off the track.

'getTurnDir()' indicates the direction myPetto needs to turn to stay on the track. This function returns "left" if the right line sensor goes off the track. Likewise, it returns "right" if the left line sensor goes off the track.

'trackMove()' is the function that governs myPetto's movement on the track. It moves myPetto forward while checking the status of each individual line sensor using 'getTurnDir()'. If it senses that a line sensor is off the track, it will proceed to rotate in the other way to adjust the robot's direction.

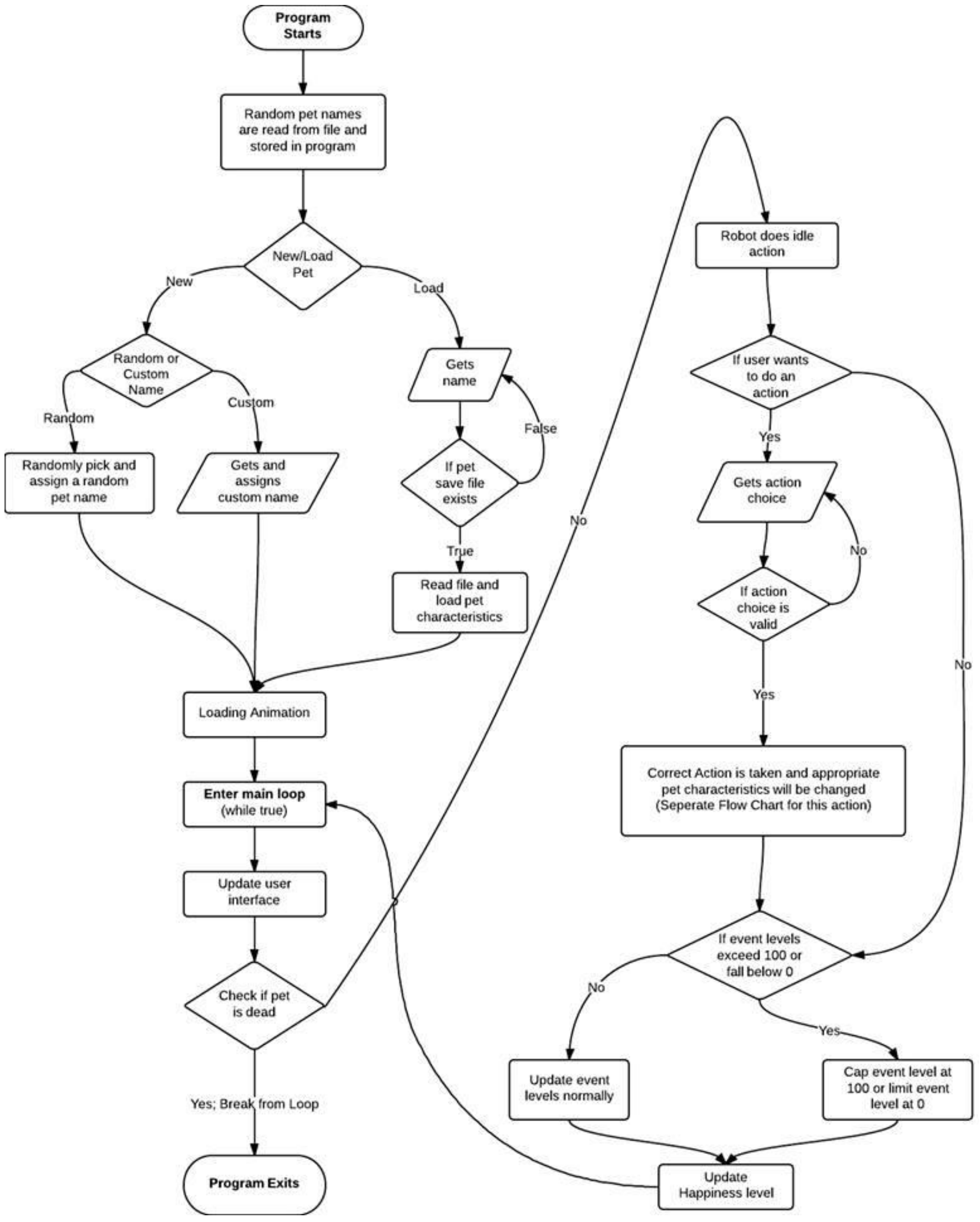
'startMovement()' handles the overall movement of myPetto. This function is called once myPetto is on the track. This function calls trackMove() to move myPetto along the track, plays a held note to simulate pooping, rotates myPetto until it relocates the track, and finally it calls trackMove() once more to return myPetto to its original starting position.

