Introduction to Game Development with Pygame
(Python 2D Game Prog.)
PHY-109 J-term 2013
Gustavus Adolphus College

Description:

This course is an introduction to computer programming through an application of the Python language. The elements of the language will be taught as students develop computer games with the Pygame interface to the Simple Directmedia Layer (SDL) library. Students will develop a simple physics engine (algorithms that model object motion and interaction) as well as apply more advanced open-source engines (Box2d). High school math will be helpful but not required. There will be instructive exercises as well as longer project work. The course concludes in a competition with teams battling for survival in multi-player network games created by the students.

Goals:

The primary goal is to introduce students to computer programming by developing games. The intent is to teach the Python language through an engaging application of the Pygame game-development environment. There will be emphasis given to Object-Oriented Programming methods (OOP).

A secondary goal is to introduce students to numerical methods for modeling physical systems. Students will develop their own simple physics engines. This will involve a very basic introduction to numerical solutions of differential equations (Euler method). A student background in high school physics and calculus is beneficial but definitely not required.

There will be some overview discussion of how the mathematics of advanced physics engines can be used in modeling more complex interactions such as non-spherical rigid-body collisions. Here again, advanced math will not be required. This is intended to spark interest in some students for future investigation and study.

This course will foster teamwork. There will be lab exercises, individual and team-based project work, and a competition between teams at the end of the course. This will necessitate both individual effort and cooperative parsing and execution of project tasks.

The course is aimed at those with little or no programing experience and is intended to attract students from all departments at Gustavus. The majority of students may come from science programs, but we also hope to see some history and philosophy majors. Programming is a useful skill for any academic focus.

Finally, I'm expecting to see amazement as students watch their programs literally become life-like on the screen. The oh-wows in this course will come fairly early and painlessly, creating appetite for learning more of the Python language and associated game development methods.

Teaching/Learning Approach:

This course will be completely project driven. Students will incrementally develop the algorithmic tools needed to assemble a 2-D game environment. This overall environment will be instructed through a series of assignments, each building on the previous, until the students have a collection of tools needed for the final project and multiplayer competition in the last week. Each assignment will involve algorithms and code that affect the visual representation of game objects as rendered in the Pygame environment. Instruction on the supporting elements of the Python language will be given as needed. The visual feedback in these exercises helps to motivate the students (and the instructor). It's fun; it can be addicting.

Daily Pattern:

Each day will start with instruction on algorithmic concepts. The ideas will be introduced during a short 20 to 40 minutes morning lecture/discussion. There will usually be some introductory explanation in each assignment description in PDF format. Then there will be a code walk-through of how these algorithms will look in the Python language. The Python code will be discussed and explained while projected to the classroom screen (but the actual code file will not be distributed at this point).

Needed elements of the Python language will be identified in the morning lecture and in the assignment description and then explained in more detail in the afternoon lecture/discussion session, also 20 to 40 minutes. The afternoon session will be directed at the students with less coding experience. Associated and supporting assignments may be given to help students gain experience with the needed language elements.

The day will end with a lab session of 2 to 3 hours. Here the students will work together and start writing the code (as individuals) for the assignment. This will be a time to discuss the assignment with other students and also the instructor. Students with Python experience will be encouraged to help those who are new to the language. We will have a lab space, lecture room, and the second floor commons area to spread out in. There will be eight lab computers that can be used by students without laptops and for teams to test out multiplayer functionality in their games.

The instructor will check individual progress at least once per week at the beginning of lab time. Visual demonstration (does it work?) of individual student code will be observed by the instructor. After progress is noted, releases of code solutions for assignments will be distributed to the class. Solution code can be used by students to modify their code or as a starting point for the next assignment.

Course Outline:

1. First week: **1D Air Track**
2. Second week: **2D Air Table**
3. Third week: **Pybox2d**
4. Forth week: **Project, Tournament, and Exam.**

Weekly Outline:

1. First week: **1D Air Track**
    a. Pygame (morning discussions)
        i. Installation and demonstration of the development environment:

1. Python, Pygame, pybox2d, Notepad++.

ii. Notepad++ (the editing/running environment):
1. Basic editing tips.
2. Starting and debugging programs from Notepad++.

iii. Pygame basics:
1. Importing modules and initializing pygame.
2. Game window and surfaces.
3. Drawing or blitting to a surface.
4. Rectangles, circles, lines, and polygons.
5. Flipping (updating) the display.
6. The event queue.

iv. Air track assignment concepts:
1. The main classes of the air track assignment:
a. Client.
b. AirTrack.
c. Detroit (track cars).
d. Environment.
e. GuiControls.
2. Flow of logic in the game loop.
3. Time-based physics calculations: the integration.
4. Cursor strings and the client class.
5. Mutiple car objects and collision physics.
6. Collisions and penetration corrections.
7. Gui interface and pgu.

b. Python (afternoon discussions)
i. Statements, functions, conditional branching and Boolean logic, loops.
ii. Date types: integer and floating point numbers.
iii. References.
iv. Mutable and immutable objects.
v. Lists and dictionaries.
vi. Naming conventions.
vii. Namespace.
viii. Classes: methods, properties, and instantiation.
ix. Enumerated lists and collision pairs.
x. Debugging.

2. Second week: **2D Air Table**
a. Pygame
i. Vector math and the vector class.
ii. The physics World and the Screen and converting back and forth.
iii. 2D collision physics for circular objects (pucks).
iv. Springs (and dampers) class, rendering the stretched spring, and the aggregate forces on the pucks.
v. Multi-player network features and PodSixNet: server, clients (the game pad), and supporting classes.
vi. Vector rotation.
vii. Multi-player stuff: cursors, cursor strings, tubes, jets, guns, and bullets.

viii. Zooming the view.

b. Python
    i. Operator overloading with the vector class.
    ii. Class inheritance.
    iii. Object deletion.

3. Third week: **pybox2d and etc…**
    a. Introduction to the Box2D physics engine.
    b. The pybox2d test-bed and framework overview.
    c. Sample code and assignments:
        i. Modification of a test-bed example.
        ii. An example without using the test-bed framework.
    d. Other Pygame things: the Twixt board game (no physics please).

4. Forth week: **Project, Exam, and the Tournament.**
    a. **The final project and tournament**: As a team, develop two or more multi-player games. Use the client-server approach demonstrated in the air-table exercise (i.e., game-pad clients with all game-object rendering done at the server). The games should support either two or four clients (players). Base one game on the air-table exercise. Base the other game on pybox2d. The only requirement is that the game has a clear metric for determining a win and that wins be establish in five to ten minutes (or less). An acceptable alternative for one of the games is to develop it without using a physics engine. For example: adapt the Twixt board game for multiplayer network use.

    We'll discuss the nature of the tournament and decide how we want to do this as a class. The initial thought on this is to have it be double elimination with brackets created via an online generator. Each round will have a pair of teams competing in multiple games. Each team contributes one game to be used in the tournament.

    b. **Exam**: a one hour WebAssign (on-line) exam on the elements of the Python language that were used in the course. Questions will be all multiple choice except for one code-composition question.

Grading:

1. Assignments (40%): This is an individual effort but collaboration is encouraged. Individual progress will be checked at least once per week during the first three weeks.
2. Exam (30%): Individual effort on the WebAssign exam.
3. Final Project (30%): Graded as a team.