MCS 260 Project Five : developing a simple video game
due Wednesday 3 December at 1PM

The goal of the project is to develop a simple video game, combining OOP with a GUI.

The objective of this single player game is to harvest apples falling from the sky. There are
good and bad apples. Harvesting a good apple increases the score by one. If a bad apple lands
in the basket, the game is over.

Two Python scripts implement this game. The first Python script 'harvest.py' outlines
the game, without animating graphics. The GUI is defined by the second script 'falling.py'
which imports class definitions from the 'harvest.py' script. The script 'harvest.py' uses
four classes: Apple, Player, Display, and Game.

The class Apple represents a falling apple, using five object data attributes: its location,
height, speed, goodness, and its identification number. By default the speed is 1, the apple is
good, and the identification number is 1. The Fall method decrements its current height by
the value of the speed. If the height becomes negative it is set to 0. For testing purposes it is
recommended to write code for __str__

The class Player represents the player, using two object data attributes: its location and the
score. The default score is 0. The score equals the number of harvested apples. The method
Catch takes on input a list of apples. Catch adds 1 to the score for each good apple that is at
height zero and at the same location of the player. If a bad apple is at height zero and at the same
location of the player, then the score is set to -1. Two other arguments of Catch are height and
margin, each having default values equal to zero. Giving nonzero values to height will consider
apples at that height for updating the score. Giving positive values to margin determines the
width of the player's basket, i.e.: to catch an apple its location should be between plus or minus
the margin of the player's location. The method Move takes on input a step size (positive or
negative) which gets added to the current location of the player.

The classes Apple and Player are imported for use in the GUI. The GUI will replace the
Display class. The dimensions of the display, the number of rows and columns, are stored as
object data attributes. For r rows and c columns, we think of the display as an r-by-c matrix of
single characters. The method Show takes on input a list of tuples (i,j,s) to put the character
s at row i and column j. Show prints a space at (i,j) when the list contains no corresponding
(i,j,s). If two elements in the list with the same (i,j) occur, we follow these rules. If a good
apple falls in the basket of the player, then the basket is displayed. If a bad apple lands in the
basket, then the bad apple is shown.

An object of the class Game contains a player, a list of apples, and a display as object data
attributes. The instantiation of the game requires the dimensions of the display. The list of
apples is initially empty and the location of the player is set at the middle column of the display.
There are four methods in this class: Update, CleanFallen, Locations, and Play. The Update
method applies the method Fall to every apple and creates one new apple at the top height at
some random location. The method CleanFallen removes all apples on the ground. The method
Locations computes the list of tuples (i,j,s) from the apples and the location of the player
needed as input of the Show method of the Display object. Good and bad apples are displayed
using 'o' and '*' respectively. The character to represent the basket of the user is 'u'. The
method Play updates the display, shows it to the player, and prompts for a move. When the apples or the player move, the player may catch good apples or bad apples may catch the player.

The function main prompts the user for the number of rows and columns in the display. After instantiating the game, the main loop calls the method Play. This loop ends when the score of the player turns $-1$ or when the user does not answers ‘y’ to the ‘continue?’ question. At the end of the game, the final score is displayed.

Running the game at the command prompt $\$ $ could go as follows:

```
$ python harvest.py
Number of rows in display : 4
Number of columns in display : 10
score : 0

o

o
give step size : +1
continue (y/n) y
score : 0

o

o

u
give step size : 0
continue (y/n) y
score : 0

*

o

o

u
give step size : 0
continue (y/n) y
score : 0

*

o

o

u
give step size : 0
continue (y/n) y
score : 1

*

o

o

u
give step size : 0
continue (y/n) y
score : 2

*

o

u
give step size : -1
continue (y/n) n
collected 2 apples

```

The script ‘harvest.py’ represents half of the project and defines the game. The main feature of the GUI will be the animation to show the falling apples.
When the script 'falling.py' is launched, we see

The GUI consists of three labels to define the purpose of the game, and to document the role of the two scales. The vertical scale controls the number of apples, the horizontal scale determines the location of the basket. The score is displayed in an entry widget. The animation begins when the user hits the start button. The stop button stops the animation.

At some point we may see the following:

The radius of the circles to draw the apples is 6 pixels. Good apples are colored in green,
circles filled with red represent the bad apples. The floor of the basket equals the diameter of the 
circles and the upper corners are 3 pixels farther and up from the floor of the basket.

The dimensions of the canvas are 400 pixels wide and high. Because of this increased resolu-
tion, we take the speed at which the apples drop to be at random between 1 and 20. The speed 
at which the canvas is refreshed is determined when the GUI is instantiated. By default, the GUI 
waits 10 time units before updating the canvas. We notice that the more apples the slower they 
drop. Apples start falling faster when they are fewer in number.

The game stops when a bad apple lands in the basket:

The development of the game consists in three parts:

1. using object data attributes to store coordinates of apples and player information, 
coding basic methods for the moves of the apples and player;

2. defining the layout of the GUI and an animation to show apples falling from the sky;

3. configuring the change in the number of apples and catching apples, updating the score.

Fixing the layout of the GUI and the definition of the basic classes in the script harvest.py can 
be done independently.
Some important points:

1. Although the code consists of two parts, there is work enough for a team of three programmers. We suggest that pairs for which the last project turned out too hard split up and join stronger pairs. All authors will receive the same score.

2. Feel free to work on your own and solve the project individually if that suits you better. However, sharing code is not allowed and will be considered as plagiarism. If you start collaborating, then you should stick with each other till the deadline.

3. One of the goals of this project is to use classes. Correct programs without the use of classes can only get half of the points.

4. Providing appropriate documentation will receive proper credit. In particular: every function must have a documentation string of at least one line long.

5. Avoid spelling mistakes in the dialogue with the user.

6. Handing in an incomplete but working program is better than handing in a program that crashes or does not run at all.

7. The solution to the project consist of two files: harvest.py and falling.py. Each have their own main program.

8. The first line of your Python files must be

```
# MCS 260 Project Five by <Author(s)>
```

where you replace the `<Author(s)>` by your name(s) (in alphabetical order). Only one of the authors should submit two files.

9. Email your solution to the project to jan@math.uic.edu before 1PM on Wednesday 3 December so the date of the email is proof of an on time submission. Also bring also a printed version of your solution to class.

10. Late submissions before 5PM on Wednesday 3 December will be discounted with 10 points.

If you have questions or difficulties with the project, feel free to come to my office for help.