6 Conclusion

In conclusion, the myPetto robot is a good solution to the problem and a good idea for the project as it gives real life functionality to the Scribbler 2 Robot. The three main functions, eating, sleeping and excretion, all simulate those of a real pet in realistic, fun and interactive way, in addition to the newly implemented "fetch" function. Regular and thorough tests of the program were conducted to keep these functions working consistently when outside factors such as testing environment and hardware condition changes. Mathematical models were devised to accurately and realistically model the food habits of the pet. In addition, a nice, visually appealing user interface is included to show all of the pet's stats and to help the user choose an action to do with the pet. Bugs were fixed in a logical and timely manner. Group decisions were made rationally and always under a consensus. The design project helped develop a foundation in Python programming as well as improving the transferable skills needed to work effectively and efficiently in a team.

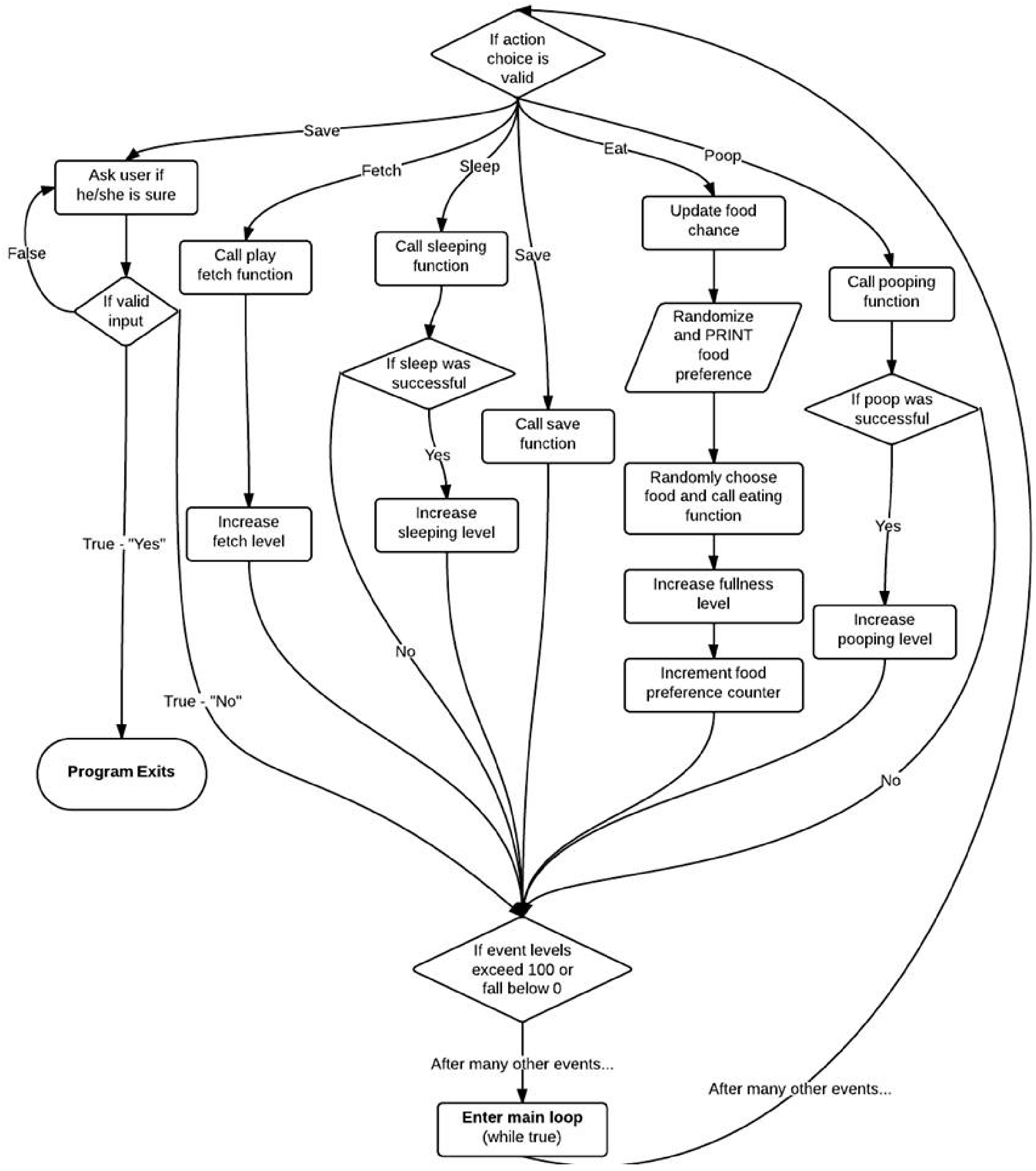
A. Appendix

A.1 Main program flow (without event chooser)


Program Starts
Random pet names are read from file and stored in program
New/Load Pet
New
Load
Random or Custom Name
Random
Custom
Randomly pick and assign a random pet name
Gets and assigns custom name
Gets name
If pet save file exists
False
True
Read file and load pet characteristics
Loading Animation
Enter main loop (while true)
Update user interface
Check if pet is dead
Yes; Break from Loop
Program Exits
No
Robot does idle action
If user wants to do an action
Yes
No
Gets action choice
If action choice is valid
No
Yes
Correct Action is taken and appropriate pet characteristics will be changed (Seperate Flow Chart for this action)
If event levels exceed 100 or fall below 0
No
Yes
Update event levels normally
Cap event level at 100 or limit event level at 0
Update Happiness level

