

CS 1400: Programming I (Python), Midterm Project

Adamic - Spring 2024

Overview

The primary goal of this midterm project is to demonstrate problem-solving skills by implementing the game logic for the classic arcade game "Pong." This project aims to showcase the application of fundamental programming concepts, including basic data structures (lists, tuples, dictionaries), functions, conditional structures, and loops.

Setup / Install

The only dependency for this project is the 3rd party package [pygame](#). To install the dependency, run the following pip install command from the terminal:

```
python -m pip install pygame
```

(replace python with python3 if needed)

Due Date / Submission (Saturday, March 2, 2024)

Final submission (due Saturday, March 2, 2024 @ 11:59p) will require uploading to canvas, as well as sharing your (private) project repository with the Instructor on github.

Make a PRIVATE repository on github for your midterm, and add share permissions with JonAdamic-Weber before the deadline. Here is a refresher on git: <https://github.com/git-guides>

Final Submission Checklist:

- Code passes pylint 10/10 AND name, date, and assignment title are included in file intro doc string (at the top of the file)
- All required functionality works (paddles have accurate collision detection with the balls, etc)
- Code is up to date in your PRIVATE github repository
- Github repository is shared with JonAdamic-Weber
- Project python file is uploaded to canvas submission by the final deadline
- Short <5 minute code/project walk through is uploaded to canvas submission

Functional Requirements

You will be given with starter code that provides an outline for the logic that must be implemented. The following functions MUST exist and implement the corresponding logic.

REQUIRED FUNCTIONS:

process_input():

- Collect keyboard input from the user, and tentatively update the position of each player's paddle.

update():

- Update Ball Position: Tentatively update the position of the ball based on the ball's velocity.
- Boundary Detection: Employ conditional structures and mathematical logic to detect when the ball has passed beyond the right/left boundaries. Increment and keep track of player scores when this happens, then reset the ball position to the center.
- Collision Detection (Ball): Employ conditional structures and mathematical logic to detect collisions between the ball and the paddles, as well as the ball and the top/bottom walls. Handle these collisions appropriately to maintain the integrity of the game (ie change velocity/direction of the ball).
- Collision Detection (Paddle): Detect collisions between each paddle and the top/bottom walls and prevent the paddle from going beyond the game window.

render(): (ALREADY IMPLEMENTED, DON'T CHANGE THIS FUNCTION)

- Draw the objects on screen based on their updated final positions.

get_scores():

- Returns the score of both players in the form of (p1_score, p2_score)

OPTIONAL/SUGGESTED FUNCTIONS:

The following functions may be helpful to break down the logic into smaller steps, but are not strictly required:

- move_player: Update the position of a paddle based on a direction/velocity etc.
- move_ball: Update the position of the ball based on a direction/velocity etc.
- check_boundary: Check if the ball has gone beyond the right/left edge and reset the ball if necessary (and increment scores).
- clamp_paddles: Restrict a paddle from going beyond the top/bottom edges.
- check_wall_collisions: Check if the ball has hit the top/bottom wall and update it's position/velocity/direction accordingly

Data Requirements

The provided starter code relies on required global variables. These required variables are defined in the global scope of the project, and will be used during the final render of the game. Please ensure your update functions properly access and update the following variables:

REQUIRED GLOBAL VARIABLES:

- player1 and player2:
 - Can be of types: dict, list, tuple (your choice).
 - Stores relevant information related to each player.
 - Required information includes the x and y position of this player's paddle (see the game coordinate system description).
 - If of type tuple or list: first two values must be x and y (anything after that is personal choice).
 - If using a dictionary, the keys "x", "y" must exist with their corresponding values.
- ball:
 - Can be of type: dict, list, tuple (your choice).
 - Stores relevant information related to the ball.
 - Required information includes the x and y position of the ball (using the game coordinate system)
 - If of type tuple or list: first two values must be x and y.
 - If using a dictionary, the keys "x", "y" must exist with their corresponding values.
 - Depending on your implementation, you may want to keep track of velocity as part of this data.

