Introducing the Pizza Panic Game

The project for this chapter, the Pizza Panic game, involves a crazy chef, a deep-dish pan, and a bunch of flying pizzas. Here's the scenario: After being pushed over the edge by one-too-many finicky diners, the chef at the local pizza parlor has taken to the rooftop and is madly flinging pizzas to their doom. Of course, the pizzas must be saved. Using the mouse, the player controls a pan that he or she maneuvers to catch the falling pizzas. The player's score increases with every pizza caught. But once a pie hits the ground, the game is over. Figures 11.1 and 11.2 show the game in action.

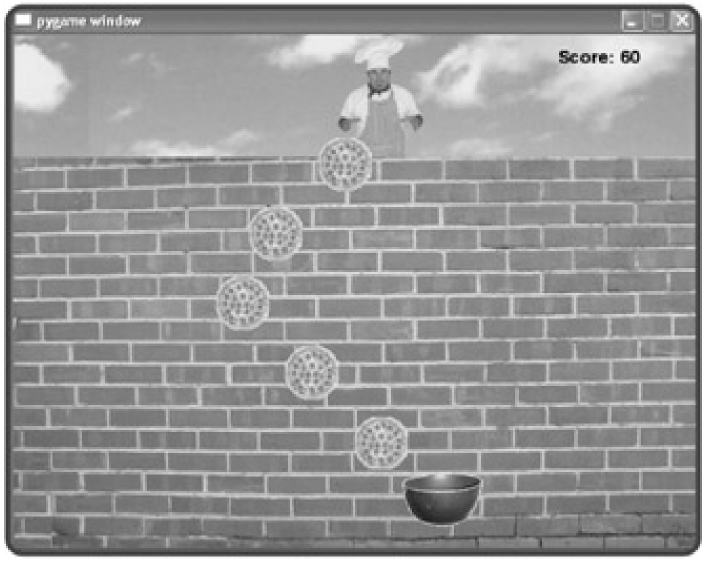

Figure 11.1: The player must catch the falling pizzas.

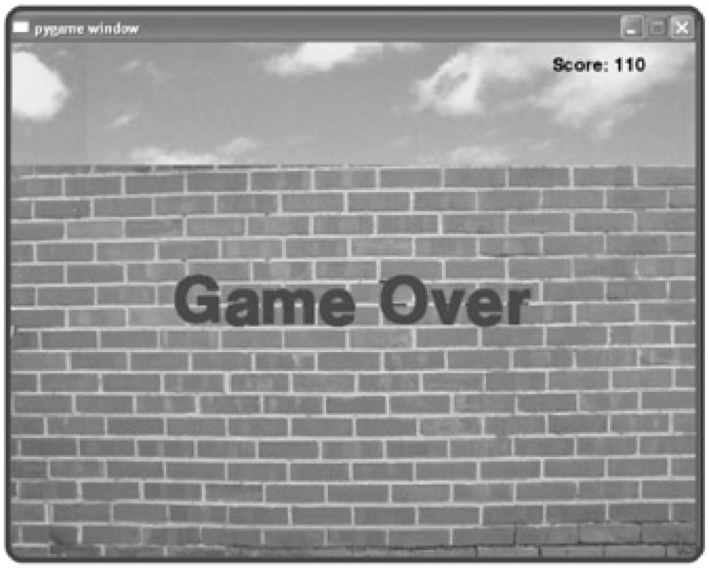

Figure 11.2: Once a pizza gets by the player, the game is over.

Back to the Pizza Panic Game

Now that you've gotten a taste of what the `livewires` multimedia package can do, it's time to create the Pizza game introduced at the beginning of the chapter. Much of the program for the game can be taken directly from the ex programs. However, I'll also introduce a few new concepts as I put the game together.

Setting Up the Program

As in all of the programs in this chapter, I begin by importing the modules and setting some global constants:

```
# Pizza Panic
# Player must catch falling pizzas before they hit the ground
# Michael Dawson 5/12/03

import random
from livewires import games, color

SCREEN_WIDTH = 640
SCREEN_HEIGHT = 480
THE_SCREEN = games.Screen(SCREEN_WIDTH, SCREEN_HEIGHT)
```

To do any graphics work, I need to import `games` , while `color` gives me access to the set of predefined colors to us creating screen text. I import `random` so that the crazy chef seems more life-like when he makes his choices. Next, I global constants for the width and height of the graphics screen. Then I do something new. I create the graphics scre assign it to the global constant `THE_SCREEN` . I do this because I need the screen object in existence before I can lo image object, which I do in several class definitions in this program, as you'll soon see.

