Building a Mobile Game: with PyGame and PyDroid

By Paul Moggridge (p.moggridge@herts.ac.uk)

1 Task

In this activity, you will develop your Python coding skills by creating a Python mobile game. By end of this activity you will have considered the design implications of building a mobile game and have developed a small mobile-friendly game using PyGame.

(a) An oven

(b) A knife

(c) A table

(d) A potato

(e) A cooked potato

(f) A burnt potato

(g) Wedges

(h) Cooked wedges

(i) Burnt wedges

Figure 1: Sprites (images) for your game. You can use these images or draw your own (making sure you use a transparent background) or chose your own from a website such as Flaticon.

2 Learning Objectives

- Knowledge of the basic structures in a real-time graphical game.
- Knowledge of some of PyGame's important functions.
- Knowledge of the design considerations for building a mobile game.
- Be able to apply your Python programming knowledge to build mobile 2D graphical game.

3 Setup

This activity sheet assumes you have successfully installed Python 3 (including Python Pip) and an Integrated Development Environment (IDE) such as Visual Studio Code.

In your terminal, in Visual Studio Code (Terminal > New Terminal). Do one of the following:

Option 1 (simplest, least storage used, beginner friendly): Simply execute `pip install pygame` this will install the PyGame module into your base environment.

Option 2: (best practice, advanced users): Create and use a python virtual environment for PyGame. In your user/home directory run `mkdir pyvenvs`, then: `cd pyvenvs` then: `python {m venv pygame-venv` and activate it (on Linux: `Source pygame-venv/bin/activate` on Windows: `C:\Users\YourUsername\pyvenvs\pygame-venv\Scripts\activate`) (don't forget you can use the tab to auto-complete!).

If you skip this step, you won't be able to run the game, and will receive an `ModuleNotFound` error when trying to run the game on your computer.

4 Download Game Files

Create folder for the game "potatochef". Then copy or download the code in the below Figure into new files "main.py" and "potatochef.py".

Download the code and resources here: **Game Files**

5 Optional Setup (for fun)

You can develop and play the game on your computer, but if you have a android device you may be play this game on your phone too!

On the Play store download "PyDriod" app. This is an Integrated Development Environment (IDE) and Python environment all-in-one, it is great for running Python programs, that have been written on your computer, or even on the app itself, on the go.

When you have completed this activity sheet, copy the game files to your phone and open PyDroid. Locate the files and open "main.py". Press the play button and your game should run.

Speaking from experience, testing this with several models of Android phone, most of the time it works fine and runs exceedingly well. However, some models (especially newer models) have tighten security around apps accessing files, and may block your code from loading the image icons.

6 Challenges

1. Find `TODO 1` and change the background colour to a colour of your choosing.
2. Find `TODO 2` and replace the code adding two potatoes, with a loop which adds six potatoes.
3. Find `TODO 3a/b/c/d/e/f/g/h/i/j/k/l/m` and add a cutting area (knife sprite) which chops the potatoes into wedges. Wedges can be dragged to oven and cooked.
4. Find `TODO 4a/b/c/d/e` and add scoring mechanism. For example: serving a raw potato to the table is worth -£5, serving a cooked potato is worth £7, serving a cooked wedges is worth £10... so on. The score is accumulated and displayed to the player on screen.

7 Advanced Challenge

"Googling"[1] for solutions to errors and bugs in your code in common practice. But another common practice in coding is "Googling" for pieces "boilerplate" code. The term "boilerplate" is used to describe code that is commonly implemented. Many good websites and communities exist to help programmers, but one of the biggest is StackOverflow. Take a look at `https://stackoverflow.com/questions/30720665/countdown-timer-in-pygame` and see how the community members have implemented a timer in PyGame. Integrate their solution into your code. Make the game end once your 20 second timer has expired and display a game over message. Use a comment to credit the post for the code you copied.

[1] Other search engines are available!

8 Further Suggestions

You could extend this game with more potatoes recipes or indeed include other foods and prepare more complex dishes.

Another fairly easy suggestion is adding a high-score feature. The game could store store the high-est scores in a text file and display them on the game over screen.

One of inspirations for this was "Overcooked", a time pressured cooking game, and one of the concepts this game, is that certain meals are being ordered and you must make requested dish. This feature could be added to this game to enhance the complexity of the game play.