Coordinate System



The following coordinate system is used to define the positions of objects on the screen. The coordinate system is based on a Cartesian coordinate system, with the origin (0, 0) located at the top-left corner of the window. The x-axis increases from left to right, and the y-axis increases from top to bottom. Here are some key points about the coordinate system:

- Origin (0, 0): The top-left corner of the window is the origin point (0, 0). It serves as the reference point for positioning objects on the screen.
- Positive X-axis: The x-axis extends horizontally from the left side of the window. As you move to the right, the x-coordinate increases.
- Positive Y-axis: The y-axis extends vertically from the top of the window. As you move down, the y-coordinate increases.
- Screen Dimensions: The screen dimensions are specified in pixels. The width and height of the window determine the coordinate range in both the x and y directions.
- Object X coordinate: Specifies the x value of the top left point of the object.
- Object Y coordinate: Specifies the y value of the top left point of the object.
- Object Width: Specifies how wide the object is in pixels (extending right from the objects top-left origin)
- Object Height: Specifies how tall/long the object is in pixels (extending down from the objects top-left origin)

Video Walkthrough / Code Demo

As part of the final submission, you will need to screen record a short video (<5 minutes) that walks through your code. The main focus is to discuss implementation choices for datatypes/structures, any helper functions you made, your collision detection logic, etc. This should be a very informal and quick video, this is simply to ensure you understand the code that you wrote and can explain your implementation decisions. (in industry, live code demos similar to this are commonly performed when code is submitted for merge requests).

If you need a screen recording software, you may want to use OBS, it is free and open source: <https://obsproject.com/>

Python CA Rubric I							
Criteria		Ratings				Pts	
Coding Standards Follows Naming Convention and language coding style		5 to >4 pts Excellent <ul style="list-style-type: none">Includes name, date, and assignment title.Excellent use of white space.Creatively organized work.Excellent use of variables (no ambiguous naming).Passes pylint 10/10	4 to >2 pts Good <ul style="list-style-type: none">Includes name, date, and assignment title.Good use of white space.Organized work.Good use of variables (no ambiguous naming).Passes pylint 8+	2 to >0 pts Satisfactory <ul style="list-style-type: none">Includes name, date, and assignment title.White space makes program fairly easy to read.Organized work.Passable use of variables (no ambiguous naming).Passes pylint 4/10	0 pts Unsatisfactory <ul style="list-style-type: none">No name, date, or assignment title includedPoor use of white space (indentation, blank lines).Disorganized and messyPoor use of variables (ambiguous naming).Pylint <4/10	5 pts	
Documentation The use of docstrings, GIT README file		5 to >3 pts Excellent <ul style="list-style-type: none">Clearly and effectively documented including descriptions of all variables.Specific purpose is noted for each function, control structure, input requirements, and output results.	3 to >2 pts Good <ul style="list-style-type: none">Clearly documented including descriptions of all variables.Specific purpose is noted for each function and control structure.	2 to >0 pts Satisfactory <ul style="list-style-type: none">Basic documentation has been completed including descriptions of all variables.Purpose is noted for each function.	0 pts Unsatisfactory <ul style="list-style-type: none">No documentation included.	5 pts	
Delivery I Include code demo		15 to >12 pts Excellent <ul style="list-style-type: none">Completed between 90-100% of the requirements.Delivered on time, and in correct format (disk, email, etc.)Shows code demo	12 to >7.5 pts Good <ul style="list-style-type: none">Completed between 80-90% of the requirements.Delivered on time, and in correct format (disk, email, etc.)	7.5 to >1.5 pts Satisfactory <ul style="list-style-type: none">Completed between 70-80% of the requirements.Delivered on time, and in correct format (disk, email, etc.)	1.5 to >0 pts Unsatisfactory <ul style="list-style-type: none">Completed less than 70% of the requirements.Not delivered on time or not in correct format (disk, email, etc.)	15 pts	
Code Walkthrough A screencast explaining your homework assignment.		5 pts Good Explains how the code works. This includes functions, classes with all the methods.		3 pts Satisfactory Explains some parts of the code.	0 pts Unsatisfactory No explanation included.	5 pts	
Total Points: 30							