The `Pan` Class

The `Pan` class is a blueprint for the pan sprite that the player controls through the mouse. However, the pan will only left and right. I'll go through the class, one section at a time.

Loading the Pan Image

I do something a little different in the beginning of this class; I load a sprite image and assign it to a class variable, `ir` do this because Pizza Panic has several classes, and loading an image in its corresponding class definition is cleare loading all of the images in the program's `main()` function.

```
class Pan(games.Sprite):
    """
    A pan controlled by player to catch falling pizzas.
    """
    image = games.load_image("pan.bmp")
```

The `__init__()` Method

Next, I write the constructor, which creates a new `Pan` object with the given coordinates. I define an attribute, `score_value` , for the player's score, which I set to `0` . I also define another attribute, `score_text` , for a `Text` obj
displays the score on the graphics screen.

```
    def __init__ (self, screen, x, y):
        """ Initialize pan object. Create a Text object for player's score. """
        self.init_sprite(screen = screen, x = x, y = y, image = Pan.image)
        self.score_value = 0
        self.score_text = games.Text(screen = self.screen, x = 550, y = 20,
                                     text = "Score: 0", size = 25, color = color.b
```

The `moved()` Method

This method moves the player's pan:

```
    def moved(self):
        """ Move pan to mouse x position. """
        x, y = self.screen.mouse_pos()
        self.move_to(x, self.get_ypos())
        if self.get_left() < 0:
            self.set_left(0)
        if self.get_right() > SCREEN_WIDTH:
            self.set_right(SCREEN_WIDTH)
        self.check_for_catch()
```

The method gets the coordinates of the mouse, then moves the player's pan to the mouse's x-coordinate and the pa
current y-coordinate. By using the pan's current y-coordinate, the pan stays locked at the same height. As a result, t
can move left and right but not up and down.

Next, I use the object's `get_left()` method to check if the left edge of the pan is less than `0` , meaning that it's bey
left edge of the graphics window. If so, I set the left edge to `0` with the object's `set_left()` method. This way, the p
never be drawn beyond the left edge of the graphics window.

Then, I use the object's `get_right()` method to check if the right edge of the pan is greater than `SCREEN_WIDTH` ,
meaning that it is beyond the right edge of the graphics window. If so, I set the right edge to `SCREEN_WIDTH` with the
object's `set_right()` method. This way, the pan will never be drawn beyond the right edge of the graphics window

Finally, I invoke the object's `check_for_catch()` method.

The `check_for_catch()` Method

This method checks if the player has caught one of the falling pizzas:

```
    def check_for_catch(self):
        """ Check if pan catches a pizza. """
        for pizza in self.overlapping_objects():
            self.handle_caught()
            pizza.handle_caught()
```

The method goes through the list of objects that overlap the player's pan. For each object that overlaps the `Pan` obje
method invokes the `Pan` object's own `handle_caught()` method and then invokes the overlapping object's `handle_caught()` method.

The `handle_caught()` Method

This method is called whenever the player's pan and a falling pizza collide:

```
    def handle_caught(self):
        """ Increase and display score. """
        self.score_value += 10
        self.score_text.set_text("Score: "+ str(self.score_value))
```

This method increases the `Pan` object's `score_value` attribute score by `10` each time a pizza is caught. In order to
the change in the player's score on the graphics screen, the `Text` object for the score must be updated. So next, thi
method invokes the `set_text()` method of the `score_text` attribute of the `Pan` object. `set_text()` assigns a n
string to the object's `text` attribute to reflect the player's new score.

The `Pizza` Class

This class is for the falling pizzas that the player must catch:

```
class Pizza(games.Sprite):
    """
    A pizza which falls to the ground.
    """
    image = games.load_image("pizza.bmp")
    START_Y = 90                # start any pizza at chef's chest-level
    speed = 1
```

I define three class variables: image for the pizza image, `START_Y` , a constant for all pizzas' starting y-coordinate, a
`speed` , a class variable for all pizzas' falling speed. `START_Y` is set to `90` so that any newly created pizza will appea
chef's chest level on the graphics screen. I set `speed` to `1` so that the pizzas fall at a fairly slow speed. I use all three
variables in the `Pizza` constructor method, as you'll soon see.