One way in which the quality of the software could be improved would be added error handling on image loading. Use a `try` block to catch errors thrown if the images cannot be found and take action in the `except` block to handle the issue. (Simplest) You could display message on screen which explains that the images could not be loaded. (Advanced) You could embed the lower quality versions images directly into the script (for example, base64 encoded) or draw some images in code (for example, pillow library), to recover from the images not loading.

9 Code

Figure 2: main.py - available for download:

```
import pygame
from potatochef import PotatoChef

def main():
    # Initialise pygame
    pygame.init()
    screen = pygame.display.set_mode((600,800)) # on mobile this size is ignored
    pygame.display.set_caption('Potato Chef (version 1.0.0.1)')
    fps_clock = pygame.time.Clock()

    # In PyDroid on Android the display size we set above is ignored
    screen_rect = screen.get_rect()

    # initialise game object
    game = PotatoChef(screen_rect.w, screen_rect.h)
    game.load_images()
    game.new_game()

    # font for on screen text
    # TODO: 4d - load a font for the on screen text

    # Is a potato being dragged?
    touched = False

    running = True
    while running:

        # Check pygame events
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                # set value which escapes this game loop
                running = False
```

```
            elif event.type == pygame.MOUSEBUTTONDOWN:
                # Checking each potato for if they are being dragged
                for potato in game.potato_group:
                    if potato.rect.collidepoint(event.pos):
                        # record which potato is being dragged
                        dragged = potato
                        touched = True
                        pygame.mouse.get_rel()
            elif event.type == pygame.MOUSEBUTTONUP:
                touched = False

        # dragged potato follows mouse
        if touched:
            dragged.rect.move_ip(pygame.mouse.get_rel())
            dragged.rect.clamp_ip(screen_rect)

        # set background colour
        screen.fill((80,80,80)) # TODO: 1 - Change the background colour.

        # update the game state
        game.update()

        # draw game object in the correct order, potatoes ontop of oven, so potatoes second
        game.oven_group.draw(screen)
        # TODO: 3e - draw the knife you created in 3d
        game.table_group.draw(screen)
        game.potato_group.draw(screen)

        # draw text
        # TODO: 4e - render and display the text onto the screen.

        # display update
        pygame.display.update()
        fps_clock.tick(30)

# Entry point
if __name__ == '__main__':
    main()
```