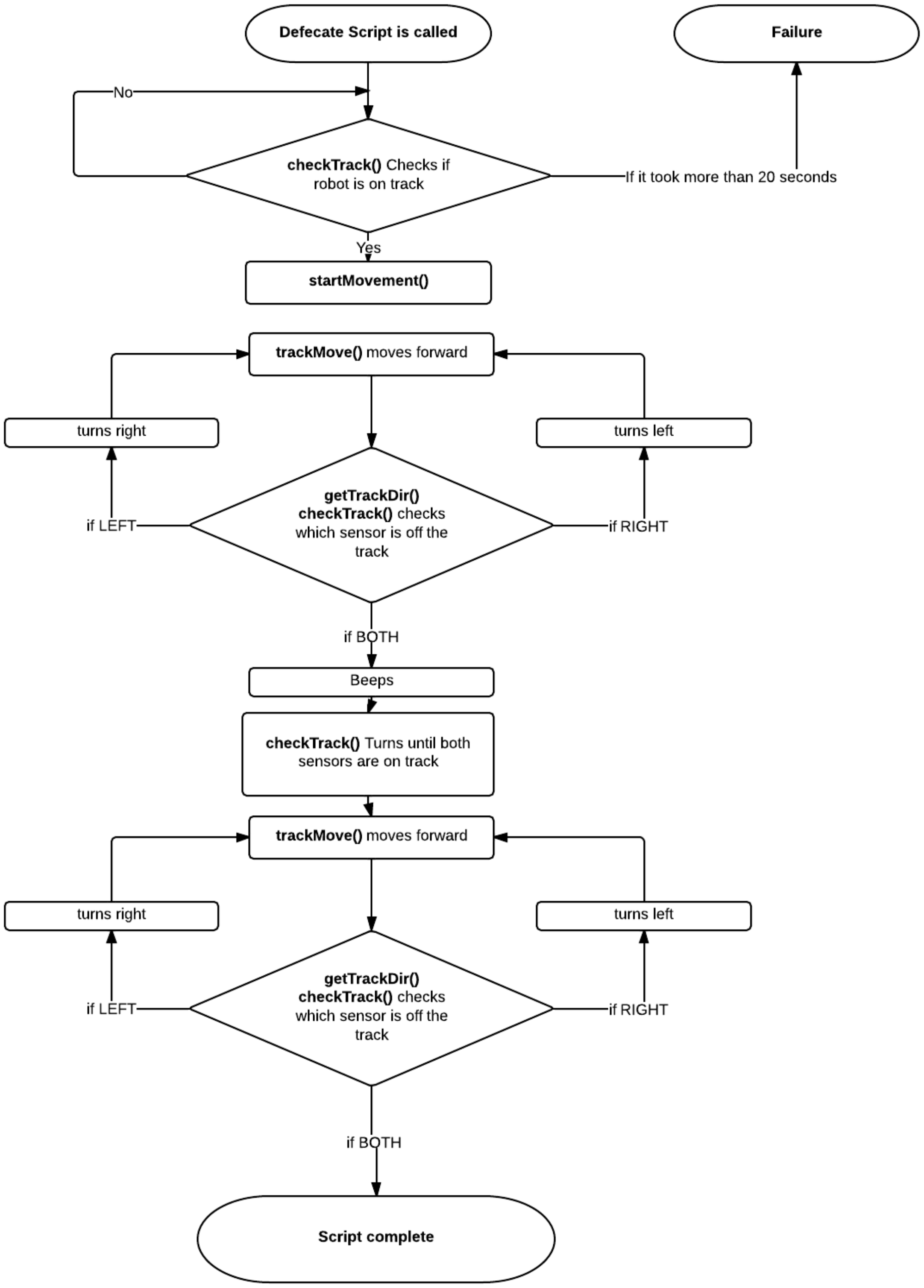
This detailed flowchart shows the main flow of events starting from the execution of the "main.py" script and ends with program termination. The section where the user chooses an action for the pet to do is omitted and is put in a separate flowchart (A2).

A.2 User event chooser flow



This chart details the logic behind the algorithm handles the event choice from the user and executes the appropriate action based on the choice.

A.3 Poop script Algorithm


Defecate Script is called
Failure
No
checkTrack() Checks if robot is on track
If it took more than 20 seconds
Yes
startMovement()
trackMove() moves forward
turns right
turns left
getTrackDir() checkTrack() checks which sensor is off the track
if LEFT
if RIGHT
if BOTH
Beeps
checkTrack() Turns until both sensors are on track
trackMove() moves forward
turns right
turns left
getTrackDir() checkTrack() checks which sensor is off the track
if LEFT
if RIGHT
if BOTH
Script complete

This chart details the flow of logic behind the algorithm that helps take the robot to the washroom and back along the track.

References

"Myro Reference Manual," IPRE Wiki, [online] 5 October 2006, http://calicoproject.org/Myro_Reference_Manual (Accessed: 24 November 2013).

"Stack Overflow," Stack Overflow, [online], http://stackoverflow.com/ (Accessed: 24 November 2013).

A. Morton, "Software Engineering Work-Term Report Checklist," Software Engineering Work-Term Report Checklist, [online], http://gsdlab.org/se101f13wiki/images/6/67/WkrptFrmtList_10.pdf (Accessed: 24 November 2013)

Fred L. Drake, Jr., "Python Library Reference," Python Documentation, [online] 18 October 2006, http://docs.python.org/2.4/lib/lib.html (Accessed: 24 November 2013).

Robert L. Oakman, "Flowchart Symbols," The Computer Triangle, [online] 10 September 1996, http://www.wiley.com/college/busin/icmis/oakman/outline/chap05/slides/symbols.html (Accessed: 24 November 2013).