I didn't make `speed` a constant because I thought I might want to change the speed at which the pizzas fall as the g
progresses in a future version of the program (or you might want to, if you accept the chapter challenges).

The `__init__()` Method

This method initializes a new `Pizza` object:

```
    def __init__(self, screen, x):
        """ Initialize a pizza object. """
        self.init_sprite(screen = screen, x = x, y = Pizza.START_Y,
                         dx = 0, dy = Pizza.speed, image = Pizza.image)
```

When the constructor method of a newly created `Pizza` object is invoked, the object's `init_sprite()` method is i to initialize the sprite.

The `moved()` Method

This method handles screen boundary checking:

```
def moved(self):
    """ Check if a pizza's bottom edge has reached screen bottom. """
    if self.get_bottom() > SCREEN_HEIGHT:
        self.game_over()
```

All this method does is check if a pizza has reached the bottom of the screen. If it has, the method invokes the objec `game_over()` method.

The `handle_caught()` Method

Remember, this method is invoked by the player's `Pan` object when the `Pizza` object collides with it:

```
def handle_caught(self):
    """ Destroy self if caught. """
    self.destroy()
```

When a pizza collides with a pan, the pizza is considered "caught" and simply ceases to exist. So, the `Pizza` object its own `destroy()` method and the pizza literally disappears.

The `game_over()` Method

This method is invoked by `moved()` when a pizza reaches the bottom of the screen. The method ends the game.

```
def game_over(self):
    """ End the game. """
    # destroy all game objects except the Text object (player's score)
    for game_object in self.screen.all_objects():
        if not isinstance(game_object, games.Text):
            game_object.destroy()

    # show 'Game Over' for 250 mainloop() cycles (at 50 fps that's 5 seconds)
    games.Message(screen = self.screen,
                  x = SCREEN_WIDTH/2, y = SCREEN_HEIGHT/2,
                  text = "Game Over", size = 90, color = color.red,
                  lifetime = 250, after_death = self.screen.quit)
```

When this method is invoked, the player's pan, the crazy chef, and all of the pizzas disappear from the screen. Then, message "Game Over" is displayed in big, red letters. About five seconds later, the program ends.

The `for` loop moves through all of the objects on the screen and destroys each one, except the `Text` object, which represents the player's score. The method checks if each object is a `Text` object with the `isinstance()` Python fu which takes an object and a class as arguments. `isinstance()` is `True` if the object is an instance of the class, ar `False` otherwise.

Next, the `game_over()` method creates a `Message` object that declares that the game is over. Since the `lifetim`
attribute is `250` and `mainloop()` is running at 50 cycles per second, the message stays on the screen for about five
seconds. At that point, the method specified in the `after_death` attribute of the `Message` object is invoked. The sp
method is the `Screen` object's `quit()` method, so the graphics window disappears and the program ends.

The `Chef` Class

The `Chef` class is used to create the crazy chef who throws the pizzas off the restaurant rooftop. The class has a
constructor method, a `moved()` method, and a `drop_pizza()` method, which, you guessed it, allows the chef to dr
new pizza.

```
class Chef(games.Sprite):
    """
    A chef which moves left and right, dropping pizzas.
    """
    image = games.load_image("chef.bmp")
    Y = 55                      # put the chef right on the top of the brick wall
```

I define two class variables. `image` is for the chef image and `Y` is for the starting y-coordinate of the `Chef` object. I se
`55`, which will put the image of the chef right at the rooftop.

The `__init__()` Method

This method creates a new chef:

```
    def __init__ (self, screen, x, speed, odds_change):
        """ Initialize the Chef object. """
        self.init_sprite(screen = screen, x = x, y = Chef.Y,
                         dx = speed, dy = 0, image = Chef.image)
        self.odds_change = odds_change
        self.time_til_drop = 0
```

First, I invoke the newly created `Chef` object's `init_sprite()` method to initialize the sprite. I pass the class cons
for the y-coordinate. `dx` is passed `speed`, which determines the chef's horizontal velocity as he moves along the roo

The method also creates two attributes, `odds_change` and `time_til_drop`. `odds_change` is an integer that rep
the odds that the chef will change his direction. For example, if `odds_change` is `250`, then there's a 1 in 250 chanc
every time the chef moves, he'll reverse direction. You'll see how this works in the `moved()` method of the class.

`time_til_drop` is an integer that represents the amount of time, in `mainloop()` cycles, until the next time the che
his next pizza. I set it to `0` initially, meaning that when a `Chef` object springs to life, it should immediately drop a pizz
see how `time_til_drop` works in the `drop_pizza()` method.