Figure 3: potatochef.py

```
import pygame
import random
from enum import Enum

class PotatoChef:

    def __init__(self, sx, sy):
        self.sx = sx
        self.sy = sy
        self.image_xy_size = self.sx // 12 # all images are square and equally sized for
            simplicity of this game
        # TODO: 4a - create variable to store the score.

    def load_images(self):
        self.potato_img = pygame.transform.scale(pygame.image.load('res/potato.png'),
            (self.image_xy_size, self.image_xy_size))
        self.potato_img_cooked =
            pygame.transform.scale(pygame.image.load('res/potato_cooked.png'),
```

```
            (self.image_xy_size, self.image_xy_size))
        self.potato_img_burnt =
            pygame.transform.scale(pygame.image.load('res/potato_burnt.png'),
            (self.image_xy_size, self.image_xy_size))
        # TODO: 3f - Load the wedges images.
        self.oven_img = pygame.transform.scale(pygame.image.load('res/oven.png'),
            (self.image_xy_size*2, self.image_xy_size*2))
        # TODO: 3b - Load the knife on a chopping board image.
        self.table_img = pygame.transform.scale(pygame.image.load('res/table.png'),
            (self.image_xy_size*4, self.image_xy_size*4))

    def new_game(self):
        self.oven_group = self._make_oven()
        # TODO: 3d - Create sprite group, using function 3c
        self.table_group = self._make_table()
        self.potato_group = self._make_potatoes()

    # Updates game state
    def update(self):

        # check each potato for colisions..
        for potato in self.potato_group:
            if potato.collide(self.oven_group):
                print("COOKING" + random.randint(1,5)*'!') # for easy debugging
            # TODO: 3m - check for collision with the knife group...
            # ...like above print a message so that you can see when the collision is
                happening.
            if potato.collide(self.table_group):
                print("!!!!\tYUM\t!!!!") # for easy debugging
                # TODO: 4c - Add the value of the potato (from 4b) to the score.

        # Update potatoes, calls all potatoes update functions
        self.potato_group.update()

    def _make_oven(self):
        oven_group = pygame.sprite.Group()
        oven_group.add(Oven((self.sx/3) * 2, self.sy/3, self.oven_img))
        return oven_group

    # TODO: 3c create a function, which returns a sprite group containing the knife
        sprite.

    def _make_table(self):
        table_group = pygame.sprite.Group()
        table_group.add(Table(self.sx/2, (self.sy/4) * 3, self.table_img))
        return table_group

    def _make_potatoes(self):
        potato_group = pygame.sprite.Group()
        # TODO: 2 - Use a loop to create potatoes, instead of the below two lines.
        potato_group.add(Potato(50, 100, self.potato_img, self.potato_img_cooked,
            self.potato_img_burnt))
        potato_group.add(Potato(50, 150, self.potato_img, self.potato_img_cooked,
            self.potato_img_burnt))
        return potato_group

class PotatoState(Enum):
    RAW = 1
    COOKED = 2
    BURNT = 3
    RAW_WEDGES = 4
```

```
    COOKED_WEDGES = 5
    BURNT_WEDGES = 6

class Potato(pygame.sprite.Sprite):
    # TODO: 3g (part 1) - Ask for the wedges images as parameters to the potato class.
    # TODO: 3g (part 2) - You will also need to update the _make_potatoes function to
        provide the wedges images.
    def __init__(self, x, y, img, img_cooked, img_burnt):
        super().__init__()
        self.image = img
        self.potato_img = img
        self.img_cooked = img_cooked
        self.img_burnt = img_burnt
        # TODO: 3f - store the wedges images you provided as paramters into object
            variables
        self.rect = self.image.get_rect()
        self.rect.center = (x, y)
        self.cooked = 0
        # TODO: 3g - add variable to store how chopped this potato is.

        # timings
        # TODO: 3h - add a variable to represent how long it takes to chop.
        self.cook_time = 300
        # TODO: 3i - add a variable to represent how long it takes to cook wedges.
        self.burn_time = 50


    def calculate_state(self):
        # TODO: 3j - update this if statment to return the correct...
        # ...state for different chopped and cooked values. See the potato state enum.
        if self.cooked > (self.cook_time + self.burn_time):
            return PotatoState.BURNT
        elif self.cooked > self.cook_time:
            return PotatoState.COOKED
        else:
            return PotatoState.RAW

    # TODO: 4b - write function which calculates the value of serving this potato, based
        on this state

    def update(self):
        # Update inage to match state
        state = self.calculate_state()
        if state == PotatoState.RAW:
            self.image = self.potato_img
        elif state == PotatoState.COOKED:
            self.image = self.img_cooked
        elif state == PotatoState.BURNT:
            self.image = self.img_burnt
        # TODO: 3k - update the image variable (which is one displayed), based on the
            state.
        else:
            self.image = self.potato_img

    def collide(self, spriteGroup):
        if pygame.sprite.spritecollide(self, spriteGroup, False):
            # Which sprite did it collide with?
            if isinstance(spriteGroup.sprites()[0], Oven):
                self.cooked = self.cooked + 1
            # TODO: 3l - If there a collision is with a Knife increment chopped variable
                from 3g
```

```
            if isinstance(spriteGroup.sprites()[0], Table):
                self.kill()
            return True
        return False

class Oven(pygame.sprite.Sprite):
    def __init__(self, x, y, img):
        super().__init__()
        self.image = img
        self.rect = self.image.get_rect()
        self.rect.center = (x, y)

# TODO: 3a - Add a class for Knife

class Table(pygame.sprite.Sprite):
    def __init__(self, x, y, img):
        super().__init__()
        self.image = img
        self.rect = self.image.get_rect()
        self.rect.center = (x, y)
```

---

10 Image Credit

- Potato - created by AomAm - Flaticon
- Oven - created by Freepik - Flaticon
- Knife - created by Freepik - Flaticon
- Table - created by Freepik - Flaticon

Icons free for personal and commercial use with attribution.