UTM Python Graphics Workshop

Michael Liut
michael.liut@utoronto.ca

June 5, 2018

1 Content

For the GitHub repository: https://github.com/MichaelLiut/UTM_PythonGraphics

For documentation on Zelle's Graphics: http://mcsp.wartburg.edu/zelle/python/graphics/graphics.pdf

For documentation on PyGame: https://www.pygame.org/docs/

2 Overview

Some of the questions that will be discussed at the workshop are:

1. What is PyGame and Zelle Graphics?
2. What are some examples I can show in class?
3. What are some smaller assignments I can give my students?
4. How can I create a larger project that involves the creation of a game?
5. Where can I go for help with Python Graphics?

3 Zelle Graphics

There is a reference PDF in the repository (called "Graphics Help"). This is a great guide to get you started! But the true idea here is to visually represent classes as objects. Think of a circle, sure you can create a class that has attributes (e.g. outer line colour, inner fill colour, size, etc...) but for students that does not mean a lot until you show them that they can create these (i.e. the students have visual representation). Then you can backtrack and speak to the object orientation and the importance of modularity; also the software design principle *low coupling, high cohesion*.

Coupling refers to the level/degree of dependence between different modules/classes, where *cohesion* refers to the level/degree of which the elements of a module/class belong together. Think of having a very high level of functionality within a module/class (a lot of collaboration) and strong independence for each module/class added to the program/application (freedom to operate without being dependant on others to survive/run).

4 PyGame

4.1 Structuring a Game

1. Set a clear set of Learning Outcomes/Objectives
2. If each student is to build their own game, then perhaps take an inverted pyramid approach; allowing those that tasked themselves with too big of a product the opportunity to narrow their focus.

3. If each student is expected to replicate a game, you have much more control! You can even provide skeleton code and have them add their own *unique* feature(s).

4. Set regular checkpoints! This is easier if you are following (3), however, if option (2) is more appealing then here are a few suggestions:

   (a) Drawing and refreshing the canvas.

   (b) User/Player input and sprite movement.

   (c) Physics/Math and collisions.

   (d) More advanced/bonus features:

      i. Randomization.

      ii. Artificial Intelligence.

4.2 Example: Particle Flocking System

This is a complex program with many parts, so some can be given! For example, it can be structured into three parts:

1. Main Program

   - Create your canvas and initialize PyGame.
   - Randomly generating the particles.
   - Displaying the particles to the screen.
   - Updating the location of particles. Look into double buffering!
   - Button/Mouse click(s) and their action(s).
   - Reset the canvas.
   - Modify range, speed, etc...
   - Maybe a bonus? Spinning the particles? Strobbing the colours?

2. Math/Physics Calculations

   - Computing the distance between two points.
   - Computing the length of a vector.
   - Normalizing the vector.
   - Moving the point.

3. Particle Class

   - Colour
   - Size
   - Direction
   - Range
   - Speed
   - Starting Position