Lastly, since I've used OOP to build Pizza Panic, it becomes a trivial task to have multiple chefs in the same game. W
one additional line of code to instantiate another `Chef` object, I can have two crazy, hat-wearing men tossing pizzas
the player's pan. Though I'll be using only one chef in this version of the game, this knowledge might come in handy (
for a chapter challenge).

The `moved()` Method

This method defines the rules for how the chef decides to slide back and forth along the rooftop:

```
    def moved(self):
        """ Determine if direction needs to be reversed. """
        if self.get_left() < 0 or self.get_right() > SCREEN_WIDTH:
            self.reverse()
        else:
            same_direction = random.randrange(self.odds_change)
            if not same_direction:
                self.reverse()
        self.drop_pizza()
```

A chef slides along the rooftop in one direction until he either reaches the edge of the screen or "decides," at random switch directions. The beginning of this method checks to see if the chef has moved beyond the left or right edge of t graphics window. If he has, then the `reverse()` method is invoked. Otherwise, the chef has a 1 in `odds_change` c of changing direction.

Regardless of whether or not the chef changes direction, the last thing the method does is invoke the `Chef` object's `drop_pizza()` method.

The `reverse()` Method

This method is invoked by `moved()` and reverses the chef's direction:

```
    def reverse(self):
        """ Reverse direction. """
        dx, dy = self.get_velocity()
        self.set_velocity((-dx, dy))
```

This method is quite simple. It reverses the horizontal velocity of the chef, changing his direction.

The `drop_pizza()` Method

This method is invoked every time `moved()` is invoked, but that doesn't mean a new pizza is dropped each time:

```
    def drop_pizza(self):
        """ Decrease countdown or drop pizza and reset countdown. """
        if self.time_til_drop:
            self.time_til_drop -= 1
        else:
            # set so buffer will be 15 pixels, regardless of pizza speed
            self.time_til_drop = int(65 / Pizza.speed)
            Pizza(self.screen, self.get_xpos())
```

`time_til_drop` represents a countdown for our chef. If `time_til_drop` is not `0` , then `1` is subtracted from it. Otherwise, `time_til_drop` is reset and a new `Pizza` object is created. The value of `time_til_drop` is determin the height of the pizza sprite image and the speed at which the pizzas are falling. Since the pizza image is 50 pixels l the formula provides a nice 15 pixel-sized gap between each pie, independent of the falling speed.

The `main()` Function

The `main()` function creates a graphics screen, creates graphics objects and then kicks off the `Screen` object's `mainloop()` to run the show:

```
def main():
    my_screen = THE_SCREEN
    my_screen.mouse_visible(False)
    wall_image = games.load_image("wall.jpg", transparent = False)
    my_screen.set_background(wall_image)

    Chef(screen = my_screen, x = SCREEN_WIDTH/2, speed = 1, odds_change = 250)
    Pan(screen = my_screen, x = SCREEN_WIDTH/2, y = 435)

    my_screen.mainloop()

# start program
main()
```

First, I assign the graphics screen to `my_screen` and set the mouse pointer to invisible. Then, I set the brick wall as background.

Next, I create a chef with a speed of 1 and a 1 in 250 chance of changing directions each move. Then, I create the pl
pan with a y-coordinate of 435, putting it at the bottom of the screen.

Finally, I invoke `my_screen'` s `mainloop()` and the game begins.

Summary

In this chapter, you saw how to use the `livewires` multimedia package to add graphics to your programs. You learned how to create a new graphics window and how to set a background image for it. You saw how to display text on a graphics window. You learned about the sprite, a special graphics object with an image. Specifically, you saw how to place and move a sprite on a graphics screen. You also saw how to test for collisions between graphics objects. You learned how to get input from the mouse. Finally, you saw how to put everything together in a fast-paced, video game, complete with a computer-controlled opponent.

Challenges

1. Improve the Pizza Panic game by increasing its difficulty as the game progresses. Think of different ways to accomplish this. You could increase the speed of the pizzas and the speed of the chef. You could raise the player's pan to a higher position on the screen. You could even increase the number of crazy chefs flinging pizzas.

2. Create a simple, one-player game of pong, where a player controls a paddle, and the ball bounces off three walls. If the ball gets by the player's paddle, the game is over.

3. Write a game where the player controls a character that must avoid falling debris. The player controls the character with the mouse, and objects fall from the sky.